SmartFusion2

DDR Controller and Serial High Speed Controller Initialization Methodology

Table of Contents

Introduction

When creating a design using a SmartFusion2 device, if you use one of the two DDR controllers (FDDR or MDDR) or any of the Serial High speed controller (SERDESIF) blocks, you must initialize the configuration registers of these blocks at run-time before they can be used. For example, for the DDR controller, you must set the DDR mode (DDR3/DDR2/LPDDR), PHY width, burst mode and ECC. Similarly, for the SERDESIF block used as a PCIe endpoint, you must set the PCIE BAR to AXI (or AHB) window.

This document describes the steps necessary to create a Libero design that automatically initializes the DDR controller and SERDESIF blocks at power up. It also describes how to generate the firmware code from Libero SOC that is used in the embedded design flow.

A detailed description of the theory of operations is provided first.

The next section describes how to create such a design using the Libero SoC System Builder, a powerful design tool that, among other features, creates the 'initialization' solution for you if you are using DDR or SERDESIF blocks in your design.

The next section describes how to put a complete 'initialization' solution together without using the SmartFusion2 System Builder. This helps explain what needs to be done if you do not wish to use the System Builder, and also describes what the System Builder tool actually generates for you. This section addresses:

- The creation of the configuration data for DDR controller and SERDESIF configuration registers
- The creation of the FPGA logic required to transfer the configuration data to the different ASIC configuration registers

Finally we describe the generated files related to:

- The creation of firmware 'initialization' solution.
- The simulation of the design for the DDR 'initialization' solution.

For details about the DDR controller and SERDESIF configuration registers, refer to the Microsemi SmartFusion2 High Speed Serial and DDR Interfaces User's Guide.

1 – Theory of Operation

The Peripheral initialization solution uses the following major components:

- The CMSIS SystemInit() function, which runs on the Cortex-M3 and orchestrates the initialization process.
- The CoreConfigP soft IP core, which initializes the peripherals' configuration registers.
- The CoreResetP soft IP core, which manages the reset sequence of the MSS, DDR controllers, and SERDESIF blocks..

The peripheral initialization process works as follows:

1. Upon reset, the Cortex-M3 runs the CMSIS SystemInit() function. This function is automatically executed before the application's main() function is executed.

   The CoreResetP output signal MSS_HPMS_READY is asserted at the beginning of the initialization process, indicating that the MSS and all the peripherals (except MDDR) are ready for communication.
2. The SystemInit() function writes configuration data to the DDR controllers and SERDESIF configuration registers via the MSS FIC_2 APB3 bus. This interface is connected to the soft CoreConfigP core instantiated in the FPGA fabric.
3. After all the registers are configured, the SystemInit() function writes to the CoreConfigP control registers to indicate the completion of the register configuration phase; the CoreConfigP output signals CONFIG1_DONE and CONIG2_DONE are then asserted.

   There are two phases of register configuration (CONFIG1 and CONFIG2) depending upon the peripherals used in the design.
4. If one or both of MDDR/FDDR are used, and none of the SERDESIF blocks are used in the design, there is only one register configuration phase. Both the CoreConfigP output signals CONFIG1_DONE and CONIG2_DONE are asserted one after the other without any wait/delay.

   If one or more SERDESIF blocks in non-PCIe mode are used in the design, there is only one phase of register configuration. CONFIG1_DONE and CONIG2_DONE are asserted one after the other without any wait/delay.

   If one or more SERDESIF blocks in PCIe mode are used in the design, there are two phases of register configuration. CONFIG1_DONE is asserted after the first phase of register configuration is complete. SERDESIF system and lane registers are configured in this phase. If SERDESIF is configured in a non-PCIE mode, CONFIG2_DONE signal is also asserted immediately.
5. The second phase of register configuration then follows (if SERDESIF is configured in PCIE mode). The following are the different events that happen in the second phase:
   – CoreResetP de-asserts PHY_RESET_N and CORE_RESET_N signals corresponding to each of the SERDESIF blocks used. It also asserts an output signal SDIF_RELEASED after all the SERDESIF blocks are out of reset. This SDIF_RELEASED signal is used to indicate to the CoreConfigP that the SERDESIF core is out of reset and is ready for the second phase of register configuration.
   – Once the SDIF_RELEASED signal is asserted, the SystemInit() function starts polling for the assertion of PMA_READY on the appropriate SERDESIF lane. Once the PMA_READY is asserted, the second set of SERDESIF registers (PCIE registers) are configured/written by the SystemInit() function.
6. After all the PCIE registers are configured, the SystemInit() function writes to the CoreConfigP control registers to indicate the completion of the second phase of register configuration; the CoreConfigP output signal CONIG2_DONE is then asserted.
7. Apart from the above signal assertions/de-assertions, CoreResetP also manages the initialization of the various blocks by performing the following functions:
   – De-asserting the FDDR core reset
   – De-asserting the SERDESIF blocks PHY and CORE resets

- Monitoring of the FDDR PLL (FPLL) lock signal. The FPLL must have locked to guarantee that the FDDR AXI/AHBLite data interface and the FPGA fabric can communicate correctly.
- Monitoring of the SERDESIF block PLL (SPLL) lock signals. The SPLL must have locked to guarantee that the SERDESIF blocks AXI/AHBLite interface (PCIe mode) or XAUI interface can communicate properly with the FPGA fabric.
- Waiting for the external DDR memories to settle and be ready to be accessed by the DDR controllers.

8. When all peripherals have completed their initialization, CoreResetP asserts the INIT_DONE signal; the CoreConfigP internal register INIT_DONE is then asserted.

   If one or both of MDDR/FDDR are used, and the DDR initialization time is reached, CoreResetP output signal DDR_READY is asserted. Assertion of this signal DDR_READY can be monitored as an indication that the DDR (MDDR/FDDR) is ready for communication.

   If one or more SERDESIF blocks are used, and the second phase of register configuration is successfully completed, CoreResetP output signal SDIF_READY is asserted. Assertion of this signal SDIF_READY can be monitored as an indication that all the SERDESIF blocks are ready for communication.
9. The SystemInit() function, which has been waiting for INIT_DONE to be asserted, completes, and the application's main() function is executed. At that time, all used DDR controllers and SERDESIF blocks have been initialized, and the firmware application and the FPGA fabric logic can reliably communicate with them.

The methodology described in this document relies on the Cortex-M3 executing the initialization process as part of the system initialization code executed before the application's main()function.

See the Flow Charts in Figure 1-1, Figure 1-2 and Figure 1-3 for the Initialization steps of FDDR/MDDR, SEREDES(non-PCIe mode) and SERDES (PCIe mode).

Figure 1-4 shows a Peripheral Initialization timing diagram.

Figure 1-1 • **MDDR/FDDR Initialization (No SERDESIF) Initialization Flow Chart**

