



Series XMC-SLX150
Spartan 6 Based FPGA
XMC Module

Getting Started with the XMC-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

Copyright 2011, Acromag, Inc., Printed in the USA.
Data and specifications are subject to change without notice.

8500-894-A11B000

Table of Contents

GETTING STARTED..... 2

Installing the Board and Device Drivers 2

Starting a New Xilinx Project 3

Modifying the Provided VHDL Code 7

 Example Change 8

Generating a Programming “MCS” File 12

Differences in VHDL between the XMC-SLX150 and the XMC-SLX150-1M 16

Xilinx ISE 12.3 Compiler Warnings 17

All trademarks are the property of their respective owners.

You must consider the possible negative effects of power, wiring, component, sensor, or software failure in the design of any type of control or monitoring system. This is very important where property loss or human life is involved. It is important that you perform satisfactory overall system design and it is agreed between you and Acromag, that this is your responsibility.

Getting Started

The purpose of this document is to provide basic instructions on using the “XMC-SLX Engineering Design Kit” with the XMC-SLX Boards. It will focus on programming the FPGA of the XMC-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 XMC-SLX150(E) and the XMC-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 **XMC-SLX Engineering Design Kit** and copy the **SLX150** folder to your computer.

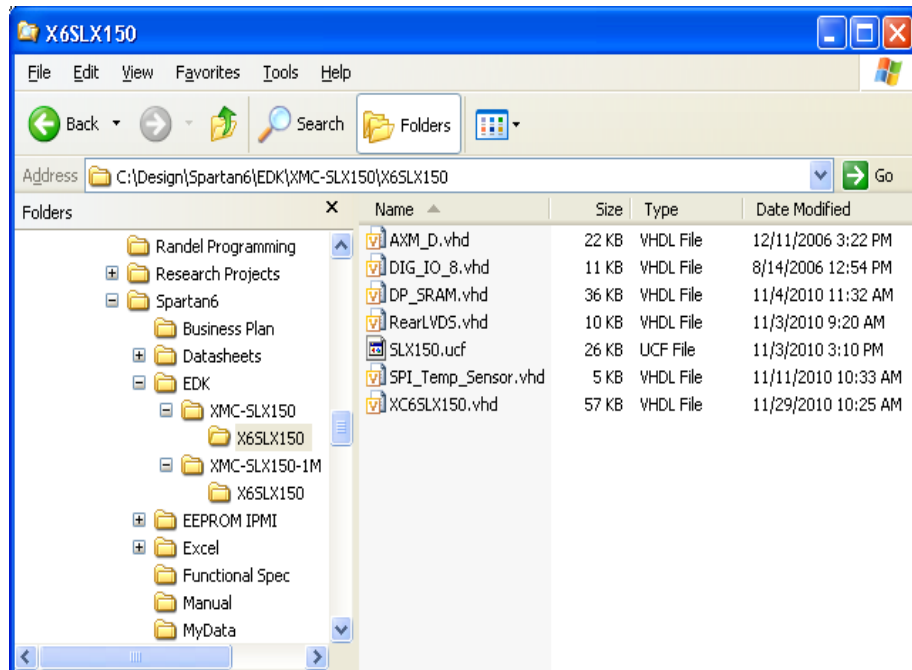
Before you start, familiarize yourself with the **XMC-SLX User’s Manual** included on the EDK CD and the **PCieSLX Driver Function Reference** included on the PCI Win32 Driver Software CD. The user’s manual gives the memory addresses of all the registers, and their purposes. The function reference gives 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).

The Configuration Xilinx steps 1-13 following are also given in the Readme-SLX150.txt file included on the EDK CD. This file also 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.

