Series PMC-SLX150

Spartan 6 Based FPGA

PMC Module

Getting Started with the PMC-SLX Engineering Design Kit

ACROMAG INCORPORATED

30765 South Wixom Road

P.O. BOX 437

Wixom, MI 48393-7037 U.S.A.

Tel: (248) 624-1541

Fax: (248) 624-9234

8500-924-B13C012

Table of Contents

Getting Started

The purpose of this document is to provide basic instructions on using the "PMC-SLX Engineering Design Kit" with the PMC-SLX Boards. It will focus on programming the FPGA of the PMC-SLX150 using VHDL, but can be easily modified to use with any model of the SLX line. This document also shows how to use the supplied DLL files with a MFC application. It is assumed that the user has a working knowledge of Xilinx, VHDL and Visual C++. Note that this document assumes Windows is used as the operating systems. Linux users can follow the same general procedure but be aware that differences exist that are not noted in this document.

There are small differences in the VHDL between the PMC-SLX150(E) and the PMC-SLX150(E)-1M models due to differences in the SRAM size. Refer to the appendix at the end of this manual for further information on the VHDL differences.

Installing the Board and Device Drivers

For first time users, turn off your computer, and unplug the power cord. Before touching either board, make sure to discharge all static electricity. Then attach the SLX150 to your carrier. Insert the carrier into an empty slot in your computer. When restarting your computer, you will be prompted to insert a CD with the drivers on it. At this point, insert the PCISW-API-WIN CD (software product sold separately from this EDK) into your CD-ROM drive. When the plug and play installation has completed, follow the steps to install the additional PCISW-API-WIN software on your computer. When finished, insert the CD titled **PMC-SLX Engineering Design Kit** and copy the **SLX150** folder to your computer.

Before you start, familiarize yourself with the **PMC-SLX User's Manual** included on the EDK CD and the **PCISLX Driver Function Reference** included on the PCI Win32 Driver Software CD. The user's manual gives the memory addresses of all the registers, and their purposes. The function reference gives information on how to use the DLL file in C/C++, Visual Basic, and LabView (we will be focusing only on using the C/C++ demo program).

A readme.txt file is included on the EDK CD. This file contains detailed information on the contents of the CD including a description of the contents of the VHDL files. Read this file prior to reading this manual.

This manual assumes that Xilinx ISE Version 13.2 in used. Note that earlier versions are not supported and later versions may have alternate procedures. Also note that VHDL line numbers in this manual may not match the line numbers of the files provided in the EDK due to revisions.

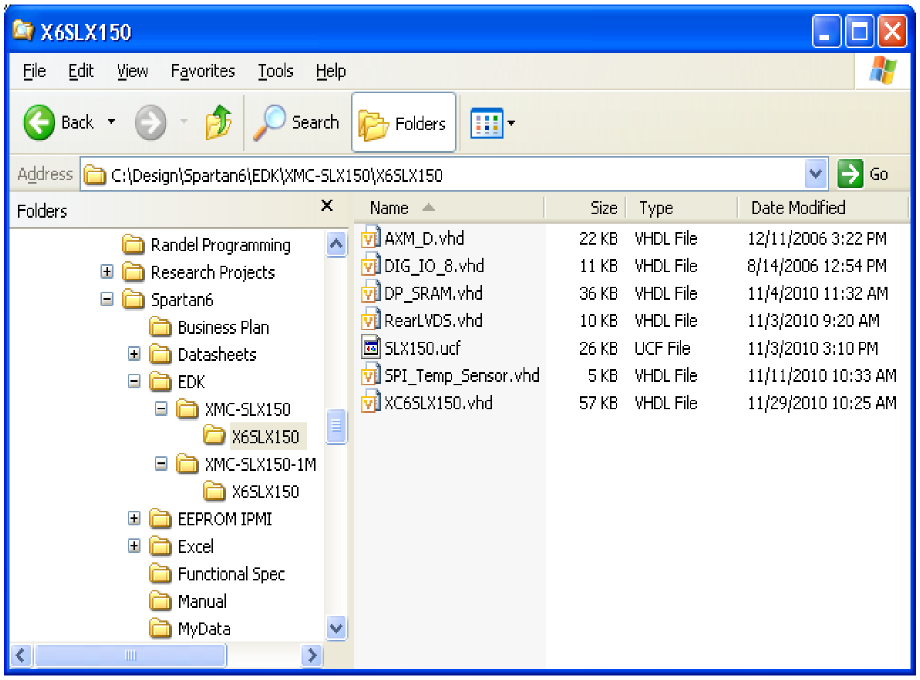
Starting a New Xilinx Project

1. Make a new directory on your computer and call it **XC6SLX150**.

2. From the **XC6SLX150** folder, copy the all the vhdl files in into the new **XC6SLX150 folder**. Then from the **XC6SLX150** folder copy the **SLX150.ucf** file to the XC6SLX150 folder. Note that all of the files are shown in the adjacent figure.

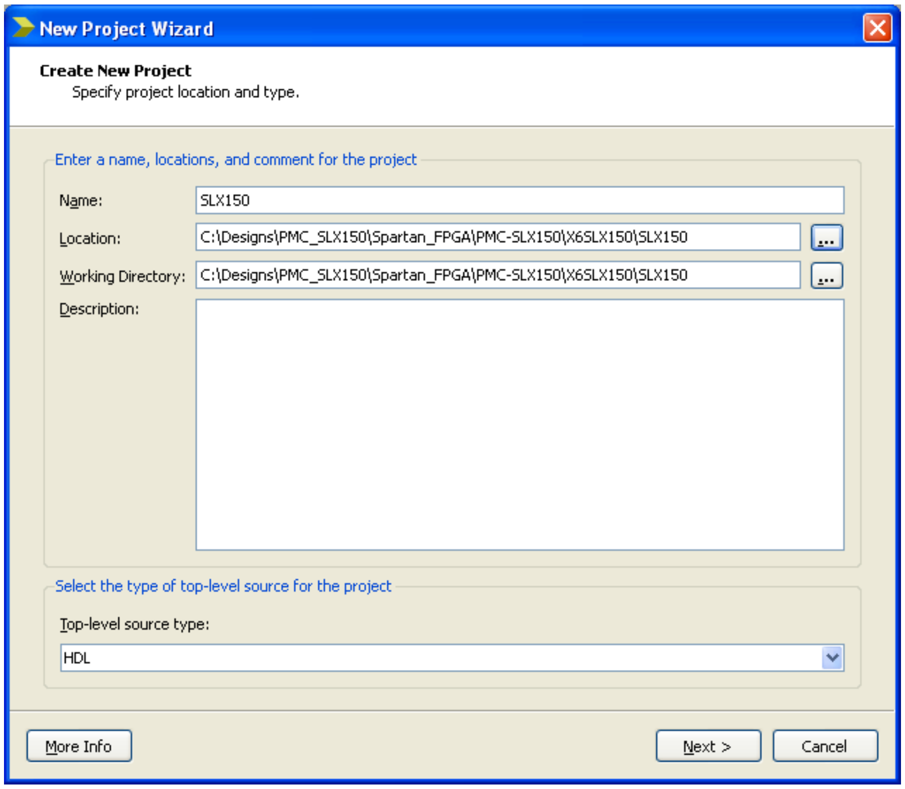
3. Start Xilinx's Project Navigator from your start menu. **Xilinx ISE Design Suite 13.2 → ISE Design Tools → Project Navigator**

4. Open a new project by selecting **File → New Project**



Name	Size	Type	Date Modified
AXM_D.vhd	22 KB	VHDL File	12/11/2006 3:22 PM
DIG_IO_8.vhd	11 KB	VHDL File	8/14/2006 12:54 PM
DP_SRAM.vhd	36 KB	VHDL File	11/4/2010 11:32 AM
RearLVDS.vhd	10 KB	VHDL File	11/3/2010 9:20 AM
SLX150.ucf	26 KB	UCF File	11/3/2010 3:10 PM
SPI_Temp_Sensor.vhd	5 KB	VHDL File	11/11/2010 10:33 AM
XC6SLX150.vhd	57 KB	VHDL File	11/29/2010 10:25 AM

5. In the **Project Name** field, type **SLX150.** In the **Location** field type the path name where to find the XC6SLX150 folder. Make sure the **Top-Level Module Type** field is **HDL**, and click **Next**.



Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name: SLX150
Location: C:\Designs\PMC_SLX150\Spartan_FPGA\PMC-SLX150\X6SLX150\SLX150
Working Directory: C:\Designs\PMC_SLX150\Spartan_FPGA\PMC-SLX150\X6SLX150\SLX150
Description:

Select the type of top-level source for the project

Top-level source type: HDL

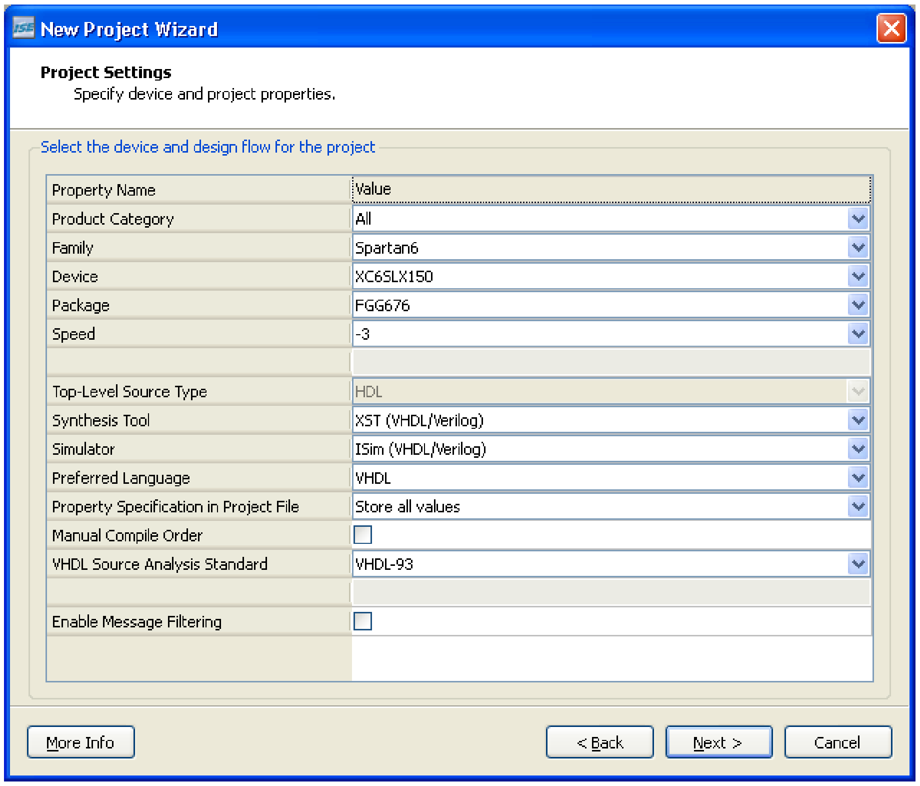
6. Enter the following information if using the SLX150. Then click **Next** and then **Finish.**

Family: Spartan6

Device: XC6SLX150

Package: FGG676

Speed: -3

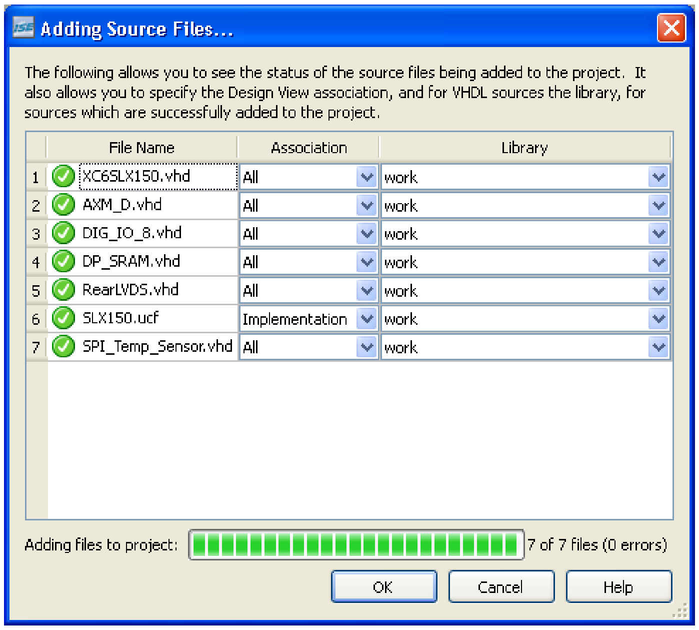


Project Settings
Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Product Category	All
Family	Spartan6
Device	XC6SLX150
Package	FGG676
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

7. We will next add the files we copied from the CD. Follow these steps:
   a. Select **Project->Add Source**.
   b. Select the .ucf and all the .vhd files
   c. Click the **Open** button.
   d. The association for all files should be *All* for the .vhd files and *Implement* for the .ucf files. There are a total of 6 .vhd files and one .ucf file.
   e. Click the **OK** button.