Figure 1-2 • **SERDESIF (Non-PCIe) Initialization Flow Chart**

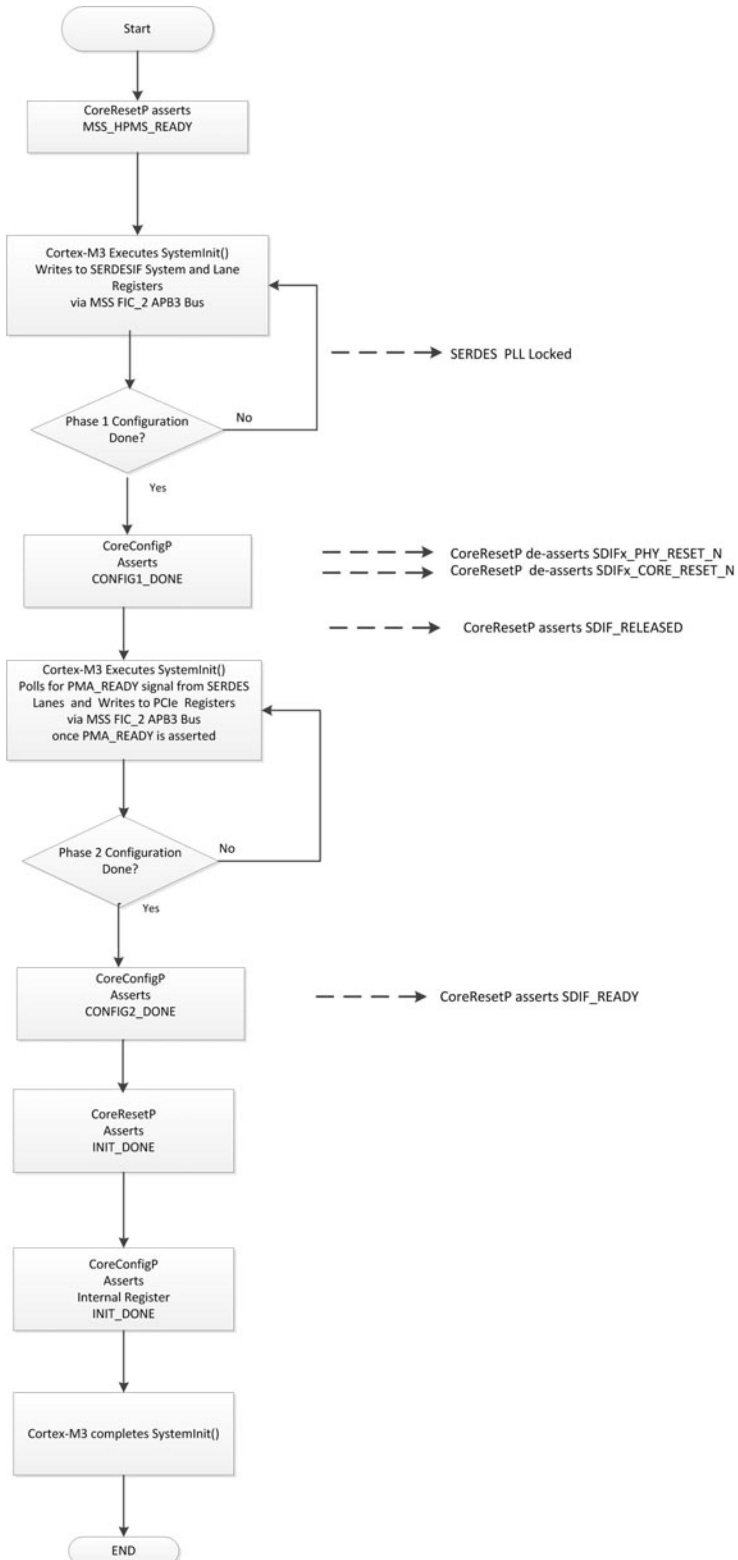

Figure 1-3 • **SERDESIF (PCIe) Initialization Flow Chart**

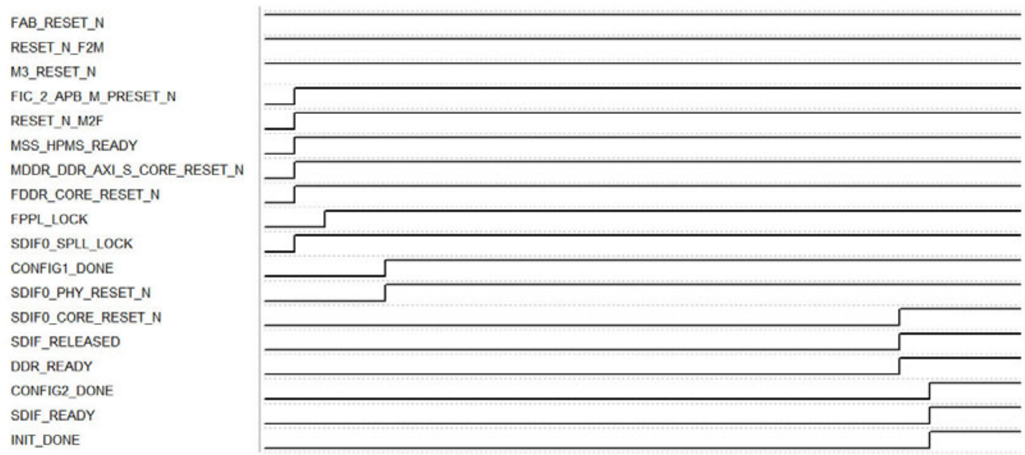

Figure 1-4 **• Peripheral Initialization Timing Diagram**

The initialization procedure described in this document requires you to run Cortex-M3 during the initialization process, even if you are not planning on running any code on the Cortex-M3. You must create a basic firmware application that does nothing (a simple loop, for example) and load that executable in the embedded Non Volatile Memory (eNVM) so the DDR controllers and SERDESIF blocks are initialized when the Cortex-M3 boots.

2 – Using System Builder to Create a Design Using DDR and SERDESIF Blocks

The SmartFusion2 System Builder is a powerful design tool that helps you capture your system-level requirements and produces a design implementing those requirements. A very important function of the System Builder is the automatic creation of the Peripheral Initialization sub-system. "Using SmartDesign to Create a Design Using DDR and SERDESIF Blocks" on page 17 describes in detail how to create such a solution without the System Builder.

If you are using System Builder, you must perform the following tasks to create a design that initializes your DDR controllers and SERDESIF blocks at power up:

1. In the **Device Features** page (Figure 2-1), specify which DDR controllers are used and how many SERDESIF blocks are used in your design.
2. In the **Memory** page, specify the type of DDR (DDR2/DDR3/LPDDR) and the configuration data for your external DDR memories. See the Memory Page section for details.
3. In the **Peripherals** page, add fabric masters configured as AHBLite/AXI to the Fabric DDR Subsystem and/or MSS DDR FIC Subsystem (optional).
4. In the **Clock Settings** page, specify the clock frequencies for the DDR sub-systems.
5. Complete your design specification and click Finish. This generates the System Builder created design, including the logic necessary for the 'initialization' solution.
6. If you are using SERDESIF blocks, you must instantiate the SERDESIF blocks in your design and connect their initializations ports to those of the System Builder generated core.

System Builder Device Features Page

In the Device Features page, specify which DDR controllers (MDDR and/or FDDR) are used and how many SERDESIF blocks are used in your design (Figure 2-1).

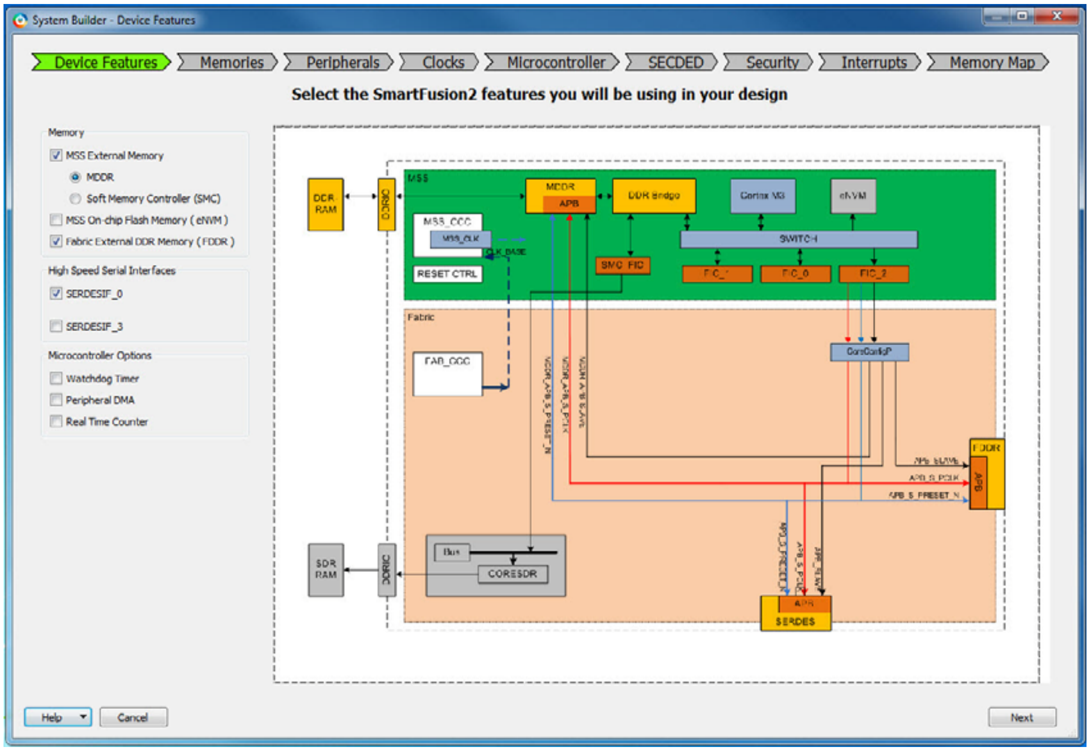

Figure 2-1 • **System Builder Device Features Page**

System Builder Memory Page

To use the MSS DDR (MDDR) or Fabric DDR (FDDR), select the Memory Type from the drop-down list (Figure 2-2).

