TEXAS INSTRUMENTS

Application Report

Lit. Number – Month Year

Getting Started With TMS570LS Microcontrollers

Sunil Oak *AEC-Automotive Microcontrollers*

ABSTRACT

This application report provides a brief overview of the TMS570LS series of microcontrollers. It describes the initialization procedure for the TMS570LS series of microcontrollers. The document also shows code fragments from source files that are generated using the HALCoGen tool. All code constructs used in this document are also defined in header files also generated by the same utility.

Contents

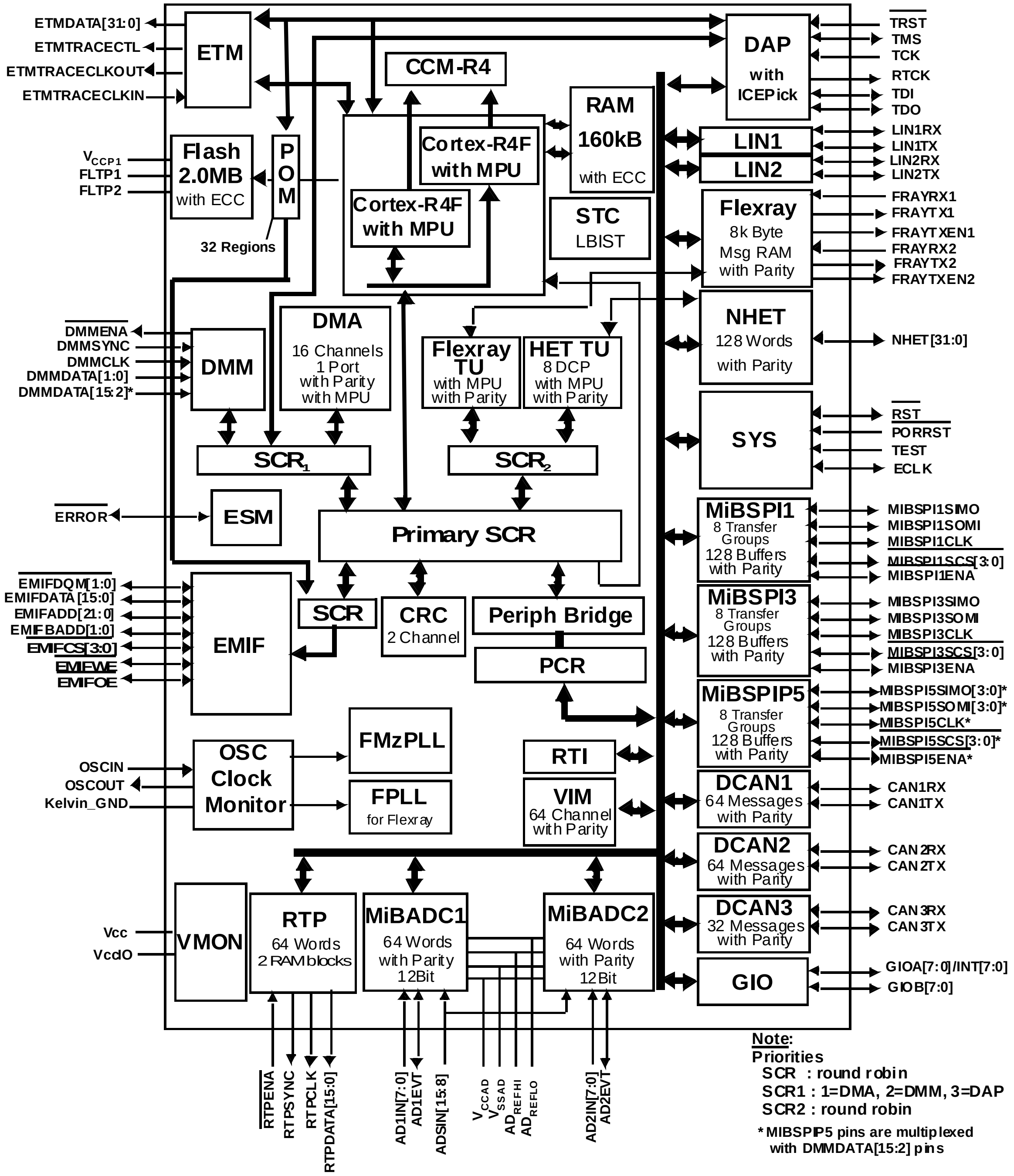
Figures

Tables

1 Block Diagram

Figure 1 shows a high-level block diagram of the TMS570LS series microcontroller.

Figure 1. Device Block Diagram



Note:
Priorities
SCR : round robin
SCR1 : 1=DMA, 2=DMM, 3=DAP
SCR2 : round robin
*MIBSPIP5 pins are multiplexed with DMMDATA[15:2] pins

2 Overview of Features

The TMS570LS series is a high-performance microcontroller family which has been certified for use in IEC 61508 SIL3 safety systems.

The microcontrollers included in the TMS570LS Series are:

- TMS570LS20216
- TMS570LS20206
- TMS570LS10216
- TMS570LS10206
- TMS570LS10116
- TMS570LS10106

TMS570LS Series microcontrollers contain the following:

- Dual ARM Cortex™-R4F processors operating in lock-step
  - 1.6 DMIPS/MHz
  - CPU clock speeds up to 160MHz, providing up to 250 DMIPS
  - Big-Endian
  - Core-Compare Module (CCM-R4) to monitor lock-step operation
  - Supports Built-In Self-Test (BIST)
- Integrated Memory
  - Up to 2MB Flash Memory
    - Non-volatile, electrically-erasable and programmable with 64-bit data bus interface
    - Single 3.3V supply needed for all read, program and erase operations
    - Connected to Tightly-Coupled Memory interface (TCM A) of the Cortex™-R4F CPU
    - Protects accesses with ECC (Single Error Correction Double Error Detection) logic
    - Supports accesses up to 160MHz in pipelined mode
  - Up to 160KB SRAM
    - Supports single-cycle accesses at the maximum specified clock speed
    - Allows read/write accesses in byte, half-word (16 bits), and word (32 bits) sizes
    - Protects accesses with ECC (Single Error Correction Double Error Detection) logic
- External Memory Interface (EMIF)
  - Provides 8- or 16-bit interface to external asynchronous memories, 22 address lines
  - Supports up to 4 chip selects
  - Each chip select can address up to 32MB in 16-bit mode or 16MB in 8-bit mode

- TMS570 High-Performance Architecture for Safety-Critical Applications
  - Consistent memory map across entire microcontroller family
  - Real-Time Interrupt (RTI) for use as OS timer
    - Up to 6 configurable periodic interrupts
  - Vectored Interrupt Manager (VIM)
    - Supports vectored interrupt controller (NVIC) port on the Cortex™-R4F CPU
    - Reduces interrupt latency even if NVIC port is not used
  - 2-Channel Cyclic Redundancy Checker (CRC)
    - Allows integrity checks for memories
- Direct Memory Access (DMA) controller
  - 16 channels
  - 32 hardware requests configured by control packets
  - Control packet information protected by parity error detection
  - Built-in Memory Protection Unit (MPU) dedicated to prevent erroneous DMA transfers
- Enhanced High-End Timer (NHET)
  - Programmable timer coprocessor with its own assembly microcode
  - 128-word instruction memory with parity protection
  - 32 programmable channels for output compare or input capture functions
  - Unused channels can be used as general-purpose I/O signals
  - Dedicated transfer unit (HTU) with 8 programmable transfer configurations
  - Control packet information protected by parity error detection
  - Built-in Memory Protection Unit (MPU) dedicated to prevent erroneous HTU transfers
- Analog-to-Digital Converters (ADC)
  - Two 12-bit-resolution Successive Approximation Resistor A2D converters
  - Dedicated memory to store up to 64 conversion results, with parity protection
  - 24 total input channels, 8 shared channels
- Communication Modules
  - Dual-channel FlexRay™ Controller
    - 8 KB message RAM with parity protection
    - Dedicated transfer unit (FTU)
    - Parity protection for transfer unit configuration memory
    - Dedicated MPU to protect against erroneous transfers by FTU
  - Three Multi-Buffered Serial Peripheral Interface (MibSPI) Controllers

    - □ 4 chip selects and 1 enable for each MibSPI
    - □ 128 transmit and receive buffers with parity protection for each MibSPI
    - □ One MibSPI supports unique parallel interface mode with up to 4-bit data
  - – Two Local Interconnect Network (LIN) Interface Controllers
    - □ Compliant to LIN specification version 2.0
    - □ Support the standard Serial Communication Interface (SCI) mode
  - – Three Controller Area Network (CAN) Controllers
    - □ Two controllers with 64 mailboxes, one controller with 32 mailboxes
    - □ Parity protection for mailbox memory

- Trace and Calibration Interfaces
  - – Embedded Trace Macrocell (ETM-R4) for Cortex™-R4F
    - □ Provides trace for instruction and data accesses by the Cortex™-R4F CPU
  - – Parameter Overlay Module (POM)
    - □ Re-routes flash accesses to external memory via the EMIF
    - □ Allows application to update parameters without re-programming the flash
  - – Data Modification Module (DMM)
    - □ Provides ability to modify data anywhere within the 4GB addressable space
  - – RAM Trace Port (RTP)
    - □ Provides a high-speed trace of RAM accesses

3 Standard Initialization Sequence for TMS570LS Microcontrollers

A typical basic sequence to be followed for initialization and configuration of key features on the TMS570LS Series microcontrollers is summarized below and detailed in the following sections. This is not a mandatory sequence. Applications that are non-safety-critical can choose to not use the ECC feature for flash and RAM accesses, for example.

- Enable the Floating Point Unit (FPU) inside the Cortex-R4F CPU
- Initialize the CPU registers and FPU registers
- Initialize stack pointers for all operating modes
- Enable CPU's dedicated vectored interrupt controller (VIC) port
- Set up flash wrapper for required wait states and pipelined mode
- Set up flash bank and pump power modes
- Configure PLL control registers
- Enable desired clock sources
- Map device clock domains to desired clock sources
- Run the Built-In Self-Test for the CPU
- Release peripherals from reset and enable clocks to all peripherals
- Run self-tests on all device memories using Programmable Built-In Self-test (PBIST)
- Perform auto-initialization for all on-chip SRAMs
- Program Vectored Interrupt Manager memory to map all interrupt service routine addresses
- Configure IRQ / FIQ interrupt priorities for all interrupt channels
- Enable the desired interrupts
- Initialize copy table, global variables, and constructors
- Call the main application

The following sections describe each of the above steps in more detail. Code examples are also provided.

3.1 Enabling Floating Point Coprocessor (FPU)

The floating point coprocessor is disabled upon a CPU reset and must be enabled.

