SmartFusion2

DDR Controller and Serial High Speed Controller Standalone Initialization Methodology



Introduction

When creating a design using a SmartFusion2 device, if you use any of the two DDR controllers (FDDR or MDDR) or 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.

In this document, we describe all the steps necessary to create a Libero design that automatically initializes the DDR controller and SERDESIF blocks at power up, with the Standalone Initialization mode ON. We also describe how to generate the firmware code from Libero SoC that will be used in the embedded design flow.

First we provide a detailed description of the theory of operation. We introduce the major components of the Peripheral Initialization Solution and outline how they interact.

Unlike the normal flow (Standalone Initialization OFF) where the initialization solution is created by the System Builder, in the case of Standalone Initialization mode ON, the initialization solution has to be put together in SmartDesign using different soft IP cores (mentioned in the latter sections), whether you choose to use System Builder or not. System Builder will not create any initialization logic for any of the peripherals. You have to build the initialization logic that sits outside the System Builder block, should you choose to use System Builder at all.

Note that as the name suggests, the standalone initialization logic has to be built separately for each of the peripherals (DDR/SERDES) used.

Next, we describe how to build designs with the Standalone Initialization mode ON in cases where you choose to use System Builder and in cases where you choose not to.

In this section we address:

- 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

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

1 – Theory of Operation

The Standalone Peripheral Initialization solution for each peripheral uses the following major components:

- The CoreABC soft IP core, which has to be loaded with a program to initialize the peripheral's configuration registers, so that it orchestrates the initialization process. The program contains the registers specific to a peripheral that's being initialized.
- The CoreConfigP soft IP core, whose function is to initialize the peripherals' configuration registers.
- The CoreResetP soft IP core, whose function is to manage the reset sequence of the MSS, DDR controllers, and SERDESIF blocks.

One set of these 3 soft IP cores is dedicated to initialize a single peripheral, and similar logic involving these cores should be built separately for each peripheral used in the design.

The peripheral initialization process works as follows:

1. Upon reset, the CoreABC runs the program it is loaded with.

2. The program starts writing to the registers of the peripheral being initialized. If the peripheral is MDDR/FDDR, then the program writes configuration data to the DDR controllers, and if the peripheral is SERDES, then the program writes the SERDESIF configuration registers, via the CoreABC master BIF. This interface is connected to the soft CoreConfigP core instantiated in the FPGA fabric.

3. After all the registers are configured, the CoreABC program writes to the CoreConfigP control registers to indicate the completion of the register configuration phase; the CoreConfigP output signal 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 the peripheral being initialized is DDR (FDDR/MDDR), then both the signals CONFIG1_DONE and CONFIG2_DONE are asserted at the same time.

5. If the peripheral being initialized is SERDESIF, then there are 2 phases of register configuration depending upon whether SERDES is configured in PCIE mode or not.

   - CONFIG1_DONE is asserted after the first phase of register configuration is complete. SERDESIF system and lane registers are configured in this phase. If SERDES is configured in a non-PCIE mode, then CONFIG2_DONE signal is also asserted immediately.

   - The second phase of register configuration then follows (if SERDESIF is configured in PCIE mode). The following are the different events that happen in this second phase:
     - Once CoreResetP de-asserts PHY_RESET_N and CORE_RESET_N signals of the SERDESIF blocks, it also asserts an output signal SDIF_RELEASED.
     - Once the SDIF_RELEASED signal is asserted, the CoreABC program 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 CoreABC program.
     - After all the PCIE registers are configured, the CoreABC program 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.

6. Apart from the above signal assertions/de-assertions, CoreResetP also manages the initialization of the peripheral being initialized by performing the following functions (depending upon the peripheral being initialized):
    - De-asserting the MDDR/FDDR core reset
    - De-asserting the SERDESIF blocks PHY and CORE resets
    - Monitoring of the FDDR PLL (FPLL) lock signal. The FPLL must be 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.

7. When the peripheral is initialized and is ready to communicate, CoreResetP asserts the INIT_DONE signal; the CoreConfigP internal register INIT_DONE is then asserted.
    - If the peripheral is MDDR/FDDR, 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 the peripheral is SERDESIF, 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 this SERDESIF block is ready for communication.

8. The CoreABC program which has been waiting for INIT_DONE to be asserted completes its execution now.

Note: In case of a SmartFusion2 design with Cortex-M3 processor waiting to communicate with the peripheral, the application's main() function should wait for the assertion of INIT_DONE (OR DDR_READY/SDIF_READY based on which peripheral is being used) signal of CoreResetP before it is executed.

In case of a SmartFusion2 design using more than 1 peripheral (combination of MDDR/FDDR/SERDESIF_n) with Cortex-M3 waiting to communicate with the peripherals, the application's main() function should wait for the assertion of INIT_DONE (OR DDR_READY/SDIF_READY based on which peripheral is being used) signals of all the CoreResetP cores each corresponding to 1 peripheral used. 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 CoreABC executing the initialization process as part of its program (microcode).

Please note that the initialization procedure explained in this document does NOT require you to run Cortex-M3 during the initialization process if you are not planning on running any code on the Cortex-M3. All the initialization logic is taken care by the CoreABC program and the soft IP cores CoreConfigP and CoreResetP.

2 – Switching the Standalone Initialization Mode ON

You can turn the Standalone Initialization mode ON when you first create a project for SmartFusion2 in the Design Methodology section in the New Project dialog (Figure 1).



Figure 1 • Design Methodology – Use Standalone Initialization for MDDR/FDDR/SERDES

If you already have your project open you can turn the Standalone Initialization mode ON from the Project Settings → Design Flow window (Figure 2).

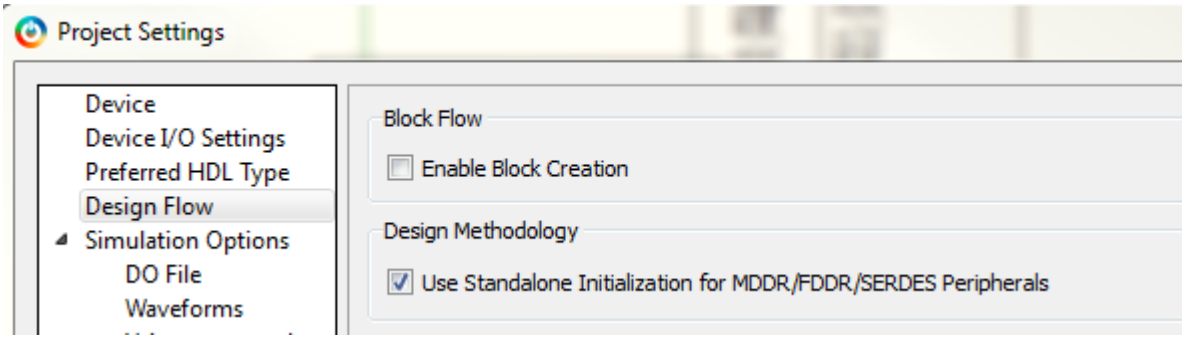


Figure 2 • Standalone Initialization from Project Settings window

3 – Using System Builder to Create a Design Using DDR 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. With the Standalone Initialization mode ON, if you are building a design using MDDR/FDDR, you can choose to use System Builder which automatically instantiates and configures the MDDR(MSS)/FDDR blocks. Alternatively, without using System Builder also, you can just instantiate MSS/FDDR (to use MDDR/FDDR blocks) manually to build your design. In any case, the peripheral initialization logic using CoreABC, CoreConfigP and CoreResetP has to be built manually for every peripheral you use. If you are building a design using SERDESIF and fabric logic only (that means if you don't want to use anything else in the MSS), you don't have to use System Builder at all. Build everything using regular Smart Design. In "Using SmartDesign to Create a Design Using DDR and SERDESIF Blocks" on page 29 we describe 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 will instantiate and configure your DDR blocks (MDDR/FDDR), and then create and interface the initialization logic required to initialize the DDR blocks (MDDR/FDDR).

1. In the **Device Features** page (Figure 3), specify which DDR controllers 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, and configure the Chip Oscillator and Fabric CCC resources required to drive the fabric logic outside the System Builder block.
5. Complete your design specification and click Finish. System Builder will then build the design instantiating and configuring the MSS(MDDR)/FDDR blocks.
6. Build CoreABC based standalone initialization logic required to initialize the DDR blocks (MDDR/FDDR).
7. Interface the initialization logic with the System Builder block (which has MDDR/FDDR), and continue with the design flow.

System Builder Device Features Page

In the Device Features page, specify which DDR controllers (MDDR and/or FDDR) are used in your design (Figure 3).

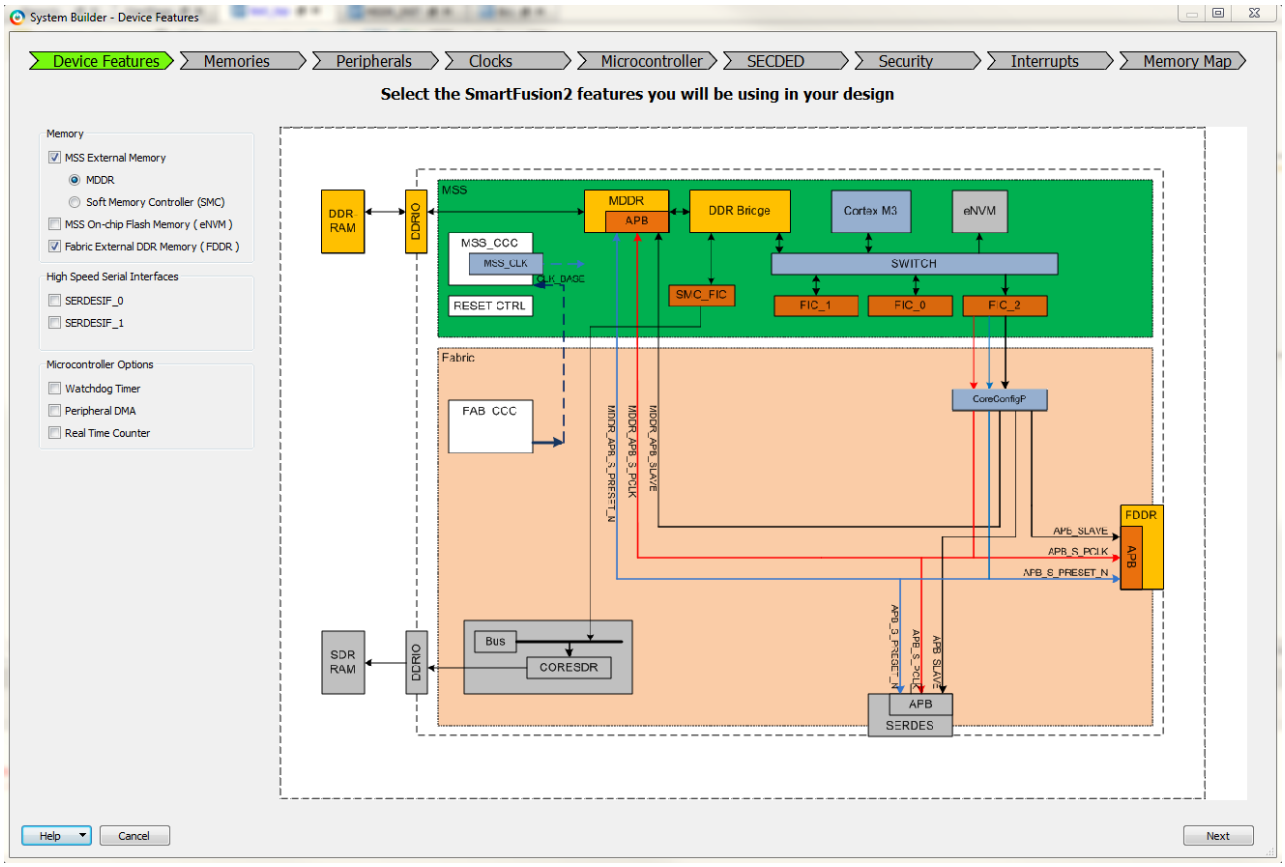


Figure 3 • System Builder Device Features Page

System Builder Memory Page

To use the MSS DDR (MDDR) or Fabric DDR (FDDR), select the Memory Type from the dropdown list (Figure 4).

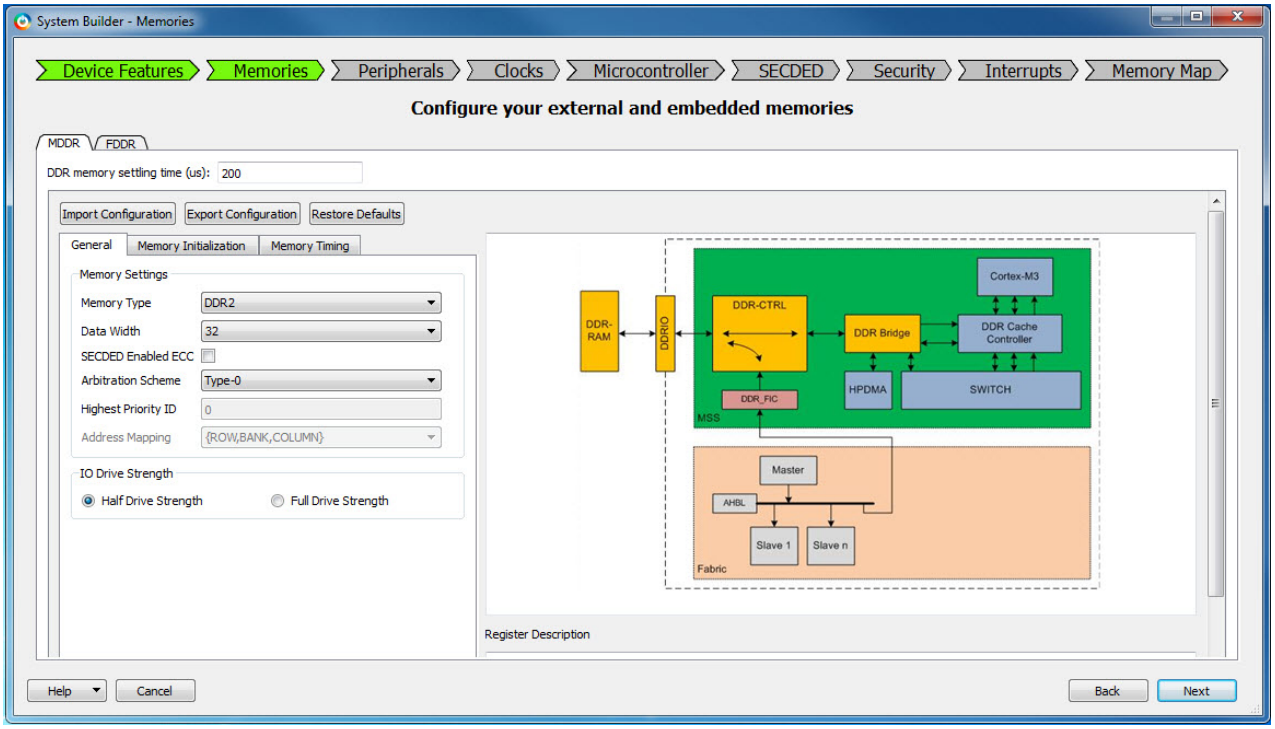


Figure 4 • MSS External Memory

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, consult the DDR Interfaces User's Guide.

This data is used to generate the CoreABC program files corresponding to the DDR registers being configured. For complete details on DDR controller configuration registers please 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 5. The register names used in this file are the same as the ones described in the 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 5 • 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.

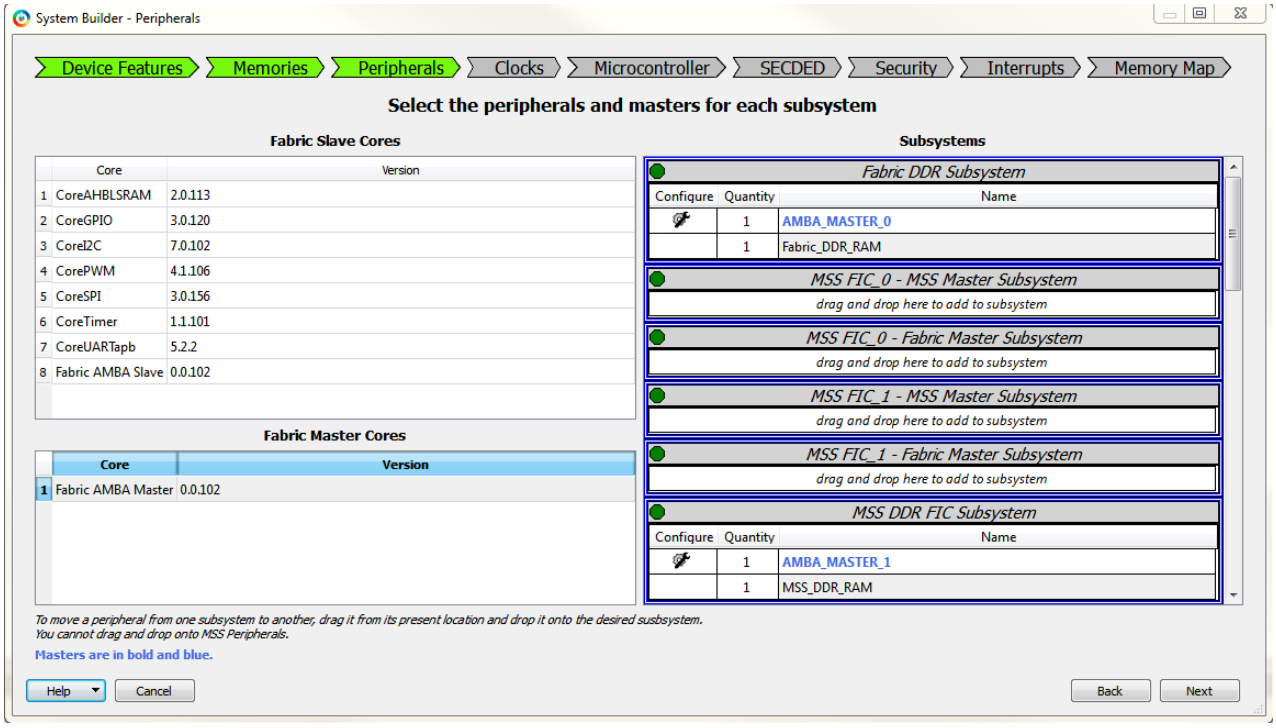


Figure 6• System Builder Peripherals Page

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. Note that this clock is defined as a multiple of the M3_CLK (Cortex-M3 and MSS Main Clock, Figure 7). 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 a ratio of the MDDR_CLK and it should match the frequency at which the FPGA fabric sub-system that accesses the MDDR is running.



Figure 7 • 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 (Figure 7). 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 8 • Fabric DDR Clocks

Chip Oscillators Tab – Clocks Page

In the Chip Oscillators tab of the System Builder Clocks page, check the 'On-chip 25/50 MHz RC Oscillator' and the 'Drives Fabric Logic' checkboxes as shown in the Figure 9 below. This exposes an output pin RCOSC_25_50MHZ_O2F under the CHIP_OSC_PINS group on the System Builder block which can be used to drive the RCOSC_25_50MHZ input pin of the CoreResetP soft IP cores used in the peripheral initialization. This helps in reusing the oscillator block that's already instantiated inside the System Builder block to drive the CoreResetP cores being used for the peripherals, that sit outside the System Builder block. This is necessary if you are using System Builder because there's only 1 RCOSC per device.

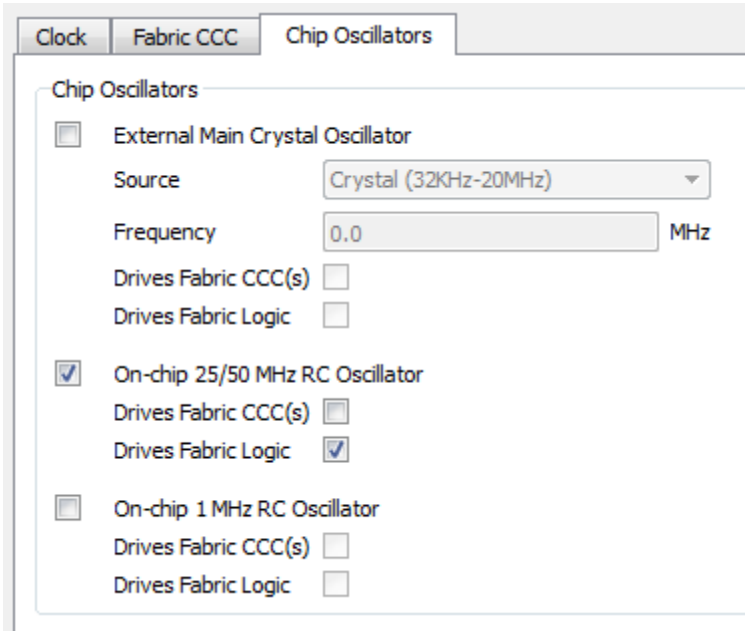


Figure 9 • Chip Oscillators tab of System Builder

Generating your System Builder design

Once you are done configuring all the System Builder pages with your desired settings, click 'Finish' in the last page. The System Builder component is generated to a SmartDesign, with all required top level pins and BIF ports exposed on the System Builder block under appropriate pin groups. Next you need to build the

initialization logic for the DDR blocks (MDDR/FDDR) used in your design, interface it to the System Builder block to initialize the DDR blocks, generate and then continue with the design flow.

Upon generating the System Builder component, separate text files containing the CoreABC program corresponding to MDDR and FDDR register configuration are created to the disk under the <project_location>/.../*_MSS/ directory and the <project_location>/.../FABDDR_0/ directories respectively with the names MDDR_init_abc.txt and FDDR_init_abc.txt. This CoreABC program generated for MDDR/FDDR has to be loaded/copied to the CoreABC instance used for the initialization of the peripheral (MDDR/FDDR). This will be discussed again in the following sections.

Building Standalone Initialization Logic for MDDR

In order to initialize the MDDR, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the CoreABC program to the DDR configuration registers, manages the reset sequences required for the MDDR block to be operational and signals when the MDDR block is ready to communicate with the rest of your design. To create the initialization subsystem you must:

- Instantiate and configure CoreABC soft IP core.
- Load CoreABC with the initialization program generated to the MDDR_init_abc.txt file.
- Instantiate and configure the CoreConfigP and the CoreResetP cores
- Connect these components to the peripheral's (MDDR) configuration interfaces, clocks, resets and PLL lock ports

CoreABC configuration

1. Create a new SmartDesign component (MDDR_INIT).
2. Instantiate CoreABC into your SmartDesign. This core can be found in the Libero Catalog (under Processors).
3. Double-click the core to open the configurator.
4. Configure the core as shown in the depiction below (Figure 10).

   Configure the data bus width to be 16.

   Configure the maximum number of instructions to at least 256.

   Configure to use AND and OR operations as optional instructions.