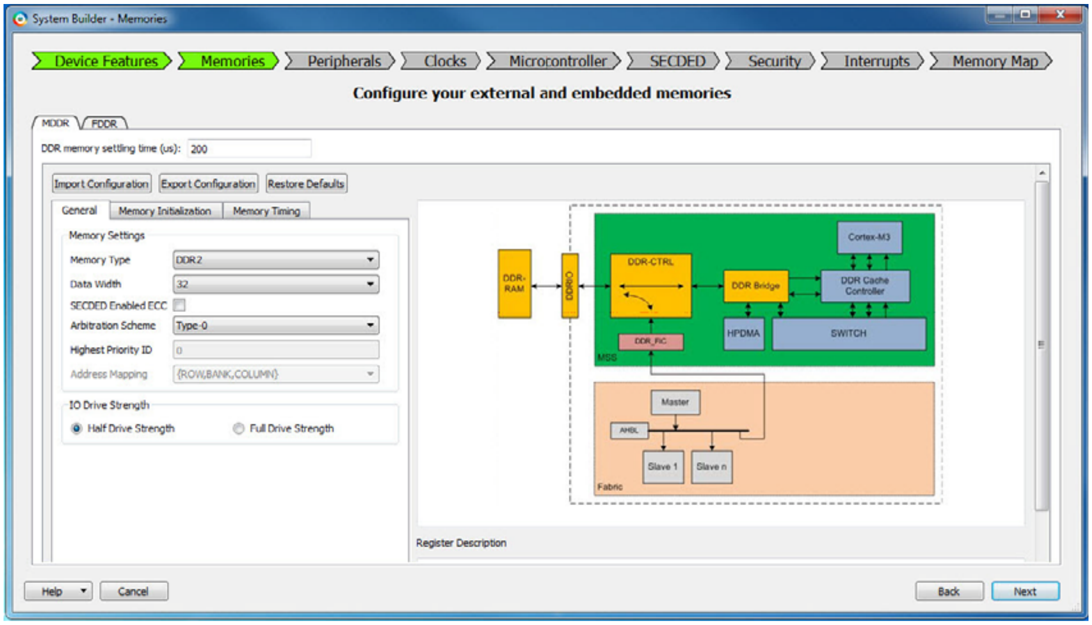

Figure 2-2 **• MSS External Memory**

You must:

1. Select the DDR type (**DDR2**, **DDR3** or **LPDDR**).
2. Define the **DDR memory settling time**. Consult your external DDR Memory Specifications to set the correct memory setting time. The DDR memory may fail to initialize correctly if the memory settling time is not correctly set.
3. Either import the DDR register configuration data or set your DDR Memory Parameters. For details, refer to the Microsemi SmartFusion2 High Speed Serial and DDR Interfaces User's Guide.

This data is used to generate the DDR register BFM and firmware configuration files as described in the "Creating and Compiling the Firmware Application" on page 26 and "BFM Files Used for Simulating the Design" on page 27. For details on DDR controller configuration registers, refer to the Microsemi SmartFusion2 High Speed Serial and DDR Interfaces User's Guide.

An example of the configuration file syntax is shown in Figure 2-3. The register names used in this file are the same as those described in the Microsemi SmartFusion2 High Speed Serial and DDR Interfaces User's Guide.

```
## PHY_16_DDR2_NO_ECC_BL8_INTER

ddrc_dyn_soft_reset_CR          0x00 ;
ddrc_dyn_refresh_1_CR           0x27DE ;
ddrc_dyn_refresh_2_CR           0x030F ;
ddrc_dyn_powerdown_CR           0x02 ;
ddrc_dyn_debug_CR               0x00 ;
ddrc_ecc_data_mask_CR           0x0000 ;
ddrc_addr_map_col_1_CR          0x3333 ;
ddrc_addr_map_col_3_CR          0x3300 ;
ddrc_init_1_CR                  0x0001 ;
ddrc_cke_rstn_cycles_CR1        0x0100 ;
ddrc_cke_rstn_cycles_CR2        0x0008 ;
ddrc_init_emr2_CR               0x0000 ;
ddrc_init_emr3_CR               0x0000 ;
ddrc_dram_bank_act_timing_CR    0x1947;
ddrc_odt_param_1_CR             0x0010 ;
ddrc_odt_param_2_CR             0x0000 ;
ddrc_debug_CR                   0x3300 ;
ddrc_mode_reg_rd_wr_CR          0x0000 ;
ddrc_mode_reg_data_CR           0x0000 ;
ddrc_pwr_save_2_CR              0x0000 ;
ddrc_hpr_queue_param_CR1        0x80F8 ;
ddrc_hpr_queue_param_CR2        0x0007 ;
ddrc_lpr_queue_param_CR1        0x80F8 ;
ddrc_lpr_queue_param_CR2        0x0007 ;
ddrc_wr_queue_param_CR          0x0200 ;
ddrc_dfi_min_ctrlupd_timing_CR  0x0003 ;
ddrc_dfi_max_ctrlupd_timing_CR  0x0040 ;
ddrc_dfi_wr_lvl_control_CR1     0x0000 ;
ddrc_dfi_wr_lvl_control_CR2     0x0000 ;
ddrc_dfi_rd_lvl_control_CR1     0x0000 ;
ddrc_dfi_rd_lvl_control_CR2     0x0000 ;
ddrc_dfi_ctrlupd_time_interval_CR  0x0309 ;
ddrc_perf_param_3_CR            0x0000 ;
ddrc_ecc_int_clr_reg            0x0000 ;
```

Figure 2-3 **• Configuration File Syntax Example**

System Builder Peripherals Page

In the Peripherals page, for each DDR controller a separate subsystem is created (Fabric DDR Subsystem for FDDR and MSS DDR FIC Subsystem for MDDR). You can add a Fabric AMBA Master (configured as AXI/AHBLite) core to each of these subsystems to enable fabric master access to the DDR controllers. Upon generation, System Builder automatically instantiates bus cores (depending on the type of AMBA Master added) and exposes the master BIF of the bus core and the clock and reset pins of the corresponding subsystems (FDDR/MDDR) under appropriate pin groups, to the top. All you have to do is connect the BIFs to the appropriate Fabric Master cores that you would instantiate in the design. In the case of MDDR, it is optional to add a Fabric AMBA Master core to the MSS DDR FIC Subsystem; Cortex-M3 is a default master on this subsystem. Figure 2-4 shows the System Builder Peripherals Page.

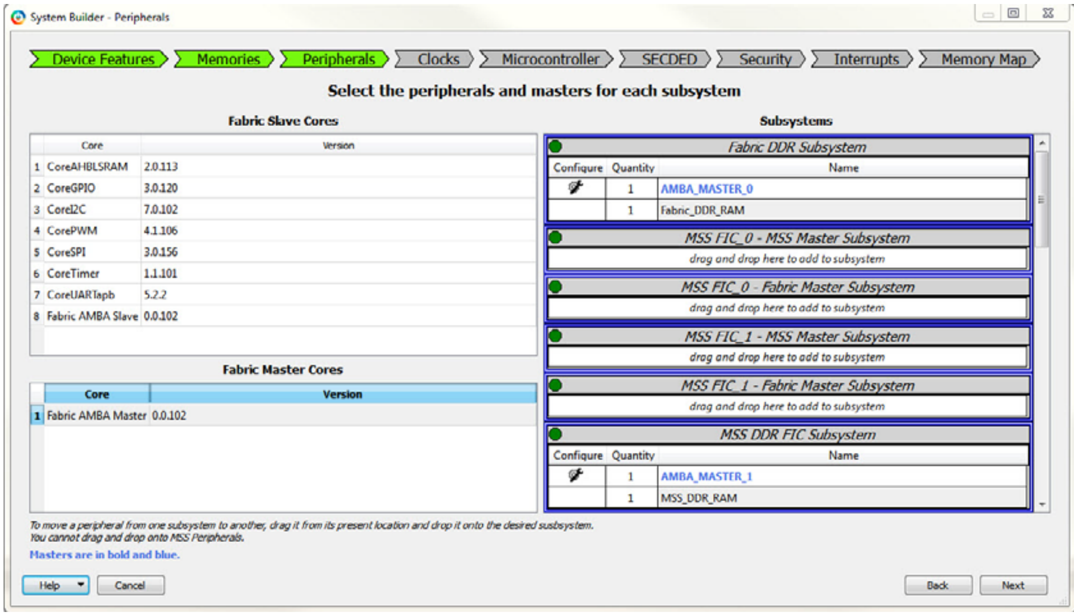

Figure 2-4 **• System Builder Peripherals Page**

System Builder Clock Settings Page

In the Clock Settings page, for each DDR controller, you must specify the clock frequencies related to each DDR (MDDR and/or FDDR) sub-system.

