Interfacing the TLV320AIC10/11 Codec to the TMS320C5402 DSP

Wendy X. Fang and Perry Miller — *AAP Data Conversion*

ABSTRACT

This report describes how the analog interface circuit (AIC) device TLV320AIC10/11, 16-bit, 22-ksps audio codec has been applied in telephony (tone generation and echo cancellation) and speech (voice security system and voice-over IP). It describes the TMS320C54xx software architecture, and the hardware that has been developed and built. The C source code developed offers good reuse capabilities.

Contents

List of Figures

List of Tables

1 Introduction

This report discusses the TMS320C54xx digital signal processor (DSP) and the analog interface circuits for interfacing an analog signal or sensor to the DSP, and to convert digital DSP–sourced signals into analog.

The TMS320C5402 DSP is a popular member of the TMS320C5000 family of fixed-point DSPs . The C5402 DSP features 16-bit resolution, 16 kword on-chip memory, up to 100 million instruction per second (MIPS) speed performance, and on-chip peripherals. The peripherals include two multichannel buffered serial ports (McBSPs), enhanced 8-bit parallel host port, two 16-bit timers, and six-channel direct memory access (DMA) controller. The TMS320C5402 DSP starter kit (DSK), one of the C54xx's development tools, is intended for DSP hardware/software designers developing a complete data acquisition system connected to a PC or a laptop computer.

A TLV320AIC10/11 device is an analog interface circuit (AIC)—also called a modem codec—that contains analog-to-digital converter (ADC) and digital-to-analog converter (DAC) paths and which interfaces with the DSP through syncronization (sync) serial bus lines. The TLV320AIC10/11 device, providing 16-bit resolution and up to 22 ksamples per second (ksps) speed, is designed for use with the TMS320C5402 DSP, or any other DSP or microprocessor that features a McBSP or sync serial peripheral interface (SPI). This general-purpose AIC device is widely used in telephony and speech applications. The TLV320AIC10/11 EVM is the codec device's evaluation tool.

This application report uses the TMS320C5402 DSK as the working platform to develop the interface to the TLV320AIC10/11 codec devices and to provide users with a hardware and software solution and some examples.

2 Hardware Interface

The hardware interface consists of the TMS320C5402 DSK (DSP starter kit) and the TLV320AIC10/11 EVM (evaluation module).

2.1 TMS320C5402 DSK

The TMS320C5402 DSK provides C5402 DSP users with a low-cost, comprehensive, stand-alone development tool. The DSK is designed specifically for digital communications applications and comes complete with a TMS320C5402-based target board, DSK-specific *Code Composer Studio* software debugger, 32K application-size-limited C compiler/assembler/linker, parallel-port interface, power supply and cables.

2.1.1 Block Diagram

Figure 1 shows a block diagram of the DSK board. The DSK board has a 100-MHz TMS3200C5402 DSP, 64K words of external SRAM, 256K words of nonvolatile flash ROM, and a complex programmable logic device (CPLD). See *TMS320VC5402 Fixed-Point Digital Signal Processor*, SPRS079 [1] for a detail description of the DSP device.

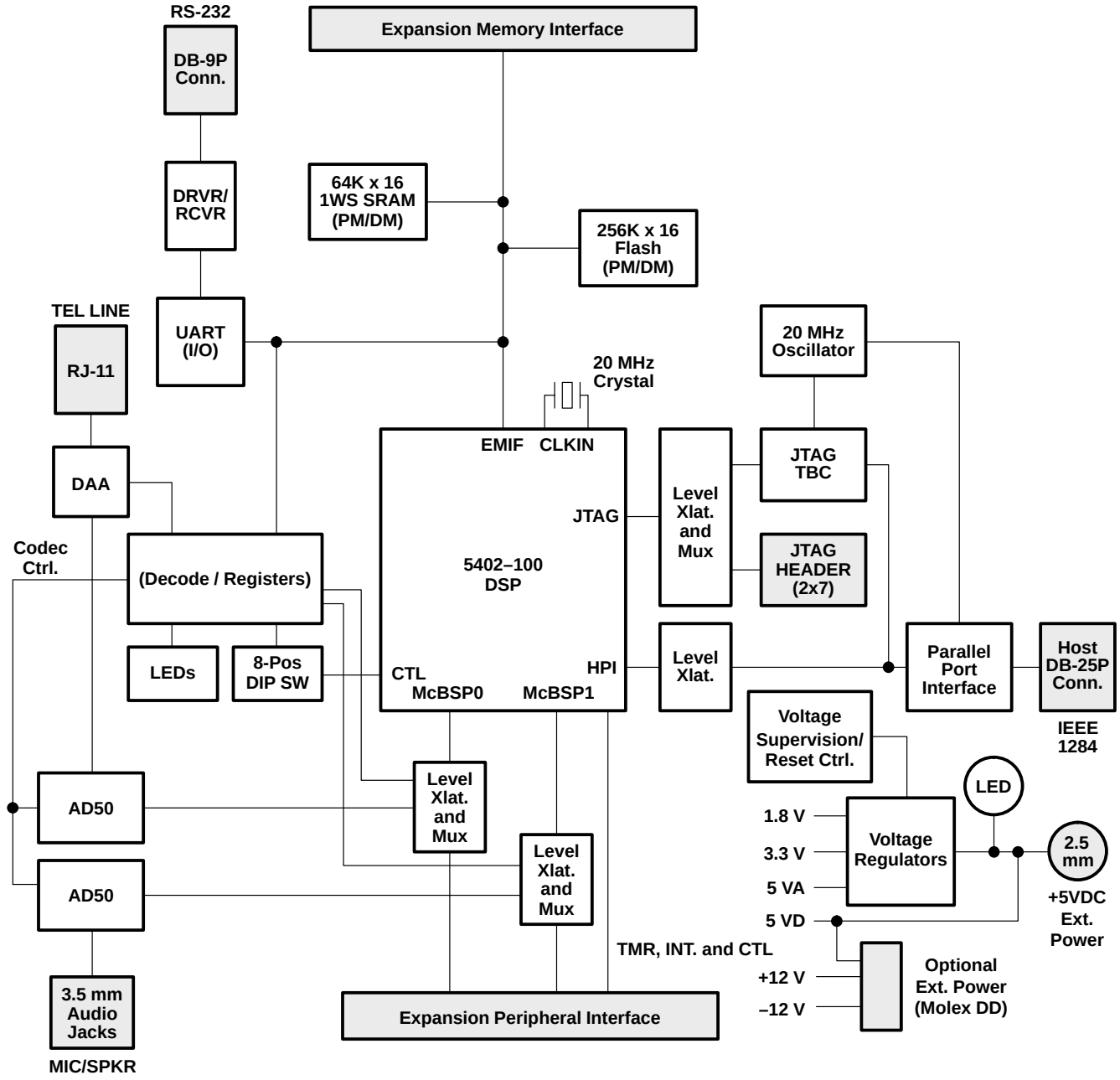


Figure 1. TMS320C5402 DSK Block Diagram

2.1.2 External Interfaces

The following external interfaces have been installed on the DSK board to enable various communication channels and to achieve a high degree of flexibility between the DSK and external interface boards (see Figure 1). These external interfaces (highlighted in Figure 1) include:

- 2.5 mm barrel-type power connector for external +5-V dc power supply
- 4-pin industry-standard Molex external power connector
- 25-pin DB-25 IEEE 1284 parallel port for JTAG/HPI access

- 14-pin JTAG emulation header
- 10-pin JTAG CPLD header
- 9-pin DB-9F RS-232 serial (UART) I/O connector
- two 3.5 mm audio jacks for microphone and speaker
- 6-pin RJ-11 modular interface connector for DAA access
- Two low-profile 80-pin daughter-board connectors: one is the expansion memory interface; the other is the expansion peripheral interface.

2.1.3 User Hardware Configuration

There is one 8-position DIP switch and 4 hardware straps provided on the DSK for the user to set up or configure the DSK, based on user options (Figure 2). Table 1 summarizes the 8 switches and their functions in both ON and OFF settings. Table 2 lists clock mode selections for the C5402 DSP. Note that an ON setting means logic low or 0; an OFF setting translates to a logic high or 1.

Table 1. 8-Position DIP Switch Description

SWITCH NO.	NAME	OFF SELECTION	ON SELECTION
1	JTAGSEL	External (e.g. XDS510PP)	Internal (test bus controller)
2	MP/MC	Microprocessor mode	Microcontroller mode
3,4,5	CLKMODE	See Table 2-1b	See Table 2-1b
6	DMSEL	External memory onboard	External memory offboard
7	USER 1	User-software defined (1)	User-software defined (0)
8	USER 0	User-software defined (1)	User-software defined (0)

Table 2. DIP Switch Controlled CLKMD Configuration

DIP SW #5 (CLKMD1)	DIP SW #4 (CLKMD2)	DIP SW #3 (CLKMD3)	CLKMD RESET DEFAULT	DSP CPU CLOCK FREQUENCY (DSK ONBOARD 20-MHz CRYSTAL)
0	0	0	E007h	x15 (not valid)
0	0	1	9007h	x10 (not valid)
0	1	0	4007h	x5 (100 MHz)
1	0	0	1007h	x2 (40 MHz)
1	1	0	F007h	x1 (20 MHz)
1	1	1	0000h	x0.5 (10 MHz)
1	0	1	F000h	x0.25 (5 MHz)
0	1	1	—	Reserved

The DSK board hardware straps provide four user options, as described in Table 3 along with the manufacturer's default strap configuration (see Figure 2 for JP1 to JP4 strap locations).

Table 3. Hardware Strap Description

NAME	DESCRIPTION	PINS 1 TO 2 FUNCTION	PINS 2 TO 3 FUNCTION	DEFAULT
JP1	CPLD program selection	CPLD program through J1 and JTAG	CPLD program through parallel port	Pins 1 to 2
JP2	Boot mode control	HPI boot selection	Normal (internal or external memory)	Pins 2 to 3
JP3	Speaker output control	Unbuffered output	Low-impedance driver output	Pin 2 to 3
JP4	DAA loop current selection	125 mA (USA) (jumper installed)	45 mA (CTR21) (jumper removed)	Jumper installed

2.1.4 System Connection and Configurations

The overall system configuration is shown in Figure 2 and the results contained in this report are based on this hardware configuration. In Figure 2, a single 5-V dc power supply was connected to a 2.5-mm barrel-type power connector; the XDS510PP emulator, installed inside the PC, was connected to the DSK's 14-pin JTAG emulation header through a JTAG adapter; and the TLV320AIC10/11 EVM communicated with the DSK through the DSK's 80-pin expansion peripheral interface connector.

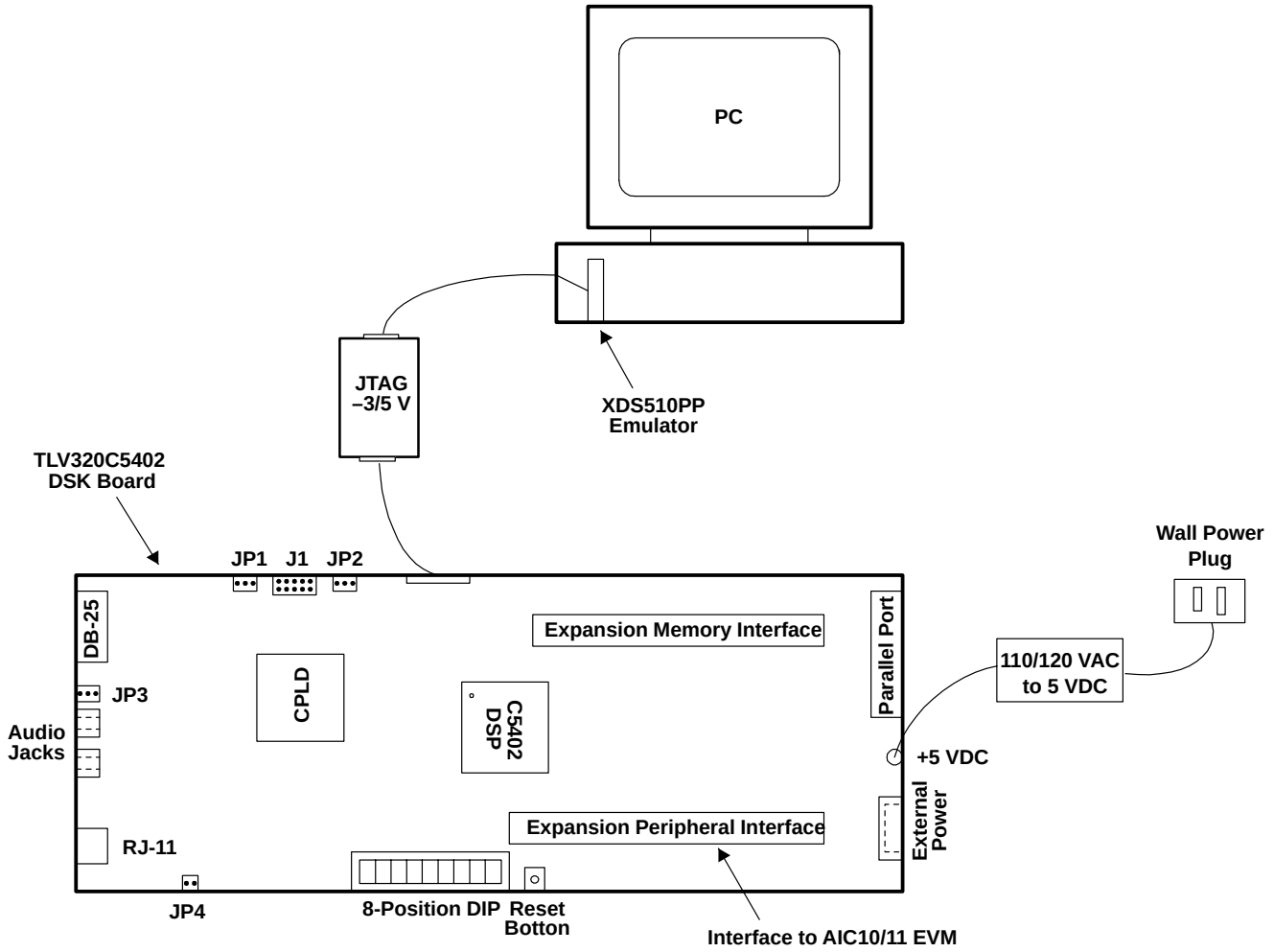


Figure 2. System Connection

Table 4 lists the C5402–DSK DIP switch settings normally used when interfacing the DSK to the TMS320AIC10/11 EVM.

Table 4. DIP Switch Configuration

SWITCH NO.	NAME	ON/OFF	DESCRIPTION
1	JTAGSEL	OFF	External (e.g. XDS510PP)
2	MP/$\overline{MC}$	ON	Microcontroller mode
3	CLKMD3	ON	The CLKMD pins are set to 0 1 0, i.e., DSP CPU clock frequency is 100 MHz (20 MHz $\times$ 5).
4	CLKMD2	OFF	
5	CLKMD1	ON	
6	DMSEL	OFF	External memory onboard
7	USER 1	ON	User-software defined (0)
8	USER 0	ON	User-software defined (0)

It is important to note that all hardware straps were kept at the default configuration of Table 3.

2.2 TMS320AIC10/11 EVM

The TLV320AIC10/11 EVM is designed to be a daughterboard that plugs directly into the expansion peripheral connector of the TMS320C5402 DSK. This is illustrated in Figure 2 where it can be seen that there are two 80–pin interface connectors provided on the DSK. The EVM has a footprint for a total of eight AIC10/11 devices, but only two TLV320AIC10/11 devices are populated and supplied with the EVM. With only two codec (AIC10/11) devices on board, the EVM can be used successfully to evaluate the various operating modes of the codec devices.

2.2.1 Block Diagram

Figure 3 shows the EVM block diagram. The nonpopulated AIC10/11 devices are shown within the dash-dot squares.

