Table of Contents

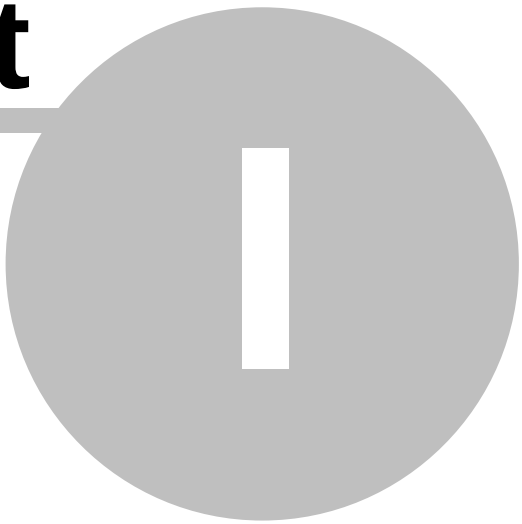
Foreword

This is just another title page
placed between table of contents
and topics

Top Level Intro

This page is printed before a new
top-level chapter starts

Part



1 Quick Start Tutorials

Overview

This following tutorials will lead you step-by-step through the compilation, execution and RTL generation of your first Impulse C applications on the Xilinx PowerPC platform.

The tutorials that follow assume that you have previously gone through at least one of the tutorials included in your standard CoDeveloper installation. It is also assumed that you are somewhat familiar with the Xilinx ISE and Platform Studio (EDK) tools.

Note: These tutorials assume that you are using Xilinx Platform Studio version 10.1 or later. Depending on the version of Platform Studio you are using, the steps may be somewhat different.

The Tutorials

Tutorial 1: Complex FIR Filter on Virtex-5 Platform (EDK 10.1)
Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)
Tutorial 3: Fractal Image Generation on the Virtex-4 Platform (EDK 10.1)

See Also

Platform Support Package Overview

1.1 Tutorial 1: Complex FIR Filter on Virtex-5 Platform (EDK 10.1)

Overview

This detailed tutorial will demonstrate how to use **Impulse C** to create, compile and optimize a digital signal processing (DSP) example for the **PowerPC** platform. We will also show how to make use of the **Auxiliary Processor Unit** (**APU**) and **Fabric Co-processor Bus** (**FCB**) provided in the **PowerPC** platform.

The goal of this application will be to compile the algorithm (a **Complex FIR Filter** function) on hardware on the FPGA. The **PowerPC** will be used to run test code (producer and consumer processes) that will pass text data into the algorithm and accept the results.

This example makes use of the **Xilinx ML507 Evaluation Platform**. The board features a **Virtex-5 FXT FPGA** with a **PowerPC 440** soft processor core. This tutorial also assumes you are using the **Xilinx EDK 10.1i** (or later) development tools.

This tutorial will require approximately two hours to complete, including software run times.

*Note: this tutorial is based on a sample DSP application developed by Bruce Karsten of Xilinx, Inc. A more complete description of the algorithm can be found in the **Impulse C User Guide**, in the Getting Started Tutorial #2. This tutorial assumes that you have are familiar with the basic steps involved in using the Xilinx EDK tools. For brevity this tutorial will omit some EDK details that are covered in introductory EDK and **Impulse C tutorials**.*

Note also that most of the detailed steps in this tutorial only need to be performed once, during the initial creation of your PowerPC application. Subsequent changes to the application do not require repeating these steps.

Steps

See Also

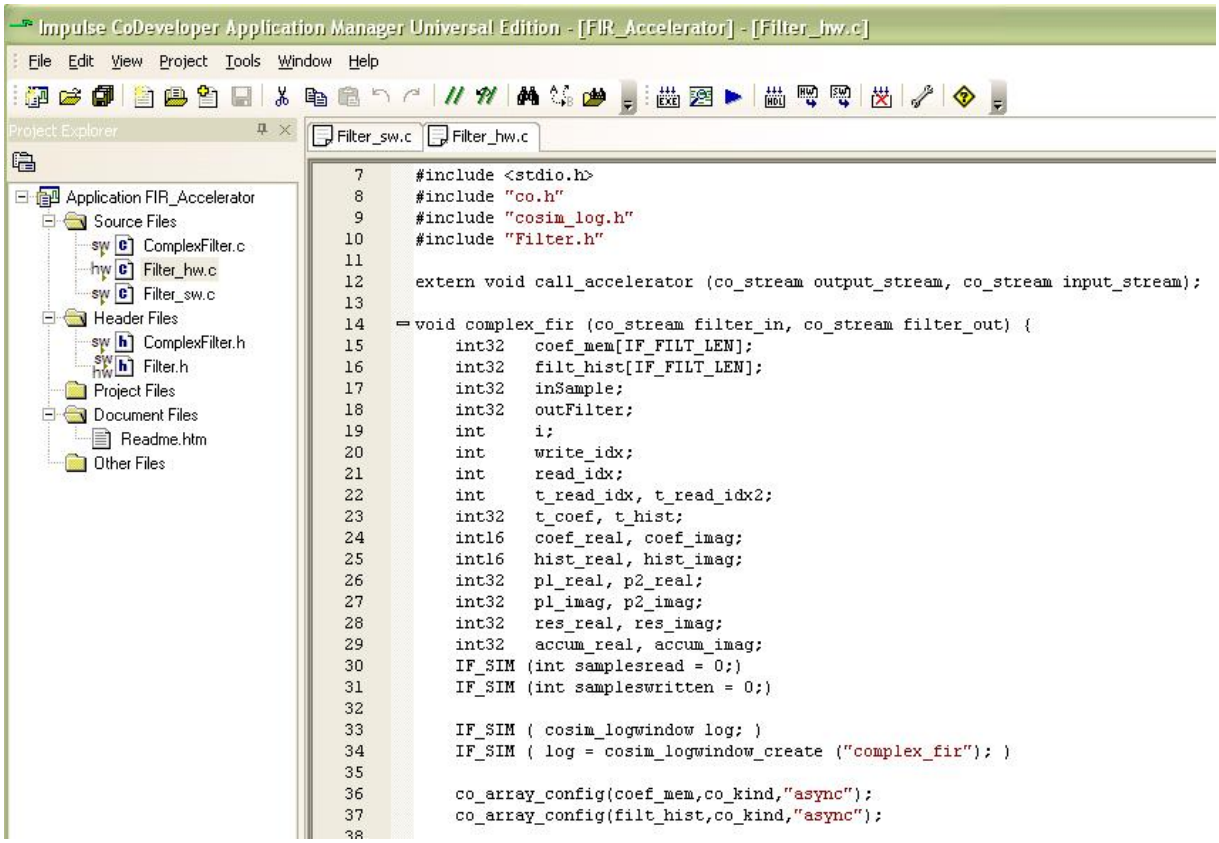
1.1.1 Loading the Complex FIR Filter Application

Complex FIR Filter Tutorial for PowerPC, Step 1

To begin, start the **CoDeveloper** Application Manager by selecting Application Manager from the **Start** -> **Programs** -> **Impulse Accelerated Technologies** -> **CoDeveloper** program group.

*Note: this tutorial assumes that you have already read and understand the Complex FIR example and tutorial presented in the main **CoDeveloper** help file.*

Open the **Xilinx PowerPC ComplexFIR** sample project by selecting **Open Project** from the **File** menu, or by clicking the **Open Project** toolbar button. Navigate to the **.\ExamplesV3\Embedded\ComplexFIR_PowerPC** directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) The project file is also available online at http://impulsec.com/ReadyToRun/. Opening the project will result in the display of a window similar to the following:



Files included in the **Complex FIR** project include:

Source files ComplexFilter.c, Filter_hw.c and Filter_sw.c - These source files represent the complete application, including the **main()** function, consumer and producer software processes and a single hardware process.

See Also

Understanding the Complex FIR Application

1.1.2 Understanding the Complex FIR Filter Application

Complex FIR Filter Tutorial for PowerPC, Step 2

Before compiling the Complex FIR application to hardware, let's first take a moment to understand its basic operation and the contents of the its primary source files, and in particular *Filter_hw.c*.

The specific process that we will be compiling to hardware is represented by the following function (located in Filter_hw.c):

```
void complex_fir(co_stream filter_in, co_stream filter_out)
```

This function reads two types of data:

- Filter coefficients used in the Complex FIR convolution algorithm.
- An incoming data stream

The results of the convolution are written by the process to the stream **filter_out**.

The **complex_fir** function begins by reading the coefficients from the **filter_in** stream and storing the resulting data into a local array (**coef_mem**). The function then reads and begins processing the data, one at a time. Results are written to the output stream **filter_out**.

The repetitive operations described in the **complex_fir** function are complex convolution algorithm.

The complete test application includes test routines (including **main**) that run on the PowerPC processor, generating test data and verifying the results against the legacy C algorithm from which **complex_fir** was adapted.

The configuration that ties these modules together appears toward the end of the Filter_hw.c file, and reads as follows:

```
void config_filt (void *arg) {
   int i;

   co_stream  to_filt, from_filt;
   co_process cpu_proc, filter_proc;

   to_filt   = co_stream_create ("to_filt",   INT_TYPE(32), 4);
   from_filt = co_stream_create ("from_filt", INT_TYPE(32), 4);

   cpu_proc    = co_process_create ("cpu_proc",    (co_function)
call_accelerator, 2, to_filt, from_filt);
   filter_proc = co_process_create ("filter_proc", (co_function)
complex_fir,      2, to_filt, from_filt);

   co_process_config (filter_proc, co_loc, "PE0");
}
```

As in the Hello World example (described in the main CoDeveloper help file), this configuration function describes the connectivity between instances of each previously defined process.

Only one process in this example (**filter_proc**) will be mapped onto hardware and compiled by the Impulse C compiler. This process (**filter_proc**) is flagged as a hardware process through the use of the **co_process_config** function, which appears here at the last statement in the configuration function. **Co_process_config** instructs the compiler to generate hardware for **complex_fir** (or more accurately, the *instance* of **complex_fir** that has been declared here as **filter_proc**).

The *ComplexFilter.c* generates a set of complex FIR coefficients and also a group of input data being processed.

The Filter_sw.c will run in the PowerPC embedded processor, controlling the stream flow and printing results.

