Application Report
SPNA106D–May 2013

Initialization of Hercules™ ARM® Cortex™-R4F Microcontrollers

Sunil Oak

ABSTRACT

This application report provides a brief overview and initialization procedure of the TMS570LS31x series and the RM4x series of microcontrollers in the Hercules family. "Hercules MCU" will be used henceforth in this document to refer to any part in these 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 defined in header files also generated by the same utility.

The HALCode Generator tool can be downloaded from the following URL: http://www.ti.com/tool/halcogen.

Contents

List of Figures

List of Tables

1 Block Diagram

Section 1 shows a high-level block diagram of the superset TMS570LS31x microcontroller. For the actual block diagram relevant for any derivative of the TMS570LS series or for the RM4x series of microcontrollers, see the device-specific data sheet.

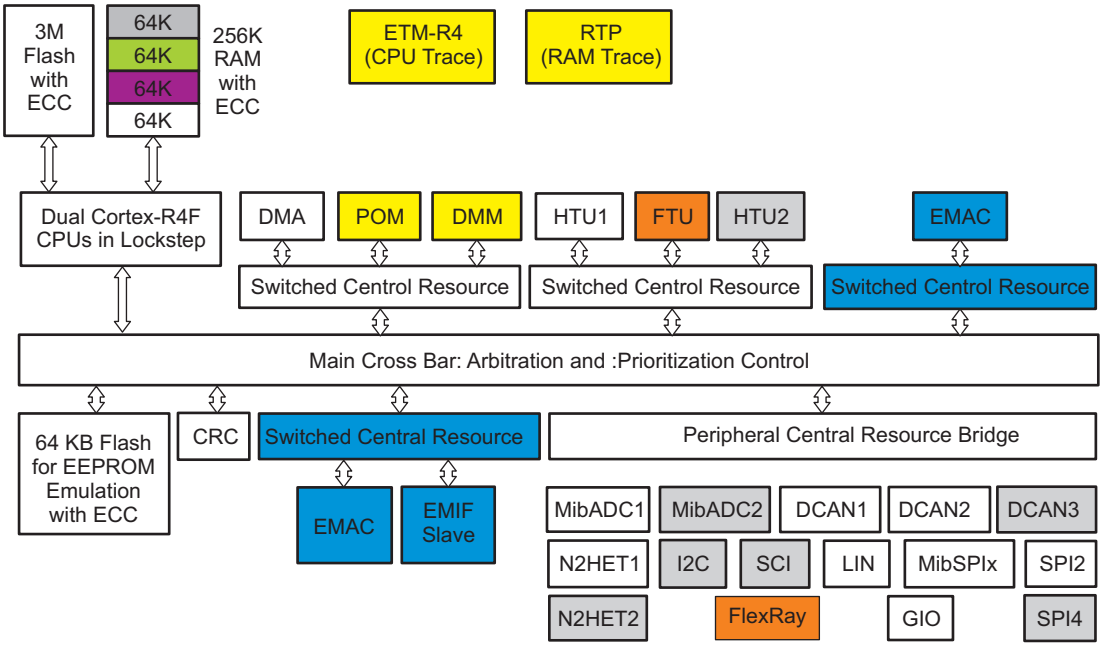

Figure 1. Device Block Diagram

The block diagram includes a color-coded representation of the individual core-power domains implemented on the microcontroller (see Figure 2). These power domains can be individually turned ON or OFF during initialization as per the application requirements.

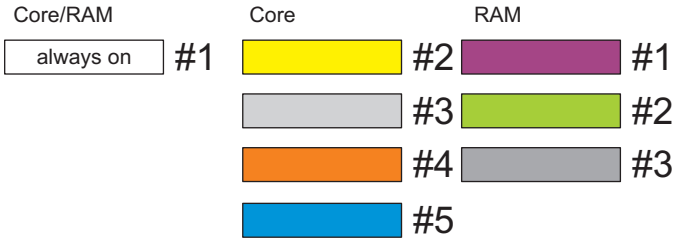

Figure 2. Color Legend for Block Diagram

2 Standard Initialization Sequence for Hercules Microcontrollers

A basic sequence for initialization and configuration of the key features on a Hercules MCU is summarized below and many steps are detailed in the following sections. The source code example accompanying this application report demonstrates many of the suggested steps. Some parts of the initialization sequence are not mandatory. Applications that are non-safety-critical can choose to not use the error correction coding (ECC) feature for Flash and RAM accesses, for example. Each application must also have its specific exception handling scheme: reset handler, abort handler, etc. The code generated using HALCoGen includes template handling routines for each exception. These routines need to be modified as required by the application.

