

TMS320TCI6488 Chip Support Library API Reference Guide

June 2006

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Networking	
		Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments

Post Office Box 655303, Dallas, Texas 75265

Copyright © 2005, Texas Instruments Incorporated

Preface

Read This First

About This Manual

The API reference guide serves as a software programmer's handbook for working with the TMS320TCI6488 CSL.

The purpose of this document is to identify the set of published Chip Support Library (CSL) APIs for the TMS320TCI6488 device. The application developer is expected to refer to this document while designing applications that use these modules.

Abbreviations

Table of Abbreviations

Abbreviation	Description
API	Application Programming Interface
CSL	Chip Support Library
EDMA	Enhanced Direct Memory Access
EMAC	Ethernet Media Access Controller
MCBSP	Multi Channel Buffered Serial Port
VCP	Viterbi Decoder Coprocessor
TCP	Turbo Decoder Coprocessor
TSC	Time Stamp Counter
IDMA	Internal DMA
DDR	Double Data Rate
DTF	DSP Trace Formatter
EMIF	External Memory Interface
ETB	Embedded Trace Buffer
GPIO	General Purpose Input/Output
I2C	Inter Integrated Circuit
INTC	Interrupt Controller
EDC	Error Detection Correction
PLL	PLL Controller
MDIO	Management Data Input/Output
SRIO	Serial Rapid IO
BWMNGMT	Bandwidth Management
MEMPROT	Memory Protection
CFG	Configuration

PWRDWN	Power Down
SGMII	Serial Gigabit Media Independent Interface
ECTL	EMAC interrupt control

TABLE OF CONTENTS

Chapter 1 INTRODUCTION	27
1.1 Introduction	28
1.2 Overview	28
1.3 CSL Interface	28
1.4 Functional Layer	29
1.4.1 CSL Basic Data Types	29
1.4.2 Functional Layer Naming Conventions	29
1.4.3 Symbolic Constants	30
1.4.4 Error Codes	31
1.5 Register Layer	32
1.5.1 Register Layer Naming Conventions	32
1.5.2 Register Overlay Structure	32
1.5.3 Register Layer Symbolic Constants	33
1.5.4 Register Layer Macros	35
1.6 C++ Compatibility	35
CHAPTER 2 CACHE MODULE	36
2.1 Overview	37
2.2 Functions	38
2.2.1 CACHE_enableCaching	38
2.2.2 CACHE_wait	38
2.2.3 CACHE_waitInternal	39
2.2.4 CACHE_freezeL1	39
2.2.5 CACHE_unfreezeL1	40
2.2.6 CACHE_setL1pSize	41
2.2.7 CACHE_freezeL1p	42
2.2.8 CACHE_unfreezeL1p	42
2.2.9 CACHE_invL1p	43
2.2.10 CACHE_invAllL1p	44
2.2.11 CACHE_setL1dSize	45
2.2.12 CACHE_freezeL1d	45
2.2.13 CACHE_unfreezeL1d	46
2.2.14 CACHE_wbL1d	47
2.2.15 CACHE_invL1d	48
2.2.16 CACHE_wbInvL1d	49
2.2.17 CACHE_wbAllL1d	50
2.2.18 CACHE_invAllL1d	50
2.2.19 CACHE_wbInvAllL1d	51
2.2.20 CACHE_setL2Size	52
2.2.21 CACHE_setL2Mode	52
2.2.22 CACHE_wbL2	53
2.2.23 CACHE_invL2	54
2.2.24 CACHE_wbInvL2	55
2.2.25 CACHE_wbAllL2	56
2.2.26 CACHE_invAllL2	57
2.2.27 CACHE_wbInvAllL2	57
2.3 Enumerations	59
2.3.1 CE_MAR	59
2.3.2 CACHE_Wait	59
2.3.3 CACHE_L1_Freeze	60
2.3.4 CACHE_L1Size	60
2.3.5 CACHE_L2Size	60
2.3.6 CACHE_L2Mode	61

2.4	Macros.....	62
CHAPTER 3	DDR2 MODULE.....	63
3.1	Overview	64
3.2	Functions	65
3.2.1	CSL_ddr2Init	65
3.2.2	CSL_ddr2Open	65
3.2.3	CSL_ddr2Close.....	66
3.2.4	CSL_ddr2HwSetup	67
3.2.5	CSL_ddr2GetHwSetup	68
3.2.6	CSL_ddr2HwControl	69
3.2.7	CSL_ddr2GetHwStatus.....	70
3.2.8	CSL_ddr2HwSetupRaw	71
3.2.9	CSL_ddr2GetBaseAddress	72
3.3	Data Structures	74
3.3.1	CSL_Ddr2Obj.....	74
3.3.2	CSL_Ddr2Config.....	74
3.3.3	CSL_Ddr2Context.....	74
3.3.4	CSL_Ddr2Param.....	75
3.3.5	CSL_Ddr2HwSetup.....	75
3.3.6	CSL_Ddr2BaseAddress.....	75
3.3.7	CSL_Ddr2Timing1	76
3.3.8	CSL_Ddr2Timing2	76
3.3.9	CSL_Ddr2Settings	77
3.3.10	CSL_Ddr2ModIdRev.....	77
3.4	Enumerations	78
3.4.1	CSL_Ddr2CasLatency	78
3.4.2	CSL_Ddr2IntBank	78
3.4.3	CSL_Ddr2PageSize.....	78
3.4.4	CSL_Ddr2SelfRefresh	78
3.4.5	CSL_Ddr2HwStatusQuery.....	79
3.4.6	CSL_Ddr2HwControlCmd.....	79
3.5	Macros.....	80
CHAPTER 4	EDMA MODULE	81
4.1	Overview	82
4.2	Functions	83
4.2.1	CSL_edma3Init	83
4.2.2	CSL_edma3Open	83
4.2.3	CSL_edma3Close.....	84
4.2.4	CSL_edma3HwSetup	85
4.2.5	CSL_edma3GetHwSetup	86
4.2.6	CSL_edma3HwControl	88
4.2.7	CSL_edma3GetHwStatus.....	89
4.2.8	CSL_edma3ccGetModuleBaseAddr.....	90
4.2.9	CSL_edma3ChannelOpen.....	91
4.2.10	CSL_edma3ChannelClose	93
4.2.11	CSL_edma3HwChannelSetupParam	94
4.2.12	CSL_edma3HwChannelSetupTriggerWord.....	95
4.2.13	CSL_edma3HwChannelSetupQueue	97
4.2.14	CSL_edma3GetHwChannelSetupParam	98
4.2.15	CSL_edma3GetHwChannelSetupTriggerWord	99
4.2.16	CSL_edma3GetHwChannelSetupQueue	100
4.2.17	CSL_edma3HwChannelControl.....	102
4.2.18	CSL_edma3GetHwChannelStatus	103
4.2.19	CSL_edma3GetParamHandle	105

4.2.20	CSL_edma3ParamSetup	106
4.2.21	CSL_edma3ParamWriteWord	108
4.3	Data Structures	111
4.3.1	CSL_Edma3Obj	111
4.3.2	CSL_Edma3ParamSetup	111
4.3.3	CSL_Edma3ChannelObj	112
4.3.4	CSL_Edma3MemFaultStat	112
4.3.5	CSL_Edma3CtrlErrStat	112
4.3.6	CSL_Edma3QueryInfo	113
4.3.7	CSL_Edma3ActivityStat	113
4.3.8	CSL_Edma3QueStat	113
4.3.9	CSL_Edma3CmdRegion	114
4.3.10	CSL_Edma3CmdQrae	114
4.3.11	CSL_Edma3CmdIntr	115
4.3.12	CSL_Edma3CmdDrae	115
4.3.13	CSL_Edma3CmdQuePri	115
4.3.14	CSL_Edma3CmdQueThr	116
4.3.15	CSL_Edma3ModuleBaseAddress	116
4.3.16	CSL_Edma3ChannelAttr	116
4.3.17	CSL_Edma3ChannelErr	116
4.3.18	CSL_Edma3HwQdmaChannelSetup	117
4.3.19	CSL_Edma3HwDmaChannelSetup	117
4.3.20	CSL_Edma3HwSetup	117
4.4	Enumerations	118
4.4.1	CSL_Edma3QuePri	118
4.4.2	CSL_Edma3QueThr	118
4.4.3	CSL_Edma3HwControlCmd	119
4.4.4	CSL_Edma3HwStatusQuery	120
4.4.5	CSL_Edma3HwChannelControlCmd	120
4.4.6	CSL_Edma3HwChannelStatusQuery	120
4.5	Macros	122
CHAPTER 5 MCBSP MODULE		127
5.1	Overview	128
5.2	Functions	129
5.2.1	CSL_mcbbspInit	129
5.2.2	CSL_mcbbspOpen	129
5.2.3	CSL_mcbbspClose	130
5.2.4	CSL_mcbbspHwSetup	131
5.2.5	CSL_mcbbspHwSetupRaw	132
5.2.6	CSL_mcbbspRead	133
5.2.7	CSL_mcbbspWrite	134
5.2.8	CSL_mcbbspIoWrite	135
5.2.9	CSL_mcbbspIoRead	136
5.2.10	CSL_mcbbspHwControl	137
5.2.11	CSL_mcbbspGetHwStatus	138
5.2.12	CSL_mcbbspGetHwSetup	139
5.2.13	CSL_mcbbspGetBaseAddress	140
5.3	Data Structures	142
5.3.1	CSL_McbbspObj	142
5.3.2	CSL_McbbspConfig	142
5.3.3	CSL_McbbspContext	143
5.3.4	CSL_McbbspHwSetup	143
5.3.5	CSL_McbbspParam	144
5.3.6	CSL_McbbspBaseAddress	144
5.3.7	CSL_McbbspBlkAssign	144

5.3.8	CSL_McbspChanControl	144
5.3.9	CSL_McbspDataSetup	145
5.3.10	CSL_McbspClkSetup	145
5.3.11	CSL_McbspGlobalSetup	146
5.3.12	CSL_McbspMulChSetup	147
5.3.13	CSL_McbspPerild	147
5.4	Enumerations	148
5.4.1	CSL_McbspWordLen	148
5.4.2	CSL_McbspCompand	148
5.4.3	CSL_McbspDataDelay	148
5.4.4	CSL_McbspIntMode	148
5.4.5	CSL_McbspFsClkMode	149
5.4.6	CSL_McbspTxRxClkMode	149
5.4.7	CSL_McbspFsPol	149
5.4.8	CSL_McbspClkPol	149
5.4.9	CSL_McbspSrgClk	149
5.4.10	CSL_McbspTxFsMode	150
5.4.11	CSL_McbspIOMode	150
5.4.12	CSL_McbspClkStp	150
5.4.13	CSL_McbspPartMode	150
5.4.14	CSL_McbspPABlk	151
5.4.15	CSL_McbspPBBlk	151
5.4.16	CSL_McbspEmu	151
5.4.17	CSL_McbspPartition	151
5.4.18	CSL_McbspBlock	152
5.4.19	CSL_McbspChCtrl	152
5.4.20	CSL_McbspChType	152
5.4.21	CSL_McbspDlbMode	152
5.4.22	CSL_McbspPhase	153
5.4.23	CSL_McbspFrmSync	153
5.4.24	CSL_McbspRjustDxena	153
5.4.25	CSL_McbspClkgSyncMode	153
5.4.26	CSL_McbspRstStat	153
5.4.27	CSL_McbspBitReversal	154
5.4.28	CSL_McbspControlCmd	154
5.4.29	CSL_McbspHwStatusQuery	155
5.5	Macros	156
CHAPTER 6	TCP2 MODULE	160
6.1	Overview	161
6.2	Functions	162
6.2.1	TCP2_setParams	162
6.2.2	TCP2_tailConfig	163
6.2.3	TCP2_genlc	165
6.2.4	TCP2_genParams	166
6.2.5	TCP2_calcSubBlocksSA	167
6.2.6	TCP2_calcSubBlocksSP	169
6.2.7	TCP2_tailConfig3GPP	170
6.2.8	TCP2_tailConfigIs2000	172
6.2.9	TCP2_deinterleaveExt	173
6.2.10	TCP2_interleaveExt	174
6.2.11	TCP2_depunctInputs	175
6.2.12	TCP2_calculateHd	176
6.2.13	TCP2_demuxInput	177
6.2.14	TCP2_normalCeil	178
6.2.15	TCP2_ceil	179

6.2.16	TCP2_setExtScaling	179
6.2.17	TCP2_makeTailArgs	180
6.2.18	TCP2_getAccessErr	181
6.2.19	TCP2_getErr	181
6.2.20	TCP2_getTcpErrors	182
6.2.21	TCP2_getFrameLenErr	182
6.2.22	TCP2_getProlLenErr	183
6.2.23	TCP2_getSubFrameErr	184
6.2.24	TCP2_getRelLenErr	184
6.2.25	TCP2_getSnrErr	185
6.2.26	TCP2_getInterleaveErr	185
6.2.27	TCP2_getOutParmErr	186
6.2.28	TCP2_getMaxMinErr	187
6.2.29	TCP2_getNumIt	187
6.2.30	TCP2_getSnrM1	188
6.2.31	TCP2_getSnrM2	188
6.2.32	TCP2_getMap	189
6.2.33	TCP2_getMap0Err	189
6.2.34	TCP2_getMap1Err	190
6.2.35	TCP2_statRun	190
6.2.36	TCP2_statError	191
6.2.37	TCP2_statWaitlc	191
6.2.38	TCP2_statWaitInter	192
6.2.39	TCP2_statWaitSysPar	192
6.2.40	TCP2_statWaitApriori	193
6.2.41	TCP2_statWaitExt	193
6.2.42	TCP2_statWaitHardDec	194
6.2.43	TCP2_statWaitOutParm	195
6.2.44	TCP2_statEmuHalt	195
6.2.45	TCP2_statActMap	196
6.2.46	TCP2_statActState	196
6.2.47	TCP2_statActIter	197
6.2.48	TCP2_statSnr	197
6.2.49	TCP2_statCrc	198
6.2.50	TCP2_statTcpState	198
6.2.51	TCP2_getExecStatus	199
6.2.52	TCP2_getExtEndian	199
6.2.53	TCP2_getInterEndian	200
6.2.54	TCP2_getSlpzvss	200
6.2.55	TCP2_getSlpzdvd	201
6.2.56	TCP2_setExtEndian	202
6.2.57	TCP2_setInterEndian	202
6.2.58	TCP2_setNativeEndian	203
6.2.59	TCP2_setPacked32Endian	203
6.2.60	TCP2_start	204
6.2.61	TCP2_debug	204
6.2.62	TCP2_debugStep	205
6.2.63	TCP2_debugComplete	205
6.2.64	TCP2_reset	206
6.2.65	TCP2_setSlpzdvd	206
6.2.66	TCP2_setSlpzvss	207
6.2.67	TCP2_getIcConfig	207
6.2.68	TCP2_icConfig	208
6.2.69	TCP2_icConfigArgs	209
6.3	Data Structures	211

6.3.1	TCP2_Configlc.....	211
6.3.2	TCP2_Params	212
6.3.3	TCP2_BaseParams	213
6.4	Enumerations	215
6.4.1	TCP2_InputSign.....	215
6.4.2	TCP2_OutputOrder.....	215
6.5	Macros.....	216
CHAPTER 7.....	218	
TIMER MODULE.....	218	
7.1	Overview	219
7.2	Functions	220
7.2.1	CSL_tmrInit	220
7.2.2	CSL_tmrOpen	220
7.2.3	CSL_tmrClose.....	221
7.2.4	CSL_tmrHwSetup	222
7.2.5	CSL_tmrHwControl	223
7.2.6	CSL_tmrGetHwStatus.....	224
7.2.7	CSL_tmrHwSetupRaw	225
7.2.8	CSL_tmrGetHwSetup	226
7.2.9	CSL_tmrGetBaseAddress	227
7.3	Data Structures	228
7.3.1	CSL_TmrObj	228
7.3.2	CSL_TmrConfig	228
7.3.3	CSL_TmrContext	229
7.3.4	CSL_TmrParam	229
7.3.5	CSL_TmrHwSetup	229
7.3.6	CSL_TmrBaseAddress	230
7.4	Enumerations	231
7.4.1	CSL_TmrHwControlCmd	231
7.4.2	CSL_TmrHwStatusQuery	232
7.4.3	CSL_TmrIpGate	232
7.4.4	CSL_TmrClksrc.....	232
7.4.5	CSL_TmrEnamode	233
7.4.6	CSL_TmrPulseWidth	233
7.4.7	CSL_TmrClockPulse.....	233
7.4.8	CSL_TmrInvInp.....	233
7.4.9	CSL_TmrInvOutp	233
7.4.10	CSL_TmrMode.....	234
7.4.11	CSL_TmrState	234
7.4.12	CSL_TmrTstat.....	234
7.4.13	CSL_TmrWdflagBitStatus	234
7.5	Macros.....	235
CHAPTER 8.....	236	
DAT MODULE.....	236	
8.1	Overview	237
8.2	Functions	238
8.2.1	DAT_open	238
8.2.2	DAT_close	238
8.2.3	DAT_copy	239
8.2.4	DAT_fill	240
8.2.5	DAT_wait	241
8.2.6	DAT_busy	242
8.2.7	DAT_copy2d	242
8.2.8	DAT_setPriority	244

8.3 Data Structures	245
8.3.1 DAT_Setup	245
8.4 Macros.....	246
CHAPTER 9.....	247
INTC MODULE.....	247
9.1 Overview	248
9.2 Functions	249
9.2.1 CSL_intcInit.....	249
9.2.2 CSL_intcOpen.....	250
9.2.3 CSL_intcClose	251
9.2.4 CSL_intcPlugEventHandler	252
9.2.5 CSL_intcHookIsr	253
9.2.6 CSL_intcHwControl.....	254
9.2.7 CSL_intcGetHwStatus	256
9.2.8 CSL_intcGlobalEnable.....	257
9.2.9 CSL_intcGlobalDisable	258
9.2.10 CSL_intcGlobalRestore	258
9.2.11 CSL_intcGlobalNmiEnable	259
9.2.12 CSL_intcGlobalExcepEnable.....	259
9.2.13 CSL_intcGlobalExtExcepEnable	259
9.2.14 CSL_intcGlobalExcepClear	260
9.2.15 CSL_intcExcepAllEnable	260
9.2.16 CSL_intcExcepAllDisable	261
9.2.17 CSL_intcExcepAllRestore.....	262
9.2.18 CSL_intcExcepAllClear.....	263
9.2.19 CSL_intcExcepAllStatus	263
9.2.20 CSL_intcQueryDropStatus	264
9.3 Data Structures	266
9.3.1 CSL_IntcObj.....	266
9.3.2 CSL_IntcContext.....	266
9.3.3 CSL_IntcEventHandlerRecord.....	266
9.3.4 CSL_IntcDropStatus	267
9.4 Enumerations	268
9.4.1 CSL_IntcVectId	268
9.4.2 CSL_IntcHwControlCmd.....	268
9.4.3 CSL_IntcHwStatusQuery.....	269
9.4.4 CSL_IntcExcepEn.....	269
9.4.5 CSL_IntcExcep	270
9.5 Macros.....	271
CHAPTER 10.....	272
TSC MODULE.....	272
10.1 Overview.....	273
10.2 Functions	274
10.2.1 CSL_tscEnable	274
10.2.2 CSL_tscRead.....	274
CHAPTER 11 GPIO MODULE.....	276
11.1 Overview.....	277
11.2 Functions	278
11.2.1 CSL_gpioInit	278
11.2.2 CSL_gpioOpen	278
11.2.3 CSL_gpioClose	279
11.2.4 CSL_gpioHwSetup.....	280
11.2.5 CSL_gpioHwSetupRaw	281
11.2.6 CSL_gpioGetHwSetup.....	282

11.2.7	CSL_gpioHwControl	282
11.2.8	CSL_gpioGetHwStatus	283
11.2.9	CSL_gpioGetBaseAddress	284
11.3	Data Structures	286
11.3.1	CSL_GpioObj	286
11.3.2	CSL_GpioConfig	286
11.3.3	CSL_GpioContext	287
11.3.4	CSL_GpioParam	287
11.3.5	CSL_GpioHwSetup	287
11.3.6	CSL_GpioBaseAddress	287
11.3.7	CSL_GpioPinConfig	288
11.3.8	CSL_GpioPinData	288
11.4	Enumerations	289
11.4.1	CSL_GpioDirection	289
11.4.2	CSL_GpioTriggerType	289
11.4.3	CSL_GpioHwControlCmd	289
11.4.4	CSL_GpioHwStatusQuery	290
11.5	Macros	291
CHAPTER 12	I2C MODULE	292
12.1	Overview	293
12.2	Functions	294
12.2.1	CSL_i2cInit	294
12.2.2	CSL_i2cOpen	294
12.2.3	CSL_i2cClose	295
12.2.4	CSL_i2cHwSetup	296
12.2.5	CSL_i2cGetHwSetup	297
12.2.6	CSL_i2cHwControl	298
12.2.7	CSL_i2cRead	299
12.2.8	CSL_i2cWrite	300
12.2.9	CSL_i2cHwSetupRaw	301
12.2.10	CSL_i2cGetHwStatus	301
12.2.11	CSL_i2cGetBaseAddress	302
12.3	Data Structures	304
12.3.1	CSL_I2cObj	304
12.3.2	CSL_I2cConfig	304
12.3.3	CSL_I2cContext	305
12.3.4	CSL_I2cParam	305
12.3.5	CSL_I2cClkSetup	305
12.3.6	CSL_I2cHwSetup	306
12.3.7	CSL_I2cBaseAddress	307
12.4	Enumerations	308
12.4.1	CSL_I2cHwStatusQuery	308
12.4.2	CSL_I2cHwControlCmd	309
12.5	Macros	311
CHAPTER 13	IDMA MODULE	314
13.1	Overview	315
13.2	Functions	316
13.2.1	IDMA1_init	316
13.2.2	IDMA1_copy	316
13.2.3	IDMA1_fill	318
13.2.4	IDMA1_getStatus	319
13.2.5	IDMA1_wait	319
13.2.6	IDMA1_setPriority	320
13.2.7	IDMA1_setInt	320

13.2.8	IDMA0_init	321
13.2.9	IDMA0_config	322
13.2.10	IDMA0_configArgs	322
13.2.11	IDMA0_getStatus	323
13.2.12	IDMA0_wait	324
13.2.13	IDMA0_setInt	324
13.3	Data Structures	326
13.3.1	idma1_handle	326
13.3.2	idma0_config	326
13.4	Enumerations	327
13.4.1	IDMA_Chan	327
13.4.2	IDMA_intEn	327
13.4.3	IDMA_priSet	327
CHAPTER 14	VCP2 MODULE	328
14.1	Overview	329
14.2	Functions	330
14.2.1	VCP2_genParams	330
14.2.2	VCP2_genIc	331
14.2.3	VCP2_ceil	332
14.2.4	VCP2_normalCeil	332
14.2.5	VCP2_getBmEndian	333
14.2.6	VCP2_getIcConfig	334
14.2.7	VCP2_getNumInFifo	334
14.2.8	VCP2_getNumOutFifo	335
14.2.9	VCP2_getSdEndian	335
14.2.10	VCP2_getStateIndex	336
14.2.11	VCP2_getYamBit	336
14.2.12	VCP2_getMaxSm	337
14.2.13	VCP2_getMinSm	338
14.2.14	VCP2_icConfig	338
14.2.15	VCP2_icConfigArgs	339
14.2.16	VCP2_setBmEndian	340
14.2.17	VCP2_setNativeEndian	340
14.2.18	VCP2_setPacked32Endian	341
14.2.19	VCP2_setSdEndian	341
14.2.20	VCP2_addPoly	342
14.2.21	VCP2_statError	343
14.2.22	VCP2_statInFifo	343
14.2.23	VCP2_statOutFifo	344
14.2.24	VCP2_statPause	345
14.2.25	VCP2_statRun	345
14.2.26	VCP2_statSymProc	346
14.2.27	VCP2_statWaitIc	346
14.2.28	VCP2_start	347
14.2.29	VCP2_pause	348
14.2.30	VCP2_unpause	348
14.2.31	VCP2_stepTraceback	349
14.2.32	VCP2_reset	349
14.2.33	VCP2_getErrors	350
14.2.34	VCP2_statEmuHalt	350
14.2.35	VCP2_getVssSleepMode	351
14.2.36	VCP2_getVddSleepMode	351
14.2.37	VCP2_setVssSleepMode	352
14.2.38	P2_setVddSleepMode	353
14.2.39	VCP2_emuEnable	353

14.2.40	VCP2_emuDisable	354
14.2.41	VCP2_getId	354
14.3	Data Structures	356
14.3.1	VCP2_ConfigIc	356
14.3.2	VCP2_Params	356
14.3.3	VCP2_BaseParams	358
14.3.4	VCP2_Errors	358
14.3.5	VCP2_Poly	359
14.4	Macros	360
CHAPTER 15		362
CHIP MODULE		362
15.1	Overview	363
15.2	Functions	365
15.2.1	CSL_chipWriteReg	365
15.2.2	CSL_chipReadReg	365
15.3	Enumerations	367
15.3.1	CSL_ChipReg	367
CHAPTER 16		368
EDC MODULE		368
16.1	Overview	369
16.2	Functions	370
16.2.1	CSL_edcEnable	370
16.2.2	CSL_edcDisable	370
16.2.3	CSL_edcSuspend	371
16.2.4	CSL_edcClear	371
16.2.5	CSL_edcGetErrorAddress	372
16.2.6	CSL_edcGetHwStatus	373
16.2.7	CSL_edcPageEnable	374
16.3	Data Structures	375
16.3.1	CSL_EdcAddrInfo	375
16.3.2	CSL_EdcStatusInfo	375
16.4	Enumerations	377
16.4.1	CSL_EdcMem	377
16.4.2	CSL_EdcClrAccessType	377
16.4.3	CSL_EdcHwStatusQuery	377
16.4.4	CSL_EdcEnableStatus	378
16.4.5	CSL_EdcErrorStatus	378
16.4.6	CSL_EdcNumErrors	378
16.4.7	CSL_EdcUmap	379
16.4.8	CSL_EdcAddrL2way	379
16.4.9	CSL_EdcAddrSram	379
CHAPTER 17 PLLC MODULE		380
17.1	Overview	381
17.2	Functions	382
17.2.1	CSL_pllCinit	382
17.2.2	CSL_pllCOpen	382
17.2.3	CSL_pllCClose	383
17.2.4	CSL_pllCHwSetup	384
17.2.5	CSL_pllCHwControl	385
17.2.6	CSL_pllCGetHwStatus	386
17.2.7	CSL_pllCHwSetupRaw	386
17.2.8	CSL_pllCGetHwSetup	387
17.2.9	CSL_pllCGetBaseAddress	388
17.3	Data Structures	390

17.3.1	CSL_PllcObj.....	390
17.3.2	CSL_PllcConfig.....	390
17.3.3	CSL_PllcContext.....	391
17.3.4	CSL_PllcHwSetup.....	391
17.3.5	CSL_PllcParam.....	392
17.3.6	CSL_PllcBaseAddress.....	393
17.3.7	CSL_PllcDivRatio.....	393
17.3.8	CSL_PllcDivideControl.....	393
17.4	Enumerations.....	394
17.4.1	CSL_PllcDivCtrl	394
17.4.2	CSL_PllcHwControlCmd.....	394
17.4.3	CSL_PllcHwStatusQuery.....	395
17.5	Macros	397
CHAPTER 18	SRIO MODULE.....	402
18.1	Overview.....	403
18.2	Functions	404
18.2.1	CSL_srioInit	404
18.2.2	CSL_srioOpen	404
18.2.3	CSL_srioClose	405
18.2.4	CSL_srioHwSetup.....	406
18.2.5	CSL_srioHwControl	407
18.2.6	CSL_srioGetHwStatus	408
18.2.7	CSL_srioHwSetupRaw	409
18.2.8	CSL_srioGetHwSetup.....	410
18.2.9	CSL_srioLsuSetup	410
18.2.10	CSL_srioGetBaseAddress	412
18.3	Data Structures.....	413
18.3.1	CSL_SrioObj	413
18.3.2	CSL_SrioConfig	413
18.3.3	CSL_SrioContext	415
18.3.4	CSL_SrioHwSetup	415
18.3.5	CSL_SrioParam	417
18.3.6	CSL_SrioBaseAddress	417
18.3.7	CSL_SrioCfgLsuRegs.....	417
18.3.8	CSL_SrioCfgPortRegs.....	418
18.3.9	CSL_SrioCfgPortErrorRegs.....	418
18.3.10	CSL_SrioCfgPortOptionRegs.....	418
18.3.11	CSL_SrioControlSetup	419
18.3.12	CSL_SrioDevInfo.....	420
18.3.13	CSL_SrioAssyInfo	420
18.3.14	CSL_SrioCntlSym	420
18.3.15	CSL_SrioLogTrErrInfo.....	421
18.3.16	CSL_SrioPortData.....	421
18.3.17	CSL_SrioPortGenConfig	422
18.3.18	CSL_SrioPortCntlConfig	422
18.3.19	CSL_SrioPortErrConfig	423
18.3.20	CSL_SrioPidNumber	423
18.3.21	CSL_SrioDevIdConfig	424
18.3.22	CSL_SrioBlkEn.....	424
18.3.23	CSL_SrioPktFwdCntl.....	425
18.3.24	CSL_SrioLsuCompStat	425
18.3.25	CSL_SrioLongAddress.....	425
18.3.26	CSL_SrioPortErrCapt	426
18.3.27	CSL_SrioPortWriteCapt	426
18.3.28	CSL_SrioDirectIO_ConfigXfr.....	427

18.3.29	CSL_SrioSerDesPllCfg	427
18.3.30	CSL_SrioSerDesRxCfg	428
18.3.31	CSL_SrioSerDesTxCfg	428
18.4	Enumerations.....	430
18.4.1	CSL_SrioHwControlCmd	430
18.4.2	CSL_SrioHwStatusQuery	432
18.4.3	CSL_SrioPortCaptType	434
18.4.4	CSL_SrioPortNum	434
18.4.5	CSL_SrioDiscoveryTimer	434
18.4.6	CSL_SrioPwTimer	435
18.4.7	CSL_SrioSilenceTimer.....	435
18.4.8	CSL_SrioBusTransPriority.....	436
18.4.9	CSL_SrioClkDiv	436
18.4.10	CSL_SrioTxPriorityWm	436
18.4.11	CSL_SrioAddrSelect	437
18.4.12	CSL_SrioBufMode.....	437
18.4.13	CSL_SrioPortWidthOverride	437
18.4.14	CSL_SrioErrRtBias.....	437
18.4.15	CSL_SrioPortLnkTimeout	438
18.4.16	CSL_SrioCompCode.....	438
18.4.17	CSL_SrioErrRtNum.....	438
18.4.18	CSL_SrioSerDesLoopBandwidth	439
18.4.19	CSL_SrioSerDesPllMply	439
18.4.20	CSL_SrioSerDesLos	439
18.4.21	CSL_SrioSerDesSymAlignment.....	440
18.4.22	CSL_SrioSerDesTermination.....	440
18.4.23	CSL_SrioSerDesRate	440
18.4.24	CSL_SrioSerDesBusWidth	440
18.4.25	CSL_SrioSerDesCommonMode	441
18.4.26	CSL_SrioSerDesSwingCfg	441
18.5	Macros	442
18.6	Typedefs.....	453
CHAPTER 19	BWMNGMT MODULE	454
19.1	Overview.....	455
19.2	Functions	456
19.2.1	CSL_bwmngmtInit.....	456
19.2.2	CSL_bwmngmtOpen.....	456
19.2.3	CSL_bwmngmtClose	457
19.2.4	CSL_bwmngmtHwSetup.....	458
19.2.5	CSL_bwmngmtGetHwSetup.....	460
19.2.6	CSL_bwmngmtHwControl	461
19.2.7	CSL_bwmngmtGetHwStatus	462
19.3	Data Structures.....	464
19.3.1	CSL_BwmngmtObj	464
19.3.2	CSL_BwmngmtHwSetup	464
19.4	Enumerations.....	465
19.4.1	CSL_BwmngmtControlBlocks.....	465
19.4.2	CSL_BwmngmtPriority.....	465
19.4.3	CSL_BwmngmtMaxwait.....	465
19.4.4	CSL_BwmngmtHwStatusQuery.....	466
19.4.5	CSL_BwmngmtHwControlCmd	466
19.5	Macros	467
19.6	Typedefs.....	468
CHAPTER 20	MEMPROT MODULE	469

20.1	Overview	470
20.2	Functions	471
20.2.1	CSL_memprotInit	471
20.2.2	CSL_memprotOpen	471
20.2.3	CSL_memprotClose	472
20.2.4	CSL_memprotHwSetup	473
20.2.5	CSL_memprotGetHwSetup	474
20.2.6	CSL_memprotHwControl	476
20.2.7	CSL_memprotGetHwStatus	477
20.2.8	CSL_memprotGetBaseAddress	479
20.3	Data Structures	481
20.3.1	CSL_MemprotObj	481
20.3.2	CSL_MemprotContext	481
20.3.3	CSL_MemprotHwSetup	481
20.3.4	CSL_MemprotBaseAddress	481
20.3.5	CSL_MemprotFaultStatus	482
20.3.6	CSL_MemprotPageAttr	482
20.3.7	CSL_MemprotParam	482
20.4	Enumerations	483
20.4.1	CSL_MemprotHwStatusQuery	483
20.4.2	CSL_MemprotHwControlCmd	483
20.4.3	CSL_MemprotLockStatus	483
20.5	Macros	484
CHAPTER 21	CFG MODULE	485
21.1	Overview	486
21.2	Functions	487
21.2.1	CSL_cfgInit	487
21.2.2	CSL_cfgOpen	487
21.2.3	CSL_cfgClose	488
21.2.4	CSL_cfgHwControl	489
21.2.5	CSL_cfgGetHwStatus	490
21.2.6	CSL_cfgGetBaseAddress	491
21.3	Data Structures	492
21.3.1	CSL_CfgObj	492
21.3.2	CSL_CfgFaultStatus	492
21.4	Enumerations	493
21.4.1	CSL_CfgHwControlCmd	493
21.4.2	CSL_CfgHwStatusQuery	493
21.5	Macros	494
21.6	Typedefs	495
CHAPTER 22	PWRDWN MODULE	496
22.1	Overview	497
22.2	Functions	498
22.2.1	CSL_pwrdownInit	498
22.2.2	CSL_pwrdownOpen	498
22.2.3	CSL_pwrdownClose	499
22.2.4	CSL_pwrdownHwSetup	500
22.2.5	CSL_pwrdownGetHwSetup	501
22.2.6	CSL_pwrdownGetHwStatus	502
22.2.7	CSL_pwrdownHwSetupRaw	503
22.2.8	CSL_pwrdownGetBaseAddress	504
22.2.9	CSL_pwrdownHwControl	505
22.3	Data Structures	507
22.3.1	CSL_PwrdownObj	507

22.3.2	CSL_PwrdownConfig.....	507
22.3.3	CSL_PwrdownContext.....	507
22.3.4	CSL_PwrdownHwSetup.....	508
22.3.5	CSL_PwrdownParam.....	508
22.3.6	CSL_PwrdownBaseAddress.....	508
22.3.7	CSL_PwrdownPortData.....	508
22.3.8	CSL_PwrdownL2Manual.....	509
22.4	Enumerations.....	510
22.4.1	CSL_PwrdownHwStatusQuery.....	510
22.4.2	CSL_PwrdownHwControlCmd.....	510
Chapter 23 DTF Module.....		dxix
23.1	Overview.....	512
23.2	Functions.....	513
23.2.1	CSL_dtfInit.....	513
23.2.2	CSL_dtfOpen.....	513
23.2.3	CSL_dtfClose.....	514
23.2.4	CSL_dtfHwControl.....	515
23.2.5	CSL_dtfGetHwStatus.....	516
23.2.6	CSL_GetBaseAddress.....	517
23.3	Data Structures.....	519
23.3.1	CSL_DtfBaseAddress.....	519
23.3.2	CSL_DtfContext.....	519
23.3.3	CSL_DtfObj.....	519
23.3.4	CSL_DtfParam.....	519
23.4	Enumerations.....	520
23.4.1	CSL_DtfControlCmd.....	520
23.4.2	CSL_DtfHwStatusQuery.....	521
23.5	Macros.....	522
23.6	Typedefs.....	523
Chapter 24 ETB Module.....		dxixv
24.1	Overview.....	525
24.2	Functions.....	526
24.2.1	CSL_etbInit.....	526
24.2.2	CSL_etbOpen.....	526
24.2.3	CSL_etbclose.....	527
24.2.4	CSL_etbHwControl.....	528
24.2.5	CSL_etbGetHwStatus.....	529
24.2.6	CSL_etbRead.....	530
24.2.7	CSL_etbWrite.....	531
24.2.8	CSL_etbGetBaseAddress.....	532
24.3	Data Structures.....	534
24.3.1	CSL_EtbBaseAddress.....	534
24.3.2	CSL_EtbContext.....	534
24.3.3	CSL_EtbObj.....	534
24.3.4	CSL_EtbParam.....	534
24.4	Enumerations.....	535
24.4.1	CSL_EtbControlCmd.....	535
24.4.2	CSL_EtbHwStatusQuery.....	536
24.5	Macros.....	539
24.6	Typedefs.....	540
Chapter 25 CIC Module.....		541
25.1	Overview.....	542
25.2	Functions.....	543
25.2.1	CSL_cicInit.....	543

25.2.2	CSL_cicOpen	543
25.2.3	CSL_cicClose	544
25.2.4	CSL_cicGetHwStatus	545
25.2.5	CSL_cicHwControl	546
25.3	Data Structures	548
25.3.1	CSL_CicContext	548
25.3.2	CSL_CicObj	548
25.3.3	CSL_CicBaseAddress	548
25.3.4	CSL_CicParam	548
25.4	Enumerations	549
25.4.1	CSL_CicEctlEvtId	549
25.4.2	CSL_CicHwControlCmd	549
25.4.3	CSL_CicHwStatusQuery	550
25.5	Macros	551
25.6	Typedefs	552
Chapter 26	FSYNC Module	553
26.1	Overview	554
26.2	Functions	555
26.2.1	CSL_fsyncClose	555
26.2.2	CSL_fsyncHwControl	555
26.2.3	CSL_fsyncGetHwStatus	557
26.2.4	CSL_fsyncHwSetup	558
26.2.5	CSL_fsyncGetBaseAddress	560
26.3	Data Structures	561
26.3.1	CSL_FsyncBaseAddress	561
26.3.2	CSL_FsyncCounterTriggerGenObj	561
26.3.3	CSL_FsyncErrEventMaskObj	561
26.3.4	CSL_FsyncMaskTriggerGenObj	561
26.3.5	CSL_FsyncObj	562
26.3.6	CSL_FsyncParam	562
26.3.7	CSL_FsyncRp1PayloadObj	562
26.3.8	CSL_FsyncSetup	563
26.3.9	CSL_FsyncTimerCountObj	564
26.3.10	CSL_FsyncTimerInitObj	564
26.3.11	CSL_FsyncTimerTermCountObj	564
26.3.12	CSL_FsyncTriggerCompareObj	565
26.3.13	CSL_FsyncTriggerMaskObj	565
26.3.14	CSL_FsyncTriggerOffsetObj	566
26.3.15	CSL_FsyncWatchDogObj	566
26.4	Enumerations	567
26.4.1	CSL_FsyncTimerType	567
26.4.2	CSL_FsyncTimerSyncSource	567
26.4.3	CSL_FsyncTimerClkSource	567
26.4.4	CSL_FsyncTimerSyncMode	567
26.4.5	CSL_FsyncTimerReSyncMode	568
26.4.6	CSL_FsyncRp1CRCUsage	568
26.4.7	CSL_FsyncTodLeapSecsUsage	568
26.4.8	CSL_FsyncRp1CrcPosition	568
26.4.9	CSL_FsyncErrMaskType	569
26.4.10	CSL_FsyncErrAlarmIndex	569
26.4.11	CSL_FsyncTriggerGenNum	569
26.4.12	CSL_FsyncRp1CrcInitValue	570
26.4.13	CSL_FsyncHwControlCmd	571
26.4.14	CSL_FsyncHwStatusQuery	573
26.5	Macros	575

26.6	Typedefs	576
Chapter 27	RAC Module	577
27.1	Overview	578
27.2	Functions	579
27.2.1	void CSL_RAC_BEII_disable	579
27.2.2	CSL_RAC_BEII_enable	579
27.2.3	CSL_RAC_BEII_getInterruptStatus	580
27.2.4	CSL_RAC_BEII_setBetieotMask	580
27.2.5	CSL_RAC_BEII_setBetieobbtRdCrossingMask	580
27.2.6	CSL_RAC_BEII_setBetieodbtRdCrossingMask	581
27.2.7	CSL_RAC_BEII_setBeWatchDogMask	581
27.2.8	CSL_RAC_BEII_setCycleOverflowMask	582
27.2.9	CSL_RAC_BEII_setFeWatchDogMask	582
27.2.10	CSL_RAC_BEII_setFifoOverflowMask	583
27.2.11	CSL_RAC_BEII_setMasterMask	583
27.2.12	CSL_RAC_BEII_setSequencerIdleMask	583
27.2.13	CSL_RAC_BETI_disable	584
27.2.14	CSL_RAC_BETI_enable	584
27.2.15	CSL_RAC_BETI_getEotInterruptStatus	585
27.2.16	CSL_RAC_BETI_getObbtRdCrossingStatus	585
27.2.17	CSL_RAC_BETI_getOdbtRdCrossingStatus	586
27.2.18	CSL_RAC_BETI_getOdbtStatus	586
27.2.19	CSL_RAC_BETI_getStatus	586
27.2.20	CSL_RAC_BETI_getWatchDogInterruptStatus	587
27.2.21	CSL_RAC_BETI_getWatchDogStatus	587
27.2.22	CSL_RAC_BETI_setWatchDog	588
27.2.23	CSL_RAC_Stats_getCfgReadAccess	588
27.2.24	CSL_RAC_Stats_getCfgTotalAccess	589
27.2.25	CSL_RAC_Stats_getCfgWriteAccess	589
27.2.26	CSL_RAC_Stats_getMasterHighPrioAccess	590
27.2.27	CSL_RAC_Stats_getMasterLowPrioAccess	590
27.2.28	CSL_RAC_Stats_getSlaveReadAccess	590
27.2.29	CSL_RAC_Stats_getSlaveTotalAccess	591
27.2.30	CSL_RAC_Stats_getSlaveWriteAccess	591
27.2.31	CSL_RAC_BEII_disable	592
27.2.32	CSL_RAC_BEII_enable	592
27.2.33	CSL_RAC_BEII_getInterruptStatus	593
27.2.34	CSL_RAC_BEII_setBetieotMask	593
27.2.35	CSL_RAC_BEII_setBetieobbtRdCrossingMask	593
27.2.36	CSL_RAC_BEII_setBetieodbtRdCrossingMask	594
27.2.37	CSL_RAC_BEII_setBeWatchDogMask	594
27.2.38	CSL_RAC_BEII_setCycleOverflowMask	595
27.2.39	CSL_RAC_BEII_setFeWatchDogMask	595
27.2.40	CSL_RAC_BEII_setFifoOverflowMask	596
27.2.41	CSL_RAC_BEII_setMasterMask	596
27.2.42	CSL_RAC_BEII_setSequencerIdleMask	597
27.2.43	CSL_RAC_BETI_disable	597
27.2.44	CSL_RAC_BETI_enable	597
27.2.45	CSL_RAC_BETI_getEotInterruptStatus	598
27.2.46	CSL_RAC_BETI_getObbtRdCrossingStatus	598
27.2.47	CSL_RAC_BETI_getOdbtRdCrossingStatus	599
27.2.48	CSL_RAC_BETI_getOdbtStatus	599
27.2.49	CSL_RAC_BETI_getStatus	600
27.2.50	CSL_RAC_BETI_getWatchDogInterruptStatus	600
27.2.51	CSL_RAC_BETI_getWatchDogStatus	601

27.2.52	CSL_RAC_BETI_setWatchDog	601
27.2.53	CSL_RAC_Stats_getCfgReadAccess	601
27.2.54	CSL_RAC_Stats_getCfgTotalAccess	602
27.2.55	CSL_RAC_Stats_getCfgWriteAccess	602
27.2.56	CSL_RAC_Stats_getMasterHighPrioAccess	603
27.2.57	CSL_RAC_Stats_getMasterLowPrioAccess	603
27.2.58	CSL_RAC_Stats_getSlaveReadAccess	604
27.2.59	CSL_RAC_Stats_getSlaveTotalAccess	604
27.2.60	CSL_RAC_Stats_getSlaveWriteAccess	604
27.2.61	CSL_RAC_FE_disable	605
27.2.62	CSL_RAC_FE_enable	605
27.2.63	CSL_RAC_FE_getGccpStatus	606
27.2.64	CSL_RAC_FE_getStatus	606
27.2.65	CSL_RAC_FE_getTimestamp	607
27.2.66	CSL_RAC_FE_getWatchDogInterruptStatus	607
27.2.67	CSL_RAC_FE_getWatchDogStatus	607
27.2.68	CSL_RAC_FE_setInputBufferDepth	608
27.2.69	CSL_RAC_FE_setMaxCyclesPerIteration	608
27.2.70	CSL_RAC_FE_setTimestamp	609
27.2.71	CSL_RAC_GCCP_disable	609
27.2.72	CSL_RAC_GCCP_enable	610
27.2.73	CSL_RAC_GCCP_getActiveCycles	610
27.2.74	CSL_RAC_GCCP_getCgtEntry	610
27.2.75	CSL_RAC_GCCP_getFifoOverflowStatus	611
27.2.76	CSL_RAC_GCCP_getHighPrioControlLevel	611
27.2.77	CSL_RAC_GCCP_getHighPrioControlWatermark	612
27.2.78	CSL_RAC_GCCP_getHighPrioDataLevel	612
27.2.79	CSL_RAC_GCCP_getHighPrioDataWatermark	612
27.2.80	CSL_RAC_GCCP_getLowPrioControlLevel	613
27.2.81	CSL_RAC_GCCP_getLowPrioControlWatermark	613
27.2.82	CSL_RAC_GCCP_getLowPrioDataLevel	614
27.2.83	CSL_RAC_GCCP_getLowPrioDataWatermark	614
27.2.84	CSL_RAC_GCCP_getPrtEntry	614
27.2.85	CSL_RAC_GCCP_getReadTime	615
27.2.86	CSL_RAC_GCCP_getSequencerCycles	615
27.2.87	CSL_RAC_GCCP_resetHighPriorityQueue	616
27.2.88	CSL_RAC_GCCP_resetLowPriorityQueue	616
27.2.89	CSL_RAC_GCCP_setCgtEntry	617
27.2.90	CSL_RAC_GCCP_setIctEntry	617
27.2.91	CSL_RAC_GCCP_setPrtEntry	618
27.2.92	CSL_RAC_GCCP_disable	618
27.2.93	CSL_RAC_GCCP_enable	619
27.2.94	CSL_RAC_GCCP_getActiveCycles	619
27.2.95	CSL_RAC_GCCP_getCgtEntry	620
27.2.96	CSL_RAC_GCCP_getCycleOverflowStatus	620
27.2.97	CSL_RAC_GCCP_getFifoOverflowStatus	620
27.2.98	CSL_RAC_GCCP_getHighPrioControlLevel	621
27.2.99	CSL_RAC_GCCP_getHighPrioControlWatermark	621
27.2.100	CSL_RAC_GCCP_getHighPrioDataLevel	622
27.2.101	CSL_RAC_GCCP_getHighPrioDataWatermark	622
27.2.102	CSL_RAC_GCCP_getLowPrioControlLevel	622
27.2.103	CSL_RAC_GCCP_getLowPrioControlWatermark	623
27.2.104	CSL_RAC_GCCP_getLowPrioDataLevel	623
27.2.105	CSL_RAC_GCCP_getLowPrioDataWatermark	624
27.2.106	CSL_RAC_GCCP_getPrtEntry	624

27.2.107	CSL_RAC_GCCP_getReadTime	625
27.2.108	CSL_RAC_GCCP_getSequencerCycles	625
27.2.109	CSL_RAC_GCCP_resetHighPriorityQueue	626
27.2.110	CSL_RAC_GCCP_resetLowPriorityQueue	626
27.2.111	CSL_RAC_GCCP_setCgtEntry	626
27.2.112	CSL_RAC_GCCP_setlctEntry	627
27.2.113	CSL_RAC_GCCP_setPrtEntry	628
27.3	Data Structures	629
27.3.1	_CSL_RAC_FE_Timestamp_req	629
27.3.2	_CSL_RAC_GCCP_cycleOverflow	629
27.3.3	_CSL_RAC_GCCP_fifoOverflowStatus	629
27.3.4	CSL_RAC_BEII_interruptStatus	630
27.4	Enumerations	631
27.4.1	CSL_RAC_BEII_interrupt	631
27.4.2	CSL_RAC_BETI_odbtStatusBit	631
27.4.3	CSL_RAC_BETI_readParamsUpdateStatus	631
27.4.4	CSL_RAC_BETI_statusBit	631
27.4.5	CSL_RAC_BETI_wdInterruptStatus	631
27.4.6	CSL_RAC_FE_gccpStatus	632
27.4.7	CSL_RAC_FE_transferState	632
27.4.8	CSL_RAC_FE_wdInterruptStatus	632
Chapter 28	EMAC Module	633
28.1	Overview	634
28.2	Functions	635
28.2.1	EMAC_close	635
28.2.2	EMAC_enumerate	635
28.2.3	EMAC_getReceiveFilter	636
28.2.4	EMAC_getStatistics	636
28.2.5	EMAC_getStatus	637
28.2.6	EMAC_open	638
28.2.7	EMAC_RxServiceCheck	639
28.2.8	EMAC_sendPacket	641
28.2.9	EMAC_setMulticast	642
28.2.10	EMAC_setReceiveFilter	643
28.2.11	EMAC_timerTick	644
28.2.12	EMAC_txChannelTeardown	645
28.2.13	EMAC_rxChannelTeardown	646
28.2.14	EMAC_TxServiceCheck	646
28.3	Data Structures	648
28.3.1	EMAC_ChannelInfo	648
28.3.2	EMAC_Config	648
28.3.3	EMAC_DescCh	650
28.3.4	EMAC_Device	650
28.3.5	EMAC_Pkt	651
28.3.6	EMAC_Statistics	653
28.3.7	EMAC_Status	655
28.4	Macros	656
28.5	Typedefs	659
Chapter 29	MDIO Module	660
29.1	Overview	661
29.2	Functions	662
29.2.1	MDIO_close	662
29.2.2	MDIO_getStatus	662
29.2.3	MDIO_initPHY	663

29.2.4	MDIO_open.....	664
29.2.5	MDIO_phyRegRead.....	664
29.2.6	MDIO_phyRegWrite.....	665
29.2.7	MDIO_timerTick.....	666
29.2.8	MDIO_initContinue.....	667
29.3	Data Structures.....	668
29.3.1	MDIO_Device.....	668
29.4	Macros.....	669
Chapter 30	SEM Module.....	675
30.1	Overview.....	676
30.2	Functions.....	677
30.2.1	CSL_semInit.....	677
30.2.2	CSL_semOpen.....	677
30.2.3	CSL_semClose.....	678
30.2.4	CSL_semGetHwStatus.....	679
30.2.5	CSL_semHwControl.....	680
30.2.6	CSL_semGetBaseAddress.....	681
30.3	Data Structures.....	682
30.3.1	CSL_SemEOISet_Arg.....	682
30.3.2	CSL_SemErrSet_Arg.....	682
30.3.3	CSL_SemFlagClear_Arg.....	682
30.3.4	CSL_SemFlagSet_Arg.....	682
30.3.5	CSL_SemFaultStatus.....	683
30.3.6	CSL_SemObj.....	683
30.3.7	CSL_SemVal.....	683
30.3.8	CSL_SemContext.....	683
30.3.9	CSL_SemParam.....	684
30.3.10	CSL_SemBaseAddress.....	684
30.4	Enumerations.....	685
30.4.1	CSL_SemError.....	685
30.4.2	CSL_SemFlag.....	685
30.4.3	CSL_SemHwControlCmd.....	685
30.4.4	CSL_SemHwStatusQuery.....	686
30.4.5	CSL_SemOwnerId.....	686
Chapter 31	AIF Module.....	688
31.1	Overview.....	689
31.2	Functions.....	690
31.2.1	CSL_aifInit.....	690
31.2.2	CSL_aifOpen.....	690
31.2.3	CSL_aifClose.....	691
31.2.4	CSL_aifHwSetup.....	692
31.2.5	CSL_aifHwControl.....	693
31.2.6	CSL_aifGetHwStatus.....	694
31.2.7	CSL_aifGetBaseAddress.....	696
31.3	Data Structures.....	697
31.3.1	CSL_AifAggregatorSetup.....	697
31.3.2	CSL_AifAggregatorStatus.....	697
31.3.3	CSL_AifBaseAddress.....	697
31.3.4	CSL_AifCdSetup.....	697
31.3.5	CSL_AifCombinerSetup.....	698
31.3.6	CSL_AifCommonLinkSetup.....	698
31.3.7	CSL_AifCpriTxMacSetup.....	698
31.3.8	CSL_AifDbFifoPtrStatus.....	699
31.3.9	CSL_AifDecombinerSetup.....	699

31.3.10	CSL_AifExcEventCmnMaskObj	699
31.3.11	CSL_AifExcEventLinkClearObj	700
31.3.12	CSL_AifExcEventLinkMaskObj	700
31.3.13	CSL_AifExcEventMode	700
31.3.14	CSL_AifExcEventQueryObj	700
31.3.15	CSL_AifExcLinkSelect	701
31.3.16	CSL_AifGlobalSetup	701
31.3.17	CSL_AifInboundFifoSetup	701
31.3.18	CSL_AifInboundLinkSetup	702
31.3.19	CSL_AifIntMemStruct	702
31.3.20	CSL_AifLinkObj	702
31.3.21	CSL_AifLinkSetup	702
31.3.22	CSL_AifOutboundLinkSetup	703
31.3.23	CSL_AifParam	703
31.3.24	CSL_AifPdCommonSetup	703
31.3.25	CSL_AifPdSetup	704
31.3.26	CSL_AifPeSetup	704
31.3.27	CSL_AifPidStatus	705
31.3.28	CSL_AifRxMacCommonSetup	706
31.3.29	CSL_AifRxMacSetup	706
31.3.30	CSL_AifSerdesCommonSetup	706
31.3.31	CSL_AifSerdesSetup	707
31.3.32	CSL_AifTxMacSetup	707
31.4	Enumerations	709
31.4.1	CSL_AifLinkProtocol	709
31.4.2	CSL_AifFrameStructure	709
31.4.3	CSL_AifAntDataWidth	709
31.4.4	CSL_AifLinkIndex	709
31.4.5	CSL_AifLinkRate	710
31.4.6	CSL_AifRxSyncState	710
31.4.7	CSL_AifTxSyncState	710
31.4.8	CSL_AifLinkDataType	710
31.4.9	CSL_AifLinkFormatType	711
31.4.10	CSL_AifCombinerIndex	711
31.4.11	CSL_AifDecombinerIndex	711
31.4.12	CSL_AifCombinerInput	711
31.4.13	CSL_AifDecombinerDest	712
31.4.14	CSL_AifAggregatorMode	712
31.4.15	CSL_AifAggrSrcSel	712
31.4.16	CSL_AifCpriCtrlWMode	712
31.4.17	CSL_AifInbndPsFifoEventDepth	713
31.4.18	CSL_AifOutboundFifoIndex	714
31.4.19	CSL_AifSerdesIndex	714
31.4.20	CSL_AifSerdesRxTerm	714
31.4.21	CSL_AifSerdesRxEqConfig	715
31.4.22	CSL_AifSerdesRxPairPolarity	715
31.4.23	CSL_AifSerdesRxCdrAlg	715
31.4.24	CSL_AifSerdesTxCommonMode	716
31.4.25	CSL_AifSerdesTxPairPolarity	716
31.4.26	CSL_AifSerdesTxAmpConfig	716
31.4.27	CSL_AifSerdesTxDeConfig	717
31.4.28	CSL_AifPIIMpyFactor	717
31.4.29	CSL_AifExcEventIndex	718
31.4.30	CSL_AifRxSetSyncState	718
31.4.31	CSL_AifTxSetSyncState	718

31.4.32	CSL_AifHwControlCmd	719
31.4.33	CSL_AifHwStatusQuery	723
31.5	Macros	727
31.6	Typedefs	735
Chapter 32	SGMII MODULE	736
32.1	Overview	737
32.2	Functions	738
32.2.1	SGMII_clearDiagnostics	738
32.2.2	SGMII_config	738
32.2.3	SGMII_getAnErrorStatus	739
32.2.4	SGMII_getLinkPartnerStatus	740
32.2.5	SGMII_getStatus	740
32.2.6	SGMII_getStatusReg	741
32.2.7	SGMII_reset	742
32.3	Data Structures	743
32.3.1	SGMII_Config	743
32.3.2	SGMII_Status	743
32.4	Macros	745
Chapter 33	ECTL MODULE	746
33.1	Overview	747
33.2	Functions	748
33.2.1	ECTL_config	748
33.2.2	ECTL_getStatus	748
33.3	Data Structures	750
33.3.1	ECTL_Config	750
33.3.2	ECTL_Status	750
33.4	Macros	752

Chapter 1 INTRODUCTION

Topics

<u>1. 1 Introduction</u>
<u>1. 2 Overview</u>
<u>1. 3 CSL Interface</u>
<u>1. 4 Functional Layer</u>
<u>1. 5 Register Layer</u>
<u>1.6 C++ Compatibility</u>

1.1 Introduction

The Chip Support Layer constitutes a set of well-defined API that abstracts low-level details of the underlying SoC device so that user can configure, control (start/stop etc.) and have read/write access to peripherals without having to worry about register bit-field details.

The CSL services are implemented as distinct modules that correspond with the underlying SoC device modules themselves. By design, CSL APIs follow a consistent style, uniformly across Processor Instruction Set Architecture and are independent of OS. This helps in improving portability of code written using CSL.

1.2 Overview

CSL is realized as twin-layer – a basic register-layer and a more abstracted functional-layer. The lower register layer comprises of a very basic set of macros and type definitions. The upper functional layer comprises of “C” functions that provide an increased degree of abstraction, but intended to provide “directed” control of underlying hardware.

It is important to note that CSL does not manage data-movement over underlying h/w devices. Such functionality is considered a prerogative of a device-driver and serious effort is made to not blur the boundary between device-driver and CSL services in this regard.

CSL does not model the device state machine. However, should there exist a mandatory (hardware dictated) sequence (possibly atomically executed) of register reads/writes to setup the device in chosen “operating modes” as per the device datasheet, then CSL does indeed support services for such operations.

The CSL services are decomposed into modules, each following the twin-layer of abstraction described above. The APIs of each such module are completely orthogonal (one module’s API does not internally call API of another module) and do not allocate memory dynamically from within. This is key to keeping CSL scalable to fit the specific usage scenarios and ease the effort to ROM a CSL based application.

1.3 CSL Interface

CSL is organized into modules by peripheral. Each module contains a twin-layer user interface, the register layer and the functional layer.

The register layer header file for a peripheral <module> is provided in a header file called `cslr_<module>.h`. The functional layer header file for a given peripheral <module> is provided in a header file called `csf_<module>.h`.

In addition to modules for individual peripherals, CSL provides some chip-level modules that perform system and device-level services. These modules are described in the table below.

Module	Description
CHIP	Contains the generic device-specific information that is not specific to a peripheral or module. It includes the chip register IDs, field definitions, register read and write functions,
VERSION	Provides for version management, such as chip ID and version ID.
INTC	The interrupt module provides interrupt management services and a dispatcher. This module is delivered as a separate library.

Table 1: Chip Level Modules

These modules follow the same naming convention for header files.

1.4 Functional Layer

The CSL Functional Layer for TMS320TCI6488 is provided as a mix of CSL 3.x style and CSL 2.x style recommended application programmer's interface to the peripheral. To take advantage of hardware abstraction and maintain maximum forward compatibility in the future, users are encouraged to make use of the Functional Layer APIs in their application and driver code.

Interface functions exported by this layer are "run to completion", meaning, they shall not support asynchronous behavior or deferred completion. If the peripheral hardware has ability to initiate a transaction and assert its completion at a later point in time via designated CPU interrupts, the same should be accommodated by higher-level software (typically device drivers). In general, CSL APIs do not perform resource management or memory allocation; this is managed by the application code or device drivers.

1.4.1 CSL Basic Data Types

The following basic data types are defined in CSL

Type	Defined as.
Bool	Unsigned short
Int	int
Char	char
String	char *
Ptr	void *
UInt32	unsigned int
UInt16	unsigned short
UInt8	unsigned char
Int32	int
Int16	short
Int8	char
CSL_BitMask16	UInt16
CSL_BitMask32	UInt32
CSL_Reg16	volatile UInt16
CSL_Reg32	volatile UInt32
CSL_Status	Int16

Table 2: CSL Basic Data Types

1.4.2 Functional Layer Naming Conventions

The CSL reserved names fall into two categories, those that are **declared** (ex: Functions, variables and so on) and those that are **symbolic constants** and **macros** that are implemented via enum or #defines. The declarative names should strictly be avoided from redefining by user. The #defines however, are open for redefinition via the standard C supported #undef construct. Regardless, user is encouraged not to redefine/conflict with CSL Namespace, as side effects are hard to predict.

The following table illustrates the CSL naming conventions:

Format	Namespace	Type
CSL_<MODULE>_<STRING>	Symbolic constant specified as either a #define or an	CSL_INTC_EVENTID_CNT Here INTC is <MODULE>

	enum. The entire name must be in upper case the <MODULE> denotes peripheral module name and the <STRING> denotes any name, representative of the item being specified or defined. The <STRING> part can have one or more underscores embedded for improved readability.	
CSL_<PeriTitleCaseName>	Peripheral module data type. The CSL_ prefix will be in upper case. The module name string is capitalized and follows title case convention without any underscores. Upper case is used to denote start of a new word or phrase.	CSL_TimerObj CSL_IntcEventId CSL_I2cHwSetup CSL_UartHandle

Table 3: CSL 3.x naming conventions

Format	Namespace	Type
<MODULE>_<STRING>	Symbolic constant specified as either a #define or an enum. The entire name must be in upper case the <MODULE> denotes peripheral module name and the <STRING> denotes any name, representative of the item being specified or defined. The <STRING> part can have one or more underscores embedded for improved readability.	MCBSP_RCV_START Here MCBSP is <MODULE>
<MODULE>_<TitleCaseName>	Peripheral module data type. The <MODULE> prefix will be in upper case. The name string is capitalized and follows title case convention without any underscores. Upper case is used to denote start of a new word or phrase.	MCBSP_Obj HPI_Config

Table 4: CSL 2.x naming conventions

1.4.3 Symbolic Constants

This section documents the symbolic (#define) constants that constitute part of published CSL APIs. The table only lists the common symbols that are applicable to all peripheral modules. However, there exists a whole host of symbolic constants that are very specific to each particular module and are **not** listed here.

Name	Description
CSL_<MODULE>_<n>_REGS	Base address of hardware registers for instance <n> of said peripheral module. Ex: CSL_TIMER_1_REGS for TimeR
CSL_<MODULE>_CHA<m>_REGS	Base address of hardware registers for channel <m> of peripheral, for modules that do support multiple channels or resources.

Table 5: Symbolic constants naming conventions

1.4.4 Error Codes

The CSL3.x will extend minimal support for error handling. Essentially, CSL will only report success or failure of APIs via their return types and/or separate status parameter passed to the call itself.

The error codes are 16bit signed binary numbers that allows us to represent 32 K unique errors. The entire space is divided into 1024 groups, each of size 32. The first group is reserved for CSL generic system errors, the second through last are distributed amongst individual CSL modules. A positive number is regarded as OK status and/or successful operation of a CSL API. All error states are represented as negative integers only.

The following table documents the base set of CSL error codes, **not** specific to any given peripheral.

Error Code	Number	Description
CSL_SOK	+1	Success
CSL_ESYS_FAIL	-1	Generic failure
CSL_ESYS_INUSE	-2	Peripheral resource is already in use
CSL_ESYS_XIO	-3	Encountered a shared I/O (XIO) pin conflict
CSL_ESYS_OVFL	-4	Encountered CSL system resource overflow
CSL_ESYS_BADHANDLE	-5	Handle passed to CSL was invalid
CSL_ESYS_INVPARAMS	-6	Invalid parameters.
CSL_ESYS_INVCMD	-7	Command passed to the CSL was invalid.
CSL_ESYS_INVQUERY	-8	Query passed to the CSL was invalid.
CSL_ESYS_NOTSUPPORTED	-9	Action not supported by CSL.
CSL_ESYS_ALREADY_INITIALIZED	-10	Module already initialized

Table 6: Common error codes

1.5 Register Layer

1.5.1 Register Layer Naming Conventions

All names are alphanumeric except for use of underscores as delimiters.

Convention	Description
CSL Module Identifiers	CSL_<MOD>_ID, where <MOD> is name of the CSL module for a specific peripheral Ex: CSL_TIMER_ID, CSL_MCBSP_ID
Peripheral Instance Identifiers	CSL_<MOD>_<NUM>, where <NUM> is Instance number as per Device Data Sheet or Peripheral Reference Guide Ex: CSL_TIMER_1, CSL_MCBSP_1, CSL_I2C_1 For CSL modules, which have single instance, the convention followed is CSL_<MOD> Ex: CSL_DMA, CSL_GPIO
Peripheral Instance Count Identifiers	CSL_<MOD>_CNT, where <MOD> is peripheral module whose number of instances is defined Ex: CSL_EMIFA_CNT, CSL_HPI_CNT
Peripheral Register Identifiers	<MOD>_<REG>, where <REG> is register name. Names used with specific peripheral instance overlays. Ex: TIMER_CNTL, CPMAC_TX_CONTROL, CPMAC_RX_CONTROL
Peripheral Register Continuous Bit-field Identifiers	<MOD>_<REG>_<FIELD>, where <FIELD> is bit-field name. Names are instance-dependent. Ex: TIMER_CNTL_CLKEN, CPMAC_TX_CONTROL_EN
Peripheral Register Bit-field Symbols	<MOD>_<REG>_<FIELD>_<SYM>, where <SYM> is bit-field setting symbolic token Names are instance-dependent. Ex: CPMAC_TX_CONTROL_EN_RESETVAL, TIMER_CNTL_CLKEN_SHIFT

Table 7: Register layer naming conventions

1.5.2 Register Overlay Structure

SoC peripherals are typically programmed by reading/writing to one or more registers in the peripheral's IO address space. In order to allow for clean and intuitive access to all the registers belonging to a given peripheral instance, CSL implements a technique called Register Overlay Structure. A C data structure template is defined with structure members corresponding to each of the registers of the peripheral device in the order in which they occur. The member types are

chosen to correspond to the widths of the register they represent. Appropriate padding is introduced to ensure alignment for proper addressing of these registers from “C”. The structure members use names that correspond to those used in the peripheral datasheet, to ease programming. Since there exists a well-formed C structure, the registers can be viewed in IDE watch windows and presumably recognized by smart-editors that can do auto-completion while typing.

It should be noted that register overlays do not consume memory, as they are not instantiated. The purpose of these structures is mainly to typify the “C” pointers

Example:

The figure below shows the layout of a TIMER peripheral device. Assuming there exist two instances of the device, one at address 0x01940000 and the other at address 0x01944000, the register overlay for such a device is specified as follows:

```
typedef struct {
    Uint32 CTL; /* Timer Control Register */
    Uint32 PRD; /* Timer Period Register */
    Uint32 CNT; /* Timer Counter Register */
} CSL_TimerRegs;
typedef volatile CSL_TimerReg *CSL_TimerRegsOvly
```

CSL_TIMER_0_REGS (0x01940000)	CTL [+0x00]
	PRD [+0x04]
	CNT [+0x08]
CSL_TIMER_1_REGS (0x01944000)	CTL [+0x00]
	PRD [+0x04]
	CNT [+0x08]

Figure 1: Register Layer overlay structure

1.5.3 Register Layer Symbolic Constants

The CSL register layer file for a given peripheral device (cslr_<module>.h) will define certain standard symbols for each peripheral register/bit field. These symbolic constants are declared with the following convention:

Notational convention: CSL_<MODULE>_<REG>_<FIELD>_<SYMBOL>

The semantics of the various parts of the symbolic name is shown in table below:

Convention	Description
<MODULE>	The CSL Peripheral module Ex: TIMER
<REG>	The peripheral device register Ex: CNT
<FIELD>	Bit-field of interest Ex: ST

<**SYMBOL**> Operational symbol for constant being defined Ex: **STOP, START**

Table 8: Symbolic names used in Register layer

Ex: CSL_TIMER_CNT_ST_STOP

The table below summarizes the standard symbols used with register bit fields:

#define Symbolic Constant	Semantics of the Value Assigned
CSL_<MODULE>_<REG>_SHIFT	The number of left shift positions to reach the register bit-field of interest
CSL_<MODULE>_<REG>_MASK	The binary *and* mask useful to extract register bit-field of interest
CSL_<MODULE>_<REG>_RESETVAL	The power-on reset value assumed by the register or bit-field of interest

Table 9: Standard Symbols used with register bit fields

NOTE: The above defines specified in **cslr_<module>.h** have math bit ordering of MSB: LSB and are regardless of what Endian-flips occur as these are read over processor memory busses. Typically, the processor hardware wiring will be such that CPU always gets to read/write its memory mapped peripherals in "Native Endian" format i.e., ready for CPU interpretation. Should there be Endian mismatch between CPU and memory-mapped peripheral, then necessary corrections (Swaps) must be handled outside, before applying the **_MASK, _SHIFT** etc., symbols shown in this file.

1.5.4 Register Layer Macros

Service	Description
CSL_FMK(field, val)	Creates an AND mask of value (val) moved to specified field location.
CSL_FMKT (field, token)	Same as CSL_FMK, but allows predefined symbolic tokens to be used as value.
CSL_FMKR (msb, lsb, val)	Same as CSL_FMK, but allows raw bit positions (msb:lsb) to specify bit-field.
CSL_FEXT(reg, field)	Evaluates the arithmetic value of bits gathered from specified field.
CSL_FEXTR(reg, msb, lsb)	Same as CSL_FEXT, but allows raw bit positions (msb:lsb) to specify bit-field.
CSL_FINS(reg, field, val)	Inserts the specified value (val) at the specified field in the register.
CSL_FINST(reg, field, token)	Same as CSL_FINS, but allows predefined symbolic tokens to be used as value.
CSL_FINSR(reg, msb, lsb, val)	Same as CSL_FINS, but allows raw bit positions (msb:lsb) to specify bit-field.

Table 10: Register Bit Field Manipulation Macro services

1.6 C++ Compatibility

CSL Functional Layer APIs are, for the most part, implemented in C, with small parts implemented in native assembly to work around some difficulties of realizing the same in C. Regardless, the APIs are declared appropriately so as to allow C++ applications to call them. Unlike C++ functions, the CSL APIs will not support specification of default values for formal arguments passed to them.

Also, in places where CSL API semantics require the user to specify function pointers, CSL3.x design does not allow the user to input a C++ function pointer. To work around above limitation, a wrapper function in C, encapsulating the C++ member function needs to be written by the user. This function can be designed to input the class instance as an argument (along with any other parameters that it requires), and invoke the appropriate class member function internally for achieving the desired objectives.

Chapter 2

CACHE MODULE

Topics

<u>2. 1 Overview</u>
<u>2. 2 Functions</u>
<u>2. 3 Enumerations</u>
<u>2. 4 Macros</u>

2.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within CACHE module.

This module use three cache architectures, Level 1 Program (L1P), Level 1 Data (L1D) and Level 2 CACHE architectures, The L1P and L1D can be configured as 0K, 4K, 8K, 16K, and 32K CACHE size. The L2 can be configured as 32KB, 64KB, 128KB, and 256KB CACHE size. This CACHE module supports the Block and Global Coherence Operations.

2.2 Functions

This section lists the functions available in the CACHE module.

2.2.1 CACHE_enableCaching

void CACHE_enableCaching (CE_MAR *mar*)

Description

This function enables caching for the specified block of memory. This is accomplished by setting the PC bit in the appropriate memory attribute register (MAR). By default, caching is disabled for all memory spaces.

Arguments

<i>mar</i>	EMIF range, Specifies a block of external memory to enable caching
------------	--

Return Value

None

Pre Condition

None

Post Condition

Caching for the specified memory range is enabled.

Modifies

MAR registers

Example

```
CACHE_enableCaching (CACHE_EMIFB_CE00);
```

2.2.2 CACHE_wait

void CACHE_wait (void)

Description

This function waits for the previously issued block operations to complete. This does a partial wait i.e. waits for the cache status register to read back as done.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
...
    CACHE_wait();
...
```

2.2.3 CACHE_waitInternal

void CACHE_waitInternal (void)

Description

This function waits for previously issued block operations to complete. This does a partial wait i.e. waits for the cache status register to read back as done.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
...
    CACHE_waitInternal();
...
```

2.2.4 CACHE_freezeL1

CACHE_L1_Freeze CACHE_freezeL1 (void)

Description

This function freezes the L1P and L1D Cache

As per the specification,

1. The new freeze state is programmed in L1DCC, L1PCC
2. The old state is read from the L1DCC, L1PCC from the POPER field.

This latter read accomplishes two things viz. ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value

CACHE_L1_Freeze

-
- CACHE_L1_FREEZE - Old Freeze State of L1 Cache
 - CACHE_L1P_FREEZE - Old Freeze State of L1P Cache
 - CACHE_L1D_FREEZE - Old Freeze State of L1D Cache
 - CACHE_L1_NORMAL - Normal State of L1 Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Freeze L1 cache

Modifies

L1DCC and L1PCC registers

Example

```
...
CACHE_L1_Freeze oldFreezeState ;

oldFreezeState = CACHE_freezeL1();
...
```

2.2.5 CACHE_unfreezeL1

CACHE_L1_Freeze CACHE_unfreezeL1 (void)

Description

This function unfreezes the L1P and L1D Cache.

As per the specification,

1. The new unfreeze state is programmed in L1DCC, L1PCC.
2. The old state is read from the L1DCC, L1PCC from the POPER field.

This latter read accomplishes 2 things viz. ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value

CACHE_L1_Freeze

- CACHE_L1_FREEZE - Old Freeze State of L1 Cache
- CACHE_L1P_FREEZE - Old Freeze State of L1P Cache
- CACHE_L1D_FREEZE - Old Freeze State of L1D Cache
- CACHE_L1_NORMAL - Normal State of L1 Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Unfreeze the L1 cache

Modifies

L1DCC and L1PCC registers

Example

```
...
CACHE_L1_Freeze oldFreezeState;

oldFreezeState = CACHE_unfreezeL1();
...
```

2.2.6 CACHE_setL1pSize

[CACHE_L1Size](#) **CACHE_setL1pSize** ([CACHE_L1Size](#) *newSize*)

Description

This function sets the L1P cache size. The configurable L1P cache sizes are 0K, 4K, 8K, 16K, and 32K.

As per the specification,

1. The new size is programmed in L1PCFG.
2. L1PCFG is read back to ensure it is set.

Arguments

newSize New Cache size to be programmed

Return Value

CACHE_L1Size

- Old size of L1 Cache

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Set L1P cache size

Modifies

L1PCFG register

Example

```
...
CACHE_L1Size oldSize;

oldSize = CACHE_setL1pSize(CACHE_L1_32KCACHE);
...
```

2.2.7 CACHE_freezeL1p

[CACHE L1 Freeze](#) CACHE_freezeL1p (void)

Description

This function freezes L1P Cache .

As per the specification,

1. The new freeze state is programmed in L1PCC.
2. The old state is read from the L1PCC from the POPER field.

This latter read accomplishes two things; viz. ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value CACHE_L1_Freeze

- CACHE_L1P_FREEZE - Old Freeze State of L1P Cache
- CACHE_L1P_NORMAL - Normal State of L1P Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Freeze L1P cache

Modifies

L1PCC register

Example

```
...
CACHE_L1_Freeze oldFreezeState;

oldFreezeState = CACHE_freezeL1p();
...
```

2.2.8 CACHE_unfreezeL1p

[CACHE L1 Freeze](#) CACHE_unfreezeL1p (void)

Description

This function unfreezes L1P Cache.

As per the specification,

1. The normal state is programmed in L1PCC
2. The old state is read from the L1PCC from the POPER field.

This latter read accomplishes 2 things, viz. Ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value CACHE_L1_Freeze

- CACHE_L1P_FREEZE - Old Freeze State of L1P Cache
- CACHE_L1P_NORMAL - Normal State of L1P Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Unfreeze L1P cache

Modifies

L1PCC register

Example

```
...
CACHE_L1_Freeze oldFreezeState;

oldFreezeState = CACHE_unfreezeL1p();
...
```

2.2.9 CACHE_invL1p

```
void CACHE_invL1p          ( void *          blockPtr,
                             Uint32         byteCnt,
                             CACHE\_Wait wait
                             )
```

Description

This function issues an L1P block invalidate command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only.

As per the specification,

1. The start of the range that needs to be invalidated is written into L1PIBAR 2
2. The byte count is programmed in L1PIWC.

Arguments

blockPtr	Start address of range to be invalidated
byteCnt	Number of bytes to be invalidated
wait	Wait flag

CACHE_NOWAIT - return immediately
 CACHE_WAIT - wait until the operation completes
 CACHE_WAITINTERNAL - wait until the relevant
 cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Invalidate L1P cache

Modifies

L1PIBAR and L1PIWC registers

Example

```

...
CACHE_invL1p ((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...

```

2.2.10 CACHE_invAllL1p

void **CACHE_invAllL1p** ([CACHE_Wait](#) *wait*)

Description

This function issues an L1P invalidate all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,

1. The L1PINV is programmed

Arguments

wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion
------	---

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Invalidate all L1P cache

Modifies

L1PINV register

Example

```
...
CACHE_invAllL1p (CACHE_NOWAIT);
...
```

2.2.11 CACHE_setL1dSize

[CACHE_L1Size](#) **CACHE_setL1dSize** ([CACHE_L1Size](#) *newSize*)

Description

This function sets the size of the L1D cache. The configurable L1D cache sizes are 0K, 4K, 8K, 16K, and 32K.

As per the specification,

1. The new size is programmed in L1DCFG
2. L1DCFG is read back to ensure it is set.

Arguments

newSize New size to be programmed

Return Value

CACHE_L1Size

- Old L1D Cache Size

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Set L1D cache size

Modifies

L1DCFG register

Example

```
...
CACHE_L1Size oldSize;

oldSize = CACHE_setL1dSize(CACHE_L1_32KCACHE);
...
```

2.2.12 CACHE_freezeL1d

[CACHE_L1_Freeze](#) **CACHE_freezeL1d** (void)

Description

This function freezes L1D Cache .

As per the specification,

1. The new freeze state is programmed in L1DCC.

2. The old state is read from the L1DCC from the POPER field.
This latter read accomplishes two things; viz. ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value

CACHE_L1_Freeze

- CACHE_L1D_FREEZE - Old Freeze State of L1D Cache
- CACHE_L1D_NORMAL - Normal State of L1D Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Freeze L1D cache

Modifies

L1DCC register

Example

```
...
CACHE_L1_Freeze oldFreezeState;

oldFreezeState = CACHE_freezeL1d();
...
```

2.2.13 CACHE_unfreezeL1d

[CACHE L1 Freeze](#) **CACHE_unfreezeL1d** (void)

Description

This API Unfreezes L1D Cache. As per the specification,

1. The normal state is programmed in L1DCC
2. The old state is read from the L1DCC from the POPER field.

This latter read accomplishes two things; viz. ensuring the new state is programmed as well as reading the old programmed value.

Arguments

None

Return Value CACHE_L1_Freeze

- CACHE_L1D_FREEZE - Old Freeze State of L1D Cache
- CACHE_L1D_NORMAL - Normal State of L1D Cache

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Unfreeze L1D cache

Modifies

L1DCC register

Example

```
...
CACHE_L1_Freeze oldFreezeState;

oldFreezeState = CACHE_unfreezeL1d();
...
```

2.2.14 CACHE_wbL1d

```
void CACHE_wbL1d          ( void *          blockPtr,
                           Uint32          byteCnt,
                           CACHE\_Wait    wait
                           )
```

Description

This function issues an L1D block write back command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument *wait* is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only.

As per the specification,

- a. The start of the range that needs to be written back is programmed into L1DWBAR.
- b. The byte count is programmed in L1DWWC.

Arguments

<code>blockPtr</code>	Start address of range to be written back
<code>byteCnt</code>	Number of bytes to be written back
<code>wait</code>	Wait flag <code>CACHE_NOWAIT</code> - return immediately <code>CACHE_WAIT</code> - wait until the operation completes <code>CACHE_WAITINTERNAL</code> - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Write back L1D cache

Modifies

L1DWWC and L1DWBAR registers

Example

```
...
    CACHE_wbL1d((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...
```

2.2.15 CACHE_invL1d

```
void CACHE_invL1d          ( void *          blockPtr,
                             Uint32         byteCnt,
                             CACHE_Wait    wait
                             )
```

Description

This function issues an L1D block invalidate command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only.

As per the specification,

1. The start of the range that needs to be invalidated is written into L1DIBAR.
2. The byte count is programmed in L1DIWC.

Arguments

blockPtr	Start address of range to be invalidated
byteCnt	Number of bytes to be invalidated
wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Invalidate the L1D cache

Modifies

L1DIWC and L1DIBAR registers

Example


```
...
CACHE_invL1d ((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...
```

2.2.16 CACHE_wbInvL1d

```
void CACHE_wbInvL1d          ( void *      blockPtr,
                               Uint32      byteCnt,
                               CACHE\_Wait wait
                               )
```

Description

This function issues an L1D block write back and invalidate command to the cache controller. If Writeback invalidates range specified in L1D. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only.

As per the specification,

1. The start of the range that needs to be writeback invalidated is programmed into L1DWIBAR.
2. The byte count is programmed in L1DWIWC.

Arguments

blockPtr	Start address of range to be written back invalidated
byteCnt	Number of bytes to be written back invalidated
wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Writeback and invalidate the L1D cache

Modifies

L1DWIWC and L1DWIBAR registers

Example

```
...
CACHE_wbInvL1d ((Uint32*)(0x1000),200,CACHE_NOWAIT);
...
```

2.2.17 CACHE_wbAII1d

void **CACHE_wbAII1d** ([CACHE_Wait](#) *wait*)

Description

This function issues an L1D writeback all command to the cache controller. . If **CACHE_wbAII1d** was also a previous cache operation which is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument *wait* is **CACHE_NOWAIT**, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,

1. The L1DWB is programmed.

Arguments

<i>wait</i>	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion
-------------	---

Return Value

None

Pre Condition

The **CACHE** must be successfully enabled via **CACHE_enableCaching()** before calling this function

Post Condition

Writeback all the L1D cache

Modifies

L1DWB register

Example

```
...
CACHE_wbAII1d (CACHE_NOWAIT);
...
```

2.2.18 CACHE_invAII1d

void **CACHE_invAII1d** ([CACHE_Wait](#) *wait*)

Description

This function issues an L1D invalidate all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument *wait* is **CACHE_NOWAIT**, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,
1. The L1DINV is programmed.

Arguments

wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion
------	--

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Invalidate the all L1D cache

Modifies

L1DINV register

Example

```
...
CACHE_invAllL1d (CACHE_NOWAIT);
...
```

2.2.19 CACHE_wbInvAllL1d

void CACHE_wbInvAllL1d ([CACHE_Wait](#) *wait*)

Description

This function issues an L1D writeback and invalidate all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,
1. The L1DWBINV is programmed.

Arguments

wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion
------	--

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Writeback and invalidate all L1D cache

Modifies

L1DWBINV register

Example

```
...
CACHE_wbInvAllL1d (CACHE_NOWAIT);
...
```

2.2.20 CACHE_setL2Size

[CACHE_L2Size](#) **CACHE_setL2Size** ([CACHE_L2Size](#) *newSize*)

Description

This function sets the L2 Cache size. The configurable L2 cache sizes are 32KB, 64KB, 128KB, and 256KB.

As per the specification,

1. The old size is read from the L2CFG.
2. The new size is programmed in L2CFG.
3. L2CFG is read back to ensure it is set.

Arguments

newSize New memory size to be programmed

Return Value [CACHE_L2Size](#)

- Old L2 Cache Size

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Set the L2 cache size

Modifies

L2CFG register

Example

```
...
CACHE_L2Size oldSize;

oldSize = CACHE_setL2Size(CACHE_32KCACHE);
...
```

2.2.21 CACHE_setL2Mode

[CACHE_L2Mode](#) **CACHE_setL2Mode** ([CACHE_L2Mode](#) *newMode*)

Description

This function sets the L2 Cache mode. The configurable L2 Cache modes are Normal and Freeze mode.

As per the specification,

1. The old mode is read from the L2CFG.
2. The new mode is programmed in L2CFG.
3. L2CFG is read back to ensure it is set.

Arguments

`newMode` New mode to be programmed

Return Value `CACHE_L2Mode`

- Old Mode set for L2

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Set L2 cache mode

Modifies

L2CFG register

Example

```
...
CACHE_L2Mode oldMode;

oldMode = CACHE_setL2Mode(CACHE_L2_NORMAL);
...
```

2.2.22 `CACHE_wbL2`

```
void CACHE_wbL2          ( void *          blockPtr,
                          Uint32         byteCnt,
                          CACHE\_Wait      wait
                          )
```

Description

This function issues an L2 block writeback command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument `wait` is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only. To prevent unintended behavior, `blockPtr` and `byteCnt` should be multiples of the cache line size.

As per the specification,

1. The start of the range that needs to be written back is programmed into L2WBAR.
2. The byte count is programmed in L2WWC

Arguments

<code>blockPtr</code>	Start address of range to be written back
<code>byteCnt</code>	Number of bytes to be written back
<code>wait</code>	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Writeback the L2 cache

Modifies

L2WWC and L2WBAR registers

Example

```
...
CACHE_wbL2((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...
```

2.2.23 CACHE_invL2

```
void CACHE_invL2          ( void *          blockPtr,
                           Uint32          byteCnt,
                           CACHE\_Wait      wait
                           )
```

Description

This function issues an L2 block invalidate command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument `wait` is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed. Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only. To prevent unintended behavior, `blockPtr` and `byteCnt` should be multiples of the cache line size.

As per the specification,

1. The start of the range that needs to be written back is programmed into L2IBAR
2. The byte count is programmed in L2IWC.

Arguments

<code>blockPtr</code>	Start address of range to be invalidated
<code>byteCnt</code>	Number of bytes to be invalidated

wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache status registers indicate completion
------	--

Return Value

None

Pre Condition

The CACHE must be successfully enabled via CACHE_enableCaching() before calling this function

Post Condition

Invalidate the L2 cache

Modifies

L2IBAR and L2IWC registers

Example

```
...
CACHE_invL2((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...
```

2.2.24 CACHE_wbInvL2

```
void CACHE_wbInvL2      ( void *           blockPtr,
                          Uint32          byteCnt,
                          CACHE_Wait    wait
                          )
```

Description

This function issues an L2 block writeback and invalidate command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument wait is CACHE_NOWAIT, then the function returns immediately, regardless of whether the operation has completed.

Although the block size can be specified in number of bytes, the cache controller operates on whole cache lines only. To prevent unintended behavior, blockPtr and byteCnt should be multiples of the cache line size.

As per the specification,

1. The start of the range that needs to be written back is programmed into L2WIBAR
2. The byte count is programmed in L2WIWC.

Arguments

blockPtr	Start address of range to be written back invalidated
byteCnt	Number of bytes to be written back invalidated
wait	Wait flag CACHE_NOWAIT - return immediately CACHE_WAIT - wait until the operation completes CACHE_WAITINTERNAL - wait until the relevant cache

status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Writeback and invalidate the L2 cache

Modifies

L2WIBAR and L2WIWC registers

Example

```
...
CACHE_wbInvL2((Uint32*)(0x1000), 200, CACHE_NOWAIT);
...
```

2.2.25 CACHE_wbAII L2

void **CACHE_wbAII L2** ([CACHE_Wait](#) *wait*)

Description

This function issues an L2 writeback all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument `wait` is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,

1. The L2WB needs to be programmed.

Arguments

<code>wait</code>	Wait flag
	<code>CACHE_NOWAIT</code> - return immediately
	<code>CACHE_WAIT</code> - wait until the operation completes
	<code>CACHE_WAITINTERNAL</code> - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Writeback all L2 cache

Modifies

L2WB register

Example


```
...
CACHE_wbAllL2(CACHE_NOWAIT);
...
```

2.2.26 CACHE_invAllL2

void CACHE_invAllL2 ([CACHE_Wait](#) *wait*)

Description

This function issues an L2 invalidate all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument *wait* is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,

1. The L2INV needs to be programmed.

Arguments

<i>wait</i>	Wait flag <code>CACHE_NOWAIT</code> - return immediately <code>CACHE_WAIT</code> - wait until the operation completes <code>CACHE_WAITINTERNAL</code> - wait until the relevant cache status registers indicate completion
-------------	---

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Invalidate all L2 cache

Modifies

L2INV register

Example

```
...
CACHE_invAllL2(CACHE_NOWAIT);
...
```

2.2.27 CACHE_wbInvAllL2

void CACHE_wbInvAllL2 ([CACHE_Wait](#) *wait*)

Description

This function issues an L2 writeback and invalidate all command to the cache controller. If a previous cache operation is still active, the function waits for its completion before initiating the new operation, in order to prevent lockout of interrupts. If the argument *wait* is `CACHE_NOWAIT`, then the function returns immediately, regardless of whether the operation has completed.

As per the specification,

1. The L2WBINV needs to be programmed.

Arguments

<code>wait</code>	Wait flag
	<code>CACHE_NOWAIT</code> - return immediately
	<code>CACHE_WAIT</code> - wait until the operation completes
	<code>CACHE_WAITINTERNAL</code> - wait until the relevant cache status registers indicate completion

Return Value

None

Pre Condition

The CACHE must be successfully enabled via `CACHE_enableCaching()` before calling this function

Post Condition

Writeback and invalidate all the L2 cache

Modifies

L2WBINV register

Example

```
...  
CACHE_wbInvAllL2(CACHE_NOWAIT);  
...
```

2.3 Enumerations

This section lists the enumerations available in the CACHE module.

2.3.1 CE_MAR

enum CE_MAR

Enumeration for Emif ranges. This is used for setting up the cache ability of the EMIF ranges.

Enumeration values:

CACHE_EMIFA_CE00	EMIF ranges from 0x80000000 – 0x80FFFFFF
CACHE_EMIFA_CE01	EMIF ranges from 0x81000000 – 0x81FFFFFF
CACHE_EMIFA_CE02	EMIF ranges from 0x82000000 – 0x82FFFFFF
CACHE_EMIFA_CE03	EMIF ranges from 0x83000000 – 0x83FFFFFF
CACHE_EMIFA_CE04	EMIF ranges from 0x84000000 – 0x84FFFFFF
CACHE_EMIFA_CE05	EMIF ranges from 0x85000000 – 0x85FFFFFF
CACHE_EMIFA_CE06	EMIF ranges from 0x86000000 – 0x86FFFFFF
CACHE_EMIFA_CE07	EMIF ranges from 0x87000000 – 0x87FFFFFF
CACHE_EMIFA_CE08	EMIF ranges from 0x88000000 – 0x88FFFFFF
CACHE_EMIFA_CE09	EMIF ranges from 0x89000000 – 0x89FFFFFF
CACHE_EMIFA_CE10	EMIF ranges from 0x8A000000 – 0x8AFFFFFF
CACHE_EMIFA_CE11	EMIF ranges from 0x8B000000 – 0x8BFFFFFF
CACHE_EMIFA_CE12	EMIF ranges from 0x8C000000 – 0x8CFFFFFF
CACHE_EMIFA_CE13	EMIF ranges from 0x8D000000 – 0x8DFFFFFF
CACHE_EMIFA_CE14	EMIF ranges from 0x8E000000 – 0x8EFFFFFF
CACHE_EMIFA_CE15	EMIF ranges from 0x8F000000 – 0x8FFFFFFF
CACHE_EMIFA_CE16	EMIF ranges from 0x90000000 – 0x90FFFFFF
CACHE_EMIFA_CE17	EMIF ranges from 0x91000000 – 0x91FFFFFF
CACHE_EMIFA_CE18	EMIF ranges from 0x92000000 – 0x92FFFFFF
CACHE_EMIFA_CE19	EMIF ranges from 0x93000000 – 0x93FFFFFF
CACHE_EMIFA_CE20	EMIF ranges from 0x94000000 – 0x94FFFFFF
CACHE_EMIFA_CE21	EMIF ranges from 0x95000000 – 0x95FFFFFF
CACHE_EMIFA_CE22	EMIF ranges from 0x96000000 – 0x96FFFFFF
CACHE_EMIFA_CE23	EMIF ranges from 0x97000000 – 0x97FFFFFF
CACHE_EMIFA_CE24	EMIF ranges from 0x98000000 – 0x98FFFFFF
CACHE_EMIFA_CE25	EMIF ranges from 0x99000000 – 0x99FFFFFF
CACHE_EMIFA_CE26	EMIF ranges from 0x9A000000 – 0x9AFFFFFF
CACHE_EMIFA_CE27	EMIF ranges from 0x9B000000 – 0x9BFFFFFF
CACHE_EMIFA_CE28	EMIF ranges from 0x9C000000 – 0x9CFFFFFF
CACHE_EMIFA_CE29	EMIF ranges from 0x9D000000 – 0x9DFFFFFF
CACHE_EMIFA_CE30	EMIF ranges from 0x9E000000 – 0x9EFFFFFF
CACHE_EMIFA_CE31	EMIF ranges from 0x9F000000 – 0x9FFFFFFF

2.3.2 CACHE_Wait

enum CACHE_Wait

Enumeration for Cache wait flags.

This is used for specifying whether the cache operations should block till the desired operation is complete.

Enumeration values:

<i>CACHE_NOWAIT</i>	No blocking, the call exits after programming the control registers
<i>CACHE_WAITINTERNAL</i>	Blocking Call, the call exits after the relevant cache status registers indicate completion
<i>CACHE_WAIT</i>	Blocking Call, the call waits not only till the cache status registers indicate completion, but also till a write read is issued to the EMIF registers (if required)

2.3.3 CACHE_L1_Freeze

enum CACHE_L1_Freeze

Enumeration for Cache Freeze flags. This is used for reporting back the current state of the L1.

Enumeration values:

<i>CACHE_L1D_NORMAL</i>	L1D is in Normal State
<i>CACHE_L1D_FREEZE</i>	L1D is in Freeze State
<i>CACHE_L1P_NORMAL</i>	L1P is in Normal State
<i>CACHE_L1P_FREEZE</i>	L1P is in Freeze State
<i>CACHE_L1_NORMAL</i>	L1D, L1P is in Normal State
<i>CACHE_L1_FREEZE</i>	L1D, L1P is in Freeze State

2.3.4 CACHE_L1Size

enum CACHE_L1Size

Enumeration for L1 (P or D) Sizes.

Enumeration values:

<i>CACHE_L1_0KCACHE</i>	No Cache
<i>CACHE_L1_4KCACHE</i>	4KB Cache
<i>CACHE_L1_8KCACHE</i>	8KB Cache
<i>CACHE_L1_16KCACHE</i>	16KB Cache
<i>CACHE_L1_32KCACHE</i>	32KB Cache

2.3.5 CACHE_L2Size

enum CACHE_L2Size

Enumeration for L2 Sizes.

Enumeration values:

<i>CACHE_0KCACHE</i>	No Cache
<i>CACHE_32KCACHE</i>	32KB Cache
<i>CACHE_64KCACHE</i>	64KB Cache
<i>CACHE_128KCACHE</i>	128KB Cache
<i>CACHE_256KCACHE</i>	256KB Cache

2.3.6 CACHE_L2Mode

enum CACHE_L2Mode
Enumeration for L2 Modes.

Enumeration values:

CACHE_L2_NORMAL
CACHE_L2_FREEZE

Enabled/Normal Mode
Freeze Mode

2.4 Macros

#define CACHE_L1D_LINESIZE 64
L1D Line Size

#define CACHE_L1P_LINESIZE 32
L1P Line Size

#define CACHE_L2_LINESIZE 128
L2 Line Size

#define CACHE_ROUND_TO_LINESIZE (CACHE,
ELCNT,
ELSIZE
)

Value:

```
((CACHE_##CACHE##_LINESIZE *
    ((ELCNT)*(ELSIZE)/CACHE_##CACHE##_LINESIZE) + 1) /
    (ELSIZE))
```

Cache Round to Line size

Chapter 3

DDR2 MODULE

Topics

<u>3. 1 Overview</u>

<u>3. 2 Functions</u>

<u>3. 3 Data Structures</u>

<u>3. 4 Enumerations</u>
--

<u>3. 5 Macros</u>

3.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within DDR2 module. This is a 32-bit DDR2 SDRAM interface. The 32-bit DDR2 Memory Controller bus is used to interface to DDR2 devices. The DDR2 external bus only interfaces to DDR2 devices; it does not share the bus with any other types of peripherals.

3.2 Functions

This section lists the functions available in the DDR2 module.

3.2.1 CSL_ddr2Init

CSL_Status CSL_ddr2Init ([CSL_Ddr2Context](#) * *pContext*)

Description

This is the initialization function for the DDR2 CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Pointer to module-context. As DDR2 doesn't have any context based information user is expected to pass NULL.

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

The CSL for DDR2 is initialized.

Modifies

None

Example

```
...
if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}
...
```

3.2.2 CSL_ddr2Open

[CSL_Ddr2Handle](#) CSL_ddr2Open ([CSL_Ddr2Obj](#) * *pDdr2Obj*,
CSL_InstNum *ddr2Num*,
[CSL_Ddr2Param](#) * *pDdr2Param*,
CSL_Status * *pStatus*
)

Description

This function returns the handle to the DDR2 instance. The open call sets up the data structures for the particular instance of DDR2. The handle returned by this call is input argument for rest of the DDR2 CSL APIs.

Arguments

<code>pDdr2Obj</code>	Pointer to the object that holds reference to the instance of DDR2 requested after the call
<code>ddr2Num</code>	Instance of DDR2 to which a handle is requested
<code>pDdr2Param</code>	Pointer to module specific parameters
<code>pStatus</code>	pointer for returning status of the function call

Return Value `CSL_Ddr2Handle`

- Valid DDR2 instance handle will be returned if status value is equal to `CSL_SOK`.

Pre Condition

The DDR2 must be successfully initialized via `CSL_ddr2Init()` before calling this function.

Post Condition

- The status is returned in the status variable. If status is `CSL_SOK` then a valid DDR2 handle is returned. If status is `NULL` then the DDR2 instance is invalid.
- DDR2 object structure is populated.

Modifies

- The status variable
- object structure

Example:

```
CSL_Status      status;
CSL_Ddr2Obj     ddr2Obj;
CSL_Ddr2Handle  hDdr2;

hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
```

3.2.3 CSL_ddr2Close

CSL_Status `CSL_ddr2Close` ([CSL_Ddr2Handle](#) *hDdr2*)

Description

This function closes the specified instance of DDR2.

Arguments

<code>hDdr2</code>	DDR2 handle returned by successful 'open'
--------------------	---

Return Value `CSL_Status`

- `CSL_SOK` - external memory interface close successful
- `CSL_ESYS_BADHANDLE` - The handle passed is invalid

Pre Condition

Both *CSL_ddr2Init()* and *CSL_ddr2Open()* must be called successfully in order before calling *CSL_ddr2Close()*.

Post Condition

The external memory interface CSL APIs can not be called until the external memory interface CSL is reopened again using *CSL_ddr2Open()*.

Modifies

Obj structure values

Example

```

CSL_Status          status;
CSL_Ddr2Obj         ddr2Obj;
CSL_Ddr2Handle      hDdr2;

if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}
hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
...
CSL_ddr2Close(hDdr2);
...

```

3.2.4 CSL_ddr2HwSetup

```

CSL_Status CSL_ddr2HwSetup      ( CSL\_Ddr2Handle          hDdr2,
                                   CSL\_Ddr2HwSetup *         setup
                                   )

```

Description

This function initializes the device registers with the appropriate values provided through the HwSetup data structure. For information passed through the HwSetup data structure, refer *CSL_Ddr2HwSetup*.

Arguments

<i>hDdr2</i>	DDR2 handle returned by successful 'open'
<i>setup</i>	Pointer to setup structure, which contains the information to program DDR2 to a required state

Return Value *CSL_Status*

- *CSL_SOK* – Hwsetup successful
- *CSL_ESYS_BADHANDLE* – Handle passed is invalid
- *CSL_ESYS_INVPARAMS* – The param passed is invalid

Pre Condition Both *CSL_ddr2Init()* and *CSL_ddr2Open()* must be called successfully in order before this function.

Post Condition

DDR2 registers are configured according to the hardware setup parameters.

Modifies

DDR2 registers

Example

```

CSL_Ddr2Handle    hDdr2;
CSL_Status        status;
CSL_Ddr2Obj       ddr2Obj;
CSL_Ddr2Timing1  tim1 = CSL_DDR2_TIMING1_DEFAULTS;
CSL_Ddr2Timing2  tim2 = CSL_DDR2_TIMING2_DEFAULTS;
CSL_Ddr2Settings set = CSL_DDR2_SETTING_DEFAULTS;
CSL_Ddr2HwSetup  hwSetup;

if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}
hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
hwSetup.refreshRate = (Uint16)0x753;
hwSetup.timing1Param = &tim1;
hwSetup.timing2Param = &tim2;
hwSetup.setParam = &set;
CSL_ddr2HwSetup(hDdr2, &hwSetup);

```

3.2.5 CSL_ddr2GetHwSetup

CSL_Status CSL_ddr2GetHwSetup ([CSL_Ddr2Handle](#) *hDdr2*,
[CSL_Ddr2HwSetup](#) * *setup*
)

Description

This function gets the current setup of the DDR2. The status is returned through *CSL_Ddr2HwSetup*. The obtaining of status is the reverse operation of *CSL_ddr2HwSetup()* function.

Arguments

<i>hDdr2</i>	DDR2 handle returned by successful 'open'
<i>setup</i>	Pointer to the hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_INVPARAMS - Param passed is invalid
- CSL_ESYS_BADHANDLE - Handle is not valid

Pre Condition Both *CSL_ddr2Init()* and *CSL_ddr2Open()* must be called successfully in order before calling *CSL_ddr2GetHwSetup()*.

Post Condition

None

Modifies

Second parameter setup value

Example

```

CSL_Ddr2Handle hDdr2;
CSL_Ddr2Obj ddr2Obj;
CSL_Status status;
CSL_Ddr2Timing1 tim1;
CSL_Ddr2Timing2 tim2;
CSL_Ddr2Settings set;
CSL_Ddr2HwSetup hwSetup;

if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}

hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
...

hwSetup.timing1Param = &tim1;
hwSetup.timing2Param = &tim2;
hwSetup.setParam = &set;
...
status = CSL_ddr2GetHwSetup(hDdr2, &hwSetup);

```

3.2.6 CSL_ddr2HwControl

CSL_Status CSL_ddr2HwControl	(<u>CSL_Ddr2Handle</u>	<i>hDdr2,</i>
		<u>CSL_Ddr2HwControlCmd</u>	<i>cmd,</i>
		void *	<i>arg</i>
)		

Description

Control operations for the DDR2. For a particular control operation, the pointer to the corresponding data type needs to be passed as argument HwControl function Call. All the arguments (structure elements included) passed to the HwControl function are inputs. For the list of commands supported and argument type that can be *void** casted and passed with a particular command refer to *CSL_Ddr2HwControlCmd*.

Arguments

hDdr2	DDR2 handle returned by successful 'open'
cmd	The command to this API indicates the action to be taken
arg	Optional argument as per the control command

Return Value CSL_Status

- CSL_SOK - Command successful
- CSL_ESYS_BADHANDLE - Handle passed is invalid
- CSL_ESYS_INVCMD - Command passed is invalid

Pre Condition

Both `CSL_ddr2Init()` and `CSL_ddr2Open()` must be called successfully in order before calling `CSL_ddr2HwControl()`.

Post Condition

DDR2 registers are configured according to the command passed.

Modifies

DDR2 registers

Example

```
CSL_Ddr2Handle    hDdr2;
CSL_Status        status;
CSL_Ddr2Obj       ddr2Obj;

    CSL_Ddr2SelfRefresh command;
    if (CSL_SOK != CSL_ddr2Init(NULL))
    {
        return;
    }
hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
...
command = CSL_DDR2_SELF_REFRESH_DISABLE;
...
status = CSL_ddr2HwControl(    hDdr2,
                              CSL_DDR2_CMD_SELF_REFRESH,
                              &command);
```

3.2.7 CSL_ddr2GetHwStatus

```
CSL_Status CSL_ddr2GetHwStatus    ( CSL\_Ddr2Handle           hDdr2,
                                   CSL\_Ddr2HwStatusQuery  query,
                                   void *                          response
                                   )
```

Description

This function is used to read the current device configuration, status flags and the value present associated registers. For details about the various status queries supported and the associated data structure to record the response, refer to *CSL_Ddr2HwStatusQuery*.

Arguments

<code>hDdr2</code>	DDR2 handle returned by successful 'open'
<code>query</code>	The query to this API, which indicates the status to be returned
<code>response</code>	Response from the query.

Return Value CSL_Status

- `CSL_SOK` - Hardware status call is successful
- `CSL_ESYS_BADHANDLE` - Not a valid Handle

- **CSL_ESYS_INVQUERY** - Invalid Query

Pre Condition Both *CSL_ddr2Init()* and *CSL_ddr2Open()* must be called successfully in order before calling *CSL_ddr2GetHwStatus()*.

Post Condition
None

Modifies
Third parameter, response value

Example:

```
CSL_Ddr2Handle    hDdr2;
CSL_Status        status;
Uint16            response;
CSL_Ddr2Obj       ddr2Obj;

if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}
hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
...
status = CSL_ddr2GetHwStatus(hDdr2,
                             CSL_DDR2_QUERY_REFRESH_RATE,
                             &response);
```

3.2.8 CSL_ddr2HwSetupRaw

CSL_Status CSL_ddr2HwSetupRaw ([CSL_Ddr2Handle](#) *hDdr2*,
[CSL_Ddr2Config](#) * *config*
)

Description

This function initializes the device registers with the register-values provided through the Config data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values.

Arguments

<i>hDdr2</i>	Handle to the DDR2 external memory interface instance
<i>config</i>	Pointer to the config structure containing the device register values

Return Value *CSL_Status*

- **CSL_SOK** - Configuration successful
- **CSL_ESYS_BADHANDLE** - Invalid handle
- **CSL_ESYS_INVPARAMS** - Configuration structure pointer is not properly initialized

Pre Condition

CSL_ddr2Init() and *CSL_ddr2Open ()* must be called successfully before calling this function.

Post Condition

The registers of the specified DDR2 instance will be setup according to the values passed through the Config structure.

Modifies

Hardware registers of the DDR2

Example

```

CSL_Ddr2Handle    hDdr2;
CSL_Ddr2Config    config = CSL_DDR2_CONFIG_DEFAULTS;
CSL_Status        status;
CSL_Ddr2Obj       ddr2Obj;

if (CSL_SOK != CSL_ddr2Init(NULL))
{
    return;
}
hDdr2 = CSL_ddr2Open(&ddr2Obj, CSL_DDR2, NULL, &status);
...
status = CSL_ddr2HwSetupRaw(hDdr2, &config);
...

```

3.2.9 CSL_ddr2GetBaseAddress

```

CSL_Status CSL_ddr2GetBaseAddress ( CSL_InstNum      ddr2Num,
                                   CSL\_Ddr2Param *  pDdr2Param,
                                   CSL\_Ddr2BaseAddress * pBaseAddress
                                   )

```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_ddr2Open() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

ddr2Num	Specifies the instance of the DDR2 external memory interface for which the base address is requested
pDdr2Param	Module specific parameters.
pBaseAddress	Pointer to the base address structure to return the base address details.

Return Value CSL_Status

- CSL_SOK - Successful on getting the base address of DDR2
- CSL_ESYS_FAIL - The external memory interface instance is not available.
- CSL_ESYS_INVPARAMS - Invalid Parameter

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure

Example

```
CSL_Status          status;  
CSL_Ddr2BaseAddress baseAddress;  
...  
status = CSL_ddr2GetBaseAddress(CSL_DDR2, NULL, &baseAddress);
```

3.3 Data Structures

This section lists the data structures available in the DDR2 module.

3.3.1 CSL_Ddr2Obj

Detailed Description

This object contains the reference to the instance of DDR2 opened using the *CSL_ddr2Open()*. The pointer to this is passed to all DDR2 CSL APIs. *CSL_ddr2Open()* function initializes this structure based on the parameters passed.

Field Documentation

CSL_InstNum CSL_Ddr2Obj::perNum

This is the instance of DDR2 being referred to by this object

CSL_Ddr2RegsOvly CSL_Ddr2Obj::regs

Pointer to the register overlay structure of the DDR2

3.3.2 CSL_Ddr2Config

Detailed Description

DDR2 config structure, which is used in *CSL_ddr2HwSetupRaw()* function. This is a structure of register values, rather than a structure of register field values like *CSL_Ddr2HwSetup*.

Field Documentation

volatile Uint32 CSL_Ddr2Config::SDCFG

SDRAM Config Register

volatile Uint32 CSL_Ddr2Config::SDRFC

SDRAM Refresh Control Register

volatile Uint32 CSL_Ddr2Config::SDTIM1

SDRAM Timing1 Register

volatile Uint32 CSL_Ddr2Config::SDTIM2

SDRAM Timing2 Register

volatile Uint32 CSL_Ddr2Config::BPRIO

VBUSM Burst Priority Register

3.3.3 CSL_Ddr2Context

Detailed Description

DDR2 specific context information. Present implementation doesn't have any Context information.

Field Documentation
UInt16 CSL_Ddr2Context::contextInfo

Context information of DDR2 external memory interface CSL passed as an argument to CSL_ddr2Init(). Present implementation of DDR2 CSL doesn't have any context information; hence assigned NULL. The declaration is just a placeholder for future implementation.

3.3.4 CSL_Ddr2Param

Detailed Description

This is module specific parameter. Present implementation of DDR2 CSL doesn't have any module specific parameters.

Field Documentation
CSL_BitMask16 CSL_Ddr2Param::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation. Passed as an argument to CSL_ddr2Open().

3.3.5 CSL_Ddr2HwSetup

Detailed Description

This has all the fields required to configure DDR2 at Power Up (after a Hardware Reset) or a Soft Reset. This structure is used to setup or obtain existing setup of DDR2 using *CSL_ddr2HwSetup()* and *CSL_ddr2GetHwSetup()* functions respectively.

Field Documentation
UInt16 CSL_Ddr2HwSetup::refreshRate

Refresh Rate

[CSL_Ddr2Settings](#)* CSL_Ddr2HwSetup::setParam

Structure for DDR2 SDRAM configuration parameter

[CSL_Ddr2Timing1](#)* CSL_Ddr2HwSetup::timing1Param

Structure for DDR2 SDRAM Timing1

[CSL_Ddr2Timing2](#)* CSL_Ddr2HwSetup::timing2Param

Structure for DDR2 SDRAM Timing2

3.3.6 CSL_Ddr2BaseAddress

Detailed Description

This structure contains the base address information for the DDR2 instance.

Field Documentation
CSL_Ddr2RegsOvly CSL_Ddr2BaseAddress::regs

Base address of the configuration registers of the peripheral

3.3.7 CSL_Ddr2Timing1

Detailed Description

Timing1 structure to set the Timing1 register of DDR2 SDRAM.

Field Documentation

UInt8 CSL_Ddr2Timing1::tras

Specifies TRAS value: Minimum number of DDR2 EMIF cycles from Activate to Pre-charge command, minus one

UInt8 CSL_Ddr2Timing1::trc

Specifies TRC value: Minimum number of DDR2 EMIF cycles from Activate command to Activate command, minus one

UInt8 CSL_Ddr2Timing1::trcd

Specifies TRCD value: Minimum number of DDR2 EMIF cycles from Active to Read or Write command, minus one

UInt8 CSL_Ddr2Timing1::trfc

Specifies TRFC value: Minimum number of DDR2 EMIF cycles from Refresh or Load command to Refresh or Activate command, minus one

UInt8 CSL_Ddr2Timing1::trp

Specifies TRP value: Minimum number of DDR2 EMIF cycles from Pre-charge to Active or Refresh command, minus one

UInt8 CSL_Ddr2Timing1::trrd

Specifies TRRD value: Minimum number of DDR2 EMIF cycles from Activate command to Activate command for a different bank, minus one

UInt8 CSL_Ddr2Timing1::twr

Specifies TWR value: Minimum number of DDR2 EMIF cycles from last write transfer to Pre-charge command, minus one

UInt8 CSL_Ddr2Timing1::twtr

Specifies the minimum number of DDR2 EMIF clock cycles from last DDR Write to DDR Read, minus one

3.3.8 CSL_Ddr2Timing2

Detailed Description

Timing2 structure to set the Timing2 register of DDR2 SDRAM.

Field Documentation

UInt8 CSL_Ddr2Timing2::tcke

Specifies the minimum number of DDR2 EMIF clock cycles between pado_mcke_o changes, minus one.

UInt8 CSL_Ddr2Timing2::trtp

Specifies the minimum number of DDR2 EMIF clock cycles from the last Read command to a Pre-charge command for DDR2 SDRAM, minus one.

UInt8 CSL_Ddr2Timing2::tsxnr

Specifies the minimum number of DDR2 EMIF clock cycles from Self-Refresh exit to any command other than a Read command, minus one.

UInt8 CSL_Ddr2Timing2::tsxrd

Specifies the minimum number of DDR2 EMIF clock cycles from Self-Refresh exit to a Read command for DDR SDRAM, minus one.

UInt8 CSL_Ddr2Timing2::todt

Specifies the minimum number of DDR2 EMIF clock cycles from Self-Refresh exit to any command other than a Read command, minus one.

3.3.9 CSL_Ddr2Settings

Detailed Description

This structure contains the fields to set the DDR2 SDRAM. All fields needed for DDR2 SDRAM settings are present in this structure.

Field Documentation
[CSL_Ddr2CasLatency](#) CSL_Ddr2Settings::casLatncy

CAS Latency

[CSL_Ddr2IntBank](#) CSL_Ddr2Settings::ibank

Defines number of banks inside connected SDRAM devices

[CSL_Ddr2PageSize](#) CSL_Ddr2Settings::pageSize

Defines the internal page size of connected SDRAM devices

CSL_Ddr2Mode CSL_Ddr2Settings::narrowMode

SDRAM data bus width

3.3.10 CSL_Ddr2ModIdRev

Detailed Description

DDR2 Module ID and Revision structure is used for querying the DDR2 module Id and revision.

Field Documentation
UInt8 CSL_Ddr2ModIdRev::majRev

DDR2 EMIF Major Revision

UInt8 CSL_Ddr2ModIdRev::minRev

DDR2 EMIF Minor Revision

UInt16 CSL_Ddr2ModIdRev::modId

DDR2 EMIF Module ID

3.4 Enumerations

This section lists the enumerations available in the DDR2 module.

3.4.1 CSL_Ddr2CasLatency

enum CSL_Ddr2CasLatency

Enumeration for bit field CL of SDRAM Config Register.

Enumeration values:

<i>CSL_DDR2_CAS_LATENCY_2</i>	Cas Latency is 2
<i>CSL_DDR2_CAS_LATENCY_3</i>	Cas Latency is 3
<i>CSL_DDR2_CAS_LATENCY_4</i>	Cas Latency is 4
<i>CSL_DDR2_CAS_LATENCY_5</i>	Cas Latency is 5

3.4.2 CSL_Ddr2IntBank

enum CSL_Ddr2IntBank

Enumeration for bit field ibank of SDRAM Config Register.

Enumeration values:

<i>CSL_DDR2_1_SDRAM_BANKS</i>	DDR2 SDRAM has one internal bank
<i>CSL_DDR2_2_SDRAM_BANKS</i>	DDR2 SDRAM has two internal banks
<i>CSL_DDR2_4_SDRAM_BANKS</i>	DDR2 SDRAM has four internal bank
<i>CSL_DDR2_8_SDRAM_BANKS</i>	DDR2 SDRAM has eight internal banks

3.4.3 CSL_Ddr2PageSize

enum CSL_Ddr2PageSize

Enumeration for bit field pagesize of SDRAM Config Register.

Enumeration values:

<i>CSL_DDR2_256WORD_8COL_ADDR</i>	256-word pages requiring 8 column address bits
<i>CSL_DDR2_512WORD_9COL_ADDR</i>	512-word pages requiring 9 column address bits
<i>CSL_DDR2_1024WORD_10COL_ADDR</i>	1024-word pages requiring 10 column address bits
<i>CSL_DDR2_2048WORD_11COL_ADDR</i>	2048-word pages requiring 11 column address bits

3.4.4 CSL_Ddr2SelfRefresh

enum CSL_Ddr2SelfRefresh

Enumeration for bit field SR of SDRAM Config Register.

Enumeration values:

<i>CSL_DDR2_SELF_REFRESH_DISABLE</i>	Disables Self Refresh on DDR2
<i>CSL_DDR2_SELF_REFRESH_ENABLE</i>	Connected DDR2 SDRAM device will enter Self Refresh Mode and DDR2 EMIF enters Self Refresh State

3.4.5 CSL_Ddr2HwStatusQuery

enum CSL_Ddr2HwStatusQuery

Enumeration for queries passed to *CSL_ddr2GetHwStatus()*.