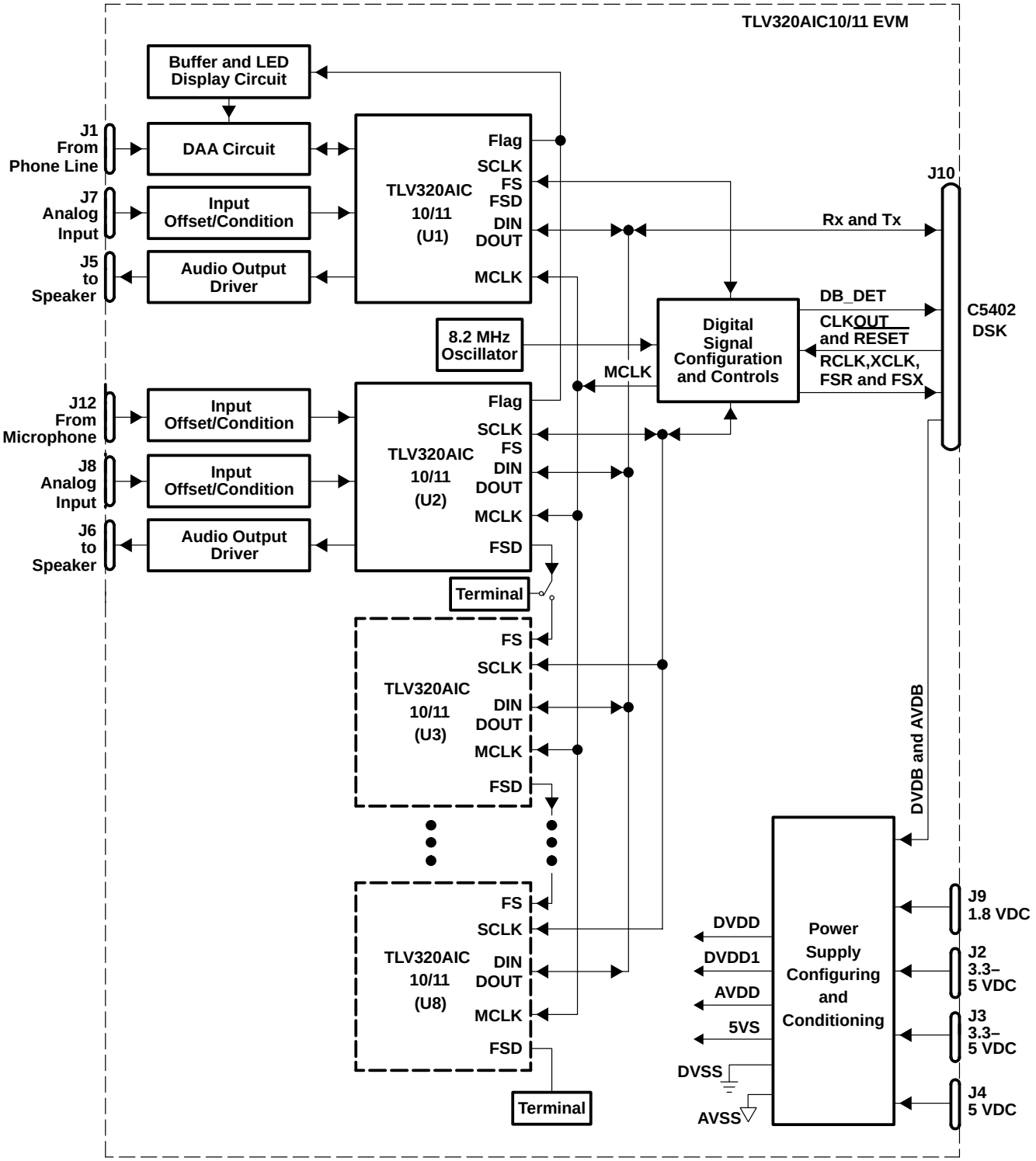


Figure 3. TLV320AIC10/11 EVM Block Diagram

2.2.2 Analog Interfaces

Not including the power supply, the EVM interfaces to external analog signals via six connectors or jacks. Two analog inputs and one analog output can be connected to each of the two populated AIC10/11 devices. The two analog inputs are multiplexed to the AIC10/11 device's ADC path, and the multiplexing selection is controlled by software. Table 5 lists the six analog interfaces and their descriptions.

Table 5. TLV320AIC10/11 EVM Analog Interfaces

CONNECTOR		CONNECTOR TYPE	DESCRIPTION
AIC10/11 #1	J1	Input: 6-pin RJ-11 modular interface connector	DAA access
	J7	Input: 3.5-mm audio jack	Analog input
	J5	Output: A pair of wires	Speaker output
AIC10/11 #2	J12	Input: 3.5-mm audio jack	Microphone input
	J8	Input: 3.5-mm audio jack	Analog input
	J6	Output: A pair of wires	Speaker output

2.2.3 Digital Interface

An 80–pin connector (J10) links the digital I/O interfacing signals between the C5402 DSK and the AIC10/11 EVM board. Connector J10 is directly plugged into the expansion peripheral interface of the DSK and, as a result, there is essentially clean bussing of the I/O signals travelling between the two boards. Table 6 lists the signals that interface between the EVM and DSK through the hardware connection, J10.

Table 6. EVM Motherboard Peripheral Connector Pinout

J10 PIN NO.	DESCRIPTION
1	+12 V dc
2	–12V dc
3, 4	Digital ground
5	+5.0 V dc, digital
6	+5.0 V dc, analog
7, 8	Digital ground
9	+5.0 V dc, digital
10–18	Not used
19	+3.3 V dc, digital
20	+3.3 V dc, analog
21	X_CLKX0, McBSP0 Tx clock
22	Not used
23	X_FSX0, McBSP0 Tx frame sync
24	X_DX0, McBSP0 Tx data
25, 26	Digital ground
27	X_CLKR0, McBSP0 Rx clock
28	Not used
29	X_FSR0, McBSP0 Rx frame sync
30	X_DR0, McBSP0 Rx data
31, 32	Digital ground
33	X_CLKX1, McBSP1 Tx clock
34	Not used
35	X_FSX1, McBSP1 Tx frame sync
36	X_DX1, McBSP1 Tx data
37, 38	Digital ground
39	X_CLKR1, McBSP1 Rx clock
40	Not used
41	X_FSR1, McBSP1 Rx frame sync
42	X_DR1, McBSP1 Rx data
43, 44	Digital ground
45–50	Not used
51, 52	Digital ground
53–58	Not used
59	X_/RESET, Reset signal from motherboard
60	Not used
61, 62	Digital ground
63– 74	Not used
75	DB_DET, Daughter board detector
76, 77	Digital ground
78	X_CLKOUT, DSP CLKOUT Pin output signal = 1/2 CPU Frequency
79, 80	Digital ground

2.2.4 System Connection and Configurations

The EVM daughter board is mounted on top of a DSK board by plugging the EVM's digital interface connector, J10, into the expansion-peripheral interface of the DSK. Another 80-pin connector, J15, is plugged into the expansion-memory interface, but J15 is not wired and only serves as a hardware support for the EVM board.

In the work done for this application report, the EVM system ran with the manufacturer's default configurations, i.e., there was a total of 2 AIC10/11 devices on the board. These 2 devices worked in parallel and cascade modes, with AIC10/11 number 1 connected as the master device and AIC10/11 number 2 connected as the slave (default condition). In this case, the serial-interface sync mode of the codec is set to the default mode, which is pulse mode. See *TLV320AIC10/11 EVM User's Guide*, SLWU003C [2] for the manufacturer's defaults.

One microphone and two speakers were connected into the analog interfaces J12, J5, and J6, respectively.

Three dc power supplies were used for the EVM board (see the EVM block diagram in Figure 3). Digital power at 3.3-V dc was connected to J2 because the codec devices used for this application report test were AIC10s. In addition, 3.3-V dc analog power was connected to J3, and +5-V dc analog power to J4.

3 Software Interface

For communication between the DSP on the DSK board and the AIC10 devices on the EVM board, a set of software drivers was developed, including initialization (software configuration) of the DSP's McBSP, CPLD registers, and AIC10/11 control registers. Sample code for ADC data receive (Rx) and DAC data transmit (Tx) functions are provided in this report. For convenience, a software structure or skeleton was implemented that can be used for a variety of applications. Refer to *TMS320C54x DSP Reference Set, Volume 4: Applications Guide*, SPRU173 [3] for a variety of applications for this device.

In this report, several application examples are also presented, and users can either adapt these examples to their application or replace them with completely new C-language code. The driver, the Rx/Tx example, and the software structure are included in the appendices.

3.1 DSP Initialization

DSP initialization configures the DSP system by setting up its memory-mapped registers (MMRs) which are mapped into the beginning of the DSP's data-memory page. For functions and descriptions of all of the MMRs, see *TMS320C54x DSP CPU and Peripherals, Reference Set Volume 1,* SPRU131F [4]. The MMRs are defined in Appendix E.

3.1.1 DSP System Clock Frequency

A 20-MHz crystal oscillator on the DSK board provides the C5402 DSP with basic frequency reference (see Figure 1). The DSP system clock can be configured to different frequencies using hardware or software. The DSP's three clock-mode pins (CLKMD1, CLKMD2, and CLKMD3) are designed for hardware configuration, which can be done on the DSK via DIP switch settings (refer to Table 1). At power up, the default DSP system-clock frequency depends on the hardware-clock mode configuration. Using the DIP switch configuration shown in Table 4, this frequency is 100 MHz. After power up, the user can use software to change to another system frequency, such as 40 MHz, 20 MHz, 10 MHz, or 5 MHz (see Table 2). Changing the value at the MMR CLKM changes the DSP system clock frequency. Note that CLKMD has to be brought into divider mode (CLKMD=0000h) before rewriting a new value. For example, if the DSP system clock is to be changed from 100 MHz to 40 MHz after power up, that task can be executed by the following code lines (at power up, the data in CLKMD are 0×4007, i.e., a system clock frequency of 100 MHz):

```
            STM     #0, CLKMD           ; switch to divider mode
TestStatus  LDM     CLKMD, A            ; test clock mode status
            AND     #0x0001, A          ; mask out PLL status bit
            BC      TestStatus, ANEQ    ; wait for clock to divider mode
            STM     #0x1007, CLKMD      ; switch to PLLx2 (= 40 MHz) mode
```

See *TMS320C54x DSP Reference Set, Volume 2:Mnemonic Instruction Set*, SPRU172B [5] for assistance in developing other code.

3.1.2 Software Loop Control

Timer0 of the DSP is used in this application report to control the software's main loop so that a fixed-rate routine can be repeated. This provides basic timing for the application software system and simplifies code development for the majority of applications. For example, with a 100-MHz system clock, a 16-kHz main application loop can be obtained in several ways, such as by setting the Timer0 divide-down ratio MMR, TDDR, to 0 (zero) and the period MMR, PRD, to 6249, (which gives 100MHz/(0+1)/(6249+1) = 16 kHz); or by setting TDDR to 4 and PRD to 1249 (which gives 100MHz/5/1250 = 16 kHz); or setting TDDR and PRD to other combinations that result in (TDDR+1)×(PRD+1) = 6250. In each instance, the routines within the loop are repeated at exactly 16 kHz.

On initialization, Timer0 is stopped, and the MMR registers for Timer0 are set up. After all initial configurations, and just before the program enters its main repeating loop, Timer0 is started. Timer0 begins to count down while the driver routine is running. After executing the application routine(s), the driver checks the Timer0 interrupt flag. Note that even if the Timer0 interrupt is disabled, its interrupt flag will still be set when the Timer0 counter reaches 0, forcing a return to the top of the code. Figure 4 illustrates the main loop control routine. The code listing for this routine can be found in Appendix A; also see *TMS320C54x Optimizing C Compiler*, SPRU103D [6].

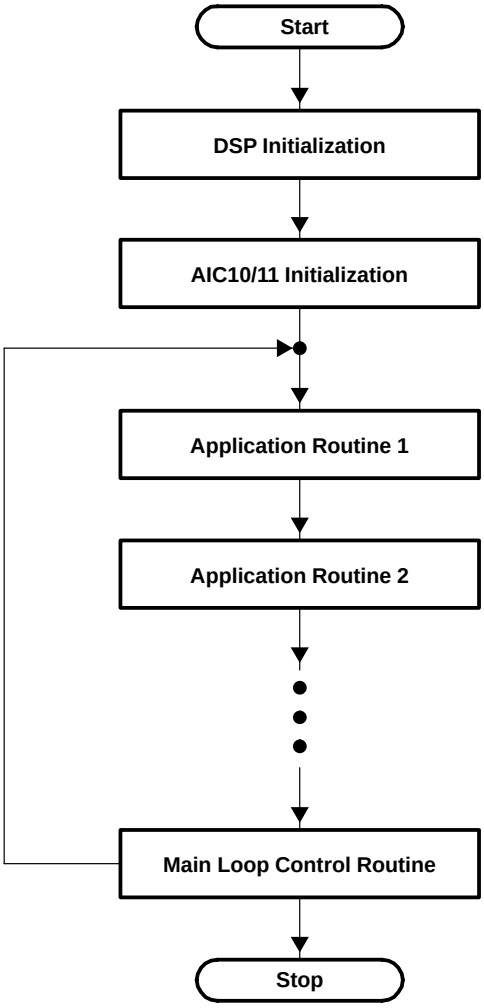


Figure 4. Software Main Loop Flow Chart

3.1.3 McBSP Initialization

McBSP is a high-speed, full-duplex serial port. Its triple-buffered input and double-buffered output-data registers allow a continuous data stream and independent framing and clocking for receiving and transmitting. With correct configuration, the McBSP can interface gluelessly with other McBSPs or other devices, such as AIC10/11 codec.

There are two McBSPs on a C5402 DSP, namely McBSP0 and McBSP1. In this application report, McBSP0 is used as the interface for initializing and communicating between the DSP and all AIC10 devices. McBSP1 is used only for the direct configuration serial interface (DCSI) that configures the AIC10/11 devices via the DCSI pin of an AIC10/11.

On the MMR memory section located on the DSP's data page, there are six 16-bit words that correspond to a McBSP. All McBSP control and status registers are within a subbank. Table 7 lists the MMRs of a McBSP. The data registers DRR1 and DXR1 are used for reading and writing data that are received from and transmitted to AIC10 devices, respectively. When the Rx/Tx data length exceeds 16 bits, DRR1 and DXR1 are supplemented with DRR2 and DXR2, respectively. The MMR registers SPSA and SPSD are used to access the McBSP's subbank registers. SPSA is used to load the subaddress of a McBSP register; the register SPSD contains the value of the McBSP's subbank register that possesses the address shown by SPSA. To configure a McBSP register in the subbank, first its subaddress is written into register SPSA, and then SPSD becomes the subbank read/write register.

Table 7. McBSP Registers

ADDRESS AT DATA MEMORY		SUBADDRESS	ACRONYM	DESCRIPTION
McBSP0	McBSP1			
0x0020	0x0040	-	DRR2	Data receive register 2
0x0021	0x0041	-	DRR1	Data receive register 1
0x0022	0x0042	-	DXR2	Data transmit register 2
0x0023	0x0043	-	DXR1	Data transmit register 1
0x0038	0x0048	-	SPSA	Serial port sub-bank address register
0x0039	0x0049	-	SPSD	Serial port sub-bank data register
		0x0000	SPSCR1	Serial port control register 1
		0x0001	SPSCR2	Serial port control register 2
		0x0002	RCR1	Receive control register 1
		0x0003	RCR2	Receive control register 2
		0x0004	XCR1	Transmit control register 1
		0x0005	XCR2	Transmit control register 2
		0x0006	SRGR1	Sample rate generator register 2
		0x0007	SRGR2	Sample rate generator register 1
		0x0008	MCR1	Multi-channel register 1
		0x0009	MCR2	Multi-channel register 2
		0x000A	RCERA	Receive-channel enable register partition A
		0x000B	RCERB	Receive-channel enable register partition B
		0x000C	XCERA	Transmit-channel enable register partition A
		0x000D	XCERB	Transmit-channel enable register partition B
		0x000E	PCR	McBSP pin-control register

For this report, both McBSPs work as slaves; therefore, the system communication clock and frame signals come from an external device, i.e., from a master AIC10. Also, both McBSPs do not work in multichannel mode. Consequently, the configuration of a McBSP involves only the following McBSP subbank registers: SPSCR1, SPSCR2, RCR1 RCR2, XCR1, XCR2, and PCR. For the bit definitions of the McBSP subbank registers, refer to *TMS320C54x Enhanced Peripherals, Reference Set Volume 2,* SPRU302 [7].

In this report, both receive and transmit data lengths are set to 16 bits and are left-justified; their frame lengths are 1 word per frame, and their interrupts, RINT and XINT, are generated by a new-frame sync signal (namely, the FS from the master AIC10 device).

There are seven hardware pins (described in Table 8) on the DSP that come from a McBSP. Setting the MMR's PCR configures the mode, direction and polarity of these pins. In this report, the pins DR/X, FSR/X and CLKR/X function as a serial port—FSR/X and CLKR/X are driven by a master AIC10 device; FSR/X are active high; and CLKR/X are active on the signal's rising edge.