See Also

Compiling the Application for Simulation

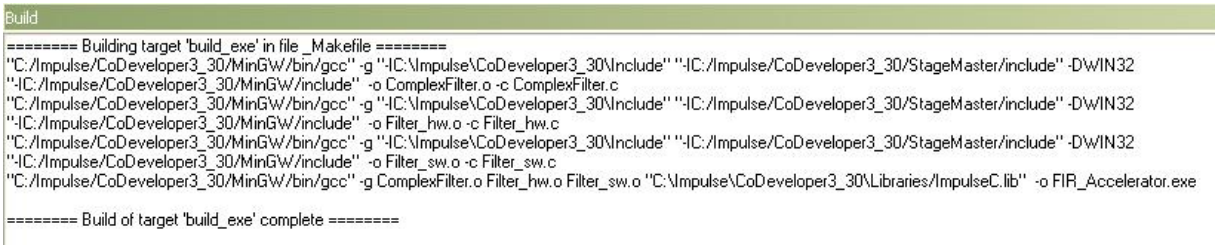
1.1.3 Compiling the Application for Simulation

Complex FIR Filter Tutorial for PowerPC, Step 3

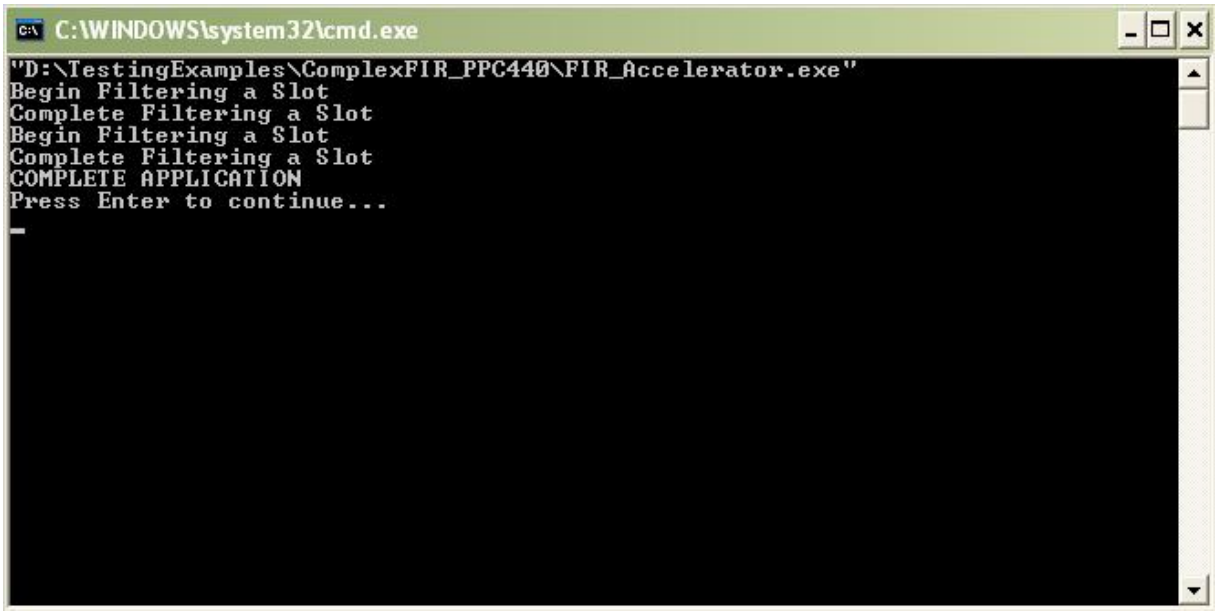
Simulation allows you to verify the correct operation and functional behavior of your algorithm before attempting to generate hardware for the FPGA. When using Impulse C, simulation simply refers to the process of compiling your C code to the desktop (host) development system using a standard C compiler, in this case the gcc compiler included with the Impulse CoDeveloper tools.

To compile and simulate the application for the purpose of functional verification:

1. Select **Project** -> **Build Simulation Executable** (or click the **Build Simulation Executable** button) to build the **ComplexFIR.exe** executable. The **Build** console window will display the compile and link messages as shown below:



2. You now have a Windows executable representing the ComplexFIR application implemented as a desktop (console) software application. Run this executable by selecting **Project** -> **Launch Simulation Executable**. A command window will open and the simulation executable will run as shown below:



Verify that the simulation produces the output shown. Note that although the messages indicate that the **ComplexFIR** algorithm is running on the FPGA, the application (represented by hardware and software processes) is actually running entirely in software as a compiled, native Windows executable. The messages you will see have been generated as a result of instrumenting the application with simple printf statements such as the following:

```
#ifdef IMPULSE_C_TARGET
     // Print Acceleration Numbers
     printf ("\r\n--> Acceleration factor: %dX\r\n\n", TimeSA/TimeHA);
     printf ("------> Visit www.ImpulseC.com to learn more!");

     #if defined(XPAR_MICROBLAZE_ID)
           // Disable DCache
           microblaze_disable_dcache();
           microblaze_init_dcache_range(0,  XPAR_MICROBLAZE_0_DCACHE_BYTE_SIZE);
           // Disable ICache
           microblaze_disable_icache();
           microblaze_init_icache_range(0,  XPAR_MICROBLAZE_0_CACHE_BYTE_SIZE);

     #elif defined(XPAR_PPC440_VIRTEX5_ID)
           // Disable DCache
           XCache_DisableDCache();
           // Disable ICache
           XCache_DisableICache();
     #endif

#else
     printf ("COMPLETE APPLICATION\r\n");
     printf ("Press Enter to continue...\r\n");
     c = getc(stdin);
#endif
```

Notice in the above C source code that **#ifdef** statements have been used to allow the software side of the application to be compiled either for the embedded PowerPC processor, or to the host development system for simulation purposes.

See Also

Building the Application for the Target Platform

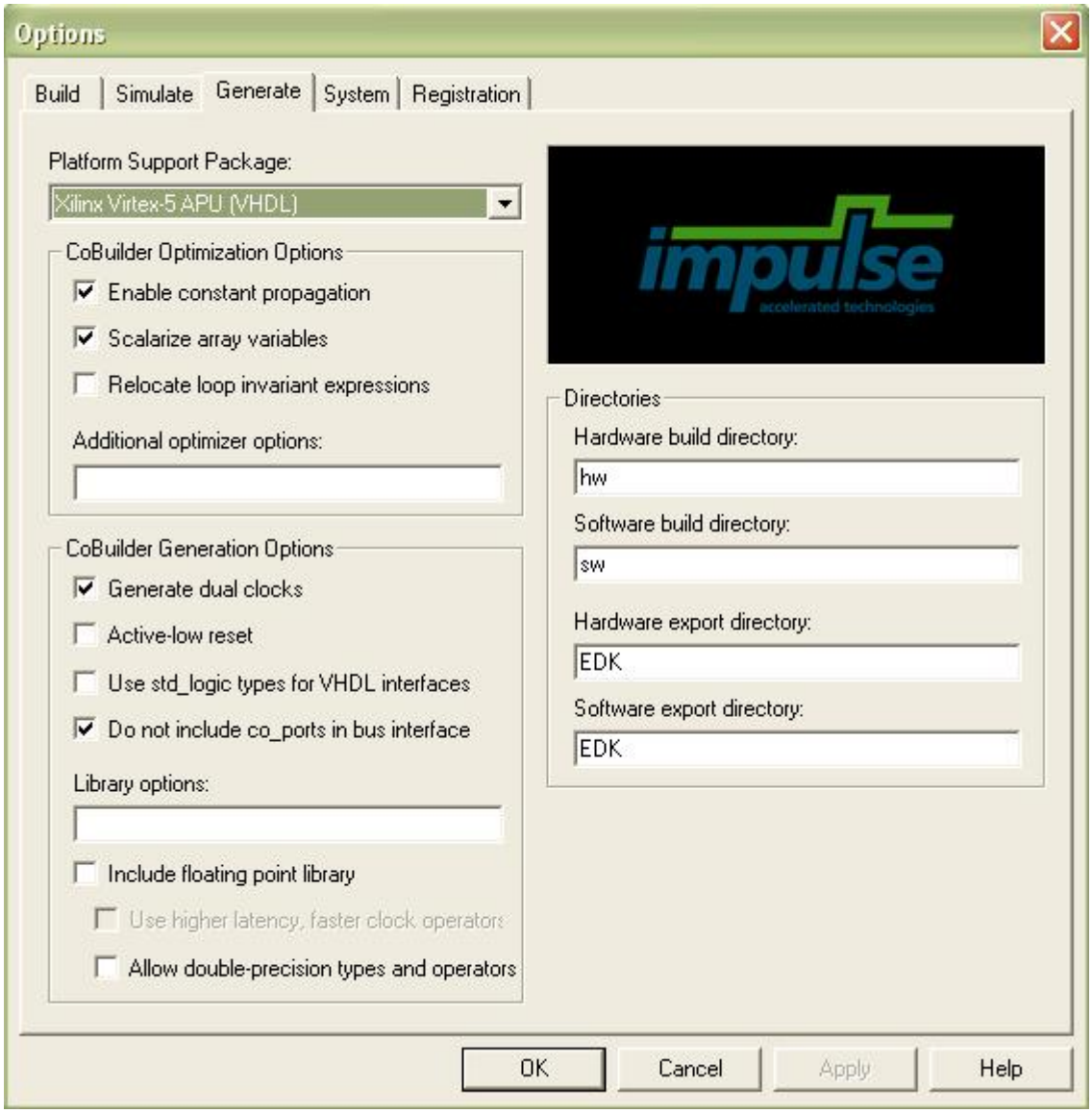
1.1.4 Building the Application for the Target Platform

Complex FIR Filter Tutorial for PowerPC, Step 4

The next step in the tutorial is to create FPGA hardware and related files from the C code found in the **Filter_hw.c** source file. This requires that we select a platform target, specify any needed options, and initiate the hardware compilation process.

Specifying the Platform Support Package

To specify a platform target, select from the menu **Project** -> **Options** to open the **Generate Options** dialog as shown below:



Specify **Xilinx Virtex-5 APU (VHDL)** as the **Platform Support Package**. Also specify **hw** and **sw** for the hardware and software directories as shown, and specify **EDK** for both the hardware and software export directories. Also ensure that the **Generate dual clocks** option is checked, which will allow the generated hardware core to run at a different clock speed than the system bus speed on the FPGA.

Click **OK** to save the options and exit the dialog.

Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces and library files, select from the menu **Project** -> **Generate HDL**. A series of processing steps will run in the **Build** console window as shown below:

Note: the processing of this example may require a few minutes to complete, depending on the performance of your system.

When processing has completed you will have a number of resulting files created in the **hw** and **sw** subdirectories of your project directory.

See Also

Exporting Files from CoDeveloper

1.1.5 Exporting Files from CoDeveloper

Complex FIR Filter Tutorial for PowerPC, Step 5

Recall that in **Step 4** you specified the directory **EDK** as the export target for hardware and software. These export directories specify where the generated hardware and software processes are to be copied when the **Export Software** and **Export Hardware** features of **CoDeveloper** are invoked. Within these target directories (in this case **EDK**), the specific destination (which may be a subdirectory under **EDK**) for each file previously generated is determined from the **Platform Support Package** architecture library files. It is therefore important that the correct **Platform Support Package** (in this case **Xilinx Virtex-5 APU**) is selected prior to starting the export process.

To export the files from the build directories (in this case **hw** and **sw**) to the export directories (in this case the **EDK** directory), select **Project** -> **Export Generated Hardware (HDL)** and **Project** -> **Export Generated Software** from the menu. The Build console window will display some processing messages, as shown below:

Export the Hardware Files

```
Build
"C:/Impulse/CoDeveloper3_30/bin/impulse_export" -hardware -srcdirhw "-aC:/Impulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml" FIR_Accelerator.xic "EDK"Impulse C
Design Exporter
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/Xilinx/V5/PPC/APU/cpu.xml ...
Loading FIR_Accelerator.xic ...

======== Build of target 'export_hardware' complete ========
```

Export the Software Files

```
Build
chmod -R +rw sw
"C:/Impulse/CoDeveloper3_30/bin/impulse_export" -software -srcdirsw "-aC:/Impulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml" FIR_Accelerator.xic "EDK"
Impulse C Design Exporter
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.Loading C:/Impulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/Xilinx/V5/PPC/APU/cpu.xml ...
Loading FIR_Accelerator.xic ...

======== Build of target 'export_software' complete ========
```

*Note: you must select BOTH **Export Software** and **Export Hardware** before going onto the next step.*

You have now exported all necessary files from **CoDeveloper** to the Xilinx tools environment.