For MDDR, you must specify:

- **MDDR_CLK** - This clock determines the operating frequency of the DDR Controller and should match the clock frequency you wish your external DDR memory to run at. This clock is defined as a multiple of the M3_CLK (Cortex-M3 and MSS Main Clock, Figure 2-5). The MDDR_CLK must be less than 333 MHz.
- **DDR_FIC_CLK** - If you have chosen to also access the MDDR from the FPGA fabric, you need to specify the DDR_FIC_CLK. This clock frequency is defined as ratio of the MDDR_CLK and should match the frequency at which the FPGA fabric sub-system that accesses the MDDR is running.

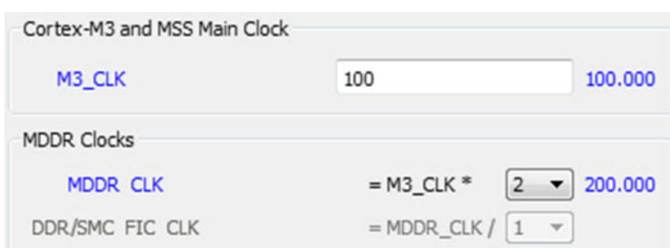

Figure 2-5 **• Cortex-M3 and MSS Main Clock; MDDR Clocks**

For FDDR, you must specify:

- **FDDR_CLK** - Determines the operating frequency of the DDR Controller and should match the clock frequency at which you wish your external DDR memory to run. Note that this clock is defined as a multiple of the M3_CLK (MSS and Cortex-M3 clock, Figure 2-5). The FDDR_CLK must be within 20 MHz and 333 MHz.

- **FDDR_SUBSYSTEM_CLK** - This clock frequency is defined as a ratio of the FDDR_CLK and should match the frequency at which the FPGA fabric sub-system that accesses the FDDR is running.

Figure 2-6 **• Fabric DDR Clocks**

SERDESIF Configuration

The SERDESIF blocks are not instantiated in the System Builder generated design. However, for all the SERDESIF blocks, initialization signals are available at the interface of the System Builder core and can be connected to the SERDESIF cores at the next level of hierarchy, as shown in Figure 2-7.

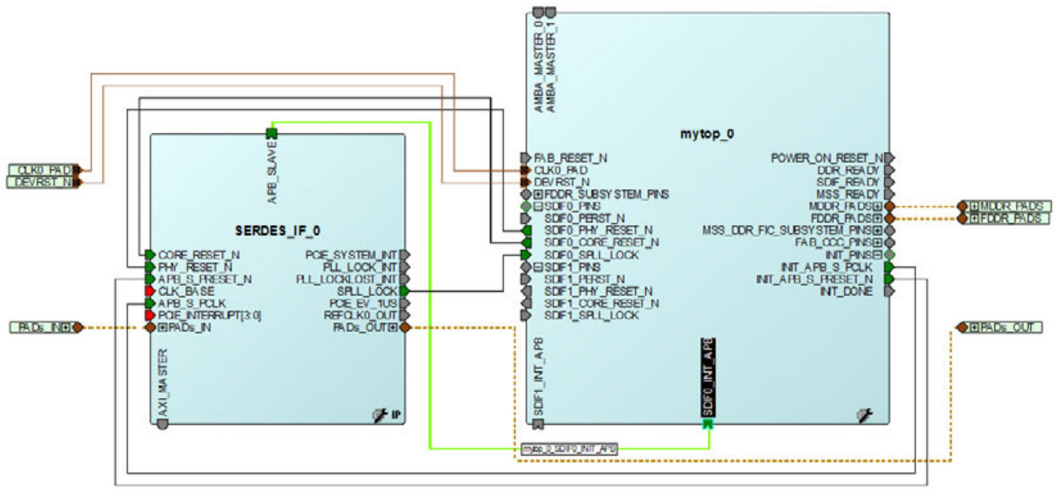

Figure 2-7 **• SERDESIF Peripheral Initialization Connectivity**

Similar to the DDR configuration registers, each SERDES block also has configuration registers that must be loaded at runtime. You can either import these register values or use the High Speed Serial Interface Configurator (Figure 2-8) to enter your PCIe or EPCS parameters and the register values are automatically computed for you. For details, refer to the SERDES Configurator User's Guide.

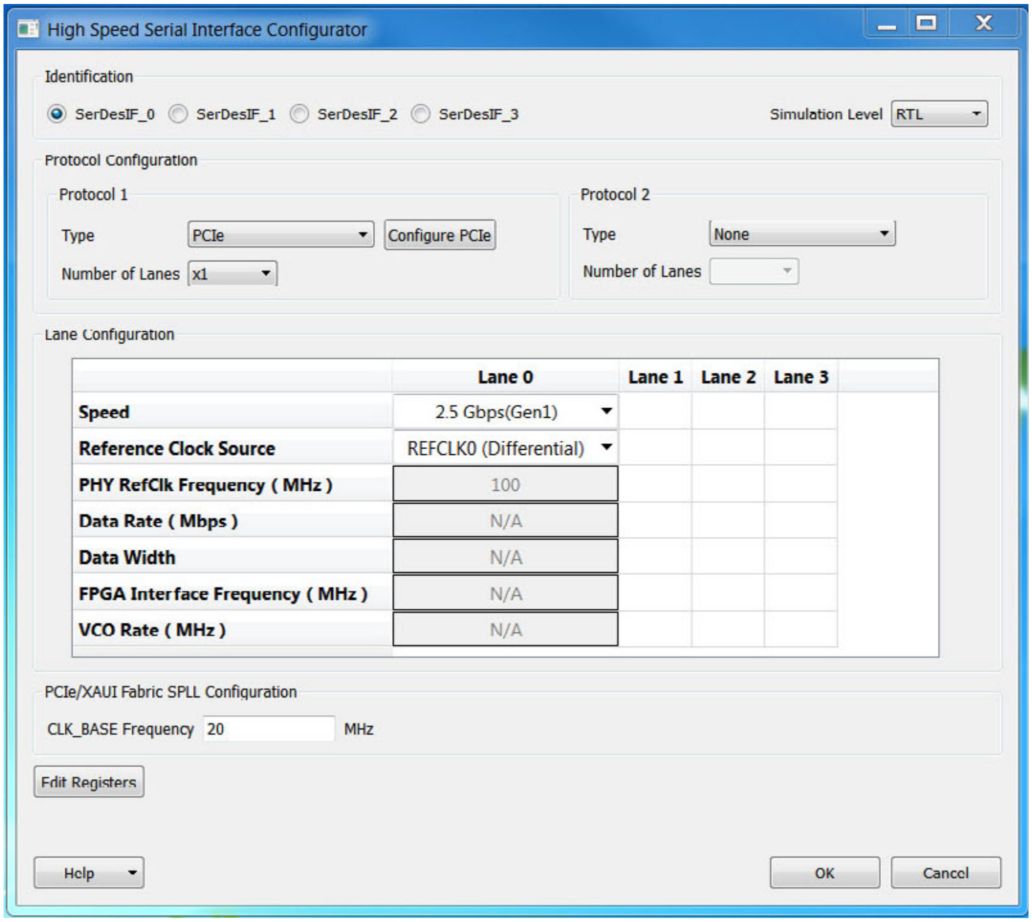

Figure 2-8 • High Speed Serial Interface Configurator

Once you have integrated your user logic with the System Builder block and SERDES block, you can generate your top level SmartDesign. This generates all HDL and BFM files that are necessary to implement and simulate your design. You can then proceed with the rest of the Design Flow.