This is used to get the status of different operations

Enumeration values:

<i>CSL_DDR2_QUERY_REV_ID</i>	Get the DDR2 EMIF module ID and revision numbers (response type: (CSL_Ddr2ModIdRev*))
<i>CSL_DDR2_QUERY_REFRESH_RATE</i>	Get the EMIF refresh rate information (response type: <i>Uint16 *</i>)
<i>CSL_DDR2_QUERY_SELF_REFRESH</i>	Get self refresh bit value (response type: (CSL_Ddr2SelfRefresh *))
<i>CSL_DDR2_QUERY_IFRDY</i>	Reflects the value on the IFRDY_ready port (active high) that defines whether the DDR PHY is ready for normal operation. (Response type: <i>Uint8 *</i>)
<i>CSL_DDR2_QUERY_ENDIAN</i>	Gets the current endian of DDR2 emif from the SDRAM Status register. (response type: @a <i>Uint8 *</i>)

3.4.6 CSL_Ddr2HwControlCmd

enum CSL_Ddr2HwControlCmd

Enumeration for commands passed to *CSL_ddr2HwControl()*.

This is used to select the commands to control the operations existing setup of DDR2. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

<i>CSL_DDR2_CMD_SELF_REFRESH</i>	Self refresh enable or disable based on arg passed: argument (CSL_Ddr2SelfRefresh *)
<i>CSL_DDR2_CMD_REFRESH_RATE</i>	Enters the Refresh rate value: argument (<i>Uint16 *</i>)
<i>CSL_DDR2_CMD_PRIO_RAISE</i>	Number of memory transfers after which the DDR2 EMIF momentarily raises the priority of old commands in the VBUSM Command FIFO. Argument (<i>Uint8 *</i>)

3.5 Macros

#define CSL_DDR2_CONFIG_DEFAULTS

Value:

```
{ \
    CSL_DDR2_SDCFG_DEFAULT,      \
    CSL_DDR2_SDRFC_DEFAULT,      \
    CSL_DDR2_SDTIM1_DEFAULT,     \
    CSL_DDR2_SDTIM2_DEFAULT,     \
    CSL_DDR2_BPRIO_RESETVAL \
}
```

Default values for Config structure.

#define CSL_DDR2_SETTING_DEFAULTS

Value:

```
{ \
    (CSL_Ddr2CasLatency)CSL_DDR2_CAS_LATENCY_5, \
    (CSL_Ddr2IntBank)CSL_DDR2_4_SDRAM_BANKS, \
    (CSL_Ddr2PageSize)CSL_DDR2_256WORD_8COL_ADDR, \
    (CSL_Ddr2Mode)CSL_DDR2_NORMAL_MODE\
}
```

The default values of DDR2 SDRAM settings.

#define CSL_DDR2_TIMING1_DEFAULTS

Value:

```
{\
    (Uint16)CSL_DDR2_TIMING1_TRFC_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TRP_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TRCD_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TWR_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1 TRAS_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TRC_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TRRD_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING1_TWTR_DEFAULT \
}
```

The default values of DDR2 SDRAM Timing1 Control structure.

#define CSL_DDR2_TIMING2_DEFAULTS

Value:

```
{ \
    (Uint8)CSL_DDR2_TIMING2_T_ODT_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING2_TSXNR_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING2_TSXRD_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING2_TRTP_DEFAULT, \
    (Uint16)CSL_DDR2_TIMING2_TCKE_DEFAULT \
}
```

The default values of DDR2 SDRAM Timing2 Control structure.

Chapter 4

EDMA MODULE

Topics

<u>4. 1 Overview</u>
<u>4. 2 Functions</u>
<u>4. 3 Data Structures</u>
<u>4. 4 Enumerations</u>
<u>4. 5 Macros</u>

4.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within EDMA module.

The EDMA controller handles all data transfers between the level-two (L2) cache/memory controller and the device peripherals. These data transfers include cache servicing, non-cacheable memory accesses, user-programmed data transfers, and host accesses. The EDMA supports up to 64-event channels and 8 QDMA channels. The EDMA consists of a scalable Parameter RAM (PaRAM) that supports flexible ping-pong, circular buffering, channel-chaining, auto-reloading, and memory protection. The EDMA allows movement of data to/from any addressable memory spaces, including internal memory (L2 SRAM), peripherals, and external memory.

4.2 Functions

This section lists the functions available in the EDMA module.

4.2.1 CSL_edma3Init

CSL_Status CSL_edma3Init (**CSL_Edma3Context *** *pContext*)

Description

This is the initialization function for the EDMA CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

PContext	Pointer to module-context. As edma doesn't have any context based information user is expected to pass NULL.
-----------------	--

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
...
if (CSL_edma3Init(NULL) != CSL_SOK) {
    return;
}
```

4.2.2 CSL_edma3Open

[CSL_Edma3Handle](#) CSL_edma3Open ([CSL_Edma3Obj *](#) *edmaObj*,
CSL_InstNum *edmaNum*,
CSL_Edma3ModuleAttr * *attr*,
CSL_Status * *status*
)

Description

This function Opens the EDMA CSL. It returns a handle to the edma instance. This handle is passed to all other CSL APIs, as the reference to the EDMA instance.

Arguments

edmaObj	Pointer to EDMA Module Object
edmaNum	Instance of EDMA to be opened
attr	EDMA Attribute pointer
status	Status of the function call

Return Value CSL_Edma3Handle

- Valid Edma handle will be returned if status value is equal to CSL_SOK.

Pre Condition

The EDMA must be successfully initialized via CSL_edma3Init() before calling this function.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid EDMA handle is returned
- CSL_ESYS_FAIL The EDMA instance is invalid
- CSL_ESYS_INVPARAMS The Parameter passed is invalid

2. EDMA object structure is populated.

Modifies

- The status variable
- EDMA object structure

Example

```
CSL_Edma3Handle    hModule;
CSL_Edma3Obj       edmaObj;
CSL_Edma3Context   context;
CSL_Status         status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);
```

4.2.3 CSL_edma3Close

CSL_Status CSL_edma3Close ([CSL_Edma3Handle](#) *hEdma*)

Description

This is a module level close required to invalidate the module handle. The module handle must not be used after this API call.

Arguments

hEdma	Handle to the EDMA Instance
-------	-----------------------------

Return Value CSL_Status

- CSL_SOK - EDMA is closed successfully.
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

The functions CSL_edma3Init() and CSL_edma3Open() have to be called in that order successfully before calling this function.

Post Condition

The EDMA CSL APIs can not be called until the EDMA CSL is reopened again using CSL_edma3Open().

Modifies

CSL_edma3Obj structure valuesNone

Example

```
CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3Context     context;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
                    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

CSL_Status status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup  = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Open Channels, setup transfers etc
// Close Module
CSL_edma3Close(hModule);
```

4.2.4 CSL_edma3HwSetup

CSL_Status **CSL_edma3HwSetup** ([CSL_Edma3Handle](#) *hMod*,
[CSL_Edma3HwSetup](#) * *setup*
)

Description

This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure, refer CSL_Edma3HwSetup. This does

the setup for all dma/qdma channels viz. the parameter entry mapping, the trigger word setting (if QDMA channels) and the event queue mapping of the channel.

Arguments

hMod	Edma module Handle
setup	Pointer to the setup structure

Return Value CSL_Status

- CSL_SOK – Successful completion of hardware setup
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() must be called successfully in that order before this API can be invoked.

Post Condition

EDMA registers are configured according to the hardware setup parameters.

Modifies

EDMA registers

Example

```

CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3Context     context;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Edma3HwQdmaChannelSetup qdmahwSetup[CSL_EDMA3_NUM_QDMACH] =
    CSL_EDMA3_QDMACHANNELSETUP_DEFAULT;
CSL_Status            status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = &qdmahwSetup[0];
CSL_edma3HwSetup(hModule, &hwSetup);

```

4.2.5 CSL_edma3GetHwSetup

CSL_Status **CSL_edma3GetHwSetup** ([CSL_Edma3Handle](#) *hMod*,
[CSL_Edma3HwSetup](#) * *setup*
)

Description

It gets the hwparameterssetup of the all edma/qdma channels.

Arguments

hMod	Edma Handle
setup	Pointer to the setup structure

Return Value CSL_Status

- CSL_SOK - Getting the msparameters is successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter passed is Invalid

Pre Condition

The functions CSL_edma3Init(), CSL_edma3Open() must be called successfully in that order before CSL_edma3GetHwSetup() can be called.

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

None

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup          hwSetup, gethwSetup;
CSL_Edma3Obj              edmaObj;
CSL_Edma3Context          context;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
                           CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Edma3HwDmaChannelSetup getdmahwSetup[CSL_EDMA3_NUM_DMACH];
CSL_Status                status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Get Module Setup
gethwSetup.dmaChaSetup = &getdmahwSetup[0];
gethwSetup.qdmaChaSetup = NULL;
CSL_edma3GetHwSetup(hModule, &gethwSetup);

```

4.2.6 CSL_edma3HwControl

```
CSL_Status CSL_edma3HwControl ( CSL\_Edma3Handle          hMod,
                                CSL\_Edma3HwControlCmd       cmd,
                                void *                      cmdArg
                                )
```

Description

This function takes a command with an optional argument and implements it. This function is used to carry out the different operations performed by EDMA.

Arguments

<code>hMod</code>	Edma module Handle
<code>cmd</code>	The command to this API which indicates the action to be taken
<code>cmdArg</code>	Pointer argument specific to the command

Return Value CSL_Status

- CSL_SOK - Command execution successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVCMD - The command passed is invalid

Pre Condition

The functions `CSL_edma3Init()`, `CSL_edma3Open()` must be called successfully in that order before this API can be invoked.

Post Condition

Edma registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

EDMA registers determined by the command

Example

```
CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3QueryInfo   info;
CSL_Edma3CmdDrae     regionAccess;
CSL_Edma3Context      context;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status           status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
```

```

hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Query Module Info
CSL_edma3GetHwStatus(hModule, CSL_EDMA3_QUERY_INFO, &info);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0 ;
regionAccess.drae = 0xFFFF ;
regionAccess.draeh = 0x0000 ;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
    &regionAccess);

```

4.2.7 CSL_edma3GetHwStatus

```

CSL_Status CSL_edma3GetHwStatus ( CSL\_Edma3Handle          hMod,
                                   CSL\_Edma3HwStatusQuery    myQuery,
                                   void *                      response
                                   )

```

Description

This function gets the status of the different operations or the current setup of EDMA module.

Arguments

<i>hMod</i>	Edma module handle
<i>myQuery</i>	Query to be performed
<i>response</i>	Pointer to buffer to return the data requested by the query passed

Return Value CSL_Status

- CSL_SOK - Getting the status of edma is successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVQUERY - The query passed is invalid

Pre Condition

The functions CSL_edma3Init(), CSL_edma3Open() must be called successfully in that order before this API can be invoked. Argument type that can be void* casted and passed with a particular command refer to CSL_Edma3HwStatusQuery.

Post Condition

None

Modifies

The input argument "response" is modified.

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3Obj             edmaObj;
CSL_Edma3QueryInfo       info;
CSL_Edma3Context         context;
CSL_Status               status;

CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
                        CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Query Module Info
CSL_edma3GetHwStatus(hModule, CSL_EDMA3_QUERY_INFO, &info);

```

4.2.8 CSL_edma3ccGetModuleBaseAddr

```

CSL_Status CSL_edma3ccGetModuleBaseAddr( CSL_InstNum          edmaNum,
                                         CSL_Edma3ModuleAttr * pAttr,
                                         CSL\_Edma3ModuleBaseAddress * pBaseAddress
                                         )

```

Description

This function is used for getting the base-address of the EDMA module. This function will be called inside the CSL_edma3Open()/ CSL_edma3ChannelOpen() function call.

Note: This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

edmaNum	Specifies the instance of the edma to be opened.
pAttr	Module specific parameters
pBaseAddress	Pointer to baseaddress structure containing base address details.

Return Value CSL_Status

- CSL_SOK - Successfully retrieved base address
- CSL_ESYS_FAIL - The instance number is invalid.
- CSL_ESYS_INVPARAMS - Invalid Parameter

Pre Condition

None

Post Condition

Base Address structure is populated.

Modifies

- The status variable
- Base address structure is modified.

Example

```
CSL_Status          status;
CSL_Edma3ModuleBaseAddress  baseAddress;

...
status = CSL_edma3ccGetModuleBaseAddr(    CSL_EDMA3,
                                           NULL,
                                           &baseAddress);
```

4.2.9 CSL_edma3ChannelOpen

CSL_Edma3ChannelHandle **CSL_edma3ChannelOpen**(**CSL_Edma3ChannelObj** * **edmaObj**,
CSL_InstNum **edmaNum**,
CSL_Edma3ChannelAttr * **chAttr**,
CSL_Status * **status**
)

Description

The API returns a handle for the specified EDMA Channel for use. The channel can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input as an essential argument for many of the APIs described for this module.

Arguments

edmaObj	pointer to the object that holds reference to the channel instance of the Specified EDMA
edmaNum	Instance of EDMA whose channel is requested
chAttr	Instance of Channel requested
status	Status of the function call

Return Value **CSL_Edma3ChannelHandle**

The requested channel instance of the specified EDMA if the call is successful, else a NULL is returned.

Pre Condition

Functions **CSL_edma3Init()**, **CSL_edma3Open()** must be invoked successfully in that order before this API can be invoked.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid channel handle is returned
- CSL_ESYS_FAIL The EDMA instance or channel is invalid

2. EDMA channel object structure is populated.

Modifies

1. The status variable
2. EDMA channel object structure

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3Obj             edmaObj;
CSL_Edma3ChannelObj      chObj;
CSL_Edma3CmdDrae         regionAccess;
CSL_Edma3ChannelHandle   hChannel;
CSL_Edma3Context         context;
CSL_Edma3ChannelAttr     chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
                        CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status               status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0 ;
regionAccess.drae = 0xFFFF ;
regionAccess.draeh = 0x0000 ;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
                    &regionAccess);
// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(    &chObj,
                                    CSL_EDMA3,
                                    &chAttr,
                                    &status);

// Manually trigger the Channel
CSL_edma3HwChannelControl(    hChannel,
                              CSL_EDMA3_CMD_CHANNEL_SET,
                              NULL);

```

```
// Close Channel
CSL_edma3ChannelClose(hChannel);
```

4.2.10 CSL_edma3ChannelClose

CSL_Status CSL_edma3ChannelClose ([CSL_Edma3ChannelHandle](#) *hEdma*)

Description

This function marks the channel cannot be accessed any more using the handle. CSL for the EDMA channel need to be reopened before using any edma channel.

Arguments

hEdma Handle to the requested channel

Return Value CSL_Status

- CSL_SOK - Edma channel is closed successfully.
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

The functions CSL_edma3Init(), CSL_edma3Open(), CSL_edma3ChannelOpen() must be invoked successfully in that order before this API can be invoked.

Post Condition

The edma channel related CSL APIs can not be called until the edma channel is reopened again using CSL_edma3ChannelOpen().

Modifies

CSL_Edma3ChannelObj structure values.

Example

```
CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3CmdDrae     regionAccess;
CSL_Edma3ChannelHandle hChannel;
CSL_Edma3Context     context;
CSL_Edma3ChannelAttr  chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

CSL_Status            status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
```

```

CSL_edma3HwSetup(hModule, &hwSetup);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0;
regionAccess.drae = 0xFFFF;
regionAccess.draeh = 0x0000;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
                    &regionAccess);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Manually trigger the Channel
CSL_edma3HwChannelControl(hChannel,
                          CSL_EDMA3_CMD_CHANNEL_SET, \
                          NULL);

// Close Channel
CSL_edma3ChannelClose(hChannel);

```

4.2.11 CSL_edma3HwChannelSetupParam

**CSL_Status CSL_edma3HwChannelSetupParam ([CSL_Edma3ChannelHandle](#) *hEdma*,
Uint16 *paramNum*
)**

Description

This function sets up the channel to parameter entry mapping. This writes the DCHMAP[]/QCHMAP appropriately.

Arguments

<i>hEdma</i>	Channel Handle
<i>paramNum</i>	Parameter Entry Number

Return Value CSL_Status

- CSL_SOK - Channel setup param successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameters passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before this API can be invoked.

Post Condition

Channel to parameter entry is configured.

Modifies
EDMA registers

Example

```

CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3ChannelHandle hChannel;
CSL_Edma3Context     context;
CSL_Edma3ChannelAttr chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

Uint16               paramNum;
CSL_Status            status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(    &chObj,
                                   CSL_EDMA3,
                                   &chAttr,
                                   &status);

// Set the parameter entry number to channel
paramNum = 100;
CSL_edma3HwChannelSetupParam(hChannel, paramNum);

```

4.2.12 CSL_edma3HwChannelSetupTriggerWord

CSL_Status CSL_edma3HwChannelSetupTriggerWord([CSL_Edma3ChannelHandle](#) hEdma,

UInt8
triggerWord

)

Description

Programs the QDMA channel triggerword. This writes the QCHMAP appropriately.

Arguments

<code>hEdma</code>	Channel Handle
<code>triggerWord</code>	Trigger word

Return Value `CSL_Status`

-
- CSL_SOK - Channel setup triggerword is successful
 - CSL_ESYS_BADHANDLE - The handle passed is invalid
 - CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before CSL_edma3HwChannelSetupTriggerWord() can be called.

Post Condition

None

Modifies

EDMA registers

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup          hwSetup;
CSL_Edma3Obj              edmaObj;
CSL_Edma3ChannelObj       chObj;
CSL_Edma3ChannelHandle    hChannel;
CSL_Edma3Context          context;
CSL_Edma3ChannelAttr      chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
                           CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status                status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_QCHA_0;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Get the trigger word programmed for a channel
CSL_edma3HwChannelSetupTriggerWord(hChannel, 3);

```


4.2.13 CSL_edma3HwChannelSetupQue

```
CSL_Status CSL_edma3HwChannelSetupQue ( CSL\_Edma3ChannelHandle hEdma,
                                         CSL_Edma3Que que
                                         )
```

Description

This function programs the channel to Queue mapping. This writes the DMAQNUM/QDAMQNUM appropriately.

Arguments

hEdma	Channel Handle
que	Event Queue Name

Return Value CSL_Status

- CSL_SOK - Channel setup queue successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before this API can be invoked.

Post Condition

Sets up the channel to Queue mapping

Modifies

EDMA registers

Example

```
CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3ChannelHandle hChannel;
CSL_Edma3Context     context;
CSL_Edma3ChannelAttr chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status           status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);
```

```
// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Set up the channel to que mapping
CSL_edma3HwChannelSetupQue(hChannel, CSL_EDMA3_QUE_3);
```

4.2.14 CSL_edma3GetHwChannelSetupParam

```
CSL_Status CSL_edma3GetHwChannelSetupParam( CSL\_Edma3ChannelHandle hEdma,
                                             Uint16 * paramNum
                                             )
```

Description

This function obtains the Channel to Parameter Set mapping. This reads the DCHMAP/QCHMAP appropriately.

Arguments

hEdma	Channel Handle
paramNum	Pointer to parameter entry

Return Value CSL Status

- CSL_SOK - Retrieving the parameter entry number to which a channel is mapped
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

The functions `CSL_edma3Init()`, `CSL_edma3Open()` and `CSL_edma3ChannelOpen()` must be called successfully in that order before `CSL_edma3GetHwChannelSetupParam()` can be invoked.

Post Condition

None

Modifies

None

Example

```
CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3Obj            edmaObj;
CSL_Edma3ChannelObj      chObj;
CSL_Edma3ChannelHandle   hChannel;
CSL_Edma3Context         context;
CSL_Edma3ChannelAttr     chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
```

```

    UInt16          paramNum;
    CSL_Status      status;

    // Module Initialization
    CSL_edma3Init(&context);

    // Module Level Open
    hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

    // Module Setup
    hwSetup.dmaChaSetup = &dmahwSetup[0];
    hwSetup.qdmaChaSetup = NULL;
    CSL_edma3HwSetup(hModule, &hwSetup);

    // Channel 0 Open in context of Shadow region 0
    chAttr.regionNum = CSL_EDMA3_REGION_0;
    chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
    hChannel = CSL_edma3ChannelOpen(&chObj,
                                    CSL_EDMA3,
                                    &chAttr,
                                    &status);

    // Get the parameter entry number to which a channel is mapped
    CSL_edma3GetHwChannelSetupParam(hChannel, &paramNum);

```

4.2.15 CSL_edma3GetHwChannelSetupTriggerWord

```

CSL_Status CSL_edma3GetHwChannelSetupTriggerWord( CSL\_Edma3ChannelHandle hEdma,
                                                    UInt8 *
                                                    triggerWord
                                                    )

```

Description

This function read the QDMA channel triggerword. This reads the QCHMAP to obtain the trigger word appropriately.

Arguments

hEdma	Channel Handle
triggerWord	Pointer to Trigger word

Return Value CSL_Status

- CSL_SOK - Retrieving the parameter entry number to which a channel is mapped
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before CSL_edma3GetHwChannelSetupTriggerWord() can be called.

Post Condition

None

Modifies

None

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3Obj             edmaObj;
CSL_Edma3ChannelObj      chObj;
CSL_Edma3ChannelHandle   hChannel;
CSL_Edma3Context         context;
CSL_Edma3ChannelAttr     chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
                        CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

Uint8                    triggerWord;
CSL_Status               status;


// Module Initialization
CSL_edma3Init(&context);


// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);


// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);


// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);


// Get the trigger word programmed for a channel
CSL_edma3GetHwChannelSetupTriggerWord(hChannel, &triggerWord);

```

4.2.16 CSL_edma3GetHwChannelSetupQue

```

CSL_Status CSL_edma3GetHwChannelSetupQue ( CSL\_Edma3ChannelHandle hEdma,
                                           CSL_Edma3Que *      que
                                           )

```

Description

This function obtains the channel to queue map for the channel. This reads the DMAQNUM/QDAMQNUM appropriately.

Arguments

hEdma	Channel Handle
que	Pointer to Event Queue structure

Return Value CSL_Status

- CSL_SOK - Retrieving the queue to which a channel is mapped
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter is Invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before CSL_edma3GetHwChannelSetupQue() can be called.

Post Condition

None

Modifies

None

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3Obj             edmaObj;
CSL_Edma3ChannelObj      chObj;
CSL_Edma3ChannelHandle   hChannel;
CSL_Edma3Context         context;
CSL_Edma3ChannelAttr     chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

CSL_Edma3Que             evtQue;
CSL_Status               status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Get the que to which a channel is mapped
CSL_edma3GetHwChannelSetupQue(hChannel, &evtQue);

```

4.2.17 CSL_edma3HwChannelControl

```
CSL_Status CSL_edma3HwChannelControl ( CSL\_Edma3ChannelHandle      hCh,
                                         CSL\_Edma3HwChannelControlCmd cmd,
                                         void *                          cmdArg
                                         )
```

Description

This function takes a command with an optional argument and implements it. This function is used to carry out the different operations performed by EDMA.

Arguments

<code>hCh</code>	Channel Handle
<code>cmd</code>	The command to this API which indicates the action to be taken
<code>cmdArg</code>	Pointer argument specific to the command

Return Value CSL_Status

- `CSL_SOK` - Command execution successful
- `CSL_ESYS_BADHANDLE` - The handle passed is invalid
- `CSL_ESYS_INVCMD` - The command passed is invalid

Pre Condition

Functions `CSL_edma3Init()`, `CSL_edma3Open()` and `CSL_edma3ChannelOpen()` must be called successfully in that order before this API can be invoked. If a Shadow region is used, then care of the DRAE settings must be taken.

Post Condition

EDMA registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

EDMA registers determined by the command

Example

```
CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3CmdIntr     regionIntr;
CSL_Edma3CmdDrae     regionAccess;
CSL_Edma3ChannelHandle hCh;
CSL_Edma3Context     context;
CSL_Edma3ChannelAttr chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status           status;
```

```
// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0 ;
regionAccess.drae = 0xFFFF ;
regionAccess.draeh = 0x0000 ;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
                    &regionAccess);

// Interrupt Enable (Bits 0-11) for the Shadow Region 0.
regionIntr.region = CSL_EDMA3_REGION_0 ;
regionIntr.intr = 0x0FFF ;
regionIntr.intrh = 0x0000 ;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_INTR_ENABLE,
                    &regionIntr);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hCh = CSL_edma3ChannelOpen(&chObj,
                           CSL_EDMA3,
                           &chAttr,
                           &status);

// Enable Channel(if the channel is meant for external event)
// This step is not required if the channel is chained to or
// manually triggered.
CSL_edma3HwChannelControl(hCh, CSL_EDMA3_CMD_CHANNEL_ENABLE, NULL);
```

4.2.18 CSL_edma3GetHwChannelStatus

```
CSL_Status CSL_edma3GetHwChannelStatus(CSL\_Edma3ChannelHandle      hCh,
                                       CSL\_Edma3HwChannelStatusQuery myQuery,
                                       void *                          response
                                       )
```

Description

This function gets the status of the different operations or the current setup of EDMA module.

Arguments

hCh	Channel Handle
myQuery	Query to be performed

response	Pointer to buffer to return the data requested by the query passed
----------	--

Return Value CSL_Status

- CSL_SOK - Getting the EDMA channel status is successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVQUERY - The query passed is invalid

Pre Condition

The functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before this API can be invoked. If a Shadow region is used, then care of the DRAE settings must be taken.

Post Condition

None

Modifies

The input argument "response" is modified.

Example

```

CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3ChannelHandle hChannel;
CSL_Edma3Context     context;
CSL_Edma3ChannelAttr chAttr;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] = \
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

CSL_Status           status;
void                 *response;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Enable Channel( .. )

```

```

CSL_edma3HwChannelControl(    hChannel,
                               CSL_EDMA3_CMD_CHANNEL_ENABLE,
                               NULL);

    // Obtain Channel Error Status
CSL_edma3GetHwChannelStatus(  hChannel,
                               CSL_EDMA3_QUERY_CHANNEL_ERR,
                               response);

```

4.2.19 CSL_edma3GetParamHandle

```

CSL_Edma3ParamHandle CSL_edma3GetParamHandle( CSL\_Edma3ChannelHandle hEdma,
                                                Int16 paramNum,
                                                CSL_Status * status
                                                )

```

Description

This function acquires the PARAM entry as specified by the argument.

Arguments

hEdma	Channel Handle
paramNum	Parameter entry number
status	Status of the function call

Return Value CSL_Edma3ParamHandle

Valid PARAM handle will be returned if status value is equal to CSL_SOK.

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before this API can be invoked.

Post Condition

- The status is returned in the status variable. If status returned is
 - CSL_SOK Valid PARAM handle is returned.
 - CSL_ESYS_INVPARAMS Parameter passed is invalid
 - CSL_ESYS_BADHANDLE Channel handle is invalid

Modifies

None

Example

CSL_Edma3Handle	hModule;
CSL_Edma3HwSetup	hwSetup;
CSL_Edma3ParamHandle	hParamBasic;
CSL_Edma3Obj	edmaObj;
CSL_Edma3ChannelObj	chObj;
CSL_Edma3CmdIntr	regionIntr;
CSL_Edma3CmdDrae	regionAccess;
CSL_Edma3ChannelHandle	hChannel;
CSL_Edma3Context	context;

```

    CSL_Edma3ChannelAttr      chAttr;
    CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
                                CSL_EDMA3_DMACHANNELSETUP_DEFAULT;

    CSL_Status                status;

    // Module Initialization
    CSL_edma3Init(&context);

    // Module Level Open
    hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

    // Module Setup
    hwSetup.dmaChaSetup = &dmahwSetup[0];
    hwSetup.qdmaChaSetup = NULL;
    CSL_edma3HwSetup(hModule, &hwSetup);

    // DRAE Enable (Bits 0-15) for the Shadow Region 0.
    regionAccess.region = CSL_EDMA3_REGION_0 ;
    regionAccess.drae = 0xFFFF ;
    regionAccess.draeh = 0x0000 ;
    CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
                      &regionAccess);

    // Interrupt Enable (Bits 0-11) for the Shadow Region 0.
    regionIntr.region = CSL_EDMA3_REGION_0 ;
    regionIntr.intr = 0x0FFF ;
    regionIntr.intrh = 0x0000 ;
    CSL_edma3HwControl(hModule,
                      CSL_EDMA3_CMD_INTR_ENABLE, &regionIntr);

    // Channel 0 Open in context of Shadow region 0
    chAttr.regionNum = CSL_EDMA3_REGION_0;
    chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
    hChannel = CSL_          (&chObj,
                             CSL_EDMA3,
                             &chAttr,
                             &status);

    // Obtain a handle to parameter entry 0
    hParamBasic = CSL_edma3GetParamHandle(hChannel, 0, NULL);

```

4.2.20 CSL_edma3ParamSetup

```

CSL_Status CSL_edma3ParamSetup    ( CSL_Edma3ParamHandle    hParam,
                                   CSL\_Edma3ParamSetup * setup
                                   )

```

Description

This function configures the EDMA parameter Entry using the values passed in through the PARAM setup structure.

Arguments

hParam Handle to the PARAM entry

setup

Pointer to PARAM setup structure

Return Value CSL_Status

- CSL_SOK - PARAM setup successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

Functions CSL_edma3Init(), CSL_edma3Open() and CSL_edma3ChannelOpen() must be called successfully in that order before this API can be invoked.

Post Condition

Configures the EDMA parameter entry

Modifies

Parameter entry

Example

```

CSL_Edma3Handle      hModule;
CSL_Edma3HwSetup     hwSetup;
CSL_Edma3ParamHandle hParamBasic;
CSL_Edma3Obj         edmaObj;
CSL_Edma3ChannelObj  chObj;
CSL_Edma3CmdIntr     regionIntr;
CSL_Edma3CmdDrae     regionAccess;
CSL_Edma3ChannelHandle hChannel;
CSL_Edma3ParamSetup  myParamSetup;
CSL_Edma3Context      context;
CSL_Edma3ChannelAttr  chAttr;
UInt32               srcBuff1 = 0;
UInt32               dstBuff1 = 0;
CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Status            status;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];
hwSetup.qdmaChaSetup = NULL;
CSL_edma3HwSetup(hModule, &hwSetup);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0 ;
regionAccess.drae = 0xFFFF ;
regionAccess.draeh = 0x0000 ;
CSL_edma3HwControl(hModule, CSL_EDMA3_CMD_DMAREGION_ENABLE, \
    &regionAccess);

```

```

// Interrupt Enable (Bits 0-11) for the Shadow Region 0.
regionIntr.region = CSL_EDMA3_REGION_0 ;
regionIntr.intr = 0x0FFF ;
regionIntr.intrh = 0x0000 ;
CSL_edma3HwControl(hModule,
                   CSL_EDMA3_CMD_INTR_ENABLE,&regionIntr);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Obtain a handle to parameter entry 0
hParamBasic = CSL_edma3GetParamHandle(hChannel,0,NULL);

// Setup the first param Entry (Ping buffer)
myParamSetup.option = CSL_EDMA3_OPT_MAKE(CSL_EDMA3_ITCCH_DIS, \
                                           CSL_EDMA3_TCCH_DIS, \
                                           CSL_EDMA3_ITCINT_DIS, \
                                           CSL_EDMA3_TCINT_EN, \
                                           0,CSL_EDMA3_TCC_NORMAL, \
                                           CSL_EDMA3_FIFOWIDTH_NONE, \
                                           CSL_EDMA3_STATIC_DIS, \
                                           CSL_EDMA3_SYNC_A, \
                                           CSL_EDMA3_ADDRMODE_INCR, \
                                           CSL_EDMA3_ADDRMODE_INCR);

myParamSetup.srcAddr = (Uint32)srcBuff1;
myParamSetup.aCntbCnt = CSL_EDMA3_CNT_MAKE(256,1);
myParamSetup.dstAddr = (Uint32)dstBuff1;
myParamSetup.srcDstBidx = CSL_EDMA3_BIDX_MAKE(1,1);
myParamSetup.linkBcntrlld = CSL_EDMA3_LINKBCNTRLD_MAKE
                             (CSL_EDMA3_LINK_NULL,0);
myParamSetup.srcDstCidx = CSL_EDMA3_CIDX_MAKE(0,1);
myParamSetup.cCnt = 1;
CSL_edma3ParamSetup(hParamBasic,&myParamSetup);

```

4.2.21 CSL_edma3ParamWriteWord

CSL_Status	CSL_edma3ParamWriteWord	(CSL_Edma3ParamHandle	hParamHndl,
			Uint16	wordOffset,
			Uint32	word
)		

Description

This is for the ease of QDMA channels. Once the QDMA channel transfer is triggered, subsequent triggers may be done with only writing the modified words in the parameter entry along with the trigger word. This API is expected to achieve this purpose. Most usage scenarios, the user should not be writing more than the trigger word entry.

Arguments

<code>hParamHndl</code>	Handle to the param entry
<code>wordOffset</code>	Word offset in the 8 word parameter entry
<code>word</code>	Word to be written

Return Value `CSL_Status`

- `CSL_SOK` - PARAM Write Word successful
- `CSL_ESYS_BADHANDLE` - The handle passed is invalid
- `CSL_ESYS_INVPARAMS` - The parameter passed is invalid

Pre Condition

Functions `CSL_edma3Init()`, `CSL_edma3Open()` and `CSL_edma3ChannelOpen()` and must be `CSL_edma3GetParamHandle()`, `CSL_edma3ParamSetup()` called successfully in that order before this API can be invoked. The main setup structure consists of pointers to sub-structures. The user has to allocate space for and fill in the parameter setup structure.

Post Condition

Configure trigger word

Modifies

None

Example

```

CSL_Edma3Handle          hModule;
CSL_Edma3HwSetup         hwSetup;
CSL_Edma3ParamHandle     hParamBasic;
CSL_Edma3Obj             edmaObj;
CSL_Edma3ChannelObj      chObj;
CSL_Edma3CmdIntr         regionIntr;
CSL_Edma3CmdQrae         regionAccess;
CSL_Edma3ChannelHandle    hChannel;
CSL_Edma3ParamSetup       myParamSetup;
CSL_Edma3Context          context;
CSL_Edma3ChannelAttr      chAttr;
CSL_Status               status;
UInt32                   srcBuff1 = 0;
UInt32                   dstBuff1 = 0;

CSL_Edma3HwDmaChannelSetup dmahwSetup[CSL_EDMA3_NUM_DMACH] =
    CSL_EDMA3_DMACHANNELSETUP_DEFAULT;
CSL_Edma3HwQdmaChannelSetup qdmahwSetup[CSL_EDMA3_NUM_QDMACH] =
    CSL_EDMA3_QDMACHANNELSETUP_DEFAULT;

// Module Initialization
CSL_edma3Init(&context);

// Module Level Open
hModule = CSL_edma3Open(&edmaObj, CSL_EDMA3, NULL, &status);

// Module Setup
hwSetup.dmaChaSetup = &dmahwSetup[0];

```

```

hwSetup.qdmaChaSetup = &qdmahwSetup[0];
CSL_edma3HwSetup(hModule,&hwSetup);

// DRAE Enable(Bits 0-15) for the Shadow Region 0.
regionAccess.region = CSL_EDMA3_REGION_0 ;
regionAccess.grae = 0x000F ;
CSL_edma3HwControl(hModule,CSL_EDMA3_CMD_QDMAREGION_ENABLE, \
                    &regionAccess);

// Interrupt Enable (Bits 0-11) for the Shadow Region 0.
regionIntr.region = CSL_EDMA3_REGION_0 ;
regionIntr.intr = 0x0FFF ;
regionIntr.intrh = 0x0000 ;
CSL_edma3HwControl(hModule,CSL_EDMA3_CMD_INTR_ENABLE,&regionIntr);

// Channel 0 Open in context of Shadow region 0
chAttr.regionNum = CSL_EDMA3_REGION_0;
chAttr.chaNum = CSL_EDMA3_CHA_TCPREVT;
hChannel = CSL_edma3ChannelOpen(&chObj,
                                CSL_EDMA3,
                                &chAttr,
                                &status);

// Obtain a handle to parameter entry 0
hParamBasic = CSL_edma3GetParamHandle(hChannel,0,NULL);

// Setup the first param Entry (Ping buffer)
myParamSetup.option = CSL_EDMA3_OPT_MAKE(CSL_EDMA3_ITCCH_DIS, \
                                           CSL_EDMA3_TCCH_DIS, \
                                           CSL_EDMA3_ITCINT_DIS, \
                                           CSL_EDMA3_TCINT_EN, \
                                           0,CSL_EDMA3_TCC_NORMAL, \
                                           CSL_EDMA3_FIFOWIDTH_NONE, \
                                           CSL_EDMA3_STATIC_EN, \
                                           CSL_EDMA3_SYNC_A, \
                                           CSL_EDMA3_ADDRMODE_INCR, \
                                           CSL_EDMA3_ADDRMODE_INCR);

myParamSetup.srcAddr = (Uint32)srcBuff1;
myParamSetup.aCntbCnt = CSL_EDMA3_CNT_MAKE(256,1);
myParamSetup.dstAddr = (Uint32)dstBuff1;
myParamSetup.srcDstBidx = CSL_EDMA3_BIDX_MAKE(1,1);
myParamSetup.linkBcntrl = CSL_EDMA3_LINKBCNTRLD_MAKE
                           (CSL_EDMA3_LINK_NULL,0);
myParamSetup.srcDstCidx = CSL_EDMA3_CIDX_MAKE(0,1);
myParamSetup.cCnt = 1;
CSL_edma3ParamSetup(hParamBasic,&myParamSetup);

// Enable Channel
CSL_edma3HwChannelControl(hChannel,
                          CSL_EDMA3_CMD_CHANNEL_ENABLE,
                          NULL);

// Write trigger word
CSL_edma3ParamWriteWord(hParamBasic,7,myParamSetup.cCnt);

```

4.3 Data Structures

This section lists the data structures available in the EDMA module.

4.3.1 CSL_Edma3Obj

Detailed Description

This object contains the reference to the instance of EDMA Module opened using the *CSL_edma3Open()*. A pointer to this object is passed to all EDMA Module level CSL APIs.

Field Documentation

CSL_InstNum CSL_Edma3Obj::instNum

This is the instance of module number i.e. CSL_EDMA3

CSL_Edma3ccRegsOvly CSL_Edma3Obj::regs

This is a pointer to the EDMA Channel Controller registers of the module requested.

4.3.2 CSL_Edma3ParamSetup

Detailed Description

Edma ParamSetup Structure. An object of this type is allocated by the user and its address is passed as a parameter to the *CSL_edma3ParamSetup()*. This structure is used to program the Param Set for EDMA/QDMA. The macros can be used to assign values to the fields of the structure. The setup structure should be setup using the macros provided OR as per the bit descriptions in the user guide.

Field Documentation

UInt32 CSL_Edma3ParamSetup::aCntbCnt

Lower 16 bits are A Count Upper 16 bits are B Count

UInt32 CSL_Edma3ParamSetup::cCnt

C count

UInt32 CSL_Edma3ParamSetup::dstAddr

Specifies the destination address

UInt32 CSL_Edma3ParamSetup::linkBcntrl

Lower 16 bits are link of the next PARAM entry Upper 16 bits are b count reload

UInt32 CSL_Edma3ParamSetup::option

Options

UInt32 CSL_Edma3ParamSetup::srcAddr

Specifies the source address

UInt32 CSL_Edma3ParamSetup::srcDstBidx

Lower 16 bits are source b index Upper 16 bits are destination b index

UInt32 CSL_Edma3ParamSetup::srcDstCidx

Lower 16 bits are source c index Upper 16 bits are destination c index

4.3.3 CSL_Edma3ChannelObj

Detailed Description

Edma Channel Object Structure. An object of this type is allocated by the user and its address is passed as a parameter to the `CSL_edma3ChannelOpen()`. The `CSL_edma3ChannelOpen()` updates all the members of the data structure and returns the objects address as a *CSL_Edma3ChannelHandle*. The *CSL_Edma3ChannelHandle* is used in all subsequent function calls.

Field Documentation

Int CSL_Edma3ChannelObj::chanum

Channel Number being requested

Int CSL_Edma3ChannelObj::edmaNum

EDMA instance whose channel is being requested

Int CSL_Edma3ChannelObj::region

Region number to which the channel belongs

CSL_Edma3ccRegsOvly CSL_Edma3ChannelObj::regs

Pointer to the EDMA Channel Controller module register overlay structure

4.3.4 CSL_Edma3MemFaultStat

Detailed Description

An object of this type is allocated by the user and its address is passed as a parameter to the `CSL_edma3GetMemoryFaultError()` / `CSL_edma3GetHwStatus()` with the relevant command. This is relevant only if MPEXIST is present for a given device.

Field Documentation

UInt32 CSL_Edma3MemFaultStat::addr

Memory Protection Fault Address

CSL_BitMask16 CSL_Edma3MemFaultStat::error

Bit Mask of the Errors

UInt16 CSL_Edma3MemFaultStat::fid

Faulted ID

4.3.5 CSL_Edma3CtrlErrStat

Detailed Description

EDMA Controller Error Status. An object of this type is allocated by the user and its address is passed as a parameter to the `CSL_edma3GetControllerError()` / `CSL_edma3GetHwStatus()`.

Field Documentation

CSL_BitMask16 CSL_Edma3CtrlErrStat::error

Bit Mask of the Queue Threshold Errors

Bool CSL_Edma3CtrlErrStat::exceedTcc

Whether number of permissible outstanding TCCs is exceeded

4.3.6 CSL_Edma3QueryInfo

Detailed Description

EDMA Controller Information. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetInfo() /CSL_edma3GetHwStatus().

Field Documentation

UInt32 CSL_Edma3QueryInfo::config

Channel Controller Configuration obtained from the CCCFG register

UInt32 CSL_Edma3QueryInfo::revision

Revision/Peripheral id of the EDMA3 Channel Controller

4.3.7 CSL_Edma3ActivityStat

Detailed Description

EDMA Channel Controller Activity Status. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetActivityStatus() /CSL_edma3GetHwStatus().

Field Documentation

Bool CSL_Edma3ActivityStat::active

Indicates if the Channel Controller is active at all

Bool CSL_Edma3ActivityStat::evtActive

Indicates whether any EDMA events are active

UInt16 CSL_Edma3ActivityStat::outstandingTcc

Number of outstanding completion requests

Bool CSL_Edma3ActivityStat::qevtActive

Indicates whether any QDMA events are active

CSL_BitMask16 CSL_Edma3ActivityStat::queueActive

BitMask of the queue active in the Channel Controller

Bool CSL_Edma3ActivityStat::trActive

Indicates whether the TR processing/submission logic is active

4.3.8 CSL_Edma3QueStat

Detailed Description

EDMA Controller Queue Status. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetQueStatus() /CSL_edma3GetHwStatus().

Field Documentation
Bool CSL_Edma3QueStat::exceed

Output field: The number of valid entries in a queue has exceeded the threshold specified in QWMTHRA has been exceeded

UInt8 CSL_Edma3QueStat::numVal

Output field: Number of valid entries in Queue N

CSL_Edma3Que CSL_Edma3QueStat::que

Input field: Event Queue. This needs to be specified by the user before invocation of the above API

UInt8 CSL_Edma3QueStat::startPtr

Output field: Start pointer/Head of the queue

UInt8 CSL_Edma3QueStat::waterMark

Output field: The most entries that have been in Queue since reset/last time the watermark was cleared

4.3.9 CSL_Edma3CmdRegion

Detailed Description

EDMA Control/Query Command Structure for querying region specific attributes. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetHwStatus/CSL_edma3HwControl with the relevant command.

Field Documentation
Int CSL_Edma3CmdRegion::region

Input field:- this field needs to be initialized by the user before issuing the query/command

CSL_BitMask32 CSL_Edma3CmdRegion::regionVal

Input/Output field. This needs to be filled by the user in case of issuing a COMMAND or it will be filled in by the CSL when used with a QUERY

4.3.10 CSL_Edma3CmdQrae

Detailed Description

EDMA Control/Query Command Structure for querying QDMA region access enable attributes. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetHwStatus/CSL_edma3HwControl with the relevant command.

Field Documentation
CSL_BitMask32 CSL_Edma3CmdQrae::qrae

This needs to be filled by the user in case of issuing a COMMAND or it will be filled in by the CSL when used with a QUERY

Int CSL_Edma3CmdQrae::region

This field needs to be initialized by the user before issuing the query/command

4.3.11 CSL_Edma3CmdIntr

Detailed Description

EDMA Control/Query Control Command structure for issuing commands for interrupt related APIs
An object of this type is allocated by the user and its address is passed to the Control API.

Field Documentation

CSL_BitMask32 CSL_Edma3CmdIntr::intr

Input/Output field: - this needs to be filled by the user in case of issuing a COMMAND or it will be filled in by the CSL when used with a QUERY

CSL_BitMask32 CSL_Edma3CmdIntr::intrh

Input/Output: - this needs to be filled by the user in case of issuing a COMMAND or it will be filled in by the CSL when used with a QUERY

Int CSL_Edma3CmdIntr::region

Input field: - this field needs to be initialized by the user before issuing the query/command

4.3.12 CSL_Edma3CmdDrae

Detailed Description

EDMA Command Structure for setting region specific attributes. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetHwStatus when

Field Documentation

CSL_BitMask32 CSL_Edma3CmdDrae::drae

DRAE Setting for the region

CSL_BitMask32 CSL_Edma3CmdDrae::draeh

DRAEH Setting for the region

Int CSL_Edma3CmdDrae::region

This field needs to be initialized by the user before issuing the command specifying the region for which attributes need to be set

4.3.13 CSL_Edma3CmdQuePri

Detailed Description

EDMA Command Structure used for setting Event Queue priority level.
An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3HwControl API.

Field Documentation

[CSL_Edma3QuePri](#) CSL_Edma3CmdQuePri::pri

Queue priority

CSL_Edma3Que CSL_Edma3CmdQuePri::que

Specifies the Queue that needs a priority change

4.3.14 CSL_Edma3CmdQueThr

Detailed Description

EDMA Command Structure used for setting Event Queue threshold level. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3HwControl API.

Field Documentation

CSL_Edma3Que CSL_Edma3CmdQueThr::que

Specifies the Queue that needs a change in the threshold setting

[CSL_Edma3QueThr](#) CSL_Edma3CmdQueThr::threshold

Queue threshold setting

4.3.15 CSL_Edma3ModuleBaseAddress

Detailed Description

This will have the base-address information for the module instance.

Field Documentation

CSL_Edma3ccRegsOvly CSL_Edma3ModuleBaseAddress::regs

Base-address of the peripheral registers

4.3.16 CSL_Edma3ChannelAttr

Detailed Description

EDMA Channel parameter structure. This is used for opening a channel.

Field Documentation

Int CSL_Edma3ChannelAttr::chanum

Channel number

Int CSL_Edma3ChannelAttr::regionNum

Region Number

4.3.17 CSL_Edma3ChannelErr

Detailed Description

Edma Channel Error structure. An object of this type is allocated by the user and its address is passed as a parameter to the CSL_edma3GetChannelError() /CSL_edma3GetHwStatus()/ CSL_edma3ChannelErrorClear() /CSL_edma3HwChannelControl().

Field Documentation

Bool CSL_Edma3ChannelErr::missed

A TRUE indicates an event is missed on this channel.

Bool CSL_Edma3ChannelErr::secEvt

A TRUE indicates an event that no events on this channel will be prioritized until this is cleared. This being TRUE does NOT necessarily mean it is an error. ONLY if both missed and ser are set, this kind of error needs to be cleared.

4.3.18 CSL_Edma3HwQdmaChannelSetup

Detailed Description

QDMA Edma Channel Setup. An array of such objects are allocated by the user and address initialized in the CSL_Edma3HwSetup structure which is passed CSL_edma3HwSetup()

Field Documentation
UInt16 CSL_Edma3HwQdmaChannelSetup::paramNum

Parameter set mapping for the channel.

CSL_Edma3Queue CSL_Edma3HwQdmaChannelSetup::que

Queue number for the channel

UInt8 CSL_Edma3HwQdmaChannelSetup::triggerWord

Trigger word for the QDMA channels.

4.3.19 CSL_Edma3HwDmaChannelSetup

Detailed Description

QDMA EDMA Channel Setup. An array of such objects are allocated by the user and address initialized in the CSL_Edma3HwSetup structure which is passed CSL_edma3HwSetup()

Field Documentation
CSL_Edma3Queue CSL_Edma3HwDmaChannelSetup::que

Queue number for the channel

UInt16 CSL_Edma3HwDmaChannelSetup::paramNum

Parameter set mapping for the channel

4.3.20 CSL_Edma3HwSetup

Detailed Description

This structure is used to setup or obtain existing setup of EDMA using CSL_edma3HwSetup () and CSL_edma3GetHwSetup () respectively.

Field Documentation
[CSL_Edma3HwDmaChannelSetup](#)* CSL_Edma3HwSetup::dmaChaSetup

Pointer to Edma Hw Channel setup structure.

[CSL_Edma3HwQdmaChannelSetup](#)* CSL_Edma3HwSetup::qdmaChaSetup

Pointer to QDMA channel setup structure

4.4 Enumerations

4.4.1 CSL_Edma3QuePri

enum CSL_Edma3QuePri

Enumeration for System priorities.

This is used for Setting up the Queue Priority level.

Enumeration values:

<i>CSL_EDMA3_QUE_PRI_0</i>	System priority level 0
<i>CSL_EDMA3_QUE_PRI_1</i>	System priority level 1
<i>CSL_EDMA3_QUE_PRI_2</i>	System priority level 2
<i>CSL_EDMA3_QUE_PRI_3</i>	System priority level 3
<i>CSL_EDMA3_QUE_PRI_4</i>	System priority level 4
<i>CSL_EDMA3_QUE_PRI_5</i>	System priority level 5
<i>CSL_EDMA3_QUE_PRI_6</i>	System priority level 6
<i>CSL_EDMA3_QUE_PRI_7</i>	System priority level 7

4.4.2 CSL_Edma3QueThr

enum CSL_Edma3QueThr

Enumeration for EDMA Queue Thresholds.

This is used for Setting up the Queue thresholds.

Enumeration values:

<i>CSL_EDMA3_QUE_THR_0</i>	EDMA Queue Threshold 0
<i>CSL_EDMA3_QUE_THR_1</i>	EDMA Queue Threshold 1
<i>CSL_EDMA3_QUE_THR_2</i>	EDMA Queue Threshold 2
<i>CSL_EDMA3_QUE_THR_3</i>	EDMA Queue Threshold 3
<i>CSL_EDMA3_QUE_THR_4</i>	EDMA Queue Threshold 4
<i>CSL_EDMA3_QUE_THR_5</i>	EDMA Queue Threshold 5
<i>CSL_EDMA3_QUE_THR_6</i>	EDMA Queue Threshold 6
<i>CSL_EDMA3_QUE_THR_7</i>	EDMA Queue Threshold 7
<i>CSL_EDMA3_QUE_THR_8</i>	EDMA Queue Threshold 8
<i>CSL_EDMA3_QUE_THR_9</i>	EDMA Queue Threshold 9
<i>CSL_EDMA3_QUE_THR_10</i>	EDMA Queue Threshold 10
<i>CSL_EDMA3_QUE_THR_11</i>	EDMA Queue Threshold 11
<i>CSL_EDMA3_QUE_THR_12</i>	EDMA Queue Threshold 12
<i>CSL_EDMA3_QUE_THR_13</i>	EDMA Queue Threshold 13
<i>CSL_EDMA3_QUE_THR_14</i>	EDMA Queue Threshold 14
<i>CSL_EDMA3_QUE_THR_15</i>	EDMA Queue Threshold 15
<i>CSL_EDMA3_QUE_THR_16</i>	EDMA Queue Threshold 16
<i>CSL_EDMA3_QUE_THR_DISABLE</i>	EDMA Queue Threshold Disable Errors

4.4.3 CSL_Edma3HwControlCmd

enum CSL_Edma3HwControlCmd

MODULE Level Commands

Enumeration values:

<i>CSL_EDMA3_CMD_DMAREGION_ENABLE</i>	(Arg: <i>CSL_Edma3CmdDrae*</i>) Enables bits as specified in the argument passed in DRAE/DRAEH. Please note: If bits are already set in DRAE/DRAEH this Control command will cause additional bits (as specified by the bitmask) to be set and does
<i>CSL_EDMA3_CMD_DMAREGION_DISABLE</i>	(Arg: <i>CSL_Edma3CmdDrae*</i>) Disables bits as specified in the argument passed in DRAE/DRAEH
<i>CSL_EDMA3_CMD_QDMAREGION_ENABLE</i>	(Arg: <i>CSL_Edma3CmdQrae*</i>) Enables bits as specified in the argument passed in QRAE. Please note: If bits are already set in QRAE/QRAEH this Control command will cause additional bits (as specified by the bitmask) to be set and does.
<i>CSL_EDMA3_CMD_QDMAREGION_DISABLE</i>	(Arg: <i>CSL_Edma3CmdQrae*</i>) Disables bits as specified in the argument passed in QRAE
<i>CSL_EDMA3_CMD_QUEPRIORITY_SET</i>	(Arg: <i>CSL_Edma3CmdQuePri*</i>) Programming QUEPRI register with the specified priority
<i>CSL_EDMA3_CMD_QUETHRESHOLD_SET</i>	(Arg: <i>CSL_Edma3CmdQueThr*</i>) Programming QUEUE Threshold levels
<i>CSL_EDMA3_CMD_ERROR_EVAL</i>	(Arg: <i>#None</i>) Sets the EVAL bit in the EEVAL register
<i>CSL_EDMA3_CMD_INTRPEND_CLEAR</i>	(Arg: <i>CSL_Edma3CmdIntr*</i>) Clears specified (Bitmask) pending interrupt at Module/Region Level
<i>CSL_EDMA3_CMD_INTR_ENABLE</i>	(Arg: <i>CSL_Edma3CmdIntr*</i>) Enables specified interrupts (BitMask) at Module/Region Level
<i>CSL_EDMA3_CMD_INTR_DISABLE</i>	(Arg: <i>CSL_Edma3CmdIntr*</i>) Disables specified interrupts (BitMask) at Module/Region Level
<i>CSL_EDMA3_CMD_INTR_EVAL</i>	(Arg: <i>#Int*</i>) Interrupt Evaluation asserted for the Module/Region
<i>CSL_EDMA3_CMD_CTRLERROR_CLEAR</i>	(Arg: <i>CSL_Edma3CtrlErrStat*</i>)
<i>CSL_EDMA3_CMD_EVENTMISSED_CLEAR</i>	(Arg: <i>#CSL_BitMask32*</i>) Pointer to an array of 3 elements, where element0 refers to the EMR register to be cleared, element1 refers to the EMRH register to be cleared, element2 refers to the QEMR register to be cleared.

4.4.4 CSL_Edma3HwStatusQuery

enum CSL_Edma3HwStatusQuery
MODULE Level Queries.

Enumeration values:

<i>CSL_EDMA3_QUERY_CTRLERROR</i>	(Arg: <i>CSL_Edma3CtrlErrStat*</i>)Return Controller Error
<i>CSL_EDMA3_QUERY_INTRPEND</i>	(Arg: <i>CSL_Edma3CmdIntr*</i>)Return pend status of specified interrupt
<i>CSL_EDMA3_QUERY_EVENTMISSED</i>	(Arg: <i>#CSL_BitMask32*</i>)Returns Miss Status of all Channels Pointer to an array of 3 elements, where element0 refers to the EMR register, element1 refers to the EMRH register, element2 refers to the QEMR register
<i>CSL_EDMA3_QUERY_QUESTATUS</i>	(Arg: <i>CSL_Edma3QueStat*</i>)Returns the Que status
<i>CSL_EDMA3_QUERY_ACTIVITY</i>	(Arg: <i>CSL_Edma3ActivityStat*</i>)Returns the Channel Controller Active Status
<i>CSL_EDMA3_QUERY_INFO</i>	(Arg: <i>CSL_Edma3QueryInfo*</i>)Returns the Channel Controller Information viz. Configuration, Revision Id

4.4.5 CSL_Edma3HwChannelControlCmd

enum CSL_Edma3HwChannelControlCmd
CHANNEL Commands.

Enumeration values:

<i>CSL_EDMA3_CMD_CHANNEL_ENABLE</i>	(Arg: <i>#None</i>)Enables specified Channel
<i>CSL_EDMA3_CMD_CHANNEL_DISABLE</i>	(Arg: <i>#None</i>)Disables specified Channel
<i>CSL_EDMA3_CMD_CHANNEL_SET</i>	(Arg: <i>#None</i>)Manually sets the Channel Event, writes into ESR/ESRH and not ER.NA for QDMA.
<i>CSL_EDMA3_CMD_CHANNEL_CLEAR</i>	(Arg: <i>#None</i>)Manually clears the Channel Event, does not write into ESR/ESRH or ER/ERH but the ECR/ECRH. NA for QDMA.
<i>CSL_EDMA3_CMD_CHANNEL_CLEARERR</i>	(Arg: <i>CSL_Edma3ChannelErr*</i>)In case of DMA channels clears SER/SERH (by writing into SECR/SECRH if "secEvt" and "missed" are both TRUE) and EMR/EMRH (by writing into EMCR/EMCRH if "missed" is TRUE). In case of QDMA channels clears QSER (by writing into QSECR if "ser" and "missed" are both TRUE) and QEMR (by writing into QEMCR if "missed" is TRUE)

4.4.6 CSL_Edma3HwChannelStatusQuery

enum CSL_Edma3HwChannelStatusQuery
CHANNEL Queries.

Enumeration values:
CSL_EDMA3_QUERY_CHANNEL_STATUS

(Arg: *#Bool**) In case of DMA channels returns TRUE if ER/ERH is set, In case of QDMA channels returns TRUE if QER is set

CSL_EDMA3_QUERY_CHANNEL_ERR

(Arg: *CSL_Edma3ChannelErr**) In case of DMA channels, 'missed' is set to TRUE if EMR/EMRH is set, 'secEvt' is set to TRUE if SER/SERH is set. In case of QDMA channels, 'missed' is set to TRUE if QEMR is set, 'secEvt' is set to TRUE if QSER is set. It should be noted that if secEvt ONLY is set to TRUE it may not be a valid error condition

4.5 Macros

#define CSL_EDMA3_ADDRMODE_FIFO 1

Address Mode is such it wraps around after reaching FIFO width

#define CSL_EDMA3_ADDRMODE_INCR 0

Address Mode is incremental

#define CSL_EDMA3_BIDX_MAKE (src, dst)

Value:

```
(Uint32)(\
    CSL_FMK(EDMA3CC_SRC_DST_BIDX_DSTBIDX,(Uint32)dst) \
    | CSL_FMK(EDMA3CC_SRC_DST_BIDX_SRCBIDX,(Uint32)src)\
)
```

Used for creating the B index entry in the parameter ram

#define CSL_EDMA3_CIDX_MAKE (src, dst)

Value:

```
(Uint32)(\
    CSL_FMK(EDMA3CC_SRC_DST_CIDX_DSTCIDX,(Uint32)dst) \
    | CSL_FMK(EDMA3CC_SRC_DST_CIDX_SRCIDX,(Uint32)src)\
)
```

Used for creating the C index entry in the parameter ram

#define CSL_EDMA3_CNT_MAKE (aCnt, bCnt)

Value:

```
(Uint32)(\
    CSL_FMK(EDMA3CC_A_B_CNT_ACNT,aCnt) \
    | CSL_FMK(EDMA3CC_A_B_CNT_BCNT,bCnt)\
)
```

Used for creating the A, B Count entry in the parameter ram

#define CSL_EDMA3_DMACHANNELSETUP_DEFAULT

Value:

```
{
    \
    { CSL_EDMA3_QUE_0,0 }, \
    { CSL_EDMA3_QUE_0,1 }, \
    { CSL_EDMA3_QUE_0,2 }, \
    { CSL_EDMA3_QUE_0,3 }, \
    { CSL_EDMA3_QUE_0,4 }, \
    { CSL_EDMA3_QUE_0,5 }, \
    { CSL_EDMA3_QUE_0,6 }, \
    { CSL_EDMA3_QUE_0,7 }, \
    { CSL_EDMA3_QUE_0,8 }, \
    { CSL_EDMA3_QUE_0,9 }, \
}
```

```

{CSL_EDMA3_QUE_0,10}, \
{CSL_EDMA3_QUE_0,11}, \
{CSL_EDMA3_QUE_0,12}, \
{CSL_EDMA3_QUE_0,13}, \
{CSL_EDMA3_QUE_0,14}, \
{CSL_EDMA3_QUE_0,15}, \
{CSL_EDMA3_QUE_0,16}, \
{CSL_EDMA3_QUE_0,17}, \
{CSL_EDMA3_QUE_0,18}, \
{CSL_EDMA3_QUE_0,19}, \
{CSL_EDMA3_QUE_0,20}, \
{CSL_EDMA3_QUE_0,21}, \
{CSL_EDMA3_QUE_0,22}, \
{CSL_EDMA3_QUE_0,23}, \
{CSL_EDMA3_QUE_0,24}, \
{CSL_EDMA3_QUE_0,25}, \
{CSL_EDMA3_QUE_0,26}, \
{CSL_EDMA3_QUE_0,27}, \
{CSL_EDMA3_QUE_0,28}, \
{CSL_EDMA3_QUE_0,29}, \
{CSL_EDMA3_QUE_0,30}, \
{CSL_EDMA3_QUE_0,31}, \
{CSL_EDMA3_QUE_0,32}, \
{CSL_EDMA3_QUE_0,33}, \
{CSL_EDMA3_QUE_0,34}, \
{CSL_EDMA3_QUE_0,35}, \
{CSL_EDMA3_QUE_0,36}, \
{CSL_EDMA3_QUE_0,37}, \
{CSL_EDMA3_QUE_0,38}, \
{CSL_EDMA3_QUE_0,39}, \
{CSL_EDMA3_QUE_0,40}, \
{CSL_EDMA3_QUE_0,41}, \
{CSL_EDMA3_QUE_0,42}, \
{CSL_EDMA3_QUE_0,43}, \
{CSL_EDMA3_QUE_0,44}, \
{CSL_EDMA3_QUE_0,45}, \
{CSL_EDMA3_QUE_0,46}, \
{CSL_EDMA3_QUE_0,47}, \
{CSL_EDMA3_QUE_0,48}, \
{CSL_EDMA3_QUE_0,49}, \
{CSL_EDMA3_QUE_0,50}, \
{CSL_EDMA3_QUE_0,51}, \
{CSL_EDMA3_QUE_0,52}, \
{CSL_EDMA3_QUE_0,53}, \
{CSL_EDMA3_QUE_0,54}, \
{CSL_EDMA3_QUE_0,55}, \
{CSL_EDMA3_QUE_0,56}, \
{CSL_EDMA3_QUE_0,57}, \
{CSL_EDMA3_QUE_0,58}, \
{CSL_EDMA3_QUE_0,59}, \
{CSL_EDMA3_QUE_0,60}, \
{CSL_EDMA3_QUE_0,61}, \
{CSL_EDMA3_QUE_0,62}, \
{CSL_EDMA3_QUE_0,63} \
}

```

DMA Channel Setup

#define CSL_EDMA3_FIFOWIDTH_128BIT 4
128 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_16BIT 1
16 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_256BIT 5
256 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_32BIT 2
32 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_64BIT 3
64 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_8BIT 0
8 bit FIFO Width

#define CSL_EDMA3_FIFOWIDTH_NONE 0
Only for ease

#define CSL_EDMA3_ITCCH_DIS 0
Intermediate transfer completion chaining disable

#define CSL_EDMA3_ITCCH_EN 1
Intermediate transfer completion chaining enable

#define CSL_EDMA3_ITCINT_DIS 0
Intermediate transfer completion interrupt disable

#define CSL_EDMA3_ITCINT_EN 1
Intermediate transfer completion interrupt enable

#define CSL_EDMA3_LINK_DEFAULT 0xFFFF
Link to a Null Param set

#define CSL_EDMA3_LINK_NULL 0xFFFF
Link to a Null Param set

#define CSL_EDMA3_LINKBCNTRLD_MAKE (link, bCntRld)

Value:

```
(Uint32)(\
    CSL_FMK(EDMA3CC_LINK_BCNTRLD_LINK,(Uint32)link) \
    | CSL_FMK(EDMA3CC_LINK_BCNTRLD_BCNTRLD,bCntRld)\
)
```

Used for creating the link and B count reload entry in the parameter ram

```

#define CSL_EDMA3_OPT_MAKE ( itcchEn,
                                tcchEn,
                                itcintEn,
                                tcintEn,
                                tcc,
                                tccMode,
                                fwid,
                                stat,
                                syncDim,
                                dam,
                                sam )

```

Value:

```

( Uint32 ) ( \
    CSL_FMKR(23,23,itcchEn) \
    | CSL_FMKR(22,22,tcchEn) \
    | CSL_FMKR(21,21,itcintEn) \
    | CSL_FMKR(20,20,tcintEn) \
    | CSL_FMKR(17,12,tcc) \
    | CSL_FMKR(11,11,tccMode) \
    | CSL_FMKR(10,8,fwid) \
    | CSL_FMKR(3,3,stat) \
    | CSL_FMKR(2,2,syncDim) \
    | CSL_FMKR(1,1,dam) \
    | CSL_FMKR(0,0,sam) )

```

Used for creating the options entry in the parameter ram

#define CSL_EDMA3_QDMACHANNELSETUP_DEFAULT
Value:

```

{ \
    { CSL_EDMA3_QUE_0,64,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,65,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,66,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,67,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,68,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,69,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,70,CSL_EDMA3_TRIGWORD_DEFAULT }, \
    { CSL_EDMA3_QUE_0,71,CSL_EDMA3_TRIGWORD_DEFAULT } \
}

```

QDMA Channel Setup

#define CSL_EDMA3_STATIC_DIS 0

Disable Static

#define CSL_EDMA3_STATIC_EN 1

Enable Static

```
#define CSL_EDMA3_SYNC_A 0
A synchronized transfer

#define CSL_EDMA3_SYNC_AB 1
AB synchronized transfer

#define CSL_EDMA3_TCC_EARLY 1
Early Completion

#define CSL_EDMA3_TCC_NORMAL 0
Normal Completion

#define CSL_EDMA3_TCCH_DIS 0
Transfer completion chaining disable

#define CSL_EDMA3_TCCH_EN 1
Transfer completion chaining enable

#define CSL_EDMA3_TCINT_DIS 0
Transfer completion interrupt disable

#define CSL_EDMA3_TCINT_EN 1
Transfer completion interrupt enable

#define CSL_EDMA3_TRIGWORD_DEFAULT 7
Last trigger word in a QDMA parameter set
```

Chapter 5

McBSP MODULE

Topics

<u>5.1 Overview</u>
<u>5.2 Functions</u>
<u>5.3 Data Structures</u>
<u>5.4 Enumerations</u>
<u>5.5 Macros</u>

5.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within McBSP module.

This multiple high-speed multichannel buffered serial ports (McBSPs) that allow direct interface to codecs and other devices in a system. The McBSP consists of a data path and a control path that connect to external devices. Separate pins for transmission and reception communicate data to these external devices. Four other pins communicate control information (clocking and frame synchronization). The device communicates to the McBSP using 32-bit-wide control registers accessible via the internal peripheral bus.

Data is communicated to devices interfacing to the McBSP via the data transmit (DX) pin for transmission and via the data receive (DR) pin for reception. Control information (clocking and frame synchronization) is communicated via CLKX, CLKR, FSX, and FSR.

The following are McBSP features supported:

- Full-Duplex communication
- Double buffered data registers which allow a continuous data stream.
- Independent framing and clocking for receive and transmit.
- External shift clock generation or an internal programmable frequency shift clock.
- Autobuffering capability through DMA controller
- A wide selection of data sizes² including 8-, 12-, 16-, 20-, 24-, or 32-bits
- 8-bit data transfers with LSB or MSB first
- Programmable polarity for both frame synchronization and data clocks
- Highly programmable internal clock and frame generation
- Support A-bis mode in normal/32/128 mode (R/XEMODE=0)
- Direct interface to industry standard Codecs, Analog Interface Chips (AICs), and other serially connected A/D and D/A devices.
- Supporting fractional T1/E1. Direct interface to:
 - T1/E1 framers
 - MVIP switching compatible and ST-BUS compliant devices including:
 - MVIP framers
 - H.100 framers
 - SCSA framers
 - IOM-2 compliant devices
 - AC97 compliant devices. (The necessary multi-phase frame synchronization provided.)
 - IIS compliant devices
 - SPI devices

5.2 Functions

This section lists the functions available in the McBSP module.

5.2.1 CSL_mcbbspInit

CSL_Status CSL_mcbbspInit ([CSL_McbbspContext](#) * *pContext*)

Description

This is the initialization function for the McBSP CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use

Arguments

pContext	Pointer to module-context. As McBSP doesn't have any context based information user is expected to pass NULL.
----------	---

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
...
CSL_mcbbspInit(NULL);
...
```

5.2.2 CSL_mcbbspOpen

[CSL_McbbspHandle](#) CSL_mcbbspOpen ([CSL_McbbspObj](#) * *pMcbbspObj*,
CSL_InstNum *mcbbspNum*,
[CSL_McbbspParam](#) * *pMcbbspParam*,
CSL_Status * *pStatus*
)

Description

This function opens the McBSP CSL. It returns a handle to the McBSP instance. This handle is passed to all other CSL APIs, as the reference to the McBSP instance. The device can be re-opened anytime after it has been normally closed, if so required.

Arguments

<code>mcbObj</code>	McBSP Module Object pointer
<code>mcbNum</code>	Instance of McBSP to be opened
<code>pMcbParam</code>	Parameter for McBSP
<code>status</code>	Status of the function call

Return Value `CSL_McbHandle`

- Valid McBSP handle will be returned if status value is equal to `CSL_SOK`.

Pre Condition

The McBSP must be successfully initialized via `CSL_mcbInit()` before calling this function.

Post Condition

- The status is returned in the status variable. If status returned is
 - `CSL_SOK` - Valid McBSP handle is returned.
 - `CSL_ESYS_FAIL` - The McBSP instance is invalid.
 - `CSL_ESYS_INVPARAMS` - The param passed is invalid
- McBSP object structure is populated.

Modifies

- The status variable
- McBSP object structure

Example

```

CSL_McbHandle    hMcb;
CSL_McbObj      mcbObj;
CSL_McbHwSetup  mcbSetup;
CSL_Status      status;
...
hMcb = CSL_mcbOpen(&mcbObj, CSL_MCBSP_0, NULL, &status);
...

```

5.2.3 CSL_mcbClose

CSL_Status `CSL_mcbClose` ([CSL_McbHandle](#) *hMcb*)

Description

Unreserves the McBSP identified by the handle passed. This is a module level close required to invalidate the module handle. The module handle must not be used after this API call.

Arguments

<code>hMcb</code>	McBSP handle returned by successful 'open'
-------------------	--

Return Value `CSL_Status`

- `CSL_SOK` – McBSP closed successfully

-
- CSL_ESYS_BADHANDLE - The handle passed is invalid
 - CSL_ESYS_INVPARAMS - The param passed is invalid

Pre Condition

CSL_mcbSPInit() and CSL_mcbSPOpen() must be called successfully in that order before CSL_mcbSPClose() can be called.

Post Condition

The McBSP CSL APIs can not be called until the McBSP CSL is reopened again using CSL_mcbSPOpen().

Modifies

CSL_mcbSPObj structure instance values

Example

```
CSL_McbSPHandle  hMcbSP;
CSL_McbSPObj     mcbSPObj;
CSL_Status       status;
hMcbSP = CSL_mcbSPOpen (&mcbSPObj, CSL_MCBSP_0, NULL, &status);
...
CSL_mcbSPClose(hMcbSP);
```

5.2.4 CSL_mcbSPHwSetup

```
CSL_Status CSL_mcbSPHwSetup ( CSL\_McbSPHandle          hMcbSP,
                             CSL\_McbSPHwSetup *         setup
                             )
```

Description

This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure, refer CSL_McbSPHwSetup.

Arguments

<i>hMcbSP</i>	McBSP handle returned by successful 'open'
<i>setup</i>	Pointer to setup structure

Return Value CSL_Status

- CSL_SOK - Hwsetup is successfully completed
- CSL_ESYS_INVPARAMS - The param passed is invalid
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

CSL_mcbSPInit() and CSL_mcbSPOpen() must be called successfully in that order before CSL_mcbSPHwSetup() can be called.

Post Condition

McBSP registers are configured according to the hardware setup parameters.

Modifies

McBSP registers

Example

```

CSL_mcbbspHandle    hMcbbsp;
CSL_McbbspHwSetup   hwSetup;    ...

// Init Successfully done
...
// Open Successfully done
...
hwSetup.global= &glbSetup;
hwSetup.rxdataset = &rxDataSetup;
hwSetup.txdataset= &txDataSetup;
hwSetup.clkset = &clkSetup;
hwSetup.mulCh = &mulChSetup;
hwSetup.emumode= CSL_MCBSP_EMU_FREERUN;
hwSetup.extendSetup=NULL;

CSL_mcbbspHwSetup(hMcbbsp, &hwSetup);
...

```

5.2.5 CSL_mcbbspHwSetupRaw

CSL_Status CSL_mcbbspHwSetupRaw ([CSL_McbbspHandle](#) *hMcbbsp*,
[CSL_McbbspConfig](#) * *config*
)

Description

This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values.

Arguments

<i>hMcbbsp</i>	Handle to the McBSP instance
<i>config</i>	Pointer to config structure

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

Both CSL_mcbbspInit() and CSL_mcbbspOpen() must be called successfully in that order before this function can be called.

Post Condition

The registers of the specified McBSP instance will be configured according to value passed.

Modifies

Hardware registers of the specified McBSP instance.

Example

```

    CSL_McbspHandle    hMcbsp;
    CSL_McbspConfig    config = CSL_MCBSP_CONFIG_DEFAULTS;
    CSL_Status          status;

    // Init Successfully done
    ...
    // Open Successfully done
    ...

    status = CSL_mcbspHwSetupRaw (hMcbsp, &config);
    ...

```

5.2.6 CSL_mcbspRead

```

CSL_Status CSL_mcbspRead      ( CSL\_McbspHandle          hMcbsp,
                               CSL\_McbspWordLen       wordLen,
                               void *                data
                               )

```

Description

This function reads the data from McBSP. The word length for the read operation is specified using *wordLen* argument. According to this word length, appropriate amount of data will be read in the data object (variable); the pointer to which is passed as the third argument.

Arguments

<i>hMcbsp</i>	Handle to the McBSP instance
<i>wordLen</i>	Word length of data to be read in
<i>data</i>	Pointer to data object (variable) that will hold the read data

Return Value CSL_Status

- CSL_SOK – Read data successful
- CSL_EMCBSP_INVSIZE - Invalid Word length

Pre Condition

CSL_mcbspInit() and CSL_mcbspOpen() must be called successfully in that order. Then CSL_mcbspHwSetup() must be called successfully before CSL_mcbspRead() can be called.

Post Condition

None

Modifies

McBSP registers

Example

```

    Uint16    inData;
    CSL_Status status;

```

```

CSL_McbbspHandle hMcbbsp;
...
// McBSP object defined and HwSetup structure defined and
initialized
...

// Init, Open, HwSetup successfully done in that order
...
// McBSP SRG, Frame sync, RCV taken out of reset in that order
...
status = CSL_mcbbspRead(hMcbbsp,
                        CSL_MCBSP_WORDLEN_16
                        &inData);

```

5.2.7 CSL_mcbbspWrite

```

CSL_Status CSL_mcbbspWrite      ( CSL\_McbbspHandle          hMcbbsp,
                                CSL\_McbbspWordLen       wordLen,
                                void *                data
                                )

```

Description

This function transmits the data from McBSP. The word length for the write operation is specified using *wordLen* argument. According to this word length, the appropriate amount of data will be transmitted from the data object (variable); the pointer to which is passed as the third argument.

Arguments

<i>hMcbbsp</i>	Handle to the McBSP instance
<i>wordLen</i>	Word length of data to be transmitted
<i>data</i>	Pointer to data object (variable) that holds the data to be sent out

Return Value CSL_Status

- CSL_SOK - Write data successful
- CSL_EMCBSP_INVSIZE - Invalid Word length

Pre Condition

CSL_mcbbspInit() and CSL_mcbbspOpen() must be called successfully in that order. Then CSL_mcbbspHwSetup() must be called successfully before CSL_mcbbspWrite() can be called.

Post Condition

Data is written to DXR register

Modifies

McBSP registers

Example

```

Uint16          outData;
CSL_Status      status;
CSL_McbbspHandle hMcbbsp;

```

```

...
// McBSP object defined and HwSetup structure defined and
initialized
...

// Init, Open, HwSetup successfully done in that order
...
// McBSP SRG, Frame sync, XMT taken out of reset in that order
...
outData = 0x1234;
status = CSL_mcbbspWrite(hMcbbsp,
                        CSL_MCBSP_WORDLEN_16
                        &outData);

```

5.2.8 CSL_mcbbspWrite

```

void CSL_mcbbspWrite      ( CSL\_McbbspHandle      hMcbbsp,
                           CSL\_BitMask16      outputSel,
                           Uint16      outputData
                           )

```

Description

This function sends the data using McBSP pin, which is configured as general purpose output. The 16-bit data transmitted is specified by 'outputData' argument. McBSP pin to use in this write operation is identified by the second argument.

Arguments

<i>hMcbbsp</i>	McBSP handle returned by successful 'open'
<i>outputSel</i>	McBSP pin to be used as general purpose output
<i>outputData</i>	16 bit output data to be transmitted

Return Value

None

Pre Condition

CSL_mcbbspInit() and CSL_mcbbspOpen() must be called successfully in that order before CSL_mcbbspWrite() can be called.

Post Condition

None

Modifies

McBSP registers

Example

```

Uint16      outData;
CSL_Status  status;
CSL_McbbspHandle  hMcbbsp;
...
// McBSP object defined and HwSetup structure defined and
initialized

```

```

...
// Init, Open, HwSetup successfully done in that order
...
outData = 1;
CSL_mcbspIoWrite(hMcbsp, CSL_MCBSP_IO_CLKX, outData);
...

```

5.2.9 CSL_mcbspIoRead

```

Uint16 CSL_mcbspIoRead ( CSL\_McbspHandle hMcbsp,
                           CSL_BitMask16 inputSel
                           )

```

Description

This function reads the data from McBSP pin, which is configured as general purpose input. The 16-bit data read from this pin is returned by this API. McBSP pin to use in this read operation is identified by the second argument.

Arguments

<code>hMcbsp</code>	McBSP handle returned by successful 'open'
<code>inputSel</code>	McBSP pin to be used as general purpose input

Return Value **Uint16**

- Data read from the pin

Pre Condition

CSL_mcbspInit() and CSL_mcbspOpen() must be called successfully in that order before CSL_mcbspIoRead() can be called.

Post Condition

None

Modifies

None

Example

```

Uint16          inData;
Uint16          clkx_data;
Uint16          clkr_data;
CSL_Status      status;
CSL_BitMask16  inMask;
CSL_McbspHandle hMcbsp;
...
// McBSP object defined and HwSetup structure defined and
// initialized
...

// Init, Open, HwSetup successfully done in that order
...
inMask = CSL_MCBSP_IO_CLKX | CSL_MCBSP_IO_CLKR;

```

```

inData = CSL_mcbspIoRead(hMcbbsp, inMask);
if ((inData & CSL_MCBSP_IO_CLKX) != 0) clkx_data = 1;
else
clkx_data = 0;
if ((inData & CSL_MCBSP_IO_CLKR) != 0) clkr_data = 1;
else
clkr_data = 0;
...

```

5.2.10 CSL_mcbspHwControl

```

CSL_Status CSL_mcbspHwControl ( CSL\_McbbspHandle          hMcbbsp,
                                CSL\_McbbspControlCmd       cmd,
                                void *                      arg
                                )

```

Description

This function takes an input control command with an optional argument and accordingly controls the operation/configuration of McBSP.

Arguments

<code>hMcbbsp</code>	McBSP handle returned by successful 'open'
<code>cmd</code>	Control command
<code>arg</code>	Optional argument as per the control command

Return Value CSL_Status

- CSL_SOK - Command successful
- CSL_ESYS_INVCMD - The Command passed is invalid
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

CSL_mcbspInit() and CSL_mcbspOpen() must be called successfully in that order before CSL_mcbspHwControl() can be called.

Post Condition

McBSP registers are configured according to the command passed.

Modifies

McBSP registers determined by the command

Example

```

CSL_Status          status;
CSL_BitMask16       ctrlMask;
CSL_McbbspHandle     hMcbbsp;
...
// McBSP object defined and HwSetup structure defined and
// initialized
...

```

```

// Init successfully done
...
// Open successfully done
...
// HwSetup sucessfully done
...
// McBSP SRG and Frame sync taken out of reset
...

ctrlMask = CSL_MCBSP_CTRL_RX_ENABLE |
            CSL_MCBSP_CTRL_TX_ENABLE;
status = CSL_mcbbspHwControl( hMcbbsp,
                             CSL_MCBSP_CMD_RESET_CONTROL,
                             &ctrlMask);

...

```

5.2.11 CSL_mcbbspGetHwStatus

```

CSL_Status CSL_mcbbspGetHwStatus ( CSL\_McbbspHandle      hMcbbsp,
                                   CSL\_McbbspHwStatusQuery myQuery,
                                   void *                      response
                                   )

```

Description

This function gets the status of different operations or some setup-parameters of MCBSP. The status is returned through the third parameter.

Arguments

<i>hMcbbsp</i>	McBSP handle returned by successful 'open'
<i>myQuery</i>	Query command
<i>response</i>	Response from the query. Pointer to appropriate object corresponding to the query command needs to be passed here

Return Value CSL_Status

- CSL_SOK - Query successful
- CSL_ESYS_INVQUERY - The Query passed is invalid
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

CSL_mcbbspInit() and CSL_mcbbspOpen() must be called successfully in that order before CSL_mcbbspGetHwStatus() can be called.

Post Condition

None

Modifies

Third parameter "response" vlaue

Example

```

CSL_McbbspHandle    hMcbbsp;
CSL_Status          status;
Uint16              response;
...
status = CSL_mcbbspGetHwStatus(hMcbbsp,
                               CSL_MCBSP_QUERY_DEV_STATUS,
                               &response);

if (response & CSL_MCBSP_RRDY)
{
    // Receiver is ready to with new data
    ...
}
...

```

5.2.12 CSL_mcbbspGetHwSetup

```

CSL_Status CSL_mcbbspGetHwSetup ( CSL\_McbbspHandle      hMcbbsp,
                                CSL\_McbbspHwSetup * myHwSetup
                                )

```

Description

This function gets the status of some or all of the setup-parameters of McBSP. To get the status of complete McBSP h/w setup, all the sub-structure pointers inside the main HwSetup structure, should be non-NULL.

Arguments

<i>hMcbbsp</i>	McBSP handle returned by successful 'open'
<i>myHwSetup</i>	Pointer to CSL_McbbspHwSetup structure

Return Value CSL_Status

- CSL_SOK - Get hwsetup successful
- CSL_ESYS_PARAM - The parameter passed is invalid
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

CSL_mcbbspInit() and CSL_mcbbspOpen() must be called successfully in that order before CSL_mcbbspGetHwSetup() can be called.

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

None

Example

```

CSL_McbbspHandle    hMcbbsp;
CSL_Status          status;

CSL_McbbspGlobalSetup gblSetup;

```

```

CSL_McbSpClkSetup clkSetup;
CSL_McbSpEmu emuMode;
CSL_McbSpHwSetup readSetup = {
    &gblSetup,
    NULL,      // RX Data-setup structure if not required
    NULL,      // TX Data-setup structure if not required
    &clkSetup,
    NULL,      // Multichannel-setup structure if not required
                emuMode
};
...
status = CSL_mcbSpGetHwSetup(hMcbSp, &readSetup);
...

```

5.2.13 CSL_mcbSpGetBaseAddress

```

CSL_Status CSL_mcbSpGetBaseAddress ( CSL_InstNum      mcbSpNum,
                                     CSL\_McbSpParam *  pMcbSpParam,
                                     CSL\_McbSpBaseAddress * pBaseAddress
                                     )

```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_mcbSpOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

mcbSpNum	Specifies the instance of the McBSP to be opened
pMcbSpParam	Module specific parameters
pBaseAddress	Pointer to baseaddress structure

Return Value CSL_Status

- CSL_SOK - Successfully retrieved base address
- CSL_ESYS_FAIL - The instance number is invalid.
- CSL_ESYS_INVPARAMS - Invalid Parameter

Pre Condition

None

Post Condition

Base Address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status          status;  
CSL_McbspBaseAddress baseAddress;  
  
...  
status = CSL_mcbSPGetBaseAddress(CSL_MCBSP_0, NULL,  
                                &baseAddress);
```

5.3 Data Structures

This section lists the data structures available in the McBSP module.

5.3.1 CSL_McbObj

Detailed Description

This structure/object holds the context of the instance of McBSP opened using CSL_mcbOpen() function. Pointer to this object is passed as McBSP Handle to all McBSP CSL APIs. CSL_mcbOpen() function initializes this structure based on the parameters passed

Field Documentation

CSL_InstNum CSL_McbObj::perNum

Instance of McBSP being referred by this object

CSL_McbRegsOvly CSL_McbObj::regs

Pointer to the register overlay structure of the McBSP

5.3.2 CSL_McbConfig

Detailed Description

This is configuration structure of McBSP. This is used to configure McBSP using CSL_HwSetupRaw function. This is a structure of register values, rather than a structure of register field values like CSL_McbHwSetup.

Field Documentation

volatile Uint32 CSL_McbConfig::MCR

Multichannel Control Register

volatile Uint32 CSL_McbConfig::PCR

Pin Control Register

volatile Uint32 CSL_McbConfig::RCERE0

Receive Channel Enable Register for Partition A and B

volatile Uint32 CSL_McbConfig::RCERE1

Receive Channel Enable Register for Partition C and D

volatile Uint32 CSL_McbConfig::RCERE2

Receive Channel Enable Register for Partition E and F

volatile Uint32 CSL_McbConfig::RCERE3

Receive Channel Enable Register for Partition G and H

volatile Uint32 CSL_McbConfig::RCR

Receive Control Register

volatile Uint32 CSL_McbConfig::SPCR

Serial Port Control Register

volatile UInt32 CSL_McbspConfig::SRGR
Sample Rate Generator Register

volatile UInt32 CSL_McbspConfig::XCERE0
Transmit Channel Enable Register for Partition A and B

volatile UInt32 CSL_McbspConfig::XCERE1
Transmit Channel Enable Register for Partition C and D

volatile UInt32 CSL_McbspConfig::XCERE2
Transmit Channel Enable Register for Partition E and F

volatile UInt32 CSL_McbspConfig::XCERE3
Transmit Channel Enable Register for Partition G and H

volatile UInt32 CSL_McbspConfig::XCR
Transmit Control Register

5.3.3 CSL_McbspContext

Detailed Description

This contains McBSP specific context information. Present implementation doesn't have any Context information.

Field Documentation

UInt16 CSL_McbspContext::contextInfo

McBSP context information. The declaration is just a placeholder for future implementation.

5.3.4 CSL_McbspHwSetup

Detailed Description

This is the hardware setup structure for configuring McBSP using CSL_mcbspHwSetup() function.

Field Documentation

[CSL_McbspClkSetup](#)* **CSL_McbspHwSetup::clkset**
Clock configuration parameters

[CSL_McbspEmu](#) **CSL_McbspHwSetup::emumode**
Emulation mode parameters

void* CSL_McbspHwSetup::extendSetup
Any extra parameters, for future use

[CSL_McbspGlobalSetup](#)* **CSL_McbspHwSetup::global**
Global configuration parameters

[CSL_McbspMulChSetup](#)* **CSL_McbspHwSetup::mulCh**
Multichannel mode configuration parameters

[CSL_McbspDataSetup](#)* **CSL_McbspHwSetup::rxdataset**
RCV data setup related parameters

[CSL_McbspDataSetup](#)* **CSL_McbspHwSetup::txdataset**
XMT data setup related parameters

5.3.5 CSL_McbspParam

Detailed Description

This contains the McBSP specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_McbspParam::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

5.3.6 CSL_McbspBaseAddress

Detailed Description

This will have the base-address information for the peripheral instance

Field Documentation

CSL_McbspRegsOvly CSL_McbspBaseAddress::regs

Base-address of the Configuration registers of McBSP

5.3.7 CSL_McbspBlkAssign

Detailed Description

Pointer to this structure is used as the third argument in CSL_mcbspHwControl() for block assignment in multichannel mode

Field Documentation

[CSL_McbspBlock](#) **CSL_McbspBlkAssign::block**

Block to choose

[CSL_McbspPartition](#) **CSL_McbspBlkAssign::partition**

Partition to choose

5.3.8 CSL_McbspChanControl

Detailed Description

Pointer to this structure is used as the third argument in CSL_mcbspHwControl() for channel control operations (Enable/Disable TX/RX) in multichannel mode.

Field Documentation

UInt16 CSL_McbspChanControl::channelNo

Channel number to control

[CSL_McbSpChCtrl](#) CSL_McbSpChanControl::operation
Control operation

5.3.9 CSL_McbSpDataSetup

Detailed Description

This is a sub-structure in CSL_McbSpHwSetup. This structure is used for configuring input/output data related parameters.

Field Documentation

[CSL_McbSpCompand](#) CSL_McbSpDataSetup::compand
Companding options

[CSL_McbSpDataDelay](#) CSL_McbSpDataSetup::dataDelay
Data delay in number of bits

UInt16 CSL_McbSpDataSetup::frmLength1
Number of words per frame in phase 1

UInt16 CSL_McbSpDataSetup::frmLength2
Number of words per frame in phase 2

[CSL_McbSpFrmSync](#) CSL_McbSpDataSetup::frmSyncIgn
Frame Sync ignore

[CSL_McbSpPhase](#) CSL_McbSpDataSetup::numPhases
Number of phases in a frame

[CSL_McbSpRjustDxena](#) CSL_McbSpDataSetup::rjust_dxenable
Controls DX delay for XMT or sign-extension and justification for RCV

[CSL_McbSpWordLen](#) CSL_McbSpDataSetup::wordLength1
Number of bits per word in phase 1

[CSL_McbSpWordLen](#) CSL_McbSpDataSetup::wordLength2
Number of bits per word in phase 2

[CSL_McbSpBitReversal](#) CSL_McbSpDataSetup::wordReverse
32-bit reversal feature

5.3.10 CSL_McbSpClkSetup

Detailed Description

This is a sub-structure in CSL_McbSpHwSetup. This structure is used for configuring Clock and Frame Sync generation parameters.

Field Documentation

[CSL_McbSpTxRxClkMode](#) CSL_McbSpClkSetup::clkRxMode
RCV clock mode

[CSL_McbspClkPol](#) **CSL_McbspClkSetup::clkRxPolarity**
RCV clock polarity

[CSL_McbspTxRxClkMode](#) **CSL_McbspClkSetup::clkTxMode**
XMT clock mode

[CSL_McbspClkPol](#) **CSL_McbspClkSetup::clkTxPolarity**
XMT clock polarity

[CSL_McbspFsClkMode](#) **CSL_McbspClkSetup::frmSyncRxMode**
RCV frame sync mode

[CSL_McbspFsPol](#) **CSL_McbspClkSetup::frmSyncRxPolarity**
RCV frame sync polarity

[CSL_McbspFsClkMode](#) **CSL_McbspClkSetup::frmSyncTxMode**
XMT frame sync mode

[CSL_McbspFsPol](#) **CSL_McbspClkSetup::frmSyncTxPolarity**
XMT frame sync polarity

UInt16 CSL_McbspClkSetup::srgClkDivide
SRG divide-down ratio

[CSL_McbspClkPol](#) **CSL_McbspClkSetup::srgClkPolarity**
SRG clock polarity

[CSL_McbspClkgSyncMode](#) **CSL_McbspClkSetup::srgClkSync**
SRG clock synchronization mode

UInt16 CSL_McbspClkSetup::srgFrmPeriod
SRG frame sync period

UInt16 CSL_McbspClkSetup::srgFrmPulseWidth
SRG frame sync pulse width

[CSL_McbspSrgClk](#) **CSL_McbspClkSetup::srgInputClkMode**
SRG input clock mode

[CSL_McbspTxFsMode](#) **CSL_McbspClkSetup::srgTxFrmSyncMode**
SRG XMT frame-synchronization mode

5.3.11 CSL_McbspGlobalSetup

Detailed Description

This is a sub-structure in *CSL_McbspHwSetup*. This structure is used for configuring the parameters global to McBSP

Field Documentation

[CSL_McbspClkStp](#) **CSL_McbspGlobalSetup::clkStopMode**
Clock stop mode

[CSL_McbspDlbMode](#) **CSL_McbspGlobalSetup::dlbMode**
Digital Loopback mode

CSL_McbspIOMode **CSL_McbspGlobalSetup::ioEnableMode**
XMT and RCV IO enable bit

5.3.12 CSL_McbspMulChSetup

Detailed Description

This is a sub-structure in *CSL_McbspHwSetup*. This structure is used for configuring Multichannel mode parameters

Field Documentation

UInt16 CSL_McbspMulChSetup::rxMulChSel
RCV multichannel selection mode

[CSL_McbspPABlk](#) **CSL_McbspMulChSetup::rxPartABlk**
RCV partition A block

[CSL_McbspPBBlk](#) **CSL_McbspMulChSetup::rxPartBBlk**
RCV partition B block

[CSL_McbspPartMode](#) **CSL_McbspMulChSetup::rxPartition**
RCV partition

UInt16 CSL_McbspMulChSetup::txMulChSel
XMT multichannel selection mode

[CSL_McbspPABlk](#) **CSL_McbspMulChSetup::txPartABlk**
XMT partition A block

[CSL_McbspPBBlk](#) **CSL_McbspMulChSetup::txPartBBlk**
XMT partition B block

[CSL_McbspPartMode](#) **CSL_McbspMulChSetup::txPartition**
XMT partition

5.3.13 CSL_McbspPerild

Detailed Description

Pointer to this structure is used as the third argument in *CSL_mcbspGetHwStatus* () for getting revision, type and class info of McBSP

Field Documentation

UInt16 CSL_McbspPerild:: type
Identifies the Type of peripheral

UInt16 CSL_McbspPerild:: devclass
Identifies the class of the McBSP peripheral

UInt16 CSL_McbspPerild:: revision
Identifies the revision level of the McBSP

5.4 Enumerations

5.4.1 CSL_McbbspWordLen

enum CSL_McbbspWordLen

This enumeration contains the word lengths supported on McBSP. Use this symbol for setting Word Length in each Phase for every Frame.

Enumeration values:

<i>CSL_MCBSP_WORDLEN_8</i>	Word Length for Frame is 8
<i>CSL_MCBSP_WORDLEN_12</i>	Word Length for Frame is 12
<i>CSL_MCBSP_WORDLEN_16</i>	Word Length for Frame is 16
<i>CSL_MCBSP_WORDLEN_20</i>	Word Length for Frame is 20
<i>CSL_MCBSP_WORDLEN_24</i>	Word Length for Frame is 24
<i>CSL_MCBSP_WORDLEN_32</i>	Word Length for Frame is 32

5.4.2 CSL_McbbspCompand

enum CSL_McbbspCompand

McBSP companding options - Use this symbol to set Companding related options.

Enumeration values:

<i>CSL_MCBSP_COMPAND_OFF_MSB_FIRST</i>	No companding for msb
<i>CSL_MCBSP_COMPAND_OFF_LSB_FIRST</i>	No companding for lsb
<i>CSL_MCBSP_COMPAND_MULAW</i>	mu-law companding enable for channel
<i>CSL_MCBSP_COMPAND_ALAW</i>	A-law companding enable for channel

5.4.3 CSL_McbbspDataDelay

enum CSL_McbbspDataDelay

Data delay in bits - Use this symbol to set XMT/RCV Data Delay (in bits).

Enumeration values:

<i>CSL_MCBSP_DATADELAY_0_BIT</i>	Sets XMT/RCV Data Delay is 0
<i>CSL_MCBSP_DATADELAY_1_BIT</i>	Sets XMT/RCV Data Delay is 1
<i>CSL_MCBSP_DATADELAY_2_BITS</i>	Sets XMT/RCV Data Delay is 2

5.4.4 CSL_McbbspIntMode

enum CSL_McbbspIntMode

This enumeration contains McBSP Interrupts modes - Use this symbol to set Interrupt mode (i.e. source of interrupt generation). This symbol is used on both RCV and XMT for RINT and XINT generation mode.

Enumeration values:

<i>CSL_MCBSP_INTMODE_ON_READY</i>	Interrupt generated on RRDY of RCV or XRDY of XMT
<i>CSL_MCBSP_INTMODE_ON_EOB</i>	Interrupt generated on end of 16-channel block transfer in multichannel mode

<i>CSL_MCBSP_INTMODE_ON_FSYNC</i>	Interrupt generated on frame sync
<i>CSL_MCBSP_INTMODE_ON_SYNCERR</i>	Interrupt generated on synchronization error

5.4.5 CSL_McbspFsClkMode

enum CSL_McbspFsClkMode

Frame sync clock source - Use this symbol to set the frame sync clock source as internal or external.

Enumeration values:

<i>CSL_MCBSP_FSCLKMODE_EXTERNAL</i>	Frame sync clock source as external
<i>CSL_MCBSP_FSCLKMODE_INTERNAL</i>	Frame sync clock source as internal

5.4.6 CSL_McbspTxRxClkMode

enum CSL_McbspTxRxClkMode

Clock source - Use this symbol to set the clock source as internal or external.

Enumeration values:

<i>CSL_MCBSP_TXRXCLKMODE_EXTERNAL</i>	Clock source as external
<i>CSL_MCBSP_TXRXCLKMODE_INTERNAL</i>	Clock source as internal

5.4.7 CSL_McbspFsPol

enum CSL_McbspFsPol

Frame sync polarity - Use this symbol to set frame sync polarity as active-high or active-low.

Enumeration values:

<i>CSL_MCBSP_FSPOL_ACTIVE_HIGH</i>	Frame sync polarity is active-high
<i>CSL_MCBSP_FSPOL_ACTIVE_LOW</i>	Frame sync polarity is active-low

5.4.8 CSL_McbspClkPol

enum CSL_McbspClkPol

Clock polarity - Use this symbol to set XMT or RCV clock polarity as rising or falling edge.

Enumeration values:

<i>CSL_MCBSP_CLKPOL_TX_RISING_EDGE</i>	XMT clock polarity is rising edge
<i>CSL_MCBSP_CLKPOL_RX_FALLING_EDGE</i>	RCV clock polarity is falling edge
<i>CSL_MCBSP_CLKPOL_SRG_RISING_EDGE</i>	SRG clock polarity is rising edge
<i>CSL_MCBSP_CLKPOL_TX_FALLING_EDGE</i>	XMT clock polarity is falling edge
<i>CSL_MCBSP_CLKPOL_RX_RISING_EDGE</i>	RCV clock polarity is rising edge
<i>CSL_MCBSP_CLKPOL_SRG_FALLING_EDGE</i>	SRG clock polarity is falling edge

5.4.9 CSL_McbspSrgClk

enum CSL_McbspSrgClk

SRG clock source - Use this symbol to select input clock source for Sample Rate Generator.

Enumeration values:

<i>CSL_MCBSP_SRGCLK_CLKS</i>	Input clock source for Sample Rate Generator is CLKS pin
<i>CSL_MCBSP_SRGCLK_CLKCPU</i>	Input clock source for Sample Rate Generator is CPU

5.4.10 CSL_McbspTxFsMode

enum CSL_McbspTxFsMode

This enumeration contains XMT Frame Sync generation mode - Use this symbol to set XMT Frame Sync generation mode.

Enumeration values:

<i>CSL_MCBSP_TXFSMODE_DXRCOPY</i>	Disables the frame sync generation mode
<i>CSL_MCBSP_TXFSMODE_SRG</i>	Enables the frame sync generation mode

5.4.11 CSL_McbspIOMode

enum CSL_McbspIOMode

XMT and RCV IO Mode - Use this symbol to Enable/Disable IO Mode for XMT and RCV.

Enumeration values:

<i>CSL_MCBSP_IOMODE_TXDIS_RXDIS</i>	Disable the both XMT and RCV IO mode
<i>CSL_MCBSP_IOMODE_TXDIS_RXEN</i>	Disable XMT and enable RCV IO mode
<i>CSL_MCBSP_IOMODE_TXEN_RXDIS</i>	Enable XMT and Disable RCV IO mode
<i>CSL_MCBSP_IOMODE_TXEN_RXEN</i>	Enable XMT and enable RCV IO mode

5.4.12 CSL_McbspClkStp

enum CSL_McbspClkStp

Clock Stop Mode - Use this symbol to Enable/Disable Clock Stop Mode.

Enumeration values:

<i>CSL_MCBSP_CLKSTP_DISABLE</i>	Disable the clock stop mode
<i>CSL_MCBSP_CLKSTP_WITHOUT_DELAY</i>	Enable the clock stop mode with out delay
<i>CSL_MCBSP_CLKSTP_WITH_DELAY</i>	Enable the clock stop mode with delay

5.4.13 CSL_McbspPartMode

enum CSL_McbspPartMode

This enumeration contains the multichannel mode partition type - Use this symbol to select the partition type in multichannel mode.

Enumeration values:

<i>CSL_MCBSP_PARTMODE_2PARTITION</i>	Two partition mode
<i>CSL_MCBSP_PARTMODE_8PARTITION</i>	Eight partition multichannel mode

5.4.14 CSL_McbPABlk

enum CSL_McbPABlk

Multichannel mode PartitionA block - Use this symbol to assign Blocks to Partition-A in multichannel mode

Enumeration values:

<i>CSL_MCBSP_PABLK_0</i>	Block 0 for partition A
<i>CSL_MCBSP_PABLK_2</i>	Block 2 for partition A
<i>CSL_MCBSP_PABLK_4</i>	Block 4 for partition A
<i>CSL_MCBSP_PABLK_6</i>	Block 6 for partition A

5.4.15 CSL_McbPBBlk

enum CSL_McbPBBlk

Multichannel mode PartitionB block - Use this symbol to assign Blocks to Partition-B in multichannel mode

Enumeration values:

<i>CSL_MCBSP_PBBLK_1</i>	Block 1 for partition B
<i>CSL_MCBSP_PBBLK_3</i>	Block 3 for partition B
<i>CSL_MCBSP_PBBLK_5</i>	Block 5 for partition B
<i>CSL_MCBSP_PBBLK_7</i>	Block 7 for partition B

5.4.16 CSL_McbEmu

enum CSL_McbEmu

Emulation mode setting - Use this symbol to set the Emulation Mode

Enumeration values:

<i>CSL_MCBSP_EMU_STOP</i>	Emulation mode stop
<i>CSL_MCBSP_EMU_TX_STOP</i>	Emulation mode TX stop
<i>CSL_MCBSP_EMU_FREERUN</i>	Emulation free run mode

5.4.17 CSL_McbPartition

enum CSL_McbPartition

Multichannel mode Partition select - Use this symbol in multichannel mode to select the Partition for assigning a block to

Enumeration values:

<i>CSL_MCBSP_PARTITION_ATX</i>	TX partition for A
<i>CSL_MCBSP_PARTITION_ARX</i>	RX partition for A
<i>CSL_MCBSP_PARTITION_BTXX</i>	TX partition for B
<i>CSL_MCBSP_PARTITION_BRXX</i>	RX partition for B

5.4.18 CSL_McbspBlock

enum CSL_McbspBlock

Multichannel mode Block select - Use this symbol in multichannel mode to select block on which the operation is to be performed

Enumeration values:

<i>CSL_MCBSP_BLOCK_0</i>	Block 0 for multichannel mode
<i>CSL_MCBSP_BLOCK_1</i>	Block 1 for multichannel mode
<i>CSL_MCBSP_BLOCK_2</i>	Block 2 for multichannel mode
<i>CSL_MCBSP_BLOCK_3</i>	Block 3 for multichannel mode
<i>CSL_MCBSP_BLOCK_4</i>	Block 4 for multichannel mode
<i>CSL_MCBSP_BLOCK_5</i>	Block 5 for multichannel mode
<i>CSL_MCBSP_BLOCK_6</i>	Block 6 for multichannel mode
<i>CSL_MCBSP_BLOCK_7</i>	Block 7 for multichannel mode

5.4.19 CSL_McbspChCtrl

enum CSL_McbspChCtrl

Channel control in multichannel mode Use this symbol to enable/disable a channel in multichannel mode. This is a member of CSL_McbspChanControl structure, which is input to CSL_mcbspHwControl() function for CSL_MCBSP_CMD_CHANNEL_CONTROL command.

Enumeration values:

<i>CSL_MCBSP_CHCTRL_TX_ENABLE</i>	TX enable for multichannel mode
<i>CSL_MCBSP_CHCTRL_TX_DISABLE</i>	TX disable for multichannel mode
<i>CSL_MCBSP_CHCTRL_RX_ENABLE</i>	RX enable for multichannel mode
<i>CSL_MCBSP_CHCTRL_RX_DISABLE</i>	RX disable for multichannel mode

5.4.20 CSL_McbspChType

enum CSL_McbspChType

Channel type: TX, RX or both - Use this symbol to select the channel type for CSL_mcbspHwControl()

Enumeration values:

<i>CSL_MCBSP_CHTYPE_RX</i>	Channel type is RX
<i>CSL_MCBSP_CHTYPE_TX</i>	Channel type is TX
<i>CSL_MCBSP_CHTYPE_TXRX</i>	Channel type is TXRX

5.4.21 CSL_McbspDlbMode

enum CSL_McbspDlbMode

Digital Loopback mode selection - Use this symbol to enable/disable digital loopback mode

Enumeration values:

<i>CSL_MCBSP_DLBMODE_OFF</i>	Disable digital loopback mode
<i>CSL_MCBSP_DLBMODE_ON</i>	Enable digital loopback mode

5.4.22 CSL_McbSPPhase

enum CSL_McbSPPhase

Phase count selection - Use this symbol to select number of phases per frame

Enumeration values:

<i>CSL_MCBSP_PHASE_SINGLE</i>	Single phase for frame
<i>CSL_MCBSP_PHASE_DUAL</i>	Dual phase for frame

5.4.23 CSL_McbSPFrmSync

enum CSL_McbSPFrmSync

Frame sync ignore status - Use this symbol to detect or ignore frame synchronization

Enumeration values:

<i>CSL_MCBSP_FRMSYNC_DETECT</i>	Detect frame synchronization
<i>CSL_MCBSP_FRMSYNC_IGNORE</i>	Ignore frame synchronization

5.4.24 CSL_McbSPRjustDxena

enum CSL_McbSPRjustDxena

RJUST or DXENA settings - Use this symbol for setting up RCV sign-extension and justification mode or enabling/disabling XMT DX pin delay

Enumeration values:

<i>CSL_MCBSP_RJUSTDXENA_RJUST_RZF</i>	RCV setting - right justify, fill MSBs with zeros
<i>CSL_MCBSP_RJUSTDXENA_DXENA_OFF</i>	XMT setting - Delay at DX pin disabled
<i>CSL_MCBSP_RJUSTDXENA_RJUST_RSE</i>	RCV setting - right justify, sign-extend the data into MSBs
<i>CSL_MCBSP_RJUSTDXENA_DXENA_ON</i>	XMT setting - Delay at DX pin enabled
<i>CSL_MCBSP_RJUSTDXENA_RJUST_LZF</i>	RCV setting - left justify, fill LSBs with zeros

5.4.25 CSL_McbSPClkgSyncMode

enum CSL_McbSPClkgSyncMode

CLKG sync mode selection - Use this symbol to enable/disable CLKG synchronization when input CLK source for SRGR is external

Enumeration values:

<i>CSL_MCBSP_CLKGSYNCMODE_OFF</i>	Disable CLKG synchronization
<i>CSL_MCBSP_CLKGSYNCMODE_ON</i>	Enable CLKG synchronization

5.4.26 CSL_McbSPRstStat

enum CSL_McbSPRstStat

Tx/Rx reset status - Use this symbol to compare the output of CSL_mcbSPGetHwStatus() for CSL_MCBSP_QUERY_TX_RST_STAT and CSL_MCBSP_QUERY_RX_RST_STAT queries

Enumeration values:

<i>CSL_MCBSP_RSTSTAT_TX_IN_RESET</i>	Disable the XRST bit
<i>CSL_MCBSP_RSTSTAT_RX_IN_RESET</i>	Disable the RRST bit
<i>CSL_MCBSP_RSTSTAT_TX_OUTOF_RESET</i>	Enable the XRST bit
<i>CSL_MCBSP_RSTSTAT_RX_OUTOF_RESET</i>	Enable the RRST bit

5.4.27 CSL_McbSPBitReversal

enum CSL_McbSPBitReversal

McBSP 32-bit reversal feature

Enumeration values:

<i>CSL_MCBSP_32BIT_REVERS_DISABLE</i>	32-bit reversal disabled
<i>CSL_MCBSP_32BIT_REVERS_ENABLE</i>	32-bit reversal enabled. 32-bit data is received LSB first. Word length should be set for 32-bit operation; else operation undefined

5.4.28 CSL_McbSPControlCmd

enum CSL_McbSPControlCmd

This is the set of control commands that are passed to `CSL_mcbSPHwControl()`, with an optional argument type-casted to `void*`. The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<i>CSL_MCBSP_CMD_ASSIGN_BLOCK</i>	Assigns a block to a particular partition in multichannel mode. Parameters: (<i>CSL_McbSPBlkAssign *</i>)
<i>CSL_MCBSP_CMD_CHANNEL_CONTROL</i>	Enables or disables a channel in multichannel mode. Parameters: (<i>CSL_McbSPChanControl *</i>)
<i>CSL_MCBSP_CMD_CLEAR_FRAME_SYNC</i>	Clears frame sync error for XMT or RCV. Parameters: (<i>CSL_McbSPChType *</i>)
<i>CSL_MCBSP_CMD_RESET</i>	Resets all the registers to their power-on default values. Parameters: <i>None</i>
<i>CSL_MCBSP_CMD_RESET_CONTROL</i>	Enable/Disable - Frame Sync, Sample Rate Generator and XMT/RCV Operation. Parameters: (<i>CSL_BitMask16 *</i>)
<i>CSL_MCBSP_CMD_TX_INT_MODE</i>	Configures Transmit INT MODE. Parameters: (<i>CSL_McbSPIntMode*</i>)
<i>CSL_MCBSP_CMD_RX_INT_MODE</i>	Configures Receive INT MODE. Parameters:

(CSL_McbspIntMode*)

5.4.29 CSL_McbspHwStatusQuery

enum CSL_McbspHwStatusQuery

This is the set of query commands to get the status of various operations in McBSP. The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<i>CSL_MCBSP_QUERY_CUR_TX_BLK</i>	Queries the current XMT block. Parameters: (CSL_McbspBlock *)
<i>CSL_MCBSP_QUERY_CUR_RX_BLK</i>	Queries the current RCV block. Parameters: (CSL_McbspBlock *)
<i>CSL_MCBSP_QUERY_DEV_STATUS</i>	Queries the status of RRDY, XRDY, RFULL, XEMPTY, RSYNCERR and XSYNCERR events and returns them in supplied CSL_BitMask16 argument. Parameters: (CSL_BitMask16 *)
<i>CSL_MCBSP_QUERY_TX_RST_STAT</i>	Queries XMT reset status. Parameters: (CSL_McbspRstStat *) Returns: CSL_SOK
<i>CSL_MCBSP_QUERY_RX_RST_STAT</i>	Queries RCV reset status. Parameters: (CSL_McbspRstStat *)
<i>CSL_MCBSP_QUERY_TX_INT_MODE</i>	Queries Transmit INT mode. Parameters: (CSL_McbspIntMode*)
<i>CSL_MCBSP_QUERY_RX_INT_MODE</i>	Queries Receive INT mode. Parameters: (CSL_McbspIntMode*)

5.5 Macros

#define CSL_EMCBSP_INVCNTLCMD (CSL_EMCBSP_FIRST - 0)
Invalid Control Command

#define CSL_EMCBSP_INVMODE (CSL_EMCBSP_FIRST - 5)
Invalid mode to conduct operation

#define CSL_EMCBSP_INVPARAMS (CSL_EMCBSP_FIRST - 2)
Invalid Parameter

#define CSL_EMCBSP_INVQUERY (CSL_EMCBSP_FIRST - 1)
Invalid Query

#define CSL_EMCBSP_INVSIZE (CSL_EMCBSP_FIRST - 3)
Invalid Size

#define CSL_EMCBSP_NOTEXIST (CSL_EMCBSP_FIRST - 4)
'Does not exist'

#define CSL_MCBSP_CLOCKSETUP_DEFAULTS
Value:

```
{
    (CSL_McbbspFsClkMode)CSL_MCBSP_FSCLKMODE_EXTERNAL, \
    (CSL_McbbspFsClkMode)CSL_MCBSP_FSCLKMODE_EXTERNAL, \
    (CSL_McbbspTxRxClkMode)CSL_MCBSP_TXRXCLKMODE_INTERNAL, \
    (CSL_McbbspTxRxClkMode)CSL_MCBSP_TXRXCLKMODE_EXTERNAL, \
    (CSL_McbbspFsPol)0, \
    (CSL_McbbspFsPol)0, \
    (CSL_McbbspClkPol)0, \
    (CSL_McbbspClkPol)0, \
    1, \
    0x40, \
    0xFF, \
    (CSL_McbbspSrgClk)0, \
    (CSL_McbbspClkPol)0, \
    (CSL_McbbspTxFsMode)CSL_MCBSP_TXFSMODE_SRG, \
    (CSL_McbbspClkgSyncMode)CSL_MCBSP_CLKGSYNCMODE_OFF } \
```

Clock Setup defaults

#define CSL_MCBSP_CONFIG_DEFAULTS
Value:

```
{ \
    CSL_MCBSP_SPCR_RESETVAL, \
    CSL_MCBSP_RCR_RESETVAL, \
    CSL_MCBSP_XCR_RESETVAL, \
    CSL_MCBSP_SRGR_RESETVAL, \
    CSL_MCBSP_MCR_RESETVAL, \
    CSL_MCBSP_RCERE0_RESETVAL, \
    CSL_MCBSP_XCERE0_RESETVAL, \
    CSL_MCBSP_PCR_RESETVAL, \
    CSL_MCBSP_RCERE1_RESETVAL, \
    CSL_MCBSP_XCERE1_RESETVAL, \
```

```

        CSL_MCBSP_RCERE2_RESETVAL, \
        CSL_MCBSP_XCERE2_RESETVAL, \
        CSL_MCBSP_RCERE3_RESETVAL, \
        CSL_MCBSP_XCERE3_RESETVAL \
    }

```

Default Values for Config structure

#define CSL_MCBSP_CTRL_FSYNC_DISABLE (64)

To disable Frame Sync Generation in resetControl Function

#define CSL_MCBSP_CTRL_FSYNC_ENABLE (16)

To enable Frame Sync Generation in resetControl Function

#define CSL_MCBSP_CTRL_RX_DISABLE (4)

To disable Receiver in resetControl Function

#define CSL_MCBSP_CTRL_RX_ENABLE (1)

To enable Receiver in resetControl Function

#define CSL_MCBSP_CTRL_SRG_DISABLE (128)

To disable Sample Rate Generator in resetControl Function

#define CSL_MCBSP_CTRL_SRG_ENABLE (32)

To enable Sample Rate Generator in resetControl Function

#define CSL_MCBSP_CTRL_TX_DISABLE (8)

To disable Transmitter in resetControl Function

#define CSL_MCBSP_CTRL_TX_ENABLE (2)

To enable Transmitter in resetControl Function

#define CSL_MCBSP_DATASETUP_DEFAULTS

Value:

```

{
    (CSL_McbbspPhase)CSL_MCBSP_PHASE_SINGLE, \
    (CSL_McbbspWordLen)CSL_MCBSP_WORDLEN_16, \
    1, \
    (CSL_McbbspWordLen)0, \
    0, \
    (CSL_McbbspFrmSync)CSL_MCBSP_FRMSYNC_DETECT, \
    (CSL_McbbspCompand)CSL_MCBSP_COMPAND_OFF_MSB_FIRST, \
    (CSL_McbbspDataDelay)CSL_MCBSP_DATADELAY_0_BIT, \
    (CSL_McbbspRjustDxena)0, \
    (CSL_McbbspBitReversal)CSL_MCBSP_32BIT_REVERS_DISABLE } \

```

Data Setup defaults

#define CSL_MCBSP_EMUMODE_DEFAULT CSL_MCBSP_EMU_STOP

Default Emulation mode - Stop

#define CSL_MCBSP_EXTENDSETUP_DEFAULT NULL

Extend Setup default - NULL

#define CSL_MCBSP_GLOBALSETUP_DEFAULTS

Value:

```

{
    \

```

```

(CSL_McbspIOMode)CSL_MCBSP_IOMODE_TXDIS_RXDIS, \
(CSL_McbspDlbMode)CSL_MCBSP_DLBMODE_OFF, \
(CSL_McbspClkStp)CSL_MCBSP_CLKSTP_DISABLE } \
Global parameters Setup defaults

```

#define CSL_MCBSP_IO_CLKR (8)
I/O Pin Input/Output configuration for CLKR Pin

#define CSL_MCBSP_IO_CLKS (64)
Not Configurable. Always Input.

#define CSL_MCBSP_IO_CLKX (1)
I/O Pin Input/Output configuration for CLKX Pin

#define CSL_MCBSP_IO_DR (32)
Not Configurable. Always Input.

#define CSL_MCBSP_IO_DX (4)
Not Configurable. Always Output.

#define CSL_MCBSP_IO_FSR (16)
I/O Pin Input/Output configuration for FSR Pin

#define CSL_MCBSP_IO_FSX (2)
I/O Pin Input/Output configuration for FSX Pin

#define CSL_MCBSP_MULTICHAN_DEFAULTS
Value:

