Series XMC-VLX85/VLX110/VLX155
Virtex-5 Based FPGA
XMC Module

Getting Started With the XMC-VLX

ACROMAG INCORPORATED
30765 South Wixom Road
P.O. BOX 437
Wixom, MI 48393-7037 U.S.A.
Tel: (248) 295-0310
Fax: (248) 624-9234

8500-903-A10M000

OBJECTIVE

The purpose of this document is to provide basic instructions on using the "XMC-VLX Engineering Design Kit" with the XMC-VLX Boards. It will focus on programming the FPGA of the XMC-VLX110 using VHDL, but can be easily modified to use with any model of the VLX 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++.

1.0 Getting Started

When using any VLX board besides the VLX110 simply substitute its model number for the “VLX110” through out this document.

Turn off your computer, and unplug the power cord. Before touching either board, make sure to discharge all static electricity. Then attach the VLX110 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-VLX Engineering Design Kit** and copy the **VLX110** folder to your computer.

Before you start, familiarize yourself with the **XMC-VLX User’s Manual** included on the EDK CD and the **PCIeVLX 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-VLX110.txt file included on the EDK CD

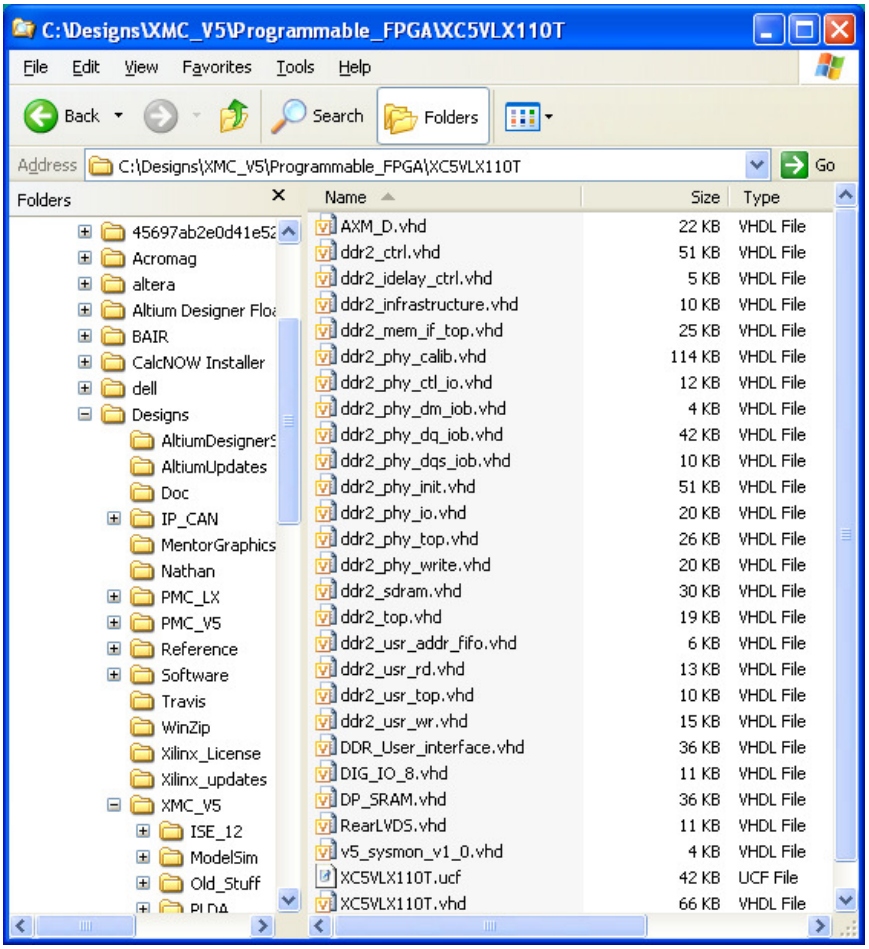
2.0 Configuring Xilinx

1. Make a new directory and call it **XC5VLX110T**.

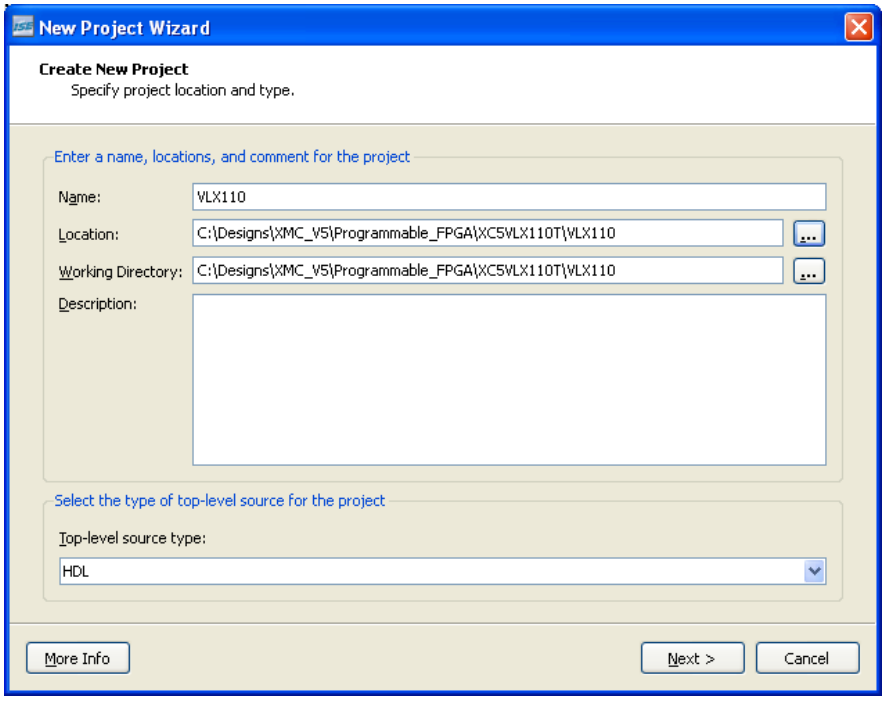
2. From the **XC5VLX110T** folder, copy the all the vhdl files in into the new **XC5VLX110T folder**. Then from the **XC5VLX110T** folder copy the **XC5VLX110T.ucf** file to the XC5VLX110T folder. Note that all of the files are shown in the adjacent figure.

3. Start Xilinx's Project Navigator from your start menu. **Xilinx ISE Design Suite 12.3 → ISE Design Tools → Project Navigator**

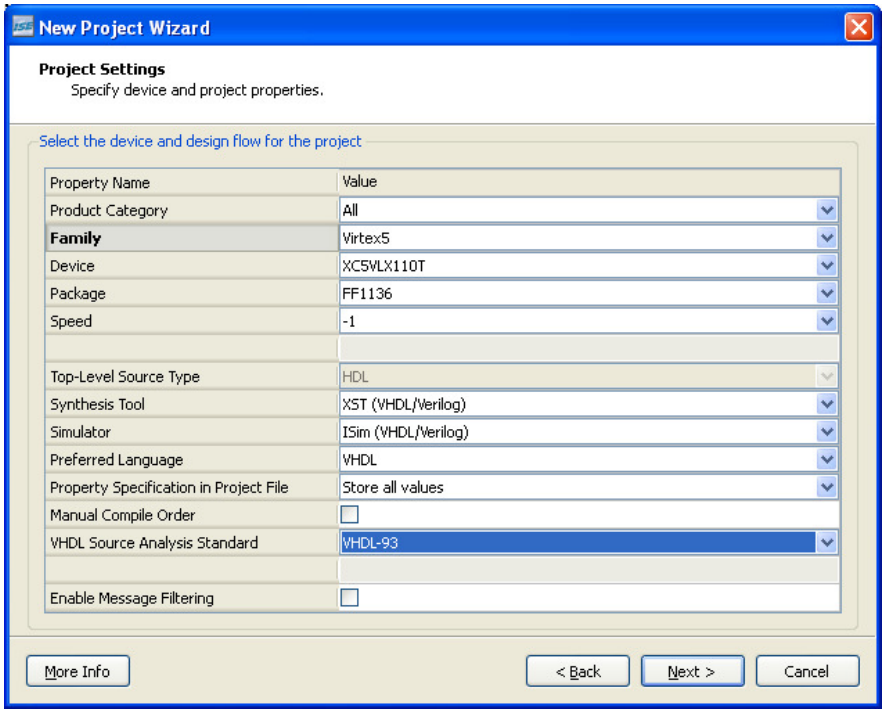
4. Open a new project by selecting **File → New Project**



5. In the **Project Name** field, type **VLX110.** In the **Location** field type the path name where to find the XC5VLX110T folder. Make sure the **Top-Level Module Type** field is **HDL**, and click **Next**.