   Configure Instruction Store to Hard (FPGA Tiles).
5. Copy the CoreABC program generated for MDDR from the MDDR_init_abc.txt file created under the <project_location>/…/*_MSS/ directory, to the CoreABC Program tab. See the figure below.

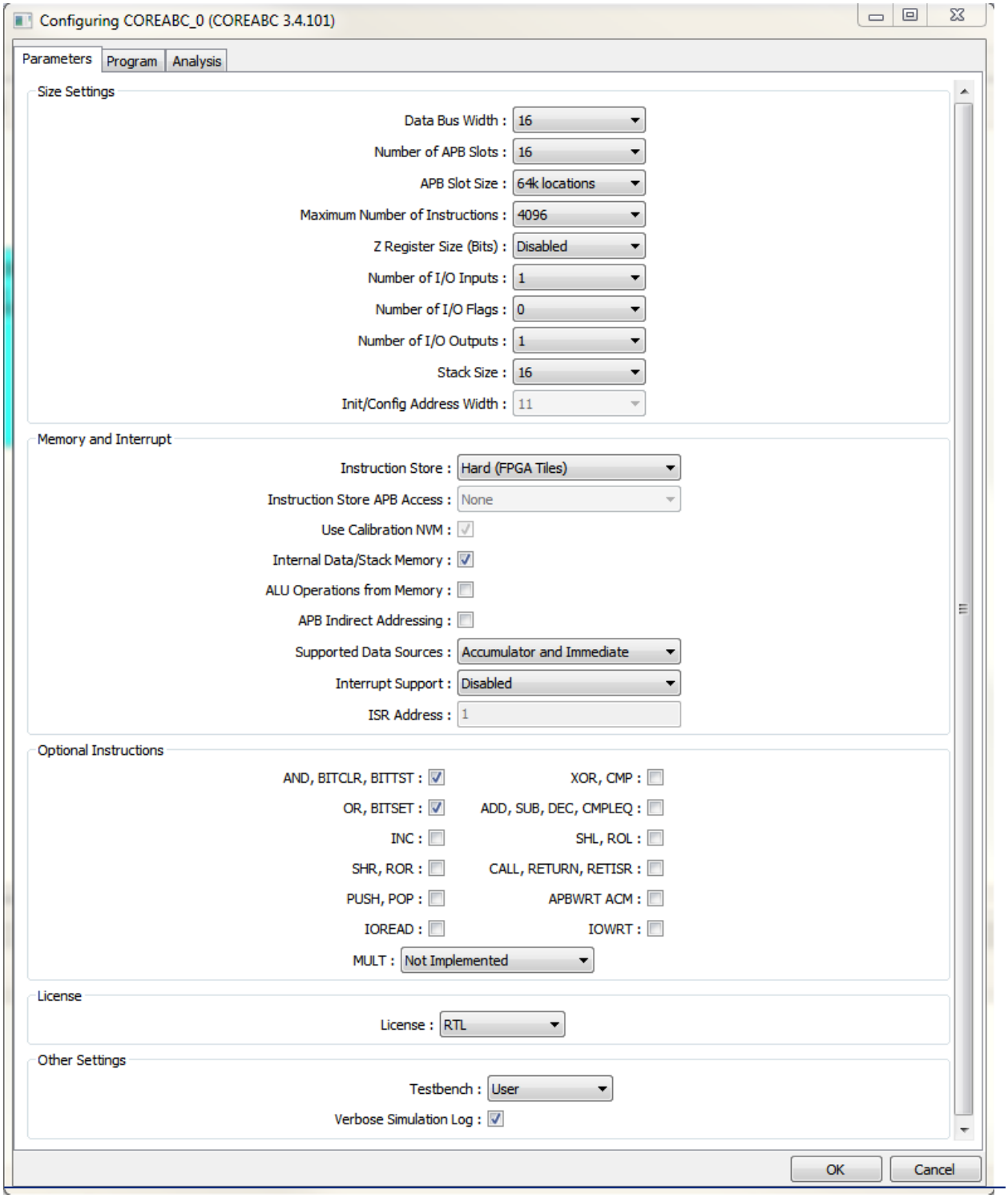


Figure 10. *CoreABC configuration*

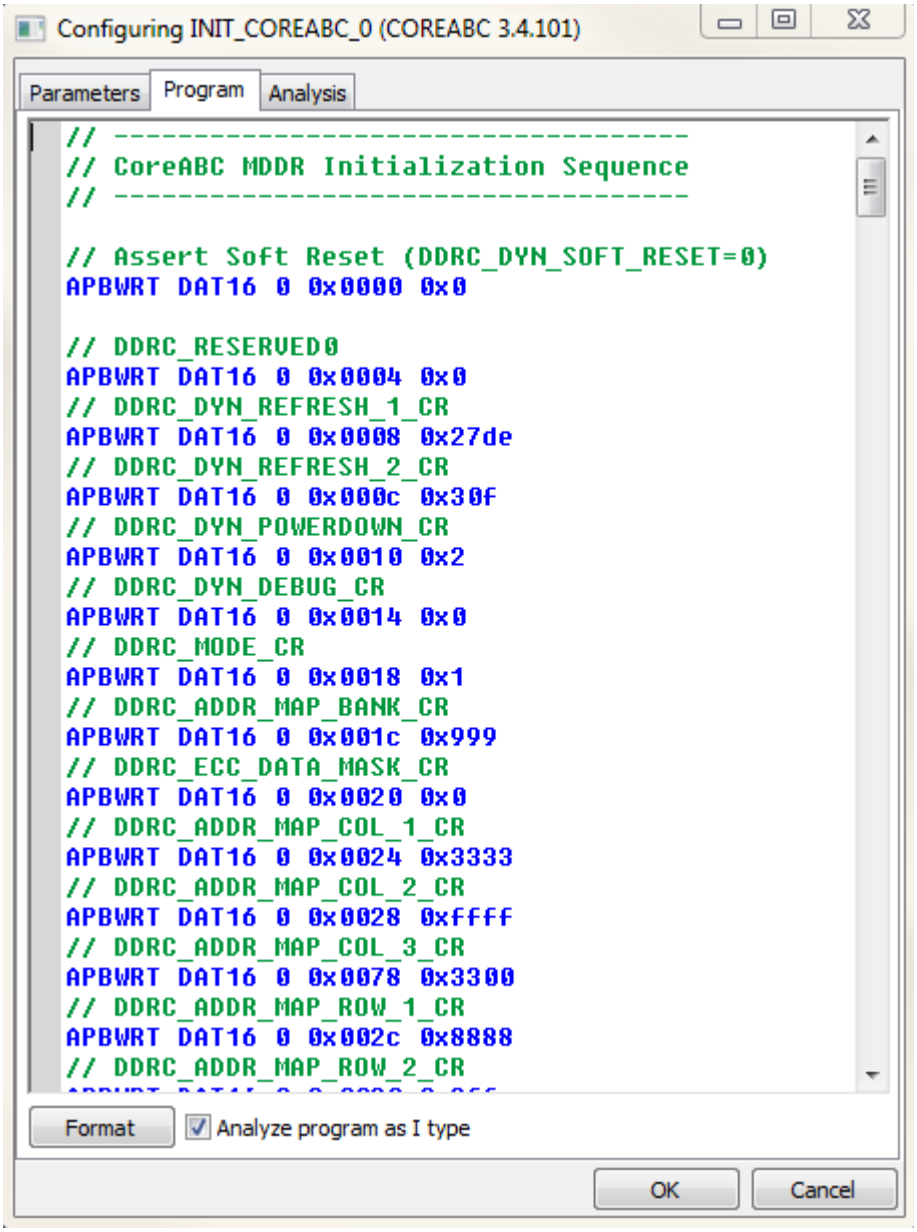


Figure X. *CoreABC program for MDDR*

CoreConfigP

1. Instantiate CoreConfigP into the same SmartDesign. 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 11)

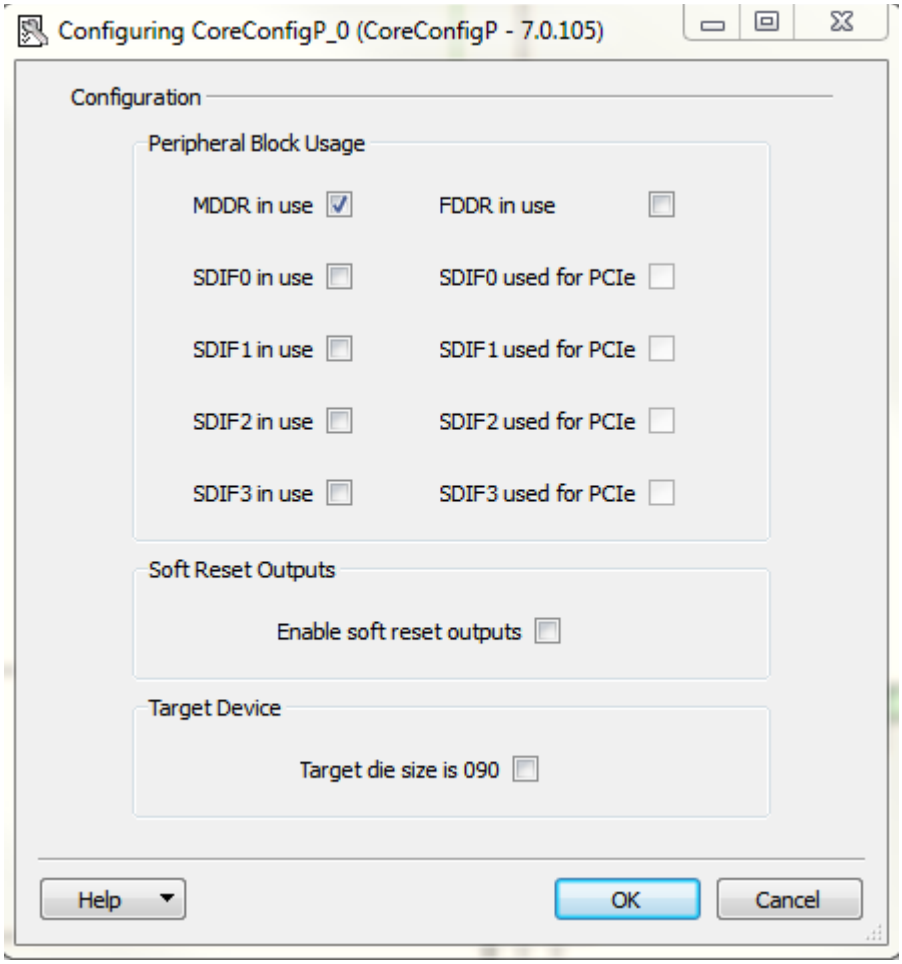


Figure 11 • CoreConfigP Dialog Box

CoreResetP

1. Instantiate CoreResetP into the same SmartDesign. 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 12– CoreResetP Configurator)
3. Configure the core to:
   - Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
     - EXT_RESET_OUT is never asserted
     - EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
     - EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
     - 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.
   - Check the appropriate checkboxes to indicate which peripherals you are using in your design..
   - Specify the external DDR memory setting time. 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. Incorrect value for the settling time may result in

simulation errors.

Refer to the CoreResetP handbook for details on the options available to you in this configurator

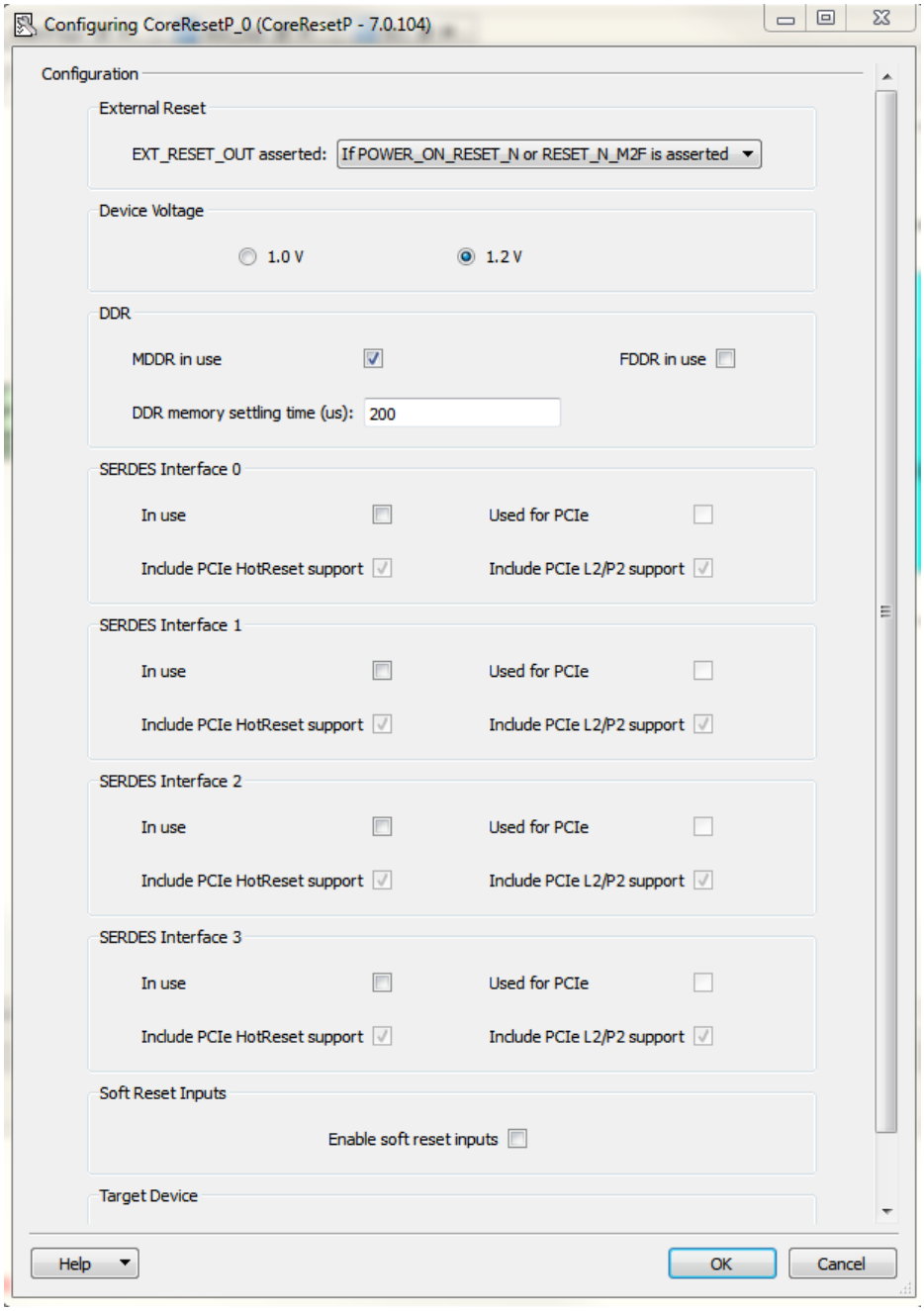


Figure 12. CoreResetP Configurator

Overall Connectivity of the initialization logic (MDDR_INIT)

After you have instantiated and configured the 3 cores CoreABC, CoreConfigP and CoreResetP, appropriate connections have to be made to make the initialization logic operational. See the depiction below in the Figure 13 to understand how the connections are made.

The following is a list of signals that need to be promoted to the top which will be needed when interfacing this initialization logic with the actual peripheral (MDDR).

- CoreConfigP:

    - APB_S_PRESET_N
    - APB_S_PCLK
    - APB_S_INIT (APB BIF MDDR_APBmslave)
- CoreResetP:
    - RCOSC_25_50MHZ
    - FAB_RESET_N
    - POWER_ON_RESET_N
    - DDR_READY
    - MDDR_DDR_AXI_S_CORE_RESET_N
- INIT_PCLK_25MHz connecting together the PCLK of CoreABC, the FIC_2_APB_M_PCLK of CoreConfigP and the CLK_BASE of CoreResetP).

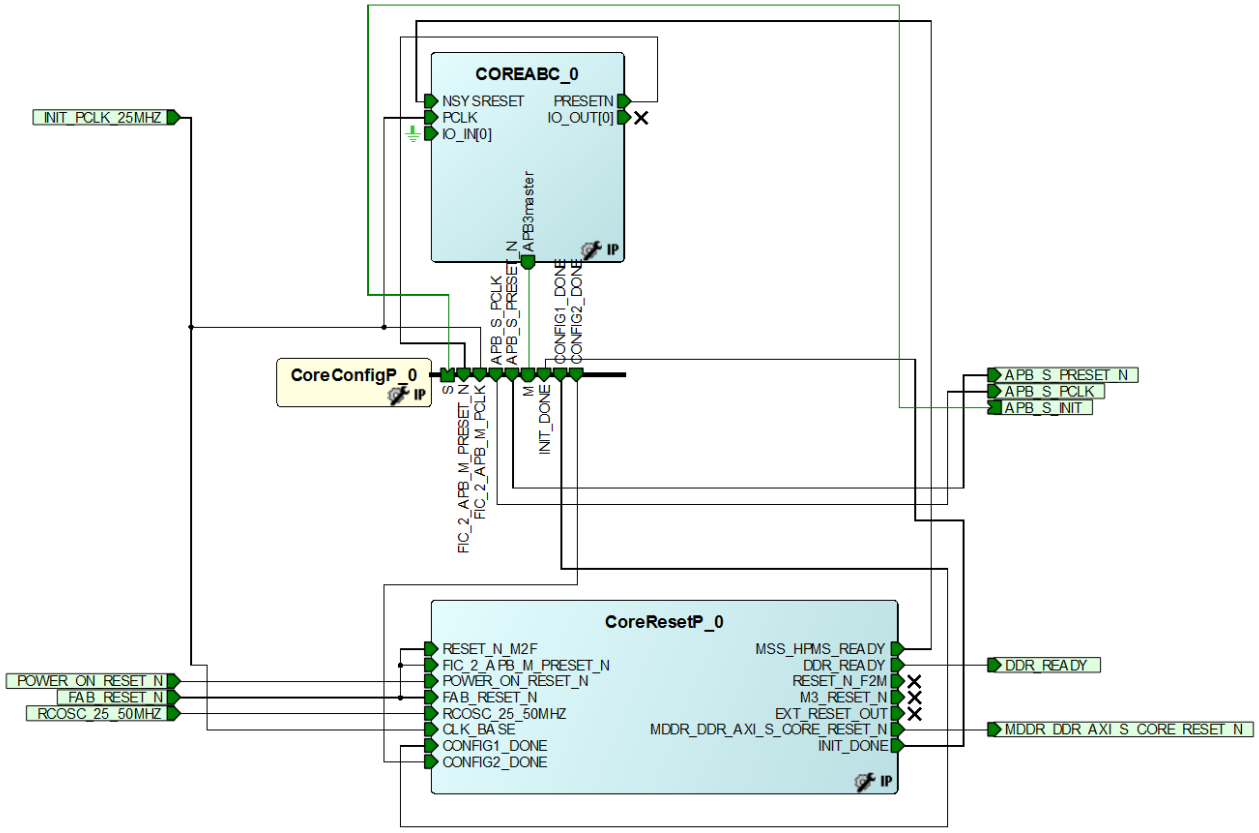


Figure 13. MDDR_INIT (MDDR Initialization Logic)

Interfacing MDDR with the Initialization Logic Built for it

In the same SmartDesign the System Builder block is present, instantiate the Smart Design containing the MDDR initialization logic (MDDR_INIT), and do necessary interconnections to interface the System Builder block (containing the MDDR) to the initialization logic. See the depiction below in the Figure 14 to understand how the connections are made.

The following is a list of signal interconnections that need to be made to properly interface the System Builder block (MDDR) to the initialization logic.

FROM Port or Bus Interface (BIF)/ Component	TO Port/Bus Interface (BIF)/Component
MDDR_APB_S_PCLK/ System Builder Block	APB_S_PCLK/ initialization logic.
MDDR_APB_S_PRESET_N/ System Builder Block	APB_S_PRESET_N/ initialization logic.
MDDR_APB_SLAVE BIF/ System Builder Block	APB_S_INIT/ initialization logic
POWER_ON_RESET_N/ System Builder Block	POWER_ON_RESET_N/ initialization logic
FAB_RESET_N / System Builder Block	FAB_RESET_N / initialization logic
RCOSC_25_50MHZ/ System Builder Block	RCOSC_25_50MHZ/ initialization logic
MDDR_CORE_RESET_N / System Builder Block	MDDR_DDR_AXI_S_CORE_RESET_N / initialization logic

Apart from the above connections, do the following also:

- Promote the FAB_RESET_N pin the initialization logic (MDDR_INIT_0 instance) to the top level (this is the warm reset).
- Promote the MSS_DDR_FIC_SUBSYSTEM_PINS to the top to drive the fabric logic that belongs to the MSS_DDR_FIC_SUBSYSTEM.
- Promote the DDR_READY of the initialization logic to the top to monitor the status of the MDDR initialization.
- Drive the INIT_PCLK_25MHz input pin of the initialization logic with 25MHz clock. You can use the unused GLx in the System Builder block from the ‘Fabric CCC’ tab of the ‘Clocks’ page to drive any clock in the fabric logic.