```

{ \
(CSL_McbspPartMode)CSL_MCBSP_PARTMODE_2PARTITION, \
(CSL_McbspPartMode)CSL_MCBSP_PARTMODE_2PARTITION, \
(Uint16)0, \
(Uint16)0, \
(CSL_McbspPABlk)CSL_MCBSP_PABLK_0, \
(CSL_McbspPBBlk)CSL_MCBSP_PBBLK_1, \
(CSL_McbspPABlk)CSL_MCBSP_PABLK_0, \
(CSL_McbspPBBlk)CSL_MCBSP_PBBLK_1, \
}\

```

Multichannel Setup defaults

#define CSL_MCBSP_RFULL 0x0004
RCV full status

#define CSL_MCBSP_RRDY 0x0001
RCV ready status

#define CSL_MCBSP_RSYNCERR 0x0010
RCV frame sync error status

#define CSL_MCBSP_XEMPTY 0x0008
XMT empty status

#define CSL_MCBSP_XRDY 0x0002
XMT ready status

```
#define CSL_MCBSP_XSYNCERR 0x0020
XMT frame sync error status
```

Chapter 6

TCP2 MODULE

Topics

6.1 Overview
6.2 Functions
6.3 Data Structures
6.4 Enumerations
6.5 Macros

6.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within TCP2 module.

The Turbo Decoder Coprocessor (TCP) is a programmable peripheral for decoding of IS2000/3GPP turbo codes. The TCP is controlled via memory mapped control registers and data buffers. The coprocessor operates either as a complete turbo decoder including the iterative structure (standalone processing mode), or it can operate as a single MAP (Maximum A Posterior) decoder (shared processing mode). In the standalone processing mode, the inputs into the TCP are channel soft decisions for systematic and parity bits, and the outputs are hard decisions. In the shared processing mode, the inputs are channel soft decisions for systematic and parity bits and apriori information for systematic bits, and the outputs are extrinsic information for systematic bits.

TCP is enhanced Turbo code system and will be able to support 44 384 Kbps data channels running at 333 MHz.

The TCP2 supports:

- Parallel concatenated convolutional turbo decoding using the MAP algorithm
- All turbo code rates greater than or equal to 1/5
- 3GPP and CDMA2000 turbo encoder trellis
- 3GPP and CDMA2000 block sizes in standalone mode
- Larger block sizes in shared processing mode
- Both max log MAP and log MAP decoding
- Sliding windows algorithm with variable reliability and prolog lengths
- The prolog reduction algorithm
- Execution of a minimum and maximum number of iterations
- The SNR stopping criteria algorithm
- The CRC stopping criteria algorithm

6.2 Functions

This section lists the functions available in the TCP2 module CSL.

6.2.1 TCP2_setParams

```
void TCP2_setParams      ( TCP2\_Params *restrict      configParams,
                          TCP2\_ConfigIc *restrict    configIc
                          )
```

Description

This function sets up the TCP input configuration parameters in the TCP2_ConfigIc structure. The configuration values are passed in the configParams input argument.

Arguments

configParams	Pointer to the structure holding the TCP configuration parameters.
configIc	Pointer to the TCP2_ConfigIc structure to be filled.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configIc argument passed.

Example

```
TCP2_ConfigIc    configIc;
TCP2_BaseParams  configBase;
TCP2_Params      configParams;
Uint32           frameLen = 40;
Uint32           cnt;

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen    = 0;
configBase.crcPoly   = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
```

```

configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate = TCP2_RATE_1_3;
configBase.snr = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

// Generate the configuration register values
TCP2_setParams (&configParams, &configIc);

```

6.2.2 TCP2_tailConfig

```

void TCP2_tailConfig      ( TCP2_Standard      standard,
                           TCP2_Mode          mode,
                           TCP2_Map           map,
                           TCP2_Rate          rate,
                           TCP2_TailData *restrict tailData,
                           TCP2\_ConfigIc *restrict configIc
                           )

```

Description

This function generates the input control values IC6-IC11 based on the processing to be performed by the TCP. These values consist of the tail data following the systematics and parities data. This function calls specific tail generation functions depending on the standard followed.

Arguments

<code>standard</code>	3G standard to be decoded.
<code>mode</code>	TCP processing mode (SA or SP)
<code>map</code>	TCP shared processing MAP
<code>rate</code>	Code rate of the TCP
<code>tailData</code>	Pointer to the tail data
<code>configIc</code>	Pointer to the TCP2_ConfigIc structure to be filled.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configlc argument passed.

Example

```
TCP2_ConfigIc      configIc;
TCP2_BaseParams    configBase;
TCP2_Params        configParams;
Uint32             frameLen = 40;
Uint32             cnt;
TCP2_TailData tailData []= { 0x2f,
                              0x31,
                              0x30,
                              0x20,
                              0x32,
                              0x27,
                              0x30,
                              0x0d,
                              0x10,
                              0x3f,
                              0x18,
                              0x3b };

TCP2_TailData *xabData = &tailData[frameLen];

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen    = 0;
configBase.crcPoly    = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate      = TCP2_RATE_1_3;
configBase.snr        = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

// Generate the configuration register values
```

```
TCP2_setParams (&configParams, &configIc);
```

```
TCP2_tailConfig ( configParams.standard,
                  configParams.mode,
                  configParams.map,
                  configParams.rate,
                  xabData, &configIc);
```

6.2.3 TCP2_genIc

```
void TCP2_genIc      ( TCP2\_Params *restrict      configParams,
                      TCP2_TailData *restrict      tailData,
                      TCP2\_ConfigIc *restrict      configIc
                      )
```

Description

This function sets up the TCP input configuration parameters in the TCP2_ConfigIc structure. The configuration values are passed in the configParams input argument.

Arguments

configParams	Pointer to the structure holding the TCP configuration parameters.
tailData	Tail data
configIc	Pointer to the TCP2_ConfigIc structure to be filled.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configIc argument passed.

Example

```
TCP2_TailData tailData []= { 0x2f,
                              0x31,
                              0x30,
                              0x20,
                              0x32,
                              0x27,
                              0x30,
                              0x0d,
                              0x10,
                              0x3f,
                              0x18,
```

```

                                0x3b };
```

```

TCP2_ConfigIc      configIc;
TCP2_BaseParams    configBase;
TCP2_Params        configParams;
TCP2_TailData      *xabData;
UInt32             frameLen = 40;
UInt32             cnt;

xabData = &tailData [frameLen];

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen     = 0;
configBase.crcPoly   = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate      = TCP2_RATE_1_3;
configBase.snr       = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

// Generate the configuration register values
TCP2_genIc (&configParams, xabData, &configIc);
```

6.2.4 TCP2_genParams

```

UInt32 TCP2_genParams      ( TCP2\_BaseParams *restrict      configBase,
                               TCP2\_Params *restrict      configParams
                               )
```

Description

This function copies the basic parameters, to the configParams parameters structure. For shared processing mode this function copies the configuration parameters for the first/middle sub-frame and the last sub frame. Hence, for this mode the function expects the configParams to be a pointer to an array of two TCP2_Params structure.

Arguments

<code>configBase</code>	Pointer to the <code>TCP2_BaseParams</code> structure
<code>configParams</code>	Pointer to the TCP configuration parameters structure.

Return Value `UInt32`

The number of sub frames for shared processing mode.

Pre Condition

`configBase` is populated with all the configuration parameters

Post Condition

None

Modifies

The `configParams` argument passed.

Example

```

TCP2_BaseParams    configBase;
TCP2_Params        configParams;
UInt32             frameLen = 40;
UInt32             cnt;

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen    = 0;
configBase.crcPoly   = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate      = TCP2_RATE_1_3;
configBase.snr       = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

```

6.2.5 TCP2_calcSubBlocksSA

```
void TCP2_calcSubBlocksSA ( TCP2\_Params * configParams )
```

Description

This function calculates the number of sub blocks for the TCP standalone processing. The reliability length is also calculated. The configParams structure is populated with the calculated parameters. User is not expected to call TCP2_calcSubBlocksSP() to configure the stand alone mode. Since TCP2_calcSubBlocksSP() does not check for error condition it configures stand alone mode also. As CSL does limited error checking user has to take care of it.

Arguments

configParams	Pointer to the structure holding the TCP configuration parameters.
--------------	--

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configParams argument passed.

Example

```

TCP2_BaseParams    configBase;
TCP2_Params        configParams;
Uint32             frameLen = 40;
Uint8              cnt;

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen    = 0;
configBase.crcPoly   = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate      = TCP2_RATE_1_3;
configBase.snr        = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

```

```
TCP2_calcSubBlocksSA (&configParams);
```

6.2.6 TCP2_calcSubBlocksSP

```
Uint32 TCP2_calcSubBlocksSP ( TCP2\_Params * configParams )
```

Description

This function calculates the number of sub blocks for the TCP shared processing. The reliability length is also calculated and the configParams structure is populated. These parameters are calculated for the first/middle sub-frame and the last sub frame. The function expects the configParams to be a pointer to an array of two TCP2_Params structure. User is not expected to call TCP2_calcSubBlocksSA() to configure the stand alone mode. Since CP2_calcSubBlocksSA() does not check for error condition it configures shared process mode also. As CSL does limited error checking user has to take care of it.

Arguments

configParams	Pointer to the structure holding the TCP configuration parameters.
--------------	--

Return Value Uint32

Number of sub frames the frame is divided into

Pre Condition

None

Post Condition

None

Modifies

The configParams argument passed.

Example

```
TCP2_BaseParams    configBase;
TCP2_Params        configParams;
Uint32             frameLen = 40;
Uint8              cnt;
Uint32             numSubFrames;

// Assign the configuration parameters
configBase.mode      = TCP2_MODE_SA;
configBase.frameLen  = frameLen;
configBase.inputSign = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag   = 1;
configBase.maxIter   = 8;
configBase.maxStarEn = TRUE;
configBase.standard  = TCP2_STANDARD_3GPP;
configBase.crcLen    = 0;
configBase.crcPoly   = 0;
configBase.minIter   = 1;
configBase.numCrcPass = 1;
configBase.outParmFlag = 0;
configBase.outputOrder = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn = FALSE;
configBase.prologSize = 24;
configBase.rate      = TCP2_RATE_1_3;
```

```

configBase.snr                = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);
numSubFrames = TCP2_calcSubBlocksSP (&configParams);
...

```

6.2.7 TCP2_tailConfig3GPP

```

void TCP2_tailConfig3GPP      ( TCP2_Mode           mode,
                                TCP2_Map            map,
                                TCP2_Rate           rate,
                                TCP2_TailData *restrict tailData,
                                TCP2\_ConfigIc *restrict configIc
                                )

```

Description

This function generates the input control values IC6-IC11 for 3GPP channels. These values consist of the tail data following the systematics and parities data. This function is called from the generic TCP2_tailConfig function. User is not expected to call TCP2_tailConfigIs2000() to configure the 3GPP channels. Since TCP2_tailConfigIs2000() does not check for error condition it configures IS2000 channels also. As CSL does limited error checking user has to take care of it.

Arguments

mode	TCP processing mode (SA or SP)
map	TCP shared processing MAP
rate	TCP data code rate
tailData	Pointer to the tail data
configIc	Pointer to the IC values structure

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configIc argument passed.

Example

```

TCP2_ConfigIc          configIc;
TCP2_BaseParams        configBase;
TCP2_Params            configParams;
UInt32                 frameLen = 40;
UInt16                 cnt;
TCP2_TailData tailData []= { 0x2f,
                              0x31,
                              0x30,
                              0x20,
                              0x32,
                              0x27,
                              0x30,
                              0x0d,
                              0x10,
                              0x3f,
                              0x18,
                              0x3b };

TCP2_TailData *xabData = &tailData[frameLen];

// Assign the configuration parameters
configBase.mode         = TCP2_MODE_SA;
configBase.frameLen     = frameLen;
configBase.inputSign    = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag      = 1;
configBase.maxIter      = 8;
configBase.maxStarEn    = TRUE;
configBase.standard     = TCP2_STANDARD_3GPP;
configBase.crcLen       = 0;
configBase.crcPoly      = 0;
configBase.minIter      = 1;
configBase.numCrcPass   = 1;
configBase.outParmFlag  = 0;
configBase.outputOrder  = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn  = FALSE;
configBase.prologSize   = 24;
configBase.rate         = TCP2_RATE_1_3;
configBase.snr          = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

// Generate the configuration register values
TCP2_setParams (&configParams, &configIc);

TCP2_tailConfig3GPP(configParams.mode,
                    configParams.map,
                    configParams.rate,
                    xabData, &configIc);

```

6.2.8 TCP2_tailConfigs2000

```
void TCP2_tailConfigs2000      ( TCP2_Mode           mode,
                                TCP2_Map            map,
                                TCP2_Rate           rate,
                                TCP2_TailData *restrict tailData,
                                TCP2\_ConfigIc *restrict configIc
                                )
```

Description

This function generates the input control values IC6-IC11 for IS2000 channels. These values consist of the tail data following the systematics and parities data. This function is called from the generic TCP2_tailConfig function. User is not expected to call TCP2_tailConfig3GPP() to configure the IS2000 channels. Since TCP2_tailConfig3GPP() does not check for error condition it configures 3GPP channels also. As CSL does limited error checking user has to take care of it.

Arguments

mode	TCP processing mode (SA or SP)
map	TCP shared processing MAP
rate	TCP data code rate
tailData	Pointer to the tail data
configIc	Pointer to the IC values structure

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The configIc argument passed.

Example

```
TCP2_ConfigIc      configIc;
TCP2_BaseParams    configBase;
TCP2_Params        configParams;
UInt32             frameLen = 40;
UInt16             cnt;
TCP2_TailData tailData []= { 0x2f,
                              0x31,
                              0x30,
                              0x20,
                              0x32,
                              0x27,
```

```

                                0x30,
                                0x0d,
                                0x10,
                                0x3f,
                                0x18,
                                0x3b };
TCP2_TailData *xabData = &tailData[frameLen];

// Assign the configuration parameters
configBase.mode                = TCP2_MODE_SA;
configBase.frameLen            = frameLen;
configBase.inputSign           = TCP2_INPUT_SIGN_POSITIVE;
configBase.intFlag             = 1;
configBase.maxIter             = 8;
configBase.maxStarEn           = TRUE;
configBase.standard            = TCP2_STANDARD_3GPP;
configBase.crcLen              = 0;
configBase.crcPoly             = 0;
configBase.minIter             = 1;
configBase.numCrcPass          = 1;
configBase.outParmFlag         = 0;
configBase.outputOrder         = TCP2_OUT_ORDER_0_31;
configBase.prologRedEn         = FALSE;
configBase.prologSize          = 24;
configBase.rate                = TCP2_RATE_1_3;
configBase.snr                 = 0;

for (cnt = 0; cnt < 16; cnt++)
    configBase.extrScaling [cnt] = 32;

// Setup the TCP configuration registers parameters
TCP2_genParams (&configBase, &configParams);

// Generate the configuration register values
TCP2_setParams (&configParams, &configIc);

TCP2_tailConfigIs2000 ( configParams.mode,
                        configParams.map,
                        configParams.rate,
                        xabData, &configIc);

```

6.2.9 TCP2_deinterleaveExt

```

void TCP2_deinterleaveExt    ( TCP2_ExtrinsicData *      aprioriMap1,
                              const TCP2_ExtrinsicData * extrinsicsMap2,
                              const Uint16 *             interleaverTable,
                              Uint32                     numExt
                              )

```

Description

This function de-interleaves the MAP2 extrinsics data to generate apriori data for the MAP1 decode. This function is for use in performing shared processing.

Arguments

<code>aprioriMap1</code>	Apriori data for MAP1 decode
<code>extrinsicsMap2</code>	Extrinsics data
<code>interleaverTable</code>	Interleaver data table
<code>numExt</code>	Number of Extrinsics

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The `aprioriMap1` argument passed is modified to contain the deinterleaved data.

Example

```
<...MAP 2 decode...>

TCP2_deinterleaveExt(aprioriMap1,
                    extrinsicsMap2,
                    interleaverTable,
                    numExt);

<...MAP 1 decode...>
```

6.2.10 TCP2_interleaveExt

```
void TCP2_interleaveExt    ( TCP2_ExtrinsicData *      aprioriMap2,
                           const TCP2_ExtrinsicData * extrinsicsMap1,
                           const Uint16 *             interleaverTable,
                           Uint32                     numExt
                           )
```

Description

This function interleaves the MAP1 extrinsics data to generate apriori data for the MAP2 decode. This function is for used in performing shared processing.

Arguments

<code>aprioriMap2</code>	Apriori data for MAP2 decode
<code>extrinsicsMap1</code>	Extrinsics data
<code>interleaverTable</code>	Interleaver data table
<code>numExt</code>	Number of Extrinsics

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The aprioriMap2 argument passed is modified to contain the interleaved data.

Example

```
<...MAP 1 decode...>

TCP2_interleaveExt(aprioriMap2,
                   extrinsicsMap1,
                   interleaverTable,
                   numExt);

<...MAP 2 decode...>
```

6.2.11 TCP2_depunctInputs

```
void TCP2_depunctInputs      (  Uint32          frameLen,
                               TCP2_UserData *  inputData,
                               TCP2_Rate        rate,
                               Uint32          scalingFact,
                               TCP2_InputData *  depunctData
                               )
```

Description

This function scales and sorts input data of any code rate into a code rate 1/5 format.

Arguments

<code>frameLen</code>	Input data length in bytes
<code>inputData</code>	Input data
<code>rate</code>	Input data code rate
<code>scalingFact</code>	Scaling factor
<code>depunctData</code>	Depunctured data

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The depunctData argument passed, to contain the data depunctured to rate 1/5

Example

```
TCP2_UserData inputData []= { 0x2f,
                                0x31,
                                0x30,
                                0x20,
                                0x32,
                                0x27,
                                0x30,
                                0x0d,
                                0x10,
                                0x3f,
                                0x18,
                                0x3b };

Uint32      rate = TCP2_RATE_1_4;
Uint32      frameLength = 40;
TCP2_InputData * depunctData;
Uint32      scalingFact;

TCP2_depunctInputs (frameLength,
                    inputData,
                    rate
                    scalingFact,
                    depunctData);
```

6.2.12 TCP2_calculateHd

```
void TCP2_calculateHd      ( Const TCP2_ExtrinsicData *      extrinsicsMap1,
                            Const TCP2_ExtrinsicData *      apriori,
                            Const TCP2_UserData *           channelData,
                            Uint32 *                         hardDecisions,
                            Uint16                           numExt
                            )
```

Description

This function calculates the hard decisions following multiple MAP decodings in shared processing mode.

Arguments

extrinsicsMap1	Extrinsics data following MAP1 decode
apriori	Apriori data following MAP2 decode
channelData	Input channel data
hardDecisions	Hard decisions

numExt Number of extrinsics

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The hardDecisions argument passed, to contain the calculated hard decisions

Example

```
<...Iterate through MAP1 and MAP2 decodes...>
void TCP2_calculateHd(extrinsicsMap1,
                      apriori,
                      channelData,
                      hardDecisions,
                      numExt);
```

6.2.13 TCP2_demuxInput

```
void TCP2_demuxInput      (  Uint32                rate,
                           Uint32                frameLen,
                           const TCP2_UserData *  input,
                           const Uint16 *         interleaver,
                           TCP2_ExtrinsicData *   nonInterleaved,
                           TCP2_ExtrinsicData *   interleaved
                           )
```

Description

This function splits the input data into two working sets. One set contains the non-interleaved input data and is used with the MAP 1 decoding. The other contains the interleaved input data and is used with the MAP2 decoding. This function is used in shared processing mode.

Arguments

rate	TCP data code rate
frameLen	Frame length
input	Input channel data
interleaver	Interleaver data table
nonInterleaved	Non Interleaved data for SP MAP0
interleaved	Interleaved data for SP MAP1

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

The nonInterleaved argument, to contain the non-interleaved data and the interleaved argument, to contain the interleaved data.

Example

```
TCP2_demuxInput ( TCP2_RATE_1_4,
                  frameLen,
                  input,
                  interleaver,
                  interleaved,
                  nonInterleaved);
```

INLINE FUNCTIONS
6.2.14 TCP2_normalCeil

```
CSL_IDEF_INLINE Uint32 TCP2_normalCeil          ( Uint32    val1,
                                                  Uint32    val2
                                                  )
```

Description

Returns the value rounded to the nearest integer, greater than or equal to (val1/val2).

Arguments

val1	Value to be augmented.
val2	Value by which val1 must be divisible.

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint32    frameLen = 51200;
Uint32    numSubFrame;
```

```
numSubFrame = TCP2_normalCeil (frameLen,
                                TCP2_SUB_FRAME_SIZE_MAX);
```

6.2.15 TCP2_ceil

```
CSL_IDEF_INLINE Uint32 TCP2_ceil          (  Uint32    val,
                                             Uint32    pwr2
                                             )
```

Description

Returns the value rounded to the nearest integer, greater than or equal to $(val/(2^{pwr2}))$.

Arguments

<code>val</code>	Value to be augmented.
<code>pwr2</code>	The power of two by which <code>val</code> must be divisible.

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint32    val1 = 512;
Uint32    val2 = 4;
Uint32    val3;

val3= TCP2_ceil((val1, val2);
```

6.2.16 TCP2_setExtScaling

```
CSL_IDEF_INLINE Uint32 TCP2_setExtScaling (  Uint8    extrVal1,
                                             Uint8    extrVal2,
                                             Uint8    extrVal3,
                                             Uint8    extrVal4
                                             )
```

Description

This function formats individual bytes into a 32-bit word, which is used to set the extrinsic configuration registers.

Arguments

<code>extrVal1</code>	Extrinsic scaling value 1
<code>extrVal2</code>	Extrinsic scaling value 2
<code>extrVal3</code>	Extrinsic scaling value 3
<code>extrVal4</code>	Extrinsic scaling value 4

Return Value

UInt32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
TCP2_Params      *configParams;
TCP2_ConfigIc    configIc;

configIc.ic12 = TCP2_setExtScaling(configParams->extrScaling [0],
                                   configParams->extrScaling [1],
                                   configParams->extrScaling [2],
                                   configParams->extrScaling [3]);
```

6.2.17 TCP2_makeTailArgs

```
CSL_IDEF_INLINE UInt32 TCP2_makeTailArgs          ( UInt8   byte17_12,
                                                    UInt8   byte11_6,
                                                    UInt8   byte5_0
                                                    )
```

Description

This function formats individual bytes into a 32-bit word, which is used to set the tail bits configuration registers.

Arguments

<code>byte17_12</code>	Byte to be placed in bits 17-12 of the 32-bit value
<code>byte11_6</code>	Byte to be placed in bits 11-6 of the 32-bit value
<code>byte5_0</code>	Byte to be placed in bits 5-0 of the 32-bit value

Return Value

UInt32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
extern TCP2_UserData    *userData;
TCP2_ConfigIc          configIc;
Uint8                  *xabData;

configIc.ic6 = TCP2_makeTailArgs( xabData[10],
                                   xabData[8],
                                   xabData[6]);
```

6.2.18 TCP2_getAccessErr

CSL_IDEF_INLINE Uint32 TCP2_getAccessErr (void)

Description

This function returns the ACC bit value of the TCPERR register indicating whether an invalid access has been made to the TCP during operation.

0 - No error

1 - TCP rams (syst, parities, hard decisions, extrinsics, aprioris) access is not allowed in state 1. This causes an error interrupt to occur.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getAccessErr ())
{
    ...
}
```

6.2.19 TCP2_getErr

CSL_IDEF_INLINE Uint32 TCP2_getErr (void)

Description

This function returns the ERR bit value of the TCPERR register indicating whether an error has occurred during TCP operation.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getErr ())
{
    ...
}
```

6.2.20 TCP2_getTcpErrors

CSL_IDEF_INLINE Uint32 TCP2_getTcpErrors

(void)

Description

This function returns the TCPERR register value.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getTcpErrors ())
{
    ...
}
```

6.2.21 TCP2_getFrameLenErr

CSL_IDEF_INLINE Uint32 TCP2_getFrameLenErr

(void)

Description

This function returns a Boolean value indicating whether an invalid frame length has been programmed in the TCP during operation.

0 - no error.

1 - (SA mode) frame length < 40 or frame length > 20730.

- (SP mode) frame length < 256 or frame length > 20480 and $f \% 256 \neq 0$ for the first or middle subframes.

- (SP mode) if $f < 128$ or $f > 20480$ for the last subframe.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getFrameLenErr ())
{
    ...
}
```

6.2.22 TCP2_getProlLenErr

CSL_IDEF_INLINE Uint32 TCP2_getProlLenErr

(void)

Description

This function returns the P bit value indicating whether an invalid prolog length has been programmed into the TCP.

0 - no error

1 - Prolog length < 4 or > 48

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getProlLenErr ())
{
    ...
}
```

6.2.23 TCP2_getSubFrameErr

CSL_IDEF_INLINE Uint32 TCP2_getSubFrameErr (void)

Description

This function returns a Boolean value indicating whether the sub-frame length programmed into the TCP is invalid.

0 - no error

1 - sub-frame length > 20480 (SP mode)

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getSubFrameErr ())
{
    ...
}
```

6.2.24 TCP2_getRelLenErr

CSL_IDEF_INLINE Uint32 TCP2_getRelLenErr (void)

Description

This function returns the R bit value indicating whether an invalid reliability length has been programmed into the TCP. The reliability length must be $40 < RL < 128$ for SA Mode, or must be 128 during first/middle subframes of SP mode, and must be > 64 in the last subframe.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getRelLenErr ())
{
    ...
}
```

6.2.25 TCP2_getSnrErr

CSL_IDEF_INLINE Uint32 TCP2_getSnrErr (void)

Description

This function returns the SNR bit value indicating whether the SNR threshold exceeded 100 (1) or not (0).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getSnrErr ())
{
    ...
}
```

6.2.26 TCP2_getInterleaveErr

CSL_IDEF_INLINE Uint32 TCP2_getInterleaveErr (void)

Description

This function returns the INTER value bit indicating whether the TCP was incorrectly programmed to receive an interleaver table. An interleaver table can only be sent when operating in standalone mode. This bit(1) indicates if an interleaver table was sent when in shared processing mode.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getInterleaveErr ())
{
    ...
}
```

6.2.27 TCP2_getOutParmErr

CSL_IDEF_INLINE Uint32 TCP2_getOutParmErr (void)

Description

This function returns the OP bit value (1) indicating whether the TCP was programmed to transfer output parameters in shared processing mode. The output parameters are only valid when operating in standalone mode.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getOutParmErr ())
{
    ...
}
```

6.2.28 TCP2_getMaxMinErr

CSL_IDEF_INLINE Uint32 TCP2_getMaxMinErr (void)

Description

This function returns the MAXMINITER bit value indicating whether the TCP was programmed with the minimum iterations value greater than the maximum iterations.

0 = no error

1 = min_iter > max_iter

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getMaxMinErr ())
{
    ...
}
```

6.2.29 TCP2_getNumIt

CSL_IDEF_INLINE Uint32 TCP2_getNumIt (void)

Description

This function returns the number of decoded iterations of the TCP in standalone processing mode. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      numIter;

    numIter = TCP2_getNumIt ();

```

6.2.30 TCP2_getSnrM1

CSL_IDEF_INLINE Uint32 TCP2_getSnrM1 (void)

Description

This function returns the 1st moment of SNR calculation. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32 snrM1;
    snrM1 = TCP2_getSnrM1 ();

```

6.2.31 TCP2_getSnrM2

CSL_IDEF_INLINE Uint32 TCP2_getSnrM2 (void)

Description

This function returns the second moment of SNR calculation. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      snrM2;
    snrM2 = TCP2_getSnrM2 ();

```

6.2.32 TCP2_getMap

CSL_IDEF_INLINE Uint32 TCP2_getMap (void)

Description

This function returns the active MAP of the TCP. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

0 - MAP 0 is active 1 - MAP 1 is active

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      activeMap;

    activeMap = _TCP2_getMap ();

```

6.2.33 TCP2_getMap0Err

CSL_IDEF_INLINE Uint32 TCP2_getMap0Err (void)

Description

This function returns the number of re-encode errors for MAP 0. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      map0Err;
    map0Err = TCP2_getMap0Err ();

```

6.2.34 TCP2_getMap1Err

CSL_IDEF_INLINE Uint32 TCP2_getMap1Err (void)

Description

This function returns the number of re-encode errors for MAP 1. This function reads the output parameters register. Alternatively, the EDMA can be used to transfer the output parameters following the hard decisions (recommended).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      map1Err;

    map1Err = TCP2_getMap1Err ();

```

6.2.35 TCP2_statRun

CSL_IDEF_INLINE Uint32 TCP2_statRun (void)

Description

This function returns a Boolean status indicating whether the TCP MAP decoder is in state 0 or state 1-8 (running or not).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
while (!TCP2_statRun());
```

6.2.36 TCP2_statError

CSL_IDEF_INLINE Uint32 TCP2_statError (void)

Description

This function returns the ERR bit value of the TCPSTAT register indicating whether TCP has encountered an error in the input register configuration..

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statError ())
{
    ...
}
```

6.2.37 TCP2_statWaitlc

CSL_IDEF_INLINE Uint32 TCP2_statWaitlc (void)

Description

This function returns the WIC bit status indicating whether the TCP is waiting to receive new IC values.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitIc ())
{
    ...
}
```

6.2.38 TCP2_statWaitInter

CSL_IDEF_INLINE Uint32 TCP2_statWaitInter (void)

Description

This function returns the WINT status indicating whether the TCP is waiting to receive interleaver table data.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitInter ())
{
    ...
}
```

6.2.39 TCP2_statWaitSysPar

CSL_IDEF_INLINE Uint32 TCP2_statWaitSysPar (void)

Description

This function returns the WSP bit status indicating whether the TCP is waiting to receive systematic and parity data.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitSysPar ())
{
    ...
}
```

6.2.40 TCP2_statWaitApriori

CSL_IDEF_INLINE Uint32 TCP2_statWaitApriori (void)

Description

This function returns the WAP bit status indicating whether the TCP is waiting to receive apriori data.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitApriori ())
{
    ...
}
```

6.2.41 TCP2_statWaitExt

CSL_IDEF_INLINE Uint32 TCP2_statWaitExt (void)

Description

This function returns the REXT bit status indicating whether the TCP is waiting for extrinsic data to be read.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitExt ())
{
    ...
}
```

6.2.42 TCP2_statWaitHardDec

CSL_IDEF_INLINE Uint32 TCP2_statWaitHardDec

(void)

Description

This function returns the RHD bit status indicating whether the TCP is waiting for the hard decisions data to be read.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitHardDec ())
{
    ...
}
```

6.2.43 TCP2_statWaitOutParm

CSL_IDEF_INLINE **Uint32** TCP2_statWaitOutParm (void)

Description

This function returns the ROP bit status indicating whether the TCP is waiting for the output parameters to be read.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statWaitOutParm ())
{
    ...
}
```

6.2.44 TCP2_statEmuHalt

CSL_IDEF_INLINE **Uint32** TCP2_statEmuHalt (void)

Description

This function returns the emuhalt bit status indicating whether the TCP is halted due to emulation.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_statEmuHalt ())
{
    ...
}
```

6.2.45 TCP2_statActMap

CSL_IDEF_INLINE Uint32 TCP2_statActMap (void)

Description

This function returns the active_map bit status of the TCPSTAT register indicating whether the TCP MAP 0 is active (0) or the TCP MAP 1 is active (1).

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint32    activeMap;
.....
activeMap = TCP2_statActMap ();
```

6.2.46 TCP2_statActState

CSL_IDEF_INLINE Uint32 TCP2_statActState (void)

Description

This function returns the active_state bit status indicating the active TCP MAP decoder state.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32    activeState;
    .....
    activeState = TCP2_statActState ();

```

6.2.47 TCP2_statActIter

CSL_IDEF_INLINE Uint32 TCP2_statActIter (void)

Description

This function returns the active_iter bit status indicating the active TCP iteration.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32    activeIter;
    .....
    activeIter = TCP2_statActIter ();

```

6.2.48 TCP2_statSnr

CSL_IDEF_INLINE Uint32 TCP2_statSnr (void)

Description

This function returns the snr_exceed bits, indicating whether the TCP MAP 0 or MAP 1 passed the SNR criteria in a particular iteration.

0 - All fail, 1 - MAP 1 failed, 2 - MAP 0 failed, 3 - All passed

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      snrExceed;
    .....
    snrExceed = TCP2_statSnr ();

```

6.2.49 TCP2_statCrc

CSL_IDEF_INLINE Uint32 TCP2_statCrc (void)

Description

This function returns the crc_pass bit boolean status indicating whether the TCP passed CRC check.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    if (TCP2_statCrc ())
    {
        ...
    }

```

6.2.50 TCP2_statTcpState

CSL_IDEF_INLINE Uint32 TCP2_statTcpState (void)

Description

This function returns the state of the TCP state machine for the standalone mode or shared processing mode.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      tcp2State;
    ....
    tcp2State = TCP2_statTcpState ();

```

6.2.51 TCP2_getExecStatus

CSL_IDEF_INLINE Uint32 TCP2_getExecStatus (void)

Description

This function returns the TCPSTAT register value.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      tcpStatus;

    tcpStatus = TCP2_getExecStatus ();

```

6.2.52 TCP2_getExtEndian

CSL_IDEF_INLINE Uint32 TCP2_getExtEndian (void)

Description

This function returns the value programmed into the TCP_END register for the extrinsic data indicating whether the data is in its native 8-bit format ('1') or consists of values packed in little endian format into 32-bit words ('0'). This should always be '0' for little endian operation.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getExtEndian ())
{
    ...
}
```

6.2.53 TCP2_getInterEndian

CSL_IDEF_INLINE Uint32 TCP2_getInterEndian (void)

Description

Returns the value programmed into the TCP_END register for the interleaver table data indicating whether the data is in its native 8-bit format ('1') or consists of values packed in little endian format into 32-bit words ('0'). This should always be '0' for little endian operation.

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getInterEndian ())
{
    ...
}
```

6.2.54 TCP2_getSlpZvss

CSL_IDEF_INLINE Uint32 TCP2_getSlpZvss (void)

Description

This function gets the configuration of the internal control of the slpZvss.

0 = sleep mode disabled
1 = internal control of slpzbss enabled

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getSlpzbss ())
{
    ...
}
```

6.2.55 TCP2_getSlpzbdd

CSL_IDEF_INLINE Uint32 TCP2_getSlpzbdd (void)

Description

This function gets the configuration of the internal control of the slpzbdd.

0 = sleep mode disabled

1 = internal control of slpzbdd enabled

Arguments

None

Return Value

Uint32

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (TCP2_getSlpzbdd ())
{
    ...
}
```

6.2.56 TCP2_setExtEndian

CSL_IDEF_INLINE void TCP2_setExtEndian (**Uint32** *endianMode*)

Description

This function programs TCP to view the format of the extrinsics data as either native 8-bit format ('1') or values packed into 32-bit words in little endian format ('0'). This should always be '0' for little endian operation.

Arguments

endianMode Endian setting for extrinsics data

Return Value

None

Pre Condition

None

Post Condition

TCPEND register bit for the extrinsic data is configured in the mode passed.

Modifies

TCPEND register

Example

```
TCP2_setExtEndian(1);
```

6.2.57 TCP2_setInterEndian

CSL_IDEF_INLINE void TCP2_setInterEndian (**Uint32** *endianMode*)

Description

This function programs TCP to view the format of the interleaver data as either native 8-bit format ('1') or values packed into 32-bit words in little endian format ('0'). This should always be '0' for little endian operation.

Arguments

endianMode Endian setting for interleaver data

Return Value

None

Pre Condition

None

Post Condition

TCPEND register bit for the interleaver data is configured in the mode passed.

Modifies

TCPEND register

Example

```
TCP2_setInterEndian (1);
```

6.2.58 TCP2_setNativeEndian

CSL_IDEF_INLINE void TCP2_setNativeEndian (void)

Description

This function programs the TCP to view the format of all data as native 8/16-bit format. This should only be used when running in big endian mode.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

TCPEND register configured to native mode for all data.

Modifies

TCPEND register

Example

```
TCP2_setNativeEndian ();
```

6.2.59 TCP2_setPacked32Endian

CSL_IDEF_INLINE void TCP2_setPacked32Endian (void)

Description

This function programs the TCP to view the format of all data as packed data in 32-bit words. This should always be used when running in little endian mode and should be used in big endian mode only if the CPU is formatting the data.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

TCPEND register configured to packed 32 mode for all data.

Modifies

TCPEND register

Example

```
TCP2_setPacked32Endian ();
```

6.2.60 TCP2_start

CSL_IDEF_INLINE void TCP2_start (void)

Description

This function starts the TCP by writing a '1h' to the EXEINST field of the TCPEXE register. See also TCP2_debug(), TCP2_debugStep() and TCP2_debugComplete().

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

TCP state machine starts executing.

Modifies

TCPEXE register

Example

```
TCP2_start ();
```

6.2.61 TCP2_debug

CSL_IDEF_INLINE void TCP2_debug (void)

Description

This function puts the TCP into debug mode by writing '4h' to the EXEINST field of the TCPEXE register. Normal initialization is performed and TCP waits in MAP state 0 for Debug Step or Debug Complete to be performed. See also TCP2_start(), TCP2_debugStep() and TCP2_debugComplete()

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

EXEINST field of the TCPEXE register is configured

Modifies

TCPEXE register

Example

```
TCP2_debug ();
```

6.2.62 TCP2_debugStep

CSL_IDEF_INLINE void TCP2_debugStep (void)

Description

This function executes one MAP decode and waits in state 6 when the TCP is in Debug mode, by writing '5h' to the EXEINST field of the TCPEXE register. See also TCP2_start(), TCP2_debug() and TCP2_debugComplete()

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

EXEINST field of the TCPEXE register is configured.

Modifies

TCPEXE register

Example

```
TCP2_debugStep ();
```

6.2.63 TCP2_debugComplete

CSL_IDEF_INLINE void TCP2_debugComplete (void)

Description

This function executes the remaining MAP decodes when the TCP is in Debug mode, by writing '6h' to the EXEINST field of the TCPEXE register. See also TCP2_start(), TCP2_debug() and TCP2_debugStep()

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

EXEINST field of the TCPEXE register is configured .

Modifies

TCPEXE register

Example

```
TCP2_debugComplete ();
```

6.2.64 TCP2_reset

CSL_IDEF_INLINE void TCP2_reset (void)

Description

This function performs a soft reset of all TCP registers except for TCPEXE and TCPEND registers.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

Performs soft reset of TCP2.

Modifies

TCPEXE register

Example

```
TCP2_reset ();
```

6.2.65 TCP2_setSlpzvdd

CSL_IDEF_INLINE void TCP2_setSlpzvdd (Uint32 *slpzvddCtrl*)

Description

This function enables/disables the internal control of the slpzvdd.

Arguments

slpzvddCtrl Enable/disable configuration of the slpzvdd

Return Value

None

Pre Condition

None

Post Condition

TCPEND register configured with the value passed.

Modifies

TCPEND register

Example

```
TCP2_setSlpzvdd (1);
```

6.2.66 TCP2_setSlpzvss

CSL_IDEF_INLINE void TCP2_setSlpzvss (**Uint32** *slpzvssCtrl*)

Description

This function enables/disables the internal control of the slpzvss.

Arguments

slpzvssCtrl Enable/disable configuration of the slpzvss

Return Value

None

Pre Condition

None

Post Condition

TCPEND register configured with the value passed.

Modifies

TCPEND register

Example

```
TCP2_setSlpzvss (1);
```

6.2.67 TCP2_getIcConfig

CSL_IDEF_INLINE void TCP2_getIcConfig ([TCP2_ConfigIc](#) * *config*)

Description

This function reads the input configuration values currently programmed into the TCP.

Arguments

config TCP configuration structure to hold the read values

Return Value

None

Pre Condition

None

Post Condition

config structure contains the TCP input configuration values.

Modifies

None

Example

```
TCP2_ConfigIc      config;

TCP2_getIcConfig (&config);
```

6.2.68 TCP2_icConfig

CSL_IDEF_INLINE void TCP2_icConfig ([TCP2_ConfigIc](#)* *config*)

Description

Programs the TCP with the input configuration values passed in the TCP2_ConfigIc structure. This is not the recommended means by which to program the TCP, as it is more efficient to transfer the IC values using the EDMA.

Arguments

<i>config</i>	TCP configuration structure containing the values to be programmed
---------------	--

Return Value

None

Pre Condition

None

Post Condition

TCP input configuration registers are programmed with the values passed.

Modifies

TCP input configuration registers.

Example

```
TCP2_ConfigIc    configIc;
configIc.ic0 = 0x00283300;
configIc.ic1 = 0x00270000;
configIc.ic2 = 0x00080118;
configIc.ic3 = 0x00000011;
configIc.ic4 = 0x00000100;
configIc.ic5 = 0x00000000;
configIc.ic6 = 0x00032c2f;
configIc.ic7 = 0x00027831;
configIc.ic8 = 0x00000000;
configIc.ic9 = 0x00018430;
configIc.ic10 = 0x0003bfcd;
configIc.ic11 = 0x00000000;
configIc.ic12 = 0x00820820;
configIc.ic13 = 0x00820820;
configIc.ic14 = 0x00820820;
configIc.ic15 = 0x00820820;

TCP2_icConfig (&configIc);
```


6.2.69 TCP2_icConfigArgs

```
CSL_IDEF_INLINE void TCP2_icConfigArgs          ( Uint32    ic0,
                                                  Uint32    ic1,
                                                  Uint32    ic2,
                                                  Uint32    ic3,
                                                  Uint32    ic4,
                                                  Uint32    ic5,
                                                  Uint32    ic6,
                                                  Uint32    ic7,
                                                  Uint32    ic8,
                                                  Uint32    ic9,
                                                  Uint32    ic10,
                                                  Uint32    ic11,
                                                  Uint32    ic12,
                                                  Uint32    ic13,
                                                  Uint32    ic14,
                                                  Uint32    ic15
                                                  )
```

Description

Programs the TCP with the input configuration values passed. This is not the recommended means by which to program the TCP, as it is more efficient to transfer the IC values using the EDMA.

Arguments

<i>ic0</i>	TCP input configuration register 0 value
<i>ic1</i>	TCP input configuration register 1 value
<i>ic2</i>	TCP input configuration register 2 value
<i>ic3</i>	TCP input configuration register 3 value
<i>ic4</i>	TCP input configuration register 4 value
<i>ic5</i>	TCP input configuration register 5 value
<i>ic6</i>	TCP input configuration register 6 value
<i>ic7</i>	TCP input configuration register 7 value

ic8	TCP input configuration register 8 value
ic9	TCP input configuration register 9 value
ic10	TCP input configuration register 10 value
ic11	TCP input configuration register 11 value
ic12	TCP input configuration register 12 value
ic13	TCP input configuration register 13 value
ic14	TCP input configuration register 14 value
ic15	TCP input configuration register 15 value

Return Value

None

Pre Condition

None

Post Condition

TCP input configuration registers are programmed with the values passed.

Modifies

TCP input configuration registers

Example

```

    Uint32 ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7, ic8, ic9, ic10,
        ic11, ic12, ic13, ic14, ic15;

    ic0 = 0x00283300;
    ic1 = 0x00270000;
    ic2 = 0x00080118;
    ic3 = 0x00000011;
    ic4 = 0x00000100;
    ic5 = 0x00000000;
    ic6 = 0x00032c2f;
    ic7 = 0x00027831;
    ic8 = 0x00000000;
    ic9 = 0x00018430;
    ic10 = 0x0003bfcd;
    ic11 = 0x00000000;
    ic12 = 0x00820820;
    ic13 = 0x00820820;
    ic14 = 0x00820820;
    ic15 = 0x00820820;

    TCP2_icConfigArgs (ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7,
        ic8, ic9, ic10, ic11, ic12, ic13, ic14, ic15);

```

6.3 Data Structures

This section lists the data structures available in the TCP2 module.

6.3.1 TCP2_Configlc

Detailed Description

The TCP input configuration structure holds all the configuration values that are to be transferred to the TCP via the EDMA

Field Documentation

Uint32 TCP2_Configlc::ic0

TCP input configuration word 0 value

Uint32 TCP2_Configlc::ic1

TCP input configuration word 1 value

Uint32 TCP2_Configlc::ic2

TCP input configuration word 2 value

Uint32 TCP2_Configlc::ic3

TCP input configuration word 3 value

Uint32 TCP2_Configlc::ic4

TCP input configuration word 4 value

Uint32 TCP2_Configlc::ic5

TCP input configuration word 5 value

Uint32 TCP2_Configlc::ic6

TCP input configuration word 6 value

Uint32 TCP2_Configlc::ic7

TCP input configuration word 7 value

Uint32 TCP2_Configlc::ic8

TCP input configuration word 8 value

Uint32 TCP2_Configlc::ic9

TCP input configuration word 9 value

Uint32 TCP2_Configlc::ic10

TCP2 input configuration word 10 value

Uint32 TCP2_Configlc::ic11

TCP input configuration word 11 value

Uint32 TCP2_Configlc::ic12

TCP input configuration word 12 value

Uint32 TCP2_Configlc::ic13

TCP input configuration word 13 value

Uint32 TCP2_Config::ic14

TCP input configuration word 14 value

Uint32 TCP2_Config::ic15

TCP input configuration word 15 value

6.3.2 TCP2_Params

Detailed Description

The TCP parameters structure holds all the information concerning the user channel. These values are used to generate the appropriate input configuration values for the TCP.

Field Documentation

Uint8 TCP2_Params::crcLen

CRC polynomial length

Uint32 TCP2_Params::crcPoly

CRC polynomial

Uint8 TCP2_Params::extrScaling[16]

Extrinsic scaling factors

Uint32 TCP2_Params::frameLen

Frame length

[TCP2_InputSign](#) TCP2_Params::inputSign

The sign of the input data (+/-)

Uint32 TCP2_Params::intFlag

Interleaver write flag

TCP2_Map TCP2_Params::map

TCP2 shared processing MAP

Uint32 TCP2_Params::maxIter

Maximum number of iterations

Bool TCP2_Params::maxStarEn

Enable/disable the max star

Uint8 TCP2_Params::minIter

Minimum number of iterations to be executed

TCP2_Mode TCP2_Params::mode

TCP mode

Uint8 TCP2_Params::numCrcPass

Number of passed CRC iterations required before decoder termination

TCP2_NumSW TCP2_Params::numSlideWin

Number of sliding window per sub block

UInt32 TCP2_Params::numSubBlock

Number of sub blocks

UInt32 TCP2_Params::outParmFlag

Output parameters read flag

[TCP2_OutputOrder](#) TCP2_Params::outputOrder

The bit ordering of the output data

Bool TCP2_Params::prologRedEn

Enable/disable the prolog reduction

UInt32 TCP2_Params::prologSize

Prolog length

TCP2_Rate TCP2_Params::rate

TCP code rate

UInt32 TCP2_Params::relLen

Reliability length

UInt32 TCP2_Params::snr

SNR threshold used for stopping test

TCP2_Standard TCP2_Params::standard

TCP standard

6.3.3 TCP2_BaseParams

Detailed Description

The TCP base parameters structure is used to set up the TCP programmable parameters. The user has to create the object and pass it to the TCP2_genParams() function which returns the TCP2_Params structure.

Field Documentation
UInt8 TCP2_BaseParams::crcLen

CRC polynomial length

UInt32 TCP2_BaseParams::crcPoly

CRC polynomial

UInt8 TCP2_BaseParams::extrScaling[16]

Extrinsic scaling factors

UInt32 TCP2_BaseParams::frameLen

Frame length

[TCP2_InputSign](#) TCP2_BaseParams::inputSign

The sign of the input data (+/-)

UInt32 TCP2_BaseParams::intFlag

Interleaver write flag

TCP2_Map TCP2_BaseParams::map

TCP shared processing MAP

UInt32 TCP2_BaseParams::maxIter

Maximum number of iterations

Bool TCP2_BaseParams::maxStarEn

Enable/disable the max star

UInt8 TCP2_BaseParams::minIter

Minimum number of iterations to be executed

TCP2_Mode TCP2_BaseParams::mode

TCP Mode

UInt8 TCP2_BaseParams::numCrcPass

Number of passed CRC iterations required before decoder termination

UInt32 TCP2_BaseParams::outParmFlag

Output parameters read flag

[TCP2_OutputOrder](#) TCP2_BaseParams::outputOrder

The bit ordering of the output data

Bool TCP2_BaseParams::prologRedEn

Enable/disable the prolog reduction

UInt32 TCP2_BaseParams::prologSize

Prolog length

TCP2_Rate TCP2_BaseParams::rate

TCP code rate

UInt32 TCP2_BaseParams::snr

SNR threshold used for stopping test

TCP2_Standard TCP2_BaseParams::standard

TCP decoder standards

6.4 Enumerations

This section lists the enumerations available in the TCP2 module.

6.4.1 TCP2_InputSign

enum TCP2_InputSign

Enum for the input sign values

Enumeration values:

TCP2_INPUT_SIGN_POSITIVE

Multiply the channel input by +1

TCP2_INPUT_SIGN_NEGATIVE

Multiply the channel input by -1

6.4.2 TCP2_OutputOrder

enum TCP2_OutputOrder

Enum for the output order values

Enumeration values:

TCP2_OUT_ORDER_0_31

Order of the bits in the output data is 0-31

TCP2_OUT_ORDER_31_0

Order of the bits in the output data is 31-0

6.5 Macros

#define tcp2CfgRegs ((CSL_Tcp2CfgRegs*)CSL_TCP2_CFG_REGS)

Address of the TCP2 configuration registers

#define tcp2Regs ((CSL_Tcp2Regs*)CSL_TCP2_0_REGS)

Address of the TCP2 registers

#define TCP2_FIRST_SF CSL_TCP2_TCP2IC0_OPMOD_SP_FF

TCP shared processing, first sub frame

#define TCP2_FLEN_MAX 20730

TCP maximum standalone mode frame size

#define TCP2_LAST_SF CSL_TCP2_TCP2IC0_OPMOD_SP_LF

TCP shared processing, last sub frame

#define TCP2_MAP_MAP1 0

TCP shared processing non interleaved MAP

#define TCP2_MAP_MAP2 1

TCP shared processing interleaved MAP

#define TCP2_MIDDLE_SF CSL_TCP2_TCP2IC0_OPMOD_SP_MF

TCP shared processing, middle sub frame

#define TCP2_MODE_SA CSL_TCP2_TCP2IC0_OPMOD_SA

TCP stand alone mode

#define TCP2_MODE_SP 1

TCP shared processing mode

#define TCP2_RATE_1_2 CSL_TCP2_TCP2IC0_RATE_1_2

define for TCP code rate 1/2

#define TCP2_RATE_1_3 CSL_TCP2_TCPIC0_RATE_1_3

define for TCP code rate 1/3

#define TCP2_RATE_1_4 CSL_TCP2_TCPIC0_RATE_1_4

define for TCP code rate 1/4

#define TCP2_RATE_1_5 CSL_TCP2_TCPIC0_RATE_1_5

define for TCP code rate 1/5

#define TCP2_RATE_3_4 CSL_TCP2_TCPIC0_RATE_3_4

define for TCP code rate 3/4

#define TCP2_RLEN_MAX 128

TCP maximum reliability length

#define TCP2_STANDARD_3GPP 0

Decoder standard 3GPP

#define TCP2_STANDARD_IS2000 1
Decoder standard IS2000

#define TCP2_SUB_FRAME_SIZE_MAX 20480
TCP maximum sub frame size

#define TCP2_SW_G128 CSL_TCP2_TCPIC0_NUMSW_G_128
Number of sliding windows per block is > 128

#define TCP2_SW_LEQ128 CSL_TCP2_TCPIC0_NUMSW_LEQ_128
Number of sliding windows per block is <= 128

Chapter 7

TIMER MODULE

Topics

<u>7.1 Overview</u>
<u>7.2 Functions</u>
<u>7.3 Data Structures</u>
<u>7.4 Enumerations</u>
<u>7.5 Macros</u>

7.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within TIMER module. This is having six 64-bit general-purpose timers. Each of the TIMER peripherals (TIMER0 to TIMER5) is configurable as either 64-bit general-purpose timer or two 32-bit general-purpose timers or a watchdog timer. Each timer is made up of two 32-bit counters: a high counter and a low counter.

The timers can be used to: time events, count events, generate pulses, interrupt the CPU, and send synchronization events to the EDMA.

7.2 Functions

This section lists the functions available in the TIMER module.

7.2.1 CSL_tmrInit

CSL_Status CSL_tmrInit ([CSL_TmrContext](#) * *pContext*)

Description

This is the initialization function for the TIMER CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext	Pointer to module-context. As General purpose timer doesn't have any context based information user is expected to pass NULL.
----------	---

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

The CSL for timer is initialized.

Modifies

None

Example

```
CSL_tmrInit(NULL);
```

7.2.2 CSL_tmrOpen

[CSL_TmrHandle](#) CSL_tmrOpen ([CSL_TmrObj](#) * *pTmrObj*,
 CSL_InstNum *tmrNum*,
[CSL_TmrParam](#) * *pTmrParam*,
 CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the TIMER instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of TIMER device. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input argument for rest of the TIMER CSL APIs.

Arguments

<code>pTmrObj</code>	Pointer to timer object.
<code>tmrNum</code>	Instance of timer CSL to be opened. There are six instances of the timers available. So, the value for this parameter will be based on the instance.
<code>pTmrParam</code>	Module specific parameters
<code>pStatus</code>	Status of the function call

Return Value

CSL_TmrHandle

- Valid gptimer handle will be returned if status value is equal to CSL_SOK.

Pre Condition

The TIMER must be successfully initialized via `CSL_tmrInit()` before calling this function.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid gptimer handle is returned
- CSL_ESYS_FAIL The gptimer instance is invalid

2. Timer object structure is populated.

Modifies

Timer object structure and pStatus variable

Example

```

CSL_Status          status;
CSL_TmrObj          tmrObj;
CSL_TmrHandle       hTmr;
...
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);
...

```

7.2.3 CSL_tmrClose

CSL_Status CSL_tmrClose ([CSL_TmrHandle](#) *hTmr*)

Description

This function closes the specified instance of TIMER. CSL for the timer instance need to be reopened before using any timer CSL API.

Arguments

hTmr	Pointer to the object that holds reference to the instance of TIMER requested after the call
------	--

Return Value CSL_Status

- CSL_SOK - Timer close successful
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling *CSL_tmrClose()*.

Post Condition

The timer CSL APIs can not be called until the timer CSL is reopened again using *CSL_tmrOpen()*

Modifies

Obj structure values for the instance

Example

```
CSL_TmrHandle      hTmr ;
...
CSL_tmrClose(hTmr);
```

7.2.4 CSL_tmrHwSetup

CSL_Status CSL_tmrHwSetup	(CSL_TmrHandle	<i>hTmr,</i>
		CSL_TmrHwSetup *	<i>hwSetup</i>
)		

Description

It configures the timer instance registers as per the values passed in the hardware setup structure.

Arguments

hTmr	Handle to the TIMER instance
hwSetup	Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling this function.

Post Condition

The specified instance will be setup according the hardware setup parameters.

Modifies

Timer registers for the specified instance

Example

```

CSL_Status          status;
CSL_TmrHwSetup      hwSetup;
CSL_TmrHandle       hTmr;
CSL_TmrObj          tmrObj;

hwSetup.tmrTimerPeriodLo = 0x100;
hwSetup.tmrTimerPeriodHi = 0x100;
...
CSL_tmrInit(NULL);
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);

...
status = CSL_tmrHwSetup(hTmr, &hwSetup);
...

```

7.2.5 CSL_tmrHwControl

```

CSL_Status CSL_tmrHwControl      ( CSL\_TmrHandle          hTmr,
                                   CSL\_TmrHwControlCmd       cmd,
                                   void *                       arg
                                   )

```

Description

This function performs various control operations on the timer instance, based on the command passed.

Arguments

<code>hTmr</code>	Handle to the timer instance
<code>cmd</code>	Operation to be performed on the timer
<code>arg</code>	Optional argument as per the control command

Return Value CSL_Status

- CSL_SOK - Command execution successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling this function.

Post Condition

Registers of the timer instance are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

TIMER Registers

Example

```
CSL_Status      status;
CSL_TmrHandle   hTmr;
CSL_TmrObj      tmrObj;

CSL_tmrInit(NULL);
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);

...
status = CSL_tmrHwControl(hTmr, CSL_TMR_CMD_START_TIMLO, NULL);
...
```

7.2.6 CSL_tmrGetHwStatus

```
CSL_Status CSL_tmrGetHwStatus ( CSL\_TmrHandle          hTmr,
                                CSL\_TmrHwStatusQuery       query,
                                void *                      response
                                )
```

Description

This function is used to get the value of various parameters of the timer instance. The value returned depends on the query passed.

Arguments

hTmr	Handle to the timer instance
query	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Query command not supported

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling this function.

Post Condition

None

Modifies

Third parameter, response value

Example

```

CSL_Status      status;
CSL_TmrHandle   hTmr;
CSL_TmrObj      tmrObj;
Uint32          count;

CSL_tmrInit(NULL);
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);

...

status = CSL_tmrGetHwStatus(hTmr, CSL_TMR_QUERY_COUNT_LO,
                           &count);

...
```

7.2.7 CSL_tmrHwSetupRaw

```

CSL_Status CSL_tmrHwSetupRaw ( CSL\_TmrHandle      hTmr,
                              CSL\_TmrConfig *    config
                              )
```

Description

This function initializes the device registers with the register-values provided through the config data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values.

Arguments

hTmr	Handle to the timer instance
config	Pointer to the config structure containing the device register values

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration structure pointer is not properly initialized

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling this function.

Post Condition

The registers of the specified timer instance will be setup according to the values passed through the Config structure.

Modifies

Hardware registers of the specified General purpose timer instance

Example

```

CSL_TmrHandle      hTmr;
```

```

CSL_TmrConfig      config = CSL_TMR_CONFIG_DEFAULTS;
CSL_Status         status;
CSL_TmrObj         tmrObj;

CSL_tmrInit(NULL);
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);
...
status = CSL_tmrHwSetupRaw(hTmr, &config);
...

```

7.2.8 CSL_tmrGetHwSetup

```

CSL_Status CSL_tmrGetHwSetup      ( CSL\_TmrHandle          hTmr,
                                   CSL\_TmrHwSetup *      hwSetup
                                   )

```

Description

This function gets the current setup of the TIMER. The status is returned through *CSL_tmrHwSetup*. The obtaining of status is the reverse operation of *CSL_tmrHwSetup()* function.

Arguments

<i>hTmr</i>	Handle to the timer instance
<i>hwSetup</i>	Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup retrieved
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

Both *CSL_tmrInit()* and *CSL_tmrOpen()* must be called successfully in order before calling this function.

Post Condition

None

Modifies

Second parameter *hwSetup* value

Example

```

CSL_Status      status;
CSL_TmrHandle   hTmr;
CSL_TmrHwSetup  hwSetup;
CSL_TmrObj      tmrObj;

CSL_tmrInit(NULL);
hTmr = CSL_tmrOpen(&tmrObj, CSL_TMR_1, NULL, &status);

...
status = CSL_tmrGetHwSetup(hTmr, &hwSetup);

```

...

7.2.9 CSL_tmrGetBaseAddress

```
CSL_Status CSL_tmrGetBaseAddress ( CSL_InstNum      tmrNum,
                                   CSL\_TmrParam * pTmrParam,
                                   CSL\_TmrBaseAddress * pBaseAddress
                                   )
```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_tmrOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

<code>tmrNum</code>	Specifies the instance of the timer to be opened
<code>pTmrParam</code>	Timer module specific parameters
<code>pBaseAddress</code>	Pointer to base address structure containing base address details

Return Value CSL_Status

- CSL_SOK - Successful on getting the base address of TIMER
- CSL_ESYS_FAIL - Timer instance is not available.
- CSL_ESYS_INVPARAMS - Invalid Parameters

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status      status;
CSL_TmrBaseAddress  baseAddress;

...
status = CSL_tmrGetBaseAddress(CSL_TMR_1, NULL, &baseAddress);
...
```

7.3 Data Structures

This section lists the data structures available in the TIMER module.

7.3.1 CSL_TmrObj

Detailed Description

Watchdog timer object structure.

Field Documentation

CSL_InstNum CSL_TmrObj::perNum

Instance of timer being referred by this object

CSL_TmrRegsOvly CSL_TmrObj::regs

Pointer to the register overlay structure of the timer

7.3.2 CSL_TmrConfig

Detailed Description

Config-structure Used to configure the timer using CSL_tmrHwSetupRaw(). This is a structure of register values, rather than a structure of register field values like CSL_TmrHwSetup.

Field Documentation

UInt32 CSL_TmrConfig::EMUMGT_CLKSPD

Emulation Management/Clock Speed Registers

UInt32 CSL_TmrConfig::PRDHI

Timer Period Register High

UInt32 CSL_TmrConfig::PRDLO

Timer Period Register Low

UInt32 CSL_TmrConfig::TCR

Timer Control Register

UInt32 CSL_TmrConfig::TGCR

Timer Global Control Register

UInt32 CSL_TmrConfig::TIMHI

Timer Counter Register High

UInt32 CSL_TmrConfig::TIMLO

Timer Counter Register Low

UInt32 CSL_TmrConfig::WDTCTCR

Watchdog Timer Control Register

7.3.3 CSL_TmrContext

Detailed Description

Module specific context information. Present implementation of timer CSL doesn't have any context information.

Field Documentation

UInt16 CSL_TmrContext::contextInfo

Context information of timer CSL. The declaration is just a placeholder for future implementation.

7.3.4 CSL_TmrParam

Detailed Description

Module specific parameters. Present implementation of timer CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_TmrParam::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

7.3.5 CSL_TmrHwSetup

Detailed Description

Hardware setup structure.

Field Documentation

[CSL_TmrClksrc](#) CSL_TmrHwSetup::tmrClksrcHi

CLKSRC determines the selected clock source for the timer

[CSL_TmrClksrc](#) CSL_TmrHwSetup::tmrClksrcLo

CLKSRC determines the selected clock source for the timer

[CSL_TmrClockPulse](#) CSL_TmrHwSetup::tmrClockPulseHi

Clock/Pulse mode for timerHigh output

[CSL_TmrClockPulse](#) CSL_TmrHwSetup::tmrClockPulseLo

Clock/Pulse mode for timerLow output

[CSL_TmrInvInp](#) CSL_TmrHwSetup::tmrInvInpHi

Timer input inverter control. Only affects operation if CLKSRC=1, Timer Input pin

[CSL_TmrInvInp](#) CSL_TmrHwSetup::tmrInvInpLo

Timer input inverter control. Only affects operation if CLKSRC=1, Timer Input pin

[CSL_TmrInvOutp](#) CSL_TmrHwSetup::tmrInvOutpHi

Timer output inverter control

[CSL_TmrInvOutp](#) CSL_TmrHwSetup::tmrInvOutpLo

Timer output inverter control

[CSL_TmrIpGate](#) CSL_TmrHwSetup::tmrIpGateHi

TIEN determines if the timer clock is gated by the timer input. Applicable only when CLKSRC=0

[CSL_TmrIpGate](#) CSL_TmrHwSetup::tmrIpGateLo

TIEN determines if the timer clock is gated by the timer input. Applicable only when CLKSRC=0

UInt8 CSL_TmrHwSetup::tmrPreScalarCounterHi

TIMHI pre-scalar counter specifies the count for TIMHI

[CSL_TmrPulseWidth](#) CSL_TmrHwSetup::tmrPulseWidthHi

Pulse width. Used in pulse mode (C/P_=0) by the timer

[CSL_TmrPulseWidth](#) CSL_TmrHwSetup::tmrPulseWidthLo

Pulse width. Used in pulse mode (C/P_=0) by the timer

UInt32 CSL_TmrHwSetup::tmrTimerCounterHi

32-bit load value to be loaded to Timer Counter Register High

UInt32 CSL_TmrHwSetup::tmrTimerCounterLo

32-bit load value to be loaded to Timer Counter Register Low

[CSL_TmrMode](#) CSL_TmrHwSetup::tmrTimerMode

Configures the GP timer in GP mode or in general purpose timer mode or Dual 32 bit timer mode

UInt32 CSL_TmrHwSetup::tmrTimerPeriodHi

32-bit load value to be loaded to Timer Period Register High

UInt32 CSL_TmrHwSetup::tmrTimerPeriodLo

32-bit load value to be loaded to Timer Period Register low

7.3.6 CSL_TmrBaseAddress

Detailed Description

This structure contains the base-address information for the peripheral instance.

Field Documentation

CSL_TmrRegsOvly CSL_TmrBaseAddress::regs

Base-address of the configuration registers of the peripheral

7.4 Enumerations

This section lists the enumerations available in the TIMER module.

7.4.1 CSL_TmrHwControlCmd

enum CSL_TmrHwControlCmd

This enum describes the commands used to control the timer through CSL_tmrHwControl().

Enumeration values:

<i>CSL_TMR_CMD_LOAD_PRDLO</i>	Loads the Timer Period Register Low. Parameters: <i>Uint32 *</i>
<i>CSL_TMR_CMD_LOAD_PRDHI</i>	Loads the Timer Period Register High. Parameters: <i>Uint32 *</i>
<i>CSL_TMR_CMD_LOAD_PSCHI</i>	Loads the Timer Pre-scalar value for TIMHI. Parameters: <i>Uint8 *</i>
<i>CSL_TMR_CMD_START_TIMLO</i>	Enable the timer Low. Parameters: <i>CSL_TmrEnamode</i>
<i>CSL_TMR_CMD_START_TIMHI</i>	Enable the timer High. Parameters: <i>CSL_TmrEnamode</i>
<i>CSL_TMR_CMD_STOP_TIMLO</i>	Stop the timer Low. Parameters: <i>None</i>
<i>CSL_TMR_CMD_STOP_TIMHI</i>	Stop the timer High. Parameters: <i>None</i>
<i>CSL_TMR_CMD_RESET_TIMLO</i>	Reset the timer Low. Parameters: <i>None</i>
<i>CSL_TMR_CMD_RESET_TIMHI</i>	Reset the timer High. Parameters: <i>None</i>
<i>CSL_TMR_CMD_START64</i>	Start the timer in GPtimer64 OR Chained mode. Parameters: <i>None</i>
<i>CSL_TMR_CMD_STOP64</i>	Stop the timer of GPtimer64 OR Chained. Parameters: <i>CSL_TmrEnamode</i>
<i>CSL_TMR_CMD_RESET64</i>	Reset the timer of GPtimer64 OR Chained. Parameters: <i>None</i>
<i>CSL_TMR_CMD_START_WDT</i>	Enable the timer in watchdog mode. Parameters:

CSL_TMR_CMD_LOAD_WDKEY	<p><i>CSL_TmrEnamode</i></p> <p>Loads the watchdog key.</p> <p>Parameters:</p> <p><i>Uint16</i></p>
-------------------------------	--

7.4.2 CSL_TmrHwStatusQuery

enum CSL_TmrHwStatusQuery

This enum describes the commands used to get status of various parameters of the timer. These values are used in CSL_tmrGetHwStatus().

Enumeration values:

CSL_TMR_QUERY_COUNT_LO	<p>Gets the current value of the timer TIMLO register.</p> <p>Parameters:</p> <p><i>Uint32 *</i></p>
CSL_TMR_QUERY_COUNT_HI	<p>Gets the current value of the timer TIMHI register.</p> <p>Parameters:</p> <p><i>Uint32 *</i></p>
CSL_TMR_QUERY_TSTAT_LO	<p>This query command returns the status about whether the TIMLO is running or stopped.</p> <p>Parameters:</p> <p><i>CSL_TmrTstat</i></p>
CSL_TMR_QUERY_TSTAT_HI	<p>This query command returns the status about whether the TIMHI is running or stopped.</p> <p>Parameters:</p> <p><i>CSL_TmrTstat</i></p>
CSL_TMR_QUERY_WDFLAG_STATUS	<p>This query command returns the status about whether the timer is in watchdog mode or not.</p> <p>Parameters:</p> <p><i>CSL_WdflagBitStatus</i></p>

7.4.3 CSL_TmrIrpGate

enum CSL_TmrIrpGate

This enum describes whether the Timer Clock input is gated or not gated.

Enumeration values:

CSL_TMR_CLOCK_INP_NOGATE	Timer input not gated
CSL_TMR_CLOCK_INP_GATE	Timer input gated

7.4.4 CSL_TmrClksrc

enum CSL_TmrClksrc

This enum describes the Timer Clock source selection.

Enumeration values:

CSL_TMR_CLKSRC_INTERNAL	Timer clock INTERNAL source selection
CSL_TMR_CLKSRC_TMRINP	Timer clock Timer input pin source selection

7.4.5 CSL_TmrEnamode

enum CSL_TmrEnamode

This enum describes the enabling/disabling of Timer.

Enumeration values:

<i>CSL_TMR_ENAMODE_DISABLE</i>	The timer is disabled and maintains current value
<i>CSL_TMR_ENAMODE_ENABLE</i>	The timer is enabled one time
<i>CSL_TMR_ENAMODE_CONT</i>	The timer is enabled continuously

7.4.6 CSL_TmrPulseWidth

enum CSL_TmrPulseWidth

This enum describes the Timer Clock cycles (1/2/3/4).

Enumeration values:

<i>CSL_TMR_PWID_ONECLK</i>	One timer clock cycle
<i>CSL_TMR_PWID_TWOCLKS</i>	Two timer clock cycle
<i>CSL_TMR_PWID_THREECLKS</i>	Three timer clock cycle
<i>CSL_TMR_PWID_FOURCLKS</i>	Four timer clock cycle

7.4.7 CSL_TmrClockPulse

enum CSL_TmrClockPulse

This enum describes the mode of Timer Clock (Pulse/Clock).

Enumeration values:

<i>CSL_TMR_CP_PULSE</i>	Pulse mode
<i>CSL_TMR_CP_CLOCK</i>	Clock mode

7.4.8 CSL_TmrInvInp

enum CSL_TmrInvInp

This enum describes the Timer input inverter control.

Enumeration values:

<i>CSL_TMR_INVINP_UNINVERTED</i>	Uninverted timer input drives timer
<i>CSL_TMR_INVINP_INVERTED</i>	Inverted timer input drives timer

7.4.9 CSL_TmrInvOutp

enum CSL_TmrInvOutp

This enum describes the Timer output inverter control.

Enumeration values:

<i>CSL_TMR_INVOUTP_UNINVERTED</i>	Uninverted timer output
<i>CSL_TMR_INVOUTP_INVERTED</i>	Inverted timer output

7.4.10 CSL_TmrMode

enum CSL_TmrMode

This enum describes the mode of Timer (GPT/WDT/Chained/Unchained).

Enumeration values:

<i>CSL_TMR_TIMMODE_GPT</i>	The timer is in 64-bit GP timer mode
<i>CSL_TMR_TIMMODE_DUAL_UNCHAINED</i>	The timer is in dual 32-bit timer, unchained mode
<i>CSL_TMR_TIMMODE_WDT</i>	The timer is in 64-bit Watchdog timer mode
<i>CSL_TMR_TIMMODE_DUAL_CHAINED</i>	The timer is in dual 32-bit timer, chained mode

7.4.11 CSL_TmrState

enum CSL_TmrState

This enum describes the reset condition of Timer (ON/OFF).

Enumeration values:

<i>CSL_TMR_TIMxxRS_RESET_ON</i>	Timer TIMxx is in reset
<i>CSL_TMR_TIMxxRS_RESET_OFF</i>	Timer TIMHI is not in reset. TIMHI can be used as a 32-bit timer

7.4.12 CSL_TmrTstat

enum CSL_TmrTstat

This enum describes the status of Timer.

Enumeration values:

<i>CSL_TMR_TSTAT_HIGH</i>	Timer status drives High
<i>CSL_TMR_TSTAT_LOW</i>	Timer status drives Low

7.4.13 CSL_TmrWdflagBitStatus

enum CSL_TmrWdflagBitStatus

This enumeration describes the flag bit status of the timer in watchdog mode.

Enumeration values:

<i>CSL_TMR_WDFLAG_NOTIMEOUT</i>	No watchdog timeout occurred
<i>CSL_TMR_WDFLAG_TIMEOUT</i>	Watchdog timeout occurred

7.5 Macros

#define CSL_TMR_CONFIG_DEFAULTS

Value:

```
{ \
    CSL_TMR_EMUMGT_CLKSPD_RESETVAL, \
    CSL_TMR_TIMLO_RESETVAL, \
    CSL_TMR_TIMHI_RESETVAL, \
    CSL_TMR_PRDLO_RESETVAL, \
    CSL_TMR_PRDHI_RESETVAL, \
    CSL_TMR_TCR_RESETVAL, \
    CSL_TMR_TGCR_RESETVAL, \
    CSL_TMR_WDTCR_RESETVAL \
}
```

Default values for Config structure.

#define CSL_TMR_HWSETUP_DEFAULTS

Value:

```
{ \
    CSL_TMR_PRDLO_RESETVAL, \
    CSL_TMR_PRDHI_RESETVAL, \
    CSL_TMR_TIMLO_RESETVAL, \
    CSL_TMR_TIMHI_RESETVAL, \
    (CSL_TmrIpGate)CSL_TMR_TCR_TIEN_HI_RESETVAL, \
    (CSL_TmrClksrc)CSL_TMR_TCR_CLKSRC_HI_RESETVAL, \
    (CSL_TmrPulseWidth)CSL_TMR_TCR_PWID_HI_RESETVAL, \
    (CSL_TmrClockPulse)CSL_TMR_TCR_CP_HI_RESETVAL, \
    (CSL_TmrInvInp)CSL_TMR_TCR_INVINP_HI_RESETVAL, \
    (CSL_TmrInvOutp)CSL_TMR_TCR_INVOUTP_HI_RESETVAL, \
    (CSL_TmrIpGate)CSL_TMR_TCR_TIEN_LO_RESETVAL, \
    (CSL_TmrClksrc)CSL_TMR_TCR_CLKSRC_LO_RESETVAL, \
    (CSL_TmrPulseWidth)CSL_TMR_TCR_PWID_LO_RESETVAL, \
    (CSL_TmrClockPulse)CSL_TMR_TCR_CP_LO_RESETVAL, \
    (CSL_TmrInvInp)CSL_TMR_TCR_INVINP_LO_RESETVAL, \
    (CSL_TmrInvOutp)CSL_TMR_TCR_INVOUTP_LO_RESETVAL, \
    CSL_TMR_TGCR_PSCHI_RESETVAL, \
    (CSL_TmrMode)CSL_TMR_TGCR_TIMMODE_RESETVAL \
}
```

Default values for hardware setup parameters.

Chapter 8

DAT MODULE

Topics

<u>8.1 Overview</u>
<u>8.2 Functions</u>
<u>8.3 Data Structures</u>
<u>8.4 Macros</u>

8.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within DAT module.

The data module (DAT) is used to move data around by means of EDMA hardware. This module serves as a level of abstraction such that it works the same for devices that have the DMA peripheral as for devices that have the EDMA peripheral.

8.2 Functions

This section lists the functions available in the DAT module.

8.2.1 DAT_open

Int16 DAT_open ([DAT_Setup](#) * **setup**)

Description

This API,

- a. Sets up the channel to Parameter set mapping
- b. Sets up the priority. This is essentially done by specifying the queue to which the channel is submitted to viz. Queue0- Queue3 with Queue 0 being the highest priority.
- c. Enables the region access bit for the channel, if a region is specified.

Arguments

setup Pointer to the DAT setup structure

Return Value

CSL_SOK

Pre Condition

None

Post Condition

The EDMA registers are configured with the setup values passed.

Modifies

EDMA registers

Example

```
DAT_Setup  datSetup;
datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
```

8.2.2 DAT_close

void DAT_close (**void**)

Description

This API disables the region access bit, if specified.

Arguments

None

Return Value

None

Pre Condition

DAT_open() must be successfully invoked prior to this call.

Post Condition

None

Modifies

None

Example

```
DAT_Setup  datSetup;
datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
DAT_close();
```

8.2.3 DAT_copy

```

Uint32 DAT_copy      (   void *      src,
                        void *      dst,
                        Uint16      byteCnt
                        )

```

Description

This API copies a linear block of data from *Src* to *Dst* using EDMA hardware, depending on the device. The arguments are checked for alignment. For best efficiency, the source and destination addresses should be aligned on an 8-byte boundary, with the transfer rate a multiple of eight.

Arguments

<i>src</i>	Source memory address for the data transfer
<i>dst</i>	Destination memory address of the data transfer
<i>byteCnt</i>	Number of bytes to be transferred

Return Value **Uint32**

tccNum - Transfer completion code

Pre Condition

DAT_open() must be successfully invoked prior to this call.

Post Condition

The EDMA registers are configured to transfer *byteCnt* bytes from the source memory address to the destination memory address.

Modifies

EDMA registers

Example

```

DAT_Setup    datSetup;
Uint8        dstld[8*16];
Uint8        srcld[8*16];
datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
DAT_copy(&srcld,&dstld,256);
...
DAT_close();

```

8.2.4 DAT_fill

```

Uint32 DAT_fill (    void *          dst,
                    Uint16          byteCnt,
                    Uint32 *        value
                  )

```

Description

This API fills a linear block of memory with the specified fill value using EDMA hardware

Arguments

<code>dst</code>	Destination memory address to be filled
<code>byteCnt</code>	Number of bytes to be filled
<code>value</code>	Value to be filled

Return Value `Uint32`

`tccNum` - Transfer completion code

Pre Condition

`DAT_open()` must be successfully invoked prior to this call.

Post Condition

The EDMA registers are configured to transfer a value to `byteCnt` bytes of the destination memory address.

Modifies

EDMA registers

Example

```

DAT_Setup    datSetup;
Uint8        dst[8*16];
Uint8        fillVal;

datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;

```

```

datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
fillVal = 0x5a;
DAT_fill(&dst,256,(Uint32*)&fillVal);
...
DAT_close();

```

8.2.5 DAT_wait

void DAT_wait (**Uint32** *id*)

Description

This API Waits for completion of the ongoing transfer.

Arguments

id Transfer completion number of the previous transfer

Return Value

None

Pre Condition

DAT_copy()/DAT_fill must be successfully invoked prior to this call.

Post Condition

Indicates that the transfer ongoing is complete.

Modifies

None

Example

```

DAT_Setup    datSetup;
Uint8        dstld[8*16];
Uint8        srcld[8*16];
Uint32        id;
datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
id = DAT_copy(&srcld,&dstld,256);

DAT_wait(id);
...
DAT_close();

```

8.2.6 DAT_busy

Int16 DAT_busy (**Uint32** *id*)

Description

This API polls for transfer completion.

Arguments

id Transfer completion number of the previous transfer

Return Value

Int16 TRUE/FALSE

Pre Condition

DAT_copy()/DAT_fill must be successfully invoked prior to this call.

Post Condition

Indicates that the transfer ongoing is complete.

Modifies

None

Example

```

DAT_Setup    datSetup;
Uint8        dstId[8*16];
Uint8        srcId[8*16];
Uint32       id;
datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
id = DAT_copy(&srcId,&dstId,256);

do {
    ...
}while (DAT_busy(id));
...
DAT_close();

```

8.2.7 DAT_copy2d

Uint32 DAT_copy2d (**Uint32** *type*,
void * *src*,
void * *dst*,
Uint16 *lineLen*,
Uint16 *lineCnt*,

Uint16
linePitch
)
Description

This API copies data from source to destination for two dimension transfer.

Arguments

type	Indicates the type of the transfer DAT_1D2D - 1 dimension to 2 dimension DAT_2D1D - 2 dimension to 1 dimension DAT_2D2D - 2 dimension to 2 dimension
src	Source memory address for the data transfer
dst	Destination memory address of the data transfer
lineLen	Number of bytes per line
lineCnt	Number of lines
linePitch	Number of bytes between start of one line to start of next line

Return Value Uint32

TccNum – Transfer completion code

Pre Condition

DAT_open() must be successfully invoked prior to this call.

Post Condition

The EDMA registers are configured for the transfer.

Modifies

EDMA registers

Example

```

DAT_Setup    datSetup;
Uint8        dst2d[8][20];
Uint8        src1d[8*16];
Uint32       id;

datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
id = DAT_copy2d(DAT_1D2D,src1d,dst2d,16,8,20);

do {
    ...
}while (DAT_busy(id));
...

```

```
DAT_close();
```

8.2.8 DAT_setPriority

```
void DAT_setPriority ( Int priority )
```

Description

Sets the priority bit value PRI of OPT register. The priority value can be set by using the type CSL_DatPriority.

Arguments

priority	Priority value
----------	----------------

Return Value

None

Pre Condition

DAT_open must be successfully invoked prior to this call.

Post Condition

OPT register is set for the priority value

Modifies

OPT register

Example

```
DAT_Setup    datSetup;

datSetup.qchNum = CSL_DAT_QCHA_0;
datSetup.regionNum = CSL_DAT_REGION_GLOBAL ;
datSetup.tccNum = 1;
datSetup.paramNum = 0 ;
datSetup.priority = CSL_DAT_PRI_0;

DAT_open(&datSetup);
...
DAT_setPriority(CSL_DAT_PRI_3);
```

8.3 Data Structures

This section lists the data structures available in the DAT module.

8.3.1 DAT_Setup

Detailed description

DAT Setup structure.

Field Documentation**Int DAT_Setup::paramNum**

Parameter set number for this channel

Int DAT_Setup::priority

Priority/Queue number on which the transfer requests are submitted

Int DAT_Setup::qchNum

QDMA Channel number being requested

Int DAT_Setup::regionNum

Region of operation

Int DAT_Setup::tccNum

Transfer completion code dedicated for DAT

8.4 Macros

#define DAT_1D2D 0x1
Transfer type is 1D2D

#define DAT_2D1D 0x2
Transfer type is 2D1D

#define DAT_2D2D 0x3
Transfer type is 2D2D

Chapter 9

INTC MODULE

Topics

<u>9.1 Overview</u>
<u>9.2 Functions</u>
<u>9.3 Data Structures</u>
<u>9.4 Enumerations</u>
<u>9.5 Macros</u>

9.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within INTC module.

The CPU has one exception input, one non-maskable interrupt, 12 maskable interrupts, and two dedicated emulation interrupts. The Interrupt Controller supports up to 128 system events. There are 128 system events that act as inputs to the Interrupt Controller. They consist of both internally-generated events (within the megamodule) and chip-level events. In addition to these 128 events, INTC also receives (and routes straight through to the CPU) the non-maskable and reset events. From these event inputs, the Interrupt Controller outputs signals to the CPU:

- One maskable, hardware exception (EXCEP)
- Twelve maskable hardware interrupts (INT4 ... INT15)
- One non-maskable signal which can be used as either an interrupt or exception (NMI)
- One reset signal (RESET)

9.2 Functions

This section lists the functions available in the INTC module.

9.2.1 CSL_intcInit

CSL_Status CSL_intcInit ([CSL_IntcContext](#) * *pContext*)

Description

This is the initialization function for the INTC CSLT. This function must be called before calling any other API from this CSL. The context should be initialized such that numEvtEntries is equal to the number of records capable of being held in the eventhandlerRecord.

Arguments

pContext Pointer to module-context structure.

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

The context should be initialized such that numEvtEntries is equal to the number of records capable of being held in the eventhandlerRecord.

Post Condition

None

Modifies

None

Example

```
CSL_IntcContext context;
CSL_IntcEventHandlerRecord recordTable[10];

context.numEvtEntries = 10;
context.eventhandlerRecord = &recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK) {
    exit;
}
```

9.2.2 CSL_intcOpen

```

CSL_IntcHandle CSL_intcOpen ( CSL_IntcObj *      intcObj,
                             CSL_IntcEventId     eventId,
                             CSL_IntcParam *      param,
                             CSL_Status *         status
                             )

```

Description

The API would reserve an interrupt-event for use. It returns a valid handle to the event only if the event is not currently allocated. The user could release the event after use by calling CSL_intcClose(..). The CSL-object ('intcObj') that the user passes would be used to store information pertaining handle.

Arguments

intcObj	Pointer to the CSL-object allocated by the user
eventId	The event-id of the interrupt
param	Pointer to the Intc specific parameter
status	(Optional) pointer for returning status of the function call

Return Value CSL_IntcHandle

Valid INTC handle identifying the event

Pre Condition

The INTC must be successfully initialized via CSL_intcInit() before calling this function.

Post Condition

1. INTC object structure is populated
2. The status is returned in the status variable. If status returned is

- CSL_SOK Valid intc handle is returned
- CSL_ESYS_FAIL The open command failed

Modifies

1. The status variable
2. INTC object structure

Example

```

CSL_IntcObj      intcObj20;
CSL_IntcGlobalEnableState state;
CSL_IntcContext  context;
CSL_Status       intStat;
CSL_IntcParam    vectId;
CSL_IntcHandle   hIntc20;

```

```

context.numEvtEntries = 0;
context.eventhandlerRecord = NULL;
// Init Module
CSL_intcInit(&context);

// NMI Enable
CSL_intcGlobalNmiEnable();

// Enable Global Interrupts
intStat = CSL_intcGlobalEnable(&state);

// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (  &intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId,
                        &intStat);

// Close handle
CSL_intcClose(hIntc20);

```

9.2.3 CSL_intcClose

CSL_Status CSL_intcClose ([CSL_IntcHandle](#) *hIntc*)

Description

This intc handle can no longer be used to access the event. The event is de-allocated and further access to the event resources are possible only after opening the event object again.

Arguments

hIntc Handle identifying the event

Return Value CSL_Status

- CSL_SOK - Close successful
- CSL_INTC_BADHANDLE - The handle passed is invalid

Pre Condition

Functions CSL_intcInit() and CSL_intcOpen() have to be called in that order successfully before calling this function.

Post Condition

1. CPU interrupt could be used again
2. The intc CSL APIs can not be called until the intc CSL is reopened again using CSL_intcOpen().

Modifies

CSL_intcObj structure values

Example

```

CSL_IntcContext    context;
CSL_Status          intStat;

```

```

CSL_IntcParam          vectId;
CSL_IntcObj            intcObj20;
CSL_IntcHandle         hIntc20;
CSL_IntcEventHandlerRecord recordTable[10];

context.numEvtEntries = 10;
context.eventHandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
{exit;
}
// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (&intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId, \
                        &intStat);

// Close handle
CSL_intcClose(hIntc20);

```

9.2.4 CSL_intcPlugEventHandler

```

CSL_Status CSL_intcPlugEventHandler( CSL\_IntcHandle          hIntc,
                                     CSL\_IntcEventHandlerRecord * eventHandlerRecord
                                   )

```

Description

Associate an event-handler with an event CSL_intcPlugEventHandler(..) ties an event-handler to an event; so that the occurrence of the event, would result in the event-handler being invoked.

Arguments

<code>hIntc</code>	Handle identifying the interrupt-event
<code>eventHandlerRecord</code>	Provides the details of the event-handler

Return Value

Returns the address of the previous handler

Example

```

CSL_IntcObj    intcObj20;
CSL_IntcGlobalEnableState state;
CSL_IntcContext context;
CSL_Status     intStat;
CSL_IntcParam vectId;
CSL_IntcHandle hIntc20;
CSL_IntcEventHandlerRecord EventRecord;

context.numEvtEntries = 0;

```

```

context.eventhandlerRecord = NULL;
// Init Module
CSL_intcInit(&context);

// NMI Enable
CSL_intcGlobalNmiEnable();

// Enable Global Interrupts
intStat = CSL_intcGlobalEnable(&state);

// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (&intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId ,
                        &intStat);

EventRecord.handler = (CSL_IntcEventHandler)&event20Handler;
EventRecord.arg = hIntc20;
CSL_intcPlugEventHandler(hIntc20,&EventRecord);
// Close handle
CSL_intcClose(hIntc20);
}

void event20Handler( CSL_IntcHandle hIntc)
{
}

```

9.2.5 CSL_intcHooklsr

```

CSL_Status CSL_intcHooklsr          ( CSL\_IntcVectId          evtId,
                                     void *                          isrAddr
                                     )

```

Description

Hook up an exception handler This API hooks up the handler to the specified exception. Note: In this case, it is done by inserting a B(ranch) instruction to the handler. Because of the restriction in the instruction, the handler must be within 32MB of the exception vector. In addition, the function assumes that the exception vector table is located at its default ("low") address.

Arguments

<i>evtId</i>	Interrupt Vector identifier
<i>isrAddr</i>	Pointer to the handler

Return Value CSL_Status

CSL_SOK - CSL_intcHooklsr Successful

Example

```

/* prototype declaration
 *      interrupt void isrVect4();
 */

```

```

CSL_IntcContext          context;
CSL_Status               intStat;
CSL_IntcParam            vectId;
CSL_IntcObj              intcObj20;
CSL_IntcHandle            hIntc20;
CSL_IntcEventHandlerRecord recordTable[10];
CSL_IntcGlobalEnableState state;

context.numEvtEntries = 10;
context.eventhandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
    exit;
// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (&intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId ,
                        &intStat);

CSL_intcGlobalNmiEnable();
// Enable Global Interrupts
intStat = CSL_intcGlobalEnable(&state);

// Hook Isr appropriately
CSL_intcHookIsr(CSL_INTC_VECTID_4,&isrVect4);
...
}
interrupt void isrVect4() {

}

```

9.2.6 CSL_intcHwControl

```

CSL_Status CSL_intcHwControl ( CSL\_IntcHandle          hIntc,
                              CSL\_IntcHwControlCmd       command,
                              void *                     commandArg
                              )

```

Description

This API perform a control-operation. This API is used to invoke any of the supported control-operations supported by the module.

Arguments

<code>hIntc</code>	Handle identifying the event
<code>command</code>	The command to this API indicates the action to be taken on INTC.

commandArg An optional argument.

Return Value CSL_Status

- CSL_SOK - HwControl successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

CSL_intcOpen() must be invoked before this call.

Post Condition

INTC registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

The hardware registers of INTC.

Example

```

CSL_IntcObj   intcObj20;
CSL_IntcGlobalEnableState state;
CSL_IntcContext   context;
CSL_Status   intStat;
CSL_IntcParam vectId;
CSL_IntcHandle hIntc20;

context.numEvtEntries = 0;
context.eventhandlerRecord = NULL;
// Init Module
CSL_intcInit(&context);

// NMI Enable
CSL_intcGlobalNmiEnable();

// Enable Global Interrupts
intStat = CSL_intcGlobalEnable(&state);

// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (  &intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId,
                        &intStat);
CSL_intcHwControl(hIntc20,CSL_INTC_CMD_EVTENABLE,NULL);

```

9.2.7 CSL_intcGetHwStatus

```
CSL_Status CSL_intcGetHwStatus      ( CSL\_IntcHandle          hIntc,
                                     CSL\_IntcHwStatusQuery    query,
                                     void *                          response
                                     )
```

Description

Queries the peripheral for status. The CSL_intcGetHwStatus(..) API could be used to retrieve status or configuration information from the peripheral. The user must allocate an object that would hold the retrieved information and pass a pointer to it to the function. The type of the object is specific to the query-command.

Arguments

hIntc	Handle identifying the event
query	The query to this API of INTC which indicates the status to be returned.
response	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_INVQUERY - Invalid query

Pre Condition

The functions CSL_intcInit(), CSL_intcOpen() must be called successfully in that order before this API can be invoked.

Post Condition

None

Modifies

The input argument "response" is modified.

Example

```
CSL_IntcContext      context;
CSL_Status           intStat;
CSL_IntcParam        vectId;
CSL_IntcObj          intcObj20;
CSL_IntcHandle       hIntc20;
CSL_IntcEventHandlerRecord recordTable[10];
CSL_IntcGlobalEnableState state;
UInt32               intrStat;

context.numEvtEntries = 10;
context.eventhandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;
```

```
// Init Module
if (CSL_intcInit(&context) != CSL_SOK)
    exit;
// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (&intcObj20,
                        CSL_INTC_EVENTID_RIOINT0,
                        &vectId ,
                        &intStat);

// NMI Enable
CSL_intcGlobalNmiEnable();

// Enable Global Interrupts
intStat = CSL_intcGlobalEnable(&state);

do
{
    CSL_intcGetHwStatus(hIntc20,
                        CSL_INTC_QUERY_PENDSTATUS,\
                        (void*)&intrStat);
} while (!intrStat);

// Close handle
CSL_intcClose(hIntc20);
```

9.2.8 CSL_intcGlobalEnable

CSL_Status CSL_intcGlobalEnable (CSL_IntcGlobalEnableState * *prevState*)

Description

Globally enable interrupts. The API enables the global interrupt by manipulating the processor's global interrupt enable/disable flag. If the user wishes to restore the enable-state at a later point, they may store the current state using the parameter, which could be used with CSL_intcGlobalRestore(..). CSL_intcGlobalEnable(..) must be called from a privileged mode.

Arguments

<i>prevState</i>	Pointer to object that would store current stateObject that contains information about previous state
------------------	---

Return Value CSL_Status

- CSL_SOK on success

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_intcGlobalEnable (NULL);
```

9.2.9 CSL_intcGlobalDisable

CSL_Status CSL_intcGlobalDisable (CSL_IntcGlobalEnableState * *prevState*)

Description

Globally disable interrupts. The API disables the global interrupt by manipulating the processor's global interrupt enable/disable flag. If the user wishes to restore the enable-state at a later point, they may store the current state using the parameter, which could be used with CSL_intcGlobalRestore(..). CSL_intcGlobalDisable(..) must be called from a privileged mode.

Arguments

prevState	Pointer to object that would store current stateObject that contains information about previous state
-----------	---

Return Value CSL_Status

- CSL_SOK on success

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_intcGlobalDisable(NULL);
```

9.2.10 CSL_intcGlobalRestore

CSL_Status CSL_intcGlobalRestore (CSL_IntcGlobalEnableState *prevState*)

Description

Restores global interrupt enable/disable to a previous state. The API restores the global interrupt enable/disable state to a previous state as recorded by the global-event-enable state passed as an argument. CSL_intcGlobalRestore(..) must be called from a privileged mode.

Arguments

prevState	Object containing information about previous state
-----------	--

Return Value CSL_Status

- CSL_SOK on success

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_IntcGlobalEnableState  prevState;
CSL_intcGlobalRestore(prevState);
```

9.2.11 CSL_intcGlobalNmiEnable

CSL_Status CSL_intcGlobalNmiEnable (void)

Description

This API enables global NMI.

Arguments

None

Return Value CSL_Status

- CSL_SOK on success

Example

```
CSL_intcGlobalNmiEnable();
```

9.2.12 CSL_intcGlobalExcepEnable

CSL_Status CSL_intcGlobalExcepEnable (void)

Description

This API enables global exception.

Arguments

None

Return Value CSL_Status

- CSL_SOK on success

Example

```
CSL_intcGlobalExcepEnable();
```

9.2.13 CSL_intcGlobalExtExcepEnable

CSL_Status CSL_intcGlobalExtExcepEnable (void)

Description

This API enables external exception.

Arguments

None

Return Value CSL_Status

- CSL_SOK on success

Example

```
CSL_intcGlobalExtExcepEnable();
```

9.2.14 CSL_intcGlobalExcepClear

CSL_Status CSL_intcGlobalExcepClear ([CSL_IntcExcep](#) exc)

Description

This API clears Global Exceptions.

Arguments

exc Exception to be cleared NMI/SW/EXT/INT

Return Value CSL_Status

- CSL_SOK on success

Example

```
CSL_IntcExcep exc;
```

```
CSL_intcGlobalExcepClear(exc);
```

9.2.15 CSL_intcExcepAllEnable

CSL_Status CSL_intcExcepAllEnable ([CSL_IntcExcepEn](#) excMask,
 CSL_BitMask32 excVal,
 CSL_BitMask32 * prevState)

Description

This API enables all exceptions.

Arguments

excMask Exception to be cleared NMI/SW/EXT/INT

excVal Event Value

prevState Pointer to Pre state information

Return Value CSL_Status

- CSL_SOK on success

Example

```

CSL_IntcContext          context;
CSL_IntcEventHandlerRecord recordTable[10];
CSL_IntcGlobalEnableState prevState;
Uint32                  intcStat;

context.numEvtEntries = 10;
context.eventhandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
{
    exit;
    // Enable exception events 9,10,11.
    intcStat = CSL_intcExcepAllEnable(CSL_INTC_EXCEP_0TO31,
                                      0x0F00,&prevState);

```

9.2.16 CSL_intcExcepAllDisable

```

CSL_Status CSL_intcExcepAllDisable ( CSL\_IntcExcepEn      excepMask,
                                     CSL_BitMask32          excVal,
                                     CSL_BitMask32 *          prevState
                                     )

```

Description

This API disables all exceptions.

Arguments

<code>excepMask</code>	Exception Mask
<code>excVal</code>	Event Value
<code>prevState</code>	Previous state information

Return Value CSL_Status

- CSL_SOK on success

Example

```

CSL_IntcContext          context;
CSL_Status               intcStat;
CSL_IntcEventHandlerRecord recordTable[10];
CSL_IntcGlobalEnableState prevState;

context.numEvtEntries = 10;
context.eventhandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;

// Init Module

```

• • •

```
if (CSL_intcInit(&context) != CSL_SOK)
{
    exit;
}
// Enable exception events 9,10,11.
intcStat = CSL_intcExcepAllDisable(CSL_INTC_EXCEP_0TO31, \
                                     0x0F00,&prevState);
```

9.2.17 CSL_intcExcepAllRestore

```
CSL_Status CSL_intcExcepAllRestore ( CSL\_IntcExcepEn excepMask,
                                     CSL_IntcGlobalEnableState restoreVal
                                   )
```

Description

This API restores all exceptions.

Arguments

excepMask	Exception Mask
restoreVal	BitMask to be restored

Return Value CSL_Status

- CSL_SOK on success

Example

```

CSL_IntcContext      context;
CSL_Status            intcStat;
CSL_IntcEventHandlerRecord recordTable[10];
CSL_IntcGlobalEnableState prevState;

context.numEvtEntries = 10;
context.eventhandlerRecord =( CSL_IntcEventHandlerRecord*)
                             &recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
{
    exit;
}
intcStat = CSL_intcExcepAllDisable(CSL_INTC_EXCEP_0TO31,0x0F00, \
                                   &prevState);

// Restore
intcStat = CSL_intcExcepAllRestore(CSL_INTC_EXCEP_0TO31,
                                   prevState);

```

9.2.18 CSL_intcExcepAllClear

```
CSL_Status CSL_intcExcepAllClear      ( CSL\_IntcExcepEn      exceptMask,
                                         CSL_BitMask32      excVal
                                         )
```

Description

This clears the exception flags.

Arguments

<i>exceptMask</i>	Exception Mask
<i>excVal</i>	Holder for the event bitmask to be cleared

Return Value CSL_Status

- CSL_SOK - Intc Excep All Clear return successful

Pre Condition

CSL_intcInit() and CSL_intcExcepAllEnable() must be called before use of this API.

Post Condition

None

Modifies

None

Example

```
CSL_IntcContext  context;
CSL_Status      intcStat;
CSL_IntcEventHandlerRecord recordTable[10];
context.numEvtEntries = 10;
context.eventhandlerRecord =
(CSL_IntcEventHandlerRecord*)&recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
{
    exit;
}
// Clear exception events 9,10,11.

intcStat = CSL_intcExcepAllClear(CSL_INTC_EXCEP_0TO31,0x0F00);
```

9.2.19 CSL_intcExcepAllStatus

```
CSL_Status CSL_intcExcepAllStatus      ( CSL\_IntcExcepEn      exceptMask,
                                         CSL_BitMask32 *      status
                                         )
```

Description

This obtains the status of the exception flags.

Arguments

<code>excepMask</code>	Exception Mask
<code>status</code>	Holder for the event bitmask to be cleared

Return Value `CSL_Status`

- `CSL_SOK` - intc Excep All Status return successful

Pre Condition

`CSL_intcInit()` must be called before use of this API.

Post Condition

None

Modifies

None

Example

```
CSL_IntcContext      context;
CSL_Status           intcStat;
CSL_BitMask32        exp0Stat;
CSL_IntcEventHandlerRecord recordTable[10];
context.numEvtEntries = 10;
context.eventhandlerRecord = (CSL_IntcEventHandlerRecord*)
                             &recordTable;

// Init Module

if (CSL_intcInit(&context) != CSL_SOK)
{
    exit;
}
intcStat = CSL_intcExcepAllStatus(
                                CSL_INTC_EXCEP_0TO31,
                                &exp0Stat);
```

9.2.20 CSL_intcQueryDropStatus

CSL_Status `CSL_intcQueryDropStatus` ([CSL_IntcDropStatus](#) * *dropStat*)

Description

Queries the peripheral for Drop status. The `CSL_intcQueryDropStatus(..)` API could be used to retrieve drop status.

Arguments

<code>dropStat</code>	Pointer to drop status structure
-----------------------	----------------------------------

Return Value `CSL_Status`

- `CSL_SOK` - Status info return successful
- `CSL_ESYS_INVPARAMS` - Invalid handle

Pre Condition

CSL_intcInit(), CSL_intcOpen() must be invoked before this call.

Post Condition

None

Modifies

None

Example

```

CSL_IntcContext          context;
CSL_IntcParam            vectId;
CSL_IntcObj              intcObj20;
CSL_IntcHandle           hIntc20;
CSL_IntcDropStatus       drop;
CSL_IntcEventHandlerRecord recordTable[10];

context.numEvtEntries = 10;
context.eventhandlerRecord = (CSL_IntcEventHandlerRecord*)
                             &recordTable;

// Init Module
...
if (CSL_intcInit(&context) != CSL_SOK)
    exit;
// Opening a handle for the Event 20 at vector id 4

vectId = CSL_INTC_VECTID_4;
hIntc20 = CSL_intcOpen (&intcObj20,
                       CSL_INTC_EVENTID_RIOINT0,
                       &vectId ,
                       NULL);

// Drop Enable
CSL_intcHwControl(hIntc20,CSL_INTC_CMD_EVTDROPENABLE,NULL);
// Query Drop status
CSL_intcQueryDropStatus(&drop);

// Close handle
CSL_intcClose(hIntc20);

```

9.3 Data Structures

This section lists the data structures available in the INTC module.

9.3.1 CSL_IntcObj

Detailed description

The interrupt handle object. This object is used referenced by the handle to identify the event.

Field Documentation

CSL_IntcEventId CSL_IntcObj::eventId

The event-id

[CSL_IntcVectId](#) **CSL_IntcObj::vectId**

The vector-id

9.3.2 CSL_IntcContext

Detailed description

INTC Module Context

Field Documentation

CSL_BitMask32 CSL_IntcContext::eventAllocMask[(CSL_INTC_EVENTID_CNT + 31) / 32]

Event allocation mask

[CSL_IntcEventHandlerRecord*](#) **CSL_IntcContext::eventhandlerRecord**

Pointer to the event handle record

UInt16 CSL_IntcContext::numEvtEntries

Number of event entries

Int8 CSL_IntcContext::offsetResv[128]

Reserved

9.3.3 CSL_IntcEventHandlerRecord

Detailed description

Event Handler Record. Used to setup the event-handler using CSL_intcPlugEventHandler(..)

Field Documentation

void* CSL_IntcEventHandlerRecord::arg

The argument to be passed to the handler when it is invoked.

CSL_IntcEventHandler CSL_IntcEventHandlerRecord::handler

Pointer to the event handler

9.3.4 CSL_IntcDropStatus

Detailed description

The drop status structure. This object is used along with the CSL_intcQueryDropStatus() API.

Field Documentation**Bool CSL_IntcDropStatus::drop**

Whether dropped/not

CSL_IntcEventId CSL_IntcDropStatus::eventId

The event-id

[CSL_IntcVectId](#) CSL_IntcDropStatus::vectId

The vect-id

9.4 Enumerations

This section lists the enumerations available in the INTC module.

9.4.1 CSL_IntcVectId

enum CSL_IntcVectId

Interrupt Vector Ids

Enumeration values:

<i>CSL_INTC_VECTID_NMI</i>	Should be used only along with <i>CSL_intcHookIsr()</i>
<i>CSL_INTC_VECTID_4</i>	CPU Vector 4
<i>CSL_INTC_VECTID_5</i>	CPU Vector 5
<i>CSL_INTC_VECTID_6</i>	CPU Vector 6
<i>CSL_INTC_VECTID_7</i>	CPU Vector 7
<i>CSL_INTC_VECTID_8</i>	CPU Vector 8
<i>CSL_INTC_VECTID_9</i>	CPU Vector 9
<i>CSL_INTC_VECTID_10</i>	CPU Vector 10
<i>CSL_INTC_VECTID_11</i>	CPU Vector 11
<i>CSL_INTC_VECTID_12</i>	CPU Vector 12
<i>CSL_INTC_VECTID_13</i>	CPU Vector 13
<i>CSL_INTC_VECTID_14</i>	CPU Vector 14
<i>CSL_INTC_VECTID_15</i>	CPU Vector 15
<i>CSL_INTC_VECTID_COMBINE</i>	Should be used at the time of opening an Event handle to specify that the event needs to go to the combiner
<i>CSL_INTC_VECTID_EXCEP</i>	Should be used at the time of opening an Event handle to specify that the event needs to go to the exception combiner.

9.4.2 CSL_IntcHwControlCmd

enum CSL_IntcHwControlCmd

Enumeration of the control commands

These are the control commands that could be used with *CSL_intcHwControl(..)*. Some of the commands expect an argument as documented along side the description of the command.

Enumeration values:

<i>CSL_INTC_CMD_EVTDISABLE</i>	Disables the event. Parameters: <i>CSL_IntcEnableState</i>
<i>CSL_INTC_CMD_EVTSET</i>	Sets the event manually. Parameters: <i>None</i>
<i>CSL_INTC_CMD_EVTCLEAR</i>	Clears the event (if pending). Parameters: <i>None</i>
<i>CSL_INTC_CMD_EVTDROPENABLE</i>	Enables the Drop Event detection feature for this event. Parameters: <i>None</i>

<code>CSL_INTC_CMD_EVTDROPDISABLE</code>	Disables the Drop Event detection feature for this event. Parameters: <i>None</i>
<code>CSL_INTC_CMD_EVTINVOKEFUNCTION</code>	To be used ONLY to invoke the associated Function handlewith Event when the user is writing an exception handling routine. Parameters: <i>None</i>
<code>CSL_INTC_CMD_EVTENABLE</code>	Enables the event. Parameters: <i>CSL_IntcEnableState</i>

9.4.3 CSL_IntcHwStatusQuery

enum CSL_IntcHwStatusQuery

Enumeration of the queries. These are the queries that could be used with `CSL_intcGetHwStatus(..)`. The queries return a value through the object pointed to by the pointer that it takes as an argument. The argument supported by the query is documented along side the description of the query.

Enumeration values:

<code>CSL_INTC_QUERY_PENDSTATUS</code>	The Pend Status of the Event is queried. Parameters: <i>Bool</i>
--	---

9.4.4 CSL_IntcExcepEn

enum CSL_IntcExcepEn

Enumeration of the exception mask registers. These are the symbols used along with the value to be programmed into the Exception mask register.

Enumeration values:

<code>CSL_INTC_EXCEP_32TO63</code>	Symbol for EXPMASK[1]. Parameters: <i>BitMask</i> for EXPMASK1
<code>CSL_INTC_EXCEP_64TO95</code>	Symbol for EXPMASK[2]. Parameters: <i>BitMask</i> for EXPMASK2
<code>CSL_INTC_EXCEP_96TO127</code>	Symbol for EXPMASK[3]. Parameters: <i>BitMask</i> for EXPMASK3
<code>CSL_INTC_EXCEP_0TO31</code>	Symbol for EXPMASK[0]. Parameters: <i>BitMask</i> for EXPMASK0

9.4.5 CSL_IntcExcep

enum CSL_IntcExcep

Enumeration of the exception

These are the symbols used along with the Exception Clear API.

Enumeration values:

<i>CSL_INTC_EXCEPTION_NMI</i>	Symbol for NMI. Parameters: <i>None</i>
<i>CSL_INTC_EXCEPTION_EXT</i>	Symbol for External Exception. Parameters: <i>None</i>
<i>CSL_INTC_EXCEPTION_INT</i>	Symbol for Internal Exception. Parameters: <i>None</i>
<i>CSL_INTC_EXCEPTION_SW</i>	Symbol for Software Exception Parameters: <i>None</i>

9.5 Macros

#define CSL_INTC_BADHANDLE (0)
Invalid handle

#define CSL_INTC_EVENTID_CNT 128
Number of Events in the System

#define CSL_INTC_EVTHANDLER_NONE ((CSL_IntcEventHandler) 0)
Indicates there is no associated event-handler

#define CSL_INTC_MAPPED_NONE (-1)
None mapped

Chapter 10

TSC MODULE

Topics

<u>10. 1 Overview</u>

<u>10. 2 Functions</u>
--

10.1 Overview

This chapter describes the Functions within TSC module.

Time Stamp Counter is a free running 64-bit CPU counter that advances each CPU clock after counting is enabled. The counter is accessed using two 32-bit read-only control registers, Time Stamp Counter Registers – Low (TSCL) and Time Stamp Counter Registers – High (TSCH). The counter is enabled by writing to TSCL. The value written is ignored. Once enabled, counting cannot be disabled under program control. Counting is disabled in the following cases:

- After exiting the reset state.
- When the CPU is fully powered down.

10.2 Functions

This section lists the functions available in the TSC module.

10.2.1 CSL_tscEnable

void CSL_tscEnable (void)

Description

This API enables the 64 bit time stamp counter. Time Stamp Counter stops only upon Reset or Power down. When time stamp counter is enabled (following a reset or power down of the CPU) it will initialize to 0 and begin incrementing once per CPU cycle. You cannot reset the time stamp counter.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

Time Stamp Counter value starts incrementing

Modifies

None

Example

```
CSL_tscEnable();
```

10.2.2 CSL_tscRead

CSL_Uint64 CSL_tscRead (void)

Description

Reads the 64-bit Time Stamp Counter and returns the 64 bit counter value.

Arguments

None

Return Value CSL_Uint64

64 Bit Time Stamp Counter Value

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Uint64      counterVal;  
  
...  
CSL_tscEnable();  
counterVal  = CSL_tscRead ();
```

Chapter 11

GPIO MODULE

Topics

<u>11. 1 Overview</u>
<u>11. 2 Functions</u>
<u>11. 3 Data Structures</u>
<u>11. 4 Enumerations</u>
<u>11. 5 Macros</u>

11.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within GPIO module. General-purpose input/output port (GPIO) with programmable interrupt/event generation modes having 16-pins.

The GPIO peripheral provides 16 dedicated general-purpose pins that can be configured as either inputs or outputs. Each GPx pin configured as an input can directly trigger a CPU interrupt or a GPIO event. The properties and functionalities of the GPx pins are covered by a set of CSL APIs.

To use the GPIO pins, you must first allocate a device using `GPIO_open()`, and then configure the Global Control register to determine the peripheral mode by using the configuration structure.

11.2 Functions

This section lists the functions available in the GPIO module.

11.2.1 CSL_gpioInit

CSL_Status CSL_gpioInit ([CSL_GpioContext](#) * *pContext*)

Description

This is the initialization function for the GPIO CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Pointer to module-context.Context information for the instance. As GPIO doesn't have any context based information user is expected to pass NULL.

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

The CSL for GPIO is initialized

Modifies

None

Example

```
CSL_gpioInit(NULL);
```

11.2.2 CSL_gpioOpen

[CSL_GpioHandle](#) CSL_gpioOpen ([CSL_GpioObj](#) * *pGpioObj*,
CSL_InstNum *gpioNum*,
[CSL_GpioParam](#) * *pGpioParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the GPIO instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of GPIO device. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input as an essential argument for rest of the GPIO CSL APIs.

Arguments

<code>pGpioObj</code>	Pointer to the GPIO instance object
<code>gpioNum</code>	Instance of the GPIO to which a handle is requested
<code>pGpioParam</code>	Pointer to module specific parameters
<code>pStatus</code>	Pointer for returning status of the function call

Return Value `CSL_GpioHandle`

Valid GPIO instance handle will be returned if status value is equal to `CSL_SOK`.

Pre Condition

The GPIO must be successfully initialized via `CSL_gpioInit()` before calling this function

Post Condition

1. GPIO object structure is populated
2. The status is returned in the status variable. If status returned is

- `CSL_SOK` - Valid gpio handle is returned
- `CSL_ESYS_FAIL` - The gpio instance is invalid
- `CSL_ESYS_INVPARAMS` - Invalid parameter

Modifies

1. The status variable
2. GPIO object structure

Example

```

CSL_Status      status;
CSL_GpioObj     gpioObj;
CSL_GpioHandle  hGpio;

hGpio = CSL_gpioOpen(&gpioObj, CSL_GPIO, NULL, &status);

```

11.2.3 CSL_gpioClose

CSL_Status CSL_gpioClose ([CSL_GpioHandle](#) *hGpio*)

Description

This function closes the specified instance of GPIO.

Arguments

<code>hGpio</code>	Handle to the GPIO instance
--------------------	-----------------------------

Return Value `CSL_Status`

- `CSL_SOK` - Close successful
- `CSL_ESYS_BADHANDLE` - Invalid handle

Pre Condition

Both `CSL_gpioInit()` and `CSL_gpioOpen()` must be called successfully in order before calling `CSL_gpioClose()`.

Post Condition

The GPIO CSL APIs can not be called until the GPIO CSL is reopened again using `CSL_gpioOpen()`.

Modifies

Obj structure values

Example

```
CSL_GpioHandle    hGpio;
CSL_Status        status;
CSL_GpioObj       gpioObj;

hGpio = CSL_gpioOpen (&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioClose(hGpio);
```

11.2.4 CSL_gpioHwSetup

```
CSL_Status CSL_gpioHwSetup ( CSL\_GpioHandle          hGpio,
                           CSL\_GpioHwSetup *        setup
                           )
```

Description

It configures the gpio registers as per the values passed in the hardware setup structure. But this is a dummy function for this module. It has been kept for future use.

Arguments

<code>hGpio</code>	Handle to the GPIO instance
<code>setup</code>	Pointer to hardware setup structure

Return Value `CSL_Status`

- `CSL_SOK` - Hardware setup successful.
- `CSL_ESYS_BADHANDLE` - Invalid handle
- `CSL_ESYS_INVPARAMS` - The param passed is invalid

Pre Condition

Both `CSL_gpiolnit()` and `CSL_gpioOpen()` must be called successfully in order before this function. The user has to allocate space for and fill in the main setup structure appropriately before calling this function.

Post Condition

GPIO registers are configured according to the hardware setup parameters.

Modifies

Registers of GPIO.

Example

```
CSL_GpioHandle    hGpio;
CSL_GpioObj       gpioObj;
CSL_GpioHwSetup    hwSetup;
CSL_Status        status;
```

```
hwSetup.extendSetup = NULL;

hGpio = CSL_gpioOpen(&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioHwSetup(hGpio, &hwSetup);
```

11.2.5 CSL_gpioHwSetupRaw

```
CSL_Status CSL_gpioHwSetupRaw      ( CSL\_GpioHandle      hGpio,
                                     CSL\_GpioConfig *    config
                                     )
```

Description

This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values.

Arguments

hGpio	Handle to the Gpio instance
config	Pointer to config structure containing the device register values

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

Both *CSL_gpioInit()* and *CSL_gpioOpen()* must be called successfully in order before this function.

Post Condition

The registers of the specified GPIO instance will be setup according to value passed.

Modifies

Hardware registers of the GPIO.

Example

```
CSL_GpioHandle    hGpio;
CSL_GpioObj       gpioObj;
CSL_GpioConfig    config = CSL_GPIO_CONFIG_DEFAULTS;
CSL_Status        status;

hGpio = CSL_gpioOpen (&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioHwSetupRaw (hGpio, &config);
```

11.2.6 CSL_gpioGetHwSetup

```
CSL_Status CSL_gpioGetHwSetup ( CSL\_GpioHandle          hGpio,
                                CSL\_GpioHwSetup *      setup
                                )
```

Description

This function gets the current setup of the GPIO. The status is returned through *CSL_GpioHwSetup*. The obtaining of status is the reverse operation of *CSL_gpioHwSetup()* function.

Arguments

<i>hGpio</i>	Handle to the GPIO instance
<i>setup</i>	Pointer to setup structure which contains the setup information of GPIO.

Return Value *CSL_Status*

- *CSL_SOK* - Hardware setup successful.
- *CSL_ESYS_BADHANDLE* - Invalid handle
- *CSL_ESYS_INVPARAMS* - Invalid parameter

Pre Condition

Both *CSL_gpioInit()* and *CSL_gpioOpen()* must be called successfully in order before this function

Post Condition

None

Modifies

Hardware registers of the specified GPIO instance.

Example

```
CSL_GpioHandle    hGpio;
CSL_GpioObj       gpioObj;
CSL_GpioHwSetup   setup;
CSL_Status        status;

hGpio = CSL_gpioOpen (&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioGetHwSetup(hGpio, &setup);
```

11.2.7 CSL_gpioHwControl

```
CSL_Status CSL_gpioHwControl ( CSL\_GpioHandle          hGpio,
                               CSL\_GpioHwControlCmd      cmd,
                               void *                    arg
                               )
```

Description

Control operations for the GPIO. For a particular control operation, the pointer to the corresponding data type needs to be passed as argument to HwControl function Call.

Arguments

hGpio	Handle to the GPIO instance
cmd	The command to this API indicates the action to be taken on GPIO.
arg	Optional argument as per the control command.