1. Enable the floating-point unit (FPU) inside the Cortex-R4F CPU (Section 2.1).
2. Initialize the CPU registers and FPU registers, including stack pointers (Section 2.2).
3. Enable the flash interface module's response to an ECC error indicated by the CPU on accesses to flash (Section 2.3).
4. Enable the CPU's Event Bus export mechanism (Section 2.4).
5. Enable the CPU's Single-Error-Correction Double-Error-Detection (SECDED) logic for accesses to Flash memory (CPU's ATCM interface) (Section 2.5).
6. Handle the cause of reset to determine whether or not to continue with the start-up sequence (Section 2.6)
7. Check if any ESM group3 error was indicated during power-up. If any ESM group3 error occurred during the power-up, it is not safe to continue code execution and the microcontroller initialization process can be stopped at this point. The subsequent steps in this sequence assume that there was no ESM group3 error during power-up.
8. Configure PLL control registers with the largest value for the last-stage of the dividers (R-dividers) (Section 2.7).
9. Enable the Phased-Locked Loops (PLLs) (Section 2.8).
10. Run the eFuse controller start-up checks and start the self-test on the eFuse controller SECDED logic (Section 2.9).
11. Release the peripherals from reset and enable clocks to all peripherals (Section 2.10).
12. Set up the device-level multiplexing options as well as the input/output (I/O) multiplexing.
13. Wait for the eFuse controller ECC logic self-test to complete and check the results.
14. Set up Flash module for the required wait states and pipelined mode (Section 2.11).
15. Set up Flash bank and pump power modes (Section 2.12).
16. Trim the LPO (Section 2.13).
17. Run the self-test on the SECDED logic embedded inside the Flash module (Section 2.14).
18. Wait for main PLL output to become valid.
19. Map the device clock domains to the desired clock sources (Section 2.15).
20. Reduce the values of the R-dividers in steps to attain the target PLL output frequency for both PLL1 and PLL2.
21. Run a diagnostic check on the CPU self-test controller (Section 2.16). A CPU reset is asserted upon completion of the CPU self-test. Therefore, the initialization steps leading up to the reset handler will be repeated.
22. Run the built-in self-test for the CPU (LBIST) (Section 2.17). A CPU reset is asserted upon completion of the CPU self-test. Therefore, the initialization steps leading up to the reset handler will be repeated.
23. Run a diagnostic check on the CPU compare module (CCM-R4F) (Section 2.18).
24. Run a diagnostic check on the memory self-test controller (Section 2.19).
25. Start a self-test on the CPU RAM using the programmable built-in self-test (PBIST) controller and wait for this self-test to complete and pass (Section 2.20).
26. Initialize the CPU RAM using the system module hardware initialization mechanism so that the ECC region for the CPU RAM is also initialized (Section 2.21).
27. Enable the CPU's Single-Error-Correction Double-Error-Detection (SECDED) logic for accesses to CPU RAM memory (CPU's B0TCM and B1TCM interfaces) (Section 2.22).

28. Start a self-test on all on-chip dual-port SRAMs using the PBIST controller (Section 2.23).
29. Run the self-test on the CPU's SECDED logic for accesses to main data RAM (B0TCM and B1TCM) (Section 2.24).
30. Run the self-test on the CPU's SECDED logic for accesses to the main Flash memory (ATCM) (Section 2.25).
31. Wait for self-test to complete and pass on all on-chip dual-port SRAMs.
32. Start a self-test on all on-chip single-port SRAMs excluding the CPU RAM using the PBIST controller (Section 2.26).
33. Wait for self-test to complete and pass on all on-chip single-port SRAMs.
34. Start auto-initialization for all other on-chip SRAMs (Section 2.27).
35. Check if the auto-initialization process for all RAMs is completed; wait here if it has not completed.
36. Check the parity error detection mechanism for all peripheral memories (Section 2.28).
37. Enable the CPU's dedicated vectored interrupt controller (VIC) port (Section 2.29).
38. Program all interrupt service routine addresses in the vectored interrupt manager (VIM) memory (Section 2.30).
39. Configure IRQ / FIQ interrupt priorities for all interrupt channels (Section 2.30.1).
40. Enable the desired interrupts (IRQ and/or FIQ) inside the CPU (Section 2.31).
41. Enable the desired interrupts in the VIM control registers (Section 2.30.2).
42. Set up the application responses to inputs to the error signaling module (ESM) (Section 2.32).
43. Initialize copy table, global variables, and constructors (Section 2.33).
44. Verify that the dual-clock-comparator (DCC) module can actually detect and flag a frequency error.
45. Configure the DCC module to continuously monitor the PLL output.
46. Verify that a memory protection unit (MPU) violation for all bus masters is flagged as an error to the ESM.
47. Run a background check on entire Flash using CRC and DMA.
48. Run the offset error calibration routine for the ADC.
49. Run a self-test on the analog-to-digital converter (ADC) analog input channels.
50. Check I/O loop-back for all peripherals.
51. Set up the MPU for the bus masters.
52. Set up the digital windowed watchdog (DWWD) module service window size and the module response on a violation (reset or NMI).
53. Configure the N2HET1-to-N2HET2 monitoring functionality.
54. Configure desired access permissions for peripherals using the Peripheral Central Resource (PCR) controller registers.
55. Configure external safety companion, e.g., TI TPS6538x, for online diagnostic operation.
56. Set up the real-time interrupt (RTI) module for generating periodic interrupts as required by the application.
57. Call the main application (Section 2.35).