Table 8. McBSP Hardware Pins

PIN	DIRECTION	DESCRIPTION
DR	Input	Received serial data
DX	Output/high-impedance	Transmitted serial data
FSR	Input/output/high-impedance	Receive frame synchronization
FSX	Input/output/high-impedance	Transmit frame synchronization
CLKR	Input/output/high-impedance	Receive clock
CLKX	Input/output/high-impedance	Transmit clock
CLKS	Input	External McBSP system clock

The McBSP initialization assembler code used for this report can be found in Appendix B. See also *TMS320C54x Assembly Language Tools*, SPRU102D [8] and *TMS320C54x Reference Set, Volume 2: Mnemonic Instruction Set*, SPRU172B [5]. Figure 5 shows the flow chart for the McBSP initialization.

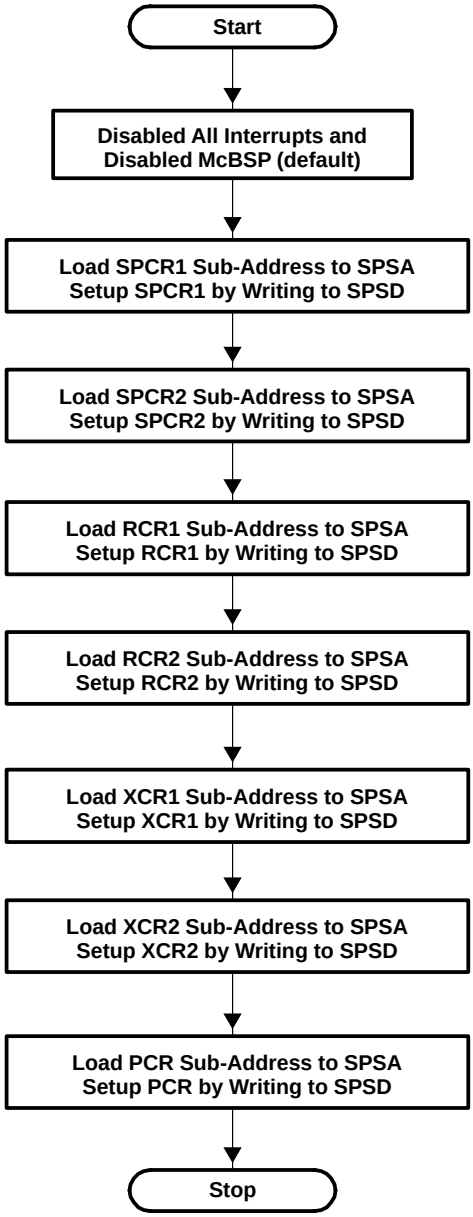


Figure 5. McBSP Initialization Flow Chart

3.1.4 Interrupts

Interrupts are hardware- or software-driven signals that cause the central processing unit (CPU) in the DSP to suspend its main program and execute an interrupt service routine (ISR). There are 30 interrupts in C5402 DSP. Among the 30 interrupts, two of them(RESET and NMI) are non-maskable, 14 are software interrupts, and 14 are peripheral interrupts. The interrupt-vector table given in Appendix F lists all the interrupts. Note that each vector occupies four words of space.

By default, the C5402 interrupt table is placed on the DSP's program-memory page with the start address 0xFF80. The table, however, can be remapped into any 128-word memory block in the program page (except the reserved areas). The interrupt vector pointer (IPTR) in CPU points to the most significant nine bits of the start address of the interrupt vector table. IPTR is mapped into the high nine bits of the processor mode status register (PMST) of MMRs. At power up, IPTR is always 0x1FF, indicating that the default address for the vector table is 0xFF80.

In this report, the interrupt vector table is remapped to address 0x0080 in the program memory. Therefore, the PMST register is initialized to 0x00A0, and the most significant nine bits are 000000001b (or equivalently, IPTR equals 0x001), which points to the address 0x0080.

All maskable interrupts are disabled during software initialization by setting the INTM bit to 1 in the ST1 register of the MMRs. All peripheral interrupts are also disabled during initialization. When a peripheral condition causes an interrupt, the corresponding interrupt flag is set and, if the process is in the idle state, it awakens; however, no ISR is executed by the CPU if the peripheral interrupt is disabled, or the INTM bit is already set to 1.

Two peripheral ISRs have been developed: one corresponds to the McBSP0 data-receive interrupt BRINT0; the other corresponds to the McBSP1 data-transmit interrupt BXINT1. The ISR for BRINT0 is designed to download AIC10 ADC data from the McBSP0 receive-data register, DRR1, and to upload digital data, processed or generated by the DSP, into the McBSP0 transmit data register, DXR1, so as to output the processed data into the AIC10's DAC. The ISR for DXINT1 is used to configure the AIC10/11 control registers through DCSI.

The interrupt service routines are listed in Appendix D. They are called if the INTM bit in ST1 is reset (CPU interrupt enabled) and the corresponding peripheral interrupts are unmasked.

3.2 CPLD Register Initialization

The C5402 DSK uses a complex programmable logic device (CPLD) to implement the required logic and to provide control and status interfaces for the DSP software. The CPLD includes seven control and status registers that are memory mapped into the DSP's lower I/O memory space, starting at address 0x0000. Table 9 lists these CPLD registers.

Table 9. CPLD Registers

ADDRESS AT DSP I/O SPACE	ACRONYM	DESCRIPTION
0x0000	CNTL1	Control register 1
0x0001	STAT	Status register
0x0002	DMCNTL	Data memory control register
0x0003	DBIO	Daughter board general purpose IO
0x0004	CNTL2	Control register 2
0x0005	SEM0	Semaphore 0
0x0006	SEM1	Semaphore 1

Only CNTL2 is used for interfacing with the EVM board; it allows the software to control the source of data for both McBSPs. The default data source for the McBSPs is the DSK board itself. To permit communication between the McBSPs and AIC10 devices on the EVM board, the data source for McBSP0 and McBSP1 needs to be configured so that the McBSPs interface with data sourced from the DSK's daughter board, i.e., from the EVM board.

Table 10 presents the bit definitions of the register CNTL2. For selecting the EVM board as data source, the 0 and 1 bits in CNTL2 are set to 1. The configuration-code listing is on the first page of Appendix C. The DSK and EVM connection routine must be executed before the DSP can access any of the AIC10 devices.

Table 10. CPLD Control Register 2 (CNTL2) Definition

BIT NO.	ACRONYM	R/W	DESCRIPTION
7	DAAOH	RW	DAA off-hook control
6	DAACID	RW	DAA caller ID enable
5	$\overline{\text{FLASHENB}}$	RW	External memory source selection (0 = Flash; 1 = SRAM)
4	$\overline{\text{INT1SEL}}$	RW	INT1 interrupt source selection (0 = UART; 1 = daughter board)
3	FC1CON	RW	Mic/speaker AD50 FC control
2	FC0CON	RW	DAA AD50 FC control
1	BSPSEL1	RW	McBSP1 data source selection (0 = Mic/speaker; 1 = daughter board)
0	BSPSEL0	RW	McBSP0 data source selection (0 = DAA; 1 = daughter board)

3.3 AIC10/11 Control Registers Initialization

In an AIC10/11 device,there are four control registers that permit the user to select and control the ADC and DAC sampling frequencies, and other devices and circuits within the AIC10/11. For the definitions of these registers, see *General-Purpose 3-V to 5.5-V 16-bit 22-ksps DSP Codec, TLV320AIC10*, SLWS093D [9] and *General-Purpose Low-Voltage 1.1-V to 3.6-V I/O 16-bit 22-ksps DSP Codec TLV320AIC11*, SLWS100 [10].

3.3.1 ADC/DAC Sampling Frequency

The sampling frequency, F_S, of an AIC10 device is:

$$F_S = \frac{f_{MCLK}}{256 \times N} \quad (1)$$

where f_{MCLK} is the master clock frequency and N is the frequency divider, which is an integer from 1 to 32, set at the AIC10/11's control registers according to the user's application requirement. The default value of N at each power up is 32, which brings the system to its slowest sample rate under the master clock (MCLK). There are two clock sources in the DSK/EVM system that can be used as the MCLK. One is from the EVM onboard crystal oscillator and the other is from its motherboard, the CLKOUT signal of the DSP. Of course, the user can bring in an external clock source as the MCLK. Note that the maximum MCLK frequency for the TLV320AIC10/11 is 40 MHz, or 15 MHz if the divider, N, is an odd number. For example, if the 8.2 MHz oscillator on the EVM board is used as the MCLK and N is 2, a 16-kHz sampling rate is obtained. As another example, if the CLKOUT from the motherboard is used as the MCLK and the CPU is running at 40 MHz (CLKOUT is at 20 MHz), the user can set N to 4 and obtain a 19.53-kHz sampling rate.

3.3.2 Communication Cycle and Phase

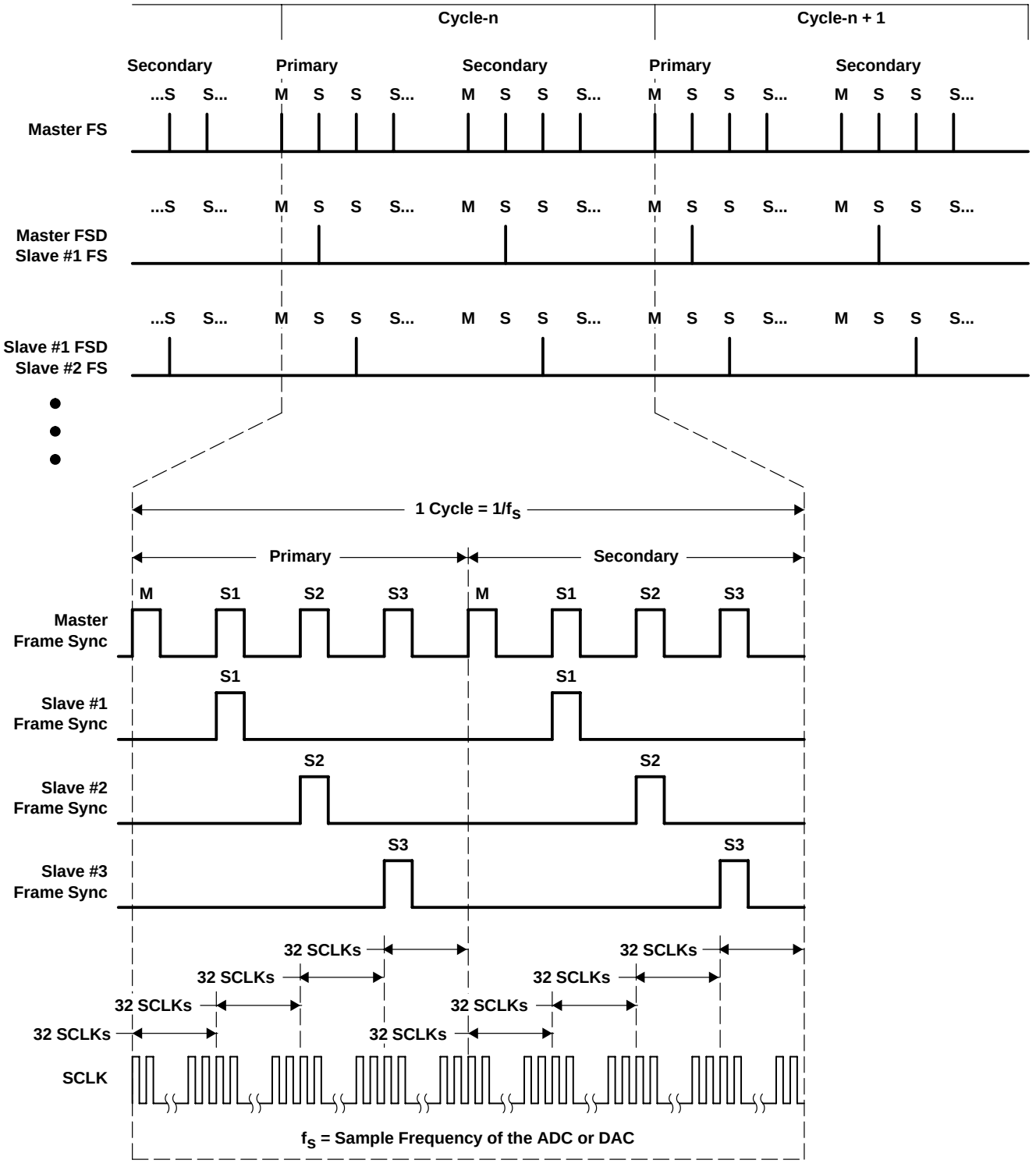
The inverse of the sample frequency, $1/F_S$, is called a *communication cycle*, which is the interval between two consecutive ADC or DAC samples. There are generally two communication phases in each communication cycle, called the primary and the secondary communication phase, respectively. During the primary phase, ADC and DAC data are received/transmitted between the McBSP0 and the AIC10/11 devices; during the secondary phase, the AIC10's control registers are read or written for checking the AIC10/11 devices' status, or for configuring the AIC10/11 devices.

The primary communication phase occurs during *every* communication cycle. The secondary phase occurs only if the request for the secondary communication has been sent during the primary phase (or by the hardware signal, FC); otherwise, time elapses but no secondary communication FS signal is generated from the master, and no data receive/transmit takes place during the entire secondary phase.

3.3.3 Sync Communication Timing

In Figure 3, it can be seen that the data Rx/Tx lines are connected from the DSP to all AIC10/11 devices. Thus, the frame-sync signal (FS) is essential for the DSP to set up an identity between the data in its DRR1 or DXR1 registers and the particular AIC10/11 device associated with the data.

A master AIC10/11 device controls the communication timing and generates the frame sync signal (FS) and the shift clock (SCLK). The SCLK signal from the master goes to the DSP and to all other slave AIC devices to set up their data-bit shift rates. The FS signal from the master goes only to the DSP and is used to set a flag in McBSP0 that indicates when the new Rx/Tx data have been written into its DRR1 and/or read from its DXR1. The FSD of the master, which is delayed by 32 SCLK ticks (or an FS) from the FS pulse for the master itself, is output to the next slave AIC10/11 device. A driven slave AIC10/11 device receives its FS either from the FSD pin of the master, or from the slave that immediately precedes it in the cascade chain. This driven slave then delays the FS 32-SCLK and outputs the pulse from its FSD. A master AIC10/11 outputs M frame sync (FS) pulses to the DSP during a communication phase (where M symbolizes the total number of AIC10 devices in the cascade chain), while a slave AIC10/11 device gets a single FS signal in its primary communication phase and gets either one FS or no FS at all, in its secondary phase. Figure 6 illustrates the frame sync timing for four cascade-connected AIC10/11 devices.



NOTES: 1. In Master FS there are 32 SCLKs between a master/slave frame and a slave/slave frame.
2. There are 256 (1 to 4'AIC10s on board) or 512 (5 to 8'AIC10s on board) SCLK pulses in each communication cycle (also called ADC/DAC sample interval), in which half (128 or 256) is for the primary phase and half is for the secondary phase.
3. The secondary communication phase occurs only if required in the primary one.

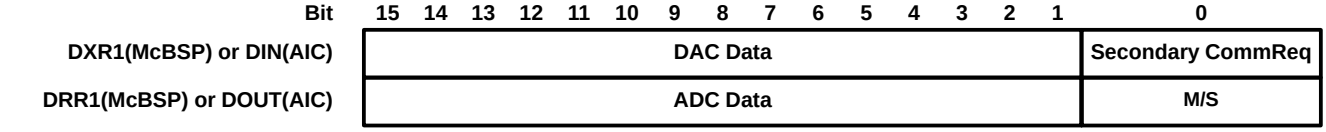
Figure 6. Cascade Paralleling TLV320AIC10 Master-Slave Frame Sync Timing Diagram

3.3.4 Interface Data Format

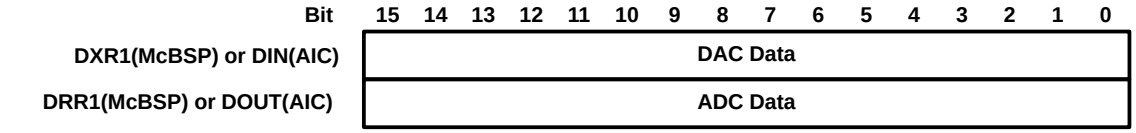
As already discussed, the data format of a McBSP is 16 bits and the 16-bit data Rx/Tx occurs at every frame sync (FS) signal. There are different data formats for transmission across the interface, depending on the data mode and communication phases. Figure 7 illustrates the McBSP and AIC10/11 interface data format.

