XAPP1303 (v1.0.2) February 27, 2017

Integrating LogiCORE SEM IP with AXI in Zynq UltraScale+ Devices

Author: Michael Welter

Summary

This application note outlines how to use a Zynq® UltraScale+™ MPSoC in conjunction with the LogiCORE™ IP UltraScale+ architecture Soft Error Mitigation (SEM) controller. The reference design provides an example of how the SEM controller is integrated with a processing system (PS). A Xilinx® ZCU102 board is targeted, but the design can be changed for different devices, family architectures, and boards. The PS interfaces with the SEM controller using the AXI4-Lite interface. The AXI4-Lite interface should be used to monitor the health of the SEM controller and to aid in debugging if an error condition is encountered. Reports from the SEM controller are read from a FIFO using this interface. These reports provide the most detailed information about the health of the SEM controller. Control, status, and interrupt information is provided through an AXI4-Lite register interface. Status from the register interface is less detailed than the report information from the FIFO. The interrupts are configurable and are used to inform the PS when the SEM controller needs to be serviced, such as when the SEM controller detects an error in configuration memory.

Alternatively, *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298) [Ref 1] outlines how to use a Zynq UltraScale+ MPSoC in conjunction with the LogiCORE IP UltraScale+ architecture SEM controller. Instead of using an AXI4-Lite interface, it uses a minimal set of the Zynq MPSoC processor's extended multiplexed I/O (EMIO) general purpose I/O (GPIO) pins to communicate with and manage the SEM controller.

Reference Design

The reference design was created using Vivado® Design Suite 2016.2 IP integrator.

Download the reference design files for this application note from the Xilinx website.

This reference design, illustrated in Figure 1 and Figure 2, uses AXI4-Lite to connect the SEM controller to the Zynq UltraScale+ MPSoC block. This design consists of the following Vivado IP integrator blocks:

- Zynq UltraScale+ MPSoC
- Processor System Reset - Controls the sequencing of reset deassertion
- AXI Interconnect - Allows multiple AXI peripherals to be connected to the Zynq UltraScale+ MPSoC block
- AXI Interrupt controller - Provides a single interrupt from all of the AXI peripherals to the Zynq UltraScale+ MPSoC block

- Utility buffer - Buffer configured as an IBUFDS
- UltraScale architecture Soft Error Mitigation controller
- AXI2SEM module - Numerous modules for providing the AXI interface to and from the UltraScale Soft Error Mitigation controller
- Constant - Used to drive a constant value on a signal or bus
- Concat - Used to concatenate signals and smaller buses of varying widths into a larger bus
- Slice - Used to rip bits of a bus

Refer to Figure 1, Figure 2, and Figure 3 in Hardware for illustrations.

AXI2SEM connects to the PS through the AXI Interconnect. The interconnect allows for other AXI peripherals to be connected to the PS. In this reference design the AXI Interrupt controller is also connected to the AXI Interconnect. The AXI2SEM module has numerous submodules and is essentially a bridge to allow communication between the PS and the SEM controller. See Figure 3 for the block design. A register interface in AXI2SEM allows the PS to send commands to the SEM controller. It is also used to provide SEM controller health and status information to the PS. See Appendix A—AXI2SEM Register Definition for the AXI2SEM register definition. Using the AXI4-Lite interface, the PS writes all SEM commands to a FIFO. All status information from the SEM controller is read from another FIFO using the same AXI4-Lite interface. The ICAP interface is arbitrated for using the AXI2SEM ICAP register.

In this reference design, the AXI4-Lite interface operates at 100 MHz while SEM operates at an asynchronous clock rate of 74.25 MHz. This reference design uses a minimal set of interrupts to demonstrate the design. After the PS initializes the SEM controller via the ICAP register, the PS detects that the SEM Ready interrupt has been asserted. This signifies that SEM initialization completed. The interrupt is serviced and the initialization report is read from the log FIFO. The log FIFO contains the SEM controller's ASCII status information because the SEM UART interface is not used. Then the PS injects a correctable error into the configuration memory of the PL. The SEM controller detects the error and performs correction. This causes a correction state interrupt to trigger (this interrupt is triggered from the status_correction signal of the SEM controller). The PS services this interrupt and reads the correction report from the log FIFO. Finally, the PS injects an uncorrectable error. The SEM controller detects the error, is unable to correct it, but an uncorrectable interrupt is generated from the status_uncorrectable signal from the SEM controller. The interrupt is serviced and you are instructed to reconfigure the FPGA.

IMPORTANT: *This application and all supporting files are for **demonstration purposes only**. All files are delivered **as is**. This reference design is not intended to be a drop-in solution. After integrating this reference design into a new design, thorough verification is required to ensure the resulting solution meets functional and reliability goals.*

Hardware

Reference Design

Figure 1 shows a high-level block diagram of the reference design. A PC connects to the evaluation board JTAG port and USB to Serial port. A more detailed Vivado IP integrator

diagram is shown in Figure 2. The PS is used to send commands to the SEM controller. The AXI2SEM module converts the AXI transaction (or transactions) to SEM commands in its register interface module. Then it sends the commands to the SEM controller. The AXI2SEM module provides SEM status to the PS through the register interface. An IP interrupt controller is instantiated in the AXI2SEM module to pass interrupts to the PS. SEM mitigation actions should be based upon the interrupts. The interrupt from AXI2SEM is connected to the AXI Interrupt controller, which then sends the interrupt to the PS. The AXI Interrupt controller is used to combine multiple AXI peripheral interrupts into a single interrupt that is connected to the PS. In this reference design, the AXI2SEM module is the only other AXI peripheral.

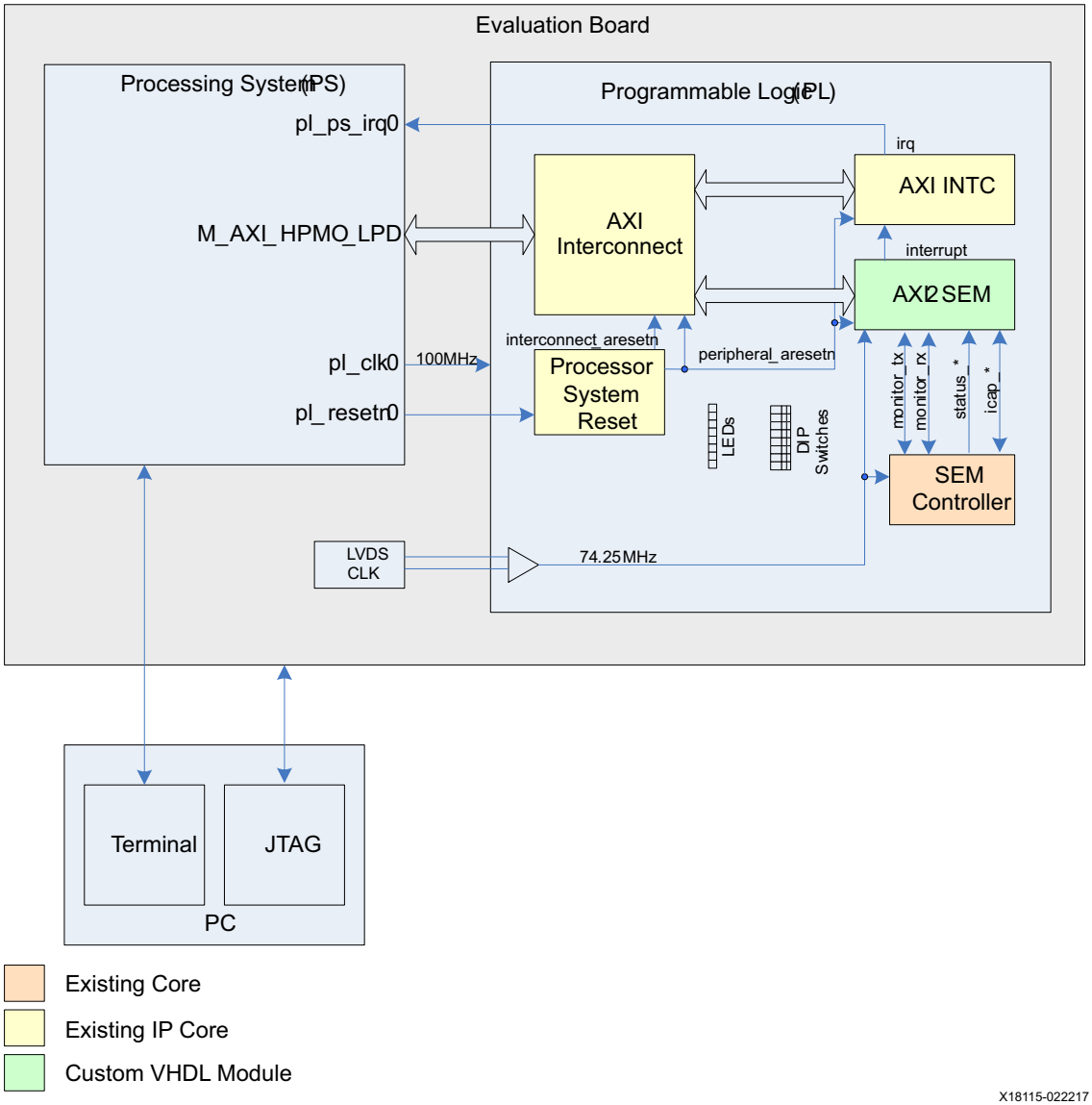

Figure 1: **High-Level Block Diagram**

AXI2SEM uses the signals listed in Table 1 to provide visual indicators by connecting them to the evaluation board GPIO LEDs.

Table 1: **AXI2SEM GPIO Connections**

Signal Name	GPIO LED	Description
irq	GPIO LED[4]	Interrupt request to the PS
cap_req	GPIO LED[5]	ICAP Request
cdc_icap_grant	GPIO LED[6]	ICAP Grant
cdc_icap_release	GPIO LED[7]	ICAP Release

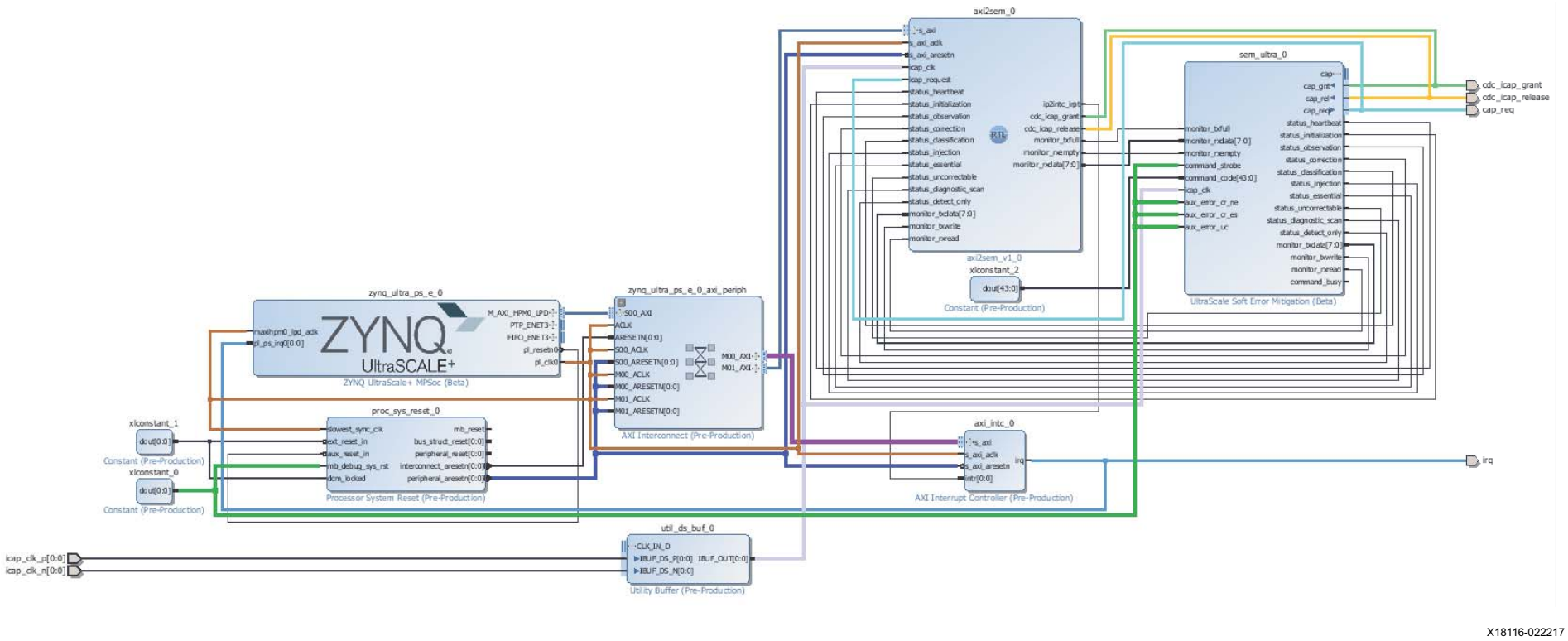

Figure 2: **Vivado IP Integrator Diagram**

The AXI2SEM block diagram is shown in Figure 3.

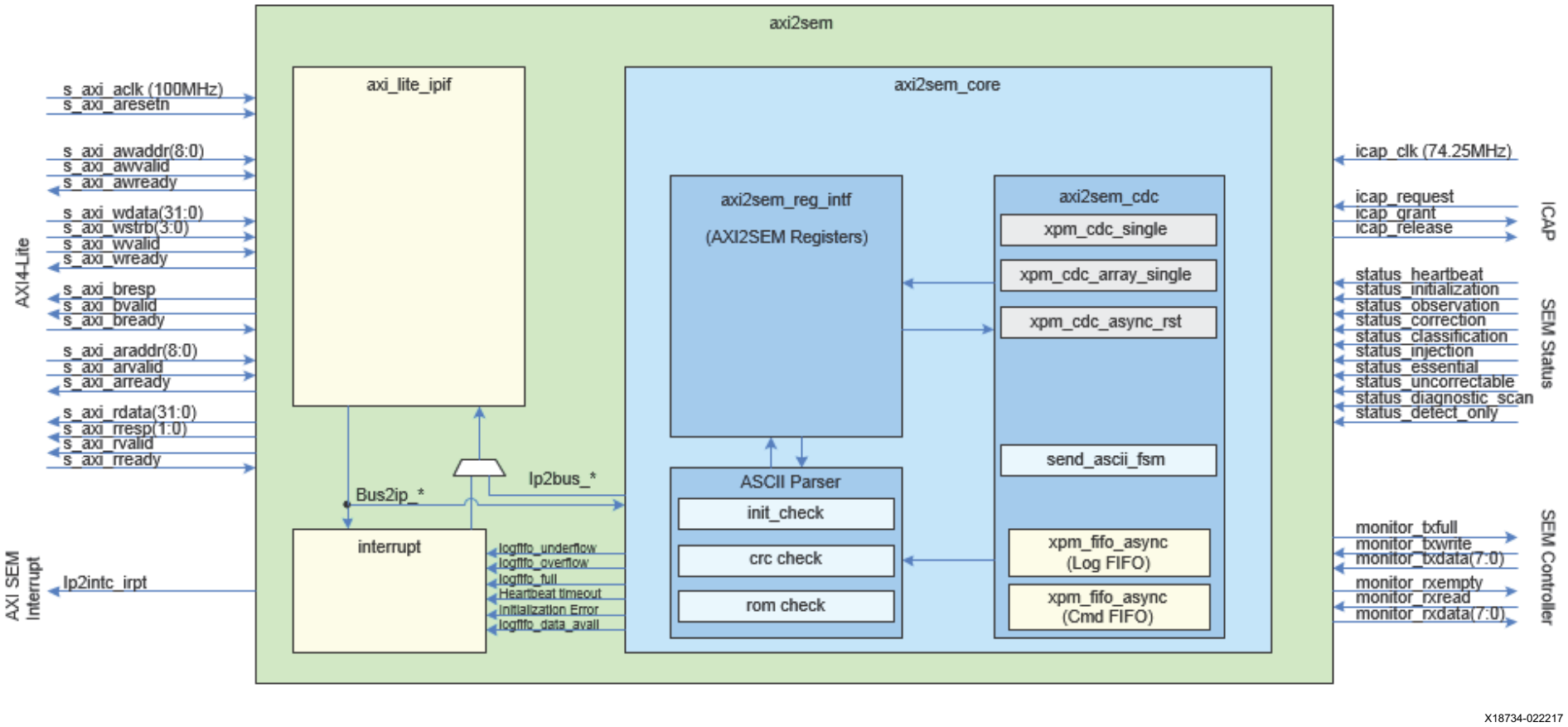

Figure 3: **AXI2SEM Block Diagram**

axi2sem.vhd

This VHDL module is a wrapper file containing the logic and submodules that provide the bridge between the AXI Interconnect and the SEM controller.