- First access to the FPU must be enabled.

```
        MRC p15, #0x0, r0, c1, c0, #2
        MOV r3, #0xf00000
        ORR r0, r0, r3
        MCR p15, #0x0, r0, c1, c0, #2
        MRC p15, #0x0, r0, c0, c0
```

- Enable the FPU

```
        MOV r0, #0x40000000
        FMXR FPEXC, r0
```

3.2 Initialization of Cortex-R4F Registers

The TMS570LS series of microcontrollers include dual Cortex-R4F CPUs running in a lock-step operation mode. A Core Compare Module (CCM-R4) compares the output signals from each R4F CPU. Any difference in the two CPUs' outputs is flagged as a fault of a high severity level. The CPU internal core registers need to be initialized to a predefined state in order to prevent an error from being flagged during system initialization.

- Initialize CPU registers R0 through R12

```
        mov   r0,         lr
        mov   r1,         #0x03D0
        mov   r2,         #0x0000
        mov   r3,         #0x0000
        mov   r4,         #0x0000
        mov   r5,         #0x0000
        mov   r6,         #0x0000
        mov   r7,         #0x0000
        mov   r8,         #0x0000
        mov   r9,         #0x0000
        mov   r10,        #0x0000
        mov   r11,        #0x0000
        mov   r12,        #0x0000
```

- Initialize CPU banked registers

```
        orr   r13,        r1,     #0x0001
        msr   cpsr_cxsf,  r13
        msr   spsr_cxsf,  r13
        mov   lr,         r0
        mov   r8,         #0x0000
        mov   r9,         #0x0000
        mov   r10,        #0x0000
        mov   r11,        #0x0000
        mov   r12,        #0x0000
        orr   r13,        r1,     #0x0002
        msr   cpsr_c,     r13
        msr   spsr_cxsf,  r13
        mov   lr,         r0
        orr   r13,        r1,     #0x0007
        msr   cpsr_c,     r13
        msr   spsr_cxsf,  r13
        mov   lr,         r0
        orr   r13,        r1,     #0x000B
        msr   cpsr_c,     r13
        msr   spsr_cxsf,  r13
        mov   lr,         r0
        orr   r13,        r1,     #0x0003
        msr   cpsr_c,     r13
        msr   spsr_cxsf,  r13
```

- Initialize the FPU registers

```
        fmdrr d0,          r1,      r1
        fmdrr d1,          r1,      r1
        fmdrr d2,          r1,      r1
        fmdrr d3,          r1,      r1
        fmdrr d4,          r1,      r1
        fmdrr d5,          r1,      r1
        fmdrr d6,          r1,      r1
        fmdrr d7,          r1,      r1
        fmdrr d8,          r1,      r1
        fmdrr d9,          r1,      r1
        fmdrr d10,         r1,      r1
        fmdrr d11,         r1,      r1
        fmdrr d12,         r1,      r1
        fmdrr d13,         r1,      r1
        fmdrr d14,         r1,      r1
        fmdrr d15,         r1,      r1
```

- Initialize CPU return stack

```
        bl    $+4
        bl    $+4
        bl    $+4
        bl    $+4
        bx    r0
```

3.3 Initialize stack pointers for all CPU operating modes

Define the base addresses for the stacks used for the different operating modes. The addresses listed below are only examples and can be defined by the application as required.

```
user:   .word 0x08000000+0x00001000
svc:    .word 0x08000000+0x00001000+0x00000100
fiq:    .word 0x08000000+0x00001000+0x00000100+0x00000100
irq:    .word 0x08000000+0x00001000+0x00000100+0x00000100+0x00000100
abort:  .word 0x08000000+0x00001000+0x00000100+0x00000100+0x00000100+0x00000100
undef:  .word
0x08000000+0x00001000+0x00000100+0x00000100+0x00000100+0x00000100+0x00000100
```

The Cortex-R4F CPU can operate in one of several modes:

- *User mode (USR)* is the usual mode for the execution of ARM or Thumb programs. It is used for executing most application programs. Many control registers on the TMS570LS microcontroller are not writable in user mode.

```
        msr   cpsr_c,   #0xDF
        ldr   sp,       user
```

- *Fast interrupt mode (FIQ)* is entered upon taking a fast interrupt.

```
        msr   cpsr_c,   #0xD1
        ldr   sp,       fiq
```

- *Interrupt mode (IRQ)* is entered on taking a normal interrupt.

```
        msr   cpsr_c,   #0xD2
        ldr   sp,       irq
```

- *Abort mode (ABT)* is entered after a data or instruction abort.
- *Undefined mode (UND)* is entered when an undefined instruction exception occurs.

```
        msr   cpsr_c,   #0xDB
        ldr   sp,       undef
```

- *System mode (SYS)* is a privileged mode for the operating system. This is also the default mode of the CPU after a CPU reset.
- *Supervisor mode (SVC)* is a protected mode for the operating system and is entered upon taking a Supervisor Call (SVC).

```
        msr   cpsr_c,   #0xD3
        ldr   sp,       svc
```

The application can initialize the stack pointers in the above sequence. This will leave the CPU in the Supervisor (SVC) mode once the stack pointers are initialized.

3.4 Enable the Cortex-R4F CPU’s Vectored Interrupt Controller (VIC) Port

The CPU has a dedicated port that enables the Vectored Interrupt Manager (VIM) module to supply the address of an interrupt service routine along with the interrupt (IRQ) signal. This provides faster entry into the interrupt service routine versus the CPU having to decode the pending interrupts and identify the highest priority interrupt to be serviced first.

The VIC port is disabled upon any CPU reset and must be enabled by the application. The VIC is enabled by setting the VE bit in the CPU’s System Control Register, as shown below.

```
        mrc   p15, #0, r0,         c1, c0,  #0
        orr   r0,  r0,    #0x01000000
        mcr   p15, #0, r0,         c1, c0,  #0
```

3.5 Configure Flash Access

The flash memory on the TMS570LS series microcontrollers is a non-volatile electrically erasable and programmable memory.

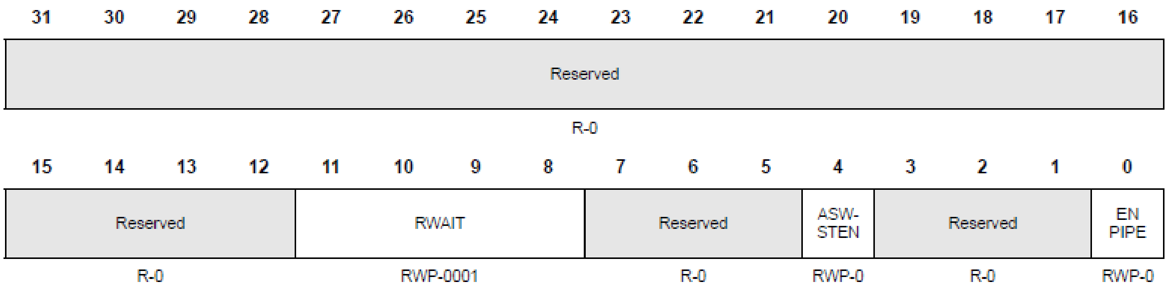
The TMS570LS microcontrollers contain a digital wrapper module that manages all accesses to the flash memory. A flash access can be completed without any wait states required for bus master clock speeds up to 36MHz. If the bus clock is faster than 36MHz, then any flash access requires the appropriate number of wait states depending on the bus clock speed. The TMS570Ls series microcontrollers support clock speeds up to 160MHz. Please refer to the device datasheet for the actual maximum allowed speed as it varies per the package type.

Suppose that the application requires the microcontroller to run at the maximum supported speed of 160MHz. This requires 1 address wait state and 3 data wait states for any access to the flash memory. These wait states need to be configured in the flash wrapper registers.

The flash wrapper also features a special pipeline mode. When this mode is enabled, the wrapper reads 128 bits from the flash memory and holds them in buffers that the CPU can read from without any wait state. The CPU can read 32 or 64 bits of instructions or data from the pipeline buffers.

The register inside the flash wrapper that controls the wait states and the pipeline mode is shown below.

Figure 2. Flash Read Control Register: FRDCNTL, Address = 0xFFF87000



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				RWAIT				Reserved			ASW-STEN	Reserved			EN PIPE
R-0				RWP-0001				R-0			RWP-0	R-0			RWP-0

-*n* = Value after reset, R=Read, WP=Write in Privilege Mode

- The RWAIT field configures the number of data read wait states.
- The ASWSTEN field enables the generation of 1 address wait state. The address bus is latched one cycle before it is decoded for a pipeline hit or miss.
- The ENPIPE field is sued to enable or disable the pipeline mode of the flash wrapper.

The sequence to configure the wait states and to enable the pipeline mode is as follows.

```
    flashWREG->FRDCNTL =  0x01000000U
                       | (3U << 8U)          // 3 data wait states
                       | (1U << 4U)          // 1 address wait state enabled
                       |  1U;                // Enable pipeline mode
```

3.6 Configure flash bank and pump power modes

The flash banks and pump used on the TMS570LS series microcontrollers support three different operating modes to optimize power consumption.

a. Active mode

□ Flash bank sense amplifiers and sense reference are enabled

□ All circuits of flash charge pump are enabled

b. Standby mode (only for flash banks)

□ Flash bank sense reference is enabled but sense amplifiers are disabled

c. Sleep Mode

□ Flash bank sense amplifiers and sense reference are disabled

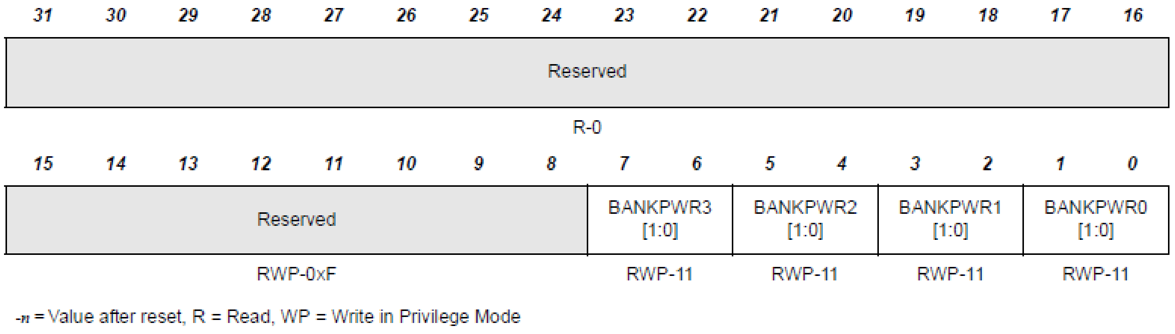
□ All circuits of flash charge pump are disabled

The flash banks and charge pump are in the active state by default and after any system reset. The flash wrapper allows the application to configure "fall back" power states for the flash banks and charge pump. The flash banks and pump automatically switch the power mode to the selected fall back state when there is no access to the flash banks detected within a user-configurable time.