2.1 Enable Floating-Point Coprocessor (FPU)

The floating-point coprocessor is disabled upon a CPU reset and must be enabled if the application requires floating-point calculations. If a floating-point instruction is executed with the FPU disabled, an undefined instruction exception is generated.

2.2 Initialize Cortex-R4F Registers

The Hercules 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 registers are not guaranteed to power up in the same state for both the CPUs. The CPU pushes the internal registers on to the stack on a function call, which could lead to the detection of a core compare error. Therefore, the CPU internal core registers need to be initialized to a predefined state before any function call is made.

The CPU's call-return stack consists of a 4-entry circular buffer. When the CPU pre-fetch unit (PFU) detects a taken procedure call instruction, the PFU pushes the return address onto the call-return stack. The instructions that the PFU recognizes as procedure calls are, in both the ARM and Thumb instruction sets:

→ BL immediate
→ BLX immediate
→ BLX Rm

When the return stack detects a taken return instruction, the PFU issues an instruction fetch from the location at the top of the return stack, and pops the return stack. The instructions that the PFU recognizes as procedure returns are, in both the ARM and Thumb instruction sets:

→ LDMIA Rn{!}, {..,pc}
→ POP {..,pc}
→ LDMIB Rn{!}, {..,pc}
→ LDMDA Rn{!}, {..,pc}
→ LDMDB Rn{!}, {..,pc}
→ LDR pc, [sp], #4
→ BX Rm

2.3 Enable Response to ECC Errors in Flash Interface Module

The Flash module has a Flash Error Detection and Correction Control Register 1 (FEDACCTRL1) at address 0xFFF87008. This register controls the ECC functionality implemented inside the Flash module, including support for the SECDED logic inside the Cortex-R4F CPU. The bits 3–0 of this register make up the EDACEN field. EDACEN is configured to 0x5 by default. The application must configure EDACEN to 0xA in order to enable the flash module's support for the CPU's SECDED logic.

2.4 Enable the Cortex-R4F CPU's Event Signaling Mechanism

The Cortex-R4F CPU has a dedicated event bus that is used to indicate that an event had occurred. This event signaling is disabled upon reset and must be enabled. The Flash module and the RAM module interfaces capture the ECC error events signaled by the CPU. This allows the application to further debug the exact address, which caused the ECC error.

The CPU event signaling can be enabled by setting the "X" bit of the performance monitoring unit's "Performance monitor control register, c9".

2.5 Enable the Cortex-R4F CPU's ECC Checking for ATCM Interface

The CPU has internal ECC logic that protects all CPU accesses to the ATCM (Flash) interface. This logic is not used by default and must be enabled by setting the ATCMPCEN bit of the System control coprocessor's Auxiliary control register, c1.

2.6 Handle the Cause of Reset

Each application has different levels of tolerance for different reset conditions.

2.7 Configure PLLs

The Hercules microcontrollers contain a frequency-modulated phase-locked loop (FMPLL) 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 FMPLL allows the flexibility to generate many different frequency options from a fixed crystal or resonator.