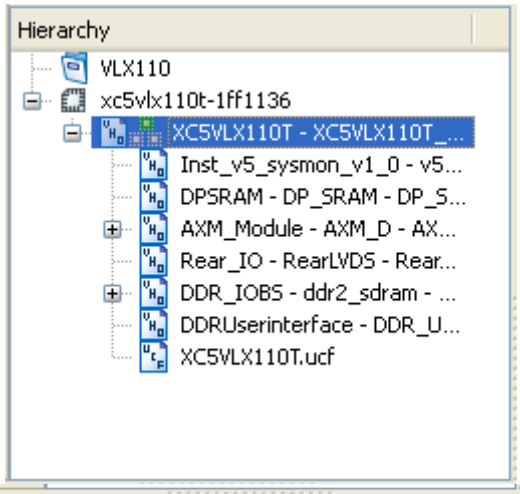


6. Enter the following information if using the VLX110. Then click **Next and then Finish.**

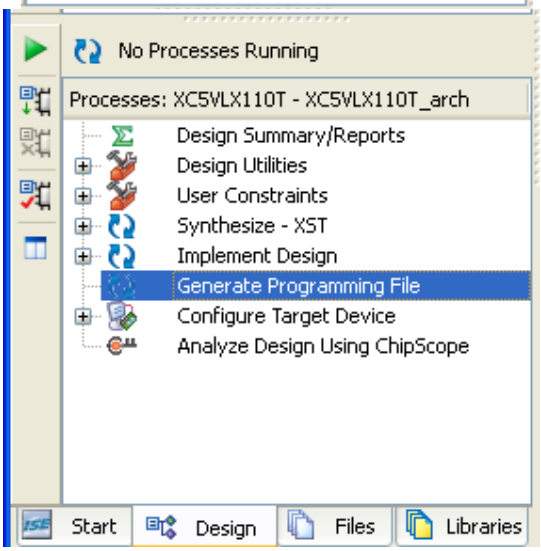


7. We will next add the files we copied from the CD. Follow these steps:
   a. Select **Project → Add Source**
   b. Select all the .ucf and .vhd files
   c. Select the **Open** button
   d. The association for all file should be All for the .vhd files and Implement for the .ucf file. There a total of 26 .vhd files and one .ucf file.
   e. Select **OK** button.

8. In the **Hierarchy Window**, click on **XC5VLX110T-XC5VLX110T_arch (XC5VLX110T.vhd)** to highlight it.



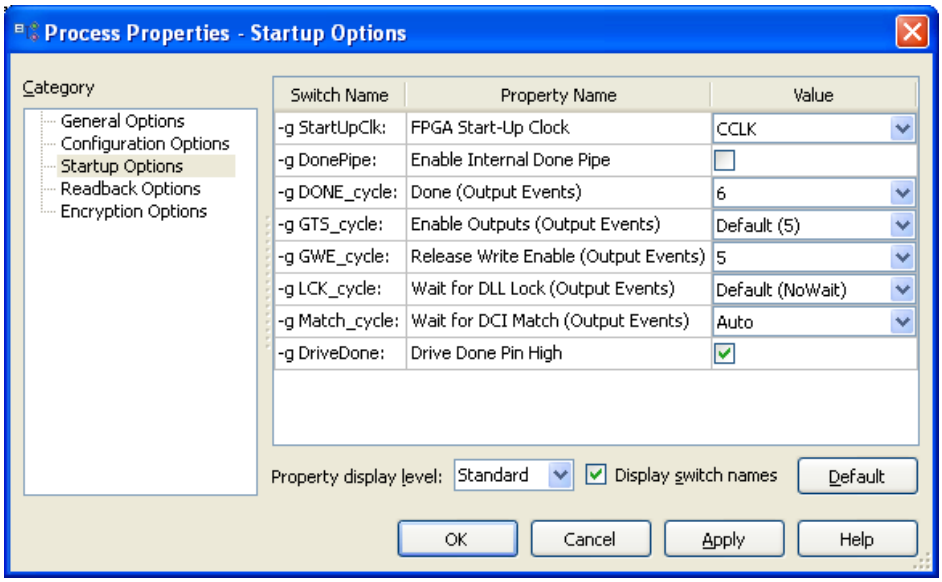
9. In the **Processes Window**, click on **Generate Programming File** so it is also highlighted.