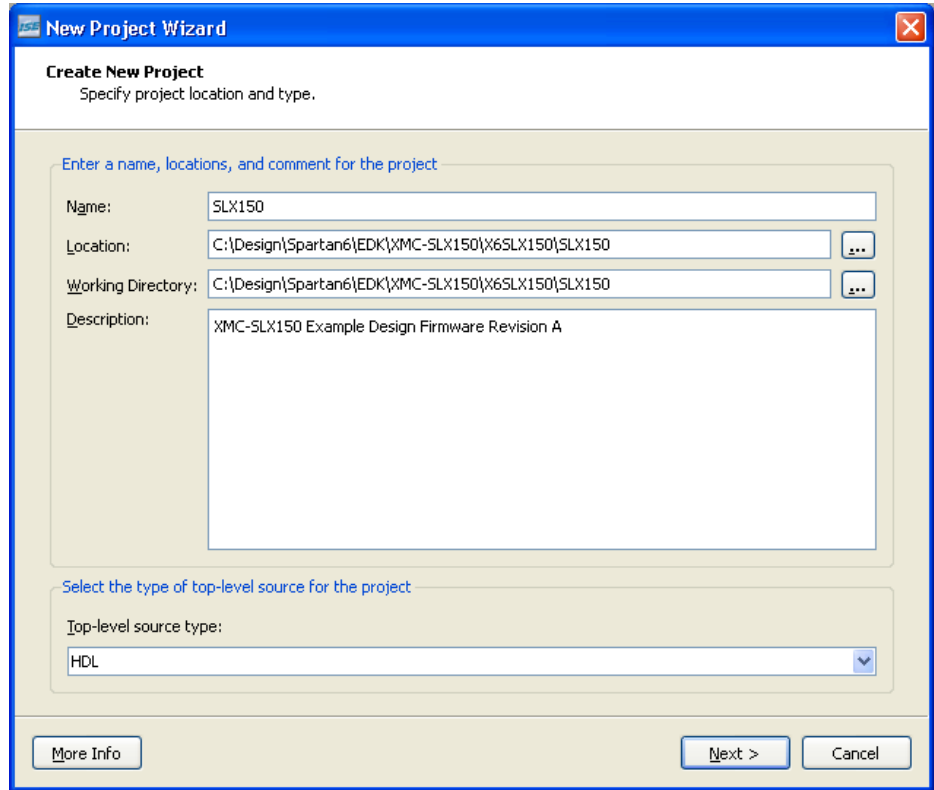
This manual assumes that Xilinx ISE Version 12.3 is used. Note that earlier versions are not supported and later versions may have alternate procedures. Also note that VHDL line numbers in this manual may not match the line numbers of the files provided in the EDK due to revisions.

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 vhd files in into the new **XC6SLX150** folder. Then from the **XC6SLX150** folder copy the **XC6SLX150.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 12.3 → ISE Design Tools → Project Navigator**
4. Open a new project by selecting **File → New Project**



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**.



The 'New Project Wizard' window shows the 'Create New Project' step. The 'Name' field is 'SLX150'. The 'Location' and 'Working Directory' fields are 'C:\Design\Spartan6\EDK\XMC-SLX150\X6SLX150\SLX150'. The 'Description' field is 'XMC-SLX150 Example Design Firmware Revision A'. The 'Top-level source type' dropdown is set to 'HDL'. Buttons for 'More Info', 'Next >', and 'Cancel' are at the bottom.

Field	Value
Name	SLX150
Location	C:\Design\Spartan6\EDK\XMC-SLX150\X6SLX150\SLX150
Working Directory	C:\Design\Spartan6\EDK\XMC-SLX150\X6SLX150\SLX150
Description	XMC-SLX150 Example Design Firmware Revision A
Top-level source type	HDL

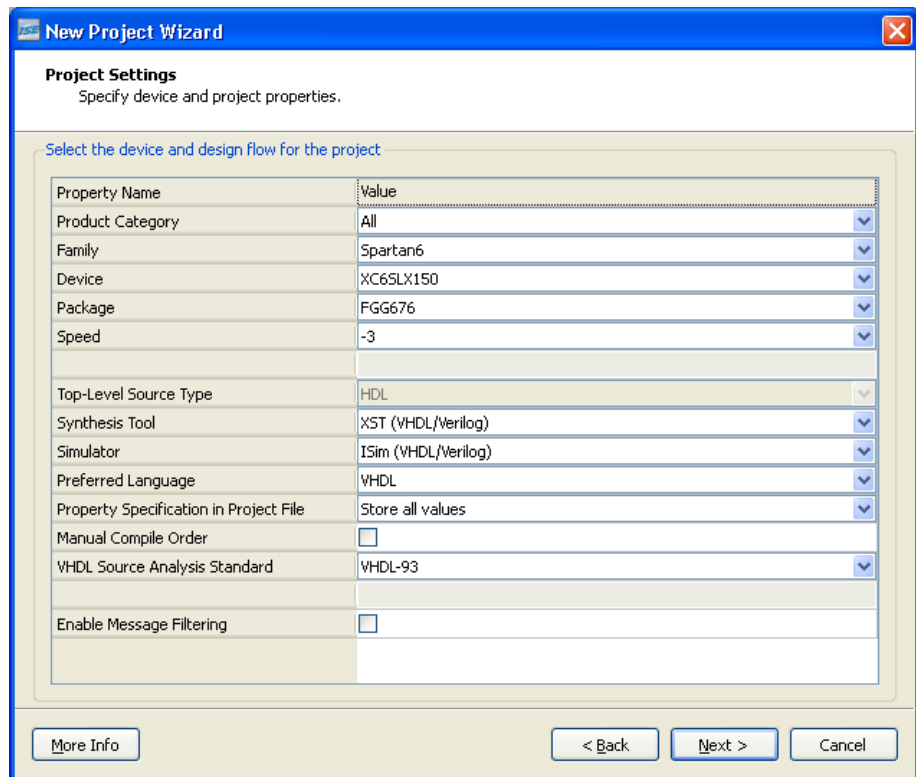
6. Enter the following information if using the SLX150. Then click **Next** and then **Finish**.