The FMPLL allows the application to superimpose a "modulation frequency" signal on the selected base frequency signal output from the FMPLL. 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 Hercules microcontrollers also contain a second non-modulating PLL macro. This PLL#2 can be independently configured to generate a second high-frequency clock source for specific uses, e.g., FlexRay communication clock source of 80 MHz.

2.7.1 FMPLL Block Diagram

Figure 3 shows a high-level block diagram of the FMPLL macro.

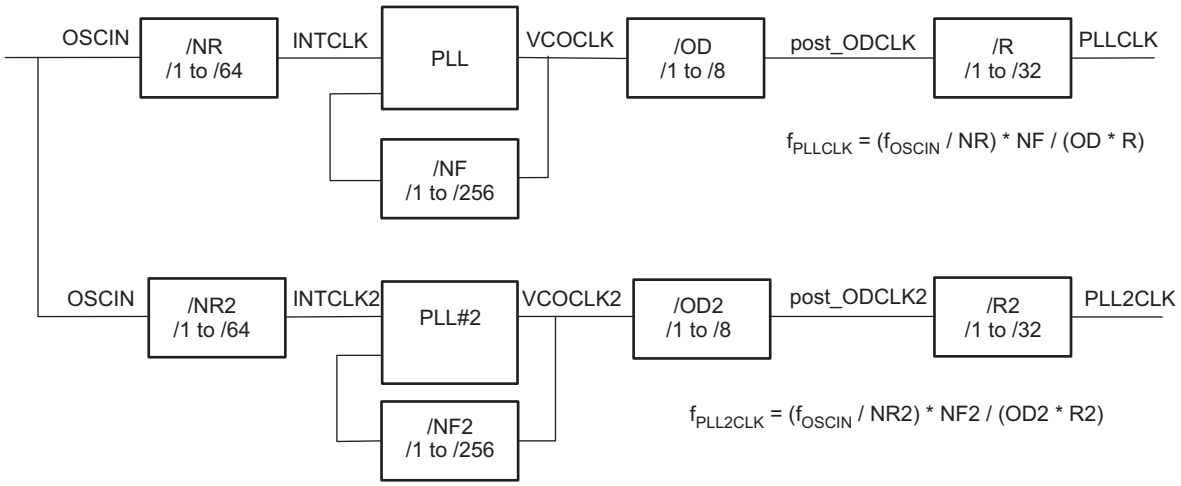

Figure 3. FMPLL Block Diagram

The parameters f_{OSCIN}, f_{post_ODCLK} and f_{HCLK} are data sheet specifications. To identify the min/max limits on these frequencies, see the device-specific data sheet.

NOTE: The FMPLL takes (127 + 1024*NR) oscillator cycles to acquire lock to the target frequency, hence it is recommended to configure the FMPLL(s) and enable them as soon as possible in the device initialization.

2.7.2 FMPLL Configuration

PLL1 is configured using two control registers, PLL Control 1 Register (PLLCTL1) and PLL Control 2 Register (PLLCTL2), located within the System module on the Hercules microcontrollers.

PLL2 is configured using a single PLL Control 3 Register (PLLCTL3) in the System module.

2.8 Enable Clock Sources

2.8.1 Available Clock Sources on Hercules Microcontrollers

The Hercules microcontrollers support seven different clock sources, as listed in Table 1.

Table 1. Clock Sources on Hercules 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 FMPLL and the FMPLL2 macros. The OSCIN frequency must be between 5 MHz and 20 MHz.
1	FMPLL#1 output	This is the output of the FMPLL, which is generated using the OSCIN as the input clock. The FMPLL output clock frequency must not exceed the maximum device frequency specified in the device-specific data sheet. The FMPLL features a modulation mode where a modulation frequency is superimposed on the FMPLL 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	EXTCLKIN1	External clock input #1. This clock source must only be enabled if there is an actual external clock source connected to the identified device terminal for EXTCLKIN1. For more information, see the device-specific data sheet.
4	LF LPO	This is the low-frequency output of the internal reference oscillator. The LF LPO is typically an 80 KHz 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 10 MHz signal, and is used as a reference clock for monitoring the main oscillator.
6	FMPLL#2 output	This is the output of the secondary FMPLL, which is generated using the OSCIN as the input clock. The FMPLL output clock frequency must not exceed the maximum device frequency specified in the device-specific data sheet.
7	EXTCLKIN2	External clock input #2. This clock source must only be enabled if there is an actual external clock source connected to the identified device terminal for EXTCLKIN2. For more information, see the device-specific data sheet.