3 – Using SmartDesign to Create a Design Using DDR and SERDESIF Blocks

This section describes how to put a complete 'initialization' solution together without using the SmartFusion2 System Builder. The goal is to help you understand what you must do if you do not wish to use the System Builder. This section also describes what the System Builder tool actually generates for you. This section describes how to:

- Input the configuration data for DDR controller and SERDESIF configuration registers.
- Instantiate and connect the Fabric Cores required to transfer the configuration data to the DDR controllers and SERDESIF configuration registers.

DDR Controller Configuration

The MSS DDR (MDDR) and Fabric DDR (FDDR) controllers must be configured dynamically (at runtime) to match the external DDR memory configuration requirements (DDR mode, PHY width, burst mode, ECC, etc.). Data entered in MDDR/FDDR configurator is written to the DDR controller configuration registers by the CMSIS SystemInit() function. The Configurator has three different tabs for entering different types of configuration data:

- General data (DDR mode, Data Width, Clock Frequency, ECC, Fabric Interface, Drive Strength)
- Memory Initialization data (Burst Length, Burst Order, Timing Mode, Latency, etc.)
- Memory Timing data

Refer to the specifications of your external DDR memory and configure the DDR Controller to match the requirements of your external DDR memory.

For details on DDR configuration, refer to the SmartFusion2 MSS DDR Configuration User Guide.

SERDESIF Configuration

Double-click the SERDES block in the SmartDesign canvas to open the Configurator to configure the SERDES (Figure 3-1). You can either import these register values or use the SERDES configurator to enter your PCIe or EPCS parameters and the register values is automatically computed for you. For details, refer to the SERDES Configurator User's Guide.

.

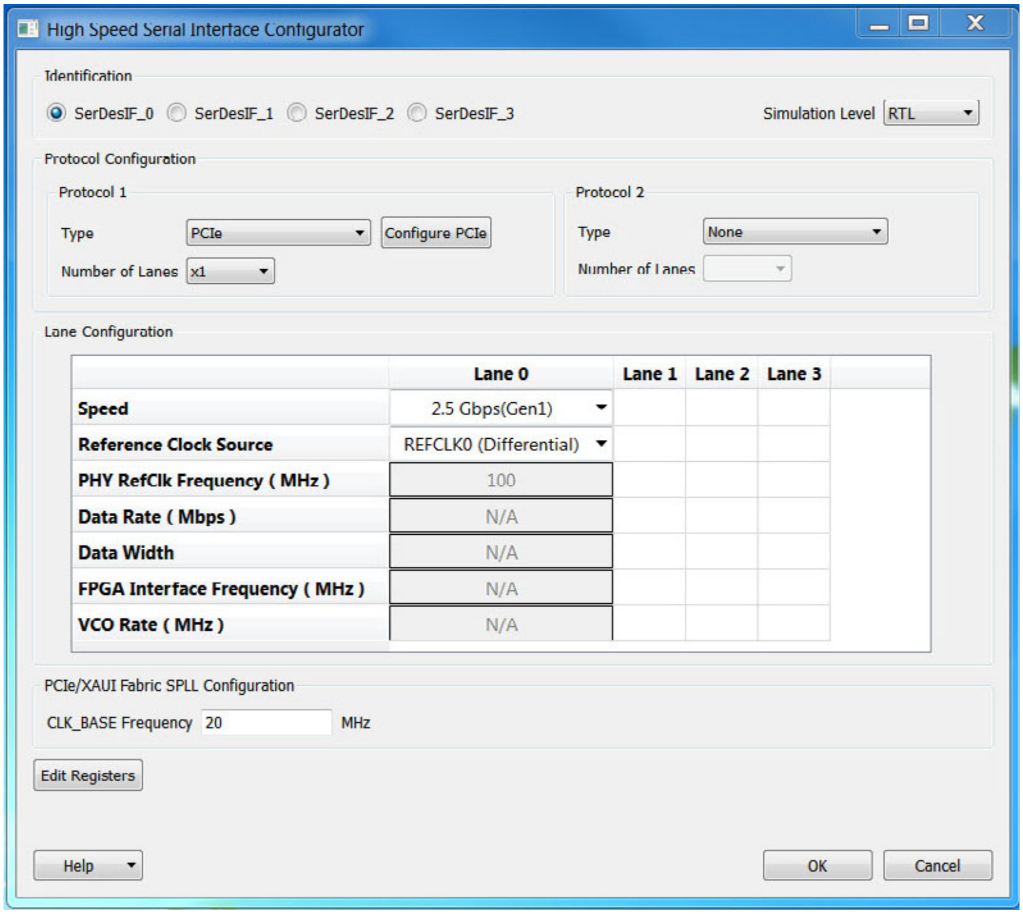

Figure 3-1 **• High Speed Serial Interface Configurator**

Creating the FPGA Design Initialization Sub-System

To initialize the DDR and SERDESIF blocks, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the Cortex-M3 to the DDR and SERDESIF configuration registers, manages the reset sequences required for these blocks to be operational and signals when these blocks are ready to communicate with the rest of your design. To create the initialization subsystem, you must:

- Configure FIC_2 inside the MSS
- Instantiate and configure the CoreConfigP and CoreResetP cores
- Instantiate the on-chip 25/50MHz RC oscillator
- Instantiate the System Reset (SYSRESET) macro
- Connect these components to each peripheral's configuration interfaces, clocks, resets and PLL lock ports

MSS FIC_2 APB Configuration

To configure the MSS FIC_2:

1. Open the FIC_2 configurator dialog box from the MSS configurator (Figure 3-2).
2. Select **Initialize peripherals using Cortex-M3**.

3. Depending on your system, check one or both of the following checkboxes:
   – MSS DDR
   – Fabric DDR and/or SERDES Blocks
4. Click **OK** and proceed to generate the MSS (you may defer this action until you have fully configured the MSS to your design requirements). The FIC_2 ports (FIC_2_APB_MASTER, FIC_2_APB_M_PCLK and FIC_2_APB_M_RESET_N) are now exposed at the MSS interface and can be connected to the CoreConfigP and CoreResetP cores.

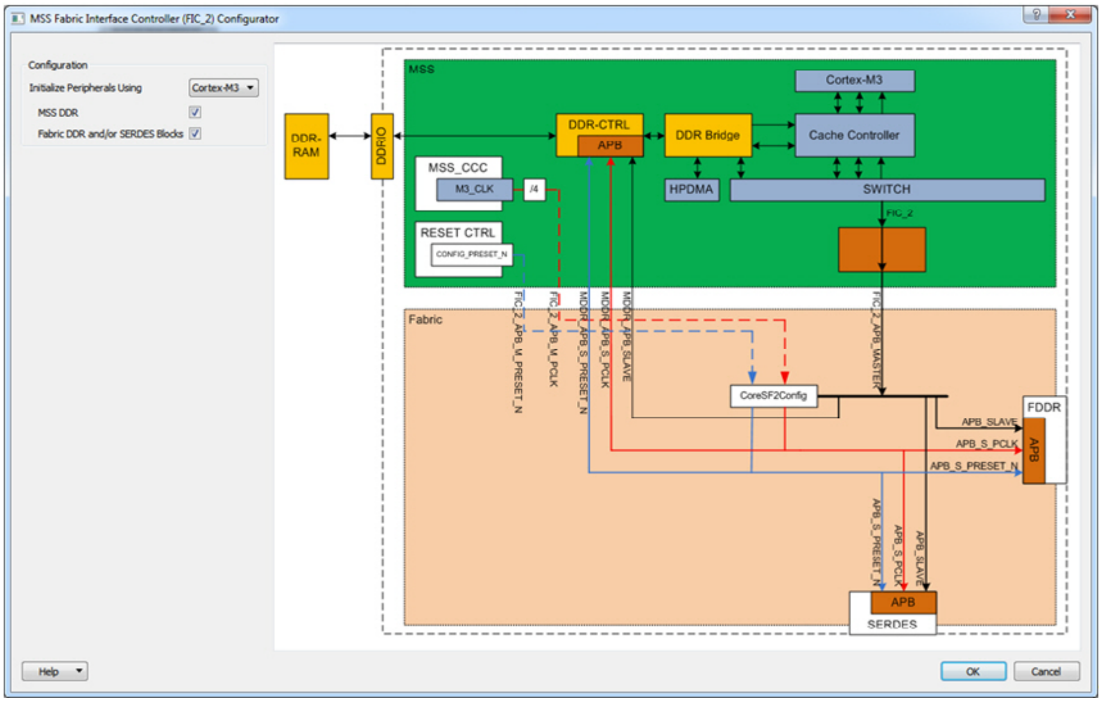

Figure 3-2 **• MSS FIC_2 Configurator**

CoreConfigP

To configure CoreConfigP:

1. Instantiate CoreConfigP into your SmartDesign (typically the one where the MSS is instantiated). This core can be found in the Libero Catalog (under Peripherals).
2. Double-click the core to open the configurator.