Family: Spartan6

Device: XC6SLX150

Package: FGG676

Speed: -3



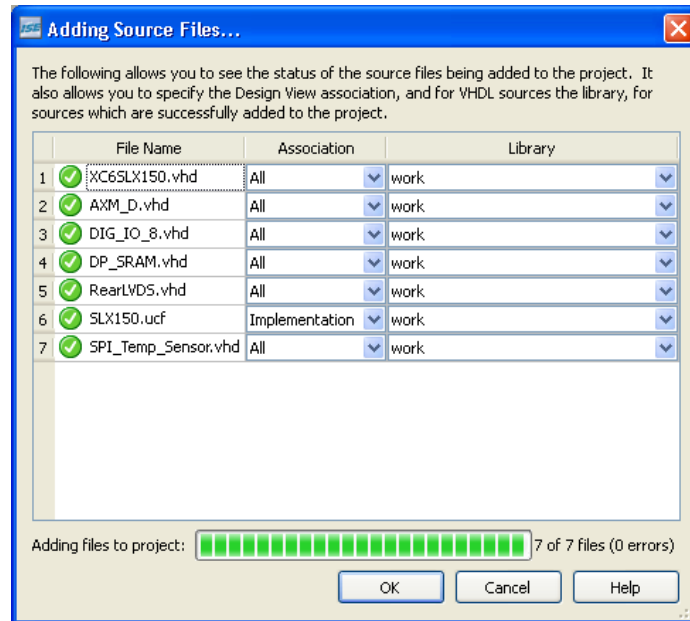
The 'New Project Wizard' window shows the 'Project Settings' step. The 'Property Name' and 'Value' table is as follows:

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	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

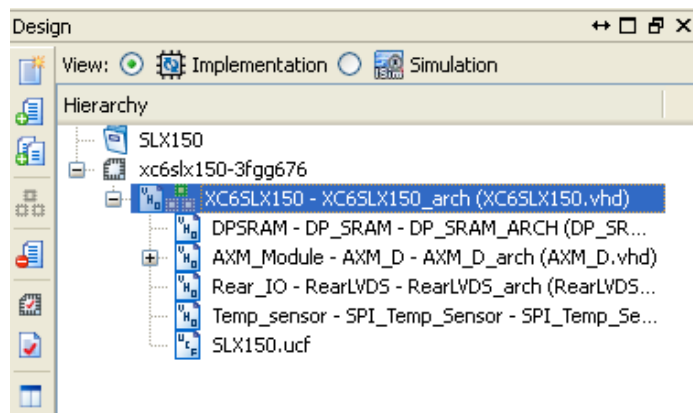
Buttons for 'More Info', '< Back', 'Next >', and 'Cancel' are at the bottom.

7. We will next add the files we copied from the CD. Follow these steps:

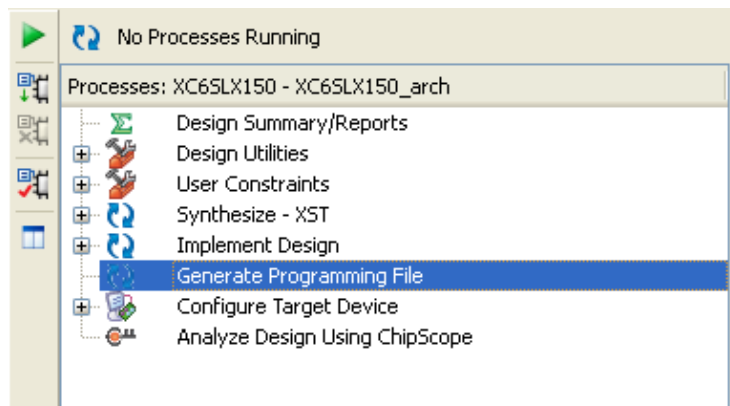
- a. Select **Project->Add Source**.
- b. Select the .ucf and all the .vhd files
- c. Click the **Open** button.
- d. The association for all files should be *All* for the .vhd files and *Implement* for the .ucf files. There are a total of 6 .vhd files and one .ucf file.
- e. Click the **OK** button.