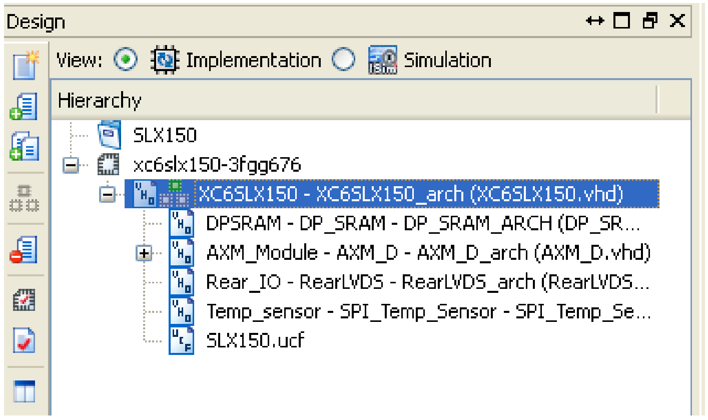
Adding Source Files...

The following allows you to see the status of the source files being added to the project. It also allows you to specify the Design View association, and for VHDL sources the library, for sources which are successfully added to the project.

	File Name	Association	Library
1	XC6SLX150.vhd	All	work
2	AXM_D.vhd	All	work
3	DIG_IO_8.vhd	All	work
4	DP_SRAM.vhd	All	work
5	RearLVDS.vhd	All	work
6	SLX150.ucf	Implementation	work
7	SPI_Temp_Sensor.vhd	All	work

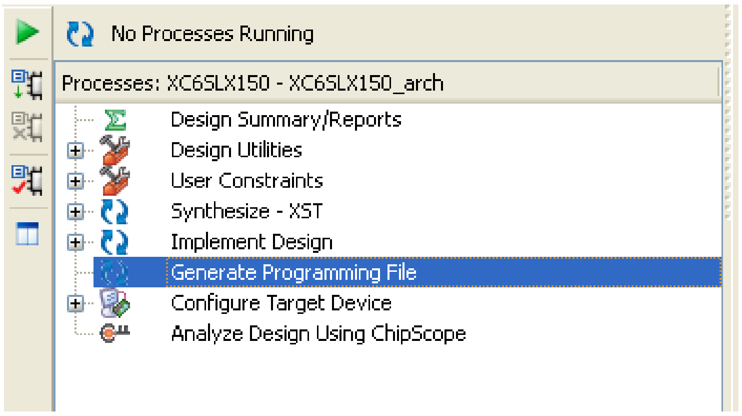
Adding files to project: 7 of 7 files (0 errors)

8. In the **Heirarchy Window**, click on **XC6SLX150_arch (XC6SLX150.vhd)** to highlight it.



```
SLX150
xc6slx150-3fgg676
  XC6SLX150 - XC6SLX150_arch (XC6SLX150.vhd)
    DPSRAM - DP_SRAM - DP_SRAM_ARCH (DP_SR...
    AXM_Module - AXM_D - AXM_D_arch (AXM_D.vhd)
    Rear_IO - RearLVDS - RearLVDS_arch (RearLVDS...
    Temp_sensor - SPI_Temp_Sensor - SPI_Temp_Se...
    SLX150.ucf
```

9. In the **Processes Window**, click on **Generate Programming File** so it is also highlighted.



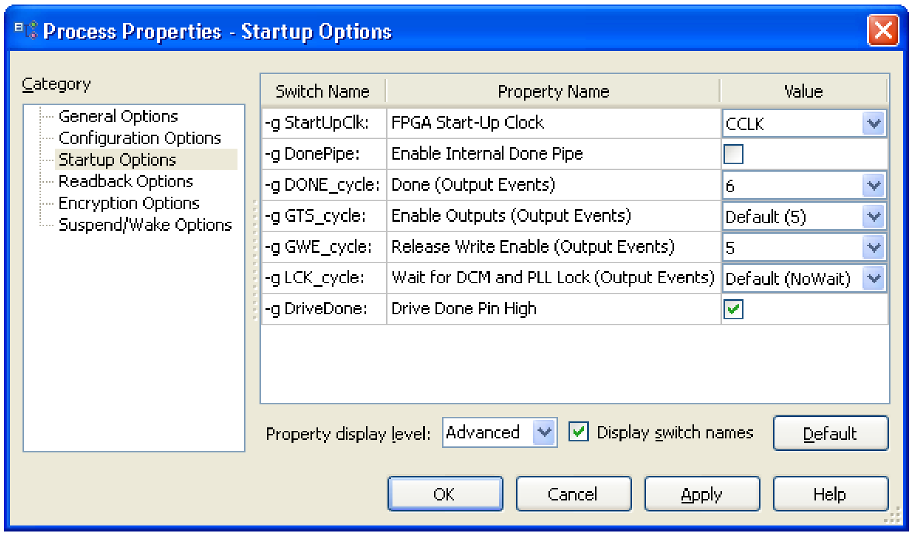
```
Processes: XC6SLX150 - XC6SLX150_arch
  Design Summary/Reports
  Design Utilities
  User Constraints
  Synthesize - XST
  Implement Design
  Generate Programming File
  Configure Target Device
  Analyze Design Using ChipScope
```

10. Click on **Process** from the menu bar, and click on **Properties.**

11. Click on the **Startup Options** tab and verify the following options are selected:

FPGA Start-up Clock:	**CCLK**
Enable Internal Done Pipe:	**Not Checked**
Done	**6**
Enable Outputs:	**5**
Release Write Enable:	**5**
Wait for DCM and PLL Lock:	**NoWait**
Drive Done Pin High:	**Checked**

Note that if not all properties are shown change the Property display level from Standard to Advanced.

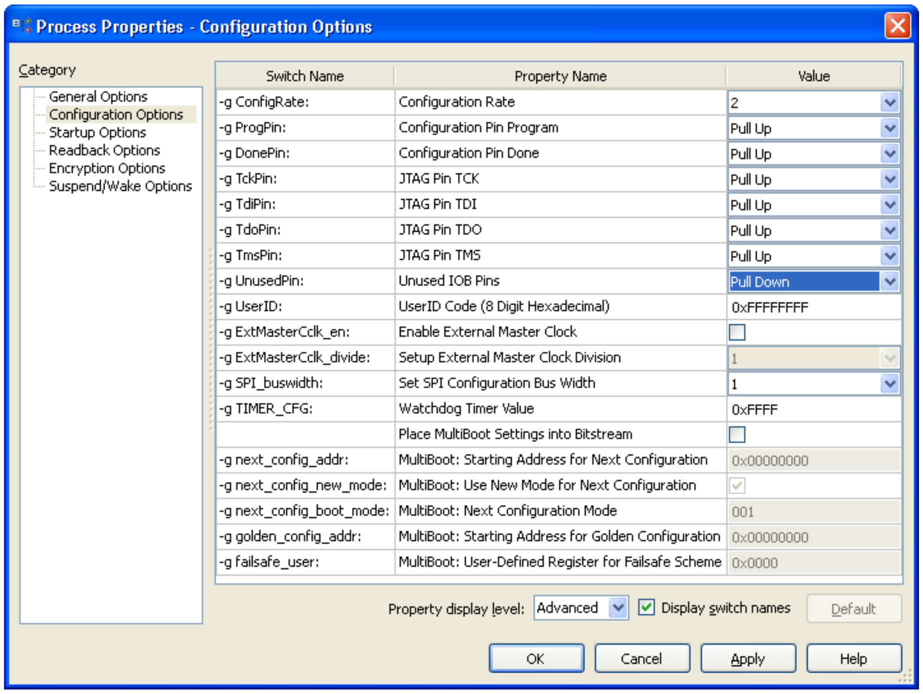


Switch Name	Property Name	Value
-g StartUpClk:	FPGA Start-Up Clock	CCLK
-g DonePipe:	Enable Internal Done Pipe	☐
-g DONE_cycle:	Done (Output Events)	6
-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
-g GWE_cycle:	Release Write Enable (Output Events)	5
-g LCK_cycle:	Wait for DCM and PLL Lock (Output Events)	Default (NoWait)
-g DriveDone:	Drive Done Pin High	✓

Property display level: Advanced ✓ Display switch names

12. Click on the **Configuration Options** tab. Verify that all options are set to default as shown in the screen shot to the right.

13. Click **OK**.



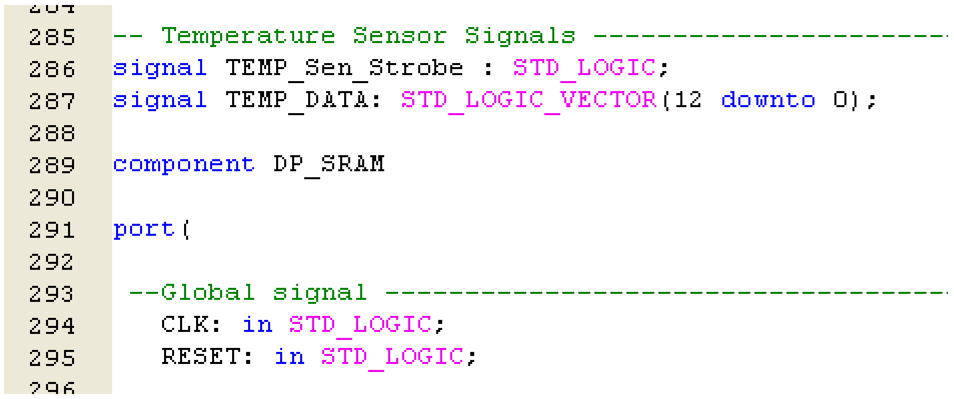
Switch Name	Property Name	Value
-g ConfigRate:	Configuration Rate	2
-g ProgPin:	Configuration Pin Program	Pull Up
-g DonePin:	Configuration Pin Done	Pull Up
-g TckPin:	JTAG Pin TCK	Pull Up
-g TdiPin:	JTAG Pin TDI	Pull Up
-g TdoPin:	JTAG Pin TDO	Pull Up
-g TmsPin:	JTAG Pin TMS	Pull Up
-g UnusedPin:	Unused IOB Pins	Pull Down
-g UserID:	UserID Code (8 Digit Hexadecimal)	0xFFFFFFFF
-g ExtMasterCclk_en:	Enable External Master Clock	☐
-g ExtMasterCclk_divide:	Setup External Master Clock Division	1
-g SPI_buswidth:	Set SPI Configuration Bus Width	1
-g TIMER_CFG:	Watchdog Timer Value	0xFFFF
	Place MultiBoot Settings into Bitstream	☐
-g next_config_addr:	MultiBoot: Starting Address for Next Configuration	0x00000000
-g next_config_new_mode:	MultiBoot: Use New Mode for Next Configuration	✓
-g next_config_boot_mode:	MultiBoot: Next Configuration Mode	001
-g golden_config_addr:	MultiBoot: Starting Address for Golden Configuration	0x00000000
-g failsafe_user:	MultiBoot: User-Defined Register for Failsafe Scheme	0x0000

Property display level: Advanced ✓ Display switch names

Modifying the Provided VHDL Code

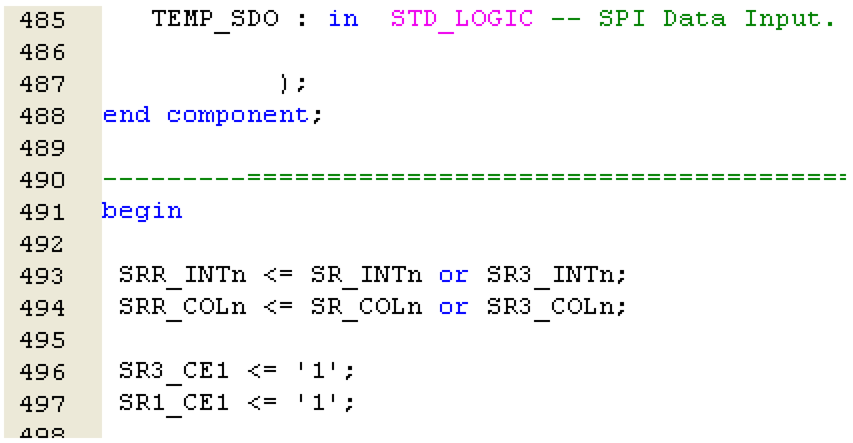
To revise or add to the provided VHDL code, begin by double clicking on the **XC6SLX150-XC6SLX150 _arch (XC6SLX150.vhd)** file located in the **Hierarchy** window. This will open the VHDL file for editing.

Additional components and signals may be added between the current definitions at line 288.



```
285  -- Temperature Sensor Signals -----------------------
286  signal TEMP_Sen_Strobe : STD_LOGIC;
287  signal TEMP_DATA: STD_LOGIC_VECTOR(12 downto 0);
288
289  component DP_SRAM
290
291  port(
292
293   --Global signal --------------------------------------
294     CLK: in STD_LOGIC;
295     RESET: in STD_LOGIC;
```

After the **begin** keyword (line 491) additional instantiations for components may be added



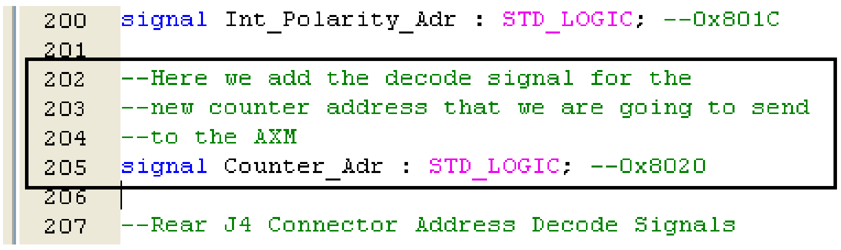
```
485     TEMP_SDO : in  STD_LOGIC -- SPI Data Input.
486
487                );
488  end component;
489
490  ---------========================================
491  begin
492
493   SRR_INTn <= SR_INTn or SR3_INTn;
494   SRR_COLn <= SR_COLn or SR3_COLn;
495
496   SR3_CE1 <= '1';
497   SR1_CE1 <= '1';
```

For simplicity we suggest starting by adding to or revising the provided VHDL code that is associated with the front I/O. To use the front I/O, begin by double clicking on the **AXM_Module-AXM_D_arch (AXM_D.vhd)** file located in the **Hierarchy** window. This will open the VHDL file for editing. To use the rear I/O, begin by double clicking on the **Rear_IO-RearLVDS_arch (RearLVDS.vhd)** file located in the **Hierarchy** window. This will likewise open the VHDL file associated with the rear I/O for editing

Example Change

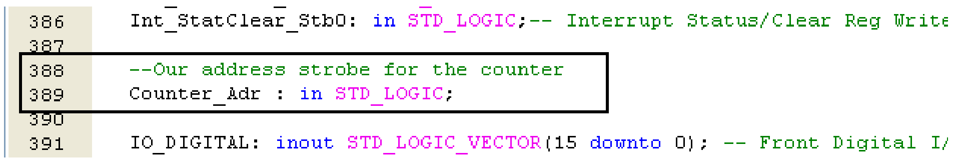
Below is a simple example of some VHDL that could be used to control five of the SLX150's Front I/O differential channels (via the AXM-D02 or AXM-D04). It is included to show how the code supplied with the Engineering Design Kit can be modified for personal use.