Primary Communication Format:

(AIC Device at 15-Bit Data Mode)

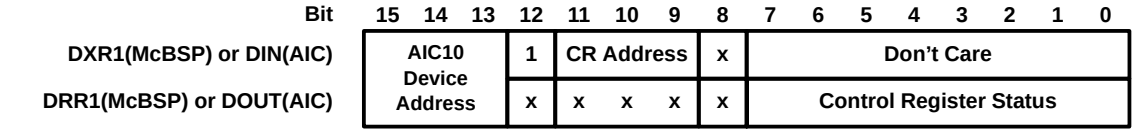


(AIC Device at 16-Bit Data Mode)

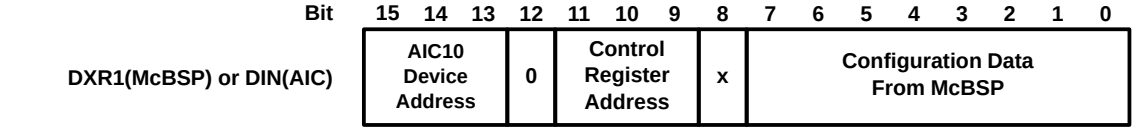


Secondary Communication Format:

(McBSP Request Reading From CR)



(McBSP Write to CR)



(McBSP Write to CR Through DCSI)

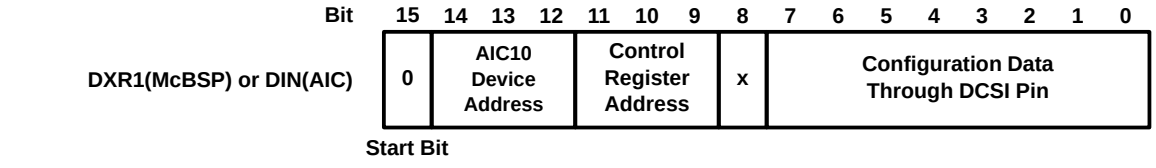


Figure 7. Interface Data Format

If the AIC10/11 is configured for 15-bit data mode, during the primary communication phase the McBSP not only receives and transmits data with an AIC10/11 device, but it also passes the AIC10/11's mode status (master vs slave) to the McBSP and sends the AIC10//11 device's secondary communication request. Conversely, in 16-bit data mode, the primary phase only receives and transmits the ADC/DAC data—the secondary communication has to be requested by a hardware FC signal.

The secondary communication is needed for reading and writing the control registers of the AIC10/11 devices. Besides the secondary communication, there is also another way to configure the AIC10/11, using the DCSI pin to input the AIC10/11 configuration data directly. The DCSI data format is also given in Figure 7. The listing of an example routine for AIC10/11 configuration via the DCSI port can be found in Appendix D.

3.3.5 Hardware Configuration Identification

For the DSP to identify the source of the data in its DRR1 and DXR1 registers, both the number of AIC10/11 devices on the EVM board and the position of the device that has just interfaced with DSP must be known to the DSP. This report applies a plug-and-play algorithm that automatically identifies this information so that the AIC10/11 configuration routine can be reused without change or, at worst, with only small changes for different numbers and hardware configurations of AIC10/11 devices on board. The method is explained further in *Hardware Auto-Identification and Software Auto-Configuration for the TLV320AIC10 DSP CODECs—A Plug-and-Play Algorithm*, SLYT023 [11].

3.3.6 AIC10/11 Control Register Configuration

As already mentioned, there are four control registers (CRs) in each of the AIC10/11 devices. To configure the control registers, a total of four full communication cycles are needed by the McBSP, with each cycle having a primary and secondary phase. Figure 8 shows the timing diagram of the master FS for configuring 3 AIC10/11 devices—1 master and 2 slaves.

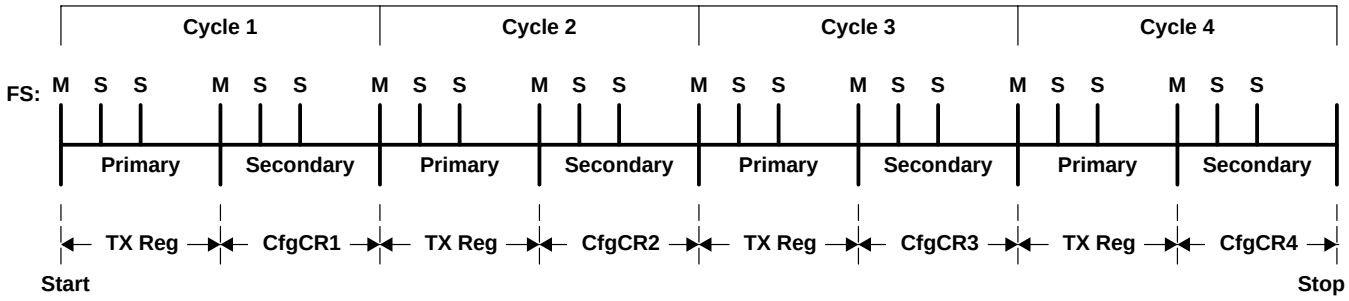


Figure 8. Timing Diagram of Autoconfiguration Procedure

In the primary phase of a cycle, the request for the secondary communication is sent, i.e., DXR1 (= 0x0001) is sent from McBSP0 to each (of the M) AIC10/11 devices. Therefore, at each master FS pulse, the send must be repeated M times and must start with the master AIC10/11 device. During the secondary communication phase, at each master FS pulse, an AIC10/11 control register is configured or written. It can also be see from Figure 8 that the control registers are configured in the following order:

in cycle 1: master CR1, slave1 CR1, slave2 CR1;
in cycle 2: master CR2, slave1 CR2, slave2 CR2;

in cycle 3: master CR3, slave1 CR3, slave2 CR3; and
in cycle 4: master CR4, slave1 CR4, slave2 CR4.

The code for configuring M cascaded AIC10/11 devices, where M = 1, 2, …, 8, is contained in the code listing of Appendix C.

3.3.7 AIC10/11 Control Register Reading

In many cases, an application must check the status of the AIC10/11 devices by reading their control registers. Appendix C contains a code example for reading all of the control registers from a master and a slave AIC10/11 device.

3.4 Data Receive (ADC) and Transmit (DAC) Programs

The major function of an AIC10/11 device is to convert analog signals into digital data for the DSP (ADC) and to convert digital data processed by the DSP back into an analog signal (DAC). The interface between the McBSP0 of the DSP on the DSK board and the AIC10/11 devices on the EVM board has been built using the hardware and software configurations already described. During the DSP's normal operation, a software routine is needed to download ADC data and upload DAC data from the McBSP0's registers DRR1 and DXR1, at the same frequency as the sampling rate, e.g., at 16 kHz.

To download the ADC data for each and every sample, the receive interrupt, RINT0, of the McBSP0 must be enabled. Also, by enabling both receive and transmit functions of the McBSP0, a data receive (Rx) and data transmit (Tx) occur at every RINT0 interrupt, permitting the ADC data to be downloaded from an AIC10/11 device through McBSP0's DRR1 register and digital data to be uploaded to the McBSP0's register DXR1. The data in DXR1 are transmitted to an ACI10/11 device, where they are further converted into an analog signal.

In this report, the RINT0 interrupt service routine (ISR) first identifies which of the two AIC10 devices has just finished its ADC. Then it downloads the data into the corresponding ADC data memory and uploads the digital data to the McBSP0's DXR1 register for transmitting to the corresponding AIC10 device. Figure 3.4-1is the ISR flowchart. For the master AIC10, the analog input from J7 is downloaded to the DSP and the DSP data (the secured voice SOUT) are uploaded so as to output to the speaker connected through J5. For the slave AIC10, a microphone input from J12 is downloaded to the DSP, and other DSP data (the DTMF tone, called a *ToneWave*) are uploaded and output to the speaker connected through J6. The code listing is in Appendix D.

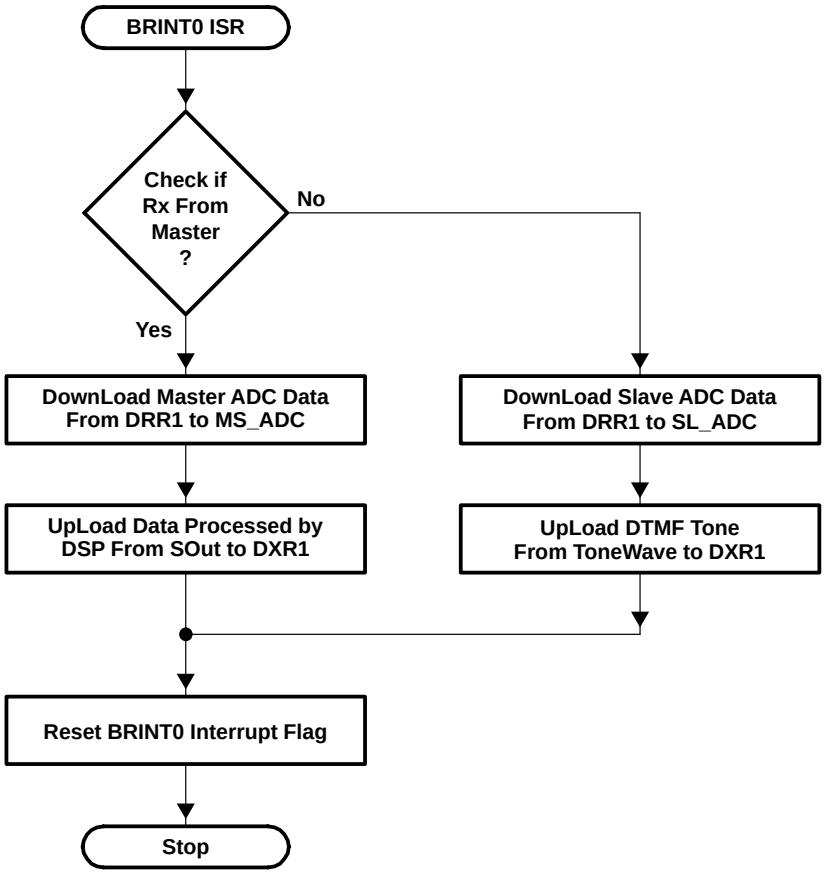


Figure 9. BRINIT ISR Flow Chart

3.5 Software Structure

The code's structure (see Figure 10) has been designed so that the driver and other system software can be reused, enabling users to concentrate their efforts on their specific application. Using this basic structure, the development of numerous other applications requires only minor changes to a few lines of code. Furthermore, user-supplied routines can be added into the structure in the same way as has been done for the examples DTMFTone.c, SecurityVoice.c, and EchoCancel.c.

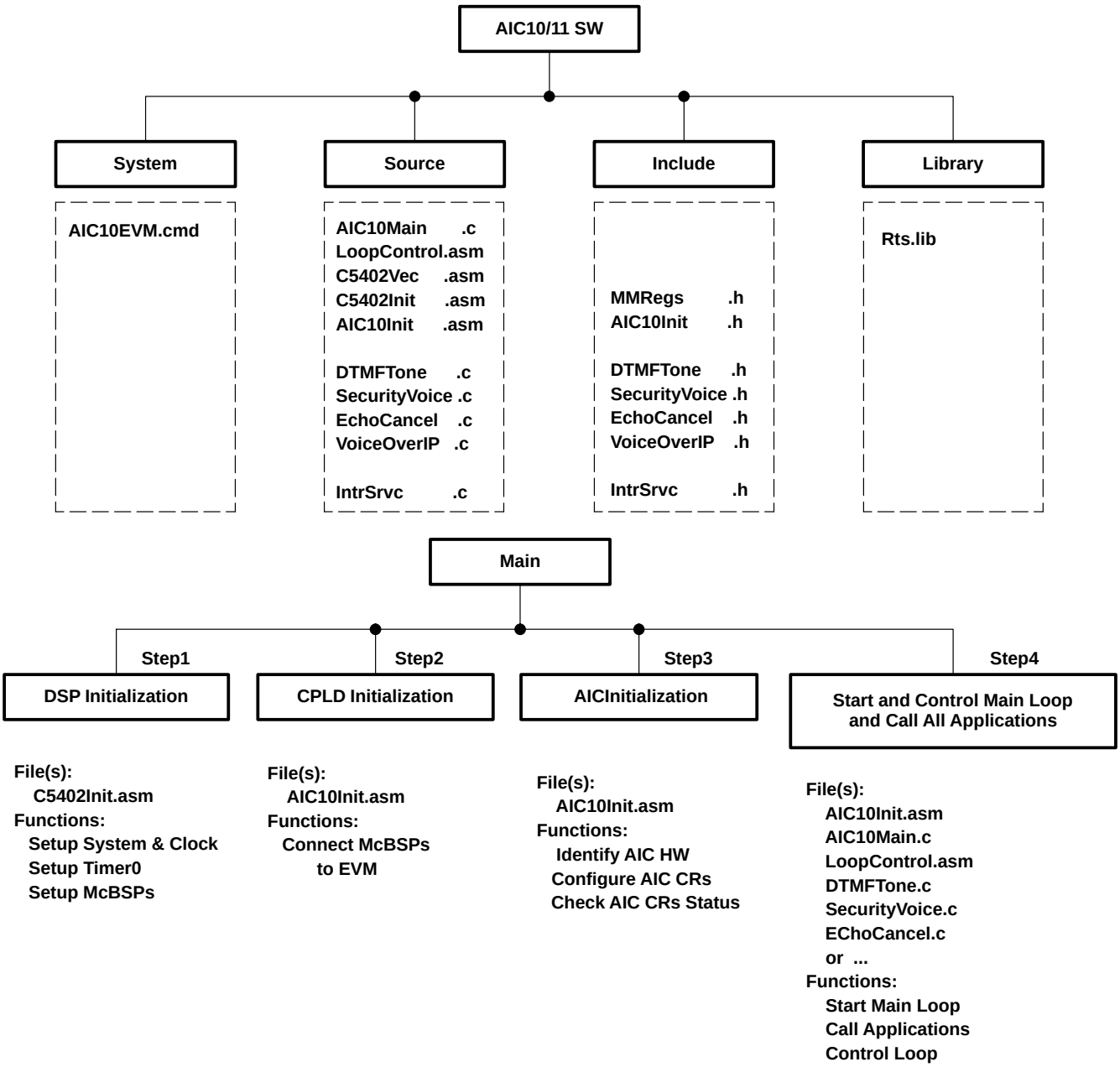


Figure 10. Software Tree Structure

3.6 Application Examples

In this report, three application examples have been implemented and developed into C-language routines that conform to the general structure of Figure 10.

3.6.1 Tone Generation

An audio tone is generated by the DSP from a group of seven sine-wave tables with different frequencies, from 697 Hz to 1477 Hz. The tone generator runs at 16 kHz. At each sample point, a high and a low tone are generated from two different frequency sine tables, and are combined to form a two-frequency tone. The seven single-frequency tones are mixed two-by-two to obtain

10 two-frequency tones. The tones are transmitted to the slave AIC10 device and output to the speaker connected on the EVM's jack, J6. Each of the 10 two-frequency tones is output to the speaker every 0.1 s and the 10-tone combinations simulate music, whose pattern is repeated at a frequency of 1 Hz.

3.6.2 Voice Security System

A microphone is plugged into J12 (connected to the ADC path of the slave AIC10 on the EVM). This signal is converted to digital by the AIC10 and sent to the DSP through the McBSP0. The DSP mixes this audio signal with a fixed-frequency sine wave, called the carrier wave, so as to shift the audio signal into a different part of the frequency spectrum. Then a 4^{th}-order Butterworth low-pass digital filter is applied to remove the right half of the mirrored frequency spectrum of the carrier frequency. The result is the same aural signal, but with a different vocal sound, as illustrated in Figure 11. The processed voice signal is output to the speaker connected by jack J5 to the master AIC10. The block diagram for the system is shown in Figure 12.