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

11. Click on the **Startup Options** tab.
12. Verify that the following options are selected:

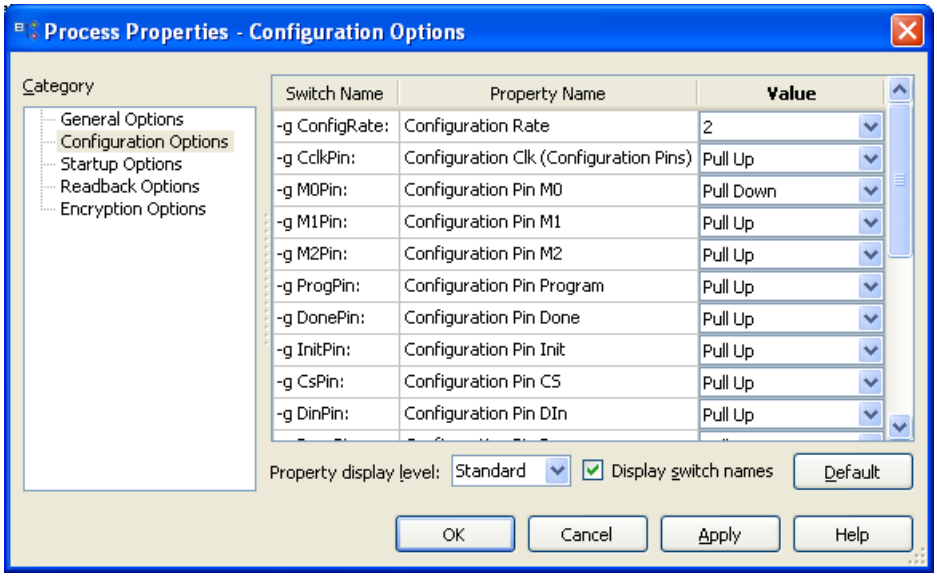
FPGA Start-Up Clock:	CCLK
Enable Internal Done Pipe:	Not Checked
Done:	6
Enable Outputs:	(Default) 5
Release Write Enable:	5
Release DLL:	Default
Match Cycle:	Auto
Drive Done Pin High:	Check



13. Click on the **Configuration Options** tab. Change only the following options

Configuration Rate:	2
Configuration Pin M0:	Pull Down

All other settings should stay the same
Click on **OK**.



3.0 Adding To The Provided VHDL Code

To revise or add to the provided VHDL code, begin by double clicking on the **XC5VLX110T-XC5VLX110T _arch (XC5VLX110T.vhd)** file located in the **Hierarchy** window. This will open the VHDL file for editing

Under the commented code (line 342) additional components and signals may be added.

```
336     signal SysMonAddrReg_Adr : std_logic;  --0x808C
337     signal SysMonAddrReg_Stb0 : std_logic;
338
339     signal DWE_hold, DRDY_OUT_i1, DRDY_OUT_reg : std_logic;
340
341   --**Insert Components Declarations: between the 'architecture
342   --- 'begin' keywords**
343
344     COMPONENT v5_sysmon_v1_0
```

After the **begin** keyword (line 672) additional instantiations for components may be added

```
668   end component;
669
670
671   ---------================================================
672   begin
673   --for debug
674   -- FIO_DIFF(0) <= ADS_n;
```

For simplicity we suggest adding to or revising the provided VHDL code that is associated with the 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

4.0 Example Use of the VHDL Code

Below is a simple example of some VHDL that could be used to control five of the VLX110's LVTTL channels. It is included to show how the code supplied with the Engineering Design Kit can be modified for personal use.