Return Value CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_EGPIO_INVPARAM - Invalid GPIO pin number

Pre Condition

Both *CSL_gpioInit()* and *CSL_gpioOpen()* must be called successfully in order before this function

Post Condition

GPIO registers are configured according to the command passed

Modifies

The hardware registers of GPIO.

Example

```

CSL_GpioHandle      hGpio;
CSL_GpioObj         gpioObj;
CSL_GpioHwControlCmd cmd = CSL_GPIO_CMD_BANK_INT_ENABLE;
CSL_Status          status;

hGpio = CSL_gpioOpen (&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioHwControl(hGpio, cmd, NULL);

```

11.2.8 CSL_gpioGetHwStatus

```

CSL_Status CSL_gpioGetHwStatus ( CSL\_GpioHandle          hGpio,
                                CSL\_GpioHwStatusQuery query,
                                void *          response
                                )

```

Description

This function is used to read the current device configuration, status flags and the value present associated registers. For details about the various status queries supported and the associated data structure to record the response, refer to *CSL_GpioHwStatusQuery*..

Arguments

hGpio	Handle to the GPIO instance
query	The query to this API of GPIO which indicates the status to be returned.
response	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Hardware status call is successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query

Pre Condition

Both *CSL_gpioInit()* and *CSL_gpioOpen()* must be called successfully in order before this function

Post Condition

None

Modifies

Third parameter, response value

Example

```

CSL_GpioHandle      hGpio;
CSL_GpioObj         gpioObj;
CSL_GpioHwStatusQuery query= CSL_GPIO_QUERY_BINTEN_STAT;
void                response;
CSL_Status           status;

hGpio = CSL_gpioOpen (&gpioObj, CSL_GPIO, NULL, &status);

status = CSL_gpioGetHwStatus(hGpio, query, &response);

```

11.2.9 CSL_gpioGetBaseAddress

```

CSL_Status CSL_gpioGetBaseAddress ( CSL_InstNum      gpioNum,
                                   CSL\_GpioParam *  pGpioParam,
                                   CSL\_GpioBaseAddress * pBaseAddress
                                   )

```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the *CSL_gpioOpen()* function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

<code>gpioNum</code>	Specifies the instance of GPIO to be opened.
<code>pGpioParam</code>	Module specific parameters.
<code>pBaseAddress</code>	Pointer to baseaddress structure containing base address details.

Return Value `CSL_Status`

- `CSL_SOK` Function call is successful
- `CSL_ESYS_FAIL` The instance number is invalid.
- `CSL_ESYS_INVPARAMS` Invalid Parameter

Pre Condition

None

Post Condition

Base Address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```

CSL_Status          status;
CSL_GpioBaseAddress baseAddress;

status = CSL_gpioGetBaseAddress(CSL_GPIO, NULL, &baseAddress);

```

11.3 Data Structures

This section lists the data structures available in the GPIO module.

11.3.1 CSL_GpioObj

Detailed Description

This object contains the reference to the instance of GPIO opened using the *CSL_gpioOpen()*. The pointer to this is passed to all GPIO CSL APIs. This structure has the fields required to configure GPIO. It should be initialized as per requirements of and passed on to the setup function

Field Documentation

CSL_InstNum CSL_GpioObj::gpioNum

This is the instance of GPIO being referred to by this object

CSL_GpioRegsOvly CSL_GpioObj::regs

Pointer to the register overlay structure of the GPIO

UInt8 CSL_GpioObj::numPins

This is the maximum number of pins supported by this instance of GPIO

11.3.2 CSL_GpioConfig

Detailed Description

Config structure of GPIO. This is used to configure GPIO using *CSL_HwSetupRaw()* function. This is a structure of register values, rather than a structure of register field values like *CSL_GpioHwSetup*.

Field Documentation

volatile UInt32 CSL_GpioConfig::BINTEN

GPIO Interrupt Per-Bank Enable Register

volatile UInt32 CSL_GpioConfig::CLR_DATA

GPIO Clear Data Register

volatile UInt32 CSL_GpioConfig::CLR_FAL_TRIG

GPIO Clear Falling Edge Interrupt Register

volatile UInt32 CSL_GpioConfig::CLR_RIS_TRIG

GPIO Clear Rising Edge Interrupt Register

volatile UInt32 CSL_GpioConfig::DIR

GPIO Direction Register

volatile UInt32 CSL_GpioConfig::OUT_DATA

GPIO Output Data Register

volatile UInt32 CSL_GpioConfig::SET_DATA

GPIO Set Data Register

volatile UInt32 CSL_GpioConfig::SET_FAL_TRIG
GPIO Set Falling Edge Interrupt Register

volatile UInt32 CSL_GpioConfig::SET_RIS_TRIG
GPIO Set Rising Edge Interrupt Register

11.3.3 CSL_GpioContext

Detailed Description

GPIO specific context information. Present implementation doesn't have any Context information.

Field Documentation

UInt16 CSL_GpioContext::contextInfo

Context information of GPIO CSL passed as an argument to CSL_gpioInit(). Present implementation of GPIO CSL doesn't have any context information; hence assigned NULL. The declaration is just a placeholder for future implementation.

11.3.4 CSL_GpioParam

Detailed Description

GPIO specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_GpioParam::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

11.3.5 CSL_GpioHwSetup

Detailed Description

Input parameters for setting up GPIO during startup. This is just a placeholder as GPIO is a simple module, which doesn't require any setup

Field Documentation

void* CSL_GpioHwSetup::extendSetup

The extendSetup is just a placeholder for future implementation.

11.3.6 CSL_GpioBaseAddress

Detailed Description

Base-address of the Configuration registers of GPIO.

Field Documentation

CSL_GpioRegsOvly CSL_GpioBaseAddress::regs

Base address of the configuration registers of the peripheral

11.3.7 CSL_GpioPinConfig

Detailed Description

Input parameters for configuring a GPIO pin. This is used to configure the direction and edge detection.

Field Documentation

[CSL_GpioDirection](#) **CSL_GpioPinConfig::direction**
Direction for GPIO pin

CSL_GpioPinNum **CSL_GpioPinConfig::pinNum**
GPIO Pin Number

CSL_GpioTriggerType **CSL_GpioPinConfig::trigger**
GPIO pin edge detection

11.3.8 CSL_GpioPinData

Detailed Description

This is used for getting a specific pin status and to set the output value of a pin.

Field Documentation

CSL_GpioPinNum **CSL_GpioPinData::pinNum**
Pin number for GPIO bank

Int16 **CSL_GpioPinData::pinVal**
Pin value of the specified pin number

11.4 Enumerations

11.4.1 CSL_GpioDirection

enum CSL_GpioDirection

Enumeration for configuring GPIO pin direction.

Enumeration values:

CSL_GPIO_DIR_OUTPUT

Output pin

CSL_GPIO_DIR_INPUT

Input pin

11.4.2 CSL_GpioTriggerType

enum CSL_GpioTriggerType

Enumeration for configuring GPIO pin edge detection.

Enumeration values:

CSL_GPIO_TRIG_CLEAR_EDGE

No edge detection

CSL_GPIO_TRIG_RISING_EDGE

Rising edge detection

CSL_GPIO_TRIG_FALLING_EDGE

Falling edge detection

CSL_GPIO_TRIG_DUAL_EDGE

Dual edge detection

11.4.3 CSL_GpioHwControlCmd

enum CSL_GpioHwControlCmd

Enumeration for control commands passed to *CSL_gpioHwControl()*.

This is the set of commands that are passed to the *CSL_gpioHwControl()* with an optional argument type-casted to *void**. The arguments to be passed with each enumeration (if any) are specified next to the enumeration.

Enumeration values:

CSL_GPIO_CMD_BANK_INT_ENABLE

Enables interrupt on bank.

Parameters:

(None)

CSL_GPIO_CMD_BANK_INT_DISABLE

Disables interrupt on bank.

Parameters:

(None)

CSL_GPIO_CMD_CONFIG_BIT

Configures GPIO pin direction and edge detection properties.

Parameters:

(*CSL_GpioPinConfig*)

CSL_GPIO_CMD_SET_BIT

Changes output state of GPIO pin to logic-1.

Parameters:

(*CSL_GpioPinNum*)

CSL_GPIO_CMD_CLEAR_BIT

Changes output state of GPIO pin to logic-0.

Parameters:

(*CSL_GpioPinNum*)

CSL_GPIO_CMD_GET_INPUTBIT

Gets the state of input pins on bank The "data"

	field acts as output parameter reporting the input state of the GPIO pins on the bank. Parameters: (CSL_BitMask16*)
<i>CSL_GPIO_CMD_GET_OUTDRVSTATE</i>	Gets the state of output pins on bank. The "data" field acts as output parameter reporting the output drive state of the GPIO pins on the bank. Parameters: (CSL_BitMask16*)
<i>CSL_GPIO_CMD_GET_BIT</i>	Gets the state of input pin on bank. Parameters: (CSL_GpioPinData*)
<i>CSL_GPIO_CMD_SET_OUT_BIT</i>	Sets the output state of GPIO pin to logic 1 or logic 0. Parameters: (CSL_GpioPinData*)

11.4.4 CSL_GpioHwStatusQuery

enum CSL_GpioHwStatusQuery

Enumeration for queries passed to *CSL_GpioGetHwStatus()*.

This is used to get the status of different operations. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

<i>CSL_GPIO_QUERY_BINTEN_STAT</i>	Queries GPIO bank interrupt enable status. Parameters: (CSL_BitMask16*)
-----------------------------------	--

11.5 Macros

#define CSL_EGPIO_INVPARAM CSL_EGPIO_FIRST
Value for invalid argument

#define CSL_GPIO_CONFIG_DEFAULTS
Value:

```
{
    \
    CSL_GPIO_BINTEN_RESETVAL ,      \
    CSL_GPIO_DIR_RESETVAL ,        \
    CSL_GPIO_OUT_DATA_RESETVAL ,   \
    CSL_GPIO_SET_DATA_RESETVAL ,   \
    CSL_GPIO_CLR_DATA_RESETVAL ,   \
    CSL_GPIO_SET_RIS_TRIG_RESETVAL , \
    CSL_GPIO_CLR_RIS_TRIG_RESETVAL , \
    CSL_GPIO_SET_FAL_TRIG_RESETVAL , \
    CSL_GPIO_CLR_FAL_TRIG_RESETVAL , \
}
```

Default values for GPIO Config structure.

Chapter 12

I2C MODULE

Topics

<u>12. 1 Overview</u>
<u>12. 2 Functions</u>
<u>12. 3 Data Structures</u>
<u>12. 4 Enumerations</u>
<u>12. 5 Macros</u>

12.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within I2C module. The I2C ports allows the DSP to easily control peripheral devices and communicate with a host processor.

The inter-integrated circuit (I2C) module provides an interface between a DSP and other devices of Inter-IC bus (I2C-bus).

12.2 Functions

This section lists the functions available in the I2C module.

12.2.1 CSL_i2cInit

CSL_Status CSL_i2cInit ([CSL_I2cContext](#) * *pContext*)

Description

This is the initialization function for the I2C CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

<i>pContext</i>	Context information for the instance. As I2C doesn't have any context based information user is expected to pass NULL.
-----------------	--

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

The CSL for I2C is initialized

Modifies

None

Example

```
CSL_i2cInit(NULL);
```

12.2.2 CSL_i2cOpen

[CSL_I2cHandle](#) CSL_i2cOpen ([CSL_I2cObj](#) * *pl2cObj*,
CSL_InstNum *i2cNum*,
[CSL_I2cParam](#) * *pl2cParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of I2C device. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input argument for rest of the I2C CSL APIs.

Arguments

<code>pI2cObj</code>	Pointer to the I2C instance object
<code>i2cNum</code>	Instance of the I2C to be opened.
<code>pI2cParam</code>	Pointer to module specific parameters
<code>pStatus</code>	Pointer for returning status of the function call

Return Value `CSL_I2cHandle`

- Valid I2C instance handle will be returned if status value is equal to `CSL_SOK`.

Pre Condition

`CSL_i2cInit()` must be called successfully.

Post Condition

- The status variable
- I2C object structure

- `CSL_SOK` Valid I2C handle is returned.
- `CSL_ESYS_FAIL` The I2C instance is invalid.
- `CSL_ESYS_INVPARAM` Invalid Parameters.

Modifies

Fills up the Obj structure

Example

```
CSL_Status      status;
CSL_I2cObj      i2cObj;
CSL_I2cHandle   hI2c;

hI2c = CSL_i2cOpen(&i2cObj, CSL_I2C, NULL, &status);
```

12.2.3 CSL_i2cClose

CSL_Status `CSL_i2cClose` ([CSL_I2cHandle](#) `hI2c`)

Description

This function closes the specified instance of I2C.

Arguments

<code>hI2c</code>	Handle to the I2C module
-------------------	--------------------------

Return Value `CSL_Status`

- `CSL_SOK` - Close Successful
- `CSL_ESYS_BADHANDLE` - Invalid handle

Pre Condition

Both `CSL_i2cInit()` and `CSL_i2cOpen()` must be called successfully in order before calling `CSL_i2cClose()`.

Post Condition

The I2C CSL APIs can not be called until the I2C CSL is reopened again using `CSL_i2cOpen()`.

Modifies

Obj structure values

Example

```
CSL_I2cHandle  hI2c;
CSL_I2cObj     i2cObj;
CSL_Status     status;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

status = CSL_i2cClose(hI2c);
```

12.2.4 CSL_i2cHwSetup

```
CSL_Status CSL_i2cHwSetup ( CSL\_I2cHandle          hI2c,
                           CSL\_I2cHwSetup *          setup
                           )
```

Description

This function initializes the device registers with the appropriate values provided through the HwSetup Data structure. After the Setup is completed, the device is ready for operation. For information passed through the HwSetup Data structure, refer *CSL_i2cHwSetup*.

Arguments

<code>hI2c</code>	Handle to the I2C
<code>setup</code>	Pointer to the setup structure which contains the setup information of I2C

Return Value `CSL_Status`

- `CSL_SOK` - Close successful
- `CSL_ESYS_BADHANDLE` - Invalid handle
- `CSL_ESYS_INVPARAMS` - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function. The user has to allocate space for and fill in the main setup structure appropriately before calling this function.

Post Condition

I2C registers are configured according to the hardware setup parameters.

Modifies

I2C registers will be setup according to value passed.

Example

```
CSL_I2cHandle hI2c;
CSL_I2cObj     i2cObj;
CSL_Status     status;
```

```

CSL_I2cHwSetup hwSetup;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

hwSetup.mode = 1;
hwSetup.dir = 0;
hwSetup.ownaddr = 0x10;
...

CSL_i2cHwSetup(hI2c, &hwSetup);

```

12.2.5 CSL_i2cGetHwSetup

```

CSL_Status CSL_i2cGetHwSetup      ( CSL\_I2cHandle          hI2c,
                                     CSL\_I2cHwSetup *      setup
                                     )

```

Description

This function gets the current setup of the I2C. The status is returned through *CSL_I2cHwSetup*. The obtaining of status is the reverse operation of *CSL_i2cHwSetup()* function.

Arguments

<i>hI2c</i>	Handle to the I2C
<i>setup</i>	Pointer to the hardware setup structure

Return Value CSL_Status

- CSL_SOK - Retrieving the hardware setup parameters is successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

hwSetup variable

Example

```

CSL_I2cHandle  hI2c;
CSL_I2cObj    i2cObj;
CSL_I2cHwSetup hwSetup;
CSL_Status     status;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

status = CSL_i2cGetHwSetup(hI2c, &hwSetup);

```

12.2.6 CSL_i2cHwControl

```
CSL_Status CSL_i2cHwControl      ( CSL\_I2cHandle           hI2c,
                                   CSL\_I2cHwControlCmd    cmd,
                                   void *          arg
                                   )
```

Description

Control operations for the I2C. For a particular control operation, the pointer to the corresponding data type need to be passed as argument to HwControl function call. All the arguments (structure element included) passed to the HwControl function are inputs. For the list of commands supported and argument type that can be *void** casted and passed with a particular command refer to *CSL_I2cHwControlCmd*.

Arguments

<code>hI2c</code>	Handle to the I2C instance
<code>cmd</code>	The command to this API indicates the action to be taken on I2C.
<code>arg</code>	An optional argument

Return Value CSL_Status

- CSL_SOK - Command successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition

I2C registers are configured according to the command passed

Modifies

The hardware registers of I2C.

Example

```
CSL_I2cHandle      hI2c;
CSL_I2cObj         i2cObj;
CSL_I2cHwControlCmd cmd = CSL_I2C_CMD_ENABLE;
CSL_Status         status;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

status = CSL_i2cHwControl(hI2c, cmd, NULL);
```

12.2.7 CSL_i2cRead

```
CSL_Status CSL_i2cRead          ( CSL\_I2cHandle          hI2c,
                                void *                    buf
                                )
```

Description

This function reads I2C data.

Arguments

<i>hI2c</i>	Handle to I2C instance
<i>buf</i>	Buffer to store the data read

Return Value CSL_Status

- CSL_SOK – Read operation Successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition

None

Modifies

Output parameter *buf* being passed.

Example

```
Uint8          inData;
CSL_Status     status;
CSL_I2cHandle  hI2c;
CSL_I2cObj     i2cObj;
CSL_I2cHwSetup setup;

setup.mode = 0;
setup.dir = 1;
...

/* Init, Open, HwSetup successfully done in that order */
CSL_i2cInit (NULL);

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

CSL_i2cHwSetup (hI2c, &setup);

status = CSL_i2cRead(hI2c, &inData);
```

12.2.8 CSL_i2cWrite

```
CSL_Status CSL_i2cWrite ( CSL\_I2cHandle      hi2c,
                          void *                buf
                          )
```

Description

This function writes the specified data into I2C data register.

Arguments

<i>hi2c</i>	Handle to I2C instance
<i>buf</i>	Data to be written

Return Value CSL_Status

- CSL_SOK – Write success (does not verify written data)
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition

Data is written to I2C data register

Modifies

I2C register

Example

```
Uint8      outData;
CSL_Status status;
CSL_I2cHandle hi2c;
CSL_I2cObj  i2cObj;
CSL_I2cHwSetup setup;

setup.mode = 0;
setup.dir = 1;
...

/* Init, Open, HwSetup successfully done in that order */
CSL_i2cInit ();

hi2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

CSL_i2cHwSetup (hi2c, &setup);

outData = 0x65;

status = CSL_i2cWrite(hi2c, &outData);
```

12.2.9 CSL_i2cHwSetupRaw

```
CSL_Status CSL_i2cHwSetupRaw ( CSL\_I2cHandle          hI2c,
                               CSL\_I2cConfig *         config
                               )
```

Description

This function initializes the device registers with the register-values provided through the Config Data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values.

Arguments

<code>hI2c</code>	Handle to the I2C
<code>config</code>	Pointer to config structure

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in order before calling this function.

Post Condition

The registers of the specified I2C instance will be setup according to value passed.

Modifies

Hardware registers of the specified I2C instance.

Example

```
CSL_I2cHandle    hI2c;
CSL_I2cObj       i2cObj;
CSL_I2cConfig    config = CSL_I2C_CONFIG_DEFAULTS;
CSL_Status       status;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

status = CSL_i2cHwSetupRaw (hI2c, &config);
```

12.2.10 CSL_i2cGetHwStatus

```
CSL_Status CSL_i2cGetHwStatus ( CSL\_I2cHandle          hI2c,
                                CSL\_I2cHwStatusQuery      query,
                                void *                    response
                                )
```

Description

This function is used to read the current device configuration, status flags and the value present associated registers. For various status queries supported and the associated data structure to record the response refer *CSL_I2cHwStatusQuery*. User should allocate memory for the said data type and pass its pointer as an unadorned void* argument to the status query call. .

Arguments

hI2c	Handle to the I2C instance
query	The query to this API of I2C which indicates the status to be returned.
response	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Hardware status call is successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both *CSL_i2cInit()* and *CSL_i2cOpen()* must be called successfully in that order before *CSL_i2cGetHwStatus()* can be called.

Post Condition

None

Modifies

Third parameter, response value

Example

```

CSL_I2cHandle      hI2c;
CSL_I2cObj         i2cObj;
CSL_I2cHwStatusQuery query = CSL_I2C_QUERY_TX_RDY;
void               response;
CSL_Status          status;

hI2c = CSL_i2cOpen (&i2cObj, CSL_I2C, NULL, &status);

status = CSL_i2cGetHwStatus(hI2c, query, &response);

```

12.2.11 CSL_i2cGetBaseAddress

```

CSL_Status CSL_i2cGetBaseAddress ( CSL_InstNum      i2cNum,
                                   CSL\_I2cParam *  pI2cParam,
                                   CSL\_I2cBaseAddress * pBaseAddress
                                   )

```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the *CSL_i2cOpen()*

function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

<code>i2cNum</code>	Specifies the instance of I2C to be opened.
<code>pI2cParam</code>	Module specific parameters.
<code>pBaseAddress</code>	Pointer to baseaddress structure containing base address details.

Return Value `CSL_Status`

- `CSL_SOK` - Successful on getting the base address of I2C
- `CSL_ESYS_FAIL` - The instance number is invalid.

Pre Condition

None

Post Condition

Base address structure is populated

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status      status;
CSL_I2cBaseAddress  baseAddress;

status = CSL_i2cGetBaseAddress(CSL_I2C, NULL, &baseAddress);
```

12.3 Data Structures

This section lists the data structures available in the I2C module.

12.3.1 CSL_I2cObj

Detailed Description

This object contains the reference to the instance of I2C opened using the *CSL_i2cOpen()*. The pointer to this is passed to all I2C CSL APIs.

Field Documentation

CSL_InstNum CSL_I2cObj::perNum

This is the instance of I2C being referred to by this object

CSL_I2cRegsOvly CSL_I2cObj::regs

The register overlay structure of I2C.

12.3.2 CSL_I2cConfig

Detailed Description

I2C Configuration Structure is used to configure I2C using *CSL_HwSetupRaw()* function. This is a structure of register values, rather than a structure of register field values like *CSL_I2cHwSetup*.

Field Documentation

volatile Uint32 CSL_I2cConfig::ICCLKH

I2C Clock High Register

volatile Uint32 CSL_I2cConfig::ICCLKL

I2C Clock Low Register

volatile Uint32 CSL_I2cConfig::ICCNT

I2C Data Count Register

volatile Uint32 CSL_I2cConfig::ICDXR

I2C Data Transmit Register

volatile Uint32 CSL_I2cConfig::ICEMDR

I2C Extended Mode Register

volatile Uint32 CSL_I2cConfig::ICIMR

I2C Interrupt Mask Register

volatile Uint32 CSL_I2cConfig::ICIVR

I2C Interrupt vector register

volatile Uint32 CSL_I2cConfig::ICMDR

I2C Data Receive Register

volatile Uint32 CSL_I2cConfig::ICOAR

I2C Own Address Register

volatile Uint32 CSL_I2cConfig::ICPSC
I2C Pre-scalar Register

volatile Uint32 CSL_I2cConfig::ICSAR
I2C Slave Address Register

volatile Uint32 CSL_I2cConfig::ICSTR
I2C Status Register

12.3.3 CSL_I2cContext

Detailed Description

I2C specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_I2cContext::contextInfo

Context information of I2C. The declaration is just a placeholder for future implementation.

12.3.4 CSL_I2cParam

Detailed Description

I2C specific parameters. Present implementation of I2C CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_I2cParam::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

12.3.5 CSL_I2cClkSetup

Detailed Description

The clock setup structure has all the fields required to configure the I2C clock.

Field Documentation

Uint32 CSL_I2cClkSetup::clkhighdiv

High time period of the clock

Uint32 CSL_I2cClkSetup::clklowdiv

Low time period of the clock

Uint32 CSL_I2cClkSetup::prescalar

Pre-scalar to the input clock

12.3.6 CSL_I2cHwSetup

Detailed Description

This has all the fields required to configure I2C at Power Up (After a Hardware Reset) or a Soft Reset. This structure is used to setup or obtain existing setup of I2C using *CSL_i2cHwSetup()* and *CSL_i2cGetHwSetup()* functions respectively.

Field Documentation

UInt32 CSL_I2cHwSetup::ackMode

ACK mode while receiver: 0==> ACK Mode, 1==> NACK Mode

UInt32 CSL_I2cHwSetup::addrMode

Addressing Mode :0==> 7-bit Mode, 1==> 10-bit Mode

UInt32 CSL_I2cHwSetup::bcm

I2C Backward Compatibility Mode : 0 ==> Not compatible, 1 ==> Compatible

[CSL_I2cClkSetup](#)* CSL_I2cHwSetup::clksetup

Prescalar, Clock Low and Clock High for Clock Setup

UInt32 CSL_I2cHwSetup::dir

Transmitter Mode or Receiver Mode: 1==> Transmitter Mode, 0 ==> Receiver Mode

UInt32 CSL_I2cHwSetup::freeDataFormat

Free Data Format of I2C: 0 ==>Free data format disable, 1 ==> Free data format enable

UInt32 CSL_I2cHwSetup::inten

Interrupt Enable mask The mask can be for one interrupt or OR of multiple interrupts.

UInt32 CSL_I2cHwSetup::loopBackMode

DLBack mode of I2C (master tx-er only): 0 ==> No loopback, 1 ==> Loopback Mode

UInt32 CSL_I2cHwSetup::mode

Master or Slave Mode: 1==> Master Mode, 0==> Slave Mode

UInt32 CSL_I2cHwSetup::ownaddr

Address of the own device

UInt32 CSL_I2cHwSetup::repeatMode

Repeat Mode of I2C: 0==> No repeat mode 1==> Repeat mode

UInt32 CSL_I2cHwSetup::resetMode

I2C Reset Mode: 0==> Reset, 1==> Out of reset

UInt32 CSL_I2cHwSetup::runMode

Run mode of I2C: 0==> No Free Run, 1==> Free Run mode

UInt32 CSL_I2cHwSetup::sttbyteen

Start Byte Mode: 1 ==> Start Byte Mode, 0 ==> Normal Mode

12.3.7 CSL_I2cBaseAddress

Detailed Description

This structure contains the base address information for I2C peripheral instance.

Field Documentation

CSL_I2cRegsOvly CSL_I2cBaseAddress::regs

Base address of the Configuration registers of I2C.

12.4 Enumerations

This section lists the enumerations available in the I2C module.

12.4.1 CSL_I2cHwStatusQuery

enum CSL_I2cHwStatusQuery

Enumeration for queries passed to *CSL_i2cGetHwStatus()*.

This is used to get the status of different operations or to get the existing setup of I2C.

Enumeration values:

<i>CSL_I2C_QUERY_CLOCK_SETUP</i>	To get current clock setup parameters. Parameters: (<i>CSL_I2cClkSetup *</i>)
<i>CSL_I2C_QUERY_BUS_BUSY</i>	To get the Bus Busy status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_RX_RDY</i>	To get the Receive Ready status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_TX_RDY</i>	To get the Transmit Ready status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_ACS_RDY</i>	To get the Register Ready status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_SCD</i>	To get the Stop Condition Data bit information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_ADO</i>	To get the Address Zero (General Call) detection status. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_RSFULL</i>	To get the Receive overflow status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_XSMT</i>	To get the Transmit underflow status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_AAS</i>	To get the Address as Slave bit information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_AL</i>	To get the Arbitration Lost status information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_RDONE</i>	To get the Reset Done status bit information. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_QUERY_BITCOUNT</i>	To get number of bits of next byte to be received or

	transmitted.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_INTCODE</i>	To get the interrupt code for the interrupt that occurred.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_REV</i>	To get the revision level of the I2C.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_CLASS</i>	To get the class of the peripheral.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_TYPE</i>	To get the type of the peripheral.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_SDIR</i>	To get the slave direction.
	Parameters: (<i>Uint32*</i>)
<i>CSL_I2C_QUERY_NACKSNT</i>	To get the acknowledgement status.
	Parameters: (<i>Uint32*</i>)

12.4.2 CSL_I2cHwControlCmd

enum CSL_I2cHwControlCmd

Enumeration for queries passed to *CSL_i2cHwControl()*.

This is used to select the commands to control the operations existing setup of I2C. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

<i>CSL_I2C_CMD_ENABLE</i>	Command to enable the I2C module.
	Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_RESET</i>	Command to reset the I2C.
	Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_OUTOFRESET</i>	Command to make the I2C out of reset.
	Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_CLEAR_STATUS</i>	Command to clear the status bits. The argument next to the command specifies the status bit to be cleared. The status bit can be: <i>CSL_I2C_CLEAR_AL</i> , <i>CSL_I2C_CLEAR_NACK</i> , <i>CSL_I2C_CLEAR_ARDY</i> , <i>CSL_I2C_CLEAR_RRDY</i> , <i>CSL_I2C_CLEAR_XRDY</i> , <i>CSL_I2C_CLEAR_GC</i> .
	Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_SET_SLAVE_ADDR</i>	Command to set the address of the Slave device.
	Parameters:

	(<i>Uint32 *</i>)
<i>CSL_I2C_CMD_SET_DATA_COUNT</i>	Command to set the Data Count. Parameters: (<i>Uint32 *</i>)
<i>CSL_I2C_CMD_START</i>	Command to set the start condition. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_STOP</i>	Command to set the stop condition. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_DIR_TRANSMIT</i>	Command to set the transmission mode. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_DIR_RECEIVE</i>	Command to set the receiver mode. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_RM_ENABLE</i>	Command to set the Repeat Mode. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_RM_DISABLE</i>	Command to disable the Repeat Mode. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_DLB_ENABLE</i>	Command to set the loop back mode. Parameters: (<i>None</i>)
<i>CSL_I2C_CMD_DLB_DISABLE</i>	Command to set the loop back mode. Parameters: (<i>None</i>)

12.5 Macros

#define CSL_I2C_ACK_DISABLE (1)

For enabling the tx of a NACK to the TX-ER, while in the RECEIVER mode

#define CSL_I2C_ACK_ENABLE (0)

For enabling the tx of a ACK to the TX-ER, while in the RECEIVER mode

#define CSL_I2C_ACS_NOT_READY (0)

For indicating that the Access ready signal is low

#define CSL_I2C_ACS_READY (1)

For indicating that the Access ready signal is high

#define CSL_I2C_ADDRSZ_SEVEN (0)

For setting the 7-bit Addressing Mode for I2C

#define CSL_I2C_ADDRSZ_TEN (1)

For setting the 10-bit Addressing Mode

#define CSL_I2C_ARBITRATION_LOST (1)

For indicating Arbitration Lost signal is set

#define CSL_I2C_BCM_DISABLE (0)

For disabling the Backward Compatibility mode of I2C

#define CSL_I2C_BCM_ENABLE (1)

For enabling the Backward Compatibility mode of I2C

#define CSL_I2C_BUS_BUSY (1)

For indicating that the bus is busy

#define CSL_I2C_BUS_NOT_BUSY (0)

For indicating that the bus is not busy

#define CSL_I2C_CLEAR_AL 0x1

Clear the Arbitration Lost status bit

#define CSL_I2C_CLEAR_ARDY 0x4

Clear the Register access ready status bit

#define CSL_I2C_CLEAR_NACK 0x2

Clear the No acknowledge status bit

#define CSL_I2C_CLEAR_RRDY 0x8

Clear the Receive ready status bit

#define CSL_I2C_CLEAR_SCD 0x20

Clear the Stop Condition Detect status bit

#define CSL_I2C_CLEAR_XRDY 0x10

Clear the Transmit ready status bit

#define CSL_I2C_CONFIG_DEFAULTS
Value:

```
{
    \
    CSL_I2C_ICOAR_RESETVAL,    \
    CSL_I2C_ICIMR_RESETVAL,    \
    CSL_I2C_ICSTR_RESETVAL,    \
    CSL_I2C_ICCLKL_RESETVAL,   \
    CSL_I2C_ICCLKH_RESETVAL,   \
    CSL_I2C_ICCNT_RESETVAL,    \
    CSL_I2C_ICSAR_RESETVAL,    \
    CSL_I2C_ICDXR_RESETVAL,    \
    CSL_I2C_ICMDR_RESETVAL,    \
    CSL_I2C_ICIVR_RESETVAL,    \
    CSL_I2C_ICEMDR_RESETVAL,   \
    CSL_I2C_ICPSC_RESETVAL,    \
}
```

Default Values for Config structure

#define CSL_I2C_DIR_RECEIVE (0)

For setting the RECEIVER Mode for I2C

#define CSL_I2C_DIR_TRANSMIT (1)

For setting the TRANSMITTER Mode for I2C

#define CSL_I2C_DLB_DISABLE (0)

For disabling DLB mode of I2C (applicable only in case of MASTER TX-ER)

#define CSL_I2C_DLB_ENABLE (1)

For enabling DLB mode of I2C (applicable only in case of MASTER TX-ER)

#define CSL_I2C_FDF_DISABLE (0)

For disabling the Free Data Format of I2C

#define CSL_I2C_FDF_ENABLE (1)

For enabling the Free Data Format of I2C

#define CSL_I2C_FREE_MODE_DISABLE (0)

For disabling the free run mode of the I2C

#define CSL_I2C_FREE_MODE_ENABLE (1)

For enabling the free run mode of the I2C

#define CSL_I2C_IRS_DISABLE (1)

For taking the I2C out of Reset

#define CSL_I2C_IRS_ENABLE (0)

For putting the I2C in Reset

#define CSL_I2C_MODE_MASTER (1)

For setting the MASTER Mode for I2C

#define CSL_I2C_MODE_SLAVE (0)

For setting the SLAVE Mode for I2C

#define CSL_I2C_RECEIVE_OVERFLOW (1)

For indicating Receive overflow signal is set

#define CSL_I2C_REPEAT_MODE_DISABLE (0)

For disabling the Repeat Mode of the I2C

#define CSL_I2C_REPEAT_MODE_ENABLE (1)

For enabling the Repeat Mode of the I2C

#define CSL_I2C_RESET_DONE (1)

For indicating the completion of Reset

#define CSL_I2C_RESET_NOT_DONE (0)

For indicating the non-completion of Reset

#define CSL_I2C_RX_NOT_READY (0)

For indicating that the Receive ready signal is low

#define CSL_I2C_RX_READY (1)

For indicating that the Receive ready signal is high

#define CSL_I2C_SINGLE_BYTE_DATA (1)

For indicating Single Byte Data signal is set

#define CSL_I2C_STB_DISABLE (0)

For Disabling the Start Byte Mode for I2C(Normal Mode)

#define CSL_I2C_STB_ENABLE (1)

For Enabling the Start Byte Mode for I2C

#define CSL_I2C_TRANSMIT_UNDERFLOW (1)

For indicating Transmit underflow signal is set

#define CSL_I2C_TX_NOT_READY (0)

For indicating that the Transmit ready signal is low

#define CSL_I2C_TX_READY (1)

For indicating that the Transmit ready signal is high

Chapter 13

IDMA MODULE

Topics

<u>13. 1 Overview</u>

<u>13. 2 Functions</u>
--

<u>13. 3 Data Structures</u>
--

<u>13. 4 Enumerations</u>

13.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within IDMA module.

The internal DMA (IDMA), is a DMA local to the megamodule- that is, it provides data move services only within the megamodule (L1P, L1D, L2, and CFG).

There are two IDMA channels (0 and 1).

- Channel 0 allows data to be transferred between the peripheral configuration space (CFG) and any local memories (L1P, L1D, and L2).
- Channel 1 is used to transfer data between the local memories (L1P, L1D, and L2).

The IDMA data transfers occur in the background of CPU operation. That is, once a channel transfer is programmed, it happens concurrent with other CPU activity, and without additional CPU intervention.

13.2 Functions

This section lists the functions available in the IDMA module.

13.2.1 IDMA1_init

```
Int IDMA1_init          ( IDMA\_priSet          priority,
                        IDMA\_intEn          interr
                        )
```

Description

This function obtains a priority and an interrupt flag and remembers them so that all future transfers on channel 1 will use these priorities. The priority is contained in the argument "priority" and interrupt flag in "interr". This function performs IDMA Channel 1 initialization by setting the priority level and the enabling/disabling the interrupt event generation for the channel.

Arguments

<code>priority</code>	Priority 0-7 of handle
<code>interr</code>	Interrupt event generated on/off

Return Value Int

Priority of IDMA relative to CPU and whether interrupt is desired or not. These values stored in the 32-bit field 'cnt' of the local IDMA1 handle structure

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Int          cnt1;

// Initialize IDMA Channel 1
// Set Chan 1 to have Priority 7 and Interrupt Event Gen On
...

cnt1 = IDMA1_init (IDMA_PRI_7, IDMA_INT_EN);
```

13.2.2 IDMA1_copy

```
Int IDMA1_copy          (   Uint32 *          src,
                          Uint32 *          dst,
                          Uint32          byteCnt
                          )
```

Description

IDMA1_copy() transfers "byteCnt" bytes from a source "src" to a destination "dst". It is assumed that both the source and destination addresses are in internal memory. Transfers from addresses that are not in the internal memory will raise an exception. No checking is performed by this function to check the correctness of any of the passed in arguments. Used to transfer "byteCnt" bytes from source "src" to destination "dst".

Arguments

src	Pointer to the source address
dst	Pointer to the destination address
byteCnt	Number of bytes to be transferred

Return Value Int

Always returns 0

Pre Condition

The function IDMA_init () must be called successfully before calling to this function. None

Post Condition

Call IDMA1_wait () function

Modifies

The hardware registers of IDMA.

Example

```
#pragma DATA_SECTION(src, "ISRAM");
#pragma DATA_ALIGN(src, 8);
#pragma DATA_SECTION(dst1, "ISRAM1");
#pragma DATA_ALIGN(dst1, 8);

Uint32      src[20] = {
    0xDEADBEEF,
    0xFADEBABA,
    0x5AA51C3A,
    0xD4536BA3,
    0x5E69BA23,
    0x4884A01F,
    0x9265ACDA,
    0xFFFF0123,
    0xBEADDABE,
    0x234A76B2,
    0x9675ABCD,
    0xABCDEF12,
    0xEEEECDEA,
    0x01234567,
    0x00000000,
    0xFEEFDADE,
    0x0A1B2C3D,
    0x4E5F6B7C,
    0x5AA5ECCE,
    0xFABEFACE };

Uint32      dst1[20];
...
```

```
void main (void)
{
...
// Copy src to dst1 - 80 bytes - 20 words
IDMA1_copy(src, dst1, 80);
... }
```

13.2.3 IDMA1_fill

```
Int IDMA1_fill (    Uint32 *      dst,
                   Uint32      byteCnt,
                   Uint32      fill_value
                  )
```

Description

IDMA1_fill() takes a fill value in "fill_value" and fills "byteCnt" bytes of the "fill_value" to destination "dst".

Arguments

dst	Pointer to the destination address
byteCnt	Number of bytes to be transferred
fill_value	Data to be filled

Return Value Int

Always returns 0

Pre Condition

The function IDMA_init () must be called successfully before calling to this function.

Post Condition

Call IDMA1_wait () function

Modifies

The hardware registers of IDMA

Example

```
#pragma DATA_SECTION(dst1, "ISRAM1");
#pragma DATA_ALIGN(dst1, 8);

Uint32      dst1[20];

void main (void)
{
...

IDMA1_fill(dst1, 80, 0xAAAABABA);
...
}
```

13.2.4 IDMA1_getStatus

Uint32 IDMA1_getStatus (void)

Description

IDMA1_getStatus() gets the active and pending status of IDMA Channel 1 and returns ACTV in the least significant bit and PEND in the 2nd least significant bit

Arguments

None

Return Value Uint32

IDMA channel 1 status

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      stat;
    ...
    stat = IDMA1_getStatus();

```

13.2.5 IDMA1_wait

void IDMA1_wait (void)

Description

IDMA1_wait() waits until all previous transfers for IDMA Channel 1 have been completed by making sure that both active and pending bits are zero. These are the two least significant bits of the status register for the channel.

Arguments

None

Return Value

None

Pre Condition

Functions IDMA1_init () and IDMA1_copy () or IDMA1_fill () must be called successfully in that order before this API can be invoked.

Post Condition

Completion of previous transfers

Modifies

IDMA channel 1 registers

Example

```
#pragma DATA_SECTION(dst, "ISRAM1");
#pragma DATA_ALIGN(dst, 8);

Uint32      dst[20];

void main (void)
{
    Uint32      stat;

    ...
    IDMA1_fill (dst, 80, 0xAAAAAAAA);
    stat = IDMA1_getStatus();
    IDMA1_wait();
}
```

13.2.6 IDMA1_setPriority

Int IDMA1_setPriority ([IDMA_priSet](#) *priority*)

Description

IDMA1_setPriority() sets a "3-bit" priority field which has a valid range of 0-7 for priorities 0-7. It returns a "32-bit" count register field back to the user. This 32-bit register field will be used in IDMA1_copy() and IDMA1_fill() to program the Priority and Interrupt options for IDMA Chan 1. Sets the priority level for IDMA channel 1 transfers.

Arguments

priority Priority 0-7 of handle

Return Value Int

Priority of IDMA relative to CPU. This value stored in the 32-bit field 'cnt' of the local IDMA1 handle structure

Pre Condition

None

Post Condition

None

Modifies

IDMA channel 1 registers

Example

```
Uint32      tempCnt;
...

// Set and test Priority level for IDMA1
tempCnt = IDMA1_setPriority(IDMA_PRI_2);
```

13.2.7 IDMA1_setInt

Int IDMA1_setInt ([IDMA_intEn](#) *interr*)

Description

IDMA1_setInt() sets the interrupt enable field which is used to enable/disable interrupts for IDMA Channel 1. It returns a "32-bit" count register field back to the user. This 32-bit register field will be used in IDMA1_copy() and IDMA1_fill() to program the Priority and Interrupt options for IDMA Channel 1

Arguments

interr	Interrupt event generated on/off
--------	----------------------------------

Return Value

Interrupt is enabled or not. This value stored in the 32-bit field 'cnt' of the local IDMA1 handle structure

Pre Condition

None

Post Condition

None

Modifies

IDMA channel 1 registers

Example

```

uint32_t tempCnt;
...

// Set and test Interrupt event gen for IDMA1
tempCnt = IDMA1_setInt(IDMA_INT_DIS);

```

13.2.8 IDMA0_init

```
Int IDMA0_init ( IDMA_intEn interr )
```

Description

This function obtains a interrupt enable setting and remembers them so that all the future transfers on Channel 0 generate interrupts or not. Initializes the Interrupt Event Generation for IDMA Channel 0.

Arguments

interr	Interrupt event generated on/off
--------	----------------------------------

Return Value Int

Interrupt is enabled or not. This value stored in the 32-bit 'cnt' field of the local IDMA0 configuration structure

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32      cnt0;
    ...

    // Initialize IDMA Channel 0
    // Set Chan 0 to have Interrupt Event Gen On
    cnt0 = IDMA0_init(IDMA_INT_EN);

```

13.2.9 IDMA0_config

```
void IDMA0_config ( IDMA0\_Config * config )
```

Description

IDMA0_config() - Configures IDMA Channel 0 to perform a transfer between Internal Memory and Configuration Space based on the data in the *config structure. "mask" provides a 1-hot encoding for the 32-word transfer that determines which of the 32-words are to be transferred. In the *config structure "src" provides the source location of the transfer and "dst" provides the destination location of the transfer and both must be word aligned. While "cnt" provides the number of 32-word transfers to perform and must not be greater than 15. Initializes the configuration for IDMA Channel 0 including 1-hot encoding mask, source location, destination location and count. This is done using the structure IDMA0_Config.

Arguments

config Pointer to the Configuration structure

Return Value

None

Pre Condition

The function IDMA0_init () must be called successfully before invoking this API.

Post Condition

Invoke IDMA0_wait () after calling this API

Modifies

The hardware registers of IDMA.

Example

```

IDMA0_Config  config;
...

IDMA0_config(&config);
IDMA0_wait();

```

13.2.10 IDMA0_configArgs

```

void IDMA0_configArgs (
    Uint32      mask,
    Uint32 *    src,
    Uint32 *    dst,
    Uint32      count
)

```

Description

IDMA0_configArgs() - Configures IMDA Channel 0 to perform a transfer between Internal Memory and Configuration Space based on the inputs to the function. "mask" provides a 1-hot encoding for the 32-word transfer that determines which of the 32-words are to be transferred. "src" provides the source location of the transfer and "dst" provides the destination location of the transfer and both must be word aligned. While "cnt" provides the number of 32-word transfers to perform and must not be greater than 15. Initializes the configuration for IDMA Channel 0 including 1-hot encoding mask, source location, destination location and count.

Arguments

mask	Encoding value for the 32-word transfer
src	Pointer to the source location of the transfer
dst	Pointer to the destination location of the transfer
cnt	Number of 32-word transfers

Return Value

None

Pre Condition

The function IDMA0_init () must be called successfully before invoking this API.

Post Condition

Invoke IDMA0_wait () after calling this API

Modifies

The hardware registers of IDMA.

Example

```

    Uint32  *src,*dst;
    Uint32  mask;
    ...

    IDMA0_configArgs(mask, src, dst, 1);
    IDMA0_wait();

```

13.2.11 IDMA0_getStatus

Uint32 IDMA0_getStatus (void)

Description

IDMA0_getStatus() gets the active and pending status of IMDA Channel 0 and returns ACTV in the least significant bit and PEND in the 2nd least significant bit

Arguments

None

Return Value

Uint32
IDMA channel 0 status

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32          stat;
    ...

    stat = IDMA0_getStatus();

```

13.2.12 IDMA0_wait

void IDMA0_wait (**void**)

Description

IDMA0_wait() waits until all previous transfers for IDMA Channel 0 have been completed by making sure that both active and pend bits are zero. These are the two least significant bits of the status register for the channel.

Arguments

None

Return Value

None

Pre Condition

Functions IDMA0_init () and IDMA0_config () or IDMA0_configArgs () must be called successfully in that order before this API can be invoked.

Post Condition

Completion of previous transfer

Modifies

IDMA channel 0 registers

Example

```

    Uint32          stat;
    ...

    stat = IDMA0_getStatus();
    IDMA0_wait();

```

13.2.13 IDMA0_setInt

Int IDMA0_setInt ([IDMA_intEn](#) *interr*)

Description

IDMA0_setInt() sets a the interrupt enable field which is used to enable/disable interrupts for IDMA Channel 0. It returns a "32-bit" count register field back to the user. This 32-bit register field will be used in IDMA0_config() and IDMA0_configArgs() to program the Interrupt option for IDMA Channel 0

Arguments

`interr` Interrupt event generated on/off

Return Value `Int`

Interrupt is enabled or not. This value stored in the 32-bit 'cnt' field of the local IDMA0 configuration structure

Pre Condition

None

Post Condition

None

Modifies

IDMA channel 0 registers

Example

```
Uint32                      tempCnt;  
...  
  
// Set and test Interrupt event gen for IDMA0  
tempCnt = IDMA0_setInt(IDMA_INT_DIS);
```

13.3 Data Structures

This section lists the data structures available in the IDMA module.

13.3.1 idma1_handle

Detailed Description

IDMA1_handle IDMA Channel 1 handle - Contains Status, Source and Destination locations and count for channel 1 transfer.

Field Documentation

UInt32 idma1_handle::cnt

Number of bytes to be transferred

UInt32* idma1_handle::dst

IDMA channel 1 destination

UInt32 idma1_handle::reserved

Reserved area

UInt32* idma1_handle::src

IDMA channel 1 source location

UInt32 idma1_handle::status

IDMA channel 1 status

13.3.2 idma0_config

Detailed Description

IDMA0_Config IDMA Channel 0 configuration - Contains Status, Mask, Source and Destination locations and count for channel 0 (configuration) transfers.

Field Documentation

UInt32 idma0_config::cnt

Number of bytes to be transferred

UInt32* idma0_config::dst

IDMA channel 0 destination

UInt32 idma0_config::mask

IDMA channel 0 mask value

UInt32* idma0_config::src

IDMA channel 0 source location

UInt32 idma0_config::status

IDMA channel 0 status

13.4 Enumerations

This section lists the enumerations available in the IDMA module.

13.4.1 IDMA_Chan

enum IDMA_Chan

This enumeration specifies which IDMA channel will be used. This is used to indicate which IDMA channel (0 or 1) will be used by API.

Enumeration values:

<i>IDMA_CHAN_0</i>	IDMA channel 0
<i>IDMA_CHAN_1</i>	IDMA channel 1

13.4.2 IDMA_intEn

enum IDMA_intEn

This enumeration specifies whether the interrupt event generation is enabled or disabled. This is used to indicate whether the interrupt event generation is enabled or disabled.

Enumeration values:

<i>IDMA_INT_DIS</i>	Idma Int Disable
<i>IDMA_INT_EN</i>	Idma Int Enable

13.4.3 IDMA_priSet

enum IDMA_priSet

This enumeration specifies what priority level the IDMA channel is set to. This is used to specify what priority level the IDMA channel is set to.

Enumeration values:

<i>IDMA_PRI_0</i>	Set Priority level 0
<i>IDMA_PRI_1</i>	Set Priority level 1
<i>IDMA_PRI_2</i>	Set Priority level 2
<i>IDMA_PRI_3</i>	Set Priority level 3
<i>IDMA_PRI_4</i>	Set Priority level 4
<i>IDMA_PRI_5</i>	Set Priority level 5
<i>IDMA_PRI_6</i>	Set Priority level 6
<i>IDMA_PRI_7</i>	Set Priority level 7
<i>IDMA_PRI_NULL</i>	No Priority level

Chapter 14

VCP2 MODULE

Topics

<u>14. 1 Overview</u>
<u>14. 2 Functions</u>
<u>14. 3 Data Structures</u>
<u>14. 4 Macros</u>

14.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within VCP2 module.

Viterbi Decoder Coprocessor 2 (VCP2) is a programmable peripheral for decoding of convolutional codes. The VCP2 is controlled via memory mapped control registers and data buffers. The VCP2 can be used for channel decoding of voice and low bit-rate data channels found in third generation cellular standards that require decoding of convolutional encoded data. The VCP coprocessor has been designed to perform forward error correction for 2G and 3G wireless systems. The VCP can support 1941 12.2 Kbps class A 3G voice channels running at 333 Mhz.

The VCP2 supports:

- Unlimited frame sizes
- Code rates 1/2, 1/3 and 1/4
- Constraint lengths 5, 6, 7, 8, and 9
- Programmable encoder polynomials
- Programmable reliability and convergence lengths
- Hard and soft decoded decisions
- Tail and convergent modes
- Yamamoto logic
- Tail biting logic
- Various input and output FIFO lengths

14.2 Functions

This section lists the functions available in the VCP2 module.

14.2.1 VCP2_genParams

```
void VCP2_genParams      ( VCP2\_BaseParams *restrict      pConfigBase,
                          VCP2\_Params *restrict      pConfigParams
                          )
```

Description

This function calculates the VCP parameters based on the input VCP2_BaseParams object values and sets the values to the output VCP2_Params parameters structure.

Arguments

<code>pConfigBase</code>	Pointer to VCP base parameters structure.
<code>pConfigParams</code>	Pointer to output VCP channel parameters structure.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

Input VCP2_Params structure instance pointed by pConfigParams.

Example

```
VCP2_Params      vcpParam;
VCP2_BaseParams  vcpBaseParam;
...
vcpBaseParam.rate          = VCP2_RATE_1_4;
vcpBaseParam.constLen      = 5;
vcpBaseParam.frameLen      = 255;
vcpBaseParam.yamTh         = 50;
vcpBaseParam.stateNum      = 63;
vcpBaseParam.tbConvrgMode  = FALSE;
vcpBaseParam.decision      = VCP2_DECISION_HARD;
vcpBaseParam.readFlag      = VCP2_OUTF_YES;
vcpBaseParam.tailBitEnable = FALSE;
vcpBaseParam.traceBackIndex = 0x0;
vcpBaseParam.outOrder       = VCP2_OUTORDER_0_31;
vcpBaseParam.perf           = VCP2_SPEED_CRITICAL;
...
VCP2_genParams (&vcpBaseParam, &vcpParam);
```

14.2.2 VCP2_genlc

```
void VCP2_genlc      ( VCP2\_Params *restrict          pConfigParams,
                      VCP2\_Configlc *restrict          pConfiglc
                      )
```

Description

This function generates the required input configuration register values needed to program the VCP based on the parameters provided by VCP2_Params object values.

Arguments

pConfigParams Pointer to channel parameters structure.

pConfiglc Pointer to input configuration structure.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

Input VCP2_Configlc structure instance pointed by pConfiglc.

Example

```
VCP2_Params          vcpParam;
VCP2_BaseParams      vcpBaseParam;
VCP2_Configlc        vcpConfig;

vcpBaseParam.rate      =  VCP2_RATE_1_4;
vcpBaseParam.constLen  =  5;
vcpBaseParam.frameLen  =  2042;
vcpBaseParam.yamTh     =  50;
vcpBaseParam.stateNum  =  63;
vcpBaseParam.tbConvrgMode =  FALSE;
vcpBaseParam.decision  =  VCP2_DECISION_HARD;
vcpBaseParam.readFlag  =  VCP2_OUTF_YES;
vcpBaseParam.tailBitEnable =  FALSE;
vcpBaseParam.traceBackIndex =  0x0;
vcpBaseParam.outOrder  =  VCP2_OUTORDER_0_31;
vcpBaseParam.perf      =  VCP2_SPEED_CRITICAL;

VCP2_genParams (&vcpBaseParam, &vcpParam);

VCP2_genlc (&vcpParam, &vcpConfig);
```

INLINE FUNCTIONS

14.2.3 VCP2_ceil

```
CSL_IDEF_INLINE Uint32 VCP2_ceil          (  Uint32    val,
                                           Uint32    pwr2
                                           )
```

Description

Returns the value rounded to the nearest integer, greater than or equal to $(val/(2^{pwr2}))$.

Arguments

val	Value to be augmented.
pwr2	The power of two by which val must be divisible.

Return Value Uint32

- Value - The smallest number which when multiplied by 2^{pwr2} is greater than val.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint32  val1 = 512;
Uint32  val2 = 4;
Uint32  val3;

val3 = VCP2_ceil (val1, val2);
```

14.2.4 VCP2_normalCeil

```
CSL_IDEF_INLINE Uint32 VCP2_normalCeil    (  Uint32    val1,
                                           Uint32    val2
                                           )
```

Description

Returns the value rounded to the nearest integer, greater than or equal to $(val1/val2)$.

Arguments

val1	Value to be augmented.
val2	Value by which val1 must be divisible.

Return Value Uint32

-
- Value - The smallest number greater than or equal to val1 that is divisible by val2.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32 bmCnt = 512;
    Uint32 numBmFrames;

    // Number of frame transfers with number of bytes
    // transferred to the VCP2 per receive event - 128

    numBmFrames = VCP2_normalCeil(bmCnt, 128);
    ...

```

14.2.5 VCP2_getBmEndian

CSL_IDEF_INLINE Uint32 VCP2_getBmEndian (void)

Description

This function returns the value programmed into the VCPEND register for the branch metrics data for Big Endian mode indicating whether the data is in its native 8-bit format ('1') or 32-bit word packed ('0').

Arguments

None

Return Value Uint32

- Value - Branch metric memory format.
- 0 - 32-bit word packed.
- 1 - Native (8 bits).

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    if (VCP2_getBmEndian ())
    {

```

```

        ...
    } /* end if */

```

14.2.6 VCP2_getIcConfig

CSL_IDEF_INLINE void VCP2_getIcConfig ([VCP2_ConfigIc](#) * *configIc*)

Description

This function gets the current VCPIC register values and puts them in a structure of type VCP2_ConfigIc.

Arguments

configIc Pointer to the structure of type VCP2_ConfigIc to hold the values of VCPIC registers.

Return Value

None

Pre Condition

None

Post Condition

The structure of type VCP2_ConfigIc passed as argument contains the values of the VCP configuration registers.

Modifies

Input structure of type VCP2_ConfigIc.

Example

```

VCP2_ConfigIc    configIc;
...
VCP2_getIcConfig (&configIc);

```

14.2.7 VCP2_getNumInFifo

CSL_IDEF_INLINE Uint32 VCP2_getNumInFifo (void)

Description

This function returns the count, number of symbols currently in the input FIFO.

Arguments

None

Return Value Uint32

- Value - Number of symbols in the branch metric input FIFO buffer.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32 numSym;
    numSym = VCP2_getNumInFifo ();

```

14.2.8 VCP2_getNumOutFifo

CSL_IDEF_INLINE Uint32 VCP2_getNumOutFifo (void)

Description

This function returns the count, number of symbols currently in the output FIFO.

Arguments

None

Return Value Uint32

- Value - Number of symbols present in the output FIFO buffer.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

    Uint32 numSym;
    numSym = VCP2_getNumOutFifo ();

```

14.2.9 VCP2_getSdEndian

CSL_IDEF_INLINE Uint32 VCP2_getSdEndian (void)

Description

This function returns the value programmed into the VCPEND register for the soft decision data for Big Endian mode indicating whether the data is in its native 8-bit format ('1') or 32-bit word packed ('0').

Arguments

None

Return Value Uint32

- Value - Soft decisions memory format.
0 - 32-bit word packed.

1 - Native (8 bits).

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
If (VCP2_getSdEndian ())
{
    ...
} /* end if */
```

14.2.10 VCP2_getStateIndex

CSL_IDEF_INLINE **Uint8** VCP2_getStateIndex (void)

Description

This function returns an index for the final maximum state metric.

Arguments

None

Return Value **Uint8**

- Value - Final maximum state metric index.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint8    index;
index = VCP2_getStateIndex();
```

14.2.11 VCP2_getYamBit

CSL_IDEF_INLINE **Uint8** VCP2_getYamBit (void)

Description

This function returns the value of the Yamamoto bit after the VCP decoding. This bit is a quality indicator and is only used if the yamamoto logic is enabled.

Arguments

None

Return Value Uint8

- Value - Yamamoto bit result.

0 - at least one trellis stage had an absolute difference less than the Yamamoto threshold and the decoded frame has poor quality.

1 - no trellis stage had an absolute difference less than the Yamamoto threshold and the frame has good quality.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
int8    yamBit;

yamBit = VCP2_getYamBit();
```

14.2.12 VCP2_getMaxSm

CSL_IDEF_INLINE Int16 VCP2_getMaxSm (void)

Description

This function returns the final maximum state metric after the VCP has completed its decoding.

Arguments

None

Return Value Int16

- Value - Maximum state metric value for the final trellis stage.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Int16    maxSm;
maxSm = VCP2_getMaxSm();
```

14.2.13 VCP2_getMinSm

CSL_IDEF_INLINE Int16 VCP2_getMinSm (void)

Description

This function returns the final minimum state metric after the VCP has completed its decoding.

Arguments

None

Return Value Int16

- Value - Minimum state metric value for the final trellis stage.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Int16    minSm;

minSm = VCP2_getMinSm();
```

14.2.14 VCP2_icConfig

CSL_IDEF_INLINE void VCP2_icConfig ([VCP2_ConfigIc](#) * vcpConfigIc)

Description

This function programs the VCP input configuration registers with the values provided through the VCP2_ConfigIc structure.

Arguments

vcpConfigIc	Pointer to VCP2_ConfigIc structure instance containing the input configuration register values.
-------------	---

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP input configuration registers.

Example

```
VCP2_ConfigIc      configIc;
configIc.ic0  =  0xf0b07050;
configIc.ic1  =  0x10320000;
configIc.ic2  =  0x000007fa;
configIc.ic3  =  0x00000054;
configIc.ic4  =  0x00800800;
configIc.ic5  =  0x51f3000c;
...
VCP2_icConfig (&configIc);
```

14.2.15 VCP2_icConfigArgs

```
CSL_IDEF_INLINE void VCP2_icConfigArgs          (  Uint32      ic0,
                                                    Uint32      ic1,
                                                    Uint32      ic2,
                                                    Uint32      ic3,
                                                    Uint32      ic4,
                                                    Uint32      ic5
                                                    )
```

Description

This function programs the VCP input configuration registers with the given values.

Arguments

<i>ic0</i>	Value to program input configuration register 0
<i>ic1</i>	Value to program input configuration register 1
<i>ic2</i>	Value to program input configuration register 2
<i>ic3</i>	Value to program input configuration register 3
<i>ic4</i>	Value to program input configuration register 4
<i>ic5</i>	Value to program input configuration register 5

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP input configuration registers.

Example

```

    Uint32 ic0, ic1, ic2, ic3, ic4, ic5;
    ...
    ic0  =  0xf0b07050;
    ic1  =  0x10320000;
    ic2  =  0x000007fa;
    ic3  =  0x00000054;
    ic4  =  0x00800800;
    ic5  =  0x51f3000c;
    ...
    VCP2_icConfigArgs (ic0, ic1, ic2, ic3, ic4, ic5);

```

14.2.16 VCP2_setBmEndian

CSL_IDEF_INLINE void VCP2_setBmEndian (Uint32 *bmEnd*)

Description

This function programs the VCP to view the format of the branch metrics data as either native 8-bit format ('1') or values packed into 32-bit words in little endian format ('0').

Arguments

bmEnd '1' for native 8-bit format and '0' for 32-bit word packed format

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```

    Uint32 bmEnd = VCP2_END_NATIVE;
    VCP2_setBmEndian (bmEnd);

```

14.2.17 VCP2_setNativeEndian

CSL_IDEF_INLINE void VCP2_setNativeEndian (void)

Description

This function programs the VCP to view the format of all data as native 8-bit format.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```
VCP2_setNativeEndian ();
```

14.2.18 VCP2_setPacked32Endian

CSL_IDEF_INLINE void VCP2_setPacked32Endian (void)

Description

This function programs the VCP to view the format of all data as packed data in 32-bit words.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```
VCP2_setPacked32Endian ();
```

14.2.19 VCP2_setSdEndian

CSL_IDEF_INLINE void VCP2_setSdEndian (Uint32 sdEnd)

Description

This function programs the VCP to view the format of the soft decision data as either native 8-bit format ('1') or values packed into 32-bit words in little endian format ('0').

Arguments

sdEnd '1' for native 8-bit format and '0' for 32-bit word packed format

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```
Uint32 sdEnd = VCP2_END_NATIVE;
VCP2_setSdEndian (sdEnd);
```

14.2.20 VCP2_addPoly

```
CSL_IDEF_INLINE void VCP2_addPoly          ( VCP2\_Poly *           poly,
                                             VCP2\_Params *         params
                                             )
```

Description

This function is used to add either predefined or user defined polynomials to the generated VCP2_Params.

Arguments

pPoly	Pointer to the structure of type VCP2_Poly containing the values of generator polynomials.
pParams	Pointer to the structure of type VCP2_Params containing the generated values for input configuration registers.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

Structure of type VCP2_Params passed as argument to the function.

Example

```
VCP2_Poly  poly = {VCP2_GEN_POLY_3, VCP2_GEN_POLY_1,
                  VCP2_GEN_POLY_2, VCP2_GEN_POLY_3};
VCP2_Params      params;
VCP2_BaseParams  baseParams;

...

VCP2_genParams (&baseParams, &params);
VCP2_addPoly (&poly, &params);
```

14.2.21 VCP2_statError

CSL_IDEF_INLINE Bool VCP2_statError (void)

Description

This function returns a boolean value indicating whether any VCP error has occurred.

Arguments

None

Return Value Bool

- bitStatus - ERR bit field value of VCP status register 0.

0 – No error.

1 – VCP paused due to error.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
VCP2_Errors error;
/* check whether an error has occurred */
if (VCP2_statError ())
{
    VCP2_getErrors (&error);
} /* end if */
```

14.2.22 VCP2_statInFifo

CSL_IDEF_INLINE Uint32 VCP2_statInFifo (void)

Description

This function returns the input FIFO's empty status flag. A '1' indicates that the input FIFO is empty and a '0' indicates it is not empty.

Arguments

None

Return Value Uint32

- bitStatus - IFEMP bit field value of VCP status register 0.

0 – Input FIFO is not empty.

1 – Input FIFO is empty.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (VCP2_statInFifo ())
{
    ...
} /* end if */
```

14.2.23 VCP2_statOutFifo

CSL_IDEF_INLINE Uint32 VCP2_statOutFifo (void)

Description

This function returns the output FIFO's full status flag. A '1' indicates that the output FIFO is full and a '0' indicates it is not full.

Arguments

None

Return Value Uint32

- bitStatus - OFFUL bit field value of VCP status register 0.

0 – Output FIFO is not full.

1 – Output FIFO is full.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (VCP2_statOutFifo ())
{
    ...
} /* end if */
```

14.2.24 VCP2_statPause

CSL_IDEF_INLINE Uint32 VCP2_statPause (void)

Description

This function returns the PAUSE bit status indicating whether the VCP is paused or not.

Arguments

None

Return Value Uint32

- bitStatus - PAUSE bit field value of VCP status register 0.

0 – VCP is not paused.

1 – VCP is paused.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
/* Pause the VCP */
VCP2_pause ();
/* Wait for pause to take place */
while (!VCP2_statPause ());
```

14.2.25 VCP2_statRun

CSL_IDEF_INLINE Uint32 VCP2_statRun (void)

Description

This function returns the RUN bit status indicating whether the VCP is running or not.

Arguments

None

Return Value Uint32

- bitStatus - RUN bit field value of VCP status register 0.

0 – VCP is not running.

1 – VCP is running.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
/* start the VCP */
VCP2_start ();
/* check that the VCP is running */
while (!VCP2_statRun ());
```

14.2.26 VCP2_statSymProc

CSL_IDEF_INLINE Uint32 VCP2_statSymProc (void)

Description

This function returns the number of symbols processed, NSYMPROC bitfield of VCP.

Arguments

None

Return Value Uint32

- Value - Number of symbols processed.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
Uint32 numSym;
...
numSym = VCP2_statSymProc ();
```

14.2.27 VCP2_statWaitlc

CSL_IDEF_INLINE Uint32 VCP2_statWaitlc (void)

Description

This function returns the WIC bit status indicating whether the VCP is waiting to receive new input configuration values.

Arguments

None

Return Value Uint32

- bitStatus - WIC bit field value of VCP status register 0.
 0 – VCP is not waiting for input configuration words.
 1 – VCP is waiting for input configuration words.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
if (VCP2_statWaitIc ())
{
    ...
} /* end if */
```

14.2.28 VCP2_start

CSL_IDEF_INLINE void VCP2_start (void)

Description

This function starts the VCP by writing a start command to the VCPEXE register.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

VCP is started.

Modifies

VCP execution register.

Example

```
VCP2_start ();
```

14.2.29 VCP2_pause

CSL_IDEF_INLINE void VCP2_pause (void)

Description

This function pauses the VCP by writing a pause command to the VCPEXE register.

Arguments

None

Return Value

None

Pre Condition

The VCP should be operating in debug/emulation mode.

Post Condition

VCP is paused.

Modifies

VCP execution register

Example

```
VCP2_pause ( );
```

14.2.30 VCP2_unpause

CSL_IDEF_INLINE void VCP2_unpause (void)

Description

This function un-pauses the VCP, previously paused by VCP2_pause() function, by writing the un-pause command to the VCPEXE register. This function restarts the VCP at the beginning of current trace back, and VCP will run to normal completion.

Arguments

None

Return Value

None

Pre Condition

The VCP should be operating in debug/emulation mode.

Post Condition

VCP is restarted.

Modifies

VCP execution register.

Example

```
VCP2_unpause ( );
```

14.2.31 VCP2_stepTraceback

CSL_IDEF_INLINE void VCP2_stepTraceback (void)

Description

This function un-pauses the VCP, previously paused by VCP2_pause() function, by writing the un-pause command to the VCPEXE register. This function restarts the VCP at the beginning of current trace back and halts at the next trace back (i.e. Step Single Trace back).

Arguments

None

Return Value

None

Pre Condition

The VCP should be operating in debug/emulation mode.

Post Condition

VCP is restarted.

Modifies

VCP execution register.

Example

```
VCP2_stepTraceback ();
```

14.2.32 VCP2_reset

CSL_IDEF_INLINE void VCP2_reset (void)

Description

This function sets all the VCP control registers to their default values.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

All registers in the VCP are reset except for the execution register, endian register, emulation register and other internal registers.

Modifies

VCP execution register

Example

```
VCP2_reset ();
```

14.2.33 VCP2_getErrors

CSL_IDEF_INLINE void VCP2_getErrors ([VCP2_Errors](#) * *pVcpErr*)

Description

This function will acquire the VCPERR register values and fill in the fields of VCP2_Error structure and pass it back as the results.

Arguments

pVcpErr Pointer to the VCP2_Errors structure instance.

Return Value

None

Pre Condition

None

Post Condition

The fields of the VCP2_Errors structure indicate the respective errors, if occurred.

Modifies

VCPSTAT0 register as a side effect. Clears ERR bit of VCPSTAT0 register.

Example

```
VCP2_Errors error;
/* Check whether an error has occurred */
if (VCP2_statError ())
{
    VCP2_getErrors (&error);
} /* end if */
```

14.2.34 VCP2_statEmuHalt

CSL_IDEF_INLINE Uint32 VCP2_statEmuHalt (void)

Description

This function returns the EMUHALT bit status indicating whether the VCP halt is due to emulation or not.

Arguments

None

Return Value Uint32

- bitStatus - Emuhalt bit field value of VCP status register 0.
- 0 – Not halt due to emulation.
- 1 – Halt due to emulation.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

If (VCP2_statEmuHalt ())
{
    ...
}/* end if */

```

14.2.35 VCP2_getVssSleepMode

CSL_IDEF_INLINE Uint32 VCP2_getVssSleepMode (void)

Description

This function returns the value programmed into VCPEND register for sleep mode indicating if sleep mode is disabled or if internal power down control is enabled for SLPZVSS.

Arguments

None

Return Value Uint32

- Value - Sleep mode enable/disable.
- 0 - Sleep mode disabled.
- 1 - Sleep mode enabled.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

Uint32 slpMode;
slpMode = VCP2_getVssSleepMode ();

```

14.2.36 VCP2_getVddSleepMode

CSL_IDEF_INLINE Uint32 VCP2_getVddSleepMode (void)

Description

This function returns the value programmed into VCPEND register for sleep mode indicating if sleep mode is disabled or if internal power down control is enabled for SLPZVDD.

Arguments

None

Return Value Uint32

- Value - Sleep mode enable/disable.
- 0 - Sleep mode disabled.
- 1 - Sleep mode enabled.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

Uint32 slpMode;
slpMode = VCP2_getVddSleepMode ();

```

14.2.37 VCP2_setVssSleepMode

CSL_IDEF_INLINE void VCP2_setVssSleepMode (Uint32 slpMode)

Description

This function either disables sleep mode or enables the internal power down control of SLPZVSS.

Arguments

slpMode '0' to disable sleep mode and '1' to enable internal power down control for SLPZVSS.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```

VCP2_setVssSleepMode (1)

```


14.2.38 P2_setVddSleepMode

CSL_IDEF_INLINE void VCP2_setVddSleepMode (**Uint32** *slpMode*)

Description

This function either disables sleep mode or enables the internal power down control of SLPZVDD.

Arguments

slpMode '0' to disable sleep mode and '1' to enable internal power down control for SLPZVDD.

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP endian register

Example

```
VCP2_setVddSleepMode (1);
```

14.2.39 VCP2_emuEnable

CSL_IDEF_INLINE void VCP2_emuEnable (**Uint16** *emuMode*)

Description

This function enables the emulation/debug mode of VCP.

Arguments

emuMode '0' to halt VCP at the end of completion of the current window of state metric processing or at the end of a frame.
'1' to halt the VCP at the end of completion of the processing of the frame.

Return Value

None

Pre Condition

None

Post Condition

Emulation mode is enabled.

Modifies

VCP emulation control register.

Example

```

    Uint16 emuMode = VCP2_EMUHALT_DEFAULT;
    VCP2_emuEnable (emuMode);

```

14.2.40 VCP2_emuDisable

CSL_IDEF_INLINE void VCP2_emuDisable (void)

Description

This function disables the emulation/debug mode of VCP.

Arguments

None

Return Value

None

Pre Condition

None

Post Condition

None

Modifies

VCP emulation control register

Example

```

    VCP2_emuDisable ();

```

14.2.41 VCP2_getId

CSL_IDEF_INLINE Uint32 VCP2_getId (void)

Description

This function returns the VCP Id.

Arguments

None

Return Value Uint32

- Value - VCP id.

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
int32 vcpid;  
vcpid = VCP2_getId ();
```

14.3 Data Structures

This section lists the data structures available in the VCP2 module.

14.3.1 VCP2_Configlc

Detailed Description

VCP input configuration structure that holds all of the configuration values that are to be transferred to the VCP via the EDMA.

Field Documentation

UInt32 VCP2_Configlc::ic0

Value of VCP input configuration register 0

UInt32 VCP2_Configlc::ic1

Value of VCP input configuration register 1

UInt32 VCP2_Configlc::ic2

Value of VCP input configuration register 2

UInt32 VCP2_Configlc::ic3

Value of VCP input configuration register 3

UInt32 VCP2_Configlc::ic4

Value of VCP input configuration register 4

UInt32 VCP2_Configlc::ic5

Value of VCP input configuration register 5

14.3.2 VCP2_Params

Detailed Description

VCP channel parameters structure that holds all of the information concerning the user channel. These values are used to generate the appropriate input configuration values for the VCP.

Field Documentation

UInt8 VCP2_Params::bmBuffLen

Branch metrics buffer length in input FIFO

UInt8 VCP2_Params::constLen

Constraint length

UInt16 VCP2_Params::convDist

Convergence distance

UInt8 VCP2_Params::decBuffLen

Decisions buffer length in output FIFO

UInt8 VCP2_Params::decision

Decision selection: hard or soft

Uint16 VCP2_Params::frameLen
Frame length i.e. number of symbols in a frame

Int16 VCP2_Params::maxSm
Maximum initial state metric

Int16 VCP2_Params::minSm
Minimum initial state metric

Uint16 VCP2_Params::numBmFrames
Number of branch metric frames

Uint16 VCP2_Params::numDecFrames
Number of decision frames

Uint16 VCP2_Params::outOrder
Hard decision output ordering

Uint8 VCP2_Params::poly0
Polynomial 0

Uint8 VCP2_Params::poly1
Polynomial 1

Uint8 VCP2_Params::poly2
Polynomial 2

Uint8 VCP2_Params::poly3
Polynomial 3

VCP2_Rate VCP2_Params::rate
Code rate

Uint8 VCP2_Params::readFlag
Output parameters read flag

Uint16 VCP2_Params::relLen
Reliability length

Uint8 VCP2_Params::stateNum
State index set to the maximum initial state metric

Uint8 VCP2_Params::traceBack
Traceback mode

Bool VCP2_Params::traceBackEn
Traceback state index enable/disable

Uint16 VCP2_Params::traceBackIndex
Traceback state index

Uint16 VCP2_Params::yamTh
Yamamoto threshold value

14.3.3 VCP2_BaseParams

Detailed Description

VCP base parameter structure that is used to configure the VCP parameters structure with the given values using VCP2_genParams() function.

Field Documentation

UInt8 VCP2_BaseParams::constLen

Constraint length

UInt8 VCP2_BaseParams::decision

Output decision type

UInt16 VCP2_BaseParams::frameLen

Frame length

UInt8 VCP2_BaseParams::outOrder

Hard decision output ordering

UInt8 VCP2_BaseParams::perf

Performance and speed

VCP2_Rate VCP2_BaseParams::rate

Code rate

UInt8 VCP2_BaseParams::readFlag

Output parameters read flag

UInt8 VCP2_BaseParams::stateNum

Maximum initial state metric value

Bool VCP2_BaseParams::tailBitEnable

Enable/Disable tail biting

Bool VCP2_BaseParams::tbConvrgMode

Traceback convergence mode

UInt16 VCP2_BaseParams::traceBackIndex

Tailbiting traceback index mode

UInt16 VCP2_BaseParams::yamTh

Yamamoto threshold value

14.3.4 VCP2_Errors

Detailed Description

VCP Error structure

Field Documentation

Bool VCP2_Errors::fctlErr

Reliability + convergence distance error

Bool VCP2_Errors::ftlErr

Frame length error

Bool VCP2_Errors::maxminErr

Max-Min error

Bool VCP2_Errors::symrErr

SYMR error

Bool VCP2_Errors::symxErr

SYMx error

Bool VCP2_Errors::tbnaErr

Traceback mode error

14.3.5 VCP2_Poly

Detailed Description

VCP generator polynomials structure

Field Documentation**Uint8 VCP2_Poly::poly0**

Generator polynomial 0

Uint8 VCP2_Poly::poly1

Generator polynomial 1

Uint8 VCP2_Poly::poly2

Generator polynomial 2

Uint8 VCP2_Poly::poly3

Generator polynomial 3

14.4 Macros

#define hVcp2 ((CSL_Vcp2ConfigRegs*)CSL_VCP2_0_REGS)

Handle to access VCP2 registers accessible through config bus

#define hVcp2Vbus ((CSL_Vcp2EdmaRegs *)CSL_VCP2_EDMA_REGS)

Handle to access VCP2 registers accessible through EDMA bus

#define VCP2_DECISION_HARD CSL_VCP2_VCP_IC5_SDHD_HARD

Output decision type: Hard decisions

#define VCP2_DECISION_SOFT CSL_VCP2_VCP_IC5_SDHD_SOFT

Output decision type: Soft decisions

#define VCP2_EMUHALT_DEFAULT CSL_VCP2_VCP_EMU_SOFT_HALT_DEFAULT

EMU mode: VCP halts at the end of completion of the current window of state metric processing or at the end of a frame

#define VCP2_EMUHALT_FRAMEEND CSL_VCP2_VCP_EMU_SOFT_HALT_FRAMEEND

EMU mode : VCP halts at the end of completion of the processing of the frame

#define VCP2_END_NATIVE CSL_VCP2_VCP_END_SD_NATIVE

Soft decisions memory format: Native (8 bits)

#define VCP2_END_PACKED32 CSL_VCP2_VCP_END_SD_32BIT

Soft decisions memory format: 32-bit word packed

#define VCP2_GEN_POLY_0 0x30

GSM/Edge/GPRS generator polynomial 0

#define VCP2_GEN_POLY_1 0xB0

GSM/Edge/GPRS generator polynomial 1

#define VCP2_GEN_POLY_2 0x50

GSM/Edge/GPRS generator polynomial 2

#define VCP2_GEN_POLY_3 0xF0

GSM/Edge/GPRS generator polynomial 3

#define VCP2_GEN_POLY_4 0x6C

GSM/Edge/GPRS generator polynomial 4

#define VCP2_GEN_POLY_5 0x94

GSM/Edge/GPRS generator polynomial 5

#define VCP2_GEN_POLY_6 0xF4

GSM/Edge/GPRS generator polynomial 6

#define VCP2_GEN_POLY_7 0xE4

GSM/Edge/GPRS generator polynomial 7

#define VCP2_GEN_POLY_GNULL 0x00

NULL generator polynomial for GSM/Edge/GPRS

#define VCP2_OUTF_NO CSL_VCP2_VCP_IC5_OUTF_NO
Output parameters read flag: VCP read event is not generated

#define VCP2_OUTF_YES CSL_VCP2_VCP_IC5_OUTF_YES
Output parameters read flag: VCP read event is generated

#define VCP2_OUTORDER_0_31 CSL_VCP2_VCP_IC3_OUT_ORDER_0_31
Out order of VCP output for decoded data: 0 to 31

#define VCP2_OUTORDER_31_0 CSL_VCP2_VCP_IC3_OUT_ORDER_31_0
Out order of VCP output for decoded data: 31 to 0

#define VCP2_PERF_CRITICAL 2
Performance critical

#define VCP2_PERF_DEFAULT VCP2_SPEED_CRITICAL
Default value

#define VCP2_PERF_MOST_CRITICAL 3
Performance most critical

#define VCP2_RATE_1_2 2
Code rate = 2

#define VCP2_RATE_1_3 3
Code rate = 3

#define VCP2_RATE_1_4 4
Code rate = 4

#define VCP2_SPEED_CRITICAL 0
Speed critical

#define VCP2_SPEED_MOST_CRITICAL 1
Speed most critical

#define VCP2_TRACEBACK_CONVERGENT CSL_VCP2_VCP_IC5_TB_CONV
Traceback mode: Convergent

#define VCP2_TRACEBACK_MIXED CSL_VCP2_VCP_IC5_TB_MIX
Traceback mode: Mixed

#define VCP2_TRACEBACK_NONE CSL_VCP2_VCP_IC5_TB_NO
No trace back allowed

#define VCP2_TRACEBACK_TAILED CSL_VCP2_VCP_IC5_TB_TAIL
Traceback mode: Tailed

#define VCP2_UNPAUSE_NORMAL CSL_VCP2_VCP_EXE_COMMAND_RESTART
VCP unpause type: VCP restarts

#define VCP2_UNPAUSE_ONESW CSL_VCP2_VCP_EXE_COMMAND_RESTART_PAUSE
VCP unpause type: VCP restarts and processes one sliding window before pausing again

Chapter 15

CHIP MODULE

Topics

<u>15. 1 Overview</u>

<u>15. 2 Functions</u>
--

<u>15. 3 Enumerations</u>

15.1 Overview

This module deals with all System On Chip (SOC) configurations. It constitutes of Configuration Registers specific for the chip. Following are the Registers associated with the CHIP module:

- Addressing Mode Register - This register specifies the addressing mode for the registers which can perform linear or circular addressing, also contain sizes for circular addressing
- Control Status Register - This register contains the control and status bits. This register is used to control the mode of cache. This is also used to enable or disable all the interrupts except reset and non maskable interrupt.
- Interrupt Flag Register – This register contains the status of INT4–INT15 and NMI interrupt. Each corresponding bit in the IFR is set to 1 when that interrupt occurs; otherwise, the bits are cleared to 0.

Interrupt Set Register - This register allows user to manually set the maskable interrupts (INT4–INT15) in the interrupt flag register (IFR). Writing a 1 to any of the bits in ISR causes the corresponding interrupt flag to be set in IFR.

- Interrupt Clear Register – This register allows user to manually clear the maskable interrupts (INT15–INT4) in the interrupt flag register (IFR). Writing a 1 to any of the bits in ICR causes the corresponding interrupt flag to be cleared in IFR.
- Interrupt Enable Register - This register enables and disables individual interrupts and this not accessible in User mode.
- Interrupt Service Table Pointer Register – This register is used to locate the interrupt service routine (ISR).
- Interrupt Return Pointer Register – This register contains the return pointer that directs the CPU to the proper location to continue program execution after processing a maskable interrupt.

Nonmaskable Interrupt (NMI) Return Pointer Register – This register contains the return pointer that directs the CPU to the proper location to continue program execution after processing of a non-maskable interrupt (NMI).

- Exception Return Pointer Register – This register contains the return pointer that directs the CPU to the proper location to continue program execution after processing of a exception.
- Time Stamp Counter Registers – The CPU contains a free running 64-bit counter that advances each CPU clock after counting is enabled. The counter is accessed using two 32-bit read-only control registers, Time Stamp Counter Registers – Low (TSCL) and Time Stamp Counter Registers – High (TSCH). The counter is enabled by writing to TSCL. The value written is ignored. Once enabled, counting cannot be disabled under program control. Counting is disabled in the following cases:
 - After exiting the reset state.
 - When the CPU is fully powered down.

SPLOOP Inner Loop Count Register - The SPLOOP or SPLOOPD instructions use the SPLOOP inner loop count register (ILC), as the count of the number of iterations left to perform. The ILC content is decremented at each stage boundary until the ILC content reaches 0.

SPLOOP Reload Inner Loop Count Register - Predicated SPLOOP or SPLOOPD instructions used in conjunction with a SPMASKR or SPKERNELR instruction use the SPLOOP reload inner

loop count register (RILC), as the iteration count value to be written to the SPLOOP inner loop count register (ILC) in the cycle before the reload operation begins.

- E1 Phase Program Counter – This register contains the 32-bit address of the fetch packet in the E1 pipeline phase.

DSP Core Number Register – This register provides an identifier to shared resources in the system which identifies which CPU is accessing those resources. The contents of this register are set to a specific value at reset.

- Saturation Status Register – This register provides saturation flags for each functional unit, making it possible for the program to distinguish between saturations caused by different instructions in the same execute packet.
- GMPY Polynomial.A Side Register – The GMPY instruction uses the 32-bit polynomial in the GMPY polynomial—A side register (GPLYA), when the instruction is executed on the M1 unit.
- GMPY Polynomial. B Side Register – The GMPY instruction uses the 32-bit polynomial in the GMPY polynomial—B side register (GPLYB), when the instruction is executed on the M2 unit.
- Galois Field Polynomial Generator Function Register – This register controls the field size and the Galois field polynomial generator of the Galois field multiply hardware.

Task State Register – This register contains all of the status bits that determine or indicate the current execution environment. TSR is saved in the event of an interrupt or exception to the ITSR or NTSR, respectively.

Interrupt Task State Register – This register is used to store the contents of the task state register (TSR) in the event of an interrupt.

- NMI/Exception Task State Register – This register is used to store the contents of the task state register (TSR) and the conditions under which an exception occurred in the event of a nonmaskable interrupt (NMI) or an exception.
- Exception Flag Register – This register contains bits that indicate which exceptions have been detected. Clearing the EFR bits is done by writing a 1 to the corresponding bit position in the exception clear register (ECR).
- Exception Clear Register – This register is used to clear individual bits in the exception flag register (EFR). Writing a 1 to any bit in ECR clears the corresponding bit in EFR.
- Internal Exception Report Register – This register contains flags that indicate the cause of the internal exception.
- Restricted Entry Point Address Register – This register is used by the SWENR instruction as the target of the change of control when an SWENR instruction is issued. The contents of REP should be preinitialized by the processor in Supervisor mode before any SWENR instruction is issued.

15.2 Functions

This section lists the functions available in the CHIP module.

15.2.1 CSL_chipWriteReg

```

Uint32 CSL_chipWriteReg ( CSL\_ChipReg reg,
                           CSL_Reg32 val
                           )

```

Description

This API writes specified control register with the specified value 'val'. The register that can be specified could be one of those enumerated in CSL_ChipReg.

Arguments

<i>reg</i>	This is the register name specified for the register through the enum.
<i>val</i>	Value to be written into the register.

Return Value

Uint32
 The value in the register before the new value being written

- Old programmed value

Pre Condition

None

Post Condition

The reg control register is written with the value passed.

Modifies

The specified register will be modified.

Usage Constraints

Please refer to the C64x+ user guide for constraints while accessing registers in different privilege levels

Example

```

Uint32 oldamr;
oldamr = CSL_chipWriteReg (AMR, 56);

```

15.2.2 CSL_chipReadReg

```

Uint32 CSL_chipReadReg ( CSL\_ChipReg reg )

```

Description

This API reads the specified control register. The register that can be specified could be one of those enumerated in CSL_ChipReg.

Arguments

<i>reg</i>	This is the register name specified for the register through the enum
------------	---

Return Value

UInt32

- The value read from the register

Pre Condition

None

Post Condition

None

Modifies

None

Usage Constraints

Please refer to the C64x+ user guide for constraints while accessing registers in different privilege levels

Example

```
UInt32 amr;  
amr = CSL_chipReadReg (AMR);
```

15.3 Enumerations

This section lists the enumerations available in the CHIP module.

15.3.1 CSL_ChipReg

enum CSL_ChipReg

Enumeration for the CHIP registers

Enumeration values:

<i>AMR</i>	Addressing Mode Register
<i>CSR</i>	Control Status Register
<i>IFR</i>	Interrupt Flag Register
<i>ISR</i>	Interrupt Set Register
<i>ICR</i>	Interrupt Clear Register
<i>IER</i>	Interrupt Enable Register
<i>ISTP</i>	Interrupt Service Table Pointer Register
<i>IRP</i>	Interrupt Return Pointer Register
<i>NRP</i>	Nonmaskable Interrupt (NMI) Return Pointer Register
<i>ERP</i>	Exception Return Pointer Register
<i>TSCL</i>	Time Stamp Counter Register - Low
<i>TSCH</i>	Time Stamp Counter Registers - High
<i>ILC</i>	SPLOOP Inner Loop Count Register
<i>RILC</i>	SPLOOP Reload Inner Loop Count Register
<i>PCE1</i>	E1 Phase Program Counter
<i>DNUM</i>	DSP Core Number Register
<i>SSR</i>	Saturation Status Register
<i>GPLYA</i>	GMPY Polynomial A Side Register
<i>GPLYB</i>	GMPY Polynomial B Side Register
<i>GFPGFR</i>	Galois Field Polynomial Generator Function Register
<i>TSR</i>	Task State Register
<i>ITSR</i>	Interrupt Task State Register
<i>NTSR</i>	NMI/Exception Task State Register
<i>EFR</i>	Exception Flag Register
<i>ECR</i>	Exception Clear Register
<i>IERR</i>	Internal Exception Report Register
<i>REP</i>	Restricted Entry Point Address Register

Chapter 16

EDC MODULE

Topics

<u>16. 1 Overview</u>
<u>16. 2 Functions</u>
<u>16. 3 Data Structures</u>
<u>16. 4 Enumerations</u>

16.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within EDC module.

L1P and L2 support error detection and correction mechanism to protect the program code and static data which are not frequently changed. L1p support error detectction and L2 support error detection and correction mechanism.

16.2 Functions

This section lists the functions available in the EDC module.

16.2.1 CSL_edcEnable

CSL_Status CSL_edcEnable ([CSL_EdcMem](#) *edcMem*)

Description

Enables the EDC for the specified memory.

Arguments

edcMem Specifies what memory EDC is to be enabled

Return Value CSL_Status

- CSL_SOK - EDC disable for specified memory is successful
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status      status;
...
status = CSL_edcEnable (CSL_EDC_L1P);
...
```

16.2.2 CSL_edcDisable

CSL_Status CSL_edcDisable ([CSL_EdcMem](#) *edcMem*)

Description

Disables the EDC for the specified memory.

Arguments

edcMem Specifies what memory EDC is to be disabled

Return Value CSL_Status

- CSL_SOK - EDC disable for specified memory is successful
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status          status;
...
status = CSL_edcDisable (CSL_EDC_L1P);
...
```

16.2.3 CSL_edcSuspend

CSL_Status CSL_edcSuspend ([CSL_EdcMem](#) *edcMem*)

Description

Suspend the EDC for the specified memory.

Arguments

edcMem Specifies what memory EDC is to be suspend

Return Value CSL_Status

- CSL_SOK - EDC suspend for specified memory is successful
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status          status;
...
status = CSL_edcSuspend (CSL_EDC_L1P);
...
```

16.2.4 CSL_edcClear

CSL_Status CSL_edcClear ([CSL_EdcMem](#) *edcMem*,
[CSL_EdcClrAccessType](#) *edcAccessType*
)

Description

Clears the Address of the parity error for the specified memory along with the access type parity error bit.

Arguments

<code>edcMem</code>	Specifies what memory EDC error address is to be cleared
<code>edcAccessType</code>	Specifies what fetch type parity error bit or parity error count type is to be cleared.

Return Value `CSL_Status`

- `CSL_SOK` - Address of parity error clear is successful
- `CSL_ESYS_INVPARAMS` - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

EDC registers

Example

```
CSL_Status          status;
...
status = CSL_edcClear (CSL_EDC_L1P, CSL_EDC_DCLR);
...
```

16.2.5 CSL_edcGetErrorAddress

```
CSL_Status CSL_edcGetErrorAddress ( CSL\_EdcMem          edcMem,
                                   CSL\_EdcAddrInfo * edcAddr
                                   )
```

Description

Gets the Address location of the parity error.

Arguments

<code>edcMem</code>	Specifies what memory EDC Address Info is to be acquired for.
<code>edcAddr</code>	Structure for returning Address, L2 Way, SRAM/Cache info bitposition for error.

Return Value `CSL_Status`

- `CSL_SOK` - EDC get error address is successful
- `CSL_ESYS_INVPARAMS` - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status          status;
CSL_EdcAddrInfo    edcAddr
...
status = CSL_edcGetErrorAddress (CSL_EDC_L1P, &edcAddr);
...
```

16.2.6 CSL_edcGetHwStatus

```
CSL_Status CSL_edcGetHwStatus ( CSL\_EdcMem          edcMem,
                                CSL\_EdcHwStatusQuery    query,
                                void *                response
                                )
```

Description

Gets the requested HW Status of the specified memory.

Arguments

<i>edcMem</i>	Specifies what memory EDC status is to be obtained.
<i>query</i>	The query to this API which indicates the status to be returned.
<i>response</i>	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - EDC get error address is successful
- CSL_ESYS_INVPARAMS - The parameter passed is invalid
- CSL_ESYS_INVQUERY - Query command not supported

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status          status;
Uint16              enStat;
...
status = CSL_edcGetHwStatus (CSL_EDC_L1P, CSL_EDC_QUERY_ENABLESTAT,
```

```

...
(void *)&enStat);

```

16.2.7 CSL_edcPageEnable

```

CSL_Status CSL_edcPageEnable ( Uint32          mask,
                               CSL\_EdcUmap      umap
                               )

```

Description

Enables the pages for EDC specified by a 32-bit mask.

Arguments

mask	Specifies what pages of the given map(s) are to be enabled by setting the bit corresponding to the page to 1
umap	Specifies which map(s) to apply mask to (MAP0, MAP1, or BOTH)

Return Value CSL_Status

- CSL_SOK - Enable pages for EDC is successful
- CSL_ESYS_INVPARAMS - The parameter passed is invalid

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

CSL_Status          status;
...
status = CSL_edcPageEnable (0x1, CSL_EDC_UMAP0);
...

```

16.3 Data Structures

This section lists the data structures available in the EDC module.

16.3.1 CSL_EdcAddrInfo

Detailed Description

CSL_EdcAddrInfo has all the fields required locate the parity error

Field Documentation

UInt32 CSL_EdcAddrInfo::addr

Address of the parity error - 5 LSBs always 0

[CSL_EdcAddrL2way](#) CSL_EdcAddrInfo::l2way

The cache way the error was detected in if in cache

[CSL_EdcAddrSram](#) CSL_EdcAddrInfo::sram

Parity error was detected in SRAM or Cache

UInt8 CSL_EdcAddrInfo::bitPos

Bit Position of the sigle bit error in the 32 bit word (of the 256 bit block address returned in the ADDR field of L2EDADDR register) identified by the BITPOS[4:0] (word32Bit) of L2EDSTAT register

UInt8 CSL_EdcAddrInfo::word32Bit

32 bit word location of the 256 bit block address returned in the ADDR field of L2EDADDR register

16.3.2 CSL_EdcStatusInfo

Detailed Description

CSL_EdcStatusInfo used to get the error status for L1 and L2

Field Documentation

UInt8CSL_EdcStatusInfo::enStat

Enable status for L1 and L2

UInt8CSL_EdcStatusInfo::disStat

Disable status for L1 and L2

UInt8CSL_EdcStatusInfo::suspStat

Suspend status for L1 and L2

UInt8CSL_EdcStatusInfo::prgErr

Intstruction fetch error status for L1 and L2

Uint8CSL_EdcStatusInfo::dmaErr

DMA error status for L1 and L2

Uint8CSL_EdcStatusInfo::dataErr

Data fetch error status for L2

16.4 Enumerations

This section lists the enumerations available in the EDC module.

16.4.1 CSL_EdcMem

enum CSL_EdcMem

Memory Specifier for EDC. Used to indicate which memories (L1P or L2) are to be affected by the API.

Enumeration values:

CSL_EDC_L1P L1P's EDC will be affected by the APIs when CSL_EDC_L1P is used
CSL_EDC_L2 L2's EDC will be affected by the APIs when CSL_EDC_L2 is used

16.4.2 CSL_EdcClrAccessType

enum CSL_EdcClrAccessType

Specifies the Access Type for which the parity error is to be cleared. Used to indicate which access parity error bit to be cleared.

Enumeration values:

CSL_EDC_DCLR Data fetch parity error bit to be cleared
CSL_EDC_PCLR Program fetch parity error bit to be cleared
CSL_EDC_DMACLR DMA read parity error bit to be cleared
CSL_EDC_CECNTCLR Correctable parity error count value to be cleared
CSL_EDC_NCECNTCLR Non-correctable parity error count value to be cleared

16.4.3 CSL_EdcHwStatusQuery

enum CSL_EdcHwStatusQuery

EDC Hardware Status Query Type. Used to indicate what HW status to query.

Enumeration values:

<i>CSL_EDC_QUERY_ENABLESTAT</i>	Query enabled/disabled status. Parameters: (<i>CSL_EdcEnableStatus</i>)
<i>CSL_EDC_QUERY_ERRORSTAT</i>	Query error status. Parameters: (<i>CSL_EdcErrorStatus</i>)
<i>CSL_EDC_QUERY_NERRSTAT</i>	Query number of bit error status (L2 only). Parameters: (<i>CSL_EdcNumErrors</i>)
<i>CSL_EDC_QUERY_BITPOS</i>	Query bit position of error (L2 only). Parameters: (<i>Uint32 *</i>)
<i>CSL_EDC_QUERY_ALLSTAT</i>	Query all status (returns all bit fields EDSTAT)

	register).
	Parameters:
	<i>CSL_EdcStatusInfo</i> *)
<i>CSL_EDC_QUERY_PAGE0</i>	Query page 0 enables (L2 only).
	Parameters:
	<i>(Uint32 *)</i>
<i>CSL_EDC_QUERY_PAGE1</i>	Query page 1 enables (L2 only).
	Parameters:
	<i>(Uint32 *)</i>
<i>CSL_EDC_QUERY_CE_CNT</i>	Query correctable error count (L2 only).
	Parameters:
	<i>(Uint32 *)</i>
<i>CSL_EDC_QUERY_NCE_CNT</i>	Query non-correctable error count (L2 only).
	Parameters:
	<i>(Uint32 *)</i>

16.4.4 CSL_EdcEnableStatus

enum CSL_EdcEnableStatus

EDC Enable/Disable Status. Used to indicate whether EDC is enabled, disabled, or suspended.

Enumeration values:

<i>CSL_EDC_ENABLED</i>	EDC enabled
<i>CSL_EDC_DISABLED</i>	EDC disabled
<i>CSL_EDC_SUSPENDED</i>	EDC suspended

16.4.5 CSL_EdcErrorStatus

enum CSL_EdcErrorStatus

EDC error status. Used to indicate EDC access error type.

Enumeration values:

<i>CSL_EDC_DERR</i>	EDC error status - data fetch parity error
<i>CSL_EDC_IERR</i>	EDC error status - program fetch parity error
<i>CSL_EDC_DMAERR</i>	EDC error status - DMA read parity error

16.4.6 CSL_EdcNumErrors

enum CSL_EdcNumErrors

Indicates the number of EDC bit errors. Used to indicate number of EDC bit errors or if bit error is in parity value.

Enumeration values:

<i>CSL_EDC_1BIT</i>	EDC number of bit errors - single bit error
<i>CSL_EDC_2BIT</i>	EDC number of bit errors - double bit error
<i>CSL_EDC_PERROR</i>	EDC number of bit errors - error in parity value

16.4.7 CSL_EdcUmap

enum CSL_EdcUmap

UMAP Specifier for EDC. Used to indicate which of the UMAPs the page enables are to be applied to.

Enumeration values:

<i>CSL_EDC_UMAP0</i>	EDC apply page enables to UMAP0 only
<i>CSL_EDC_UMAP1</i>	EDC apply page enables to UMAP1 only
<i>CSL_EDC_UMAPBOTH</i>	EDC apply page enables to both UMAP0 and UMAP1

16.4.8 CSL_EdcAddrL2way

enum CSL_EdcAddrL2way

L2 way specifier for EDC error. Provides the L2 way for the Address of the detected error.

Enumeration values:

<i>CSL_EDC_L2WAY_0</i>	L2 way 0
<i>CSL_EDC_L2WAY_1</i>	L2 way 1
<i>CSL_EDC_L2WAY_2</i>	L2 way 2
<i>CSL_EDC_L2WAY_3</i>	L2 way 3

16.4.9 CSL_EdcAddrSram

enum CSL_EdcAddrSram

EDC error in SRAM or Cache. Specifies whether the EDC error was located in the SRAM or Cache.

Enumeration values:

<i>CSL_EDC_CACHE</i>	EDC error is in cache
<i>CSL_EDC_SRAM</i>	EDC error is in SRAM

Chapter 17

PLLC MODULE

Topics

<u>17. 1 Overview</u>
<u>17. 2 Functions</u>
<u>17. 3 Data Structures</u>
<u>17. 4 Enumerations</u>
<u>17. 5 Macros</u>

17.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within PLLC module.

The PLL controller features software-configurable PLL multiplier controller (PLLM), dividers, and reset controller. The PLL controller offers flexibility and convenience by way of software-configurable multiplier and dividers to modify the input signal internally. The resulting clock outputs are passed to the DSP core, peripherals, and other modules inside the DSP.

- The input reference clocks to the PLL controller:
 - CLKIN: output signal from external oscillator
- The resulting output clocks from the PLL controller:
 - AUXCLK: internal clock output signal directly from CLKIN.
 - SYSCLKn: internal clock output of divider Dn.

17.2 Functions

This section lists the functions available in the PLLC module.

17.2.1 CSL_pllInit

CSL_Status CSL_pllInit ([CSL_PllContext](#) * *pContext*)

Description

This is the initialization function for the pll CSL. The function must be called before calling any other API from this CSL. This function is idem-potent. Currently, the function just return status CSL_SOK, without doing anything.

Arguments

pContext Pointer to module-context. As PLLC doesn't have any context based information user is expected to pass NULL.

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_pllInit(NULL);
```

17.2.2 CSL_pllOpen

[CSL_PllHandle](#) CSL_pllOpen ([CSL_PllObj](#) * *pPllObj*,
CSL_InstNum *pllNum*,
[CSL_PllParam](#) * *pPllParam*,
CSL_Status * *pStatus*
)

Description

This function returns the handle to the PLLC instance. This handle is passed to all other CSL APIs.

Arguments

pPllObj Pointer to PLLC object.

pllNum Instance of PLLC to be opened.

pPllcParam Module specific parameters.

pStatus Status of the function call

Return Value

CSL PllcHandle

Valid PLLC handle will be returned if status value is equal to CSL_SOK.

Pre Condition

None

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid PLLC handle is returned
- CSL_ESYS_FAIL - The PLLC instance is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

2. PLLC object structure is populated.

Modifies

1. The status variable
2. PLLC object structure

Example

```
CSL_Status          status;
CSL_PllcObj         pllObj;
CSL_PllcHandle      hPllc;
...

hPllc = CSL_pllOpen(&pllObj, CSL_PLLC_0, NULL, &status);
...
```

17.2.3 CSL_pllClose

CSL_Status CSL_pllClose ([CSL_PllCHandle](#) *hPllc*)

Description

This function closes the specified instance of PLLC.

Arguments

hPllc	Handle to the PLLC
-------	--------------------

Return Value

CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Example

```
CSL_PllcHandle    hPllc;
CSL_Status        status;
...
```

```
status = CSL_pllClose(hPllc, &hwSetup);
```

17.2.4 CSL_pllHwSetup

```
CSL_Status CSL_pllHwSetup      ( CSL\_PllCHandle          hPllc,  
                                CSL\_PllcHwSetup *    hwSetup  
                                )
```

Description

It configures the pll registers as per the values passed in the hardware setup structure.

Arguments

hPllc	Handle to the PLLC
hwSetup	Pointer to hardware setup structure

Return Value

CSL_Status

- CSL_SOK – Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS – Hardware structure is not properly initialized

Pre Condition

None

Post Condition

PLLC registers are configured according to the hardware setup parameters

Modifies

PLLC registers.

Example

```
CSL_PllCHandle    hPllc;
CSL_PllcObj       pllObj;
CSL_PllcHwSetup   hwSetup;
CSL_Status        status;
...

hPllc = CSL_pllOpen(&pllObj, CSL_PLLC_0, NULL, &status);
...

status = CSL_pllHwSetup(hPllc, &hwSetup);
```

17.2.5 CSL_pllCwControl

```
CSL_Status CSL_pllCwControl      ( CSL\_PllCHandle          hPllc,
                                   CSL\_PllCwControlCmd cmd,
                                   void *          arg
                                   )
```

Description

Takes a command of PLLC with an optional argument and implements it.

Arguments

<code>hPllc</code>	Handle to the PLLC instance
<code>cmd</code>	The command to this API indicates the action to be taken on PLLC.
<code>arg</code>	An optional argument.

Return Value

CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None.

Post Condition

None.

Modifies

The hardware registers of PLLC.

Example

```
CSL_PllCHandle    hPllc;
CSL_PllCwControlCmd cmd;
void              arg;
...

status = CSL_pllCwControl (hPllc, cmd, &arg);
```

17.2.6 CSL_pllGetHwStatus

```
CSL_Status CSL_pllGetHwStatus      ( CSL\_PllcHandle          hPllc,
                                     CSL\_PllcHwStatusQuery query,
                                     void *                          response
                                     )
```

Description

Gets the status of the different operations of PLLC.

Arguments

<code>hPllc</code>	Handle to the PLLC instance
<code>query</code>	The query to this API of PLLC which indicates the status to be returned.
<code>response</code>	Placeholder to return the status.

Return Value

CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None

Post Condition

None

Modifies

None.

Example

```
CSL_PllcHandle    hPllc;
CSL_PllcHwStatusQuery query;
void              reponse;
...

status = CSL_pllGetHwStatus (hPllc, query, &response);
```

17.2.7 CSL_pllHwSetupRaw

```
CSL_Status CSL_pllHwSetupRaw      ( CSL\_PllcHandle          hPllc,
                                     CSL\_PllcConfig *      config
                                     )
```

Description

This function initializes the device registers with the register-values provided through the config data structure. This configures registers based on a structure of register values, as compared to CSL_pllCwSetup (), which configures registers based on structure of bit field values.

Arguments

hpllC	Handle to the PLLC instance
config	Pointer to config structure

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration is not properly initialized

Pre Condition

None

Post Condition

The registers of the specified PLLC instance will be setup according to input configuration structure values.

Modifies

Hardware registers of the specified PLLC instance.

Example

```
CSL_PllcHandle    hPllc;
CSL_PllcConfig    config = CSL_PLLC_CONFIG_DEFAULTS;
CSL_Status        status;
...

config.PLLCTL      = 0x00000040u;
config.PLLM        = 0x00000000u;
config.PLLDIV1     = 0x00000000u;
config.PLLCMD      = 0x00000001u;

status = CSL_pllCwSetupRaw (hPllc, &config);
```

17.2.8 CSL_pllCwGetHwSetup

CSL_Status **CSL_pllCwGetHwSetup** ([CSL_PllcHandle](#) *hPllc*,
[CSL_PllcHwSetup](#) * *hwSetup*
)

Description

It retrieves the hardware setup parameters of the PLLC specified by the given handle.

Arguments

hPllc	Handle to the PLLC
hwSetup	Pointer to the hardware setup structure

Return Value CSL_Status

-
- CSL_SOK - Retrieving the hardware setup parameters is successful
 - CSL_ESYS_BADHANDLE - The handle passed is invalid
 - CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

hwSetup variable

Example

```
CSL_PllcHandle    hPllc;
CSL_PllcHwSetup   hwSetup;
...

status = CSL_pllGetHwSetup(hPllc, &hwSetup);
```

17.2.9 CSL_pllGetBaseAddress

```
CSL_Status CSL_pllGetBaseAddress ( CSL_InstNum      pllNum,
                                   CSL\_PllcParam *  pPllcParam,
                                   CSL\_PllcBaseAddress * pBaseAddress
                                   )
```

Description

This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_pllOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

pllNum	Specifies the instance of the PLLC to be opened.
pPllcParam	Module specific parameters
pBaseAddress	Pointer to base address structure containing base address details.

Return Value

CSL_Status

- CSL_SOK - Open call is successful
- CSL_ESYS_FAIL - The instance number is invalid.
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status          status;  
CSL_PllcBaseAddress baseAddress;  
...  
status = CSL_pllGetBaseAddress(CSL_PLLC_0, NULL, &baseAddress);  
...
```

17.3 Data Structures

This section lists the data structures available in the PLLC module.

17.3.1 CSL_PllcObj

Detailed Description

This object contains the reference to the instance of PLLC opened using the *CSL_pllOpen()*. The pointer to this is passed to all PLLC CSL APIs.

Field Documentation

CSL_InstNum CSL_PllcObj::pllNum

This is the instance of PLLC being referred to by this object

CSL_PllcRegsOnly CSL_PllcObj::regs

This is a pointer to the registers of the instance of PLLC referred to by this object

17.3.2 CSL_PllcConfig

Detailed Description

Config-structure. Used to configure the PLLC using *CSL_pllHwSetupRaw()*.

Field Documentation

Volatile Uint32 CSL_PllcConfig::RSTDEF

Reset definition register

volatile Uint32 CSL_PllcConfig::PLLCTL

PLL Control register

Volatile Uint32 CSL_PllcConfig::OCSSEL

OBSCLK Select register

Volatile Uint32 CSL_PllcConfig::SECCTL

PLL Secondary Control register

volatile Uint32 CSL_PllcConfig::PLLM

PLL Multiplier Control register

volatile Uint32 CSL_PllcConfig::PLLDIV1

PLL Controller Divider 1 register (for SYSCLK0)

volatile Uint32 CSL_PllcConfig::PLLDIV2

PLL Controller Divider 2 register (for SYSCLK1)

volatile Uint32 CSL_PllcConfig::PLLDIV3

PLL Controller Divider 3 register (for SYSCLK2)

volatile Uint32 CSL_PllcConfig::OSCDIV1

Oscillator Divider1 register

Volatile Uint32 CSL_PllcConfig::BPDIV

Bypass Divider register

Volatile Uint32 CSL_PllcConfig::WAKEUP

Wakeup register

Volatile Uint32 CSL_PllcConfig::PLLCMD

PLL Controller Command register

Volatile Uint32 CSL_PllcConfig::CKEN

Clock Enable Control register

volatile Uint32 CSL_PllcConfig::PLLDIV4

PLL Controller Divider 4 register (for SYSCLK3)

volatile Uint32 CSL_PllcConfig::PLLDIV5

PLL Controller Divider 5 register (for SYSCLK4)

volatile Uint32 CSL_PllcConfig::PLLDIV6

PLL Controller Divider 6 register (for SYSCLK5)

volatile Uint32 CSL_PllcConfig::PLLDIV11

PLL Controller Divider 11 register (for SYSCLK10)

volatile Uint32 CSL_PllcConfig::PLLDIV13

PLL Controller Divider 13 register (for SYSCLK12)

volatile Uint32 CSL_PllcConfig::PLLDIV14

PLL Controller Divider 14 register (for SYSCLK13)

17.3.3 CSL_PllcContext

Detailed Description

Module specific context information. Present implementation of PLLC CSL doesn't have any context information.

Field Documentation
Uint16 CSL_PllcContext::contextInfo

Context information of PLLC CSL. The declaration is just a placeholder for future implementation.

17.3.4 CSL_PllcHwSetup

Detailed Description

Input parameters for setting up PLL Controller. Used to put PLLC known useful state /sa CSL_PLLC_DIVEN_DEFINE

Field Documentation
CSL_BitMask32 CSL_PllcHwSetup::divEnable

Divider Enable/Disable.

Parameters:*CSL_BitMask32*

Uint32 CSL_PllcHwSetup::pllM

Pre-Divider.

Parameters:Uint32

Uint32 CSL_PllcHwSetup::pllDiv0

PLL Divider 0.

Parameters: *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv1

PLL Divider 1.

Parameters: *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv2

 PLL Divider 2. **Parameters:** *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv3

 PLL Divider 3. **Parameters:** *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv4

 PLL Divider 4. **Parameters:** *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv5

 PLL Divider 5. **Parameters:** *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv10

PLL Divider 10.

Parameters: *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv12

PLL Divider 12.

Parameters: *Uint32*
Uint32 CSL_PllcHwSetup::pllDiv13

PLL Divider 13.

Parameters: *Uint32*
Uint32 CSL_PllcHwSetup::oscDiv1

 Oscillator Divider 1. **Parameters:** *Uint32*
void* CSL_PllcHwSetup::extendSetup

 Setup that can be used for future implementation. **Parameters:** *void**

17.3.5 CSL_PllcParam

Detailed Description

Module specific parameters. Present implementation of PLLC CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_PllcParam::flags

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation.

17.3.6 CSL_PllcBaseAddress

Detailed Description

This structure contains the base-address information for the peripheral instance of the PLLC.

Field Documentation

CSL_PllcRegsOvly CSL_PllcBaseAddress::regs

Base-address of the configuration registers of the peripheral

17.3.7 CSL_PllcDivRatio

Detailed Description

Input parameters for setting up PLL Divide ratio.

Field Documentation

UInt32 CSL_PllcDivRatio::divNum

Divider number

UInt32 CSL_PllcDivRatio::divRatio

Divider Ratio

17.3.8 CSL_PllcDivideControl

Detailed Description

Input parameters for enabling PLL Divide ratio

Field Documentation

[CSL_PllcDivCtrl](#) **CSL_PllcDivideControl::divCtrl**

Divider Control (Enable/Disable)

UInt32 CSL_PllcDivideControl::divNum

Divider Number

17.4 Enumerations

This section lists the enumerations available in the PLLC module.

17.4.1 CSL_PllcDivCtrl

enum CSL_PllcDivCtrl

Enums for PLL divide enable/ disable

Enumeration values:

CSL_PLLC_PLLDIV_DISABLE

PLL Divider Disable

CSL_PLLC_PLLDIV_ENABLE

PLL Divider Enable

17.4.2 CSL_PllcHwControlCmd

enum CSL_PllcHwControlCmd

This is the set of commands that are passed to the *CSL_pllHwControl()* with an optional argument type-casted to *void**. The arguments to be passed with each enumeration (if any) are specified next to the enumeration.

Enumeration values:

CSL_PLLC_CMD_PLLCONTROL

Control PLL based on the bits set in the input argument. **Parameters:**

CSL_BitMask32

Returns:

CSL_SOK

See also:

CSL_PLLC_CTRL_DEFINE

CSL_PLLC_CMD_CLOCK_ENABLE

Enable the clocks as specified by input bitmask.

Parameters:

CSL_BitMask32

Returns:

CSL_SOK

See also:

CSL_PLLC_CLKEN_DEFINE

CSL_PLLC_CMD_CLOCK_DISABLE

Disable the clocks specified by input bitmask.

Parameters:

CSL_BitMask32

Returns:

CSL_SOK

See also:

CSL_PLLC_CLKEN_DEFINE

CSL_PLLC_CMD_DEFINE_RESET

Disable the clocks specified by input bitmask.

Parameters:

CSL_PllcResetDef

Returns:

CSL_SOK

See also:

CSL_PllcResetDef

<i>CSL_PLLC_CMD_SET_PLLM</i>	Set PLL multiplier value. Parameters: <i>Uint32</i> Returns: CSL_SOK
<i>CSL_PLLC_CMD_SET_OSCRATIO</i>	Set oscillator divide ratio. Parameters: <i>Uint32</i> Returns: CSL_SOK
<i>CSL_PLLC_CMD_SET_PLLRATIO</i>	Set PLL divide ratio. Parameters: <i>CSL_PllcDivRatio</i> Returns: CSL_SOK
<i>CSL_PLLC_CMD_OSCDIV_CONTROL</i>	Enable/disable oscillator divider. Parameters: <i>CSL_PllcOscDivCtrl</i> Returns: CSL_SOK See also: CSL_PllcOscDivCtrl
<i>CSL_PLLC_CMD_PLLDIV_CONTROL</i>	Enable/disable PLL divider. Parameters: <i>CSL_PllcDivideControl</i> Returns: CSL_SOK See also: CSL_PllcOscDivCtrl
<i>CSL_PLLC_CMD_WAKEUP</i>	Enable/disable Wake Up functionality of wakeup pins. Parameters: <i>CSL_BitMask16</i> Returns: CSL_SOK

17.4.3 CSL_PllcHwStatusQuery

enum CSL_PllcHwStatusQuery

This is used to get the status of different operations. The status is returned in the argument passed.