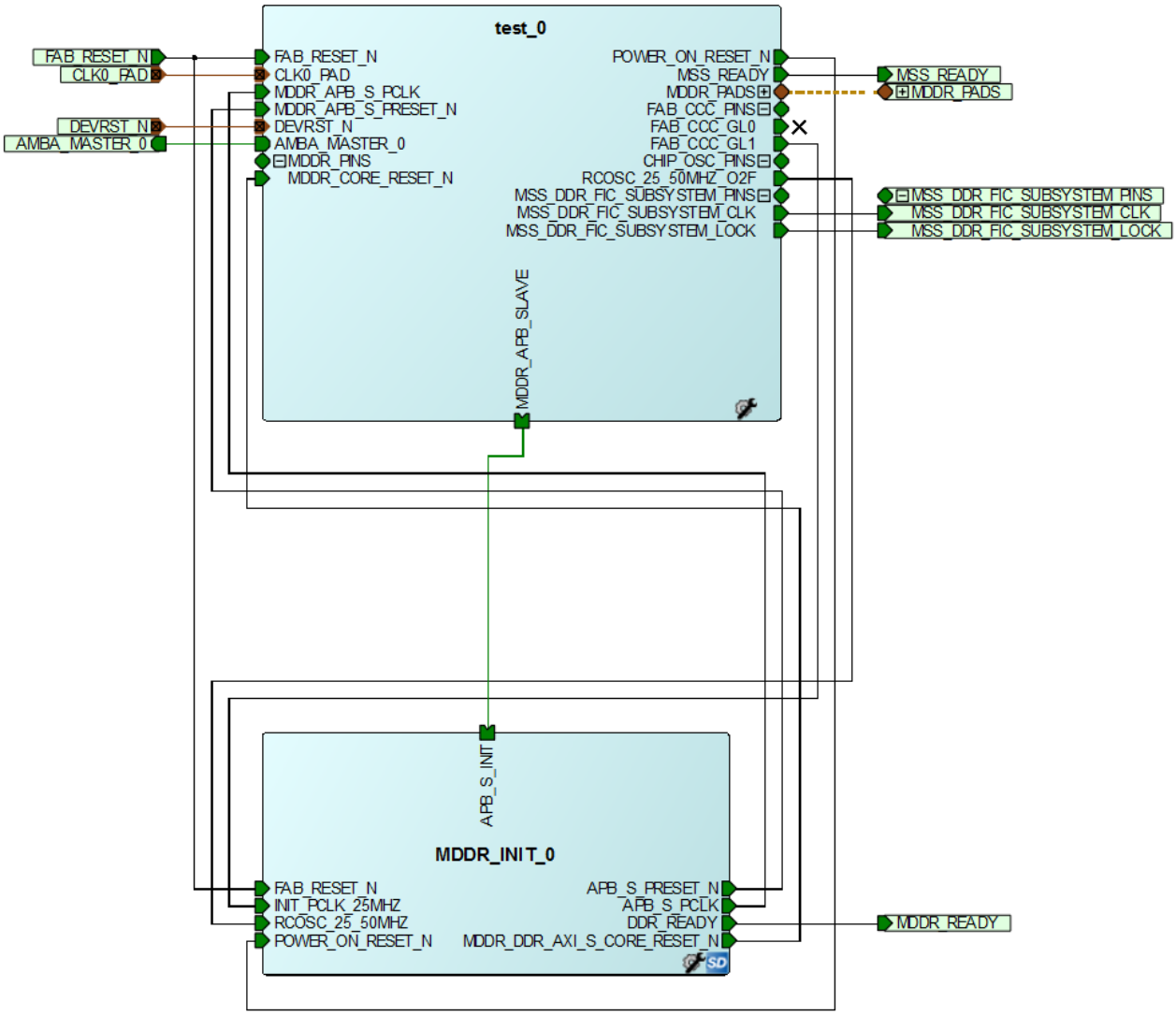


Figure 14. Interfacing System Builder (MDDR) with the Initialization Logic

Building Standalone Initialization Logic for FDDR

In order to initialize the FDDR, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the CoreABC program to the DDR configuration registers, manages the reset sequences required for the FDDR block to be operational and signals when the FDDR block is ready to communicate with the rest of your design. To create the initialization subsystem you must:

- Instantiate and configure CoreABC soft IP core.
- Load CoreABC with the initialization program generated to the FDDR_init_abc.txt file.
- Instantiate and configure the CoreConfigP and CoreResetP cores
- Connect these components to the peripheral's (FDDR) configuration interfaces, clocks, resets and PLL lock ports

CoreABC configuration

1. Create a new SmartDesign component (FDDR_INIT).
2. Instantiate CoreABC into your SmartDesign. This core can be found in the Libero

Catalog (under Processors).

3. Double-click the core to open the configurator.
4. Configure the core as shown in the depiction below (Figure 15).
    Configure the data bus width to be 16.
    Configure the maximum number of instructions to at least 256.
    Configure to use AND and OR operations as optional instructions.
    Configure Instruction Store to Hard (FPGA Tiles).

5. Copy the CoreABC program generated for FDDR from the FDDR_init_abc.txt file created under the <project_location>/.../FABDDR_0/ directory, to the CoreABC Program tab. See the figure below.

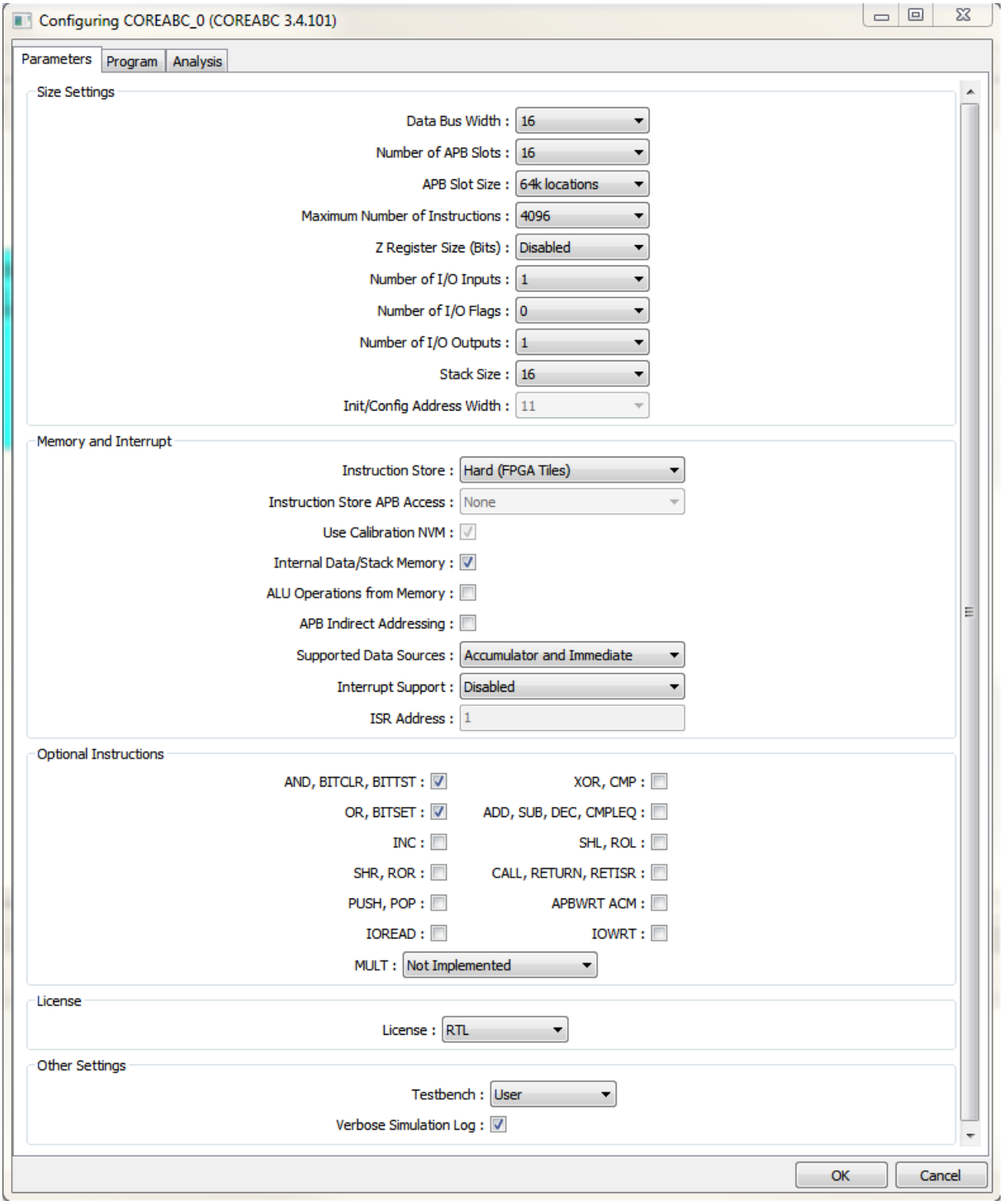


Figure 15. *CoreABC configuration*

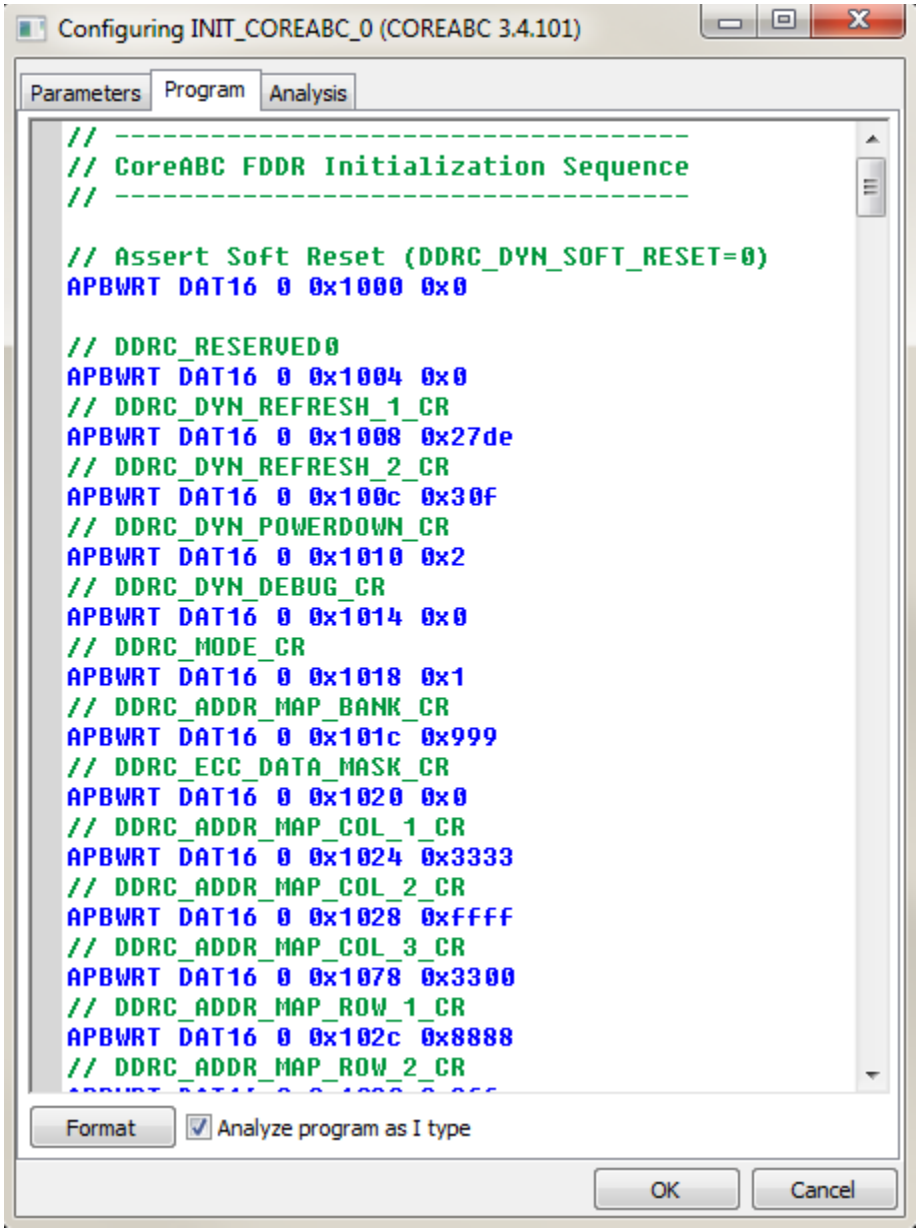


Figure X. *CoreABC program for FDDR*

CoreConfigP

1. Instantiate CoreConfigP into the same SmartDesign. 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 16)

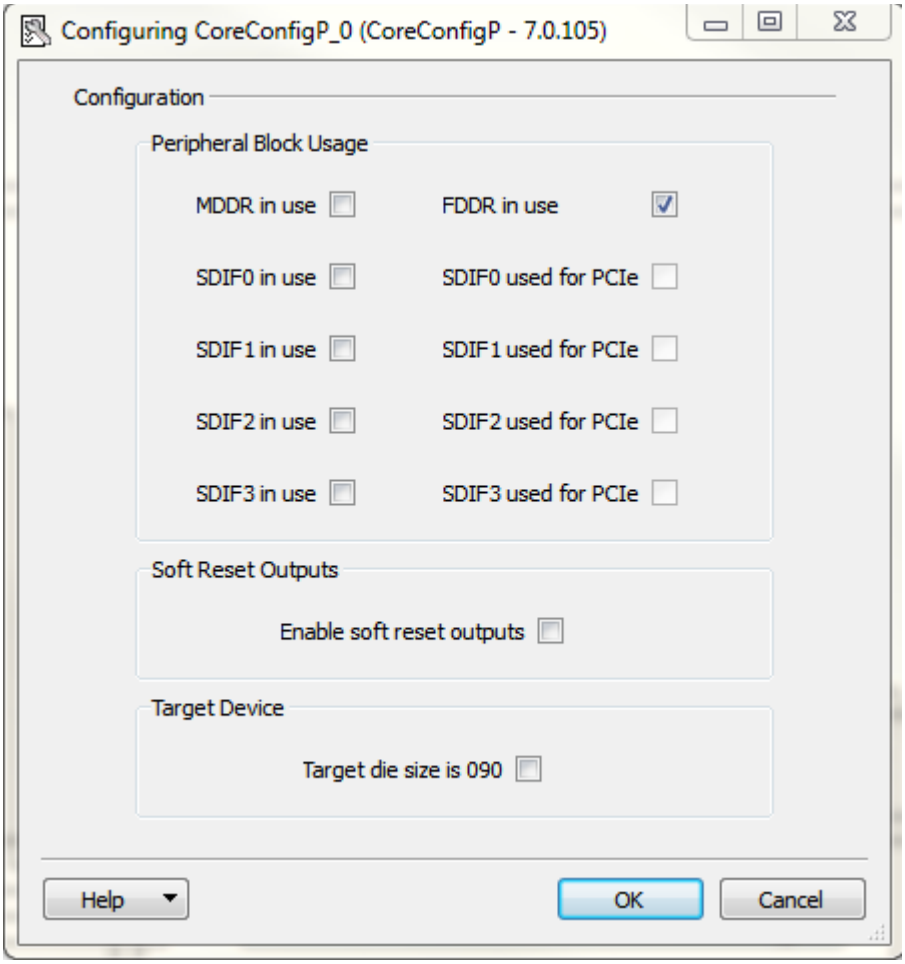


Figure 16 • CoreConfigP Dialog Box

CoreResetP

1. Instantiate CoreResetP into the same SmartDesign. 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 17 – CoreResetP Configurator)
3. Configure the core to:
   - Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
     - EXT_RESET_OUT is never asserted
     - EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
     - EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
     - 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.
   - Check the appropriate checkboxes to indicate which peripherals you are using in your design.
   - Specify the external DDR memory setting time. 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. Incorrect value for the settling time may result in

simulation errors.

Refer to the CoreResetP handbook for details on the options available to you in this configurator

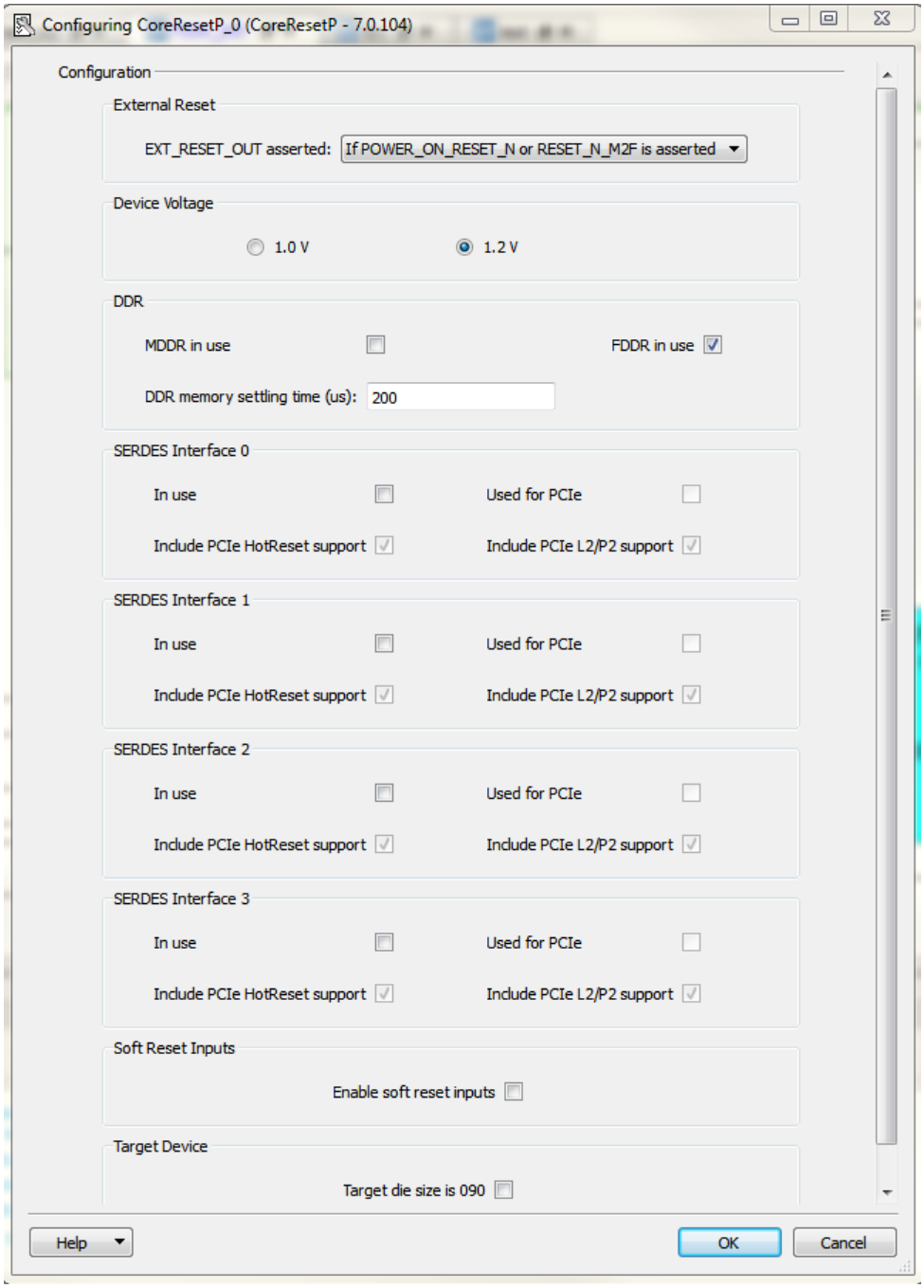


Figure 17. CoreResetP Configurator

Overall Connectivity of the initialization logic (FDDR_INIT)

After you have instantiated and configured the 3 cores CoreABC, CoreConfigP and CoreResetP, appropriate connections have to be made to make the initialization logic operational. See the depiction below in the Figure 18 to understand how the connections are made.

The following is a list of signals that need to be promoted to the top which will be needed when interfacing this initialization logic with the actual peripheral (FDDR).

- CoreConfigP:

    - APB_S_PRESET_N
    - APB_S_PCLK
    - APB_S_INIT (APB BIF FDDR_APBmslave)
- CoreResetP:
    - RCOSC_25_50MHZ
    - FAB_RESET_N
    - POWER_ON_RESET_N
    - DDR_READY
    - FDDR_CORE_RESET_N
    - FPLL_LOCK
- INIT_PCLK_25MHz (connecting together the PCLK of CoreABC, the FIC_2_APB_M_PCLK of CoreConfigP and the CLK_BASE of CoreResetP).

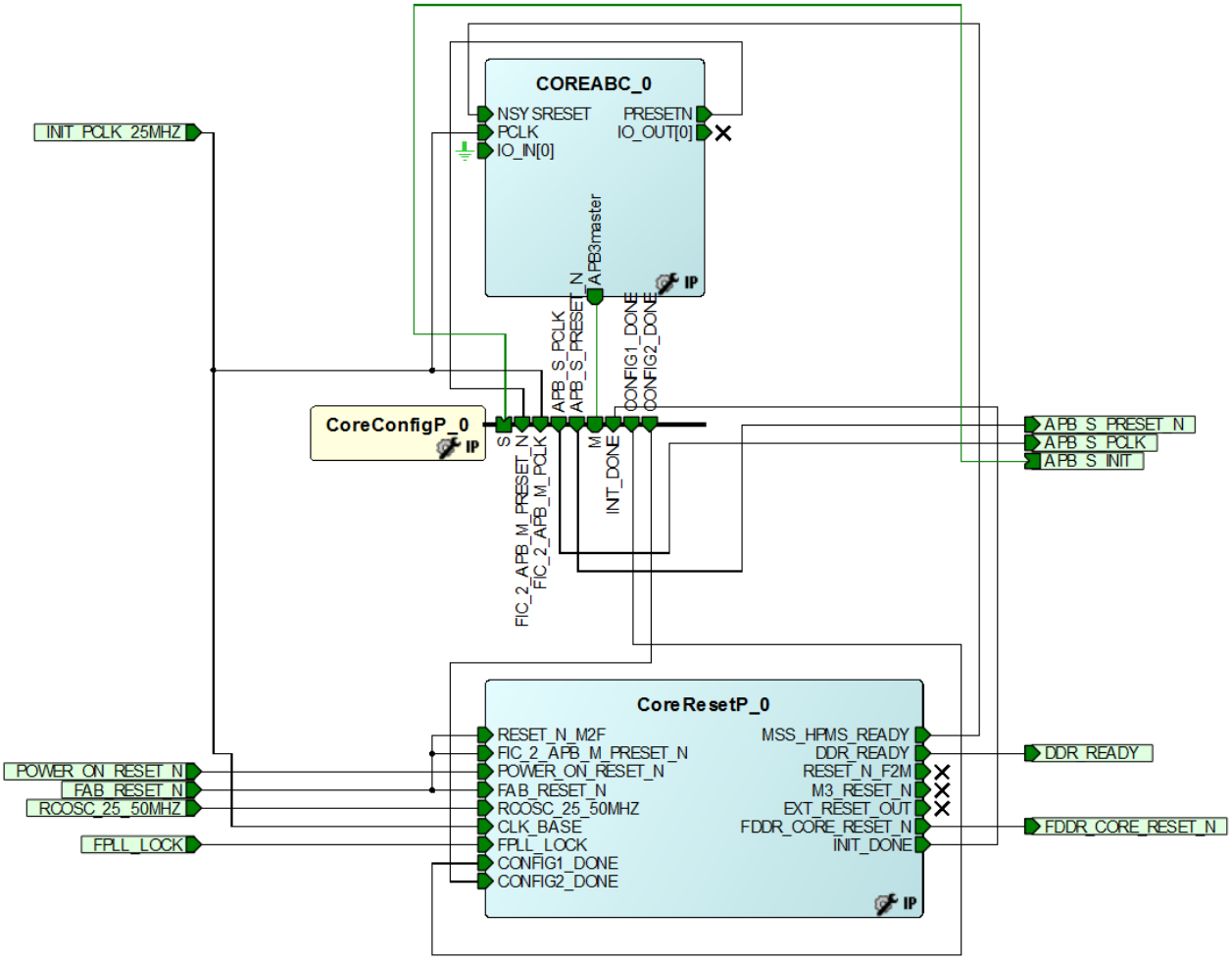


Figure 18. FDDR_INIT (FDDR Initialization Logic)

Interfacing FDDR with the Initialization Logic Built for it

In the same SmartDesign the System Builder block is present, instantiate the Smart Design containing the FDDR initialization logic (FDDR_INIT), and do necessary interconnections to interface the System Builder block containing the FDDR, to the initialization logic. See the depiction below in the Figure 19 to understand how the connections are made.