3. Configure the core to specify which peripherals need to be initialized (Figure 3-3)

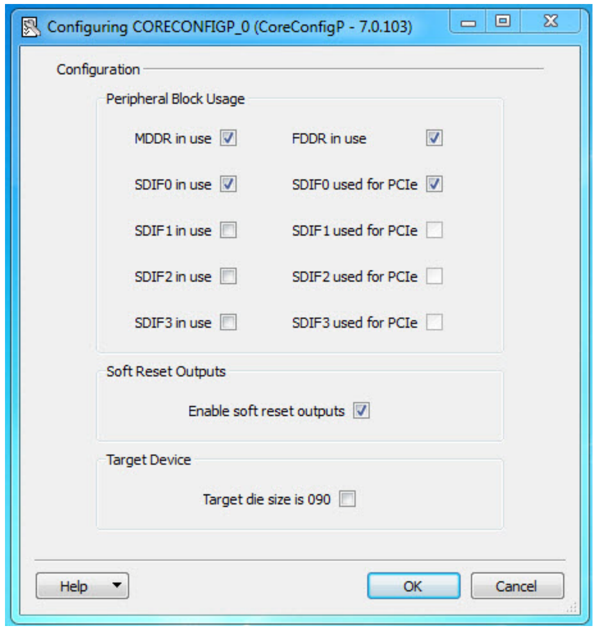

Figure 3-3 **• CoreConfigP Dialog Box**

CoreResetP

To configure CoreResetP:

1. Instantiate CoreResetP into your SmartDesign (typically the one where the MSS is instantiated). This core can be found in the Libero Catalog, under Peripherals.
2. Double-click the core inside the SmartDesign Canvas to open the Configurator (Figure 3-4).
3. Configure the core to:
   - Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
     o EXT_RESET_OUT is never asserted
     o EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
     o EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
     o EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) or FAB_RESET_N is asserted
   - Specify the Device Voltage. The selected value should match the voltage you selected in the Libero Project Settings dialog box.
   - Check the appropriate checkboxes to indicate which peripherals you are using in your design.
   - Specify the external DDR memory setting time. This is the maximum value for all DDR memories used in your application (MDDR and FDDR). Refer to the external DDR memory vendor datasheet to configure this parameter. 200us is a good default value for DDR2 and DDR3 memories running at 200MHz. This is a very important parameter to guarantee a working simulation and a working system on silicon. An incorrect value for the settling time may result in simulation errors. Refer to the DDR memory vendor datasheet to configure this parameter.
   - For each SERDES block in your design, check the appropriate boxes to indicate whether:
     o PCIe is used
     o Support for PCIe Hot Reset is required
     o Support for PCIe L2/P2 is required

Note: If you are using the 090 die(M2S090) and your design uses SERDESIF, you do not have to check any of the following checkboxes: 'Used for PCIe', 'Include PCIe HotReset support' and 'Include PCIe L2/P2 support'. If you are using any non-090 device and using one or more SERDESIF blocks, you have to check all four checkboxes under the appropriate SERDESIF section.

Note: For details on the options available to you in this configurator, refer to the CoreResetP Handbook.

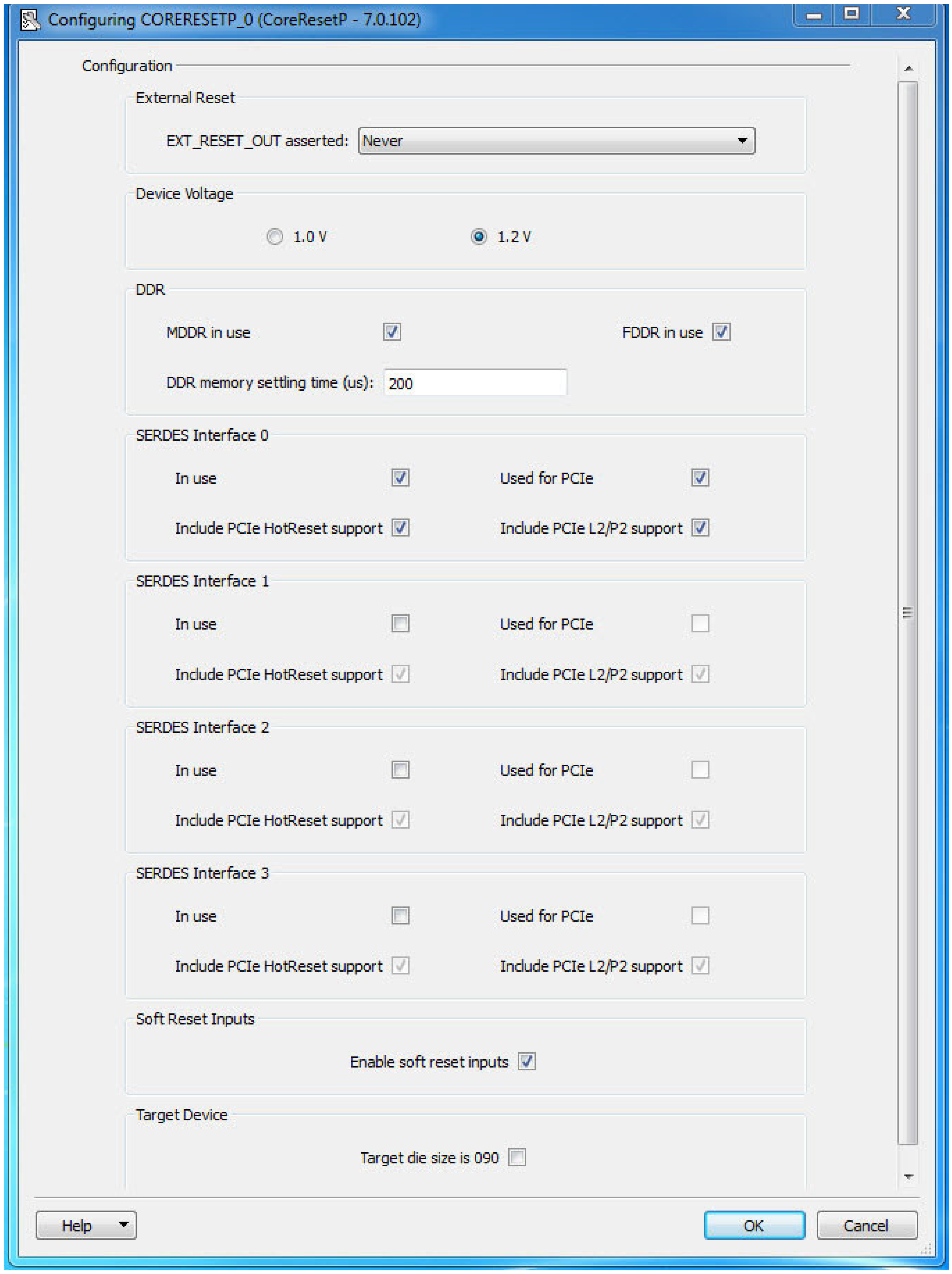

***Figure 3-4* • CoreResetPConfigurator**

25/50MHz Oscillator Instantiation

CoreConfigP and CoreResetP are clocked by the on-chip 25/50MHz RC oscillator. You must instantiate a 25/50MHz Oscillator and connect it to these cores.