Enumeration values:

<i>CSL_PLLC_QUERY_PID</i>	Queries PLL Control Peripheral ID. Parameters: <i>(Uint32*)</i> Returns: CSL_SOK
---------------------------	--

<i>CSL_PLLC_QUERY_STATUS</i>	<p>Queries PLL Controller Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_STATUS_DEFINE</p>
<i>CSL_PLLC_QUERY_DIVRATIO_CHANGE</i>	<p>Queries PLLDIV Modified Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_DCHANGESTAT_DEFINE</p>
<i>CSL_PLLC_QUERY_CLKSTAT</i>	<p>Queries Clock Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_CLKSTAT_DEFINE</p>
<i>CSL_PLLC_QUERY_SYSCLKSTAT</i>	<p>Queries PLL SYSCLK Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_SYSCLKSTAT_DEFINE</p>
<i>CSL_PLLC_QUERY_CLKENSTAT</i>	<p>Queries CLK Enable Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_CLKEN_DEFINE</p>
<i>CSL_PLLC_QUERY_RESETSTAT</i>	<p>Queries Reset Type Status.</p> <p>Parameters: (<i>CSL_BitMask32*</i>)</p> <p>Returns: CSL_SOK</p> <p>See also: CSL_PLLC_RESETSTAT_DEFINE</p>
<i>CSL_PLLC_QUERY_EFUSEERR</i>	<p>Queries Fuse Farm Error Status.</p> <p>Parameters: (<i>Uint32*</i>)</p> <p>Returns: CSL_SOK</p>

17.5 Macros

PLL Controller Status

#define CSL_PLLC_STATUS_GO CSL_FMKT (PLL_CPLLSTAT_GOSTAT, INPROG)

Set when GO operation (divide-ratio change and clock alignment) is in progress

#define CSL_PLLC_STATUS_LOCK CSL_FMKT (PLL_CPLLSTAT_LOCK, YES)

Set when PLL core is locked

#define CSL_PLLC_STATUS_STABLE CSL_FMKT (PLL_CPLLSTAT_STABLE, YES)

Set when OSCIN/CLKIN is assumed to be stable

PLL Divider Ratio Modified Status

#define CSL_PLLC_DCHANGESTAT_SYS10 CSL_FMKT (PLL_CCHANGE_SYS10, YES)

SYSCLK10 divide ratio is modified

#define CSL_PLLC_DCHANGESTAT_SYS12 CSL_FMKT (PLL_CCHANGE_SYS12, YES)

SYSCLK12 divide ratio is modified

#define CSL_PLLC_DCHANGESTAT_SYS13 CSL_FMKT (PLL_CCHANGE_SYS13, YES)

SYSCLK13 divide ratio is modified

PLL Clock Status

#define CSL_PLLC_CLKSTAT_AUXON CSL_FMKT (PLL_CKSTAT_AUXEN, ON)

AUXCLK is ON

#define CSL_PLLC_CLKSTAT_BPON CSL_FMKT (PLL_CKSTAT_BPON, ON)

SYSCLKBP is ON

#define CSL_PLLC_CLKSTAT_OBSON CSL_FMKT (PLL_CKSTAT_OBSEN, ON)

OBSCCLK is ON

PLL Clock Enable Status

#define CSL_PLLC_CLKEN_AUXEN CSL_FMKT (PLL_CKEN_AUXEN, ENABLE)

AUXCLK enable

#define CSL_PLLC_CLKEN_OBSEN CSL_FMKT (PLL_CKEN_OBSEN, ENABLE)

OBSCCLK enable

PLL SYSCLK Status

#define CSL_PLLC_SYSCCLKSTAT_SYS1ON CSL_FMKT (PLL_CKSTAT_SYS1ON, ON)

SYSCLK0 is ON

#define CSL_PLLC_SYSCCLKSTAT_SYS2ON CSL_FMKT (PLL_CKSTAT_SYS2ON, ON)

SYSCLK1 is ON

#define CSL_PLLC_SYSCCLKSTAT_SYS3ON CSL_FMKT (PLL_CKSTAT_SYS3ON, ON)

SYSClk2 is ON

#define CSL_PLLC_SYSClkSTAT_SYS4ON CSL_FMKT (PLLC_CKSTAT_SYS4ON, ON)
SYSClk3 is ON

#define CSL_PLLC_SYSClkSTAT_SYS5ON CSL_FMKT (PLLC_CKSTAT_SYS5ON, ON)
SYSClk4 is ON

#define CSL_PLLC_SYSClkSTAT_SYS6ON CSL_FMKT (PLLC_CKSTAT_SYS6ON, ON)
SYSClk5 is ON

#define CSL_PLLC_SYSClkSTAT_SYS7ON CSL_FMKT (PLLC_CKSTAT_SYS7ON, ON)
SYSClk6 is ON

#define CSL_PLLC_SYSClkSTAT_SYS8ON CSL_FMKT (PLLC_CKSTAT_SYS8ON, ON)
SYSClk7 is ON

#define CSL_PLLC_SYSClkSTAT_SYS9ON CSL_FMKT (PLLC_CKSTAT_SYS9ON, ON)
SYSClk8 is ON

#define CSL_PLLC_SYSClkSTAT_SYS10ON CSL_FMKT (PLLC_CKSTAT_SYS10ON, ON)
SYSClk9 is ON

#define CSL_PLLC_SYSClkSTAT_SYS11ON CSL_FMKT (PLLC_CKSTAT_SYS11ON, ON)
SYSClk10 is ON

#define CSL_PLLC_SYSClkSTAT_SYS12ON CSL_FMKT (PLLC_CKSTAT_SYS12ON, ON)
SYSClk11 is ON

#define CSL_PLLC_SYSClkSTAT_SYS13ON CSL_FMKT (PLLC_CKSTAT_SYS13ON, ON)
SYSClk12 is ON

#define CSL_PLLC_SYSClkSTAT_SYS14ON CSL_FMKT (PLLC_CKSTAT_SYS14ON, ON)
SYSClk13 is ON

PLLC Last Reset Status

#define CSL_PLLC_RESETSTAT_MRST CSL_FMKT (PLLC_RSTYPE_MRST, YES)
Maximum Reset

#define CSL_PLLC_RESETSTAT_POR CSL_FMKT (PLLC_RSTYPE_POR, YES)
Power On Reset

#define CSL_PLLC_RESETSTAT_SRST CSL_FMKT (PLLC_RSTYPE_SRST, YES)
System/Chip Reset

#define CSL_PLLC_RESETSTAT_XWRST CSL_FMKT (PLLC_RSTYPE_XWRST, YES)
External Warm Reset

PLLC Control Mask

#define CSL_PLLC_CTRL_ALIGN_PHASE (CSL_FMKT (PLLC_PLLCMD_GOSET, SET)<< 16)

A write of 1 to this bit signifies that the new divide ratios in PLLDIV[1:n] are taken into account at the nearest possible rising edge to phase align the clocks. The actual SYSCLKx to be aligned are selected in register ALNCTL

#define CSL_PLLC_CTRL_ASSERT_DIS CSL_FMKT (PLLC_PLLCTL_PLLRST, YES)
 PLL Disable Asserted

#define CSL_PLLC_CTRL_BYPASS CSL_FMKT (PLLC_PLLCTL_PLEN, BYPASS)
 PreDiv, PLL, and PostDiv are bypassed. SYSCLK divided down directly from input reference clock refclk

#define CSL_PLLC_CTRL_ENABLE CSL_FMKT (PLLC_PLLCTL_PLEN, PLL)
 PLL is used. SYSCLK divided down from PostDiv output

#define CSL_PLLC_CTRL_OPERATIONAL CSL_FMKT (PLLC_PLLCTL_PLLPWRDN, NO)
 Selected PLL Operational

#define CSL_PLLC_CTRL_POWERDOWN CSL_FMKT (PLLC_PLLCTL_PLLPWRDN, YES)
 Selected PLL Placed In Power Down State

#define CSL_PLLC_CTRL_RELEASE_DIS CSL_FMKT (PLLC_PLLCTL_PLDIS, NO)
 PLL Disable Released

#define CSL_PLLC_CTRL_RELEASE_RESET CSL_FMKT (PLLC_PLLCTL_PLLRST, NO)
 PLL Reset Released

#define CSL_PLLC_CTRL_RESET CSL_FMKT (PLLC_PLLCTL_PLLRST, YES)
 PLL Reset Asserted

#define CSL_PLLC_CTRL_SELECT_CLKIN CSL_FMKT (PLLC_PLLCTL_CLKMODE, CLKIN)
 clkin_pi is the reference clock

#define CSL_PLLC_CTRL_SELECT_OSCIN CSL_FMKT (PLLC_PLLCTL_CLKMODE, OSCIN)
 oscin_pi is the reference clock

PLLC Divider Enable

#define CSL_PLLC_DIVEN_OSCDIV1 (1 << 1)
 Oscillator Divider OD1 Enable

#define CSL_PLLC_DIVEN_PLLDIV0 (1 << 2)
 Enable divider D0 for SYSCLK0

#define CSL_PLLC_DIVEN_PLLDIV1 (1 << 3)
 Enable divider D1 for SYSCLK1

#define CSL_PLLC_DIVEN_PLLDIV2 (1 << 4)
 Enable divider D2 for SYSCLK2

#define CSL_PLLC_DIVEN_PLLDIV3 (1 << 5)
 Enable divider D3 for SYSCLK3

```
#define CSL_PLLC_DIVEN_PLLDIV4 (1 << 6)
```

```
    Enable divider D4 for SYSCLK4
```

```
#define CSL_PLLC_DIVEN_PLLDIV5 (1 << 7)
```

```
    Enable divider D5 for SYSCLK5
```

```
#define CSL_PLLC_DIVEN_PLLDIV10 (1 << 11)
```

```
    Enable divider D10 for SYSCLK10
```

```
#define CSL_PLLC_DIVEN_PLLDIV12 (1 << 13)
```

```
    Enable divider D12 for SYSCLK12
```

```
#define CSL_PLLC_DIVEN_PLLDIV13 (1 << 14)
```

```
    Enable divider D13 for SYSCLK13
```

Divider Select for SYSCLKs

```
#define CSL_PLLC_DIVSEL_PLLDIV0 (1)
```

```
    Divider D1 for SYSCLK0
```

```
#define CSL_PLLC_DIVSEL_PLLDIV1 (2)
```

```
    Divider D1 for SYSCLK1
```

```
#define CSL_PLLC_DIVSEL_PLLDIV2 (3)
```

```
    Divider D2 for SYSCLK2
```

```
#define CSL_PLLC_DIVSEL_PLLDIV3 (4)
```

```
    Divider D3 for SYSCLK3
```

```
#define CSL_PLLC_DIVSEL_PLLDIV4 (5)
```

```
    Divider D4 for SYSCLK4
```

```
#define CSL_PLLC_DIVSEL_PLLDIV5 (6)
```

```
    Divider D5 for SYSCLK5
```

```
#define CSL_PLLC_DIVSEL_PLLDIV10 (11)
```

```
    Divider D10 for SYSCLK10
```

```
#define CSL_PLLC_DIVSEL_PLLDIV12 (13)
```

```
    Divider D12 for SYSCLK12
```

```
#define CSL_PLLC_DIVSEL_PLLDIV13 (14)
```

```
    Divider D13 for SYSCLK13
```

PLL Default HwSetup Structure

```
#define CSL_PLLC_HWSETUP_DEFAULTS
```

```
    Value:
```

```
{ \
    0, \
    0, \
    0, \
}
```

```

    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    NULL
}

```

PLL Default Config Structure

```

#define CSL_PLL_CONFIG_DEFAULTS
Value:

```

```

{ \
    CSL_PLL_RSTDEF_RESETVAL, \
    CSL_PLL_PLLCTL_RESETVAL, \
    CSL_PLL_OCSEL_RESETVAL, \
    CSL_PLL_SECCTL_RESETVAL, \
    CSL_PLL_PLLM_RESETVAL, \
    CSL_PLL_PLLDIV1_RESETVAL, \
    CSL_PLL_PLLDIV2_RESETVAL, \
    CSL_PLL_PLLDIV3_RESETVAL, \
    CSL_PLL_OSCDIV1_RESETVAL, \
    CSL_PLL_BP_DIV_RESETVAL, \
    CSL_PLL_WAKEUP_RESETVAL, \
    CSL_PLL_PLLCMD_RESETVAL, \
    CSL_PLL_CKEN_RESETVAL, \
    CSL_PLL_PLLDIV4_RESETVAL, \
    CSL_PLL_PLLDIV5_RESETVAL, \
    CSL_PLL_PLLDIV6_RESETVAL, \
    CSL_PLL_PLLDIV11_RESETVAL, \
    CSL_PLL_PLLDIV13_RESETVAL, \
    CSL_PLL_PLLDIV14_RESETVAL \
}

```

Chapter 18

SRIO MODULE

Topics

<u>18. 1 Overview</u>
<u>18. 2 Functions</u>
<u>18. 3 Data Structures</u>
<u>18. 4 Enumerations</u>
<u>18. 5 Macros</u>
<u>18.6 Typedefs</u>

18.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within SRIO module.

RapidIO™ is a non-proprietary high-bandwidth system level interconnect, it is a packet-switched interconnect intended primarily as an intra-system interface for chip-to-chip and board-to-board communications at Gigabyte-per-second performance levels. Uses for the architecture can be found in connected microprocessors, memory, and memory mapped I/O devices that operate in networking equipment, memory subsystems, and general purpose computing.

Features Supported in SRIO:

- RapidIO Interconnect Specification V1.2 compliant, Errata 1.2
- LP-Serial Specification V1.2 compliant
- 4X Serial RapidIO with auto-negotiation to 1X port, optional operation for (4) 1X ports
- Integrated Clock Recovery with TI SERDES
- Hardware Error handling including CRC
- Differential CML signaling supporting AC and DC coupling
- Support for 1.25, 2.5, and 3.125Gbps rates
- Power-down option for unused ports
- Read, write, write w/response, streaming write, out-going Atomic, maintenance operations
- Shall generate interrupts to the CPU (Doorbell packets and Internal scheduling)
- Support for 8b and 16b device ID
- Support for receiving 34b addresses
- Support for generating 34b, 50b, and 66b addresses
- Support for data sizes: byte, half-word, word, double-word
- Defined as Big Endian
- Direct IO transfers
- Message passing transfers
- Data payloads to 256B
- Single message generation up to 16 packets
- Elastic Store FIFO for clock domain handoff
- Short Run and Long Run compliant
- CBA3.0 compliant – generate DMA BUS commands and data transfers
- Support for Error Management Extensions
- Support for Congestion Control Extensions
- Support for one multi-cast ID

The SRIO CSL supports functional layer API doesnot support the functional layer API for message passing data transfer.

18.2 Functions

This section lists the functions available in the SRIO module.

18.2.1 CSL_srioInit

CSL_Status CSL_srioInit ([CSL_SrioContext](#) * *pContext*)

Description

This is the initialization function for the SRIO CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Pointer to module-context. As SRIO doesn't have any context based information user is expected to pass NULL.

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

The CSL for SRIO is initialized

Modifies

None

Example

```
CSL_srioInit(NULL);
```

18.2.2 CSL_srioOpen

[CSL_SrioHandle](#) CSL_srioOpen ([CSL_SrioObj](#) * *pSrioObj*,
CSL_InstNum *srioNum*,
[CSL_SrioParam](#) * *pSrioParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the SRIO instance and returns a handle to the instance. The handle returned by this call is input as an essential argument for the rest of the APIs described for this module.

Arguments

pSrioObj Pointer to SRIO object.

srioNum	Instance of SRIO CSL to be opened. There is one instance of the SRIO available. So, the value for this parameter will be CSL_SRIO always.
pSrioParam	Module specific parameters.
pStatus	Status of the function call

Return Value CSL_SrioHandle

Valid SRIO handle will be returned if status value is equal to CSL_SOK.

Pre Condition

The SRIO must be successfully initialized via CSL_srioInit () before calling this function.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid SRIO handle is returned
- CSL_ESYS_FAIL - The SRIO instance is invalid
- CSL_ESYS_INVPARAMS – Invalid parameters.

2. SRIO object structure is populated.

Modifies

1. The status variable
2. SRIO object structure

Example

```

CSL_Status      status;
CSL_SrioObj     srioObj;
CSL_SrioHandle  hSrio;
...
hSrio = CSL_srioOpen(&srioObj, CSL_SRIO, NULL, &status);
...

```

18.2.3 CSL_srioClose

CSL_Status CSL_srioClose ([CSL_SrioHandle](#) *hSrio*)

Description

This function closes the specified instance of SRIO.

Arguments

hSrio Handle to the SRIO

Return Value CSL_Status

- CSL_SOK - SRIO is closed successfully
- CSL_ESYS_BADHANDLE - The handle passed is invalid

Pre Condition

Both *CSL_srioInit()* and *CSL_srioOpen()* must be called successfully in order before calling *CSL_srioClose()*.

Post Condition

The SRIO CSL APIs can not be called until the SRIO CSL is reopened again using *CSL_srioOpen()*.

Modifies

The peripheral data object.

Example

```
CSL_SrioHandle hSrio;
CSL_Status     status;
...
status = CSL_srioClose(hSrio);
```

18.2.4 CSL_srioHwSetup

```
CSL_Status CSL_srioHwSetup ( CSL\_SrioHandle          hSrio,
                             CSL\_SrioHwSetup        *hwSetup
                             )
```

Description

It configures the SRIO instance registers as per the values passed in the hardware setup structure.

Arguments

<i>hSrio</i>	Handle to the SRIO instance
<i>hwSetup</i>	Pointer to hardware setup structure

Return Value *CSL_Status*

- *CSL_SOK* - Hardware setup successful
- *CSL_ESYS_BADHANDLE* - Invalid handle
- *CSL_ESYS_INVPARAMS* - Hardware structure is not properly initialized

Pre Condition

Both *CSL_srioInit()* and *CSL_srioOpen()* must be called successfully in order before calling this API.

Post Condition

The specified instance will be setup according to value passed.

Modifies

Hardware registers for the specified instance.

Example

```
CSL_SrioHandle hSrio;
CSL_SrioObj    srioObj;
CSL_SrioHwSetup hwSetup =
CSL_SRIO_HWSETUP_DEFAULTS;
```

```

CSL_Status          status;
CSL_SrioControlSetup    periSetup;
CSL_SrioBlkEn          blockSetup;
CSL_SrioPktFwdCntl      pktFwdSetup;

hSrio = CSL_srioOpen (&srioObj, CSL_SRIO, NULL, &status);

status = CSL_srioHwSetup(hSrio, &hwSetup);

```

18.2.5 CSL_srioHwControl

```

CSL_Status CSL_srioHwControl      ( CSL\_SrioHandle          hSrio,
                                   CSL\_SrioHwControlCmd      cmd,
                                   void *                      arg
                                   )

```

Description

This function performs various control operations on the SRIO instance, based on the command passed.

Arguments

<code>hSrio</code>	Handle to the SRIO instance
<code>cmd</code>	Operation to be performed on the SRIO
<code>arg</code>	Argument specific to the command

Return Value CSL_Status

- CSL_SOK - Command execution successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

Both `CSL_srioInit()` and `CSL_srioOpen()` must be called successfully in order before calling this API.

Post Condition

Registers of the SRIO instance are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

Registers determined by the command

Example

```

CSL_SrioHandle  hSrio;
CSL_SrioPortData clearData;
CSL_Status      status;
Uint32          mask;
Uint8           index;
...
// for clearing LSU interrupts status [0..3]

```

```

index = 1;
mask = CSL_SRIO_LSU_INTR3 | CSL_SRIO_LSU_INTR2 |
      CSL_SRIO_LSU_INTR1 | CSL_SRIO_LSU_INTR0;
clearData.index = index;
clearData.data = mask;
...
CSL_srioHwControl(hSrio, CSL_SRIO_CMD_LSU_INTR_CLEAR, &clearData);
...

```

18.2.6 CSL_srioGetHwStatus

```

CSL_Status CSL_srioGetHwStatus ( CSL\_SrioHandle           hSrio,
                                CSL\_SrioHwStatusQuery       query,
                                void *                       response
                                )

```

Description

This function is used to get the value of various parameters of the SRIO instance. The value returned depends on the query passed.

Arguments

<code>hSrio</code>	Handle to the SRIO instance
<code>query</code>	Query to be performed
<code>response</code>	Pointer to buffer to return the data requested by the query passed

Return Value CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Query command not supported
- CSL_ESYS_INVPARAMS - Invalid parameters

Pre Condition

Both *CSL_srioInit()* and *CSL_srioOpen()* must be called successfully in order before calling this API.

Post Condition

Data requested by the query is returned through the variable "response".

Modifies

The input argument "response" is modified.

Example

```

CSL_Status      status;
CSL_SrioHandle  hSrio;
CSL_SrioPidNumber response;
...

```



```

Status=CSL_srioGetHwStatus(hSrio,
                           CSL_SRIO_QUERY_PID_NUMBER,
                           &response);
...

```

18.2.7 CSL_srioHwSetupRaw

```

CSL_Status CSL_srioHwSetupRaw      ( CSL\_SrioHandle          hSrio,
                                     CSL\_SrioConfig *        config
                                     )

```

Description

This function initializes the device registers with the register-values provided through the config data structure. This configures registers based on a structure of register values, as compared to CSL_SrioHwSetup, which configures registers based on structure of bit field values.

Arguments

<code>hSrio</code>	Handle to the SRIO instance
<code>config</code>	Pointer to the config structure containing the device register values

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration structure pointer is not properly initialized

Pre Condition

Both `CSL_srioInit()` and `CSL_srioOpen()` must be called successfully in order before calling this API.

Post Condition

The registers of SRIO will be setup according to the values passed through the config structure.

Modifies

Hardware registers of SRIO

Example

```

CSL_SrioHandle hSrio;
CSL_SrioConfig config = CSL_SRIO_CONFIG_DEFAULTS;
CSL_Status      status;
...

status = CSL_srioHwSetupRaw(hSrio, &config);
...

```

18.2.8 CSL_srioGetHwSetup

```
CSL_Status CSL_srioGetHwSetup ( CSL\_SrioHandle          hSrio,
                                CSL\_SrioHwSetup *        hwSetup
                                )
```

Description

It retrieves the hardware setup parameters.

Arguments

<code>hSrio</code>	Handle to the SRIO instance
<code>hwSetup</code>	Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup retrieved
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS – Invalid parameters

Pre Condition

Both *CSL_srioInit()* and *CSL_srioOpen()* must be called successfully in order before calling this API.

Post Condition

The hardware set up structure will be populated with values from the registers.

Modifies

None

Example

```
CSL_Status      status;
CSL_SrioHwSetup hwSetup;
...
status = CSL_srioGetHwsetup(hSrio, &hwSetup);
...
```

18.2.9 CSL_srioLsuSetup

```
CSL_Status CSL_srioLsuSetup ( CSL\_SrioHandle          hSrio,
                              CSL_SrioDirectIO_ConfigXfr * IsuConfig,
                              Uint8                      index
                              )
```

Description

Function to configure the LSU module for Direct IO transfer.

Arguments

<code>hSrio</code>	Handle to the SRIO instance
--------------------	-----------------------------

lsuConfig	Pointer to the direct IO configuration structure
index	Index to the LSU block number

Return Value CSL_Status

- CSL_SOK - Successfully configured the LSU module
- CSL_ESYS_BADHANDLE - Invalid handle is passed
- CSL_ESYS_INVPARAMS – Invalid parameter

Pre Condition

None

Post Condition

The LSU module registers are configured with the passed parameters and the data transfer starts.

Modifies

LSU module registers

Example

```

#define LARGE_DEV_ID          0xBEEF
#define SRIO_PKT_TYPE_NWRITE  0x54
CSL_Status                    status;
CSL_SrioDirectIO_ConfigXfr    lsuConfig;
UInt8                         index;
UInt32                        dst=0x20000000;
UInt32                        src=0x30000000;
index = 1;
lsuConfig.srcNodeAddr         = (UInt32)src; /* Source address */
...
lsuConfig.dstNodeAddr.addressHi = 0;
lsuConfig.dstNodeAddr.addressLo = (UInt32)dst; /* Destination
address */
lsuConfig.byteCnt              = 256;
lsuConfig.idSize               = 1; /* 16 bit device id*/
lsuConfig.priority             = 2; /* PKT priority is 2*/
lsuConfig.xambs                = 0; /* Not an extended
                                address */
lsuConfig.dstId                = LARGE_DEV_ID;
lsuConfig.intrReq              = 0; /* No interrupts */
lsuConfig.pktType              = SRIO_PKT_TYPE_NWRITE;
                                /* write with no response */
lsuConfig.hopCount             = 0; /*Valid for maintainance pkt
                                */
lsuConfig.doorbellInfo         = 0; /* Not a doorbell pkt */
lsuConfig.outPortId            = 3; /* Tx on Port
                                SELECTED_PORT */
status = CSL_srioLsuSetup(hSrio, &lsuConfig, index);

```

18.2.10 CSL_srioGetBaseAddress

```
CSL_Status CSL_srioGetBaseAddress ( CSL_InstNum      srioNum,
                                   CSL\_SrioParam * pSrioParam,
                                   CSL\_SrioBaseAddress * pBaseAddress
                                   )
```

Description

This function gets the base address of the given SRIO instance. This function will be called inside the CSL_srioOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral. MMRs go to an alternate location.

Arguments

<code>srioNum</code>	Specifies the instance of the SRIO to be opened
<code>pSrioParam</code>	SRIO module specific parameters
<code>pBaseAddress</code>	Pointer to base address structure containing base address details

Return Value CSL_Status

- CSL_SOK Open call is successful
- CSL_ESYS_FAIL SRIO instance is not available.
- CSL_ESYS_INVPARAMS Invalid Parameters

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status      status;
CSL_SrioBaseAddress baseAddress;
...
status = CSL_SrioGetBaseAddress(CSL_SRIO, NULL, &baseAddress);
...
```

18.3 Data Structures

This section lists the data structures available in the SRIO module.

18.3.1 CSL_SrioObj

Detailed Description

Serial Rapid IO object structure.

Field Documentation

CSL_InstNum CSL_SrioObj::perNum

Instance of SRIO being referred by this object

CSL_SrioRegsOvly CSL_SrioObj::regs

Pointer to the register overlay structure of the SRIO

18.3.2 CSL_SrioConfig

Detailed Description

Config-structure used to configure the SRIO using CSL_srioHwSetupRaw(). This is a structure of register values, rather than a structure of register field values like CSLSrioHwSetup

Field Documentation

UInt32 CSL_SrioConfig::BASE_ID

Base device ID CSR register

UInt32 CSL_SrioConfig::BLK_EN[9]

Block enable registers

UInt32 CSL_SrioConfig::COMP_TAG

Component tag CSR

UInt32 CSL_SrioConfig::DEVICEID_REG1

Device ID register 1

UInt32 CSL_SrioConfig::DEVICEID_REG2

Device ID register 2

UInt32 CSL_SrioConfig::DOORBELL_ICCR[4]

Doorbell interrupt clear registers

UInt32 CSL_SrioConfig::ERR_DET

Logical/Transport layer error detect CSR

UInt32 CSL_SrioConfig::ERR_EN

Logical/Transport layer error enable CSR

UInt32 CSL_SrioConfig::ERR_RST_EVNT_ICCR

Error, Reset, and Special event interrupt clear registers

Uint32 CSL_SrioConfig::FLOW_CNTL[16]

Flow control table entry registers

Uint32 CSL_SrioConfig::GBL_EN

Peripheral global enable register

Uint32 CSL_SrioConfig::HOST_BASE_ID_LOCK

Host base device ID lock CSR

CSL_SrioHw_pkt_fwdRegs CSL_SrioConfig::HW_PKT_FWD[4]

Packet forwarding registers for 16-bit and 8-bit device IDs

Uint32 CSL_SrioConfig::INTDST_RATE_CNTL

INTDST interrupt rate control register for DST 0

Uint32 CSL_SrioConfig::IP_PRESCALAR

Serial port IP prescalar

[**CSL_SrioCfgLsuRegs CSL_SrioConfig::LSU\[4\]**](#)

LSU registers

Uint32 CSL_SrioConfig::LSU_ICCR

LSU interrupt clear registers

Uint32 CSL_SrioConfig::PCR

Peripheral control register

Uint32 CSL_SrioConfig::PE_LL_CTL

Processing element logical layer control CSR register

Uint32 CSL_SrioConfig::PER_SET_CNTL

Peripheral settings control register

[**CSL_SrioCfgPortRegs CSL_SrioConfig::PORT\[4\]**](#)

Port registers

[**CSL_SrioCfgPortErrorRegs CSL_SrioConfig::PORT_ERROR\[4\]**](#)

Port error CSR

[**CSL_SrioCfgPortOptionRegs CSL_SrioConfig::PORT_OPTION\[4\]**](#)

Port options CSR

Uint32 CSL_SrioConfig::PW_TGT_ID

Port-write target device ID CSR

Uint32 CSL_SrioConfig::SERDES_CFG_CNTL[4]

SerDes macros configuration control registers

Uint32 CSL_SrioConfig::SERDES_CFGRX_CNTL[4]

SerDes RX channels configuration control registers

Uint32 CSL_SrioConfig::SERDES_CFGTX_CNTL[4]

SerDes TX channels configuration control registers

Uint32 CSL_SrioConfig::SP_GEN_CTL

Port general control CSR

UInt32 CSL_SrioConfig::SP_IP_DISCOVERY_TIMER

Port IP discovery timer in 4x mode

UInt32 CSL_SrioConfig::SP_IP_MODE

Port IP mode CSR

UInt32 CSL_SrioConfig::SP_LT_CTL

Port link time-out control CSR

UInt32 CSL_SrioConfig::SP_RT_CTL

Port link response time-out control CSR

18.3.3 CSL_SrioContext

Detailed Description

Module specific context information. Present implementation of SRIO CSL doesn't have any context information.

Field Documentation

UInt16 CSL_SrioContext::contextInfo

Context information of SRIO CSL. The declaration is just a placeholder for future implementation.

18.3.4 CSL_SrioHwSetup

Detailed Description

Hardware setup structure.

Field Documentation

UInt32 CSL_SrioHwSetup::blkEn[9]

Controls reset to logical block n

UInt32 CSL_SrioHwSetup::componentTag

Software defined component Tag for PE (processing element). Useful for devices without device IDs

UInt32 CSL_SrioHwSetup::devicId1

This value is equal to the value of the RapidIO Base Device ID CSR. The CPU must read the CSR value and set this register, so that out-going packets contain the correct SOURCEID value. This field contains both 16bit and 8bit IDs

UInt32 CSL_SrioHwSetup::devicId2

This is a secondary supported DeviceID checked against an in-coming packet's DestID field. Typically used for Multi-cast support. This field contains both 16bit and 8bit IDs

[CSL_SrioDevIdConfig](#) **CSL_SrioHwSetup::devIdSetup**

Base device configuration

[CSL_SrioDiscoveryTimer](#) CSL_SrioHwSetup::discoveryTimer

Discovery Timer in 4x mode. The discovery-timer allows time for the link partner to enter its DISCOVERY state and if the link partner is supporting 4x mode, for all 4 lanes to be aligned

UInt16 CSL_SrioHwSetup::flowCntlId[16]

Destination ID of flow n

UInt8 CSL_SrioHwSetup::flowCntlIdLen[16]

Selects flow control ID length

Bool CSL_SrioHwSetup::gblEn

Controls reset to all clock domains within the peripheral

UInt32 CSL_SrioHwSetup::lgclTransErrEn

Enable/disable logical/transport layer errors. Macros can be OR'ed to get the value to pass the argument

[CSL_SrioAddrSelect](#) CSL_SrioHwSetup::peLIAddrCtrl

Sets the number of address bits generated by the PE as a source and processed by the PE as the target of an operation

Bool CSL_SrioHwSetup::perEn

Peripheral enable. Controls the flow of data in the logical layer of the peripheral

[CSL_SrioControlSetup](#) CSL_SrioHwSetup::periCntlSetup

This is used to hold the information for local SRIO's control setup

[CSL_SrioPktFwdCntl](#) CSL_SrioHwSetup::pktFwdCntl[4]

Sets the boundaries for device IDs that are part of the chain and the packet can be forwarded to

UInt32 CSL_SrioHwSetup::portCntlIndpEn[4]

Port control independent error reporting enable. Macros can be OR'ed to get the value

[CSL_SrioPortCntlConfig](#) CSL_SrioHwSetup::portCntlSetup[4]

Port control configuration

[CSL_SrioPortErrConfig](#) CSL_SrioHwSetup::portErrSetup[4]

Port error configuration

[CSL_SrioPortGenConfig](#) CSL_SrioHwSetup::portGenSetup

Port General configuration

UInt32 CSL_SrioHwSetup::portIpModeSet

This configures the SP_IP_MODE register

UInt32 CSL_SrioHwSetup::portIpPrescaler

This configures the SP_IP_PRESCALE register

[CSL_SrioPwTimer](#) CSL_SrioHwSetup::pwTimer

Port-Write Timer. The timer defines a period to repeat sending an error reporting Port-Write request for software assistance. The timer stopped by software writing to the error detect registers

Uint32 CSL_SrioHwSetup::serDesRxChannelCfg [4]
SERDES RX channel configure

Uint32 CSL_SrioHwSetup::serDesPllCfg[4]
General Purpose I/O bits can be used to control any SerDes PLL control functions. Mapping of GPIO bits is device specific based on the SERDES macro that is implemented

[CSL_SrioSilenceTimer](#) **CSL_SrioHwSetup::silenceTimer[4]**
Silence timer. Defines the time of the port in the SILENT state

Uint32 CSL_SrioHwSetup::serDesTxChannelCfg [4]
SERDES TX channel configure

18.3.5 CSL_SrioParam

Detailed Description

Module specific parameters. Present implementation of SRIO CSL doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_SrioParam::flags
Bit mask to be used for module specific parameters. The declaration is just a place-holder for future implementation.

18.3.6 CSL_SrioBaseAddress

Detailed Description

This structure contains the base-address information for the peripheral instance.

Field Documentation

CSL_SrioRegsOvly CSL_SrioBaseAddress::regs
Base-address of the configuration registers of the peripheral

18.3.7 CSL_SrioCfgLsuRegs

Detailed Description

This structure contains the control and congestion flow mask registers for the configuration of Load/Store module in SRIO.

Field Documentation

Uint32 CSL_SrioCfgLsuRegs::LSU_FLOW_MASKS
Core LSU congestion control flow mask register

Uint32 CSL_SrioCfgLsuRegs::LSU_REG0
LSU control register 0

Uint32 CSL_SrioCfgLsuRegs::LSU_REG1
LSU control register 1

Uint32 CSL_SrioCfgLsuRegs::LSU_REG2
LSU control register 2

Uint32 CSL_SrioCfgLsuRegs::LSU_REG3
LSU control register 3

Uint32 CSL_SrioCfgLsuRegs::LSU_REG4
LSU control register 4

18.3.8 CSL_SrioCfgPortRegs

Detailed Description

This structure contains port configuration CSR registers.

Field Documentation

Uint32 CSL_SrioCfgPortRegs::SP_ACKID_STAT
Port local ACK ID status CSR

Uint32 CSL_SrioCfgPortRegs::SP_CTL
Port control CSR

Uint32 CSL_SrioCfgPortRegs::SP_ERR_STAT
Port error and status CSR

Uint32 CSL_SrioCfgPortRegs::SP_LM_REQ
Port link maintenance request CSR

18.3.9 CSL_SrioCfgPortErrorRegs

Detailed Description

This structure contains port error configuration CSR registers.

Field Documentation

Uint32 CSL_SrioCfgPortErrorRegs::SP_ERR_DET
Port error detect CSR

Uint32 CSL_SrioCfgPortErrorRegs::SP_ERR_RATE
Port error rate CSR

Uint32 CSL_SrioCfgPortErrorRegs::SP_ERR_THRESH
Port error rate threshold CSR

Uint32 CSL_SrioCfgPortErrorRegs::SP_RATE_EN
Port error enable CSR

18.3.10 CSL_SrioCfgPortOptionRegs

Detailed Description

This structure contains port error configuration CSR registers.

Field Documentation
UInt32 CSL_SrioCfgPortOptionRegs::SP_CS_TX

Port control symbol transmit register

UInt32 CSL_SrioCfgPortOptionRegs::SP_CTL_INDEP

Port control independent register

UInt32 CSL_SrioCfgPortOptionRegs::SP_MULT_EVNT_CS

Port multicast-event control symbol request register

UInt32 CSL_SrioCfgPortOptionRegs::SP_RST_OPT

Port reset option CSR

UInt32 CSL_SrioCfgPortOptionRegs::SP_SILENCE_TIMER

Port silence timer register

18.3.11 CSL_SrioControlSetup

Detailed Description

This structure contains the control parameters of SRIO.

Field Documentation
Bool CSL_SrioControlSetup::bootComplete

Controls ability to write any register during initialization. It also includes read only registers during normal mode of operation that have application defined reset value. 0 - write enabled, 1 - write to read only registers disabled. Usually the boot_complete is asserted once after reset to define power on configuration

[CSL_SrioBufMode](#) CSL_SrioControlSetup::bufferMode

UDI buffering setup (priority versus port)

[CSL_SrioBusTransPriority](#) CSL_SrioControlSetup::busTransPriority

Internal bus transaction priority

Bool CSL_SrioControlSetup::loopback

0 - Normal operation, 1 - Loop back. Transmit data to receive on the same port. Packet data is looped back in the digital domain before the SerDes macros

UInt8 CSL_SrioControlSetup::pllEn

SERDES macros PLL enable/disable. Enable/disable macros are OR' ed to get the value

[CSL_SrioClkDiv](#) CSL_SrioControlSetup::prescalar

Internal clock frequency pre-scalar, used to drive the request to response timers

Bool CSL_SrioControlSetup::swMemSleepOverride

Puts the memories in either in sleep mode or in awake mode, while in shutdown

[CSL_SrioTxPriorityWm](#) CSL_SrioControlSetup::txPriority0Wm

Sets the required number of logical layer TX buffers needed to send priority 0 packets across the UDI interface

[CSL_SrioTxPriorityWm](#) CSL_SrioControlSetup::txPriority1Wm

Sets the required number of logical layer TX buffers needed to send priority 1 packets across the UDI interface

[CSL_SrioTxPriorityWm](#) CSL_SrioControlSetup::txPriority2Wm

Sets the required number of logical layer TX buffers needed to send priority 2 packets across the UDI interface

18.3.12 CSL_SrioDevInfo

Detailed Description

This structure contains SRIO vendor related information.

Field Documentation
UInt16 CSL_SrioDevInfo::devId

Identifies the vendor specific type of device

UInt32 CSL_SrioDevInfo::devRevision

Vendor supplied device revision

UInt16 CSL_SrioDevInfo::devVendorId

Device vendor ID assigned by RapidIO TA

18.3.13 CSL_SrioAssyInfo

Detailed Description

This structure contains the information about SRIO assembly.

Field Documentation
UInt16 CSL_SrioAssyInfo::assyId

Identifies the vendor specific type of assembly

UInt16 CSL_SrioAssyInfo::assyRevision

Vendor supplied assembly revision

UInt16 CSL_SrioAssyInfo::assyVendorId

Assembly vendor ID assigned by RapidIO TA

18.3.14 CSL_SrioCntlSym

Detailed Description

This structure contains control symbols used for packet acknowledgment.

Field Documentation
UInt8 CSL_SrioCntlSym::cmd

Used in conjunction with stype1 encoding to define the link maintenance commands

Bool CSL_SrioCntlSym::emb

When set, force the outbound flow to insert control symbol into packet. Used in debug mode

UInt8 CSL_SrioCntlSym::par0

Used in conjunction with stype0 encoding

UInt8 CSL_SrioCntlSym::par1

Used in conjunction with stype0 encoding

UInt8 CSL_SrioCntlSym::stype0

Encoding for control symbol that make use of parameters PAR_0 and PAR_1

UInt8 CSL_SrioCntlSym::stype1

Encoding for control symbol that make use of parameter CMD

CSL_SrioPortNum CSL_SrioCntlSym:: portNum
--

Port number

18.3.15 CSL_SrioLogTrErrInfo

Detailed Description

This structure contains captured error information of logical/transport layer.

Field Documentation

UInt16 CSL_SrioLogTrErrInfo::destId

The destination ID associated with the error

UInt32 CSL_SrioLogTrErrInfo::errAddrHi

The address associated with the error (only valid for devices supporting 66 and 50 bit addresses)

UInt32 CSL_SrioLogTrErrInfo::errAddrLo

The address associated with the error (only valid for devices supporting 66 and 50 bit addresses)

UInt8 CSL_SrioLogTrErrInfo::ftype

Format type associated with the error

UInt16 CSL_SrioLogTrErrInfo::impSpecific

Implementation specific information associated with the error

UInt16 CSL_SrioLogTrErrInfo::srcId

The source ID associated with the error

UInt8 CSL_SrioLogTrErrInfo::tType

Transaction type associated with the error

UInt8 CSL_SrioLogTrErrInfo::xambs

Extended address bits of the address associated with the error

18.3.16 CSL_SrioPortData

Detailed Description

This structure is used to hold the configuration/status information of different SRIO ports.

Field Documentation
CSL_BitMask32 CSL_SrioPortData::data

Desired information in the registers

UInt32 CSL_SrioPortData::index

Port selection

18.3.17 CSL_SrioPortGenConfig

Detailed Description

This structure contains information to configure port.

Field Documentation
Bool CSL_SrioPortGenConfig::hostEn

A Host device enable 0b0 - agent or slave device 0b1 - host device

Bool CSL_SrioPortGenConfig::masterEn

It controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests

UInt32 CSL_SrioPortGenConfig::portLinkTimeout

Timeout value for all ports on the device. This timeout is for link events such as sending a packet to receiving the corresponding ACK

UInt32 CSL_SrioPortGenConfig::portRespTimeout

Timeout value for all ports on the device. This timeout is for sending a packet to receiving the corresponding response packet

18.3.18 CSL_SrioPortCntlConfig

Detailed Description

This structure contains information to configure port parameters.

Field Documentation
Bool CSL_SrioPortCntlConfig::dropPktEn

Enabling this bit causes the port to drop packets that are acknowledged with a packet-not-accepted control symbol when the error failed threshold is exceeded

Bool CSL_SrioPortCntlConfig::errCheckDis

Disables/Enables all RapidIO transmission error checking

Bool CSL_SrioPortCntlConfig::inPortEn

Input port receive enable. Controls input port to respond to any packet

Bool CSL_SrioPortCntlConfig::multicastRcvEn

Disables/Enables the multicast event reception on this port

Bool CSL_SrioPortCntlConfig::outPortEn

Controls output port to issue any packets and control symbols

Bool CSL_SrioPortCntlConfig::portDis

Controls port receivers/drivers to receive/transmit to any packets or control symbols

Bool CSL_SrioPortCntlConfig::portLockoutEn

When the bit is set the port is stopped and is not enabled to issue or receive any packets

[CSL_SrioPortWidthOverride](#) CSL_SrioPortCntlConfig::portWidthOverride

Soft port configuration to override the hardware size

Bool CSL_SrioPortCntlConfig::stopOnPortFailEn

Enabling this bit causes the port to stop attempting to send packets to the connected device when the output failed-encountered bit is set.

18.3.19 CSL_SrioPortErrConfig

Detailed Description

This structure contains information to configure port error enable and error rate thresholds.

Field Documentation

UInt32 CSL_SrioPortErrConfig::portErrRateEn

Enable/disable port error interrupts. Macros can be OR'ed to get the value to pass the argument

UInt8 CSL_SrioPortErrConfig::portErrRtDegrdThresh

The threshold value for reporting an error condition due to a degrading link

UInt8 CSL_SrioPortErrConfig::portErrRtFldThresh

The threshold value for reporting an error condition due to a possibly broken link

[CSL_SrioErrRtNum](#) CSL_SrioPortErrConfig::portErrRtRec

Limit value to the error rate counter above the failed threshold trigger

[CSL_SrioErrRtBias](#) CSL_SrioPortErrConfig::prtErrRtBias

The error rate bias value

18.3.20 CSL_SrioPidNumber

Detailed Description

This structure is used to return the contents of the Peripheral Identification register, which has the versioning information, used to identify the specific SRIO peripheral.

Field Documentation

UInt8 CSL_SrioPidNumber::srioClass

Identifies the class of peripheral

UInt8 CSL_SrioPidNumber::srioRevision

Identifies the revision of SRIO

UInt8 CSL_SrioPidNumber::srioType

Identifies the type of peripheral

18.3.21 CSL_SrioDevIdConfig

Detailed Description

This structure contains base device configuration parameters.

Field Documentation

UInt16 CSL_SrioDevIdConfig::hostBaseDevId

This is the base ID for the Host PE that is initializing this PE (processing element)

UInt16 CSL_SrioDevIdConfig::largeTrBaseDevId

This is the base ID of the device in a large common transport system (Only valid for end points, and if bit 4 of the PEFTR register is set)

UInt8 CSL_SrioDevIdConfig::smallTrBaseDevId

This is the base ID of the device in small common transport system (End points only)

18.3.22 CSL_SrioBlkEn

Detailed Description

This structure is used to enable/disable the blocks within the SRIO peripheral.

Field Documentation

Bool CSL_SrioBlkEn::block0

Enable/disable MMR non-Reset/PD control Registers (Logical Block 0)

Bool CSL_SrioBlkEn::block1

Enable/disable LSU (Direct I/O Initiator)

Bool CSL_SrioBlkEn::block2

Enable/disable MAU (Direct I/O Target)

Bool CSL_SrioBlkEn::block3

Enable/disable TXU (Message Passing Initiator)

Bool CSL_SrioBlkEn::block4

Enable/disable RXU (Message Passing Target)

Bool CSL_SrioBlkEn::block5

Enable/disable Port 0 Data path

Bool CSL_SrioBlkEn::block6

Enable/disable Port 1 Data path

Bool CSL_SrioBlkEn::block7

Enable/disable port 2 Data path

Bool CSL_SrioBlkEn::block8

Enable/disable Port 3 Data path

18.3.23 CSL_SrioPktFwdCntl

Detailed Description

This structure is used to configure hardware packet forwarding.

Field Documentation

UInt16 CSL_SrioPktFwdCntl::largeLowBoundDevId

Lower 16-bit Device ID boundary. Destination ID lower than this number cannot use the table entry

UInt16 CSL_SrioPktFwdCntl::largeUpBoundDevId

Upper 16-bit Device ID boundary. Destination ID above this range cannot use the table entry

[CSL_SrioPortNum](#) CSL_SrioPktFwdCntl::outBoundPort

Output port number for packet's whose destination ID falls within the 8b or 16b range for this table entry

UInt8 CSL_SrioPktFwdCntl::smallLowBoundDevId

Lower 8-bit Device ID boundary. Destination ID lower than this number cannot use the table entry

UInt8 CSL_SrioPktFwdCntl::smallUpBoundDevId

Upper 8-bit Device ID boundary. Destination ID above this range cannot use the table entry

18.3.24 CSL_SrioLsuCompStat

Detailed Description

This structure is used to return the completion status of the LSU command.

Field Documentation

[CSL_SrioCompCode](#) CSL_SrioLsuCompStat::lsuCompCode

This is used to return the LSU command completion code

[CSL_SrioPortNum](#) CSL_SrioLsuCompStat::portNum

Port number

18.3.25 CSL_SrioLongAddress

Detailed Description

This structure contains local configuration base address.

Field Documentation

UInt32 CSL_SrioLongAddress::addressHi

Configuration address high

UInt32 CSL_SrioLongAddress::addressLo

Configuration address low

18.3.26 CSL_SrioPortErrCapt

Detailed Description

This structure is used to return the error capture information for the specified port.

Field Documentation**UInt32 CSL_SrioPortErrCapt::capture0**

This contains the control symbol information or 0-3 bytes of packet header

UInt32 CSL_SrioPortErrCapt::capture1

This contains the control symbol information or 4-7 bytes of packet header

UInt32 CSL_SrioPortErrCapt::capture2

This contains the control symbol information or 8-11 bytes of packet header

UInt32 CSL_SrioPortErrCapt::capture3

This contains the control symbol information or 12-15 bytes of packet header

UInt8 CSL_SrioPortErrCapt::errorType

Encoded error type

UInt32 CSL_SrioPortErrCapt::impSpecData

Implementation specific data

[CSL_SrioPortCaptType](#) CSL_SrioPortErrCapt::portErrCaptType

Type of information logged

[CSL_SrioPortNum](#) CSL_SrioPortErrCapt::portNum

Port number for which the error data is to be captured

18.3.27 CSL_SrioPortWriteCapt

Detailed Description

This structure is used to return the port write capture information.

Field Documentation**UInt32 CSL_SrioPortWriteCapt::capture0**

Port-Write payload, word 0

UInt32 CSL_SrioPortWriteCapt::capture1

Port-Write payload, word 1

UInt32 CSL_SrioPortWriteCapt::capture2

Port-Write payload, word 2

UInt32 CSL_SrioPortWriteCapt::capture3

Port-Write payload, word 3

18.3.28 CSL_SrioDirectIO_ConfigXfr

Detailed Description

This structure is used to configure LSU module for Transfer enable.

Field Documentation

UInt16 CSL_SrioDirectIO_ConfigXfr::byteCnt

Number of data bytes to Read/Write - up to 4KB. (Used in conjunction with RapidIO address to create WRSIZE/RDSIZE and WDPTR in RapidIO packet header)

UInt16 CSL_SrioDirectIO_ConfigXfr::doorbellInfo

Doorbell info

UInt16 CSL_SrioDirectIO_ConfigXfr::dstId

RapidIO destination ID field specifying target device

[CSL_SrioLongAddress](#) CSL_SrioDirectIO_ConfigXfr::dstNodeAddr

Destination node address

UInt8 CSL_SrioDirectIO_ConfigXfr::hopCount

RapidIO hop count

UInt8 CSL_SrioDirectIO_ConfigXfr::idSize

RapidIO tt field specifying 8 or 16bit Device IDs

Bool CSL_SrioDirectIO_ConfigXfr::intrReq

RapidIO Lsu module interrupt request

UInt8 CSL_SrioDirectIO_ConfigXfr::outPortId

Out port ID

UInt8 CSL_SrioDirectIO_ConfigXfr::pktType

Packet type

UInt8 CSL_SrioDirectIO_ConfigXfr::priority

This field specifies packet priority

UInt32 CSL_SrioDirectIO_ConfigXfr::srcNodeAddr

Source node address

UInt8 CSL_SrioDirectIO_ConfigXfr::xambs

RapidIO xambs field specifying extended address MSB

18.3.29 CSL_SrioSerDesPllCfg

Detailed Description

This structure configures SERDES PLL

Field Documentation

CSL_SrioSerDesLoopBandwidth CSL_SrioSerDesPllCfg::loopBandwidth

Loop bandwidth

Bool CSL_SrioSerDesPllCfg::pllEnable

Enables the internal PLL of the SERDES

CSL_SrioSerDesPllMply CSL_SrioSerDesPllCfg::pllMplyFactor

PLL multiplication factor

18.3.30 CSL_SrioSerDesRxCfg

Detailed Description

This structure configures the SERDES receiver

Field Documentation

CSL_SrioSerDesBusWidth CSL_SrioSerDesRxCfg::busWidth

Bus width

UInt8 CSL_SrioSerDesRxCfg::clockDataRecovery

Clock/data recovery configuration

Bool CSL_SrioSerDesRxCfg::enRx

Enable receiver

UInt8 CSL_SrioSerDesRxCfg::equalizer

Configure the adaptive equalizer

Bool CSL_SrioSerDesRxCfg::invertedPolarity

Inverted polarity

CSL_SrioSerDesLos CSL_SrioSerDesRxCfg::los

Loss of signal detection, with selectable thresholds

CSL_SrioSerDesRate CSL_SrioSerDesRxCfg::rate

Operating rate

CSL_SrioSerDesSymAlignment CSL_SrioSerDesRxCfg::symAlign

Enables internal or external symbol alignment.

CSL_SrioSerDesTermination CSL_SrioSerDesRxCfg::termination

Selects input termination options suitable for a variety of AC or DC coupled scenarios

18.3.31 CSL_SrioSerDesTxCfg

Detailed Description

This structure configures the SERDES transmitter.

Field Documentation

CSL_SrioSerDesBusWidth CSL_SrioSerDesTxCfg::busWidth

Bus width

CSL_SrioSerDesCommonMode CSL_SrioSerDesTxCfg::commonMode

Common mode configuration

Bool [CSL_SrioSerDesTxCfg::enableFixedPhase](#)

Enable fixed TXBCLKIN[i] phase with TXBCLK [i]

Bool [CSL_SrioSerDesTxCfg::enTx](#)

Enable transmitter

Bool [CSL_SrioSerDesTxCfg::invertedPolarity](#)

Inverted polarity

Uint8 [CSL_SrioSerDesTxCfg::outputDeEmphasis](#)

Output de-emphasis select

[CSL_SrioSerDesSwingCfg](#) [CSL_SrioSerDesTxCfg::outputSwing](#)

Output swing configuration

[CSL_SrioSerDesRate](#) [CSL_SrioSerDesTxCfg::rate](#)

Operating rate

18.4 Enumerations

This section lists the enumerations available in the SRIO module.

18.4.1 CSL_SrioHwControlCmd

enum CSL_SrioHwControlCmd

This enum describes the commands used to control the SRIO through CSL_srioHwControl().

Enumeration values:

<i>CSL_SRIO_CMD_PER_ENABLE</i>	Enables/disables the peripheral. Parameters: <i>Bool *</i>
<i>CSL_SRIO_CMD_PLL_CONTROL</i>	Enable/disable the SERDES PLLs. PLL enable macros can be OR'ed to get the value. Parameters: <i>Uint8 *</i>
<i>CSL_SRIO_CMD_DOORBELL_INTR_CLEAR</i>	Clears doorbell interrupts. Macros can be OR'ed to get the value. Parameters: <i>CSL_SrioPortData *</i>
<i>CSL_SRIO_CMD_LSU_INTR_CLEAR</i>	Clear load/store module interrupts. Macros can be OR'ed to get the value. Parameters: <i>Uint32 *</i>
<i>CSL_SRIO_CMD_ERR_RST_INTR_CLEAR</i>	Clears Error, Reset, and Special Event interrupts. Macros can be OR'ed to get the value. Parameters: <i>Uint32 *</i>
<i>CSL_SRIO_CMD_DIRECTIO_SRC_NODE_ADDR_SET</i>	Sets 32-bit DSP byte source address. Parameters: <i>CSL_SrioPortData *</i>
<i>CSL_SRIO_CMD_DIRECTIO_DST_ADDR_MSB_SET</i>	Sets the rapid IO destination MSB address. Parameters: <i>CSL_SrioPortData *</i>
<i>CSL_SRIO_CMD_DIRECTIO_DST_ADDR_LSB_SET</i>	Sets the rapid IO destination LSB address. Parameters: <i>CSL_SrioPortData *</i>
<i>CSL_SRIO_CMD_DIRECTIO_XFR_BYTECNT_SET</i>	Number of data bytes to Read/Write - up to 4KB. Parameters: <i>CSL_SrioPortData *</i>

<i>CSL_SRIO_CMD_DIRECTIO_LSU_XFR_TYPE_SET</i>	<p>Sets 4 MSBs to 4-bit ftype field for all packets and 4 LSBs to 4-bit trans field for Packet types 2,5 and 8.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_DOORBELL_XFR_SET</i>	<p>Sets RapidIO doorbell info field for type 10 packets and sets the packet type to 10.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_DIRECTIO_LSU_FLOW_MASK_SET</i>	<p>Sets LSU flow masks. Port number is passed as input. Macros can be OR'ed to get the value for argument.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_PORT_COMMAND_SET</i>	<p>Sets the command to be sent in the link-request control symbol.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_SP_ERR_STAT_CLEAR</i>	<p>Clear fields' status of SP_ERR_STAT register. Macros can be OR'ed to get the value to pass the argument.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_LGCL_TRANS_ERR_STAT_CLEAR</i>	<p>Clear status of Logical/Transport layer errors. Macros can be OR'ed to get the value to pass the argument.</p> <p>Parameters: <i>Uint32*</i></p>
<i>CSL_SRIO_CMD_SP_ERR_DET_STAT_CLEAR</i>	<p>Clears status of port errors interrupts. Macros can be OR'ed to get the value to pass the argument.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_SP_CTL_INDEP_ERR_STAT_CLEAR</i>	<p>Clear the fields status of the SP_CTL_INDEP register.</p> <p>Parameters: <i>CSL_SrioPortData*</i></p>
<i>CSL_SRIO_CMD_CNTL_SYM_SET</i>	<p>Set control symbols used for packet acknowledgment.</p> <p>Parameters: <i>CSL_SrioCntlSym*</i></p>
<i>CSL_SRIO_CMD_INTDST_RATE_CNTL</i>	<p>Sets interrupt rate control counter.</p> <p>Parameters: <i>Uint32*</i></p>

18.4.2 CSL_SrioHwStatusQuery

enum CSL_SrioHwStatusQuery

This enum describes the commands used to get status of various parameters of the SRIO. These values are used in CSL_srioGetHwStatus().

Enumeration values:

<i>CSL_SRIO_QUERY_PID_NUMBER</i>	This query command returns the SRIO Peripheral Identification number. Parameters: <i>CSL_SrioPidNumber</i>
<i>CSL_SRIO_QUERY_GBL_EN_STAT</i>	Gets global enable status. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_BLK_EN_STAT</i>	Gets block enable status for all the blocks. Parameters: <i>CSL_SrioBlkEn</i>
<i>CSL_SRIO_QUERY_DOORBELL_INTR_STAT</i>	Get doorbell interrupts status. The port number is passed as input. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_LSU_INTR_STAT</i>	Get the LSU interrupts status. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_ERR_RST_INTR_STAT</i>	Gets Error, Reset, and Special Event interrupts status. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_LSU_INTR_DECODE_STAT</i>	Get status of LSU interrupts decode for DST 0. Parameters: <i>Bool</i>
<i>CSL_SRIO_QUERY_ERR_INTR_DECODE_STAT</i>	Get Error, Reset, and Special Event interrupts decode status for DST 0. Parameters: <i>Bool</i>
<i>CSL_SRIO_QUERY_LSU_COMP_CODE_STAT</i>	Gets the status of the pending command of LSU registers for a particular port. Parameters: <i>CSL_SrioLsuCompStat</i>
<i>CSL_SRIO_QUERY_LSU_BSY_STAT</i>	Gets status of the command registers of LSU module for a particular port. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_DEV_ID_INFO</i>	Gets the type of device (Vendor specific). Parameters:

<i>CSL_SRIO_QUERY_ASSY_ID_INFO</i>	<i>CSL_SrioDevInfo</i> Gets vendor specific assembly information. Parameters: <i>CSL_SrioAssyInfo</i>
<i>CSL_SRIO_QUERY_PE_FEATURE</i>	Gets processing element features. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_SRC_OPERN_SUPPORT</i>	Get source operations CAR status. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_DST_OPERN_SUPPORT</i>	Get destination operations CAR status. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_LCL_CFG_BAR</i>	Get local configuration space base addresses. Parameters: <i>CSL_SrioLongAddress</i>
<i>CSL_SRIO_QUERY_SP_LM_RESP_STAT</i>	Get status of SP_LM_RESP register fields. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_SP_ACKID_STAT</i>	Get status of SP_ACKID_STAT register fields. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_SP_ERR_STAT</i>	Get status of SP_ERR_STAT register fields. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_SP_CTL</i>	Gets SP_CTL register status fields. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_LGCL_TRNS_ERR_STAT</i>	Get the status of logical/transport layer errors. Parameters: <i>Uint32</i>
<i>CSL_SRIO_QUERY_LGCL_TRNS_ERR_CAPT</i>	Get captured error info of logical/transport layer. Parameters: <i>CSL_SrioLogTrErrInfoCapt</i>
<i>CSL_SRIO_QUERY_SP_ERR_DET_STAT</i>	Get status of port error detect CSR fields. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_PORT_ERR_CAPT</i>	Get the port error captured information. Parameters: <i>CSL_SrioPortErrCapt</i>
<i>CSL_SRIO_QUERY_SP_CTL_INDEP</i>	Get port control independent register

<i>CSL_SRIO_QUERY_PW_CAPTURE</i>	fields status. Parameters: <i>CSL_SrioPortData</i> Get the port write capture information. Parameters: <i>CSL_SrioPortWriteCapt</i>
<i>CSL_SRIO_QUERY_ERR_RATE_CNTR_READ</i>	Reads the count of the number of transmission errors that have occurred. Parameters: <i>CSL_SrioPortData</i>
<i>CSL_SRIO_QUERY_PEAK_ERR_RATE_READ</i>	Reads the peak value of the error rate counter. Parameters: <i>CSL_SrioPortData</i>

18.4.3 CSL_SrioPortCaptType

enum CSL_SrioPortCaptType

This enum describes type of the captured information type at the time of port error.

Enumeration values:

<i>CSL_SRIO_CAPT_TYPE_PKT</i>	Port captured packet data during error
<i>CSL_SRIO_CAPT_TYPE_CNTL_SYM</i>	Port captured control symbols during error
<i>CSL_SRIO_CAPT_TYPE_IMP_SPEC</i>	Port captured implementation specific during error

18.4.4 CSL_SrioPortNum

enum CSL_SrioPortNum

This enum describes the port number configuration for SRIO.

Enumeration values:

<i>CSL_SRIO_PORT_0</i>	Port number 0
<i>CSL_SRIO_PORT_1</i>	Port number 1
<i>CSL_SRIO_PORT_2</i>	Port number 2
<i>CSL_SRIO_PORT_3</i>	Port number 3

18.4.5 CSL_SrioDiscoveryTimer

enum CSL_SrioDiscoveryTimer

This enum describes the discovery time for the link partner to enter its discovery state.

Enumeration values:

<i>CSL_SRIO_DISCOVERY_TIME_0</i>	Discovery time is 102.4ps (for debug mode only)
<i>CSL_SRIO_DISCOVERY_TIME_1</i>	Discovery time is 0.84ms
<i>CSL_SRIO_DISCOVERY_TIME_2</i>	Discovery time is 0.84ms*2
<i>CSL_SRIO_DISCOVERY_TIME_3</i>	Discovery time is 0.84ms*3
<i>CSL_SRIO_DISCOVERY_TIME_4</i>	Discovery time is 0.84ms*4
<i>CSL_SRIO_DISCOVERY_TIME_5</i>	Discovery time is 0.84ms*5
<i>CSL_SRIO_DISCOVERY_TIME_6</i>	Discovery time is 0.84ms*6

<i>CSL_SRIO_DISCOVERY_TIME_7</i>	Discovery time is 0.84ms*7
<i>CSL_SRIO_DISCOVERY_TIME_8</i>	Discovery time is 0.84ms*8
<i>CSL_SRIO_DISCOVERY_TIME_9</i>	Discovery time is 0.84ms*9
<i>CSL_SRIO_DISCOVERY_TIME_10</i>	Discovery time is 0.84ms*10
<i>CSL_SRIO_DISCOVERY_TIME_11</i>	Discovery time is 0.84ms*11
<i>CSL_SRIO_DISCOVERY_TIME_12</i>	Discovery time is 0.84ms*12
<i>CSL_SRIO_DISCOVERY_TIME_13</i>	Discovery time is 0.84ms*13
<i>CSL_SRIO_DISCOVERY_TIME_14</i>	Discovery time is 0.84ms*14
<i>CSL_SRIO_DISCOVERY_TIME_15</i>	Discovery time is 0.84ms*15

18.4.6 CSL_SrioPwTimer

enum CSL_SrioPwTimer

This enum describes the port write time for request.

Enumeration values:

<i>CSL_SRIO_PW_TIME_0</i>	Port write is sent only once (disabled)
<i>CSL_SRIO_PW_TIME_1</i>	Port write time is 107ms - 214ms
<i>CSL_SRIO_PW_TIME_2</i>	Port write time is 214ms - 321ms
<i>CSL_SRIO_PW_TIME_6</i>	Port write time is 428ms - 535ms
<i>CSL_SRIO_PW_TIME_8</i>	Port write time is 856ms - 963ms
<i>CSL_SRIO_PW_TIME_15</i>	Port write time is 0.82 - 1.64us

18.4.7 CSL_SrioSilenceTimer

enum CSL_SrioSilenceTimer

This enum describes the time values for the port in silent state.

Enumeration values:

<i>CSL_SRIO_SILENCE_TIME_0</i>	Port in silent state for 64ns (debug mode)
<i>CSL_SRIO_SILENCE_TIME_1</i>	Port in silent state for 13.1us*1
<i>CSL_SRIO_SILENCE_TIME_2</i>	Port in silent state for 13.1us*2
<i>CSL_SRIO_SILENCE_TIME_3</i>	Port in silent state for 13.1us*3
<i>CSL_SRIO_SILENCE_TIME_4</i>	Port in silent state for 13.1us*4
<i>CSL_SRIO_SILENCE_TIME_5</i>	Port in silent state for 13.1us*5
<i>CSL_SRIO_SILENCE_TIME_6</i>	Port in silent state for 13.1us*6
<i>CSL_SRIO_SILENCE_TIME_7</i>	Port in silent state for 13.1us*7
<i>CSL_SRIO_SILENCE_TIME_8</i>	Port in silent state for 13.1us*8
<i>CSL_SRIO_SILENCE_TIME_9</i>	Port in silent state for 13.1us*9
<i>CSL_SRIO_SILENCE_TIME_10</i>	Port in silent state for 13.1us*10
<i>CSL_SRIO_SILENCE_TIME_11</i>	Port in silent state for 13.1us*11
<i>CSL_SRIO_SILENCE_TIME_12</i>	Port in silent state for 13.1us*12
<i>CSL_SRIO_SILENCE_TIME_13</i>	Port in silent state for 13.1us*13
<i>CSL_SRIO_SILENCE_TIME_14</i>	Port in silent state for 13.1us*14
<i>CSL_SRIO_SILENCE_TIME_15</i>	Port in silent state for 13.1us*15

18.4.8 CSL_SrioBusTransPriority

enum CSL_SrioBusTransPriority

This enum describes the bus transaction priority values for SRIO.

Enumeration values:

<i>CSL_SRIO_BUS_TRANS_PRIORITY_0</i>	Sets internal bus priority to 0(highest)
<i>CSL_SRIO_BUS_TRANS_PRIORITY_1</i>	Sets internal bus priority to 1
<i>CSL_SRIO_BUS_TRANS_PRIORITY_2</i>	Sets internal bus priority to 2
<i>CSL_SRIO_BUS_TRANS_PRIORITY_3</i>	Sets internal bus priority to 3
<i>CSL_SRIO_BUS_TRANS_PRIORITY_4</i>	Sets internal bus priority to 4
<i>CSL_SRIO_BUS_TRANS_PRIORITY_5</i>	Sets internal bus priority to 5
<i>CSL_SRIO_BUS_TRANS_PRIORITY_6</i>	Sets internal bus priority to 6
<i>CSL_SRIO_BUS_TRANS_PRIORITY_7</i>	Sets internal bus priority to 7

18.4.9 CSL_SrioClkDiv

enum CSL_SrioClkDiv

This enum describes the internal clock prescale values for SRIO.

Enumeration values:

<i>CSL_SRIO_CLK_PRESCALE_0</i>	Sets the internal clock frequency Min 44.7 and Max 89.5
<i>CSL_SRIO_CLK_PRESCALE_1</i>	Sets the internal clock frequency Min 89.5 and Max 179.0
<i>CSL_SRIO_CLK_PRESCALE_2</i>	Sets the internal clock frequency Min 134.2 and Max 268.4
<i>CSL_SRIO_CLK_PRESCALE_3</i>	Sets the internal clock frequency Min 180.0 and Max 360.0
<i>CSL_SRIO_CLK_PRESCALE_4</i>	Sets the internal clock frequency Min 223.7 and Max 447.4
<i>CSL_SRIO_CLK_PRESCALE_5</i>	Sets the internal clock frequency Min 268.4 and Max 536.8
<i>CSL_SRIO_CLK_PRESCALE_6</i>	Sets the internal clock frequency Min 313.2 and Max 626.4
<i>CSL_SRIO_CLK_PRESCALE_7</i>	Sets the internal clock frequency Min 357.9 and Max 715.8
<i>CSL_SRIO_CLK_PRESCALE_8</i>	Sets the internal clock frequency Min 402.6 and Max 805.4
<i>CSL_SRIO_CLK_PRESCALE_9</i>	Sets the internal clock frequency Min 447.4 and Max 894.8
<i>CSL_SRIO_CLK_PRESCALE_10</i>	Sets the internal clock frequency Min 492.1 and Max 984.2
<i>CSL_SRIO_CLK_PRESCALE_11</i>	Sets the internal clock frequency Min 536.9 and Max 1073.8
<i>CSL_SRIO_CLK_PRESCALE_12</i>	Sets the internal clock frequency Min 581.6 and Max 1163.2
<i>CSL_SRIO_CLK_PRESCALE_13</i>	Sets the internal clock frequency Min 626.3 and Max 1252.6
<i>CSL_SRIO_CLK_PRESCALE_14</i>	Sets the internal clock frequency Min 671.1 and Max 1342.2
<i>CSL_SRIO_CLK_PRESCALE_15</i>	Sets the internal clock frequency Min 715.8 and Max 1431.6

18.4.10 CSL_SrioTxPriorityWm

enum CSL_SrioTxPriorityWm

This enum describes required buffer count for packets to be sent across the UDI interface.

Enumeration values:

<i>CSL_SRIO_TX_PRIORITY_WM_0</i>	Transmit credit threshold 1
<i>CSL_SRIO_TX_PRIORITY_WM_1</i>	Transmit credit threshold 2
<i>CSL_SRIO_TX_PRIORITY_WM_2</i>	Transmit credit threshold 3
<i>CSL_SRIO_TX_PRIORITY_WM_3</i>	Transmit credit threshold 4

<i>CSL_SRIO_TX_PRIORITY_WM_4</i>	Transmit credit threshold 5
<i>CSL_SRIO_TX_PRIORITY_WM_5</i>	Transmit credit threshold 6
<i>CSL_SRIO_TX_PRIORITY_WM_6</i>	Transmit credit threshold 7
<i>CSL_SRIO_TX_PRIORITY_WM_7</i>	Transmit credit threshold 8

18.4.11 CSL_SrioAddrSelect

enum CSL_SrioAddrSelect

This enum describes extended addressing control bits.

Enumeration values:

<i>CSL_SRIO_ADDR_SELECT_66BIT</i>	PE supports 66 bit addresses
<i>CSL_SRIO_ADDR_SELECT_50BIT</i>	PE supports 50 bit addresses
<i>CSL_SRIO_ADDR_SELECT_34BIT</i>	PE supports 34 bit addresses (default)

18.4.12 CSL_SrioBufMode

enum CSL_SrioBufMode

This enum describes UDI buffers setup.

Enumeration values:

<i>CSL_SRIO_1X_MODE_PORT</i>	UDI buffers are port based
<i>CSL_SRIO_1X_MODE_PRIORITY</i>	UDI buffers are priority based

18.4.13 CSL_SrioPortWidthOverride

enum CSL_SrioPortWidthOverride

This enum describes the port width override options.

Enumeration values:

<i>CSL_SRIO_PORT_WIDTH_NO_OVERRIDE</i>	No override to the port width
<i>CSL_SRIO_PORT_WIDTH_LANE_0</i>	Force single lane, lane 0
<i>CSL_SRIO_PORT_WIDTH_LANE_2</i>	Force single lane, lane 2

18.4.14 CSL_SrioErrRtBias

enum CSL_SrioErrRtBias

This enum describes the error rate bias values.

Enumeration values:

<i>CSL_SRIO_ERR_RATE_BIAS_0</i>	Error rate counter do not decrement
<i>CSL_SRIO_ERR_RATE_BIAS_1MS</i>	Error rate counter decrements every 1ms
<i>CSL_SRIO_ERR_RATE_BIAS_10MS</i>	Error rate counter decrements every 10ms
<i>CSL_SRIO_ERR_RATE_BIAS_100MS</i>	Error rate counter decrements every 100ms
<i>CSL_SRIO_ERR_RATE_BIAS_1S</i>	Error rate counter decrements every 1s
<i>CSL_SRIO_ERR_RATE_BIAS_10S</i>	Error rate counter decrements every 10s
<i>CSL_SRIO_ERR_RATE_BIAS_100S</i>	Error rate counter decrements every 100s
<i>CSL_SRIO_ERR_RATE_BIAS_1000S</i>	Error rate counter decrements every 1000s

CSL_SRIO_ERR_RATE_BIAS_10000S

Error rate counter decrements every 10000s

18.4.15 CSL_SrioPortLnkTimeout

enum CSL_SrioPortLnkTimeout

This enum describes the port link timeout values.

Enumeration values:

CSL_SRIO_PORT_LNK_TIMEOUT_0	Timer disabled
CSL_SRIO_PORT_LNK_TIMEOUT_1	Timeout value is 205ns
CSL_SRIO_PORT_LNK_TIMEOUT_2	Timeout value is 3.1us
CSL_SRIO_PORT_LNK_TIMEOUT_3	Timeout value is 52.4us
CSL_SRIO_PORT_LNK_TIMEOUT_4	Timeout value is 839.5us
CSL_SRIO_PORT_LNK_TIMEOUT_5	Timeout value is 13.4ms
CSL_SRIO_PORT_LNK_TIMEOUT_6	Timeout value is 215ms
CSL_SRIO_PORT_LNK_TIMEOUT_7	Timeout value is 3.4s

18.4.16 CSL_SrioCompCode

enum CSL_SrioCompCode

This enumeration indicates the status of the pending command.

Enumeration values:

CSL_SRIO_TRANS_NO_ERR	Transaction complete, no errors (Posted/Non-posted)
CSL_SRIO_TRANS_TIMEOUT	Transaction timeout occurred on non-posted transaction
CSL_SRIO_FLOW_CNTL_BLOCKADE	Transaction complete, packet not sent due to flow control blockade (Xoff)
CSL_SRIO_RESP_PKT_ERR	Transaction complete, non-posted response packet (type 8 and 13) contained ERROR status, or response payload length was in error
CSL_SRIO_INV_PROG_ENCODING	Transaction complete, packet not sent due to unsupported transaction type or invalid programming encoding for one or more LSU register fields
CSL_SRIO_DMA_TRANS_ERR	DMA data transfer error
CSL_SRIO_RETRY_DRBL_RESP_RCVD	"Retry" DOORBELL response received, or atomic test-and-swap was not allowed (semaphore in use)
CSL_SRIO_UNAVAILABLE_OUTBOUND_CR	Transaction complete, packet not sent due to unavailable outbound credit at given priority

18.4.17 CSL_SrioErrRtNum

enum CSL_SrioErrRtNum

This enum describes error rate counter threshold values.

Enumeration values:

<i>CSL_SRIO_ERR_RATE_COUNT_2</i>	Only count 2 errors and above
<i>CSL_SRIO_ERR_RATE_COUNT_4</i>	Only count 4 errors and above
<i>CSL_SRIO_ERR_RATE_COUNT_16</i>	Only count 16 errors and above
<i>CSL_SRIO_ERR_RATE_COUNT_NO_LIMIT</i>	No limit in incrementing the error rate count

18.4.18 CSL_SrioSerDesLoopBandwidth

enum CSL_SrioSerDesLoopBandwidth

Enum for SERDES Loop bandwidth.

Enumeration values:

<i>CSL_SRIO_SERDES_LOOP_BANDWIDTH_FREQ_DEP</i>	Frequency dependant loop bandwidth
<i>CSL_SRIO_SERDES_LOOP_BANDWIDTH_LOW</i>	Low loop bandwidth
<i>CSL_SRIO_SERDES_LOOP_BANDWIDTH_HIGH</i>	High loop bandwidth

18.4.19 CSL_SrioSerDesPllMply

enum CSL_SrioSerDesPllMply

Enum for SERDES PLL multiplication factor values.

Enumeration values:

<i>CSL_SRIO_SERDES_PLL_MPLY_BY_4</i>	SERDES PLL multiplication factor 4
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_5</i>	SERDES PLL multiplication factor 5
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_6</i>	SERDES PLL multiplication factor 6
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_8</i>	SERDES PLL multiplication factor 8
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_10</i>	SERDES PLL multiplication factor 10
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_12</i>	SERDES PLL multiplication factor 12
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_12_5</i>	SERDES PLL multiplication factor 12.5
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_15</i>	SERDES PLL multiplication factor 15
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_20</i>	SERDES PLL multiplication factor 20
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_25</i>	SERDES PLL multiplication factor 25
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_50</i>	SERDES PLL multiplication factor 50
<i>CSL_SRIO_SERDES_PLL_MPLY_BY_60</i>	SERDES PLL multiplication factor 60

18.4.20 CSL_SrioSerDesLos

enum CSL_SrioSerDesLos

Enum for SERDES loss of signal detection configuration.

Enumeration values:

<i>CSL_SRIO_SERDES_LOS_DET_DISABLE</i>	Loss of signal detection disabled
<i>CSL_SRIO_SERDES_LOS_DET_HIGH_THRESHOLD</i>	Loss of signal detection threshold in the range 85 to 195mVdfpp.
<i>CSL_SRIO_SERDES_LOS_DET_LOW_THRESHOLD</i>	Loss of signal detection threshold in the range 65 to 175mVdfpp.

18.4.21 CSL_SrioSerDesSymAlignment

enum CSL_SrioSerDesSymAlignment

Enum for SERDES symbol alignment configuration.

Enumeration values:

CSL_SRIO_SERDES_SYM_ALIGN_DISABLE

CSL_SRIO_SERDES_SYM_ALIGN_COMMA

CSL_SRIO_SERDES_SYM_ALIGN_JOG

Symbol alignment disabled

Comma alignment: Symbol alignment will be performed whenever a misaligned comma symbol is received.

Alignment Jog. The symbol alignment will be adjusted by one bit position

18.4.22 CSL_SrioSerDesTermination

enum CSL_SrioSerDesTermination

Enum for SERDES input termination.

Enumeration values:

CSL_SRIO_SERDES_TERMINATION_VDDT

CSL_SRIO_SERDES_TERMINATION_0_8_VDDT

CSL_SRIO_SERDES_TERMINATION_FLOATING

Input termination is to VDDT

Input termination is to 0.8 VDDT

Input termination is floating

18.4.23 CSL_SrioSerDesRate

enum CSL_SrioSerDesRate

Enum for the SERDES operating rate configuration.

Enumeration values:

CSL_SRIO_SERDES_RATE_FULL

CSL_SRIO_SERDES_RATE_HALF

CSL_SRIO_SERDES_RATE_QUARTER

Full rate operation

Half rate operation

Quarter rate operation

18.4.24 CSL_SrioSerDesBusWidth

enum CSL_SrioSerDesBusWidth

Enum for the SERDES bus width configuration.

Enumeration values:

CSL_SRIO_SERDES_BUS_WIDTH_10_BIT

CSL_SRIO_SERDES_BUS_WIDTH_8_BIT

10 bit bus width

8 bit bus width

18.4.25 CSL_SrioSerDesCommonMode

enum CSL_SrioSerDesCommonMode

Enum for SERDES TX common mode configuration.

Enumeration values:

CSL_SRIO_SERDES_COMMON_MODE_NORMAL

Normal: Common mode not adjusted

CSL_SRIO_SERDES_COMMON_MODE_RAISED

Raised: Common mode raised by 5% of e54

18.4.26 CSL_SrioSerDesSwingCfg

enum CSL_SrioSerDesSwingCfg

Enum for SERDES output swing configuration.

Enumeration values:

CSL_SRIO_SERDES_SWING_AMPLITUDE_125

Output swing amplitude 125

CSL_SRIO_SERDES_SWING_AMPLITUDE_250

Output swing amplitude 250

CSL_SRIO_SERDES_SWING_AMPLITUDE_500

Output swing amplitude 500

CSL_SRIO_SERDES_SWING_AMPLITUDE_625

Output swing amplitude 625

CSL_SRIO_SERDES_SWING_AMPLITUDE_750

Output swing amplitude 750

CSL_SRIO_SERDES_SWING_AMPLITUDE_1000

Output swing amplitude 1000

CSL_SRIO_SERDES_SWING_AMPLITUDE_1125

Output swing amplitude 1125

CSL_SRIO_SERDES_SWING_AMPLITUDE_1250

Output swing amplitude 1250

18.5 Macros

```
#define CSL_SRIO_DOORBELL_INTR0 (0x00000001)
#define CSL_SRIO_DOORBELL_INTR1 (0x00000002)
#define CSL_SRIO_DOORBELL_INTR2 (0x00000004)
#define CSL_SRIO_DOORBELL_INTR3 (0x00000008)
#define CSL_SRIO_DOORBELL_INTR4 (0x00000010)
#define CSL_SRIO_DOORBELL_INTR5 (0x00000020)
#define CSL_SRIO_DOORBELL_INTR6 (0x00000040)
#define CSL_SRIO_DOORBELL_INTR7 (0x00000080)
#define CSL_SRIO_DOORBELL_INTR8 (0x00000100)
#define CSL_SRIO_DOORBELL_INTR9 (0x00000200)
#define CSL_SRIO_DOORBELL_INTR10 (0x00000400)
#define CSL_SRIO_DOORBELL_INTR11 (0x00000800)
#define CSL_SRIO_DOORBELL_INTR12 (0x00001000)
#define CSL_SRIO_DOORBELL_INTR13 (0x00002000)
#define CSL_SRIO_DOORBELL_INTR14 (0x00004000)
#define CSL_SRIO_DOORBELL_INTR15 (0x00008000)
```

Doorbell interrupts clear macros

```
#define CSL_SRIO_ERR_DEV_RST_INTR (0x00010000)
#define CSL_SRIO_ERR_PORT3_INTR (0x00000800)
#define CSL_SRIO_ERR_PORT2_INTR (0x00000400)
#define CSL_SRIO_ERR_PORT1_INTR (0x00000200)
#define CSL_SRIO_ERR_PORT0_INTR (0x00000100)
#define CSL_SRIO_ERR_LGCL_INTR (0x00000004)
#define CSL_SRIO_ERR_PW_INTR (0x00000002)
#define CSL_SRIO_ERR_MULTICAST_INTR (0x00000001)
```

Error, Reset, and Special Event Status Interrupt clear macros

```
#define CSL_SRIO_ERR_IMP_SPECIFIC ~(0x80000000)
#define CSL_SRIO_CORRUPT_CNTL_SYM ~(0x00400000)
#define CSL_SRIO_CNTL_SYM_UNEXPECTED_ACKID ~(0x00200000)
#define CSL_SRIO_RCVD_PKT_NOT_ACCPT ~(0x00100000)
#define CSL_SRIO_PKT_UNEXPECTED_ACKID ~(0x00080000)
#define CSL_SRIO_RCVD_PKT_WITH_BAD_CRC ~(0x00040000)
#define CSL_SRIO_RCVD_PKT_OVER_276B ~(0x00020000)
#define CSL_SRIO_NON_OUTSTANDING_ACKID ~(0x00000020)
#define CSL_SRIO_PROTOCOL_ERROR ~(0x00000010)
#define CSL_SRIO_UNSOLICITED_ACK_CNTL_SYM ~(0x00000002)
#define CSL_SRIO_LINK_TIMEOUT ~(0x00000001)
```

Port error detect clear macros

```
#define CSL_SRIO_ERR_IMP_SPECIFIC_ENABLE (0x80000000)
#define CSL_SRIO_CORRUPT_CNTL_SYM_ENABLE (0x00400000)
#define CSL_SRIO_CNTL_SYM_UNEXPECTED_ACKID_ENABLE (0x00200000)
#define CSL_SRIO_RCVD_PKT_NOT_ACCPT_ENABLE (0x00100000)
#define CSL_SRIO_PKT_UNEXPECTED_ACKID_ENABLE (0x00080000)
#define CSL_SRIO_RCVD_PKT_WITH_BAD_CRC_ENABLE (0x00040000)
#define CSL_SRIO_RCVD_PKT_OVER_276B_ENABLE (0x00020000)
#define CSL_SRIO_NON_OUTSTANDING_ACKID_ENABLE (0x00000020)
#define CSL_SRIO_PROTOCOL_ERROR_ENABLE (0x00000010)
#define CSL_SRIO_UNSOLICITED_ACK_CNTL_SYM_ENABLE (0x00000002)
```

```
#define CSL_SRIO_LINK_TIMEOUT_ENABLE (0x00000001)
```

Port error detect enable macros

```
#define CSL_SRIO_ERR_OUTPUT_PKT_DROP (0x04000000)
#define CSL_SRIO_ERR_OUTPUT_FLD_ENC (0x02000000)
#define CSL_SRIO_ERR_OUTPUT_DEGRD_ENC (0x01000000)
#define CSL_SRIO_ERR_OUTPUT_RETRY_ENC (0x00100000)
#define CSL_SRIO_OUTPUT_ERROR_ENC (0x00020000)
#define CSL_SRIO_INPUT_ERROR_ENC (0x00000200)
#define CSL_SRIO_PORT_WRITE_PND (0x00000010)
#define CSL_SRIO_PORT_ERROR (0x00000004)
```

Port error Status clear macros

```
#define CSL_SRIO_IO_ERR_RESP_ENABLE (0x80000000)
#define CSL_SRIO_ILL_TRANS_DECODE_ENABLE (0x08000000)
#define CSL_SRIO_ILL_TRANS_TARGET_ERR_ENABLE (0x04000000)
#define CSL_SRIO_PKT_RESP_TIMEOUT_ENABLE (0x01000000)
#define CSL_SRIO_UNSOLICITED_RESP_ENABLE (0x00800000)
#define CSL_SRIO_UNSUPPORTED_TRANS_ENABLE (0x00400000)
```

Logical/transport layer error enable

```
#define CSL_SRIO_IO_ERR_RSPNS ~(0x80000000)
#define CSL_SRIO_ILL_TRANS_DECODE ~(0x08000000)
#define CSL_SRIO_PKT_RSPNS_TIMEOUT ~(0x01000000)
#define CSL_SRIO_UNSOLICITED_RSPNS ~(0x00800000)
#define CSL_SRIO_UNSUPPORTED_TRANS ~(0x00400000)
```

Logical/transport layer error status clear

```
#define CSL_SRIO_LSU_INTR0 (0x00000001)
#define CSL_SRIO_LSU_INTR1 (0x00000002)
#define CSL_SRIO_LSU_INTR2 (0x00000004)
#define CSL_SRIO_LSU_INTR3 (0x00000008)
#define CSL_SRIO_LSU_INTR4 (0x00000010)
#define CSL_SRIO_LSU_INTR5 (0x00000020)
#define CSL_SRIO_LSU_INTR6 (0x00000040)
#define CSL_SRIO_LSU_INTR7 (0x00000080)
#define CSL_SRIO_LSU_INTR8 (0x00000100)
#define CSL_SRIO_LSU_INTR9 (0x00000200)
#define CSL_SRIO_LSU_INTR10 (0x00000400)
#define CSL_SRIO_LSU_INTR11 (0x00000800)
#define CSL_SRIO_LSU_INTR12 (0x00001000)
#define CSL_SRIO_LSU_INTR13 (0x00002000)
#define CSL_SRIO_LSU_INTR14 (0x00004000)
#define CSL_SRIO_LSU_INTR15 (0x00008000)
#define CSL_SRIO_LSU_INTR16 (0x00010000)
#define CSL_SRIO_LSU_INTR17 (0x00020000)
#define CSL_SRIO_LSU_INTR18 (0x00040000)
#define CSL_SRIO_LSU_INTR19 (0x00080000)
#define CSL_SRIO_LSU_INTR20 (0x00100000)
#define CSL_SRIO_LSU_INTR21 (0x00200000)
#define CSL_SRIO_LSU_INTR22 (0x00400000)
#define CSL_SRIO_LSU_INTR23 (0x00800000)
#define CSL_SRIO_LSU_INTR24 (0x01000000)
```

```

#define CSL_SRIO_LSU_INTR25      (0x02000000)
#define CSL_SRIO_LSU_INTR26      (0x04000000)
#define CSL_SRIO_LSU_INTR27      (0x08000000)
#define CSL_SRIO_LSU_INTR28      (0x10000000)
#define CSL_SRIO_LSU_INTR29      (0x20000000)
#define CSL_SRIO_LSU_INTR30      (0x40000000)
#define CSL_SRIO_LSU_INTR31      (0x80000000)

```

LSU interrupts clear macros

```

#define CSL_SRIO_PLL1_ENABLE (0x00000001)
#define CSL_SRIO_PLL2_ENABLE (0x00000002)
#define CSL_SRIO_PLL3_ENABLE (0x00000004)
#define CSL_SRIO_PLL4_ENABLE (0x00000008)

```

PLL enable macros

#define CSL_SRIO_HWSETUP_DEFAULTS

Value:

```

{ \
    CSL_SRIO_PCR_PEREN_RESETVAL, \
    { \
        CSL_SRIO_PER_SET_CNTL_SW_MEM_SLEEP_OVERRIDE_RESETVAL, \
        CSL_SRIO_PER_SET_CNTL_LOOPBACK_RESETVAL, \
        CSL_SRIO_PER_SET_CNTL_BOOT_COMPLETE_RESETVAL, \
        CSL_SRIO_TX_PRIORITY_WM_3, \
        CSL_SRIO_TX_PRIORITY_WM_2, \
        CSL_SRIO_TX_PRIORITY_WM_1, \
        CSL_SRIO_BUS_TRANS_PRIORITY_0, \
        CSL_SRIO_1X_MODE_PRIORITY, \
        CSL_SRIO_CLK_PRESCALE_0, \
        0x0 \
    }, \
    0x1, \
    { \
        0x1, \
        0x0, \
        0x0, \
        0x0, \
        0x0, \
        0x0, \
        0x0, \
        0x0, \
        0x0 \
    }, \
    CSL_SRIO_DEVICEID_REG1_RESETVAL, \
    CSL_SRIO_DEVICEID_REG2_RESETVAL, \
    { \
        {0x0000FFFF, 0x0000FFFF, CSL_SRIO_PORT_3, 0x000000FF, \
        0x000000FF}, \
        {0x0000FFFF, 0x0000FFFF, CSL_SRIO_PORT_3, 0x000000FF, \
        0x000000FF}, \
        {0x0000FFFF, 0x0000FFFF, CSL_SRIO_PORT_3, 0x000000FF, \
        0x000000FF}, \
        {0x0000FFFF, 0x0000FFFF, CSL_SRIO_PORT_3, 0x000000FF, 0x000000FF} \
    }, \
}

```

```

{ \
{ \
FALSE, \
CSL_SRIO_SERDES_PLL_MPLY_BY_4, \
CSL_SRIO_SERDES_LOOP_BANDWIDTH_FREQ_DEP \
}, \

{ \
FALSE, \
CSL_SRIO_SERDES_PLL_MPLY_BY_4, \
CSL_SRIO_SERDES_LOOP_BANDWIDTH_FREQ_DEP \
}, \

{ \
FALSE, \
CSL_SRIO_SERDES_PLL_MPLY_BY_4, \
CSL_SRIO_SERDES_LOOP_BANDWIDTH_FREQ_DEP \
}, \

{ \
FALSE, \
CSL_SRIO_SERDES_PLL_MPLY_BY_4, \
CSL_SRIO_SERDES_LOOP_BANDWIDTH_FREQ_DEP \
} \
}, \
{ \
{ \
FALSE, \
CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
CSL_SRIO_SERDES_RATE_FULL, \
FALSE, \
CSL_SRIO_SERDES_TERMINATION_VDDT, \
CSL_SRIO_SERDES_SYM_ALIGN_DISABLE, \
CSL_SRIO_SERDES_LOS_DET_DISABLE, \
0x0, \
0x0 \
}, \

{ \
FALSE, \
CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
CSL_SRIO_SERDES_RATE_FULL, \
FALSE, \
CSL_SRIO_SERDES_TERMINATION_VDDT, \
CSL_SRIO_SERDES_SYM_ALIGN_DISABLE, \
CSL_SRIO_SERDES_LOS_DET_DISABLE, \
0x0, \
0x0 \
}, \

{ \
FALSE, \
CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
CSL_SRIO_SERDES_RATE_FULL, \
FALSE, \
CSL_SRIO_SERDES_TERMINATION_VDDT, \
CSL_SRIO_SERDES_SYM_ALIGN_DISABLE, \
CSL_SRIO_SERDES_LOS_DET_DISABLE, \
0x0, \
0x0 \
}, \

```

```

    { \
        FALSE, \
        CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
        CSL_SRIO_SERDES_RATE_FULL, \
        FALSE, \
        CSL_SRIO_SERDES_TERMINATION_VDDT, \
        CSL_SRIO_SERDES_SYM_ALIGN_DISABLE, \
        CSL_SRIO_SERDES_LOS_DET_DISABLE, \
        0x0, \
        0x0 \
    } \
}, \
{ \
    { \
        FALSE, \
        CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
        CSL_SRIO_SERDES_RATE_FULL, \
        FALSE, \
        CSL_SRIO_SERDES_COMMON_MODE_NORMAL, \
        CSL_SRIO_SERDES_SWING_AMPLITUDE_125, \
        0x0, \
        FALSE \
    }, \
    { \
        FALSE, \
        CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
        CSL_SRIO_SERDES_RATE_FULL, \
        FALSE, \
        CSL_SRIO_SERDES_COMMON_MODE_NORMAL, \
        CSL_SRIO_SERDES_SWING_AMPLITUDE_125, \
        0x0, \
        FALSE \
    }, \
    { \
        FALSE, \
        CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
        CSL_SRIO_SERDES_RATE_FULL, \
        FALSE, \
        CSL_SRIO_SERDES_COMMON_MODE_NORMAL, \
        CSL_SRIO_SERDES_SWING_AMPLITUDE_125, \
        0x0, \
        FALSE \
    }, \
    { \
        FALSE, \
        CSL_SRIO_SERDES_BUS_WIDTH_10_BIT, \
        CSL_SRIO_SERDES_RATE_FULL, \
        FALSE, \
        CSL_SRIO_SERDES_COMMON_MODE_NORMAL, \
        CSL_SRIO_SERDES_SWING_AMPLITUDE_125, \
        0x0, \
        FALSE \
    } \
}, \
{0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1,
  0x1, \0x1, 0x1 }, \

```

447

```

        }, \
        {\
            FALSE, \
            FALSE, \
            FALSE, \

(CSL_SrioPortWidthOverride)CSL_SRIO_SP_CTL_PORT_WIDTH_OVERRIDE_RESETVAL
, \
            FALSE, \
            FALSE, \
            FALSE, \
            FALSE, \
            FALSE \
        } \
    }, \
    CSL_SRIO_ERR_EN_RESETVAL, \
    {\
        {\
            CSL_SRIO_SP_RATE_EN_RESETVAL, \
(CSL_SrioErrRtBias)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_BIAS_RESETVAL, \

(CSL_SrioErrRtNum)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_RECOVERY_RESETVAL, \

CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_FAILED_THRESHOLD_RESETVAL, \
CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_DEGRADED_THRES_RESETVAL \
        }, \
        {\
            CSL_SRIO_SP_RATE_EN_RESETVAL, \

(CSL_SrioErrRtBias)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_BIAS_RESETVAL, \

(CSL_SrioErrRtNum)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_RECOVERY_RESETVAL, \

CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_FAILED_THRESHOLD_RESETVAL, \
CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_DEGRADED_THRES_RESETVAL \
        }, \
        {\
            CSL_SRIO_SP_RATE_EN_RESETVAL, \
(CSL_SrioErrRtBias)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_BIAS_RESETVAL, \

(CSL_SrioErrRtNum)CSL_SRIO_SP_ERR_RATE_ERROR_RATE_RECOVERY_RESETVAL, \

CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_FAILED_THRESHOLD_RESETVAL, \
CSL_SRIO_SP_ERR_THRESH_ERROR_RATE_DEGRADED_THRES_RESETVAL \
        } \
    } \

```

```

    }, \
} \

(CSL_SrioDiscoveryTimer)CSL_SRIO_SP_IP_DISCOVERY_TIMER_DISCOVERY_TIMER_
RESETVAL, \
    CSL_SRIO_SP_IP_MODE_RESETVAL, \
    CSL_SRIO_IP_PRESCAL_RESETVAL, \
    (CSL_SrioPwTimer)CSL_SRIO_SP_IP_DISCOVERY_TIMER_PW_TIMER_RESETVAL,
\
    { \

(CSL_SrioSilenceTimer)CSL_SRIO_SP_SILENCE_TIMER_SILENCE_TIMER_RESETVAL,
\

(CSL_SrioSilenceTimer)CSL_SRIO_SP_SILENCE_TIMER_SILENCE_TIMER_RESETVAL,
\

(CSL_SrioSilenceTimer)CSL_SRIO_SP_SILENCE_TIMER_SILENCE_TIMER_RESETVAL,
\

(CSL_SrioSilenceTimer)CSL_SRIO_SP_SILENCE_TIMER_SILENCE_TIMER_RESETVAL
\
    }, \
    { \
        CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
        CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
        CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
        CSL_SRIO_SP_CTL_INDEP_RESETVAL \
    } \
}

```

Default hardware setup parameters

#define CSL_SRIO_CONFIG_DEFAULTS

Value:

```

{ \
    CSL_SRIO_PCR_RESETVAL, \
    CSL_SRIO_PER_SET_CNTL_RESETVAL, \
    0x1, \
    {0x1, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, \
    CSL_SRIO_DEVICEID_REG1_RESETVAL, \
    CSL_SRIO_DEVICEID_REG2_16BNODEID_RESETVAL, \
    { \
        {0xFFFFFFFF, 0x0003FFFF}, \
        {0xFFFFFFFF, 0x0003FFFF}, \
        {0xFFFFFFFF, 0x0003FFFF}, \
        {0xFFFFFFFF, 0x0003FFFF} \
    }, \
    {0x00000000, 0x00000000, 0x00000000, 0x00000000}, \
    {0x00000000, 0x00000000, 0x00000000, 0x00000000}, \
    {0x00000000, 0x00000000, 0x00000000, 0x00000000}, \
    {0x00000000, 0x00000000, 0x00000000, 0x00000000}, \
    CSL_SRIO_LSU_ICCR_RESETVAL, \
    CSL_SRIO_ERR_RST_EVNT_ICCR_RESETVAL, \
    CSL_SRIO_INTDST_RATE_CNTL_RESETVAL, \
    { \

```

```

        {\
            CSL_SRIO_LSU_REG0_RESETVAL, \
            CSL_SRIO_LSU_REG1_RESETVAL, \
            CSL_SRIO_LSU_REG2_RESETVAL, \
            CSL_SRIO_LSU_REG3_RESETVAL, \
            CSL_SRIO_LSU_REG4_RESETVAL, \
            CSL_SRIO_LSU_FLOW_MASKS_RESETVAL \
        }, \
        {\
            CSL_SRIO_LSU_REG0_RESETVAL, \
            CSL_SRIO_LSU_REG1_RESETVAL, \
            CSL_SRIO_LSU_REG2_RESETVAL, \
            CSL_SRIO_LSU_REG3_RESETVAL, \
            CSL_SRIO_LSU_REG4_RESETVAL, \
            CSL_SRIO_LSU_FLOW_MASKS_RESETVAL \
        }, \
        {\
            CSL_SRIO_LSU_REG0_RESETVAL, \
            CSL_SRIO_LSU_REG1_RESETVAL, \
            CSL_SRIO_LSU_REG2_RESETVAL, \
            CSL_SRIO_LSU_REG3_RESETVAL, \
            CSL_SRIO_LSU_REG4_RESETVAL, \
            CSL_SRIO_LSU_FLOW_MASKS_RESETVAL \
        }, \
        {\
            CSL_SRIO_LSU_REG0_RESETVAL, \
            CSL_SRIO_LSU_REG1_RESETVAL, \
            CSL_SRIO_LSU_REG2_RESETVAL, \
            CSL_SRIO_LSU_REG3_RESETVAL, \
            CSL_SRIO_LSU_REG4_RESETVAL, \
            CSL_SRIO_LSU_FLOW_MASKS_RESETVAL \
        } \
    }, \
    {\
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL, \
        CSL_SRIO_FLOW_CNTL_RESETVAL \
    }, \
    CSL_SRIO_PE_LL_CTL_RESETVAL, \
    CSL_SRIO_BASE_ID_RESETVAL, \
    CSL_SRIO_HOST_BASE_ID_LOCK_RESETVAL, \
    CSL_SRIO_COMP_TAG_RESETVAL, \
    CSL_SRIO_SP_LT_CTL_RESETVAL, \

```

```

CSL_SRIO_SP_RT_CTL_RESETVAL, \
CSL_SRIO_SP_GEN_CTL_RESETVAL, \
{\
    {\
        CSL_SRIO_SP_LM_REQ_RESETVAL, \
        CSL_SRIO_SP_ACKID_STAT_RESETVAL, \
        CSL_SRIO_SP_ERR_STAT_RESETVAL, \
        CSL_SRIO_SP_CTL_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_LM_REQ_RESETVAL, \
        CSL_SRIO_SP_ACKID_STAT_RESETVAL, \
        CSL_SRIO_SP_ERR_STAT_RESETVAL, \
        CSL_SRIO_SP_CTL_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_LM_REQ_RESETVAL, \
        CSL_SRIO_SP_ACKID_STAT_RESETVAL, \
        CSL_SRIO_SP_ERR_STAT_RESETVAL, \
        CSL_SRIO_SP_CTL_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_LM_REQ_RESETVAL, \
        CSL_SRIO_SP_ACKID_STAT_RESETVAL, \
        CSL_SRIO_SP_ERR_STAT_RESETVAL, \
        CSL_SRIO_SP_CTL_RESETVAL \
    } \
}, \
CSL_SRIO_ERR_DET_RESETVAL, \
CSL_SRIO_ERR_EN_RESETVAL, \
CSL_SRIO_PW_TGT_ID_RESETVAL, \
{\
    {\
        CSL_SRIO_SP_ERR_DET_RESETVAL, \
        CSL_SRIO_SP_RATE_EN_RESETVAL, \
        CSL_SRIO_SP_ERR_RATE_RESETVAL, \
        CSL_SRIO_SP_ERR_THRESH_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_ERR_DET_RESETVAL, \
        CSL_SRIO_SP_RATE_EN_RESETVAL, \
        CSL_SRIO_SP_ERR_RATE_RESETVAL, \
        CSL_SRIO_SP_ERR_THRESH_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_ERR_DET_RESETVAL, \
        CSL_SRIO_SP_RATE_EN_RESETVAL, \
        CSL_SRIO_SP_ERR_RATE_RESETVAL, \
        CSL_SRIO_SP_ERR_THRESH_RESETVAL \
    }, \
    {\
        CSL_SRIO_SP_ERR_DET_RESETVAL, \
        CSL_SRIO_SP_RATE_EN_RESETVAL, \
        CSL_SRIO_SP_ERR_RATE_RESETVAL, \
        CSL_SRIO_SP_ERR_THRESH_RESETVAL \
    } \
} \

```

```

    }, \
    CSL_SRIO_SP_IP_DISCOVERY_TIMER_RESETVAL, \
    CSL_SRIO_SP_IP_MODE_RESETVAL, \
    CSL_SRIO_IP_PRESCAL_RESETVAL, \
    {\
        {\
            CSL_SRIO_SP_RST_OPT_RESETVAL, \
            CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
            CSL_SRIO_SP_SILENCE_TIMER_RESETVAL, \
            CSL_SRIO_SP_MULT_EVNT_CS_RESETVAL, \
            CSL_SRIO_SP_CS_TX_RESETVAL \
        }, \
        {\
            CSL_SRIO_SP_RST_OPT_RESETVAL, \
            CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
            CSL_SRIO_SP_SILENCE_TIMER_RESETVAL, \
            CSL_SRIO_SP_MULT_EVNT_CS_RESETVAL, \
            CSL_SRIO_SP_CS_TX_RESETVAL \
        }, \
        {\
            CSL_SRIO_SP_RST_OPT_RESETVAL, \
            CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
            CSL_SRIO_SP_SILENCE_TIMER_RESETVAL, \
            CSL_SRIO_SP_MULT_EVNT_CS_RESETVAL, \
            CSL_SRIO_SP_CS_TX_RESETVAL \
        }, \
        {\
            CSL_SRIO_SP_RST_OPT_RESETVAL, \
            CSL_SRIO_SP_CTL_INDEP_RESETVAL, \
            CSL_SRIO_SP_SILENCE_TIMER_RESETVAL, \
            CSL_SRIO_SP_MULT_EVNT_CS_RESETVAL, \
            CSL_SRIO_SP_CS_TX_RESETVAL \
        } \
    } \
} \
}

```

Default values for config structure

18.6 Typedefs

typedef CSL_SrioObj * CSL_SrioHandle

This data type is used to return the handle to the CSL of the SRIO.

Chapter 19

BWMNGMT MODULE

Topics

<u>19. 1 Overview</u>
<u>19. 2 Functions</u>
<u>19. 3 Data Structures</u>
<u>19. 4 Enumerations</u>
<u>19. 5 Macros</u>
<u>19.6 Typedefs</u>

19.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within BWMNGMT module.

The Bandwidth management module used to avoid a requestors (CPU, SDMA, IDMA, and Coherence Operations) being blocked from accessing a resources (L1P, L1D, L2, and configuration bus) for a long period of time.

The following four resources are managed by the BWM control hardware:

- Level 1 Program (L1P) SRAM/Cache
- Level 1 Data (L1D) SRAM/Cache
- Level 2 (L2) SRAM/Cache
- Memory-mapped registers configuration bus

19.2 Functions

This section lists the functions available in the BWMNGMT module.

19.2.1 CSL_bwmngmtInit

CSL_Status CSL_bwmngmtInit (**CSL_BwmngmtContext *** *pContext*)

Description

This is the initialization function for the BWMNGMT CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Context information for the instance. Should be NULL

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_bwmngmtInit(NULL);
```

19.2.2 CSL_bwmngmtOpen

[CSL_BwmngmtHandle](#) CSL_bwmngmtOpen ([CSL_BwmngmtObj *](#) *hBwmngmtObj*,
CSL_InstNum *bwmngmtNum*,
CSL_BwmngmtParam * *pBwmParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the instance and returns a handle to the BWMNGMT instance. The open call sets up the data structures for the particular instance of BWMNGMT device. The device can be re-opened anytime after it has been normally closed, if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

pBwmngmtObj Pointer to the BWMNGMT instance object

bwmngmtNum	Instance of the BWMNGMT to be opened.
pBwmngmtParam	Pointer to module specific parameters
pStatus	Pointer for returning status of the function call

Return Value CSL_BwmngmtHandle

Valid BWMNGMT instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition

The BWMNGMT module must be successfully initialized via *CSL_bwmngmtInit()* before calling this function.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid BWMNGMT handle is returned.
- CSL_ESYS_FAIL - The BWMNGMT instance is invalid.
- CSL_ESYS_INVPARAMS Invalid parameters

2. BWMNGMT object structure is populated.

Modifies

1. The status variable
2. BWMNGMT object structure

Example

```
CSL_Status      status;
CSL_BwmngmtObj bwmngmtObj;
CSL_BwmngmtHandle hBwmngmt;

hBwmngmt = CSL_bwmngmtOpen (&bwmngmtObj,
                             CSL_BWMNGMT,
                             NULL,
                             &status
                             );
```

19.2.3 CSL_bwmngmtClose

CSL_Status CSL_bwmngmtClose ([CSL_BwmngmtHandle](#) *hBwmngmt*)

Description

This function closes the specified instance of BWMNGMT.

Arguments

hBwmngmt Handle to the BWMNGMT instance

Return Value CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

Both `CSL_bwmngmtInit()` and `CSL_bwmngmtOpen()` must be called successfully in that order before `CSL_bwmngmtClose()` can be called.

Post Condition

The BWMNGMT CSL APIs can not be called until the BWMNGMT CSL is reopened again using CSL_bwmngmtOpen().

Modifies

CSL BwmngmtObj structure instance values

Example

```

CSL_Status          status;
CSL_BwmngmtObj      bwmngmtObj;
CSL_BwmngmtHandle    hBwmngmt;

hBwmngmt = CSL_bwmngmtOpen (&bwmngmtObj, CSL_BWMNGMT, NULL, &status);

CSL_bwmngmtClose (hBwmngmt);

```

19.2.4 CSL_bwmngmtHwSetup

```
CSL_Status CSL_bwmngmtHwSetup ( CSL\_BwmngmtHandle hBwmngmt,
                                CSL\_BwmngmtHwSetup * setup
                                )
```

Description

Configures the BWMNGMT using the values passed in through the setup structure. For information passed through the HwSetup Data structure, refer CSL BwmngmtHwSetup.

Arguments

hBwmngmt	Handle to the BWMNGMT instance
setup	Setup structure for BWMNGMT

Return Value CSL Status

- CSL_SOK – Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - If setup is NULL

Pre Condition

Both `CSL_bwmngmtInit()` and `CSL_bwmngmtOpen()` must be called successfully in that order before this function can be called. The main setup structure consists of fields used for the configuration at start up. The user must allocate space for it and fill in the main setup structure fields appropriately before a call to this function is made.

Post Condition

BWMNGMT registers are configured according to the hardware setup parameters.

Modifies

The following registers and fields are programmed by this API

- ## 1. CPU Arbitration Parameters

-
- PRI field set in L1D, L2 and/or EXT
 - MAXWAIT field set in L1D, L2 and/or EXT

2. IDMA Arbitration Parameter

- MAXWAIT field set in L1D, L2 and/or EXT

3. SLAP Arbitration Parameter

- MAXWAIT field set in L1D, L2 and/or EXT

4. MAP Arbitration Parameter

- PRI field set in EXT

5. UC Arbitration Parameter

- MAXWAIT field set in L1D and/or L2

The **control**: bitmask indicates which of the three control blocks (L1D, L2 and EXT) will be set with the associated PRI and MAXWAIT values

Note: That if associated control block is not programmable for given requestor then it will not ignored but no error will be provide. This allows the user to set control to CSL_BWMNGMT_BLOCK_ALL which is the default value. This will set all programmed arbitration values for a given requestor to the same value across the blocks that are recommended.

If PRI is set to CSL_BWMNGMT_PRI_NULL (-1) then no change will be made for the corresponding requestors priority level.

If MAXWAIT is set to CSL_BWMNGMT_MAXWAIT_NULL (-1) then no change will be made for the corresponding requestors maxwait setting.

Examples

Example 1: Sets Priorities and Maxwaits to default values

```
CSL_BwmngmtHandle hBwmngmt;
CSL_BwmngmtHwSetup hwSetup;
hwSetup = CSL_BWMNGMT_HWSETUP_DEFAULTS;
...

// Init successfully done
...
// Open successfully done
...
CSL_bwmngmtHwSetup(hBwmngmt, &hwSetup);
```

Example 2: Sets CPU Priority to 1, CPU Maxwait to 8, MAP Priority to 6 for all blocks (L1D, L2 and EXT)

```
CSL_BwmngmtHandle hBwmngmt;
CSL_BwmngmtHwSetup hwSetup;
hwSetup.cpuPriority = CSL_BWMNGMT_PRI_1;
hwSetup.cpuMaxwait = CSL_BWMNGMT_MAXWAIT_8;
```

```

hwSetup.idmaMaxwait = CSL_BWMNGMT_MAXWAIT_NULL;
hwSetup.slapMaxwait = CSL_BWMNGMT_MAXWAIT_NULL;
hwSetup.mapPriority = CSL_BWMNGMT_PRI_6;
hwSetup.ucMaxwait = CSL_BWMNGMT_MAXWAIT_NULL;
hwSetup.control = CSL_BWMNGMT_BLOCK_ALL;
...

// Init successfully done
...
// Open successfully done
...
CSL_bwmngmtHwSetup(hBwmngmt, &hwSetup);

```

19.2.5 CSL_bwmngmtGetHwSetup

CSL_Status CSL_bwmngmtGetHwSetup ([CSL_BwmngmtHandle](#) *hBwmngmt*,
[CSL_BwmngmtHwSetup](#) * *hwSetup*
)

Description

Gets the current set up of BWMNGMT.

Arguments

<i>hBwmngmt</i>	Handle to the BWMNGMT instance
<i>hwSetup</i>	Setup structure for BWMNGMT

Return Value CSL_Status

- CSL_SOK - Get hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - If setup is NULL

Pre Condition

Both *CSL_bwmngmtInit()* and *CSL_bwmngmtOpen()* must be called successfully in that order before this function can be called.

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

1. CPU Arbitration Parameters

- PRI field set in Control Block Specified by "control"
- MAXWAIT field set in Control Block Specified by "control"

2. IDMA Arbitration Parameter

- MAXWAIT field set in Control Block Specified by "control"

3. SLAP Arbitration Parameter

- MAXWAIT field set in Control Block Specified by "control"

4. MAP Arbitration Parameter

- PRI field set in Control Block Specified by "control"
if not EXT then returns CSL_BWMNGMT_PRI_NULL

5. UC Arbitration Parameter

- MAXWAIT field set in Control Block Specified by "control"
if not L1D or L2 then returns CSL_BWMNGMT_MAXWAIT_NULL

Example

```
CSL_BwmngmtHandle      hBwmngmt;
CSL_BwmngmtHwSetup     hwSetup;
hwSetup.control = CSL_BWMNGMT_BLOCK_L1D;
// Only CSL_BWMNGMT_BLOCK_L1D, CSL_BWMNGMT_BLOCK_L2, or
// CSL_BWMNGMT_BLOCK_EXT are valid
...

// Init successfully done
...
// Open successfully done
...
CSL_bwmngmtGetHwSetup(hBwmngmt, &hwSetup);
```

19.2.6 CSL_bwmngmtHwControl

```
CSL_Status CSL_bwmngmtHwControl ( CSL\_BwmngmtHandle      hBwmngmt,
                                   CSL\_BwmngmtHwControlCmd cmd,
                                   void *                      cmdArg
                                   )
```

Description

Takes a command of BWMNGMT with an optional argument & implements it. Not Implemented. For future use.

Arguments

hBwmngmt	Handle to the BWMNGMT instance
cmd	The command to this API indicates the action to be taken on BWMNGMT.
cmdArg	An optional argument

Return Value CSL_Status

- CSL_SOK - Control info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

Both *CSL_bwmngmtInit()* and *CSL_bwmngmtOpen()* must be called successfully in that order before this function can be called

Post Condition

BWMNGMT registers are configured according to the command and the command arguments. The command determines which registers are modified

Modifies

The hardware registers of BWMNGMT

Example

```

CSL_BwmngmtHandle      hBwmngmt;
CSL_BwmngmtHwControlCmd cmd;
void                    *arg;

...
status = CSL_bwmngmtHwControl (hBwmngmt, cmd, &arg);
...

```

19.2.7 CSL_bwmngmtGetHwStatus

```

CSL_Status CSL_bwmngmtGetHwStatus ( CSL\_BwmngmtHandle      hBwmngmt,
                                     CSL\_BwmngmtHwStatusQuery myQuery,
                                     void *                      response
                                   )

```

Description

Gets the status of the different operations of BWMNGMT. Not Implemented. For future use.

Arguments

<i>hBwmngmt</i>	Handle to the BWMNGMT instance
<i>myQuery</i>	The query to this API of BWMNGMT which indicates the status to be returned.
<i>response</i>	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_INVQUERY - Invalid query

Pre Condition

Both *CSL_bwmngmtInit()* and *CSL_bwmngmtOpen()* must be called successfully in that order before this function can be called

Post Condition

None

Modifies

The input argument "response" is modified

Example

```
CSL_BwmngmtHandle      hBwmngmt ;
CSL_BwmngmtHwStatusQuery  query;
void                    *response;

...
status = CSL_bwmngmtGetHwStatus(hBwmngmt, query, &response);
...
```

19.3 Data Structures

This section lists the data structures available in the BWMNGMT module.

19.3.1 CSL_BwmngmtObj

Detailed Description

This object contains the reference to the instance of BWMNGMT opened using the `CSL_bwmngmtOpen()`. The pointer to this object is passed as BWMNGMT handle to all BWMNGMT CSL APIs. `CSL_bwmngmtOpen()` function initializes this structure based on the parameters passed.

Field Documentation

CSL_InstNum CSL_BwmngmtObj::bwmngmtNum

This is the instance of BWMNGMT being referred to by this object

CSL_BwmngmtRegsOvly CSL_BwmngmtObj::regs

This is a pointer to the registers of the instance of BWMNGMT referred to by CSL_BwmngmtObj object.

19.3.2 CSL_BwmngmtHwSetup

Detailed Description

CSL_BwmngmtHwSetup has all the fields required to configure BWMNGMT. This structure has the substructures required to configure BWMNGMT at Power-Up/Reset.

Field Documentation

[CSL_BwmngmtControlBlocks](#) **CSL_BwmngmtHwSetup::control**

Controller(s) to be set with Requestors Settings L1D, L2 and/or EXT

[CSL_BwmngmtMaxwait](#) **CSL_BwmngmtHwSetup::cpuMaxwait**

CPU - Requestor Arbitration Settings - MAXWAIT

[CSL_BwmngmtPriority](#) **CSL_BwmngmtHwSetup::cpuPriority**

CPU - Requestor Arbitration Settings - PRI

[CSL_BwmngmtMaxwait](#) **CSL_BwmngmtHwSetup::idmaMaxwait**

IDMA (Internal DMA) Requestor Arbitration Settings - MAXWAIT

[CSL_BwmngmtPriority](#) **CSL_BwmngmtHwSetup::mapPriority**

MAP (Master Port) Requestor Arbitration Settings - PRI

[CSL_BwmngmtMaxwait](#) **CSL_BwmngmtHwSetup::slapMaxwait**

SLAP (Slave Port) Requestor Arbitration Settings - MAXWAIT

[CSL_BwmngmtMaxwait](#) **CSL_BwmngmtHwSetup::ucMaxwait**

UC (User Coherence) Requestor Arbitration Settings – MAXWAIT

19.4 Enumerations

This section lists the enumerations available in the BWMNGMT module.

19.4.1 CSL_BwmngmtControlBlocks

enum CSL_BwmngmtControlBlocks

Control Block Set for BWMNGMT.

This is used to indicate which control blocks (L1D, L2, and/or EXT) are to be set within BWMNGMT for the given requestor (CPU, IDMA, SLAP, MAP, UC) arbitration settings.

Enumeration values:

<i>CSL_BWMNGMT_BLOCK_ALL</i>	All controller blocks will be update with given requestors arbitration setting
<i>CSL_BWMNGMT_BLOCK_L1D</i>	L1D controller block will be update with given requestors arbitration setting
<i>CSL_BWMNGMT_BLOCK_L2</i>	L2 controller block will be update with given requestors arbitration setting
<i>CSL_BWMNGMT_BLOCK_EXT</i>	EXT controller block will be update with given requestors arbitration setting

19.4.2 CSL_BwmngmtPriority

enum CSL_BwmngmtPriority

Priority Settings for BWMNGMT.

This is used to indicate to set the Priority arbitration settings for the Requestors (CPU, IDMA, SLAP, MAP, UC)

Enumeration values:

<i>CSL_BWMNGMT_PRI_0</i>	Priority arbitration setting 0 - Highest priority requestor
<i>CSL_BWMNGMT_PRI_1</i>	Priority arbitration setting 1 - 2nd Highest priority requestor
<i>CSL_BWMNGMT_PRI_2</i>	Priority arbitration setting 2 - 3rd Highest priority requestor
<i>CSL_BWMNGMT_PRI_3</i>	Priority arbitration setting 3 - 4th Highest priority requestor
<i>CSL_BWMNGMT_PRI_4</i>	Priority arbitration setting 4 - 5th Highest priority requestor
<i>CSL_BWMNGMT_PRI_5</i>	Priority arbitration setting 5 - 6th Highest priority requestor
<i>CSL_BWMNGMT_PRI_6</i>	Priority arbitration setting 6 - 7th Highest priority requestor
<i>CSL_BWMNGMT_PRI_7</i>	Priority arbitration setting 7 - Lowest priority requestor
<i>CSL_BWMNGMT_PRI_NULL</i>	Priority arbitration setting NULL - Due Not Program PRIORITY for this requestor

19.4.3 CSL_BwmngmtMaxwait

enum CSL_BwmngmtMaxwait

Maxwait Settings for BWMNGMT.