1. Open **XC6SLX150.vhd** and scroll down to around **line 205**. Add the line of code and accompanying comments as indicated by the box to the right. This creates a new address strobe for our counter. It will be located in register 0x8020.



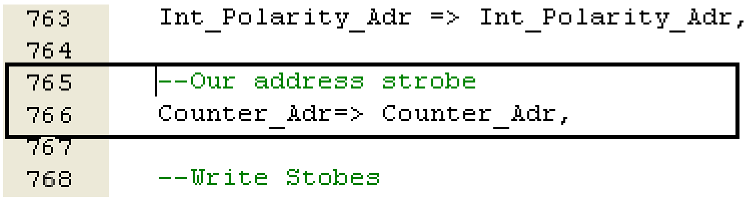
```
200  signal Int_Polarity_Adr : STD_LOGIC; --0x801C
201
202  --Here we add the decode signal for the
203  --new counter address that we are going to send
204  --to the AXM
205  signal Counter_Adr : STD_LOGIC; --0x8020
206
207  --Rear J4 Connector Address Decode Signals
```

2. Around **line 388** insert these two lines of code to the declaration of the AXM_D component. We will soon be changing the AXM_D.vhd file to match this declaration.



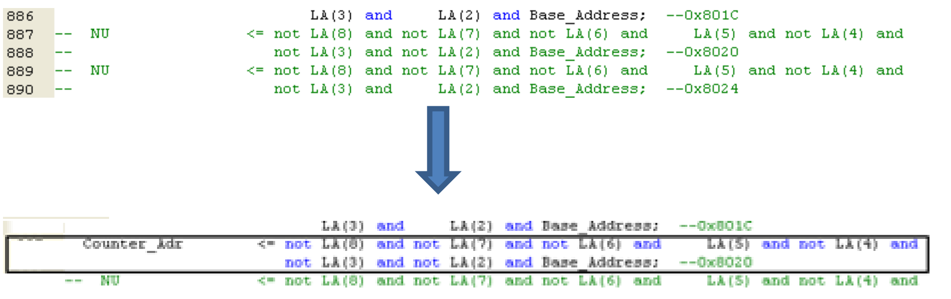
```
386      Int_StatClear_Stb0: in STD_LOGIC;-- Interrupt Status/Clear Reg Write
387
388      --Our address strobe for the counter
389      Counter_Adr : in STD_LOGIC;
390
391      IO_DIGITAL: inout STD_LOGIC_VECTOR(15 downto 0); -- Front Digital I/
```

3. Add the following around **line 765** our mapping of the Counter_Adr strobe to the AXM_D instantiation. Now the AXM_D component will receive all the information it needs for the design.



```
763      Int_Polarity_Adr => Int_Polarity_Adr,
764
765      --Our address strobe
766      Counter_Adr=> Counter_Adr,
767
768      --Write Stobes
```

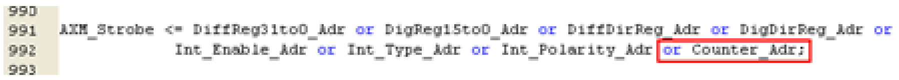
4. We will now replace a previously unused memory address. **Uncomment line 928 and 929** and replace the "NU" with **Counter_Adr**. This will be the location in memory to access the counter.



```
886                          LA(3) and     LA(2) and Base_Address;  --0x801C
887  --  NU               <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
888  --                      not LA(3) and not LA(2) and Base_Address;  --0x8020
889  --  NU               <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
890  --                      not LA(3) and     LA(2) and Base_Address;  --0x8024
```

```
                             LA(3) and     LA(2) and Base_Address;  --0x801C
     Counter_Adr          <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
                             not LA(3) and not LA(2) and Base_Address;  --0x8020
     --  NU               <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
```

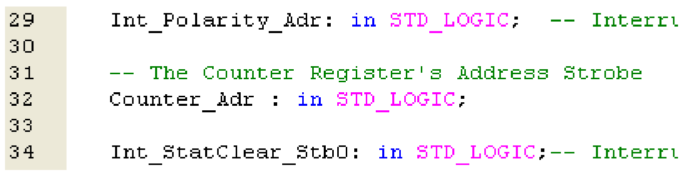
5. To **line 992** add the code in the red box. This is added to strobe the AXM (where the rest of the counter code will be located) when the Counter_Adr has received a read or write command.



```
990
991 AXM_Strobe <= DiffReg31to0_Adr or DigReg15to0_Adr or DiffDirReg_Adr or DigDirReg_Adr or
992              Int_Enable_Adr or Int_Type_Adr or Int_Polarity_Adr or Counter_Adr;
993
```

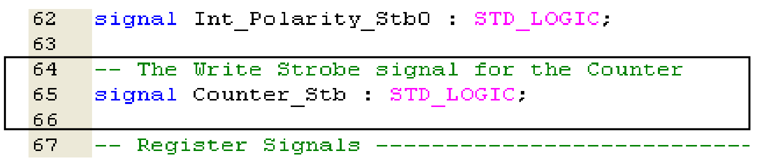
We are finished editing the XC5VLX110T.vhd file and will now be **editing the AXM_D.vhd** file.

6. After opening **AXM_D.vhd**, scroll down to **line 31** and add the Counter_Adr port. This is how the counter will be receiving the address strobe from the main vhdl code.



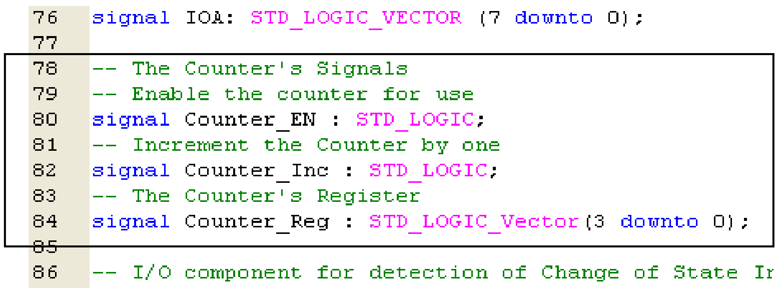
```
29     Int_Polarity_Adr: in STD_LOGIC;   -- Interru
30
31     -- The Counter Register's Address Strobe
32     Counter_Adr : in STD_LOGIC;
33
34     Int_StatClear_Stb0: in STD_LOGIC;-- Interru
```

7. To **line 64** add the write strobe for the counter. This will pulse when a write command is issued the counter address.



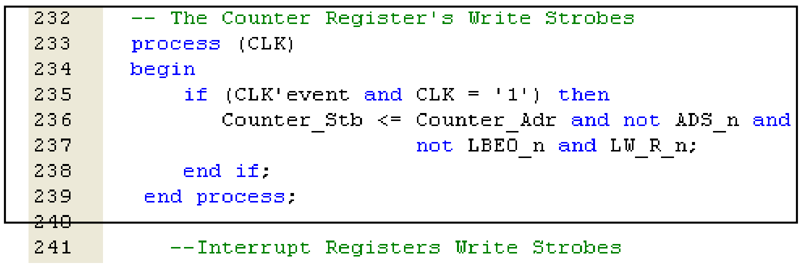
```
62  signal Int_Polarity_Stb0 : STD_LOGIC;
63
64  -- The Write Strobe signal for the Counter
65  signal Counter_Stb : STD_LOGIC;
66
67  -- Register Signals ---------------------------
```

8. At **line 78** add the signals (registers) that the counter will be using. **Counter_EN** will enable the counter, **Counter_Inc** will determine if the counter is incrementing or not, and **Counter_Reg** is the binary counter.



```
76  signal IOA: STD_LOGIC_VECTOR (7 downto 0);
77
78  -- The Counter's Signals
79  -- Enable the counter for use
80  signal Counter_EN : STD_LOGIC;
81  -- Increment the Counter by one
82  signal Counter_Inc : STD_LOGIC;
83  -- The Counter's Register
84  signal Counter_Reg : STD_LOGIC_Vector(3 downto 0);
85
86  -- I/O component for detection of Change of State Ir
```

9. At around **line 231** we will insert the counter's write strobe. This will pulse **Counter_Stb** when there is a write command to the **Counter_Adr.**



```
232     -- The Counter Register's Write Strobes
233     process (CLK)
234     begin
235         if (CLK'event and CLK = '1') then
236            Counter_Stb <= Counter_Adr and not ADS_n and
237                           not LBE0_n and LW_R_n;
238         end if;
239      end process;
240
241        --Interrupt Registers Write Strobes
```

10. At **line 350** there is the process statement to control the Differential Direction Control Register. Add the red code to cause channels 0-3 to become outputs when there is a write to the counter address, and make sure that channel 4 is an input to handle the increment line.

```
--Front I/O Differential Direction Control Register 0x8008
process (CLK, RESET)
begin
    if (RESET = '1') then
       DiffDir_Reg(7 downto 0) <= "00000000";
    elsif (CLK'event and CLK = '1') then
       if (DiffDirReg_Stb0 = '1') then
          DiffDir_Reg(7 downto 0) <= LD(7 downto 0);
       -- If there is a Counter_Stb pre-config the direction
       -- to channel 4 as an input and channels 3-0 as outputs
       elsif (Counter_Stb = '1') then
          DiffDir_Reg(7 downto 0) <= "00001111";
       else
          DiffDir_Reg(7 downto 0) <= DiffDir_Reg(7 downto 0);
       end if;
    end if;
end process;
```

11. Add this process statement at **line 434** to handle the enable for the counter. Notice that **Counter_EN** receives its information from the local data bus (LD) bit-5.

```
-- Counter_EN Register 0x8020 bit 5
-- Turns on the functionality of the Counter
process (CLK, RESET)
begin
    if (RESET = '1') then
       Counter_EN <= '0';
    elsif (CLK'event and CLK = '1') then
       if (Counter_Stb = '1') then
          Counter_EN <= LD(5);
       else
          Counter_EN <= Counter_EN;
       end if;
    end if;
 end process;
```

12. Add this process statement at **line 449** to handle the external increment line for the counter. Notice that the **Counter_Inc** receives its information from channel 4. The counter is stopped and started using this input line.

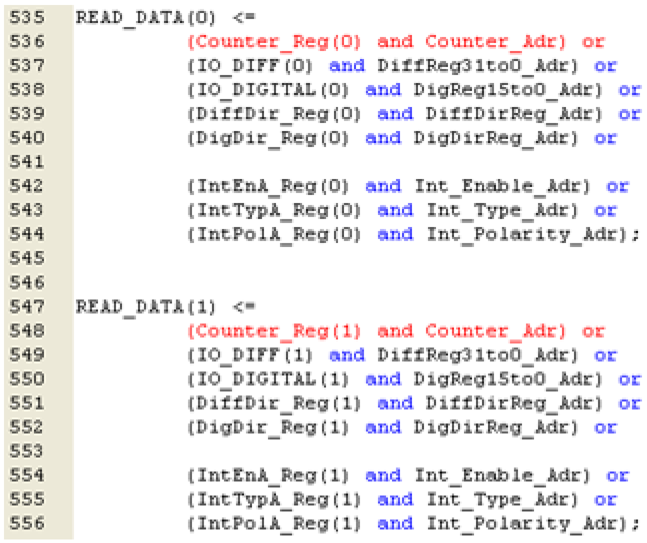
```
-- Counter_Inc determine when to load the counter
-- Register 0x8020 bit 4
process (CLK, RESET)
begin
    if (RESET = '1') then
       Counter_Inc <= '0';
    elsif (CLK'event and CLK = '1') then
       if (Counter_EN = '1') then
          Counter_Inc <= IO_DIGITAL(4);
       else
          Counter_Inc <= Counter_Inc;
       end if;
    end if;
end process;
```

13. Add this process statement at **line 464** to handle the counter. When the counter is enabled, it will check the Counter_Inc line to see if it has a positive logic equivalence of ‘1’ every positive clock edge. If it does then the counter will be incremented.

```
-- Counter_Reg determine when to increment the counter
process (CLK, RESET)
begin
    if (RESET = '1') then
       Counter_Reg <= "0000";
    elsif (CLK'event and CLK = '1') then
       if (Counter_EN = '1' and Counter_Inc = '1') then
          Counter_Reg <= Counter_Reg + 1;
       else
          Counter_Reg <= Counter_Reg;
       end if;
    end if;
end process;
```

14. Add the following lines of red code to the **READ_DATA MUX**. This will allow the read and write commands to access the counter address at 8020H.