See Also

Creating the Platform Using the Xilinx Tools

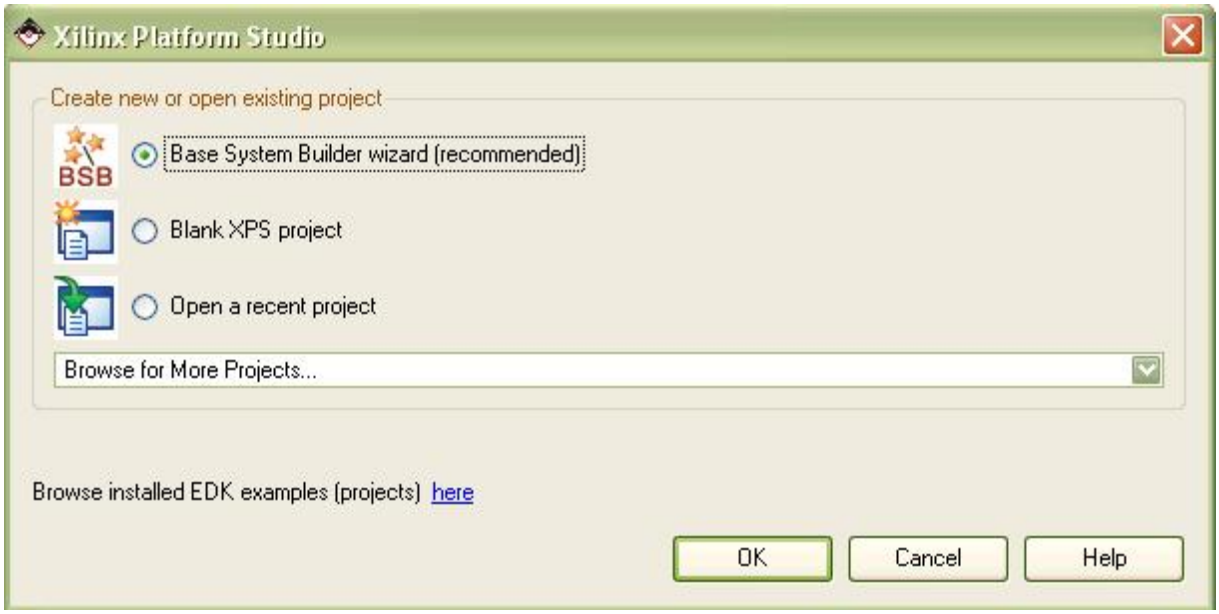
1.1.6 Creating a Platform Using Xilinx Tools

Complex FIR Filter Tutorial for PowerPC, Step 6

From the previous step, **CoDeveloper** creates a number of hardware and software-related output files that must all be used to create a complete hardware/software application on the target platform (in this case a **Xilinx FPGA** with an embedded **PowerPC** processor). This section will walk you through the file export/import process for this example, using the **Xilinx EDK System Builder, Xilinx Platform Studio**.

Creating a New Xilinx Platform Studio Project

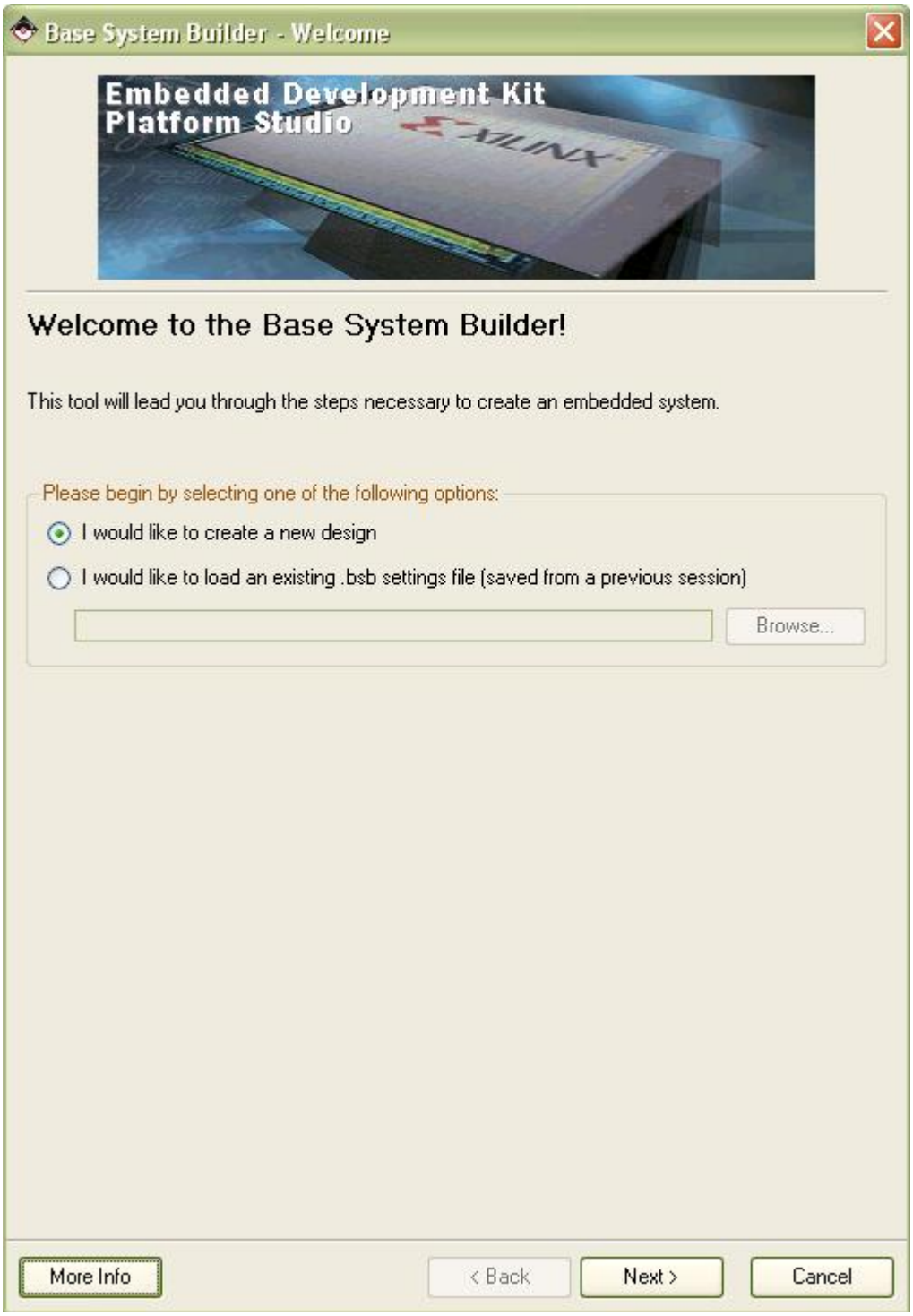
Now we'll move into the Xilinx tool environment. Begin by launching **Xilinx Platform Studio** from the **Windows Start** ->**Xilinx ISE Design Suite 10.1** -> **EDK** -> **Xilinx Platform Studio**. The **Xilinx Platform Studio** dialog appears as shown below:



Select the **Base System Builder wizard (recommended)**, and click **OK**.

Next, in the **Create New XPS Project Using BSB Wizard** dialog, click **Browse** and navigate to the directory you created for your **Xilinx EDK** project files. For this tutorial we choose the directory name **EDK** , which is also the directory name we specified earlier in the **Generate Options** dialog. Click **Save** to create a project file called **system.xmp** (you can specify a different project name if desired):

Now click **OK** in the **Create New XPS Project Using BSB Wizard** dialog. The **Base System Builder - Welcome** page will appear as shown below:

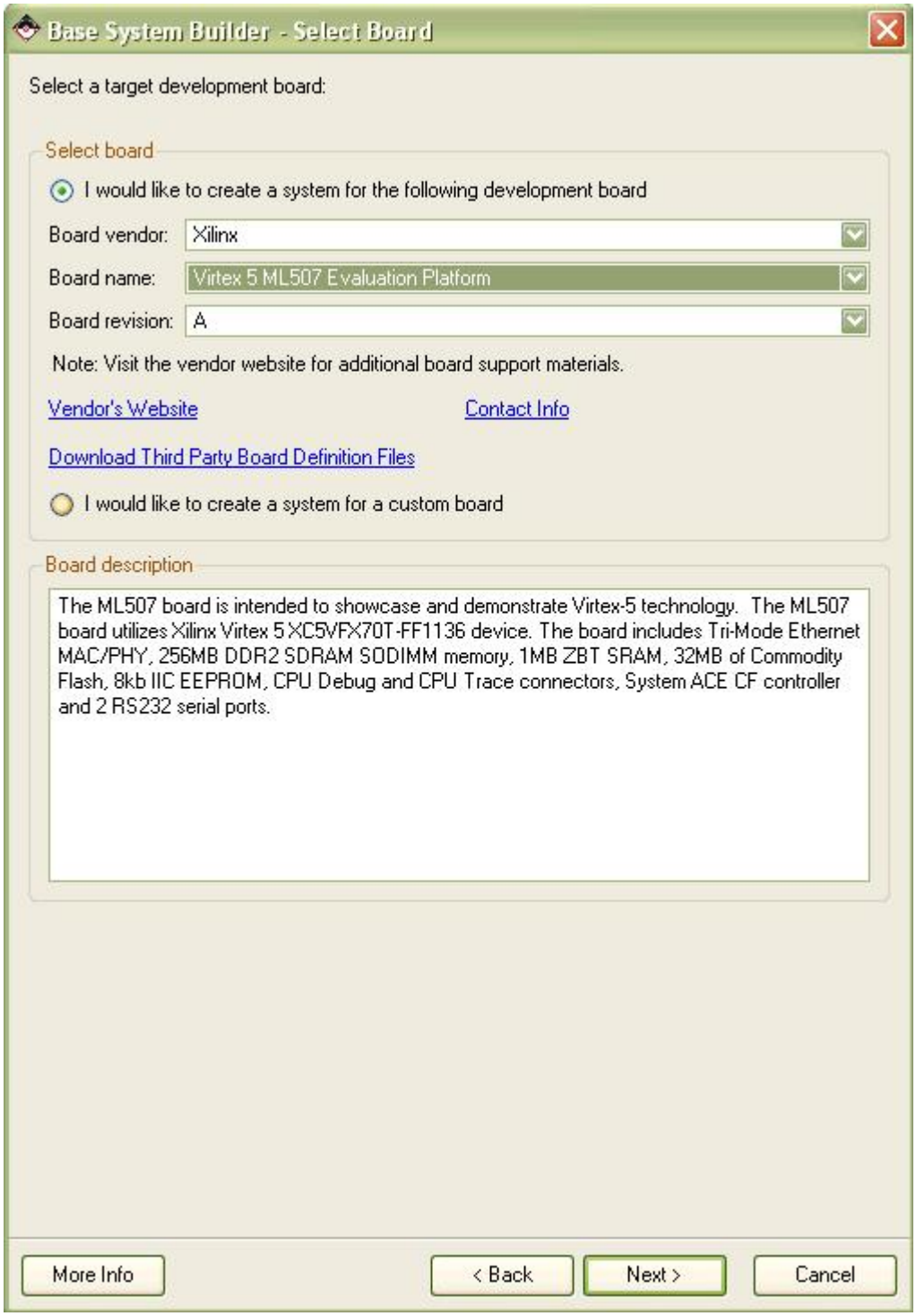


Select **I would like to create a new design** (the default), then click **Next** to choose your target board.