This is used to indicate to set Maxwait arbitration settings for the Requestors (CPU, IDMA, SLAP, MAP, UC).

Enumeration values:

<i>CSL_BWMNGMT_MAXWAIT_0</i>	Maxwait arbitration setting 0 - Always stall due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_1</i>	Maxwait arbitration setting 1 - Stall max of 1 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_2</i>	Maxwait arbitration setting 2 - Stall max of 2 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_4</i>	Maxwait arbitration setting 4 - Stall max of 4 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_8</i>	Maxwait arbitration setting 8 - Stall max of 8 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_16</i>	Maxwait arbitration setting 16 - Stall max of 16 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_32</i>	Maxwait arbitration setting 32 - Stall max of 32 cycle due to higher priority requestor
<i>CSL_BWMNGMT_MAXWAIT_NULL</i>	Maxwait arbitration setting NULL - Due Not Program MAXWAIT for this requestor

19.4.4 CSL_BwmngmtHwStatusQuery

enum CSL_BwmngmtHwStatusQuery

Enumeration for Hardware status query

Enumeration values:

<i>PLACEHOLDER0</i>	Placeholder for future implementation
---------------------	---------------------------------------

19.4.5 CSL_BwmngmtHwControlCmd

enum CSL_BwmngmtHwControlCmd

Enumeration for Hardware control command

Enumeration values:

<i>PLACEHOLDER1</i>	Placeholder for future implementation
---------------------	---------------------------------------

19.5 Macros

#define CSL_BWMNGMT_HWSETUP_DEFAULTS

Value:

```
{ \
    (CSL_BwmngmtPriority)CSL_BWMNGMT_CPUARBL1D_PRI_RESETVAL, \
    (CSL_BwmngmtMaxwait)CSL_BWMNGMT_CPUARBL1D_MAXWAIT_RESETVAL, \
    (CSL_BwmngmtMaxwait)CSL_BWMNGMT_IDMAARBL2_MAXWAIT_RESETVAL, \
    (CSL_BwmngmtMaxwait)CSL_BWMNGMT_SLAPARBL1D_MAXWAIT_RESETVAL, \
    (CSL_BwmngmtPriority)CSL_BWMNGMT_MAPARBEXT_PRI_RESETVAL, \
    (CSL_BwmngmtMaxwait)CSL_BWMNGMT_UCARBL1D_MAXWAIT_RESETVAL, \
    (CSL_BwmngmtControlBlocks)CSL_BWMNGMT_BLOCK_ALL \
}
```

The #define CSL_BWMNGMT_HWSETUP_DEFAULTS is meant to simplify the implementation in C code by the customer

19.6 Typedefs

typedef CSL_BwmngmtObj * CSL_BwmngmtHandle

This is a pointer to CSL_BwmngmtObj & is passed as the first parameter to all BWMNGMT CSL APIs

Chapter 20

MEMPROT MODULE

Topics

<u>20. 1 Overview</u>

<u>20. 2 Functions</u>
--

<u>20. 3 Data Structures</u>
--

<u>20. 4 Enumerations</u>

<u>20. 5 Macros</u>

20.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within MEMPROT module

Memory protection used to support resources (L1P, L2, L1D not an Internal CFG space).

Memory protection provides many benefits to a system. Memory protection functionality can:

- Protect operating system data structures from poorly behaving code.
- Aid in debugging by providing greater information about illegal memory accesses.
- Allow the operating system to enforce clearly defined boundaries between supervisor and user mode accesses, leading to greater system robustness.

20.2 Functions

This section lists the functions available in the MEMPROT module.

20.2.1 CSL_memprotInit

CSL_Status CSL_memprotInit ([CSL_MemprotContext](#) * *pContext*)

Description

This is the initialization function for the MEMPROT CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Context information for the instance. Should be NULL

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_memprotInit(NULL);
```

20.2.2 CSL_memprotOpen

[CSL_MemprotHandle](#) CSL_memprotOpen ([CSL_MemprotObj](#) * *pMemprotObj*,
CSL_InstNum *memprotNum*,
[CSL_MemprotParam](#) * *pMemprotParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the instance and returns a handle to the MEMPROT instance. The open call sets up the data structures for the particular instance of MEMPROT device. The device can be re-opened anytime after it has been normally closed, if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

pMemprotObj Pointer to the MEMPROT instance object

memprotNum	Instance of the MEMPROT to be opened
pMemprotParam	Pointer to module specific parameters
pStatus	Pointer for returning status of the function call

Return Value CSL_MemprotHandle

Valid MEMPROT instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition

Memory protection must be successfully initialized via *CSL_memprotInit()* before calling this function. Memory for the *CSL_MemprotObj* must be allocated outside this call. This object must be retained while usage of this module. Depending on the module opened some inherent constraints need to be kept in mind. When a handle for the Config block is opened the only operation possible is a query for the fault Status. No other control command/ query/ setup must be used. When a handle for L1D/L1P is opened, then the constraints with respect to the number of Memory pages must be kept in mind.

Post Condition

1. MEMPROT object structure is populated
2. The status is returned in the status variable. If status returned is
 - CSL_SOK - Valid MEMPROT module handle is returned
 - CSL_ESYS_FAIL - The MEMPROT instance is invalid
 - CSL_ESYS_INVPARAMS - Invalid parameter

Modifies

1. The status variable
2. MEMPROT object structure

Example

```
CSL_MemprotObj mpL2Obj;
CSL_MemprotHandle hmpL2;
CSL_Status status;

// Initializing the module
CSL_memprotInit(NULL);

// Opening the Handle for the L2
hmpL2 = CSL_memprotOpen(&mpL2Obj,
                        CSL_MEMPROT_L2,
                        NULL,
                        &status);
```

20.2.3 CSL_memprotClose

CSL_Status CSL_memprotClose ([CSL_MemprotHandle](#) *hMemprot*)

Description

This function closes the specified instance of MEMPROT.

Arguments

hMemprot Handle to the MEMPROT instance

Return Value CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

CSL_memprotInit(), CSL_memprotOpen() must be opened prior to this call.

Post Condition

The MEMPROT CSL APIs can not be called until the MEMPROT CSL is reopened again using CSL_memprotOpen().

Modifies

CSL_memprotObj structure values

Example

```
CSL_MemprotHandle    hMemprot;
CSL_Status           status;
CSL_MemprotObj       mpL2Obj;

hMemprot = CSL_memprotOpen(&mpL2Obj,
                           CSL_MEMPROT_L2,
                           NULL,
                           &status);

status = CSL_memprotClose(hMemprot);
```

20.2.4 CSL_memprotHwSetup

CSL_Status CSL_memprotHwSetup ([CSL_MemprotHandle](#) *hMemprot*,
[CSL_MemprotHwSetup](#) * *setup*
)

Description

This function initializes the module registers with the appropriate values provided through the HwSetup Data structure. For information passed through the HwSetup data structure, refer *CSL_memprotHwSetup*.

Arguments

hMemprot	Handle to the memprot instance
setup	Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition

Both *CSL_memprotInit()* and *CSL_memprotOpen()* must be called successfully in that order before this function can be called. The user has to allocate space for & fill in the main setup structure appropriately before calling this function. Ensure numpages is not set to > 32 for handles for L1D/L1P. Ensure numpages is not > 64 for L2.

Post Condition

1. MEMPROT object structure is populated
2. The status is returned in the status variable. If status returned is
 - CSL_SOK Valid MEMPROT handle is returned
 - CSL_ESYS_FAIL The MEMPROT instance is invalid
 - CSL_ESYS_INVPARAMS Invalid parameter

Modifies

The hardware registers of MEMPROT.

Example

```
#define PAGE_ATTR 0xFFFF0

CSL_MemprotObj mpL2Obj;
CSL_MemprotHandle hmpL2;
CSL_Status status;
CSL_MemprotHwSetup L2MpSetup;
Uint16 pageAttrTable[10] = { PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR};

Uint32 key[2] = {0x11223344, 0x55667788};
// Initializing the module
CSL_memprotInit(NULL);

// Opening the Handle for the L2
hmpL2 = CSL_memprotOpen(&mpL2Obj, CSL_MEMPROT_L2, NULL, &status);
L2MpSetup.memPageAttr = pageAttrTable;
L2MpSetup.numPages = 10;
L2MpSetup.key = key;

// Do Setup for the L2 Memory protection/
CSL_memprotHwSetup (hmpL2, &L2MpSetup);
```

20.2.5 CSL_memprotGetHwSetup

```
CSL_Status CSL_memprotGetHwSetup ( CSL\_MemprotHandle      hMemprot,
                                   CSL\_MemprotHwSetup * setup
                                   )
```

Description

This function gets the current setup of the Memory Protection registers. The status is returned through *CSL_MemprotHwSetup*. The obtaining of status is the reverse operation of *CSL_MemprotHwSetup()* function. Only the Memory Page attributes are read and filled into the HwSetup structure.

Arguments

<i>hMemprot</i>	Handle to the MEMPROT instance
<i>setup</i>	Pointer to setup structure which contains the setup information of MEMPROT.

Return Value *CSL_Status*

- *CSL_SOK* - Setup info load successful.
- *CSL_ESYS_BADHANDLE* - Invalid handle
- *CSL_ESYS_INVPARAMS* - Invalid parameter

Pre Condition

Both *CSL_memprotInit()* and *CSL_memprotOpen()* must be called successfully in that order before *CSL_memprotGetHwSetup()* can be called. Ensure *numpages* is initialized depending on the number of desired attributes in the setup. Make sure to set *numpages* <= 32 for handles for L1D/L1P. Ensure *numpages* <= 64 for L2.

Post Condition

The hardware setup structure is populated with the hardware setup parameters.

Modifies

None.

Example

```
#define PAGE_ATTR 0xFFFF0

CSL_MemprotObj mpL2Obj;
CSL_MemprotHandle hmpL2;
CSL_Status status;
CSL_MemprotHwSetup L2MpSetup, L2MpGetSetup;
Uint16 pageAttrTable[10] = { PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR};

Uint32 key[2] = {0x11223344, 0x55667788};

// Initializing the module
CSL_memprotInit(NULL);

// Opening the Handle for the L2
hmpL2 = CSL_memprotOpen(&mpL2Obj, CSL_MEMPROT_L2, NULL, &status);
```

```

L2MpSetup.memPageAttr = pageAttrTable;
L2MpSetup.numPages = 10;
L2MpSetup.key = key;

// Do Setup for the L2 Memory protection/
CSL_memprotHwSetup (hmpL2, &L2MpSetup);
CSL_memprotGetHwSetup(hmpL2, &L2MpGetSetup);

```

20.2.6 CSL_memprotHwControl

```

CSL_Status CSL_memprotHwControl ( CSL\_MemprotHandle          hMemprot,
                                  CSL\_MemprotHwControlCmd       cmd,
                                  void *                          arg
                                )

```

Description

Control operations for the Memory protection registers. For a particular control operation, the pointer to the corresponding data type needs to be passed as argument HwControl function Call. All the arguments (Structure elements included) passed to the HwControl function are inputs. For the list of commands supported and argument type that can be *void** casted & passed with a particular command refer to *CSL_MemprotHwControlCmd*.

Arguments

<i>hMemprot</i>	Handle to the MEMPROT instance
<i>cmd</i>	The command to this API indicates the action to be taken on MEMPROT.
<i>arg</i>	An optional argument

Return Value CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_FAIL - Invalid lock status

Pre Condition

Both *CSL_memprotInit()* and *CSL_memprotOpen()* must be called successfully in that order before *CSL_memprotHwControl()* can be called. For the argument type that can be *void** casted and passed with a particular command refer to *CSL_MemprotHwControlCmd*.

Post Condition

MEMPROT registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

The hardware registers of MEMPROT.

Example

```
#define PAGE_ATTR 0xFFFF0

Uint16 pageAttrTable[10] = { PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR};

Uint32 key[2] = {0x11223344,0x55667788};

CSL_MemprotObj mpL2Obj;
CSL_MemprotHandle hmpL2;
CSL_Status status;
CSL_MemprotHwSetup L2MpSetup,L2MpGetSetup;
CSL_MemprotLockStatus lockStat;
CSL_MemprotPageAttr pageAttr;
CSL_MemprotFaultStatus queryFaultStatus;

// Initializing the module
CSL_memprotInit(NULL);

// Opening the Handle for the L2
hmpL2 = CSL_memprotOpen(&mpL2Obj,CSL_MEMPROT_L2,NULL,&status);
L2MpSetup.memPageAttr = pageAttrTable;
L2MpSetup.numPages = 10;
L2MpSetup.key = key;

// Do Setup for the L2 Memory protection/
CSL_memprotHwSetup (hmpL2,&L2MpSetup);

// Query Lock Status
CSL_memprotGetHwStatus( hmpL2,
                       CSL_MEMPROT_QUERY_LOCKSTAT,
                       &lockStat);

// Unlock the Unit if Locked
if ((lockStat == CSL_MEMPROT_LOCKSTAT_LOCK)
{
    CSL_memprotHwControl(hmpL2,CSL_MEMPROT_CMD_UNLOCK,key);
}
```

20.2.7 CSL_memprotGetHwStatus

CSL_Status CSL_memprotGetHwStatus	(<u>CSL_MemprotHandle</u> <u>CSL_MemprotHwStatusQuery</u> void *)	<i>hMemprot,</i> <i>query,</i> <i>response</i>
--	---	---

Description

This function is used to read the current module configuration, status flags and the value present associated registers. User should allocate memory for the said data type and pass its pointer as an unadorned void* argument to the status query call. For details about the various status queries supported and the associated data structure to record the response, refer to *CSL_MemprotHwStatusQuery*.

Arguments

hMemprot	Handle to the MEMPROT instance
query	The query to this API of MEMPROT which indicates the status to be returned.
response	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Usage Constraints

Both *CSL_memprotInit()* and *CSL_memprotOpen()* must be called successfully in that order before *CSL_memprotGetHwStatus()* can be called. For the argument type that can be void* casted and passed with a particular command refer to *CSL_MemprotHwStatusQuery*.

Post Condition

None

Modifies

Third parameter "response" vlaue

Example

```
#define PAGE_ATTR 0xFFFF0

Uint16 pageAttrTable[10] = { PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR,
                             PAGE_ATTR};

Uint32 key[2] = {0x11223344, 0x55667788};
CSL_MemprotObj mpL2Obj;
CSL_MemprotHandle hmpL2;
CSL_Status status;
CSL_MemprotHwSetup L2MpSetup, L2MpGetSetup;
CSL_MemprotLockStatus lockStat;
CSL_MemprotPageAttr pageAttr;
CSL_MemprotFaultStatus queryFaultStatus;
```

```
// Initializing the module
CSL_memprotInit(NULL);

// Opening the Handle for the L2
hmpL2 = CSL_memprotOpen(&mpL2Obj,CSL_MEMPROT_L2,NULL,&status);
L2MpSetup.memPageAttr = pageAttrTable;
L2MpSetup.numPages = 10;
L2MpSetup.key = key;

// Do Setup for the L2 Memory protection/
CSL_memprotHwSetup (hmpL2,&L2MpSetup);

// Query Lock Status
CSL_memprotGetHwStatus( hmpL2,
                        CSL_MEMPROT_QUERY_LOCKSTAT,
                        &lockStat);
```

20.2.8 CSL_memprotGetBaseAddress

```
CSL_Status CSL_memprotGetBaseAddress( CSL_InstNum      memprotNum,
                                     CSL\_MemprotParam * pMemprotParam,
                                     CSL\_MemprotBaseAddress * pBaseAddress
                                     )
```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_memprotOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral. MMRs go to an alternate location.

Arguments

memprotNum	Specifies the instance of the memprot to be opened.
pMemprotParam	Module specific parameters.
pBaseAddress	Pointer to base address structure containing base address details.

Return Value CSL_Status

- CSL_SOK - Successfully retrieved base address
- CSL_ESYS_FAIL - The instance number is invalid.
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status status;  
CSL_MemprotBaseAddress  baseAddress;  
  
...  
status = CSL_memprotGetBaseAddress( CSL_MEMPROT_L2,  
                                   NULL,  
                                   &baseAddress );
```


20.3 Data Structures

This section lists the data structures available in the MEMPROT module.

20.3.1 CSL_MemprotObj

Detailed Description

This object contains the reference to the instance of memory Protection Module opened using the *CSL_memprotOpen()*. A pointer to this object is passed to all Memory Protection CSL APIs.

Field Documentation

CSL_InstNum CSL_MemprotObj::modNum

This is the instance of module number i.e. L2/L1D/L1P/CONFIG

CSL_MemprotRegsOvly CSL_MemprotObj::regs

This is a pointer to the memory protection registers of the module for which memory protection is requested.

20.3.2 CSL_MemprotContext

Detailed Description

Module specific context information. Present implementation doesn't have any Context information.

Field Documentation

UInt16 CSL_MemprotContext::contextInfo

Context information of Memory Protection. The declaration is just a placeholder for future implementation.

20.3.3 CSL_MemprotHwSetup

Detailed Description

This is the setup structure used with the HwSetup API.

Field Documentation

UInt32* CSL_MemprotHwSetup::key

This should point to an array of 2 32 bit elements (constituting the key)

UInt16* CSL_MemprotHwSetup::memPageAttr

This should point to a table of memory page attributes

UInt16 CSL_MemprotHwSetup::numPages

This is the number of pages which need to be programmed starting from 0

20.3.4 CSL_MemprotBaseAddress

Detailed Description

This will have the base-address information for the module instance.

Field Documentation**CSL_MemprotRegsOvly CSL_MemprotBaseAddress::regs**

Base-address of the memory protection registers

20.3.5 CSL_MemprotFaultStatus

Detailed Description

This will be used to query the memory fault status.

Field Documentation**UInt32 CSL_MemprotFaultStatus::addr**

Memory Protection Fault Address

CSL_BitMask16 CSL_MemprotFaultStatus::errorMask

Bit Mask of the Errors

UInt16 CSL_MemprotFaultStatus::fid

Faulted ID

20.3.6 CSL_MemprotPageAttr

Detailed Description

This will be used to set/query the memory page attributes.

Field Documentation**CSL_BitMask16 CSL_MemprotPageAttr::attr**

Memory Protection Page attributes

UInt16 CSL_MemprotPageAttr::page

Memory Protection Page number

20.3.7 CSL_MemprotParam

Detailed Description

This is module specific parameter. Present implementation of Memprot CSL doesn't have any module specific parameters.

Field Documentation**CSL_BitMask16 CSL_MemprotParam::flags**

Bit mask to be used for module specific parameters. The declaration is just a placeholder for future implementation. Passed as an argument to CSL_memprotOpen().

20.4 Enumerations

This section lists the enumerations available in the MEMPROT module.

20.4.1 CSL_MemprotHwStatusQuery

enum CSL_MemprotHwStatusQuery

Enumeration for queries passed to *CSL_memprotGetHwStatus()*.

This is used to get the status of different operations or the current register settings.

Enumeration values:

<i>CSL_MEMPROT_QUERY_FAULT</i>	Gets the fault status from the unit. Parameters: (<i>CSL_MemprotFaultStatus *</i>)
<i>CSL_MEMPROT_QUERY_PAGEATTR</i>	Get the memory protection page attributes. Parameters: (<i>CSL_MemprotPageAttr *</i>)
<i>CSL_MEMPROT_QUERY_LOCKSTAT</i>	Memory protection Lock status. Parameters: (<i>CSL_MemprotLockStatus *</i>)

20.4.2 CSL_MemprotHwControlCmd

enum CSL_MemprotHwControlCmd

Enumeration for commands passed to *CSL_memprotHwControl()*.

This is used to select the commands to control the operations in the Module.

Enumeration values:

<i>CSL_MEMPROT_CMD_LOCK</i>	Locks the Memory Protection Unit (command argument Parameters: <i>Uint32*</i> (An array of 2 32 bits elements constituting the key))
<i>CSL_MEMPROT_CMD_UNLOCK</i>	Unlocks the Memory Protection Unit (command argument Parameters: <i>Uint32*</i> (An array of 2 32 bits elements constituting the key))
<i>CSL_MEMPROT_CMD_PAGEATTR</i>	Sets the page attributes Parameters: (<i>CSL_MemprotPageAttr*</i>)

20.4.3 CSL_MemprotLockStatus

enum CSL_MemprotLockStatus

Enumeration for queried lock status.

Enumeration values:

<i>CSL_MEMPROT_LOCKSTAT_LOCK</i>	Non secure Lock
<i>CSL_MEMPROT_LOCKSTAT_UNLOCK</i>	Non secure UnLock

20.5 Macros

#define CSL_MEMPROT_MEMACCESS_AID0 0x0400

Allowed ID '0'

#define CSL_MEMPROT_MEMACCESS_AID1 0x0800

Allowed ID '1'

#define CSL_MEMPROT_MEMACCESS_AID2 0x1000

Allowed ID '2'

#define CSL_MEMPROT_MEMACCESS_AID3 0x2000

Allowed ID '3'

#define CSL_MEMPROT_MEMACCESS_AID4 0x4000

Allowed ID '4'

#define CSL_MEMPROT_MEMACCESS_AID5 0x8000

Allowed ID '5'

#define CSL_MEMPROT_MEMACCESS_EXT 0x0200

External Allowed ID. VBus requests with PrivID >= '6' are permitted if access type is allowed

#define CSL_MEMPROT_MEMACCESS_LOCAL 0x0100

Local Access

#define CSL_MEMPROT_MEMACCESS_SR 0x0020

Supervisor Read permission

#define CSL_MEMPROT_MEMACCESS_SW 0x0010

Supervisor Write permission

#define CSL_MEMPROT_MEMACCESS_SX 0x0008

Supervisor Execute permission

#define CSL_MEMPROT_MEMACCESS_UR 0x0004

User Read permission

#define CSL_MEMPROT_MEMACCESS_UW 0x0002

User Write permission

#define CSL_MEMPROT_MEMACCESS_UX 0x0001

User Execute permission

Chapter 21

CFG MODULE

Topics

<u>21. 1 Overview</u>
<u>21. 2 Functions</u>
<u>21. 3 Data Structures</u>
<u>21. 4 Enumerations</u>
<u>21. 5 Macros</u>
<u>21.6 Typedefs</u>

21.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within CFG module.

This module provides memory protection for Internal configuration space. If Invalid write accesses to reserved regions of Internal configuration Space will generate an exception. If a series of protection faults occurs to the CFG space, only that first such violation is recorded and only one exception is generated via the Extended Memory Controller CPU memory protection fault interrupt. Once this fault is cleared, a new protection violation will result in its information being recorded and new exception being generated.

21.2 Functions

This section lists the functions available in the CFG module.

21.2.1 CSL_cfgInit

CSL_Status CSL_cfgInit (**CSL_CfgContext *** *pContext*)

Description

This is the initialization function for the CFG CSL. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext Context information for the instance. Should be NULL

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_cfgInit(NULL);
```

21.2.2 CSL_cfgOpen

[CSL_CfgHandle](#) CSL_cfgOpen ([CSL_CfgObj *](#) *pCfgObj*,
CSL_InstNum *cfgNum*,
CSL_CfgParam * *pBwmParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of CFG device. The device can be re-opened anytime after it has been normally closed, if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

pCfgObj Pointer to the CFG instance object

cfgNum	Instance of the CFG to be opened.
pCfgParam	Pointer to module specific parameters
pStatus	Pointer for returning status of the function call

Return Value CSL_CfgHandle

Valid CFG instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition

CSL_cfgInit has to be called before calling this function.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid CFG handle is returned.
- CSL_ESYS_FAIL - The CFG instance is invalid.
- CSL_ESYS_INVPARAMS Invalid parameters

2. CFG object structure is populated.

Modifies

1. The status variable
2. CFG object structure

Example

```
CSL_Status      status;
CSL_CfgObj     cfgObj;
CSL_CfgHandle  hCfg;

hCfg = CSL_cfgOpen (&cfgObj, CSL_MEMPROT_CONFIG, NULL, &status);
```

21.2.3 CSL_cfgClose

CSL_Status CSL_cfgClose ([CSL_CfgHandle](#) **hCfg**)

Description

This function closes the specified instance of CFG.

Arguments

hCfg Handle to the CFG instance

Return Value CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

Both *CSL_cfgInit()* and *CSL_cfgOpen()* must be called successfully in that order before *CSL_cfgClose()* can be called.

Post Condition

The CFG CSL APIs can not be called until the CFG CSL is reopened again using *CSL_cfgOpen()*.

Modifies

CSL_cfgObj structure values

Example

```

CSL_Status      status;
CSL_CfgObj      cfgObj;
CSL_CfgHandle   hCfg;

hCfg = CSL_cfgOpen (&cfgObj, CSL_MEMPROT_CONFIG, NULL, &status);

CSL_cfgClose(hCfg);

```

21.2.4 CSL_cfgHwControl

```

CSL_Status CSL_cfgHwControl ( CSL\_CfgHandle           hCfg,
                             CSL\_CfgHwControlCmd        cmd,
                             void *                       arg
                             )

```

Description

Takes a command of CFG with an optional argument and implements it.

Arguments

hCfg	Handle to the CFG instance
cmd	The command to this API indicates the action to be taken on CFG.
arg	An optional argument.

Return Value CSL_Status

- CSL_SOK - Command successful.
- CSL_ESYS_INVCMD - Invalid command
- CSL_ESYS_BADHANDLE - Invalid parameter

Pre Condition

 Both *CSL_cfgInit()* and *CSL_cfgOpen()* must be called successfully in that order before *CSL_cfgHwControl()* can be called.

Post Condition

CFG registers are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

The registers of CFG.

Example

```

CSL_CfgHandle   hCfg;
CSL_Status      status;
CSL_CfgObj      cfgObj;
CSL_CfgHwControlCmd cmd = CSL_CFG_CMD_CLEAR;

```

```
hCfg = CSL_cfgOpen (&cfgObj, CSL_MEMPROT_CONFIG, NULL, &status);

status = CSL_cfgHwControl(hCfg, cmd, NULL);
```

21.2.5 CSL_cfgGetHwStatus

```
CSL_Status CSL_cfgGetHwStatus      ( CSL\_CfgHandle           hCfg,
                                     CSL\_CfgHwStatusQuery    query,
                                     void *                          response
                                     )
```

Description

Gets the status of the different operations of CFG.

Arguments

hCfg	Handle to the CFG instance
query	The query to this API of CFG which indicates the status to be returned.
response	Placeholder to return the status.

Return Value CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_INVQUERY - Invalid query command
- CSL_ESYS_INVPARAMS - Invalid parameter
- CSL_ESYS_BADHANDLE – Invalid Handle

Pre Condition

Both *CSL_cfgInit()* and *CSL_cfgOpen()* must be called successfully in that order before *CSL_cfgGetHwStatus()* can be called.

Post Condition

None

Modifies

Third parameter “response” vlaue

Example

```
CSL_CfgHandle      hCfg;
CSL_CfgHwStatusQuery query = CSL_CFG_QUERY_FAULT_ADDR;
CSL_Status          status;
CSL_CfgObj          cfgObj;
void                response;

hCfg = CSL_cfgOpen (&cfgObj, CSL_MEMPROT_CONFIG, NULL, &status);

status = CSL_cfgGetHwStatus(hCfg, query, &response);
```

21.2.6 CSL_cfgGetBaseAddress

```
CSL_Status CSL_cfgGetBaseAddress ( CSL_InstNum      cfgNum,
                                   CSL_CfgParam *    pCfgParam,
                                   CSL_CfgBaseAddress * pBaseAddress
                                   )
```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the CSL_cfgOpen() function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMRs go to an alternate location.

Arguments

cfgNum	Specifies the instance of the CFG for which the base address is requested
pCfgParam	Module specific parameters
pBaseAddress	Pointer to the base address structure to return the base address details

Return Value

CSL_Status

- CSL_SOK - Successful on getting the base address of CFG
- CSL_ESYS_FAIL - The CFG instance is not available.

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure

Example

```
CSL_Status      status;
CSL_CfgBaseAddress  baseAddress;

...
status = CSL_cfgGetBaseAddress(CSL_MEMPROT_CONFIG,
                               NULL,
                               &baseAddress);
...
```

21.3 Data Structures

This section lists the data structures available in the CFG module.

21.3.1 CSL_CfgObj

Detailed Description

This object contains the reference to the instance of CFG opened using the *CSL_cfgOpen()*. The pointer to this is passed as CFG Handle to all CFG CSL APIs. *CSL_cfgOpen()* function initializes this structure based on the parameters passed

Field Documentation

CSL_InstNum CSL_CfgObj::cfgNum

This is the instance of CFG being referred to by this object

CSL_CfgRegsOvly CSL_CfgObj::regs

This is a pointer to the registers of the instance of CFG referred to by this object

21.3.2 CSL_CfgFaultStatus

Detailed Description

CSL_CfgStatus has all the fields required for the status information of CFG module.

Field Documentation

CSL_BitMask16 CSL_CfgFaultStatus::errorMask

Bit Mask of the Errors

UInt16 CSL_CfgFaultStatus::faultId

Fault Id. The ID of the originator of the faulting access

21.4 Enumerations

This section lists the enumerations available in the CFG module.

21.4.1 CSL_CfgHwControlCmd

enum CSL_CfgHwControlCmd

Enumeration for queries passed to *CSL_cfgHwControl()*.

This is used to select the commands to control the operations existing setup of CFG. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

CSL_CFG_CMD_CLEAR CFG Hardware control command to clears the error conditions stored in MPFAR and MPFSR.

Parameters:

None

21.4.2 CSL_CfgHwStatusQuery

enum CSL_CfgHwStatusQuery

Enumeration for queries passed to *CSL_cfgGetHwStatus()*.

This is used to get the status of different operations or to get the existing setup of CFG.

Enumeration values:

CSL_CFG_QUERY_FAULT_ADDR Status query command to get the Fault Address.

Parameters:

*(Uint16 *)*

CSL_CFG_QUERY_FAULT_STATUS Status query command to get the Status information of CSL_CfgStatus.

Parameters:

*(CSL_CfgStatus *)*

21.5 Macros

#define CSL_CFG_FAULT_STAT_FID (0x0000F700u)

Mask value of Fault ID

#define CSL_CFG_FAULT_STAT_LOCAL (0x00000080u)

Mask value to get the status of Local memory (L1/L2)

#define CSL_CFG_FAULT_STAT_SR (0x00000020u)

Mask value for Supervisor Read

#define CSL_CFG_FAULT_STAT_SW (0x00000010u)

Mask value for Supervisor Write

#define CSL_CFG_FAULT_STAT_SX (0x00000008u)

Mask value for Supervisor Execute

#define CSL_CFG_FAULT_STAT_UR (0x00000004u)

Mask value for User Read

#define CSL_CFG_FAULT_STAT_UW (0x00000002u)

Mask value for User Write

#define CSL_CFG_FAULT_STAT_UX (0x00000001u)

Mask value for User Execute

21.6 Typedefs

typedef CSL_CfgObj * CSL_CfgHandle

This is a pointer to CSL_CfgObj & is passed as the first parameter to all CFG CSL APIs

Chapter 22

PWRDWN MODULE

Topics

<u>22. 1 Overview</u>

<u>22. 2 Functions</u>
--

<u>22. 3 Data Structures</u>
--

<u>22. 4 Enumerations</u>

22.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within PWRDWN module.

The power-down controller allows software-driven power-down management for all of the C64x+ megamodule components. The CPU can power-down part or all of the C64x+ megamodule through the power-down controller based on its own execution thread or in response to an external stimulus from a host or global controller. These power-down features can be used to design systems for lower overall system power requirements.

22.2 Functions

This section lists the functions available in the PWRDWN module.

22.2.1 CSL_pwrdownInit

CSL_Status CSL_pwrdownInit ([CSL_PwrdownContext](#) * *pContext*)

Description

CSL_pwrdownInit(.) initializes the PWRDWN module. The function must be called before calling any other API from this CSL. This function does not modify any registers or check status. It returns status CSL_SOK. It has been kept for future use.

Arguments

pContext	Pointer to module-context. As PWRDWN doesn't have any context based information user is expected to pass NULL.
----------	--

Return Value CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_pwrdownInit(NULL);
```

22.2.2 CSL_pwrdownOpen

[CSL_PwrdownHandle](#) CSL_pwrdownOpen ([CSL_PwrdownObj](#) * *pPwrdownObj*,
CSL_InstNum *pwrdownNum*,
[CSL_PwrdownParam](#) * *pPwrdownParam*,
CSL_Status * *pStatus*
)

Description

This function populates the peripheral data object for the PWRDWN instance and returns a handle to the instance. The open call sets up the data structures for the particular instance of PWRDWN device. The device can be re-opened anytime after it has been normally closed, if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

pPwrdownObj	Pointer to PWRDWN object.
pwrdownNum	Instance of pwrdown CSL to be opened. There are three instance of the PWRDWN available. So, the value for this parameter will be based on the instance.
pPwrdownParam	Module specific parameters
pStatus	Status of the function call

Return Value CSL_pwrdownHandle

Valid pwrdown handle will be returned if status value is equal to CSL_SOK.

Pre Condition

CSL_pwrdownInit() must be called prior to this call.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK Valid pwrdown handle is returned
- CSL_ESYS_FAIL The pwrdown instance is invalid
- CSL_ESYS_INVPARAMS Invalid parameters.

2. Pwrdown object structure is populated.

Modifies

1. The status variable
2. pwrdown object structure

Example

```
CSL_PwrdownObj    pwrObj;
CSL_Status        status;
CSL_PwrdownConfig pwrConfig;
CSL_PwrdownHandle hPwr;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments fof the Config structure
...
```

22.2.3 CSL_pwrdownClose

CSL_Status CSL_pwrdownClose ([CSL_PwrdownHandle](#) *hPwrdown*)

Description

This function closes the specified instance of pwrdown.

Arguments

hPwrdown Handle to the PWRDWN instance

Return Value CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

CSL_pwrdownInit(), CSL_pwrdownOpen() must be opened prior to this call.

Post Condition

1. The PWRDWN CSL APIs can not be called until the PWRDWN CSL is reopened again using CSL_pwrdownOpen()

Modifies

CSL_pwrdownObj structure values

Example

```
CSL_PwrdownObj    pwrObj;
CSL_Status        status;
CSL_PwrdownConfig pwrConfig;
CSL_PwrdownHandle hPwr;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen(&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments fof the Config structure
...
// Close
CSL_pwrdownClose(hPwr);
```

22.2.4 CSL_pwrdownHwSetup

CSL_Status **CSL_pwrdownHwSetup** ([CSL_PwrdownHandle](#) *hPwrdown*,
[CSL_PwrdownHwSetup](#) * *setup*
)

Description

It configures the PWRDWN instance registers as per the values passed in the hardware setup structure.

Arguments

hPwrdown	Handle to the pwrdown instance
setup	Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition

CSL_pwrdownInit(), CSL_pwrdownOpen() must be opened prior to this call.

Post Condition

PWRDWN registers instance will be setup according to value passed.

Modifies

PWRDWN hardware registers

Example

```

CSL_PwrdownObj      pwrObj;
CSL_Status           status;
CSL_PwrdownHwSetup  pwrSetup;
CSL_PwrdownHandle   hPwr;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments for the Setup structure
...

// Setup
CSL_pwrdownHwSetup(hPwr, &pwrSetup);

// Close handle
CSL_pwrdownClose(hPwr);

```

22.2.5 CSL_pwrdownGetHwSetup

CSL_Status **CSL_pwrdownGetHwSetup** ([CSL_PwrdownHandle](#) *hPwrdown*,
[CSL_PwrdownHwSetup](#) * *setup*
)

Description

It retrieves the hardware setup parameters.

Arguments

hPwrdown Handle to the PWRDWN instance

setup Pointer to hardware setup structure

Return Value CSL_Status

- CSL_SOK - Hardware setup retrieved
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS – Invalid Parameters.

Pre Condition

CSL_pwrdownInit(), CSL_pwrdownOpen() must be opened prior to this call.

Post Condition

The hardware set up structure will be populated with values from the registers.

Modifies

Second parameter "setup"

Example

```

CSL_PwrdownObj      pwrObj;
CSL_Status           status;
CSL_PwrdownHwSetup  pwrSetup, querySetup;
CSL_PwrdownHandle   hPwr;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments for the Setup structure
...

// Setup
CSL_pwrdownHwSetup(hPwr, &pwrSetup);

// Query Setup
CSL_pwrdownGetHwSetup(hPwr, &querySetup);

// Close handle
CSL_pwrdownClose(hPwr);

```

22.2.6 CSL_pwrdownGetHwStatus

CSL_Status	CSL_pwrdownGetHwStatus	(<u>CSL_PwrdownHandle</u>	<i>hPwrdown,</i>
			<u>CSL_PwrdownHwStatusQuery</u>	<i>query,</i>
			void *	<i>response</i>
)		

Description

This function is used to get the value of various parameters of the PWRDWN instance. The value returned depends on the query passed.

Arguments

hPwrdown	Handle to the PWRDWN instance
query	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value CSL_Status

- CSL_SOK - Successful completion of the query

-
- CSL_ESYS_BADHANDLE - Invalid handle
 - CSL_ESYS_INVQUERY - Query command not supported

Pre Condition

CSL_pwrdownInit(), CSL_pwrdownOpen() must be opened prior to this call.

Post Condition

Data requested by the query is returned through the variable "response".

Modifies

The input argument "response" is modified.

Example

```

CSL_PwrdownObj      pwrObj;
CSL_Status           status;
CSL_PwrdownHwSetup  pwrSetup;
CSL_PwrdownHandle    hPwr;
CSL_PwrdownPortData  pageSleep;

pageSleep.portNum = 0x0;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments for the Setup structure
...

// Setup
CSL_pwrdownHwSetup(hPwr, &pwrSetup);

// Hw Status Query
CSL_pwrdownGetHwStatus( hPwr,
                        CSL_PWRDWN_QUERY_PAGE0_STATUS,
                        &pageSleep);

// Close handle
CSL_pwrdownClose(hPwr);

```

22.2.7 CSL_pwrdownHwSetupRaw

CSL_Status **CSL_pwrdownHwSetupRaw** ([CSL_PwrdownHandle](#) *hPwrdown*,
[CSL_PwrdownConfig](#) * *config*
)

Description

This function initializes the device registers with the register-values provided through the config data structure. This configures registers based on a structure of register values, as compared to HwSetup, which configures registers based on structure of bit field values

Arguments

hPwrDwn	Pointer to the object that holds reference to the instance of PWRDWN requested after the call
config	Pointer to the config structure containing the device register values

Return Value CSL_Status

- CSL_SOK - Configuration successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Configuration structure pointer is not properly initialized

Pre Condition

CSL_pwrDwnInit(), CSL_pwrDwnOpen() must be opened prior to this call.

Post Condition

The registers of the specified PWRDWN instance will be setup according to the values passed through the config structure.

Modifies

Hardware registers of the specified PWRDWN instance

Example

```

CSL_PwrDwnObj      pwrObj;
CSL_Status          status;
CSL_PwrDwnConfig   pwrConfig;
CSL_PwrDwnHandle    hPwr;
// Init Module
CSL_pwrDwnInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrDwnOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments for the Config structure
...

// Setup
CSL_pwrDwnHwSetupRaw(hPwr, &pwrConfig);

// Close handle
CSL_pwrDwnClose(hPwr);

```

22.2.8 CSL_pwrDwnGetBaseAddress

```

CSL_Status CSL_pwrDwnGetBaseAddress ( CSL_InstNum      pwrDwnNum,
                                     CSL\_PwrDwnParam * pPwrDwnParam,
                                     CSL\_PwrDwnBaseAddress * pBaseAddress
                                     )

```

Description

This function gets the base address of the given pwrDwn instance.

Arguments

pwrdownNum	Specifies the instance of the pwrdown to be opened.
pPwrdownParam	pwrdown module specific parameters.
pBaseAddress	Pointer to base address structure containing base address details.

Return Value CSL_Status

- CSL_SOK - Successfully retrieved base address
- CSL_ESYS_FAIL - pwrdown instance is not available.
- CSL_ESYS_INVPARAMS - Invalid Parameter

Pre Condition

None

Post Condition

Base address structure is populated.

Modifies

1. The status variable
2. Base address structure is modified.

Example

```

CSL_PwrdownHandle    hPwrdown;
CSL_PwrdownBaseAddress baseAddress;
CSL_PwrdownParam      params;

CSL_pwrdownGetBaseAddress(CSL_PWRDWN, &params, &baseAddress) ;

```

22.2.9 CSL_pwrdownHwControl

```

CSL_Status CSL_pwrdownHwControl ( CSL\_PwrdownHandle      hPwrdown,
                                   CSL\_PwrdownHwControlCmd cmd,
                                   void *          arg
                                   )

```

Description

This function performs various control operations on the PWRDWN instance based on the command passed.

Arguments

hPwrdown	Handle to the PWRDWN instance
cmd	Operation to be performed on the PWRDWN
arg	Argument specific to the command

Return Value CSL_Status

-
- CSL_SOK - Command execution successful
 - CSL_ESYS_INVCMD - Invalid command
 - CSL_ESYS_BADHANDLE - Invalid parameter

Pre Condition

CSL_pwrdownInit(), CSL_pwrdownOpen() must be opened prior to this call

Post Condition

Registers of the PWRDWN instance are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

Registers determined by the command

Example

```

CSL_PwrdownObj          pwrObj;
CSL_PwrdownHwSetup      pwrSetup;
CSL_PwrdownHandle       hPwr;
CSL_PwrdownPortData     pageSleep;
CSL_Status              status;

// Init Module
CSL_pwrdownInit(NULL);

// Opening a handle for the Module
hPwr = CSL_pwrdownOpen (&pwrObj, CSL_PWRDWN, NULL, &status);

// Setup the arguments for the Setup structure
...

// Setup
CSL_pwrdownHwSetup(hPwr, &pwrSetup);

// Hw Control
pageSleep.portNum = 0x1;
pageSleep.data = 0x0;

CSL_pwrdownHwControl(hPwr, CSL_PWRDWN_CMD_PAGE0_SLEEP, &pageSleep);

// Close handle
CSL_pwrdownClose(hPwr);

```

22.3 Data Structures

This section lists the data structures available in the PWRDWN module.

22.3.1 CSL_PwrdownObj

Detailed Description

This object contains the reference to the instance of PWRDWN opened using the *CSL_pwrdownOpen()*.

Field Documentation

CSL_InstNum CSL_PwrdownObj::instNum

This is the instance of PWRDWN being referred to by this object

CSL_L2pwrdownRegsOvly CSL_PwrdownObj::l2pwrdownRegs

This is a pointer to the registers of the instance of L2 PWRDWN referred to by this object

CSL_PdcRegsOvly CSL_PwrdownObj::pdcRegs

This is a pointer to the registers of the instance of PDC referred to by this object

22.3.2 CSL_PwrdownConfig

Detailed Description

The config-structure.Used to configure the PWRDWN using *CSL_pwrdownHwSetupRaw(..)*.This is a structure of register values, rather than a structure of register field values like *CSL_PwrdownHwSetup*

Field Documentation

UInt32 CSL_PwrdownConfig::L2PDSLEEP0

Per page manual sleep for port0

UInt32 CSL_PwrdownConfig::L2PDSLEEP1

Per page manual sleep for port1

UInt32 CSL_PwrdownConfig::L2PDWAKE0

Per page manual awake for port0

UInt32 CSL_PwrdownConfig::L2PDWAKE1

Per page manual awake for port1

UInt32 CSL_PwrdownConfig::PDCCMD

Power down command register

22.3.3 CSL_PwrdownContext

Detailed Description

Module specific context information. Present implementation doesn't have any Context information.

Field Documentation
Uint16 CSL_PwrdownContext::contextInfo

Context information of PWRDWN. The declaration is just a placeholder for future implementation. This is a Dummy.

22.3.4 CSL_PwrdownHwSetup

Detailed Description

This has all the fields required to configure PWRDWN at Power Up (After a Hardware Reset) or a Soft Reset. This structure is used to setup or obtain existing setup of PWRDWN using *CSL_pwrdownHwSetup()* and *CSL_pwrdownGetHwSetup()* functions respectively.

Field Documentation
Bool CSL_PwrdownHwSetup::idlePwrdown

Idle powerdown

[CSL_PwrdownL2Manual*](#) CSL_PwrdownHwSetup::manualPwrdown

Manual power down setup

22.3.5 CSL_PwrdownParam

Detailed Description

Module specific parameters. None in this implementation.

Field Documentation
void* CSL_PwrdownParam::futureUse

Perhaps useful for future use

22.3.6 CSL_PwrdownBaseAddress

Detailed Description

This will have the base-address information for the module instance.

Field Documentation
CSL_L2pwrdownRegsOvly CSL_PwrdownBaseAddress::l2pwrdownRegs

Base-address of the L2 Powerdown registers

CSL_PdcRegsOvly CSL_PwrdownBaseAddress::regs

Base-address of the PDC registers

22.3.7 CSL_PwrdownPortData

Detailed Description

This will have the port specific information. It contains port number and data used in *CSL_pwrdownGetHwStatus()* and *CSL_pwrdownHwControl()*

Field Documentation

Bool CSL_PwrdownPortData::portNum
Port number

CSL_BitMask8 CSL_PwrdownPortData::data
8-bit mask

22.3.8 CSL_PwrdownL2Manual

Detailed Description

The manual powerdown setup structure.

Field Documentation

CSL_BitMask8 CSL_PwrdownL2Manual::port0PageSleep
Bitmask of the pages that need to be put to sleep on UMAP0

CSL_BitMask8 CSL_PwrdownL2Manual::port0PageWake
Bitmask of the pages that need to be woken on UMAP0

CSL_BitMask8 CSL_PwrdownL2Manual::port1PageSleep
Bitmask of the pages that need to be put to sleep on UMAP1

CSL_BitMask8 CSL_PwrdownL2Manual::port1PageWake
Bitmask of the pages that need to be woken on UMAP1

22.4 Enumerations

This section lists the enumerations available in the PWRDWN module.

22.4.1 CSL_PwrdownHwStatusQuery

enum CSL_PwrdownHwStatusQuery

Default values for the config-structure Enumeration for queries passed to *CSL_pwrdownGetHwStatus()*. This is used to get the status of different operations or to get the existing setup of PWRDWN.

Enumeration values:

<i>CSL_PWRDWN_QUERY_PAGE0_STATUS</i>	Gets the page0 sleep status Parameters: (<i>CSL_PwrdownPortData *</i>)
<i>CSL_PWRDWN_QUERY_PAGE1_STATUS</i>	Gets the page1 sleep status Parameters: (<i>CSL_PwrdownPortData *</i>)

22.4.2 CSL_PwrdownHwControlCmd

enum CSL_PwrdownHwControlCmd

Enumeration for queries passed to *CSL_pwrdownHwControl()*. This is used to select the commands to control the operations existing setup of PWRDWN. The arguments to be passed with each enumeration if any are specified next to the enumeration.

Enumeration values:

<i>CSL_PWRDWN_CMD_PAGE0_SLEEP</i>	Manual power down, port0 or port1, page0 sleep Parameters: (<i>CSL_PwrdownPortData *</i>)
<i>CSL_PWRDWN_CMD_PAGE1_SLEEP</i>	Manual power down, port0 or port1, page1 sleep Parameters: (<i>CSL_PwrdownPortData *</i>)
<i>CSL_PWRDWN_CMD_PAGE0_WAKE</i>	Manual power down, port0 or port1, page0 wake Parameters: (<i>CSL_PwrdownPortData *</i>)
<i>CSL_PWRDWN_CMD_PAGE1_WAKE</i>	Manual power down, port0 or port1, page1 wake Parameters: (<i>CSL_PwrdownPortData *</i>)

Chapter 23 DTF Module

Topics

<u>23. 1 Overview</u>
<u>23. 2 Functions</u>
<u>23. 3 Data Structure</u>
<u>23.4 Enumerations</u>
<u>23.5 Macros</u>
<u>23.6 Typedefs</u>

23.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within DTF module.

GEM's trace port sends the trace data to DTF (DSP Trace Formatter). This trace data is 22bits per clock cycle in FullGEM. The ETB has a 32bit data interface. The primary function of DTF is to pack the 22bit trace data into a 32bit packet. All the DTF's MMR's are located in the chip level register space and thus it does not have a VBUS interface. DTF also forwards the trace data as is to the SPM (Static Pin Merge).

23.2 Functions

23.2.1 CSL_dtfInit

CSL_Status CSL_dtfInit (**CSL_DtfContext *** *pContext*)

Description

This is the initialization function for the DTF. This function must be called before calling any other API from this CSL. This function is idem-potent. Currently, the function just returns status CSL_SOK, without doing anything.

Arguments

pContext Context information for the instance. Should be NULL

Return Value

CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status status;
...
status = CSL_dtfInit(NULL);
...
```

23.2.2 CSL_dtfOpen

[CSL_DtfHandle](#) CSL_dtfOpen (**CSL_DtfObj *** *pDtfObj,*
 CSL_InstNum *dtfNum,*
 CSL_DtfParam * *pDtfParam,*
 CSL_Status * *pStatus*
)

Description

This function returns the handle to the DTF controller instance. This handle is passed to all other CSL APIs.

Arguments

pDtfObj Pointer to dtf object.

dtfNum Instance of DSP DTF to be opened.
 There are three instances of the dtf
 available.

pDtfParam	Module specific parameters.
pStatus	Status of the function call

Return Value

CSL_DtfHandle

- Valid dtf handle will be returned if status value is equal to CSL_SOK.

Pre Condition
CSL_dtfInit() must be called successfully in order before calling **CSL_dtfOpen()**.

Post Condition

1. The status is returned in the status variable. If status returned is

- CSL_SOK - Valid dtf handle is returned
- CSL_ESYS_FAIL - The dtf instance is invalid
- CSL_ESYS_INVPARAMS - Invalid parameter

2. DTF object structure is populated

Modifies

- The status variable
- DTF object structure

Example

```

CSL_Status      status;
CSL_DtfObj      dtfObj;
CSL_DtfHandle   hDtf;
...
hDtf = CSL_dtfOpen(&dtfObj, CSL_DTF_0, NULL, &status);
...

```

23.2.3 CSL_dtfClose

CSL_Status CSL_dtfClose ([CSL_DtfHandle](#) hDtf)
Description

This function closes the specified instance of DTF.

Arguments

hDtf	Handle to the DTF
------	-------------------

Return Value

CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

Both **CSL_dtfInit()** and **CSL_dtfOpen()** must be called successfully in order before calling **CSL_dtfClose()**.

Post Condition

The DTF CSL APIs can not be called until the DTF CSL is reopened again using `CSL_dtfOpen()`.

Modifies

Obj structure values

Example

```
CSL_DtfHandle      hDtf;
CSL_Status          status;
...

status = CSL_dtfClose(hDtf);
...
```

23.2.4 CSL_dtfHwControl

```
CSL_Status CSL_dtfHwControl ( CSL\_DtfHandle          hDtf,
                             CSL\_DtfControlCmd       cmd,
                             void *                  arg
                             )
```

Description

This function takes an input control command with an optional argument and accordingly controls the operation/configuration of DTF.

Arguments

<code>hDtf</code>	DTF handle returned by successful 'open'
<code>cmd</code>	The command to this API indicates the action to be taken on DTF. Control command, refer @a <code>CSL_DtfControlCmd</code> for the list of commands supported
<code>arg</code>	An optional argument. Optional argument as per the control command, @a void * casted

Return Value

`CSL_Status`

- `CSL_SOK` - Status info return successful.
- `CSL_ESYS_BADHANDLE` - Invalid handle
- `CSL_ESYS_INVCMD` - Invalid command

Pre Condition

Both `CSL_dtfInit()` and `CSL_dtfOpen()` must be called successfully in order before calling `CSL_dtfHwControl()`.

Refer to `CSL_DtfHwControlCmd` for the argument type (`void*`) that needs to be passed with the control command

Post Condition

None

Modifies

The hardware registers of DTF.

Example

```

CSL_Status status;
Uint32 arg;
CSL_DtfHandle hDtf;
...
// Init successfully done
...
// Open successfully done
...

arg = 1;
status = CSL_dtfHwControl(hDtf,
                           CSL_DTF_CMD_SET_DTFENABLE,
                           &arg);
...

```

23.2.5 CSL_dtfGetHwStatus

```

CSL_Status CSL_dtfGetHwStatus ( CSL\_DtfHandle          hDtf,
                                CSL\_DtfHwStatusQuery query,
                                void *                response
                                )

```

Description

Gets the status of different operations or some setup-parameters of DTF. The status is returned through the third parameter.

Arguments

hDtf	DTF handle returned by successful 'open'
query	The query to this API of DTF which indicates the status to be returned. Query command, refer @a CSL_DtfHwStatusQuery for the list of query commands supported
response	Response from the query. Pointer to appropriate object corresponding to the query command needs to be passed here

Return Value

CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command

Pre Condition

Both **CSL_dtfInit()** and **CSL_dtfOpen()** must be called successfully in order before calling **CSL_dtfGetHwStatus()**. Refer to *CSL_DtfHwStatusQuery* for the argument to be passed along with the corresponding query command.

Post Condition

None

Modifies

Third parameter response

Example

```
CSL_DtfHandle hDtf;
CSL_Status status;
Uint32 response;
...

status = CSL_dtfGetHwStatus(hDtf,
                           CSL_DTF_QUERY_DTFOWN_STATUS,
                           &response);
...
```

23.2.6 CSL_GetBaseAddress

```
CSL_Status CSL_dtfGetBaseAddress (
                                CSL_InstNum          dtfNum,
                                CSL_DtfParam *        pDtfParam,
                                CSL_DtfBaseAddress *   pBaseAddress
                                )
```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the **CSL_dtfOpen()** function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral. MMR's go to an alternate location.

Arguments

dtfNum	Specifies the instance of the dtf to be opened.
pDtfParam	Module specific parameters.
pBaseAddress	Pointer to baseaddress structure containing base address details.

Return Value

CSL_Status

- CSL_SOK - Successful on getting the base address of dtf
- CSL_ESYS_FAIL - The instance number is invalid.

Pre Condition

None

Post Condition

Base Address structure is populated

Modifies

1. The status variable
2. Base address structure is modified.

Example

```
CSL_Status      status;  
CSL_DtfBaseAddress  baseAddress;  
...  
status = CSL_dtfGetBaseAddress(CSL_DTF_0, NULL, &baseAddress);  
...
```

23.3 Data Structures

23.3.1 CSL_DtfBaseAddress

This structure will have the base-address information for the peripheral instance.

Detailed Description

This structure will have the base-address information for the peripheral instance.

Field Documentation

CSL_DtfRegsOvly CSL_DtfBaseAddress::regs

Base-address of the Configuration registers of DTF.

23.3.2 CSL_DtfContext

Detailed Description

DTF specific context information. Present implementation doesn't have any Context information.

Field Documentation

UInt16 CSL_DtfContext::contextInfo

Context information of DTF. The below declaration is just a place-holder for future implementation.

23.3.3 CSL_DtfObj

Detailed Description

This structure/object holds the context of the instance of DTF opened using **CSL_dtfOpen()** function.

Pointer to this object is passed as DTF Handle to all DTF CSL APIs. **CSL_dtfOpen()** function initializes this structure based on the parameters passed

Field Documentation

CSL_InstNum CSL_DtfObj::perNum

Instance of DTF being referred by this object

CSL_DtfRegsOvly CSL_DtfObj::regs

Pointer to the register overlay structure of the DTF

23.3.4 CSL_DtfParam

Detailed Description

DTF specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_DtfParam::flags

Bit mask to be used for module specific parameters. The below declaration is just a place-holder for future implementation.

23.4 Enumerations

23.4.1 CSL_DtfControlCmd

This is the set of control commands that are passed to **CSL_dtfHwControl()** , with an optional argument type-casted to *void**

The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<i>CSL_DTF_CMD_ENA_AOWN_APP</i>	Setup the DTF ownership to Application Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_ENA_AOWN_EMU</i>	Setup the DTF ownership to Emulation Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_FLUSH_DTFFLUSH</i>	Setup the DTF Flush Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_SET_DTFENABLE</i>	Setting the DTFENABLE Parameters: 0 - Disable 1 - Enable Returns: CSL_SOK
<i>CSL_DTF_CMD_DIS_SPMDISABLE</i>	Disable trace output to Static-Pin_merge Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_ENA_SPMDISABLE</i>	Enable trace output to Static-Pin_merge Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_RELEASE_OWNERSHIP</i>	Releasing the DTF ownership Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_CLAIM_OWNERSHIP</i>	Claiming the DTF ownership Parameters: <i>None</i> Returns: CSL_SOK
<i>CSL_DTF_CMD_ENA_OWNERSHIP</i>	Enabling the DTF ownership Parameters: <i>None</i> Returns: CSL_SOK

23.4.2 CSL_DtfHwStatusQuery

This is the set of query commands to get the status of various operations in DTF
The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

CSL_DTF_QUERY_AOWNERSHIP

Queries the DTF ownership

Parameters: (*Uint32 **)

Returns: CSL_SOK

*CSL_DTF_QUERY_DTFOWN_
STATUS*

Queries DTF ownership state control

Parameters: (*Uint32 **)

Returns: CSL_SOK

CSL_DTF_QUERY_DTFENABLE

Queries DTFENABLE bit

Parameters: (*Uint32 **)

Returns: CSL_SOK

CSL_DTF_QUERY_SPMDISABLE

Queries Static-Pin-Merger

Parameters: (*Uint32 **)

Returns: CSL_SOK

23.5 Macros

#define CSL_DTF_AOWN_APPLICATION 1
Application owns DTF ownership

#define CSL_DTF_AOWN_EMULATION 0
Debugger owns DTF ownership

#define CSL_DTF_OWNERSHIP_RELEASE 0
Value to release the DTF ownership

#define CSL_DTF_OWNERSHIP_CLAIM 1
Value to claim the DTF ownership

#define CSL_DTF_OWNERSHIP_ENABLE 2
Value to enable the DTF ownership

#define CSL_DTF_DTFENABLE_DISABLE 0
Value to disable DTFENABLE

#define CSL_DTF_DTFENABLE_ENABLE 1
Value to enable DTFENABLE

#define CSL_DTF_SPM_DISABLE 0
Value to disable the SPM

#define CSL_DTF_SPM_ENABLE 1
Value to enable the SPM

23.6 Typedefs

typedef struct CSL_DtfObj CSL_DtfObj

This structure/object holds the context of the instance of DTF opened using **CSL_dtfOpen()** function.

typedef CSL_DtfObj * CSL_DtfHandle

This is a pointer to **CSL_DtfObj** and is passed as the first parameter to all DTF CSL APIs.

Chapter 24 ETB Module

Topics

<u>24. 1 Overview</u>
<u>24. 2 Functions</u>
<u>24. 3 Data Structures</u>
<u>24.4 Enumerations</u>
<u>24.5 Macros</u>
<u>24.6 Typedefs</u>

24.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within FTB module.

The purpose of the embedded trace buffer (ETB) is to provide a buffering mechanism to hold the PC discontinuity trace data before the data leaves the chip. The buffer is a dedicated buffer on chip. In this case, there is a 4KB buffer per core.

24.2 Functions

24.2.1 CSL_etbInit

CSL_Status CSL_etbInit ([CSL_EtbContext](#) * *pContext*)

Description

This is the initialization function for the ETB. This function must be called before calling any other API from this CSL. This function is idem-potent. Currently, the function just returns status CSL_SOK, without doing anything.

Arguments

pContext Context information for the instance. Should be NULL

Return Value

CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
CSL_Status status;
...
status = CSL_etbInit(NULL);
...
```

24.2.2 CSL_etbOpen

CSL_EtbHandle CSL_etbOpen (**CSL_EtbObj *** *pEtbObj,*
 CSL_InstNum *etbNum,*
 CSL_EtbParam * *pEtbParam,*
 CSL_Status * *pStatus*
)

Description

This function returns the handle to the ETB controller instance. This handle is passed to all other CSL APIs. After successful open, unlocks ETB i.e., by setting the LAR register of ETB in order to enable accesses to any ETB registers.

Arguments

pEtbObj Pointer to etb object.

etbNum Instance of DSP ETB to be opened.

There are three instances of the etb available. The instances are CSL_ETB_0, CSL_ETB_1 and CSL_ETB_2.

pEtbParam	Module specific parameters.
pStatus	Status of the function call

Return Value

CSL_EtbHandle

- Valid etb handle will be returned if status value is equal to CSL_SOK.

Pre Condition

CSL_etbInit() must be called successfully in order before calling **CSL_etbOpen()**.

Post Condition

- The status is returned in the status variable. If status returned is
 - CSL_SOK - Valid etb handle is returned
 - CSL_ESYS_FAIL - The etb instance is invalid
 - CSL_ESYS_INVPARAMS - Invalid parameter
- ETB object structure is populated

Modifies

- The status variable
- ETB object structure

Example

```

CSL_Status      status;
CSL_EtbObj      etbObj;
CSL_EtbHandle   hEtb;

...

hEtb = CSL_etbOpen(&etbObj,
                  CSL_ETB_0,
                  NULL,
                  &status
                  );

...

```

24.2.3 CSL_etbclose

CSL_Status CSL_etbClose ([CSL_EtbHandle](#) *hEtb*)

Description

This function closes the specified instance of ETB. After successful close, locks ETB i.e., by resetting the LAR register of ETB in order to disable accesses to any ETB registers.

Arguments

hEtb	Handle to the ETB
------	-------------------

Return Value

CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Pre Condition

Both **CSL_etbInit()** and **CSL_etbOpen()** must be called successfully in order before calling **CSL_etbClose()**.

Post Condition

The ETB CSL APIs can not be called until the ETB CSL is reopened again using *CSL_etbOpen()*.

Modifies

Obj structure values

Example

```

CSL_EtbHandle      hEtb;
CSL_Status          status;
...

status = CSL_etbClose(hEtb);
...
```

24.2.4 CSL_etbHwControl

```

CSL_Status CSL_etbHwControl ( CSL\_EtbHandle      hEtb,
                             CSL\_EtbControlCmd   cmd,
                             void *             arg
                             )
```

Description

This function takes an input control command with an optional argument and accordingly controls the operation/configuration of ETB.

Arguments

hEtb	ETB handle returned by successful 'open'
cmd	The command to this API indicates the action to be taken on ETB. Control command, refer @a CSL_EtbControlCmd for the list of commands supported
arg	An optional argument. Optional argument as per the control command, void * casted

Return Value

CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle

-
- CSL_ESYS_INVCMD - Invalid command
 - CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

Both **CSL_etbInit()** and **CSL_etbOpen()** must be called successfully in order before calling **CSL_etbHwControl()**.

Refer to *CSL_EtbHwControlCmd* for the argument type (*void**) that needs to be passed with the control command

Post Condition

None

Modifies

The hardware registers of ETB.

Example

```
CSL_Status      status;
Uint32          arg;
CSL_EtbHandle   hEtb;
...

// Init successfully done
...
// Open successfully done
...

arg = CSL_ETB_TRACEACPEN_ENABLE;
status = CSL_etbHwControl(hEtb,
                           CSL_ETB_CMD_ENA_TRACE_CAPTURE,
                           &arg);
...
```

24.2.5 CSL_etbGetHwStatus

```
CSL_Status CSL_etbGetHwStatus ( CSL\_EtbHandle           hEtb,
                                CSL\_EtbHwStatusQuery query,
                                void *                response
                                )
```

Description

Gets the status of different operations or some setup-parameters of ETB. The status is returned through the third parameter.

Arguments

hEtb	ETB handle returned by successful 'open'
query	The query to this API of ETB which indicates the status to be returned. Query command, refer @a CSL_EtbHwStatusQuery for the list of query commands supported

response	Response from the query. Pointer to appropriate object corresponding to the query command needs to be passed here
----------	---

Return Value

CSL_Status

- CSL_SOK - Status info return successful
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Invalid query command

Pre Condition

Both **CSL_etbInit()** and **CSL_etbOpen()** must be called successfully in order before calling **CSL_etbGetHwStatus()**. Refer to *CSL_EtbHwStatusQuery* for the argument to be passed along with the corresponding query command.

Post Condition

None

Modifies

Third parameter response

Example

```

CSL_EtbHandle    hEtb;
CSL_Status       status;
Uint32          response;
...
status = CSL_etbGetHwStatus( hEtb,
                             CSL_ETB_QUERY_ACQUISITION_COMPLETE,
                             &response);
...

```

24.2.6 CSL_etbRead

```

CSL_Status CSL_etbRead ( CSL\_EtbHandle          hEtb,
                        void *          rdData
                        )

```

Description

This function reads ETB data.

Arguments

hEtb	ETB handle returned by successful 'open'
rdData	read data from the RRD

Return Value

CSL_Status

- CSL_SOK – Read operation successful.
- CSL_ESYS_BADHANDLE - Invalid handle

Both **CSL_etbInit()** and **CSL_etbOpen()** must be called successfully in order before calling **CSL_etbRead()**.

None

None

```
CSL_Status      status;  
Uint32          rdData;  
CSL_EtbHandle  hEtb;  
...  
  
// Init successfully done  
...  
// Open successfully done  
...  
  
status = CSL_etbRead(hEtb, &rdData);  
...
```

24.2.7 CSL_etbWrite

```
CSL_Status CSL_etbWrite ( CSL_EtbHandle hEtb,  
                          void * wrData  
                          )
```

This function writes the specified data into ETB data register.

hEtB	ETB handle returned by successful 'open'
wrData	write data into the RWD

CSL_Status

- CSL_SOK - Success (doesnot verify written data).
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Invalid parameter

Both **CSL_etbInit()** and **CSL_etbOpen()** must be called successfully in order before calling **CSL_etbWrite()**.

Data is written to ETB RAM Write register

Modifies

ETB register.

Example

```

CSL_Status      status;
Uint32          arg;
CSL_EtbHandle   hEtb;
...

// Init successfully done
...
// Open successfully done
...

status = CSL_etbWrite(hEtb, &wrData);
...
```

24.2.8 CSL_etbGetBaseAddress

```

CSL_Status CSL_etbGetBaseAddress (      CSL_InstNum      etbNum,
                                       CSL_EtbParam *    pEtbParam,
                                       CSL_EtbBaseAddress * pBaseAddress
                                       )
```

Description

Function to get the base address of the peripheral instance. This function is used for getting the base address of the peripheral instance. This function will be called inside the **CSL_etbOpen()** function call. This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral. MMR's go to an alternate location.

Arguments

<code>etbNum</code>	Specifies the instance of the etb to be opened.
<code>pEtbParam</code>	Module specific parameters.
<code>pBaseAddress</code>	Pointer to baseaddress structure containing base address details.

Return Value

CSL_Status

- CSL_SOK - Successful on getting the base address of etb
- CSL_ESYS_FAIL - The instance number is invalid.

Pre Condition

None

Post Condition

Base Address structure is populated

Modifies

1. The status variable

2. Base address structure is modified.

Example

```
CSL_Status          status;  
CSL_EtbBaseAddress  baseAddress;  
...  
status = CSL_etbGetBaseAddress(CSL_ETB_0, NULL, &baseAddress);
```

24.3 Data Structures

24.3.1 CSL_EtbBaseAddress

Detailed Description

This structure will have the base-address information for the peripheral instance.

Field Documentation

CSL_EtbRegsOvly CSL_EtbBaseAddress::regs
Base-address of the Configuration registers of ETB.

24.3.2 CSL_EtbContext

Detailed Description

ETB specific context information. Present implementation doesn't have any Context information.

Field Documentation

Uint16 CSL_EtbContext::contextInfo
Context information of ETB. The below declaration is just a place-holder for future implementation.

24.3.3 CSL_EtbObj

Detailed Description

This structure/object holds the context of the instance of ETB opened using **CSL_etbOpen()** function.

Pointer to this object is passed as ETB Handle to all ETB CSL APIs. **CSL_etbOpen()** function initializes this structure based on the parameters passed

Field Documentation

CSL_InstNum CSL_EtbObj::perNum
Instance of ETB being referred by this object

CSL_EtbRegsOvly CSL_EtbObj::regs
Pointer to the register overlay structure of the ETB

24.3.4 CSL_EtbParam

Detailed Description

ETB specific parameters. Present implementation doesn't have any specific parameters.

Field Documentation

CSL_BitMask16 CSL_EtbParam::flags
Bit mask to be used for module specific parameters. The below declaration is just a place-holder for future implementation.

24.4 Enumerations

24.4.1 CSL_EtbControlCmd

This is the set of control commands that are passed to **CSL_etbHwControl()**, with an optional argument type-casted to *void**

The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

CSL_ETB_CMD_SET_RAM_RD_POINTER	Setup RAM read pointer register Parameters: <i>arg – value to set Trace RAM</i> Returns: CSL_SOK
CSL_ETB_CMD_SET_RAM_WR_POINTER	Setup RAM write pointer register Parameters: <i>arg – value to set the Trace RAM</i> Returns: CSL_SOK
CSL_ETB_CMD_SET_TRIGG_COUNT	Setup Trigger Counter Register Parameters: <i>arg – value to set the trigger counter</i> Returns: CSL_SOK
CSL_ETB_CMD_ENA_TRACE_CAPTURE	Enable Trace Capture Parameters: <i>None</i> Returns: CSL_SOK
CSL_ETB_CMD_DIS_TRACE_CAPTURE	Disable Trace Capture Parameters: <i>None</i> Returns: CSL_SOK
CSL_ETB_CMD_SET_ENAFORMATTING	Enable Formatting Parameters: <i>arg (0-Disable, 1-Enable)</i> Returns: CSL_SOK
CSL_ETB_CMD_SET_CONT_FORMATTING	Setup Continuous Formatting Parameters: <i>arg (0-Disable, 1-Enable)</i> Returns: CSL_SOK
CSL_ETB_CMD_ENA_FLUSHIN	Enable FLUSHIN Parameters: <i>None</i> Returns: CSL_SOK
CSL_ETB_CMD_SET_MANUAL_FLUSH	Setup Manual Flush - FONMAN Parameters: <i>None</i> Returns: CSL_SOK
CSL_ETB_CMD_ENA_STOP_FLUSH	Enable stop flush - STOPFI Parameters: <i>None</i> Returns: CSL_SOK

<i>CSL_ETB_CMD_SET_ACQCOMP</i>	Setup the value of ACQCOMP Parameters: <i>arg</i> (1 –ACQ Complete 0- ACQ not complete) Returns: CSL_SOK
<i>CSL_ETB_CMD_SET_FULL</i>	Setup the value of FULL Parameters: <i>arg</i> (0-not Full, 1-Full) Returns: CSL_SOK
<i>CSL_ETB_CMD_SET_TRIGACK</i>	Setup the value of TRIGACK Parameters: <i>arg</i> (0-TrigInAck disable, 1-TrigInACK enable) Returns: CSL_SOK
<i>CSL_ETB_CMD_SET_FLUSHACK</i>	Setup the value of FLUSHACK Parameters: <i>arg</i> (0-FlushInAck disable, 1-FlushInACK enable) Returns: CSL_SOK
<i>CSL_ETB_CMD_SET_ATREADY</i>	Setup the value of ATREADY Parameters: <i>arg</i> (0-Disable, 1-Enable) Returns: CSL_SOK
<i>CSL_ETB_CMD_SET_ATVALID</i>	Setup the value of ATVALID Parameters: <i>arg</i> (0-Disable, 1-Enable) Returns: CSL_SOK

24.4.2 CSL_EtbHwStatusQuery

This is the set of query commands to get the status of various operations in ETB
The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<i>CSL_ETB_QUERY_RAM_FULL</i>	Queries the RAM Full (RAM write pointer has wrapped around) Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK
<i>CSL_ETB_QUERY_TRIG_STAUS</i>	Queries the Trigger bit set when a trigger has been observed Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK
<i>CSL_ETB_QUERY_ACQUISITION_COMPLETE</i>	Queries the Acquisition complete Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK
<i>CSL_ETB_QUERY_FORMAT_PIPELINE</i>	Queries the Formatterpipeline empty, All data stored to RAM Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK
<i>CSL_ETB_QUERY_FLUSH</i>	Queries the Flush in progress

	Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_FORMAT_STOP	Queries the Formatter stopped Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_ITATBDATA0_STATUS	Queries the Integration register Status ITATBDATA0 Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_ITATBCTR1_STATUS	Queries the Integration register Status ITATBCTR1 Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_ITATBCTR0_STATUS	Queries the Integration register Status ITATBCTR0 Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_SECURITY_LEVEL	Queries the Reports security level Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_DEVICE_ID	Queries the DID - Device ID Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_DEVICE_TYPE	Queries the Device Type Identification Register Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_TRIGIN_VALUE	Queries the value of TRIGIN Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_FLUSHIN_VALUE	Queries the value of FLUSHIN Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_RAM_DEPTH	Queries the value of RAM DEPTH Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK
CSL_ETB_QUERY_READ_POINTER	Queries the value of READ Pointer Parameters: <i>(Uint32 *)</i>

Returns:CSL_SOK

CSL_ETB_QUERY_WRITE_POINTER Queries the value of WRITE POINTER
Parameters:(*Uint32 **)
Returns:CSL_SOK

CSL_ETB_QUERY_TRACECAP_STATUS Queries the value of TRACE Status
Parameters:(*Uint32 **)
Returns:CSL_SOK

CSL_ETB_QUERY_TRIGGERCOUNT Queries the value of Trigger Counter
Parameters:(*Uint32 **)
Returns:CSL_SOK

CSL_ETB_QUERY_ENAFORMATTING Queries the value of Enable Formatting bit
Parameters:(*Uint32 **)
Returns:CSL_SOK

CSL_ETB_QUERY_ENACONTFORMATTING Queries the value of Enable Continuous Formatting bit
Parameters:(*Uint32 **)
Returns:CSL_SOK

24.5 Macros

#define CSL_ETB_STS_ACQ_COMPLETE Acquisition complete	1
#define CSL_ETB_STS_ACQ_NOTCOMPLETE Acquisition NOT complete	0
#define CSL_ETB_UNLOCK_VAL Value to unlock ETB for register accesses	(0xc5acce55u)
#define CSL_ETB_TRACEACPEN_ENABLE Value for trace capture enable	1
#define CSL_ETB_TRACEACPEN_DISABLE Value for trace capture disable	0

24.6 Typedefs

typedef struct CSL_EtbObj CSL_EtbObj

This structure/object holds the context of the instance of ETB opened using **CSL_etbOpen()** function

Chapter 25 CIC Module

Topics

<u>25.1 Overview</u>
<u>25.2 Functions</u>
<u>25.3 Data Structures</u>
<u>25.4 Enumerations</u>
<u>25.5 Macros</u>
<u>25.6 Typedefs</u>

25.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within CIC module.

25.2 Functions

25.2.1 CSL_cicInit

CSL_Status CSL_cicInit ([CSL_CicContext](#) * *pContext*)

Description

This is the initialization function for the CIC. This function is idempotent in that calling it many times is same as calling it once. This function initializes the CSL data structures, and doesn't affect the H/W.

Arguments

pContext Pointer to module-context structure

Return Value

CSL_Status

- CSL_SOK - Always returns

Pre Condition

None

Post Condition

None

Modifies

None

Example

```
// Init Module
...
if (CSL_cicInit(&context) != CSL_SOK) {
    exit;
}
```

25.2.2 CSL_cicOpen

CSL_CicHandle CSL_cicOpen (**CSL_CicObj *** *cicObj,*
 CSL_CicEventId *eventId,*
 CSL_CicParam * *param,*
 CSL_Status * *status*
)

Description

The API would reserve an interrupt-event for use. It returns a valid handle to the event only if the event is not currently allocated. The user could release the event after use by calling CSL_cicClose(..)

Arguments

cicObj Pointer to the CSL-object allocated by the user

eventId The event-id of the interrupt

param	Pointer to the Cic specific parameter
status	Pointer for returning status of the function call

Return Value

CSL_CicHandle

- Valid CIC handle identifying the event

Pre Condition

None

Post Condition

- CIC object structure is populated
- The status is returned in the status variable. If status returned is

CSL_SOK Valid cic handle is returned

CSL_ESYS_FAIL The open command failed

Modifies

- The status variable
- CIC object structure

Example

```

CSL_CicObj cicObj20;
CSL_CicContext context;
CSL_Status intStat;
CSL_CicHandle hCic20;
CSL_CicParam param;

// Init Module
CSL_cicInit(&context);

// Opening a handle for the CSL_CIC_EVENTID_FSEVT18 at Cic Output
Event ID 4
param.eventId = CSL_CIC_EVENTID_FSEVT18;
param.ectlEvtId = CSL_CIC_ECTL_EVT4;

hCic20 = CSL_cicOpen(&cicObj20, CSL_CIC_0, &param, &intStat);

// Close handle
CSL_cicClose(hCic20);

```

25.2.3 CSL_cicClose

 CSL_Status CSL_cicClose ([CSL_CicHandle](#) hCic)

Description

This cic Handle can no longer be used to access the event. The event is de-allocated and further access to the event resources are possible only after opening the event object again.

Arguments

hCic	Handle identifying the event
------	------------------------------

Return Value

CSL_Status

-
- CSL_SOK - Close successful
 - CSL_CIC_BADHANDLE - The handle passed is invalid

Pre Condition

Functions **CSL_cicInit()** and **CSL_cicOpen()** have to be called in that order successfully before calling this function.

Post Condition

The cic CSL APIs can not be called until the cic CSL is reopened again using **CSL_cicOpen()**

Modifies

None

Example

```

CSL_Status          intStat;
CSL_CicParam        param;
CSL_CicObj          cicObj20;
CSL_CicHandle       hCic20;
CSL_CicContext      context;
// Init Module
...
if (CSL_cicInit(&context) != CSL_SOK) {
exit;
//Opening a handle for the CSL_CIC_EVENTID_FSEVT18 at Cic Output
Event ID 4
param.eventId = CSL_CIC_EVENTID_FSEVT18;
param.ectlEvtId = CSL_CIC_ECTL_EVT4;

hCic20 = CSL_cicOpen(&cicObj20, CSL_CIC_0, &param, &intStat);

// Close handle
CSL_CicClose(hCic20);

```

25.2.4 CSL_cicGetHwStatus

```

CSL_Status CSL_cicGetHwStatus ( CSL\_CicHandle          hCic,
                                CSL\_CicHwStatusQuery       myQuery,
                                void *                    answer
                                )

```

Description

Queries the peripheral for status. The **CSL_cicGetHwStatus(..)** API could be used to retrieve status or configuration information from the peripheral. The user must allocate an object that would hold the retrieved information and pass a pointer to it to the function. The type of the object is specific to the query-command.

Arguments

hCic	Handle identifying the event
query	The query to this API of CIC which indicates the status to be returned.
answer	Placeholder to return the status.

Return Value

CSL Status

- CSL_SOK - Status info return successful
- CSL_ESYS_INVQUERY - Invalid query
- CSL_ESYS_INVPARAMS - Invalid parameter

Pre Condition

None

Post Condition

None

Modifies

None

Example

```

CSL_CicContext          context;
CSL_Status              intStat;
CSL_CicParam            param;
CSL_CicObj              cicObj20;
CSL_CicHandle           hCic20;
CSL_CicGlobalEnableState state;
Uint32                  intrStat;

// Init Module
...
if (CSL_cicInit(&context) != CSL_SOK)
exit;
// Opening a handle for the CSL_CIC_EVENTID_FSEVT18 at Cic Output
Event ID 4
param.eventId = CSL_CIC_EVENTID_FSEVT18;
param.ectlEvtId = CSL_CIC_ECTL_EVT4;

hCic20 = CSL_cicOpen (&cicObj20, CSL_CIC_0, &param, &intStat);

do {
CSL_cicGetHwStatus(hCic20, CSL_CIC_QUERY_PENDSTATUS, \
                    (void*)&intrStat);
} while (!stat);

// Close handle
CSL_CicClose(hCic20);

```

25.2.5 CSL cicHwControl

```
CSL_Status CSL_cicHwControl ( CSL\_CicHandle hCic,  
                             CSL_CicHwControlCmd controlCommand,  
                             void * commandArg  
                             )
```

Description

Perform a control-operation. This API is used to invoke any of the supported control-operations supported by the module.

Arguments

hCic	Handle identifying the event
------	------------------------------

<code>controlCommand</code>	The command to this API indicates the action to be taken on CIC.
<code>commandArg</code>	An optional argument.

Return Value`CSL_Status`

- `CSL_SOK` - HwControl successful.
- `CSL_ESYS_BADHANDLE` - Invalid handle
- `CSL_ESYS_INVCMD` - Invalid command

Pre Condition`CSL_cicOpen()` must be invoked before this call.**Post Condition**

None

Modifies

The hardware registers of CIC.

Example

```
CSL_CicObj      cicObj20;
CSL_CicGlobalEnableState state;
CSL_CicContext  context;
CSL_Status      intStat;
CSL_CicHandle   hCic20;
CSL_CicParam    param;
// Init Module
CSL_cicInit(&context);

// Opening a handle for the CSL_CIC_EVENTID_FSEVT18 at Cic Output
Event ID 4
param.eventId = CSL_CIC_EVENTID_FSEVT18;
param.ectlEvtId = CSL_CIC_ECTL_EVT4;

hCic20 = CSL_cicOpen (&cicObj20, CSL_CIC_0, &param, &intStat);

CSL_cicHwControl(hCic20, CSL_CIC_CMD_EVTENABLE, NULL);
```

25.3 Data Structures

25.3.1 CSL_CicContext

Detailed Description

CIC Module Context.

Field Documentation

CSL_BitMask32 CSL_CicContext::eventAllocMask[$\text{NUM_CIC_INST} \left[\frac{(\text{CSL_CIC_EVENTID_CNT} + 31)}{32} \right]$]

Event allocation mask

25.3.2 CSL_CicObj

Detailed Description

The interrupt handle object This object is used by the handle to identify the event.

Field Documentation

CSL_CicEctlEvtId CSL_CicObj::ectlEvtId

The vector-id

CSL_CicEventId CSL_CicObj::eventId

The event-id

CSL_CicRegsOvly CSL_CicObj::regs

base address of CIC instant used

CSL_InstNum CSL_CicObj::cicNum

IC instance id, Range 0,1..3

25.3.3 CSL_CicBaseAddress

Detailed Description

Holds the base-address information for CIC peripheral instance.

Field Documentation

CSL_CicRegsOvly CSL_CicBaseAddress::regs

Base-address of the Configuration registers of CIC

25.3.4 CSL_CicParam

Detailed Description

this is equivalent to the Vector Id for the event number.

Field Documentation

CSL_CicEctlEvtId CSL_CicObj::ectlEvtId

The vector-id, Range is 0,1..13 for CIC #0-2 and 0,1,..15 for CIC #3

CSL_CicEventId CSL_CicObj::eventId

system event into the CIC range is 0..63

25.4 Enumerations

25.4.1 CSL_CicEctlEvtId

enum CSL_CicEctlEvtId

Cic Output Event IDs

Enumeration values

<i>CSL_CIC_ECTL_EVT0</i>	Cic output event0
<i>CSL_CIC_ECTL_EVT1</i>	Cic output event1
<i>CSL_CIC_ECTL_EVT2</i>	Cic output event2
<i>CSL_CIC_ECTL_EVT3</i>	Cic output event3
<i>CSL_CIC_ECTL_EVT4</i>	Cic output event4
<i>CSL_CIC_ECTL_EVT5</i>	Cic output event5
<i>CSL_CIC_ECTL_EVT6</i>	Cic output event6
<i>CSL_CIC_ECTL_EVT7</i>	Cic output event7
<i>CSL_CIC_ECTL_EVT8</i>	Cic output event8
<i>CSL_CIC_ECTL_EVT9</i>	Cic output event9
<i>CSL_CIC_ECTL_EVT10</i>	Cic output event10
<i>CSL_CIC_ECTL_EVT11</i>	Cic output event11
<i>CSL_CIC_ECTL_EVT12</i>	Cic output event12
<i>CSL_CIC_ECTL_EVT13</i>	Cic output event13
<i>CSL_CIC_ECTL_EVT14</i>	Cic output event14 only for CIC3
<i>CSL_CIC_ECTL_EVT15</i>	Cic output event15 only for CIC3

25.4.2 CSL_CicHwControlCmd

enum CSL_CicHwControlCmd

Enumeration of the control commands

These are the control commands that could be used with `CSL_cicHwControl(..)`. Some of the commands expect an argument as documented along-side the description of the command.

Enumeration values

<i>CSL_CIC_CMD_EVTENABLE</i>	Enables the event. Parameters: <i>CSL_CicEnableState</i>
------------------------------	--

<i>CSL_CIC_CMD_EVTDISABLE</i>	Disables the event. Parameters: <i>CSL_CicEnableState</i>
<i>CSL_CIC_CMD_EVTSET</i>	Sets the event manually. Parameters: <i>None</i>
<i>CSL_CIC_CMD_EVTCLEAR</i>	Clears the event (if pending). Parameters: <i>None</i>

25.4.3 CSL_CicHwStatusQuery

enum CSL_CicHwStatusQuery

Enumeration of the queries

These are the queries that could be used with `CSL_cicGetHwStatus(..)`. The queries return a value through the object pointed to by the pointer that it takes as an argument. The argument supported by the query is documented along-side the description of the query.

Enumeration values

<i>CSL_CIC_QUERY_PENDSTATUS</i>	The Pend Status of the Event is queried. Parameters: <i>Bool</i>
---------------------------------	--

25.5 Macros

#define CSL_CIC_BADHANDLE (0)

Invalid handle

#define CSL_CIC_EVENTID_CNT 64

Number of Events in the System

#define CSL_CIC_MAPPED_NONE (-1)

None mapped

#define NUM_CIC_INST 4

Number of CIC instances

25.6 Typedefs

typedef Uint32 CSL_CicEventEnableState

Event enable state

typedef Int CSL_CicEventId

Interrupt Event IDs

typedef struct CSL_CicObj* CSL_CicHandle

The CIC handle

This is returned by the CSL_cicOpen(..) API. The handle is used to identify the event of interest in all CIC calls.

Chapter 26 FSYNC Module

Topics

<u>26.1 Overview</u>
<u>26.2 Functions</u>
<u>26.3 Data Structures</u>
<u>26.4 Enumerations</u>
<u>26.5 Macros</u>
<u>25.6 Typedefs</u>

26.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within FSYNC module.

The FSync module is responsible for creating system events that are used to trigger various events on chip and off chip. Examples of events generated by FSync module include:

1. A system event occurring every 128 chips to the DL DSP that starts the downlink processing.
2. A tick every 4 chip tick to the EDMA module which starts the DMA transfer of outbound data from DSP L2 memory to AIF data memory.

There are 3 sets synchronization/clock input pairs to the FSync module. The FSync module has 3 timers – RP3 timer, system timer and a Time of Day (TOD) timer. In addition to the 3 timers, there is also a watchdog timer which is used to detect failure when synchronization updates on RP1 interface have not occurred within a programmable time. Each of the timers can be programmed to be synchronized and clocked by any of the 3 available synchronization/clock input pairs. The timers can be initialized using the CSL if the FSync is operated in non-RP1 mode (FRAME_BURST and FSYNC_CLOCK are not used to synchronize and clock the timers).

There are 30 programmable trigger generators that can be used to generate system events which are routed to GEM cores and other modules on the TCI6488 chip. Trigger generators use the timers and the conditions programmed using the CSL to generate triggers for on chip peripherals (GEM cores, EDMA, AIF, timers) and off chip peripherals. Trigger generators 0 through 9 and 18 through 29 will be mask based. Trigger generators 10 through 17 will be counter based. Mask based generators are used when system events need to be generated every 2^n chips. Counter based event generators are used when events need to be generated every n chips (where n is not a power of 2). All event generators can be programmed with a programmable offset which delays the enabling of system events by up to 1 frame. The offsets can be programmed in sub-chips to control precisely when event generation begins. Offset conditions are re-evaluated after synchronization or resynchronization of timers occurs. All trigger event generator output are timed at VBUS clock rate, however there are 2 additional special outputs driven by triggers generators 0 and 1 which are clocked directly at the input clock rate. There are AIF_FrameSync driven by trigger generator 1 and SM_FRAME_CLK driven by trigger generator 0. The AIF_FrameSync is used by the Antenna Interface and the SM_FRAME_CLK can be used by off-chip peripherals. There are 2 error/alarm circuits which can be configured to raise system events on any of the GEM cores. The events that cause the error/alarm events must be enabled to cause system events.

Any of the three GEM cores can configure the FSync module using the configuration switch fabric and configuration interface.

26.2 Functions

26.2.1 CSL_fsycnClose

CSL_Status CSL_fsycnClose ([CSL_FsyncHandle](#) *hFsync*)

Description

This function closes the specified instance of FSYNC.

Arguments

hFsync Handle to the FSYNC instance

Return Value

CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Example

```
// handle for Fsync
CSL_FsyncHandle handleFsync;
// Fsync object
CSL_FsyncObj myFsyncObj;
// CSL status
CSL_Status status;

// Initialize CSL library, this step is not required
CSL_fsycnInit(NULL);

// Open handle to FSync
handleFsync = CSL_fsycnOpen(&myFsyncObj, CSL_FSYNC, NULL,
&status);
. . .
. . .
. . .
CSL_fsycnClose(handleFsync);
```

Parameters

hFsync Pointer to the object that holds reference to the instance of FSYNC link requested after the CSL_fsycnOpen(...) call

26.2.2 CSL_fsycnHwControl

CSL_Status CSL_fsycnHwControl ([CSL_FsyncHandle](#) *hFsync*,
CSL_FsyncHwControlCmd *ctrlCmd*,
void * *arg*
)

Description

Takes a command of FSYNC with an optional argument & implements it.

Arguments

hFsync Handle to the FSYNC instance

cmd	The command to this API indicates the action to be taken on FSYNC.
arg	An optional argument.

Return Value

CSL_Status

- CSL_SOK - Status info return successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

None

Post Condition

None

Modifies

The hardware registers of FSYNC.

Example

```
// handle for Fsync
CSL_FsyncHandle  handleFsync;
// Fsync object
CSL_FsyncObj     myFsyncObj;
// Fsync setup
CSL_FsyncSetup   myFsyncCfg;
// CSL status
CSL_Status       status;

CSL_FsyncTimerTermCountObj terminalCount;

// ctrl arg for Hw command
CSL_BitMask16 ctrlArg;

// Do config Fsync with RP1 interface
myFsyncCfg.syncRP3Timer = CSL_FSYNC_FRAME_BURST;

// Initialize CSL library, this step is not required
CSL_fsyncInit(NULL);

// Open handle to FSync
handleFsync = CSL_fsyncOpen(  &myFsyncObj,
                              CSL_FSYNC,
                              NULL,
                              &status);

// FSync configuration
myFsyncCfg.syncRP3Timer = CSL_FSYNC_FRAME_BURST;
myFsyncCfg.syncSystemTimer = CSL_FSYNC_FRAME_BURST;
myFsyncCfg.clkRP3Timer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.clkSystemTimer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.pTerminalCountRP3Timer = &terminalCount;
myFsyncCfg.pTerminalCountSystemTimer = &terminalCount;
```

```

myFsyncCfg.systemTimerRplSync =
    CSL_FSYNC_RP1_TYPE_WCDMA_FDD_FRAME_NUM;

myFsyncCfg.rp3EqualsSysTimer = FALSE;
myFsyncCfg.syncMode = CSL_FSYNC_RP1_SYNC_MODE;
myFsyncCfg.reSyncMode = CSL_FSYNC_AUTO_RESYNC_MODE;
myFsyncCfg.crcUsage = CSL_FSYNC_USE_SYNC_BURST_ON_CRC_FAIL;

// FSync h/w setup
CSL_fsyncHwSetup(handleFsync, &myFsyncCfg);
// Enable timer
ctrlArg = CSL_FSYNC_RP3_TIMER_ENABLE |
    CSL_FSYNC_SYSTEM_TIMER_ENABLE |
    CSL_FSYNC_TOD_TIMER_ENABLE;

CSL_fsyncHwControl(handleFsync,
    CSL_FSYNC_CMD_ENABLE_TIMER,
    (void *)&ctrlArg);

```

26.2.3 CSL_fsyncGetHwStatus

CSL_Status CSL_fsyncGetHwStatus ([CSL_FsyncHandle](#) *hFsync,*
 CSL_FsyncHwStatusQuery *query,*
 void * *response*
)

Description

This function is used to get the value of various parameters of the fsync instance. The value returned depends on the query passed.

Arguments

hFsync	Handle to the FSYNC instance
query	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value

CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_INVQUERY - Query command not supported

Pre Condition

None

Post Condition

Data requested by query is returned through the variable "response"

Modifies

The input argument "response" is modified.

Example

```

// handle for Fsync
CSL_FsyncHandle handleFsync;

```

```
// Fsync object
CSL_FsyncObj myFsyncObj;
// Fsync setup
CSL_FsyncSetup myFsyncCfg;
// CSL status
CSL_Status status;
CSL_FsyncTimerTermCountObj terminalCount;
// ctrl arg for Hw command
CSL_BitMask16 ctrlArg;
// query response
CSL_FsyncTimerCountObj queryRp3TimerValue;

// Do config Fsync with RP1 interface
myFsyncCfg.syncRP3Timer = CSL_FSYNC_FRAME_BURST;

// Initialize CSL library, this step is not required
CSL_fsyncInit(NULL);

// Open handle to FSync
handleFsync = CSL_fsyncOpen(&myFsyncObj, CSL_FSYNC, NULL,
&status);

// FSync configuration
myFsyncCfg.syncRP3Timer = CSL_FSYNC_FRAME_BURST;
myFsyncCfg.syncSystemTimer = CSL_FSYNC_FRAME_BURST;
myFsyncCfg.clkRP3Timer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.clkSystemTimer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.pTerminalCountRP3Timer = &terminalCount;
myFsyncCfg.pTerminalCountSystemTimer = &terminalCount;
myFsyncCfg.systemTimerRp1Sync =
CSL_FSYNC_RP1_TYPE_WCDMA_FDD_FRAME_NUM;
myFsyncCfg.rp3EqualsSysTimer = FALSE;
myFsyncCfg.syncMode = CSL_FSYNC_RP1_SYNC_MODE;
myFsyncCfg.reSyncMode = CSL_FSYNC_AUTO_RESYNC_MODE;
myFsyncCfg.crcUsage = CSL_FSYNC_USE_SYNC_BURST_ON_CRC_FAIL;

// FSync h/w setup
CSL_fsyncHwSetup(handleFsync, &myFsyncCfg);
// Enable timer
ctrlArg = CSL_FSYNC_RP3_TIMER_ENABLE |
CSL_FSYNC_SYSTEM_TIMER_ENABLE |
CSL_FSYNC_TOD_TIMER_ENABLE;

CSL_fsyncHwControl(handleFsync, CSL_FSYNC_CMD_ENABLE_TIMER, (void
*)&ctrlArg);
// Read value of RP3 timer
CSL_fsyncGetHwStatus(handleFsync,
CSL_FSYNC_QUERY_RP3_TIMER_VALUE, (void *)&queryRp3TimerValue);
```

26.2.4 CSL_fsyncHwSetup

CSL_Status CSL_fsyncHwSetup ([CSL_FsyncHandle](#)
CSL_FsyncSetup *
)

hFsync,
pFsyncSetup

Description

It configures the FSYNC instance registers as per the values passed in the hardware setup structure.

Arguments

hFsync	Handle to the FSYNC instance
pFsyncSetup	Pointer to hardware setup structure

Return Value

CSL_Status

- CSL_SOK - Hardware setup successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition

None

Post Condition

The specified instance will be setup according to value passed.

Modifies

The hardware registers of FSYNC.

Example

```
// handle for Fsync
CSL_FsyncHandle handleFsync;
// Fsync object
CSL_FsyncObj myFsyncObj;
// Fsync setup
CSL_FsyncSetup myFsyncCfg;

CSL_FsyncTimerTermCountObj terminalCount;
// CSL status
CSL_Status status;

// FSync configuration
myFsyncCfg.syncRP3Timer =
CSL_FSYNC_FRAME_BURST;myFsyncCfg.syncSystemTimer =
CSL_FSYNC_FRAME_BURST;
myFsyncCfg.clkRP3Timer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.clkSystemTimer = CSL_FSYNC_FRAME_SYNC_CLK;
myFsyncCfg.pTerminalCountRP3Timer = &terminalCount;
myFsyncCfg.pTerminalCountSystemTimer = &terminalCount;
myFsyncCfg.systemTimerRplSync =
CSL_FSYNC_RP1_TYPE_WCDMA_FDD_FRAME_NUM;
myFsyncCfg.rp3EqualsSysTimer = FALSE;
myFsyncCfg.syncMode = CSL_FSYNC_RP1_SYNC_MODE;
myFsyncCfg.reSyncMode = CSL_FSYNC_AUTO_RESYNC_MODE;
myFsyncCfg.crcUsage = CSL_FSYNC_USE_SYNC_BURST_ON_CRC_FAIL;

// Initialize CSL library, this step is not required
CSL_fsycInit(NULL);
// Open handle to FSync
```

```
handleFsync = CSL_fsyncOpen(&myFsyncObj, CSL_FSYNC, NULL,
&status);
```

```
// Do setup for FSync
CSL_fsyncHwSetup(handleFsync, &myFsyncCfg);
```

26.2.5 CSL_fsyncGetBaseAddress

```
CSL_Status CSL_fsyncGetBaseAddress (   CSL_InstNum           fsyncNum,
                                       CSL_FsyncParam *       pFsyncParam,
                                       CSL_FsyncBaseAddress *   pBaseAddress
                                       )
```

Description

Function to get the Base-address of the peripheral instance.

This function is used for getting the base-address of the peripheral instance.

Return Value CSL_Status

- CSL_SOK Function call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid Parameter

Parameters

fsyncNum Peripheral instance number

pFsyncParam Module specific parameters.

pBaseAddress Base address details.

Example

```
CSL_Status status;
CSL_FsyncBaseAddress baseAddress;
...
status = CSL_fsyncGetBaseAddress(   CSL_FSYNC_0,
                                   NULL,
                                   &baseAddress);
```

26.3 Data Structures

26.3.1 CSL_FsyncBaseAddress

Detailed Description

This will have the base-address information for the peripheral instance.

Field Documentation

CSL_FsyncRegsOvly CSL_FsyncBaseAddress::regs

This is a pointer to the registers of the FSYNCR

26.3.2 CSL_FsyncCounterTriggerGenObj

Detailed Description

This object is used to specify a counter based trigger event.

Field Documentation

UInt32 CSL_FsyncCounterTriggerGenObj::eventCount

counter used to count down to event generation, used by counter based trigger generators,

CSL_FsyncTriggerGenNum CSL_FsyncCounterTriggerGenObj::eventGenUsed

specify the trigger event generator used

CSL_FsyncTriggerOffsetObj CSL_FsyncCounterTriggerGenObj::offset

specifies trigger offset in slots,chip terminal count index,chips,sample

CSL_FsyncTimerType CSL_FsyncCounterTriggerGenObj::timerUsed

specify the type of timer to be used for event trigger generation

26.3.3 CSL_FsyncErrEventMaskObj

Detailed Description

This object is used to specify the masks for enable,disable,clear,set operations for the 2 error/alarm circuits.

Field Documentation

CSL_BitMask32 CSL_FsyncErrEventMaskObj::errEventMask0

Error event mask 0

CSL_BitMask32 CSL_FsyncErrEventMaskObj::errEventMask1

Error event mask 1

26.3.4 CSL_FsyncMaskTriggerGenObj

Detailed Description

This object is used to specify a mask based trigger event.

Field Documentation

CSL_FsyncTriggerCompareObj CSL_FsyncMaskTriggerGenObj::compareValue

specifies the trigger compare value used for frame,slot,chip, terminal chip count index and sample fields

CSL_FsyncTriggerGenNum CSL_FsyncMaskTriggerGenObj::eventGenUsed
specify the trigger event generator used

CSL_FsyncTriggerMaskObj CSL_FsyncMaskTriggerGenObj::mask
specifies the trigger mask used for frame,slot,chip terminal count index, chip,sample fields and is used for mask based trigger event generation

CSL_FsyncTriggerOffsetObj CSL_FsyncMaskTriggerGenObj::offset
specifies trigger offset in slots,chip terminal count index,chips,sample

CSL_FsyncTimerType CSL_FsyncMaskTriggerGenObj::timerUsed
specify the type of timer to be used for event trigger generation

26.3.5 CSL_FsyncObj

Detailed Description

This object contains the reference to the instance of FSYNC opened using the **CSL_fsyncOpen()**. The pointer to this object, is passed as FSYNC handles to all FSYNC CSL APIs. **CSL_fsyncOpen()** function initializes this structure based on the parameters passed.

Field Documentation

CSL_InstNum CSL_FsyncObj::perNum
This is the instance of FSYNC being referred to by this object

CSL_FsyncRegsOvly CSL_FsyncObj::regs
This is a pointer to the registers of the FSYNC

26.3.6 CSL_FsyncParam

Detailed Description

Object hold module specific parameters. Present implementation doesn't have any module specific parameters.

Field Documentation

CSL_BitMask16 CSL_FsyncParam::flags
Bit mask to be used for module specific parameters. The below declaration is just a place-holder for future implementation.

26.3.7 CSL_FsyncRp1PayloadObj

Detailed Description

This object is used to query the RP1 payload captured during a sync burst.

Field Documentation

UInt32 CSL_FsyncRp1PayloadObj::lsb
lsb of RP1 payload captured

UInt32 CSL_FsyncRp1PayloadObj::msb
msb of RP1 payload captured

26.3.8 CSL_FsyncSetup

Detailed Description

This is the Setup structure for configuring FSYNC using **CSL_fsyncHwSetup()** function.

Field Documentation

CSL_FsyncTimerClkSource CSL_FsyncSetup::clkRP3Timer

Clock source for RP3 timer

CSL_FsyncTimerClkSource CSL_FsyncSetup::clkSystemTimer

Clock source for System timer

CSL_FsyncRp1CrcPosition CSL_FsyncSetup::crcPosition

specify CRC position in RP1 mode

CSL_FsyncRp1CRCUsage CSL_FsyncSetup::crcUsage

Specify CRC usage in RP1 mode

CSL_FsyncTimerTermCountObj* CSL_FsyncSetup::pTerminalCountRP3Timer

Terminal count for frame,slot,chip,subchip for RP3 timer

CSL_FsyncTimerTermCountObj* CSL_FsyncSetup::pTerminalCountSystemTimer

Terminal count for frame,slot,chip,subchip for System timer

CSL_FsyncTimerReSyncMode CSL_FsyncSetup::reSyncMode

Specify if auto resynchronization occurs if new sync is out of alignment

Bool CSL_FsyncSetup::rp3EqualsSysTimer

Specify if RP3 timer equals system timer, TRUE - RP3 equal system timer, FALSE - RP3 and system timer are different

UInt8 CSL_FsyncSetup::rp3SyncDelay

delay for RP3 sync in input clock cycles, delay can be 0-16 clk cycles

CSL_FsyncWatchDogObj CSL_FsyncSetup::setupWatchDog

specify the frame update rates for RP3/WCDMA/TOD sync burts in watch dog timer

CSL_FsyncTimerSyncMode CSL_FsyncSetup::syncMode

Specify the sync mode - RP1(OBSA)/Non RP1

CSL_FsyncTimerSyncSource CSL_FsyncSetup::syncRP3Timer

Sync source for RP3 timer - SyncBurst/UmtsSync/TRTSync/TestSync

CSL_FsyncTimerSyncSource CSL_FsyncSetup::syncSystemTimer

Sync source for System timer

UInt8 CSL_FsyncSetup::systemSyncDelay

delay for System sync in input clock cycles, delay can be 0-16 clk cycles

CSL_FsyncRp1TypeField CSL_FsyncSetup::systemTimerRp1Sync

specifies the type field in RP1 sync burst to which system timer will sync

CSL_FsyncTimerInitObj CSL_FsyncSetup::timerInit

specify the init values for RP3,system timers if non-RP1 based interface is used

CSL_FsyncTodLeapSecsUsage CSL_FsyncSetup::todLeapUsage

specify if leap secs are to be added to TOD

UInt8 CSL_FsyncSetup::todSyncDelay

delay for Time Of Day sync in input clock cycles, delay can be 0-16 clk cycles

26.3.9 CSL_FsyncTimerCountObj

Detailed Description

This object is used to query RP3/System/TRT timer count (see @ CSL_fsyncGetHwStatus) and timer init objects (see **CSL_FsyncTimerInitObj**).

Field Documentation
UInt16 CSL_FsyncTimerCountObj::chipNum

Specifies the chip no. of RP3 timer

UInt16 CSL_FsyncTimerCountObj::frameNum

Specifies the frame no. of RP3 timer

UInt8 CSL_FsyncTimerCountObj::slotNum

Specifies the slot no. of RP3 timer

26.3.10 CSL_FsyncTimerInitObj

Detailed Description

This object is used to initialise RP3 and system timers when a non-RP1 interface is used.

Field Documentation
CSL_FsyncTimerCountObj* CSL_FsyncTimerInitObj::pRp3TimerInit

This is pointer which initializes Rp3 Timer

CSL_FsyncTimerCountObj* CSL_FsyncTimerInitObj::pSystemTimerInit

This is pointer which initializes System Timer

26.3.11 CSL_FsyncTimerTermCountObj

Detailed Description

This object is used to define RP3/System terminal counts in @ **CSL_FsyncSetup**.

Field Documentation
UInt8 CSL_FsyncTimerTermCountObj::lastSampleNum

last subchip number

UInt8 CSL_FsyncTimerTermCountObj::lastSlotNum

last slot no.

UInt8 CSL_FsyncTimerTermCountObj::numChipTerminalCount

Specifies number of terminal counts for chips count - value will be 1 for WCDMA(FDD) and up to 10 for TDSCDMA

UInt16* CSL_FsyncTimerTermCountObj::pLastChipNum

pointer to array containing terminal counts for chips

26.3.12 CSL_FsyncTriggerCompareObj

Detailed Description

This object is used to specify a compare value for mask based trigger generators (see @ [CSL_FsyncMaskTriggerGenObj](#)).

Field Documentation

UInt8 CSL_FsyncTriggerCompareObj::chipTerminalCountIndexValue

specifies chip terminal counter value used to compare masked chip term. counter in RP3/system timer

UInt16 CSL_FsyncTriggerCompareObj::chipValue

specifies chip value used to compare masked chip counter in RP3/system timer

UInt8 CSL_FsyncTriggerCompareObj::frameValue

specifies frame value used to compare masked frame counter in RP3/system timer, only the 2 lsb's can be specified for frame compare value

UInt8 CSL_FsyncTriggerCompareObj::sampleValue

specifies sample value used to compare masked sample counter in RP3/system timer

UInt8 CSL_FsyncTriggerCompareObj::slotValue

specifies slot value used to compare masked slot counter in RP3/system timer

26.3.13 CSL_FsyncTriggerMaskObj

Detailed Description

This object is used to define masks for mask based trigger generators (see @ [CSL_FsyncMaskTriggerGenObj](#)).

Field Documentation

UInt16 CSL_FsyncTriggerMaskObj::chipMask

specifies chip mask used to mask chip number of RP3/system timer

UInt8 CSL_FsyncTriggerMaskObj::chipTerminalCountIndexMask

specifies chip terminal count index mask used to mask chip terminal index number of Rp3/system timer

UInt8 CSL_FsyncTriggerMaskObj::frameMask

specifies frame mask used to mask frame number of RP3/system timer, only the 2 lsb's can be masked

UInt8 CSL_FsyncTriggerMaskObj::sampleMask

specifies sample mask used to mask sample number of RP3/system timer

UInt8 CSL_FsyncTriggerMaskObj::slotMask

specifies slot mask used to mask slot number of RP3/system timer

26.3.14 CSL_FsyncTriggerOffsetObj

Detailed Description

This object is used to define trigger offset relative to RP3/System sync and is used in specifying mask based (see @ **CSL_FsyncMaskTriggerGenObj**) and counter based trigger generators (see @ **CSL_FsyncCounterTriggerGenObj**).

Field Documentation

UInt16 CSL_FsyncTriggerOffsetObj::chipOffset

specifies chip offset

UInt8 CSL_FsyncTriggerOffsetObj::chipTerminalCountIndex

Index into the array of terminal chip count array, value can be 0,1..(numChipTerminalCount-1)

UInt8 CSL_FsyncTriggerOffsetObj::sampleOffset

specifies chip offset

UInt8 CSL_FsyncTriggerOffsetObj::slotOffset

specifies slot offset

26.3.15 CSL_FsyncWatchDogObj

Detailed Description

This object is used to specify the update rates for different timers.

Field Documentation

UInt16 CSL_FsyncWatchDogObj::rp3FrameUpdateRate

Frame update rate for Rp3 timer

UInt16 CSL_FsyncWatchDogObj::todFrameUpdateRate

Frame update rate for TOD

UInt16 CSL_FsyncWatchDogObj::wcdmaFrameUpdateRate

26.4 Enumerations

26.4.1 CSL_FsyncTimerType

enum CSL_FsyncTimerType

Timer types in Frame Sync.

Use this symbol to specify timer type for FSYNC

Enumeration values

<i>CSL_FSYNC_SYSTEM_TIMER</i>	Specifies system timer
<i>CSL_FSYNC_RP3_TIMER</i>	Specifies Rp3 timer
<i>CSL_FSYNC_TOD_TIMER</i>	Specifies TOD timer

26.4.2 CSL_FsyncTimerSyncSource

enum CSL_FsyncTimerSyncSource

Timer sync sources in Frame Sync.

Use this symbol to specify timer sync source for FSYNC

Enumeration values

<i>CSL_FSYNC_FRAME_BURST</i>	Specifies frame burst sync source
<i>CSL_FSYNC_UMTS_SYNC</i>	Specifies UMTS sync source
<i>CSL_FSYNC_TRT</i>	Specifies TRT sync source
<i>CSL_FSYNC_SYSTEM_TEST_SYNC</i>	Specifies system test sync source

26.4.3 CSL_FsyncTimerClkSource

enum CSL_FsyncTimerClkSource

Clock source for frame sync timers.

Use this symbol to specify clock source for FSYNC timers

Enumeration values

<i>CSL_FSYNC_VBUS_CLK_DIV_3</i>	Specifies VBUS clock source
<i>CSL_FSYNC_TRT_CLK</i>	Specifies TRT clock source
<i>CSL_FSYNC_FRAME_SYNC_CLK</i>	Specifies Frame Sync clock source
<i>CSL_FSYNC_UMTS_CLK</i>	Specifies UMTS clock source

26.4.4 CSL_FsyncTimerSyncMode

enum CSL_FsyncTimerSyncMode

Sync mode for frame sync timers.

Use this symbol to specify sync mode for FSYNC timers

Enumeration values

CSL_FSYNC_NON_RP1_SYNC_MODE Specifies non RP1 sync mode

CSL_FSYNC_RP1_SYNC_MODE Specifies Rp1 sync mode

26.4.5 CSL_FsyncTimerReSyncMode

enum CSL_FsyncTimerReSyncMode

Re-Sync mode for FSync timers.

Use this symbol to specify re-sync mode for FSYNC timers

Enumeration values

CSL_FSYNC_NO_AUTO_RESYNC_MODE Specifies no_auto re-sync mode

CSL_FSYNC_AUTO_RESYNC_MODE Specifies auto re-sync mode

26.4.6 CSL_FsyncRp1CRCUsage

enum CSL_FsyncRp1CRCUsage

CRC usage in RP1 sync mode for FSYNC.

Use this symbol to specify CRC usage mode when FSYNC is used in RP1 sync mode

Enumeration values

CSL_FSYNC_USE_SYNC_BURST_ON_CRC_FAIL Specifies usage of sync burst on crc fail

CSL_FSYNC_DISCARD_SYNC_BURST_ON_CRC_FAIL Specifies discarding of sync burst on crc fail

26.4.7 CSL_FsyncTodLeapSecsUsage

enum CSL_FsyncTodLeapSecsUsage

Specifies if leap secs are to be added to TOD.

Use this symbol to specify if leap secs are to be added to TOD

Enumeration values

CSL_FSYNC_DONT_ADD_LEAPSECS Specifies no addition of leap secs to TOD

CSL_FSYNC_ADD_LEAPSECS Specifies addition of leap secs to TOD

26.4.8 CSL_FsyncRp1CrcPosition

enum CSL_FsyncRp1CrcPosition

Specifies CRC position in RP1 sync burst.

Use this symbol to specify CRC position in RP1 sync burst

Enumeration values

CSL_FSYNC_CRC_BIT_16_RCVD_FIRST Specifies bit 16 as CRC position in RP1 sync burst

CSL_FSYNC_CRC_BIT_0_RCVD_FIRST Specifies bit 0 as CRC position in RP1 sync burst

26.4.9 CSL_FsyncErrMaskType

enum CSL_FsyncErrMaskType

Error mask type for FSYNC.

Use this symbol to specify the error mask for FSYNC

Enumeration values

CSL_FSYNC_ERROR_MASK_0 Specifies error mask 0

CSL_FSYNC_ERROR_MASK_1 Specifies error mask 1

26.4.10 CSL_FsyncErrAlarmIndex

enum CSL_FsyncErrAlarmIndex

Error Alarm Index for FSYNC.

Use this symbol to specify the Error Alarm Index for FSYNC

Enumeration values

CSL_FSYNC_ERR_ALARM_0 Specifies error alarm index 0

CSL_FSYNC_ERR_ALARM_1 Specifies error alarm index 1

26.4.11 CSL_FsyncTriggerGenNum

enum CSL_FsyncTriggerGenNum

Use this symbol to specify the trigger generator used for event generation Trigger geerator can be mask or event based, depending on the trigger generator used, trigger parameters should be specified See *CSL_FsyncMaskTriggerGenObj* for details on mask based triggers and *CSL_FsyncCounterTriggerEventObj* for details on counter based triggers

Enumeration values

CSL_FSYNC_TRIGGER_GEN_0 specifies mask based trigger generator 0

CSL_FSYNC_TRIGGER_GEN_1 specifies mask based trigger generator 1

CSL_FSYNC_TRIGGER_GEN_2 specifies mask based trigger generator 2

CSL_FSYNC_TRIGGER_GEN_3 specifies mask based trigger generator 3

CSL_FSYNC_TRIGGER_GEN_4 specifies mask based trigger generator 4

CSL_FSYNC_TRIGGER_GEN_5 specifies mask based trigger generator 5

CSL_FSYNC_TRIGGER_GEN_6 specifies mask based trigger generator 6

CSL_FSYNC_TRIGGER_GEN_7 specifies mask based trigger generator 7

CSL_FSYNC_TRIGGER_GEN_8 specifies mask based trigger generator 8

CSL_FSYNC_TRIGGER_GEN_9 specifies mask based trigger generator 9

CSL_FSYNC_TRIGGER_GEN_10 specifies counter based trigger generator 10

<i>CSL_FSYNC_TRIGGER_GEN_11</i>	specifies counter based trigger generator 11
<i>CSL_FSYNC_TRIGGER_GEN_12</i>	specifies counter based trigger generator 12
<i>CSL_FSYNC_TRIGGER_GEN_13</i>	specifies counter based trigger generator 13
<i>CSL_FSYNC_TRIGGER_GEN_14</i>	specifies counter based trigger generator 14
<i>CSL_FSYNC_TRIGGER_GEN_15</i>	specifies counter based trigger generator 15
<i>CSL_FSYNC_TRIGGER_GEN_16</i>	specifies counter based trigger generator 16
<i>CSL_FSYNC_TRIGGER_GEN_17</i>	specifies counter based trigger generator 17
<i>CSL_FSYNC_TRIGGER_GEN_18</i>	specifies mask based trigger generator 18
<i>CSL_FSYNC_TRIGGER_GEN_19</i>	specifies mask based trigger generator 19
<i>CSL_FSYNC_TRIGGER_GEN_20</i>	specifies mask based trigger generator 20
<i>CSL_FSYNC_TRIGGER_GEN_21</i>	specifies mask based trigger generator 21
<i>CSL_FSYNC_TRIGGER_GEN_22</i>	specifies mask based trigger generator 22
<i>CSL_FSYNC_TRIGGER_GEN_23</i>	specifies mask based trigger generator 23
<i>CSL_FSYNC_TRIGGER_GEN_24</i>	specifies mask based trigger generator 24
<i>CSL_FSYNC_TRIGGER_GEN_25</i>	specifies mask based trigger generator 25
<i>CSL_FSYNC_TRIGGER_GEN_26</i>	specifies mask based trigger generator 26
<i>CSL_FSYNC_TRIGGER_GEN_27</i>	specifies mask based trigger generator 27
<i>CSL_FSYNC_TRIGGER_GEN_28</i>	specifies mask based trigger generator 28
<i>CSL_FSYNC_TRIGGER_GEN_29</i>	specifies mask based trigger generator 29

26.4.12 CSL_FsyncRp1CrclnitValue

enum CSL_FsyncRp1CrclnitValue

CRC init one for FSYNC.