Bits 3-0 will hold the four bits of the counter, bit 4 will hold the increment line, and bit 5 will hold the enable.

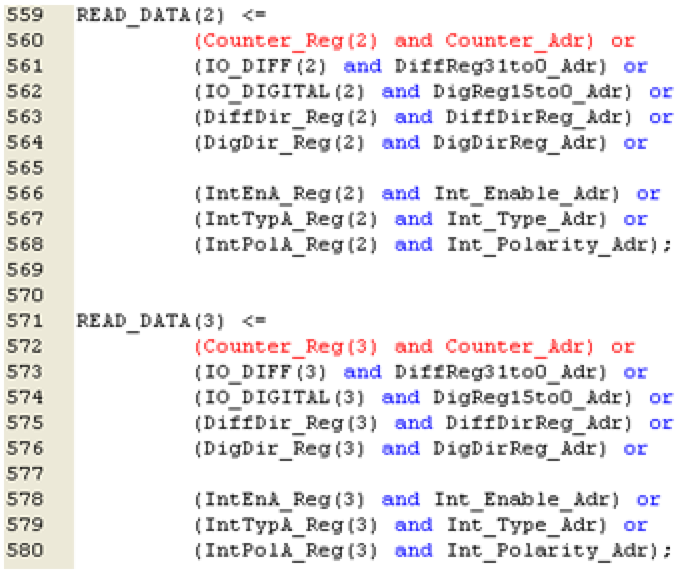


```
READ_DATA(0) <=
         (Counter_Reg(0) and Counter_Adr) or
         (IO_DIFF(0) and DiffReg31to0_Adr) or
         (IO_DIGITAL(0) and DigReg15to0_Adr) or
         (DiffDir_Reg(0) and DiffDirReg_Adr) or
         (DigDir_Reg(0) and DigDirReg_Adr) or

         (IntEnA_Reg(0) and Int_Enable_Adr) or
         (IntTypA_Reg(0) and Int_Type_Adr) or
         (IntPolA_Reg(0) and Int_Polarity_Adr);


READ_DATA(1) <=
         (Counter_Reg(1) and Counter_Adr) or
         (IO_DIFF(1) and DiffReg31to0_Adr) or
         (IO_DIGITAL(1) and DigReg15to0_Adr) or
         (DiffDir_Reg(1) and DiffDirReg_Adr) or
         (DigDir_Reg(1) and DigDirReg_Adr) or

         (IntEnA_Reg(1) and Int_Enable_Adr) or
         (IntTypA_Reg(1) and Int_Type_Adr) or
         (IntPolA_Reg(1) and Int_Polarity_Adr);
```

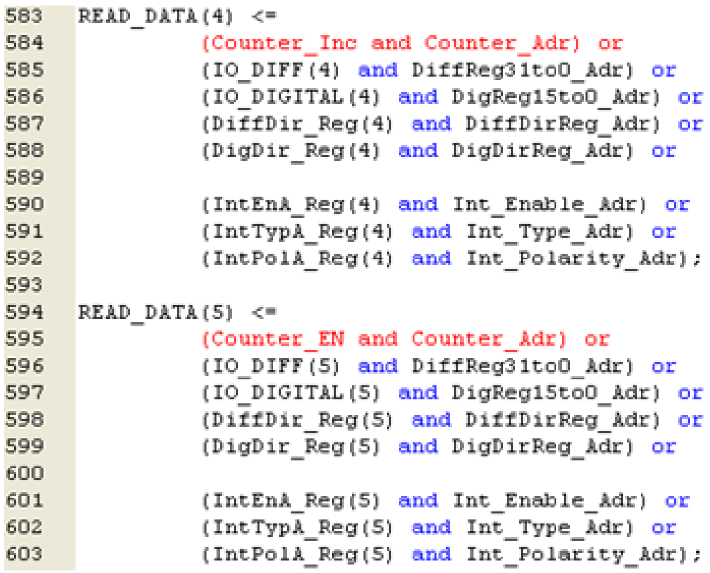


```
READ_DATA(2) <=
         (Counter_Reg(2) and Counter_Adr) or
         (IO_DIFF(2) and DiffReg31to0_Adr) or
         (IO_DIGITAL(2) and DigReg15to0_Adr) or
         (DiffDir_Reg(2) and DiffDirReg_Adr) or
         (DigDir_Reg(2) and DigDirReg_Adr) or

         (IntEnA_Reg(2) and Int_Enable_Adr) or
         (IntTypA_Reg(2) and Int_Type_Adr) or
         (IntPolA_Reg(2) and Int_Polarity_Adr);


READ_DATA(3) <=
         (Counter_Reg(3) and Counter_Adr) or
         (IO_DIFF(3) and DiffReg31to0_Adr) or
         (IO_DIGITAL(3) and DigReg15to0_Adr) or
         (DiffDir_Reg(3) and DiffDirReg_Adr) or
         (DigDir_Reg(3) and DigDirReg_Adr) or

         (IntEnA_Reg(3) and Int_Enable_Adr) or
         (IntTypA_Reg(3) and Int_Type_Adr) or
         (IntPolA_Reg(3) and Int_Polarity_Adr);
```



```
READ_DATA(4) <=
         (Counter_Inc and Counter_Adr) or
         (IO_DIFF(4) and DiffReg31to0_Adr) or
         (IO_DIGITAL(4) and DigReg15to0_Adr) or
         (DiffDir_Reg(4) and DiffDirReg_Adr) or
         (DigDir_Reg(4) and DigDirReg_Adr) or

         (IntEnA_Reg(4) and Int_Enable_Adr) or
         (IntTypA_Reg(4) and Int_Type_Adr) or
         (IntPolA_Reg(4) and Int_Polarity_Adr);

READ_DATA(5) <=
         (Counter_EN and Counter_Adr) or
         (IO_DIFF(5) and DiffReg31to0_Adr) or
         (IO_DIGITAL(5) and DigReg15to0_Adr) or
         (DiffDir_Reg(5) and DiffDirReg_Adr) or
         (DigDir_Reg(5) and DigDirReg_Adr) or

         (IntEnA_Reg(5) and Int_Enable_Adr) or
         (IntTypA_Reg(5) and Int_Type_Adr) or
         (IntPolA_Reg(5) and Int_Polarity_Adr);
```

Simulating the Example Design Using ISIM

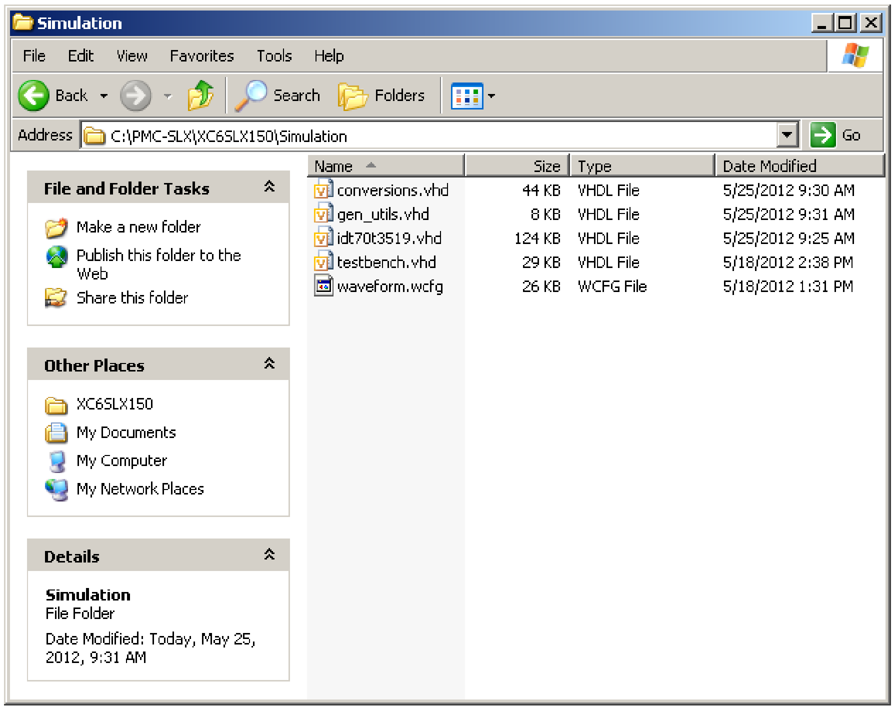
Xilinx ISE Design Suite provides an ISim simulator as a means to simulate designs. This tutorial will take you through the procedure for simulating the example design, reading and writing to the Dual-Port SRAM. We use a Dual-Port SRAM VHDL model provided by the Free Model Foundry.

1. Make a new folder in the the previously created XC6SLX150 directory and call it **Simulation**.

2. From the Simulation folder, copy the .vhd file and the .wcfg file into the new Simulation folder.

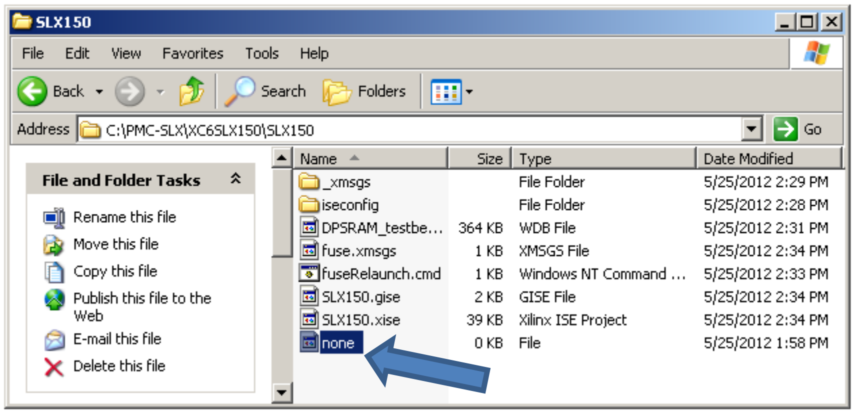
3. To obtain the DP-SRAM model go to http://www.freemodelfoundry.com/ram.php and download **idt70t3519.vhd**. Save it in the Simulation folder.

4. To get the library files for the model, go to http://www.freemodelfoundry.com/packages.php and download **conversions.vhd** and **gen_utils.vhd**. Save them both in the Simulation folder. The Simulation folder should now contain all of the files shown in the adjacent screenshot.



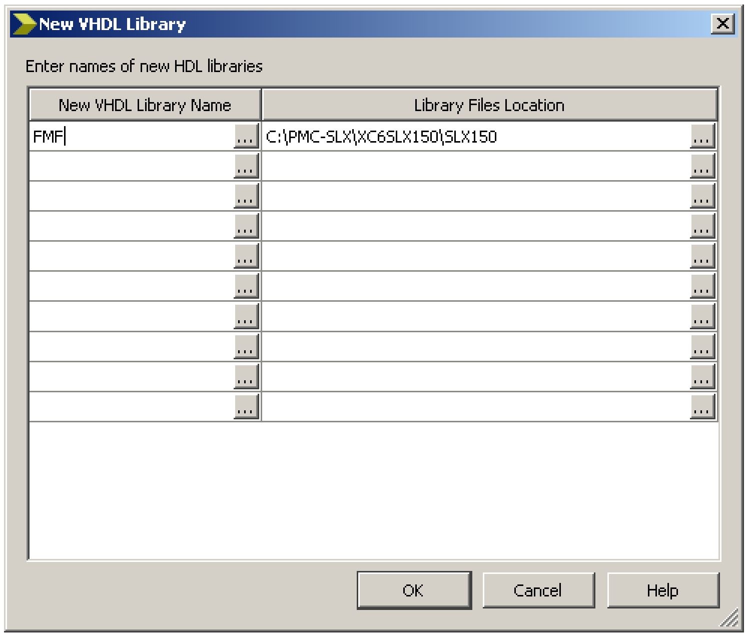
Name	Size	Type	Date Modified
conversions.vhd	44 KB	VHDL File	5/25/2012 9:30 AM
gen_utils.vhd	8 KB	VHDL File	5/25/2012 9:31 AM
idt70t3519.vhd	124 KB	VHDL File	5/25/2012 9:25 AM
testbench.vhd	29 KB	VHDL File	5/18/2012 2:38 PM
waveform.wcfg	26 KB	WCFG File	5/18/2012 1:31 PM

5. For simulation with the DP-SRAM model we must create an empty file and call it **none** inside the SLX150 folder as shown in the screen shot to the right (This can not be a text file. No .txt extention.)