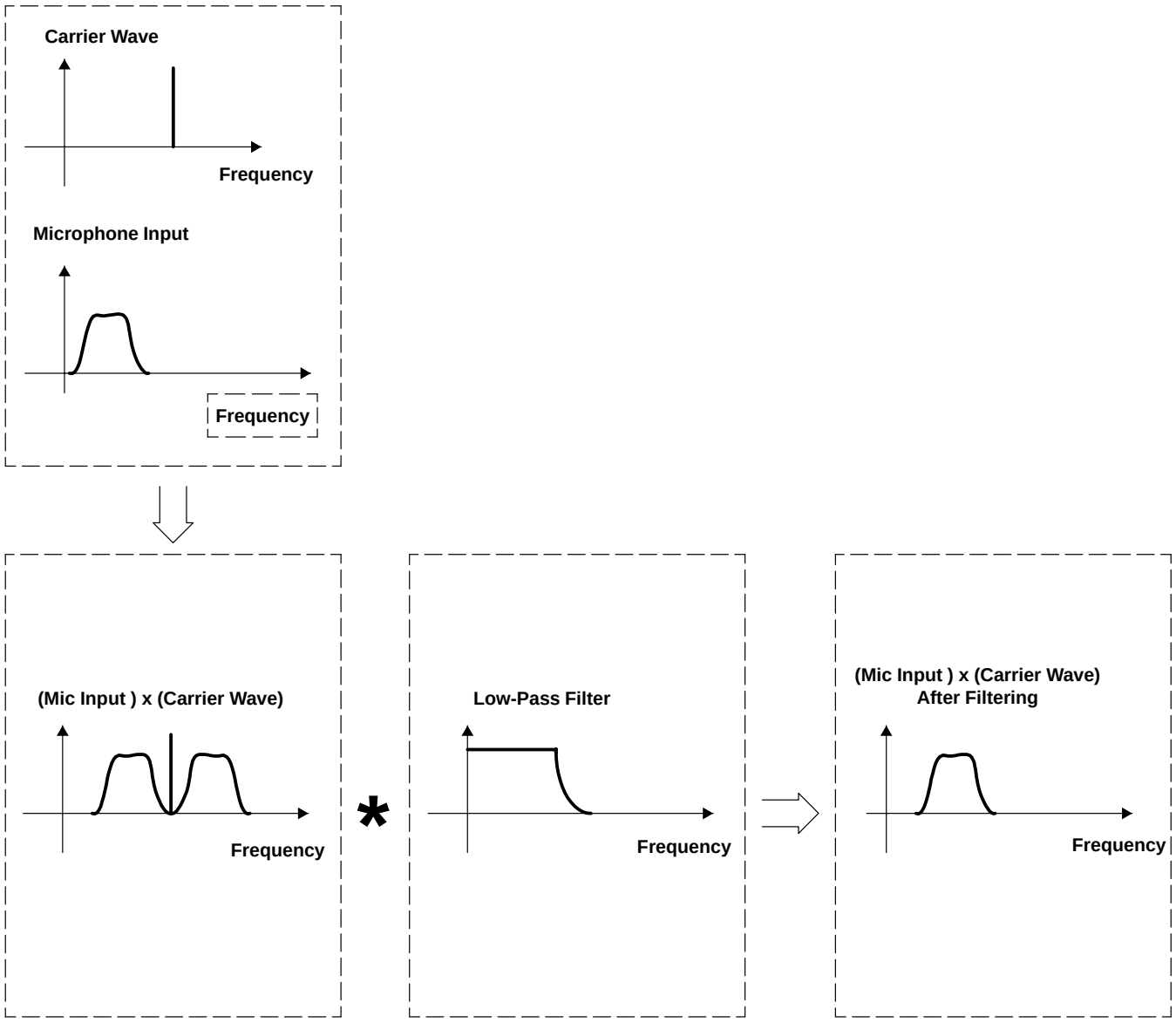


Figure 11. Principle of Voice Security System

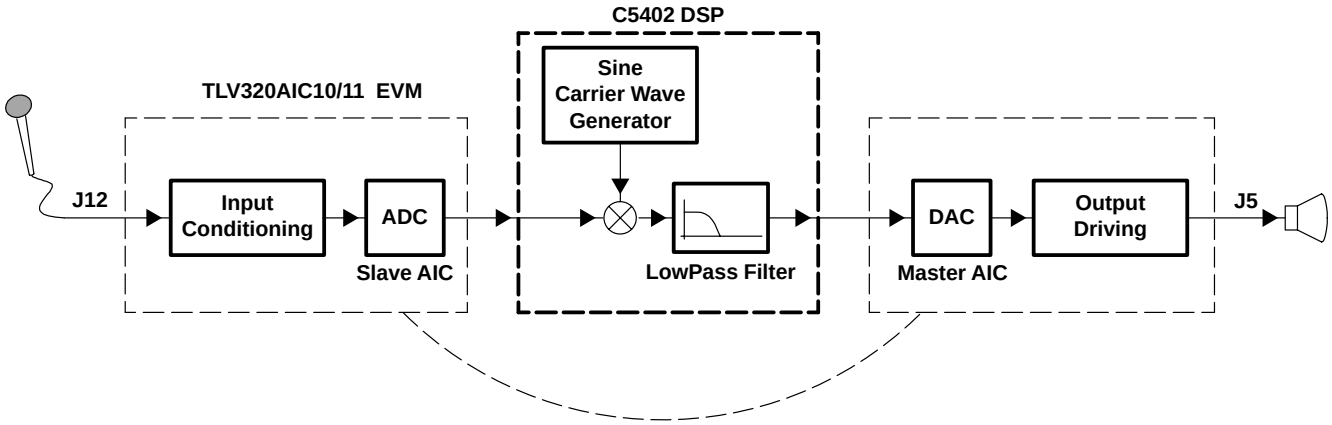


Figure 12. Voice Security System Block Diagram

This kind of application is sometimes called a *voice security system* because the voice delivered by the speaker differs from the voice speaking into the microphone. This masks the identity of the speaker.

3.6.3 Echo Cancellation

TI has developed a number of algorithms and routines for echo cancellation. These are used with many different DSP and analog devices. The C5402 DSP and two AIC10 devices are used in this application report. Figure 13 illustrates the principle.

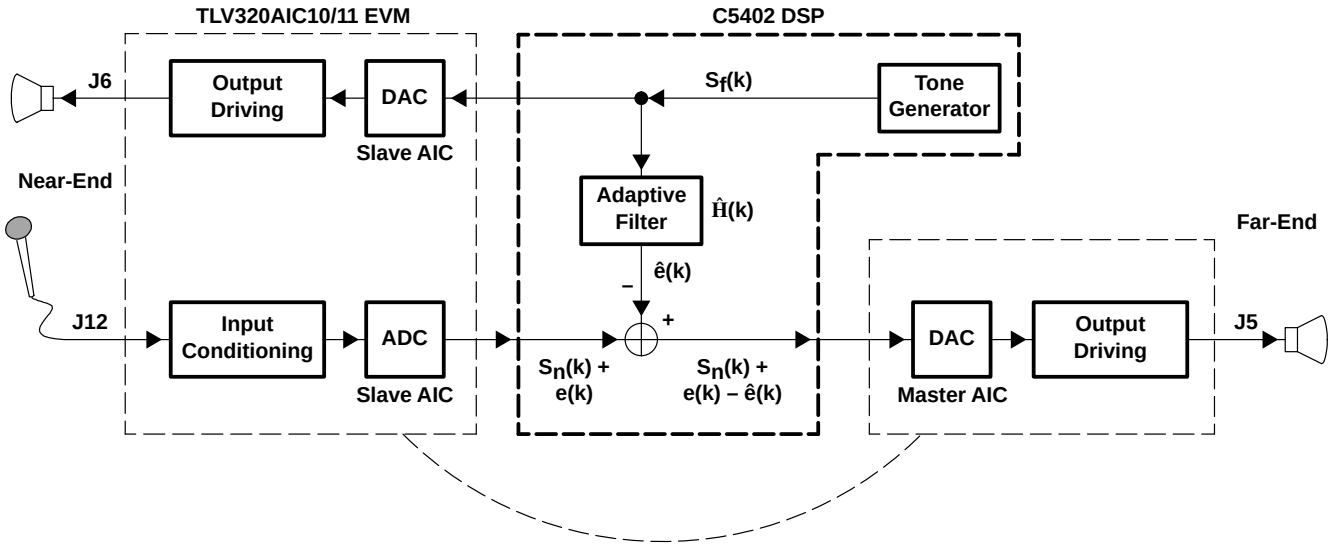


Figure 13. Echo Cancellation Block Diagram

The tone from the tone-generation code (already described) is used as the far-end input signal S_f(k). This signal is output to the near-end speaker which is the speaker connected to the slave AIC through J6. Also, $S_f(k)$ is fed into the DSP's echo-cancellation routine for the echo estimation calculation. It is estimated that the echo in the near-end for $S_f(k)$, denoted by $\hat{e}(k)$, is:

$$\hat{e}(k) = \hat{H}(z) \times S_f(k) \quad (2)$$

where $\hat{H}(z)$ is an adaptive digital filter whose parameters are updated at every sample to minimize the estimation error. See *Acoustic-Echo Cancellation Software for Hands-Free Wireless Systems,* SPRA162 [12].

The real echo, $e(k)$, of the far-end input signal travels from the near-end speaker to the near-end microphone, or from the slave AIC10's speaker (connected to J6) to its microphone (connected to J12). The near-end microphone signal and the echo signal are both picked up by the near-end microphone, transferred the slave AIC10's ADC and input to the DSP. The echo-cancellation software processes the ADC data so that the output to the far-end speaker, J5, which is the master AIC10 DAC output, is: $S_n(k) + e(k) - \hat{e}(k)$, where $S_n(k)$ is the *pure* near-end microphone signal, $e(k)$ is the near-end echo disturbance, and $\hat{e}(k)$ is the estimated $e(k)$, expressed by equation 2.

If the adaptive filter is reasonably good, or $\hat{e}(k) \approx e(k)$, the far-end speaker will output $S_n(k)$, the pure near-end microphone signal, without echo disturbance effects.

The echo cancellation routine runs at a loop rate of 8 kHz. The ADC/DAC sample rate and the adaptive filter parameter adapting rate are both 8 kHz.

4 References

1. *TMS320VC5402 Fixed-Point Digital Signal Processor*, Data Sheet, Texas Instruments Literature Number SPRS079D
2. *TLV320AIC10/11 EVM,* User's Guide, Texas Instruments Literature Number SLWU003C
3. *TMS320C54x DSP Reference Set, Volume 4: Applications Guide*, Texas Instruments Literature Number SPRU173
4. *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals*, Texas Instruments Literature Number SPRU131F
5. *TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set*, Texas Instruments Literature Number SPRU172B
6. *TMS320C54x Optimizing C Compiler,* User's Guide, Texas Instruments Literature Number SPRU103D
7. *TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals*, Texas Instruments Literature Number SPRU302
8. *TMS320C54x Assembly Language Tools,* User's Guide, Texas Instruments Literature Number SPRU102D
9. *General-Purpose 3 V to 5.5 V 16-bit 22-ksps DSP Codec TLV320AIC10,* Data Manual, Texas Instruments Literature Number SLWS093D
10. *General-Purpose Low-Voltage 1.1 V to 3.6 V I/O 16-bit 22-ksps DSP Codec TLV320AIC11,* Data Manual, Texas Instruments Literature Number SLWS100
11. *Hardware Auto-Identification and Software Auto-Configuration for the TLV320AIC10 DSP Codec—a Plug-and-Play Algorithm*, Analog Applications Journal, November, 2000, Texas Instruments Literature Number SLYT023
12. *Acoustic-Echo Cancellation Software for Hands-Free Wireless Systems*, Application Report, Texas Instruments Literature Number SPRA162

Appendix A TMS320C5402 DSK C Program Main Routine

```
/************************************************************************
**  File Name:    AIC10Main.c
**  Part Number:  TLV320AIC10/11EVM–SW–00001
************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10–11–2000    Wendy X Fang   Original Release
**
************************************************************************
**
**  Function:
**    This C code is written for using with the TI's C54 Code Composer
**    Studio platform. It serves as the main program for C5402 DSP. It
**    communicates and controls AIC10/11 CODECs on the (daughter) EVM;
**    and calls other application routines, include: DTMF tone
**    generation, security voice system, acoustic echo cancellation
**    (AEC), and voice over internet protocol (VoIP).
**
************************************************************************
**  References:
**    (1) TMS320C54x DSP, CPU and Peripheral (SPRU131)
************************************************************************/
/************************************************************************
** External Functions Called by Main()
************************************************************************/
/******* DSP & AIC10 System Routines *******/
extern void InitC5402(void);
extern void InitAIC10(void);
extern void LoopControl(void);
/******* Application Routines *******/
extern void DTMFTone(void);
extern void SecurityVoice(void);
extern void EchoCancel(void);
extern void VoiceOverIP(void);
/************************************************************************
** Main Function Program
************************************************************************/
void  main(void)
{
      InitC5402();                  /* initialize C5402 DSP */
      InitAIC10();                  /* initialize AIC10s    */
      while (1)
      {
          DTMFTone();               /* eg#1: generate DTMF tone    */
          SecurityVoice();          /* eg#2: security voice system */
/*          EchoCancel();             eg#3: echo cancellation    */
/*          VoiceOverIP();            eg#4: VoIP function         */
                                    /* add more applications here  */
```

```
            LoopControl();            /* control main-loop rate      */
        }
    }
    /************************************************************************
    **  End of File -- AIC10Main.c
    ************************************************************************/
```

Appendix B DSK Initialization Assembly Routine

```
*************************************************************************
**  File Name:    InitC5402.asm
**  Part Number:  TLV320AIC10/11EVM–SW–0011
*************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
*************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10–11–2000    Wendy X Fang   Original Release
**
*************************************************************************
**
**  Function:
**    This routine initializes C5402 DSP system and McBSP registers
**    for communication to AIC10/11EVM.
**
*************************************************************************
**  References:
**    (1) TMS320C54x DSP, CPU and Peripheral (SPRU131)
**    (2) TMS320C54x DSP, Enhanced Peripheral (SPRU302)
*************************************************************************
      .global     _InitC5402
*************************************************************************
** Include Statements
*************************************************************************
      .include    MMRegs.h
*************************************************************************
**  C5402 DSP Memory Mapping Register Initialization
*************************************************************************
_InitC5402:
      NOP
      LD    #0, DP                  ; reset data–page pointer
      STM    #0, CLKMD              ; software setting of DSP clock
      STM    #0, CLKMD              ; (to divider mode before setting)
      STM    #0x1007, CLKMD         ; set C5402 DSP clock to 40MHz
*      STM   #0x4007, CLKMD          ; set C5402 DSP clock to 100MHz
                                    ; (based on DSK crystal at 20MHz)

******* Configure C5402 System Registers *******
      STM   #0x2000, SWWSR          ; 2 wait cycle for IO space &
                                    ; 0 wait cycle for data&prog spaces
      STM   #0x0000,BSCR            ; set wait states for bank switch:
                                    ; 64k mem bank, extra 0 cycle between
                                    ; consecutive prog/data read
      STM   #0x1800,ST0             ; ST0 at default setting
      STM   #0x2900,ST1             ; ST1 at default setting(note:INTX=1)
      STM   #0x00A0,PMST            ; MC mode & OVLY=1, vectors at 0080h
******* Set up Timer Control Registers *******
      STM   #0x0010, TCR            ; stop on–chip timer0
      STM   #0x0010, TCR1           ; stop on–chip timer1
```

```
                                    ; Timer0 is used as main loop timer
      STM   #2499, PRD              ; timer0 rate=CPUCLK/1/(PRD+1)
                                    ;            =40M/2500=16KHz
*      STM   #6249, PRD             ;  if CPU at 100M/6250=16KHz

******* Initialize McBSP0 Registers *******
      STM   SPCR1,  McBSP0_SPSA     ; register subaddr of SPCR1
      STM   #4020h, McBSP0_SPSD     ; McBSP0 recv = left–justify
                                    ; RINT generated by frame sync

      STM   SPCR2,  McBSP0_SPSA     ; register subaddr for SPCR2
                                    ; XINT generated by frame sync
      STM   #0020h, McBSP0_SPSD     ; McBSP0 Tx = FREE(clock stops
                                    ; to run after SW breakpoint)

      STM   RCR1,   McBSP0_SPSA     ; register subaddr of RCR1
      STM   #0040h, McBSP0_SPSD     ; recv frame1 Dlength = 16 bits
      STM   RCR2,   McBSP0_SPSA     ; register subaddr of RCR2
      STM   #0041h, McBSP0_SPSD     ; recv Phase = 1
                                    ; Set frame2 Dlength = 16bits
      STM   XCR1,   McBSP0_SPSA     ; register subaddr of XCR1
      STM   #0040h, McBSP0_SPSD     ; set the same as recv
      STM   XCR2,   McBSP0_SPSA     ; register subaddr of XCR2
      STM   #0041h, McBSP0_SPSD     ; set the same as recv
; Use this code lines if McBSP0 is used as a master
;      STM   SRGR1,  McBSP0_SPSA     ; register subaddr of sample rate
;                                    ; generator register
;      STM   #0001h, McBSP0_SPSD     ; set to default since CLKR/X
;      STM   SRGR2,  McBSP0_SPSA     ; is input.
;      STM   #2000h, McBSP0_SPSD     ; CLKS is derived from CPU clock
      STM   PCR,    McBSP0_SPSA     ; register subaddress of PCR
      STM   #0000h, McBSP0_SPSD     ; clk and frame from external (slave)
                                    ; FS at pulse–mode(00)

; Use this code lines if McBSP0 is used as a master
;      STM   SPCR2,  McBSP0_SPSA     ; enable sample rate generator
;      LDM   McBSP0_SPSD, A
;      OR    #0x0040, A
;      STLM  A, McBSP0_SPSD
;      RPT   #5                      ; Wait 2 sample rate Clk for
;      NOP                           ;    SRG to stabilize

******* Initialize McBSP1 Registers *******
      STM   SPCR1,  McBSP1_SPSA     ; register subaddr of SPCR1
      STM   #4000h, McBSP1_SPSD     ; McBSP1 recv = left–justify
                                    ; RINT generated by frame sync
      STM   SPCR2,  McBSP1_SPSA     ; register subaddr for SPCR2
                                    ; XINT generated by frame sync
      STM   #0000h, McBSP1_SPSD     ; McBSP1 Tx = FREE(clock stops
                                    ; to run after SW breakpoint
      STM   RCR1,   McBSP1_SPSA     ; register subaddr of RCR1
      STM   #0040h, McBSP1_SPSD     ; recv frame1 Dlength = 16 bits
      STM   RCR2,   McBSP1_SPSA     ; register subaddr of RCR2
```

```
      STM   #0041h, McBSP1_SPSD     ; recv Phase = 1
                                    ; ret frame2 Dlength = 16bits
      STM   XCR1,   McBSP1_SPSA     ; register subaddr of XCR1
      STM   #0040h, McBSP1_SPSD     ; set the same as recv
      STM   XCR2,   McBSP1_SPSA     ; register subaddr of XCR2
      STM   #0041h, McBSP1_SPSD     ; set the same as recv
      STM   PCR,    McBSP1_SPSA     ; register subaddress of PCR
      STM   #0000h, McBSP1_SPSD     ; clk and frame from external (slave)
                                    ; FS at pulse–mode(00)
******* Finish DSP Initialization *******
      STM   #0x0000, IMR            ; disable peripheral interrupts
      STM   #0xFFFF, IFR            ; clear the intrupts' flags

      RET                           ; return to main
      NOP
      NOP

      .end
*************************************************************************
**  End of File –– InitC5402.asm
*************************************************************************
```