The flash wrapper also contains special timers to automatically sequence the flash banks and pump between the active and the selected fall-back states. A read access to any flash bank which is in a non-active power state will "wake up" both the selected bank and the charge pump to active power state. Programming and erase operations are only allowed on banks in active state.

The flash wrapper register that controls the flash banks' power states is shown below.

Figure 3. Flash Bank Fall-Back Control Register: FBFALLBACK, Address = 0xFFF87040



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved								BANKPWR3 [1:0]		BANKPWR2 [1:0]		BANKPWR1 [1:0]		BANKPWR0 [1:0]	
RWP-0xF								RWP-11		RWP-11		RWP-11		RWP-11	

-*n* = Value after reset, R = Read, WP = Write in Privilege Mode

Each of the BANKPWRx fields configures the fall-back mode for a single flash bank. The TMS570LS microcontrollers support up to 4 flash banks.

Configuration of fall back mode for the flash banks:

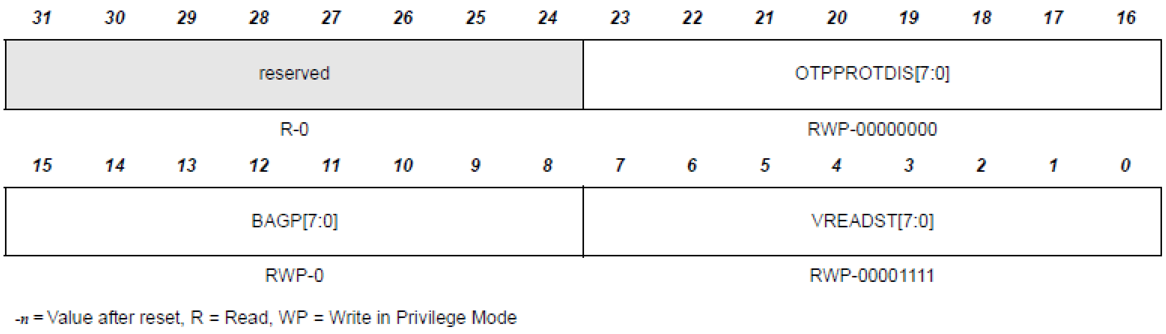
```
enum flashWPowerModes
{
    SYS_SLEEP   = 0U, /** Flash bank power mode sleep   */
    SYS_STANDBY = 1U, /** Flash bank power mode standby */
    SYS_ACTIVE  = 3U  /** flash bank power mode active  */
};

flashWREG->FBFALLBACK = 0x00000000
                      | (SYS_SLEEP << 6U)       // Bank3 falls back to SLEEP
                      | (SYS_SLEEP << 4U)       // Bank2 falls back to SLEEP
                      | (SYS_SLEEP << 2U)       // Bank1 falls back to SLEEP
                      |  SYS_SLEEP;             // Bank0 falls back to SLEEP
```

The above code fragment configures even the fall-back mode for each available flash bank to be the sleep mode. The application can choose to configure these modes differently as required. The power savings can be disabled completely by selecting the active state to also be the fall-back power state, which is the default.

There are a few other registers that control the timing sequence for entry to a fall-back mode and wake up to active mode. These are described now.

Figure 4. Flash Bank Access Control Register: FBAC, Address = 0xFFF8703C



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved								OTPPROTDIS[7:0]							
R-0								RWP-00000000							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BAGP[7:0]								VREADST[7:0]							
RWP-0								RWP-00001111							

-*n* = Value after reset, R = Read, WP = Write in Privilege Mode

- The OTPPROTDIS field is not relevant to power modes.
- The BAGP field configures the flash banks’ Active Grace Period (AGP). This is the starting count value for a down-counter. An access to a flash bank before this counter counts down to 0 causes a reload of this counter to the configured AGP value. In effect, the AGP delays the flash banks’ entry into the selected fall-back mode by 0 to 255 HCLK/16 cycles. This value must be greater than 1 when the fall-back mode is not ACTIVE.
- The VREADST field controls the delay, in terms of HCLK cycles, between the time when the charge pump generates the required read voltage (VREAD) and the time when the flash bank starts its own power up sequence.

NOTE: The flash banks have hard-coded timings for transitioning from sleep to standby to active power states. These timings are not configurable by the application.

```
flashWREG->FMAC = 0x00000003;      // Select flash bank3
flashWREG->FBAC |= 0x0000FF00;     // Select 255 HCLK/16 cycles as the bank3 AGP

flashWREG->FMAC = 0x00000002;      // Select flash bank2
flashWREG->FBAC |= 0x0000FF00;     // Select 255 HCLK/16 cycles as the bank2 AGP

flashWREG->FMAC = 0x00000001;      // Select flash bank1
flashWREG->FBAC |= 0x0000FF00;     // Select 255 HCLK/16 cycles as the bank1 AGP

flashWREG->FMAC = 0x00000000;      // Select flash bank0
flashWREG->FBAC |= 0x0000FF00;     // Select 255 HCLK/16 cycles as the bank0 AGP
```

Figure 5. Flash Pump Access Control Register 1: FPAC1, Address = 0xFFF87048

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved					PSLEEP[10:0]										
R-0					RWP-0x64										
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved															PUMP PWR
R-0															RWP-1

-n = Value after reset, R = Read, WP = Write in Privilege Mode

- The PSLEEP field configures the time that the flash pump takes for transitioning from the sleep state to the standby state. This is specified in terms of HCLK/2 cycles. Please check the TMS570LS datasheet to identify the minimum time required for the flash pump to switch from the sleep state to the standby state.

- The PUMPPWR field defines whether the flash pump falls back into sleep mode, or remains active.

```
flashWREG->FPAC1 = 0x00640000;     // PSLEEP = 100 HCLK/2 cycles,
                                   // Pump fall-back state = SLEEP
```

Figure 6. Flash Pump Access Control Register 2: FPAC2, Address = 0xFFF8704C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PAGP[15:0]															
RWP-0															

-n = Value after reset, R = Read, WP = Write in Privilege Mode

- The PAGP field defines the active grace period for the flash charge pump. This defines the starting count for a down counter. An access to the flash memory reloads this counter with the selected PAGP value. After the last access to flash memory, the down counter delays the flash pump's entry to the selected fall-back mode by 0 to 65536 HCLK/16 cycles.

```
flashWREG->FPAC2 = 0x000000FF;     // PSLEEP = 255 HCLK/16 cycles
```

3.7 Configure PLLs

The TMS570LS series of microcontrollers contain a Frequency-Modulated Phase-Locked-Loop (FMzPLL) macro that allows the input oscillator frequency to be multiplied to a higher frequency than can be conveniently achieved with an external resonator or crystal. Additionally the FMzPLL allows the flexibility to generate many different frequency options from a fixed crystal or resonator.

The FMzPLL allows the application to superimpose a "modulation frequency" signal on the selected base frequency signal output from the FMzPLL. This reduces the electromagnetic energy of the output signal by spreading it across a controlled frequency range around the base frequency. This mode is disabled by default, and the application can enable it in applications sensitive to noise emissions.

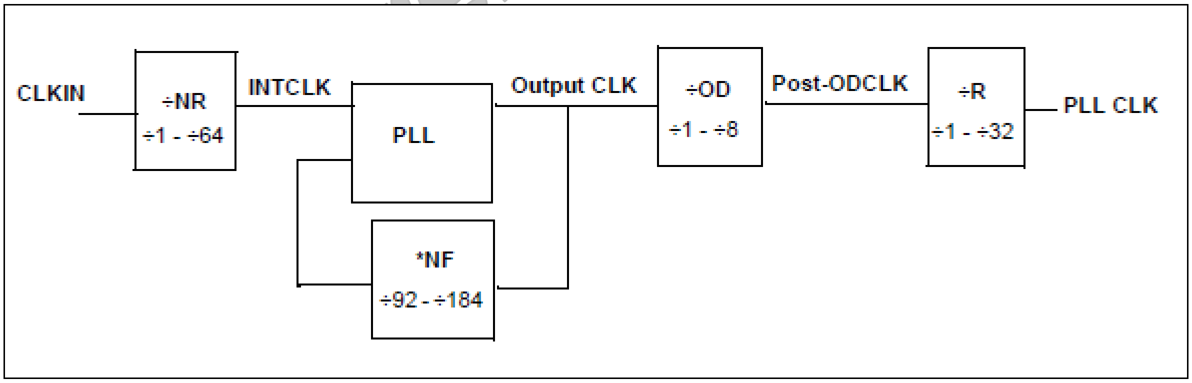
The TMS570LS microcontrollers also contain a second non-modulating PLL macro. This is the FPLL macro. The FPLL can be independently configured to generate a second high-frequency clock source for specific uses.

3.7.1 FMzPLL

3.7.1.1 FMzPLL Block Diagram

Figure 7 shows a high-level block diagram of the FMzPLL macro.

Figure 7. FMzPLL Block Diagram



- The oscillator circuit drives an external crystal or resonator, which generates the reference input clock (*CLKIN*). The FMzPLL macro supports CLKIN from 5MHz to 20MHz.
- The FMzPLL divides down this CLKIN by a value, NR, such that the divided clock (*INTCLK*) is between 1.63MHz and 6.53MHz. NR can be between 1 and 64.
- The FMzPLL multiplies INTCLK by a value NF, such that the multiplication result (*VCOCLK*) is between 120MHz and 500MHz. NF can be between 92 and 184.
- The VCOCLK is divided down by a value OD. The divided clock (*Post-ODCLK*) frequency must not exceed the maximum device frequency. OD can be between 1 and 8.

- The post-ODCLK is further divided by a value R to generate the FMzPLL output clock (*PLLCLK*). The PLLCLK frequency must not exceed the device maximum frequency specified in the datasheet.