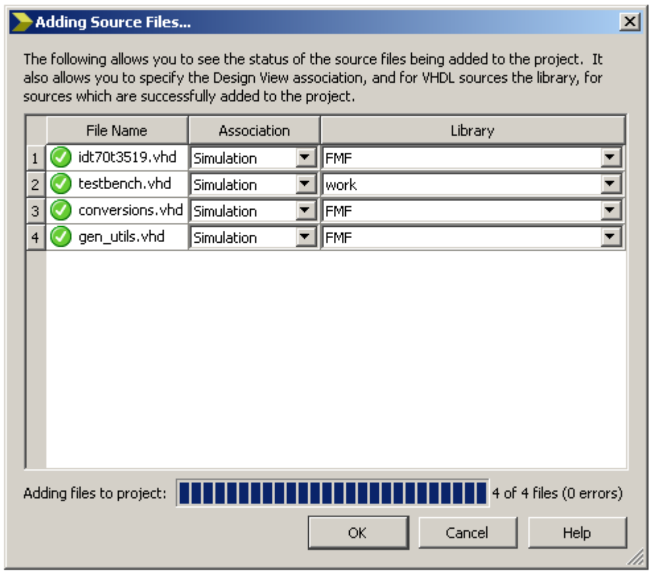
Name	Size	Type	Date Modified
_xmsgs		File Folder	5/25/2012 2:29 PM
iseconfig		File Folder	5/25/2012 2:28 PM
DPSRAM_testbe...	364 KB	WDB File	5/25/2012 2:31 PM
fuse.xmsgs	1 KB	XMSGS File	5/25/2012 2:34 PM
fuseRelaunch.cmd	1 KB	Windows NT Command ...	5/25/2012 2:33 PM
SLX150.gise	2 KB	GISE File	5/25/2012 2:34 PM
SLX150.xise	39 KB	Xilinx ISE Project	5/25/2012 2:34 PM
none	0 KB	File	5/25/2012 1:58 PM

6. Next we will add the files needed for simulation to the previously created project. Follow these steps:
   a. Select **Project → Add Source**.
   b. Select the .vhd files from the Simulation folder.
   c. Click Open.
   d. The association for all four files should be *Simulation*.
   e. Click the drop-down menu for Library to select *New VHDL Library*. The New VHDL Library window will open.
   f. Create the new library in the SLX150 folder and name it **FMF** as shown in the adjacent screenshot.



Enter names of new HDL libraries

New VHDL Library Name	Library Files Location
FMF	C:\PMC-SLX\XC6SLX150\SLX150



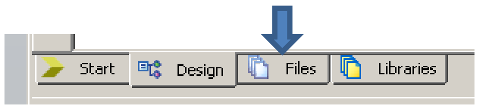
The following allows you to see the status of the source files being added to the project. It also allows you to specify the Design View association, and for VHDL sources the library, for sources which are successfully added to the project.

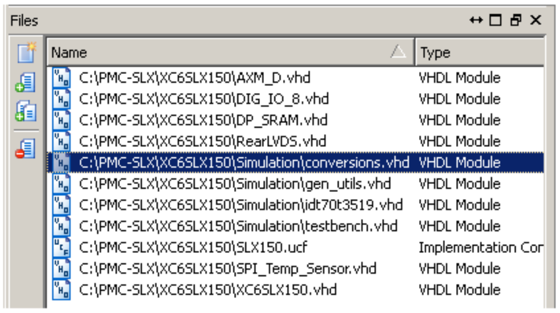
	File Name	Association	Library
1	idt70t3519.vhd	Simulation	FMF
2	testbench.vhd	Simulation	work
3	conversions.vhd	Simulation	FMF
4	gen_utils.vhd	Simulation	FMF

Adding files to project: 4 of 4 files (0 errors)

   g. Back in the Adding Source Files window, change the libraries for the .vhd file as shown in the screenshot on the right. Testbench.vhd should be the only one with library work.
   h. Click OK.

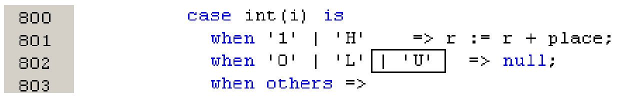
7. Next we need to edit conversion.vhd. Follow these steps:
   a. Click the **Files** tab.
   b. Resize the Columns to view the names of the files and double-click **conversions.vhd** to open it.




Files

Name	Type
C:\PMC-SLX\XC6SLX150\AXM_D.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\DIG_IO_8.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\DP_SRAM.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\RearLVDS.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\Simulation\conversions.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\Simulation\gen_utils.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\Simulation\idt70t3519.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\Simulation\testbench.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\SLX150.ucf	Implementation Cor
C:\PMC-SLX\XC6SLX150\SPI_Temp_Sensor.vhd	VHDL Module
C:\PMC-SLX\XC6SLX150\XC6SLX150.vhd	VHDL Module

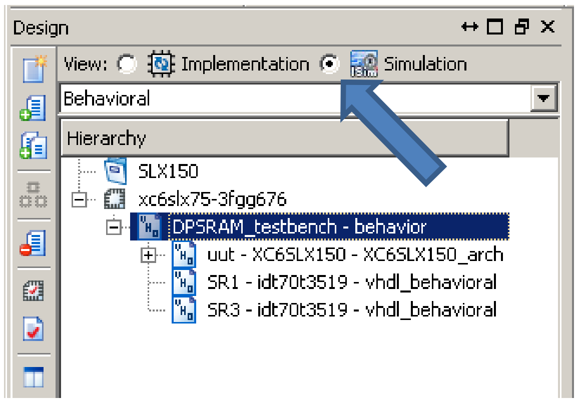
   c. Modify line 802 as shown in the screenshot to the right.
   d. Save and close conversions.vhd.



```
800      case int(i) is
801        when '1' | 'H'    => r := r + place;
802        when '0' | 'L' | 'U'  => null;
803        when others =>
```

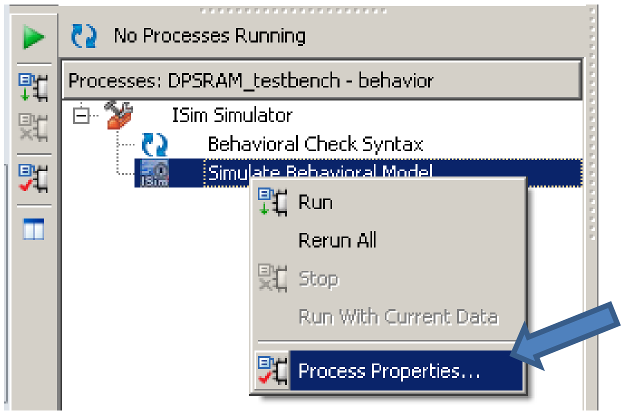
Select the **Design** tab.

8. Click the **Simulation** radio button for ISim.

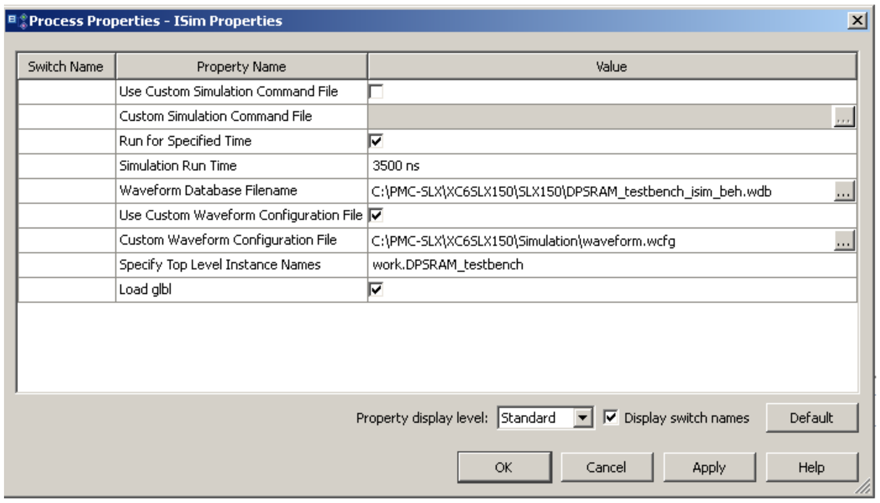

Design
View:
Implementation
Simulation
Behavioral
Hierarchy
SLX150
xc6slx75-3fgg676
DPSRAM_testbench - behavior
uut - XC6SLX150 - XC6SLX150_arch
SR1 - idt70t3519 - vhdl_behavioral
SR3 - idt70t3519 - vhdl_behavioral

9. The design hierarchy for simulation should appear in the window as shown in the adjacent screenshot. Click on **DPSRAM_testbench - behavior** to highlight it.

10. In the Processes Window, expand **ISim Simulator** and right-click *Simulate Behavioral Model*. Select *Process Properties*.



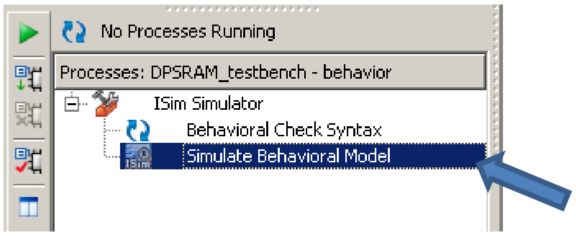
11. The Process Properties window will appear. Change the simulation run time to 3500 ns and check the Use Custom Waveform Configuration box. Browse to the file **waveform.wcfg** in the Simulation folder for the configuration file. Click OK.


Process Properties - ISim Properties

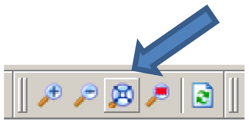
Switch Name	Property Name	Value
	Use Custom Simulation Command File	☐
	Custom Simulation Command File	
	Run for Specified Time	☑
	Simulation Run Time	3500 ns
	Waveform Database Filename	C:\PMC-SLX\XC6SLX150\SLX150\DPSRAM_testbench_isim_beh.wdb
	Use Custom Waveform Configuration File	☑
	Custom Waveform Configuration File	C:\PMC-SLX\XC6SLX150\Simulation\waveform.wcfg
	Specify Top Level Instance Names	work.DPSRAM_testbench
	Load glbl	☑

Property display level: Standard ☑ Display switch names

12. Double-click *Simulate Behavioral Model*.

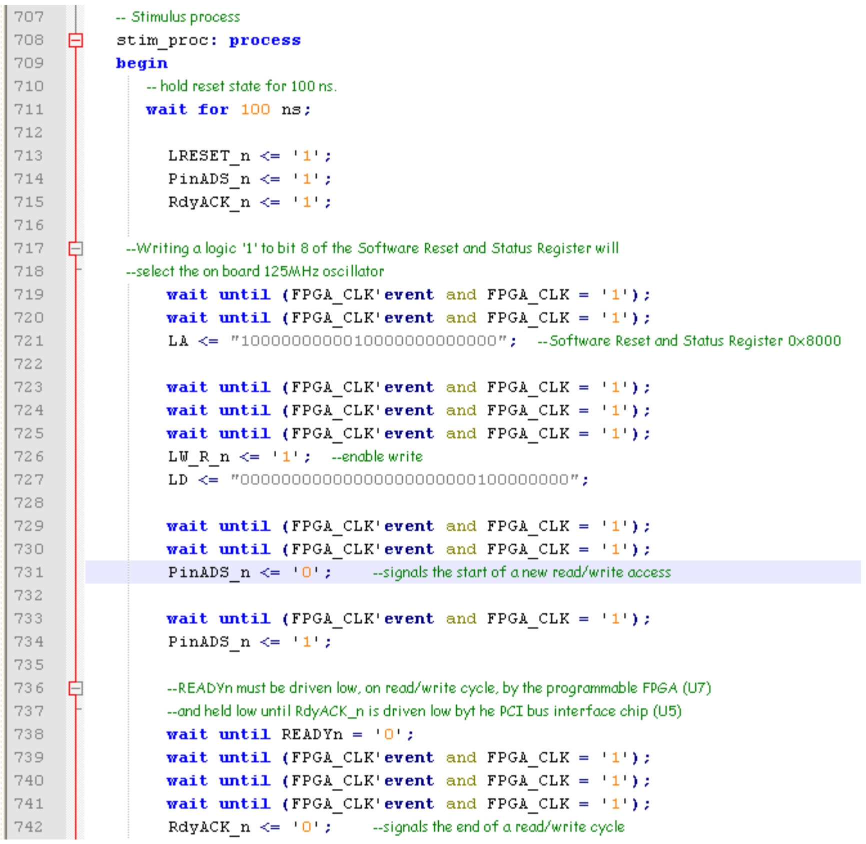

No Processes Running
Processes: DPSRAM_testbench - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model

13. After the simulation is complete and the ISIM window has appeared, click the Zoom to Full View Button in the ISIM toolbar.

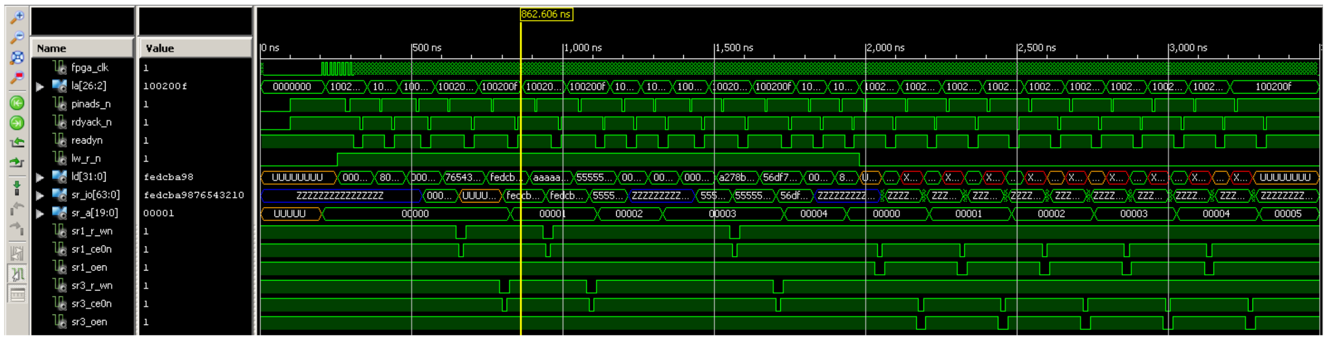


14. The waveforms are now expanded and should look like the screenshot below. Click on the waveforms and a cursor will appear.