Appendix C AIC10/11 Devices Initialization Assembly Routine

```
*************************************************************************
**  File Name:    InitAIC10.asm
**  Part Number:  TLV320AIC10/11EVM–SW–0012
*************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
*************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10–11–2000    Wendy X Fang   Original Release
**
*************************************************************************
**
**  Function:
**    This routine identifies AIC10 device hardware configuration, by
**    means of a plug–and–playing algorithm (refer to ...);  and
**    initializes AIC10/11 CODECs (master and slave)control registers,
**    correspondingly.
**
*************************************************************************
**  References:
**    (1) Data Manual: General–Purpose 3V to 5.5V 16bit 22KSPS DSP Codec
**        – TLV320AIC10
*************************************************************************
      .global     _InitAIC10
*************************************************************************
** Include Statements
*************************************************************************
      .include  MMRegs.h
      .include  InitAIC10.h
*************************************************************************
**  Function Code
*************************************************************************
     .text

_InitAIC10:
      NOP
******* Connect MCBSP to AIC10EVM *******
      PORTR  DSP_CPLD_CNTL2, 0x0060 ; select the AIC10–EVM
      NOP                           ; for McBSP read DSP_CNTL2 reg
      NOP
      ANDM   0xFF00, 0x0060         ; masking default
      ORM    0x0003, 0x0060         ; Connect McBSP to
      PORTW  0x0060, DSP_CPLD_CNTL2 ;     AIC10–EVM board
      NOP
      NOP
******* Put Data to a Default Condition *******
      LD     #InitVari, DP          ; set page pointer to current page
      NOP
      NOP
      ST     #1, AIC10Num           ; set default value of AIC10 number
```

```
      ST     #0, MasterOnFlag       ; set no–master on–board
      ST     #0, IdentErrs          ; set no identification errors
*************************************************************************
**  AIC10 Hardware Configuration Identification
*************************************************************************

******* Step1: (Use McBSP0 for Auto–Master–Detecter)
*******      Clear FSCount & Enable McBSP0 Rx only
IdentStart:
      NOP
      ST     #0, FSCount            ; clear FSCount
      STM    SPCR1, McBSP0_SPSA     ; enable McBSP0 Rx
      LDM    McBSP0_SPSD, A         ;   (by setting bit0 of SPCR1)
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      RPT    #4                     ; wait for 2 SCLKs
      NOP
******* Step2&3:
*******      Wait for Rx finished, count FS & check FSCounter max; &
*******      check DR bit0 (master/slave falg)
FSCycle1:                           ; detect the first master primary–frame
      NOP
      CALL   IfRxRDY0               ; check if a Rx from McBSP0 finished
      ADDM   #1, FSCount            ; increase FS counter
      LD     FSCount, A             ; check if FS counter>16 (max8AICs)
      SUB    #0x0010, A             ;   FSCount – 16
      BC     InitErr1, AGT          ;   to no–master err if FSCount > 16
      NOP
      NOP
      LDM    McBSP0_DRR1, A         ; load Rx data to regA & clr RRDY flag
      AND    #0x0001, A             ; mask out D0: M/S bit to find master:
      BC     FSCycle1, AEQ          ; wait for next RX if
      NOP                           ;   not found 1st master primary frame
      NOP                           ; else ... to cycle2

******* Step4:
*******      when the 1st master AIC10 is  detected,
*******      set flag & clear FSCounter
      NOP
      ST     #0x0001, MasterOnFlag  ; set master on board flag
      ST     #0x0000, FSCount       ; clear frame sync counter
******* Step5:
*******      count AIC10 device number between 2 master FSs
*******      check if there are ant AIC HW errors
FSCycle2:
      NOP
      CALL   IfRxRDY0               ; check if an Rx data ready
      ADDM   #1, FSCount            ; increase FS counter
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      AND    #0x0001, A             ; mask out D0: M/S bit
      BC     FSCycle2, AEQ          ; wait for next RX if
      NOP                           ;   not found 2ns master primary frame
      NOP                           ; else:
```

```
      LD     FSCount, A             ; store number of AIC10 devices
      STL    A, AIC10Num
      BC     InitErr0, ALEQ         ; limit AIC10Num from 1 to 8
      NOP                           ; (InitErr0: no AIC10 on board; &
      NOP                           ;  InitErr2: more than 8 AIC10s or
      SUB    #0x0008, A             ;            other mulfuctions)
      BC     InitErr2, AGT
      NOP
      NOP
******* Step6:
*******      wait for the next(3rd) FS for master &
*******      check to make sure no multiple masters
FSC3Wait:
      NOP
      LD     FSCount, A             ; decrease FSCount by 1
      SUB    #0x0001, A
      STL    A, FSCount
      BC     FSC4Wait, ALEQ         ; if current cycle finished
      NOP                           ; skip & go to next cycle
      NOP                           ; else:
      CALL   IfRxRDY0               ; wait for Rx from McBSP0 finished
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      AND    #0x0001, A             ; mask out D0: M/S bit
      BC     FSC3Wait, AEQ          ; back loop if no multi–master
      NOP                           ; else: to error routine
      NOP
******* Errors
*******
InitErr2:                           ; number of AICs > 8 (not possible)
      NOP                           ; or multi–master or other HW errors
      NOP
      ADDM   #1, IdentErrs          ; set error flag
      LD     IdentErrs, A
      SUB    #1, A
      BC     IdentStart, AEQ
      NOP
      NOP
      B      InitErr2
      NOP
      NOP
InitErr1:                           ; there is no master AIC10 on board
      NOP
      NOP
      ST     #0x0000, MasterOnFlag  ; clear master on board flag
      B      InitErr1
      NOP
      NOP

InitErr0:                           ; there is no AIC10 on board
      NOP
      NOP
      ST     #0x0000, MasterOnFlag  ; clear master on board flag
      ST     #0x0000, AIC10Num      ; clear AIC10 number
```

```
      B      InitFinish
      NOP
      NOP

******* Step7
*******     Enable McBSP TX &
*******     Transfer Dx with 2nd–comm Request for AIC10Num
*******     to duble check no errors
FSC4Wait:
      NOP
      STM    SPCR2, McBSP0_SPSA     ; enable McBSP0 Tx
      LDM    McBSP0_SPSD, A         ; (by setting bit0 at SPCR2)
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      NOP                           ; wait for stablizing
      NOP
      ST     #0x0000, FSCount       ; reset FS counter
      NOP
FSC4Prim:                           ; cycle4 primary
      NOP
      STM    #SECRequ, McBSP0_DXR1  ; set 2nd comm request
      CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
      CALL   IfRxRDY0               ; also wait for Rx in

      LD     FSCount, A             ; Check if 1st FS (for master AIC)
      BC     FSC4PM, AEQ            ; yes: to FSC4PM
      NOP
      NOP                           ; no: make sure all are slaves
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      AND    #0x0001, A             ; mask out D0: M/S bit
      BC     InitErr2, ANEQ
      NOP
      NOP
        B      FSC4Prim1
      NOP
      NOP
FSC4PM:                             ; Cycle4, primary & master FS:
      NOP                           ; make sure it is from master ––
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      AND    #0x0001, A             ; mask out D0: M/S bit
      BC     InitErr2, AEQ
      NOP
      NOP
FSC4Prim1:                          ; check FSCount < AIC10Num
      NOP
      ADDM   #1, FSCount            ; increase FS counter
      LD     FSCount, A             ; if (FSCount–AIC10Num) < 0
      SUB    AIC10Num, A
      BC     FSC4Prim, ALT          ; yes: next 2nd comm request
      NOP                           ; no:  contine
      NOP
******* Step8:
*******     Read AIC10 device ID for master AIC10 to double check no err
```

```
FSC4Second:
      NOP
      ST     #0, FSCount            ; clear FSCount
      NOP
FSC4Sec1:
      NOP
      LD     FSCount, A             ; check if the master device by
      BC     FSC4SecS, ANEQ         ;     FSCount (= 0?)
      NOP                           ; NO: skip ID check
      NOP                           ; YES:
      LD     AIC10Num, A            ; get master AIC ID from Ident
      SUB    #1, A                  ; ID = (AIC10Num–1) << 13
      SFTA   A, 13, A
      OR     #ReadCR1, A            ; ID.OR.CR1, request to read master
      STLM   A, McBSP0_DXR1         ; set to request read master AIC CR1
      CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
      CALL   IfRxRDY0               ; also check Rx get a data

      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      AND    #0xE000, A             ; mask out D15~D13: AIC device ID
      SFTA   A, –13, A              ; shft ID to lowest 3 bits
      ADD    #1, A                  ; get AIC device number from ID
                                    ;   AIC10Num = ((DOUT&0xE000)>>13)+1
      SUB    AIC10Num, A            ; check if ID–DeviceNum = identified
      BC     InitErr2, ANEQ         ; no: to errors
      NOP                           ; else:  to next AIC10 device
      NOP
      BD      FSC4Sec2
      NOP
      NOP
FSC4SecS:                           ; secondary comm slave frame(s)
      NOP
      STM    #0x0000, McBSP0_DXR1   ; no secondary comm requested
      CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
      CALL   IfRxRDY0               ; also check Rx get a data
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      NOP
FSC4Sec2:
      NOP
      ADDM   #1, FSCount            ; increase FS counter
      LD     FSCount, A             ; if (FSCount–AIC10Num) < 0
      SUB    AIC10Num, A
      BC     FSC4Sec1, ALT          ; yes: next secondary comm cycle
      NOP                           ; no:  finish
      STM    SPCR1, McBSP0_SPSA     ; disable McBSP0 RX
      LDM    McBSP0_SPSD,A
      AND    #0xFFFE, A
      STLM   A, McBSP0_SPSD

      STM    SPCR2, McBSP0_SPSA     ; disable McBSP0 TX
      LDM    McBSP0_SPSD,A
      AND    #0xFFFE, A
      STLM   A, McBSP0_SPSD
```

```
      RPT    #5
      NOP
      NOP
*************************************************************************
******* Use McBSP0 to Initalize all AIC Control Registers          *******
*******  --  AIC10 Configuration                                  *******
*************************************************************************
      NOP
      ST     #0x0004, CRegCount     ; set AIC control reg counter (4 CRs)
      NOP
      STM    SPCR2, McBSP0_SPSA     ; enable McBSP0 Tx
      LDM    McBSP0_SPSD, A         ; (by setting bit0 at SPCR2)
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      NOP
      NOP
      STM    #0x0020, IMR           ; unmask DXINT0 for 'IDLE'

******* Load Sencondary Comm Phase Configuring AIC10 CRs *******
******* Note: DX = [(AIC ID) 0 (CR #) x |Config]
*******            [ 15 ~ 13 12 11~9  8 | 7 ~ 0]
*******
      NOP                          ; contents for configuring CR1
      STM    #WriteMCR1, AR1
      STM    #WriteSCR1, AR2
      LD     AIC10Num, A            ; set FSCount
      STL    A, FSCount
      BD     InitStart
      NOP
      NOP

InitAICR2:
      NOP                           ; contents for configuring CR2
      LD     CRegCount, A
      SUB    #3, A
      BC     InitAICR3, ALT
      NOP
      NOP
      STM    #WriteMCR2, AR1
      STM    #WriteSCR2, AR2
      LD     AIC10Num, A            ; set FSCount
      STL    A, FSCount
      BD     InitStart
      NOP
      NOP

InitAICR3:
      NOP                           ; contents for configuring CR3
      LD     CRegCount, A
      SUB    #2, A
      BC     InitAICR4, ALT
      NOP
      NOP
```

```
      STM    #WriteMCR3, AR1
      STM    #WriteSCR3, AR2
      LD     AIC10Num, A            ; set FSCount
      STL    A, FSCount
      BD     InitStart
      NOP
      NOP
InitAICR4:
      NOP                           ; contents for configuring CR4
      STM    #WriteMCR4, AR1
      STM    #WriteSCR4, AR2
      LD     AIC10Num, A            ; set FSCount
      STL    A, FSCount
      NOP
      NOP

InitStart:
      NOP                           ; this is a primary cycle
      STM    #0x3FFF, IFR           ; Clear interrupt flag
      STM    #SECRequ,McBSP0_DXR1   ; load 2nd req to DX (at primary)
      IDLE   1                      ; wait for the TX finished
      LD     FSCount, A             ; check if all AIC are requested
      SUB    #1, A
      STL    A, FSCount
      BC     InitStart, AGT         ; no:  back to do more request
      NOP                           ; yes: to secondary cycle

******* Configuring Master AIC10 *******
      NOP
      LD     AIC10Num, A            ; get master AIC ID from Ident
      SUB    #1, A                  ; ID = (AIC10Num–1) << 13
      SFTA   A, 13, A
      LDM    AR1, B                 ; get CRx for master
      OR     B, A                   ; ID.OR.CRx, configuring contents
      STM    #0x3FFF, IFR           ; Clear interrupt flag
      STLM   A, McBSP0_DXR1         ; put config data to CRx
      IDLE   1                      ; wait for the TX finished

      ST     #1, FSCount            ; clear frame sync counter
******* Configuring Slave AIC10(s) *******
InitAICSec:
      NOP
      ADDM   #1, FSCount            ; increase FSCount
      LD     FSCount, A             ; check if FSCount > AIC10Num
      SUB    AIC10Num, A
      BC     InitAICSec1, AGT       ; yes: to next CRs
      NOP                           ; no:
      NOP

      LD     AIC10Num, A            ; get master AIC ID from Ident
      SUB    FSCount, A             ; ID = (AIC10Num–FSCount) << 13
      SFTA   A, 13, A
      LDM    AR2, B                 ; get CRx for slave
```

```
      OR     B, A                  ; ID.OR.CRx, configuring contents
      STM    #0x3FFF, IFR          ; Clear interrupt flag
      STLM   A, McBSP0_DXR1        ; put config data to CRx
      IDLE   1                     ; wait for the TX finished
      NOP
      BD     InitAICSec             ; to next slave
      NOP
      NOP

InitAICSec1:                       ; check if all 4 CRs have been conf
      NOP
      LD    CRegCount, A
      SUB   #1, A
      STL   A, CRegCount
      BC    InitAICR2, AGT
      NOP
      NOP
*******************************************************************************
******* InitAIC10 Return to Main Routine *******
*******************************************************************************
InitFinish:
      NOP

      STM    SPCR1, McBSP0_SPSA     ; disable McBSP0 RX
      LDM    McBSP0_SPSD,A
      AND    #0xFFFE, A
      STLM   A, McBSP0_SPSD
      STM    SPCR2, McBSP0_SPSA     ; disable McBSP0 TX
      LDM    McBSP0_SPSD,A
      AND    #0xFFFE, A
      STLM   A, McBSP0_SPSD
      RPT    #7
      NOP
      STM    0x0000, IMR            ; disable peripheral interrupts
      STM    #0x3FFF, IFR           ; reset all interrupt flags
      CALL   ReadCRs                ; read & store AIC10's CRs status
      NOP

      STM    #0x0010, IMR           ; enable BRINT0 interrupts
*      STM    #0x0800, IMR           ; enable BXINT1 interrupts
      STM    #0x3FFF, IFR           ; reset all interrupt flags
      RSBX   INTM                   ; enable system interrupts
      STM    SPCR1, McBSP0_SPSA     ; ena McBSP0 RX for ADC data in
      LDM    McBSP0_SPSD,A
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      STM    SPCR2, McBSP0_SPSA     ; enable McBSP0 TX for DTMF out
      LDM    McBSP0_SPSD,A
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD

      LDM    TCR, A                 ; start timer0 for main loop
      AND    #0xFFEF, A             ;   (by clear bit4 of TCR)
```

```
      STLM   A, TCR
      ST     #0, LoopCount          ; reset main loop counter
      NOP
      RETD                          ; return to main
      NOP
      NOP
*************************************************************************
** Local Subrotuines
*************************************************************************
******* Waiting for McBSP0 RX Finished *******
IfRxRDY0:
      NOP
      STM    SPCR1, McBSP0_SPSA     ; enable McBSP0 Rx
      LDM    McBSP0_SPSD, A
      AND    #0002h, A              ; mask RRDY bit
      BC     IfRxRDY0, AEQ          ; keep checking
      RETD                          ; return
      NOP
      NOP
******* Waiting for McBSP0 TX Finished *******
IfTxRDY0:
      NOP
      STM    SPCR2, McBSP0_SPSA     ; enable McBSP0 Tx
      LDM    McBSP0_SPSD, A
      AND    #0002h, A              ; mask TRDY bit
      BC     IfTxRDY0, AEQ          ; keep checking
      RETD                          ; return
      NOP
      NOP

******* Read & Store AIC10 Control Register Values After Config *******
ReadCRs:
      NOP
      STM    SPCR1, McBSP0_SPSA     ; ena McBSP0 RX for ADC data in
      LDM    McBSP0_SPSD,A
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      STM    SPCR2, McBSP0_SPSA     ; enable McBSP0 TX for DTMF out
      LDM    McBSP0_SPSD,A
      OR     #0x0001, A
      STLM   A, McBSP0_SPSD
      LD     #InitVari, DP          ; load data page
      RPT    #7
      NOP

      STM    #0x0001, McBSP0_DXR1
      CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
      CALL   IfRxRDY0               ; also check Rx get a data
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
      STM    #0x0001, McBSP0_DXR1
      CALL   IfTxRDY0
      CALL   IfRxRDY0               ; also check Rx get a data
      LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
```

```
STM    #0x3200, McBSP0_DXR1   ; set to request read master AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, MstCR1              ; save master CR1 to MstCR4

STM    #0x1200, McBSP0_DXR1   ; set to request read slave AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, SlvCR1              ; save master CR1 to SlvCR4
NOP

STM    #0x0001, McBSP0_DXR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STM    #0x0001, McBSP0_DXR1
CALL   IfTxRDY0
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STM    #0x3400, McBSP0_DXR1   ; set to request read master AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, MstCR2              ; save master CR1 to MstCR4

STM    #0x1400, McBSP0_DXR1   ; set to request read slave AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, SlvCR2              ; save master CR1 to SlvCR4
NOP

STM    #0x0001, McBSP0_DXR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STM    #0x0001, McBSP0_DXR1
CALL   IfTxRDY0
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STM    #0x3600, McBSP0_DXR1   ; set to request read master AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, MstCR3              ; save master CR1 to MstCR4

STM    #0x1600, McBSP0_DXR1   ; set to request read slave AIC CR1
CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
CALL   IfRxRDY0               ; also check Rx get a data
LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
STL    A, SlvCR3              ; save master CR1 to SlvCR4
```

```
        NOP

        STM    #0x0001, McBSP0_DXR1
        CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
        CALL   IfRxRDY0               ; also check Rx get a data
        LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
        STM    #0x0001, McBSP0_DXR1
        CALL   IfTxRDY0
        CALL   IfRxRDY0               ; also check Rx get a data
        LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
        STM    #0x3800, McBSP0_DXR1   ; set to request read master AIC CR1
        CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
        CALL   IfRxRDY0               ; also check Rx get a data
        LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
        STL    A, MstCR4              ; save master CR1 to MstCR4

        STM    #0x1800, McBSP0_DXR1   ; set to request read slave AIC CR1
        CALL   IfTxRDY0               ; wait for Tx from McBSP0 finished
        CALL   IfRxRDY0               ; also check Rx get a data
        LDM    McBSP0_DRR1, A         ; ld Rx data to regA & clr RRDY flag
        STL    A, SlvCR4              ; save master CR1 to SlvCR4
        NOP

        STM    SPCR1, McBSP0_SPSA     ; disable McBSP0 RX
        LDM    McBSP0_SPSD,A
        AND    #0xFFFE, A
        STLM   A, McBSP0_SPSD
        STM    SPCR2, McBSP0_SPSA     ; disable McBSP0 TX
        LDM    McBSP0_SPSD,A
        AND    #0xFFFE, A
        STLM   A, McBSP0_SPSD
        RPT    #7
        NOP
        RET
        NOP
        .end
*************************************************************************
**  End of File -- InitAIC10.asm
*************************************************************************
*************************************************************************
**  File Name:    InitAIC10.h
**  Part Number:  TLV320AIC10/11EVM-SW-0111
*************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
*************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10-11-2000    Wendy X Fang   Original Release
**
*************************************************************************
**
**  Function:
```

```
**    This header file defines values for AIC10 CODEC control registers,
**    which will be updated by costoms when additional AICs is added to
**    the EVM board (?? should fixed before final release ??).
**
*************************************************************************
**  References:
**    (1) Data Manual: General–Purpose 3V to 5.5V 16bit 22KSPS DSP Codec
**        – TLV320AIC10
*************************************************************************
*************************************************************************
**  Define Statements
*************************************************************************
SECRequ      .set   0x0001          ; in prim comm, request for 2nd comm
ReadCR1      .set   0x1200          ; in 2nd comm, read AIC10 CR1 request
******* Master AIC Control Register Values (in 2nd Comm) *******
WriteMCR1    .set   0x0200          ; use default
WriteMCR2    .set   0x0402          ; N=2 for fs = 16Khz
WriteMCR3    .set   0x0618          ; turn on OFF–HOOK to DAA
WriteMCR4    .set   0x0800          ; gain: Input=0dB, Output=24db
******* Slave AIC(s) Control Register Values (in 2nd Comm) *******
WriteSCR1    .set   0x0250          ; enable MIC op–amp interface
WriteSCR2    .set   0x0402          ; N=2 for fs = 16Khz
WriteSCR3    .set   0x0600          ; use default
WriteSCR4    .set   0x0800          ; gain: Input=24dB, Output=24db
*************************************************************************
**  Varaible & Initialization
*************************************************************************
             .global  AIC10Num, MasterOnFlag, IdentErrs

InitVari     .usect ".variable", 1  ; initial point of data
FSCount      .usect ".variable", 1  ; comm FS (frame sync) counter
CRegCount    .usect ".variable", 1  ; AIC control register counter
AIC10Num     .usect ".variable", 1  ; number of AIC10 chips on EVM
MasterOnFlag .usect ".variable", 1  ; a master AIC10 on board flag
IdentErrs    .usect ".variable", 1  ; AIC10 hw errors identified
MstCR1       .usect ".variable", 1  ; master AIC10 CR1 status
MstCR2       .usect ".variable", 1  ; master AIC10 CR2 status
MstCR3       .usect ".variable", 1  ; master AIC10 CR3 status
MstCR4       .usect ".variable", 1  ; master AIC10 CR4 status
SlvCR1       .usect ".variable", 1  ; slave AIC10 CR1 status
SlvCR2       .usect ".variable", 1  ; slave AIC10 CR2 status
SlvCR3       .usect ".variable", 1  ; slave AIC10 CR3 status
SlvCR4       .usect ".variable", 1  ; slave AIC10 CR4 status
*************************************************************************
** external variables
*************************************************************************
             .ref   LoopCount       ; main loop counter
*************************************************************************
**  End of File –– InitAIC10.h
*************************************************************************
```