The source code for this module is provided in the application note ZIP file at `<path>/Projects/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source`. This is the design hierarchy:

```
axi2sem.vhd
|-- axi_lite_ipif.vhd
|-- interrupt_control.vhd
`-- axi2sem_core.vhd
    |-- axi2sem_reg_intf.vhd
    |-- axi2sem_cdc.vhd
    |   |-- send_ascii_fsm
    |   |-- xpm_cdc_single *
    |   |-- xpm_cdc_array_single *
    |   |-- xpm_cdc_async_rst *
    |   `-- xpm_fifo_async **
    `-- ascii_parser.vhd
        |-- init_check.vhd
        |-- rom_check.vhd
        `-- crc_check.vhd

* The XPM_CDC modules are provided with Vivado tools.
** The XPM_FIFO modules are enabled when the reference design is created. Refer to Create
Project for more information.
```

This module instantiates the LogiCORE IP AXI4-Lite intellectual property interface (IPIF) module (`axi_lite_ipif.vhd`), LogiCORE IP Interrupt Control module (`interrupt_control.vhd`), and the `axi2sem_core` module. The previously mentioned LogiCORE IP modules are existing designs that were leveraged for this reference design. Refer to their documentation for a better understanding of how the modules operate:

- *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) [Ref 2]
- *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155) [Ref 3]

The LogiCORE IP Interrupt Control module [Ref 2] was used to combine multiple AXI2SEM interrupts into a single interrupt (`ip2intc_irpt`) that is connected to the AXI Interrupt controller input signal (`intr`) [Ref 4]. Any action taken for mitigation should be based upon the interrupts received. After any interrupt occurs, the AXI2SEM registers should be used for debugging purposes. These registers should also be used to periodically monitor the health of the SEM controller.

The LogiCORE IP Interrupt controller can be used to generate up to 32 interrupts. In this design, 12 interrupts are connected but only 4 interrupts are enabled in the reference design. The enabled interrupts are bold in Table 2. Each interrupt can be independently enabled. More can be added if desired. Each interrupt can be independently configured to be pass-through (inverting or non-inverting), registered level (inverting or non-inverting), positive edge detect, or negative edge detect. See the *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for more details [Ref 2]. Each interrupt in the reference design is listed in Table 2.

Table 2: **AXI2SEM Interrupts**

Interrupt Name	Configuration	Enabled/Disabled	Test Status
SEM Ready	**rising edge detect**	**enabled**	**tested**
CRC Detected	rising edge detect	disabled	untested
ROM Detected	rising edge detect	disabled	untested
Status Uncorrectable	**rising edge detect**	**enabled**	**tested**
Status Correction	**falling edge detect**	**enabled**	**tested**
Status Initialization	falling edge detect	disabled	tested
Initialization Error	rising edge detect	disabled	untested
Heartbeat Timeout	**rising edge detect**	**enabled**	**untested**
Log FIFO Overflow	rising edge detect	disabled	untested
Log FIFO Full	rising edge detect	disabled	untested
Log FIFO Underflow	rising edge detect	disabled	untested
Log FIFO Data Available	rising edge detect	disabled	untested

IMPORTANT: *This application and all supporting files are for **demonstration purposes only**. All files are delivered **as is.** This reference design is not intended to be a drop-in solution. After integrating this reference design into a new design, thorough verification is required to ensure the resulting solution meets functional and reliability goals.*

The interrupts in Table 2 marked as untested were not tested but are expected to work as intended. In a user system, Xilinx recommends these interrupts be enabled and monitored:

SEM Ready, status uncorrectable, status correction, status initialization, initialization error, heartbeat timeout, Log FIFO Overflow, Log FIFO Full, and Log FIFO Underflow.

The ROM Detected and CRC Detected interrupts do not necessarily need to be monitored because the SEM controller's status correction signal generates a correction state interrupt any time the correction state is passed through. This behavior is also the same with the Status Uncorrectable interrupt. Any time an uncorrectable interrupt occurs, it is preceded by a Status Correction interrupt. So the correctable interrupt needs to be serviced, then the uncorrectable interrupt needs to be serviced. When the correctable interrupt is asserted, all of the SEM log information is already written to the Log FIFO regardless if the error SEM detected was correctable or uncorrectable. If desired, these interrupts could be used to determine the type of interrupt that was asserted, but the Error register in the AXI2SEM registers can also be read to determine what type of error occurred.

Note: When an interrupt from the `status_correction` signal occurs, it does not mean the SEM controller performed a correction. It only indicates the SEM controller's correction state was passed through. Logic would need to be added to the design to create an interrupt that asserts when a correction occurs. This could be done by using the falling edge of the `status_correction` signal in conjunction with the `status_uncorrectable` signal. If `status_uncorrectable` is 0 at the falling edge of the `status_correction` signal, a correction occurred. However, if `status_uncorrectable` is 1 at the falling edge of the `status_correction` signal, an uncorrectable error occurred.

The status initialization interrupt has been tested, but for this reference design, it was preferred to use the SEM Ready interrupt instead. The SEM Ready interrupt is driven by the `init_check.vhd` module. It does not assert until the SEM controller initialization process has completed and the controller enters either the Idle, Detect Only, or Observation state. The initialization interrupt is asserted when initialization completes, but it does not have any information about whether the SEM controller transitioned to the Idle, Observation, or Detect Only state. In addition, the software application verifies the SEM Ready register is set to a particular value, and this register can be read immediately after the SEM Ready interrupt is detected without having to add a wait in software before reading this register.

axi_lite_ipif.vhd

Refer to *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155) [Ref 3]. The source code for this module is provided in the ZIP file at `MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0`.

interrupt_control.vhd

See *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) [Ref 2]. The source code for this module is provided in the ZIP file at `MPSoC_AXI2SEM/Vivado/hdl/interrupt_control_v3_1`.

axi2sem_core.vhd

This module is a wrapper file that instantiates the SEM register interface (`axi2sem_reg_intf.vhd`), SEM/AXI clock domain crossing (`axi2sem_cdc`), and ASCII parse logic (`ascii_parser.vhd`) modules.

axi2sem_reg_intf.vhd

This module instantiates all the registers in Appendix A—AXI2SEM Register Definition. It provides various command, health, and status information of the SEM controller, and provides an AXI interface to this information. See Table 4, AXI2SEM Register Map for the register definition. See *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 5] to understand the operation of the SEM controller.

axi2sem_cdc.vhd

This module provides all of the Clock Domain Crossing logic between AXI and SEM. It uses Xilinx Parameterized Macros (XPMs). Clock Domain Crossing logic uses the XPM_CDC macro, and the asynchronous FIFOs (Log FIFO and Command FIFO) use the XPM_FIFO macro. See the *UltraScale Architecture Libraries Guide* (UG974) [Ref 6] for more information on the XPM_CDC macros. These XPM_CDC macros were implemented:

- xpm_cdc_single
- xpm_cdc_array_single
- xpm_cdc_async_rst

At the time of this application note, the XPM_FIFO macro was in beta release for 2016.2. Therefore, this macro has some limitations. Examine the following XPM_FIFO Unsupported Features (2016.2 xpm_fifo beta release) and XPM_FIFO Supported Features (2016.2 xpm_fifo beta release) lists and refer to the `MPSoC_AXI2SEM/ip/xpm_fifo/limitations` directory for further information. Do not assume something is supported if it is not listed. Contact your FAE if you have any questions or concerns. The templates required to build the XPM_FIFO macro are provided in the ZIP file and must be enabled. Refer to Generate the PL Design for more details.

XPM_FIFO Unsupported Features (2016.2 xpm_fifo beta release)

- Asymmetric data width
- ECC
- DOUT reset value
- Programmable full/empty
- Data count
- Option for FULL to reset to 1

XPM_FIFO Supported Features (2016.2 xpm_fifo beta release)

- Native interface
- Clock domain (common, independent)
- Read mode (Standard, first word fall through (FWFT))
- Width 1–1024 (symmetric only)
- Depth up to 8192 (depth must be a power of 2)
- FIFO read latency—0, 1
  - Value 0 applies only for FWFT.
  - Value ≥ 1 applies only for Standard Read mode.
- Underflow/overflow

It is expected that the instantiation of this module will change with later Vivado tool releases. Therefore if targeting a version of Vivado later than 2016.2, this part of the code would need to be modified, and the templates for XPM_FIFO in this reference design would need to be disabled. Generate the PL Design shows how they were enabled.

The Log FIFO contains any SEM controller log reports such as the initialization, status, and correction reports. It has the following configuration:

```
FIFO_MEMORY_TYPE   => "bram",  --string; "auto", "bram", "lutram", "uram" or "builtin";
FIFO_WRITE_DEPTH   => 8192,              --positive integer
WRITE_DATA_WIDTH   => 8,                 --positive integer
READ_MODE          => "fwft",            --string; "std" or "fwft";
FIFO_READ_LATENCY  => 0,                 --positive integer; 0 or 1;
READ_DATA_WIDTH    => 8,                 --positive integer
CDC_SYNC_STAGES    => 2,                 --positive integer
WRCOUNT_TYPE       => "disable_wr_dc",   --do not change
PROG_FULL_THRESH   => 10,                --do not change
RDCOUNT_TYPE       => "disable_rd_dc"    --do not change
PROG_EMPTY_THRESH  => 10,                --do not change
DOUT_RESET_VALUE   => "0",               --do not change
RDCLK_FASTER       => 0,                 --do not change
ECC_MODE           => "no_ecc",          --do not change
EN_ECC_PIPE        => 0,                 --do not change
WAKEUP_TIME        => 0,                 --do not change
AUTO_SLEEP_TIME    => 0                  --do not change
```

The Command FIFO is used to send commands to the SEM controller. It has the following configuration:

```
FIFO_MEMORY_TYPE   => "bram",    --string; "auto", "bram", "lutram", "uram" or "builtin";
FIFO_WRITE_DEPTH   => 1024,            --positive integer
WRITE_DATA_WIDTH   => 8,               --positive integer
READ_MODE          => "fwft",          --string; "std" or "fwft";
FIFO_READ_LATENCY  => 0,               --positive integer; 0 or 1;
READ_DATA_WIDTH    => 8,               --positive integer
CDC_SYNC_STAGES    => 2,               --positive integer
WRCOUNT_TYPE       => "disable_wr_dc", --do not change
PROG_FULL_THRESH   => 10,              --do not change
RDCOUNT_TYPE       => "disable_rd_dc", --do not change
PROG_EMPTY_THRESH  => 10,              --do not change
DOUT_RESET_VALUE   => "0",             --do not change
RDCLK_FASTER       => 0,               --do not change
ECC_MODE           => "no_ecc",        --do not change
EN_ECC_PIPE        => 0,               --do not change
WAKEUP_TIME        => 0,               --do not change
AUTO_SLEEP_TIME    => 0                --do not change
```

send_ascii_fsm.vhd

This module sends SEM commands to the SEM controller. The command is in ASCII format and it is not initiated until one of the command bits (Command [10:0]) in the Command register is written. Each bit in the Command register is used to create a command in the same format as the SEM command. The commands are found in *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 5]. Each command is written to the Command FIFO one byte at a time. When the Command is initiated by writing a 1 to only one bit in Command[10:0], the write busy signal (Command[27]) asserts and remains asserted until the last byte of the command has been written to the command FIFO.

ascii_parser.vhd

This module is a wrapper file that instantiates the `init_check.vhd`, `crc_check.vhd`, and `rom_check.vhd` modules.

init_check.vhd

This module provides status information about the initialization of the SEM controller. It waits for SEM to have access to ICAP (icap_request = 1, icap_grant = 1, and icap_release = 0), then it checks for ASCII patterns in the initialization report. Using this sample initialization report, the order of checks performed are numbered:

```
SEM_ULTRA_V3_1
SC 01       1. Checks for SC
FS 04
AF 01
ICAP OK     2. Checks for OK
RDBK OK     3. Checks for OK
INIT OK     4. Checks for OK
SC 02
O>          5. Checks for 'I>' or 'O>' or 'D>'
```

Bits in the SEM Ready register are set accordingly as the FSM in this module advances through the checks. After SEM initialization completes, if SEM loses control of the ICAP, the process of checking the initialization report restarts. After initialization completes, the FSM also checks for the Soft Reset command (R XX where XX are don't cares). If the FSM detects the Soft Reset command followed by a carriage return, the process of checking the initialization report restarts.

rom_check.vhd

This module checks if the SEM controller has encountered a ROM error. It checks the ASCII characters received from the SEM controller for the ASCII representation of ROM (52 4F 4D). The check is performed before the ASCII characters are written to the Log FIFO. When the ROM pattern is matched, Error register [3] (ROM Detected) is asserted High. After it is asserted, this bit remains asserted until it is cleared. It can be cleared by writing a 1 to Error register [3].

crc_check.vhd

This module checks if the SEM controller has encountered an uncorrectable device-level CRC error. It checks the ASCII characters received from the SEM controller for the ASCII representation of CRC (43 52 43). The check is performed before the ASCII characters are written to the Log FIFO. When the CRC pattern is matched, Error Register [1] (CRC Detected) is asserted High. After it is asserted, this bit remains asserted. It cannot be cleared.

Software Application

Application Files

The software application for the hardware design is provided with this application note. The application is comprised of three files:

- `MPSoC_AXI2SEM/SDK/source/axi2sem_pl_intr_test.h`: This file provides the different offsets which are beneficial for accessing the AXI Interrupt controller registers, the AXI2SEM registers, and the AXI2SEM IP Interrupt controller registers.
- `MPSoC_AXI2SEM/SDK/source/axi2sem_pl_intr_test.c`: This is the main application that interfaces with the user via the terminal and the SEM controller via AXI4-Lite. This file contains comments to help understand the steps involved in creating the software application.
- `MPSoC_AXI2SEM/SDK/source/axi2sem_intr.c`: This is the AXI2SEM interrupt handler. It looks for the SEM Ready, Correction interrupt, Uncorrectable interrupt, and the Heartbeat Present interrupt. If other interrupts are enabled, all software files need to be modified to accommodate the additions. Upon detecting an interrupt, first the IPISR bit is cleared, then the AXI INTC IPR bit is cleared by writing to the AXI INTC IAR bit.

A full log for the application is provided in the application note ZIP file:

`MPSoC_AXI2SEM/teraterm_axi2sem_intc.log`

RECOMMENDED: *Become familiar with the Zynq UltraScale+ MPSoC Technical Reference Manual (UG1085) [Ref 7] and Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 8], which were used to create the application.*

Tool Flow and Verification

Table 3 lists the tool flow and verification procedures used for the provided reference design.