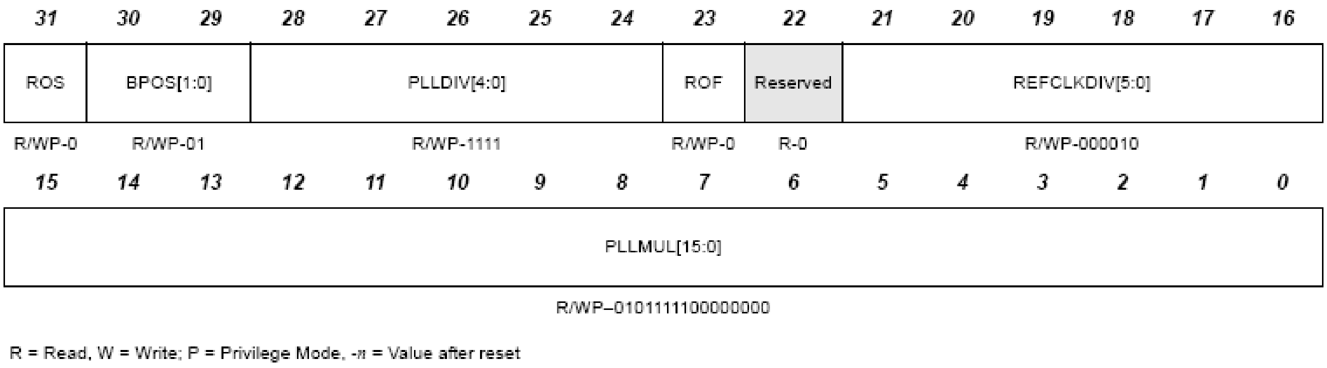
3.7.1.2 FMzPLL Slip Detector

The FMzPLL macro has a slip detector circuit that compares the CLKIN to the VCOCLK and flags any 2-cycle slips. The application can choose the response to a PLL slip indication from among 3 choices: do nothing, or cause a system reset, or bypass the FMzPLL such that the CLKIN frequency itself is supplied as the output from the FMzPLL macro.

3.7.1.3 FMzPLL Configuration

The FMzPLL has two registers (PLLCTL1 and PLLCTL2) located within the System module on the TMS570LS microcontrollers. These are described now.

Figure 8. PLL Control Register 1: PLLCTL1, Address = 0xFFFFFF70



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ROS	BPOS[1:0]		PLLDIV[4:0]					ROF	Reserved	REFCLKDIV[5:0]					
R/WP-0	R/WP-01		R/WP-1111					R/WP-0	R-0	R/WP-000010					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PLLMUL[15:0]															
R/WP–0101111100000000															

R = Read, W = Write; P = Privilege Mode, -*n* = Value after reset

- Reset-On-Slip (ROS) selects whether a PLL slip condition causes a system reset or not.
  - ROS = 1 causes a system reset when a PLL slip is flagged.

NOTE: The Bypass On Slip (BPOS) functionality must be disabled to use the ROS functionality.

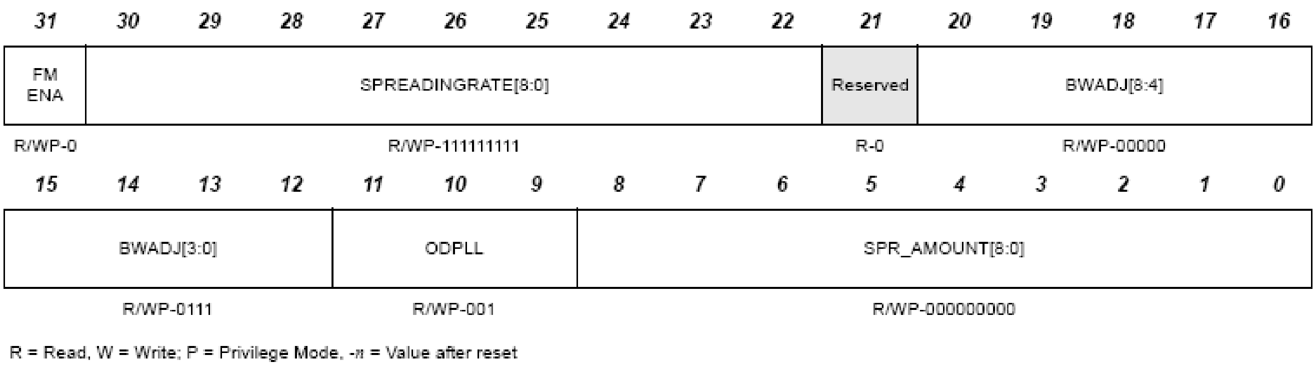
- Bypass-On-Slip (BPOS) defines the PLL behavior when a slip is flagged.
  - BPOS = 10 ignores a PLL slip condition flagged by the FMzPLL macro.
  - Writing any other value to BPOS causes the FMzPLL to be bypassed so that the CLKIN is used as the output from the FMzPLL macro.

NOTE: If the ROS bit is also ‘1’ when the FMzPLL is bypassed, then a system reset occurs and the FMzPLL output is not bypassed.

- PLLDIV defines the R-divider.
  - R = PLLDIV + 1
  - $f_{PLLCLK} = f_{post\text{-}ODCLK} / R$
- Reset-on-Oscillator-Fail (ROF) controls the response to an oscillator failure detected by the clock monitor and is not relevant to the PLL configuration discussion.

- REFCLKDIV defines the NR-divider.
  - NR = REFCLKDIV + 1
  - $f_{INTCLK} = f_{CLKIN} / NR$
- PLLMUL defines the NF multiplier
  - NF = (PLLMUL / 256) + 1
  - $f_{VCOCLK} = f_{INTCLK} * NF$

Figure 9. PLL Control Register 2: PLLCTL2, Address = 0xFFFFFF74



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FM ENA	SPREADINGRATE[8:0]									Reserved	BWADJ[8:4]				
R/WP-0	R/WP-111111111									R-0	R/WP-00000				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BWADJ[3:0]				ODPLL			SPR_AMOUNT[8:0]								
R/WP-0111				R/WP-001			R/WP-000000000								

R = Read, W = Write; P = Privilege Mode, -n = Value after reset

- Setting the FM ENA bit enables the modulation frequency to be superimposed on the output of the FMzPLL macro.
- SPREADINGRATE defines the modulation frequency used.
  - NS = SPREADINGRATE + 1
  - Modulation frequency, $f_{mod} = fs = f_{INTCLK} / (2 * NS)$
- BWADJ defines the FMzPLL modulation bandwidth adjustment.
  - NB = BWADJ + 1
  - $f_{BW} = f_{nom\_BW} / NB$

NOTE: NB must be set to 7 when modulation is not used. This is also the default value.

- ODPLL defines the OD-divider.
  - OD = ODPLL + 1
  - $f_{post\text{-}ODCLK} = f_{VCOCLK} / OD$
- SPR_AMOUNT defines the frequency modulation depth divider.
  - NV = SPR_AMOUNT + 1

3.7.1.4 Example FMzPLL configuration

```
    systemREG1->PLLCTL1 =  0x00000000U
                        |  0x20000000U          // No reset on slip, bypass on slip
                        | (0U << 24U)           // R = 1
                        | (5U << 16U)           // NR = 6
                        | (119U  << 8U);        // NF = 120

    systemREG1->PLLCTL2 = 0x00000000U           // Modulation disabled
                        | (255U << 22U)         // NS = 256
                        | (7U << 12U)           // NB = 8
                        | (1U << 9U)            // OD = 2
                        |  61U;                 // NV = 62
```

This example configuration results in a FMzPLL output clock frequency of:

$f_{PLLCLK} = (f_{CLKIN} / 6) * 120 / 2 / 1 = f_{CLKIN} * 10$

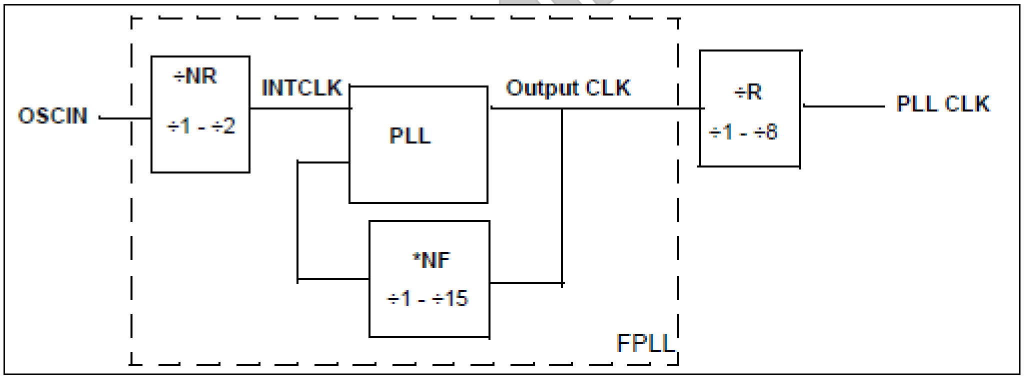
TI provides a GUI-based utility that allows the user to easily calculate the PLL control register settings.

This utility can be found at: http://focus.ti.com/docs/toolsw/folders/print/fmzpll_calculator.html

3.7.2 FPLL

3.7.2.1 FPLL Block Diagram

Figure 10. FPLL Block Diagram

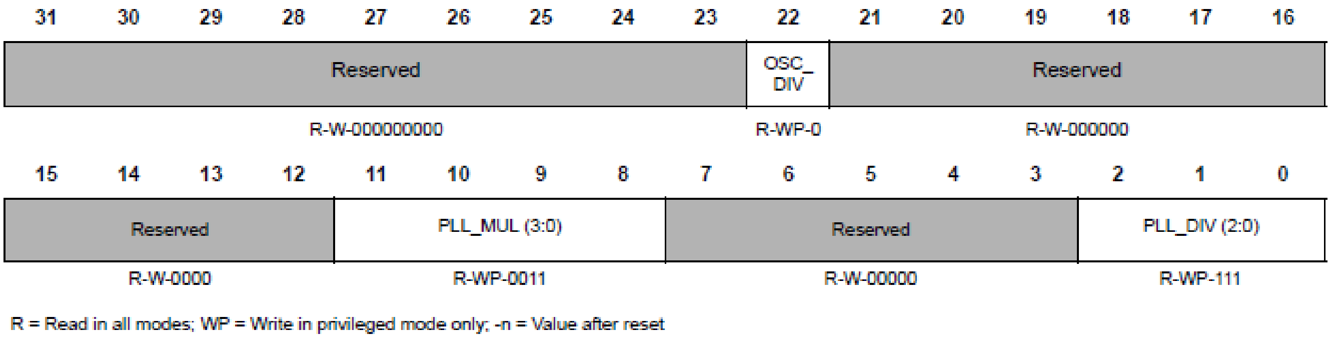


- The FPLL generates a clock output signal using an external resonator or crystal.
- The NR-divider divides the input frequency OSCIN by 1 or 2 to generate the PLL internal clock INTCLK.
- INTCLK is multiplied by a programmable value NF to generate an output clock. The frequency of this output clock must be between 10MHz and 100MHz. The value of NF ranges from 1 to 15.
- The output clock is subsequently divided by an R-divider, which ranges from 1 to 8.
- The output of the R-divider is available for use as a clock source for any device clock domain.

3.7.2.2 FPLL Configuration

The FPLL is configured using the PLL control register 3 in the system module of the TMS570LS microcontroller.