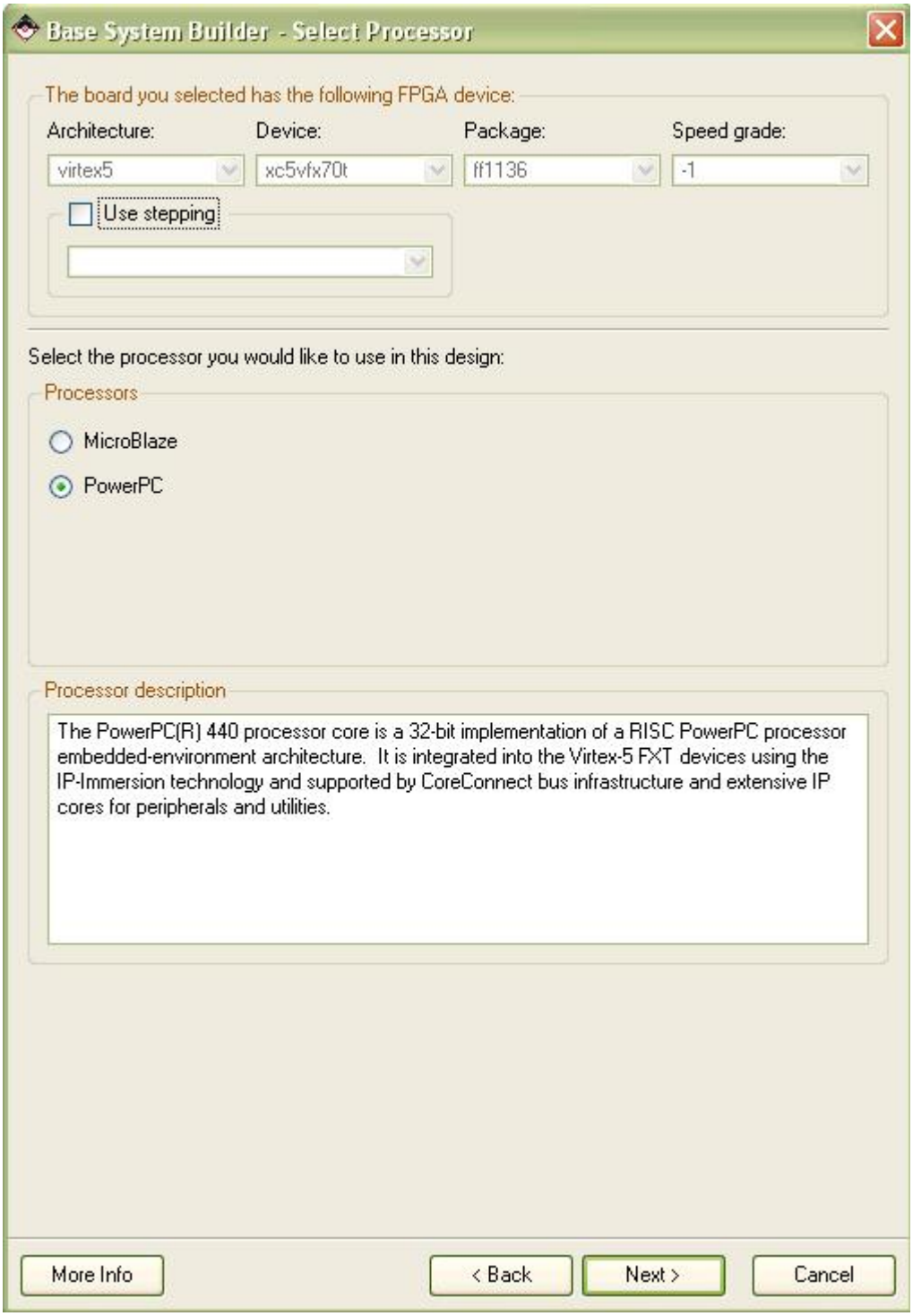
Choose your development board from the dropdown boxes. This example will use the following board (you should choose the reference board you have available for this step):

Board Vendor: **Xilinx**

Board Name: **Virtex 5 ML507 Evaluation Platform**
Board Revision: **A**



Click **Next** to continue with the **Base System Builder Wizard**. In the next wizard page, make sure that **PowerPC** is selected as the processor:



Click **Next** to continue with the **Base System Builder Wizard**.

*Note: the **Base System Builder** options that follow may be different depending on the development board you are using.*

The next steps will demonstrate how to configure the **PowerPC** processor and create the necessary I/O interfaces for our sample application.

See Also

Configuring the New Platform

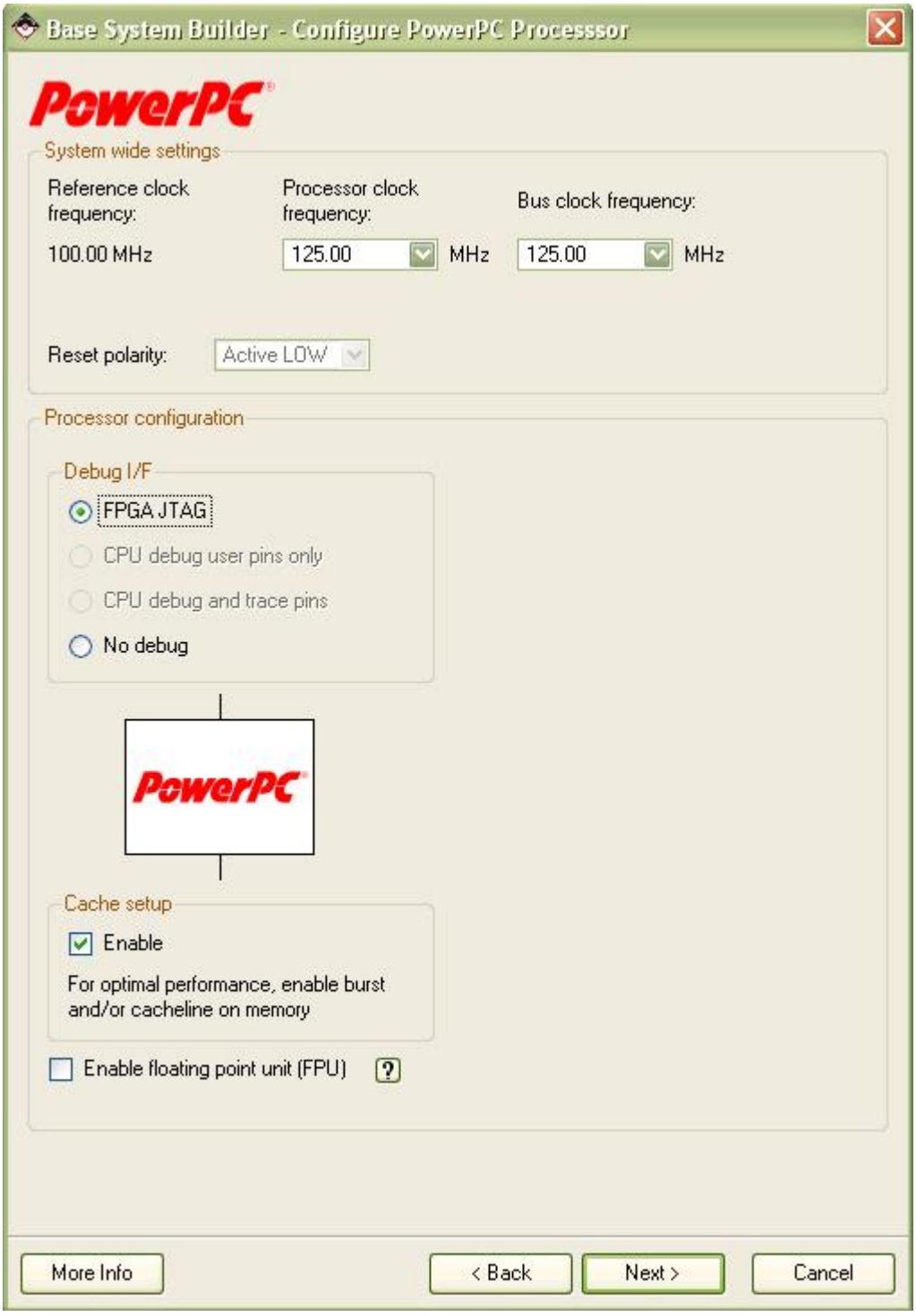
1.1.7 Configuring the New Platform

Complex FIR Filter Tutorial for PowerPC, Step 7

Now that you have created a basic PowerPC project in the **Base System Builder Wizard**, you will need to specify additional information about your platform in order to support the requirements of your software/hardware application. Continuing with the steps provided in the **Base System Builder Wizard**, specify the following information in the Configure processor page, making sure to increase the local data and instruction memory as shown:

System Wide Setting
Reference Clock Frequency: **125 MHz**
Bus Clock Frequency: **125 MHz**

Processor Configuration
Debug I/F: **FPGA JTAG** (default setting)
Cache setup: **Enable**
Unselect **Enable floating point unit (FPU)**