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



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

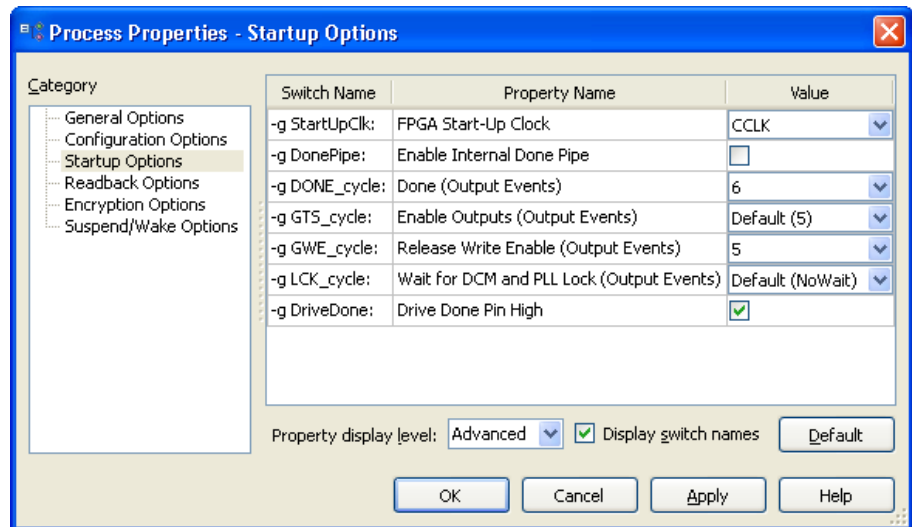


10. Click on **Process** from the menu bar, and click on **Properties**.

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

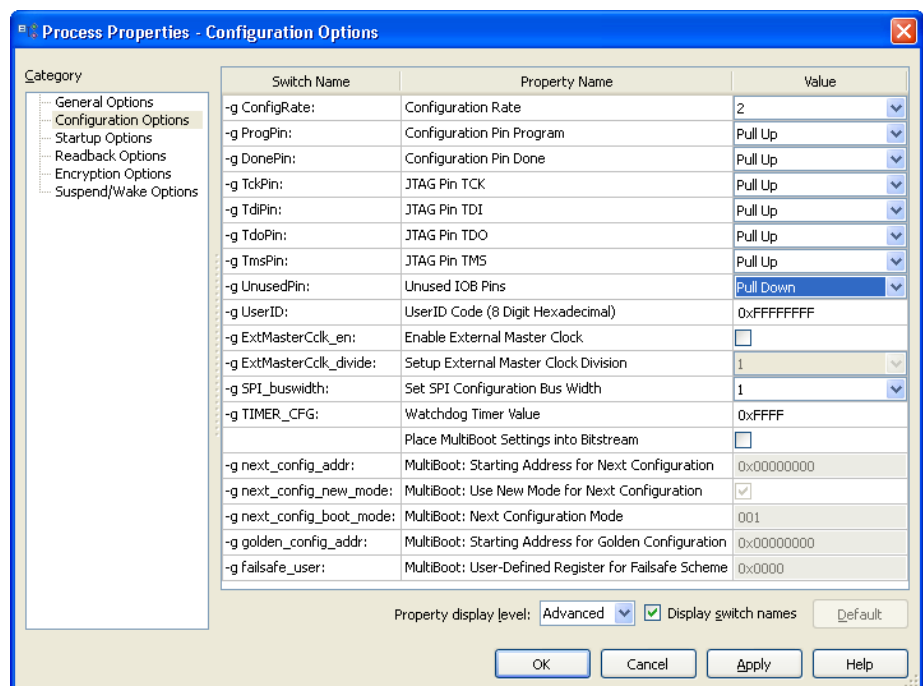
Note that if not all properties are shown change the Property display level from Standard to Advanced.

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



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**.



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;
296
```

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';
498
```

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 725** 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.

```
723 Int_Polarity_Adr => Int_Polarity_Adr,
724
725 --Our address strobe
726 Counter_Adr=> Counter_Adr,
727
728 --Write Stobes
```

4. We will now replace a previously unused memory address. **Uncomment line 887 and 888** 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 -- NU <= 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
```

↓

```
886 LA(3) and LA(2) and Base_Address; --0x801C
887 Counter_Adr <= 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
```

5. To **line 952** 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.