Figure 11. PLL Control Register 3: PLLCTL3, Address = 0xFFFFE100



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									OSC_ DIV	Reserved					
R-W-000000000									R-WP-0	R-W-000000					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				PLL_MUL (3:0)				Reserved					PLL_DIV (2:0)		
R-W-0000				R-WP-0011				R-W-00000					R-WP-111		

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- OSC_DIV defines the NR-divider of the FPLL.
  - NR = OSC_DIV + 1
  - $f_{INTCLK} = f_{OSCIN} / NR$
- PLL_MUL defines the NF-multiplier of the FPLL.
  - NF = PLL_MUL + 1
  - $f_{Output\_CLK} = f_{INTCLK} * NF$
- PLL_DIV defines the R-divider of the FPLL
  - R = PLL_DIV + 1
  - $f_{PLLCLK} = f_{Output\_CLK} / R$

3.7.2.3 Example FPLL Configuration

```
    systemREG1->PLLCTL3 =  0x00000000U
                          | (1U << 22U)           // NR = 2
                          | (5U << 8U)            // NF = 6
                          | 0U                    // R = 1
```

This example configuration generates an output clock frequency of:

$f_{PLLCLK} = (f_{OSCIN} / NR) * NF / R = (f_{OSCIN} / 2) * 6 / 1 = f_{OSCIN} * 3$

TI provides a GUI-based utility that allows the user to easily calculate the FPLL control register settings.

This utility can be found at: http://focus.ti.com/docs/toolsw/folders/print/fpll_calculator.html

3.8 Enable Clock Sources

3.8.1 Available Clock Sources on TMS570LS Microcontrollers

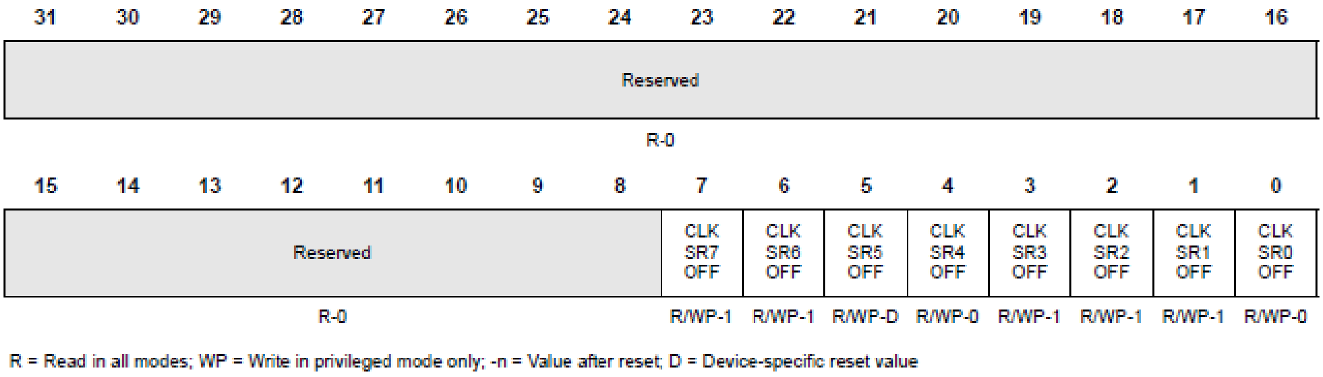
The TMS570LS microcontrollers support 5 different clock sources, as listed in the table below.

Table 1. Clock Sources on TMS570LS Microcontrollers

Clock Source Number	Clock Source Name	Description
0	OSCIN	This is the primary oscillator, typically driven by an external resonator or crystal. This is the only available input to the FMzPLL and the FPLL macros. The OSCIN frequency must be between 5MHz and 20MHz.
1	FMzPLL output	This is the output of the FMzPLL, which is generated using the OSCIN as the input clock. The FMzPLL output clock frequency must not exceed the maximum device frequency specified in the device datasheet. The FMzPLL features a modulation mode where a modulation frequency is superimposed on the FMzPLL output signal.
2	Not implemented	No clock signal is connected to source # 2. This clock source must not be enabled or chosen for any clock domain.
3	Not implemented	No clock signal is connected to source # 3. This clock source must not be enabled or chosen for any clock domain.
4	LF LPO	This is the low-frequency output of the internal reference oscillator. The LF LPO is typically an 80KHz signal, and is generally used for low power mode use cases.
5	HF LPO	This is the high-frequency output of the internal reference oscillator. The HF LPO is typically a 10MHz signal, and is used as a reference clock for monitoring the main oscillator.
6	FPLL output	This is the output of the FPLL, which is generated using the OSCIN as the input clock. The FPLL does not support modulation, and its output is typically used to support communication protocols with strict jitter requirements on the clock used to generate the baud rate.
7	Not implemented	No clock signal is connected to source # 7. This clock source must not be enabled or chosen for any clock domain.

3.8.2 Control Registers for Enabling and Disabling Clock Sources

Figure 12. Clock Source Disable Register: CSDIS, Address = 0xFFFFFF30



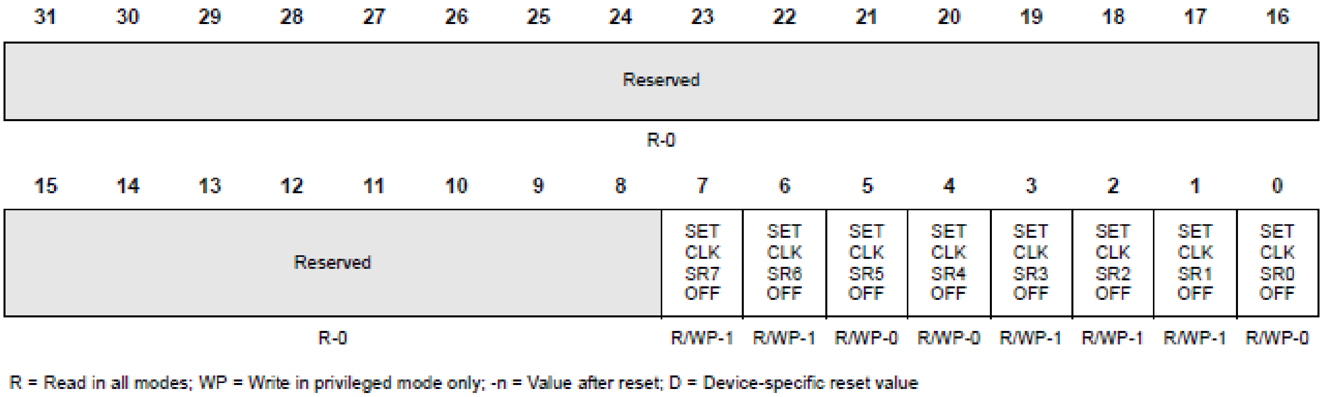
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CLK SR7 OFF	CLK SR6 OFF	CLK SR5 OFF	CLK SR4 OFF	CLK SR3 OFF	CLK SR2 OFF	CLK SR1 OFF	CLK SR0 OFF
R-0								R/WP-1	R/WP-1	R/WP-D	R/WP-0	R/WP-1	R/WP-1	R/WP-1	R/WP-0

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset; D = Device-specific reset value

- Each bit of the CSDIS controls the clock source of the same number: bit 0 controls clock source 0, bit 1 controls clock source 1, and so on.
- Figure 12 also shows the default states of the clock sources supported on the TMS570LS microcontrollers:
  - Clock sources 0, 4 and 5 are enabled, while clock sources 1 and 6 are disabled upon any system reset
  - Clock sources 2, 3 and 7 are not implemented and cannot be enabled in the application.
- Setting any bit commands the corresponding clock source to be disabled.
  - The clock source can only be disabled once there is no clock domain or secondary clock source (FMzPLL, FPLL) using the clock source to be disabled.

Figure 13. Clock Source Disable Set Register: CSDISSET, Address = 0xFFFFFF34



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								SET CLK SR7 OFF	SET CLK SR6 OFF	SET CLK SR5 OFF	SET CLK SR4 OFF	SET CLK SR3 OFF	SET CLK SR2 OFF	SET CLK SR1 OFF	SET CLK SR0 OFF
R-0								R/WP-1	R/WP-1	R/WP-0	R/WP-0	R/WP-1	R/WP-1	R/WP-1	R/WP-0

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset; D = Device-specific reset value

Figure 14. Clock Source Disable Clear Register: CSDISCLR, Address = 0xFFFFFF38

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CLR CLK SR7 OFF	CLR CLK SR6 OFF	CLR CLK SR5 OFF	CLR CLK SR4 OFF	CLR CLK SR3 OFF	CLR CLK SR2 OFF	CLR CLK SR1 OFF	CLR CLK SR0 OFF
R-0								R/WP-1	R/WP-1	R/WP-D	R/WP-0	R/WP-1	R/WP-1	R/WP-1	R/WP-0

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset; D = Device-specific reset value

- The system module also contains two additional registers that can be used to enable or disable clock sources. These registers are provided so that the application can avoid using read-modify-write operations for enabling or disabling clock sources.
- Setting any bit in the CSDISSET register commands the corresponding clock source to be disabled.
- Setting any bit in the CSDISCLR register enables the corresponding clock source.

3.8.3 Example Clock Source Configuration

```
    systemREG1->CSDISCLR = 0x00000000U
                         | 0x00000001U          // Enable clock source 0
                         | 0x00000002U          // Enable clock source 1
                         | 0x00000010U          // Enable clock source 4
                         | 0x00000020U          // Enable clock source 5
                         | 0x00000040U;         // Enable clock source 6
```

The above configuration enables clock sources 0, 1, 4, 5, and 6. The clock sources 2, 3 and 7 are not implemented on the TMS570LS microcontrollers and must be left disabled.

Of the clock sources that are enabled, number 0, 4 and 5 are enabled by default and will have become valid by the time the processor is released from reset upon a power-up. These are the main oscillator and the two outputs from the internal reference oscillator.

Clock source 1 and 6 are the two PLL outputs. The FMzPLL as well as the FPLL have a defined start-up time, and their outputs are not available for use until this time. The application must wait for the valid status flags for these clock sources to be set before using the PLL outputs for any clock domain.

```
while (!((systemREG1->CSVSTAT & 2)              // Wait for FMzPLL to become valid
       & (systemREG1-> CSVSTAT & 0x40)));       // Wait for FPLL to become valid
```

3.9 Clock Domains

There are multiple clock domains on the TMS570LS microcontrollers to ease the configuration and controllability of the different modules using these clock domains.