1. Open **XC5VLX110T.vhd** and scroll down to around **line 218**. Add these four lines of code that create a new address strobe for our counter. It will be located in register 0x8020.

Note: Only add the code within the box. The lines above and below are included as a reference.

```
215   signal Int_Type_Adr : STD_LOGIC; --0x8018
216   signal Int_Polarity_Adr : STD_LOGIC; --0x801C
217
218   -- Here we add the decode signal for the
219   -- new counter addresss we are going to send
220   -- to the AXM
221   signal Counter_Adr : STD_LOGIC;  --0x8020
222
223   --Rear J4 Connector Address Decode Signals
```

2. Around **line 455** 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.

```
451     Int_StatClear_Stb0: in STD_LOGIC;-- Interrupt Sta
452
453     -- Our Address strobe for the counter
454     Counter_Adr : in STD_LOGIC;
455
456     IO_DIGITAL: inout STD_LOGIC_VECTOR(15 downto 0);
```

3. Add the following around **line 958** 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.

```
953     Int_Polarity_Adr => Int_Polarity_Adr,
954
955     -- Our Address Strobe
956     Counter_Adr => Counter_Adr,
957
958     --Write Stobes
```

4. We will now replace a previously unused memory address. **Uncomment line 1186 and 1187** and replace the "NU" with **Counter_Adr**. This will be the location in memory to access the counter.

```
1169                        LA(3) and     LA(2) and Base_Address;  --0x801C
1170  --  NU          <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
1171  --                    not LA(3) and not LA(2) and Base_Address;  --0x8020
1172  --  NU          <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
```

V

```
1169                        LA(3) and     LA(2) and Base_Address;  --0x801C
1170    Counter_Adr   <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
1171                     not LA(3) and not LA(2) and Base_Address;  --0x8020
1172  --  NU          <= not LA(8) and not LA(7) and not LA(6) and     LA(5) and not LA(4) and
```

5. To **line 1247** add the red colored code. This is added to strobe the AXM (where the res of the counter code will be located) when the Counter_Ad has received a read or write command.
   We are finished editing the XC5VLX110T.vhd file and will now be **editing the AXM_D.vhd** file.

```
1229          SRAM_Reset0_Adr or SRAM_Reset1_Adr or SRAM_Read_Adr or SRAM_Read_Adr2;
1230
1231  AXM_Strobe <= DiffReg31to0_Adr or DigReg15to0_Adr or DiffDirReg_Adr or DigDirReg_Adr or
1232                Int_Enable_Adr or Int_Type_Adr or Int_Polarity_Adr or Counter_Adr;
1233
1234  Rear_Strobe <= Rear_LVDS_Wr_Adr or Rear_LVDS_Rd_Adr;
```