```
950
951 AXM_Strobe <= DiffReg31to0_Adr or DigReg1Sto0_Adr or DiffDirReg_Adr or DigDirReg_Adr or
952 Int_Enable_Adr or Int_Type_Adr or Int_Polarity_Adr or Counter_Adr;
953
```

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 vhd code.

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

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 LBEO_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 (DiffDir_Reg_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.

```
435 -- Counter_EN Register 0x8020 bit 5
436 -- Turns on the functionality of the Counter
437 process (CLK, RESET)
438 begin
439     if (RESET = '1') then
440         Counter_EN <= '0';
441     elsif (CLK'event and CLK = '1') then
442         if (Counter_Stb = '1') then
443             Counter_EN <= LD(5);
444         else
445             Counter_EN <= Counter_EN;
446         end if;
447     end if;
448 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.

```
450 -- Counter_Inc determine when to load the counter
451 -- Register 0x8020 bit 4
452 process (CLK, RESET)
453 begin
454     if (RESET = '1') then
455         Counter_Inc <= '0';
456     elsif (CLK'event and CLK = '1') then
457         if (Counter_EN = '1') then
458             Counter_Inc <= IO_DIGITAL(4);
459         else
460             Counter_Inc <= Counter_Inc;
461         end if;
462     end if;
463 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.

```
465 -- Counter_Reg determine when to increment the counter
466 process (CLK, RESET)
467 begin
468     if (RESET = '1') then
469         Counter_Reg <= "0000";
470     elsif (CLK'event and CLK = '1') then
471         if (Counter_EN = '1' and Counter_Inc = '1') then
472             Counter_Reg <= Counter_Reg + 1;
473         else
474             Counter_Reg <= Counter_Reg;
475         end if;
476     end if;
477 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.

```

535 READ_DATA(0) <=
536     (Counter_Reg(0) and Counter_Adr) or
537     (IO_DIFF(0) and DiffReg31to0_Adr) or
538     (IO_DIGITAL(0) and DigReg15to0_Adr) or
539     (DiffDir_Reg(0) and DiffDirReg_Adr) or
540     (DigDir_Reg(0) and DigDirReg_Adr) or
541
542     (IntEnA_Reg(0) and Int_Enable_Adr) or
543     (IntTypA_Reg(0) and Int_Type_Adr) or
544     (IntPolA_Reg(0) and Int_Polarity_Adr);
545
546
547 READ_DATA(1) <=
548     (Counter_Reg(1) and Counter_Adr) or
549     (IO_DIFF(1) and DiffReg31to0_Adr) or
550     (IO_DIGITAL(1) and DigReg15to0_Adr) or
551     (DiffDir_Reg(1) and DiffDirReg_Adr) or
552     (DigDir_Reg(1) and DigDirReg_Adr) or
553
554     (IntEnA_Reg(1) and Int_Enable_Adr) or
555     (IntTypA_Reg(1) and Int_Type_Adr) or
556     (IntPolA_Reg(1) and Int_Polarity_Adr);
557
558
559 READ_DATA(2) <=
560     (Counter_Reg(2) and Counter_Adr) or
561     (IO_DIFF(2) and DiffReg31to0_Adr) or
562     (IO_DIGITAL(2) and DigReg15to0_Adr) or
563     (DiffDir_Reg(2) and DiffDirReg_Adr) or
564     (DigDir_Reg(2) and DigDirReg_Adr) or
565
566     (IntEnA_Reg(2) and Int_Enable_Adr) or
567     (IntTypA_Reg(2) and Int_Type_Adr) or
568     (IntPolA_Reg(2) and Int_Polarity_Adr);
569
570
571 READ_DATA(3) <=
572     (Counter_Reg(3) and Counter_Adr) or
573     (IO_DIFF(3) and DiffReg31to0_Adr) or
574     (IO_DIGITAL(3) and DigReg15to0_Adr) or
575     (DiffDir_Reg(3) and DiffDirReg_Adr) or
576     (DigDir_Reg(3) and DigDirReg_Adr) or
577
578     (IntEnA_Reg(3) and Int_Enable_Adr) or
579     (IntTypA_Reg(3) and Int_Type_Adr) or
580     (IntPolA_Reg(3) and Int_Polarity_Adr);
581
582
583 READ_DATA(4) <=
584     (Counter_Inc and Counter_Adr) or
585     (IO_DIFF(4) and DiffReg31to0_Adr) or
586     (IO_DIGITAL(4) and DigReg15to0_Adr) or
587     (DiffDir_Reg(4) and DiffDirReg_Adr) or
588     (DigDir_Reg(4) and DigDirReg_Adr) or
589
590     (IntEnA_Reg(4) and Int_Enable_Adr) or
591     (IntTypA_Reg(4) and Int_Type_Adr) or
592     (IntPolA_Reg(4) and Int_Polarity_Adr);
593
594
595 READ_DATA(5) <=
596     (Counter_EN and Counter_Adr) or
597     (IO_DIFF(5) and DiffReg31to0_Adr) or
598     (IO_DIGITAL(5) and DigReg15to0_Adr) or
599     (DiffDir_Reg(5) and DiffDirReg_Adr) or
600     (DigDir_Reg(5) and DigDirReg_Adr) or
601
602     (IntEnA_Reg(5) and Int_Enable_Adr) or
603     (IntTypA_Reg(5) and Int_Type_Adr) or
604     (IntPolA_Reg(5) and Int_Polarity_Adr);

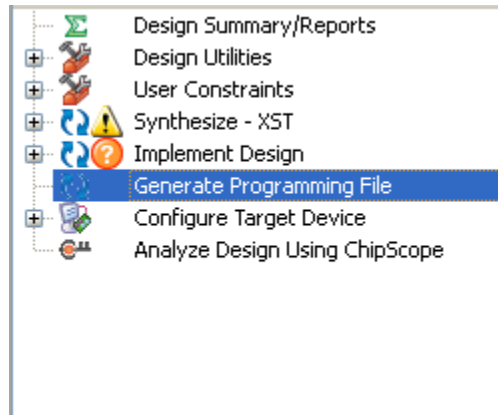
```