The following is a list of signal interconnections that need to be made to properly interface the FDDR to the initialization logic.

FROM Port or Bus Interface (BIF)/ Component	TO Port/Bus Interface (BIF)/ Component
FDDR_APB_S_PCLK/ System Builder Block	APB_S_PCLK/ initialization logic
FDDR_APB_S_PRESET_N/ System Builder Block	APB_S_PRESET_N/ initialization logic
FDDR_APB_SLAVE BIF/ System Builder Block	APB_S_INIT/ initialization logic
POWER_ON_RESET_N/ System Builder Block	POWER_ON_RESET_N/ initialization logic
FAB_RESET_N / System Builder Block	FAB_RESET_N / initialization logic
RCOSC_25_50MHZ/ System Builder Block	RCOSC_25_50MHZ/ initialization logic
FDDR_CORE_RESET_N / System Builder Block	FDDR_CORE_RESET_N / initialization logic
FDDR_FPLL_LOCK/ System Builder Block	FPLL_LOCK/ initialization logic

Apart from the above connections, do the following also:

- Promote the FAB_RESET_N pin the initialization logic (FDDR_INIT_0 instance) to the top level (this is the warm reset).
- Promote the DDR_READY of the initialization logic to the top to monitor the status of the FDDR initialization.
- Promote the FDDR_SUSBSYSTEM_RESET_N pin to the top or drive it appropriately from the fabric logic.
- Instantiate a FABCCC block and do the following connections:
    - Drive the INIT_PCLK_25MHZ input pin of the initialization logic with the GLx of FABCCC block configured to 25MHz frequency.
    - Drive the FDDR_SUBSYSTEM_CLK input pin under the FDDR_SUBSYSTEM_PINS group of the System Builder block with the GLx of FABCCC block (configured to appropriate frequency).
    - Drive the FDDR_SUBSYSTEM_LOCK input pin under the FDDR_SUBSYSTEM_PINS group of the System Builder block with the LOCK pin of the FABCCC block.

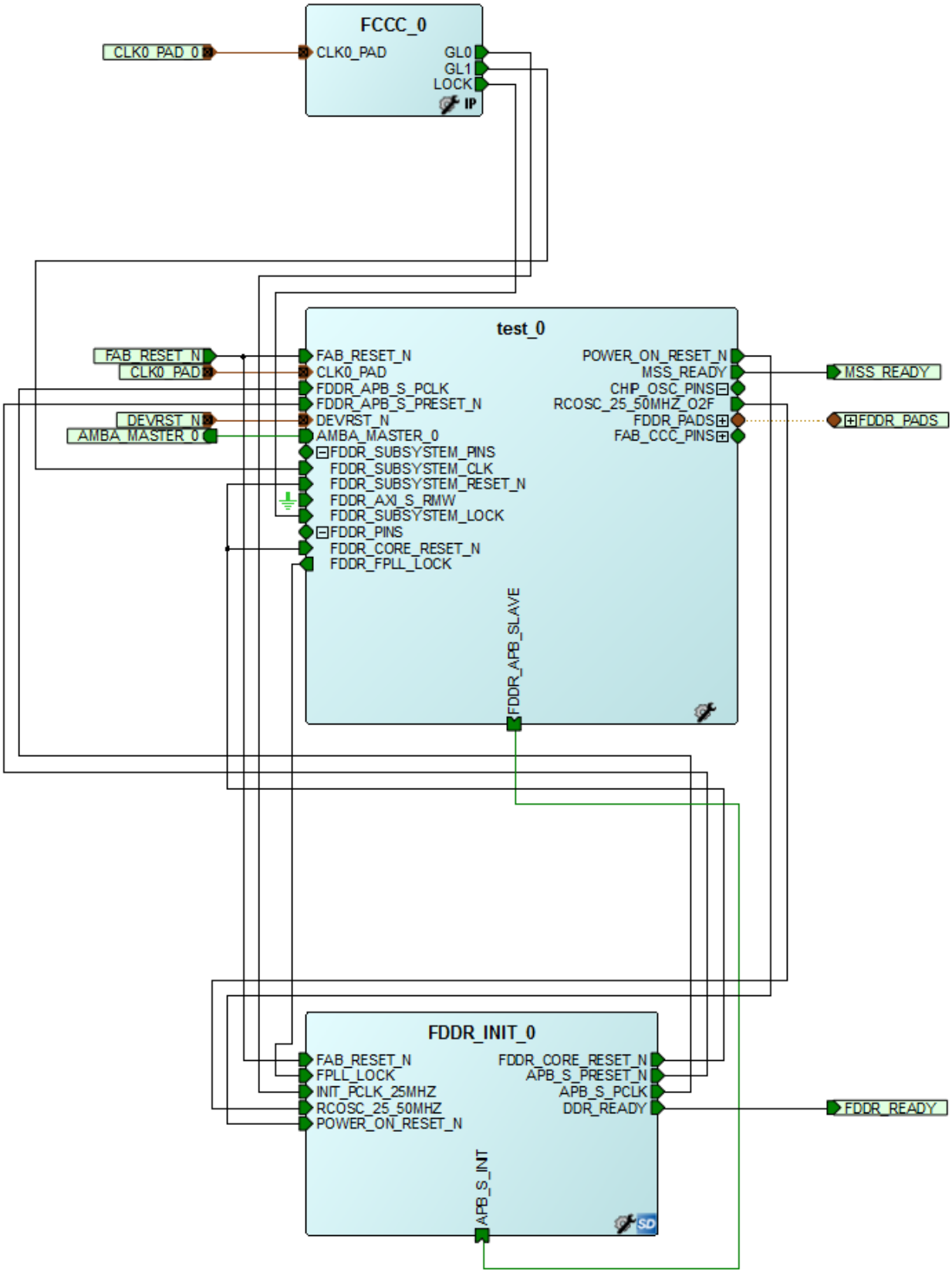


Figure 19. Interfacing System Builder (FDDR) with the Initialization Logic

Continuing with the Design Flow

Next step is to integrate any user logic that you might have with the System Builder block and the initialization logic. Once you have done that, you can generate your top level SmartDesign. This will generate all files that are necessary to implement and simulate your design. You can then proceed with the rest of the Design Flow.

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

In this section we describe 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. In this section we describe 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.

Design using SERDESIF_n (n=0/1/2/3)

Building Standalone Initialization Logic for SERDESIF_n

In order to initialize the SERDESIF_n registers, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the CoreABC program to the SERDESIF_n configuration registers, manages the reset sequences required for the SERDESIF_n block to be operational and signals when the SERDESIF_n block is ready to communicate with the rest of your design. To create the initialization subsystem you must:

- Instantiate and configure CoreABC soft IP core.
- Load CoreABC with the initialization program generated to the SERDESIF_n_init_abc.txt file.
- Instantiate and configure the CoreConfigP and CoreResetP cores.
- Instantiate and configure the on-chip 50MHz RC oscillator.
- Instantiate the System Reset (SYSRESET) macro.
- Connect these components to the peripheral's (SERDESIF_n) configuration interface, clocks, resets and PLL lock ports.

CoreABC configuration

1. Create a new SmartDesign component (SERDESIF_n_INIT).
2. Instantiate CoreABC into your SmartDesign. This core can be found in the Libero Catalog (under Processors).
3. Double-click the core to open the configurator.
4. Configure the core as shown in the depiction below (Figure 20).

   Configure the data bus width to be 32 (as 32-bit data needs to be written to some of the SERDES registers).

   Configure the maximum number of instructions to at least 256.

   Configure to use AND and OR operations as optional instructions.

   Configure Instruction Store to Hard (FPGA Tiles).
5. Copy the CoreABC program generated for SERDESIF_n from the SERDESIF_n_init_abc.txt file created under the <project_location>/../SERDES_IF_n/ directory, to the CoreABC Program tab. See the figure below.

NOTE: The SERDESIF_n_init_abc.txt file will be generated only after you generate the Smart Design containing the SERDESIF_n block. So after you've made all the connections and generated all the blocks (including SERDESIF_n), you will need to copy the contents of the SERDESIF_n_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC. You may defer doing this until you completely configure your SERDESIF_n block and generate the

SmartDesign component containing it.

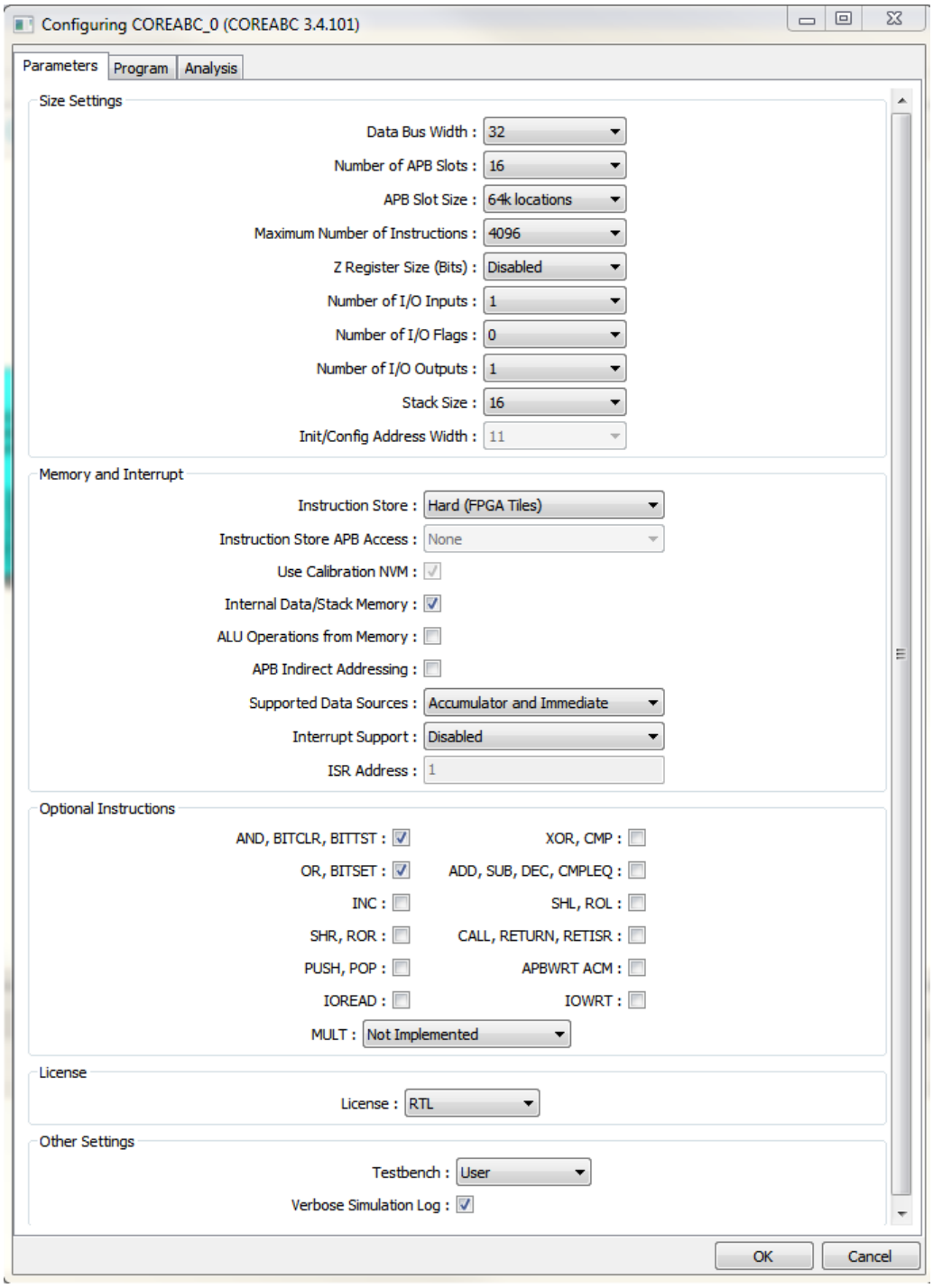


Figure 20. *CoreABC configuration*

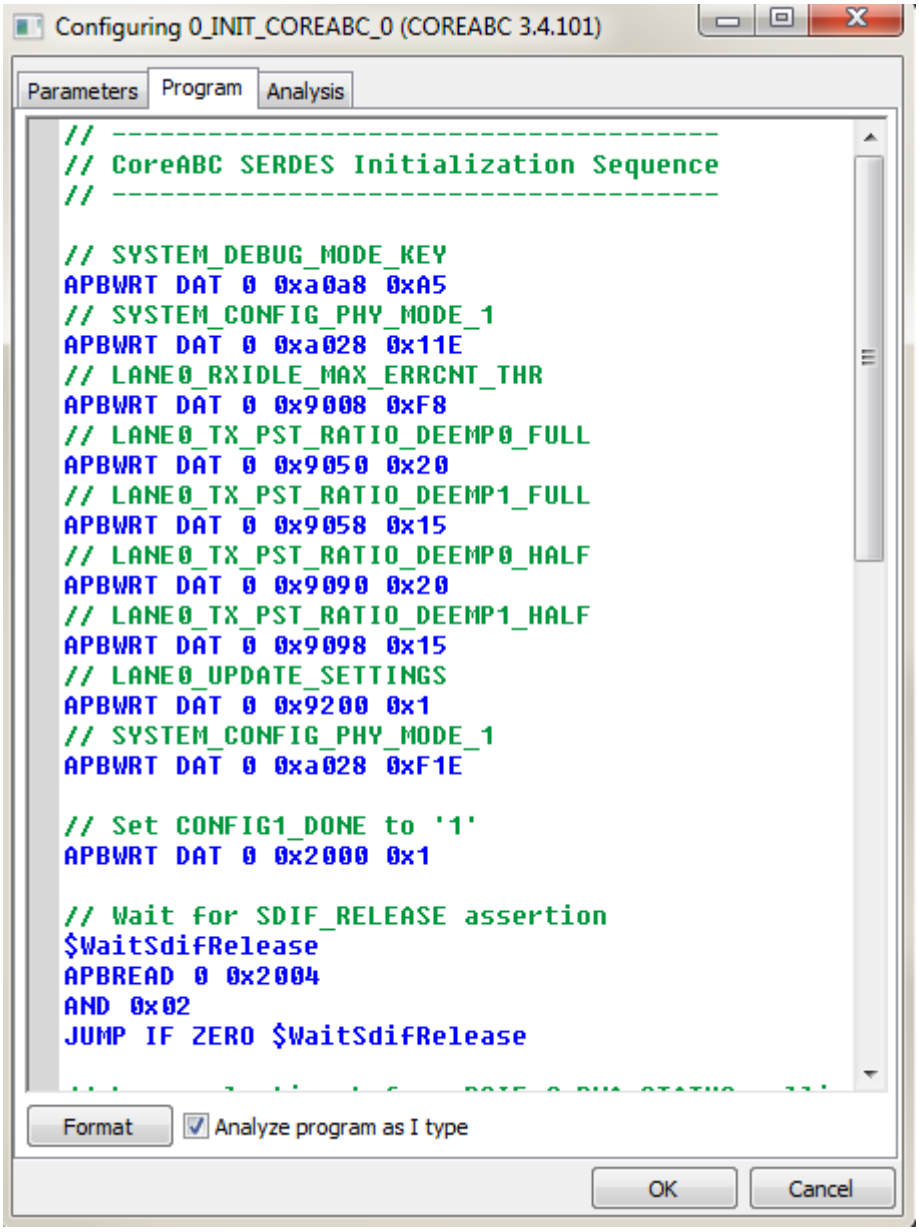


Figure X. *CoreABC program for SERDESIF_n*

CoreConfigP

1. Instantiate CoreConfigP into the same SmartDesign. This core can be found in the Libero Catalog (under Peripherals).
2. Double-click the core to open the configurator.
3. Check the appropriate checkboxes as shown in the figure below (Figure 21).

NOTE: Irrespective of the SERDESIF_n location (n=0/1/2/3) you want to configure using this initialization logic, check only the “SDIF0 in use” and “SDIF0 used for PCIe” checkboxes. For example, even if you are building this initialization logic for say, the SERDESIF_1/2/3 location, you need to check the “SDIF0 in use” and “SDIF0 used for PCIe” checkboxes in the CoreConfigP instance. Do not check the checkboxes corresponding to the SERDESIF_1/2/3 locations in the CoreConfigP instance. This is a requirement for the Libero generated CoreABC code to work for all

the SERDESIF_n locations (n = 0/1/2/3). And this rule is applicable only when you are configuring the CoreConfigP and CoreResetP instances as a part of building the initialization logic; you will have to specify the exact SERDESIF_n location in the SERDESIF configurator later when you instantiate and configure the SERDESIF block.

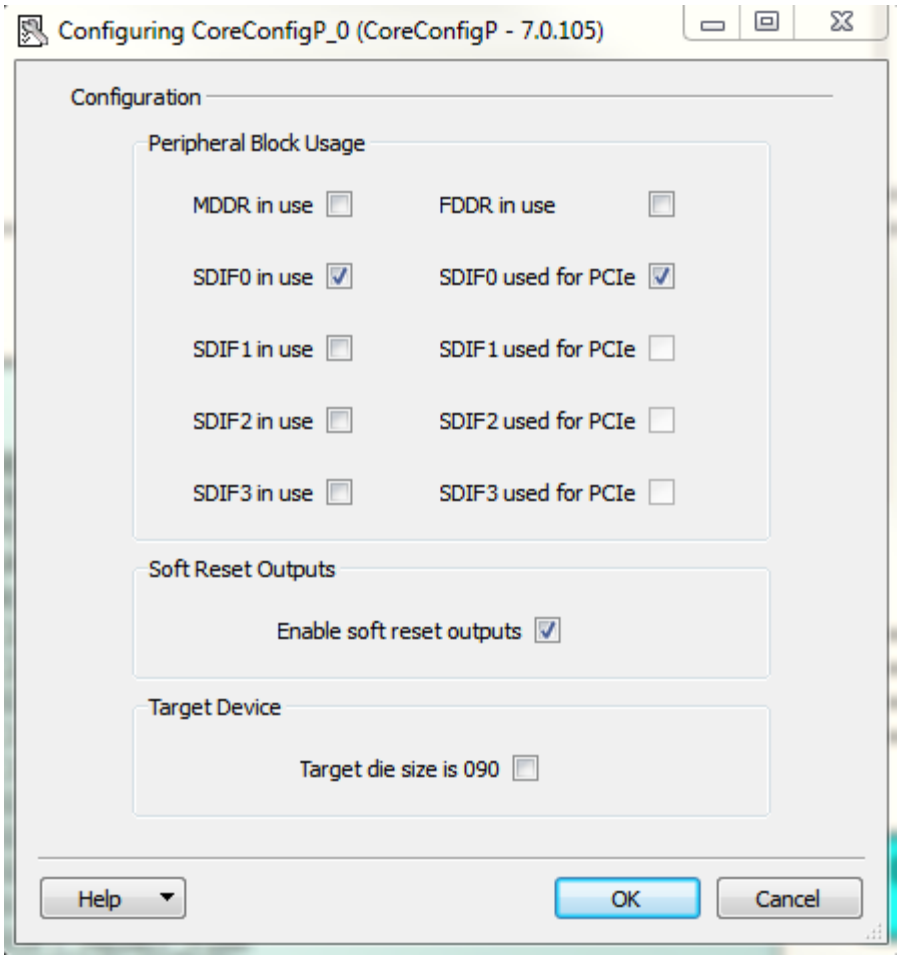


Figure 21• CoreConfigP Dialog Box

CoreResetP

1 . Instantiate CoreResetP into the same SmartDesign. 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 22 – CoreResetP Configurator)

3. Configure the core to:

– Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
  - EXT_RESET_OUT is never asserted
  - EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
  - EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
  - 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.

– Check the appropriate checkboxes as shown in the figure below. Refer to the CoreResetP

handbook for details on the options available to you in this configurator.

NOTE: Irrespective of the SERDESIF_n location (n=0/1/2/3) you want to configure using this initialization logic, check only the checkboxes under the “SERDES Interface 0”. For example, even if you are building this initialization logic for say, the SERDESIF_1/2/3 location, you need to check the checkboxes under the “SERDES Interface 0” in the CoreResetP instance. Do not check the checkboxes corresponding to the SERDESIF_1/2/3 locations. This is a requirement for the Libero generated CoreABC code to work for all the SERDESIF_n locations (n = 0/1/2/3). And this rule is applicable only when you are configuring the CoreConfigP and CoreResetP instances as a part of building the initialization logic; you will have to specify the exact SERDESIF_n location in the SERDESIF configurator later when you instantiate and configure the SERDESIF block.

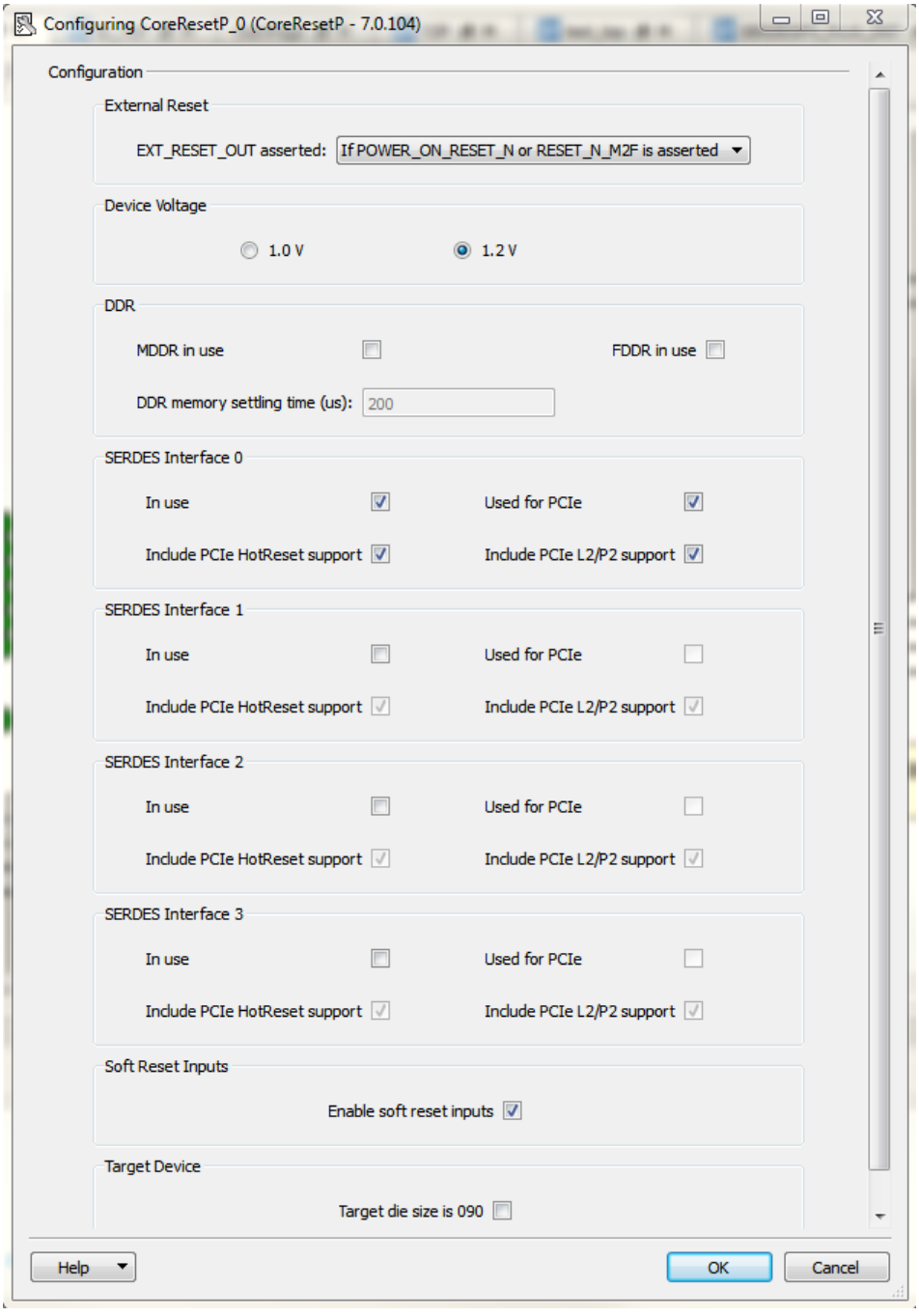


Figure 22. CoreResetP Configurator

Overall Connectivity of the initialization logic (SERDESIF_n_INIT)

After you have instantiated and configured the 3 cores CoreABC, CoreConfigP and CoreResetP, appropriate connections have to be made to make the initialization logic operational. See the depiction below in the Figure 23 to understand how the connections are made.