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 a **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 a **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 a **line 464** to handle the counter. When the counter is enabled, i 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 o 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);
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);
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  READ_DATA(5) <=
595            (Counter_EN and Counter_Adr) or
596            (IO_DIFF(5) and DiffReg31to0_Adr) or
597            (IO_DIGITAL(5) and DigReg15to0_Adr) or
598            (DiffDir_Reg(5) and DiffDirReg_Adr) or
599            (DigDir_Reg(5) and DigDirReg_Adr) or
600
601            (IntEnA_Reg(5) and Int_Enable_Adr) or
602            (IntTypA_Reg(5) and Int_Type_Adr) or
603            (IntPolA_Reg(5) and Int_Polarity_Adr);
```

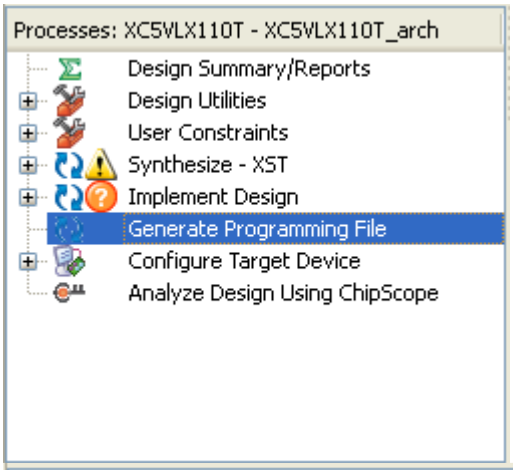