2.8.2 Control Registers for Enabling and Disabling Clock Sources

There are seven available clock sources on the Hercules microcontrollers:

- Clock sources 0, 4 and 5 are enabled, while clock sources 1, 3, 6 and 7 are disabled upon any system reset.
- Clock source 2 is not implemented and must not be enabled in the application.
- Each bit of the system module Clock Source Disable Register (CSDIS) controls the clock source of the same number: bit 0 controls clock source 0, bit 1 controls clock source 1, and so on.
- There are also dedicated Clock Source Disable Set (CSDISSET) and Clock Source Disable Clear (CSDISCLR) registers to allow the application to avoid using read-modify-write operations.
- 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 (FMPLL, FMPLL#2) using the clock source to be disabled.

2.8.3 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.

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 FMPLL as well as the FMPLL#2 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. The example initialization sequence makes use of this PLL lock time to perform all initialization actions that don'e have to be done at the maximum operating frequency chosen for the application.

2.9 Run Self-Test on the eFuse Controller SECDED Logic

Electrically programmable fuses (eFuses) are used to configure the part after de-assertion of power-on reset (nPORRST). The eFuse values are read and loaded into internal registers as part of the power-on-reset sequence. This is called the eFuse autoload. The eFuse values are protected with single-bit error-correction, double-bit error-detection (SECDED) codes. These fuses are programmed during the initial factory test of the device. The eFuse controller is designed so that the state of the eFuses cannot be changed once the device is packaged.

For safety critical systems, it is important for the application to check the status of the eFuse controller after a device reset. For more details on eFuse controller errors and the application sequence to check for these errors, see the *eFuse Controller* chapter of the device-specific technical reference manual.

2.10 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 (CLKCNTL).

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 Peripheral Central Resource (PCR) Control Registers for clearing the power down states of peripheral modules (Peripheral Power-Down Clear Register [0:3] (PSPWRDWNCLRx)). For information on the peripheral select quadrants for each peripheral, see the device-specific data sheet.

2.11 Configure Flash Access

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

The Hercules microcontrollers contain a digital 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 45 MHz. If the bus clock is faster than 45 MHz, then any Flash access requires the appropriate number of wait states depending on the bus clock speed. The Hercules series microcontrollers support clock speeds up to 180 MHz. For the actual maximum allowed speed and the number of corresponding address and data wait states, see the device-specific data sheet.

Suppose that the application requires a CPU clock speed of 180 MHz. 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 module registers.

The Flash module also features a pipelined mode of operation. When this mode is enabled, the module 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 Flash Read Control Register (FRDCNTL) inside the Flash module controls the wait states and the pipeline mode.

The Hercules MCUs also have a separate Flash bank (bank #7) that is dedicated for data storage. This bank can be used to emulate an EEPROM. Accesses to this Flash bank are configured via a separate EEPROM Emulation Configuration Register (EEPROM_CONFIG) in the Flash module. A write operation to the EEPROM_CONFIG register must first be enabled by configuring the Flash State Machine Write Enable Control Register (FSM_WR_ENA).

Once the access to the FSM control registers is enabled, the read access to the Flash bank 7 can be configured.

2.12 Configure Flash Bank and Pump Power Modes

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

- Active mode
  - Flash bank sense amplifiers and sense reference are enabled
  - All circuits of Flash charge pump are enabled
- Standby mode (only for Flash banks)
  - Flash bank sense reference is enabled but sense amplifiers are disabled
- 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 module 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 module 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 that is in a non-active power state “wakes 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 Bank Access Control Register (FBAC) controls the Flash banks’ power states.

The Flash Pump Access Control Registers (FPAC1, FPAC2) control the Flash pump's power states.

2.13 Configure Oscillator Monitor

The HF LPO clock source is used as a reference clock for monitoring the main oscillator. A failure is detected if the oscillator frequency falls outside the range: $\{f_{HFLPO} / 4, f_{HFLPO}*4\}$.

The HF LPO frequency varies significantly over process corners as well as with changes in the core supply (VCC) and temperature. The Hercules microcontrollers allow the application to trim the HF LPO such that the application can choose the operating frequency point of the HF LPO. This in turn determines the valid range of oscillator frequency.

During device test, a trim value is written into the one-time programmable section of the Flash memory (OTP), address 0xF008_01B4. Bits 31:16 of this OTP word contain a 16-bit value that may be programmed into Low Power Oscillator Monitor Control Register (LPOMONCTL) in order to initialize the trim for HF LPO.