Table 3: **Reference Design Checklist**

Parameter	Description
General	
Developer Name	Xilinx
Target Devices	xczu9eg-ffvb1156
Source code provided?	Yes
Source code format (if provided)	VHDL, C
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	UltraScale Soft Error Mitigation (Beta) XPM_FIFO_ASYNC (2016.2 Beta) XPM_CDC_SINGLE XPM_CDC_ARRAY_SINGLE XPM_CDC_ASYNC_RST Zynq UltraScale+ MPSoC Processor System Reset AXI Interconnect AXI Interrupt Controller Utility Buffer Constant AXI4-Lite IPIF Interrupt Control
Simulation	
Functional simulation performed	No
Timing simulation performed?	No
Test bench provided for functional and timing simulation?	No
Test bench format	
Simulator software and version	
SPICE/IBIS simulations	No
Implementation	
Synthesis software tools/versions used	Vivado design suite 2016.2
Implementation software tool(s) and version	Vivado design suite 2016.2
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	Zynq UltraScale+ MPSoC ZCU102 Evaluation Board

Requirements

The design was developed on the following hardware and software. Refer to Limitations and Considerations for additional information regarding AXI2SEM, XPM_FIFO, SEM, and Targeting Other Software Versions.

Hardware

- HW-Z1-ZCU102 Rev B or newer
- ZU9EG silicon
- JTAG programming cable
- USB cable for the JTAG connection
- USB cable for the UART connection

Software

- Vivado Design Suite 2016.2
- Software Development Kit (SDK) 2016.2
- Tera Term 4.83 for serial UART connection

Reference Design Files

The top-level reference design file, `xapp1303-integrating-sem-ip-with-axi.zip`, contains one reference design. The directory structure for the reference design is as follows.

Directory Structure

```
MPSoC_AXI2SEM
|-- teraterm_axi2sem_intc.log
|      - Full log file from running the SW Application
|-- readme.txt
|      - This file
|-- ip
|   |-- sem_ultra_v3_1              <-- Packaged IP required for ZU9EG support
|   `-- xpm_fifo                    <-- In Beta release - see limitations
|       |-- limitations
|       |   |-- LIMITATIONS_XPM_FIFO.txt
|       |   `-- XPM_EA_Cust_r3.pptx
|       `-- templates
|           |-- debug.xml
|           |-- index.html
|           |-- systemverilog.xml
|           |-- template.xsl
|           |-- verilog.xml
|           |-- vhdl.xml
|           `-- xdc.xml
|-- SDK
|   |-- boot_img
|   |   |-- axi2sem_intc_a53_0.elf
|   |   |-- BOOT.bin
|   |   |-- fsbl_a53_0.elf
|   |   |-- output.bif
|   |   `-- zu9eg_sem_wrapper.bit
|   `-- source
|       |-- axi2sem_intr.c
|       |-- axi2sem_pl_intr_test.c
|       `-- axi2sem_pl_intr_test.h
`-- Vivado
    |-- hdl
    |   |-- axi_lite_ipif_v3_0      <-- VHDL source provided in XAPP
    |   |-- axi2sem
    |   |   |-- constraints
    |   |   |   `-- zu9eg_sem.xdc
    |   |   `-- axi2sem
    |   |       |-- ascii_parser.vhd
    |   |       |-- axi2sem.vhd
    |   |       |-- axi2sem_cdc.vhd
    |   |       |-- axi2sem_core.vhd
    |   |       |-- axi2sem_reg_intf.vhd
    |   |       |-- crc_check.vhd
    |   |       |-- init_check.vhd
    |   |       |-- rom_check.vhd
    |   |       `-- send_ascii_fsm.vhd
    |   `-- interrupt_control_v3_1  <-- VHDL source provided in XAPP
    `-- tcl
        |-- board_repository.tcl
        |-- ip_repository.tcl
        `-- source_hdl.tcl
```

ip

This directory contains the SEM UltraScale+ family packaged design (sem_ultra_v3_1) that supports the ZU9EG device on the ZCU102 board.

SDK

- SDK/boot_img
  - BOOT.bin - Boot image file that contains the ELF and BIT files
  - fsbl_a53_0.elf - Contains the first stage boot loader software application data
  - output.bif - SDK generated file that references the files necessary to create the BOOT image
  - axi2sem_intc_a53_0.elf - Contains the AXI2SEM software application data
  - zu9eg_sem_wrapper.bit
- SDK/source
  - axi2sem_pl_intr_test.h - Header file containing the interrupt masks and address offsets for accessing the different AXI2SEM registers
  - axi2sem_pl_intr_test.c - Software application for exercising this reference design
  - axi2sem_intr.c - Interrupt handler for the software application

Vivado

This directory contains the VHDL source files for AXI2SEM and its instantiated submodules (axi_lite_ipif_v3_0 and interrupt_control_v3_1). It also contains Tcl files that can help when creating the reference design from scratch.

- hdl/axi_lite_ipif_v3_0 - The source code for the AX4-Lite IP interface (IPIF) is provided in this directory. This is instantiated in the AXI2SEM module. Refer to *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155) [Ref 3] for more details.
- hdl/axi2sem - Contains all of the VHDL source files for the AXI2SEM module:
  - ascii_parser.vhd
  - axi2sem.vhd
  - axi2sem_cdc.vhd
  - axi2sem_core.vhd
  - axi2sem_reg_intf.vhd
  - crc_check.vhd
  - init_check.vhd
  - rom_check.vhd
  - send_ascii_fsm.vhd
- hdl/interrupt_control_v3_1 - The source code for the IP interrupt controller which is instantiated in the AXI2EM module. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) [Ref 2] for more details.

Three Tcl files are provided for reference. These files can be modified to accommodate a different directory structure, then sourced in the Vivado Tcl console when creating the reference design from scratch.

- `tcl/board_repository.tcl` - Contains the command used to set the path for the board files downloaded from the Zynq UltraScale+ ZCU102 HeadStart website.
- `tcl/ip_repository.tcl` - Contains the commands to set the SEM controller repository from the build to a local repository
- `tcl/source_hdl.tcl` - Contains the Tcl commands to add, import, and assign the axi_lite_ipif_v3_0, axi2sem, and interrupt_control_v3_1 designs to separate VHDL libraries in the Vivado tools.

Licensing

SEM Controller Licensing

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property website. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Reference Design Steps

Setup

Setup instructions are included in the following sections. If you are creating the reference design from scratch, refer to Creating the Reference Design. However, to run the reference design with the supplied ZIP files, refer to Running the Reference Design. This section also contains any other setup information required to run the reference design.

Creating the Reference Design

These instructions describe how to generate the reference design from scratch. Refer to axi2sem_cdc.vhd for XPM_FIFO limitations. Follow these steps to create the reference design.

Download ZCU102 Board Files for Vivado Tool Release 2016.2

This file download procedure is required for the design.

Note: The ZCU102 board files are not released with Vivado Design Suite 2016.2, and they are not provided in this application note. The board files must be downloaded from the Zynq UltraScale+ ZCU102 HeadStart website.

IMPORTANT: *It could take up to two days to access the files after you register.*

1. Go to the Zynq UltraScale+ ZCU102 HeadStart website.
2. Click the **Documentations and Designs** tab.
3. Scroll down to the **ZCU102 ES1/ES2 Board Files.**
4. Click the **ZCU102 Board Files** link for files with the description: ZCU102 2016.2 board files.

   Note: Use only 2016.2 board files.
5. If you agree to the license agreement, select **I Accept**.
6. Download the design file and extract the contents.

Generate the PL Design

These instructions describe how to generate the PL bitstream of the reference design from scratch. Unzip the `xapp1303-integrating-sem-ip-with-axi.zip` file. See the `readme.txt` file in the MPSoC_AXI2SEM folder.

Create Project

Note: `<path> = C:/Projects` in the following steps.

1. Change directory to `MPSoC_AXI2SEM`:

   ```
   cd <path>/MPSoC_AXI2SEM
   ```
2. Open the Vivado Design Suite.
3. Before creating a project, in the Tcl console, set the path to the ZCU102 board repository files and enable the Beta XPM FIFO templates. For example:

   a. `set_param board.repoPaths C:/Projects/zcu102`

      (source `C:/Projects/MPSoC_AXI2SEM/Vivado/tcl/board_repository.tcl`)

   b. `set_param templates.setDir`

      `<path>/MPSoC_AXI2SEM/ip/xpm_fifo/templates`
4. Create a new project and configure it as follows:
   a. Project Name: **zu9eg_sem**
   b. Project Location: **<path>/MPSoC_AXI2SEM**. Select **Next**.
   c. Select **RTL Project** and check **Do not specify sources at this time**. Select **Next**.

d. Select **Boards**, set **Xilinx** as the vendor, and select **Zynq UltraScale+ ZCU102 Evaluation Board**. See Figure 4.

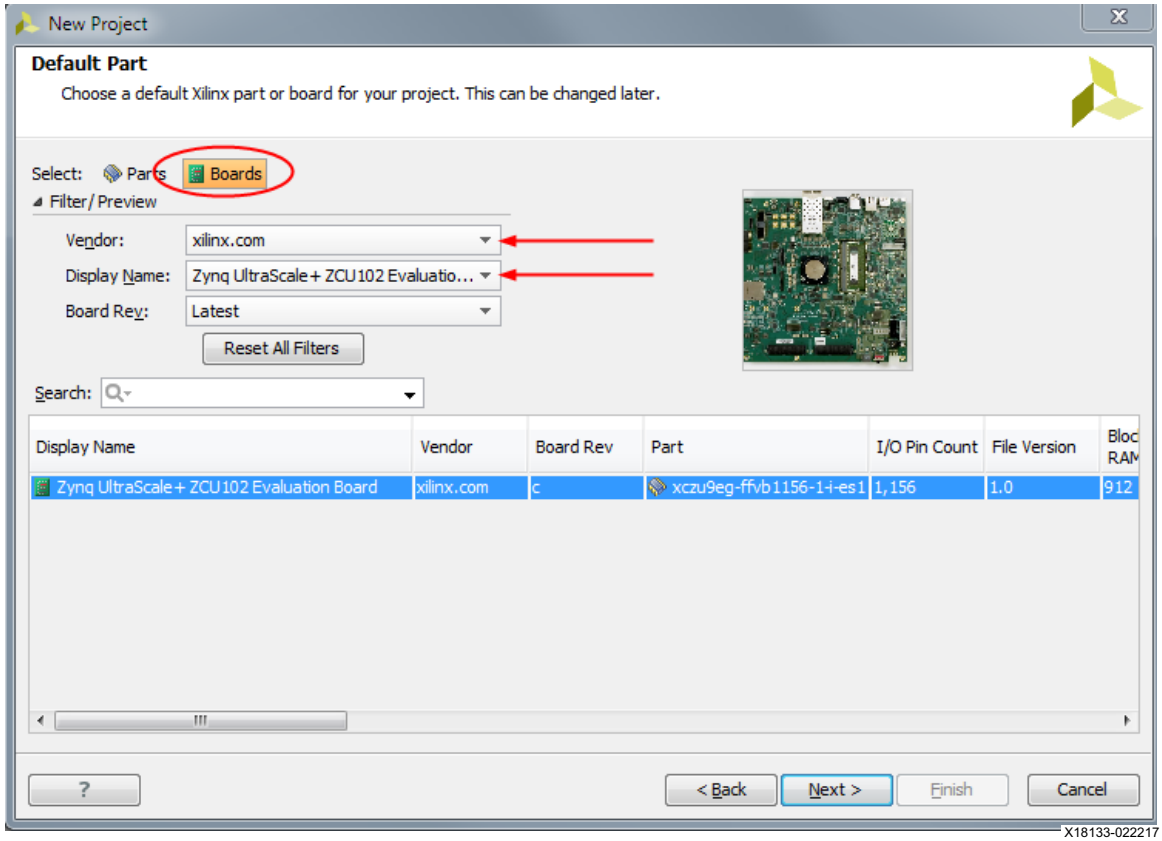

Figure 4: **Board Selection**

e. Click **Next**, then **Finish**.

5. After the project opens, in the Vivado Tcl console, set the IP Repository path for the SEM controller that supports the ZU9EG device. Also enable the XPM_CDC and XPM_FIFO macros:

a. `set_property ip_repo_paths C:/Projects/MPSoC_AXI2SEM/ip/sem_ultra_v3_1 [current_project]`

b. `update_ip_catalog -rebuild`

Ignore this duplicate IP warning:

```
WARNING: [IP_Flow 19-1663] Duplicate IP found for
xilinx.com:ip:sem_ultra:3.1. The one found in IP location
c:/Projects/MPSoC_AXI2SEM/sem_ultra_v3_1 will take precedence
over the same IP in the Xilinx installed IP.
```

c. `set_property XPM_LIBRARIES {XPM_CDC XPM_FIFO} [current_project]`

(source `C:/Projects/MPSoC_AXI2SEM/Vivado/tcl/ip_repository.tcl`)

Add Files

1. Add AXI2SEM, IP Interrupt control, AXI_LITE_IPIF source files, and set libraries to the project.

   To do this, modify the Tcl file and source it in the Vivado tools, or modify the following commands for the path information. For example, enter the following command in the Vivado Tcl console:

```
source C:/Projects/MPSoC_AXI2SEM/Vivado/tcl/source_hdl.tcl
```

a.
```
add_files -norecurse -scan_for_includes
{<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/crc_check.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem_core.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/init_check.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/ascii_parser.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem_cdc.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/
axi2sem_reg_intf.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/send_ascii_fsm.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/rom_check.vhd}

import_files -force -norecurse
{<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/crc_check.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem_core.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/init_check.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/ascii_parser.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem_cdc.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/axi2sem_reg_intf.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/send_ascii_fsm.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/source/rom_check.vhd}

set_property library axi2sem_v0_0 [get_files
{<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/axi2sem_reg_intf.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/send_ascii_fsm.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/rom_check.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/axi2sem.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/crc_check.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/axi2sem_core.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/init_check.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/ascii_parser.vhd
```

```
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
source/axi2sem_cdc.vhd}]
```

b.
```
add_files -norecurse -scan_for_includes
<path>/MPSoC_AXI2SEM/Vivado/hdl/interrupt_control_v3_1/hdl/src/
vhdl/interrupt_control.vhd

import_files -force -norecurse
<path>/MPSoC_AXI2SEM/Vivado/hdl/interrupt_control_v3_1/hdl/src/
vhdl/interrupt_control.vhd

set_property library interrupt_control_v3_1_3 [get_files
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/interrupt_control.vhd]
```

c.
```
add_files {<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/
hdl/src/vhdl/pselect_f.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
address_decoder.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
slave_attachment.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
ipif_pkg.vhd}

import_files -force -norecurse
{<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
axi_lite_ipif.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
pselect_f.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
address_decoder.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
slave_attachment.vhd
<path>/MPSoC_AXI2SEM/Vivado/hdl/axi_lite_ipif_v3_0/hdl/src/vhdl/
ipif_pkg.vhd}

set_property library axi_lite_ipif_v3_0_3 [get_files
{<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/axi_lite_ipif.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/pselect_f.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/address_decoder.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/slave_attachment.vhd
<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.srcs/sources_1/imports/
vhdl/ipif_pkg.vhd}]
```

2. Add constraints to the project.

   a. `add_files -fileset constrs_1 -norecurse <path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/constraints/zu9eg_sem.xdc`

   b. `import_files -fileset constrs_1 -force <path>/MPSoC_AXI2SEM/Vivado/hdl/axi2sem/constraints/zu9eg_sem.xdc`

3. Select **Create Block Design** and enter **zu9eg_sem** as the Design name in the dialog box.

Add IP to the Diagram

1. In the Vivado **Diagram** tab, right-click **Add IP** and select **Zynq UltraScale+ MPSoC.**

   a. Select **Run Block Automation**. See Figure 5.