The following is a list of signals that need to be promoted to the top which will be needed when interfacing this initialization logic with the actual peripheral (SERDESIF_n).

- CoreConfigP:
  - APB_S_PRESET_N
  - APB_S_PCLK
  - APB_S_INIT (APB BIF SDIF0_APBmslave)
- CoreResetP:
  - RCOSC_25_50MHZ
  - CLK_LTSSM (promoted as CLK_LTSSM_125MHZ)
  - FAB_RESET_N
  - POWER_ON_RESET_N
  - SDIF_READY
  - SDIFn_PHY_RESET_N
  - SDIFn_CORE_RESET_N
  - SDIFn_SPLL_LOCK
  - SDIFn_PERST_N
- INIT_PCLK_25MHz (connecting together the PCLK of CoreABC, the FIC_2_APB_M_PCLK of CoreConfigP and the CLK_BASE of CoreResetP).

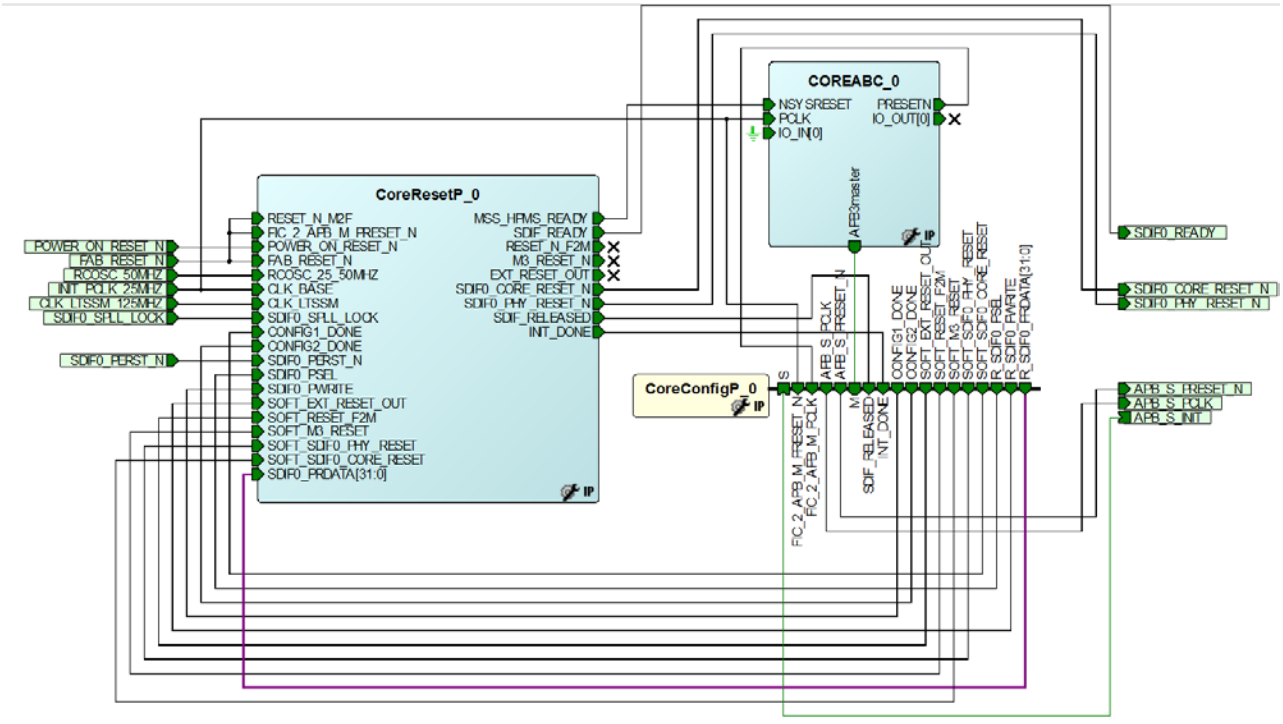


Figure 23. SERDESIF_n_INIT (SERDESIF_n Initialization Logic)

Instantiating and Configuring the SERDESIF_n block

Create a Smart Design component and instantiate a SERDESIF block from the catalog window (from under the Peripherals). Double-click the SERDESIF block in the SmartDesign canvas to open the configurator to select a location (n= 0/1/2/3) and configure the SERDES. (Figure 24 -High Speed Serial Interface Configurator). 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 24 SERDES Configurator) to enter your PCIe or EPCS parameters and the register values will automatically be computed for you. Refer to the SERDES Configurator User's Guide for details.

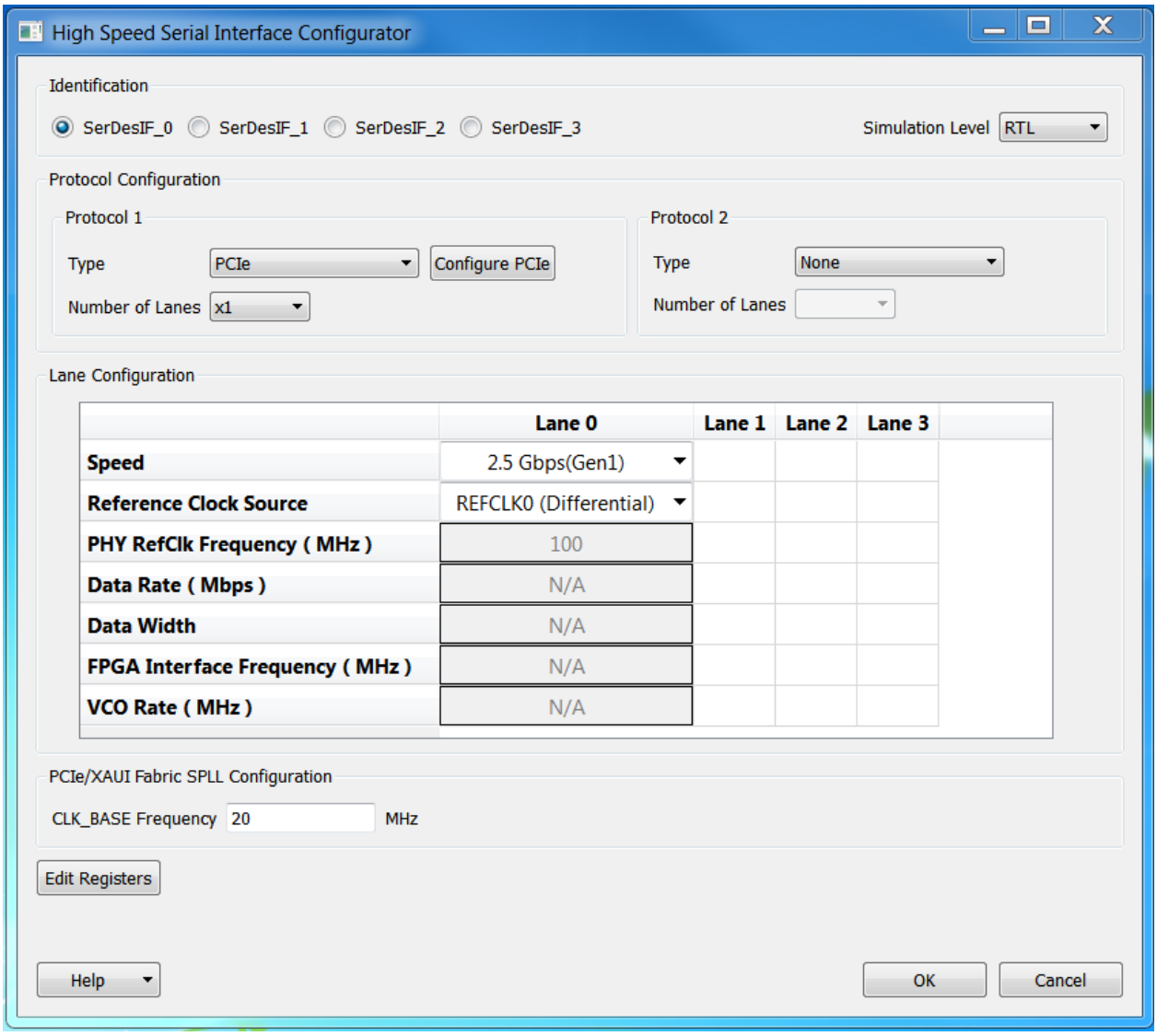


Figure 24. High Speed Serial Interface Configurator

Interfacing SERDESIF_n with the Initialization Logic Built for it

In the same SmartDesign the SERDESIF_n block is present, instantiate the Smart Design containing the SERDESIF_n initialization logic (SERDESIF_n_INIT), and do necessary interconnections to interface the SERDESIF_n block to the initialization logic. See the depiction below in the Figure 27 to understand how the connections are made.

The following is a list of signal interconnections that need to be made to properly interface the SERDESIF_n to the initialization logic.

FROM Port or Bus Interface (BIF)/ Component	TO Port/Bus Interface (BIF)/ Component
APB_S_PCLK/ SERDESIF_n	APB_S_PCLK/ initialization logic
APB_S_PRESET_N/ SERDESIF_n	APB_S_PRESET_N/ initialization logic
APB_SLAVE BIF/ SERDESIF_n	APB_S_INIT/ initialization logic
PHY_RESET_N/ SERDESIF_n	SDIFn_PHY_RESET_N/ initialization logic
CORE_RESET_N/ SERDESIF_n	SDIFn_CORE_RESET_N/ initialization logic
SPLL_LOCK/ SERDESIF_n	SDIFn_SPLL_LOCK/ initialization logic

50MHz Oscillator Instantiation

CoreResetP needs to be clocked by the on-chip 50MHz RC oscillator. You must instantiate a 50MHz Oscillator for this purpose.

1. Instantiate the Chip Oscillators core into the same SmartDesign the SERDESIF_n block is present. This core can be found in the Libero Catalog under Clock & Management.
2. Configure this core such that the oscillator drives the FPGA fabric, as shown in Figure 25.
3. Click "**OK**".
4. Connect the RCOSC_25_50MHz_O2F output of the Oscillator to the RCOSC_25_50MHz input of SERDESIF_n_INIT block (SERDESIF_n initialization logic).

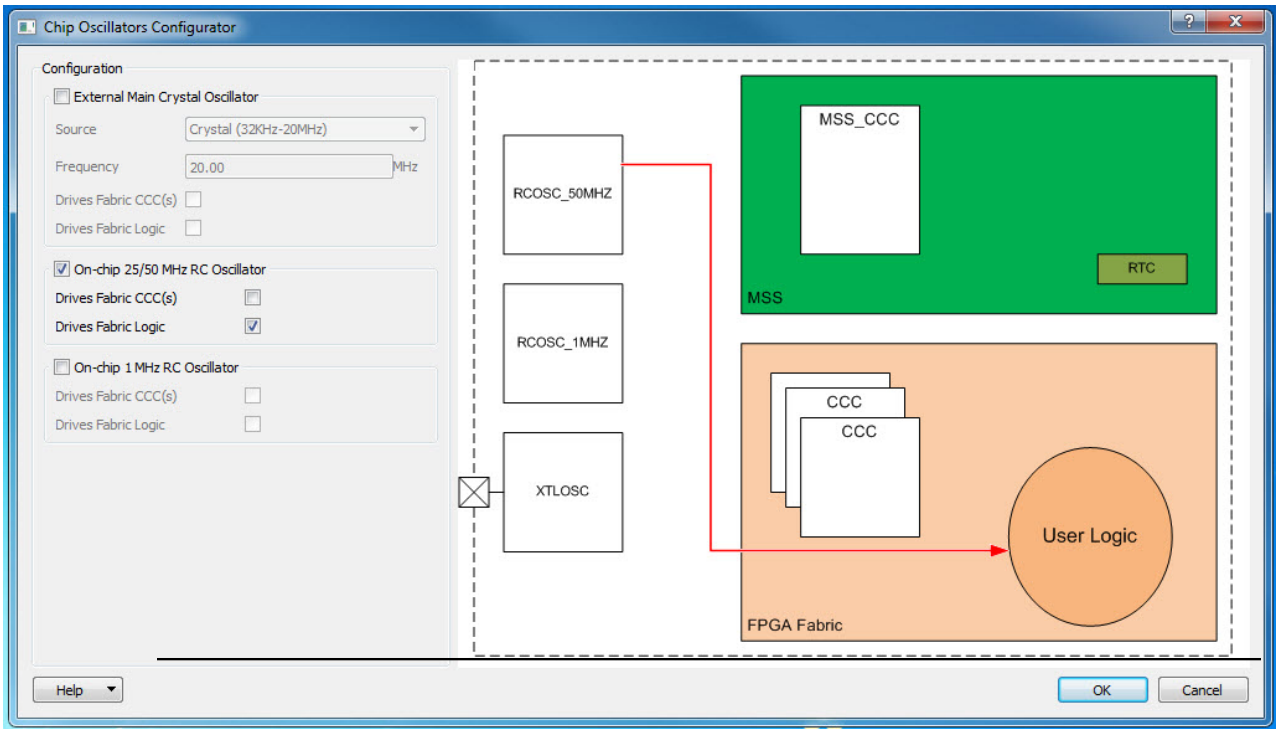


Figure 25 • Chip Oscillators Configurator

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 26). Instantiate the SYSRESET macro into the same SmartDesign the SERDESIF_n block is present. This macro can be found in the Libero Catalog under Macro Library. No configuration of this macro is necessary. Drive the POWER_ON_RESET_N input of SERDESIF_n_INIT

block (SERDESIF_n initialization logic with the POWER_ON_RESET_N output signal of this SYSRESET macro.



Figure 26 • SYSRESET Macro

Apart from the above connections, do the following also:

- Promote the FAB_RESET_N pin of the initialization logic (SERDESIF_n_INIT_0 instance) to the top level (this is the warm reset).
- Promote the SDIFn_READY pin of the initialization logic to the top to monitor the status of the SERDESIF_n initialization.
- Promote the SDIFn_PERST_N pin of the initialization logic to the top or tie it high.
- Instantiate a FABCCC block and do the following connections:
    - Drive the INIT_PCLK_25MHZ input pin of the initialization logic with the GLx of FABCCC block configured to 25MHz frequency.
    - Drive the CLK_BASE input pin of the SERDESIF_n block with the GLx of FABCCC block (configured to appropriate frequency).
    - Drive the CLK_LTSSM_125MHZ input pin of the initialization logic with the GLx of FABCCC block configured to 125MHz frequency.

NOTE: If more than 1 SERDESIF blocks are used in a design then you should put together separate initialization logic (using the CoreABC, CoreConfigP, and CoreResetP) for each one of them and do appropriate connections with the SERDESIF_n blocks.

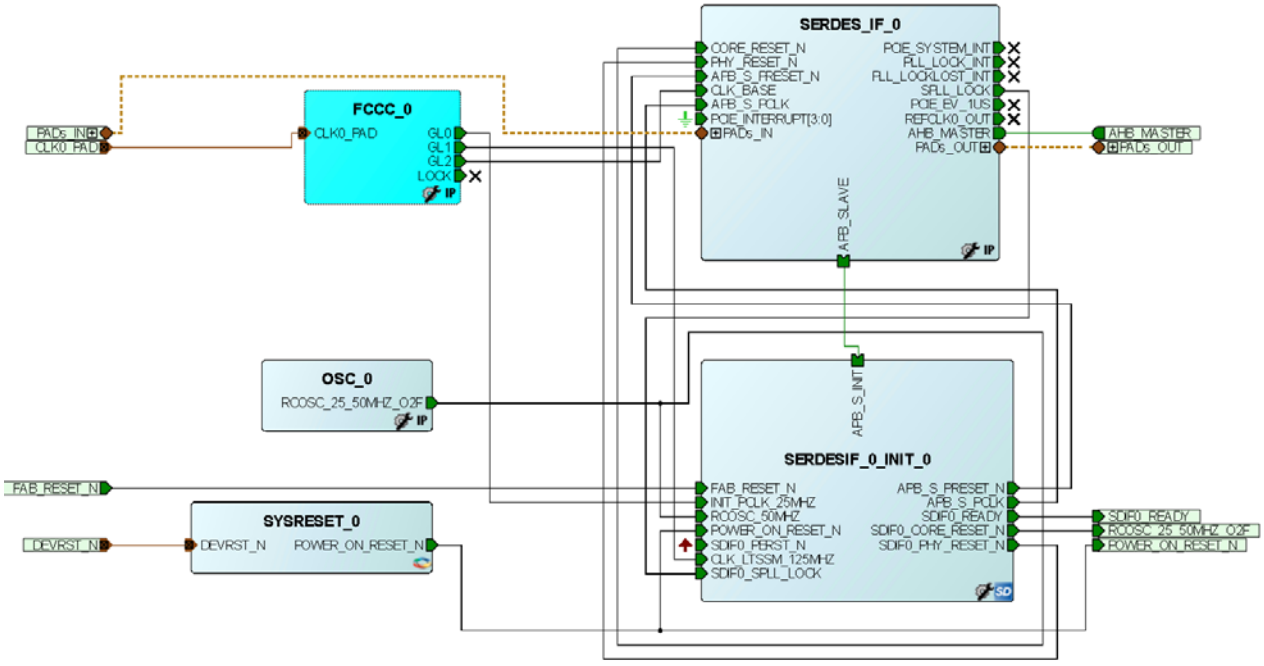


Figure 27. Interfacing SERDESIF_n with the Initialization Logic

Continuing with the Design Flow

Next step is to integrate any user logic that you might have with the SERDESIF_n block and the initialization logic. Once you have done that, you can generate your top level SmartDesign. This will generate all files that are necessary to implement and simulate your design. You can then proceed with the rest of the Design Flow.

NOTE: After configuring all the desired SERDESIF_n registers in the SERDESIF configurator and upon generating the Smart Design component containing the SERDESIF_n block , the SERDESIF_n_init_abc.txt file will be generated to the disk. You will need to copy the contents of the SERDESIF_n_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC.

Design using FDDR block

Building Standalone Initialization Logic for FDDR

In order to initialize the FDDR, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the CoreABC program to the DDR configuration registers, manages the reset sequences required for the FDDR block to be operational and signals when the FDDR block is ready to communicate with the rest of your design. To create the initialization subsystem you must:

- Instantiate and configure CoreABC soft IP core.
- Load CoreABC with the initialization program generated to the FDDR_init_abc.txt file.
- Instantiate and configure the CoreConfigP and CoreResetP cores.
- Instantiate and configure the on-chip 50MHz RC oscillator.
- Instantiate the System Reset (SYSRESET) macro.
- Connect these components to the peripheral's (FDDR) configuration interface, clocks, resets and PLL lock ports.

CoreABC configuration

1. Create a new SmartDesign component (FDDR_INIT).
2. Instantiate CoreABC into your SmartDesign. This core can be found in the Libero Catalog (under Processors).
3. Double-click the core to open the configurator.
4. Configure the core as shown in the depiction below (Figure 28).

   Configure the data bus width to be 16.

   Configure the maximum number of instructions to at least 256.

   Configure to use AND and OR operations as optional instructions.

   Configure Instruction Store to Hard (FPGA Tiles).
5. Copy the CoreABC program generated for FDDR from the FDDR_init_abc.txt file created under the <project_location>/../FABDDR_0/ directory, to the CoreABC Program tab. See the figure below.

NOTE: The FDDR_init_abc.txt file will be generated only when you generate the Smart Design containing the FDDR block. So after you've made all the connections and generated all the blocks (including FDDR), you will need to copy the contents of the FDDR_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC. You may defer doing this until you completely configure your FDDR block and generate the SmartDesign component containing it.