Use this symbol to specify the CRC init one for FSYNC

Enumeration values

<i>CSL_FSYNC_CRC_INIT_VALUE_ZEROS</i>	specifies CRC init value as 0
<i>CSL_FSYNC_CRC_INIT_VALUE_ONES</i>	specifies CRC init value as 1

26.4.13 CSL_FsyncHwControlCmd

enum CSL_FsyncHwControlCmd

This is the set of control commands that are passed to *CSL_fsyncHwControl()*, with an optional argument type-casted to *void**. The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values

<i>CSL_FSYNC_CMD_ENABLE_TIMER</i>	Enable Fsync timers so it can start when sync Happens Parameters: (<i>CSL_BitMask16 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_DISABLE_TIMER</i>	Disable Fsync timers Parameters: (<i>CSL_BitMask16 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_HALT_TIMER</i>	Halt Fsync timers Parameters: (<i>CSL_BitMask16 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_ENABLE_RP1_SYNC_MODE</i>	Enable RP1 sync mode Parameters: (<i>NULL</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_DISABLE_RP1_SYNC_MODE</i>	Disable RP1 sync mode Parameters: (<i>NULL</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_TRIGGER_SYSTEM_TEST_SYNC</i>	Trigger system test sync Parameters: (<i>NULL</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CHNG_SYNC_BURST_FRAME_UPDATE_RATE_TOD_INFO</i>	Change sync burst frame update rate for Time of Day info used in watchdog timer Parameters: (<i>UInt32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CHNG_SYNC_BURST_FRAME_UPDATE_RATE_RP3_INFO</i>	Change sync burst frame update rate for RP3 info used in watchdog timer Parameters: (<i>UInt32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CHNG_SYNC_BURST_FRAME_UPDATE_RATE_WCDMA_INFO</i>	Change sync burst frame update rate for WCDMA info used in watchdog timer Parameters: (<i>UInt32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CHNG_INIT_RP3_TIMER_VALUE</i>	Change initial value of RP3 timer Parameters: (<i>CSL_FsyncTimerCountObj *</i>) Return values: CSL_SOK

<i>CSL_FSYNC_CMD_CHNG_INIT_SYSTEM_TIMER_VALUE</i>	Change initial value of System timer Parameters: (<i>CSL_FsyncTimerCountObj *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_SET_EMU_CONTROL</i>	Specify emulation control bits Parameters: (<i>CSL_BitMask16 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_SET_EMU_MASK</i>	Specify emulation mask Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_ENABLE_ERR_EVENT</i>	Enable specified event(s) in the FSYNC error/alarm event enable reg Parameters: (<i>CSL_FsyncErrEventMaskObj *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_DISABLE_ERR_EVENT</i>	Disable specified event(s) in the FSYNC error/alarm event enable reg Parameters: (<i>CSL_FsyncErrEventMaskObj *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CLEAR_ERR_EVENT</i>	Clear specified event(s) in the FSYNC error/alarm event reg Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_SET_ERR_EVENT</i>	Set specified event(s) in the FSYNC error/alarm event set reg Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_REEVALUATE_ERR_INTERRUPT_LINE</i>	Re-evaluates the specified interrupt line for interrupts Parameters: (<i>CSL_FsyncErrAlarmIndex *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CONFIG_MASK_BASED_TRIGGER_GEN</i>	Configure mask based trigger generator Parameters: (<i>CSL_FsyncMaskTriggerGenObj *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_CONFIG_COUNTER_BASED_TRIGGER_GEN</i>	Configure counter based trigger generator Parameters: <i>CSL_FsyncCounterTriggerGenObj *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_ENABLE_TRIGGER_GEN</i>	Enable specified trigger generator Parameters: (<i>CSL_FsyncTriggerGenNum *</i>) Return values: CSL_SOK
<i>CSL_FSYNC_CMD_DISABLE_TRIGGER_GEN</i>	Disable specified trigger generator Parameters: (<i>CSL_FsyncTriggerGenNum *</i>) Return values: CSL_SOK

CSL_FSYNC_CMD_SET_TRIGGER_EVENT	Force trigger generator to generate system event Parameters: (CSL_FsyncTriggerGenNum *) Return values: CSL_SOK
CSL_FSYNC_CMD_SET_OK_STATUS_BIT	Set the OK status bit value in FSync Parameters: (NULL) Return values: CSL_SOK
CSL_FSYNC_CMD_CLEAR_OK_STATUS_BIT	Clear the OK status bit value in FSync Parameters: (NULL) Return values: CSL_SOK
CSL_FSYNC_CMD_CHNG_CRC_INIT_VALUE	Set CRC init Parameters: (CSL_FsyncRp1CrcInitValue*) Return values: CSL_SOK
CSL_FSYNC_CMD_ENABLE_CRC_INVERT	Enable CRC Invert Parameters: (NULL) Return values: CSL_SOK
CSL_FSYNC_CMD_CLEAR_TYPE_SELECT_CRC	clear Type select Parameters: (NULL) Return values: CSL_SOK
CSL_FSYNC_CMD_CHNG_SYS_TIMER_RP1_SYNC	Set System timer RP1 type Parameters: (CSL_FsyncRp1TypeField*) Return values: CSL_SOK
CSL_FSYNC_CMD_CHNG_CRC_POSITION	Set CRC position Parameters: (CSL_FsyncRp1CrcPosition*) Return values: CSL_SOK

26.4.14 CSL_FsyncHwStatusQuery

enum CSL_FsyncHwStatusQuery

This is the set of query commands to get the status of various operations in FSYNC
The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values

CSL_FSYNC_QUERY_PID	Queries Peripheral Identification Parameters: (CSL_BitMask32 *) Return values: CSL_SOK
CSL_FSYNC_QUERY_RP3_TIMER_VALUE	Queries the value of RP3 timer Parameters: (CSL_FsyncTimerCountObj *) Return values: CSL_SOK
CSL_FSYNC_QUERY_SYSTEM_TIMER_VALUE	Queries the value of system timer Parameters: (CSL_FsyncTimerCountObj *) Return values: CSL_SOK
CSL_FSYNC_QUERY_TOD_TIMER_VALUE	Queries the value of time of day timer in seconds

	Parameters: (<i>Uint32 *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_ERR_EVENT_0	Queries the value contained in the error event register 0 Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_ERR_EVENT_1	Queries the value contained in the error event register 1 Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_RP1_TYPE_CAPTURE	Queries the type field captured when a RP1 Parameters: (<i>CSL_FsyncRp1TypeField *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_RP1_PAYLOAD_CAPTURE_TOD_TYPE	Queries the payload captured when a RP1 sync burst of type Time of Day occurs Parameters: (<i>CSL_FsyncRp1PayloadObj *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_RP1_PAYLOAD_CAPTURE_RP3_TYPE	Queries the payload captured when a RP1 sync burst of type RP3 occurs Parameters: (<i>CSL_FsyncRp1PayloadObj *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_RP1_PAYLOAD_CAPTURE_WCDMA_FDD_TYPE	Queries the payload captured when a RP1 sync burst of type WCDMA(FDD) occurs Parameters: (<i>CSL_FsyncRp1PayloadObj *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_OK_STATUS_BIT	Queries the value of FSync Run bit value Parameters: (<i>NULL</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_ERR_INT_MASK0_REG	Queries the value contained in the error interrupt mask 0 register Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_ERR_INT_MASK1_REG	Queries the value contained in the error interrupt mask 1 register Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK
CSL_FSYNC_QUERY_ERR_INT_SRC_RAW_REG	Queries the value contained in the error interrupt source raw register Parameters: (<i>CSL_BitMask32 *</i>) Return values: CSL_SOK

Frame update rate for WCDMA

26.5 Macros

#define CSL_FSYNC_RP3_TIMER_ENABLE (1)

specifies position of RP3 timer in bit mask to enable/disable timers

#define CSL_FSYNC_SYSTEM_TIMER_ENABLE (2)

specifies position of system timer in bit mask to enable/disable timers

#define CSL_FSYNC_TOD_TIMER_ENABLE (4)

specifies position of TOD(time of day) timer in bit mask to enable/disable timers

#define CSL_FSYNC_NOS_MASK_EVENT_GEN (22)

#define CSL_FSYNC_NOS_COUNTER_EVENT_GEN (8)

#define CSL_FSYNC_FIRST_COUNTER_EVENT_GEN (10)

#define CSL_FSYNC_LAST_COUNTER_EVENT_GEN (17)

**#define CSL_FSYNC_EVENTGEN_TO_MASK_EVENTGEN_NUM(eventgenNum) **
 **((eventgenNum < CSL_FSYNC_FIRST_COUNTER_EVENT_GEN) ? (eventgenNum) : **
 (eventgenNum-CSL_FSYNC_NOS_COUNTER_EVENT_GEN)) // event gen 0,..9 18..29

**#define CSL_FSYNC_EVENTGEN_TO_COUNTER_EVENTGEN_NUM(eventgenNum) **
 (eventgenNum - CSL_FSYNC_FIRST_COUNTER_EVENT_GEN) // eventgenNum 10,..17

Use above symbols for defining FSync constants

#define CSL_FSYNC_SYSTEM_FRAME_FAIL (0x00000001)

#define CSL_FSYNC_RP3_FRAME_FAIL (0x00000002)

#define CSL_FSYNC_TOD_FRAME_FAIL (0x00000004)

#define CSL_FSYNC_FS_WDOG_FAIL (0x00000008)

#define CSL_FSYNC_RP3_WDOG_FAIL (0x00000010)

#define CSL_FSYNC_TOD_WDOG_FAIL (0x00000020)

#define CSL_FSYNC_CRC_FAIL (0x00000040)

#define CSL_FSYNC_BIT_WIDTH_FAIL (0x00000080)

#define CSL_FSYNC_TYPE_SPARE (0x00000100)

#define CSL_FSYNC_TYPE_RESERVE (0x00000200)

#define CSL_FSYNC_TYPE_UNSUPPORTED (0x00000400)

#define CSL_FSYNC_TYPE_TOD (0x00000800)

#define CSL_FSYNC_TYPE_RP3 (0x00001000)

#define CSL_FSYNC_TYPE_SYSTEM (0x00002000)

#define CSL_FSYNC_TYPE_OVERFLOW (0x00004000)

Use this symbols for defining FSync error alarms

26.6 Typedefs

typedef CSL_FsyncObj * CSL_FsyncHandle

typedef void * CSL_FsyncContext

FSync context is void pointer.

Chapter 27 RAC Module

Topics

<u>27.1 Overview</u>
<u>27.2 Functions</u>
<u>27.3 Data Structures</u>
<u>27.4 Enumerations</u>

27.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within RAC module.

The TCI6488 device is single chip DSP platform for 3GPP WCDMA baseband processing. One peripheral is called Receive Accelerator (RAC) and is used to perform chip-rate correlations operations

The RAC subsystem, which provides acceleration for chip rate receive functions, consists of several components:

- 2 GCCP correlation accelerators for Finger Despread (FD), Path Monitor (PM), Preamble Detection (PD), and Stream Power Estimator (SPE).
- Back-end Interface (BEI) for management of the RAC configuration and the data output.
- Front-end Interface (FEI) for reception of the antenna data for processing and access to all MMRs and memories in the RAC components.

27.2 Functions

27.2.1 void CSL_RAC_BEII_disable

void CSL_RAC_BEII_disable ()

Description

This function disables the RAC Back End Interrupt Interface (BEII) by writing 0x0 into the RAC_BEII_CTRL register.

CSL_RAC_BEII_disable

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BEII_CTRL register.

Pre condition

None

Post condition

The Back End Interrupt Interface is then disabled.

27.2.2 CSL_RAC_BEII_enable

void CSL_RAC_BEII_enable ()

Description

This function enables the RAC Back End Interrupt Interface (BEII) by writing 0x1 into the RAC_BEII_CTRL register.

CSL_RAC_BEII_enable

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BEII_CTRL register.

Pre condition

None

Post condition

The Back End Interrupt Interface is then enabled.

27.2.3 CSL_RAC_BEII_getInterruptStatus

void CSL_RAC_BEII_getInterruptStatus (Uint8 *cpuld*, CSL_RAC_BEII_interruptStatus **cpulntStatus*)

Description

This function reads the BEII Interrupt Status for the specified cpu interrupt controller.
CSL_RAC_BEII_getInterruptStatus

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

CSL_RAC_BEII_interruptStatus

Modifies

This operation modifies the RAC_BEII_INT_STATx register.

Pre condition

None

Post Condition

None

27.2.4 CSL_RAC_BEII_setBetieotMask

void CSL_RAC_BEII_setBetieotMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End End-Of-Transfer Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.5 CSL_RAC_BEII_setBetieobbtRdCrossingMask

void CSL_RAC_BEII_setBetieobbtRdCrossingMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End OBBT Read Pointer Crossing Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.6 CSL_RAC_BEII_setBetIOdbtRdCrossingMask

void CSL_RAC_BEII_setBetIOdbtRdCrossingMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End ODBT Read Pointer Crossing Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre Condition

None

Post Condition

None

27.2.7 CSL_RAC_BEII_setBeWatchDogMask

void CSL_RAC_BEII_setBeWatchDogMask (Uint8 *cpuld*, Uint8 *gccpld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End Min Activity Watch Dog Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

`gccpld` Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.8 CSL_RAC_BEII_setCycleOverflowMask

void CSL_RAC_BEII_setCycleOverflowMask (Uint8 *cpuld*, Uint8 *gccpld*, This function configures the Cycle Overflow Mask for the specified cpu interrupt controller.

Arguments

<i>cpuld</i>	Which CPU interrupt controller to configure.
<i>gccpId</i>	Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.9 CSL_RAC_BEII_setFeWatchDogMask

void CSL_RAC_BEII_setFeWatchDogMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Front End Watch Dog Mask for the specified cpu interrupt controller.

Arguments

<i>cpuld</i>	Which CPU interrupt controller to configure.
--------------	--

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.10 CSL_RAC_BEII_setFifoOverflowMask

**void CSL_RAC_BEII_setFifoOverflowMask (Uint8 *cpuld*, Uint8 *gccpld*,
CSL_RAC_BEII_interrupt *interruptEnable*)**

Description

This function configures the FIFO Overflow Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

gccpId Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[*cpuld*] register.

Pre condition

None

Post condition

None

27.2.11 CSL_RAC_BEII_setMasterMask

**void CSL_RAC_BEII_setMasterMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt
cpulntContrlEnable)**

Description

This function configures the BEII Master Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the RAC_BEII_MINT_ENA register.

Pre condition

None

Post condition

None

27.2.12 CSL_RAC_BEII_setSequencerIdleMask

**void CSL_RAC_BEII_setSequencerIdleMask (Uint8 *cpuld*, Uint8 *gccpld*,
CSL_RAC_BEII_interrupt *interruptEnable*)**

Description

This function configures the Sequencer Idle Mask for the specified cpu interrupt controller.

Arguments

<code>cupid</code>	Which CPU interrupt controller to configure.
<code>gccpId</code>	Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.13 CSL_RAC_BETI_disable

void CSL_RAC_BETI_disable ()

Description

This function disables the RAC Back End Transfer Interface (BETI) by writing 0x0 into the RAC_BETI_CTRL register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BETI_CTRL register.

Pre condition

None

Post condition

The Back End Transfer Interface is then disabled.

27.2.14 CSL_RAC_BETI_enable

void CSL_RAC_BETI_enable ()

Description

This function enables the RAC Back End Transfer Interface (BETI) by writing 0x1 into the RAC_BETI_CTRL register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BETI_CTRL register.

Pre condition

None

Post condition

The Back End Transfer Interface is then enabled.

27.2.15 CSL_RAC_BETI_getEotInterruptStatus

Uint32 CSL_RAC_BETI_getEotInterruptStatus (Uint8 *cpuld*)

Description

This function reads the content of the End-Of-Transfer Interrupt Status register to know which Output Descriptor Buffer has a new descriptor.

Arguments

`cpuld` Which CPU interrupt register to read.

Returns

All the EOT Interrupt registers.

Modifies

This operation reads the corresponding RAC_BETI_EOT_STAT# register.

Pre condition

None

Post condition

None

27.2.16 CSL_RAC_BETI_getObbtRdCrossingStatus

Uint32 CSL_RAC_BETI_getObbtRdCrossingStatus ()

Description

This function reads the content of the OBBT read pointer crossing register.

Arguments

None

Returns

All the OBBT Status.

Modifies

This operation reads the RAC_BETI_OBBT_STAT.

Pre condition

None

Post condition

None

27.2.17 CSL_RAC_BETI_getOdbtRdCrossingStatus

UInt32 CSL_RAC_BETI_getOdbtRdCrossingStatus ()

Description

This function reads the content of the ODBT read pointer crossing register.

Arguments

None

Returns

All the ODBT Status.

Modifies

This operation reads the RAC_BETI_ODBT_STAT.

Pre condition

None

Post condition

None

27.2.18 CSL_RAC_BETI_getOdbtStatus

CSL_RAC_BETI_odbStatusBit CSL_RAC_BETI_getOdbtStatus (UInt8 odbtEntryId)

Description

This function returns if the corresponding Output Descriptor Buffer has received at least one new output block descriptor.

Arguments

odbEntryId Which buffer status to get.

Returns

CSL_RAC_BETI_odbStatusBit_noNewObd When no new Output Block Descriptor has been generated for this output buffer.

CSL_RAC_BETI_odbStatusBit_newObd A new Output Block Descriptor has been written to the buffer.

Modifies

This operation reads the corresponding RAC_BETI_NEW_OBD register and extracts status bit for the given buffer.

Pre condition

None

Post condition

None

27.2.19 CSL_RAC_BETI_getStatus

CSL_RAC_BETI_statusBit CSL_RAC_BETI_getStatus ()

Description

This function returns the BETI status indicating whether the BETI is enabled or not.

Arguments

None

Returns

CSL_RAC_BETI_statusBit_Disable When the BETI is disabled.

CSL_RAC_BETI_statusBit_Enable When the BETI is enabled.

Modifies

This function reads the RAC_BETI_STAT register. Reading this register is relevant especially when disabling the BETI. When disabling, a transfer can be in process and the BETI has to complete it before being disabled.

Pre condition

None

Post condition

None

27.2.20 CSL_RAC_BETI_getWatchDogInterruptStatus

CSL_RAC_BETI_wdInterruptStatus **CSL_RAC_BETI_getWatchDogInterruptStatus (Uint8 gccpId)**

Description

This function reads the content of the minimal activity watch dog interrupt status for a given GCCP ID.

Arguments

`gccpId` Which GCCP minimal activity watch-dog status to read.

Returns

CSL_RAC_BETI_wdInterruptStatus_NoInt No interrupt has been generated.

CSL_RAC_BETI_wdInterruptStatus_Int An interrupt has been generated and forwarded to the BEII.

Modifies

This operation reads the corresponding RAC_BETI_MIN_WST[gccpId] register and extracts the interrupt status field.

Pre condition

None

Post condition

When the user reads the RAC_BETI_MIN_WST[gccpId] register, the interrupt status field is cleared by the H/W.

27.2.21 CSL_RAC_BETI_getWatchDogStatus

Uint32 **CSL_RAC_BETI_getWatchDogStatus (Uint8 gccpId)**

Description

This function reads the content of the minimal activity watch dog decouner for a given GCCP ID.

Arguments

<code>gccpId</code>	Which GCCP minimal activity watch-dog decouner to read.
---------------------	---

Returns

Uint16 Current value of the decouner.

Modifies

This operation reads the corresponding RAC_BETI_MIN_WST[`gccpId`] register and extracts the decouner field.

Pre condition

None

Post condition

None

27.2.22 CSL_RAC_BETI_setWatchDog

void CSL_RAC_BETI_setWatchDog (Uint16 *nbCyclesToReload*)

Description

This function configures the minimal activity watch dog in the BETI.

Arguments

<code>nbCycles</code>	ToReload Half-word with the value to set in the register.
-----------------------	---

Returns

None

Modifies

This operation modifies the RAC_BETI_MIN_WCFG register with the `nbCyclesToReload` value.

Pre condition

None

Post condition

The Back End watch-dogs are now configured.

27.2.23 CSL_RAC_Stats_getCfgReadAccess

Uint32 CSL_RAC_Stats_getCfgReadAccess ()

Description

This function reads the content of the statistics register for the total number of read data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_READ_REG register.

Pre condition

None

Post condition

None

27.2.24 CSL_RAC_Stats_getCfgTotalAccess

Uint32 CSL_RAC_Stats_getCfgTotalAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_TOT_REG register.

Pre condition

None

Post condition

None

27.2.25 CSL_RAC_Stats_getCfgWriteAccess

Uint32 CSL_RAC_Stats_getCfgWriteAccess ()

Description

This function reads the content of the statistics register for the total number of write data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_WRITE_REG register.

Pre condition

None

Post condition

None

27.2.26 CSL_RAC_Stats_getMasterHighPrioAccess

Uint32 CSL_RAC_Stats_getMasterHighPrioAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the high priority master interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_MST_HP_TOT_REG register.

Pre condition

None

Post condition

None

27.2.27 CSL_RAC_Stats_getMasterLowPrioAccess

Uint32 CSL_RAC_Stats_getMasterLowPrioAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the low priority master interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_MST_LP_TOT_REG register.

Pre condition

None

Post condition

None

27.2.28 CSL_RAC_Stats_getSlaveReadAccess

Uint32 CSL_RAC_Stats_getSlaveReadAccess ()

Description

This function reads the content of the statistics register for the total number of read data phases used by the DMA Slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_READ_REG register.

Pre condition

None

Post condition

None

27.2.29 CSL_RAC_Stats_getSlaveTotalAccess

Uint32 CSL_RAC_Stats_getSlaveTotalAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the DMA slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_TOT_REG register.

Pre condition

None

Post condition

None

27.2.30 CSL_RAC_Stats_getSlaveWriteAccess

Uint32 CSL_RAC_Stats_getSlaveWriteAccess ()

Description

This function reads the content of the statistics register for the total number of write data phases used by the DMA Slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_WRITE_REG register.

Pre condition

None

Post condition

None

27.2.31 CSL_RAC_BEII_disable

void CSL_RAC_BEII_disable ()

Description

This function disables the RAC Back End Interrupt Interface (BEII) by writing 0x0 into the RAC_BEII_CTRL register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BEII_CTRL register.

Pre condition

None

Post condition

The Back End Interrupt Interface is then disabled.

27.2.32 CSL_RAC_BEII_enable

void CSL_RAC_BEII_enable ()

Description

This function enables the RAC Back End Interrupt Interface (BEII) by writing 0x1 into the RAC_BEII_CTRL register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BEII_CTRL register.

Pre condition

None

Pst condition

The Back End Interrupt Interface is then enabled.

27.2.33 CSL_RAC_BEII_getInterruptStatus

void CSL_RAC_BEII_getInterruptStatus (Uint8 *cpuld*, CSL_RAC_BEII_interruptStatus **cpulntStatus*)

Description

This function reads the BEII Interrupt Status for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

CSL_RAC_BEII_interruptStatus

Modifies

This operation modifies the RAC_BEII_INT_STATx register.

Pre condition

None

Post condition

None

27.2.34 CSL_RAC_BEII_setBetiEotMask

void CSL_RAC_BEII_setBetiEotMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End End-Of-Transfer Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[*cpuld*] register.

Pre condition

None

Post Condition

None

27.2.35 CSL_RAC_BEII_setBetiObbtRdCrossingMask

void CSL_RAC_BEII_setBetiObbtRdCrossingMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End OBBT Read Pointer Crossing Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.36 CSL_RAC_BEII_setBetIOdbtRdCrossingMask

void CSL_RAC_BEII_setBetIOdbtRdCrossingMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End ODBT Read Pointer Crossing Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Precondition

None

Post condition

None

27.2.37 CSL_RAC_BEII_setBeWatchDogMask

void CSL_RAC_BEII_setBeWatchDogMask (Uint8 *cpuld*, Uint8 *gccpld*, CSL_RAC_BEII_interrupt *interruptEnable*)

Description

This function configures the Back End Min Activity Watch Dog Mask for the specified cpu interrupt controller.

Arguments

`cpuld` Which CPU interrupt controller to configure.

`gccpld` Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.38 CSL_RAC_BEII_setCycleOverflowMask

**void CSL_RAC_BEII_setCycleOverflowMask (Uint8 *cpuld*, Uint8 *gccpld*,
CSL_RAC_BEII_interrupt *interruptEnable*)**

Description

This function configures the Cycle Overflow Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

gccpld Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Postcondition

None

27.2.39 CSL_RAC_BEII_setFeWatchDogMask

**void CSL_RAC_BEII_setFeWatchDogMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt
interruptEnable)**

Description

This function configures the Front End Watch Dog Mask for the specified cpu interrupt controller.

Arguments

cpuld Which CPU interrupt controller to configure.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.40 CSL_RAC_BEII_setFifoOverflowMask

**void CSL_RAC_BEII_setFifoOverflowMask (Uint8 *cpuld*, Uint8 *gccpld*,
CSL_RAC_BEII_interrupt *interruptEnable*)**

Description

This function configures the FIFO Overflow Mask for the specified cpu interrupt controller.

Arguments

<code>cpuld</code>	Which CPU interrupt controller to configure.
<code>gccpId</code>	Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEII_INT_MSK[cpuld] register.

Pre condition

None

Post condition

None

27.2.41 CSL_RAC_BEII_setMasterMask

**void CSL_RAC_BEII_setMasterMask (Uint8 *cpuld*, CSL_RAC_BEII_interrupt
cpulntContrlEnable)**

Description

This function configures the BEII Master Mask for the specified cpu interrupt controller.

Arguments

<code>cpuld</code>	Which CPU interrupt controller to configure.
--------------------	--

Returns

None

Modifies

This operation modifies the RAC_BEII_MINT_ENA register.

Pre condition

None

Post condition

None

27.2.42 CSL_RAC_BEI_setSequencerIdleMask

void CSL_RAC_BEI_setSequencerIdleMask (Uint8 *cpuld*, Uint8 *gccpld*,
CSL_RAC_BEI_interrupt *interruptEnable*)

Description

This function configures the Sequencer Idle Mask for the specified cpu interrupt controller.

Arguments

<i>cpuld</i>	Which CPU interrupt controller to configure.
<i>gccpld</i>	Which Gccp source to enable.

Returns

None

Modifies

This operation modifies the corresponding RAC_BEI_INT_MSK[*cpuld*] register.

Pre condition

None

Post condition

None

27.2.43 CSL_RAC_BETI_disable

void CSL_RAC_BETI_disable ()

Description

This function disables the RAC Back End Transfer Interface (BETI) by writing 0x0 into the RAC_BETI_CTRL register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BETI_CTRL register.

Pre condition

None

Post condition

The Back End Transfer Interface is then disabled.

27.2.44 CSL_RAC_BETI_enable

void CSL_RAC_BETI_enable ()

Description

This function enables the RAC Back End Transfer Interface (BETI) by writing 0x1 into the RAC_BETI_CTRL register.

CSL_RAC_BETI_enable

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_BETI_CTRL register.

Pre condition

None

Post condition

The Back End Transfer Interface is then enabled.

27.2.45 CSL_RAC_BETI_getEotInterruptStatus

Uint32 CSL_RAC_BETI_getEotInterruptStatus (Uint8 *cpuld*)

Description

This function reads the content of the End-Of-Transfer Interrupt Status register to know which Output Descriptor Buffer has a new descriptor.

Arguments

cpuld Which CPU interrupt register to read.

Returns

All the EOT Interrupt registers.

Modifies

This operation reads the corresponding RAC_BETI_EOT_STAT# register.

Pre condition

None

Post condition

None

27.2.46 CSL_RAC_BETI_getObbtRdCrossingStatus

Uint32 CSL_RAC_BETI_getObbtRdCrossingStatus ()

Description

This function reads the content of the OBBT read pointer crossing register.

Arguments

None

Returns

All the OBBT Status.

Modifies

This operation reads the RAC_BETI_OBBT_STAT.

Pre condition

None

Post condition

None

27.2.47 CSL_RAC_BETI_getOdbtRdCrossingStatus

Uint32 CSL_RAC_BETI_getOdbtRdCrossingStatus ()

Description

This function reads the content of the ODBT read pointer crossing register.

Arguments

None

Returns

All the ODBT Status.

Modifies

This operation reads the RAC_BETI_ODBT_STAT.

Pre condition

None

Post condition

None

27.2.48 CSL_RAC_BETI_getOdbtStatus

CSL_RAC_BETI_odbStatusBit CSL_RAC_BETI_getOdbtStatus (Uint8 odbtEntryId)

Description

This function returns if the corresponding Output Descriptor Buffer has received at least one new output block descriptor.

Arguments

odbEntryId Which buffer status to get.

Returns

CSL_RAC_BETI_odbStatusBit_noNewObd When no new Output Block Descriptor has been generated for this output buffer.

CSL_RAC_BETI_odbStatusBit_newObd A new Output Block Descriptor has been written to the buffer.

Modifies

This operation reads the corresponding RAC_BETI_NEW_OBD register and extracts status bit for the given buffer.

Pre condition

None

Post condition

None

27.2.49 CSL_RAC_BETI_getStatus

CSL_RAC_BETI_statusBit CSL_RAC_BETI_getStatus ()

Description

This function returns the BETI status indicating whether the BETI is enabled or not.

Arguments

None

Returns

CSL_RAC_BETI_statusBit_Disable When the BETI is disabled.

CSL_RAC_BETI_statusBit_Enable When the BETI is enabled.

Modifies

This function reads the RAC_BETI_STAT register. Reading this register is relevant especially when disabling the BETI. When disabling, a transfer can be in process and the BETI has to complete it before being disabled.

Pre condition

None

Post condition

None

27.2.50 CSL_RAC_BETI_getWatchDogInterruptStatus

CSL_RAC_BETI_wdInterruptStatus CSL_RAC_BETI_getWatchDogInterruptStatus (UInt8 gccpld)

Description

This function reads the content of the minimal activity watch dog interrupt status for a given GCCP ID.

CSL_RAC_BETI_getWatchDogInterruptStatus

Arguments

gccpld Which GCCP minimal activity watch-dog status to read.

Returns

CSL_RAC_BETI_wdInterruptStatus_NoInt No interrupt has been generated.

CSL_RAC_BETI_wdInterruptStatus_Int An interrupt has been generated and forwarded to the BEII.

Modifies

This operation reads the corresponding RAC_BETI_MIN_WST[gccpld] register and extracts the interrupt status field.

Pre condition

None

Post condition

When the user reads the RAC_BETI_MIN_WST[gccpld] register, the interrupt status field is cleared by the H/W.

27.2.51 CSL_RAC_BETI_getWatchDogStatus

Uint32 CSL_RAC_BETI_getWatchDogStatus (Uint8 *gccpId*)

Description

This function reads the content of the minimal activity watch dog decouner for a given GCCP ID.

Arguments

<i>gccpId</i>	Which GCCP minimal activity watch-dog decouner to read.
---------------	---

Returns

Uint16 Current value of the decouner.

Modifies

This operation reads the corresponding RAC_BETI_MIN_WST[*gccpId*] register and extracts the decouner field.

Pre condition

None

Post condition

None

27.2.52 CSL_RAC_BETI_setWatchDog

void CSL_RAC_BETI_setWatchDog (Uint16 *nbCyclesToReload*)

Description

This function configures the minimal activity watch dog in the BETI.

Arguments

<i>nbCycles</i>	ToReload Half-word with the value to set in the register.
-----------------	---

Returns

None

Modifies

This operation modifies the RAC_BETI_MIN_WCFG register with the *nbCyclesToReload* value.

Pre condition

None

Post condition

The Back End watch-dogs are now configured.

27.2.53 CSL_RAC_Stats_getCfgReadAccess

Uint32 CSL_RAC_Stats_getCfgReadAccess ()

Description

This function reads the content of the statistics register for the total number of read data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_READ_REG register.

Pre condition

None

Post condition

None

27.2.54 CSL_RAC_Stats_getCfgTotalAccess

Uint32 CSL_RAC_Stats_getCfgTotalAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_TOT_REG register.

Pre condition

None

Post condition

None

27.2.55 CSL_RAC_Stats_getCfgWriteAccess

Uint32 CSL_RAC_Stats_getCfgWriteAccess ()

Description

This function reads the content of the statistics register for the total number of write data phases used by the Configuration interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_CFG_WRITE_REG register.

Pre condition

None

Post condition

None

27.2.56 CSL_RAC_Stats_getMasterHighPrioAccess

Uint32 CSL_RAC_Stats_getMasterHighPrioAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the high priority master interface.

CSL_RAC_Stats_getMasterHighPrioAccess

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_MST_HP_TOT_REG register.

Pre condition

None

Post condition

None

27.2.57 CSL_RAC_Stats_getMasterLowPrioAccess

Uint32 CSL_RAC_Stats_getMasterLowPrioAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the low priority master interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_MST_LP_TOT_REG register.

Pre condition

None

Post condition

None

27.2.58 CSL_RAC_Stats_getSlaveReadAccess

Uint32 CSL_RAC_Stats_getSlaveReadAccess ()

Description

This function reads the content of the statistics register for the total number of read data phases used by the DMA Slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_READ_REG register.

Pre condition

None

Post condition

None

27.2.59 CSL_RAC_Stats_getSlaveTotalAccess

Uint32 CSL_RAC_Stats_getSlaveTotalAccess ()

Description

This function reads the content of the statistics register for the total number of data phases used by the DMA slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_TOT_REG register.

Pre condition

None

Post condition

None

27.2.60 CSL_RAC_Stats_getSlaveWriteAccess

Uint32 CSL_RAC_Stats_getSlaveWriteAccess ()

Description

This function reads the content of the statistics register for the total number of write data phases used by the DMA Slave interface.

Arguments

None

Returns

The value of the register.

Modifies

This operation reads the corresponding RAC_SLV_WRITE_REG register.

Pre condition

None

Post condition

None

27.2.61 CSL_RAC_FE_disable

void CSL_RAC_FE_disable ()

Description

This function disables the RAC Front End Interface (FEI) by writing 0x0 into the RAC_FE_ENA register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_FE_ENA register.

Pre condition

None

Post condition

The Front End Interface is then disabled.

27.2.62 CSL_RAC_FE_enable

void CSL_RAC_FE_enable ()

Description

This function enables the RAC Front End Interface (FEI) by writing 0x1 into the RAC_FE_ENA register.

Arguments

None

Returns

None

Modifies

This operation modifies the RAC_FE_ENA register.

Pre condition

None

Post condition

The Front End Interface is then enabled.

27.2.63 CSL_RAC_FE_getGccpStatus

CSL_RAC_FE_gccpStatus **CSL_RAC_FE_getGccpStatus (Uint8 gccpId)**

Description

This function returns the corresponding GCCP Status by reading the Front End status register.

Arguments

gccpId Byte to select the GCCP status.

Returns

CSL_RAC_FE_gccpStatus_Idle When the GCCP is idle. The Data memories are accessible for debug.

CSL_RAC_FE_gccpStatus_Busy When the GCCP is busy. The Data memories are not accessible for debug.

Modifies

This function reads the RAC_FE_STAT register.

Pre condition

None

Post condition

None

27.2.64 CSL_RAC_FE_getStatus

CSL_RAC_FE_transferState **CSL_RAC_FE_getStatus ()**

Description

This function returns the Front End Transfer State by reading the Front End status register.

Arguments

None

Returns

CSL_RAC_FE_transferState_WaitingTimestamp When the Front End is waiting for the timestamp write.

CSL_RAC_FE_transferState_ReceivingSamples When the Front End receives samples.

CSL_RAC_FE_transferState_ReadyToStart When the Front End has received all samples and is ready to start the GCCPs.

CSL_RAC_FE_transferState_StartGccpIteration When the Front End enables the GCCPs.

Modifies

This function reads the RAC_FE_STAT register.

Pre condition

None

Post condition

None

27.2.65 CSL_RAC_FE_getTimestamp

void CSL_RAC_FE_getTimestamp (CSL_RAC_FE_Timestamp_req * *timestamp*)

Description

This function reads the Front End timestamp register.

Arguments

`timestamp` structure with timestamp parameters.

Returns

None

Modifies

This operation reads the content of the Front-End timestamp register.

Pre structure

None

Post Structure

None

27.2.66 CSL_RAC_FE_getWatchDogInterruptStatus

CSL_RAC_FE_wdInterruptStatus CSL_RAC_FE_getWatchDogInterruptStatus ()

Description

This function reads the content of the watch dog interrupt status.

Arguments

None

Returns

CSL_RAC_FE_wdInterruptStatus_NoInt No interrupt has been generated.

CSL_RAC_FE_wdInterruptStatus_Int An interrupt has been generated and forwarded to the BEII.

Modifies

This operation reads the RAC_FE_WINT register and extracts the interrupt status field.

Pre condition

None

Post condition

When the user reads the RAC_FE_WINT register, the H/W clears the interrupt status bit. If the user reads once again the register, 'no-interrupt' status will be return.

27.2.67 CSL_RAC_FE_getWatchDogStatus

Uint16 CSL_RAC_FE_getWatchDogStatus ()

Description

This function reads the content of the watch dog decounter.

Arguments

None

Returns

Uint16 Current value of the decounter.

Modifies

This operation reads the RAC_FE_WST register and extracts the decounter field.

Pre condition

None

Post condition

None

27.2.68 CSL_RAC_FE_setInputBufferDepth

void CSL_RAC_FE_setInputBufferDepth (Uint8 *ibDepth*)

Description

This function configures the Input Buffer Depth register.

Arguments

ibDepth Byte with the value to set in the register.

Returns

None

Modifies

This operation modifies the RAC_FE_IB_DEPTH register with the *ibDepth* value.

Pre condition

None

Post condition

The Front End is now able to compute the read time for the GCCPs.

27.2.69 CSL_RAC_FE_setMaxCyclesPerIteration

void CSL_RAC_FE_setMaxCyclesPerIteration (Uint16 *maxCyclesNb*)

Description

This function configures the Front End Max Cycles watch-dog.

Arguments

maxCyclesNb Half-word with the value to set in the register.

Returns

None

Modifies

This operation modifies the RAC_FE_MAX register with the *maxCyclesNb* value.

Pre condition

None

Post condition

The Front End watch-dog is configured.

27.2.70 CSL_RAC_FE_setTimestamp

void CSL_RAC_FE_setTimestamp (CSL_RAC_FE_Timestamp_req * *timestamp*)

Description

This function configures the Front End timestamp register.

Arguments

`Timestamp` structure with timestamp parameters.

Returns

None

Modifies

This operation modifies the RAC_FE_TIME register with the timestamp parameters. This function can be used for a non-periodic usage of the RAC.

Pre condition

None

Post condition

The Front End is ready to receive antenna samples.

27.2.71 CSL_RAC_GCCP_disable

void CSL_RAC_GCCP_disable (Uint8 *gccpId*)

Description

This function disables the corresponding Generic Correlation Co-Processor (GCCP) by writing 0x1 into the RAC_GCCP_SEQ_ENA register.

Arguments

`gccpId` Which GCCP to deactivate.

Returns

None

Modifies

This operation modifies the RAC_GCCP_SEQ_ENA register.

Pre condition

None

Post condition

The GCCP Sequencer is then disabled.

27.2.72 CSL_RAC_GCCP_enable

void CSL_RAC_GCCP_enable (Uint8 *gccpId*)

Description

This function enables the corresponding Generic Correlation Co-Processor (GCCP) by writing 0x1 into the RAC_GCCP_SEQ_ENA register.

Arguments

gccpId Which GCCP to activate.

Returns

None

Modifies

This operation modifies the RAC_GCCP_SEQ_ENA register.

Pre condition

None

Post condition

The GCCP Sequencer is then enabled.

27.2.73 CSL_RAC_GCCP_getActiveCycles

Uint16 CSL_RAC_GCCP_getActiveCycles (Uint8 *gccpId*)

Description

This function returns the number of active cycles in the previous iteration.

Arguments

gccpId Which GCCP to get status from.

Returns

The number of active cycles

Modifies

This function reads the RAC_GCCP_SEQ_ACT register.

Pre condition

None

Post condition

None

27.2.74 CSL_RAC_GCCP_getCgtEntry

Uint32 CSL_RAC_GCCP_getCgtEntry (Uint8 *gccpId*, Uint8 *entryId*)

Description

This function reads one of the RAC_GCCP_CGT registers.

Arguments

gccpId Which GCCP CGT to configure.

entryId Which entry to configure.

Returns

None

Modifies

This operation returns the Y part for the given GCCP and entry ID.

Pre condition

None

Post condition

None

27.2.75 CSL_RAC_GCCP_getFifoOverflowStatus

**void CSL_RAC_GCCP_getFifoOverflowStatus (Uint8 *gccpId*,
CSL_RAC_GCCP_fifoOverflowStatus * *fifoOverflowStatus*)**

Description

This function returns the last active TRM address.

Arguments

`gccpId` Which GCCP to get status from.

Returns

`CSL_RAC_GCCP_fifoOverflowStatus` The last active TRM address before a FIFO overflow

Modifies

This function reads the RAC_GCCP_FIFO_OVER register.

Pre condition

None

Post condition

None

27.2.76 CSL_RAC_GCCP_getHighPrioControlLevel

Uint32 CSL_RAC_GCCP_getHighPrioControlLevel (Uint8 *gccpId*)

Description

This function returns the High Priority Control component level of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_HCQ_CURR_LVL register.

Pre condition

None

Post condition

None

27.2.77 CSL_RAC_GCCP_getHighPrioControlWatermark

Uint32 CSL_RAC_GCCP_getHighPrioControlWatermark (Uint8 *gccpId*)

Description

This function reads the High Priority Control component water-mark of the output queue for a given GCCP ID.

Arguments

gccpId Which GCCP status to read.

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_HCQ_WTMK_LVL register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.78 CSL_RAC_GCCP_getHighPrioDataLevel

Uint32 CSL_RAC_GCCP_getHighPrioDataLevel (Uint8 *gccpId*)

Description

This function returns the High Priority Data component level of the output queue for a given GCCP ID.

Arguments

gccpId Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_HDQ_CURR_LVL register.

Pre condition

None

Post condition

None

27.2.79 CSL_RAC_GCCP_getHighPrioDataWatermark

Uint32 CSL_RAC_GCCP_getHighPrioDataWatermark (Uint8 *gccpId*)

Description

This function reads the High Priority Data component water-mark of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

Uint16 Current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_HDQ_WTMK_LVL register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.80 CSL_RAC_GCCP_getLowPrioControlLevel

Uint32 CSL_RAC_GCCP_getLowPrioControlLevel (Uint8 *gccpId*)

Description

This function returns the Low Priority Control component level of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_LCQ_CURR_LVL register.

Pre condition

None

Post condition

None

27.2.81 CSL_RAC_GCCP_getLowPrioControlWatermark

Uint32 CSL_RAC_GCCP_getLowPrioControlWatermark (Uint8 *gccpId*)

Description

This function reads the Low Priority Control component water-mark of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_LCQ_WTMK_LVL register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.82 CSL_RAC_GCCP_getLowPrioDataLevel

Uint32 CSL_RAC_GCCP_getLowPrioDataLevel (Uint8 gccpId)

Description

This function returns the Low Priority Data component level of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_LDQ_CURR_LVL register.

Pre condition

None

Post condition

None

27.2.83 CSL_RAC_GCCP_getLowPrioDataWatermark

Uint32 CSL_RAC_GCCP_getLowPrioDataWatermark (Uint8 gccpId)

Description

This function reads the Low Priority Data component water-mark of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_LDQ_WTMK_LVL register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.84 CSL_RAC_GCCP_getPrtEntry

Uint32 CSL_RAC_GCCP_getPrtEntry (Uint8 gccpId, Uint8 entryId, Uint8 wordId)

Description

This function reads one of the RAC_GCCP_PRT registers.

Arguments

<code>gccpId</code>	Which GCCP PRT to configure.
<code>entryId</code>	Which entry to configure.
<code>worded</code>	Which slot to configure.

Returns

None

Modifies

This operation returns the slot format for the given slot ID.

Pre condition

None

Post condition

None

27.2.85 CSL_RAC_GCCP_getReadTime

void CSL_RAC_GCCP_getReadTime (CSL_RAC_FE_Timestamp_req * *timestamp*)

Description

This function reads the GCCP[0] read timestamp register.

Arguments

<code>timestamp</code>	structure with timestamp parameters.
------------------------	--------------------------------------

Returns

None

Modifies

This operation reads the content of the GCCP[0] read timestamp register.

Pre condition

None

Post condition

None

27.2.86 CSL_RAC_GCCP_getSequencerCycles

Uint16 CSL_RAC_GCCP_getSequencerCycles (Uint8 *gccpId*)

Description

This function returns the number of sequencer-used cycles in the previous iteration.

Arguments

<code>gccpId</code>	Which GCCP to get status from.
---------------------	--------------------------------

Returns

The number of sequencer cycles

Modifies

This function reads the RAC_GCCP_SEQ_CYC register.

Pre condition

None

Post condition

None

27.2.87 CSL_RAC_GCCP_resetHighPriorityQueue

void CSL_RAC_GCCP_resetHighPriorityQueue (Uint8 gccpId)

Description

This function resets the High Priority Output Queue of the corresponding GCCP by writing into the RAC_GCCP_FIFO_RESET register.

Arguments

gccpId Which GCCP FIFO to reset.

Returns

None

Modifies

This operation modifies the RAC_GCCP_FIFO_RESET register.

Pre condition

None

Post condition

The GCCP High Priority Output Queue is reset.

27.2.88 CSL_RAC_GCCP_resetLowPriorityQueue

void CSL_RAC_GCCP_resetLowPriorityQueue (Uint8 gccpId)

Description

This function resets the Low Priority Output Queue of the corresponding GCCP by writing into the RAC_GCCP_FIFO_RESET register.

Arguments

gccpId Which GCCP FIFO to reset.

Returns

None

Modifies

This operation modifies the RAC_GCCP_FIFO_RESET register.

Pre condition

None

Post condition

The GCCP Low Priority Output Queue is reset.

27.2.89 CSL_RAC_GCCP_setCgtEntry

void CSL_RAC_GCCP_setCgtEntry (Uint8 *gccpId*, Uint8 *entryId*, Uint32 *partY*)

Description

This function configures the Code Generation Table by writing into one of the RAC_GCCP_CGT registers.

Arguments

<i>gccpId</i>	Which GCCP CGT to configure.
<i>entryId</i>	Which entry to configure.
<i>partY</i>	Value to write into the register.

Returns

None

Modifies

This operation modifies one of the RAC_GCCP_CGT register.

Pre condition

None

Post condition

None

27.2.90 CSL_RAC_GCCP_setIctEntry

void CSL_RAC_GCCP_setIctEntry (Uint8 *gccpId*, Uint8 *entryId*, Int8 *coeff0*, Int8 *coeff1*, Int8 *coeff2*, Int8 *coeff3*)

Description

This function configures the Interpolation Coefficients Table by writing into one of the RAC_ICT registers.

Arguments

<i>gccpId</i>	Which GCCP ICT to configure.
<i>entryId</i>	Which GCCP ICT to configure. It ranges from 0 to 4.
<i>coeff0</i>	Value to set into the coefficient #0 field. It should range from -64 to 64
<i>coeff1</i>	Value to set into the coefficient #1 field. It should range from -64 to 64
<i>coeff2</i>	Value to set into the coefficient #2 field. It should range from -64 to 64
<i>coeff3</i>	Value to set into the coefficient #3 field. It should range from -64 to 64

Returns

None

Modifies

This operation modifies the RAC_GCCP_ICT register.

Pre condition

None

Post condition

None

27.2.91 CSL_RAC_GCCP_setPrtEntry

void CSL_RAC_GCCP_setPrtEntry (Uint8 *gccpId*, Uint8 *entryId*, Uint8 *wordId*, Uint32 *format*)

Description

This function configures the Pilot Rotation Table by writing into one of the RAC_GCCP_PRT registers.

Arguments

<i>gccpId</i>	Which GCCP PRT to configure.
<i>entryId</i>	Which entry to configure.
<i>worded</i>	Which slot to configure.
<i>format</i>	Value to insert into the register.

Returns

None

Modifies

This operation modifies one of the RAC_GCCP_PRT register.

Pre condition

None

Post condition

None

27.2.92 CSL_RAC_GCCP_disable

void CSL_RAC_GCCP_disable (Uint8 *gccpId*)

Description

This function disables the corresponding Generic Correlation Co-Processor (GCCP) by writing 0x1 into the RAC_GCCP_SEQ_ENA register.

Arguments

<i>gccpId</i>	Which GCCP to deactivate.
---------------	---------------------------

Returns

None

Modifies

This operation modifies the RAC_GCCP_SEQ_ENA register.

Pre condition

None

Post condition

The GCCP Sequencer is then disabled.

27.2.93 CSL_RAC_GCCP_enable

void CSL_RAC_GCCP_enable (Uint8 *gccpId*)

Description

This function enables the corresponding Generic Correlation Co-Processor (GCCP) by writing 0x1 into the RAC_GCCP_SEQ_ENA register.

Argument

gccpId Which GCCP to activate.

Returns

None

Modifies

This operation modifies the RAC_GCCP_SEQ_ENA register.

Pre condition

None

Post condition

The GCCP Sequencer is then enabled.

27.2.94 CSL_RAC_GCCP_getActiveCycles

Uint16 CSL_RAC_GCCP_getActiveCycles (Uint8 *gccpId*)

Description

This function returns the number of active cycles in the previous iteration.

Arguments

gccpId Which GCCP to get status from.

Returns

The number of active cycles

Modifiers

This function reads the RAC_GCCP_SEQ_ACT register.

Pre condition

None

Post condition

None

27.2.95 CSL_RAC_GCCP_getCgtEntry

Uint32 CSL_RAC_GCCP_getCgtEntry (Uint8 *gccpId*, Uint8 *entryId*)

Description

This function reads one of the RAC_GCCP_CGT registers.

Arguments

gccpId Which GCCP CGT to configure.

entryId Which entry to configure.

Returns

None

Modifies

This operation returns the Y part for the given GCCP and entry ID.

Pre condition

None

Post condition

None

27.2.96 CSL_RAC_GCCP_getCycleOverflowStatus

**void CSL_RAC_GCCP_getCycleOverflowStatus (Uint8 *gccpId*,
CSL_RAC_GCCP_cycleOverflowStatus * *cycleOverflowStatus*)**

Description

This function collects cycle over flow status.

Arguments

gccpId Which GCCP PRT to configure.

Returns

None

Modifies

This function collects cycle over flow status.

Pre condition

None

Post conditions

None

27.2.97 CSL_RAC_GCCP_getFifoOverflowStatus

**void CSL_RAC_GCCP_getFifoOverflowStatus (Uint8 *gccpId*,
CSL_RAC_GCCP_fifoOverflowStatus * *fifoOverflowStatus*)**

Description

This function returns the last active TRM address.

Arguments

`gccpId` Which GCCP to get status from.

Returns

`CSL_RAC_GCCP_fifoOverflowStatus` The last active TRM address before a FIFO overflow

Modifies

This function reads the `RAC_GCCP_FIFO_OVER` register.

Pre condition

None

Post conditions

None

27.2.98 CSL_RAC_GCCP_getHighPrioControlLevel

Uint32 CSL_RAC_GCCP_getHighPrioControlLevel (Uint8 *gccpId*)

Description

This function returns the High Priority Control component level of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding `RAC_GCCP_HCQ_CURR_LVL` register.

Pre condition

None

Post condition

None

27.2.99 CSL_RAC_GCCP_getHighPrioControlWatermark

Uint32 CSL_RAC_GCCP_getHighPrioControlWatermark (Uint8 *gccpId*)

Description

This function reads the High Priority Control component water-mark of the output queue for a given GCCP ID.

Arguments

`gccpId` Which GCCP status to read.

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding `RAC_GCCP_HCQ_WTMK_LVL` register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.100 CSL_RAC_GCCP_getHighPrioDataLevel

Uint32 CSL_RAC_GCCP_getHighPrioDataLevel (Uint8 *gccpId*)

Description

This function returns the High Priority Data component level of the output queue for a given GCCP ID.

Arguments

gccpId Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_HDQ_CURR_LVL register.

Pre conditions

None

Post conditions

None

27.2.101 CSL_RAC_GCCP_getHighPrioDataWatermark

Uint32 CSL_RAC_GCCP_getHighPrioDataWatermark (Uint8 *gccpId*)

Description

This function reads the High Priority Data component water-mark of the output queue for a given GCCP ID.

Arguments

gccpId Which GCCP status to read.

Returns

Uint16 Current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_HDQ_WTMK_LVL register.

Pre conditions

None

Post conditions

The register is reset with the current level value.

27.2.102 CSL_RAC_GCCP_getLowPrioControlLevel

Uint32 CSL_RAC_GCCP_getLowPrioControlLevel (Uint8 *gccpId*)

Description

This function returns the Low Priority Control component level of the output queue for a given GCCP ID.

CSL_RAC_GCCP_getLowPrioControlLevel

Arguments

gccpId Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_LCQ_CURR_LVL register.

Pre condition

None

Post condition

None

27.2.103 CSL_RAC_GCCP_getLowPrioControlWatermark

Uint32 CSL_RAC_GCCP_getLowPrioControlWatermark (Uint8 *gccpId*)

Description

This function reads the Low Priority Control component water-mark of the output queue for a given GCCP ID.

Arguments

GccpId Which GCCP status to read.

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_LCQ_WTMK_LVL register.

Pre conditions

None

Post conditions

The register is reset with the current level value.

27.2.104 CSL_RAC_GCCP_getLowPrioDataLevel

Uint32 CSL_RAC_GCCP_getLowPrioDataLevel (Uint8 *gccpId*)

Description

This function returns the Low Priority Data component level of the output queue for a given GCCP ID.

Arguments

gccpId Which GCCP status to read.

Returns

The current value of the level

Modifies

This operation reads the corresponding RAC_GCCP_LDQ_CURR_LVL register.

Pre conditions

None

Post condition

None

27.2.105 CSL_RAC_GCCP_getLowPrioDataWatermark

Uint32 CSL_RAC_GCCP_getLowPrioDataWatermark (Uint8 *gccpId*)

Description

This function reads the Low Priority Data component water-mark of the output queue for a given GCCP ID.

Arguments

<i>gccpId</i>	Which GCCP status to read.
---------------	----------------------------

Returns

The current value of the water-mark

Modifies

This operation reads the corresponding RAC_GCCP_LDQ_WTMK_LVL register.

Pre condition

None

Post condition

The register is reset with the current level value.

27.2.106 CSL_RAC_GCCP_getPrtEntry

Uint32 CSL_RAC_GCCP_getPrtEntry (Uint8 *gccpId*, Uint8 *entryId*, Uint8 *wordId*)

Description

This function reads one of the RAC_GCCP_PRT registers.

Arguments

<i>gccpId</i>	Which GCCP PRT to configure.
<i>entryId</i>	Which entry to configure.
<i>worded</i>	Which slot to configure.

Returns

None

Modifies

This operation returns the slot format for the given slot ID.

Pre condition

None

Post condition

None

27.2.107 CSL_RAC_GCCP_getReadTime

void CSL_RAC_GCCP_getReadTime (CSL_RAC_FE_Timestamp_req * *timestamp*)

Description

This function reads the GCCP[0] read timestamp register.

Arguments

`Timestamp` structure with timestamp parameters.

Returns

None

Modifies

This operation reads the content of the GCCP[0] read timestamp register.

Pre condition

None

Post condition

None

27.2.108 CSL_RAC_GCCP_getSequencerCycles

Uint16 CSL_RAC_GCCP_getSequencerCycles (Uint8 *gccpId*)

Description

This function returns the number of sequencer-used cycles in the previous iteration.

`CSL_RAC_GCCP_getSequencerCycles`

Arguments

`gccpId` Which GCCP to get status from.

Returns

The number of sequencer cycles

Modifies

This function reads the RAC_GCCP_SEQ_CYC register.

Pre conditions

None

Post conditions

None

27.2.109 CSL_RAC_GCCP_resetHighPriorityQueue

void CSL_RAC_GCCP_resetHighPriorityQueue (Uint8 *gccpId*)

Description

This function resets the High Priority Output Queue of the corresponding GCCP by writing into the RAC_GCCP_FIFO_RESET register.

Arguments

gccpId Which GCCP FIFO to reset.

Returns

None

Modifies

This operation modifies the RAC_GCCP_FIFO_RESET register.

Pre conditions

None

Post conditions

The GCCP High Priority Output Queue is reset.

27.2.110 CSL_RAC_GCCP_resetLowPriorityQueue

void CSL_RAC_GCCP_resetLowPriorityQueue (Uint8 *gccpId*)

Description

This function resets the Low Priority Output Queue of the corresponding GCCP by writing into the RAC_GCCP_FIFO_RESET register.

Arguments

gccpId Which GCCP FIFO to reset.

Returns

None

Modifies

This operation modifies the RAC_GCCP_FIFO_RESET register.

Pre condition

None

Post condition

The GCCP Low Priority Output Queue is reset.

27.2.111 CSL_RAC_GCCP_setCgtEntry

void CSL_RAC_GCCP_setCgtEntry (Uint8 *gccpId*, Uint8 *entryId*, Uint32 *partY*)

Description

This function configures the Code Generation Table by writing into one of the RAC_GCCP_CGT registers.

Arguments

gccpId Which GCCP CGT to configure.

<code>entryId</code>	Which entry to configure.
<code>party</code>	Value to write into the register.

Returns

None

Modifies

This operation modifies one of the RAC_GCCP_CGT register.

Pre condition

None

Post condition

None

27.2.112 CSL_RAC_GCCP_setIctEntry

void CSL_RAC_GCCP_setIctEntry (Uint8 *gccpId*, Uint8 *entryId*, Int8 *coeff0*, Int8 *coeff1*, Int8 *coeff2*, Int8 *coeff3*)

Description

This function configures the Interpolation Coefficients Table by writing into one of the RAC_ICT registers.

Arguments

<code>gccpId</code>	Which GCCP ICT to configure.
<code>entryId</code>	Which GCCP ICT to configure. It ranges from 0 to 4.
<code>coeff0</code>	Value to set into the coefficient #0 field. It should range from -64 to 64
<code>coeff1</code>	Value to set into the coefficient #1 field. It should range from -64 to 64
<code>coeff2</code>	Value to set into the coefficient #2 field. It should range from -64 to 64
<code>coeff3</code>	Value to set into the coefficient #3 field. It should range from -64 to 64

Returns

None

Modifies

This operation modifies the RAC_GCCP_ICT register.

Pre condition

None

Post condition

None

27.2.113 CSL_RAC_GCCP_setPrtEntry

void CSL_RAC_GCCP_setPrtEntry (Uint8 *gccpId*, Uint8 *entryId*, Uint8 *wordId*, Uint32 *format*)

Description

This function configures the Pilot Rotation Table by writing into one of the RAC_GCCP_PRT registers.

Arguments

<code>gccpId</code>	Which GCCP PRT to configure.
<code>entryId</code>	Which entry to configure.
<code>worded</code>	Which slot to configure.
<code>format</code>	Value to insert into the register.

Returns

None

Modifies

This operation modifies one of the RAC_GCCP_PRT register.

Pre condition

None

Post condition

None

27.3 Data Structures

27.3.1 `_CSL_RAC_FE_Timestamp_req`

Detailed Description

This descriptor specifies the parameters required to setup a timestamp.

Field Documentation

UInt16 `_CSL_RAC_FE_Timestamp_req::chipId`

Specifies the chip value.

UInt16 `_CSL_RAC_FE_Timestamp_req::frameId`

Specifies the slot value.

UInt8 `_CSL_RAC_FE_Timestamp_req::slotId`

Specifies the frame value.

27.3.2 `_CSL_RAC_GCCP_cycleOverflow`

Detailed Description

This descriptor specifies the parameters required obtained when reading the cycle overflow status register.

Field Documentation

UInt8 `_CSL_RAC_GCCP_cycleOverflowStatus::cycOverFlag`

Denotes the cycle overflow flag

UInt16 `_CSL_RAC_GCCP_cycleOverflowStatus::iteNb`

Denotes the iteration Number.

UInt8 `_CSL_RAC_GCCP_cycleOverflowStatus::pageIdx`

Denotes the page index.

UInt8 `_CSL_RAC_GCCP_cycleOverflowStatus::taskIdx`

Denotes the task index.

27.3.3 `_CSL_RAC_GCCP_fifoOverflowStatus`

Detailed Description

This descriptor specifies the parameters required obtained when reading the fifo overflow status register.

Field Documentation

UInt8 `_CSL_RAC_GCCP_fifoOverflowStatus::cycOverFlag`

Denotes the cycle overflow flag

UInt8 `_CSL_RAC_GCCP_fifoOverflowStatus::fifId`

Denotes the FIFO ID

UInt16 `_CSL_RAC_GCCP_fifoOverflowStatus::iteNb`

Denotes the iteration Number.

UInt8 _CSL_RAC_GCCP_fifoOverflowStatus::pageIdx

Denotes the page index.

UInt8 _CSL_RAC_GCCP_fifoOverflowStatus::taskIdx

Denotes the task index.

27.3.4 CSL_RAC_BEII_interruptStatus

Detailed Description

This descriptor specifies the parameters obtained from the interrupt status register.

Field Documentation

UInt8 CSL_RAC_BEII_interruptStatus::eotStat

Denotes the BETI 'end-of-transfer' status.

UInt8 CSL_RAC_BEII_interruptStatus::feWdStat

Denotes the Front-end Watchdog status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp0CycOverStat

Denotes the GCCP0 Cycle Overflow status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp0FifoOverStat

Denotes the GCCP0 FIFO Overflow status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp0SeqStat

Denotes the GCCP0 Sequencer Idle status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp0WdStat

Denotes the GCCP0 Watchdog status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp1CycOverStat

Denotes the GCCP#1 Cycle Overflow status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp1FifoOverStat

Denotes the GCCP1 FIFO Overflow status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp1SeqStat

Denotes the GCCP1 Sequencer Idle status.

UInt8 CSL_RAC_BEII_interruptStatus::gccp1WdStat

Denotes the GCCP1 Watchdog status.

UInt8 CSL_RAC_BEII_interruptStatus::obbtRdPtrStat

Denotes the BETI OBBT read pointer crossing status.

UInt8 CSL_RAC_BEII_interruptStatus::odbtRdPtrStat

Denotes the BETI ODBT read pointer crossing status.

27.4 Enumerations

27.4.1 CSL_RAC_BEI_interrupt

enum CSL_RAC_BEI_interrupt

Enumeration values

CSL_RAC_BEI_interrupt_Disable specifies no interrupt forwarded.

CSL_RAC_BEI_interrupt_Enable specifies interrupt forwarded.

27.4.2 CSL_RAC_BETI_odbtStatusBit

enum CSL_RAC_BETI_odbtStatusBit

Enumeration values:

CSL_RAC_BETI_odbtStatusBit_noNewObd specifies no new Output Block Descriptor has been generated.

CSL_RAC_BETI_odbtStatusBit_newObd specifies a new Output Block Descriptor has been written to the buffer.

27.4.3 CSL_RAC_BETI_readParamsUpdateStatus

enum CSL_RAC_BETI_readParamsUpdateStatus

Enumeration values

CSL_RAC_BETI_readParams_UpdateStatus_Idle Specifies No pending request. A new request can be sent.

CSL_RAC_BETI_readParams_UpdateStatus_Pending A current request is still pending. No new request can be sent.

27.4.4 CSL_RAC_BETI_statusBit

enum CSL_RAC_BETI_statusBit

Enumeration values

CSL_RAC_BETI_statusBit_Disable Specifies BETI is disabled.

27.4.5 CSL_RAC_BETI_wdInterruptStatus

enum CSL_RAC_BETI_wdInterruptStatus

Enumeration values

CSL_RAC_BETI_wdInterrupt_Status_NoInt specifies no interrupt has been generated.

CSL_RAC_BETI_ specifies an interrupt has been generated and forwarded

wdInterruptStatus_Int to the BEII.

27.4.6 CSL_RAC_FE_gccpStatus

enum CSL_RAC_FE_gccpStatus

Enumeration values

CSL_RAC_FE_gccpStatus_Idle The GCCP is idle. The Data memories are accessible for debug.

CSL_RAC_FE_gccpStatus_Busy The GCCP is busy. The Data memories are not accessible for debug.

27.4.7 CSL_RAC_FE_transferState

enum CSL_RAC_FE_transferState

Enumeration values

CSL_RAC_FE_transferState_WaitingTimestamp The Front End is waiting for the timestamp write.

CSL_RAC_FE_transferState_ReceivingSamples The Front End receives samples.

CSL_RAC_FE_transferState_ReadyToStart The Front End has received all samples and is ready to start the GCCPs.

CSL_RAC_FE_transferState_StartGccpIteration The Front End enables the GCCPs.

27.4.8 CSL_RAC_FE_wdInterruptStatus

enum CSL_RAC_FE_wdInterruptStatus

Enumeration values

CSL_RAC_FE_wdInterruptStatus_NoInt Specifies no interrupt has been generated.

CSL_RAC_FE_wdInterruptStatus_Int Specifies an interrupt has been generated and forwarded to the BEII.

Chapter 28 EMAC Module

Topics

<u>28.1 Overview</u>
<u>28.2 Functions</u>
<u>28.3 Data Structures</u>

28.1 Overview

The Ethernet Media Access Controller (EMAC) module provides an interface between the TCI6488 DSP core processor and the networked community. The EMAC supports 10Base-T (10 Mbps/second [Mbps]), and 100BaseTX (100 Mbps), in either half- or full-duplex mode, and 1000BaseT (1000 Mbps) in full-duplex mode, with hardware flow control and quality-of-service (QoS) support.

The EMAC module conforms to the IEEE 802.3-2002 standard, describing the “Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer” specifications. The IEEE 802.3 standard has also been adopted by ISO/IEC and re-designated as ISO/IEC 8802-3:2000(E). Deviation from this standard, the EMAC module does not use the Transmit Coding Error signal MTXER. Instead of driving the error pin when an underflow condition occurs on a transmitted frame, the EMAC will intentionally generate an incorrect checksum by inverting the frame CRC, so that the transmitted frame will be detected as an error by the network.

The EMAC control module is the main interface between the device core processor, the MDIO module, and the EMAC module. The EMAC control module contains the necessary components to allow the EMAC to make efficient use of device memory, plus it controls device interrupts. The EMAC control module incorporates 8K-bytes of internal RAM to hold EMAC buffer descriptors.

Post Condition

None

Modifies

None

Example

```
EMAC_enumerate( );
```

28.2.3 EMAC_getReceiveFilter

Uint32 EMAC_getReceiveFilter (Handle *hEMAC*, Uint32 * *pReceiveFilter*)

Description

Called to get the current packet filter setting for received packets. The filter values are the same as those used in **EMAC_setReceiveFilter()**.

The current filter value is written to the pointer supplied in *pReceiveFilter*.

The function returns zero on success, or an error code on failure.

Possible error code include: EMAC_ERROR_INVALID - A calling parameter is invalid

Arguments

<i>hEMAC</i>	handle to the opened EMAC device
<i>pReceiveFilter</i>	Current receive packet filter

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened before calling this API and must be set the packet filter value.

Post Condition

The current filter value is written to the pointer supplied

Modifies

Output parameter *pReceiveFilter* being passed

Example

```
#define EMAC_RXFILTER_DIRECT      1
EMAC_Config    ecfg;
Handle        hEMAC = 0;
Uint32        pReceiveFilter;

EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);

EMAC_setReceiveFilter(hEMAC, EMAC_RXFILTER_DIRECT );

EMAC_getReceiveFilter(hEMAC, &pReceiveFilter );
```

28.2.4 EMAC_getStatistics

Uint32 EMAC_getStatistics (Handle *hEMAC*, EMAC_Statistics * *pStatistics*)

Description

Called to get the current device statistics. The statistics structure contains a collection of event counts for various packet sent and receive properties. Reading the statistics also clears the current statistic counters, so the values read represent a delta from the last call.

The statistics information is copied into the structure pointed to by the *pStatistics* argument.

The function returns zero on success, or an error code on failure.

Possible error code include: `EMAC_ERROR_INVALID` - A calling parameter is invalid

Arguments

<code>hEMAC</code>	handle to the opened EMAC device
<code>pStatistics</code>	Get the device statistics

Return Value

Success (0)

`EMAC_ERROR_INVALID` - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened before calling this API

Post Condition

1. Statistics are read for various packets sent and received.
2. Reading the statistics also clears the current statistic counters, so the values read represent a delta from the last call.

Modifies

Output parameter *pStatistics* being passed

Example

```
EMAC_Config      ecfg;
Handle          hEMAC = 0;
EMAC_Statistics  pStatistics;

EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);

EMAC_getStatistics(hEMAC, &pStatistics );
```

28.2.5 EMAC_getStatus

Uint32 EMAC_getStatus (Handle *hEMAC*, EMAC_Status * *pStatus*, Uint8 *coreNum*)

Description

Called to get the current status of the device. The device status is copied into the supplied data structure.

The function returns zero on success, or an error code on failure.

Possible error code include: `EMAC_ERROR_INVALID` - A calling parameter is invalid

Arguments

<code>hEMAC</code>	handle to the opened EMAC device
<code>pStatus</code>	Status of the EMAC

coreNum	core number
---------	-------------

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened before calling this API.

Post Condition

The current status of the device is copied into the supplied data structure.

Modifies

Output parameter *pStatus* being passed

Example

```
EMAC_Status status;
EMAC_Config ecfg;
Handle      hEMAC = 0;

//Open the EMAC peripheral
EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);

EMAC_getStatus( hEMAC, &status, 0 );
```

28.2.6 EMAC_open

**Uint32 EMAC_open (int *physicalIndex*, Handle *hApplication*, EMAC_Config *
pEMACConfig, Handle * *phEMAC*)**

Description

Opens the EMAC peripheral at the given physical index and initializes it to an embryonic state. The calling application must supply a operating configuration that includes a callback function table. Data from this config structure is copied into the device's internal instance structure so the structure may be discarded after **EMAC_open()** returns. In order to change an item in the configuration, the EMAC device must be closed and then re-opened with the new configuration. The application layer may pass in an hApplication callback handle, that will be supplied by the EMAC device when making calls to the application callback functions.

An EMAC device handle is written to phEMAC. This handle must be saved by the caller and then passed to other EMAC device functions.

The default receive filter prevents normal packets from being received until the receive filter is specified by calling EMAC_receiveFilter().

A device reset is achieved by calling **EMAC_close()** followed by **EMAC_open()**.

The function returns zero on success, or an error code on failure.

Possible error codes include: EMAC_ERROR_ALREADY - The device is already open

EMAC_ERROR_INVALID - A calling parameter is invalid

Arguments

physicalIndex	physical index
hApplication	application handle
pEMACConfig	EMAC's configuration structure

phEMAC handle to the EMAC device

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_ALREADY - The device is already open

Pre Condition

None

Post Condition

Opens the EMAC peripheral at the given physical index and initializes it.

Modifies

EMAC configuration registers

Example

```
#define  EMAC_CONFIG_MODEFLG_MACLOOPBACK  0x0002
#define  MDIO_MODEFLG_FD1000              0x0020
#define  MDIO_MODEFLG_EXTLOOPBACK         0x0100

EMAC_Config  ecfg;
Handle      hEMAC = 0;
Uint32      i = 0, j = 0;
// Setup the EMAC local loopback
ecfg.ModeFlags      = EMAC_CONFIG_MODEFLG_MACLOOPBACK;
ecfg.MdioModeFlags  = MDIO_MODEFLG_FD1000 |
                      MDIO_MODEFLG_EXTLOOPBACK;

ecfg.TxChannels      = 1;
ecfg.RxChannels      = 1;
ecfg.RxMaxPktPool    = 8;
//EMAC address from where it is stored in hardware.
//For this example set EMAC address for core 0 to be
00:01:02:03:04:05
for( i=0; i<3; i++ ){
    for (j=0; j<6; j++){
        if(j==0)
            ecfg.MacAddr[i][j] = j;
        else
            ecfg.MacAddr[i][j] = 0;
    }
}
```

```
EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);
```

28.2.7 EMAC_RxServiceCheck

Uint32 EMAC_RxServiceCheck (Handle *hEMAC*, Uint8 *coreNum*)

Description

This function should be called every time there is an EMAC device Rx interrupt. It maintains the status the EMAC.

Note that the application has the responsibility for mapping the physical device index to the correct EMAC_serviceCheck() function. If more than one EMAC device is on the same interrupt, the function must be called for each device.

Possible error codes include: EMAC_ERROR_INVALID - A calling parameter is invalid
EMAC_ERROR_MACFATAL - Fatal error in the MAC - Call **EMAC_close()**

Arguments

hEMAC	handle to the opened EMAC device
coreNum	core number

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_MACFATAL - Fatal error in the MAC - Call **EMAC_close()**

Pre Condition

EMAC_open function must be called before calling this API.

Post Condition

None

Modifies

Rx buffers and EMAC Rx head descriptor pointer and completion pointer registers

Example

```
static CSL_IntcContext context;
static CSL_IntcEventHandlerRecord Record[13];
static CSL_IntcObj intcEMACRx;
static CSL_IntcHandle hIntcEMACRx;

//CSL_IntcParam vectId1;
CSL_IntcParam vectId2;

CSL_IntcGlobalEnableState state;

/* Setup the global Interrupt */
context.numEvtEntries = 13;
context.eventhandlerRecord = Record;

/* VectorID for the Event */
vectId2 = CSL_INTC_VECTID_6;

CSL_intcInit(&context);
/* Enable NMIs */
CSL_intcGlobalNmiEnable();
/* Enable Global Interrupts */
CSL_intcGlobalEnable(&state);

/* Opening a handle for EMAC Rx interrupt */

hIntcEMACRx=CSL_intcOpen(&intcEMACRx,CSL_INTC_EVENTID_6,&vectId2,NULL);

/* Hook the ISRs */
CSL_intcHookIsr(vectId2,&HwRxInt);

CSL_intcHwControl(hIntcEMACRx, CSL_INTC_CMD_EVTENABLE, NULL);

/* This function is called when Rx interrupt occurs */
Void HwRxInt (void)
{
    /** Note : get the Emac Handle(hEMAC) by calling EMAC_open
function*/
    EMAC_RxServiceCheck(hEMAC, 0);
}
```

28.2.8 EMAC_sendPacket

Uint32 EMAC_sendPacket (Handle *hEMAC*, EMAC_Pkt * *pPkt*)

Description

Sends a Ethernet data packet out the EMAC device. On a non-error return, the EMAC device takes ownership of the packet. The packet is returned to the application's free pool once it has been transmitted.

The function returns zero on success, or an error code on failure. When an error code is returned, the EMAC device has not taken ownership of the packet.

Possible error codes include: EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_BADPACKET - The packet structure is invalid

Arguments

hEMAC	handle to the opened EMAC device
pPkt	EMAC packet structure

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_BADPACKET - The packet structure is invalid

Pre Condition

EMAC peripheral instance must be opened and get a packet buffer from private queue

Post Condition

Sends an ethernet data packet out the EMAC device and is returned to the application's free pool once it has been transmitted.

Modifies

None

Example

```
#define EMAC_RXFILTER_DIRECT      1
#define EMAC_PKT_FLAGS_SOP      0x80000000u
#define EMAC_PKT_FLAGS_EOP      0x40000000u

EMAC_Config ecfg;
EMAC_Pkt    *pPkt;
Handle      hEMAC = 0;
Uint32      size, TxCount = 0;

//open the EMAC device
EMAC_open( 1, (Handle)0x12345678, &ecfg, &hEMAC );

//set the receive filter
EMAC_setReceiveFilter( hEMAC, EMAC_RXFILTER_DIRECT );

//Fill the packet options fields
size = TxCount + 60;
pPkt->Flags      = EMAC_PKT_FLAGS_SOP | EMAC_PKT_FLAGS_EOP;
pPkt->ValidLen    = size;
pPkt->DataOffset = 0;
pPkt->PktChannel  = 0;
pPkt->PktLength   = size;
pPkt->PktFrag     = 1;

EMAC_sendPacket( hEMAC, pPkt );
```

28.2.9 EMAC_setMulticast

Uint32 EMAC_setMulticast (Handle hEMAC, Uint32 AddrCnt, Uint8 * pMCastList)

Description

This function is called to install a list of multicast addresses for use in multicast address filtering. Each time this function is called, any current multicast configuration is discarded in favor of the new list. Thus a set with a list size of zero will remove all multicast addresses from the device.

Note that the multicast list configuration is stateless in that the list of multicast addresses used to build the configuration is not retained. Thus it is impossible to examine a list of currently installed addresses.

The addresses to install are pointed to by pMCastList. The length of this list in bytes is 6 times the value of AddrCnt. When AddrCnt is zero, the pMCastList parameter can be NULL.

The function returns zero on success, or an error code on failure. The multicast list settings are not altered in the event of a failure code.

Possible error code include: EMAC_ERROR_INVALID - A calling parameter is invalid

Arguments

hEMAC	handle to the opened EMAC device
AddrCount	number of addresses to multicast
pMCastList	pointer to the multi cast list

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened and set multicast filter.

Post Condition

1. Install a list of multicast addresses for use in multicast address filtering.
2. A set with a list size of zero will remove all multicast addresses from the device.

Modifies

EMAC registers

Example

```
#define EMAC_RXFILTER_ALLMULTICAST 4

Handle      hEMAC = 0;
Uint32      AddrCnt;
Uint8       pMCastList;
EMAC_Config ecfg;

EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);

EMAC_setReceiveFilter( hEMAC, EMAC_RXFILTER_ALLMULTICAST );

EMAC_setMulticast( hEMAC, AddrCnt, &pMCastList );
```

28.2.10 EMAC_setReceiveFilter

Uint32 EMAC_setReceiveFilter (Handle hEMAC, Uint32 ReceiveFilter)

Description

Called to set the packet filter for received packets. The filtering level is inclusive, so BROADCAST would include both BROADCAST and DIRECTED (UNICAST) packets.

Available filtering modes include the following:

EMAC_RXFILTER_NOTHING - Receive nothing

EMAC_RXFILTER_DIRECT - Receive only Unicast to local MAC addr

EMAC_RXFILTER_BROADCAST	- Receive direct and Broadcast
EMAC_RXFILTER_MULTICAST	- Receive above plus multicast in mcast list
EMAC_RXFILTER_ALLMULTICAST	- Receive above plus all multicast
EMAC_RXFILTER_ALL	- Receive all packets

Note that if error frames and control frames are desired, reception of these must be specified in the device configuration.

The function returns zero on success, or an error code on failure.

Possible error code include: EMAC_ERROR_INVALID - A calling parameter is invalid

Arguments

hEMAC handle to the opened EMAC device

ReceiveFilter Filtering modes

Return Value

Success (0)

EMAC ERROR INVALID - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened before calling this API

Post Condition

Sets the packet filter for received packets

Modifies

EMAC registers

Example

```
#define EMAC_RXFILTER_DIRECT      1
EMAC_Config    ecfg;
Handle        hEMAC = 0;

EMAC_open(1, (Handle)0x12345678, &ecfg, &hEMAC);

EMAC setReceiveFilter(hEMAC, EMAC_RXFILTER_DIRECT);
```

28.2.11 EMAC timerTick

Uin32 EMAC timerTick (Handle *hEMAC*, *Uin8 coreNum*)

Description

This function should be called for each device in the system on a periodic basis of 100mS (10 times a second). It is used to check the status of the EMAC and MDIO device, and to potentially recover from low Rx buffer conditions.

Strict timing is not required, but the application should make a reasonable attempt to adhere to the 100mS mark. A missed call should not be "made up" by making multiple sequential calls. A "polling" driver (one that calls `EMAC_serviceCheck()` in a tight loop), must also adhere to the 100mS timing on this function.

Possible error codes include: `EMAC_ERROR_INVALID` - A calling parameter is invalid

Arguments

```
hEMAC          handle to the opened EMAC device
```

coreNum

core number

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

Pre Condition

EMAC peripheral instance must be opened

Post Condition

None

Modifies

Re-fill Rx buffer queue if needed and modifies EMAC CONTROL register.

Example

```

EMAC_Config ecfg;
Handle      hEMAC = 0;

//open the EMAC device

EMAC_open( 1, (Handle)0x12345678, &ecfg, &hEMAC );

EMAC_timerTick( hEMAC, 0 );

```

28.2.12 EMAC_txChannelTeardown

static inline void EMAC_txChannelTeardown (Uint32 val)
Description

Tear down selective transmit channel/channels

Arguments

val mask of selective tx channels to be torn down.

Return Value

None

Pre Condition

EMAC Tx channels should be in use

Post Condition

Tear down specific tx channels

Modifies

EMAC registers

Example

```

Uint32      val = EMAC_TEARDOWN_CHANNEL(0) | EMAC_TEARDOWN_CHANNEL(1);

EMAC_txChannelTeardown (val);

```

28.2.13 EMAC_rxChannelTeardown

static inline void EMAC_rxChannelTeardown (Uint32 val)

Description

Tear down selective receive channel/channels

Arguments

val mask of selective rx channels to be torn down.

Return Value

None

Pre Condition

EMAC Rx channels should be in use

Post Condition

Tear down specific rx channels

Modifies

EMAC registers

Example

```

Uint32              val = EMAC_TEARDOWN_CHANNEL(0) | EMAC_TEARDOWN_CHANNEL(1);

                    EMAC_rxChannelTeardown (val);

```

28.2.14 EMAC_TxServiceCheck

Uint32 EMAC_TxServiceCheck (Handle hEMAC, Uint8 coreNum)

Description

This function should be called every time there is an EMAC device Tx interrupt. It maintains the status the EMAC.

Note that the application has the responsibility for mapping the physical device index to the correct EMAC_serviceCheck() function. If more than one EMAC device is on the same interrupt, the function must be called for each device.

Possible error codes include: EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_MACFATAL - Fatal error in the MAC - Call **EMAC_close()**

Arguments

hEMAC handle to the opened EMAC device

coreNum core number

Return Value

Success (0)

EMAC_ERROR_INVALID - A calling parameter is invalid

EMAC_ERROR_MACFATAL - Fatal error in the MAC - Call **EMAC_close()**

Pre Condition

EMAC_open function must be called before calling this API.

Post Condition

None

Modifies

Tx buffers and EMAC Tx head descriptor pointer and completion pointer registers

Example

```
static CSL_IntcContext context;
static CSL_IntcEventHandlerRecord Record[13];
static CSL_IntcObj intcEMACTx;
static CSL_IntcHandle hIntcEMACTx;

    //CSL_IntcParam vectId1;
    CSL_IntcParam vectId2;

    CSL_IntcGlobalEnableState state;

    /* Setup the global Interrupt */
    context.numEvtEntries = 13;
    context.eventhandlerRecord = Record;

    /* VectorID for the Event */
    vectId2 = CSL_INTC_VECTID_6;

    CSL_intcInit(&context);
    /* Enable NMIs */
    CSL_intcGlobalNmiEnable();
    /* Enable Global Interrupts */
    CSL_intcGlobalEnable(&state);

    /* Opening a handle for EMAC Tx interrupt */

hIntcEMACTx=CSL_intcOpen(&intcEMACTx,CSL_INTC_EVENTID_6,&vectId2,NULL);

    /* Hook the ISR */
    CSL_intcHookIsr(vectId2,&HwTxInt);

    CSL_intcHwControl(hIntcEMACTx, CSL_INTC_CMD_EVTENABLE, NULL);

/* This function is called when Tx interrupt occurs */
Void HwTxInt (void)
{
    /** Note : get the Emac Handle(hEMAC) by calling EMAC_open
function*/
    EMAC_TxServiceCheck(hEMAC, 0);
}
```

28.3 Data Structures

28.3.1 EMAC_ChannelInfo

Detailed Description

Transmit/Receive Channel info Structure.

(one receive and up to 8 transmit per core in this example)

Field Documentation

Uint32 EMAC_ChannelInfo::RxChannels

Number of Rx Channels to use (1-8) if RxChannels = 0 does not set the Rx channels for the core if RxChannel = 1, 2 ... 8 uses specific channel for the core. User has to take care of allocating a portion from total allocation for the cores

Uint32 EMAC_ChannelInfo::TxChannels

Number of Tx Channels to use (1-8) if TxChannels = 0 does not allocate the Tx channels for the core if TxChannels = 1, 2, ... 8 allocates that many number of TxChannels for the core. User has to take care of allocating a portion from total allocation for the cores

28.3.2 EMAC_Config

Detailed Description

EMAC_Config.

The config structure defines how the EMAC device should operate. It is passed to the device when the device is opened, and remains in effect until the device is closed.

The following is a short description of the configuration fields:

UseMdio - Uses MDIO configuration if required. In case of SGMII MAC to MAC communication MDIO is not required. If this field is one (1) configures MDIO zero (0) does not configure MDIO

ModeFlags - Specify the Fixed Operating Mode of the Device:

EMAC_CONFIG_MODEFLG_CHPRIORITY - Treat TX channels as Priority Levels (Channel 7 is highest, 0 is lowest)

EMAC_CONFIG_MODEFLG_MACLOOPBACK - Set MAC in Internal Loopback for Testing

EMAC_CONFIG_MODEFLG_RXCRC - Include the 4 byte EtherCRC in RX frames

EMAC_CONFIG_MODEFLG_TXCRC - Assume TX Frames Include 4 byte EtherCRC

EMAC_CONFIG_MODEFLG_PASSERROR - Receive Error Frames for Testing

EMAC_CONFIG_MODEFLG_PASSCONTROL - Receive Control Frames for Testing

MdioModeFlags - Specify the MDIO/PHY Operation (See csl_MDIO.H)

TxChannels - Number of TX Channels to use (1-8, usually 1)

MacAddr - Device MAC address

RxMaxPktPool - Max Rx packet buffers to get from pool (Must be in the range of 8 to 192)

A list of callback functions is used to register callback functions with a particular instance of the EMAC peripheral. Callback functions are used by EMAC to communicate with the application.

These functions are REQUIRED for operation. The same callback table can be used for multiple driver instances.

The callback functions can be used by EMAC during any EMAC function, but mostly occur during calls to EMAC_statusIsr() and EMAC_statusPoll().

pfcbGetPacket - Called by EMAC to get a free packet buffer from the application layer for receive data. This function should return NULL is no free packets are available. The size of the packet buffer must be large enough to accommodate a full sized packet (1514 or 1518 depending

on the EMAC_CONFIG_MODEFLG_RXCRC flag), plus any application buffer padding (DataOffset).

pfcbFreePacket - Called by EMAC to give a free packet buffer back to the application layer. This function is used to return transmit packets. Note that at the time of the call, structure fields other than pDataBuffer and BufferLen are in an undefined state.

pfcbRxPacket - Called to give a received data packet to the application layer. The application must accept the packet. When the application is finished with the packet, it can return it to its own free queue. This function also returns a pointer to a free packet to replace the received packet on the EMAC free list. It returns NULL when no free packets are available. The return packet is the same as would be returned by pfcbGetPacket. Thus if a newly received packet is not desired, it can simply be returned to EMAC via the return value.

pfcbStatus - Called to indicate to the application that it should call **EMAC_getStatus()** to read the current device status. This call is made when device status changes.

pfcbStatistics - Called to indicate to the application that it should call **EMAC_getStatistics()** to read the current Ethernet statistics. Called when the statistic counters are to the point of overflow. The hApplication calling argument is the application's handle as supplied to the EMAC device in the **EMAC_open()** function.

Field Documentation

EMAC_ChannelInfo EMAC_Config::ChannelInfo[3]

Tx Channels information to use

UInt8 EMAC_Config::MacAddr[3][6]

Mac Address

UInt32 EMAC_Config::MdioModeFlags

CSL_MDIO Mode Flags (see CSL_MDIO.H)

UInt32 EMAC_Config::ModeFlags

Configuration Mode Flags

void(* EMAC_Config::pfcbFreePacket)(Handle hApplication, EMAC_Pkt *pPacket)

Free packet call back

EMAC_Pkt*(*) EMAC_Config::pfcbGetPacket)(Handle hApplication)

Get packet call back

EMAC_Pkt*(*) EMAC_Config::pfcbRxPacket)(Handle hApplication, EMAC_Pkt *pPacket)

Receive packet call back

void(* EMAC_Config::pfcbStatistics)(Handle hApplication)

Get statistics call back

void(* EMAC_Config::pfcbStatus)(Handle hApplication)

Get status call back

UInt32 EMAC_Config::RxMaxPktPool

Max Rx packet buffers to get from pool

UInt8 EMAC_Config::UseMdio

MDIO Configuration select. User has to pass one (1) if MDIO Configuration is needed, if not should pass zero (0)

28.3.3 EMAC_DescCh

Detailed Description

Transmit/Receive Descriptor Channel Structure.
(One receive and up to 8 transmit in this example)

Field Documentation

Uint32 EMAC_DescCh::ChannelIndex
Channel index 0-7

Uint32 EMAC_DescCh::DescCount
Current number of desc

Uint32 EMAC_DescCh::DescMax
Max number of desc (buffs)

PKTQ EMAC_DescCh::DescQueue
Packets queued as desc

struct _EMAC_Device* EMAC_DescCh::pd
Pointer to parent structure

EMAC_Desc* EMAC_DescCh::pDescFirst
First desc location

EMAC_Desc* EMAC_DescCh::pDescLast
Last desc location

EMAC_Desc* EMAC_DescCh::pDescRead
Location to read next desc

EMAC_Desc* EMAC_DescCh::pDescWrite
Location to write next desc

PKTQ EMAC_DescCh::WaitQueue
Packets waiting for TX desc

28.3.4 EMAC_Device

Detailed Description

EMAC Main Device Instance Structure.

Field Documentation

EMAC_Config EMAC_Device::Config
Original User Configuration

Uint32 EMAC_Device::DevMagic
Magic ID for this instance

Uint32 EMAC_Device::FatalError
Fatal Error Code

Handle EMAC_Device::hApplication

Calling Application's Handle

Handle EMAC_Device::hMDIO

Handle to MDIO Module

Uint32 EMAC_Device::MacHash1

Hash value cache

Uint32 EMAC_Device::MacHash2

Hash value cache

Uint32 EMAC_Device::PktMTU

Max physical packet size

EMAC_DescCh EMAC_Device::RxCh[8]

Receive channel status

Uint32 EMAC_Device::RxFilter

Current RX filter value

EMAC_Statistics EMAC_Device::Stats

Current running statistics

EMAC_DescCh EMAC_Device::TxCh[8]

Transmit channel status

28.3.5 EMAC_Pkt

Detailed Description

EMAC_Pkt.

The packet structure defines the basic unit of memory used to hold data packets for the EMAC device.

A packet is comprised of one or more packet buffers. Each packet buffer contains a packet buffer header, and a pointer to the buffer data. The EMAC_Pkt structure defines the packet buffer header.

The pDataBuffer field points to the packet data. This is set when the buffer is allocated, and is not altered.

BufferLen holds the the total length of the data buffer that is used to store the packet (or packet fragment). This size is set by the entity that originally allocates the buffer, and is not altered.

The Flags field contains additional information about the packet

ValidLen holds the length of the valid data currently contained in the data buffer.

DataOffset is the byte offset from the start of the data buffer to the first byte of valid data. Thus $(ValidLen + DataOffset) \leq BufferLen$.

Note that for receive buffer packets, the DataOffset field may be assigned before there is any valid data in the packet buffer. This allows the application to reserve space at the top of data buffer for private use. In all instances, the DataOffset field must be valid for all packets handled by EMAC.

The data portion of the packet buffer represents a packet or a fragment of a larger packet. This is determined by the Flags parameter. At the start of every packet, the SOP bit is set in Flags. If the EOP bit is also set, then the packet is not fragmented. Otherwise; the next packet structure pointed to by the pNext field will contain the next fragment in the packet. On either type of buffer,

when the SOP bit is set in Flags, then the PktChannel, PktLength, and PktFrag fields must also be valid. These fields contain additional information about the packet.

The PktChannel field determines what channel the packet has arrived on, or what channel it should be transmitted on. The EMAC library supports only a single receive channel, but allows for up to eight transmit channels. Transmit channels can be treated as round-robin or priority queues. The PktLength field holds the size of the entire packet. On single frag packets (both SOP and EOP set in BufFlags), PktLength and ValidLen will be equal.

The PktFrag field holds the number of fragments (EMAC_Pkt records) used to describe the packet. If more than 1 frag is present, the first record must have EMAC_PKT_FLAGS_SOP flag set, with corresponding fields validated. Each frag/record must be linked list using the pNext field, and the final frag/record must have EMAC_PKT_FLAGS_EOP flag set and pNext=0.

In systems where the packet resides in cacheable memory, the data buffer must start on a cache line boundary and be an even multiple of cache lines in size. The EMAC_Pkt header must not appear in the same cache line as the data portion of the packet. On multi-fragment packets, some packet fragments may reside in cacheable memory where others do not.

NOTE: It is up to the caller to assure that all packet buffers residing in cacheable memory are not currently stored in L1 or L2 cache when passed to any EMAC function.

Some of the packet Flags can only be set if the device is in the proper configuration to receive the corresponding frames. In order to enable these flags, the following modes must be set: Rx_crc Flag : RXCRC Mode in EMAC_Config RxErr Flags : PASSERROR Mode in EMAC_Config RxCtrl Flags : PASSCONTROL Mode in EMAC_Config RxPrm Flag : EMAC_RXFILTER_ALL in **EMAC_setReceiveFilter()**

Field Documentation

Uint32 EMAC_Pkt::AppPrivate

For use by the application

Uint32 EMAC_Pkt::BufferLen

Physical Length of buffer (read only)

Uint32 EMAC_Pkt::DataOffset

Byte offset to valid data

Uint32 EMAC_Pkt::Flags

Packet Flags

Uint8* EMAC_Pkt::pDataBuffer

Pointer to Data Buffer (read only)

Uint32 EMAC_Pkt::PktChannel

Tx/Rx Channel/Priority 0-7 (SOP only)

Uint32 EMAC_Pkt::PktFrag

No of frags in packet (SOP only) frag is EMAC_Pkt record-normally 1

Uint32 EMAC_Pkt::PktLength

Length of Packet (SOP only) (same as ValidLen on single frag Pkt)

struct _EMAC_Pkt* EMAC_Pkt::pNext

Next record

struct _EMAC_Pkt* EMAC_Pkt::pPrev
Previous record

UInt32 EMAC_Pkt::ValidLen
Length of valid data in buffer

28.3.6 EMAC_Statistics

Detailed Description
EMAC_Statistics.

The statistics structure is the used to retrieve the current count of various packet events in the system. These values represent the delta values from the last time the statistics were read.

Field Documentation

UInt32 EMAC_Statistics::Frame1024tUp
Total Tx&Rx with Octet Size of >=1024

UInt32 EMAC_Statistics::Frame128t255
Total Tx&Rx with Octet Size of 128 to 255

UInt32 EMAC_Statistics::Frame256t511
Total Tx&Rx with Octet Size of 256 to 511

UInt32 EMAC_Statistics::Frame512t1023
Total Tx&Rx with Octet Size of 512 to 1023

UInt32 EMAC_Statistics::Frame64
Total Tx&Rx with Octet Size of 64

UInt32 EMAC_Statistics::Frame65t127
Total Tx&Rx with Octet Size of 65 to 127

UInt32 EMAC_Statistics::NetOctets
Sum of all Octets Tx or Rx on the Network

UInt32 EMAC_Statistics::RxAlignCodeErrors
Frames Received with Alignment/Code Errors

UInt32 EMAC_Statistics::RxBCastFrames
Good Broadcast Frames Received

UInt32 EMAC_Statistics::RxCRCErrors
Frames Received with CRC Errors

UInt32 EMAC_Statistics::RxDMAOverruns
Total Rx DMA Overruns

UInt32 EMAC_Statistics::RxFiltered
Rx Frames Filtered Based on Address

UInt32 EMAC_Statistics::RxFragments
Rx Frame Fragments Received

Uint32 EMAC_Statistics::RxGoodFrames

Good Frames Received

Uint32 EMAC_Statistics::RxJabber

Jabber Frames Received

Uint32 EMAC_Statistics::RxMCastFrames

Good Multicast Frames Received

Uint32 EMAC_Statistics::RxMOFOverruns

Total Rx Middle of Frame Overruns

Uint32 EMAC_Statistics::RxOctets

Total Received Bytes in Good Frames

Uint32 EMAC_Statistics::RxOversized

Oversized Frames Received

Uint32 EMAC_Statistics::RxPauseFrames

PauseRx Frames Received

Uint32 EMAC_Statistics::RxQOSFiltered

Rx Frames Filtered Based on QoS Filtering

Uint32 EMAC_Statistics::RxSOFOverruns

Total Rx Start of Frame Overruns

Uint32 EMAC_Statistics::RxUndersized

Undersized Frames Received

Uint32 EMAC_Statistics::TxBCastFrames

Good Broadcast Frames Sent

Uint32 EMAC_Statistics::TxCarrierSLoss

Tx Frames Lost Due to Carrier Sense Loss

Uint32 EMAC_Statistics::TxCollision

Total Frames Sent With Collision

Uint32 EMAC_Statistics::TxDeferred

Frames Where Transmission was Deferred

Uint32 EMAC_Statistics::TxExcessiveColl

Tx Frames Lost Due to Excessive Collisions

Uint32 EMAC_Statistics::TxGoodFrames

Good Frames Sent

Uint32 EMAC_Statistics::TxLateColl

Tx Frames Lost Due to a Late Collision

Uint32 EMAC_Statistics::TxMCastFrames

Good Multicast Frames Sent

Uint32 EMAC_Statistics::TxMultiColl

Frames Sent with Multiple Colisions

Uint32 EMAC_Statistics::TxOctets

Total Transmitted Bytes in Good Frames

Uint32 EMAC_Statistics::TxPauseFrames

PauseTx Frames Sent

Uint32 EMAC_Statistics::TxSingleColl

Frames Sent with Exactly One Collision

Uint32 EMAC_Statistics::TxUnderrun

Tx Frames Lost with Tx Underrun Error

28.3.7 EMAC_Status

Detailed Description

EMAC_Status.

The status structure contains information about the MAC's run-time status.

The following is a short description of the configuration fields:

MdioLinkStatus - Current link stat (non-zero on link; see CSL_MDIO.H)

PhyDev - Current PHY device in use (0-31)

RxPktHeld - Current number of Rx packets held by the EMAC device

TxPktHeld - Current number of Tx packets held by the EMAC device

FatalError - Fatal Error Code (TBD)

Field Documentation

Uint32 EMAC_Status::FatalError

Fatal Error when non-zero

Uint32 EMAC_Status::MdioLinkStatus

CSL_MDIO Link status (see csl_mdio.h)

Uint32 EMAC_Status::PhyDev

Current PHY device in use (0-31)

Uint32 EMAC_Status::RxPktHeld

Number of packets held for Rx

Uint32 EMAC_Status::TxPktHeld

Number of packets held for Tx

28.4 Macros

#define EMAC_DEVMAGIC 0x0aceface

Device Magic number

#define EMAC_NUMSTATS 36

Number of statistics regs

#define EMAC_PKT_FLAGS_EOP 0x40000000u

End of packet

#define EMAC_PKT_FLAGS_SOP 0x80000000u

Start of packet

#define EMAC_PKT_FLAGS_ALIGNERROR 0x00040000u

RxErr: Alignment Error

#define EMAC_PKT_FLAGS_CODEERROR 0x00080000u

RxErr: Code Error

#define EMAC_PKT_FLAGS_CONTROL 0x00200000u

RxCtl: Control Frame

#define EMAC_PKT_FLAGS_CRCERROR 0x00020000u

RxErr: Bad CRC

#define EMAC_PKT_FLAGS_FRAGMENT 0x00800000u

RxErr: Fragment

#define EMAC_PKT_FLAGS_HASCRC 0x04000000u

RxCrc: PKT has 4byte CRC

#define EMAC_PKT_FLAGS_JABBER 0x02000000u

RxErr: Jabber

#define EMAC_PKT_FLAGS_NOMATCH 0x00010000u

RxPrm: No Match

#define EMAC_PKT_FLAGS_OVERRUN 0x00100000u

RxErr: Overrun

#define EMAC_PKT_FLAGS_OVERSIZE 0x01000000u

RxErr: Oversize

#define EMAC_PKT_FLAGS_UNDERSIZED 0x00400000u

RxErr: Undersized

#define EMAC_CONFIG_MODEFLG_CHPRIORITY 0x0001

Use Tx channel priority

#define EMAC_CONFIG_MODEFLG_MACLOOPBACK 0x0002

MAC internal loopback

#define EMAC_CONFIG_MODEFLG_PASSCONTROL 0x0020
Pass control frames

#define EMAC_CONFIG_MODEFLG_PASSError 0x0010
Pass error frames

#define EMAC_CONFIG_MODEFLG_RXCRC 0x0004
Include CRC in RX frames

#define EMAC_CONFIG_MODEFLG_TXCRC 0x0008
Tx frames include CRC

#define EMAC_CONFIG_MODEFLG_PASSALL 0x00040
Pass all frames

#define EMAC_CONFIG_MODEFLG_RXQOS 0x00080
Enable QOS at receive side

#define EMAC_CONFIG_MODEFLG_RXNOCHAIN 0x00100
Select no buffer chaining

#define EMAC_CONFIG_MODEFLG_RXOFFLENBLOCK 0x00200
Enable offset/length blocking

#define EMAC_CONFIG_MODEFLG_RXOWNERSHIP 0x00400
Use ownership bit as 1

#define EMAC_CONFIG_MODEFLG_RXFIFOLOWCNTL 0x00800
Enable rx fifo flow control

#define EMAC_CONFIG_MODEFLG_CMDIDLE 0x01000
Enable IDLE command

#define EMAC_CONFIG_MODEFLG_TXSHORTGAPEN 0x02000
Enable tx short gap

#define EMAC_CONFIG_MODEFLG_TXPACE 0x04000
Enable tx pacing

#define EMAC_CONFIG_MODEFLG_TXFLOWCNTL 0x08000
Enable tx flow control

#define EMAC_CONFIG_MODEFLG_RXBUFFERFLOWCNTL 0x10000
Enable rx buffer flow control

#define EMAC_CONFIG_MODEFLG_FULLDUPLEX 0x20000
Set full duplex mode

#define EMAC_CONFIG_MODEFLG_GIGABIT 0x40000
Set gigabit

#define EMAC_TEARDOWN_CHANNEL(x) (1 << x)
Macro to tear down selective Rx/Tx channels

#define EMAC_RXFILTER_ALL 5

Receive filter set to All

#define EMAC_RXFILTER_ALLMULTICAST 4

Receive filter set to All Mcast

#define EMAC_RXFILTER_BROADCAST 2

Receive filter set to Broadcast

#define EMAC_RXFILTER_DIRECT 1

Receive filter set to Direct

#define EMAC_RXFILTER_MULTICAST 3

Receive filter set to Multicast

#define EMAC_RXFILTER_NOTHING 0

Receive filter set to Nothing

#define EMAC_ERROR_ALREADY 1

Operation has already been started

#define EMAC_ERROR_BADPACKET 5

Supplied packet was invalid

#define EMAC_ERROR_DEVICE 3

Device hardware error

#define EMAC_ERROR_INVALID 4

Function or calling parameter is invalid

#define EMAC_ERROR_MACFATAL 6

Fatal Error - EMAC_close() required

#define EMAC_ERROR_NOTREADY 2

Device is not open or not ready

#define EMAC_REGS ((CSL_EmacRegs *)CSL_EMAC_0_REGS)

EMAC Module registers

#define ECTL_REGS ((CSL_EctlRegs *)CSL_ECTL_0_REGS)

EMAC Control Module registers

28.5 Typedefs

typedef struct [_EMAC_Config](#) [EMAC_Config](#)

The config structure defines how the EMAC device should operate. It is passed to the device when the device is opened, and remains in effect until the device is closed.

typedef struct [_EMAC_DescCh](#) [EMAC_DescCh](#)

Transmit/Receive Descriptor Channel Structure.
(One receive and up to 8 transmit in this example)

typedef struct [_EMAC_Pkt](#) [EMAC_Pkt](#)

The packet structure defines the basic unit of memory used to hold data packets for the EMAC device.

typedef struct [_EMAC_Statistics](#) [EMAC_Statistics](#)

The statistics structure is the used to retrieve the current count of various packet events in the system. These values represent the delta values from the last time the statistics were read.

typedef struct [_EMAC_Status](#) [EMAC_Status](#)

The status structure contains information about the MAC's run-time status.

Chapter 29 MDIO Module

Topics

<u>29.1 Overview</u>
<u>29.2 Functions</u>
<u>29.3 Data Structures</u>
<u>29.4 Macros</u>

29.1 Overview

The Management Data Input/Output (MDIO) module implements the 802.3 serial management interface to interrogate and controls up to 32 Ethernet PHY(s) connected to the device, using a shared two-wire bus. Application software uses the MDIO module to configure the auto-negotiation parameters of each PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC module for correct operation. The module is designed to allow almost transparent operation of the MDIO interface, with very little maintenance from the core processor.

29.2 Functions

29.2.1 MDIO_close

void MDIO_close (Handle *hMDIO*)

Description

Close the MDIO peripheral and disable further operation.

Arguments

<i>hMDIO</i>	handle to the opened MDIO instance
--------------	------------------------------------

Return Value

None

Pre Condition

MDIO module must be reset and opened before calling this function.

Post Condition

MDIO module is closed. No further operations are possible.

Modifies

PHY and MDIO registers

Example

```
#define MDIO_MODEFLG_FD1000      0x0020
#define MDIO_MODEFLG_EXTLOOPBACK 0x0100

Uint32    mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
Handle    hMDIO;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_close( hMDIO );
```

29.2.2 MDIO_getStatus

void MDIO_getStatus (Handle *hMDIO*, Uint32 * *pPhy*, Uint32 * *pLinkStatus*)

Description

Called to get the status of the MDIO/PHY

Arguments

<i>hMDIO</i>	handle to the opened MDIO instance
<i>pPhy</i>	pointer to the physical address
<i>pLinkStatus</i>	Status pointer to the link status

Return Value

None

Pre Condition

MDIO module must be reset and opened before calling this API

Post Condition

MDIO status is returned through the parameters passed in this API.

Modifies

Output parameters *pPhy* and *pLinkStatus* being passed

Example

```
#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100
Uint32    mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
Handle    hMDIO;
Uint32    pPhy, pLinkStatus;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_getStatus( hMDIO, &pPhy, &pLinkStatus );
```

29.2.3 MDIO_initPHY

Uint32 MDIO_initPHY (Handle *hMDIO*, volatile Uint32 *phyAddr*)

Description

Force a switch to the specified PHY, and start the negotiation process.

Arguments

<i>hMDIO</i>	handle to the opened MDIO instance
<i>phyAddr</i>	Physical address

Return Value

Returns 1 if the PHY selection completed OK,
else 0

Pre Condition

MDIO module must be reset.

Post Condition

Initializes the specific PHY device.

Modifies

PHY registers

Example

```
#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100

Uint32    mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
```

```

Handle          hMDIO;
volatile Uint32  phyAddr;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_initPHY( hMDIO, phyAddr );

```

29.2.4 MDIO_open

Handle MDIO_open (Uint32 *mdioModeFlags*)

Description

Opens the MDIO peripheral and start searching for a PHY device.
It is assumed that the MDIO module is reset prior to calling this function.

Arguments

<code>mdioModeFlags</code>	mode flags pof MII
----------------------------	--------------------

Return Value

Handle to the opened MDIO instance

Pre Condition

The MDIO module must be reset prior to calling this function.

Post Condition

Opens the MDIO peripheral and start searching for a PHY device.

Modifies

MDIO configuration registers

Example

```

#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100
Uint32 mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;

MDIO_open ( mdioModeFlags );

```

29.2.5 MDIO_phyRegRead

Uint32 MDIO_phyRegRead (volatile Uint32 *phyIdx*, volatile Uint32 *phyReg*, Uint16 * *pData*)

Description

Raw data read of a PHY register.

Arguments

<code>phyIdx</code>	Physical Index
<code>phyReg</code>	Physical register
<code>pData</code>	Data read

Return Value

Returns 1 if the PHY ACK'd the read,
else 0

Pre Condition

MDIO module must be reset and opened. PHY device must be initialized.

Post Condition

Raw data is read from a PHY register.

Modifies

Output parameter *pData* being passed.

Example

```
#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100

volatile Uint32 phyIdx = PHYREG_CONTROL;
volatile Uint32 phyReg;
Uint16 pData;

Uint32      mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
Handle      hMDIO;
volatile Uint32 phyAddr;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_initPHY( hMDIO, phyAddr );

MDIO_phyRegRead( phyIdx, ((MDIO_Device *)hMDIO)->phyReg, pData );
```

29.2.6 MDIO_phyRegWrite

Uint32 MDIO_phyRegWrite (volatile Uint32 *phyIdx*, volatile Uint32 *phyReg*, Uint16 *data*)

Description

Raw data write of a PHY register.

Arguments

<i>phyIdx</i>	Physical Index
<i>phyReg</i>	Physical register
<i>pData</i>	Data written

Return Value

Returns 1 if the PHY ACK'd the write,
else 0

Pre Condition

MDIO module must be reset and opened. PHY device must be initialized.

Post Condition

Raw data is written to a PHY register.

Modifies

None

Example

```
#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100
#define PHYREG_SHADOW_EXTLOOPBACK    0x8400

volatile Uint32 phyIdx = PHYREG_CONTROL;
volatile Uint32 phyReg;
Uint16 pData = PHYREG_SHADOW_EXTLOOPBACK;

Uint32    mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
Handle     hMDIO;
volatile Uint32  phyAddr;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_initPHY( hMDIO, phyAddr );

MDIO_phyRegWrite( phyIdx, ((MDIO_Device *)hMDIO)->phyReg, pData
);
```

29.2.7 MDIO_timerTick

Uint32 MDIO_timerTick (Handle *hMDIO*)

Description

Called to signify that approx 100mS have elapsed

Arguments

hMDIO	Handle to the opened MDIO instance
-------	------------------------------------

Return Value

MDIO event code (see MDIO Events).

Pre Condition

MDIO module must be reset and opened before calling this API.

Post Condition

Gets approximately 100mS delay

Modifies

PHY and MDIO registers

Example

```
#define MDIO_MODEFLG_FD1000          0x0020
#define MDIO_MODEFLG_EXTLOOPBACK    0x0100
```

```

UInt32      mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
        Handle  hMDIO;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );
MDIO_timerTick( hMDIO );

```

29.2.8 MDIO initContinue

```

uint32 MDIO initContinue (MDIO Device * pd) [static]

```

Description

Continues the initialization process started in **MDIO_initPHY()**

Arguments

Pd pointer to MDIO device object

Return Value

Returns 0 on an error,
1 on success

Pre Condition

MDIO_initPHY function must be called before calling this API

Post Condition

Continues and completes the initialization process started in **MDIO_initPHY()**

Modifies

PHY registers

Example

```
#define MDIO_MODEFLG_FD1000      0x0020
#define MDIO_MODEFLG_EXTLOOPBACK  0x0100

Uint32      mdioModeFlags = MDIO_MODEFLG_FD1000 |
MDIO_MODEFLG_LOOPBACK;
Handle      hMDIO;
volatile Uint32  phyAddr;

//Open the MDIO module
hMDIO = MDIO_open ( mdioModeFlags );

MDIO_initPHY( hMDIO, phyAddr );
MDIO_initContinue( (MDIO_Device *)hMDIO );
```

29.3 Data Structures

This section lists Data Structures available in the MDIO module.

29.3.1 MDIO_Device

MDIO_Device This is the MDIO object that contains the MDIO device object characteristics.

Field documentation

UInt32 _MDIO_Device::LinkStatus

Link State PHYREG_STATUS_LINKSTATUS

UInt32 _MDIO_Device::ModeFlags

User specified configuration flags

UInt32 _MDIO_Device::PendingStatus

Pending Link Status

UInt32 _MDIO_Device::phyAddr

Current (or next) PHY addr (0-31)

UInt32 _MDIO_Device::phyState

PHY State

UInt32 _MDIO_Device::phyStateTicks

Ticks elapsed in this PHY state

29.4 Macros

#define DEV_REGS ((CSL_DevRegs *) CSL_DEV_REGS)

Base address of device registers

#define MDIO_REGS ((CSL_MdioRegs *) CSL_MDIO_0_REGS)

Base address of MDIO registers

#define PHYREG_read(regadr, phyadr)

Value:

```
MDIO_REGS->USERACCESS0 = \
    CSL_FMK(MDIO_USERACCESS0_GO, 1u) | \
    CSL_FMK(MDIO_USERACCESS0_REGADR, regadr) | \
    CSL_FMK(MDIO_USERACCESS0_PHYADR, phyadr)
```

This macro reads from the PHY register through the USERACCESS register provided in MDIO module

#define PHYREG_wait() while(CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_GO))

Waits for GO bit to clear

#define PHYREG_waitResults(results)

Value:

```
{ \
    while( CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_GO) \
); \
    results = CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_DATA); }
```

Waits for GO bit to clear and reads data from the PHY register

#define PHYREG_waitResultsAck(results, ack)

Value:

```
{ \
    while( CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_GO) \
); \
    results = CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_DATA); \
    ack = CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_ACK); }
```

Waits for GO bit to clear, reads data from the PHY register and checks for ACK

#define PHYREG_write(regadr, phyadr, data)

Value:

```
MDIO_REGS->USERACCESS0 = \
    CSL_FMK(MDIO_USERACCESS0_GO, 1u) | \
    CSL_FMK(MDIO_USERACCESS0_WRITE, 1) | \
    CSL_FMK(MDIO_USERACCESS0_REGADR, regadr) | \
    CSL_FMK(MDIO_USERACCESS0_PHYADR, phyadr) | \
    CSL_FMK(MDIO_USERACCESS0_DATA, data)
```

This macro writes to the PHY register through the USERACCESS register provided in MDIO module

```
#define VBUSCLK 165  
Standard defines/assumptions for MDIO interface  
  
#define MDIO_MODEFLG_AUTONEG 0x0001  
Use Autonegotiate  
  
#define MDIO_MODEFLG_EXTLOOPBACK 0x0100  
Use external PHY Loopback, with plug  
  
#define MDIO_MODEFLG_FD10 0x0004  
Use 10Mb/s Full Duplex  
  
#define MDIO_MODEFLG_FD100 0x0010  
Use 100Mb/s Full Duplex  
  
#define MDIO_MODEFLG_FD1000 0x0020  
Use 1000Mb/s Full Duplex  
  
#define MDIO_MODEFLG_HD10 0x0002  
Use 10Mb/s Half Duplex  
  
#define MDIO_MODEFLG_HD100 0x0008  
Use 100Mb/s Half Duplex  
  
#define MDIO_MODEFLG_LOOPBACK 0x0040  
Use PHY Loopback  
  
#define MDIO_MODEFLG_NWAYACTIVE 0x0080  
NWAY currently active  
  
#define MDIO_LINKSTATUS_FD10 2  
Link Status: FD10  
  
#define MDIO_LINKSTATUS_FD100 4  
Link Status: FD100  
  
#define MDIO_LINKSTATUS_FD1000 5  
Link Status: FD1000  
  
#define MDIO_LINKSTATUS_HD10 1  
Link Status: HD10  
  
#define MDIO_LINKSTATUS_HD100 3  
Link Status: HD100  
  
#define MDIO_LINKSTATUS_NOLINK 0  
Link Status: No Link  
  
#define MDIO_EVENT_LINKDOWN 1  
Link down event  
  
#define MDIO_EVENT_LINKUP 2  
Link (or re-link) event
```

#define MDIO_EVENT_NOCHANGE 0

No change from previous status

#define MDIO_EVENT_PHYERROR 3

No PHY connected

#define PHYREG_CONTROL 0

Control register

#define PHYREG_CONTROL_AUTONEGEN (1<<12)

Auto Negate Enable bit

#define PHYREG_CONTROL_AUTORESTART (1<<9)

Set Auto restart bit

#define PHYREG_CONTROL_DUPLEXFULL (1<<8)

Set Full Duplex bit

#define PHYREG_CONTROL_ISOLATE (1<<10)

Set Isolate bit

#define PHYREG_CONTROL_LOOPBACK (1<<14)

Set Loop back bit

#define PHYREG_CONTROL_POWERDOWN (1<<11)

Set Power Down bit

#define PHYREG_CONTROL_RESET (1<<15)

Set Reset bit

#define PHYREG_CONTROL_SPEEDLSB (1<<13)

Set Speed LSB bit

#define PHYREG_CONTROL_SPEEDMSB (1<<6)

Set Speed MSB bit

#define PHYREG_STATUS 1

Status register

#define PHYREG_STATUS_AUTOCAPABLE (1<<3)

Set Auto Capable bit

#define PHYREG_STATUS_AUTOCOMPLETE (1<<5)

Set Auto complete bit

#define PHYREG_STATUS_EXTENDED (1<<0)

Set Extended bit

#define PHYREG_STATUS_EXTSTATUS (1<<8)

Set External Status bit

#define PHYREG_STATUS_FD10 (1<<12)

Set FD10 bit

```
#define PHYREG_STATUS_FD100 (1<<14)
Set FD100 bit

#define PHYREG_STATUS_HD10 (1<<11)
Set HD10bit

#define PHYREG_STATUS_HD100 (1<<13)
Set HD100 bit

#define PHYREG_STATUS_JABBER (1<<1)
Set Jabber bit

#define PHYREG_STATUS_LINKSTATUS (1<<2)
Set Link status bit

#define PHYREG_STATUS_NOPREAMBLE (1<<6)
Set No preamble bit

#define PHYREG_STATUS_REMOTEFAULT (1<<4)
Set Reomte default bit

#define PHYREG_ID1 2
Physical ID 1 register

#define PHYREG_ID2 3
Physical ID 1 register

#define PHYREG_ADVERTISE 4
Physical Advertise reg

#define PHYREG_ADVERTISE_FAULT (1<<13)
Set Fault bit

#define PHYREG_ADVERTISE_FD10 (1<<6)
Set FD10 bit

#define PHYREG_ADVERTISE_FD100 (1<<8)
Set FD100 bit

#define PHYREG_ADVERTISE_HD10 (1<<5)
Set HD10 bit

#define PHYREG_ADVERTISE_HD100 (1<<7)
Set HD100 bit

#define PHYREG_ADVERTISE_MSG (1)
Set Message bit

#define PHYREG_ADVERTISE_MSGMASK (0x1F)
Set Message mask bit

#define PHYREG_ADVERTISE_NEXTPAGE (1<<15)
Set next page bit
```

#define PHYREG_ADVERTISE_PAUSE (1<<10)

Set Pause bit

#define PHYREG_1000CONTROL 9

Physical 1000 Ctrl reg

#define PHYREG_1000STATUS 0xA

Phy 1000 Status reg

#define PHYREG_ADVERTISE_FD1000 (1<<9)

Advertise FD1000 bit

#define PHYREG_EXTSTATUS 0x0F

Physical Ext status reg

#define PHYREG_EXTSTATUS_FD1000 (1<<13)

Ext Status FD1000 bit

#define PHYREG_PARTNER 5

Physical Partner reg

#define PHYREG_PARTNER_ACK (1<<14)

Set Acknowledge bit

#define PHYREG_PARTNER_FAULT (1<<13)

Set Fault bit

#define PHYREG_PARTNER_FD10 (1<<6)

Set FD10 bit

#define PHYREG_PARTNER_FD100 (1<<8)

Set FD100 bit

#define PHYREG_PARTNER_FD1000 (1<<11)

Partner FD1000 bit

#define PHYREG_PARTNER_HD10 (1<<5)

Set HD10 bit

#define PHYREG_PARTNER_HD100 (1<<7)

Set HD100 bit

#define PHYREG_PARTNER_MSGMASK (0x1F)

Set Message mask bit

#define PHYREG_PARTNER_NEXTPAGE (1<<15)

Set next page bit

#define PHYREG_PARTNER_PAUSE (1<<10)

Set Pause bit

#define PHYREG_ACCESS 0x1C

Physical Access

#define PHYREG_ACCESS_COPPER 0xFC00

Access Copper

#define PHYREG_SHADOW 0x18

Physical shadow reg

#define PHYREG_SHADOW_EXTLOOPBACK 0x8400

Shadow Ext Loopback bit

#define PHYREG_SHADOW_INBAND 0xF1C7

Shadow In band bit

#define PHYREG_SHADOW_RGMII MODE 0xF080

Shadow RGMII mode bit

#define PHYSTATE_LINKED 5

MDIO Linked

#define PHYSTATE_LINKWAIT 4

MDIO Wait for link

#define PHYSTATE_MDIOINIT 0

MDIO Initialization state

#define PHYSTATE_NWAYSTART 2

MDIO N Way start

#define PHYSTATE_NWAYWAIT 3

MDIO N Way wait

#define PHYSTATE_RESET 1

MDIO Reset State

Chapter 30 SEM Module

Topics

<u>30.1 Overview</u>
<u>30.2 Functions</u>
<u>30.3 Data Structures</u>
<u>30.4 Enumerations</u>

30.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within SEM module.