Generating a Programming “MCS” File

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

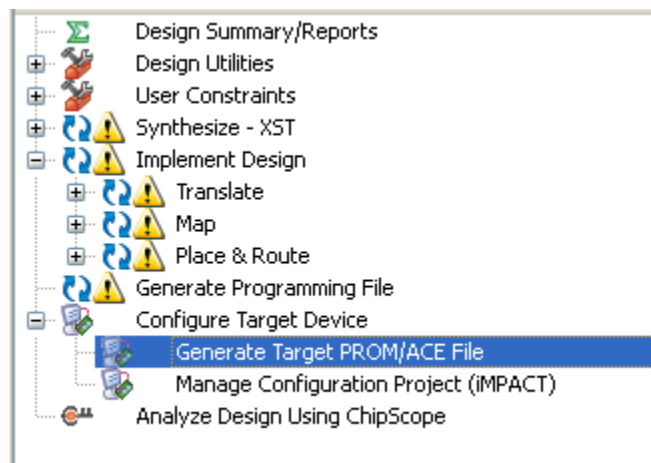
1. Select **XC5VLX110T-
XC5VLX110T_arch
(XC5VLX110T.vhd)** in the
Hierarchy Window.

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

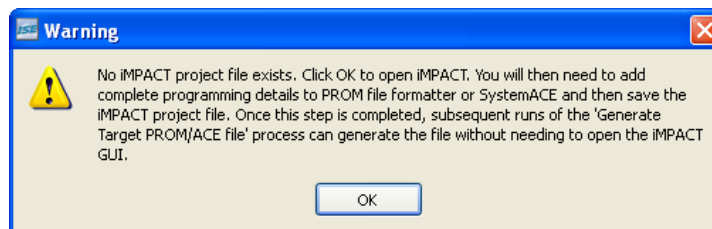


Note: If there are any errors, correct them and repeat steps 1 and 2. There may be a few warnings.

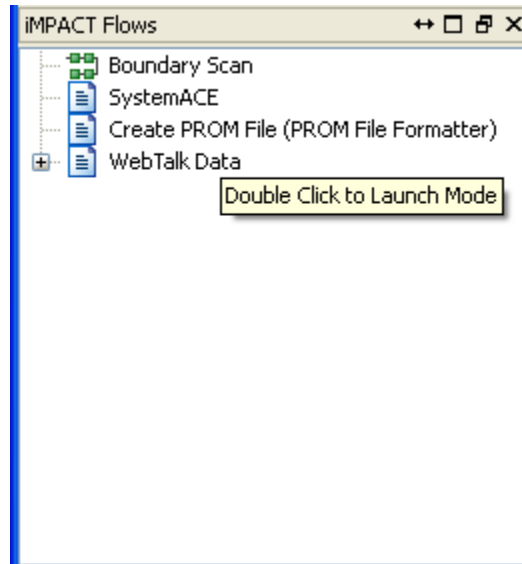
3. Right-click on **Generate
Target PROM/ACE File**, and
click on **Run**.