1. Instantiate the Chip Oscillators core into your SmartDesign (typically the one where the MSS is instantiated). This core can be found in the Libero Catalog under Clock & Management.
2. Configure this core such that the RC oscillator drives the FPGA fabric, as shown in Figure 3-5.

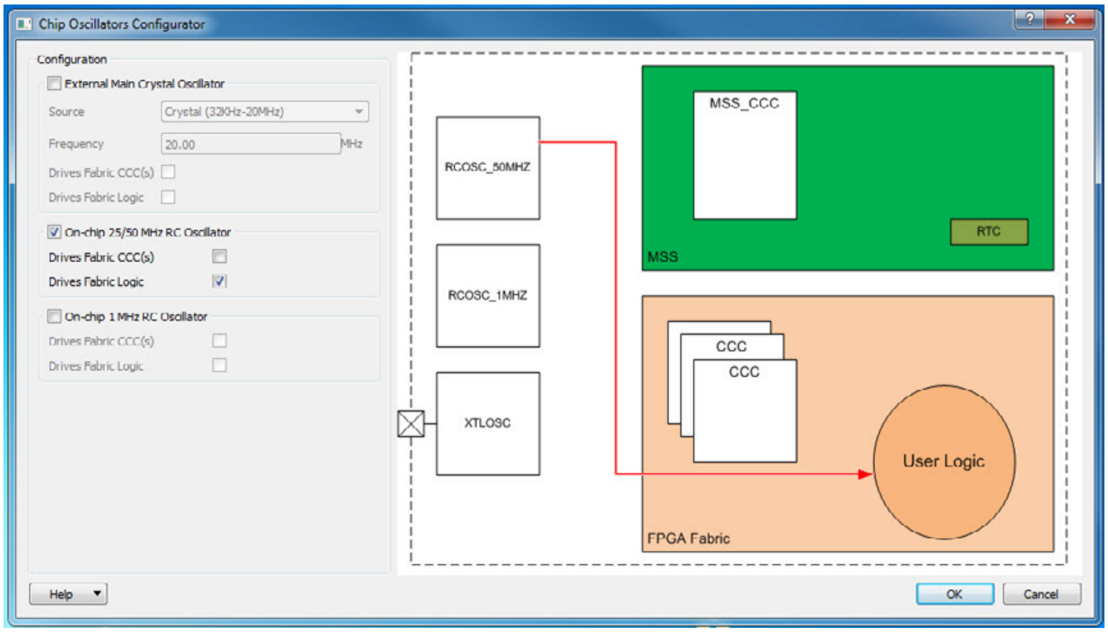

Figure 3-5 **• Chip Oscillators Configurator**

System Reset (SYSRESET) Instantiation

The SYSRESET macro provides device level reset functionality to your design. The POWER_ON_RESET_N output signal is asserted/de-asserted whenever the chip is powered up or the external pin DEVRST_N is asserted/de-asserted (Figure 3-6).

Instantiate the SYSRESET macro into your SmartDesign (typically the one where the MSS is instantiated). This macro can be found in the Libero Catalog under Macro Library.No configuration of this macro is necessary.

Figure 3-6 **• SYSRESET Macro**

Overall Connectivity

After you have instantiated and configured the MSS, FDDR, SERDESIF, OSC, SYSRESET, CoreConfigP and CoreResetP cores in your design, you need to connect them to form the Peripheral Initialization sub-system. To simplify the connectivity description in this document, it is broken into the APB3 compliant configuration data path connectivity associated with the CoreConfigP and the CoreResetP related connections.

Configuration Data Path Connectivity

Figure 3-7 shows how to connect the CoreConfigP to the MSS FIC_2 signals and the peripherals' APB3 compliant configuration interfaces.

Table 3-1 • **Configuration Data Path Port/BIF Connections**

FROM Port/Bus Interface (BIF)/ Component	**TO Port/Bus Interface (BIF)/Component**		
APB_S_PRESET_N/ CoreConfigP	APB_S_PRESET_N/ SDIF<0/1/2/3>	APB_S_PRESET_N/ FDDR	MDDR_APB_S_PRESET_N/MSS
APB_S_PCLK/ CoreConfigP	APB_S_PCLK/SDIF<0/ 1/2/3>	APB_S_PCLK/FDDR	MDDR_APB_S_PCLK/ MSS
MDDR_APBmslave/ CoreConfig			MDDR_APB_SLAVE (BIF)/MSS
SDIF<0/1/2/ 3>_APBmslave/Config	APB_SLAVE (BIF)/ SDIF<0/1/2/3>		
FDDR_APBmslave		APB_SLAVE (BIF)/ FDDR	
FIC_2_APBmmaster/ CoreConfigP			FIC_2_APB_MASTER/ MSS

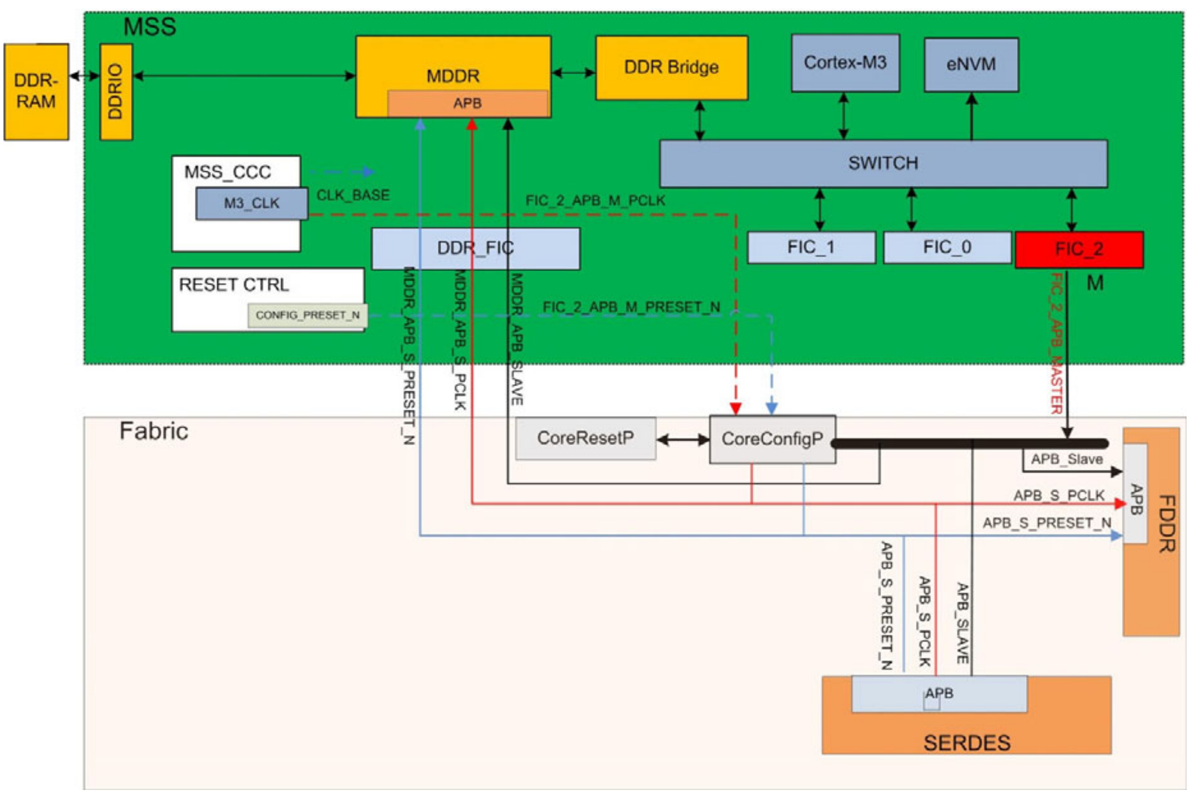

Figure 3-7 • **FIC_2 APB3 Sub-System Connectivity**

Clocks and Resets Connectivity

Figure 3-8 shows how to connect the CoreResetP to the external reset sources and the peripherals' core reset signals. It also shows how to connect the CoreResetP to the peripherals' clock synchronization status signals (PLL lock signals). In addition, it shows how the CoreConfigP and CoreResetP are connected.