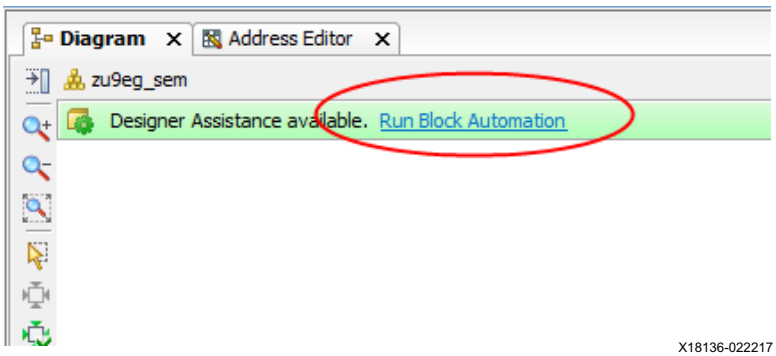

Figure 5: **Run Block Automation**

   b. Verify **Apply Board Presets** is selected. Then select **OK**.

   c. Double-click the Zynq UltraScale+ MPSoC block and configure it as follows:

   d. Set the **Clock Configuration > Low Power Domain Clocks > PL Fabric Clocks > PL0 clock** to **100 MHz**. See Figure 6.

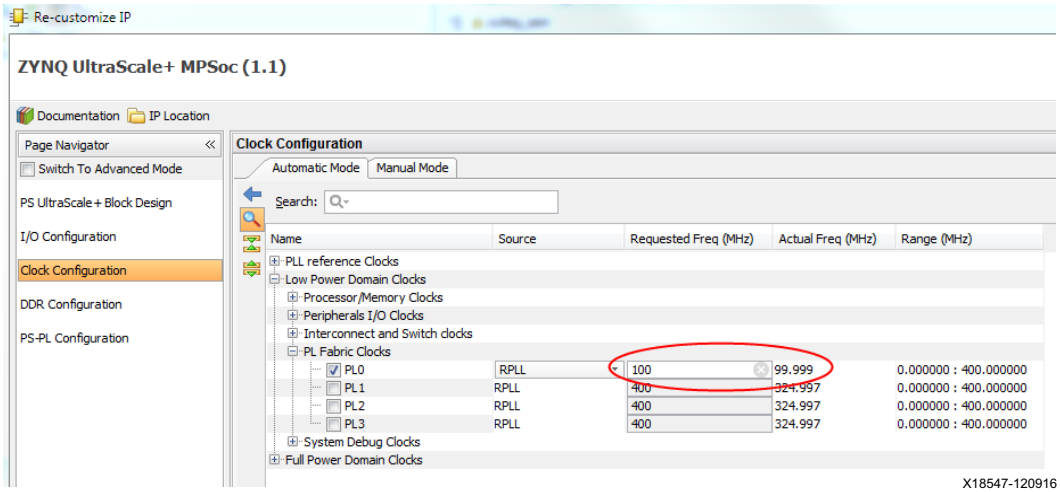

Figure 6: **Recustomize IP - Clock Configuration**

e. Set the **PS-PL Configuration > General > Interrupts > PL to PS > IRQ0[0:7]** to **1.** See Figure 7.

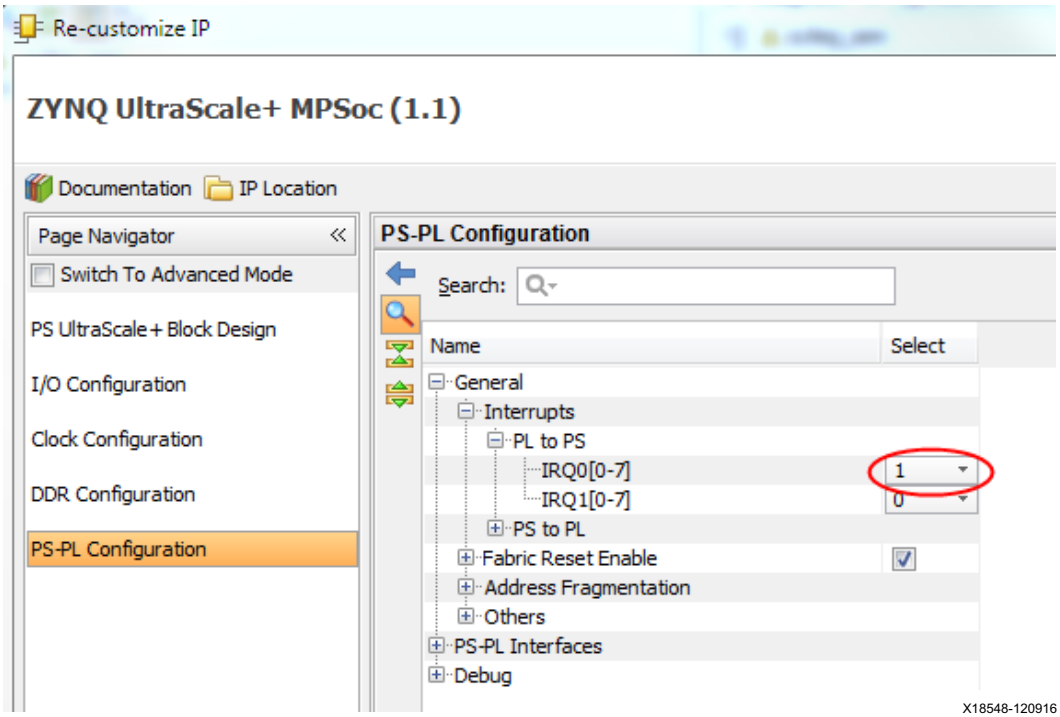

Figure 7: **Recustomize IP - PL to PS Configuration**

2. Right-click on an empty area of the **Diagram** tab and select **Add IP.**
   a. Type **Processor System Reset** in the search box, and select it. Use the default settings.
3. Right-click on an empty area of the **Diagram** tab and select **Add IP.**
   a. Type **AXI Interconnect** in the search box, and select it. Use the default settings.
4. Right-click on an empty area of the **Diagram** tab and select **Add IP**.
   a. Type **Utility Buffer** in the search box, and select it.
   b. Click each input pin to highlight it, right-click, and select **Make External.**
   c. Rename IBUF_DS_P to **icap_clk_p** (click to highlight the pin name in the **Diagram**, and edit the name in the External Port Properties window). See Figure 8.

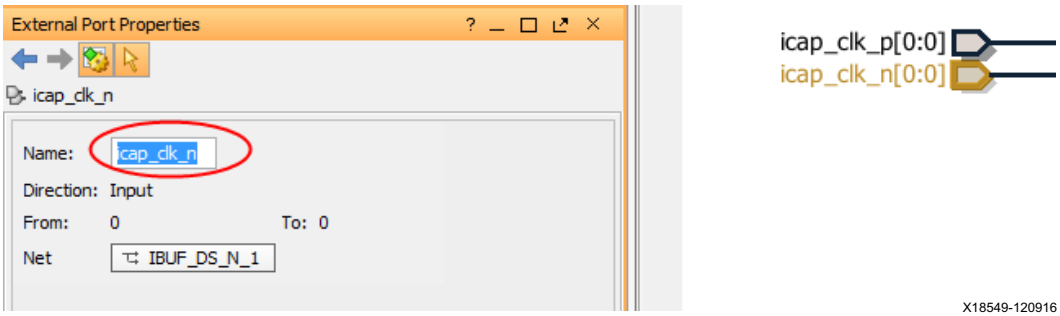

Figure 8: **External Port Properties - icap_clk_p**

d. Rename IBUF_DS_N to **icap_clk_n** (click to highlight the pin name in the **Diagram**, and edit the name in the External Port Properties window). See Figure 9.

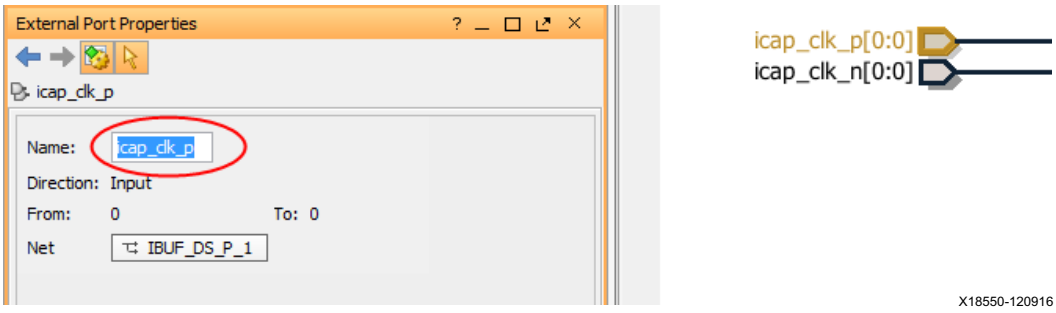

Figure 9: **External Port Properties - icap_clk_n**

5. Right-click on an empty area of the **Diagram** tab and select **Add IP**.
   a. Type **AXI Interrupt Controller** in the search box, and select it.
   b. Double-click the AXI Interrupt Controller icon. Change Interrupt Output Connection to **Single**.
   c. Click the **irq** pin to highlight it, right-click, and select **Make External.**
6. Right-click an empty area of the **Diagram** tab and select **Add IP.**
   a. Type **Ultrascale Soft Error Mitigation** in the search box, and select it.
   b. In the Duplicate IP window, click **Add Active IP**.
   c. Double-click the icon. Select **Mitigation and Testing** from the drop-down list. Ensure the Controller Clock Period is set to **13468 ns (74.25 MHz).** Click **OK**.
   d. Click the **cap_req** pin to highlight it, right-click, and select **Make External.**
7. Right-click on an empty area of the **Diagram** tab and select **Add Module**.
   a. Type **axi2sem.vhd** in the search box, and select it.
   b. Click each pin below to highlight it, right-click, and select **Make External**:
      - **cdc_icap_grant**
      - **cdc_icap_release**
8. Right-click on an empty area of the **Diagram** tab and select **Add IP**.
   a. Type **Constant** in the search box and select it.
   b. Configure it to be **1-bit wide** and to output **0**. (Connects to UltraScale Soft Error Mitigation `command_strobe`, `aux_error_cr_ne`, `aux_error_cr_es`, `aux_error_uc`, and Processor System Reset `mb_debug_sys_rst` signals.)
9. Right-click on an empty area of the **Diagram** tab and select **Add IP**.
   a. Type **Constant** in the search box and select it.
   b. Configure it to be 1-bit wide and to output 1. (Connects to the processor reset block `ext_reset_in` and `dcm_locked` signals.)

10. Right-click on an empty area of the **Diagram** tab and select **Add IP**.
    a. Type **Constant** in the search box and select it.
    b. Configure it to be **44-bits wide** and to output **0**. (Connects to UltraScale Soft Error Mitigation `command_code` signal.)

Make Final Diagram Connections

1. Connect the design as shown in Figure 2, Vivado IP Integrator Diagram.
2. In the **Address Editor** tab, drag and drop **axi_intc_0** so it appears under **Data**.
    a. Repeat for **axi2sem_0**. See Figure 10.

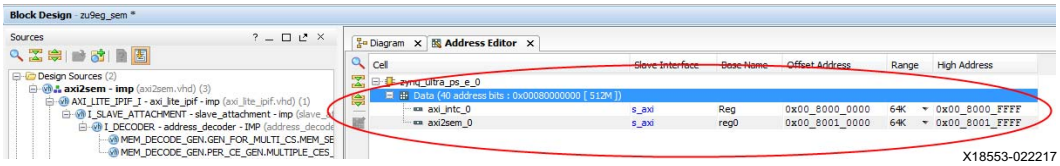

Figure 10: **Address Editor**

3. Right-click on an empty area of the **Diagram** tab and select **Validate Design**.
    a. Select **OK** in the pop-up window.

    Ignore the following warnings:

    - `WARNING: [BD 17-145] Zynq UltraScale IP doesn't support simulation`
    - `WARNING: [BD 41-702] Propagation TCL tries to overwrite USER strength parameter CLOCK_PERIOD(13468) on '/sem_ultra_0' with propagated value(10000). Command ignored`
    - `WARNING: [xilinx.com:ip:axi_intc:4.1-4] /axi_intc_0: Could not determine interrupt input port type - using default interrupt type Rising Edge. Please change this manually if necessary.`
    - `WARNING: [BD 41-237] Bus Interface property AWUSER_WIDTH does not match between /zynq_ultra_ps_e_0_axi_periph/s00_couplers/auto_ds/S_AXI(0) and /zynq_ultra_ps_e_0/M_AXI_HPM0_LPD(16)`
    - `WARNING: [BD 41-237] Bus Interface property ARUSER_WIDTH does not match between /zynq_ultra_ps_e_0_axi_periph/s00_couplers/auto_ds/S_AXI(0) and /zynq_ultra_ps_e_0/M_AXI_HPM0_LPD(16)`

4. In the **Hierarchy > Sources** tab, right-click **zu9eg_sem (zu9eg_sem.bd)** and select **Create HDL Wrapper**.
    a. Select **Let Vivado manage wrapper and auto-update** in the pop-up window. Click **OK**.
5. Right-click **zu9eg_sem_wrapper.v** and select **Set as Top**.

6. Right-click in an open area of the **Block Design** and select **Validate Design**.
   a. Select **Rerun Validate** Design (step 3).

Generate Block Design and Bitstream

1. Select **Generate Block Design** in the Flow Navigator.
   a. Select **Generate**. See Figure 11.

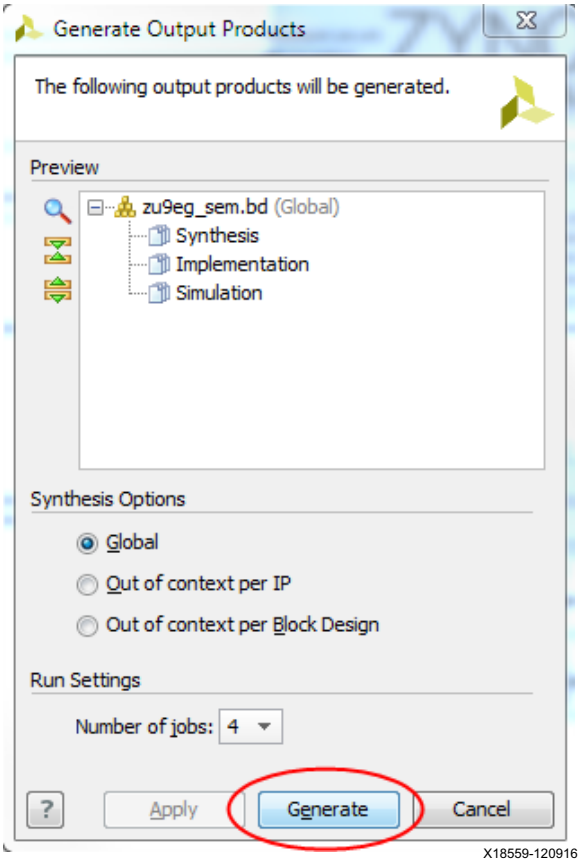

Figure 11: **Generate Output Products**

2. In the Flow Navigator, select **Generate Bitstream**, open the implemented design, and verify all timing was met.

   Note: The XPM_CDC Critical Warnings listed in Figure 12 do not affect the functionality of the design and can be safely ignored:

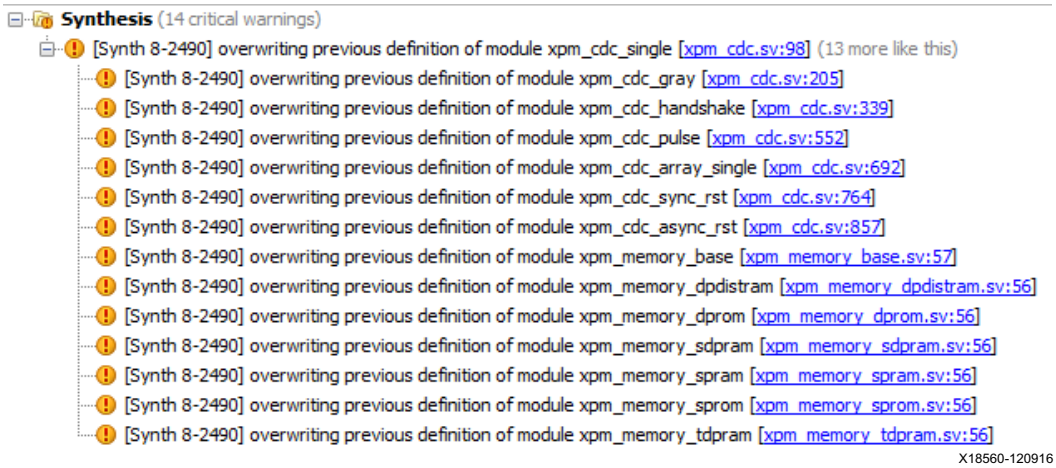

Figure 12: **Ignored XPM_CDC Critical Warnings**

3. To export the bitstream, select **File > Export > Export Hardware.** Check **Include bitstream**.
4. Select **File > Launch SDK**.