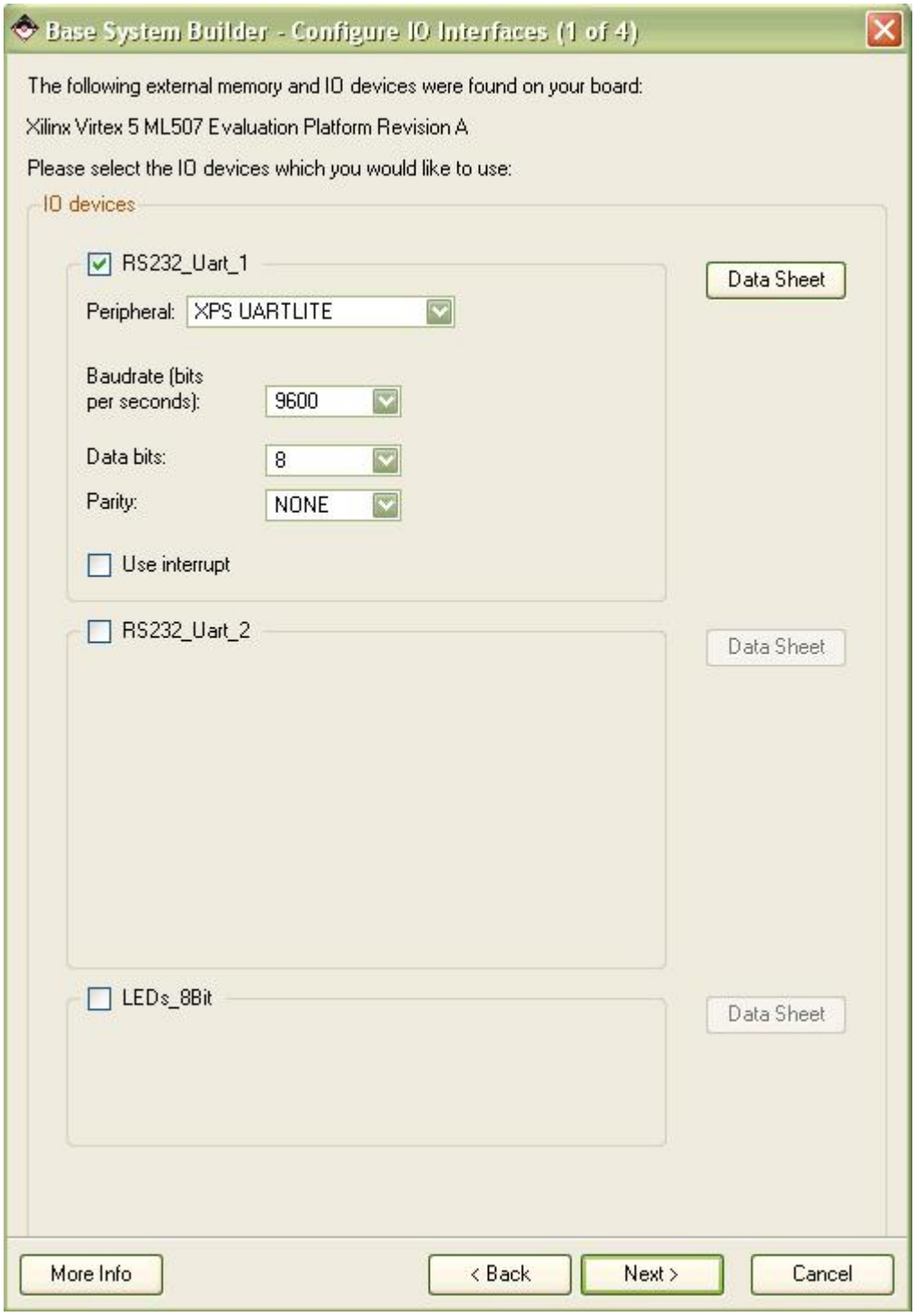


Click **Next** to continue with the wizard. You will now be presented with a series of pages specifying the I/O peripherals to be included with your processor. (The actual layout of these pages will depend on your screen resolution.) Select one **RS232** device peripheral by setting the following options:

I/O Device: RS232_Uart_1
Peripheral: XPS UARTLITE
Baudrate: 9600

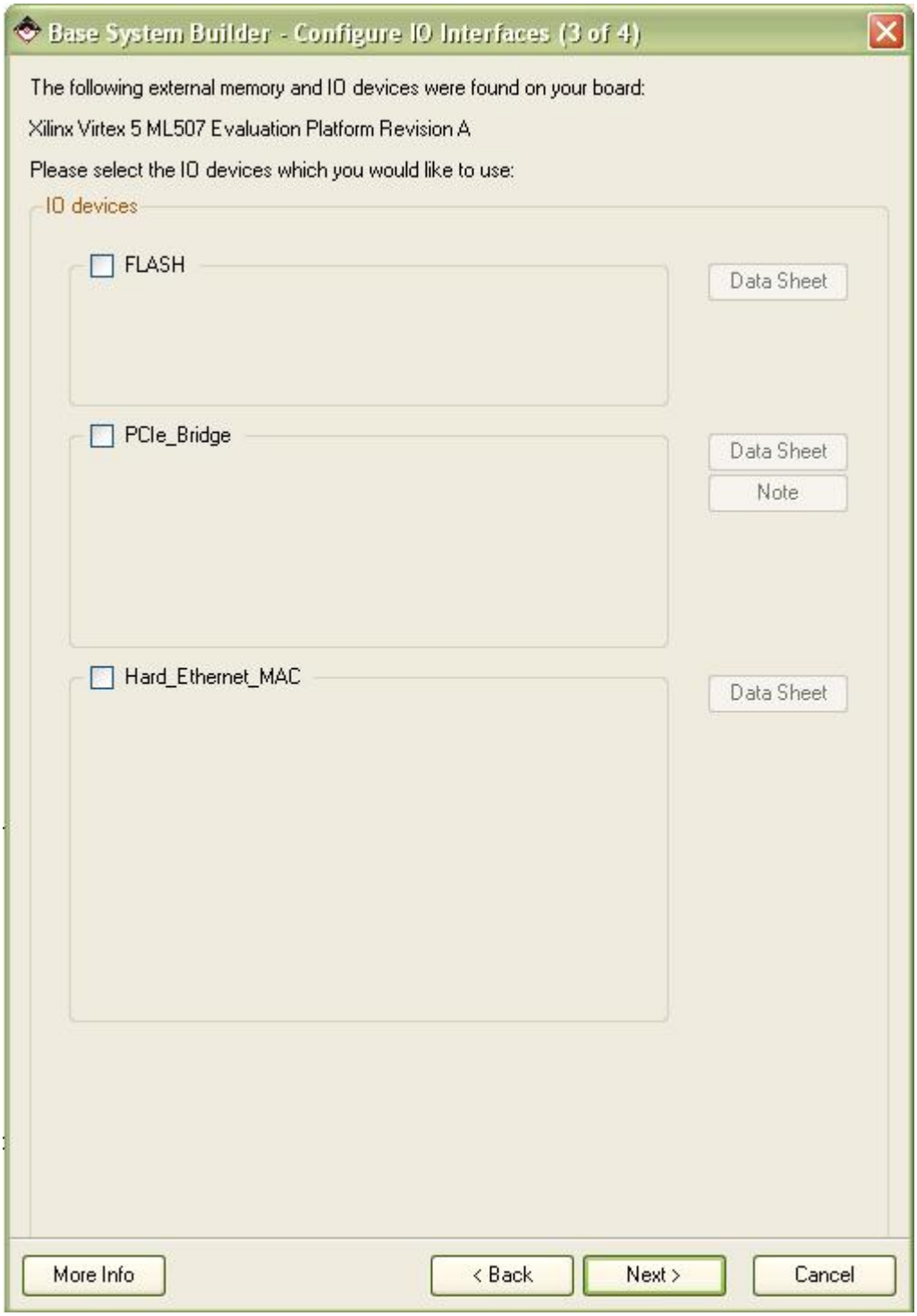
Data Bits: 8
Parity: NONE
Use Interrupt: disabled

Unselect the **RS232_Uart_2** and **LED_8Bit** devices.



Click **Next** to continue. In the next wizard page, unselect all the **IO devices** as shown below:

Click **Next** to continue. Again, in the next wizard page, unselect all the **IO devices** as shown below:

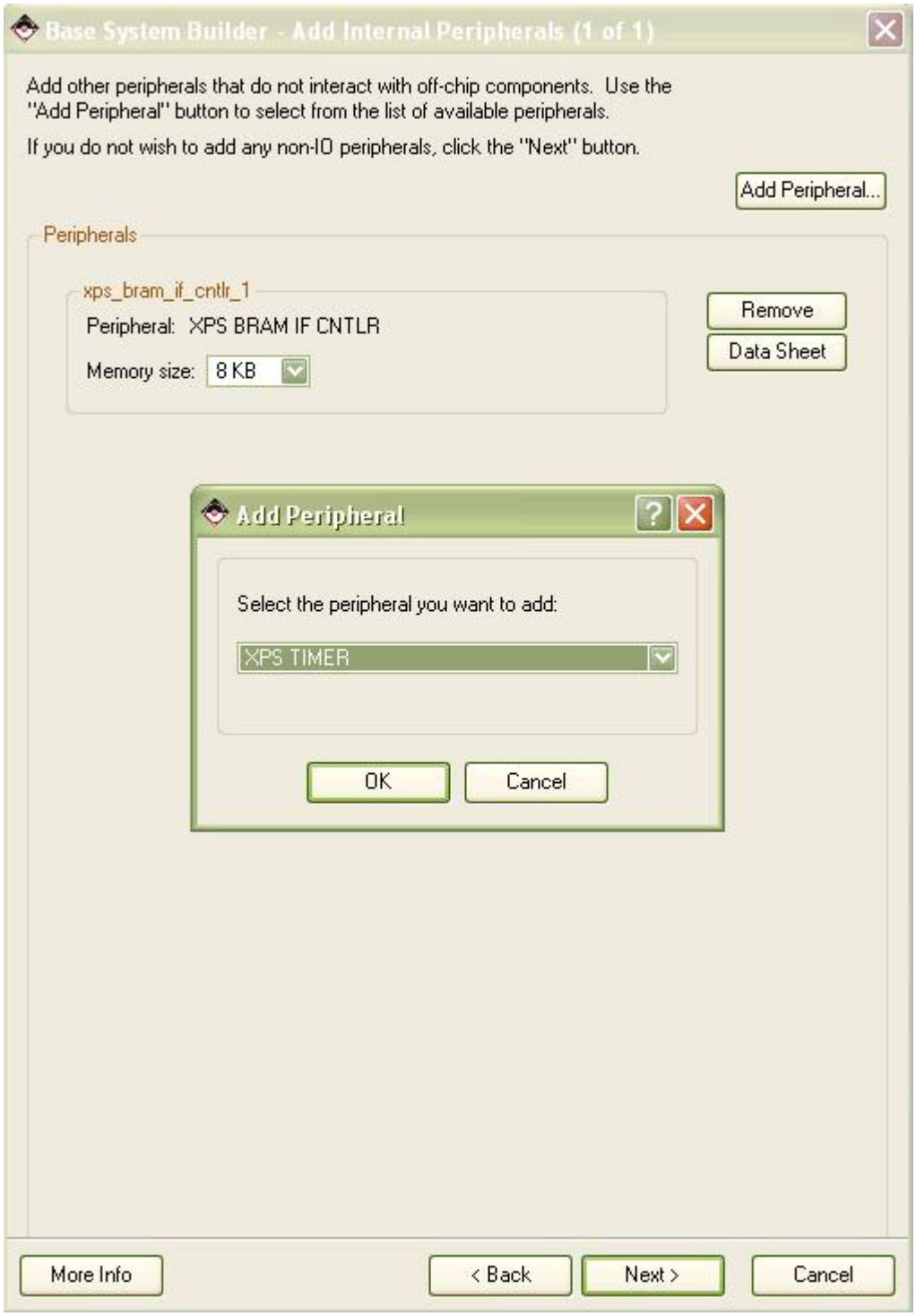


Click **Next** to continue. In the next wizard page, disable all the **IO interfaces** except the **DDR2_SDRAM**:

I/O Devices: DDR2_SDRAM
Peripheral: MPMC

Click **Next** to continue. In the **Add Internal Peripherals** page, the **xps_bram_if_cntlr** is being added with **Memory size** of **8KB**.