Alternatively, the application can use the dual-clock compare (DCC) module to determine the trim setting for the HF LPO. The DCC module allows for comparison of two clock frequencies. Once the HF LPO is determined to be in-range with the initial HFTRIM setting from the OTP, the crystal oscillator may be used as a reference against which the HF LPO and LF LPO may be further adjusted. For more details, see the device-specific technical reference manual.

2.14 Run Self-Test on the Flash Module SECDED Logic

The Flash module reads the “reset configuration vector” from address 0xF0080140 in the TI OTP region of Flash bank 0. This is a 64-bit value that is used to configure the device power domains, etc. The Flash module has built-in SECDED logic to correct any single-bit error in this vector or detect and flag and double-bit error in this vector. If a double-bit error is detected during this read from the OTP, an ESM group3 error condition is flagged and the nERROR signal is asserted low. If a single-bit error is detected during the read from the OTP, this error is corrected by the SECDED logic – no flag is set and no error signal is sent to the ESM.

There are dedicated locations within the TI OTP sector of Flash bank 0 that are programmed to have single-bit and double-bit errors. Specifically, a 32-bit or 64-bit read from the address 0xF00803F0 results in a single-bit error indication, and a 32-bit or 64-bit read from the address 0xF00803F8 results in a double-bit error indication. These locations can be read by the application to ensure that the Flash interface module is capable of detecting single-bit and double-bit errors upon reads from the OTP.

2.15 Clock Domains

All further initializatio steps are now required to be performed at the max operating frequency for the application. The application must now wait for the PLLs to lock to their target frequencies, and then map the device clock domains to the desired clock sources. There are multiple clock domains on the Hercules microcontrollers to ease the configuration and controllability of the different modules using these clock domains (see Table 2).

Table 2. Clock Domains on Hercules 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, VCLK2, VCLK3	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}. VCLK3 is also divided down from HCLK by a programmable divider from 1 to 16, and is used for the Ethernet and EMIF modules on the TMS570LS3x microcontrollers.
Asynchronous clock domains	VCLKA1, VCLKA2, and VCLKA4	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, Ethernet) 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.

2.15.1 Mapping Clock Domains to Clock Sources

The system module on the Hercules microcontrollers contains registers that allow the clock domains to be mapped to any of the available clock sources.

The clock source for the GCLK, HCLK , and VCLKx domains is selected by the GCLK, HCLK, VCLK, and VCLK2 Source Register (GHVSRC).

The clock sources for the VCLKA1 and VCLKA2 domains are selected via the Peripheral Asynchronous Clock Source Register (VCLKASRC).

The clock sources for the VCLKA3 and VCLKA4 domains are selected via the Peripheral Asynchronous Clock Configuration 1 Register (VCLKACON1).

The clock source for the RTI1CLK domain is selected via the RTI Clock Source Register (RCLKSRC).