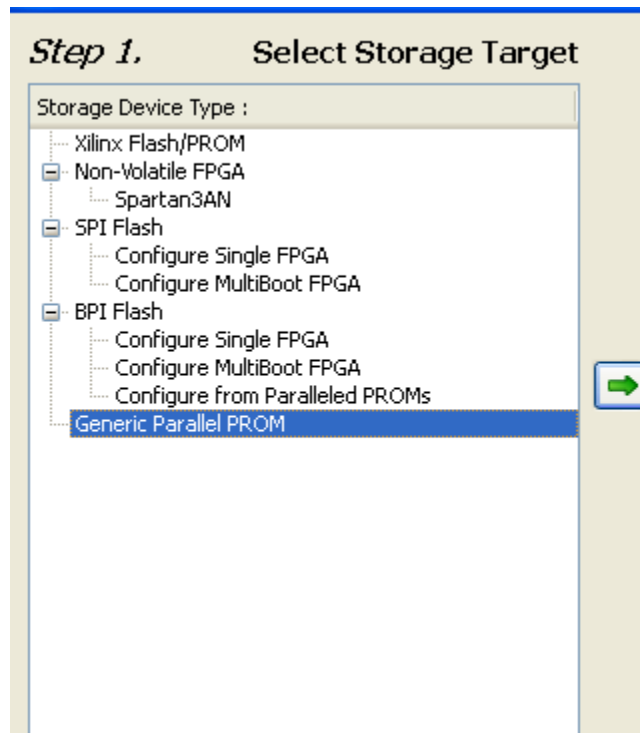
4. Select **OK**.



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



6. Select **Generic Parallel PROM** and then select the green arrow.




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

Step 2. Add Storage Device(s)

Parallel PROM (Bytes) 8M [8388608]

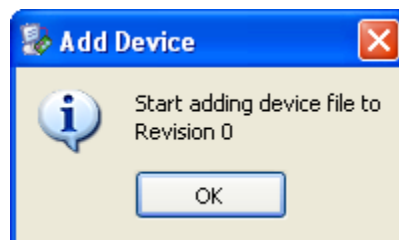
8. Verify the Output File location and enter an Output File Name. For example SLX150. Leave all other default options and select **OK**.

Step 3. Enter Data

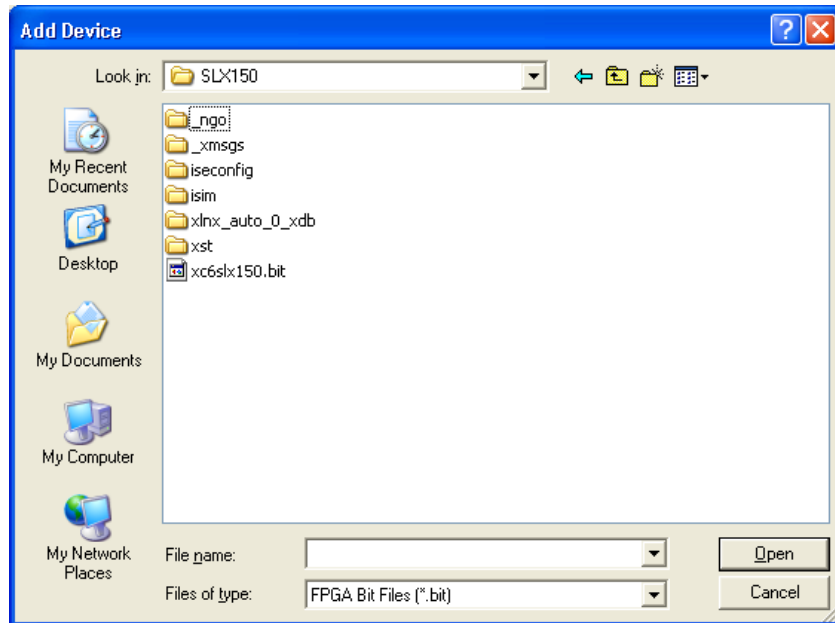
General File Detail	Value
Checksum Fill Value	FF
Output File Name	SLX150
Output File Location	sign/Spartan6/EDK/XMC-SLX150 

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

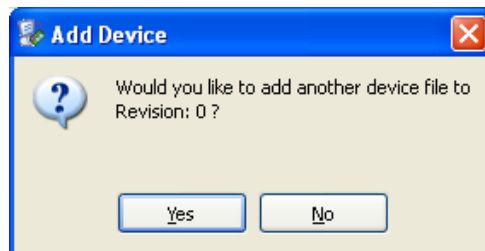
9. Click **OK**.