Note:
The testbench.vhd file starting at line707 provides additional documentation of the stimulus provided to generate this simulation.

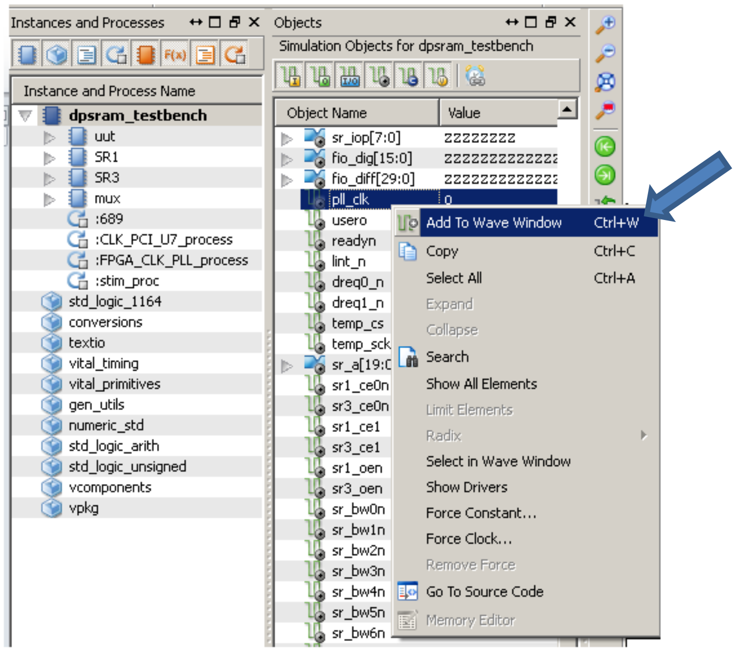


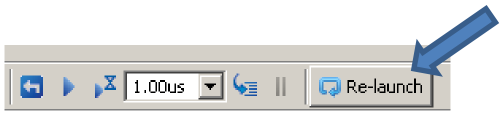
```
707    -- Stimulus process
708    stim_proc: process
709    begin
710        -- hold reset state for 100 ns.
711        wait for 100 ns;
712
713           LRESET_n <= '1';
714           PinADS_n <= '1';
715           RdyACK_n <= '1';
716
717     --Writing a logic '1' to bit 8 of the Software Reset and Status Register will
718     --select the on board 125MHz oscillator
719           wait until (FPGA_CLK'event and FPGA_CLK = '1');
720           wait until (FPGA_CLK'event and FPGA_CLK = '1');
721           LA <= "1000000000010000000000000";   --Software Reset and Status Register 0x8000
722
723           wait until (FPGA_CLK'event and FPGA_CLK = '1');
724           wait until (FPGA_CLK'event and FPGA_CLK = '1');
725           wait until (FPGA_CLK'event and FPGA_CLK = '1');
726           LW_R_n <= '1';  --enable write
727           LD <= "00000000000000000000000100000000";
728
729           wait until (FPGA_CLK'event and FPGA_CLK = '1');
730           wait until (FPGA_CLK'event and FPGA_CLK = '1');
731           PinADS_n <= '0';     --signals the start of a new read/write access
732
733           wait until (FPGA_CLK'event and FPGA_CLK = '1');
734           PinADS_n <= '1';
735
736           --READYn must be driven low, on read/write cycle, by the programmable FPGA (U7)
737           --and held low until RdyACK_n is driven low byt he PCI bus interface chip (U5)
738           wait until READYn = '0';
739           wait until (FPGA_CLK'event and FPGA_CLK = '1');
740           wait until (FPGA_CLK'event and FPGA_CLK = '1');
741           wait until (FPGA_CLK'event and FPGA_CLK = '1');
742           RdyACK_n <= '0';     --signals the end of a read/write cycle
```

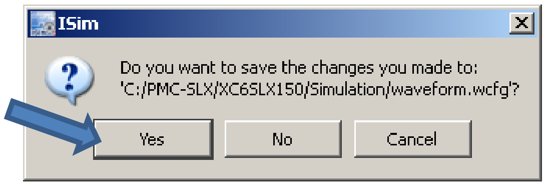


Name	Value
fpga_clk	1
la[26:2]	100200f
pinads_n	1
rdyack_n	1
readyn	1
lw_r_n	1
ld[31:0]	fedcba98
sr_io[63:0]	fedcba9876543210
sr_a[19:0]	00001
sr1_r_wn	1
sr1_ce0n	1
sr1_oen	1
sr3_r_wn	1
sr3_ce0n	1
sr3_oen	1

15. More objects can be added to the waveform window for simulation, for example, we will add the **pll_clk** signal:
    a. In the *Instances and Processes* window, find the correct instance or process that the object belongs to and click on it to highlight it, in this case it is **dpsram_testbench**.
    b. In the *Objects* window, find the object's name, **pll_clk**, right-click the object, and select **Add to Wave Window**.
    c. The name of the object should now appear in the waveform window.
    d. Rerun the simulation; click **Re-launch** at the top of the ISIM window. Be sure to save the new waveform configuration file when prompted.
    e. After simulation is complete, the signal will appear in the waveform window. Signals can also be dragged and dropped to different locations within the waveform window.





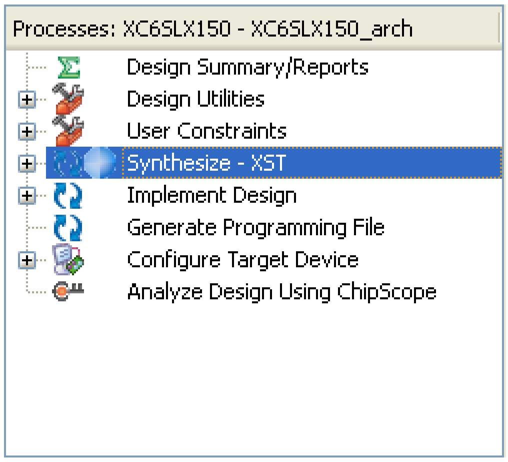

ISim

Do you want to save the changes you made to:
'C:/PMC-SLX/XC6SLX150/Simulation/waveform.wcfg'?

Generating a Programming "MCS" File

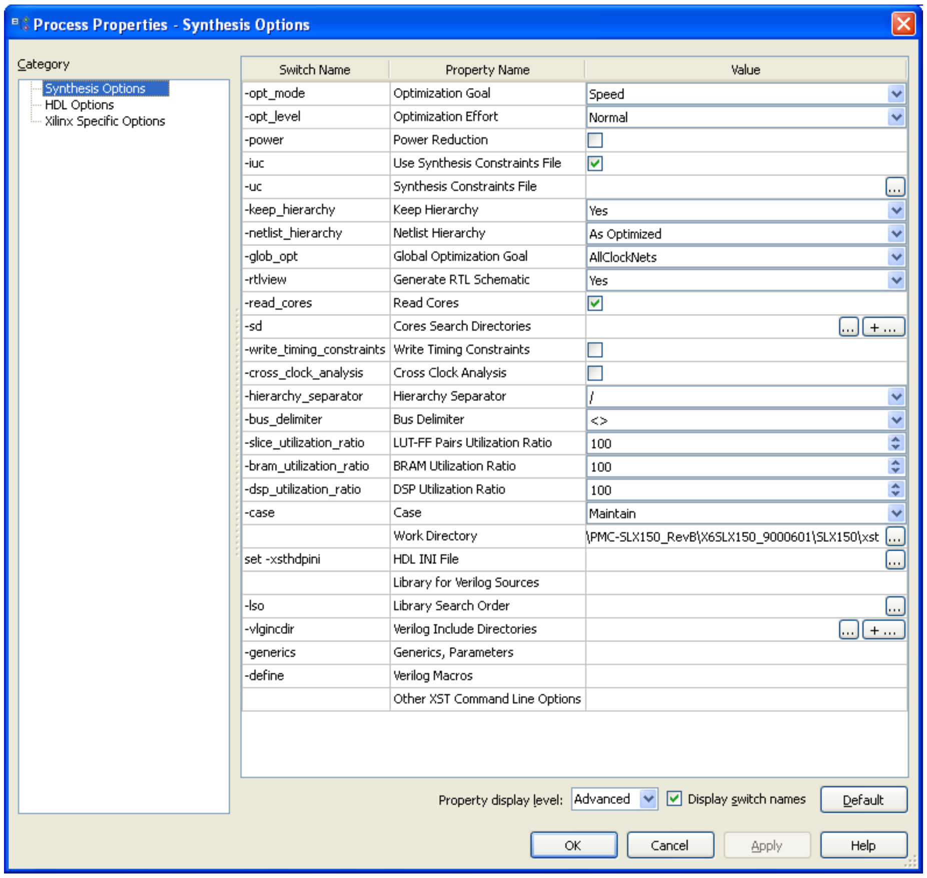
The Xilinx "MCS" contains the information to program the X6SLX150 through either FLASH or JTAG. These instructions will take you through the procedure for creating a MCS file.

1. Select **XC6SLX150-XC6SLX150 _arch (XC6SLX150.vhd)** in the **Hierarchy Window**.



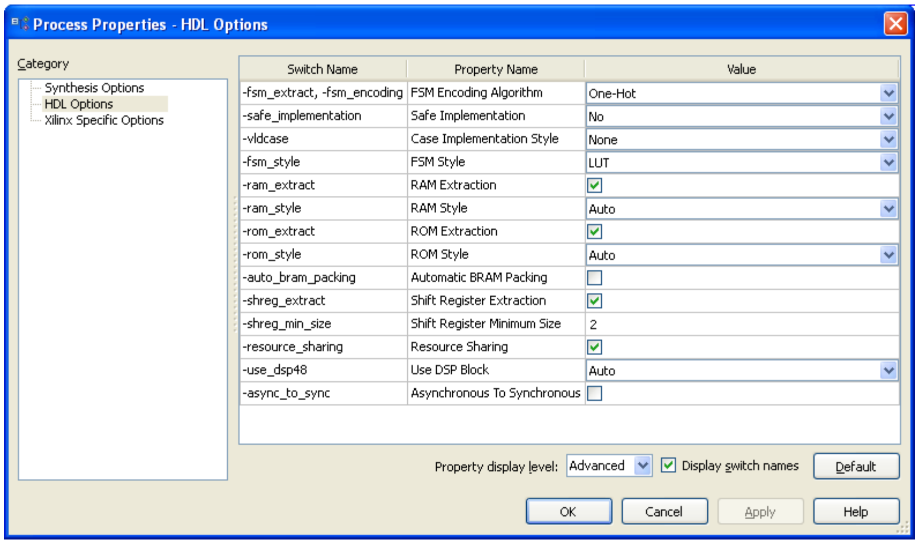
2. Select **Synthesize-XST** in the **Processes Window**. Then select Process Properties and verify the setting shown in figure to right. Then select **Process →Run**.

Note: If there are any errors, correct them and repeat steps 1 and 2. There are 8 warnings.



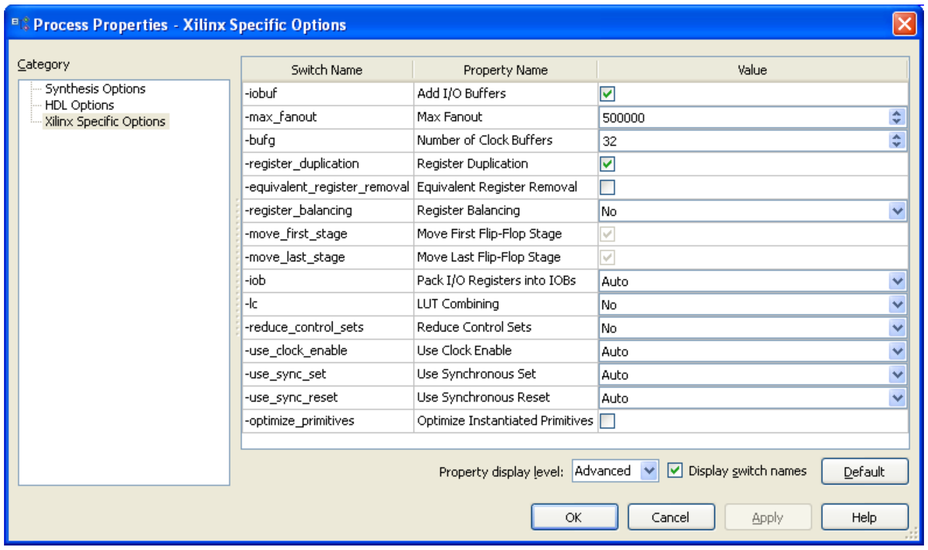
Switch Name	Property Name	Value
-opt_mode	Optimization Goal	Speed
-opt_level	Optimization Effort	Normal
-power	Power Reduction	☐
-iuc	Use Synthesis Constraints File	✓
-uc	Synthesis Constraints File	
-keep_hierarchy	Keep Hierarchy	Yes
-netlist_hierarchy	Netlist Hierarchy	As Optimized
-glob_opt	Global Optimization Goal	AllClockNets
-rtlview	Generate RTL Schematic	Yes
-read_cores	Read Cores	✓
-sd	Cores Search Directories	
-write_timing_constraints	Write Timing Constraints	☐
-cross_clock_analysis	Cross Clock Analysis	☐
-hierarchy_separator	Hierarchy Separator	/
-bus_delimiter	Bus Delimiter	<>
-slice_utilization_ratio	LUT-FF Pairs Utilization Ratio	100
-bram_utilization_ratio	BRAM Utilization Ratio	100
-dsp_utilization_ratio	DSP Utilization Ratio	100
-case	Case	Maintain
	Work Directory	\PMC-SLX150_RevB\X6SLX150_9000601\SLX150\xst
set -xsthdpini	HDL INI File	
	Library for Verilog Sources	
-lso	Library Search Order	
-vlgincdir	Verilog Include Directories	
-generics	Generics, Parameters	
-define	Verilog Macros	
	Other XST Command Line Options	