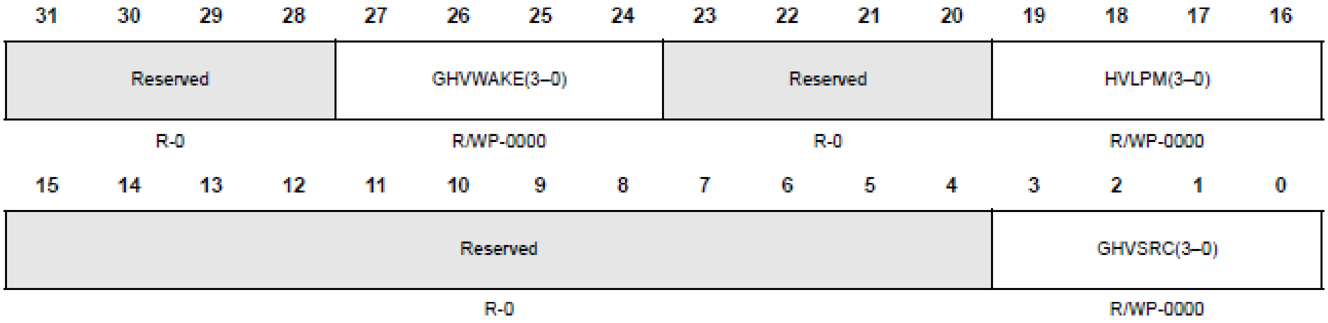
Table 2. Clock Domains on TMS570LS Microcontrollers

Domain Name	Clock Name	Comments
CPU clock domain	GCLK	GCLK controls all the CPU sub-systems, including the floating point unit (FPU), and the memory protection unit (MPU)
System bus clock domain	HCLK	HCLK shares the same clock source as GCLK, and is always the same frequency as HCLK.
System peripheral clock domain	VCLK_sys	VCLK_sys is used for the system modules such as VIM, ESM, SYS, etc. VCLK_sys is divided down from HCLK by a programmable divider from 1 to 16.
Peripheral clock domains	VCLK and VCLK2	VCLK is the primary peripheral clock, and is synchronous with VCLK_sys. VCLK2 is a secondary peripheral clock and is reserved for use by the enhanced timer module (NHET) and the associated transfer unit (HTU). VCLK2 is also divided down from HCLK by a programmable divider from 1 to 16. f_{HCLK} must be an integer multiple of f_{VCLK2}, f_{VCLK2} must be an integer multiple of f_{VCLK}.
Asynchronous clock domains	VCLKA1 and VCLKA2	These clock domains are reserved for use by special communication modules that have strict jitter constraints. The protocols for these communication modules (e.g. CAN, FlexRay) do not allow modulated clocks to be used for the baud rate generation. The asynchronous clocks allow the clock sources for the baud clocks to be decoupled from the GCLK, HCLK and VCLKx clock domains.
Real-Time Interrupt clock domains	RTI1CLK	This clock is used for generating the periodic interrupts by the RTI module.

3.9.1 Mapping Clock Domains to Clock Sources

The system module on the TMS570LS microcontrollers contains registers that allow the clock domains to be mapped to any of the available clock sources. These registers are defined now.

Figure 15. GCLK, HCLK , VCLKx Source Register: GHVSRC, Address = 0xFFFFFF48



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				GHVWAKE(3–0)				Reserved				HVLPM(3–0)			
R-0				R/WP-0000				R-0				R/WP-0000			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												GHVSRC(3–0)			
R-0												R/WP-0000			

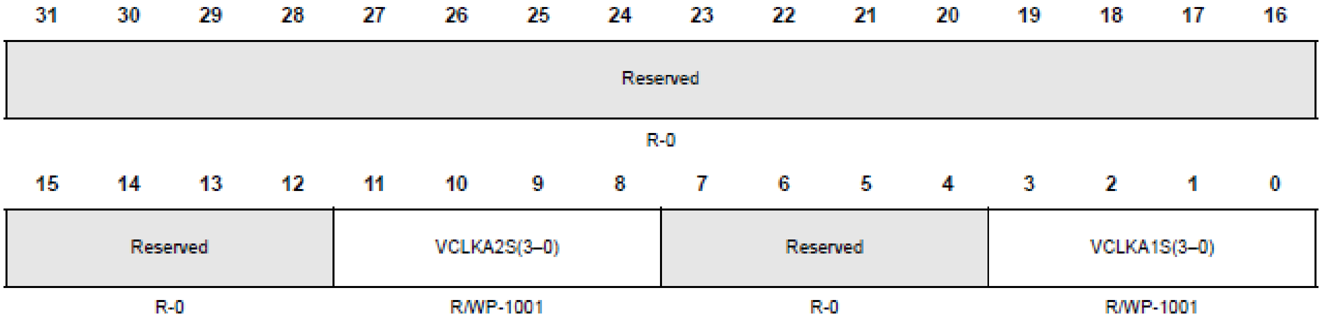
R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- GHVWAKE defines the clock source that will be used for the GCLK, HCLK and VCLKx domains when the microcontroller wakes up from a low power mode. Please refer to the platform architecture user guide for the TMS570LS microcontrollers for more details on the low power modes supported.

- HVLPM defines the clock source used for the HCLK and VCLKx domains when the CPU clock domain GCLK is disabled.

- GHVSRC defines the clock source to be currently used for the GCLK, HCLK and VCLKx domains. As shown by the reset value of the GHVSRC field, the clock source # 0, that is, the main oscillator, is used as the default clock source for the GCLK, HCLK and VCLKx domains.

Figure 16. Asynchronous Clock Source Register: VCLKASRC, Address = 0xFFFFFF4C



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				VCLKA2S(3–0)				Reserved				VCLKA1S(3–0)			
R-0				R/WP-1001				R-0				R/WP-1001			

R = Read in all modes; WP = Write in privileged mode only; -n values after reset

- VCLKA1 is used for generating the DCAN bit timings, and the VCLKA1S field defines the clock source used for the VCLKA1 domain.

- VCLKA2 is used for generating the FlexRay timings, and the VCLKA2S field defines the clock source used for the VCLKA2 domain.

Figure 17. RTI Clock Source Register: RCLKSRC, Address = 0xFFFFFF50

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						RTI1DIV(1–0)		Reserved				RTI1SRC(3–0)			
R-0						R/WP-01		R-0				R/WP-1001			

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- RTI1SRC field defines the clock source used for the RTI1CLK domain. This domain is mapped to VCLK by default.
- If the clock source for RTI1CLK is selected to be something other than VCLK, then the RTI1CLK frequency must be at least 1/3rd of the VCLK frequency. This can be achieved by using the RTI2DIV field, which defines the divider values used to divide down the clock source selected for RTI1CLK.

3.9.2 Example Clock Domain Mapping

```
systemREG1->GHVSRC = (0U << 24U)  // Use main oscillator as wake up source for GHV CLK
                   | (0U << 16U)  // Use main oscillator for HV CLK when GCLK is off
                   | (1U);        // Use FMzPLL as current source for GHV CLK

systemREG1->VCLKASRC = (0U << 8U) // Use main oscillator for FlexRay bit timing
                     | (0U);      // Use main oscillator for DCANx bit timings

systemREG1->RCLKSRC = (1U << 8U)  // Set the RTI1CLK divider to divide-by-2
                    | (0U);       // Use FMzPLL as source for RTI1CLK
```

3.9.3 Configuring VCLK and VCLK2 Frequencies

The VCLK and VCLK2 clock signals are divided down from the HCLK clock signal. These are independent dividers that can be configured via the system module Clock Control Register (CLKCNTL), as shown below.

Figure 18. Peripheral Clock Control Register: CLKCNTL, Address = 0xFFFFFFD0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				VCLK2R				Reserved				VCLKR			
R-0				R/WP-0001				R-0				R/WP-0001			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							PENA	Reserved							
R-0							R/WP-0	R-0							

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- VCLK2R defines the divide ratio between HCLK and VCLK2.
- VCLKR defines the divide ratio between HCLK and VCLK.
  - VCLK2 and VCLK can be from HCLK/1 to HCLK/16

NOTE: VCLK2 frequency must also be an integer multiple of VCLK frequency.

NOTE: There must be some delay between configuring the divide ratios for VCLK2 and VCLK.

```
systemREG1->CLKCNTL |= 0x00000000U;                  // VCLK2 = HCLK/1
temp = systemREG1->CLKCNTL;                          // dummy read to cause delay
systemREG1->CLKCNTL |= 0x00010000U;                  // VCLK = HCLK/2
```

3.10 Run CPU Self-Test

Please refer to chapter 7 of the Technical Reference Manual for the TMS570LS Microcontroller for information on the configuration and execution of the CPU self-test.

3.11 Release Reset and Clocks to Peripherals

The peripherals are kept under reset, and need to be explicitly brought out of reset by the application. This can be done by setting the Peripheral Enable (PENA) bit of the Clock Control Register, shown below.

```
systemREG1->CLKCNTL |= 0x00000100U;            // Release peripheral reset
```

The clocks to the peripheral modules are also disabled upon any system reset and need to be explicitly enabled by the application. This can be done by setting the bits corresponding to the peripheral select quadrant occupied by the peripheral module in the PCR module registers for clearing the power down states of peripheral modules (PSPWRDWNCLRx). Please refer to the datasheet for the TMS570LS Microcontrollers for information on the peripheral select quadrants for each peripheral.

In the example below, the clocks to all implemented peripherals are being enabled.

```
pcrREG->PSPWRDWNCLR0 = 0xFFFFFFFFU;
pcrREG->PSPWRDWNCLR0 = 0xFFFFFFFFU;
pcrREG->PSPWRDWNCLR0 = 0xFFFFFFFFU;
pcrREG->PSPWRDWNCLR0 = 0xFFFFFFFFU;
```

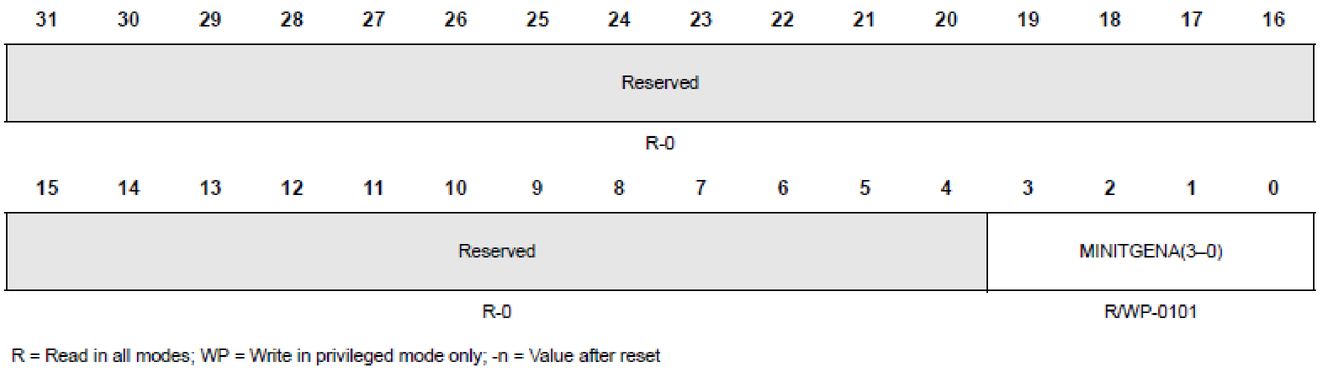
3.12 Memories' Self-Test

Please refer to the Technical Reference Manual for the TMS570LS microcontrollers for information on executing the self-test on the on-chip memories using the programmable BIST (PBIST) engine.