Appendix D DSP Interrupt Service Routines

```
/************************************************************************
**  File Name:    IntrSrvc.c
**  Part Number:  TLV320AIC10/11EVM–SW–00002
************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10–11–2000    Wendy X Fang   Original Release
**
************************************************************************
**
**  Function:
**    This file includes all interrupt service routines (ISR) for
**    T5402 DSP that communicates to AIC10 CODEC EVM.
**
************************************************************************
**  References:
**    (1) TMS320C54x DSP, CPU and Peripheral (SPRU131)
**    (2) TMS320C54x DSP, Enhanced Peripheral (SPRU302)
************************************************************************/
/************************************************************************
** Include Statements
************************************************************************/
#include   ”IntrSrvc.h”
/************************************************************************
**  Interrupt Serivece Routines
************************************************************************/
/******** This ISR responds to McBSP0 receive ready
          flags to load ADC data from McBSP RX register
          and send DTMF tone to AIC10 for output
          This interrupt occures at 16Khz frequency!  *******/
interrupt void McBSP0RXISR()
{
    McBSP0Frame=McBSP0_DRR1&0x0001; /* get master/slave bit */
    if (McBSP0Frame == 1)
    {                               /* while McBSP0 comm to master AIC */
       MS_ADC = McBSP0_DRR1&0xFFFE; /* load master AIC10 ADC result    */
       McBSP0_DXR1 = SOut;          /* securied voice to mstr AIC DAC  */
    }
    else
    {                               /* while McBSP0 comm to slave AIC */
       SL_ADC = McBSP0_DRR1&0xFFFE; /* load slave AIC10 ADC result    */
       McBSP0_DXR1 = ToneWave;      /* DTMF tone to slave AIC DAC     */
    }
    IFR   |= 0x0010;                /* reset BRINT0 interrupt flag    */
}
/******** This ISR responds to FS to generate
          input for DCSI input on the AIC10–EVM
          via McBSP1 TX ready interrupt (BXINT1)
```

```
           This routine will be call if DCSI is used
           as for configuration for AIC10 device
           control registers. Do the following
           before enable this interrupt:
           (1) Set DCSICount = 0;
           (2) Clear McBSP1's RX/TX interrupt flags first;
           (3) Enable McBSP1's TX; and
           (4) Enable BXINT1                        *******/
interrupt void McBSP1TXISR()
{
    if (McBSP1Frame == 0)          /* while at master phase  */
    {
       DCSIData  = SlaveCRValues[DCSICount]; /* load slave data  */
       DCSICount += 1;            /* point to next CR        */
       McBSP1Frame = 1;
    }
    else                           /* while at slave phase       */
    {
       DCSIData = MasterCRValues[DCSICount];
       if (DCSICount >= 4)         /* total 4 CRs at each AIC */
       {                           /* while finished config:  */
           DCSICount  = 0;         /* clear Counter           */
           DCSIConfig = 0;         /* clear DCSI Config flag  */
       }
       McBSP1Frame = 0;
    }
    IFR |= 0x0800;                 /* reset BXINT1 interrupt flag */
    McBSP1_DXR1 = DCSIData;        /* put data to McBSP1 TX reg   */
}
/************************************************************************
**  End of File -- IntrSrvc.c
************************************************************************/
/************************************************************************
**  File Name:    IntrSrvc.h
**  Part Number:  TLV320AIC10/11EVM-SW-00102
************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10-11-2000    Wendy X Fang   Original Release
**
************************************************************************
**
**  Function:
**      This is the head file of the ISR file -- IntrSrvc.c, which defines
**      or refers to all variables and parameters the ISR file uses.
**
************************************************************************/
/************************************************************************
**  Define Statements for MMRs
************************************************************************/
```

```
#define IMR          (*(volatile unsigned int *)0x0000)
#define IFR          (*(volatile unsigned int *)0x0001)
#define McBSP0_DRR2  (*(volatile unsigned int *)0x0020)
#define McBSP0_DRR1  (*(volatile unsigned int *)0x0021)
#define McBSP0_DXR2  (*(volatile unsigned int *)0x0022)
#define McBSP0_DXR1  (*(volatile unsigned int *)0x0023)
#define McBSP1_DRR2  (*(volatile unsigned int *)0x0040)
#define McBSP1_DRR1  (*(volatile unsigned int *)0x0041)
#define McBSP1_DXR2  (*(volatile unsigned int *)0x0042)
#define McBSP1_DXR1  (*(volatile unsigned int *)0x0043)
/************************************************************************
**  Global Varaibles Used for TX from McBSP0 to AIC10s
************************************************************************/
extern   int   ToneWave;            /* DTMF tone wave form */
extern   int   SOut;                /* security voice or farend output*/
extern   int   MicPhIn;
/************************************************************************
**  Global Varaibles Used for Loading AIC10s' ADC Data
************************************************************************/
unsigned int   MS_ADC    = 0;       /* master AIC10 ADC data storage */
unsigned int   SL_ADC    = 0;       /* slave1 AIC10 ADC data storage */
/************************************************************************
**  Global Varaibles Used for DCSI Interfacing (direct AIC10 CRs Config)
**  DCSI Data Format:  [ D15(=0)  D14~D12  D11~D9  D8  D7 ~ D1 ]
**                      StartBit DeviceAdd CRs'ID  x  ConfigData
************************************************************************/
unsigned int  DCSIConfig ;
unsigned int  DCSICount   = 0;      /* DCSI interface interval counter */
int           DCSIData    = 0;      /* DCSI input(from DSP to CODEC) */
int     MasterCRValues[4] =         /* master AIC10's CRs values    */
        { 0x1200,                   /* use default for master CR1   */
          0x1402,                   /* N=2 for Fs=16KHz, master CR2 */
          0x1618,                   /* turn ON OFF-HOOK to DAA, CR3 */
          0x1800 };                 /* gain: Input=0dB, Output=0dB  */
int     SlaveCRValues[4] =          /* slave AIC10's CRs values     */
        { 0x0250,                   /* enable MIC op-amp interface  */
          0x0402,                   /* N=2 for Fs=16KHz, master CR2 */
          0x0618,                   /* use default                  */
          0x0800 };                 /* gain: Input=0dB, Output=0dB  */
/************************************************************************
**  Local Variables
************************************************************************/
/******* McBSP0 *******/
unsigned int  McBSP0Frame = 0;      /* McBSP0 comm to AIC10EVM frame */
/******* McBSP1 *******/
unsigned int  McBSP1Frame = 0;      /* McBSP1 comm to AIC10EVM frame */

/************************************************************************
**  End of File -- IntrSrvc.h
************************************************************************/
```