This launches the SDK where the Software application can be created. Proceed to Generate the PS Software Application.

Generate the PS Software Application

These instructions describe how to generate the software application portion of the reference design from scratch.

Note: `<path> = C:/Projects` in the following steps.

1. Create the FSBL Application Project by selecting **File > New > Application Project**.
   a. Enter **fsbl_a53_0** as the **Project Name.** Click **Next**.
   b. Select **Zynq MP FSBL.** Click **Finish**. See Figure 13.

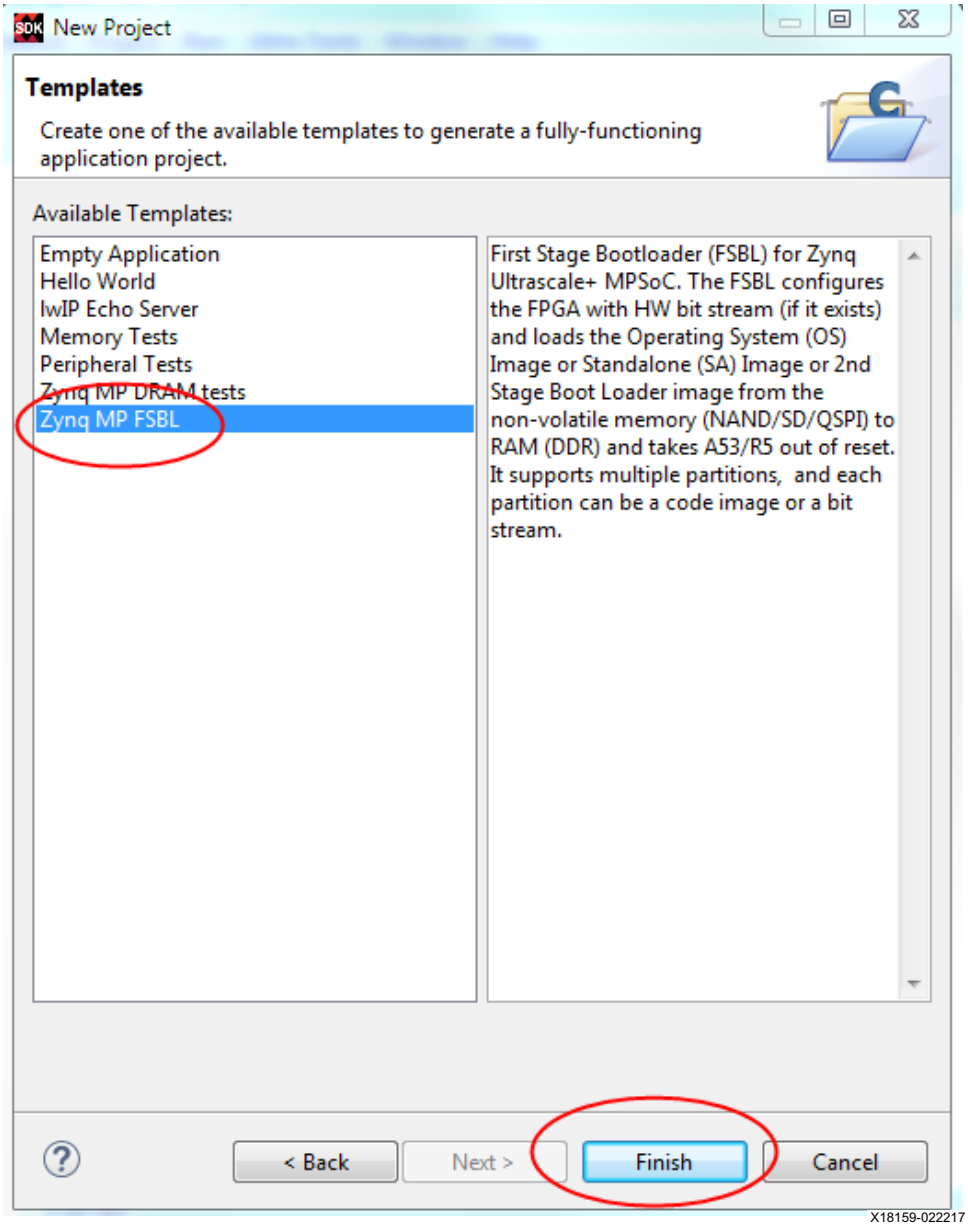

Figure 13: **Zynq MP FSBL Template**

2. Create the Application Project.
   a. **File > New > Application Project**.
   b. Enter **axi2sem_intc_a53_0** as the **Project Name.** Click **Next**.
   c. Select **Hello World**. Click **Finish**.
   d. Right-click **helloworld.c** and select **Delete** in the project explorer. See Figure 14.

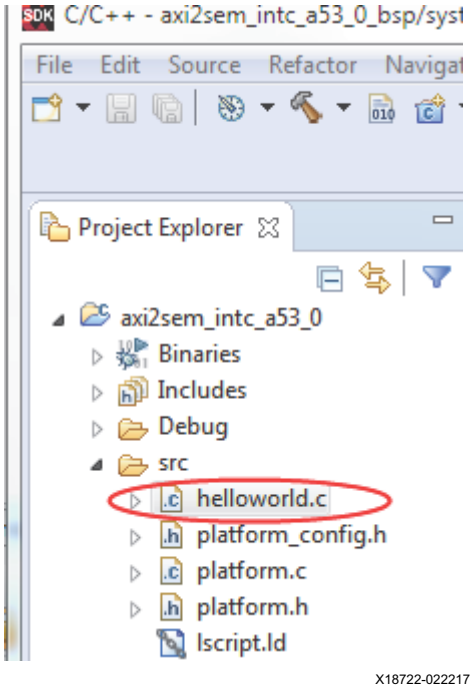

Figure 14: **Delete Hello World**

   e. Copy `axi2sem_intr.c` from `<path>/MPSoC_AXI2SEM/SDK/source/axi2sem_intr.c` to `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/axi2sem_intc_a53_0/src/axi2sem_intr.c`.
   f. Copy `axi2sem_pl_intr_test.c` from `<path>/MPSoC_AXI2SEM/SDK/source/axi2sem_pl_intr_test.c` to `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/axi2sem_intc_a53_0/src/axi2sem_pl_intr_test.c`.
   g. Copy `axi2sem_pl_intr_test.h` from `<path>/MPSoC_AXI2SEM/SDK/source/axi2sem_pl_intr_test.h` to `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/axi2sem_intc_a53_0/src/axi2sem_pl_intr_test.h`.

3. Right-click the **src** directory. Select **Refresh** to make the copied files appear. See Figure 15.

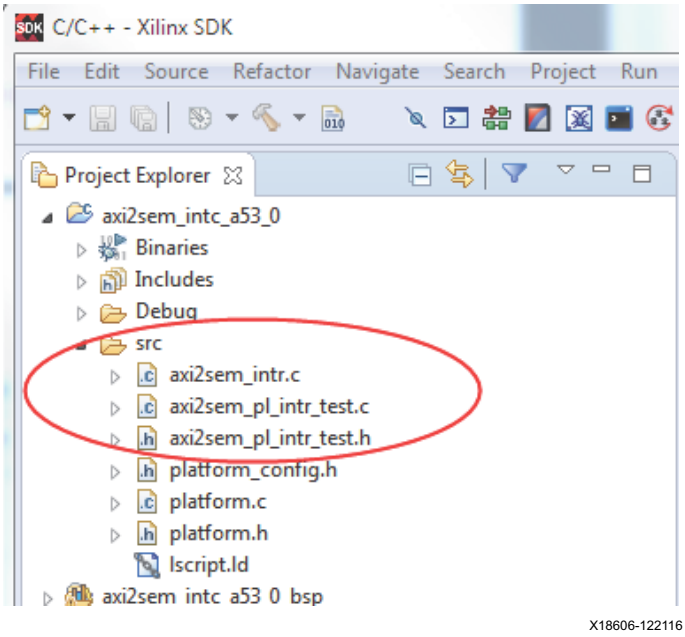

Figure 15: **Project Explorer**

If the following error is generated:

'XPAR_FABRIC_AXI_INTC_0_IRQ_INTR' undeclared (first use in this function)

Add the following lines to the `xparameters.h file`:

```
/* Definitions for Fabric interrupts connected to psu_acpu_gic */
#define XPAR_FABRIC_AXI_INTC_0_IRQ_INTR 121
```

Create the Boot Image

1. Copy the FSBL ELF file to the `boot_img` directory from `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/fsbl_a53_0/Debug/fsbl_a53_0.elf`

   to `<path>/MPSoC_AXI2SEM/SDK/boot_img/fsbl_a53_0.elf`.

2. Copy the BIT file to the `boot_img` directory, from `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/zu9eg_sem_wrapper_hw_platform_0/zu9eg_sem_wrapper.bit`

   to `<path>/MPSoC_AXI2SEM/SDK/boot_img/zu9eg_sem_wrapper.bit`.

3. Copy the `axi2sem_intc_a53_0` ELF file to the `boot_img` directory, from `<path>/MPSoC_AXI2SEM/zu9eg_sem/zu9eg_sem.sdk/axi2sem_intc_a53_0/Debug/axi2sem_intc_a53_0.elf`

   to `<path>/MPSoC_AXI2SEM/SDK/boot_img/axi2sem_intc_a53_0.elf`.

4. Select **Xilinx Tools > Create Boot Image**. Three partitions are added:
   - `fsbl_a53_0.elf`
   - `zu9eg_sem_wrapper.bit`
   - `axi2sem_intc_a53_0.elf`
5. Change Architecture to **Zynq MP**.
6. In the **Create Boot Image** window, set Output BIF file path to `<path>/MPSoC_AXI2SEM/SDK/boot_img/output.bif`.
   a. Verify the Output path is the same path. See Figure 16.

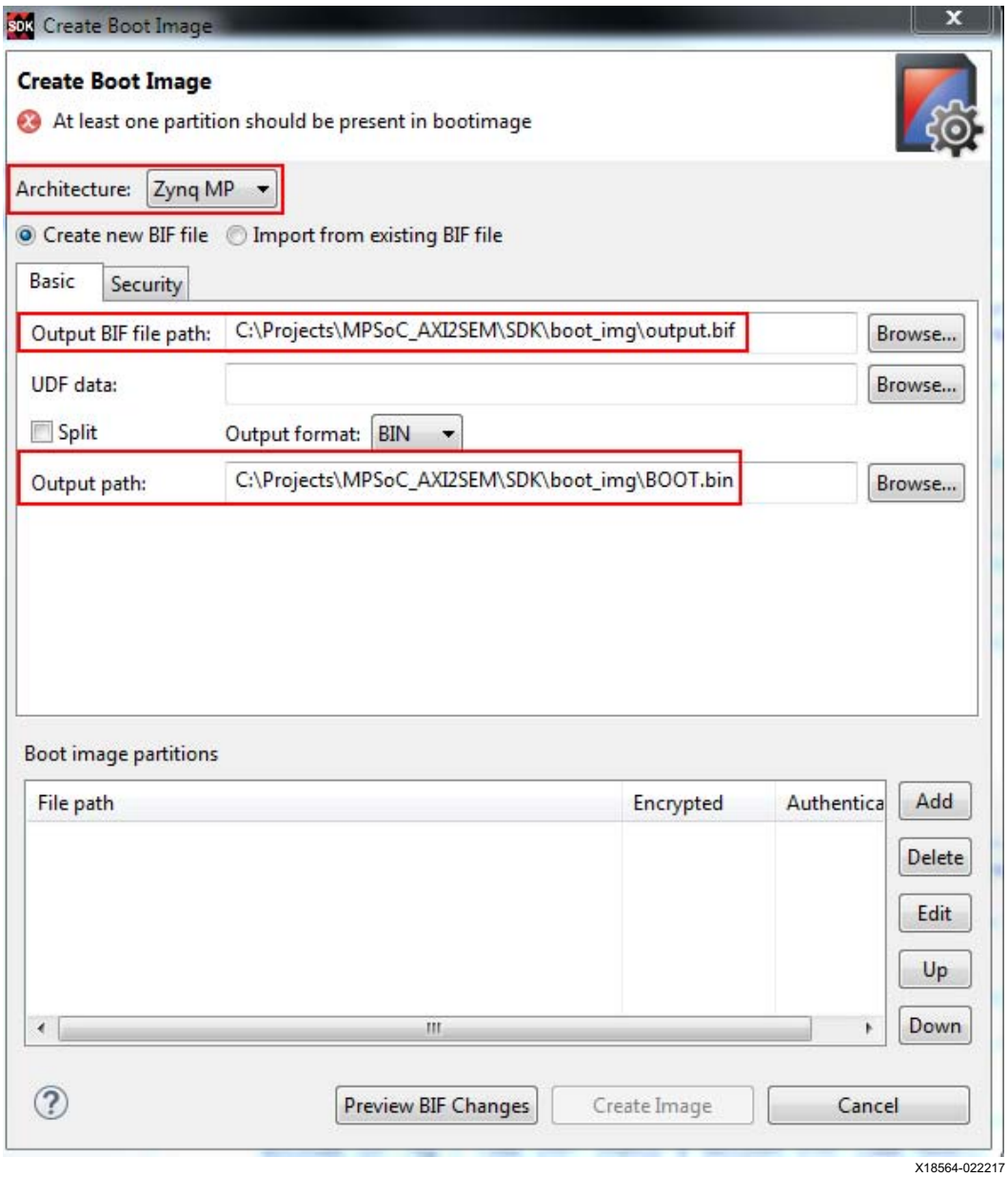

Figure 16: **Create Boot Image**

7. Add the partitions by selecting **Add**.
   a. Add the `fsbl_a53_0.elf` partition by selecting **Add** in the **Create Boot Image** window.
   b. In the **Add Partition** window, set File path to `<path>/MPSoC_AXI2SEM/SDK/boot_img/fsbl_a53_0.elf`. Click **OK**.
   c. Verify the Partition Type is **bootloader**, the Destination Device is **PS**, and the Destination CPU is **A53 x64**. See Figure 17.