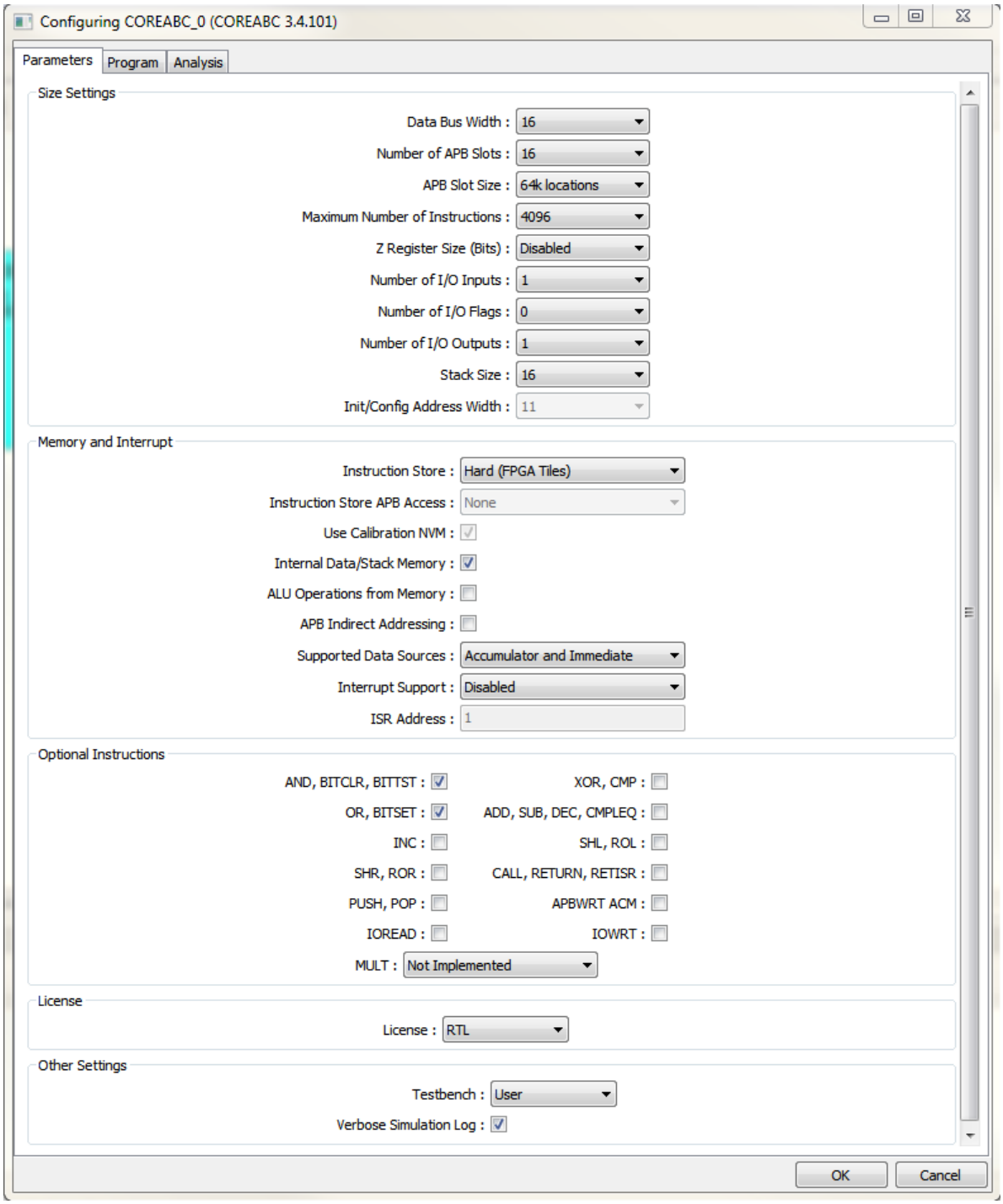


Figure 28. *CoreABC configuration*

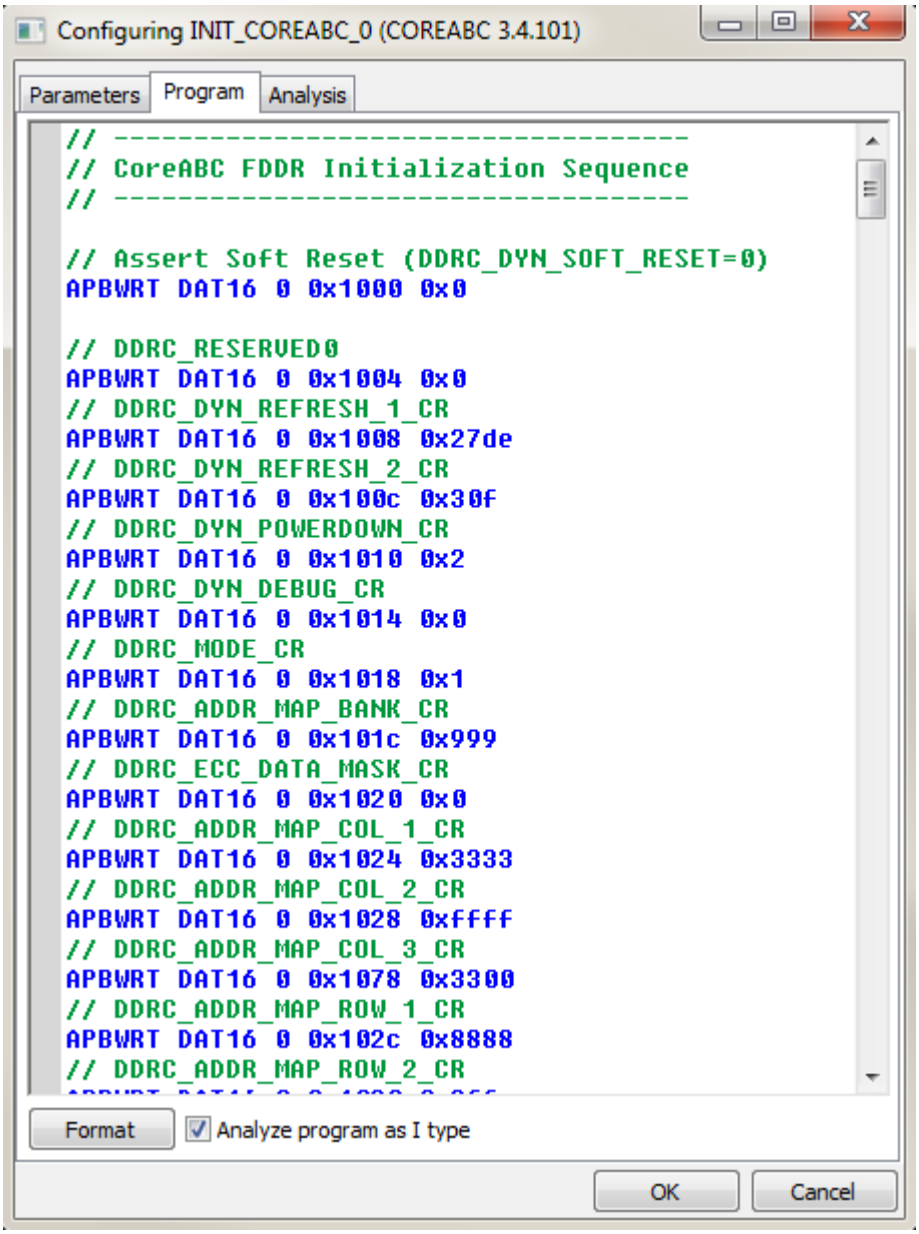


Figure X. *CoreABC program for FDDR*

CoreConfigP

1. Instantiate CoreConfigP into the same SmartDesign. 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 29)

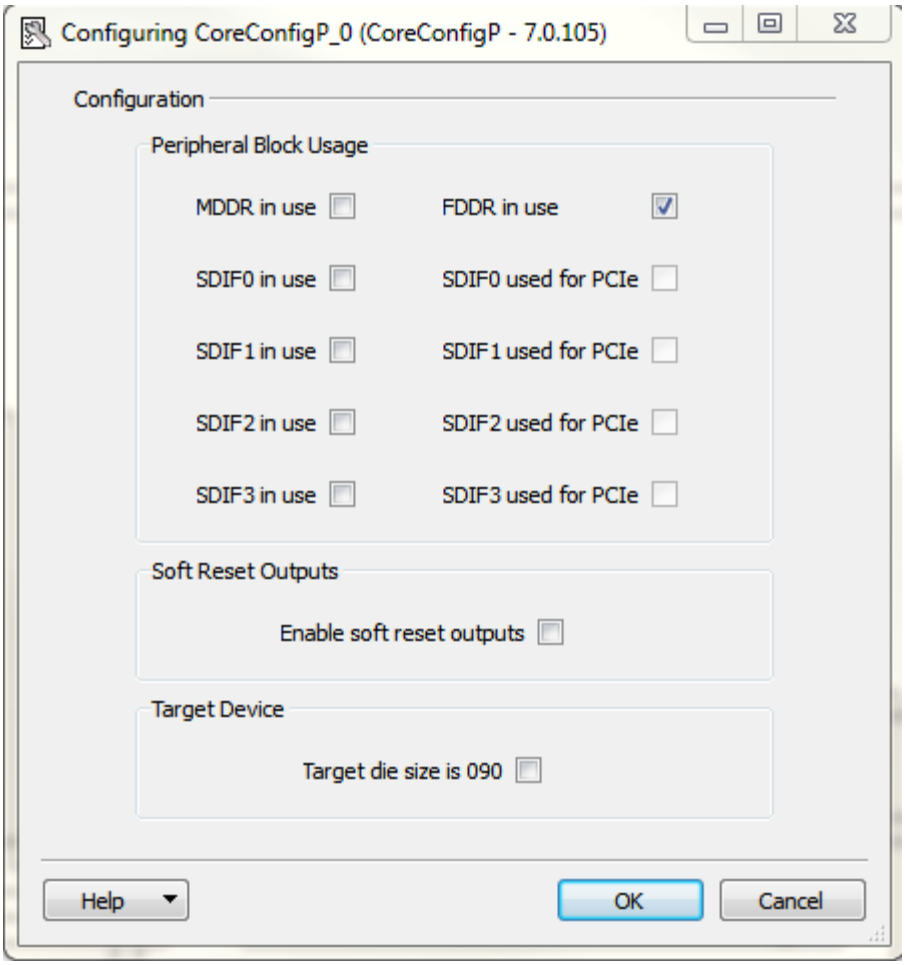


Figure 29 • CoreConfigP Dialog Box

CoreResetP

1 . Instantiate CoreResetP into the same SmartDesign. 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 30 – CoreResetP Configurator)

3. Configure the core to:

– Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
  - EXT_RESET_OUT is never asserted
  - EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
  - EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
  - 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.

– Check the appropriate checkboxes to indicate which peripherals you are using in your design.

– Specify the external DDR memory setting time. 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. Incorrect value for the settling time may result in

simulation errors.

Refer to the CoreResetP handbook for details on the options available to you in this configurator.

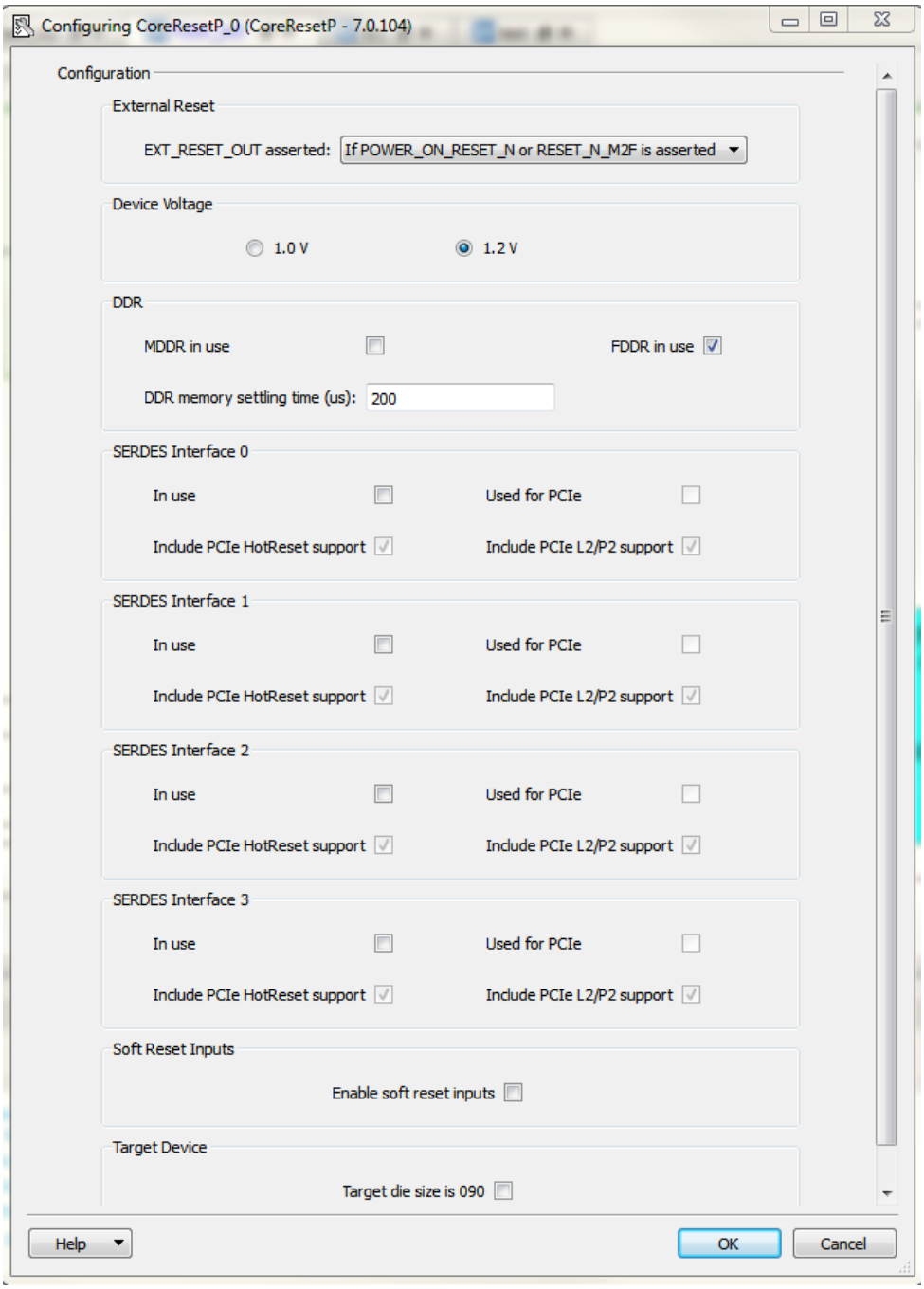


Figure 30. CoreResetP Configurator

Overall Connectivity of the initialization logic (FDDR_INIT)

After you have instantiated and configured the 3 cores CoreABC, CoreConfigP and CoreResetP, appropriate connections have to be made to make the initialization logic operational. See the depiction below in the Figure 31 to understand how the connections are made.

The following is a list of signals that need to be promoted to the top which will be needed when interfacing this

initialization logic with the actual peripheral (FDDR).

- CoreConfigP:
    - APB_S_PRESET_N
    - APB_S_PCLK
    - APB_S_INIT (APB BIF FDDR_APBmslave)
- CoreResetP:
    - RCOSC_25_50MHZ
    - FAB_RESET_N
    - POWER_ON_RESET_N
    - DDR_READY
    - FDDR_CORE_RESET_N
    - FPLL_LOCK
- INIT_PCLK_25MHz (connecting together the PCLK of CoreABC, the FIC_2_APB_M_PCLK of CoreConfigP and the CLK_BASE of CoreResetP).

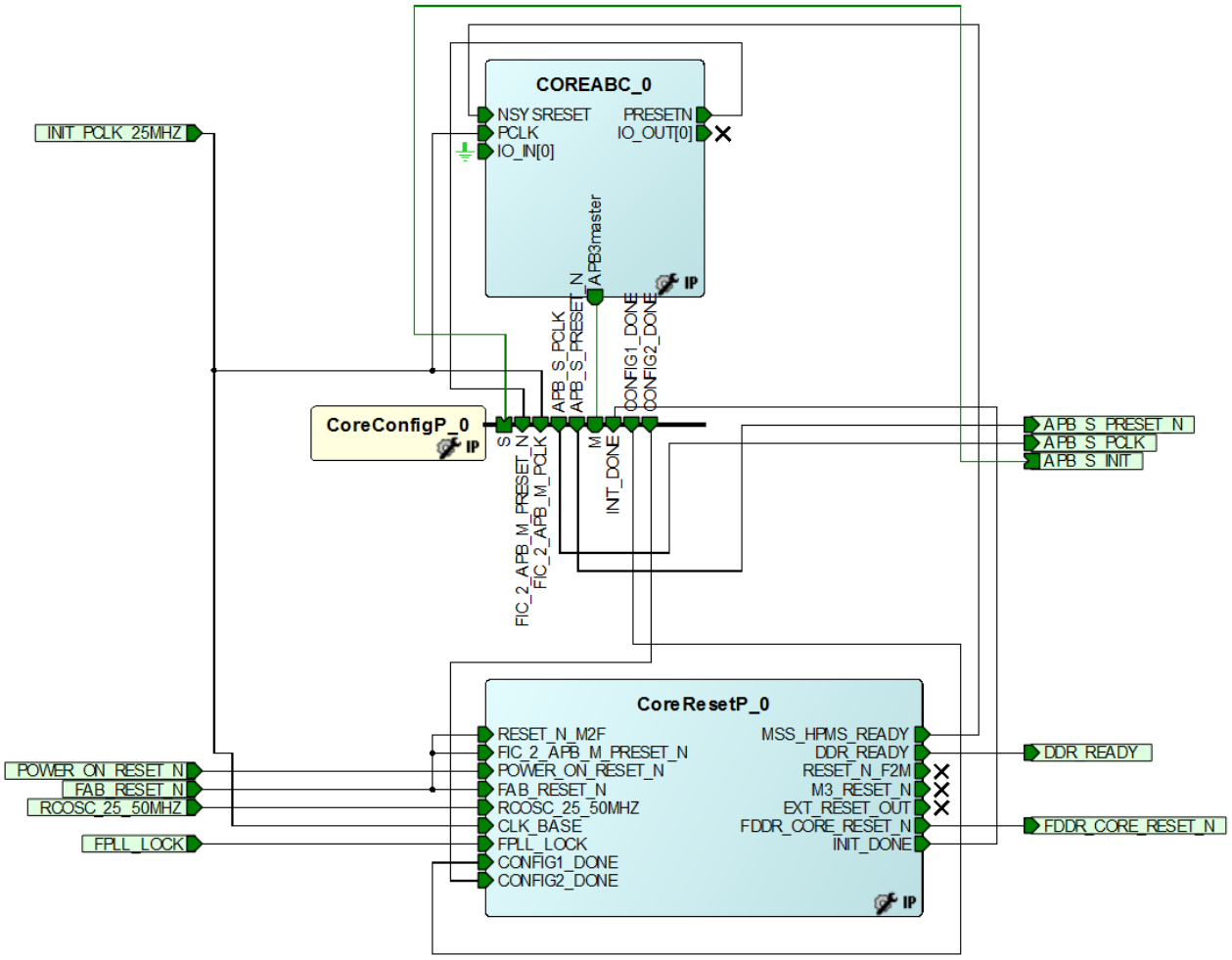


Figure 31. FDDR_INIT (FDDR Initialization Logic)

Instantiating and configuring the FDDR block

Create a Smart Design component and instantiate the FDDR block from the catalog window (from under the Memory and Controllers). Double-click the FDDR block in the SmartDesign canvas to open the configurator and configure the FDDR. (Figure 32). The Fabric DDR (FDDR) controller 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 the FDDR configurator is written to the DDR controller configuration registers by the CoreABC program. 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

Consult 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.

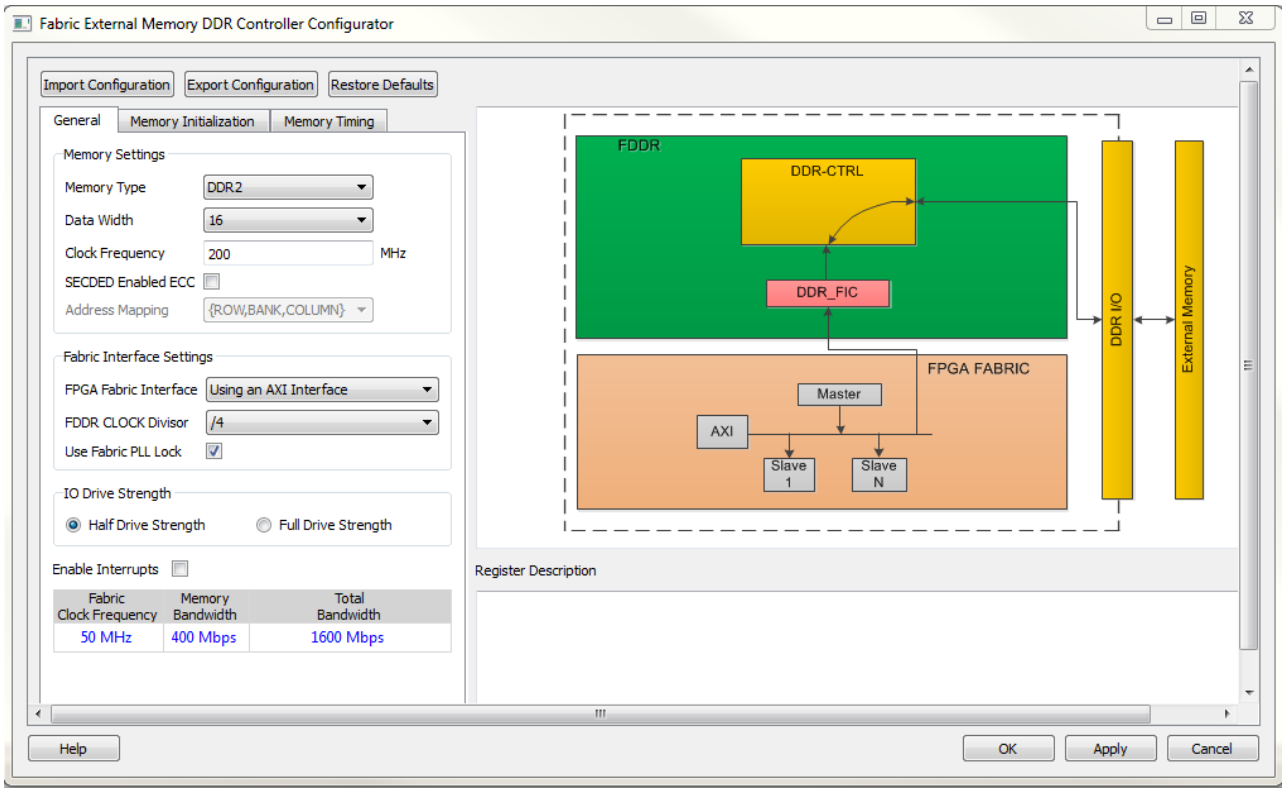


Figure 32. Fabric DDR configurator

Interfacing the FDDR with the Initialization Logic Built for it

In the same SmartDesign the FDDR block is present, instantiate the Smart Design containing the FDDR initialization logic (FDDR_INIT), and do necessary interconnections to interface the FDDR block to the initialization logic. See the depiction below in the Figure 35 to understand how the connections are made.

The following is a list of signal interconnections that need to be made to properly interface the FDDR to the initialization logic.

FROM Port or Bus Interface (BIF)/ Component	TO Port/Bus Interface (BIF)/ Component
APB_S_PCLK/ FDDR	APB_S_PCLK/ initialization logic.
APB_S_PRESET_N/ FDDR	APB_S_PRESET_N/ initialization logic.
APB_SLAVE BIF/ FDDR	APB_S_INIT/ initialization logic
FPLL_LOCK/ FDDR	FPLL_LOCK/ initialization logic
CORE_RESET_N / FDDR	FDDR_CORE_RESET_N / initialization logic

50MHz Oscillator Instantiation

CoreResetP needs to be clocked by the on-chip 50MHz RC oscillator. You must instantiate a 50MHz Oscillator for this purpose.

1. Instantiate the Chip Oscillators core into the same SmartDesign the FDDR block is present. This core can be found in the Libero Catalog under Clock & Management.
2. Configure this core such that the oscillator drives the FPGA fabric, as shown in Figure 33 below.
3. Click "**OK**".
4. Connect the RCOSC_25_50MHz_O2F output of the Oscillator to the RCOSC_25_50MHz input of FDDR_INIT block (FDDR initialization logic).