Property display level: Advanced ✓ Display switch names



Switch Name	Property Name	Value
-fsm_extract, -fsm_encoding	FSM Encoding Algorithm	One-Hot
-safe_implementation	Safe Implementation	No
-vldcase	Case Implementation Style	None
-fsm_style	FSM Style	LUT
-ram_extract	RAM Extraction	✓
-ram_style	RAM Style	Auto
-rom_extract	ROM Extraction	✓
-rom_style	ROM Style	Auto
-auto_bram_packing	Automatic BRAM Packing	☐
-shreg_extract	Shift Register Extraction	✓
-shreg_min_size	Shift Register Minimum Size	2
-resource_sharing	Resource Sharing	✓
-use_dsp48	Use DSP Block	Auto
-async_to_sync	Asynchronous To Synchronous	☐

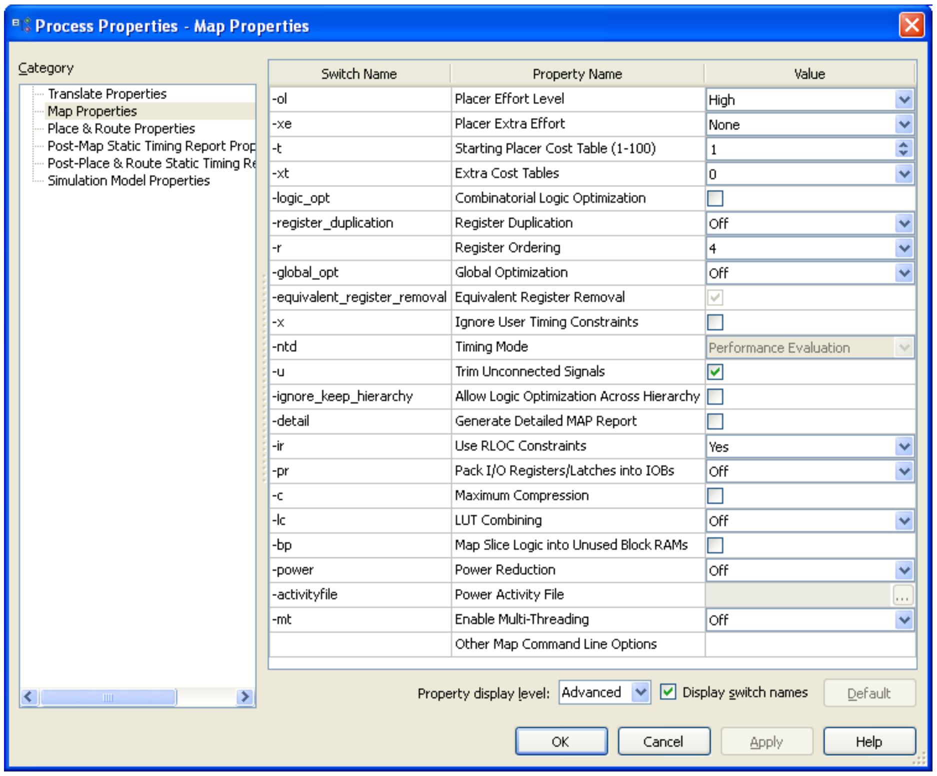
Property display level: Advanced ✓ Display switch names



Switch Name	Property Name	Value
-iobuf	Add I/O Buffers	✓
-max_fanout	Max Fanout	500000
-bufg	Number of Clock Buffers	32
-register_duplication	Register Duplication	✓
-equivalent_register_removal	Equivalent Register Removal	☐
-register_balancing	Register Balancing	No
-move_first_stage	Move First Flip-Flop Stage	✓
-move_last_stage	Move Last Flip-Flop Stage	✓
-iob	Pack I/O Registers into IOBs	Auto
-lc	LUT Combining	No
-reduce_control_sets	Reduce Control Sets	No
-use_clock_enable	Use Clock Enable	Auto
-use_sync_set	Use Synchronous Set	Auto
-use_sync_reset	Use Synchronous Reset	Auto
-optimize_primitives	Optimize Instantiated Primitives	☐

Property display level: Advanced ✓ Display switch names

3. Select **Implement Design** in the **Processes Window**. Then select Process → **Process Properties**. Verify the settings shown in the figure to the right..
4. Select OK
5. Select **Implement Design** in the **Processes Window**. Then right mouse select Process → **Run**
6. **Note:** Translate, Map, and Place and Route should complete with no errors and no additional warnings.

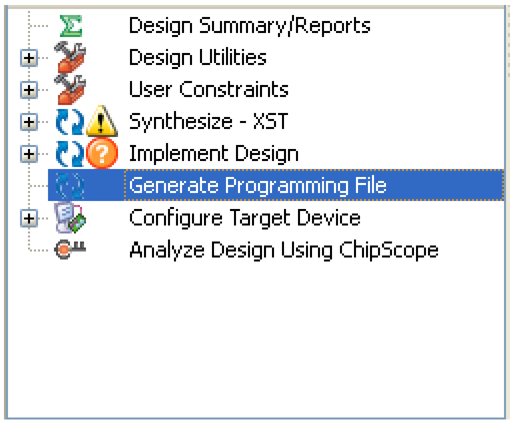

Process Properties - Map Properties
Category
Translate Properties
Map Properties
Place & Route Properties
Post-Map Static Timing Report Prop
Post-Place & Route Static Timing Re
Simulation Model Properties
OK
Cancel
Apply
Help
Default

Switch Name	Property Name	Value
-ol	Placer Effort Level	High
-xe	Placer Extra Effort	None
-t	Starting Placer Cost Table (1-100)	1
-xt	Extra Cost Tables	0
-logic_opt	Combinatorial Logic Optimization	☐
-register_duplication	Register Duplication	Off
-r	Register Ordering	4
-global_opt	Global Optimization	Off
-equivalent_register_removal	Equivalent Register Removal	✓
-x	Ignore User Timing Constraints	☐
-ntd	Timing Mode	Performance Evaluation
-u	Trim Unconnected Signals	✓
-ignore_keep_hierarchy	Allow Logic Optimization Across Hierarchy	☐
-detail	Generate Detailed MAP Report	☐
-ir	Use RLOC Constraints	Yes
-pr	Pack I/O Registers/Latches into IOBs	Off
-c	Maximum Compression	☐
-lc	LUT Combining	Off
-bp	Map Slice Logic into Unused Block RAMs	☐
-power	Power Reduction	Off
-activityfile	Power Activity File	
-mt	Enable Multi-Threading	Off
	Other Map Command Line Options	

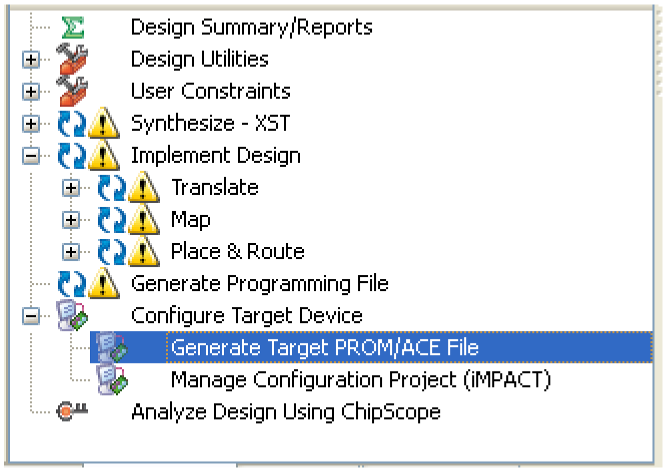
Property display level: Advanced
✓ Display switch names

7. Select **XC6SLX150-XC6SLX150 _arch (XC6SLX150.vhd)** in the **Hierarchy Window**.

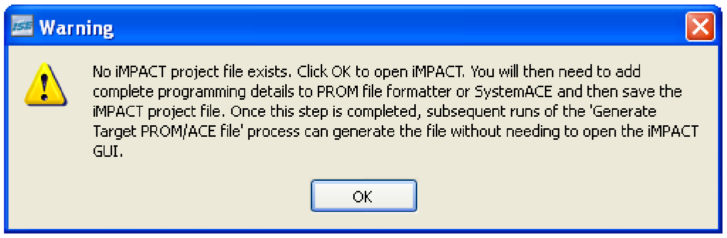
8. Select **Generate Programming File** in the **Processes Window**. Then select **Process** →**Run**.



9. Expand Configure Target Device and then Right-click on **Generate Target PROM/ACE File,** and click on **Run**.

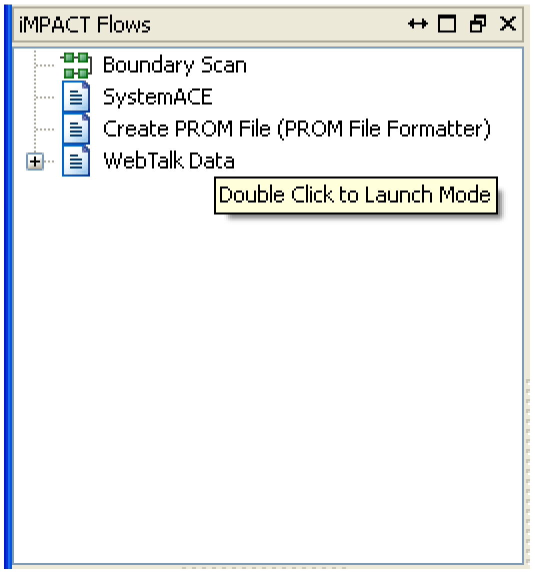


10. Select **OK**.

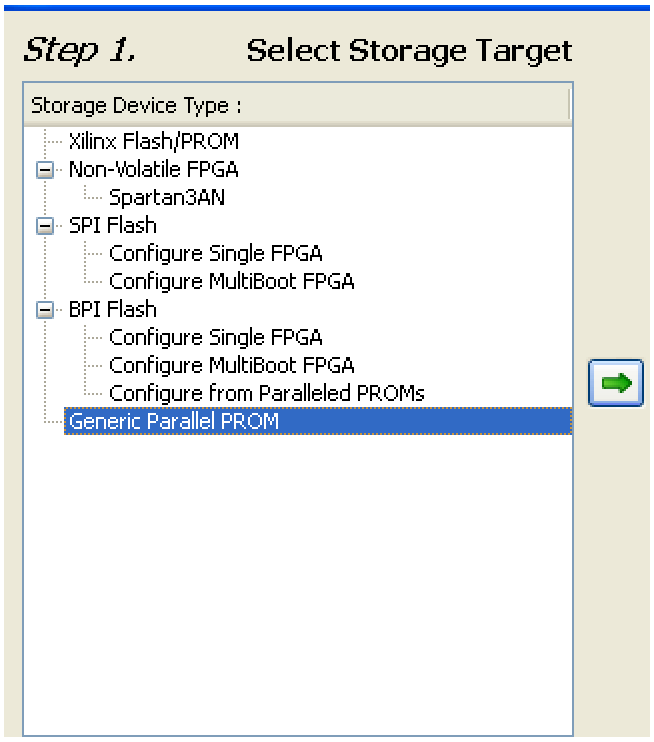

Warning

No iMPACT project file exists. Click OK to open iMPACT. You will then need to add complete programming details to PROM file formatter or SystemACE and then save the iMPACT project file. Once this step is completed, subsequent runs of the 'Generate Target PROM/ACE file' process can generate the file without needing to open the iMPACT GUI.

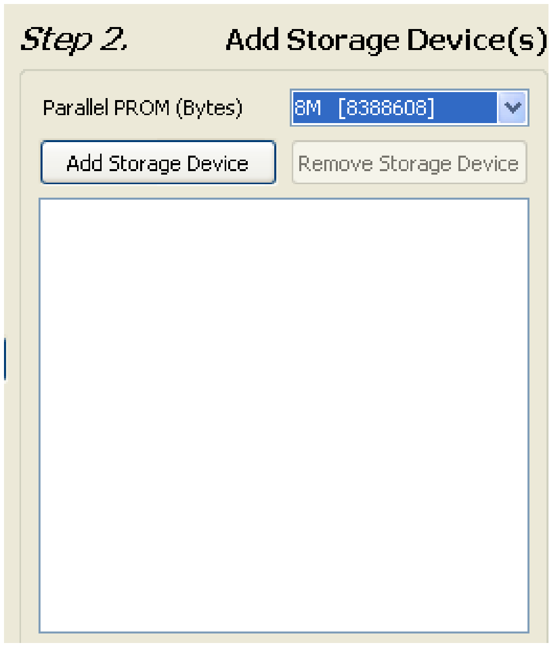
11. Double select **Create PROM File (PROM File Formatter)**


iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Formatter)
WebTalk Data
Double Click to Launch Mode

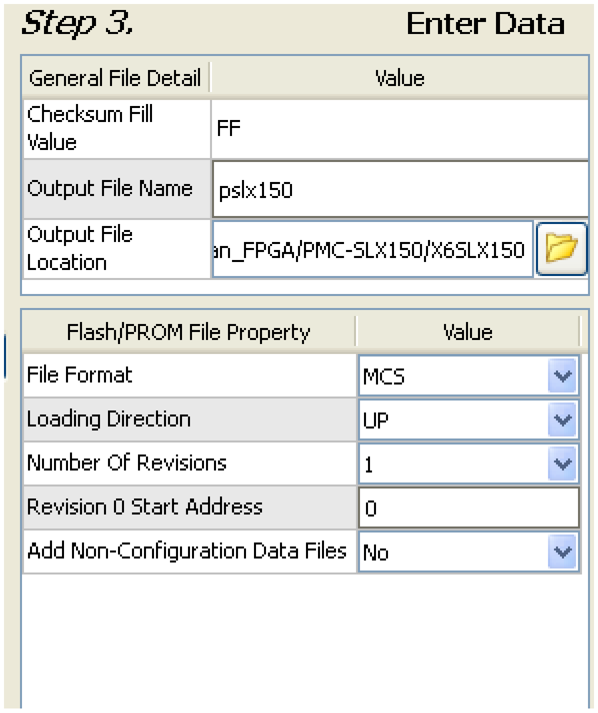
12. Select **Generic Parallel PROM** and the select the green arrow.