5.0 Create A Program MCS File

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

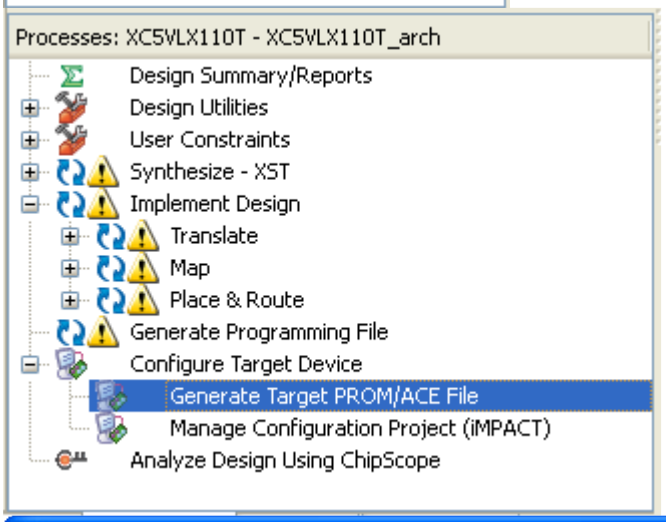
2. Select **Generate Programming File** in the **Processes Window**. Then select **Process →Run**

   Note: If there are any errors, correct them and repeat steps 1 and 2. There will be 875 warnings.

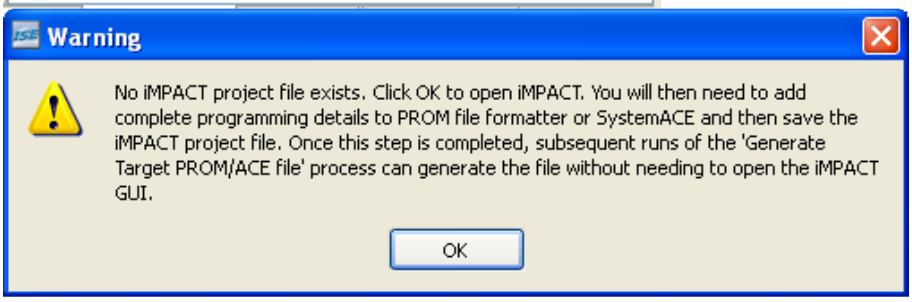
The purpose of this document is to provide basic instructions on using the "XMC-VLX Engineering Design Kit" with the XMC-VLX Boards. It will focus on programming the FPGA of the XMC-VLX110 using VHDL, but can be easily modified to use with any model of the VLX line. This document also shows how to use the supplied dll files with a MFC application. It is assumed that the user has a working knowledge of Xilinx, VHDL and Visual C++.



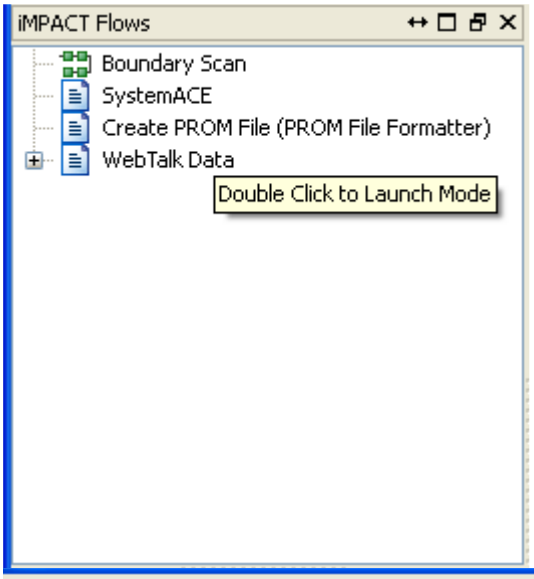
3. Right-click on **Generate Target PROM/ACE File,** and click on **Run**.