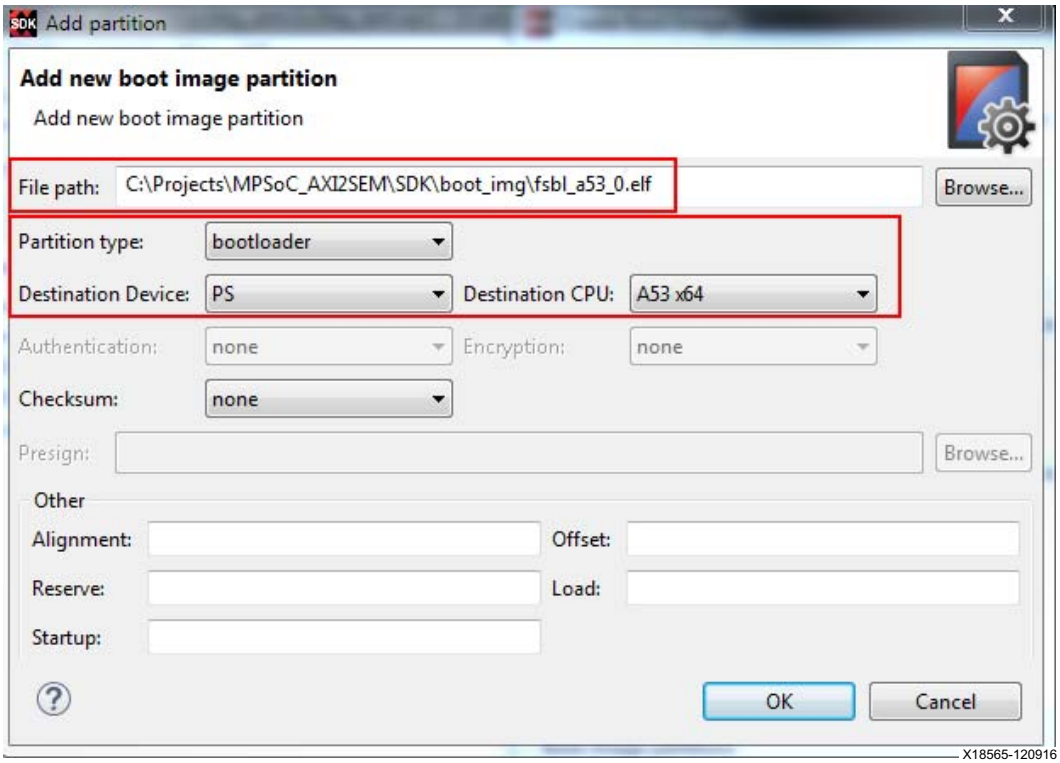

Figure 17: **FSBL Boot Partition**

8. Add the BIT file to the second partition by selecting **Add** in the **Create Boot Image** window.
   a. In the **Add Partition** window, set File path to `<path>/MPSoC_AXI2SEM/SDK/boot_img/zu9eg_sem_wrapper.bit`. Select **OK**.
   b. Verify the Partition Type is **datafile** and the Destination Device is **PL**. See Figure 18.

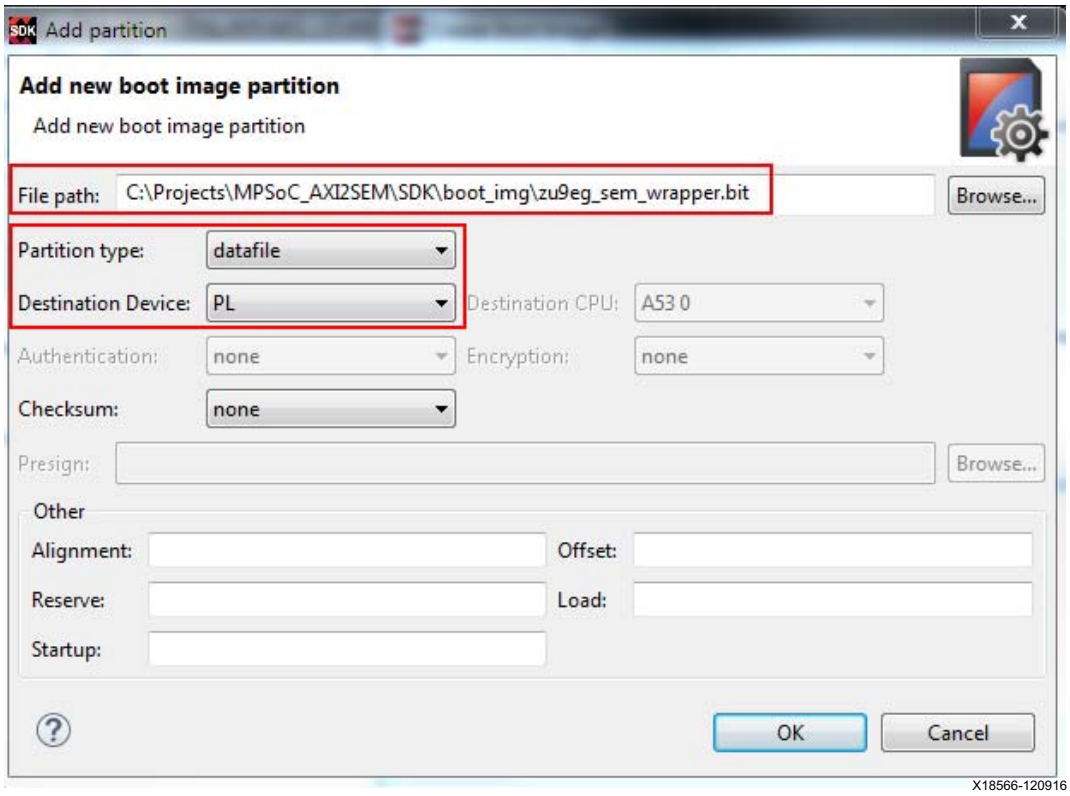

Figure 18: **BIT Partition**

9. Add the `axi2sem_intc_a53_0.elf` file to the third partition by selecting **Add** in the **Create Boot Image** window.
   a. In the **Add Partition** window, set File path to `<path>/MPSoC_AXI2SEM/SDK/boot_img/axi2sem_intc_a53_0.elf`. Select **OK**.
   b. Verify the Partition Type is **datafile**, the Destination Device is **PS**, and the Destination CPU is **A53 0**. See Figure 19.

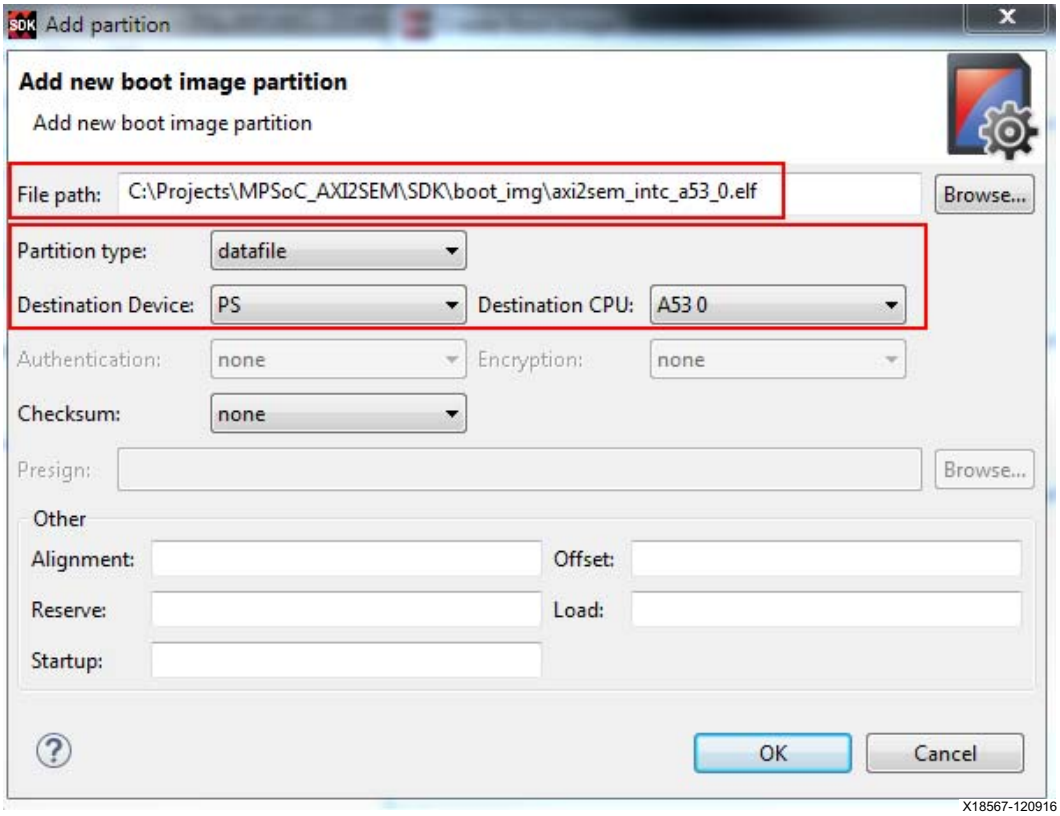

Figure 19: **AXI2SEM Boot Partition**

10. Verify the Boot image partitions. See Figure 20. Select **Create Image**.

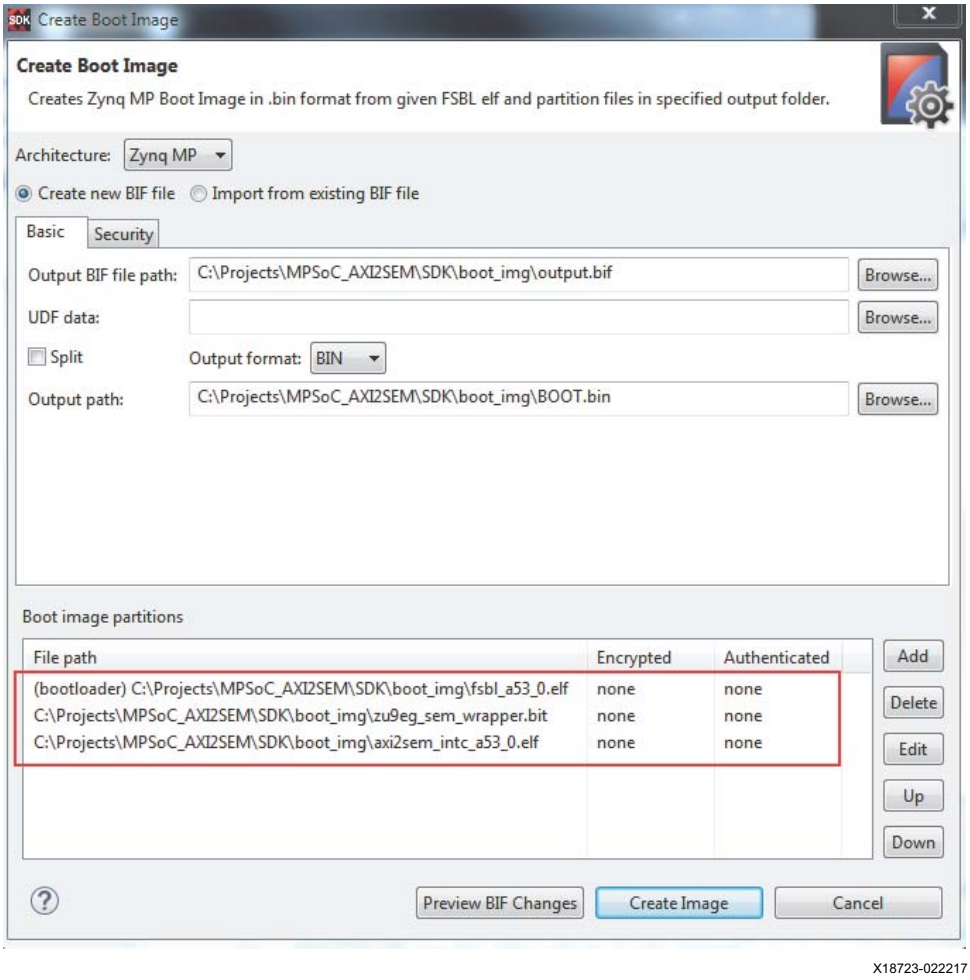

Figure 20: **Verify Boot Image Partitions**

The reference design can be run with the created design files. Proceed to Running the Reference Design step 2. **Set up the board**.

Running the Reference Design

This section provides a guide for downloading the image files through JTAG to the ZCU102 board. It also shows how to use the PS to initialize the SEM controller and interface with the SEM controller via the AXI4-Lite interface. The PS UART terminal is used to display SEM status information.

1. **Unzip the reference design**: Unzip the `xapp1303-integrating-sem-ip-with-axi.zip` file. See the `readme.txt` file in the MPSoC_AXI2SEM folder.
2. **Set up the board**: Connect the power, USB UART cable, and JTAG programming cable to the board. Set MODE[3:0] pins to ON ON ON ON (SW6). Note that for this DIP switch, in relation to the arrow, moving the switch toward the label ON is a 0. DIP switch labels 1 through 4 are equivalent to Mode pins 0 through 3. See Figure 21 and Figure 22. Apply power.

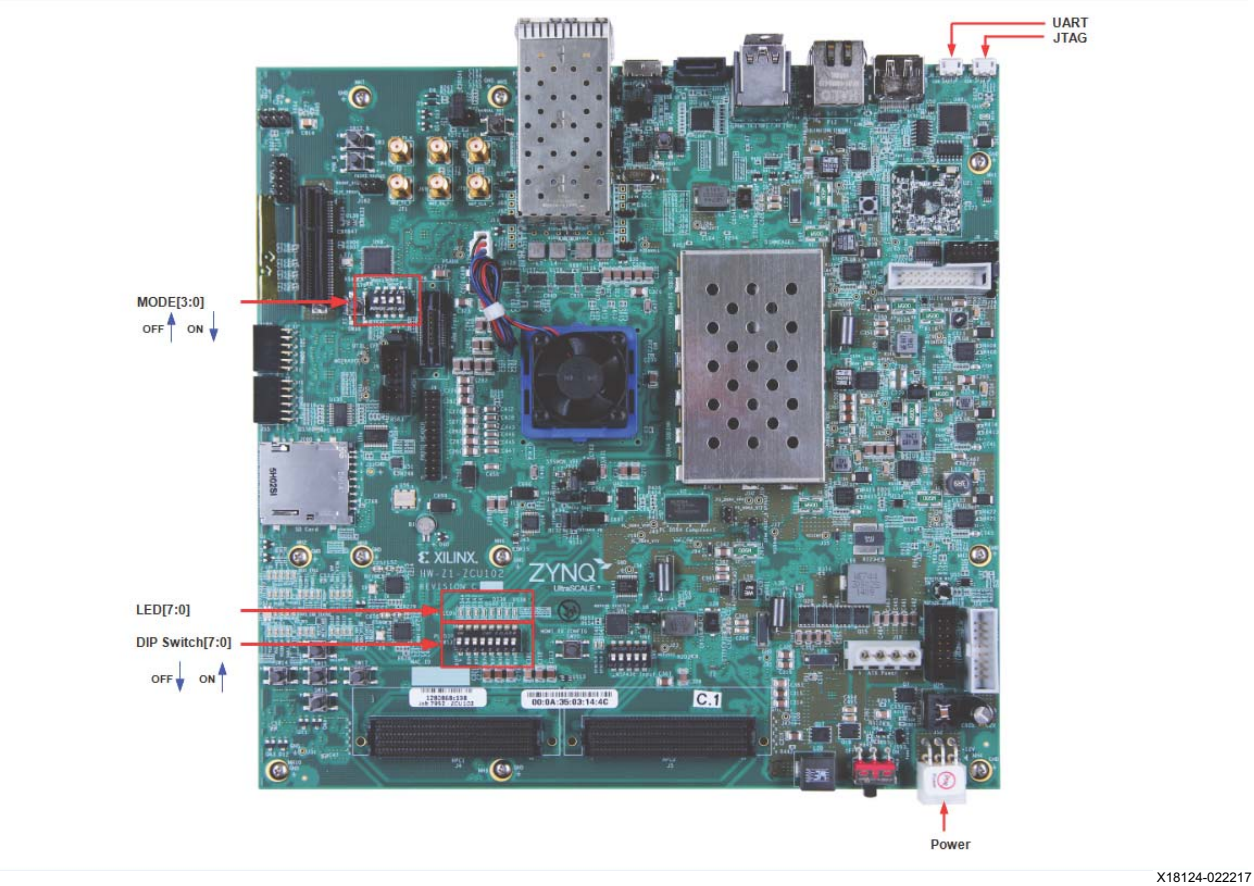

Figure 21: **ZCU102 Board—MODE Pins and DIP Switches**

Figure 22: **MODE[3:0] DIP Switch Setting (ON ON ON ON)**

IMPORTANT: *Ensure the ZCU102 board is powered on, the USB UART cable is connected, the JTAG cable is connected, and the JTAG cable drivers have been installed. See step 3.*

3. Set up the UART driver and select the COM port.
   a. Open two Tera Term terminals and set up a serial connection with these settings:

Baud	115200
Settings	Data 8; Parity None; Stop 1
Flow Control	None
Terminal ID	VT100
New-line Transmit	CR (terminal transmits CR as end of line)
New-line Receive	CR + LF (terminal receives CR as end of line and expands to CR + LF)
Local Echo	No

RECOMMENDED: *The Silicon Labs USB UART drivers might need to be installed to get this connection to work for the first time. For more information, see the Silicon Labs CP210x USB-to-UART Installation Guide (UG1033) [Ref 9].*

b. In Tera Term, select the **COM XX** port associated with the **Silicon Labs Quad CP210x USB to UART Bridge: Interface 0 (COMXX)** for the PS UART in the **Device Manager**, where **COM XX** depends on your computer. See Figure 23.