TCI6488 will include a semaphore hardware block to allow atomic access to shared resources on the device. The semaphore block will have unique interrupts to each of the cores to identify when that core has acquired the resource.

30.2 Functions

30.2.1 CSL_semInit

CSL_Status CSL_semInit ([CSL_SemContext](#) * pContext)

Description

This is the initialization function for the SEM CSL. The function must be called before calling any other API from this CSL. This function is idem-potent. Currently, the function just return status CSL_SOK, without doing anything.

Arguments

pContext	Pointer to module-context. As SEM doesn't have any context based information user is expected to pass NULL.
----------	---

Return Value

CSL_Status
CSL_SOK - Always returns

Pre Condition

This function should be called before using any of the CSL APIs in the SEM module.

Post Condition

The CSL for SEM is initialized

Modifies

None

Example

```
CSL_SemContext SemContext;
if (CSL_SOK != CSL_semInit(&SemContext))
{
    // Init CSL for sem module

return;
}
```

30.2.2 CSL_semOpen

CSL_SemHandle CSL_semOpen (CSL_SemObj * pSemObj, CSL_InstNum *instNum*, CSL_SemParam *pSemParam, CSL_Status *pStatus)

Description

The open call sets up the data structures for the particular instance of SEM device. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

pSemObj	Pointer to the object that holds reference to the instance of SEM requested after the call
instNum	Instance of SEM to which a handle is requested

SemParam	pointer to module specific parameters (Number of SEM resources)
pStatus	pointer for returning status of the function call

Return Value

CSL_SemHandle Valid SEM instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition

SEM must be successfully initialized via **CSL_semInit()** before calling this function. Memory for the **CSL_SemObj** must be allocated outside this call. This object must be retained while using this peripheral.

Post Condition

1. The status is returned in the status variable. If status returned is CSL_SOK Valid SEM handle is returned
CSL_ESYS_FAIL The SEM instance is invalid
CSL_ESYS_INVPARAMS Invalid Parameters
2. SEM object structure is populated

Modifies

1. The status variable
2. SEM object structure

Example

```
// handle for SEM
CSL_SemHandle handleSem;
// SEM object
CSL_SemObj      *pSemObj;
// CSL status
CSL_Status status;
CSL_SemParam    pSemParam;
//Number of SEM resources
pSemParam.flags = 2

CSL_semInit(NULL); // Init CSL for SEM module, this step is not
required

// Open handle for SEM module
handleSem = CSL_semOpen (pSemObj, CSL_SEM, &pSemParam, &status);

if ((handleSem == NULL) || (status != CSL_SOK))
{
    printf ("\nError opening CSL_SEM");
    exit(1);
}
```

30.2.3 CSL_semClose

CSL_Status CSL_semClose ([CSL_SemHandle](#) hSem)

Description

The Close call releases the channel of the peripheral.

Arguments

hSem	Handle to the SEM instance
------	----------------------------

Return Value

CSL_Status

CSL_SOK - Close successful

CSL_ESYS_BADHANDLE - Invalid handle

Example

```
// handle for SEM
CSL_SemHandle handleSem;
// SEM object
CSL_SemObj *pSemObj;
// CSL status
CSL_Status status;
CSL_SemParam pSemParam;
//Number of SEM resources
pSemParam.flags = 2

CSL_semInit(NULL); // Init CSL for SEM module, this step is not
required

// Open handle for SEM module
handleSem = CSL_semOpen (pSemObj, CSL_SEM, &pSemParam, &status);

...
...
...

CSL_semClose(handleSem);
```

30.2.4 CSL_semGetHwStatus

CSL_Status CSL_semGetHwStatus ([CSL_SemHandle](#) hSem, CSL_SemHwStatusQuery query, void * response)

Description

The Status Query Command call queries the semaphore module

Arguments

hSem	Handle to the SEM instance
query	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value

CSL_Status

CSL_SOK - Successful completion of the query

CSL_ESYS_BADHANDLE - Invalid Handle

CSL_ESYS_INVQUERY - Invalid query

Pre Condition

None

Post Condition

Data requested by query is returned through the variable "response"

Modifies

The input argument "response" is modified.

Example

```
// handle for SEM
CSL_SemHandle handleSem;
// SEM object
CSL_SemObj      *pSemObj;
// CSL status
CSL_Status status;
void semVal;
CSL_SemParam      pSemParam;
//Number of SEM resources
pSemParam.flags = 2

CSL_semInit(NULL); // Init CSL for SEM module, this step is not
required
...
// Open handle for SEM module
handleSem = CSL_semOpen (pSemObj, CSL_SEM, &pSemParam, &status);
..
status = CSL_semGetHwStatus(hSem, CSL_SEM_QUERY_READ_PEND,
&semVal);
```

30.2.5 CSL_semHwControl

CSL_Status CSL_semHwControlCmd ([CSL_SemHandle](#) *hSem*, **CSL_SemHwControlCmd** *ctrlCmd*, void * *arg*)

Description

The Control Command call writes a command value to the semaphore

Arguments

<i>hSem</i>	Handle to the SEM instance
<i>ctrlCmd</i>	The command to this API indicates the action to be taken on SEM.
<i>arg</i>	An optional argument.

Return Value

CSL_Status

CSL_SOK - Status info return successful.

Pre Condition

Both **CSL_semInit()** (optional) and **CSL_semOpen()** must be called successfully in that order before **CSL_semHwControlCmd()** can be called.

Post Condition

None

Modifies

The hardware registers of SEM.

Example

```
// handle for SEM
CSL_SemHandle handleSem;
// SEM object
CSL_SemObj      *pSemObj;
// CSL status
CSL_Status status;
CSL_SemParam    pSemParam;
//Number of SEM resources
pSemParam.flags = 2

CSL_semInit(NULL); // Init CSL for SEM module, this step is not
required
...
// Open handle for SEM module
handleSem = CSL_semOpen (pSemObj, CSL_SEM, &pSemParam, &status);
...
status = CSL_semHwControlCmd(handleSem,
                             CSL_SEM_CMD_WRITE_POST_DIRECT, NULL);
```

30.2.6 CSL_semGetBaseAddress

**CSL_Status CSL_semGetBaseAddress (CSL_InstNum instNum, CSL_SemParam
*pSemParam, CSL_SemBaseAddress * pBaseAddress)**

Description

Function to get the Base-address of the peripheral instance.

This function is used for getting the base-address of the peripheral instance.

Example

```
CSL_Status status;
CSL_semGetBaseAddress      baseAddress;
...
status = CSL_semGetBaseAddress(CSL_SEM, NULL, baseAddress);
```

Return Value CSL_Status

- CSL_SOK Function call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid Parameter

30.3 Data Structures

30.3.1 CSL_SemEOISet_Arg

Detailed Description

Object for Semaphore end of interrupt set.

Field Documentation

int _CSL_SemEOISet_Arg::enableErrorInt

Enable error interrupt

int _CSL_SemEOISet_Arg::masterId

master Id

30.3.2 CSL_SemErrSet_Arg

Detailed Description

Object for Semaphore error settings.

Field Documentation

int _CSL_SemErrSet_Arg::errCode

Error code

int _CSL_SemErrSet_Arg::mstId

Master Id

int _CSL_SemErrSet_Arg::semNum

Semaphore number

30.3.3 CSL_SemFlagClear_Arg

Detailed Description

Object to clear flags and get the status.

Field Documentation

unsigned int _CSL_SemFlagClear_Arg::BitMask32

32 bit mask

int _CSL_SemFlagClear_Arg::masterId

master Id

30.3.4 CSL_SemFlagSet_Arg

Detailed Description

Object for Semaphore flags to set.

Field Documentation

int _CSL_SemFlagSet_Arg::masterId

master Id

unsigned int _CSL_SemFlagSet_Arg::semId
Semaphore Id

30.3.5 CSL_SemFaultStatus

Detailed Description
Object for Fault Condition.

Field Documentation
CSL_BitMask2 CSL_SemFaultStatus::errorMask
Error mask

UInt16 CSL_SemFaultStatus::faultId
fault Id

UInt32 CSL_SemFaultStatus::semNum
Semaphore number

30.3.6 CSL_SemObj

Detailed Description
Object for semaphore handle.

Field Documentation
CSL_InstNum CSL_SemObj::instNum
Semaphore module instance

Int CSL_SemObj::semNum
Semaphore resource number

CSL_SemRegsOvly CSL_SemObj:: regs
Semaphore register overlay

30.3.7 CSL_SemVal

Detailed Description
Object for Semaphore Values.

Field Documentation
CSL_SemFlag CSL_SemVal::semFree
Semaphore free flag

Int CSL_SemVal::semNum
Semaphore number

CSL_SemOwnerId CSL_SemVal::semOwner
Semaphore owner Id

30.3.8 CSL_SemContext

Detailed Description
SEM specific context information. Present implementation doesn't have any Context information.

Field Documentation**Uint16 CSL_SemContext::contextInfo**

SEM context information. The declaration is just a placeholder for future implementation.

30.3.9 CSL_SemParam

Detailed Description

This is module specific parameter. Present implementation is for number of semaphore resources

Field Documentation**CSL_BitMask16 CSL_SemParam::flags**

Bit mask to be used for module specific parameters. The below declaration is for number of semaphore resources. Passed as an argument to CSL_semOpen().

30.3.10 CSL_SemBaseAddress

Detailed Description

This will have the base-address information for the peripheral instance

Field Documentation**CSL_SemRegsOvly CSL_SemBaseAddress::regs**

Base-address of the Configuration registers of SEM

30.4 Enumerations

30.4.1 CSL_SemError

Enumeration values

<i>CSL_SEM_ERR0</i>	semaphore error 0
<i>CSL_SEM_ERR1</i>	semaphore error 1
<i>CSL_SEM_ERR2</i>	semaphore error 2
<i>CSL_SEM_ERR3</i>	semaphore error 3

30.4.2 CSL_SemFlag

Enumeration values

<i>CSL_SEM_NOTFREE</i>	Semaphore is not available
<i>CSL_SEM_FREE</i>	Semaphore is available

30.4.3 CSL_SemHwControlCmd

Enumeration values

<i>CSL_SEM_CMD_SCRATCH_WRITE</i>	Semaphore scratch write Parameters: <i>NULL</i>
<i>CSL_SEM_CMD_RESET_RUN_WRITE</i>	Semaphore reset run write Parameters: <i>NULL</i>
<i>CSL_SEM_CMD_EOI_WRITE</i>	Semaphore EOI write Parameters: <i>CSL_SemEOISet_Arg</i>
<i>CSL_SEM_CMD_ERR_SET</i>	Semaphore error reset Parameters: <i>CSL_SemErrSet_Arg</i>
<i>CSL_SEM_CMD_FLAG_SET</i>	Semaphore flag set Parameters: <i>CSL_SemFlagSet_Arg</i>
<i>CSL_SEM_CMD_FREE_DIRECT</i>	Release semaphore (write to SEM register) Parameters: <i>NULL</i>
<i>CSL_SEM_CMD_WRITE_POST_DIRECT</i>	Perform Indirect acquisition (write to SEM register) Parameters: <i>NULL</i>
<i>CSL_SEM_CMD_FREE_INDIRECT</i>	Release semaphore (write to ISEM register) Parameters: <i>NULL</i>
<i>CSL_SEM_CMD_WRITE_POST_INDIRECT</i>	Perform Indirect acquisition (write to ISEM register) Parameters: <i>NULL</i>

<code>CSL_SEM_CMD_FREE_QSEM</code>	Release semaphore (write to QSEM register) Parameters: <i>NULL</i>
<code>CSL_SEM_CMD_WRITE_POST_QSEM</code>	Perform Indirect acquisition (write to QSEM register) Parameters: <i>NULL</i>
<code>CSL_SEM_CLEAR_ERR</code>	Clear error condition Parameters: <i>NULL</i>
<code>CSL_SEM_CLEAR_FLAGS</code>	Clear semaphore flags Parameters: <i>CSL_SemFlagClear_Arg</i>

30.4.4 CSL_SemHwStatusQuery

Enumeration values

<code>CSL_SEM_QUERY_SCRATCH_READ</code>	Queries scratch read Parameters: <i>NULL</i>
<code>CSL_SEM_QUERY_RESET_RUN_READ</code>	Queries reset run read Parameters: <i>NULL</i>
<code>CSL_SEM_QUERY_REVISION</code>	Queries revision Parameters: <i>Uint32*</i>
<code>CSL_SEM_QUERY_ERROR</code>	Queries revision Parameters: <i>CSL_SemFaultStatus*</i>
<code>CSL_SEM_QUERY_FLAGS</code>	Query semaphore flags Parameters: <i>CSL_BitMask32*</i>
<code>CSL_SEM_QUERY_STATUS</code>	Queries semaphore status Parameters: <i>CSL_SemVal*</i>
<code>CSL_SEM_QUERY_READ_PEND</code>	Queries semaphore read pend Parameters: <i>CSL_SemVal*</i>
<code>CSL_SEM_QUERY_READ_POST</code>	Queries semaphore read post Parameters: <i>CSL_SemVal*</i>

30.4.5 CSL_SemOwnerId

Enumeration values

<code>CSL_SEM_ID0</code>	semaphore ID 0
<code>CSL_SEM_ID1</code>	semaphore ID 1
<code>CSL_SEM_ID2</code>	semaphore ID 2
<code>CSL_SEM_ID3</code>	semaphore ID 3
<code>CSL_SEM_ID4</code>	semaphore ID 4

<i>CSL_SEM_ID5</i>	semaphore ID 5
<i>CSL_SEM_ID6</i>	semaphore ID 6
<i>CSL_SEM_ID7</i>	semaphore ID 7

Chapter 31 AIF Module

Topics

<u>31.1 Overview</u>
<u>31.2 Functions</u>
<u>31.3 Data Structures</u>
<u>31.4 Enumerations</u>
<u>31.5 Macros</u>
<u>31.6 Typedefs</u>

31.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within AIF module.

The AIF module converts serial data flowing on the backplane to byte format data which is captured and stored in the data buffer. The links carry circuit switched and packet switched data. The RAM in the AIF module contains separate inbound and outbound buffers to store circuit switched data and packet switched data. In addition, the AIF can perform combining, de-combining and re-direction of inbound link(s) to a specified outbound link(s). The AIF can also perform aggregation of an inbound link with an outbound link. The AIF has an external and internal interface.

The external interface of the AIF is used to interact with RF modules. The Tx and Rx part of the SERDES constitute the external interface through which inbound and outbound data flow in and out of the AIF (and TCI6488).

The internal interface is used to configure the AIF and to transport inbound and outbound data between GEM cores/RAC/EMIF and AIF. The inbound and outbound data flow through the DMA switch fabric and use the VBUSM (128 bit bus). The data flow for configuration, control and status accesses flow through the configuration switch fabric and use the VBUSP (32 bit bus).

In addition to the external and internal interface, the AIF requires system events from the frame sync module to perform scheduling. The AIF can be programmed to generate system event when error/alarm conditions occur. The AIF generated system events are routed to all GEM cores.

31.2 Functions

31.2.1 CSL_aifInit

CSL_Status CSL_aifInit (CSL_AifContext * *pContext*)

Description

This is the peripheral specific initialization function. This function is idempotent in that calling it many times is same as calling it once. This function initializes the CSL data structures, and doesn't touches the hardware.

Arguments

<i>pContext</i>	Pointer to module-context. As AIF doesn't have any context based information user is expected to pass NULL.
-----------------	---

Return Value

CSL_Status

- CSL_SOK - Always returns

Pre Condition

This function should be called before using any of the CSL APIs in the AIF module.

Post Condition

The CSL for AIF is initialized

Modifies

None

Example

```
CSL_aifInit(NULL); // Init CSL for Aif module
```

31.2.2 CSL_aifOpen

CSL_AifHandle CSL_aifOpen (CSL_AifLinkObj * *pAifLinkObj*, CSL_InstNum *aifNum*, CSL_AifParam * *paifParam*, CSL_Status * *pStatus*)

Description

The open call sets up the data structures for the particular instance of AIF device. The device can be re-opened anytime after it has been normally closed if so required. The handle returned by this call is input as an essential argument for rest of the APIs described for this module.

Arguments

<i>pAifLinkObj</i>	Pointer to the object that holds reference to the instance of AIF requested after the call
<i>aifNum</i>	Instance of AIF to which a handle is requested
<i>paifParam</i>	Module specific parameters
<i>pStatus</i>	pointer for returning status of the function call

Return Value

CSL_AifHandle

Valid AIF instance handle will be returned if status value is equal to CSL_SOK.

Pre Condition

AIF must be successfully initialized via *CSL_AIFInit()* before calling this function. Memory for the *CSL_AifObj* must be allocated outside this call. This object must be retained while usage of this peripheral.

Post Condition

1. The status is returned in the status variable. If status returned is
 - CSL_SOK Valid AIF handle is returned
 - CSL_ESYS_FAIL The AIF instance is invalid
2. AIF object structure is populated

Modifies

1. The status variable
2. AIF object structure

Example

```
// handle for link 0
CSL_AifHandle handleAifLink0;

// link object for link 0
CSL_AifLinkObj AifLinkObj0;

//AIF module specific parameters
CSL_AifParam aifParam;

// CSL status
CSL_Status status;

aifParam.LinkIndex = CSL_AIF_LINK_0;

// Open handle for link 0 - for use
handleAifLink0 = CSL_aifOpen( &AifLinkObj0,
                             CSL_AIF,
                             &aifParam,
                             &status);
if ((handleAifLink0 == NULL) || (status != CSL_SOK))
{
    printf ("\nError opening CSL_AIF");
    exit(1);
}
```

31.2.3 CSL_aifClose

CSL_Status CSL_aifClose (CSL_AifHandle hAifLink)

Description

The Close call releases the resources of the peripheral.

Arguments

hAifLink Handle to the aif instance

Return Value

CSL_Status

- CSL_SOK - Close successful
- CSL_ESYS_BADHANDLE - Invalid handle

Usage Constraints

Both **CSL_aiflnit()** and **CSL_aifOpen()** must be called successfully in that order before **CSL_aifClose()** can be called.

Example

```
CSL_AifHandle hAifLink;
// Properly initialize and open desired link for use
hAifLink = CSL_aifOpen(&AifLinkObj0, CSL_AIF, &aifParam,
&status);
CSL_aifClose(hAifLink)
```

31.2.4 CSL_aifHwSetup

CSL_Status CSL_aifHwSetup (CSL_AifHandle hAifLink, CSL_AifLinkSetup * linkSetup)

Description

It configures the AIF instance registers as per the values passed in the hardware setup structure.

Arguments

hAifLink Handle to the AIF instance

linkSetup Pointer to hardware setup structure

Return Value

CSL_Status

- CSL_SOK - Hardware setup successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVPARAMS - Hardware structure is not properly initialized

Pre Condition

Both **CSL_aiflnit()** and **CSL_aifOpen()** must be called successfully in that order before **CSL_aifHwSetup()** can be called. The user has to allocate space for & fill in the main setup structure appropriately before calling this function.

Post Condition

The specified instance will be setup according to value passed.

Modifies

The hardware registers of AIF.

Example

```
// handle for link 0
CSL_AifHandle handleAifLink0;

// link object for link 0
```

```

CSL_AifLinkObj AifLinkObj0;

//AIF module specific parameters
CSL_AifParam aifParam;

// CSL status
CSL_Status status;

// global config for AIF
CSL_AifGlobalSetup gblCfg = {...};

// Setup objects for configuring link 0
CSL_AifLinkSetup ConfigLink0;

// Setup for common params for link 0
CSL_AifCommonLinkSetup commoncfg0= {CSL_AIF_LINK_0};

// Setup for inbound for link 0
CSL_AifInboundLinkSetup inboundCfg0 = {...};

// set AIF module specific params
aifParam.LinkIndex = CSL_AIF_LINK_0;

// Open handle for link 0 - for use
handleAifLink0 = CSL_aifOpen( &AifLinkObj0,
                             CSL_AIF,
                             &aifParam,
                             &status);
if ((handleAifLink0 == NULL) || (status != CSL_SOK))
{
    printf ("\nError opening CSL_AIF");
    exit(1);
}

// Do config for link 0
ConfigLink0.globalSetup = &gblCfg;
ConfigLink0.commonlinkSetup = &commoncfg0;
// no outbound link setup for now
ConfigLink0.outboundlinkSetup = NULL;
ConfigLink0.inboundlinkSetup = &inboundCfg0;

//Do setup for link - 0
CSL_aifHwSetup(handleAifLink0, &ConfigLink0);

```

31.2.5 CSL_aifHwControl

CSL_Status CSL_aifHwControl (CSL_AifHandle *hAifLink*, CSL_AifHwControlCmd *cmd*, void * *arg*)

Description

This function performs various control operations on aif link, based on the command passed.

Arguments

haif	Handle to the aif instance
------	----------------------------

cmd	Operation to be performed on the aif
arg	Argument specific to the command

Return Value

CSL_Status

- CSL_SOK - Command execution successful.
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVCMD - Invalid command

Pre Condition

None

Post Condition

Registers of aif instance are configured according to the command and the command arguments. The command determines which registers are modified.

Modifies

Registers determined by the command

Example

```
// handle for link 0
CSL_AifHandle handleAifLink0;
// other link related declarations
...
// ctrl argument for hw command
Bool ctrlArg;

// Open handle for link 0 - for use
handleAifLink0 = CSL_aifOpen( &AifLinkObj0,
                             CSL_AIF,
                             &aifParam,
                             &status);

if ((handleAifLink0 == NULL) || (status != CSL_SOK))
{
    printf ("\nError opening CSL_AIF");
    exit(1);
}
// Do config for link 0
ConfigLink0.globalSetup = &gblCfg;
...
//Do setup for link - 0
CSL_aifHwSetup(handleAifLink0, &ConfigLink0);

ctrlArg = CSL_AIF_CTRL_RX_LINK_ENABLE;

// Send hw control command to enable Tx/Rx of link 0
CSL_aifHwControl(handleAifLink0,
                 CSL_AIF_CMD_ENABLE_DISABLE_RX_LINK, (void *)&ctrlArg);
```

31.2.6 CSL_aifGetHwStatus

CSL_Status CSL_aifGetHwStatus (CSL_AifHandle *hAifLink*, CSL_AifHwStatusQuery *Query*, void * *response*)

Description

This function is used to get the value of various parameters of the aif instance. The value returned depends on the query passed.

Arguments

hAifLink	Handle to the aif instance
Query	Query to be performed
response	Pointer to buffer to return the data requested by the query passed

Return Value

CSL_Status

- CSL_SOK - Successful completion of the query
- CSL_ESYS_BADHANDLE - Invalid handle
- CSL_ESYS_INVQUERY - Query command not supported
- CSL_ESYS_FAIL - Generic failure

Pre Condition

None

Post Condition

Data requested by query is returned through the variable "response"

Modifies

The input argument "response" is modified

Example

```
#define CSL_AIF_MAX_RX_MASTER_FRAME_OFFSET                256

// handle for link 0
CSL_AifHandle handleAifLink0;
// other link related declarations
...
// ctrl argument for hw command
Bool ctrlArg;
// query response
Uint16 response;

// Open handle for link 0 - for use
handleAifLink0 = CSL_aifOpen( &AifLinkObj0,
                             CSL_AIF,
                             &aifParam,
                             &status);
if ((handleAifLink0 == NULL) || (status != CSL_SOK))
{
    printf ("\nError opening CSL_AIF");
    exit(1);
}

// Do config for link 0
ConfigLink0.globalSetup = &gblCfg;
...
```

```

//Do setup for link - 0
CSL_aifHwSetup(handleAifLink0, &ConfigLink0);

ctrlArg = CSL_AIF_CTRL_RX_LINK_ENABLE;

// Send hw control command to enable Tx/Rx of link 0
CSL_aifHwControl(handleAifLink0,
CSL_AIF_CMD_ENABLE_DISABLE_RX_LINK, (void *)&ctrlArg);
...
//wait for a sufficient length of time, so Rx link has enough
time //to sync; 100 cycles wait time is arbitrarily chosen
wait(100);

// Get status of Rx master frame offset
CSL_aifGetHwStatus(      handleAifLink0,
                        CSL_AIF_QUERY_RM_LINK_RCVD_MSTR_FRAME_OFFSET,
                        (void *)&response);

if (response > CSL_AIF_MAX_RX_MASTER_FRAME_OFFSET)
{
    printf("\nRx Master Frame Offset exceeds bounds");
}

```

31.2.7 CSL_aifGetBaseAddress

CSL_Status CSL_aifGetBaseAddress (CSL_InstNum *aifNum*, CSL_AifParam * *paifParam*, CSL_AifBaseAddress * *pBaseAddress*)

Description

The **CSL_aifGetBaseAddress()** function returns the base-address of the specified aif instance.

Note: This function is open for re-implementing if the user wants to modify the base address of the peripheral object to point to a different location and there by allow CSL initiated write/reads into peripheral MMR's go to an alternate location. Please refer documentation for more details.

Arguments

CSL_InstNum
CSL_AifParam
CSL_AifBaseAddress

Return Value CSL_Status

- CSL_SOK Function call is successful
- CSL_ESYS_FAIL The instance number is invalid.
- CSL_ESYS_INVPARAMS Invalid Parameter

Example:

```

CSL_Status      status;
CSL_InstNum     aifNum; // Peripheral instance number
CSL_AifParam *  paifParam; // Module specific parameters.
CSL_AifBaseAddress * pBaseAddress; // Base address details
...
status = CSL_aifGetBaseAddress(CSL_AIF, paifParam, pBaseAddress);

```


31.3 Data Structures

31.3.1 CSL_AifAggregatorSetup

Detailed Description

This is a sub-structure in **CSL_AifOutboundLinkSetup** . This structure is used for configuring the parameters of aggregator.

Field Documentation

CSL_AifAggregatorMode CSL_AifAggregatorSetup::aggrMode

aggr mode - disabled, cd only(passthru), pe only, normal

Bool CSL_AifAggregatorSetup::bEnableAggrErr

Boolean indicating if aggregator error checking is enabled

31.3.2 CSL_AifAggregatorStatus

Detailed Description

This object contains the aggregator status fields.

Field Documentation

UInt8 CSL_AifAggregatorStatus::axCOverFlowErr

Overflow error for each antenna carrier

UInt8 CSL_AifAggregatorStatus::txRuleErr

[CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES]

Aggregator error for the specified transmission rule, Repeats 84 slots for 4x link, 42 slots for 2x link, 21 for 1 link. Valid values are 0 - No Error 1 - Overflow Error

31.3.3 CSL_AifBaseAddress

Detailed Description

This will have the base-address information for the peripheral instance.

Field Documentation

CSL_AifRegsOvly CSL_AifBaseAddress::regs

This is a pointer to the registers of the AIF

31.3.4 CSL_AifCdSetup

Detailed Description

This is a sub-structure in **CSL_AifGlobalSetup** . This structure is used for configuring the parameters for CD params for the AIF.

Field Documentation

Bool CSL_AifCdSetup::bCdEnableMask[CSL_AIF_MAX_NUM_LINKS]

Array indicating which output links of CD are enabled, values can be TRUE, FALSE

UInt8 CSL_AifCdSetup::cdOutSrcSel[CSL_AIF_MAX_NUM_LINKS]

Array containing src for each output of CD output values can be
CSL_AIF_CD_OUT_SRC_LINK_0, CSL_AIF_CD_OUT_SRC_LINK_1...,
CSL_AIF_CD_OUT_SRC_CB_1,

CSL_AifCombinerSetup*
CSL_AifCdSetup::pCombinerSetup[CSL_AIF_MAX_NUM_COMBINERS]

Pointer to combiner setup

CSL_AifDecombinerSetup*
CSL_AifCdSetup::pDecombinerSetup[CSL_AIF_MAX_NUM_DECOMBINERS]

Pointer to decombiner setup

31.3.5 CSL_AifCombinerSetup

Detailed Description

This is a sub-structure in **CSL_AifOutboundLinkSetup** . This structure is used for configuring the parameters of combiner.

Field Documentation

Bool CSL_AifCombinerSetup::bCombinerEnable

Enable the Combiner

UInt8 CSL_AifCombinerSetup::combinerInput[CSL_AIF_MAX_NUM_COMBINER_INPUTS]

Array indicating which inbound links are to be combined. Values can be CSL_AIF_LINK_0,...,CSL_AIF_LINK_5

UInt16 CSL_AifCombinerSetup::frameSyncOffset

combiner offset in vbus clk ticks from frame sync

UInt8 CSL_AifCombinerSetup::windowMasterFrameOffset

valid window for master frame offsets in vbus clock ticks 0,1..31

31.3.6 CSL_AifCommonLinkSetup

Detailed Description

This is a sub-structure in **CSL_AifHwSetup** . This structure is used for configuring the parameters of a link.

Field Documentation

CSL_AifLinkIndex CSL_AifCommonLinkSetup::linkIndex

Link index 0-5

CSL_AifLinkRate CSL_AifCommonLinkSetup::linkRate

Link rate

CSL_AifSerdesSetup* CSL_AifCommonLinkSetup::pSerdesSetup

pointer to Serdes setup

31.3.7 CSL_AifCpriTxMacSetup

Detailed Description

This is a sub-structure in **CSL_AifTxMacSetup** . This structure is used for configuring the parameters of the CPRI params relating to Tx MAC.

Field Documentation

Bool CSL_AifCpriTxMacSetup::bTxLof

boolean indicating if Z.130.0, b4 is transmitted

Bool CSL_AifCpriTxMacSetup::bTxLos

boolean indicating if Z.130.0, b3 is transmitted

Bool CSL_AifCpriTxMacSetup::bTxRai

boolean indicating if Z.130.0, b1 is transmitted

Bool CSL_AifCpriTxMacSetup::bTxReset

boolean indicating if Z.130.0, b0 is transmitted

Bool CSL_AifCpriTxMacSetup::bTxSdi

boolean indicating if Z.130.0, b2 is transmitted

UInt8 CSL_AifCpriTxMacSetup::TxPointerP

transmit pointer p

UInt8 CSL_AifCpriTxMacSetup::TxProtocolVer

protocol ver Z.2.0

UInt8 CSL_AifCpriTxMacSetup::TxStartup

start up info Z.66.0

31.3.8 CSL_AifDbFifoPtrStatus

Detailed Description

This object contains the data buffer fifo pointer status fields.

Field Documentation
UInt16 CSL_AifDbFifoPtrStatus::fifoHeadPtr[CSL_AIF_MAX_NUM_OF_OUTBOUND_FIFOS]

Packet Switched FIFO #0-29, Current state of head pointer

UInt16 CSL_AifDbFifoPtrStatus::fifoTailPtr[CSL_AIF_MAX_NUM_OF_OUTBOUND_FIFOS]

Packet Switched FIFO #0-29, Current state of tail pointer

31.3.9 CSL_AifDecombinerSetup

Detailed Description

This is a sub-structure in **CSL_AifOutboundLinkSetup** . This structure is used for configuring the parameters of decombiner.

Field Documentation
UInt8 CSL_AifDecombinerSetup::decombinerDest

[CSL_AIF_MAX_NUM_DECOMBINER_OUTPUTS]

indicating destination of decombiner outputs, Values can CSL_AIF_LINK_0,...CSL_AIF_LINK_5

CSL_AifLinkIndex CSL_AifDecombinerSetup::decombinerSrcSel

src link which is to be decombiner, must match with redirectionSrcSel. Values can CSL_AIF_LINK_0, ...CSL_AIF_LINK_5

31.3.10 CSL_AifExcEventCmnMaskObj

Detailed Description

This object contains the exception event common masks.

Field Documentation

CSL_BitMask32 CSL_AifExcEventCmnMaskObj::excCommonMask
Common Exception Event Register

CSL_AifExcEventIndex CSL_AifExcEventCmnMaskObj::excEvtType
specifies the AIF err event type Evt0/Evt1/Evt2/Evt3

31.3.11 CSL_AifExcEventLinkClearObj

Detailed Description

This object contains the exception event clear fields.

Field Documentation

CSL_BitMask32 CSL_AifExcEventLinkClearObj::excClearA
Exception Event Clear Register A

CSL_BitMask32 CSL_AifExcEventLinkClearObj::excClearB
Exception Event Clear Register B

31.3.12 CSL_AifExcEventLinkMaskObj

Detailed Description

This object contains the exception event mask fields.

Field Documentation

CSL_AifExcEventIndex CSL_AifExcEventLinkMaskObj::excEvtType
specifies the AIF err event type Evt0/Evt1/Evt2/Evt3

CSL_BitMask32 CSL_AifExcEventLinkMaskObj::excMaskA
Exception Event Register A

CSL_BitMask32 CSL_AifExcEventLinkMaskObj::excMaskB
Exception Event Register A

31.3.13 CSL_AifExcEventMode

Detailed Description

This object contains the exception event mode and index field.

Field Documentation

Bool CSL_AifExcEventMode::excEventMode
trigger / err_alarm

CSL_AifExcEventIndex CSL_AifExcEventMode::excEvtType
specifies the AIF err event type Evt0/Evt1/Evt2/Evt3

31.3.14 CSL_AifExcEventQueryObj

Detailed Description

This object is for exception event query.

Field Documentation

CSL_BitMask32 CSL_AifExcEventQueryObj::excCommon
Common Exception Event Register

CSL_BitMask32 CSL_AifExcEventQueryObj::excEventA
Exception Event Register A

CSL_BitMask32 CSL_AifExcEventQueryObj::excEventB
Exception Event Register B

31.3.15 CSL_AifExcLinkSelect

Detailed Description

This object contains the exception link select fields.

Field Documentation

CSL_AifExcEventIndex CSL_AifExcLinkSelect::excEvtType
specifies the AIF exception event type Evt2/Evt3

CSL_BitMask32 CSL_AifExcLinkSelect::excLinkSelA
Link Select A values for the AIF modules that are used to select the error/alarm that are aggregated to each event

CSL_BitMask32 CSL_AifExcLinkSelect::excLinkSelB
Link Select B values for the AIF modules that are used to select the error/alarm that are aggregated to each event

31.3.16 CSL_AifGlobalSetup

Detailed Description

This is a sub-structure in *CSL_AifHwSetup* . This structure is used for configuring the parameters global to AIF.

Field Documentation

CSL_AifLinkProtocol CSL_AifGlobalSetup::linkProtocol
Link protocol OBSAI/CPRI

CSL_AifCdSetup* CSL_AifGlobalSetup::pCdSetup
Frame structure used - WCDMA(FDD)/CDMA2000/WIMAX

31.3.17 CSL_AifInboundFifoSetup

Detailed Description

This is a sub-structure in *CSL_AifGlobalSetup* . This structure is used for configuring the parameters of inbound FIFO.

Field Documentation

CSL_AifInbndPsFifoEventDepth

CSL_AifInboundFifoSetup::eventDepth[CSL_AIF_MAX_NUM_INBOUND_PS_FIFO]
specifies fifo depth 1-16

31.3.18 CSL_AifInboundLinkSetup

Detailed Description

This is a sub-structure in *CSL_AifHwSetup* . This structure is used for configuring the parameters of an inbound link.

Field Documentation

CSL_AifAntDataWidth CSL_AifInboundLinkSetup::antDataWidth

Data width to be used on the link 7/8/15/16 bits

CSL_AifLinkDataType CSL_AifInboundLinkSetup::linkDataType

Type of link data - UL/DL/MAI/Control/NoData

UInt16 CSL_AifInboundLinkSetup::numActiveAxC

Num of active antenna carriers

CSL_AifPdSetup* CSL_AifInboundLinkSetup::pPdSetup

pointer to Protocol decoder link setup

31.3.19 CSL_AifIntMemStruct

Detailed Description

This object contains the elements required by AIF CSL to perform abstraction. This structure is used to create a static object which is the private/context memory used by AIF memory to store lookup tables, etc.

31.3.20 CSL_AifLinkObj

Detailed Description

This object contains the reference to the instance of AIF opened using the *CSL_aifOpen()* . The pointer to this, is passed to all AIF CSL APIs.

Field Documentation

UInt8 CSL_AifLinkObj::linkIndex

This is the link index being referred to by this object

CSL_InstNum CSL_AifLinkObj::perNum

This is the instance of AIF being referred to by this object

CSL_AifRegsOvly CSL_AifLinkObj::regs

This is a pointer to the registers of the AIF

31.3.21 CSL_AifLinkSetup

Detailed Description

This is the Setup structure for configuring AIF using *CSL_aifHwSetup()* function.

Field Documentation

CSL_AifGlobalSetup* CSL_AifLinkSetup::globalSetup

Pointer to the global AIF setup structure

CSL_AifInboundLinkSetup* CSL_AifLinkSetup::inboundlinkSetup

Pointer to the inbound link setup structure

CSL_AifOutboundLinkSetup* CSL_AifLinkSetup::outboundlinkSetup
 Pointer to the outbound link setup structure

31.3.22 CSL_AifOutboundLinkSetup

Detailed Description

This is a sub-structure in *CSL_AifHwSetup* . This structure is used for configuring the parameters of an outbound link.

Field Documentation

CSL_AifAntDataWidth CSL_AifOutboundLinkSetup::antDataWidth
 Data width to be used on the link 7/8/15/16 bit

CSL_BitMask8 CSL_AifOutboundLinkSetup::fifoEnBitMask
 Fifo enable bitmask for this link, max of CSL_AIF_MAX_NUM_OF_OUTBOUND_FIFOS per link
 Eg. to enable fifos 0,1 value is 0x3

CSL_AifLinkDataType CSL_AifOutboundLinkSetup::linkDataType
 Type of link data - DL/UL RSA/MAI/Generic

UInt16 CSL_AifOutboundLinkSetup::numActiveAxC
 Num of active antenna carriers

CSL_AifAggregatorSetup* CSL_AifOutboundLinkSetup::pAggrSetup
 pointer to aggregator setup

CSL_AifPeSetup* CSL_AifOutboundLinkSetup::pPeSetup
 pointer to protocol encoder setup

CSL_AifTxMacSetup* CSL_AifOutboundLinkSetup::pTxMacSetup
 pointer to Tx MAC setup

31.3.23 CSL_AifParam

Detailed Description

Module specific parameters.

Field Documentation

CSL_BitMask16 CSL_AifParam::flags
 Bit mask to be used for selecting link specific parameters. For this module, this is not used as there is only one module specific parameter. So user need not worry about this.

CSL_AifLinkIndex CSL_AifParam::linkIndex
 The link index for the SERDES link is to be specified

31.3.24 CSL_AifPdCommonSetup

Detailed Description

This is a sub-structure in *CSL_AifPdSetup* . This structure is used for configuring the parameters of protocol decoder which are common to all links.

Field Documentation

UInt16 CSL_AifPdCommonSetup::addressMask
 Mask specifying which 10 bits of 13 bit address header are used for inbound link

UInt8 CSL_AifPdCommonSetup::numTypeFieldEntries

Num of entries in type field lookup table used for inbound link

UInt16* CSL_AifPdCommonSetup::pInboundPsAddrFieldLut

[CSL_AIF_MAX_NUM_INBOUND_PS_FIFO]

Pointer to lookup table for mapping packet switched and control data to the correct packet switchd queue on inbound. Elements can range 0,1...(2¹³-1)

UInt8* CSL_AifPdCommonSetup::pInboundTypeFieldLut

Pointer to lookup table for type field of OBSAI header, elements can be either Inactive=0x0, CS=0x1, PS=0x2, Err = 0x4 Max size of LUT not to exceed
CSL_AIF_MAX_SIZE_TYPE_FIELD_LUT Note: index into LUT is the value of decoded type field

UInt16 CSL_AifPdCommonSetup::sizeInboundPsAddrFieldLut

[CSL_AIF_MAX_NUM_INBOUND_PS_FIFO]

Size of lookup table mapping PS/Ctrl data to correct queue on inbound link. Max size of LUT not to exceed CSL_AIF_MAX_SIZE_INBND_LINK_PS_ADDR_LUT

31.3.25 CSL_AifPdSetup

Detailed Description

This is a sub-structure in **CSL_AifInboundLinkSetup** . This structure is used for configuring the link parameters of protocol decoder.

Field Documentation
Bool CSL_AifPdSetup::bCpriCtrlWordCapture

Boolean indicating if CPRI control word is captured to PS RAM

Bool CSL_AifPdSetup::bEnablePd

Boolean indicating if PD is enabled for this link

UInt16* CSL_AifPdSetup::pInboundAxCAddrFieldLut

Pointer to lookup table for mapping AxC to its header field address for inbound link, range is 0,1,...(2¹³-1)

CSL_AifPdCommonSetup* CSL_AifPdSetup::pPdCommonSetup

pointer to Protocol decoder setup common for all links

UInt8 CSL_AifPdSetup::sizeInboundAxCAddrFieldLut

Num of entries in AxC address field lookup table used for inbound link. Max size of LUT not to exceed numActiveAxC on inbound link

31.3.26 CSL_AifPeSetup

Detailed Description

This is a sub-structure in **CSL_AifOutboundLinkSetup** . This structure is used for configuring the parameters of decombimer.

Field Documentation
UInt8 CSL_AifPeSetup::aAggrCtrlAxC Lut

[CSL_AIF_MAX_NUM_CS_TRANSMISSION_RULES]

Aggregator mode look up tablefor AxC, table can have follow values
CSL_AIF_PE_AGGR_CTRL_ADD_7_8_BIT - add 7/8 bit,

CSL_AIF_PE_AGGR_CTRL_ADD_15_16_BIT - add 15/16 bit,
 CSL_AIF_PE_AGGR_CTRL_INSERT - insert, CSL_AIF_PE_AGGR_CTRL_NOP - NOP Note:
 for first TCI6488 in chain use add for aggr ctrl and set aggregator to PE only mode

UInt8 CSL_AifPeSetup::aAggrCtrlCtrlDataLut
[CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES]
 Aggregator mode look up table for ctrl data, table can have follow values
 CSL_AIF_PE_AGGR_CTRL_INSERT - insert, CSL_AIF_PE_AGGR_CTRL_NOP - NOP

UInt16 CSL_AifPeSetup::aAxCCompare
[CSL_AIF_MAX_NUM_CS_TRANSMISSION_RULES]
 Array containing index of AxCs. Value is 0,1..aAxCMask[], set to
 CSL_AIF_PE_INVALID_COMPARE for non-active AxC

UInt16 CSL_AifPeSetup::aAxCMask [CSL_AIF_MAX_NUM_CS_TRANSMISSION_RULES]
 Array containing masks for AxC, Elements in array can take on values $(2^n - 1)$ $n=1,2,3,4,..11$.
 Value is CSL_AIF_PE_INVALID_AXC_MASK for non-active AxC

UInt16 CSL_AifPeSetup::aCtrlDataCompare
[CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES]
 Array containing index of control slots. Value is 0,1..aCtrlDataTCount[], set to 0 for non-active
 ctrl data counter

CSL_AifOutboundFifoIndex CSL_AifPeSetup::aCtrlDataSrcSel
[CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES]
 LUT for fifo index from which ctrl messages will be pulled, values can be 0,1..4.
 aCtrlDataSrcSel[0] is for ctrl msg 0, aCtrlDataSrcSel[1] for ctrl msg 1,..

UInt16 CSL_AifPeSetup::aCtrlDataTCount
[CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES]
 Array containing terminal count for ctrl slots, Elements in array can take on values 0,1,..1919.
 Value is 0 for non-active ctrl counter

UInt8 CSL_AifPeSetup::aggrCtrlIPsData
 Aggregator mode look up table for PS data, table can have follow values
 CSL_AIF_PE_AGGR_CTRL_INSERT - insert, CSL_AIF_PE_AGGR_CTRL_NOP - NOP

UInt8 CSL_AifPeSetup::obsaiAxCType
 OBSAI type that gets put in OBSAI header

UInt16 CSL_AifPeSetup::psDataCompare
 Compare value for packet switched index. Value is 0,1..PsDataMask, set to
 CSL_AIF_PE_INVALID_COMPARE for non-active Ps data

UInt16 CSL_AifPeSetup::psDataMask
 Mask for packet switched data, values are $2^n - 1$ where $n=0,1,..,11$. Value is
 CSL_AIF_PE_INVALID_PS_DATA_MASK for non-active PS Data

CSL_AifOutboundFifoIndex CSL_AifPeSetup::psDataSrcSel
 fifo index from which PS messages will be pulled, values can be 0,1..4

31.3.27 CSL_AifPidStatus

Detailed Description

This object contains the device revision number.

Field Documentation**UInt8 CSL_AifPidStatus::custom**

Custom version code

UInt8 CSL_AifPidStatus::major

Major revision (X) code

UInt8 CSL_AifPidStatus::minor

Minor revision (Y) code

31.3.28 CSL_AifRxMacCommonSetup

Detailed Description

This is a sub-structure in **CSL_AifRxMacSetup**. This structure is used for configuring the parameters of Rx MAC common to all inbound links.

Field Documentation**UInt16 CSL_AifRxMacCommonSetup::frameSyncT**

Threshold for correctly rx message groups to cause transition to ST3 state

UInt16 CSL_AifRxMacCommonSetup::frameUnSyncT

Threshold for incorrectly rx message group causing transition to ST1 state

UInt16 CSL_AifRxMacCommonSetup::syncT

Threshold for correctly rx blocks to cause transition to ST1 state

UInt16 CSL_AifRxMacCommonSetup::unSyncT

Threshold for num of incorrectly rx blocks to cause transition to ST0 state

31.3.29 CSL_AifRxMacSetup

Detailed Description

This is a sub-structure in **CSL_AifInboundLinkSetup**. This structure is used for configuring the parameters of Rx MAC specific to an inbound link.

Field Documentation**UInt32 CSL_AifRxMacSetup::losDetThreshold**

8b10b los detect threshold for line code violations received

UInt16 CSL_AifRxMacSetup::maxMasterFrameOffset

Width of max search window for master frame offset in vbus clock tics, 1...4095, 0 disables frame alignment checking

Int32 CSL_AifRxMacSetup::piOffset

PiOffset specified in vbus clock tics.

UInt8 CSL_AifRxMacSetup::validMasterFrameOffset

valid master frame offset in vbus clock tics 1..255, 0 disables frame alignment checking

31.3.30 CSL_AifSerdesCommonSetup

Detailed Description

This is a sub-structure in **CSL_AifSerdesSetup**. This structure is used for configuring the parameters of a SERDES module, the link index specifies which SERDES module is used links 0-3 use SERDES module 0, links 4-5 use SERDES module 1.

Field Documentation**Bool CSL_AifSerdesCommonSetup::bEnablePll**

Boolean indicating if PLL is to be enabled

CSL_AifPllMpyFactor CSL_AifSerdesCommonSetup::pllMpyFactor

PLL mpy setting 4,5,..25

31.3.31 CSL_AifSerdesSetup

Detailed Description

This is a sub-structure in **CSL_AifCommonLinkSetup**. This structure is used for configuring the parameters for SERDES params specific to a link.

Field Documentation**Bool CSL_AifSerdesSetup::bEnableRxAlign**

Rx Align

Bool CSL_AifSerdesSetup::bEnableRxLos

Rx loss of Signal

CSL_AifSerdesCommonSetup* CSL_AifSerdesSetup::pCommonSetup

pointer to common SERDES setup, links 0-3 use SERDES module 0, links 4,5 use SERDES module 1

CSL_AifSerdesRxCdrAlg CSL_AifSerdesSetup::rxCdrAlgorithm

specifies the clock/data recovery algorithm

CSL_AifSerdesRxEqConfig CSL_AifSerdesSetup::rxEqualizerConfig

Rx equalizer configuration

CSL_AifSerdesRxPairPolarity CSL_AifSerdesSetup::rxPairPolarity

polairty of Rx differential i/p - normal/inverted

CSL_AifSerdesRxTerm CSL_AifSerdesSetup::rxTermination

specifies the Rx termination options for AC/DC coupled scenarios

CSL_AifSerdesTxAmpConfig CSL_AifSerdesSetup::txAmpConfig

Tx amplitude setting, 1 of 8 values between 125-1250 mV dfpp

CSL_AifSerdesTxCommonMode CSL_AifSerdesSetup::txCommonMode

Tx common mode

CSL_AifSerdesTxDeConfig CSL_AifSerdesSetup::txDeEmphasisConfig

Tx de-emphasis setting 1 of 15 values between 4.76 - 71.42%

CSL_AifSerdesTxPairPolarity CSL_AifSerdesSetup::txPairPolarity

polairty of tx differential i/p - normal/inverted

31.3.32 CSL_AifTxMacSetup

Detailed Description

This is a sub-structure in **CSL_AifOutboundLinkSetup**. This structure is used for configuring the parameters of the Tx MAC.

Field Documentation**Bool CSL_AifTxMacSetup::bEnableFrameXmit**

enable Tx FSM to enter ST2 allowing frame transmission

Bool CSL_AifTxMacSetup::bEnableRxLos

Boolean indicating if Loss of Sync on Rx is to be enabled

Int16 CSL_AifTxMacSetup::deltaOffset

Delta offset specified in byte clocks, normalized to 1x link byte clock ticks (13ns)

CSL_AifCpriTxMacSetup* CSL_AifTxMacSetup::pCpriTxMacSetup

pointer to CPRI setup for Tx MAC

UInt8 CSL_AifTxMacSetup::threshTxMacFifo

threshold for full indication flag for the Tx MAC fifo

31.4 Enumerations

31.4.1 CSL_AifLinkProtocol

enum CSL_AifLinkProtocol

Enumeration values:

CSL_AIF_LINK_PROTOCOL_OBSAI Selects the OBSAI protocol

CSL_AIF_LINK_PROTOCOL_CPRI Selects the CPRI protocol

31.4.2 CSL_AifFrameStructure

enum CSL_AifFrameStructure

Enumeration values

CSL_AIF_FRAME_STRUCT_WCDMA_FDD Selects WCDMA FDD frame structure

CSL_AIF_FRAME_STRUCT_CDMA2000 Selects CDMA2000 frame structure

CSL_AIF_FRAME_STRUCT_WIMAX Selects WIMAX frame structure

31.4.3 CSL_AifAntDataWidth

enum CSL_AifAntDataWidth

Enumeration values

CSL_AIF_DATA_WIDTH_7_BIT Selects 7bit data width

CSL_AIF_DATA_WIDTH_8_BIT Selects 8bit data width

CSL_AIF_DATA_WIDTH_15_BIT Selects 15bit data width

CSL_AIF_DATA_WIDTH_16_BIT Selects 16bit data width

31.4.4 CSL_AifLinkIndex

enum CSL_AifLinkIndex

Enumeration values:

CSL_AIF_LINK_0 Selects link0

CSL_AIF_LINK_1 Selects link1

CSL_AIF_LINK_2 Selects link2

CSL_AIF_LINK_3 Selects link3

CSL_AIF_LINK_4 Selects link4

<i>CSL_AIF_LINK_5</i>	Selects link5
<i>CSL_AIF_MAX_NUM_LINKS</i>	Maximum number of links

31.4.5 CSL_AifLinkRate

enum CSL_AifLinkRate

Enumeration values:

<i>CSL_AIF_LINK_RATE_1x</i>	Selects 1X link rate
<i>CSL_AIF_LINK_RATE_2x</i>	Selects 2X link rate
<i>CSL_AIF_LINK_RATE_4x</i>	Selects 4X link rate

31.4.6 CSL_AifRxSyncState

enum CSL_AifRxSyncState

Enumeration values:

<i>CSL_AIF_RX_MAC_ST_0</i>	Selects the RxMAC state 0
<i>CSL_AIF_RX_MAC_ST_1</i>	Selects the RxMAC state 1
<i>CSL_AIF_RX_MAC_ST_2</i>	Selects the RxMAC state 2
<i>CSL_AIF_RX_MAC_ST_3</i>	Selects the RxMAC state 3

31.4.7 CSL_AifTxSyncState

enum CSL_AifTxSyncState

Enumeration values:

<i>CSL_AIF_TX_MAC_ST_0</i>	Selects the TxMAC state 0
<i>CSL_AIF_TX_MAC_ST_1</i>	Selects the TxMAC state 1
<i>CSL_AIF_TX_MAC_ST_2</i>	Selects the TxMAC state 2

31.4.8 CSL_AifLinkDataType

enum CSL_AifLinkDataType

Enumeration values:

<i>CSL_AIF_LINK_DATA_TYPE_DL</i>	Selects the link data type as downlink
<i>CSL_AIF_LINK_DATA_TYPE_UL_RSA</i>	Selects the link data type as uplink
<i>CSL_AIF_LINK_DATA_TYPE_MAI</i>	Selects the link data type as MAI
<i>CSL_AIF_LINK_DATA_TYPE_GENERIC</i>	Selects the link data type as generic

31.4.9 CSL_AifLinkFormatType

enum CSL_AifLinkFormatType

Enumeration values:

<i>CSL_AIF_LINK_FORMAT_1x_RATE</i>	Selects 1X link format rate
<i>CSL_AIF_LINK_FORMAT_2x_RATE</i>	Selects 2X link format rate
<i>CSL_AIF_LINK_FORMAT_4x_RATE</i>	Selects 4X link format rate
<i>CSL_AIF_MAX_NUM_LINK_FORMATS</i>	Maximum number of link format rate

31.4.10 CSL_AifCombinerIndex

enum CSL_AifCombinerIndex

Enumeration values:

<i>CSL_AIF_COMBINER_0</i>	Selects the Combiner 0
<i>CSL_AIF_COMBINER_1</i>	Selects the Combiner 1
<i>CSL_AIF_MAX_NUM_COMBINERS</i>	Maximum number of Combiners

31.4.11 CSL_AifDecombinerIndex

enum CSL_AifDecombinerIndex

Enumeration values:

<i>CSL_AIF_DECOMBINER_0</i>	Selects the DeCombiner 0
<i>CSL_AIF_DECOMBINER_1</i>	Selects the DeCombiner 1
<i>CSL_AIF_MAX_NUM_DECOMBINERS</i>	Maximum number of DeCombiners

31.4.12 CSL_AifCombinerInput

enum CSL_AifCombinerInput

Enumeration values:

<i>CSL_AIF_CB_SRC_A</i>	Selects the combiner source index A
<i>CSL_AIF_CB_SRC_B</i>	Selects the combiner source index B
<i>CSL_AIF_CB_SRC_C</i>	Selects the combiner source index C
<i>CSL_AIF_CB_SRC_D</i>	Selects the combiner source index D
<i>CSL_AIF_MAX_NUM_COMBINER_INPUTS</i>	Maximum number of Combiner inputs

31.4.13 CSL_AifDecombinerDest

enum CSL_AifDecombinerDest

Enumeration values:

<i>CSL_AIF_DC_DEST_A</i>	Selects the Decombiner destination index A
<i>CSL_AIF_DC_DEST_B</i>	Selects the Decombiner destination index B
<i>CSL_AIF_DC_DEST_C</i>	Selects the Decombiner destination index C
<i>CSL_AIF_DC_DEST_D</i>	Selects the Decombiner destination index D
<i>CSL_AIF_MAX_NUM_DECOMBINER_OUTPUTS</i>	Maximum number of Decombiner outputs

31.4.14 CSL_AifAggregatorMode

enum CSL_AifAggregatorMode

Enumeration values:

<i>CSL_AIF_AGGR_MODE_DISABLED</i>	Selects Aggregator mode disabled
<i>CSL_AIF_AGGR_MODE_CD_IP_ONLY</i>	Aggregator takes combiner input only or pass thru mode
<i>CSL_AIF_AGGR_MODE_PE_IP_ONLY</i>	Aggregator takes protocol encoder input only
<i>CSL_AIF_AGGR_MODE_NORMAL</i>	All aggregator fns enabled

31.4.15 CSL_AifAggrSrcSel

enum CSL_AifAggrSrcSel

Enumeration values:

<i>CSL_AIF_AGGR_SRC_SEL_CD</i>	Aggregator source select combiner Decombiner
<i>CSL_AIF_AGGR_SRC_SEL_PE</i>	Aggregator source select Protocol encoder

31.4.16 CSL_AifCpriCtrlIWMode

enum CSL_AifCpriCtrlIWMode

Enumeration values:

<i>CSL_AIF_CPRI_CTRL_WORD_ZEROS</i>	Control word zeros expect for K chars
<i>CSL_AIF_CPRI_CTRL_WORD_READ_FROM_RAM</i>	Selects Control word read from RAM

31.4.17 CSL_AifInbndPsFifoEventDepth

enum CSL_AifInbndPsFifoEventDepth

Enumeration values:

<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_1</i>	Selects inbound packet switched FIFO event depth 1
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_2</i>	Selects inbound packet switched FIFO event depth 2
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_3</i>	Selects inbound packet switched FIFO event depth 3
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_5</i>	Selects inbound packet switched FIFO event depth 5
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_6</i>	Selects inbound packet switched FIFO event depth 6
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_7</i>	Selects inbound packet switched FIFO event depth 7
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_8</i>	Selects inbound packet switched FIFO event depth 8
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_9</i>	Selects inbound packet switched FIFO event depth 9
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_10</i>	Selects inbound packet switched FIFO event depth 10
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_11</i>	Selects inbound packet switched FIFO event depth 11
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_12</i>	Selects inbound packet switched FIFO event depth 12
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_13</i>	Selects inbound packet switched FIFO event depth 13
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_14</i>	Selects inbound packet switched FIFO event depth 14
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_15</i>	Selects inbound packet switched FIFO event depth 15
<i>CSL_AIF_INBOUND_PS_FIFO_EVENT_DEPTH_16</i>	Selects inbound packet switched FIFO event depth 16

31.4.18 CSL_AifOutboundFifoIndex

enum CSL_AifOutboundFifoIndex

Enumeration values:

<i>CSL_AIF_OUT_PS_FIFO_0</i>	Selects outbound packet switched FIFO 0
<i>CSL_AIF_OUT_PS_FIFO_1</i>	Selects outbound packet switched FIFO 1
<i>CSL_AIF_OUT_PS_FIFO_2</i>	Selects outbound packet switched FIFO 2
<i>CSL_AIF_OUT_PS_FIFO_3</i>	Selects outbound packet switched FIFO 3
<i>CSL_AIF_OUT_PS_FIFO_4</i>	Selects outbound packet switched FIFO 4
<i>CSL_AIF_MAX_NUM_OUTBOUND_PS_FIFO_PER_LINK</i>	Maximum number of outbound packet switched FIFO per link

31.4.19 CSL_AifSerdesIndex

enum CSL_AifSerdesIndex

Enumeration values:

<i>CSL_AIF_SERDES_MODULE_0</i>	Selects the SERDES module 0
<i>CSL_AIF_SERDES_MODULE_1</i>	Selects the SERDES module 1
<i>CSL_AIF_MAX_NUM_SERDES_MODULES</i>	Maximum number of serdes modules

31.4.20 CSL_AifSerdesRxTerm

enum CSL_AifSerdesRxTerm

Enumeration values:

<i>CSL_AIF_SERDES_RX_TERM_COMMON_POINT_VDDT</i>	Selects the Rx termination common point connected to VDDT
<i>CSL_AIF_SERDES_RX_TERM_COMMON_POINT_0_8</i>	Selects the Rx termination common point set to 0.8 VDDT
<i>CSL_AIF_SERDES_RX_TERM_COMMON_POINT_FLOATING</i>	Selects the Rx termination common point floating

31.4.21 CSL_AifSerdesRxEqConfig

enum CSL_AifSerdesRxEqConfig

Enumeration values:

<i>CSL_AIF_SERDES_RX_EQ_MAXIMUM</i>	Selects the maximum Receiver equalizer
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE</i>	Selects the adaptive Receiver equalizer
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_365</i>	Selects the adaptive Receiver equalizer to 365MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_275</i>	Selects the adaptive Receiver equalizer to 275MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_195</i>	Selects the adaptive Receiver equalizer to 195MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_140</i>	Selects the adaptive Receiver equalizer to 140MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_105</i>	Selects the adaptive Receiver equalizer to 105MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_75</i>	Selects the adaptive Receiver equalizer to 75MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_55</i>	Selects the adaptive Receiver equalizer to 55MHz
<i>CSL_AIF_SERDES_RX_EQ_ADAPTIVE_50</i>	Selects the adaptive Receiver equalizer to 50MHz

31.4.22 CSL_AifSerdesRxPairPolarity

enum CSL_AifSerdesRxPairPolarity

Enumeration values:

<i>CSL_AIF_SERDES_RX_PAIR_NORMAL_POLARITY</i>	Selects the Receive pair normal polarity
<i>CSL_AIF_SERDES_RX_PAIR_INVERTED_POLARITY</i>	Selects the Receive pair inverted polarity

31.4.23 CSL_AifSerdesRxCdrAlg

enum CSL_AifSerdesRxCdrAlg

Enumeration values:

<i>CSL_AIF_SERDES_RX_CDR_FIRST_ORDER_THRESH_1</i>	Selects the first order, threshold of 1
---	---

<i>CSL_AIF_SERDES_RX_CDR_FIRST_ORDER_THRESH_16</i>	Selects the first order, threshold of 16
<i>CSL_AIF_SERDES_RX_CDR_SECOND_ORDER_HP_THRESH_1</i>	Selects the first order,High precision, threshold of 1
<i>CSL_AIF_SERDES_RX_CDR_SECOND_ORDER_HP_THRESH_16</i>	Selects the first order,High precision, threshold of 16
<i>CSL_AIF_SERDES_RX_CDR_SECOND_ORDER_LP_THRESH_1</i>	Selects the second order,low precision, threshold of 1
<i>CSL_AIF_SERDES_RX_CDR_SECOND_ORDER_LP_THRESH_16</i>	Selects the second order,low precision, threshold of 16
<i>CSL_AIF_SERDES_RX_CDR_FIRST_ORDER_THRESH_1_FAST_LOCK</i>	Selects the first order threshold of 1 with fast lock
<i>CSL_AIF_SERDES_RX_CDR_SECOND_ORDER_LP_FAST_LOCK</i>	Selects the second order,low precision, with fast lock

31.4.24 CSL_AifSerdesTxCommonMode

enum CSL_AifSerdesTxCommonMode

Enumeration values:

<i>CSL_AIF_SERDES_TX_NORMAL_COMMON_MODE</i>	Selects normal common mode
<i>CSL_AIF_SERDES_TX_RAISED_COMMON_MODE</i>	Selects Raised common mode

31.4.25 CSL_AifSerdesTxPairPolarity

enum CSL_AifSerdesTxPairPolarity

Enumeration values:

<i>CSL_AIF_SERDES_TX_PAIR_NORMAL_POLARITY</i>	Selects Tx pair normal polarity
<i>CSL_AIF_SERDES_TX_PAIR_INVERTED_POLARITY</i>	Selects Tx pair inverted polarity

31.4.26 CSL_AifSerdesTxAmpConfig

enum CSL_AifSerdesTxAmpConfig

Enumeration values:

<i>CSL_AIF_SERDES_TX_AMP_CONFIG_125</i>	Selects the Tx ouput swing to 125 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_250</i>	Selects the Tx ouput swing to 250 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_500</i>	Selects the Tx ouput swing to 500 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_625</i>	Selects the Tx ouput swing to 625 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_750</i>	Selects the Tx ouput swing to 750 mV

<i>CSL_AIF_SERDES_TX_AMP_CONFIG_1000</i>	Selects the Tx ouput swing to 1000 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_1250</i>	Selects the Tx ouput swing to 1250 mV
<i>CSL_AIF_SERDES_TX_AMP_CONFIG_1375</i>	Selects the Tx ouput swing to 1375 mV

31.4.27 CSL_AifSerdesTxDeConfig

enum CSL_AifSerdesTxDeConfig

Enumeration values:

<i>CSL_AIF_SERDES_TX_DE_CONFIG_0</i>	Delects the ouput de-emphasis to 0dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_1</i>	Delects the ouput de-emphasis to -0.42dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_2</i>	Delects the ouput de-emphasis to -0.87dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_3</i>	Delects the ouput de-emphasis to -1.34dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_4</i>	Delects the ouput de-emphasis to -1.83dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_5</i>	Delects the ouput de-emphasis to -2.36dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_6</i>	Delects the ouput de-emphasis to -2.92dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_7</i>	Delects the ouput de-emphasis to -3.52dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_8</i>	Delects the ouput de-emphasis to -4.16dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_9</i>	Delects the ouput de-emphasis to -4.86dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_10</i>	Delects the ouput de-emphasis to -5.61dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_11</i>	Delects the ouput de-emphasis to -6.44dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_12</i>	Delects the ouput de-emphasis to -7.35dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_13</i>	Delects the ouput de-emphasis to -8.38dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_14</i>	Delects the ouput de-emphasis to -9.54dB
<i>CSL_AIF_SERDES_TX_DE_CONFIG_15</i>	Delects the ouput de-emphasis to -10.87dB

31.4.28 CSL_AifPIIMpyFactor

enum CSL_AifPIIMpyFactor

Enumeration values:

<i>CSL_AIF_PLL_MUL_FACTOR_4X</i>	Select 4x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_5X</i>	Select 5x PLL multiply factor

<i>CSL_AIF_PLL_MUL_FACTOR_6X</i>	Select 6x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_8X</i>	Select 8x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_10X</i>	Select 10x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_12X</i>	Select 12x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_12_5X</i>	Select 12.5x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_15X</i>	Select 15x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_20X</i>	Select 20x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_25X</i>	Select 25x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_50X</i>	Select 50x PLL multiply factor
<i>CSL_AIF_PLL_MUL_FACTOR_60X</i>	Select 60x PLL multiply factor

31.4.29 CSL_AifExcEventIndex

enum CSL_AifExcEventIndex

Enumeration values:

<i>CSL_AIF_ERR_EVENT_0</i>	Selects the error event 0
<i>CSL_AIF_ERR_EVENT_1</i>	Selects the error event 1
<i>CSL_AIF_ERR_EVENT_2</i>	Selects the error event 2
<i>CSL_AIF_ERR_EVENT_3</i>	Selects the error event 3
<i>CSL_MAX_NUM_ERR_EVENTS</i>	Maximum number of error events

31.4.30 CSL_AifRxSetSyncState

enum CSL_AifRxSetSyncState

Enumeration values:

<i>CSL_AIF_SET_RX_MAC_ST_0</i>	Force receiver state machine to ST0 state
<i>CSL_AIF_SET_RX_MAC_ST_1</i>	Force receiver state machine to ST1 state
<i>CSL_AIF_SET_RX_MAC_ST_2</i>	Force receiver state machine to ST2 state
<i>CSL_AIF_SET_RX_MAC_ST_3</i>	Force receiver state machine to ST3 state
<i>CSL_AIF_SET_RX_MAC_ST_INACTIVE</i>	Force receiver state machine to INACTIVE state

31.4.31 CSL_AifTxSetSyncState

enum CSL_AifTxSetSyncState

Enumeration values:

<i>CSL_AIF_SET_TX_MAC_ST_0</i>	Force transmitter state machine to ST0 state
<i>CSL_AIF_SET_TX_MAC_ST_1</i>	Force transmitter state machine to ST1 state

<code>CSL_AIF_SET_TX_MAC_ST_2</code>	Force transmitter state machine to ST2 state
<code>CSL_AIF_SET_TX_MAC_ST_INACTIVE</code>	Force transmitter state machine to INACTIVE state

31.4.32 CSL_AifHwControlCmd

enum CSL_AifHwControlCmd

This is the set of control commands that are passed to `CSL_aifHwControl()`, with an optional argument type-casted to `void*`

The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<code>CSL_AIF_CMD_ENABLE_DISABLE_RX_LINK</code>	Starts a Rx link (argument type: Bool *)
<code>CSL_AIF_CMD_ENABLE_DISABLE_TX_LINK</code>	Starts a Tx link (argument type: Bool *)
<code>CSL_AIF_CMD_ENABLE_DISABLE_SD0_PLL</code>	Enable SD0 PLL (argument type: Bool *)
<code>CSL_AIF_CMD_MOD_SD0_PLL_MUL_FACT</code>	Select SD0 PLL multiply factor (argument type: Uint8 *)
<code>CSL_AIF_CMD_ENABLE_DISABLE_SD1_PLL</code>	Enable SD1 PLL (argument type: Bool *)
<code>CSL_AIF_CMD_MOD_SD1_PLL_MUL_FACT</code>	Select SD1 PLL multiply factor (argument type: Uint8 *)
<code>CSL_AIF_CMD_MOD_RM_LINK_RX_STATE</code>	Force receiver state machine state (argument type: <code>CSL_AifRxSetSyncState *</code>)
<code>CSL_AIF_CMD_MOD_RM_LINK_VALID_MSTR_FRAME_WIND</code>	Modify measurement window for valid Master Frame Offset (argument type: Int16 *)
<code>CSL_AIF_CMD_MOD_RM_LINK_RCVD_MSTR_FRAME_WIND</code>	Modify the measurement window for received Master Frame Offset (argument type: Int16 *)
<code>CSL_AIF_CMD_MOD_RM_LINK_PI_OFFSET</code>	Modify the Pi Offset of Rx link (argument type: Int16 *)
<code>CSL_AIF_CMD_SET_RM_LINK_LOS_DET_THOLD</code>	Sets 8b10b los detect threshold value (argument type: Uint32 *)
<code>CSL_AIF_CMD_ENABLE_DISABLE_RM_LINK_ERR_SUPPRESS</code>	Suppress error reporting when the receiver state machine is not in state ST3 (argument type: Bool *)
<code>CSL_AIF_CMD_ENABLE_DISABLE_RM_LINK_EXTRA_K28_7_ERR_SUPPRESS</code>	Suppress error reporting of "extra" K28.7 characters detected in the data stream (argument type: Bool *)
<code>CSL_AIF_CMD_MOD_RM_SYNC_THRESH</code>	Modify Threshold value for consecutive

	valid blocks of bytes which result in state ST1 (argument type: Uint16 *)
<i>CSL_AIF_CMD_MOD_RM_FRAME_SYNC_THRESH</i>	Modify Threshold value for consecutive valid message groups which result in state ST3 (argument type: Uint16 *)
<i>CSL_AIF_CMD_MOD_RM_UNSYNC_THRESH</i>	Modify Threshold value for consecutive invalid blocks of bytes which result in state ST0 (argument type: Uint16 *)
<i>CSL_AIF_CMD_MOD_RM_FRAME_UNSYNC</i>	Modify Threshold value for consecutive invalid message groups which result in state ST1 (argument type: Uint16 *)
<i>CSL_AIF_CMD_MOD_TM_LINK_TX_STATE</i>	Modify transmitter state machine state (argument type: CSL_AifTxSetSyncState *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_TM_LINK_XMIT</i>	Enable allowing the TX FSM from entering state ST2 (argument type: Bool *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_TM_LINK_LOS</i>	Enable impact of the rm_loss_of_signal to the transmitter state machine (argument type: Bool *)
<i>CSL_AIF_CMD_MOD_TM_LINK_FIFO_FULL_THOLD</i>	Modify the threshold for the full indication flag for the TX MAC FIFO (argument type: Uint8 *)
<i>CSL_AIF_CMD_MOD_TM_LINK_DELTA_OFFSET</i>	Modify the Delta Offset of Tx Link (argument type: Uint16 *)
<i>CSL_AIF_CMD_SET_TM_LINK_BFN_LOW</i>	Transmit Node B Frame number low byte (Z.128.0) - loads u_cntr low byte (argument type: Uint8 *)
<i>CSL_AIF_CMD_SET_TM_LINK_BFN_HIGH</i>	Transmit Node B Frame number high byte (Z.130.0) - loads u_cntr high byte (argument type: Uint8 *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_CB0</i>	Enable Combiner0 (argument type: Bool *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_CB1</i>	Enable Combiner1 (argument type: Bool *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_DC0</i>	Enable Decombiner0 (argument type: Bool *)
<i>CSL_AIF_CMD_ENABLE_DISABLE_DC1</i>	Enable Decombiner1 (argument type: Bool *)
<i>CSL_AIF_CMD_MOD_CB0_FS_OFFSET</i>	Modify Combiner0 offset from frame

	sync (argument type: Int16 *)
<i>CSL_AIF_CMD_MOD_CB1_FS_OFFSET</i>	Modify Combiner1 offset from frame sync (argument type: Int16 *)
<i>CSL_AIF_CMD_MOD_CB0_VALID_DATA_WIND</i>	Modify Combiner0 valid data window for Master Frame Offset (argument type: Uint8 *)
<i>CSL_AIF_CMD_MOD_CB1_VALID_DATA_WIND</i>	Modify Combiner1 valid data window for Master Frame Offset (argument type: Uint8 *)
<i>CSL_AIF_CMD_MOD_AG_LINK_MODE</i>	Modify the Aggregator Mode (argument type: Uint8 *)
<i>CSL_AIF_CMD_ENABLE_AG_LINK_ERR_CHECK</i>	Aggregator error checking enable (argument type: Uint8 *)
<i>CSL_AIF_CMD_SET_AG_LINK_HDR_ERR_SEL</i>	Controls the data source selected by the Aggregator when there is an OBSAI header mismatch and the PE is requesting aggregation (argument type: Uint8 *)
<i>CSL_AIF_CMD_PD_ENABLE_DISABLE_LINK</i>	Enable PD link (argument type: Bool *)
<i>CSL_AIF_CMD_PD_ENABLE_DISABLE_LINK_CPRI_CTRL_CAPT</i>	Enables CPRI control word capture (argument type: Bool *)
<i>CSL_AIF_CMD_PD_MOD_LINK_84CNT_TS_INCR_LUT0</i>	Modify Time Stamp to increment, Incremented value is used for next message (argument type: Uint32 *)
<i>CSL_AIF_CMD_PD_MOD_LINK_84CNT_TS_INCR_LUT1</i>	Modify Time Stamp to increment, Incremented value is used for next message (argument type: Uint32 *)
<i>CSL_AIF_CMD_PD_MOD_LINK_84CNT_TS_INCR_LUT2</i>	Modify Time Stamp to increment, Incremented value is used for next message (argument type: Uint32 *)
<i>CSL_AIF_CMD_PD_RESET_TYPE_LUT</i>	Reset the cir,pkt and err PD type lut
<i>CSL_AIF_CMD_PD_RESET_ADR_LUT</i>	Reset PD addr lut
<i>CSL_AIF_CMD_PE_ENABLE_DISABLE_LINK</i>	Enable the PE link (argument type: Bool *)
<i>CSL_AIF_CMD_PE_MOD_LINK_84CNT_LUT</i>	Enable of a particular message slot (arg is pointer to array of 84)
<i>CSL_AIF_CMD_PE_RESET_LINK_84CNT_LUT</i>	Disable all message slot (arg is pointer to array of 84)

<i>CSL_AIF_CMD_PE_MOD_LINK_21CNT_ID_LUT0</i>	Modify PE Link 21 Count Identity Look Up Table0
<i>CSL_AIF_CMD_PE_RESET_LINK_21CNT_ID_LUT0</i>	Reset PE Link 21 Count Identity Look Up Table0
<i>CSL_AIF_CMD_PE_MOD_LINK_21CNT_ID_LUT1</i>	Modify PE Link 21 Count Identity Look Up Table1
<i>CSL_AIF_CMD_PE_RESET_LINK_21CNT_ID_LUT1</i>	Modify PE Link 21 Count Identity Look Up Table1
<i>CSL_AIF_CMD_DB_ENABLE_MEM_LEAK_FLUSH</i>	Flushes all outbound pktsw fifo's if a memory leak is detected (argument type: Uint8 *)
<i>CSL_AIF_CMD_DB_ENABLE_LINK_CAPTURE</i>	Enables "Logic Analyzer" capture (argument type: Uint8 *)
<i>CSL_AIF_CMD_DB_OUT_ENABLE_PKTSW_FIFO</i>	Enables Packet Switched Outbound FIFOs per link (argument type: Uint32 *)
<i>CSL_AIF_CMD_DB_OUT_ENABLE_PKTSW_FIFO_FLUSH</i>	Flush Fifo, Fifo Pointers are cleared, FIFO will still be enabled (argument type: Uint32 *)
<i>CSL_AIF_CMD_EE_ENABLE_LINK_MASK</i>	Enables the Exception Event Link mask (argument type: CSL_AifExcEventLinkMaskObj *)
<i>CSL_AIF_CMD_EE_DISABLE_LINK_MASK</i>	Disables the Exception Event Link mask (argument type: CSL_AifExcEventLinkMaskObj *)
<i>CSL_AIF_CMD_EE_CLEAR_LINK_EVENT</i>	Clears the Exception Link Event (argument type: CSL_AifExcEventLinkClearObj *)
<i>CSL_AIF_CMD_EE_ENABLE_COMMON_MASK</i>	Enables the Exception Event Common mask (argument type: CSL_AifExcEventCmnMaskObj *)
<i>CSL_AIF_CMD_EE_DISABLE_COMMON_MASK</i>	Disables the Exception Event Common mask (argument type: CSL_AifExcEventCmnMaskObj *)
<i>CSL_AIF_CMD_EE_CLEAR_COMMON_EVENT</i>	Clears the Exception Common Event (argument type: CSL_AifExcEventCmnMaskObj *)
<i>CSL_AIF_CMD_EE_REEVALUATE_INTERRUPT_LINE</i>	Whenever this register is written, the ai_event[3:0] signal addressed by this field is allowed to generate another event if the any aggregated bits in the associated Interrupt Source Masked

	Status Registers are set to a 1 (argument type: CSL_AifExcEventIndex*)
<i>CSL_AIF_CMD_EE_SET_EVENT_MODE</i>	Set the Event mode error alarm/trigger (argument type: CSL_AifExcEventMode*)
<i>CSL_AIF_CMD_EE_SET_AIF_RUN_STATUS_BIT</i>	Sets the AIF run status bit (argument type: NULL)
<i>CSL_AIF_CMD_EE_CLEAR_AIF_RUN_STATUS_BIT</i>	Clears the AIF run status bit (argument type: NULL)
<i>CSL_AIF_CMD_EE_SET_LINK_SELECT</i>	Sets Link select bits for the AIF modules (argument type: CSL_AifExcLinkSelect *)

31.4.33 CSL_AifHwStatusQuery

enum CSL_AifHwStatusQuery

This is the set of query commands to get the status of various operations in AIF
The arguments, if any, to be passed with each command are specified next to that command.

Enumeration values:

<i>CSL_AIF_QUERY_VERSION</i>	Queries the version of the module accessed Parameters: (CSL_AifPidStatus *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_SD_RX_LOS_DET</i>	Queries SD receive loss of signal detect Parameters: (UInt8 *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_PLL_LOCK</i>	Queries status of PLL lock on SERDES Parameters: (UInt16 *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_RM_LINK_SYNC_STATE</i>	Queries the status of the AIF Rx MAC state machine Parameters: (CSL_AifRxSyncState *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_RM_LINK_LOS_OF_SIG</i>	Queries the value of the Rx loss of signal Parameters: (UInt8 *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_RM_LINK_LOS_DET</i>	Queries the value of the Rx loss detection :(UInt8 *) Returns: CSL_SOK
<i>CSL_AIF_QUERY_RM_LINK_FRAME_BNDY_RANGE</i>	Queries the value of the RxMac frame boundary range Parameters: (UInt8 *) Returns: CSL_SOK

<code>CSL_AIF_QUERY_RM_LINK_NUM_LOS</code>	<p>Queries the value of the RxMac number of loss detect in master frame Parameters: <i>(Uint32 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_RM_LINK_RCVD_</code>	<p>Queries the value of the RxMac vbus_clk tick from after pioffest that the frame boundary was recieved Parameters: <i>(Uint16 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_TM_LINK_SYNC_STATE</code>	<p>Queries the status of the AIF Tx MAC state machine Parameters: <i>(CSL_AifTxSyncState *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_TM_LINK_FIFO_OVERFLOW</code>	<p>Queries the status of the AIF Tx MAC fifo overflow Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_TM_LINK_FRAME_NOT_ALIGNED</code>	<p>Queries the status of the AIF TM link frame not aligned Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_TM_LINK_DATA_NOT_ALIGNED</code>	<p>Queries the status of the AIF TM link data not aligned Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_CB01_ALIGN_ERR</code>	<p>Queries the status of the AIF Combiner alignment error Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_CB01_OUT_FIFO_OVF</code>	<p>Queries the status of the AIF Combiner output fifo overflow Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_CB01_OUT_FIFO_UNF</code>	<p>Queries the status of the AIF Combiner output fifo underflow Parameters: <i>(Uint8 *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_AG_LINK_STATUS</code>	<p>Queries the status of the aggregator Parameters: <i>(CSL_AifAggregatorStatus *)</i> Returns: CSL_SOK</p>
<code>CSL_AIF_QUERY_DB_IN_LINK_DMA_DONE_COUNT</code>	<p>Queries input DMA done count Parameters: <i>(Uint16 *)</i> Returns: CSL_SOK</p>

<i>CSL_AIF_QUERY_DB_OUT_LINK_DMA_DONE_COUNT</i>	<p>Queries output DMA done count Parameters: (<i>Bool *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_IN_LINK_DMA_DEPTH</i>	<p>Queries number of DMA bursts written into DB by PD Parameters: (<i>Uint8 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_LINK_DMA_DEPTH</i>	<p>Queries value of queue depth in inbound packet switched FIFOs Parameters: (<i>Uint8 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_RD_PTR</i>	<p>Queries Current state of the "lowest" Read/head pointer Parameters: (<i>Uint16 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_WR_PTR</i>	<p>Queries current state of the circular write pointer of Outbound Packet Switched Memory Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_RD_INDEX</i>	<p>Queries Fifo Index 0-to-29 indicating which of 30 FIFOs has the lowest, non-read memory location Parameters: (<i>Uint8 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_DEPTH</i>	<p>Queries depth of Packet Switched Memory currently available for write Parameters: (<i>Uint16 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_NE</i>	<p>Queries Outbound packet switched FIFO 0-29, 1'b1 indicates FIFO is currently not empty Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_DB_OUT_PKTSW_FIFO_PTR_STATUS</i>	<p>Queries Packet Switched FIFO #0-29, Current state of head and tail pointer Parameters: (<i>Uint32 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_EE_INTR_VECTOR_EVT_0</i>	<p>Any link handle can be used to access the interrupt vector reg for evt 0 Parameters: (<i>CSL_BitMask32 *</i>) Returns: CSL_SOK</p>
<i>CSL_AIF_QUERY_EE_INTR_VECTOR_EVT_1</i>	<p>Any link handle can be used to access the interrupt vector reg for evt 1 Parameters: (<i>CSL_BitMask32 *</i>) Returns: CSL_SOK</p>

CSL_AIF_QUERY_EE_INTR_VECTOR_EVT_2	Any link handle can be used to access the interrupt vector reg for evt 0 Parameters: (CSL_BitMask32 *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_INTR_VECTOR_EVT_3	Any link handle can be used to access the interrupt vector reg for evt 1 Parameters: (CSL_BitMask32 *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_LINK_MASK_EVT_0	Queries masked status register for link events Parameters: (CSL_AifExcEventQueryObj *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_LINK_MASK_EVT_1	Queries masked status register for link events Parameters: (CSL_AifExcEventQueryObj *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_LINK_MASK_EVT_2	Queries masked status register for link events Parameters: (CSL_AifExcEventQueryObj *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_LINK_MASK_EVT_3	Queries masked status register for link events Parameters: (CSL_AifExcEventQueryObj *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_LINK_RAW_STATUS	Queries raw status register for link events Parameters: (CSL_AifExcEventQueryObj *) Returns: CSL_SOK
CSL_AIF_QUERY_EE_AIF_RUN_STATUS_BIT	Queries the AIF run status Parameters: (NULL) Returns: CSL_SOK

31.5 Macros

#define CSL_AIF_CD_OUT_SRC_LINK_0 (0x0)

#define CSL_AIF_CD_OUT_SRC_LINK_1 (0x1)

#define CSL_AIF_CD_OUT_SRC_LINK_2 (0x2)

#define CSL_AIF_CD_OUT_SRC_LINK_3 (0x3)

#define CSL_AIF_CD_OUT_SRC_LINK_4 (0x4)

#define CSL_AIF_CD_OUT_SRC_LINK_5 (0x5)

#define CSL_AIF_CD_OUT_SRC_CB_0 (0x6)

#define CSL_AIF_CD_OUT_SRC_CB_1 (0x7)

Source for cd out - either links 0-5 or combiner 0-1

#define CSL_AIF_CTRL_LINK_0_ENABLE (0x1)

Enable the output of combiner/decombiner for link 0

#define CSL_AIF_CTRL_LINK_1_ENABLE (0x2)

Enable the output of combiner/decombiner for link 1

#define CSL_AIF_CTRL_LINK_2_ENABLE (0x4)

Enable the output of combiner/decombiner for link 2

#define CSL_AIF_CTRL_LINK_3_ENABLE (0x8)

Enable the output of combiner/decombiner for link 3

#define CSL_AIF_CTRL_LINK_4_ENABLE (0x10)

Enable the output of combiner/decombiner for link 4

#define CSL_AIF_CTRL_LINK_5_ENABLE (0x20)

Enable the output of combiner/decombiner for link 5

#define CSL_AIF_CTRL_RX_LINK_DISABLE (0)

To enable Rx link in enable link Function

#define CSL_AIF_CTRL_RX_LINK_ENABLE (1)

To enable Rx link in enable link Function

#define CSL_AIF_CTRL_TX_LINK_DISABLE (0)

To enable Tx link in enable link Function

#define CSL_AIF_CTRL_TX_LINK_ENABLE (2)

To enable Tx link in enable link Function

#define CSL_AIF_EE_A_CI_CPRI_FSYNC_ERR (0x00100000)

This error occurs when a frame boundary is detected when the CI is not expecting it. (CPRI only)

#define CSL_AIF_EE_A_PD_CPRI_HFN_ERR (0x10000000)

Indicates CPRI hyper frame number different than FSM expected

#define CSL_AIF_EE_A_PD_FSYNC_OR_K_ERR (0x02000000)

Indicates Frame sync or K character different than FSM expected

#define CSL_AIF_EE_A_PD_OBSAI_ADR_ERR (0x04000000)

Indicates OBSAI header address look up resulted in illegal address

#define CSL_AIF_EE_A_PD_OBSAI_TYPE_ERR (0x08000000)

Indicates OBSAI header address look up resulted in illegal address

#define CSL_AIF_EE_A_PD_TIME_STAMP_ERR (0x01000000)

Indicates OBSAI time stamp error

#define CSL_AIF_EE_A_RM_8B10B_DECODE_ERR (0x00000004)

Indicates 8b10b error has occurred

#define CSL_AIF_EE_A_RM_BLK_BNDRY_DET (0x00000010)

Indicates Block boundary (OBSAI) is detected or at a Hyperframe boundary (CPRI)

#define CSL_AIF_EE_A_RM_FRM_BNDRY_DET (0x00000008)

Indicates Master frame boundary (OBSAI) is detected or at a Hyperframe boundary (CPRI) that delimits a UMTS frame

#define CSL_AIF_EE_A_RM_FRM_BNDRY_RNG_ERR (0x00000080)

This error is indicated when received Master Frame is detected outside the programmable window valid_mstr_frame_wind (typically 65ns)

#define CSL_AIF_EE_A_RM_HFNSYNC_STATE (0x00004000)

Indicates CPRI receiver in FSM State 3

#define CSL_AIF_EE_A_RM_K30P7_DET (0x00000100)

Indicates that a K30.7 character was received (OBSAI only)

#define CSL_AIF_EE_A_RM_LOF_STATE (0x00008000)

Indicates CPRI receiver entered State 0 or State 1

#define CSL_AIF_EE_A_RM_LOS_DET (0x00000002)

Indicates los_thold is reached

#define CSL_AIF_EE_A_RM_MISSING_FRM (0x00000040)

Indicates that a K28.7 (OBSAI) or a K28.5 (CPRI) character was missing and assumes receiver is in state ST3

#define CSL_AIF_EE_A_RM_MISSING_K28P5 (0x00000020)

Indicates that a K28.5 characters was missing (OBSAI only and assumes receiver is in frame sync state ST3.

#define CSL_AIF_EE_A_RM_RCVD_LOF (0x00002000)

Indicates Received lof (Z.130.0, b0) (CPRI only)

#define CSL_AIF_EE_A_RM_RCVD_LOS (0x00001000)

Indicates Received los (Z.130.0, b0) (CPRI only)

#define CSL_AIF_EE_A_RM_RCVD_RAI (0x00000400)

Indicates Received rai (Z.130.0, b0) (CPRI only)

#define CSL_AIF_EE_A_RM_RCVD_RST (0x00000200)

Indicates Received reset (Z.130.0, b0) (CPRI only)

#define CSL_AIF_EE_A_RM_RCVD_SDI (0x00000800)

Indicates Received sdi (Z.130.0, b0) (CPRI only)

#define CSL_AIF_EE_A_RM_SYNC_STAT_CHNG (0x00000001)

Indicates the status of the rx state machine

#define CSL_AIF_EE_A_SD_LOS (0x00010000)

Indicates Loss of signal condition

#define CSL_AIF_EE_B_AG_FRM_ALIGN_ERR (0x00000200)

Indicates Frame alignment error between PE and CD

#define CSL_AIF_EE_B_AG_LINK_HDR_ERR (0x00000100)

Indicates Link header error on any message slot

#define CSL_AIF_EE_B_AG_LINK_SUM_OVF (0x00000400)

Indicates Summation overflow on any AxC

#define CSL_AIF_EE_B_CD_OUT_FIFO_OVF (0x00001000)

Indicates Output FIFO overflow

#define CSL_AIF_EE_B_CD_OUT_FIFO_UNF (0x00002000)

Indicates Output FIFO underflow

#define CSL_AIF_EE_B_DB_CIRSW_IN_DMA_OFLOW (0x00010000)

Indicates Circuit Switch RAM Inbound DMA overflow

#define CSL_AIF_EE_B_DB_CIRSW_IN_DMA_UFLOW (0x00080000)

Indicates Circuit Switch RAM Inbound DMA underflow

#define CSL_AIF_EE_B_DB_CIRSW_OUT_BRST_OFLOW (0x00040000)

Indicates Circuit Switch RAM Outbound RAM burst strobe overflow

#define CSL_AIF_EE_B_DB_CIRSW_OUT_DMA_OFLOW (0x00020000)

Indicates Circuit Switch RAM Outbound DMA overflow

#define CSL_AIF_EE_B_DB_CIRSW_OUT_DMA_UFLOW (0x00100000)

Indicates Circuit Switch RAM Outbound DMA underflow

#define CSL_AIF_EE_B_PE_CWORD_K_ERR (0x01000000)

This error occurs when the CPRI control word from the DSP does not match the k28.5 character expected by the PE.

#define CSL_AIF_EE_B_TM_DATA_NOT_ALIGN (0x00000004)

This error is indicated when data to be transmitted is detected as not aligned with Master Frame + delta_offset (i.e. Frame boundary to be transmitted is not aligned with frame boundary received internally)

#define CSL_AIF_EE_B_TM_FIFO_OVF (0x00000008)

Indicates TX MAC FIFO Overflow flag

#define CSL_AIF_EE_B_TM_FRM_NOT_ALIGN (0x00000002)

This error is indicated when the frame strobe from the Frame Sync Module is not aligned with Master Frame + delta_offset.

#define CSL_AIF_EE_B_TM_SYNC_STAT_CHNG (0x00000001)

Indicates a change of tx state machine state

#define CSL_AIF_EE_CMN_CD0_ALIGN_ERR (0x00000001)

Indicates RM inputs not aligned to CD #0

#define CSL_AIF_EE_CMN_CD1_ALIGN_ERR (0x00000002)

Indicates RM inputs not aligned to CD #1

#define CSL_AIF_EE_CMN_DB_CAPT_RAM_VBUS_ERR (0x01000000)

Indicates Capture buffer, dma reading wrong/wr half of RAM

#define CSL_AIF_EE_CMN_DB_CIRSW_IN_WR_DEBUG (0x00400000)

Indicates Circuit Switched Inbound RAM VBUS write (OK for offline debug)

#define CSL_AIF_EE_CMN_DB_CIRSW_OUT_RD_DEBUG (0x00800000)

Indicates Circuit Switched Outbound RAM VBUS read (OK for offline debug and delayed streams)

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_OFLOW0 (0x00000100)

Indicates Packet Switched Inbound RAM/FIFO #0 DMA Overflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_OFLOW1 (0x00000200)

Indicates Packet Switched Inbound RAM/FIFO #1 DMA Overflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_OFLOW2 (0x00000400)

Indicates Packet Switched Inbound RAM/FIFO #2 DMA Overflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_OFLOW3 (0x00000800)

Indicates Packet Switched Error FIFO DMA Overflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_UFLOW0 (0x00001000)

Indicates Packet Switched Inbound RAM/FIFO #0 DMA Underflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_UFLOW1 (0x00002000)

Indicates Packet Switched Inbound RAM/FIFO #1 DMA Underflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_UFLOW2 (0x00004000)

Indicates Packet Switched Inbound RAM/FIFO #2 DMA Underflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_FIFO_UFLOW3 (0x00008000)

Indicates Packet Switched Inbound RAM/FIFO #3 DMA Underflow

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_RAM_RD (0x00010000)

Indicates Packet Switched Inbound RAM rd (OBSAI only; OK for CPRI)

#define CSL_AIF_EE_CMN_DB_PKTSW_IN_WR_DEBUG (0x00100000)

Indicates Packet Switched Inbound RAM VBUS write (OK for offline debug)

#define CSL_AIF_EE_CMN_DB_PKTSW_OUT_FIFO_OFLOW (0x00080000)

Indicates Packet Switched Outbound RAM/FIFO overflow write pointer wrap into read

#define CSL_AIF_EE_CMN_DB_PKTSW_OUT_MEM_LEAK (0x00040000)

Indicates Packet Switched Outbound RAM/FIFO linked list error

#define CSL_AIF_EE_CMN_DB_PKTSW_OUT_RAM_WR (0x00020000)

Indicates Packet Switched Outbound RAM wr (OBSAI only; OK for CPRI)

#define CSL_AIF_EE_CMN_DB_PKTSW_OUT_RD_DEBUG (0x00200000)

Indicates Packet Switched Outbound RAM VBUS read (OK for offline debug)

#define CSL_AIF_MAX_NUM_CONTROL_TRANSMISSION_RULES (4)

Max number of control slot transmission rule

**#define CSL_AIF_MAX_NUM_CS_TRANSMISSION_RULES
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_4X_LINK)**

Max number of circuit switched transmission rules

#define CSL_AIF_MAX_NUM_PS_TRANSMISSION_RULES (1)

Max number of packet switch data transimission rule

**#define CSL_AIF_MAX_SIZE_INBND_LINK_CS_ADDR_LUT
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_4X_LINK)**

Max size of inbound address field look-up-table for a link

#define CSL_AIF_MAX_SIZE_INBND_LINK_PS_ADDR_LUT (1024)

Max size of inbound address field look-up-table

**#define CSL_AIF_MAX_SIZE_INBOUND_REVERSE_ADDR_LUT (1 <<
CSL_AIF_OBSAI_ADDRESS_FIELD_LENGTH_BITS_PER_MSG)**

Max size of reverse address LUT mapping 13 bit OBSAI address to 10 bit AIF address

**#define CSL_AIF_MAX_SIZE_PE_TRANSMISSION_RULE_LUT
(CSL_AIF_OBSAI_NUM_SLOTS_IN_MG_WCDMA_4X_LINK)**

Max size of outbound PE transmission rule LUT

**#define CSL_AIF_MAX_SIZE_TYPE_FIELD_LUT (1 <<
CSL_AIF_OBSAI_TYPE_FIELD_LENGTH_BITS_PER_MSG)**

Max size of lookup table for type feild of OBSAI message

#define CSL_AIF_OBSAI_MAX_NUM_ADDRESS_FIELD_BITS (10)

Max number of PD address field bits

#define CSL_AIF_MAX_NUM_OF_OUTBOUND_FIFOS (5)

Max numbers of FIFOs

#define CSL_AIF_PD_ADDRESS_LUT_VALUE_INVALID (0xFFFF)

Invalid address value in address lookup table

#define CSL_AIF_PD_OBSAI_TYPE_VALUE_INACTIVE (0x0)

#define CSL_AIF_PD_OBSAI_TYPE_VALUE_CIR_SW (0x1)

#define CSL_AIF_PD_OBSAI_TYPE_VALUE_PKT_SW (0x2)

#define CSL_AIF_PD_OBSAI_TYPE_VALUE_ERR (0x4)

Type values for OBSAI messages used in PD

#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT0_1x (0x11108888)

#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT1_1x (0x42222211)

#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT2_1x (0x00084444)

Time stamp values for the link rate 1x

#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT0_2x (0x80808080)

#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT1_2x (0x02020200)

```
#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT2_2x      (0x00080202)
Time stamp values for the link rate 2x
```

```
#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT0_4x      (0x80008000)
#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT1_4x      (0x80008000)
#define CSL_AIF_PD_TIME_STAMP_INCR_NXT_LUT2_4x      (0x00080000)
Time stamp values for the link rate 4x
```

```
#define CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_20  (20)
#define CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_41  (41)
#define CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_62  (62)
#define CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_83  (83)
```

```
#define CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_40  (40)
#define CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_41  (41)
#define CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_82  (82)
#define CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_83  (83)
```

```
#define CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_80  (80)
#define CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_81  (81)
#define CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_82  (82)
#define CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_83  (83)
Position for ctrl slots for 1x,2x,4x link rates used in PE 84 CNT LUT
```

```
#define CSL_AIF_PE_1X_LINK_TIME_STAMP_MASK
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_1X_LINK - 1)
```

```
#define CSL_AIF_PE_2X_LINK_TIME_STAMP_MASK
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_2X_LINK - 1)
```

```
#define CSL_AIF_PE_4X_LINK_TIME_STAMP_MASK
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_4X_LINK - 1)
Timestamp pattern for different link rate(1X,2X and 4X) used in PE
```

```
#define CSL_AIF_PE_AGGR_CTRL_NOP              (0)
#define CSL_AIF_PE_AGGR_CTRL_INSERT          (1)
#define CSL_AIF_PE_AGGR_CTRL_ADD_15_16_BIT  (2)
#define CSL_AIF_PE_AGGR_CTRL_ADD_7_8_BIT    (3)
Values used in PE aggregator control
```

```
#define CSL_AIF_PE_CTRL_ID_LUT_INDEX_0
(CSL_AIF_OBSAI_MAX_NUM_AXC_PER_4X_LINK)
#define CSL_AIF_PE_CTRL_ID_LUT_INDEX_1
(CSL_AIF_PE_CTRL_ID_LUT_INDEX_0 + 1)
#define CSL_AIF_PE_CTRL_ID_LUT_INDEX_2
(CSL_AIF_PE_CTRL_ID_LUT_INDEX_1 + 1)
#define CSL_AIF_PE_CTRL_ID_LUT_INDEX_3
(CSL_AIF_PE_CTRL_ID_LUT_INDEX_2 + 1)
#define CSL_AIF_PE_PS_ID_LUT_INDEX
(CSL_AIF_PE_CTRL_ID_LUT_INDEX_3 + 1)
Index for accessing identity LUT used in protocol encoder
```

```
#define CSL_AIF_PE_CTRL_SLOT_POS_1X_LINK ( x )
Value:
```

```
((x == CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_20) || (x ==
    CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_41) || \
    (x == CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_62) || (x ==
    CSL_AIF_PE_1X_LINK_CTRL_SLOT_POS_83))
```

#define CSL_AIF_PE_CTRL_SLOT_POS_2X_LINK (x)

Value:

```
((x == CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_40) || (x ==
    CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_41) || \
    (x == CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_82) || (x ==
    CSL_AIF_PE_2X_LINK_CTRL_SLOT_POS_83))
```

#define CSL_AIF_PE_CTRL_SLOT_POS_4X_LINK (x)

Value:

```
((x == CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_80) || (x ==
    CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_81) || \
    (x == CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_82) || (x ==
    CSL_AIF_PE_4X_LINK_CTRL_SLOT_POS_83))
```

#define CSL_AIF_PE_INVALID_AXC_MASK (0xFFFF)

#define CSL_AIF_PE_INVALID_PS_DATA_MASK (0xFFFF)

#define CSL_AIF_PE_INVALID_COMPARE (0xFFFF)

#define CSL_AIF_PE_INVALID_ADDR (0xFFFF)

Values used in PE for populating 84 LUT

#define CSL_AIF_PE_LINK_84_EN_LUT0_ENABLE (0xFFFFFFFF)

#define CSL_AIF_PE_LINK_84_EN_LUT1_ENABLE (0xFFFFFFFF)

#define CSL_AIF_PE_LINK_84_EN_LUT2_ENABLE (0x00FFFFFF)

PE enable LUT values

#define CSL_AIF_PE_LINK_84_EN_LUT_DISABLE (0x0)

PE disable LUT values

#define CSL_AIF_PE_LINK_CNT_ID_LUT_SIZE (21)

PE link count identity LUT size

#define CSL_AIF_PE_LINK_CNT_LUT_SIZE (84)

PE link count LUT size

#define CSL_AIF_PE_TIME_STAMP_INCR_NXT_LUT_1X (0x7FFFFF)

Values used in PE time stamp to increment to next LUT at 1X rate

#define CSL_AIF_PE_TIME_STAMP_INCR_NXT_LUT_2X (0x01FF)

Values used in PE time stamp to increment to next LUT at 2X rate

#define CSL_AIF_PE_TIME_STAMP_INCR_NXT_LUT_4X (0x000F)

Values used in PE time stamp to increment to next LUT at 4X rate

#define CSL_AIF_CB_SRC_NULL (0x7)

This macro specifies null link bitfield value for combiner. NULL link is also a valid input.

#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_1X_LINK_7_15_BIT (4)

#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_2X_LINK_7_15_BIT (8)

#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_4X_LINK_7_15_BIT (16)

Max number of AxC per link

```
#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_1X_LINK_8_16_BIT      (3)
#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_2X_LINK_8_16_BIT      (7)
#define CSL_AIF_CPRI_MAX_NUM_AXC_PER_4X_LINK_8_16_BIT      (15)
Max number of AxC per link */
```

31.6 Typedefs

typedef CSL_AifLinkObj * CSL_AifHandle

typedef CSL_AifIntMemStruct * CSL_AifContext

Aif context info is a pointer.

Chapter 32

SGMII MODULE

<u>32.1 Overview</u>
<u>32.2 Functions</u>
<u>32.3 Data Structures</u>
<u>32.4 Macros</u>

32.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within SGMII module.

The SGMII is one of the components of EMAC sub system on TCI6488 DSP. The SGMII gasket must be designed to translate the (G)MII signaling native to the CP-GMAC to the SerDes interface. (CPSGMII_S_VBUSP) is used to configure the SGMII Gasket. The SGMII protocol takes a GMII data stream and converts it to a serial stream using the SerDes macro, sending the same amount of data with an embedded clock (using 8b/10b). The CPSGMII logic derives its operating clocks from the SerDes output, which has a dedicated PLL to set the link rate. The SGMII protocol also allows for dynamic switching between 10/100/1000 Mbps modes. This negotiation data is embedded in the incoming data stream from the external PHY and can happen at any time. Since the CP-SGMII logic only supports the protocol with an embedded clock, 10/100Mbps rates are supported by duplicating the data across multiple dataphases (modified by 8b/10b at the physical interface), allowing the CP-SGMII to keep the same data on the pins for the slower rates to the CP-GMAC module.

32.2 Functions

32.2.1 SGMII_clearDiagnostics

Uint32 SGMII_clearDiagnostics ()

Description

Clears diagnostics register

Return Value

Uint32

0 - Clears diagnostics register successfully.

Pre Condition

None.

Post Condition

None

Modifies

SGMII_clearDiagnostics() API modifies DIAG_CLEAR register..

Example

```
SGMII_reset();  
SGMII_clearDiagnostics ( );
```

32.2.2 SGMII_config

Uint32 SGMII_config ([SGMII_Config](#) * config)

Description

Module configuration is achieved by calling SGMII_config().

Arguments

config	Reference to struture which contains configuration for the SGMII module
--------	---

Return Value

Uint32

0 - Configure successful

SGMII_ERROR_INVALID - Invalid parameter

Pre Condition

None.

Post Condition

Module configuration is achieved.

Modifies

SGMII registers are modified.

Example

```
...
SGMII_Config SgmiiCfg;

SgmiiCfg.masterEn    = 0x0;
SgmiiCfg.loopbackEn  = 0x1;
SgmiiCfg.speed       = 0x0;
SgmiiCfg.txConfig    = 0x00000e13;
SgmiiCfg.rxConfig    = 0x00081013;
SgmiiCfg.auxConfig   = 0x0000000b;
SgmiiCfg.diagSmSel   = 0x0;
SgmiiCfg.diagEdgeSel = 0x0;
SgmiiCfg.modeOfOperation = 0x0;

SGMII_reset();
SGMII_config (&SgmiiCfg);
...
```

32.2.3 SGMII_getAnErrorStatus

Uint32 SGMII_getAnErrorStatus ()

Description

The function returns Error status of Auto negotiation process.

Return Value

Uint32

Error status of Auto negotiation process.

Pre Condition

None.

Post Condition

None

Modifies

None.

Example

```
SGMII_Config SgmiiCfg;

SgmiiCfg.masterEn    = 0x0;
SgmiiCfg.loopbackEn  = 0x1;
SgmiiCfg.speed       = 0x0;
SgmiiCfg.txConfig    = 0x00000e13;
SgmiiCfg.rxConfig    = 0x00081013;
SgmiiCfg.auxConfig   = 0x0000000b;
SgmiiCfg.diagSmSel   = 0x0;
SgmiiCfg.diagEdgeSel = 0x0;
SgmiiCfg.modeOfOperation = 0x0;
```

```
SGMII_reset();  
SGMII_config (&SgmiiCfg);  
  
SGMII_getAnErrorStatus ( );
```

32.2.4 SGMII_getLinkPartnerStatus

Uint32 SGMII_getLinkPartnerStatus ()

Description

Gets the status value of link partner.

Return Value

Uint32

SGMII_ERROR_DEVICE - auto negotiation not complete

Pre Condition

None.

Post Condition

On success returns auto negotiation status value

Modifies

None.

Example

```
SGMII_Config SgmiiCfg;  
  
SgmiiCfg.masterEn    = 0x0;  
SgmiiCfg.loopbackEn = 0x1;  
SgmiiCfg.speed       = 0x0;  
SgmiiCfg.txConfig    = 0x00000e13;  
SgmiiCfg.rxConfig    = 0x00081013;  
SgmiiCfg.auxConfig   = 0x0000000b;  
SgmiiCfg.diagSmSel   = 0x0;  
SgmiiCfg.diagEdgeSel = 0x0;  
SgmiiCfg.modeOfOperation = 0x0;  
  
SGMII_reset();  
SGMII_config (&SgmiiCfg);  
  
SGMII_getLinkPartnerStatus ( );  
...
```

32.2.5 SGMII_getStatus

Uint32 SGMII_getStatus ([SGMII_Status](#) * status)

Description

Module status is obtained by calling SGMII_getStatus().

Arguments

status Reference to an SGMII module Status Structure.

Return Value

UInt32

0 - Status is obtained successfully.

SGMII_ERROR_INVALID - Invalid parameter.

Pre Condition

None.

Post Condition

None

Modifies

Updates the argument "status"..

Example

```
...
SGMII_Status status;
SGMII_Config SgmiiCfg;

SgmiiCfg.masterEn    = 0x0;
SgmiiCfg.loopbackEn  = 0x1;
SgmiiCfg.speed       = 0x0;
SgmiiCfg.txConfig    = 0x00000e13;
SgmiiCfg.rxConfig    = 0x00081013;
SgmiiCfg.auxConfig   = 0x0000000b;
SgmiiCfg.diagSmSel   = 0x0;
SgmiiCfg.diagEdgeSel = 0x0;
SgmiiCfg.modeOfOperation = 0x0;

SGMII_reset();
SGMII_config (&SgmiiCfg);

SGMII_getStatus (&status);
...
```

32.2.6 SGMII_getStatusReg

UInt32 SGMII_getStatusReg ()

Description

The function returns the value read from STATUS register

Return Value

UInt32

Returns value read from STATUS register

Pre Condition

None.

Post Condition

None

Modifies

None.

Example

```
SGMII_reset();  
SGMII_getStatusReg ( );
```

32.2.7 SGMII_reset

Uint32 SGMII_reset ()

Description

Module reset is achieved by calling SGMII_reset().

Return Value

Uint32

Always returns a '0' if reset happens properly.

Pre Condition

None.

Post Condition

SGMII logic is reset

Modifies

Memory mapped register SOFT_RESET is modified.

Example

```
SGMII_reset ( );
```

32.3 Data Structures

32.3.1 SGMII_Config

Detailed Description

This structure is used for configuration of SGMII module

Field Documentation

UInt32 SGMII_Config::auxConfig

Auxiliary configuration to control SERDES

UInt8 SGMII_Config::diagEdgeSel

Diagnostic Hold signals Edge select

UInt8 SGMII_Config::diagSmSel

Diagnostic select

UInt8 SGMII_Config::loopbackEn

Enables loopback

UInt8 SGMII_Config::masterEn

Enables master mode

UInt8 SGMII_Config::modeOfOperation

SGMII mode of operation

UInt32 SGMII_Config::rxConfig

Receive configuration to control SERDES

UInt8 SGMII_Config::speed

SGMII operating speed 2.5MHZ or 25MHZ

UInt32 SGMII_Config::txConfig

Transmit configuration to control SERDES

32.3.2 SGMII_Status

Detailed Description

SGMII module Status Structure

Field Documentation

UInt32 SGMII_Status::auxCfgStatus

Gets the status of Transmit configuration to control SERDES

UInt16 SGMII_Status::diagStatus

Diagnostic status

UInt32 SGMII_Status::rxCfgStatus

Gets the status of Transmit configuration to control SERDES

Uint32 SGMII_Status::txCfgStatus

Gets the status of Transmit configuration to control SERDES

32.4 Macros

#define SGMII_ERROR_DEVICE 2

Device hardware error

#define SGMII_ERROR_INVALID 1

Function or calling parameter is invalid

#define SGMII_MODE_OF_OPERATION_WITH_AN 0x1

To select auto negotiation mode

#define SGMII_MODE_OF_OPERATION_WITHOUT_AN 0x2

Not to select auto negotiation mode

#define SGMII_OPERATE_SPEED_25MHZ 0x2

To select 100Mbps speed

#define SGMII_OPERATE_SPEED_2_5MHZ 0x1

To select 10Mbps speed

#define SGMII_STATUS_AN_ERROR 0x00000002

Auto negotiation status

#define SGMII_STATUS_FIB_SIG_DETECT 0x00000020

Fiber signal detect status

#define SGMII_STATUS_LINK 0x00000001

link indicator

#define SGMII_STATUS_LOCK 0x00000010

lock status

#define SGMII_STATUS_MR_AN_COMPLTE 0x00000004

Auto negotiation complete status

#define SGMII_STATUS_MR_PAGE_NEXT 0x00000008

Next page received status

Chapter 33 ECTL MODULE

<u>33.1 Overview</u>
<u>33.2 Functions</u>
<u>33.3 Data Structures</u>
<u>33.4 Macros</u>

33.1 Overview

This chapter describes the Functions, Data Structures, Enumerations and Macros within ECTL module.

The ECTL is one of the components of EMAC sub system on TCI6488 DSP. EMAC control is also known as EMAC interrupt controller. This is a wrapper kind of module combine EMAC and MDIO interrupts

33.2 Functions

33.2.1 ECTL_config

Uint32 ECTL_config ([ECTL_Config](#) * pEctlConfig)

Description

Module configuration is achieved by calling ECTL_config().

Return Value

Uint32

'0' - Configuration successful

ECTL_ERROR_INVALID - Invalid config structure

Pre Condition

None.

Post Condition

ECTL module is configured

Modifies

Memory mapped registers ECTL are modified.

Example

```
ECTL_Config pEctlConfig;

pEctlConfig.intrPaceEn = 0x1;
pEctlConfig.intrPrescale = 0x100;
...

ECTL_config (&pEctlConfig);
```

33.2.2 ECTL_getStatus

Uint32 ECTL_getStatus ([ECTL_Status](#) * pStatus)

Description

Module status is queried by calling ECTL_getStatus().

Return Value

Uint32

'0' - Read status successful

ECTL_ERROR_INVALID - Invalid status structure

Pre Condition

None.

Post Condition

ECTL module status is read

Modifies

Modifies the structure passed in as argument.

Example

```
ECTL_Config pEctlConfig;
ECTL_Status pStatus;

pEctlConfig.intrPaceEn = 0x1;
pEctlConfig.intrPrescale = 0x100;
...

ECTL_config(&pEctlConfig);
ECTL_getStatus(&pStatus);
```

33.3 Data Structures

33.3.1 ECTL_Config

Detailed Description

This structure is used for configuring the ECTL module

Field Documentation

UInt8 ECTL_Config::intrPaceEn

Enables interrupt pacing for core(0, 1 and 2)

UInt16 ECTL_Config::intrPrescale

sets the no of VBUSP_CLK periods in 4us

UInt8 ECTL_Config::miscEn[NUM_OF_CORES]

array to enable misc interrupts for core(0, 1 and 2)

UInt8 ECTL_Config::rxEn[NUM_OF_CORES]

array to enable Rx interrupts for core(0, 1 and 2)

UInt8 ECTL_Config::rxImaxLoad[NUM_OF_CORES]

Loads imax for Rx interrupts

UInt8 ECTL_Config::rxThreshEn[NUM_OF_CORES]

array to enable Rx threshold interrupts for core(0, 1 and 2)

UInt8 ECTL_Config::txEn[NUM_OF_CORES]

array to enable Tx interrupts for core(0, 1 and 2)

UInt8 ECTL_Config::txImaxLoad[NUM_OF_CORES]

Loads imax for Tx interrupts

33.3.2 ECTL_Status

Detailed Description

This structure is used to get the status values of ECTL

Field Documentation

UInt8 ECTL_Status::miscStat[NUM_OF_CORES]

Misc(STAT_PEND, HOST_PEND, MDIO_LINKINT and MDIO_USERINT) masked interrupt status

UInt8 ECTL_Status::rxImaxRead[NUM_OF_CORES]

read imax count for Rx interrupts

UInt8 ECTL_Status::rxStat[NUM_OF_CORES]

Rx masked interrupt status

UInt8 ECTL_Status::rxThreshStat[NUM_OF_CORES]

Rx thresh masked interrupt status

Uint8 ECTL_Status::txImaxRead[NUM_OF_CORES]
read imax count for Tx interrupts

Uint8 ECTL_Status::txStat[NUM_OF_CORES]
Tx masked interrupt status

33.4 Macros

#define ECTL_ERROR_INVALID 4

Function or calling parameter is invalid

#define ECTL_INTR_PACE_CORE2_TX 0x00000020

core(0, 1 and 2) interrupts pace enable macros

#define ECTL_MISC_STAT_PEND 0x00000008

core(0, 1 and 2) miscellaneous interrupts enable/status macros

#define ECTL_REGS ((CSL_EctlRegs *)CSL_ECTL_0_REGS)

ECTL module base address

#define ECTL_RX_CHA7 0x00000080

core(0, 1 and 2) Rx interrupts enable/status macros

#define ECTL_RX_THRESH_CHA7 0x00000080

core(0, 1 and 2) Rx thresh interrupts enable/status macros

#define ECTL_TX_CHA7 0x00000080

core(0, 1 and 2) Tx interrupts enable/status macros

#define NUM_OF_CORES 3

number of DSP cores on chip