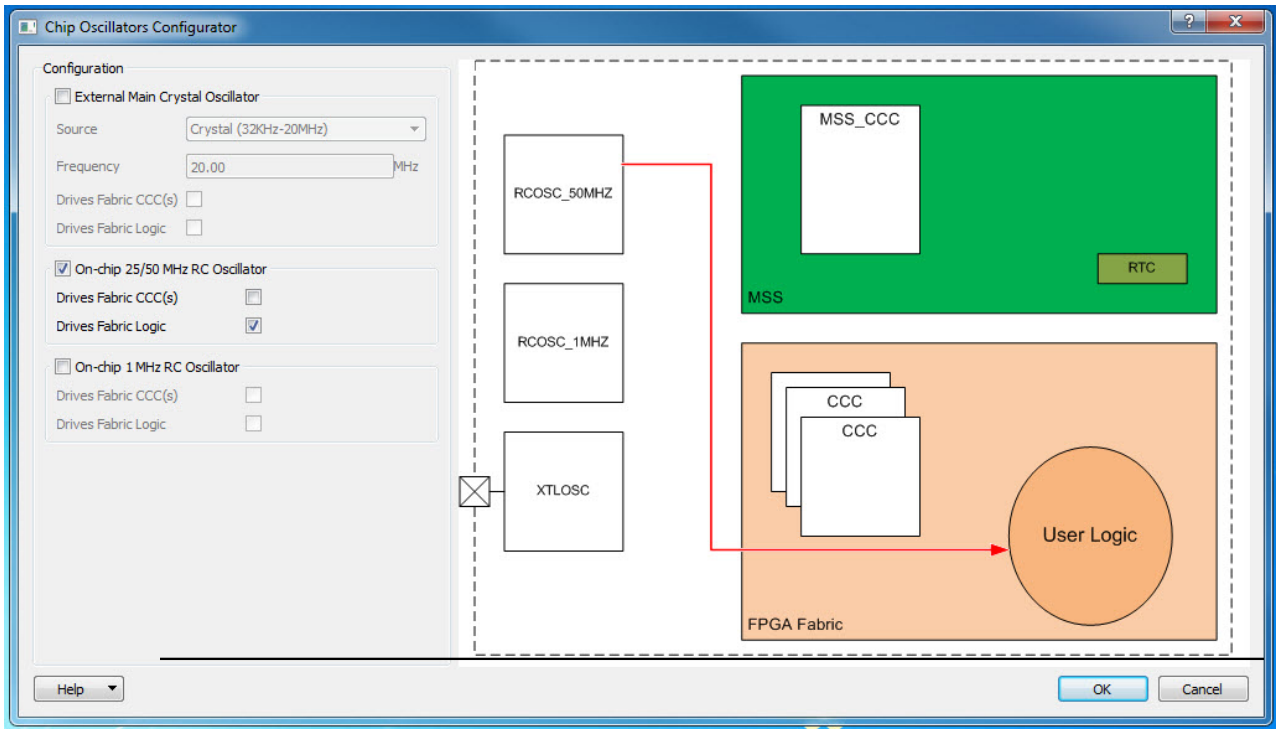


Figure 33 • Chip Oscillators Configurator

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 34).Instantiate the SYSRESET macro into the same SmartDesign the FDDR block is present. This macro can be found in the Libero Catalog under Macro Library. No configuration of this macro is necessary. Drive the POWER_ON_RESET_N input of FDDR_INIT block (FDDR initialization logic with the POWER_ON_RESET_N output signal of this SYSRESET macro.



Figure 34 • SYSRESET Macro

Apart from the above connections, do the following also:

- Promote the FAB_RESET_N pin the initialization logic (FDDR_INIT_0 instance) to the top level (this is the warm reset).
- Promote the DDR_READY of the initialization logic to the top to monitor the status of the FDDR initialization.
- Instantiate a FABCCC block and do the following connections:
    - Drive the INIT_PCLK_25MHZ input pin of the initialization logic with the GLx of FABCCC block configured to 25MHz frequency.
    - Drive the CLK_BASE input pin of the FDDR block with the GLx of FABCCC block (configured to appropriate frequency).
    - Drive the CLK_BASE_PLL_LOCK input pin of the FDDR block with the LOCK pin of the FABCCC block.

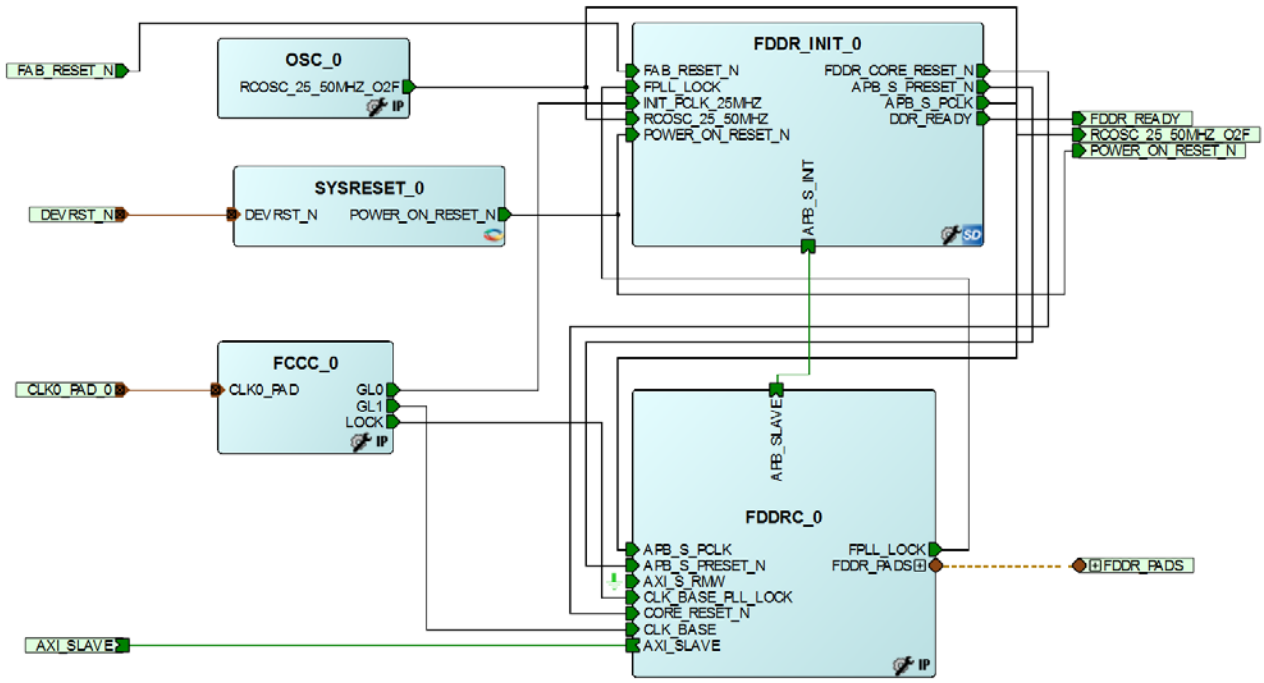


Figure 35. Interfacing FDDR with the Initialization Logic

Continuing with the Design Flow

Next step is to integrate any user logic that you might have with the FDDR block and the initialization logic. Once you have done that, you can generate your top level SmartDesign. This will generate all files that are necessary to implement and simulate your design. You can then proceed with the rest of the Design Flow.

NOTE: After configuring all the desired FDDR registers in the FDDR configurator and upon generating the Smart Design component containing the FDDR block , the FDDR_init_abc.txt file will be generated to the disk. You will need to copy the contents of the FDDR_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC.

Design using MSS_MDDR block

Building Standalone Initialization Logic for MDDR

In order to initialize the MDDR, you must create the initialization subsystem in the FPGA fabric. The FPGA fabric initialization subsystem moves data from the CoreABC program to the DDR configuration registers, manages the reset sequences required for the MDDR block to be operational and signals when the MDDR block is ready to communicate with the rest of your design. To create the initialization subsystem you must:

- Instantiate and configure CoreABC soft IP core.
- Load CoreABC with the initialization program generated to the MDDR_init_abc.txt file.
- Instantiate and configure the CoreConfigP and CoreResetP cores.
- Configure FIC_2 inside the MSS.
- Instantiate the on-chip 50MHz RC oscillator.
- Instantiate the System Reset (SYSRESET) macro.
- Connect these components to peripheral's (MDDR) configuration interface, clocks, resets and PLL lock ports.

CoreABC configuration

1. Create a new SmartDesign component (MDDR_INIT).
2. Instantiate CoreABC into your SmartDesign. This core can be found in the Libero Catalog (under Processors).
3. Double-click the core to open the configurator.
4. Configure the core as shown in the depiction below (Figure 36).

   Configure the data bus width to be 16.

   Configure the maximum number of instructions to at least 256.

   Configure to use AND and OR operations as optional instructions.

   Configure Instruction Store to Hard (FPGA Tiles).
5. Copy the CoreABC program generated for MDDR from the MDDR_init_abc.txt file created under the <project_location>/../*_MSS/ directory, to the CoreABC Program tab. See the figure below.

NOTE: The MDDR_init_abc.txt file will be generated only when you generate the MSS(MDDR) block. So after you've made all the connections and generated all the blocks (including MSS), you will need to copy the contents of the MDDR_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC. You may defer doing this until you completely configure your MSS_MDDR block and generate the SmartDesign component containing it.

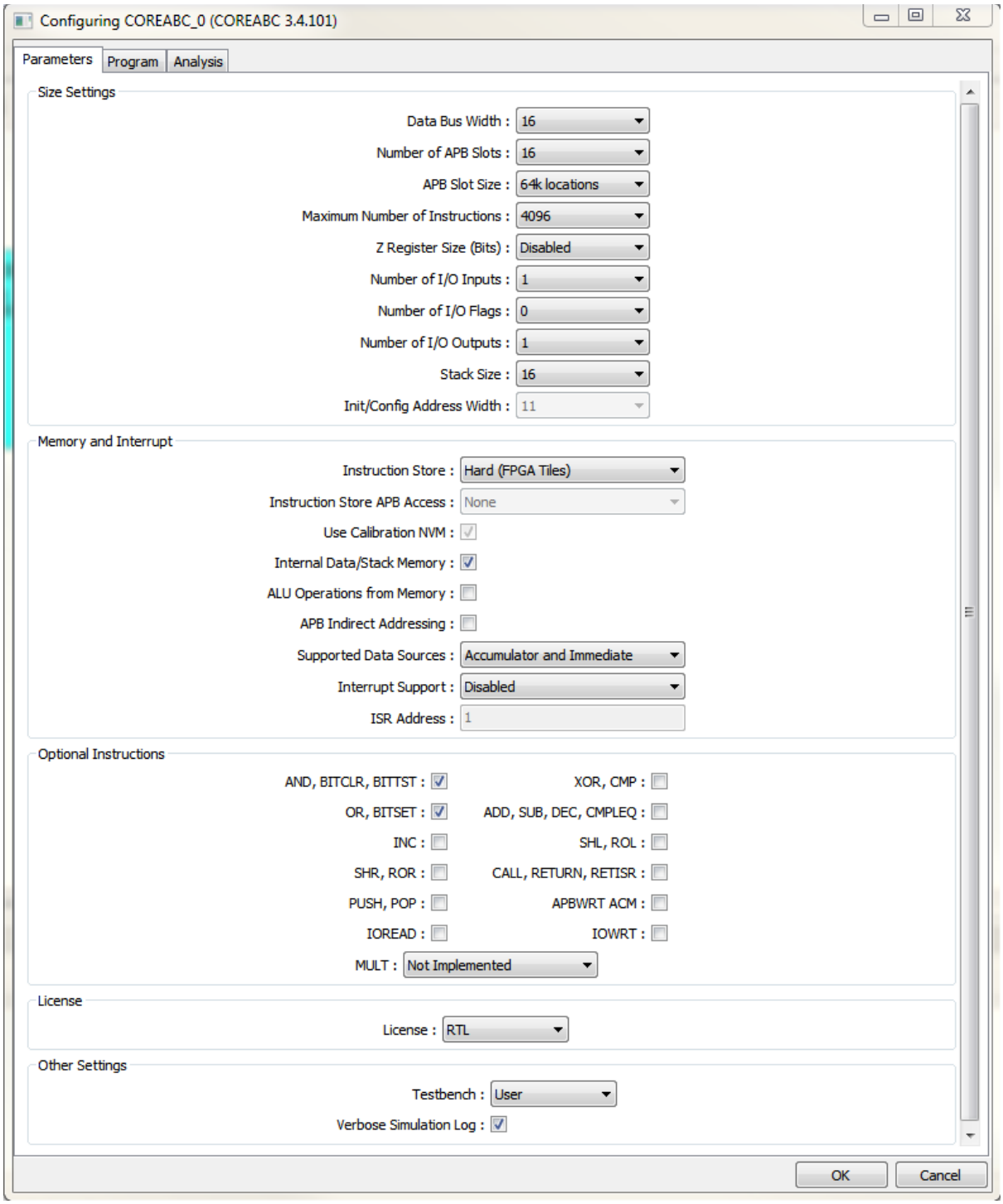


Figure 36. *CoreABC configuration*

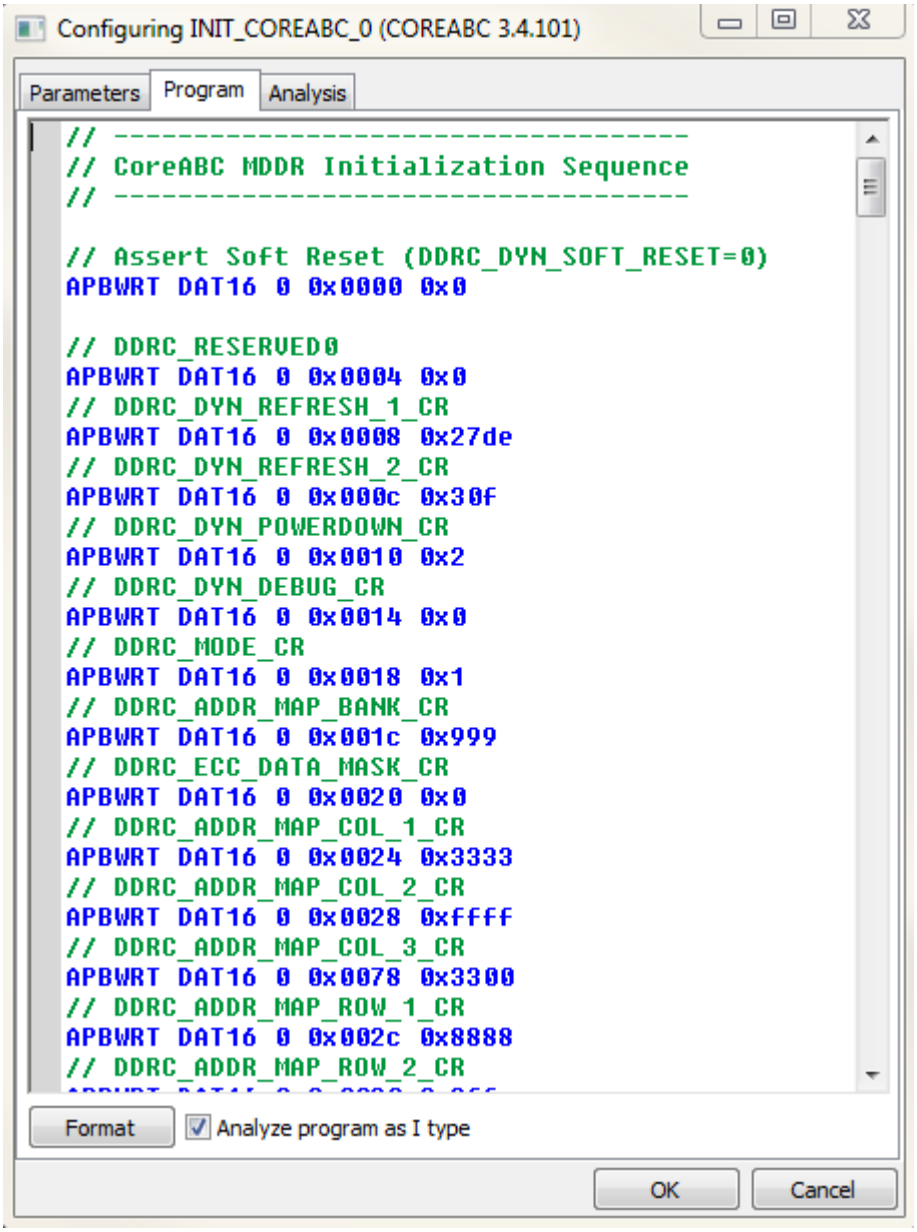


Figure X. *CoreABC program for MDDR*

CoreConfigP

1. Instantiate CoreConfigP into the same SmartDesign. 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 37)

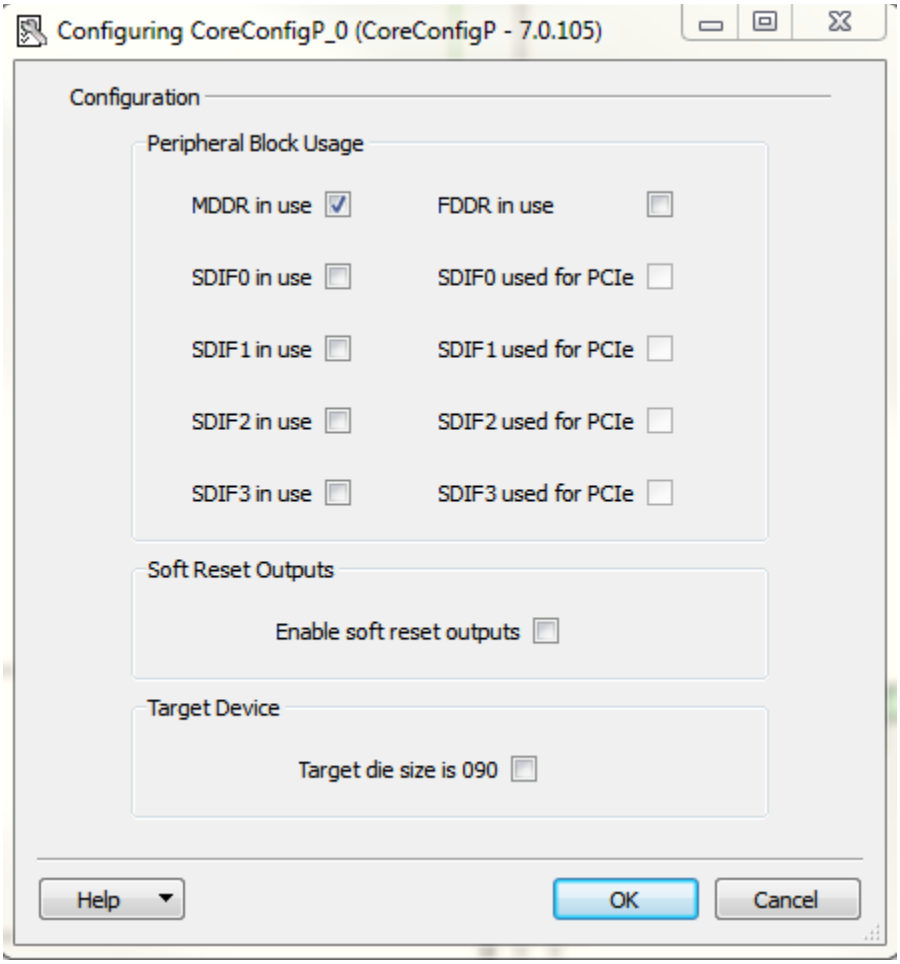


Figure 37 • CoreConfigP Dialog Box

CoreResetP

1. Instantiate CoreResetP into the same SmartDesign. 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 38 – CoreResetP configurator)
3. Configure the core to:
   - Specify the external reset behavior (EXT_RESET_OUT asserted). Choose one of four options:
     - EXT_RESET_OUT is never asserted
     - EXT_RESET_OUT is asserted if power up reset (POWER_ON_RESET_N) is asserted
     - EXT_RESET_OUT is asserted if FAB_RESET_N is asserted
     - 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.
   - Check the appropriate checkboxes to indicate which peripherals you are using in your design.
   - Specify the external DDR memory setting time. 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. Incorrect value for the settling time may result in simulation errors.

   Refer to the CoreResetP handbook for details on the options available to you in this configurator.

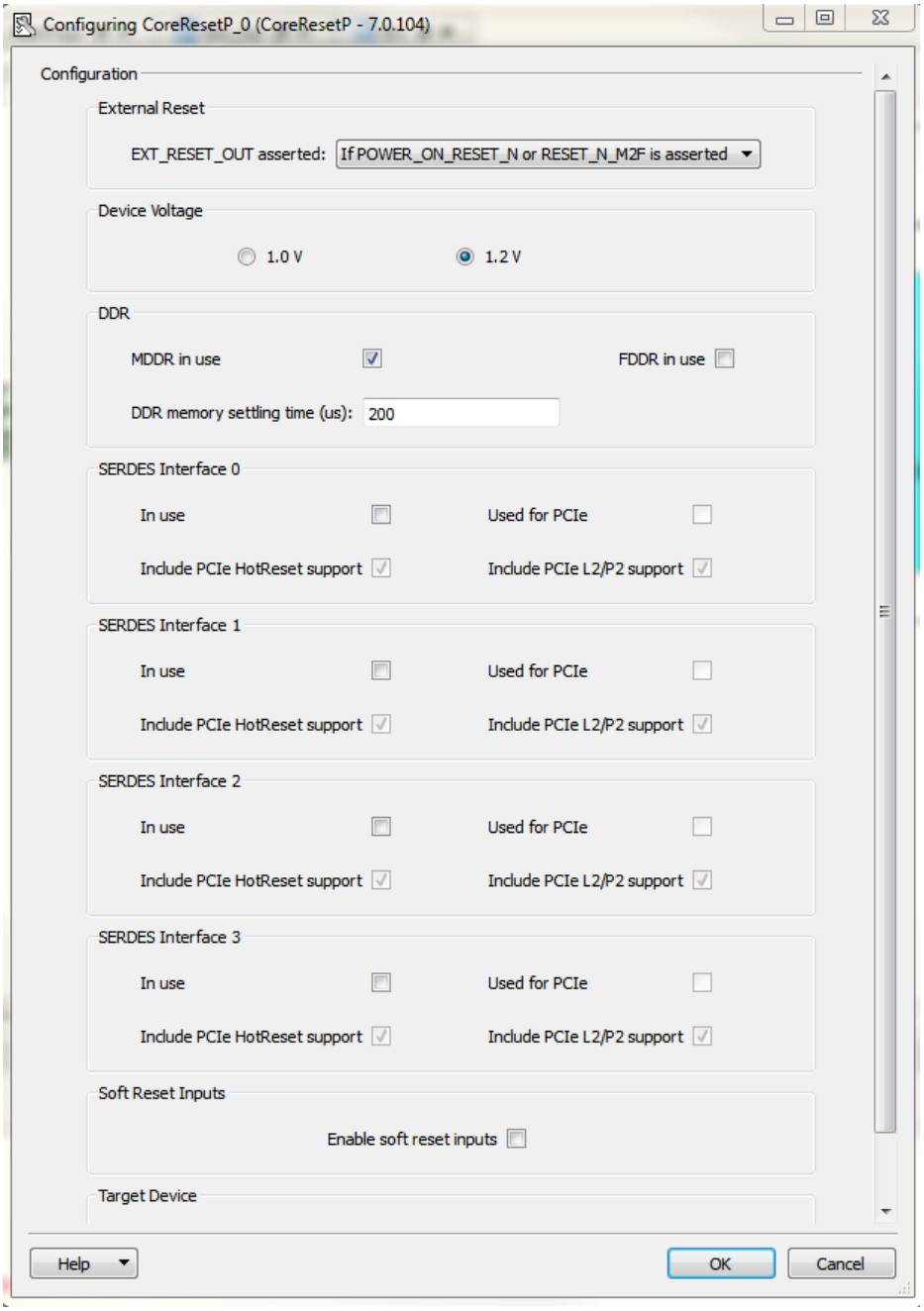


Figure 38. CoreResetP Configurator

Overall Connectivity of the initialization logic (MDDR_INIT)

After you have instantiated and configured the 3 cores CoreABC, CoreConfigP and CoreResetP, appropriate connections have to be made to make the initialization logic operational. See the depiction below in the Figure 39 to understand how the connections are made.