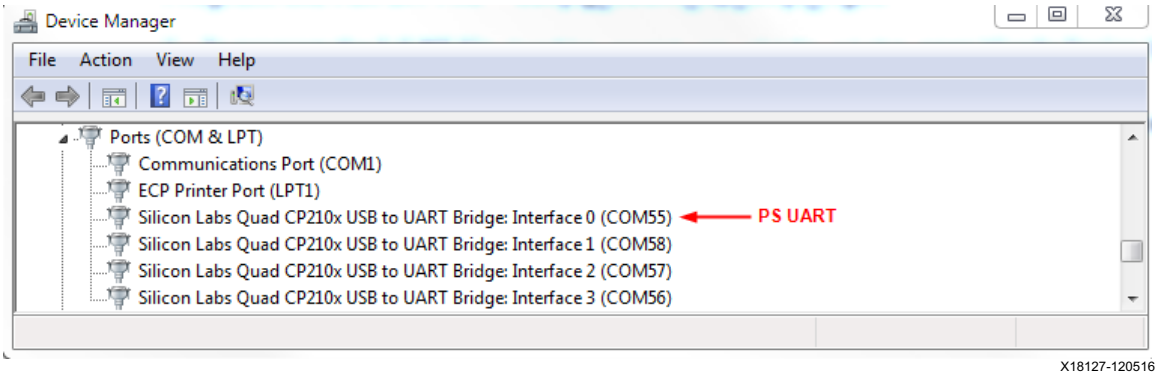

Figure 23: **COM Port Selection for the Design**

Program the Flash Memory

1. Open SDK.
2. **Xilinx Tools > Program Flash**. Settings follow:

Image File	`MPSoC_AXI2SEM/SDK/boot_img/BOOT.bin`
Offset	0x00000000
Flash Type	qspi_quad_parallel
FSBL File	`MPSoC_AXI2SEM/SDK/boot_img/fsbl_a53_0.elf`

3. After programming finishes, turn off the board.
4. Set MODE[3:0] pins (SW6) to ON ON OFF ON. Note that for this DIP switch, in relation to the arrow, moving the switch toward the label ON is a 0. DIP switch labels 1 through 4 are equivalent to Mode pins 0 through 3. See Figure 24.

Figure 24: **MODE[3:0] DIP Switch Setting (ON ON OFF ON)**

5. Turn on power to the board.
6. The application starts. Follow the directions in the PS UART terminal and verify the SEM controller is operational. Refer to the Results for Software Application description for more details on the software application.

Results

Refer to the log file provided under the main directory of the application note ZIP file for the output of the reference design:

- `teraterm_axi2sem_intc.log` - Log file that contains the UART output from the PS. It also contains the SEM controller's output because this information is read from the Log FIFO.

For register definitions, refer to Appendix A—AXI2SEM Register Definition.

Results for Software Application

This application initializes the PS UART, controls the transfer of control of the PL configuration logic from PCAP to ICAP (ICAP is required by SEM), and uses AXI4-Lite to interface with the SEM controller. The register interface is used to read status and health information for SEM and to send commands to SEM. Interrupts are used to perform mitigation actions. Three snippets, which make up the full log, are provided in Figure 25, Figure 26, and Figure 27.

The PS disables PCAP and enables ICAP. The PS issues a PL reset (this is not required, but is performed to show it works) and enables the interrupts in the AXI2SEM IP Interrupt controller (IER and GIE) and the AXI Interrupt controller (IER and MER). After the interrupts have been enabled, the PS grants CAP access to the SEM controller.

ICAP Arbitration

The PS uses these steps to give the SEM controller access to the CAP:

1. Read the ICAP register to make sure bit 2 (`icap_request`) is asserted. This means SEM is requesting access to the CAP.
2. If bit 2 is set, the PS writes `0x00000002` to the ICAP register to assert the `ICAP Grant` signal. At this point, the SEM controller initializes and automatically transitions to the Observation state.

SEM Ready Interrupt

After SEM initializes, it causes an interrupt (SEM Ready) to be generated. To service the interrupt, the PS must read the AXI Interrupt controller register to determine which AXI peripheral generated the interrupt. In this reference design, the only AXI peripheral connected to the AXI Interrupt controller is AXI2SEM. Any time an AXI2SEM interrupt occurs, bit 0 in the AXI interrupt controller (AXI INTC) interrupt pending register (IPR) is set. The interrupt handler must then determine which interrupt from AXI2SEM generated the PS interrupt. This is done by ANDing the IP interrupt enable register (IPIER) and IP interrupt status register (IPISR). An IPISR bit is set any time an interrupt occurs whether the interrupt is enabled or not. The IPIER is used to enable each interrupt independent of the other interrupts in the register. If an interrupt is not enabled in the IPIER, even though the interrupt is present in the IPISR, it does not assert the interrupt signal to the AXI Interrupt controller. The result of the AND provides the PS with only the active enabled interrupts.

Four interrupts are currently monitored in the interrupt handler. If an interrupt is detected, then the relevant IPISR bit is cleared by writing to it, then the AXI INTC ISR bit is cleared by writing to the AXI INTC IAR register. Refer to *AXI Interrupt Controller (INTC) LogiCORE IP Product Guide* (PG099) for more information [Ref 4]. If an interrupt is enabled, but the interrupt handler does not look for that interrupt, it causes the system to hang because the interrupt never gets serviced. In this reference design, only four of the twelve interrupts are handled by the interrupt handler. If more interrupts are enabled, the interrupt handler must also be modified.

In the snippet in Figure 25, an interrupt is received from the AXI Interrupt controller (AXI_INTC_IPR [0]). First the AXI2SEM interrupt is cleared (AXI2SEM_INTR_IP_ISR [11]) by writing to the IPISR. Then the AXI INTC IPR [0] bit is cleared by writing to AXI INTC IAR [0]. The log shows `0x00000041` after the SEM Ready interrupt is cleared. This is because other interrupts have occurred (Log FIFO Data Available and status initialization), but because they are not enabled, these interrupts cannot assert the interrupt to the AXI Interrupt controller. Next, the status indicators from the interrupt handler are cleared by the software application (sem_intr_status [11] and axiintc_intr_status [0]).

```
Xilinx Zynq MP First Stage Boot Loader
Release 2016.2   Jul 12 2016  -  12:05:31
FlashID=0x20 0xBB 0x20

Demonstration of UltraScale+ Zynq MPSoC with AXI2SEM & SEM IP
*** PCAP Register Access  ***
    INFO: PCAP is disable, ICAP is enabled
  Press Enter/Return to continue...
*** Reset PL  ***
    INFO: PL is in reset.
    INFO: PL was taken out of reset.
  Press Enter/Return to continue...
*** Wait for ICAP Request from SEM IP ***
    INFO: SEM IP has asserted ICAP Request
*** Assert ICAP Grant, but continue to Deassert ICAP Release  ***

    INFO: ICAP Release, ICAP Request, and ICAP Grant are LOW, HIGH, and HIGH respectively.
    INFO: SEM Ready Interrupt will be set.

    INFO: Received AXI Interrupt Controller Interrupt from AXI2SEM and cleared it (AXI_INTC_IPR[0]) 0x00000000

    INFO: Received SEM Ready Interrupt and cleared it (AXI2SEM_INTR_IP_ISR[11]) 0x00000041
    INFO: Cleared indicator from Interrupt Handler (sem_intr_status[11]) 0x00000000

    INFO: Cleared indicator from Interrupt Handler (axiintc_intr_status[0]) 0x00000000

    INFO: SEM Initialization completed successfully. SEM Ready Register: 0x000000E9

***  Reading the Log FIFO!!!  ***

SEM_ULTRA_V3_1
SC 01
FS 04
AF 01
ICAP OK
RDBK OK
INIT OK
SC 02
O>

*** Done reading initialization report!!!  ***

*** Reading Error Register to verify no errors have occurred!  ***

    INFO: No Errors are set 0x00000000
```

X18120-120516

Figure 25: **AXI2SEM Application Initialization Log**

Reading the Log FIFO

After the SEM Ready interrupt is serviced, the PS reads the Log FIFO until it is empty, using this process:

1. Read bit 0 of the Log FIFO Status Register.
2. If data is in the FIFO, read bits [7:0] of the Log Data register.
3. Repeat steps 1 and 2 until the Flog FIFO is empty.

Error Register

The Error register is read to make sure no errors have occurred after initialization.

Inject a Correctable Error

Next, the application injects a correctable error in the PL configuration memory. The commands in the following steps are equivalent to the following commands from the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 5]:

I

N C000A098000

O

AXI2SEM uses this process:

1. Write `0x00000002` to the Command register. This puts the SEM controller into the Idle state.
2. Read bit 27 of the Command register to make sure wr_busy is not set.
3. Write `0xC000A098` to the Command Base register (CBR).
4. Read bit 27 of the Command register to make sure wr_busy is not set.
5. Write `0x00000000` to the Command Extended register (CER).
6. Read bit 27 of the Command register to make sure wr_busy is not set.
7. Write `0x00000200` to the Command register to send the Injection command to the SEM controller.
8. Read bit 27 of the Command register to make sure wr_busy is not set.
9. Write `0x00000001` to the Command register to transition the SEM controller to the Observation state.

Correction Interrupt

The SEM controller detects the error and the Correction interrupt is generated. This interrupt is generated any time the SEM controller exits the Correction state. It does not mean that an error was corrected. This interrupt is also generated upon exiting the correction state of an uncorrectable error. To determine if a correctable error occurred, the Log FIFO data should be analyzed and the Error register should be read.

In the snippet in Figure 26, an interrupt is received from the AXI Interrupt controller (AXI_INTC_IPR [0]). The interrupt handler detects the correction state interrupt and clears the associated IPISR bit by writing to it (AXI2SEM_INTR_IP_ISR [7]). Then the AXI INTC IPR [0] bit is cleared by writing to AXI INTC IAR [0]. Refer to *AXI Interrupt Controller (INTC) LogiCORE IP Product Guide* (PG099) for more information [Ref 4].

The log shows `0x00000041` after the interrupt is cleared. This is because other interrupts have occurred (Log FIFO Data Available and status initialization), but because they are not enabled, these interrupts cannot assert the interrupt to the AXI Interrupt controller. Next, the status indicators from the interrupt handler are cleared by the software application (sem_intr_status [7] and axiintc_intr_status [0]).

After the Correction State interrupt is serviced, the PS reads the Log FIFO until it is empty. This is done by using the process described in Reading the Log FIFO.

After the correction report information is read from the Log FIFO, the Error register is read to make sure a correctable error was set. The Correctable Error bit is cleared by writing a 1 to the bit. In a real-life application, this register can be used to determine if the error was correctable, uncorrectable, a ROM error, or a CRC error. Only the ROM and Correctable Error bits can be cleared by writing a 1 to the bit in the register.

```
**************************************************************************
*** Inject a single bit error at LFA 0xC000A098000                   ***
**************************************************************************
    I
    N C000A098000
    O
  Press Enter/Return to continue...

    INFO: Received AXI Interrupt Controller Interrupt from AXI2SEM and cleared it (AXI_INTC_IPR[0]) 0x00000000

    INFO: Received Correction State Interrupt, and cleared it (AXI2SEM_INTR_IP_ISR[7]) 0x00000041
    INFO: Cleared indicator from Interrupt Handler (sem_intr_status[7]) 0x00000000

    INFO: Cleared indicator from Interrupt Handler (axiintc_intr_status[0]) 0x00000000

***  Reading the Log FIFO!!!  ***

I
SC 00
I> N C000A098000
SC 10
SC 00
I> O
SC 02
O>
RI 00
SC 04
ECC
TS 00000046
PA 00180200
LA 0000A098
COR
WD 00 BT 00
END
FC 00
SC 08
FC 40
SC 02
O>

*** Done reading Correction State report!!!  ***

*** Reading Error Register to verify a Correctable Error occurred!  ***

    INFO: Correctable Error bit is set (AXI2SEM_ERROR[2]) 0x00000004
    INFO: Clearing Correctable Error bit
    INFO: Correctable Error bit was cleared  (AXI2SEM_ERROR[2]) 0x00000000
```

X18121-120516

Figure 26: **AXI2SEM Application Correctable Error Log**

Inject an Uncorrectable Error

Next, the application injects an Uncorrectable error in the PL configuration memory. The commands in the following steps are equivalent to the following commands from *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 5]:

I

N C000A098000

N C000A098004

O

AXI2SEM uses this process:

1. Write `0x00000002` to the Command register. This puts the SEM controller into the Idle state.
2. Read bit 27 of the Command register to make sure wr_busy is not set.
3. Write `0xC000A098` to the Command Base register (CBR).
4. Read bit 27 of the Command register to make sure wr_busy is not set.
5. Write `0x00000000` to the Command Extended register (CER).
6. Read bit 27 of the Command register to make sure wr_busy is not set.
7. Write `0x00000200` to the Command register to send the Injection command to the SEM controller.
8. Read bit 27 of the Command register to make sure wr_busy is not set.
9. Write `0xC000A098` to the Command Base register (CBR).
10. Read bit 27 of the Command register to make sure wr_busy is not set.
11. Write `0x00400000` to the Command Extended register (CER).
12. Read bit 27 of the Command register to make sure wr_busy is not set.
13. Write `0x00000200` to the Command register to send the Injection command to the SEM controller.
14. Read bit 27 of the Command register to make sure wr_busy is not set.
15. Write `0x00000001` to the Command register to transition the SEM controller to the Observation state.

Uncorrectable Interrupt

The SEM controller detects the error and the Correction State interrupt is generated. This interrupt is generated any time the SEM controller exits the Correction state. It does not mean that an error was corrected. This interrupt is also generated upon exiting the correction state of an uncorrectable error. To determine if an uncorrectable error occurred, the Log FIFO data should be analyzed and the Error register should be read.

In the snippet in Figure 27, an interrupt is received from the AXI Interrupt controller (AXI_INTC_IPR [0]). The interrupt handler detects the correctable state interrupt. It clears the correction state interrupt by writing to the IPISR (AXI2SEM_INTR_IP_ISR [7]). Next, the uncorrectable error interrupt is detected and cleared by writing to the IPISR (AXI2SEM_INTR_IP_ISR [8]). Then the AXI INTC IPR [0] bit is cleared by writing to AXI INTC IAR [0]. Refer to *AXI Interrupt Controller (INTC) LogiCORE IP Product Guide* (PG099) [Ref 4] for more information.

The log shows `0x00000041` after each interrupt is cleared. This is because other interrupts have occurred (Log FIFO Data Available and status initialization), but because they are not enabled, these interrupts cannot assert the interrupt to the AXI Interrupt controller.

After the status indicator for the correction state interrupt is cleared by the software application (sem_intr_status [7]), it returns `0x00000100`. The bit that is set is the status indicator for the uncorrectable interrupt (sem_intr_status [8]). After the status indicator for the uncorrectable interrupt is cleared by the software application, it returns `0x00000000`. All status indicators have been cleared.

After all interrupt status indicators have been cleared, the PS reads the Log FIFO until it is empty. This is done by using the process described in Reading the Log FIFO.