2.15.2 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 FMPLL as current source for GHV CLK
systemREG1->VCLKASRC = (6U << 8U)    // Use second PLL output 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 FMPLL as source for RTI1CLK
```

2.15.3 Configuring VCLK , VCLK2 and VCLK3 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).

NOTE:

- VCLK2 frequency must also be an integer multiple of VCLK frequency.
- There must be some delay between configuring the divide ratios for VCLK2 and VCLK.

The VCLK3 clock signal is also divided down from the HCLK clock signal. This divider is in the Clock Control Register 2 (CLK2CNTL).

2.16 Run a Diagnostic Check on CPU Self-Test Controller (STC)

This involves running one CPU self-test interval in STC check mode. The STC self-check mode causes a stuck-at-0 fault to be introduced inside one of the two CPUs for there to be an STC failure. If no STC failure is indicated, this would mean that the STC is not capable of detecting a fault inside the CPU, and device operation is not reliable. For information on the configuration and execution of the STC self-test, see the device-specific technical reference manual. The CPU will be reset once the STC self-test is completed. The reset handler routine can resume the device initialization from the next step in the sequence.

2.17 Run CPU Self-Test (LBIST)

For information on the configuration and execution of the CPU self-test, see the device-specific technical reference manual. The CPU will be reset once the self-test is completed. The reset handler routine can resume the device initialization from the next step in the sequence.

2.18 Run a Diagnostic Check on the CPU Compare Module (CCM-R4F)

The CCM-R4F compares the dual Cortex-R4F CPU outputs on each CPU clock cycle. Any mismatch is indicated as an ESM group2 error. This ensures that the two CPUs are indeed operating in a lock-step mode. The CCM-R4F module also allows the application to test the different error conditions using built-in self-test routines. For information on how to configure the CCM-R4F in a self-test mode, see the device-specific technical reference manual.

2.19 Run a Diagnostic Check on the Programmable Built-In Self-Test (PBIST) Controller

The PBIST engine is used to run memory test routines on all on-chip memories. It is critical for the application to rely on this engine being able to detect and report a memory fault condition. Therefore it is necessary for the application to test this error detection and reporting mechanism before actually using it to test the on-chip memories. This is done by choosing to run a RAM test routine on a ROM memory. This test must generate a memory test failure. The application can look for the error flag to ensure that the PBIST controller can indeed detect and report a memory test failure. For information on how to configure the PBIST controller for executing specific memory test algorithms on selected on-cip memories, see the device-specific technical reference manual .

2.20 Start a Self-Test on the CPU RAM Using the PBIST Controller

The CPU RAM is tested first, so that the application can continue to execute while other memories are being tested later. For information on configuring the PBIST controller, see the device-specific technical reference manual.

2.21 Initialize the CPU RAM

The system module hardware for auto-initialization of on-chip memories also initializes the associated ECC or parity locations. This mechanism is now used to initialize the CPU RAM. This process clears the CPU RAM to all zeros and also programs the corresponding ECC locations.

2.22 Enable the Cortex-R4F CPU's ECC Checking for BxTCM Interface

The CPU has internal ECC logic that protects all CPU accesses to the BTCM (RAM) interfaces. This logic is not used by default and must be enabled by setting the B1TCMPCEN and B0TCMPCEN bits of the System control coprocessor's Auxiliary control register, c1.

2.23 Start a Self-Test on All Dual-Port Memories' Using the PBIST Controller

Separate algorithms are used for testing single-port versus dual-port on-chip SRAMs. For information on executing the self-test on the on-chip memories using the programmable BIST (PBIST) engine, see the device-specific technical reference manual.

2.24 Run a Self-Test on CPU's ECC Logic for Accesses to TCRAM

The CPU TCRAM was initialized earlier, so that all TCRAM is cleared to zeros and the corresponding correct ECC locations are programmed. The test of the CPU's ECC logic for accesses to TCRAM involves corrupting the ECC locations to create single-bit and two-bit ECC errors. For the sequence to test the CPU's ECC logic for accesses to TCRAM, see the device-specific technical reference manual or the initialization example project. Note that reading from a TCRAM location with a double-bit ECC error causes the CPU to take a data abort exception. The initialization example project also includes an example data abort handler.

2.25 Run a Self-Test on CPU's ECC Logic for Accesses to Program Flash

The Flash interface module supports a diagnostic mode (mode 7) that allows the application to test the CPU's ECC logic for accesses to program Flash. For the sequence to test the CPU's ECC logic for accesses to program Flash, see the device-specific technical reference manual or the initialization example project. Note that reading from a program Flash location with a double-bit ECC error causes the CPU to take a data abort exception. The initialization example project also includes an example data abort handler.

2.26 Start a Self-Test on All Single-Port Memories' Using the PBIST Controller