3.13 Memories' Auto-Initialization

The system module on the TMS570LS microcontroller allows all on-chip SRAMs to be initialized in hardware. This is especially essential since all the on-chip memories support some form of read protection. The CPU data RAM supports ECC while the peripheral memories support parity error detection. This mechanism also automatically initializes the ECC or parity memories, as required. The following registers are used in this process.

Figure 19. Memory Hardware Initialization Global Control Register: MINITGCR, Address = 0xFFFFFF5C



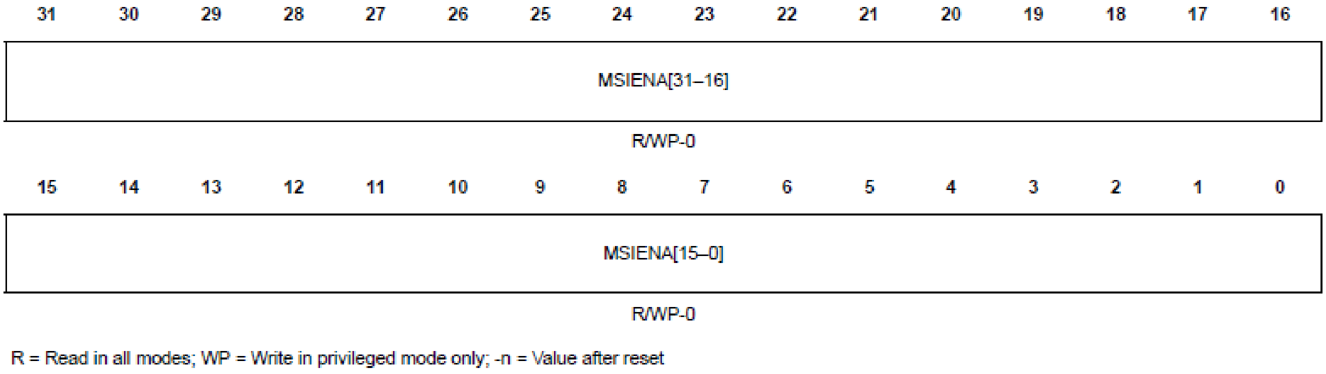
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												MINITGENA(3–0)			
R-0												R/WP-0101			

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- MINITGENA must be configured to 1010b to enable the hardware memory initialization mechanism.

Figure 20. Memory Self-Test / Initialization Control Register: MSIENA, Address = 0xFFFFFF60



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSIENA[31–16]															
R/WP-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSIENA[15–0]															
R/WP-0															

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- Each bit of MSIENA refers to a single SRAM module on the microcontroller. Refer to the device datasheet for the on-chip SRAM mapping to the initialization channel number.

Figure 21. Memory Self-Test / Initialization Status Register: MSTCGSTAT, Address = 0xFFFFFF68

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							MINI DONE	Reserved							MST DONE
R-0							R/WP-0	R-0							R/WP-0

R = Read in all modes; WP = Write in privileged mode only; -n = Value after reset

- MINI DONE gets set when all memories selected via the MSIENA field have been initialized to zeros. The application can poll this bit.

```
systemREG1->MINITGCR = 0xA;                         // Enable memory init
systemREG1->MSIENA   = 0xFFFFFFFF;                  // Select all SRAMs for init
while (!(systemREG1->MSTCGSTAT & 0x00000100U)); // Wait until memory init is done
```

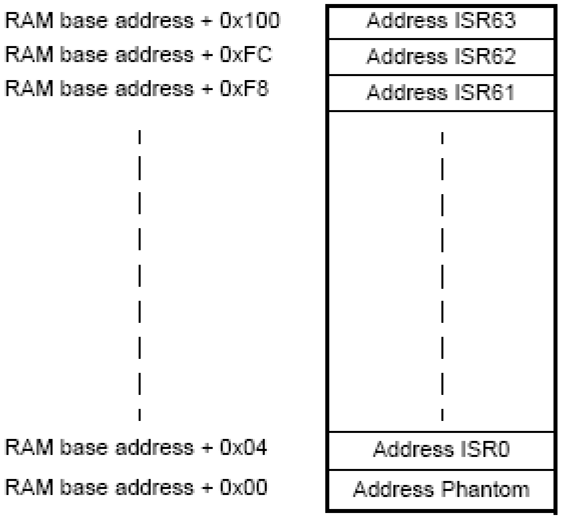
3.14 Vectored Interrupt Manager Configuration

The Vectored Interrupt Manager (VIM) module on the TMS570LS microcontrollers supports flexible mapping of interrupt request channels and the interrupt generating sources. The default mapping between the channel number and the interrupting module is defined in the device datasheet. The interrupt channel number also defines the inherent priority between the channels, with the lower numbered channel having the higher priority. That is, the priority decreases in the following order: channel 0 → channel 1 → channel 2 → … channel 63.

For this app note, assume that the application prefers to keep the default priority order between the channels. Refer to the Technical Reference Manual for details on the control registers for changing the mapping between interrupt channels and sources.

The VIM module contains a memory that holds the starting addresses of the interrupt service routines for each interrupt enabled in the application. This memory starts at base address 0xFFF82000 on the TMS570LS microcontrollers. It is organized in 65 words of 32 bits.

Figure 22. VIM Interrupt Address Memory Map



3.14.1 Example VIM RAM Configuration

The below code fragment shows the configuration of the interrupt service routine addresses' in the VIM memory.

Definitions:

```
typedef void (*t_isrFuncPTR)();
#define VIM_CHANNELS 64U
typedef volatile struct vimRam
{
    t_isrFuncPTR ISR[VIM_CHANNELS];
} vimRAM_t;
#define vimRAM ((vimRAM_t *)0xFFF82000U)
```

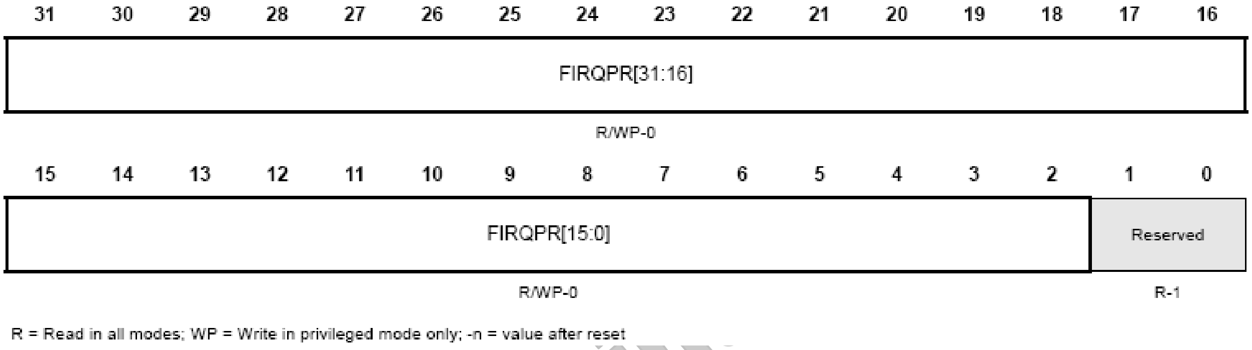
Configuration:

```
static const t_isrFuncPTR s_vim_init[] =
{
        phantomInterrupt,
        phantomInterrupt,
        phantomInterrupt,
        rtiCompare0Interrupt,
        rtiCompare1Interrupt,
        rtiCompare2Interrupt,
        rtiCompare3Interrupt,
        rtiOverflow5Interrupt,
        rtiOverflow1Interrupt,
        rtiTimebaseInterrupt,
        gioHighLevelInterrupt,
        hetHighLevelInterrupt,
        htuHighLevelInterrupt,
        spi1HighLevelInterrupt,
        sci1HighLevelInterrupt,
        adc1Group0Interrupt,
        adc1Group1Interrupt,
        can1HighLevelInterrupt,
        phantomInterrupt,
        erayHighLevelInterrupt,
        crcInterrupt,
        esmLowLevelInterrupt,
        swInterrupt,
        pmuInterrupt,
        gioLowLevelInterrupt,
        hetLowLevelInterrupt,
        htuLowLevelInterrupt,
        spi1LowLevelInterrupt,
        sci1LowLevelInterrupt,
        adc1Group2Interrupt,
        can1LowLevelInterrupt,
        phantomInterrupt,
        adc1MagInterrupt,
        erayLowLevelInterrupt,
        dmaFTCAInterrupt,
        dmaLFSAInterrupt,
        can2HighLevelInterrupt,
        dmmHighLevelInterrupt,
        spi3HighInterruptLevel,
        spi3LowLevelInterrupt,
        dmaHBCAInterrupt,
        dmaBTCAInterrupt,
        phantomInterrupt,
        can2LowLevelInterrupt,
        dmmLowLevelInterrupt,
        can1IF3Interrupt,
        can3HighLevelInterrupt,
        can2IF3Interrupt,
        fpuInterrupt,
        ftuXferStatusInterrupt,
        sci2HighLevelInterrupt,
        adc2Group0Interrupt,
        adc2Group1Interrupt,
        erayT0CInterrupt,
        spi5HighLevelInterrupt,
        sci2LowLevelInterrupt,
        can3LowLevelInterrupt,
        spi5LowLevelInterrupt,
        adc2Group2Interrupt,
        ftuErrorInterrupt,
        adc2MagInterrupt,
        can3IF3Interrupt,
        phantomInterrupt,
        erayT1CInterrupt,
        phantomInterrupt
    };
```

3.14.2 Configure Interrupts to be Fast Interrupts or Normal Interrupts

Two registers in the VIM module allow each of the 64 interrupts to be assigned to either the fast interrupt (FIQ) queue, or the normal interrupt queue (IRQ). These registers are shown now.