After the uncorrectable report information is read from the Log FIFO, the Error register is read to make sure the error was uncorrectable. The application instructs you to reconfigure the device.

```
**************************************************************************
*** Inject a double bit error at LFA 0xC000A098000 and 0xC000A098004 ***
**************************************************************************
    I
    N C000A098000
    N C000A098004
    O
  Press Enter/Return to continue...

    INFO: Received AXI Interrupt Controller Interrupt from AXI2SEM and cleared it (AXI_INTC_IPR[0]) 0x00000000

    INFO: Received Correction State Interrupt, and cleared it (AXI2SEM_INTR_IP_ISR[7]) 0x00000041
    INFO: Cleared indicator from Interrupt Handler (sem_intr_status[7]) 0x00000100

    INFO: Received Uncorrectable Error Interrupt and cleared it (AXI2SEM_INTR_IP_ISR[8]) 0x00000041
    INFO: Cleared indicator from Interrupt Handler (sem_intr_status[8]) 0x00000000

    INFO: Cleared indicator from Interrupt Handler (axiintc_intr_status[0]) 0x00000000

***  Reading the Log FIFO!!!  ***

I
SC 00
I> N C000A098000
SC 10
SC 00
I> N C000A098004
SC 10
SC 00
I> O
SC 02
O>
RI 00
SC 04
ECC
TS 00000059
PA 00180200
LA 0000A098
COR
END
FC 60
SC 08
FC 60
SC 00
I>

*** Done reading Uncorrectable Error report!!!  ***

*** Reading Error Register to verify an Uncorrectable Error occurred!  ***

    INFO: Uncorrectable Error bit is set (AXI2SEM_ERROR[0]) 0x00000001
    INFO: Cannot clear uncorrectable error!!!
    INFO: Reprogram the device!!!

  Press Enter/Return to exit program...
Exiting Program!!!
```

X18122-120516

Figure 27: **AXI2SEM Application Uncorrectable Error Log**

Debugging

Ensure all diagram connections match those in Figure 2.

Limitations and Considerations

AXI2SEM

IMPORTANT: *This application and all supporting files are for **demonstration purposes only**. All files are delivered **as is**. This reference design is not intended to be a drop-in solution. After integrating this reference design into a new design, thorough verification is required to ensure the resulting solution meets functional and reliability goals.*

XPM_FIFO

This reference design uses XPM_FIFO which is currently in Beta release for Vivado Design Suite 2016.2, and there are limitations to its usage. See the provided package for details on the limitations (`MPSoC_AXI2SEM/ip/xpm_fifo/limitations`).

Understand the following supported and unsupported features. Do not assume something is supported if it is not listed. Contact your FAE if you have any questions or concerns.

XPM_FIFO Supported Features (2016.2 xpm_fifo beta release)

These features are supported:

- Native interface
- Clock domain (common, independent)
- Read mode (Standard, FWFT)
- Width 1–1024 (symmetric only)
- Depth up to 8192 (depth must be a power of 2)
- FIFO read latency—0, 1
  - Value 0 applies only for FWFT. Value ≥ 1 applies only for Standard Read mode.
- Underflow/Overflow

XPM_FIFO Unsupported Features (2016.2 xpm_fifo beta release)

The following features are not supported:

- Asymmetric data width
- ECC
- DOUT reset value
- Programmable Full/Empty
- Data count

- Option for FULL to reset to 1

SEM Controller IP

For the latest information, see the *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 5]. Note the following:

- The SEM controller does not operate on soft errors in block memory, distributed memory, or flip-flops. Soft error mitigation in these memory resources must be addressed by the user logic through preventive measures such as redundancy or error detection and correction codes.
- The SEM controller initializes and manages the FPGA integrated silicon features for soft error mitigation and when included in a design, does not include any design constraints or options that would enable the built-in detection functions. For example, do not set POST_CRC, POST_CONFIG_CRC, or any other related constraints. Similarly, do not include options to modify GLUTMASK.
- Software-computed ECC and CRC values are not supported.
- Design simulations that instantiate the SEM controller are supported. However, it is not possible to observe the controller behaviors in simulation. Design simulation including the controller compiles, but the controller does not exit the Initialization state. Hardware-based evaluation of the controller behaviors is required.
- Use of bitstream security (encryption and authentication) is currently not supported by the controller.
- Use of SelectMAP persistence is not supported by the controller.
- Only a single ICAP instance is supported for each SEM controller and it must reside at the primary/top physical location.

The SEM controller IP is not supported when targeting ES1 UltraScale+ devices where the GTH and GTY transceivers' dynamic reconfiguration port (DRP) is connected to user logic or IP.

Targeting Other Software Versions

SEM IP

The ZCU102 board uses a ZU9EG device. SEM IP support of this device is not available in the Vivado IP integrator catalog before Vivado release 2016.3. The `MPSoC_SEM_ICAP_MBoot/ip` directory contains the SEM UltraScale+ family packaged design (`sem_ultra_v3_1`) which is required only for designs before release 2016.3. From release 2016.3 onward, you can generate the SEM IP directly in the Vivado IP integrator using the IP catalog.

ZCU102 Board Files

Starting in Vivado release 2016.4, the ZCU102 board files are included with the Vivado tools. These board files are for board revision 1.0. For designs targeting release 2016.4 or later with board revision 1.0 or later, you do not need to download any board files.

Download the board files from the Zynq UltraScale+ ZCU102 HeadStart website:

- If you are using Vivado release 2016.4 with a board revision before 1.0 (such as Rev D).
- If you are using a Vivado release prior to 2016.4.

IMPORTANT: Be careful to download the board files for your specific Vivado release and board revision.

XPM_FIFO

In Vivado release 2016.2, XPM_FIFO was in Beta release. The templates required to build the XPM_FIFO macro included in the ZIP file for this application note only apply to Vivado 2016.2. The XPM_FIFO templates are provided with Vivado 2016.3 or later. When targeting Vivado 2016.3 or later, the XPM_FIFO instantiations in `axi2sem_cdc.vhd` need to be updated. These instantiation templates can be found in the *UltraScale Architecture Libraries Guide* (UG974) [Ref 6].

Appendix A—AXI2SEM Register Definition

Register Definition

Table 4: **AXI2SEM Register Map**

Address	Name	Read/Write	Description
0x0	SEM Ready Register	Read only	SEM Ready register: Indicates if the SEM controller initialized. If it did, then it also sets a bit to indicate what state it initialized to: Idle (I), Observation (O), or Detect-only (D). This register is cleared when a SEM Soft Reset is issued, or when icap_rel = 1 and icap_request = 0. All bits are active-High. 31:8 - Reserved 7 - Received 'ICAP OK' in initialization report 6 - Received 'RDBK OK' in initialization report 5 - Received 'INIT OK' in initialization report 4 - Booted into Idle 3 - Booted into Observation 2 - Booted into Detect-only 1- Initialization Error. Initialization did not complete within 1 second. Count starts after ICAP has been arbitrated for (icap_release = 0, icap_grant = 1, and icap_request = 1) or after a Soft Reset is detected by the `init_check.vhd` module. 0- SEM Ready Asserted when the SEM controller initialized. All other bits in this register are used to help debug if the SEM controller ever fails to initialize.
0x4	ICAP Register	Read/write	ICAP Register: ICAP arbitration interface signals. 31:3 - Reserved 2 - icap_request 1 - icap_grant 0 - icap_release

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x8	Status Register	Read only	Status Register: Status signals indicating the health of the SEM controller. All bits are active-High. 31:10 - Reserved 9 - heartbeat_present – 0 when heartbeat is not detected for one second or more; otherwise 1 8 - status_initialization (from SEM controller) 7 - status_observation (from SEM controller) 6 - status_correction (from SEM controller) 5 - status_classification (from SEM controller) 4 - status_injection (from SEM controller) 3 - status_essential (from SEM controller) 2 - status_uncorrectable (from SEM controller) 1 - status_diagnostic_scan (from SEM controller) 0 - status_detect_only (from SEM controller)
0xC	Reserved	Read only	31:0 Reserved
0x10	Command Register	Read/write	Command Register: Used to set the SEM commands (Query, Inject, Translate, External Memory, Peek, Soft Reset, Diagnostic Scan, Detect Only, Status, Idle, and Observation). Also contains the Command FIFO status and Write Busy status. Only one command bit can be set at a time. A write to this register initiates the command transaction to SEM. All status bits are active-High. 31 - rd_rst_busy (Read only) status from command FIFO 30 - wr_rst_busy (Read only) status from command FIFO 29 - overflow (Read only) status from command FIFO 28 - full (Read only) status from command FIFO 27 - wr_busy (Read only) - stays High from the time this register is written until the last character has been written to the command FIFO 26:11 - Reserved 10 - Query command 9 - Injection command 8 - Translate command 7 - External Memory command 6 - Peek command 5 - Soft Reset command 4 - Diagnostic Scan command 3 - Detect-only command 2 - Status command 1 - Idle command 0 - Observation command

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x14	Command Base Register (CBR)	Read/write	Command Base Register (CBR): Used to hold the first 8 bits to 32 bits of the Soft Reset, Peek, External Memory, Translate, Inject, Translate, and Query commands. This register must be written before the Command register is written. The Command Extended register must be used in conjunction with this register to hold the remaining 12 bits of the Inject, Translate, and Query commands. See the following examples: **Soft Reset** – Example 'R **04**' **0 = CBR(31:28)** **4 = CBR(27:24)** **Peek** – Example 'P **0C**' **0 = CBR(31:28)** **C = CBR(27:24)** **External Memory** – Example 'X **01234567**' **0 = CBR(31:28)** **1 = CBR(27:24)** **2 = CBR(23:20)** **3 = CBR(19:16)** **4 = CBR(15:12)** **5 = CBR(11:8)** **6 = CBR(7:4)** **7 = CBR(3:0)** **Inject** – Example 'I **01234567**89A' **0 = CBR(31:28)** **1 = CBR(27:24)** **2 = CBR(23:20)** **3 = CBR(19:16)** **4 = CBR(15:12)** **5 = CBR(11:8)** **6 = CBR(7:4)** **7 = CBR(3:0)** Must use the Command Extended Register (CER) for the remaining command bits. **Translate** – Example 'T **01234567**89A' **0 = CBR(31:28)** **1 = CBR(27:24)** **2 = CBR(23:20)** **3 = CBR(19:16)** **4 = CBR(15:12)** **5 = CBR(11:8)** **6 = CBR(7:4)** **7 = CBR(3:0)** Must use the Command Extended Register (CER) for the remaining command bits.

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x14 (continued)	Command Base Register (CBR)	Read/write	**Query** – Example 'Q **01234567**89A' **0 = CBR(31:28)** **1 = CBR(27:24)** **2 = CBR(23:20)** **3 = CBR(19:16)** **4 = CBR(15:12)** **5 = CBR(11:8)** **6 = CBR(7:4)** **7 = CBR(3:0)** Must use the Command Extended Register (CER) for the remaining command bits.
0x18	Command Extended Register (CER)	Read/write	Command Extended Register (CER): Used to hold the last 12 bits of the Translate, Inject, and Query commands. This register must be written before the Command register is written. This register must be used in conjunction with the Command Base Register which holds the first 32 bits of these commands. 31:20 - Remaining Translate, Inject, and Query Command bits 19:0 - Reserved See the following examples: **Injection** - Example 'I 0123456**789A**' **8 = CBR(31:28)** **9 = CBR(27:24)** **A = CBR(23:20)** **Translate** – Example 'T 0123456**789A**' **8 = CBR(31:28)** **9 = CBR(27:24)** **A = CBR(23:20)** **Query** - Example 'Q 0123456**789A**' **8 = CBR(31:28)** **9 = CBR(27:24)** **A = CBR(23:20)** 19:0 - Reserved
0x1C	Reserved	Read only	31:0- Reserved
0x20	Log FIFO Status	Read only	Log FIFO Status: Provides status of the log FIFO. All bits are active-High. 31:7 - Reserved 6 - Empty 5 - Underflow 4- Full 3 - Overflow 2 - FIFO Write not ready 1 - FIFO Read not ready 0 - Data available

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x24	Log FIFO Data	Read only	Log FIFO Data: ASCII data from SEM. Each read from this register provides one byte of ASCII data. 31:8 - Reserved 7:0 - ASCII Data
0x28	Reserved	Read only	31:0 - Reserved
0x2C	Reserved	Read only	31:0 - Reserved
0x30	Error Register	Read/write Read only	Error Register: Indicates when a ROM, ECC Correctable, CRC, and/or Uncorrectable ECC error has occurred. The ROM and ECC Correctable bits can be cleared by writing a '1' to the bits. The CRC and Uncorrectable ECC bits cannot be cleared. The FPGA must be reconfigured to clear these bits. 31:4 - Reserved 3 - ROM error detected 2 - ECC correctable error detected 1 - CRC error detected 0 - Uncorrectable ECC error detected
0x3C - 0x7F	Reserved	Read only	31:0 - Reserved
0x100	Device Interrupt Status Register	Read/write	Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for usage details [Ref 2].
0x104	Device Interrupt Pending Register	Read only	Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for usage details [Ref 2].
0x108	Device Interrupt Enable Register	Read/write	Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for usage details [Ref 2].
0x118	Device Interrupt ID Register	Read only	Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for usage details [Ref 2].
0x11C	Global Interrupt Enable	Read/write	Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) for usage details [Ref 2].

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x120	IP Interrupt Status Register (IPISR)	Read/write	IP Interrupt Status Register (IPISR) Each bit in this register represents an interrupt bit. Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) [Ref 2] for usage details. 1 = Interrupt is pending 0 = No interrupt is pending 31–12 - Reserved 11 - SEM ready 10 - CRC detected 9 - ROM detected 8 - Status uncorrectable 7 - Status correction 6 - Status initialization 5 - Initialization error 4 - Heartbeat timeout 3 - Log FIFO overflow 2 - Log FIFO full 1 - Log FIFO underflow 0 - Log FIFO data available

Table 4: **AXI2SEM Register Map *(Cont'd)***

Address	Name	Read/Write	Description
0x128	IP Interrupt Enable Register (IPIER)	Read/write	IP Interrupt Enable Register (IPIER). Each bit in this register enables or masks an interrupt. Not used. Refer to *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166) [Ref 2] for usage details. 1 = Interrupt enabled 0 = Interrupt masked 31–12 - Reserved 11 - SEM ready 10 - CRC detected 9 - ROM detected 8 - Status uncorrectable 7 - Status correction 6 - Status initialization 5 - Initialization error 4 - Heartbeat timeout 3 - Log FIFO overflow 2 - Log FIFO full 1 - Log FIFO underflow 0 - Log FIFO data available

References

RECOMMENDED: *This application note was developed using Vivado Design Suite 2016.2. It is preferable to use the 2016.2 versions of Zynq UltraScale+ MPSoC Technical Reference Manual (UG1085) [Ref 7] and Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 8].*

1. *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298)
2. *LogiCORE IP Interrupt Control Product Guide for Vivado Design Suite* (PG166)
3. *AXI4-Lite IPIF LogiCORE IP Product Guide* (PG155)
4. *AXI Interrupt Controller (INTC) LogiCORE IP Product Guide* (PG099)
5. *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187)
6. *UltraScale Architecture Libraries Guide, v2016.2* (UG974)
7. *Zynq UltraScale+ MPSoC Technical Reference Manual* (UG1085)
8. *Zynq UltraScale+ MPSoC Register Reference* (UG1087)
9. *Silicon Labs CP210x USB-to-UART Installation Guide* (UG1033)

10. *Vivado Design Suite User Guide: Designing with IP* (UG896)

11. *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898)

12. *Processor System Reset Module LogiCORE IP Product Guide* (PG164)

13. *AXI Interconnect LogiCORE IP Product Guide* (PG059)

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
02/10/2017	1.0	Initial Xilinx release.
02/13/2017	1.0.1	Made typographical changes.
02/27/2017	1.0.2	Added link to XAPP1298 in References and made typographical changes.

Please Read: Important Legal Notices