13. Select **8M** and then select **Add Storage Device**. Then select the next green arrow.



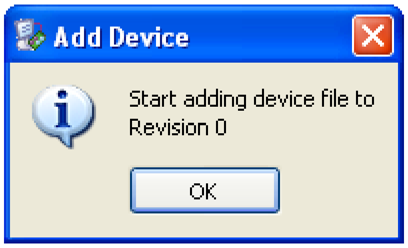
14. Verify the Output File location and enter an Output File Name. For example PSLX150. Leave all other default options and select **OK**.


Step 3. Enter Data

General File Detail	Value
Checksum Fill Value	FF
Output File Name	pslx150
Output File Location	an_FPGA/PMC-SLX150/X6SLX150

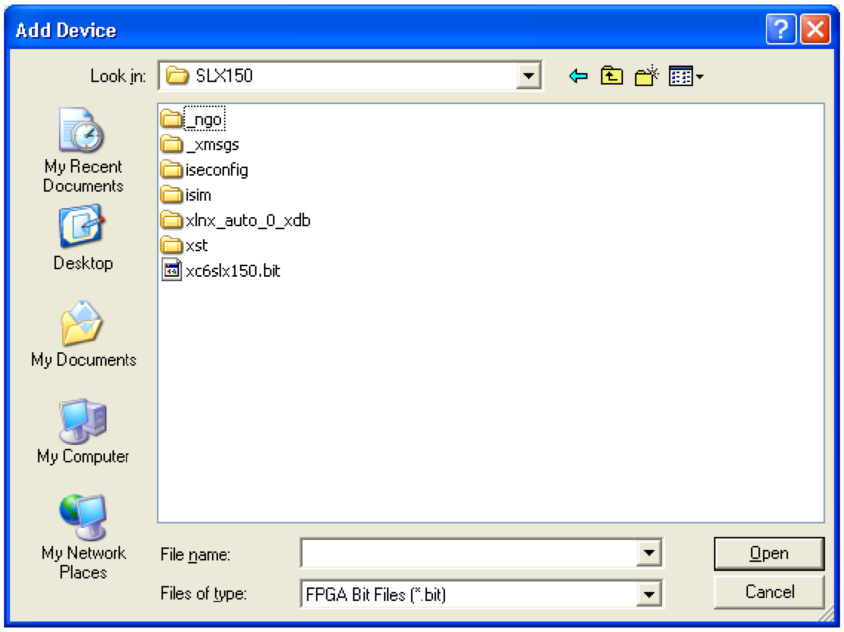
Flash/PROM File Property	Value
File Format	MCS
Loading Direction	UP
Number Of Revisions	1
Revision 0 Start Address	0
Add Non-Configuration Data Files	No

15. Click **OK**.

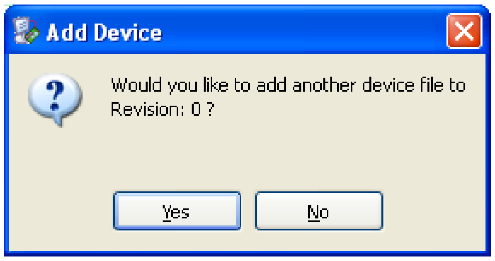

Add Device

Start adding device file to Revision 0

16. Select **xc6slx150.bit** file and then click **Open**.

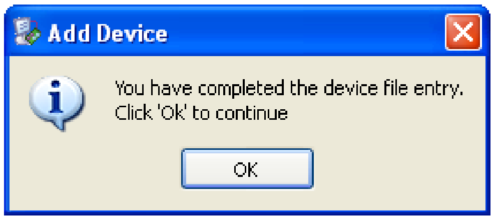


17. Click **NO**.



Would you like to add another device file to Revision: 0 ?

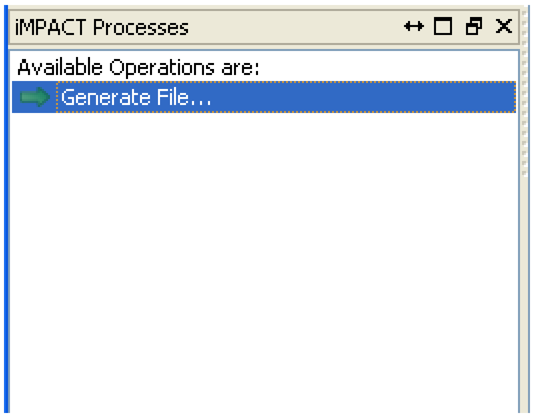
18. Click **OK**.



You have completed the device file entry. Click 'Ok' to continue

19. Click **OK** again.

20. In the left pane iMPACT Process Windows Double click **Generate File.**



If the process completed with no errors the *Generate Succeeded* message will be displayed.

The PSLX150.mcs file now resides in the targeted directory. From here the file can be downloaded to FLASH or directly to the FPGA using Acromag's software demonstration program for Windows or Linux. The file can also be downloaded via JTAG via the AXM-EDK adapter board when used in conjunction with a compatible Xilinx download cable.

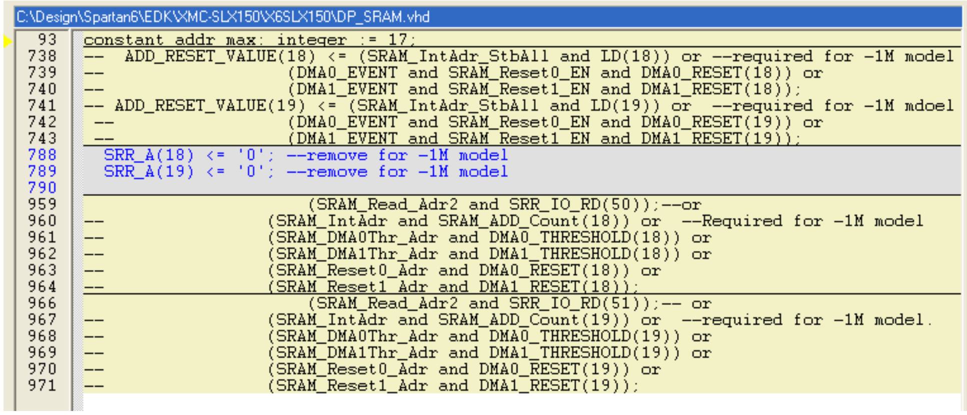
Differences in VHDL between the XMC-SLX150 and the XMC-SLX150-1M

The Dual-Port SRAM on the XMC-SLX150-1M is four times the size as on the base model. The result is that all registers that manage or use the address of the SRAM must have two extra bits. The majority of these registers are defined via a constant declaration on line 93 of DP_SRAM.vhd. The declaration is **constant addr_max: integer := 19;** (or 17 for the base model). The registers affected are listed below.

Register/Counter	Description
DMA0_THRESHOLD	Register corresponding to 0x8048.
DMA1_THRESHOLD	Register corresponding to 0x804C.
DMA0_RESET	Register corresponding to 0x8050.
DMA1_RESET	Register corresponding to 0x8054.
SRAM_ADD_Count	Counter with the current address of the SRAM. Can be set and read via 0x8038 & 0x803C.
ADD_RESET_VALUE	Counter that contains a reset value for the SRAM which could from either of the RESET registers above..

In addition, there are some minor changes to the read logic. All differences are shown on the following page. Please note that there are no differences in the top level *x6slx150.vhd* file.

DP_SRAM.vhd for XMC-SLX150

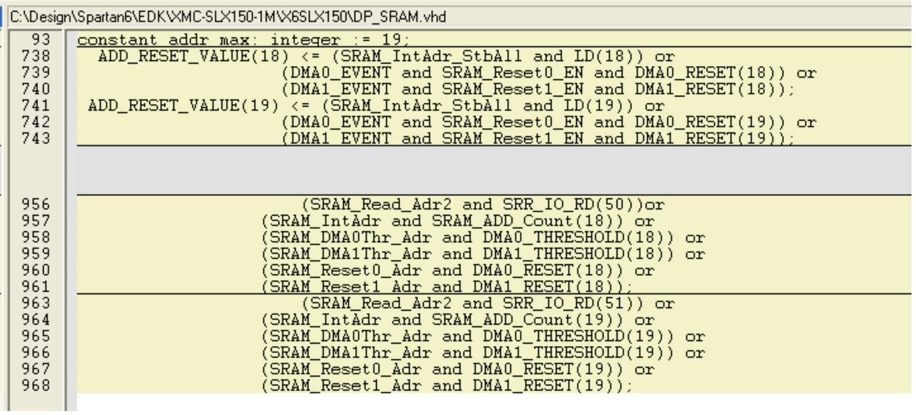


C:\Design\Spartan6\EDK\XMC-SLX150\X6SLX150\DP_SRAM.vhd

```
constant addr_max: integer := 17;
--  ADD_RESET_VALUE(18) <= (SRAM_IntAdr_StbAll and LD(18)) or --required for -1M model
--                         (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(18)) or
--                         (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(18));
-- ADD_RESET_VALUE(19) <= (SRAM_IntAdr_StbAll and LD(19)) or  --required for -1M mdoel
 --                        (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(19)) or
 --                        (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(19));
  SRR_A(18) <= '0'; --remove for -1M model
  SRR_A(19) <= '0'; --remove for -1M model

                             (SRAM_Read_Adr2 and SRR_IO_RD(50));--or
--                       (SRAM_IntAdr and SRAM_ADD_Count(18)) or  --Required for -1M model
--                       (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(18)) or
--                       (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(18)) or
--                       (SRAM_Reset0_Adr and DMA0_RESET(18)) or
--                       (SRAM_Reset1_Adr and DMA1_RESET(18));
                             (SRAM_Read_Adr2 and SRR_IO_RD(51));-- or
--                       (SRAM_IntAdr and SRAM_ADD_Count(19)) or  --required for -1M model.
--                       (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(19)) or
--                       (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(19)) or
--                       (SRAM_Reset0_Adr and DMA0_RESET(19)) or
--                       (SRAM_Reset1_Adr and DMA1_RESET(19));
```

DP_SRAM.vhd for XMC-SLX150-1M



C:\Design\Spartan6\EDK\XMC-SLX150-1M\X6SLX150\DP_SRAM.vhd

```
constant addr_max: integer := 19;
  ADD_RESET_VALUE(18) <= (SRAM_IntAdr_StbAll and LD(18)) or
                         (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(18)) or
                         (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(18));
 ADD_RESET_VALUE(19) <= (SRAM_IntAdr_StbAll and LD(19)) or
                         (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(19)) or
                         (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(19));

                             (SRAM_Read_Adr2 and SRR_IO_RD(50))or
                         (SRAM_IntAdr and SRAM_ADD_Count(18)) or
                         (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(18)) or
                         (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(18)) or
                         (SRAM_Reset0_Adr and DMA0_RESET(18)) or
                         (SRAM_Reset1_Adr and DMA1_RESET(18));
                             (SRAM_Read_Adr2 and SRR_IO_RD(51)) or
                         (SRAM_IntAdr and SRAM_ADD_Count(19)) or
                         (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(19)) or
                         (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(19)) or
                         (SRAM_Reset0_Adr and DMA0_RESET(19)) or
                         (SRAM_Reset1_Adr and DMA1_RESET(19));
```

Xilinx ISE 13.2 Compiler Warnings

Note that ISE 13.2 will generate the following warnings when compiling the example design. These warnings can be safety ignored as they reference unused signals.

- WARNING:Xst:647 - Input <LD<18:30>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
- WARNING:Xst:647 - Input <LD<31:30>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
- WARNING:Xst:2677 - Node <Latch_data_0> of sequential type is unconnected in block <Temp_sensor>.
- WARNING:Xst:2677 - Node <Latch_data_1> of sequential type is unconnected in block <Temp_sensor>.
- WARNING:Xst:2677 - Node <Latch_data_2> of sequential type is unconnected in block <Temp_sensor>.
- WARNING:Xst:2677 - Node <Latch_data_0> of sequential type is unconnected in block <SPI_Temp_Sensor>.
- WARNING:Xst:2677 - Node <Latch_data_1> of sequential type is unconnected in block <SPI_Temp_Sensor>.
- WARNING:Xst:2677 - Node <Latch_data_2> of sequential type is unconnected in block <SPI_Temp_Sensor>.