The following is a list of signals that need to be promoted to the top which will be needed when interfacing this initialization logic with the actual peripheral (MSS_MDDR).

- CoreConfigP:
  - APB_S_PRESET_N
  - APB_S_PCLK
  - APB_S_INIT (APB BIF MDDR_APBmslave)
- CoreResetP:
  - RCOSC_25_50MHZ
  - FAB_RESET_N

- POWER_ON_RESET_N
- DDR_READY
- MDDR_DDR_AXI_S_CORE_RESET_N
- MSS_RESET_N_M2F (RESET_N_M2F)
- MSS_RESET_N_F2M (RESET_N_F2M)
- FIC_2_APB_M_PRESET_N

- INIT_PCLK_25MHz (connecting together the PCLK of CoreABC, the FIC_2_APB_M_PCLK of CoreConfigP and the CLK_BASE of CoreResetP).

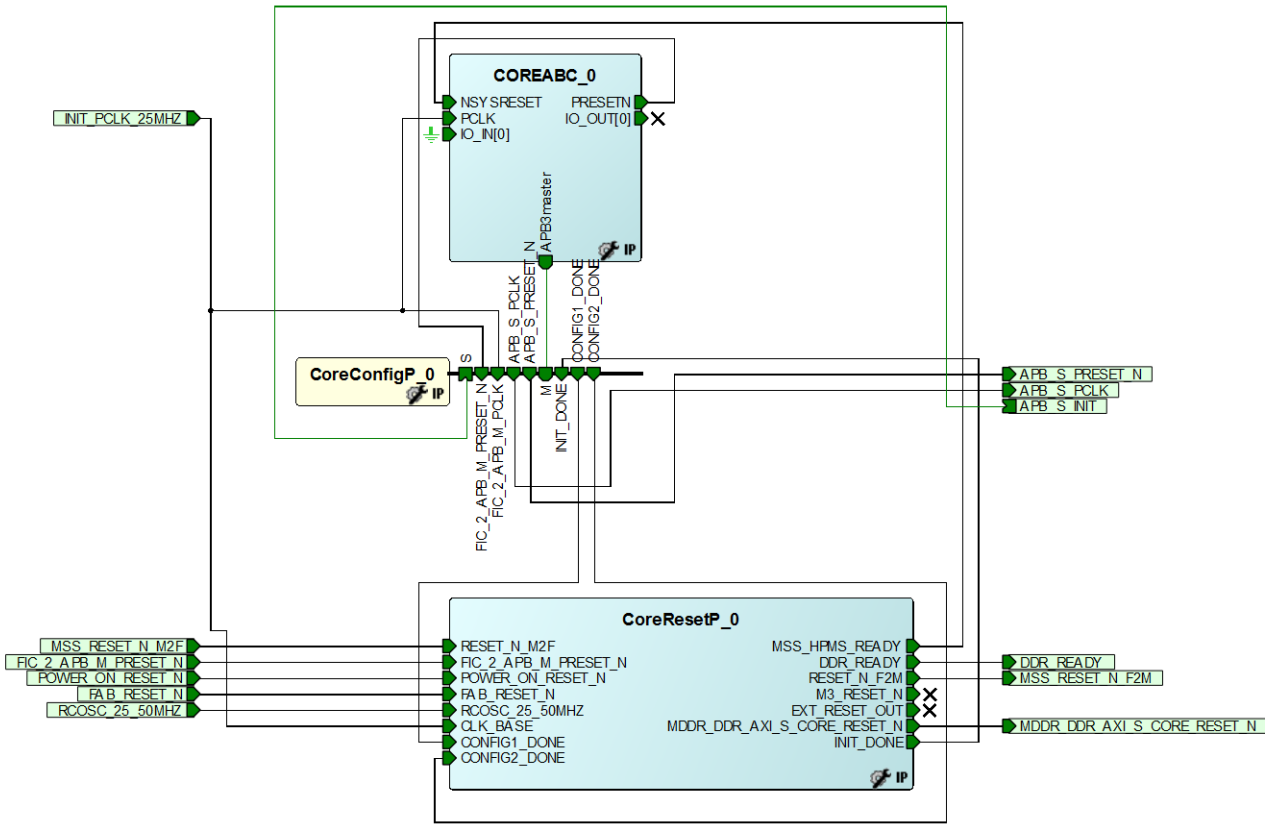


Figure 39. MDDR_INIT (MDDR Initialization Logic)

Instantiating and configuring the MSS(MDDR) block

Create a Smart Design component and instantiate the MSS block from the catalog window (from under the Processors). Double-click the MSS block in the SmartDesign canvas to open the MSS configurator window, and then double-click the MDDR block in the MSS to configure the MDDR (Figure 40). The MSS_DDR (MDDR) controller 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 the MDDR configurator is written to the DDR controller configuration registers by the CoreABC program. 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

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

If you want to access the MSS_DDR controller from the fabric (fabric master), you can configure the 'Fabric Interface Settings' section and choose the type of fabric interface. This would enable the MDDR FIC_64, the fabric interface to the MDDR.

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

.

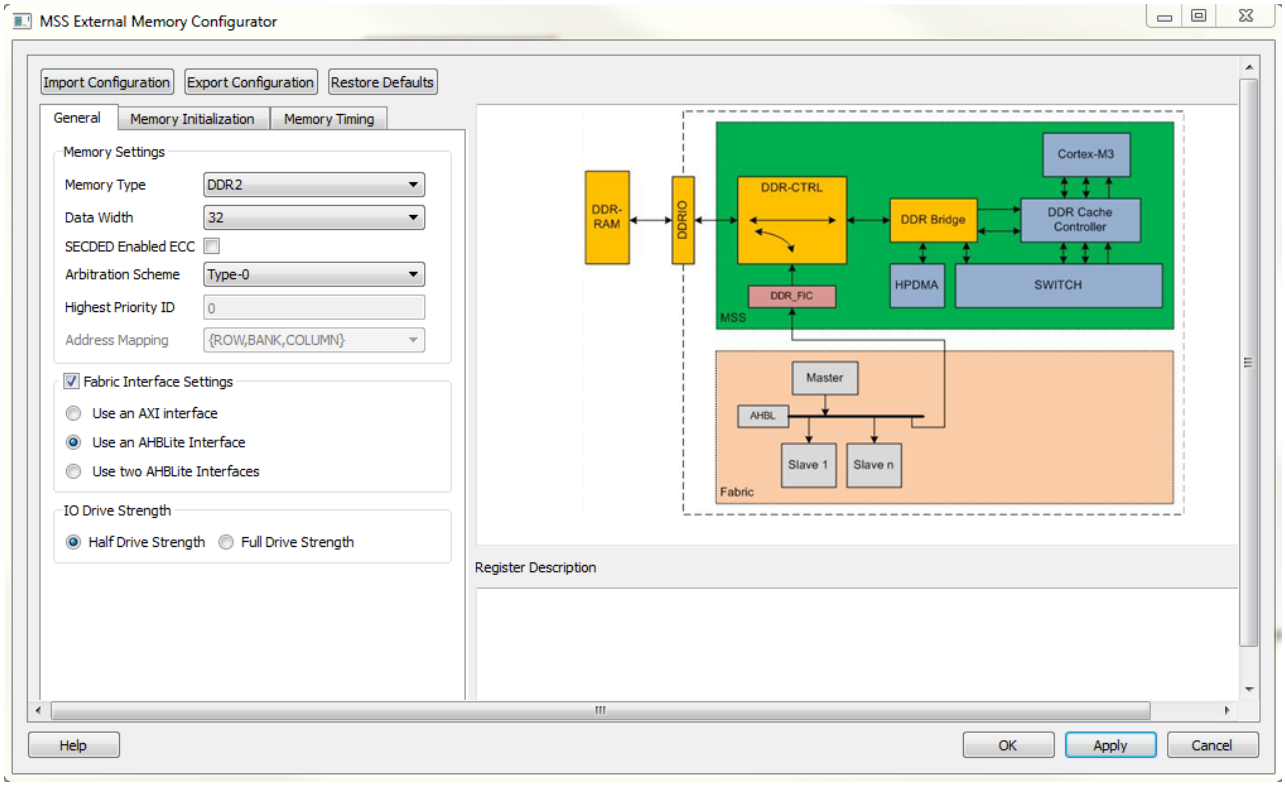


Figure 40. MSS MDDR configurators

MSS FIC_2 APB Configuration

To expose the clock and reset ports and the BIF ports corresponding to the MDDR block, you need to configure the MSS FIC_2 block inside the MSS. To configure the MSS FIC_2:

1. Open the FIC_2 configurator dialog from the MSS configurator (Figure 41)
2. Check the following checkbox:
   – MSS DDR
3. 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 MSS_DDR related ports MDDR_APB_S_PRESET_N, MDDR_APB_S_PCLK and MDDR_APB_SLAVE (APB BIF) along with 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 MDDR initialization logic.

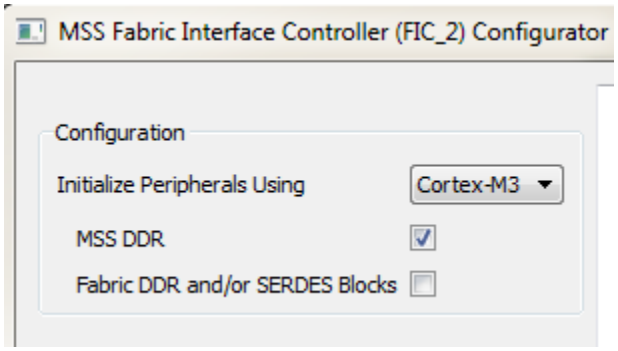


Figure 41. MSS FIC_2 configurator

Interfacing the MSS_MDDR with the Initialization Logic Built for it

In the same SmartDesign the MSS block is present, instantiate the Smart Design containing the MDDR initialization logic (MDDR_INIT), and do necessary interconnections to interface the MSS_MDDR to the initialization logic. See the depiction below in the Figure 44 to understand how the connections are made.

The following is a list of signal interconnections that need to be made to properly interface the MDDR to the initialization logic.

FROM Port or Bus Interface (BIF)/ Component	TO Port/Bus Interface (BIF)/ Component
MDDR_APB_S_PCLK/ MSS	APB_S_PCLK/ initialization logic.
MDDR_APB_S_PRESET_N/ MSS	APB_S_PRESET_N/ initialization logic.
MDDR_APB_SLAVE BIF/ MSS	APB_S_INIT/ initialization logic
MDDR_DDR_CORE_RESET_N / MSS	MDDR_DDR_AXI_S_CORE_RESET_N / initialization logic
MSS_RESET_N_M2F/ MSS	MSS_RESET_N_M2F/ initialization logic
FIC_2_APB_M_PRESET_N/ MSS	FIC_2_APB_M_PRESET_N/ initialization logic
MSS_RESET_N_F2M/ MSS	MSS_RESET_N_F2M/ initialization logic

50MHz Oscillator Instantiation

CoreResetP needs to be clocked by the on-chip 50MHz RC oscillator. You must instantiate a 50MHz Oscillator for this purpose.

1. Instantiate the Chip Oscillators core into the same SmartDesign the MSS block is present. This core can be found in the Libero Catalog under Clock & Management.
2. Configure this core such that the oscillator drives the FPGA fabric, as shown in Figure 42 below.
3. Click "**OK**".
4. Connect the RCOSC_25_50MHz_O2F output of the Oscillator to the RCOSC_25_50MHz input of MDDR_INIT block (MDDR initialization logic).

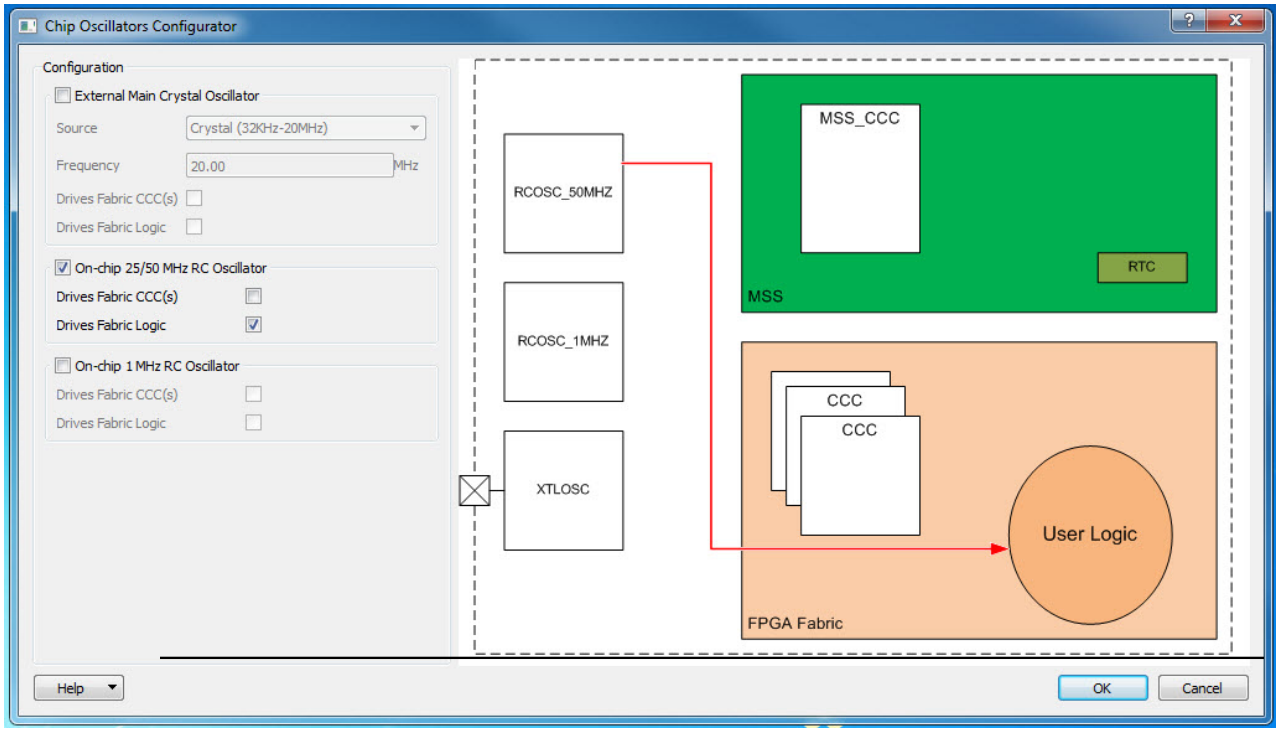


Figure 42 • Chip Oscillators Configurator

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 43).Instantiate the SYSRESET macro into the same SmartDesign the MSS block is present. This macro can be found in the Libero Catalog under Macro Library. No configuration of this macro is necessary. Drive the POWER_ON_RESET_N input of MDDR_INIT block (MDDR initialization logic with the POWER_ON_RESET_N output signal of this SYSRESET macro.



Figure 43 • SYSRESET Macro

Apart from the above connections, do the following also:

- Promote the FAB_RESET_N pin the initialization logic (MDDR_INIT_0 instance) to the top level (this is the warm reset).
- Promote the DDR_READY of the initialization logic to the top to monitor the status of the MDDR initialization.
- Instantiate a FABCCC block and do the following connections:
    - Drive the INIT_PCLK_25MHZ input pin of the initialization logic with the GLx of FABCCC block configured to 25MHz frequency.
    - Drive the MCCC_CLK_BASE input pin of the MSS block with the GLx of FABCCC block (configured to appropriate frequency).
    - Drive the MCCC_CLK_BASE_PLL_LOCK input pin of the MSS block with the LOCK pin of the FABCCC block.

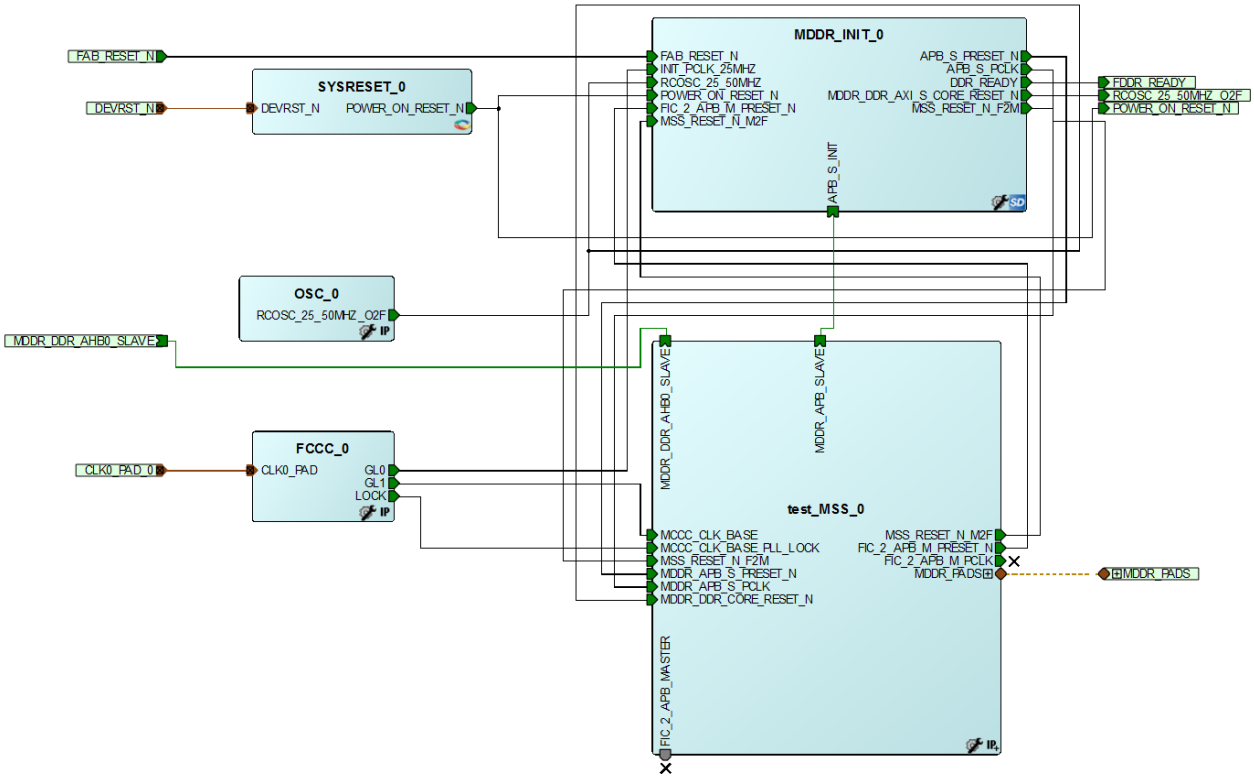


Figure 44. Interfacing MSS(MDDR) with the Initialization Logic

Continuing with the Design Flow

Next step is to integrate any user logic that you might have with the MSS(MDDR) block and the initialization logic. Once you have done that, you can generate your top level SmartDesign. This will generate all files that are necessary to implement and simulate your design. You can then proceed with the rest of the Design Flow.

NOTE: After configuring all the desired MDDR registers in the MSS_MDDR configurator and upon generating the Smart Design component containing the MSS block , the MDDR_init_abc.txt file will be generated to the disk. You will need to copy the contents of the MDDR_init_abc.txt file to the CoreABC Program tab and regenerate the initialization logic Smart Design component containing CoreABC.

5 – Creating and Compiling the Firmware Application

In the case of a SmartFusion2 design with Cortex-M3 processor waiting to communicate with any peripheral, the application's main() function should wait for the assertion of INIT_DONE (OR DDR_READY/SDIF_READY based on which peripheral is being used) signal of CoreResetP before it is executed.

In the case of a SmartFusion2 design using more than 1 peripheral (combination of MDDR/FDDR/SERDESIF_n) with Cortex-M3 waiting to communicate with the peripherals, the application's main() function should wait for the assertion of INIT_DONE (OR DDR_READY/SDIF_READY based on which peripheral is being used) signals of all the CoreResetP cores each corresponding to a peripheral used. 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.

6 – Simulating the Design

Unlike the normal flow (Standalone Initialization OFF) where the peripheral initialization involves various *_init.bfm files to mimic the initialization in simulations, no such files are required in case of the Standalone Initialization mode ON. The CoreABC program is solely responsible for the peripheral initialization both in simulations and on board (device).

Simulating a SmartFusion2 design which uses peripherals and Cortex M3 trying to communicate with them:

In a SmartFusion2 design, when you generate the MSS component containing the Cortex M3, the following simulation files 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 the user.bfm directly.
- peripheral_init.bfm, FDDR_init.bfm, MDDR_init.bfm and SERDESIF_n_init.bfm are not executed at all.
- **user.bfm** - Contains the user commands. Edit this file to enter your BFM commands. This is similar to the Cortex M3 application's main() function. If the BFM commands in the user.bfm file are targeted to a peripheral that needs initialization, it is the responsibility of the user to add logic to the user.bfm to wait for the peripheral to be ready for communication before attemting to communicate with it.

When you generate a Smart Design component containing SERDESIF_n (configured in PCIe mode), then the following files are generated in the <project dir>/simulation directory:

- **SERDESIF_n_user.bfm** - Contains the user commands. Edit this file to enter your BFM commands that would exercise the SERDESIF_n PCIe.

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**.

© 2014 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

5-02-00384-1/09.14