Appendix E TMS320C5402 DSP Memory Mapped Registers

```
;=============================================================================
;  File Name:    MMRegs.h
;  Part Number:  TLV320AIC10/11EVM–SW–0100
;=============================================================================
;  Copyright (c) Texas Instruments, Inc. 2000
;=============================================================================
;
;  Release History:
;     Version      Date          Engr          Description
;      1.00     10–11–2000    Wendy X Fang   Original Release
;
;=============================================================================
;
;  Function:
;   1. Memory–Maped Registers (MMR) are mapped to data–page memory
;      (Address: 0x0000 to 0x005F). Note that the assembly tool
;      ".mmregs" can NOT be applied for C5402 directly!!
;   2. Sub–bank addresses for McBSP and DMA are defined, and
;   3. Addresses for C5402 DSK on–board IO ports are defined.
;
;=============================================================================
;  References:
;    (1) TMS320VC5402 Fixed–Point Digital Signal Processor (SPRS079D)
;    (2) TMS320C54x DSP, CPU and Peripheral (SPRU131)
;    (3) TMS320C54x DSP, Enhanced Peripheral (SPRU302)
;=============================================================================
;=============================================================================
; C5402 Memory Mapped Register Defination (On Data–Page or Page1)
;=============================================================================
******** Map Interrupt Registers to Data Page Addresses
IMR         .set    0x0000    ; interrupt mask reg
IFR         .set    0x0001    ; interrupt mask reg
******** Map CPU Registers to Data Page Addresses
ST0         .set    0x0006    ; CPU status reg0
ST1         .set    0x0007    ; CPU status reg1
A           .set    0x0008    ; CPU accumulator A
AL          .set    0x0008    ; CPU accumulator A low word
AH          .set    0x0009    ; CPU accumulator A high word
AG          .set    0x000A    ; CPU accumulator A guard word
B           .set    0x000B    ; CPU accumulator B
BL          .set    0x000B    ; CPU accumulator B low word
BH          .set    0x000C    ; CPU accumulator B high word
BG          .set    0x000D    ; CPU accumulator B guard word
TREG        .set    0x000E    ; CPU temporary reg
TRN         .set    0x000F    ; CPU transition reg
* AR0         .set    0x0010    ; CPU auxiliary reg0
* AR1         .set    0x0011    ; CPU auxiliary reg1
* AR2         .set    0x0012    ; CPU auxiliary reg2
* AR3         .set    0x0013    ; CPU auxiliary reg3
* AR4         .set    0x0014    ; CPU auxiliary reg4
* AR5         .set    0x0015    ; CPU auxiliary reg5
```

```
* AR6          .set    0x0016    ; CPU auxiliary reg6
* AR7          .set    0x0017    ; CPU auxiliary reg7
* SP           .set    0x0018    ; CPU stack pointer reg
BK           .set    0x0019    ; CPU circular buffer size reg
BRC          .set    0x001A    ; CPU block repeat counter
RSA          .set    0x001B    ; CPU block repeat start address
REA          .set    0x001C    ; CPU block repeat end address
******* Map System Registers to Data Page Addresses
PMST         .set    0x001D    ; processor mode status reg
XPC          .set    0x001E    ; extended program page reg
SWWSR        .set    0x0028    ; software wait–state reg
BSCR         .set    0x0029    ; bank–switching control reg
SWCR         .set    0x002B    ; software wait–state control reg
CLKMD        .set    0x0058    ; clock mode reg
******** Map McBSP0 Registers to Data Page Addresses
McBSP0_DRR2 .set    0x0020    ; McBSP0 data Rx reg2
McBSP0_DRR1 .set    0x0021    ; McBSP0 data Rx reg1
McBSP0_DXR2 .set    0x0022    ; McBSP0 data Tx reg2
McBSP0_DXR1 .set    0x0023    ; McBSP0 data Tx reg1
McBSP0_SPSA .set    0x0038    ; McBSP0 sub bank addr reg
McBSP0_SPSD .set    0x0039    ; McBSP0 sub bank data reg
******** Map McBSP1 Registers to Data Page Addresses
McBSP1_DRR2 .set    0x0040    ; McBSP1 data Rx reg2
McBSP1_DRR1 .set    0x0041    ; McBSP1 data Rx reg1
McBSP1_DXR2 .set    0x0042    ; McBSP1 data Tx reg2
McBSP1_DXR1 .set    0x0043    ; McBSP1 data Tx reg1
McBSP1_SPSA .set    0x0048    ; McBSP1 sub bank addr reg
McBSP1_SPSD .set    0x0049    ; McBSP1 sub bank data reg
******** Map Timer0 Registers to Data Page Addresses
TIM          .set    0x0024    ; timer0 reg
PRD          .set    0x0025    ; timer0 period reg
TCR          .set    0x0026    ; timer0 control reg
******** Map Timer1 Registers to Data Page Addresses
TIM1         .set    0x0030    ; timer1 reg
PRD1         .set    0x0031    ; timer1 period reg
TCR1         .set    0x0032    ; timer1 control reg
******** Map HPI Registers to Data Page Addresses
HPIC         .set    0x002C    ; HPI control reg
******** Map General IO Port (Pins) Registers to Data Page Addresses
GPIOCR       .set    0x003C    ;GP I/O Pins Control Reg
GPIOSR       .set    0x003D    ;GP I/O Pins Status Reg
******* Map DMA Registers to Data Page Addresses
DMPREC       .set    0x0054    ; DMA channel priority and ebanle control
DMSA         .set    0x0055    ; DMA subbank address reg
DMSDI        .set    0x0056    ; DMA subbank data reg w/autoincrement
DMSDN        .set    0x0057    ; DMA subbank data reg
************************************************************************
** Sub–Bank Address Defination
************************************************************************
******** McBSP Sub–Bank Register Addresses
SPCR1        .set    0x0000    ; McBSP Ser Port Ctrl Reg1
SPCR2        .set    0x0001    ; McBSP Ser Port Ctrl Reg2
RCR1         .set    0x0002    ; McBSP Rx Ctrl Reg1
```

```
RCR2        .set    0x0003    ; McBSP Rx Ctrl Reg2
XCR1        .set    0x0004    ; McBSP Tx Ctrl Reg1
XCR2        .set    0x0005    ; McBSP Tx Ctrl Reg2
SRGR1       .set    0x0006    ; McBSP Sample Rate Gen Reg1
SRGR2       .set    0x0007    ; McBSP Sample Rate Gen Reg2
MCR1        .set    0x0008    ; McBSP Multichan Reg1
MCR2        .set    0x0009    ; McBSP Multichan Reg2
RCERA       .set    0x000A    ; McBSP Rx Chan Enable Reg PartA
RCERB       .set    0x000B    ; McBSP Rx Chan Enable Reg PartB
XCERA       .set    0x000C    ; McBSP Tx Chan Enable Reg PartA
XCERB       .set    0x000D    ; McBSP Tx Chan Enable Reg PartB
PCR         .set    0x000E    ; McBSP Pin Ctrl Reg
******* DMA Sub–Bank Register Addresses
DMARC0      .set    0x0000    ; DMA channel0 source address reg
DMDST0      .set    0x0001    ; DMA channel0 destination address reg
DMCTR0      .set    0x0002    ; DMA channel0 element count reg
DMDFC0      .set    0x0003    ; DMA channel0 sync sel & frame count reg
DMMCR0      .set    0x0004    ; DMA channel0 transfer mode cntrl reg
DMARC1      .set    0x0005    ; DMA channel1 source address reg
DMDST1      .set    0x0006    ; DMA channel1 destination address reg
DMCTR1      .set    0x0007    ; DMA channel1 element count reg
DMDFC1      .set    0x0008    ; DMA channel1 sync sel & frame count reg
DMMCR1      .set    0x0009    ; DMA channel1 transfer mode cntrl reg
DMARC2      .set    0x000A    ; DMA channel2 source address reg
DMDST2      .set    0x000B    ; DMA channel2 destination address reg
DMCTR2      .set    0x000C    ; DMA channel2 element count reg
DMDFC2      .set    0x000D    ; DMA channel2 sync sel & frame count reg
DMMCR2      .set    0x000E    ; DMA channel2 transfer mode cntrl reg
DMARC3      .set    0x000F    ; DMA channel3 source address reg
DMDST3      .set    0x0010    ; DMA channel3 destination address reg
DMCTR3      .set    0x0011    ; DMA channel3 element count reg
DMDFC3      .set    0x0012    ; DMA channel3 sync sel & frame count reg
DMMCR3      .set    0x0013    ; DMA channel3 transfer mode cntrl reg
DMARC4      .set    0x0014    ; DMA channel4 source address reg
DMDST4      .set    0x0015    ; DMA channel4 destination address reg
DMCTR4      .set    0x0016    ; DMA channel4 element count reg
DMDFC4      .set    0x0017    ; DMA channel4 sync sel & frame count reg
DMMCR4      .set    0x0018    ; DMA channel4 transfer mode cntrl reg
DMARC5      .set    0x0019    ; DMA channel5 source address reg
DMDST5      .set    0x001A    ; DMA channel5 destination address reg
DMCTR5      .set    0x001B    ; DMA channel5 element count reg
DMDFC5      .set    0x001C    ; DMA channel5 sync sel & frame count reg
DMMCR5      .set    0x001D    ; DMA channel5 transfer mode cntrl reg
DMSRCP      .set    0x001E    ; DMA source prog page address
DMDSTP      .set    0x001F    ; DMA destination prog page address
DMIDX0      .set    0x0020    ; DMA element index address reg0
DMIDX1      .set    0x0021    ; DMA element index address reg1
DMFRI0      .set    0x0022    ; DMA frame index reg0
DMFRI1      .set    0x0023    ; DMA frame index reg1
DMGSA       .set    0x0024    ; DMA global source address reload reg
DMGDA       .set    0x0025    ; DMA global destination address reload reg
DMGCA       .set    0x0026    ; DMA global counter reload reg
DMGFA       .set    0x0027    ; DMA global frame count reload reg
```

```
*************************************************************************
** C5402 DSK On-Board I/O Memory Mapped Registers
*************************************************************************
DSP_CPLD_CNTL1    .set    0000h     ;Control Reg1
DSP_CPLD_STAT     .set    0001h     ;Status Reg
DSP_CPLD_DMCNTL   .set    0002h     ;Data Memory Control Reg
DSP_CPLD_DBIO     .set    0003h     ;Daughter Brd / GPIO Reg
DSP_CPLD_CNTL2    .set    0004h     ;Control Reg2
DSP_CPLD_SEM0     .set    0005h     ;Semaphore 0
DSP_CPLD_SEM1     .set    0006h     ;Semaphore 1
*************************************************************************
** End of File -- MMRegs.h
*************************************************************************
```

Appendix F Interrupt Vector Table Initialization

```
*************************************************************************
**  File Name:    C5402Vec.asm
**  Part Number:  TLV320AIC10/11EVM–SW–0010
*************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
*************************************************************************
**
**  Release History:
**     Version      Date          Engr          Description
**      1.00     10–11–2000    Wendy X Fang   Original Release
**
*************************************************************************
**  Function:
**    This routine initializes the 54xx DSK PLUS vector table.
**    (Note that each vector must occupates four(4)16–bit space.)
**
*************************************************************************
**  References:
**  (1) TMS320C54x DSP, CPU and Peripheral (SPRU131)
**  (2) TMS320C54x DSP, Enhanced Peripheral (SPRU302)
*************************************************************************
*************************************************************************
** Include Statements
*************************************************************************
      .global    _c_int00
      .global    _McBSP0RXISR, _McBSP1TXISR
      .sect   ”.vectors”
*************************************************************************
**  Unmaskable Interrupts
*************************************************************************
RESET:      BD    _c_int00          ; HW/SW RESET vector
            NOP
            NOP
NMI:        RETE                    ; ~NMI, Non–Maskable interrupt
            NOP
            NOP
            NOP
*************************************************************************
**  S/W Interrupts
*************************************************************************
SINT17:     RETE
            NOP
            NOP
            NOP
SINT18:     RETE
            NOP
            NOP
            NOP
SINT19:     RETE
            NOP
            NOP
```

```
            NOP
SINT20:     RETE
            NOP
            NOP
            NOP
SINT21:     RETE
            NOP
            NOP
            NOP
SINT22:     RETE
            NOP
            NOP
            NOP
SINT23:     RETE
            NOP
            NOP
            NOP
SINT24:     RETE
            NOP
            NOP
            NOP
SINT25:     RETE
            NOP
            NOP
            NOP
SINT26:     RETE
            NOP
            NOP
            NOP
SINT27:     RETE
            NOP
            NOP
            NOP
SINT28:     RETE
            NOP
            NOP
            NOP
SINT29:     RETE
            NOP
            NOP
            NOP
SINT30:     RETE
            NOP
            NOP
            NOP
*************************************************************************
**   Rest of the Interrupts
*************************************************************************
INT0:       RETE                    ; external user interrupt #0
            NOP
            NOP
            NOP
INT1:       RETE                    ; external user interrupt #1
```

```
            NOP
            NOP
            NOP
INT2:       RETE                    ; external user interrupt #2
            NOP
            NOP
            NOP
TINT0:      RETE                    ; Timer0 interrupt
            NOP
            NOP
            NOP
BRINT0:     BD    _McBSP0RXISR      ; McBSP#0 receive interrupt
            NOP
            NOP
BXINT0:     RETE                    ; McBSP#0 transmit interrupt
            NOP
            NOP
            NOP
DMAC0:      RETE                    ; DMA channel0 interrupt
            NOP
            NOP
            NOP
TINT1:      RETE                    ; Timer1 interrupt
            NOP
            NOP
            NOP
INT3:       RETE                    ; external user interrupt #3
            NOP
            NOP
            NOP
HPINT:      RETE                    ; HPI interrupt
            NOP
            NOP
            NOP
BRINT1:     RETE                    ; McBSP#1 receive interrupt
            NOP
            NOP
            NOP
BXINT1:     BD    _McBSP1TXISR      ; McBSP#1 transmit interrupt
            NOP
            NOP
DMAC4:      RETE                    ; DMA channel4 interrupt
            NOP
            NOP
            NOP
DMAC5:      RETE                    ; DMA channel5 interrupt
            NOP
            NOP
            NOP
        .end
*************************************************************************
**  End of File -- C5402Vec.asm
*************************************************************************
```

Appendix G Linker Command Program

```
/************************************************************************
**  File Name:   AIC10EVM.cmd
**  Part Number: TLV320AIC10/11EVM–SW–00301
************************************************************************
**  Copyright (c) Texas Instruments, Inc. 2000
************************************************************************
**
**  Release History:
**     Version       Date           Engr          Description
**      1.00      10–11–2000      Wendy X Fang   Original Release
**
************************************************************************
**
**  Function:
**     This is the TMS320C54xx DSK linker command and memory mapping
**     file, which consists of programs for Linking (options have been
**     defined through Code–Composer); and maps DSP memory and its
**     sections.
**
************************************************************************/
/************************************************************************
** Link Files
************************************************************************/
/******* Basic DSP & AIC10 Management/Setup Files *******/
AIC10Main.obj                       /* DSP main code for AIC10EVM */
InitC5402.obj                       /* C5402 DSP initialization */
InitAIC10.obj                       /* AIC10 initialization */
LoopControl.obj                     /* main program loop control */
/******* Application Support Program *******/
C5402Vec.obj                        /* C5402 interrupt vector table */
IntrSrvc.obj                        /* DSP Interrupt service routines */
/******* Application Programs *******/
DTMFTone.obj                        /* DTMF tone generation */
SecurityVoice.obj                   /* security voice system */
EchoCancel.obj                      /* echo cancellation */
VoiceOverIP.obj                     /* voice over IP */
/************************************************************************
**  Memory Map and Section Definition
************************************************************************/
MEMORY
{
  PAGE 0:                           /* program space */
    VECS: origin = 0x0080, length = 0x0080 /* 128bytes vector table space */
    PROG: origin = 0x0100, length = 0x1F00 /* 8K program memory space */
  PAGE 1:                           /* data space */
    SCRA: origin = 0x0060, length = 0x0020 /* scratch pad mem space */
    STCK: origin = 0x2000, length = 0x0400 /* 1K words for stack  */
    DAT1: origin = 0x2400, length = 0x0100 /* 256 words for sys data */
    DAT2: origin = 0x2500, length = 0x1B00 /* 12K words for appl data */
}
SECTIONS
```

```
{
    .vectors  : {} > VECS  PAGE 0   /* interrupt vector table */
    .text     : {} > PROG  PAGE 0   /* program code */
    .data     : {} > PROG  PAGE 0   /* initialized data */
    .coeffs   : {} > PROG  PAGE 0   /* initialized parameters */
    .stack    : {} > STCK  PAGE 1   /* software stack section */
    .variable : {} > DAT1  PAGE 1   /* uninitialized vars for DSP&AIC10 */
    .bss      : {} > DAT2  PAGE 1   /* uninitialized vars for applications */
}
/************************************************************************
**  End of File -- AIC10EVM.cmd
************************************************************************/
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's 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, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265