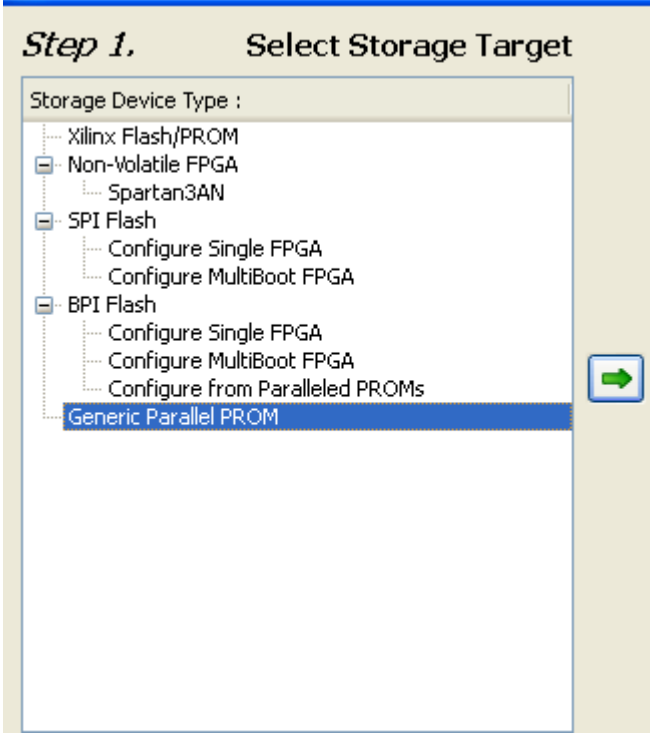
4. Select **OK**



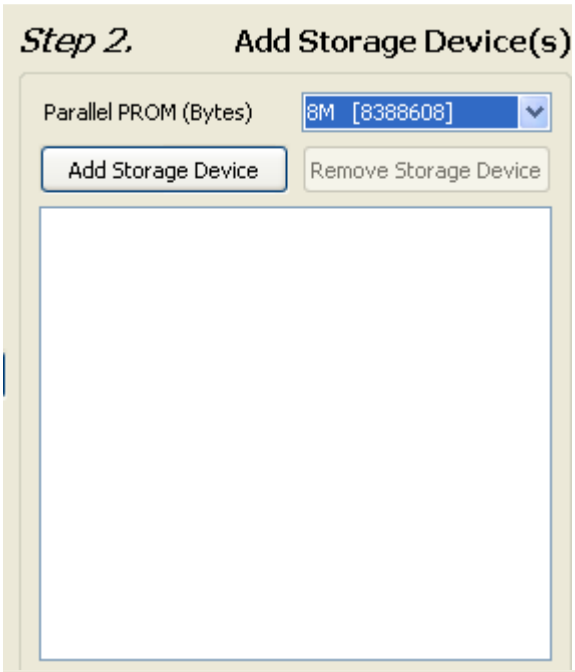
5. Double select **Create PROM File (PROM File Formatter)**



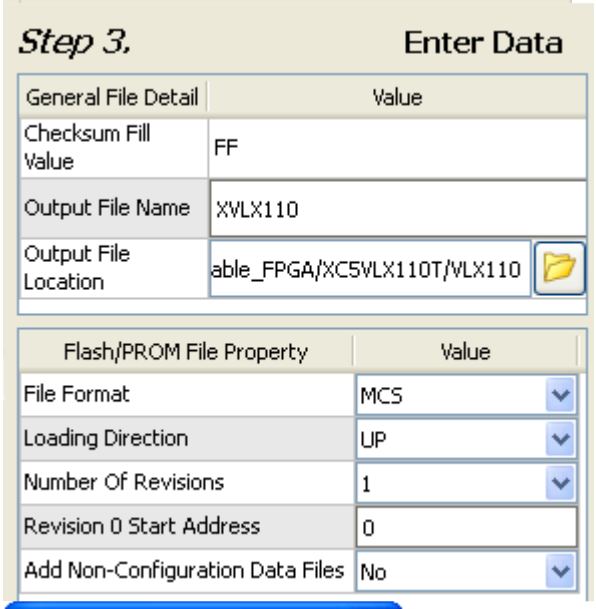
6. Select **Generic Parallel PROM** and the select the green arrow.



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

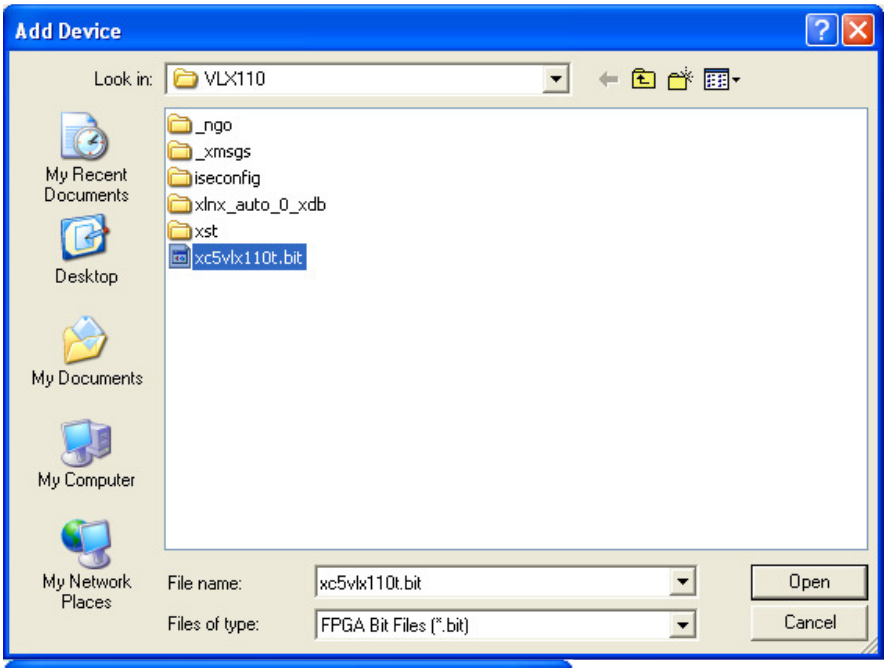


8. Verify the Output File location and enter an Output File Name. For example XVLX110. Leave all other default options and select **OK**.

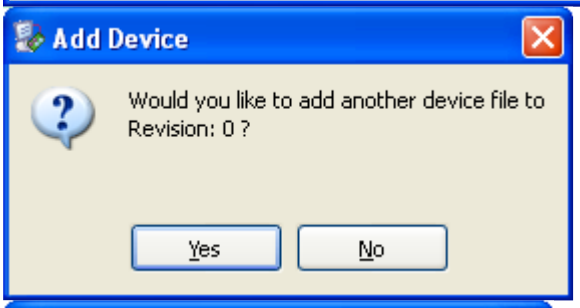


9. Select **OK**

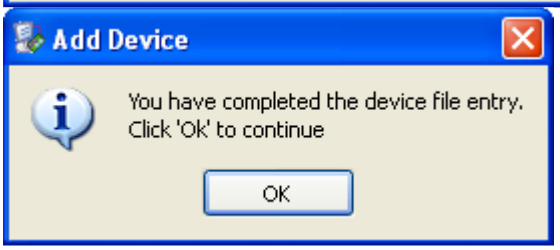
10. Select **xc5vlx110t.bit** file and then select **Open**.



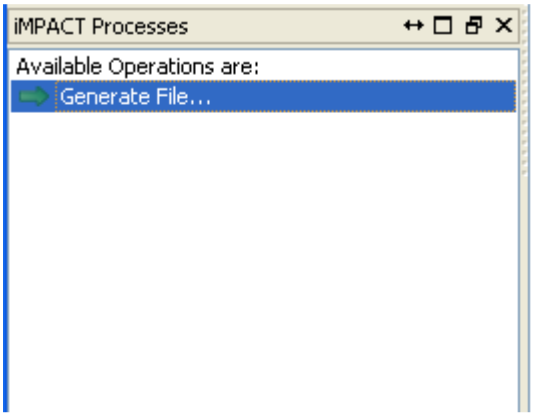
11. Select No



12. Select OK



13. Select OK again.
14. Double select **Generate File.**



Generate Succeeded will be displayed.

The XVLX110.mcs file has been successfully generated.