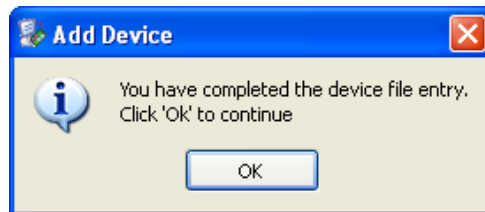
10. Select **xc6slx150.bit** file and then click **Open**.



11. Click **NO**.

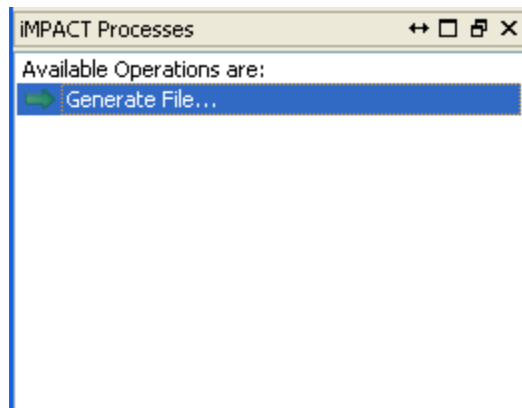


12. Click **OK**.



13. Click **OK** again.

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



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

A blue rectangular box with a slight 3D effect, containing the text "Generate Succeeded" in white, bold, sans-serif font.

The SLX150.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
93  constant addr_max: integer := 17;
738  -- ADD_RESET_VALUE(18) <= (SRAM_IntAdr_StbAll and LD(18)) or --required for -1M model
739  --      (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(18)) or
740  --      (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(18));
741  ADD_RESET_VALUE(19) <= (SRAM_IntAdr_StbAll and LD(19)) or --required for -1M model
742  --      (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(19)) or
743  --      (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(19));
788  SRR_A(18) <= '0'; --remove for -1M model
789  SRR_A(19) <= '0'; --remove for -1M model
790
959  (SRAM_Read_Adr2 and SRR_IO_RD(50)) or
960  -- (SRAM_IntAdr and SRAM_ADD_Count(18)) or --Required for -1M model
961  -- (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(18)) or
962  -- (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(18)) or
963  -- (SRAM_Reset0_Adr and DMA0_RESET(18)) or
964  -- (SRAM_Reset1_Adr and DMA1_RESET(18));
966  (SRAM_Read_Adr2 and SRR_IO_RD(51)) or
967  -- (SRAM_IntAdr and SRAM_ADD_Count(19)) or --required for -1M model
968  -- (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(19)) or
969  -- (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(19)) or
970  -- (SRAM_Reset0_Adr and DMA0_RESET(19)) or
971  -- (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
93  constant addr_max: integer := 19;
738  ADD_RESET_VALUE(18) <= (SRAM_IntAdr_StbAll and LD(18)) or
739  --      (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(18)) or
740  --      (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(18));
741  ADD_RESET_VALUE(19) <= (SRAM_IntAdr_StbAll and LD(19)) or
742  --      (DMA0_EVENT and SRAM_Reset0_EN and DMA0_RESET(19)) or
743  --      (DMA1_EVENT and SRAM_Reset1_EN and DMA1_RESET(19));
956  (SRAM_Read_Adr2 and SRR_IO_RD(50)) or
957  (SRAM_IntAdr and SRAM_ADD_Count(18)) or
958  (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(18)) or
959  (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(18)) or
960  (SRAM_Reset0_Adr and DMA0_RESET(18)) or
961  (SRAM_Reset1_Adr and DMA1_RESET(18));
963  (SRAM_Read_Adr2 and SRR_IO_RD(51)) or
964  (SRAM_IntAdr and SRAM_ADD_Count(19)) or
965  (SRAM_DMA0Thr_Adr and DMA0_THRESHOLD(19)) or
966  (SRAM_DMA1Thr_Adr and DMA1_THRESHOLD(19)) or
967  (SRAM_Reset0_Adr and DMA0_RESET(19)) or
968  (SRAM_Reset1_Adr and DMA1_RESET(19));

```

Xilinx ISE 12.3 Compiler Warnings

Note that ISE 12.3 will generate the following warnings when compiling the example design. These warnings can be safely ignored as they reference unused signals.

- WARNING:Xst:647 - Input <LD<20:30>> 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>.