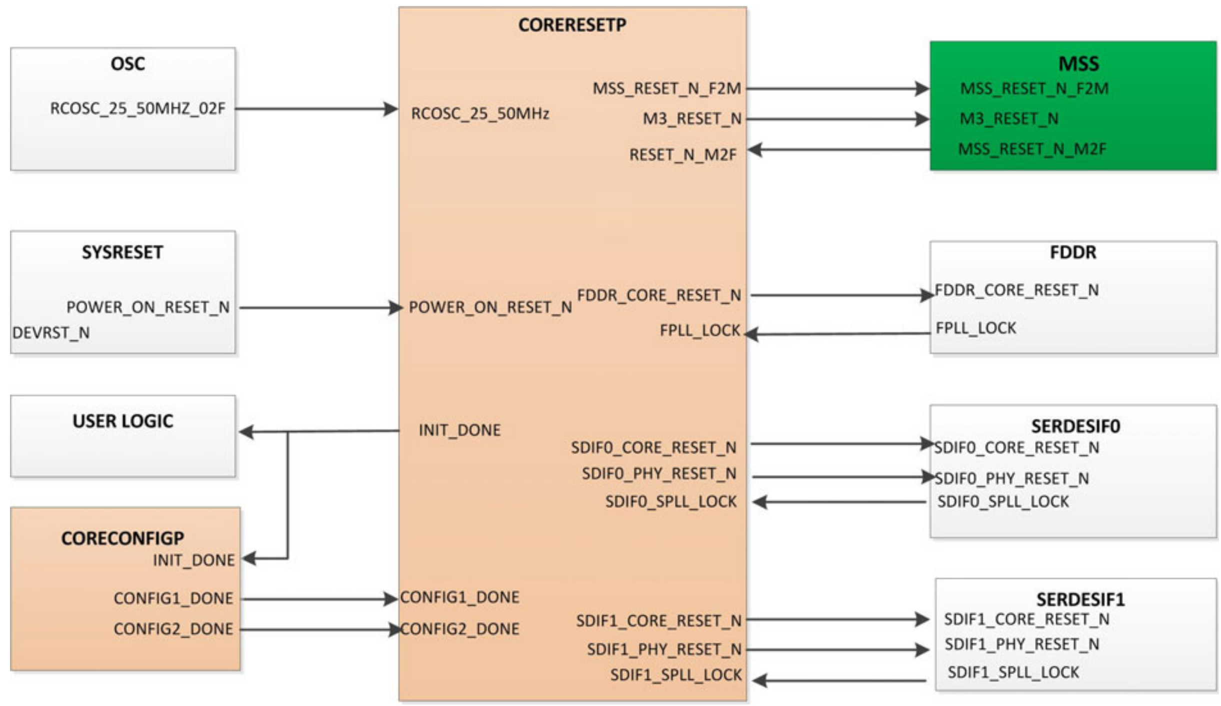

***Figure 3-8* • Core SF2Reset Sub-System Connectivity**

4 – Creating and Compiling the Firmware Application

When you export the firmware from LiberoSoC (Design Flow Window > Export Firmware > Export Firmware), Libero generates the following files in the <project_folder>/firmware/drivers_config/ sys_config folder:

- **sys_config.c** - Contains the data structures that hold the values for the peripheral registers.
- **sys_config.h** - Contains the #define statements that specify which peripherals are used in the design and need to be initialized.
- **sys_config_mddr_define.h** - Contains the MDDR controller configuration data entered in the Registers Configuration dialog box.
- **sys_config_fddr_define.h** - Contains the FDDR controller configuration data entered in the Registers Configuration dialog box.
- **sys_config_mss_clocks.h** - This file contains the MSS clock frequencies as defined in the MSS CCC configurator. These frequencies are used by the CMSIS code to provide correct clock information to many of the MSS drivers that must have access to their Peripheral Clock (PCLK) frequency (e.g., MSS UART baud rate divisors are a function of the baud rate and the PCLK frequency).
- **sys_config_SERDESIF_<n>.c** - Contains the SERDESIF_<n> register configuration data provided during the SERDESIF_<n> block configuration in design creation.
- **sys_config_SERDESIF_<n>.h** - Contains the #define statements that specify the number of register configuration pairs and the lane number that needs to be polled for PMA_READY(only in PCIe mode).

These files are required for the CMSIS code to compile properly and contain information regarding your current design, including peripheral configuration data and clock configuration information for the MSS. Do not edit these files manually; they are created to the corresponding component/peripheral directories every time the SmartDesign components containing the respective peripherals are generated. If any changes are made to the configuration data of any of the peripherals, you need to re-export the firmware projects so that the updated firmware files (see the list above) are exported to the <project_folder>/ firmware/drivers_config/sys_config folder.

When you export the firmware, Libero SoC creates the firmware projects: a library where your design configuration files and drivers are compiled.

If you check the Create project <SoftConsole/IAR/Keil> checkbox when you export the firmware, a software SoftConsole/IAR/Keil project is created to hold the application project where you can edit the main.c and user C/H files. Open the SoftConSole/IAR/Keil project to compile the CMSIS code correctly and have your firmware application properly configured to match your hardware design.

5 – BFM Files Used for Simulating the Design

When you generate the SmartDesign components containing the peripherals associated with your design, the simulation files corresponding to the respective peripherals are generated in the <project dir>/simulation directory:

- **test.bfm** - Top-level BFM file that is first executed during any simulation that exercises the SmartFusion2 MSS Cortex-M3 processor. It executes peripheral_init.bfm and user.bfm, in that order.
- **MDDR_init.bfm** - If your design uses the MDDR, Libero generates this file; it contains BFM write commands that simulate writes of the MSS DDR configuration register data you entered (using the Edit Registers dialogbox or in the MSS_MDDR GUI) into the MSS DDR Controller registers.
- **FDDR_init.bfm** - If your design uses the FDDR, Libero generates this file; it contains BFM write commands that simulate writes of the Fabric DDR configuration register data you entered (using the Edit Registers dialogbox or in the FDDR GUI) into the Fabric DDR Controller registers.
- **SERDESIF_<n>_init.bfm** - If your design uses one or more SERDESIF blocks, Libero generates this file for each of the SERDESIF_<n> blocks used; it contains BFM write commands that simulate writes of the SERDESIF configuration register data you entered (using the Edit Registers dialog box or in the SERDESIF_<n> GUI) into the SERDESIF_<n> registers. If the SERDESIF block is configured as PCIe, this file also has some #define statements that control the execution of the 2 register configuration phases in perfect order.
- **user.bfm** - Contains the user commands. These commands are executed after peripheral_init.bfm has completed. Edit this file to enter your BFM commands.
- **SERDESIF_<n>_user.bfm** - Contains the user commands. Edit this file to enter your BFM commands. Use this if you have configured SERDESIF_<n> block in BFM PCIe simulation mode and as an AXI/AHBLite master. If you have configured SERDESIF_<n> block in RTL simulation mode, you will not need this file.

When you invoke simulation every time, the following two simulation files are re-created to the <project dir>/simulation directory with updated contents:

- **subsystem.bfm** - Contains the #define statements for each peripheral used in your design, that specify the particular section of the peripheral_init.bfm to be executed corresponding to each peripheral.
- **operipheral_init.bfm** - Contains the BFM procedure that emulates the CMSIS::SystemInit() function run on the Cortex-M3 before you enter the main() procedure. It copies the configuration data for any peripheral used in the design to the correct peripheral configuration registers and then waits for all the peripherals to be ready before asserting that you can use these peripherals. It executes MDDR_init.bfm and FDDR_init.bfm.

Using these generated files, the DDR controllers in your design are configured automatically, simulating what would happen on a SmartFusion2 device. You can edit the user.bfm file to add any commands required to simulate your design (Cortex-M3 is the master). These commands are executed after the peripherals have been initialized. Do not edit the test.bfm, subsystem.bfm, peripheral_init.bfm, MDDR_init.bfm, FDDR_init.bfm files and the SERDESIF_<n>_init.bfm files.

A – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call 800.262.1060
From the rest of the world, call 650.318.4460
Fax, from anywhere in the world, 408.643.6913

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

5-02-00384-1/08.14