The CPU RAM can be excluded from this testing as it has already been verified before. For information on executing the self-test on the on-chip memories using the programmable BIST (PBIST) engine, see the device-specific technical reference manual.

2.27 On-Chip SRAM Auto-Initialization

The system module on the Hercules 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 error detection. The CPU data RAM supports ECC while the peripheral memories support parity error detection. The auto-initialization mechanism also initializes the ECC or parity memories, as required.

2.28 Run a Self-Test on All Peripheral RAMs' Parity Protection Mechanism

Accesses to most peripheral RAMs on this microcontroller are protected by parity error detection. Each of the peripherals with the parity error detection for its associated memory also includes a self-test mode to ensure that it is indeed capable of detecting and reporting a parity error on an access to the peripheral RAM. These self-test mechanisms can be used by the application before enabling use of the concerned peripheral.

2.29 Enable the Cortex-R4F CPU's Vectored Interrupt Controller (VIC) Port

The CPU has a dedicated port that enables the 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 (SYS).

2.30 Vectored Interrupt Manager (VIM) Configuration

The VIM module on the Hercules 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-specific data sheet. 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 95.

For this application report, assume that the application prefers to keep the default priority order between the channels. For details on the control registers for changing the mapping between interrupt channels and sources, see the device-specific technical reference manual.

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 Hercules microcontrollers. It is organized in 97 words of 32 bits. The VIM address memory map is shown in Figure 4.

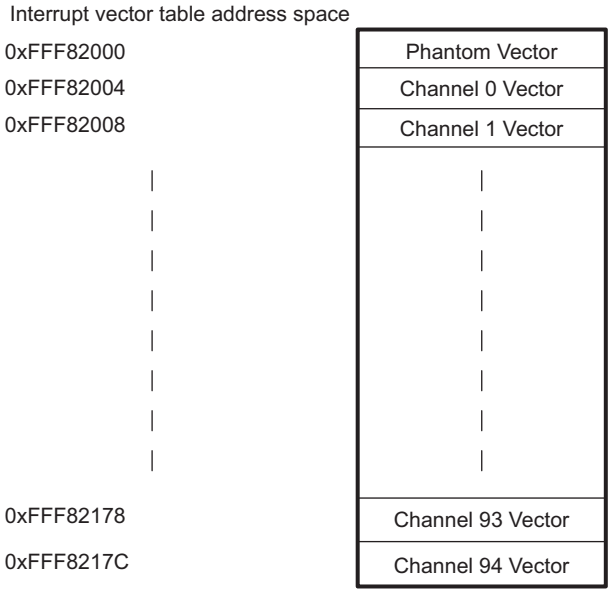

Figure 4. VIM Interrupt Address Memory Map

2.30.1 Configure Interrupts to be Fast Interrupts or Normal Interrupts

Each interrupt request to the VIM can be configured to be forwarded to the CPU as a fast interupt request (FIQ) or a normal interrupt request (IRQ). The FIQ/IRQ Program Control Registers (FIRQPRx) allow this selection.

Interrupt requests 0 and 1 are always FIQ. All others are IRQ interrupts by default.

NOTE: An interrupt request mapped to FIQ cannot use the CPU's VIC port.

2.30.2 Enabling and Disabling Interrupts

Each interrupt request can be enabled or disabled using the Interrupt Enable Set (REQENASETx) and Interrupt Enable Clear (REQENACLRx) registers. The interrupt requests 0 and 1 are always enabled and cannot be disabled. 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.

2.31 Enable Interrupts in the Cortex-R4F CPU

Interrupts (IRQ and FIQ) are disabled inside the Cortex-R4F CPU by default and after a CPU reset. The normal interrupt can be enabled by clearing the "I" bit of the Current Program Status Register (CPSR) inside the Cortex-R4F CPU, while the fast interrupt (FIQ) can be enabled by clearing the "F" bit of the CPSR.

2.32 Setup the Error Signaling Module (ESM) Responses to Group1 Errors

The ESM allows the application to choose the module response to errors in the Group1 classification. These are errors of the lowest severity and can be handled by the application by generating an interrupt to the CPU. The ESM also offers the capability to indicate any group1 errors on the external nERROR pin.

2.33 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.

2.34 Other Initialization Steps Not Described in this Document

The following is an additional list of operations that an application can perform during the device initialization.

- Verify that the DCC module can detect and report a frequency mismatch error.
- Configure the DCC module to continuously monitor the PLL output frequency.
- Several bus masters on this microcontroller include their own memory protection units to protect against accesses to certain parts of the memory map. It is recommended to ensure that violations of these MPU restrictions are detected and flagged as ESM errors.
- Configure the MPU for each bus masters.
- Run a background check on the program Flash memory using CRC and DMA.
- Calibrate the embedded ADC module for any offset error.
- Run a self-test on all ADC inputs to ensure that they are not open or shorted to power or ground.
- Run an I/O loop-back check on all peripheral signals.
- Configure the windowed watchdog module service window size as well as the module response to a window violation.
- Configure the N2HET1/N2HET2 monitoring capability.
- Setup the RTI module to generate periodic interrupts as necessary.
- Configure desired access permissions for peripherals using the PCR registers.
- Configure any external safety companion chip, e.g., TI TPS6538x, for online diagnostic operation.

2.35 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();
```

3 References

- *TMS570LSxxx7 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS162)
- *TMS570LSxxx5 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS164)
- *TMS570LSxxx4 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS165)
- *RM48Lx50 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS174)
- *RM48Lx40 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS175)
- *RM48Lx30 16/32-Bit Risc Flash Microcontroller Data Sheet* (SPNS176)
- *TMS570LS31/21 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU499)
- *RM48 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU503)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265