Figure 23. FIQ/IRQ Control Register 0: FIRQPR0, Address = 0xFFFFFE10

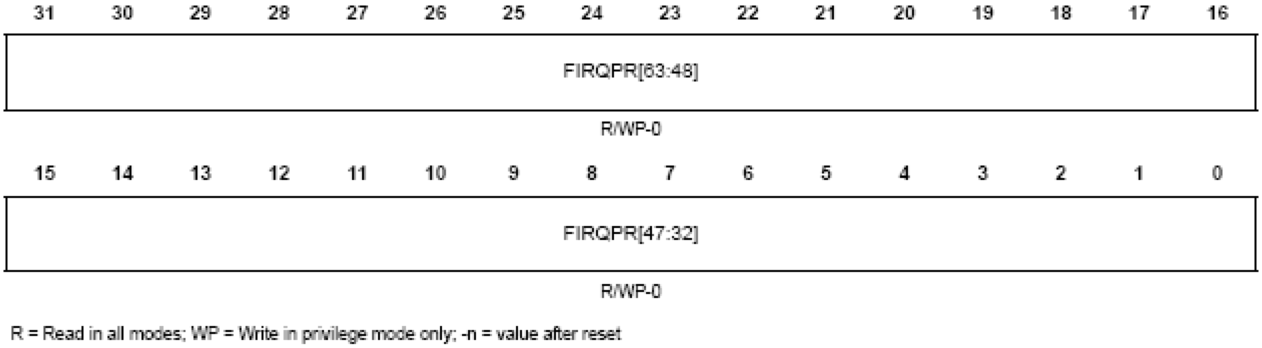


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIRQPR[31:16]															
R/WP-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIRQPR[15:0]														Reserved	
R/WP-0														R-1	

R = Read in all modes; WP = Write in privileged mode only; -n = value after reset

Figure 24. FIQ/IRQ Control Register 1: FIRQPR1, Address = 0xFFFFFE14



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIRQPR[63:48]															
R/WP-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIRQPR[47:32]															
R/WP-0															

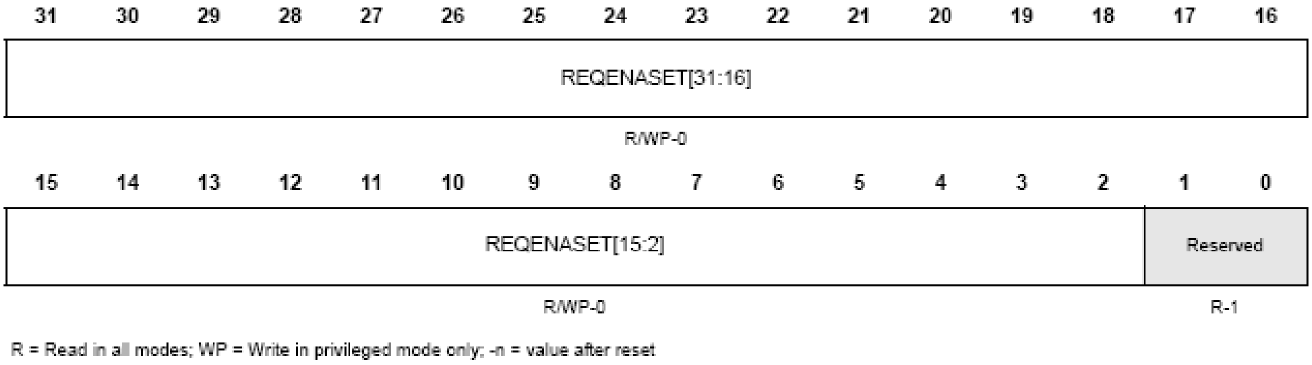
R = Read in all modes; WP = Write in privilege mode only; -n = value after reset

Setting any bit in the above two registers makes the corresponding interrupt request become an FIQ interrupt. As shown, the interrupt requests 0 and 1 are always FIQ. All others are IRQ interrupts by default.

3.14.3 Enabling Interrupts

Control registers in the VIM module allow each interrupt request to be enabled or disabled.

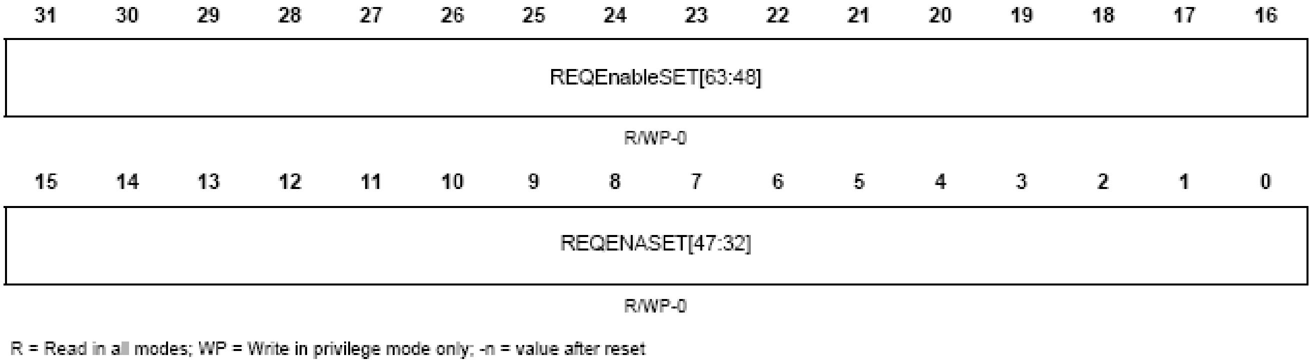
Figure 25. Interrupt Enable Set Register 0: REQENASET0, Address = 0xFFFFFE30



<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="16">REQENASET[31:16]</td></tr>
<tr><td colspan="16">R/WP-0</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="14">REQENASET[15:2]</td><td colspan="2">Reserved</td></tr>
<tr><td colspan="14">R/WP-0</td><td colspan="2">R-1</td></tr>
</table>

R = Read in all modes; WP = Write in privileged mode only; -n = value after reset

Figure 26. Interrupt Enable Set Register 1: REQENASET1, Address = 0xFFFFFE34



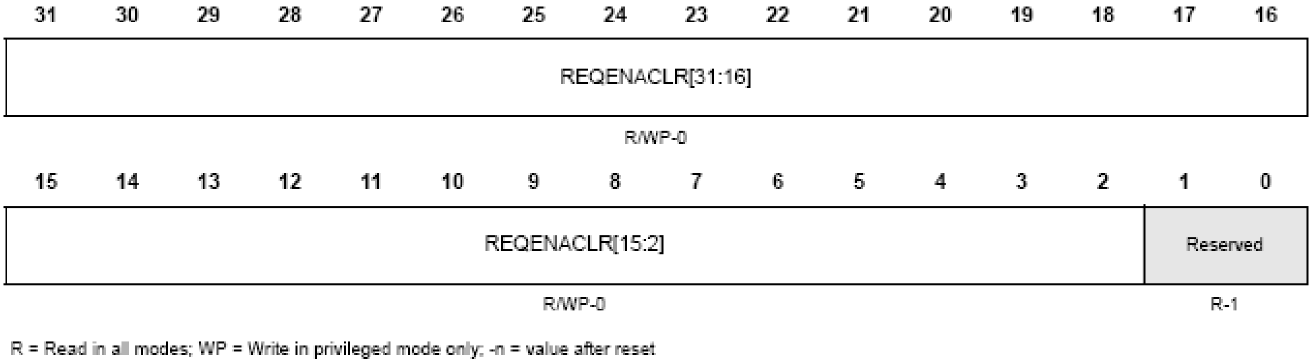
<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="16">REQEnableSET[63:48]</td></tr>
<tr><td colspan="16">R/WP-0</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">REQENASET[47:32]</td></tr>
<tr><td colspan="16">R/WP-0</td></tr>
</table>

R = Read in all modes; WP = Write in privilege mode only; -n = value after reset

Setting any bit in the above two registers enables the corresponding interrupt request to trigger either an IRQ or an FIR exception to the Cortex-R4F CPU.

The interrupt requests 0 and 1 are always enabled and cannot be disabled.

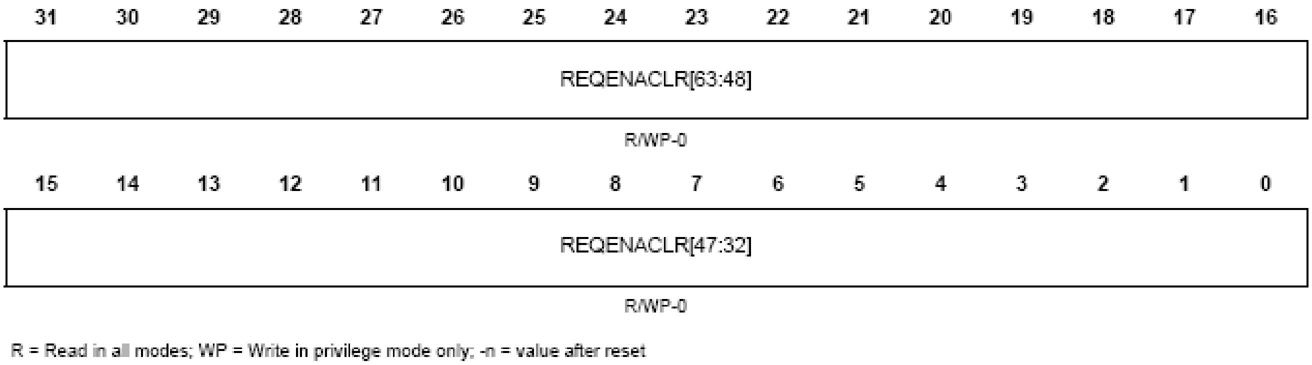
Figure 27. Interrupt Enable Clear Register 0: REQENACLR0, Address = 0xFFFFFE40



<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="16">REQENACLR[31:16]</td></tr>
<tr><td colspan="16">R/WP-0</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="14">REQENACLR[15:2]</td><td colspan="2">Reserved</td></tr>
<tr><td colspan="14">R/WP-0</td><td colspan="2">R-1</td></tr>
</table>

R = Read in all modes; WP = Write in privileged mode only; -n = value after reset

Figure 28. Interrupt Enable Clear Register 1: REQENACLR1, Address = 0xFFFFFE44



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REQENACLR[63:48]															
R/WP-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REQENACLR[47:32]															
R/WP-0															

R = Read in all modes; WP = Write in privilege mode only; -n = value after reset

Setting any bit in the interrupt enable clear registers disables the corresponding interrupt. When an interrupt is disabled, it does not prevent the interrupt flag to get set when the interrupt condition is generated but no IRQ or FIR exception is generated for the Cortex-R4F CPU.

3.15 Additional Initializations Required by Compiler

If the source program is written using C or C++, the TI compiler requires the creation of the C/C++ run-time environment. This includes:

- Initialization of copy table, if required
- Initialization of global and static variables defines in C/C++
- Initialization of global constructors
- Make a function call to branch to the main application

These requirements could be different for each compiler. The compiler reference manual must be referred to identify the specific requirements for the compiler being used.

3.16 Call the Main Application

This is a normal function call when using C/C++. It could be a branch or branch-link to the name of the routine that executes the application.

For example:

```
    main();
    exit();
```