Click the **Add Peripheral...** button, and select the **XPS TIMER** peripheral from the dropdown list as shown below:

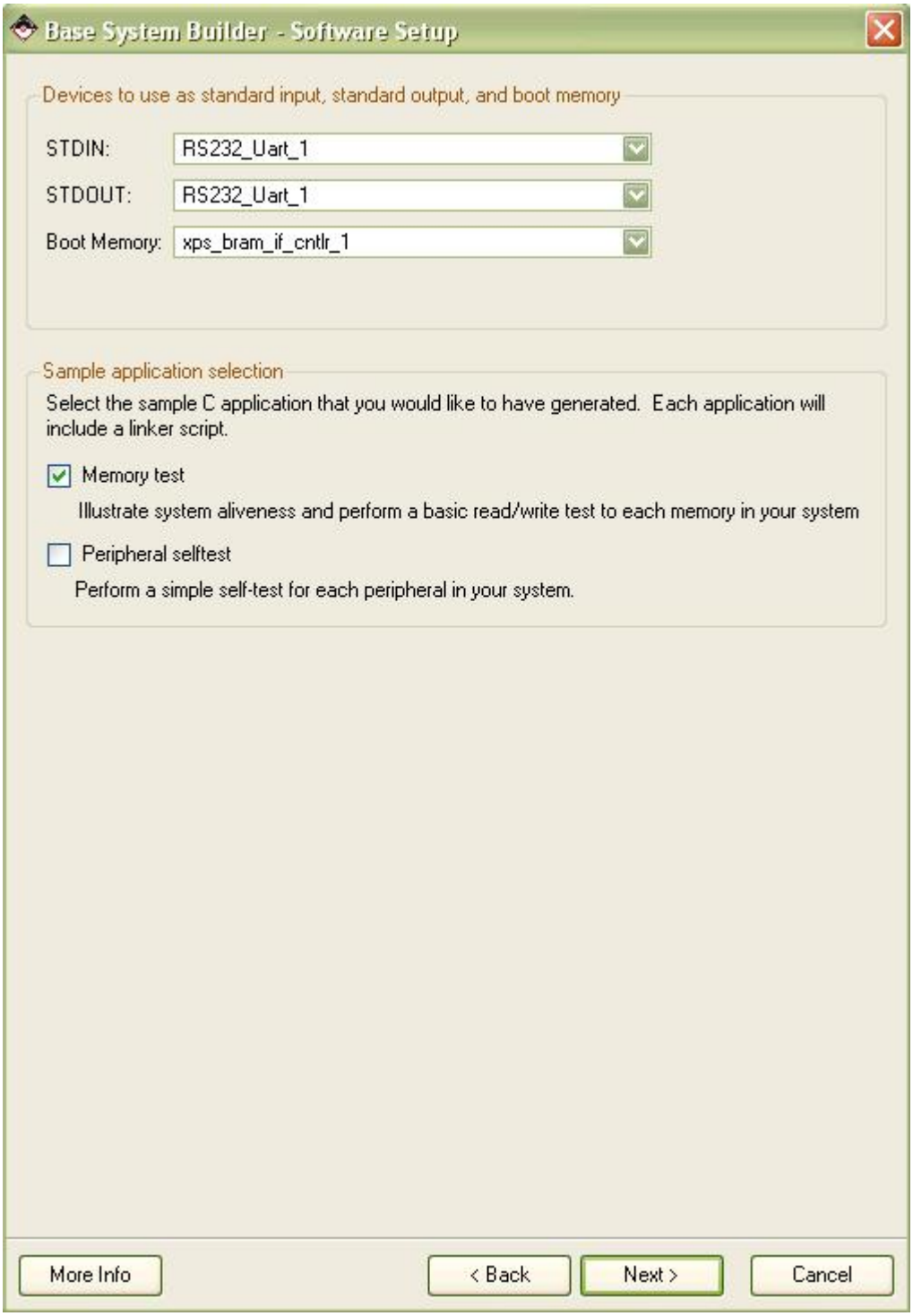


Click **Next**, and the **xps_timer_1** appears in the **Peripherals** list. Choose to the **Timer mode** as **One timer is present**, and do not **Use interrupt**.

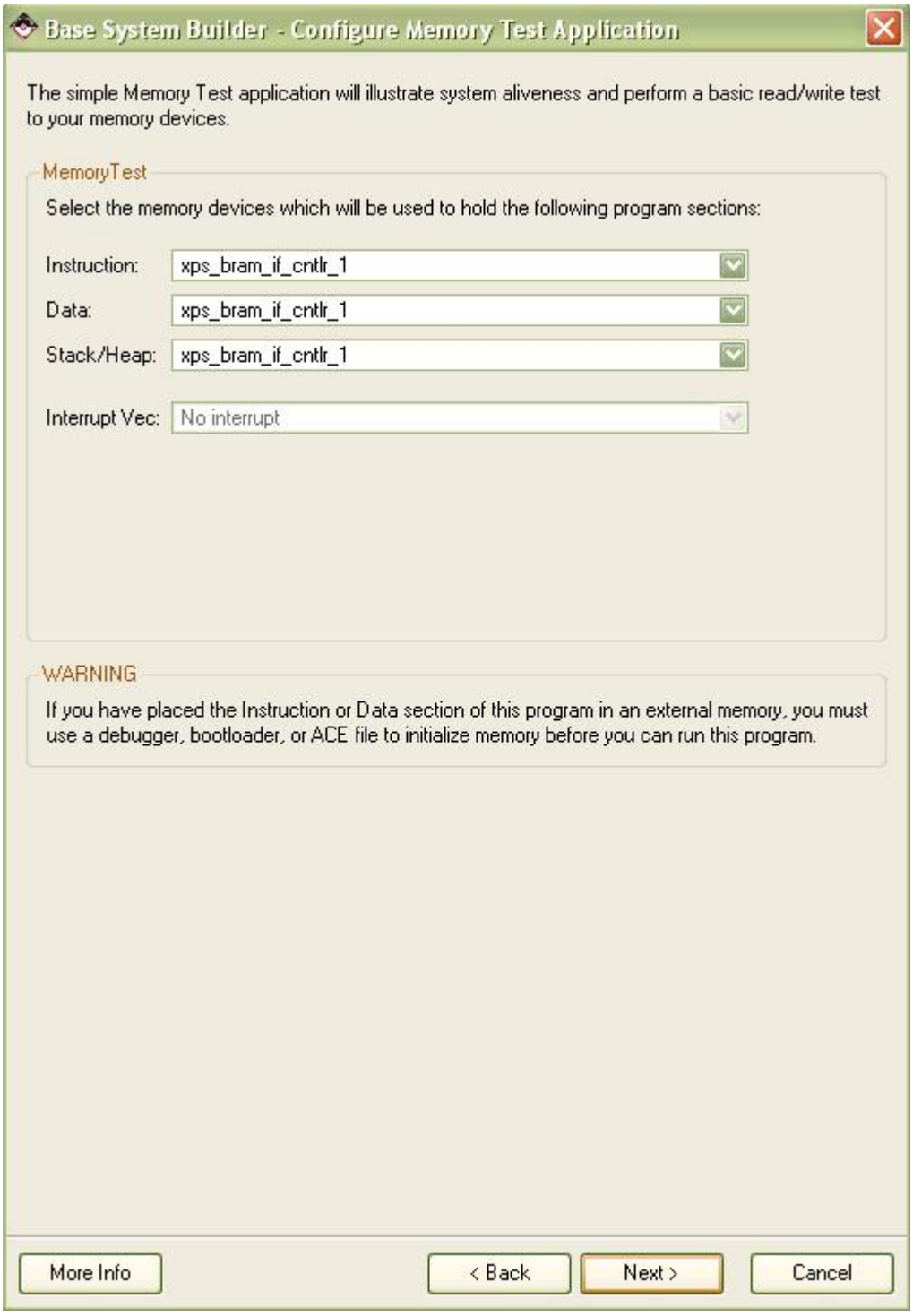
Click **OK** to add the above two peripherals.

In the **Cache Setup** page, choose the cache settings as follows:

On the **Software Setup** dialog that appears, select the **Memory test** option, and unselect the **Peripheral selftest** option:

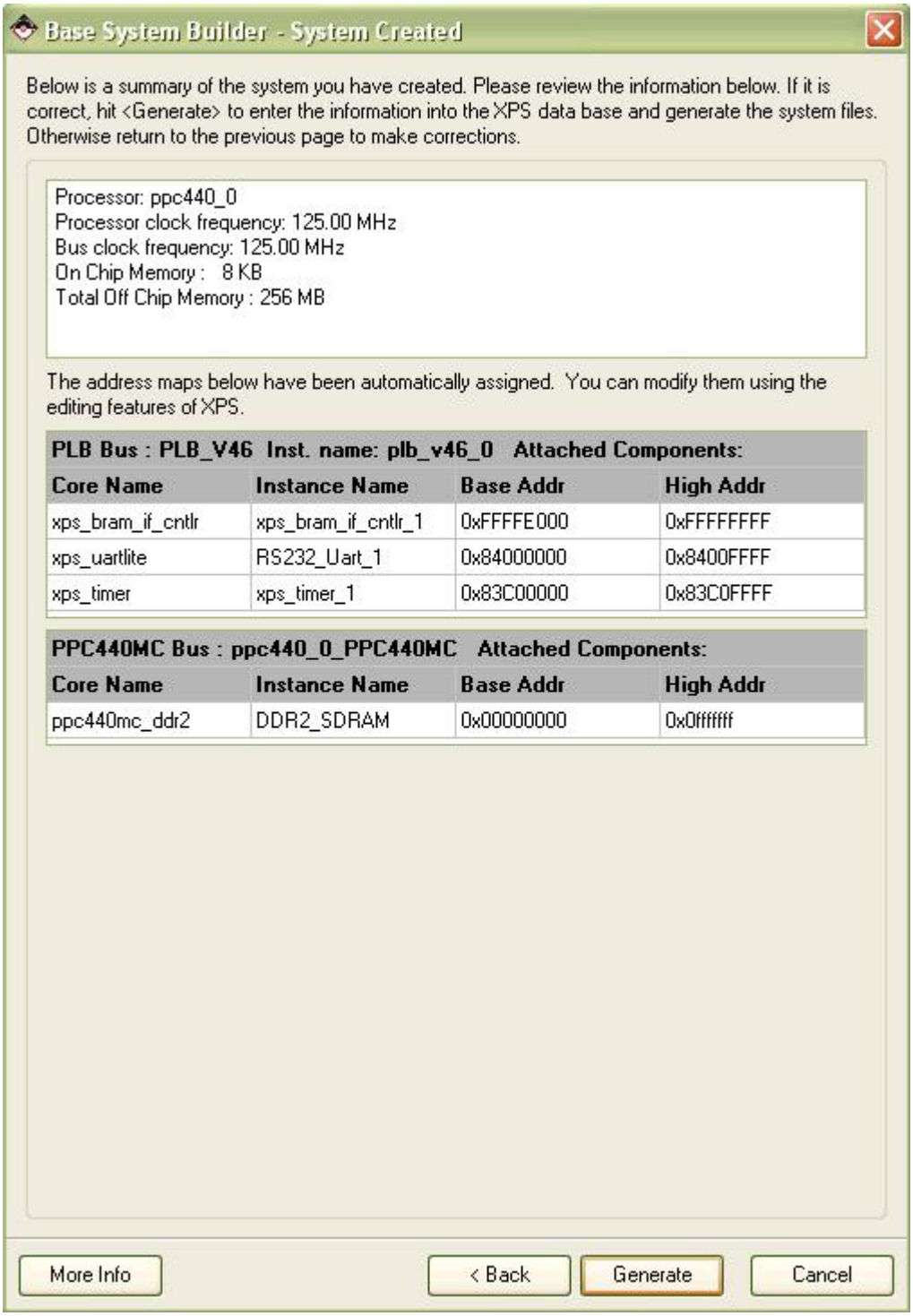


Click **Next** to view the **Memory Test** software settings as shown below:

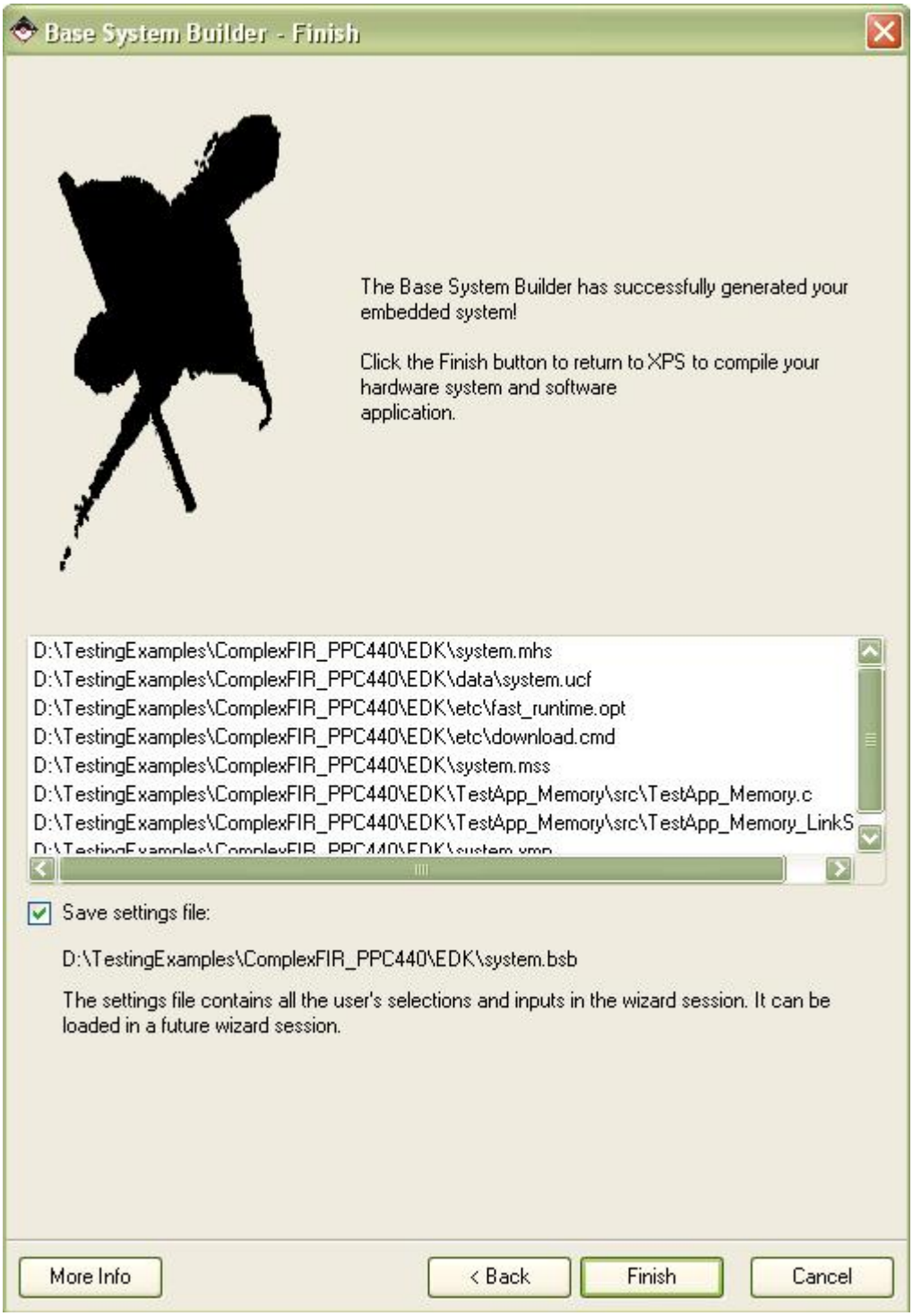


Click **Next** to accept the default **Memory Test** memory settings.

You have now configured the platform and processor features. The **Base System Builder Wizard** displays a summary of the system you have created:

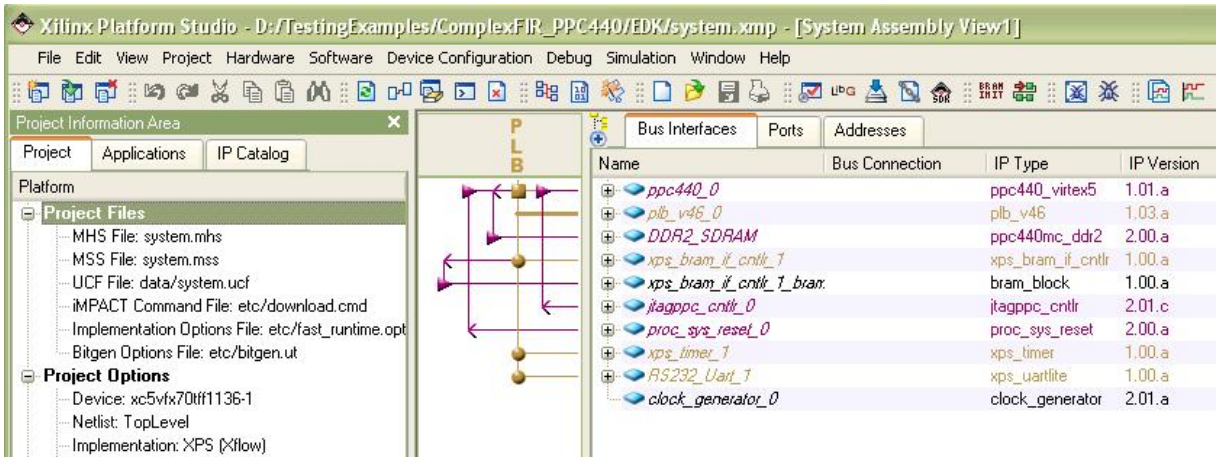


Click **Generate** to generate the system and project files. After this is done, the **Base System Builder** will display the **Finish** page as shown below:



Click **Finish** to close the wizard.

The **System Assembly View** of the **Platform Studio** should look like this:



See Also

Importing the Generated Hardware

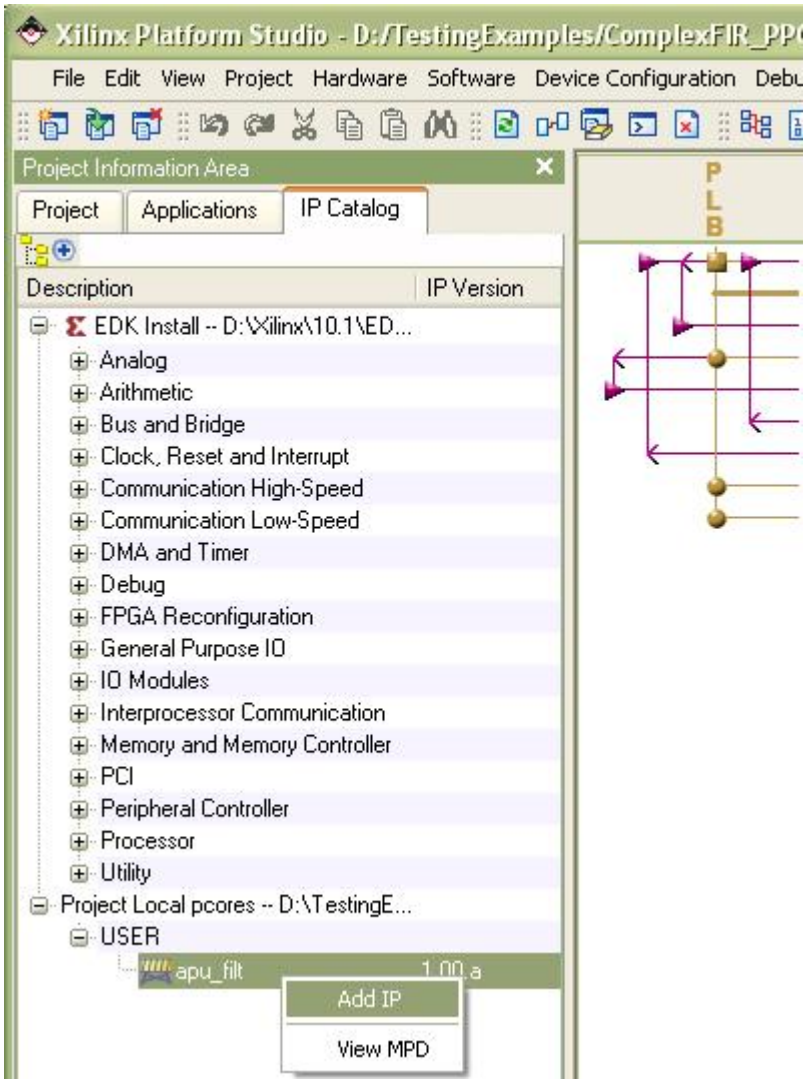
1.1.8 Importing the Generated Hardware

Complex FIR Filter Tutorial for PowerPC, Step 8

You will now create the target platform in the Xilinx Platform Studio. This procedure is somewhat lengthy but will only need to be done once for any new project.

Add the ComplexFIR Hardware IP Core

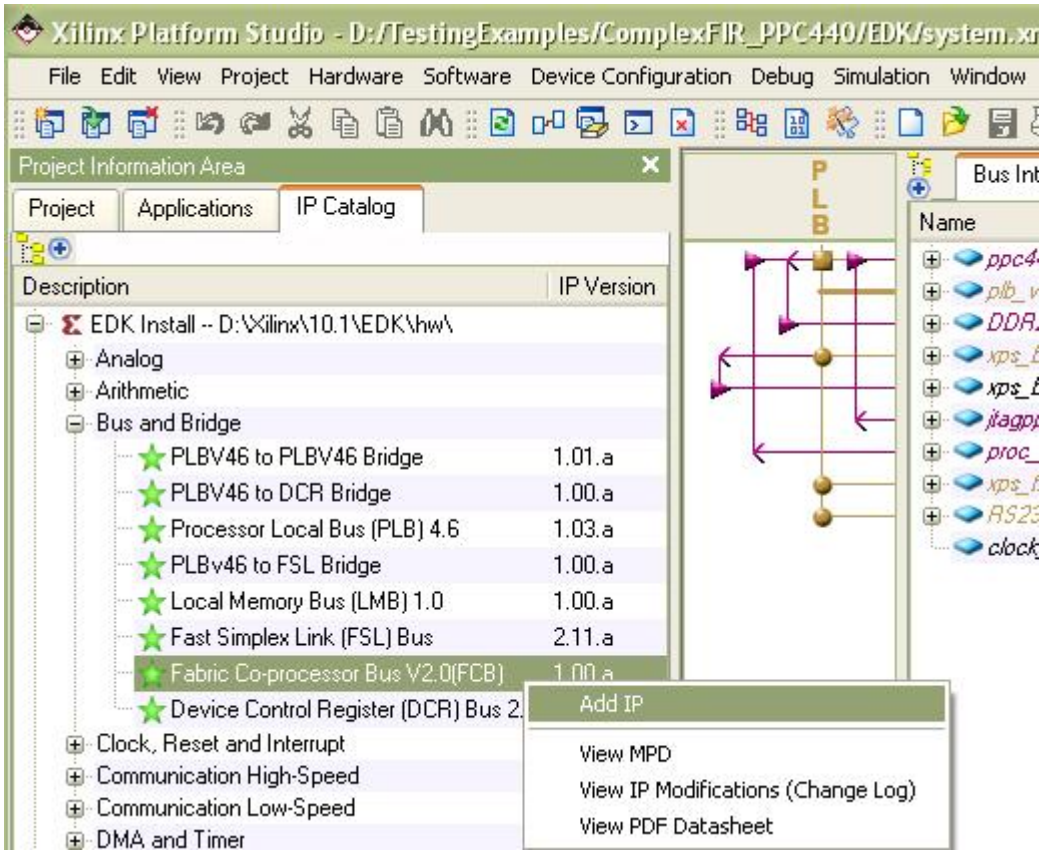
First, add the module representing the **ComplexFIR** hardware process to your development system. Switch to the **IP Catalog** tab in the **Project Information Area**. Expand **Project Local pcores** -> **USER** to reveal the generated **apu_filt** IP core. Right-click the **apu_filt** and select **Add IP** as shown below:



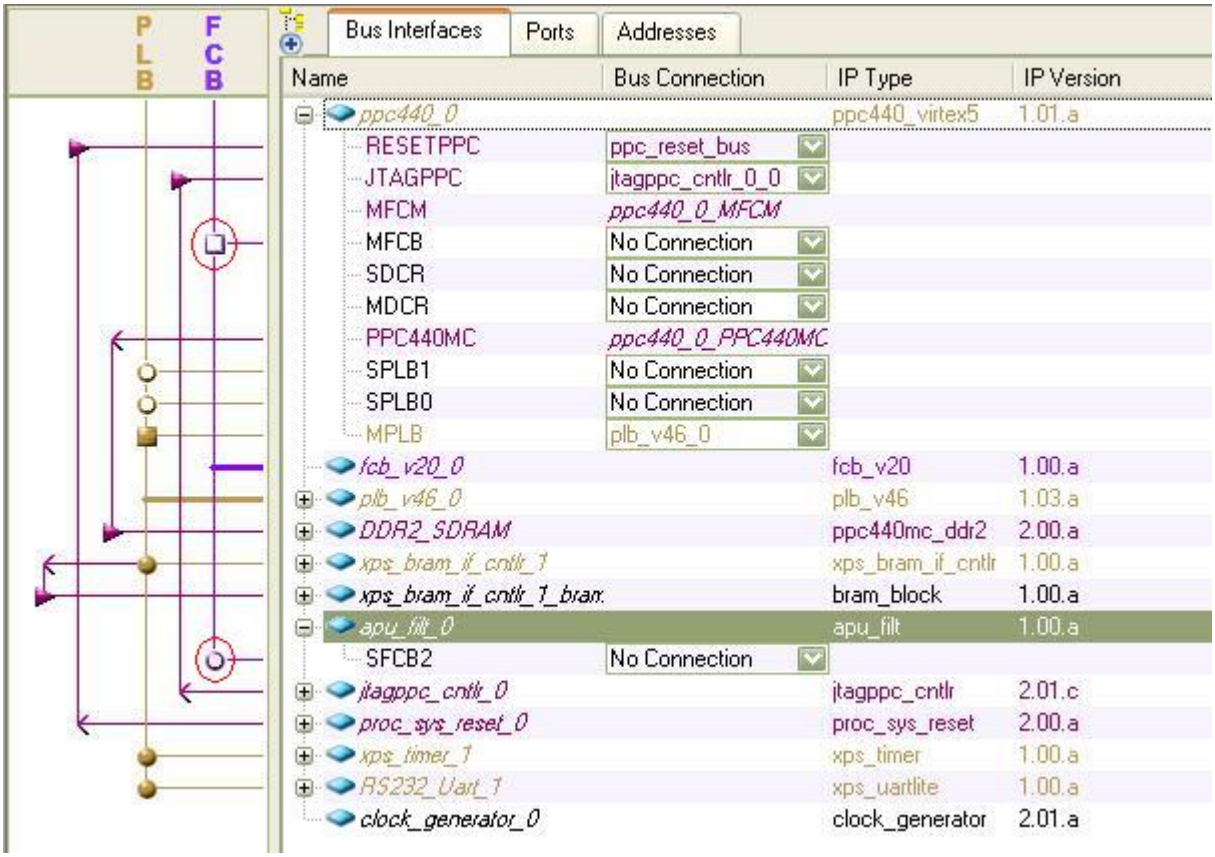
The **apu_filt** module will appear in the list of peripherals in the **System Assembly View** on the right.

Connect the IP Core to PowerPC

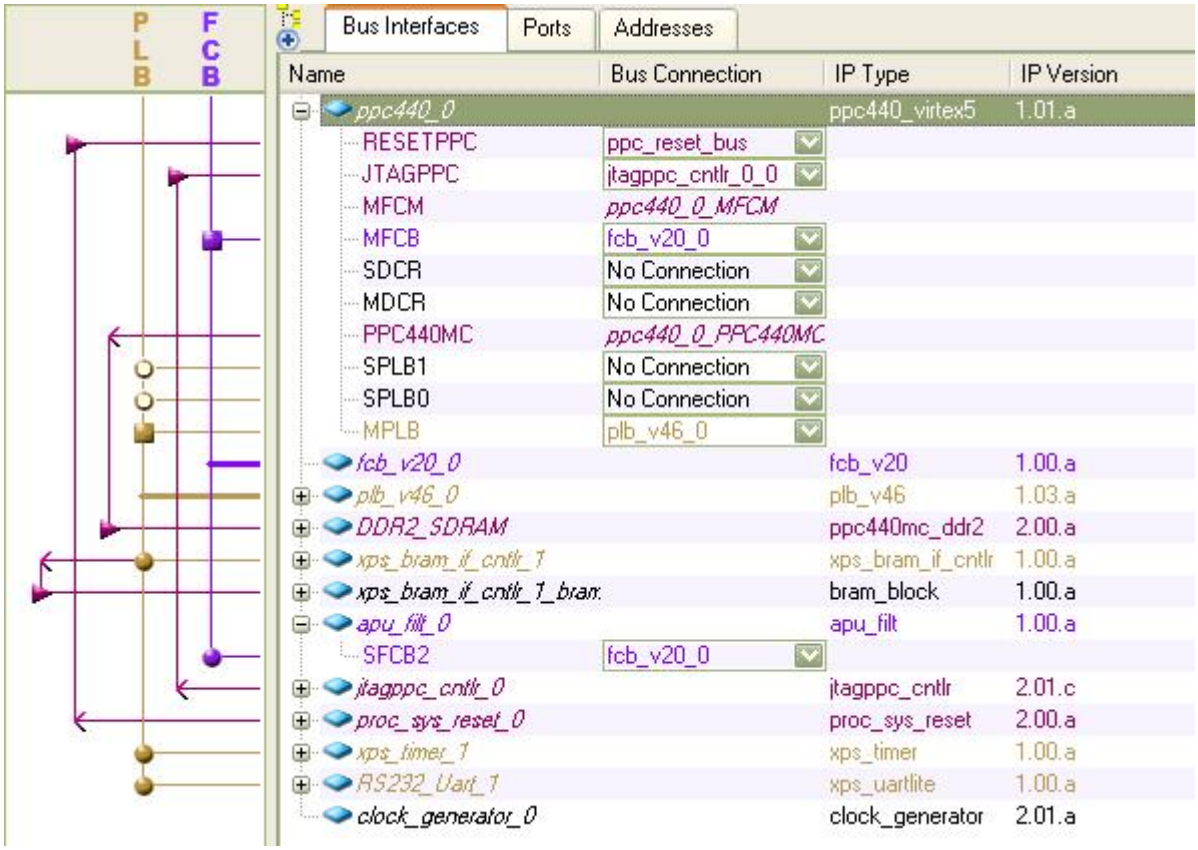
In order to connect the apu_filt IP core to the APU of the processor, a **Fabric Co-processor Bus** is needed for interfacing purposes. Click on the **IP Catalog** tab and select the **Fabric Co-processor Bus V2.0 (FCB)** IP core from the **Bus and Bridge** category. Right-click it and select **Add IP** as shown:



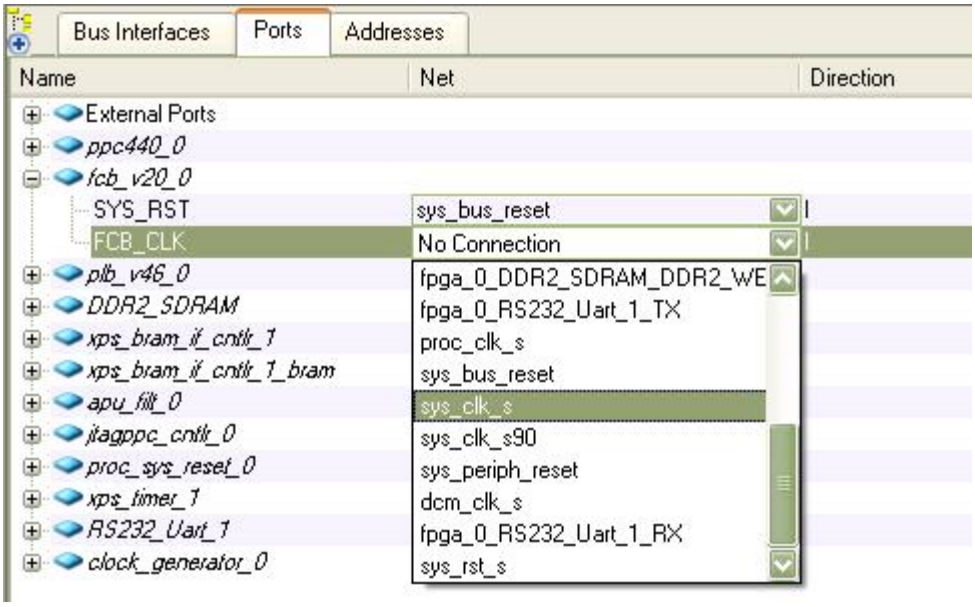
When you have added the **FCB** IP core, your project should look like this:



Notice that there are an empty square and an empty circle on the **FCB** bus. Click the empty square to connect the **ppc440_0**'s **MFCB** interface to the **fcb_v20_0** bus. Click the empty circle to connect the **apu_filt_0**'s **SFCB2** interface to the **fcb_v20_0** bus. They should be connected as shown below:

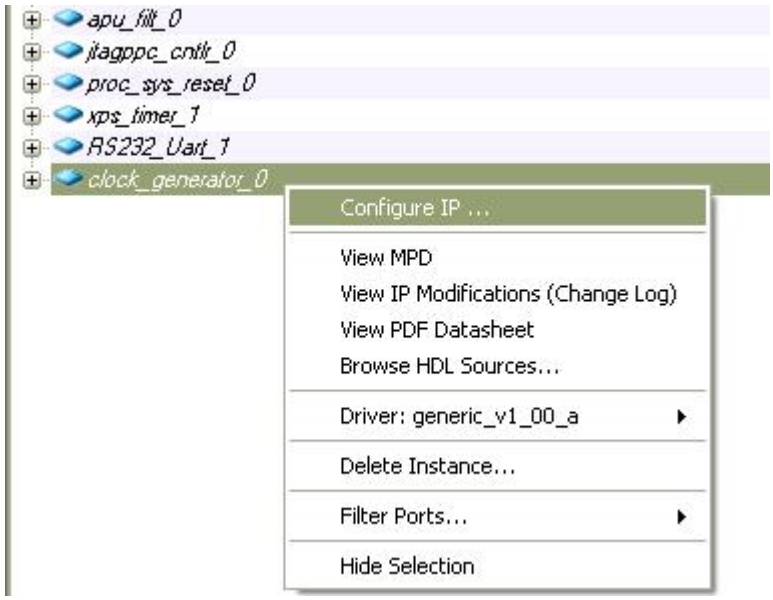


Two ports have to configured for the **FCB**. Click on the **Ports** tab in the **System Assembly View** and expand **fcb_v20_0** IP core. Set **SYS_Rst** to **sys_bus_reset** and set **FCB_Clk** to **sys_clk_s** as shown.

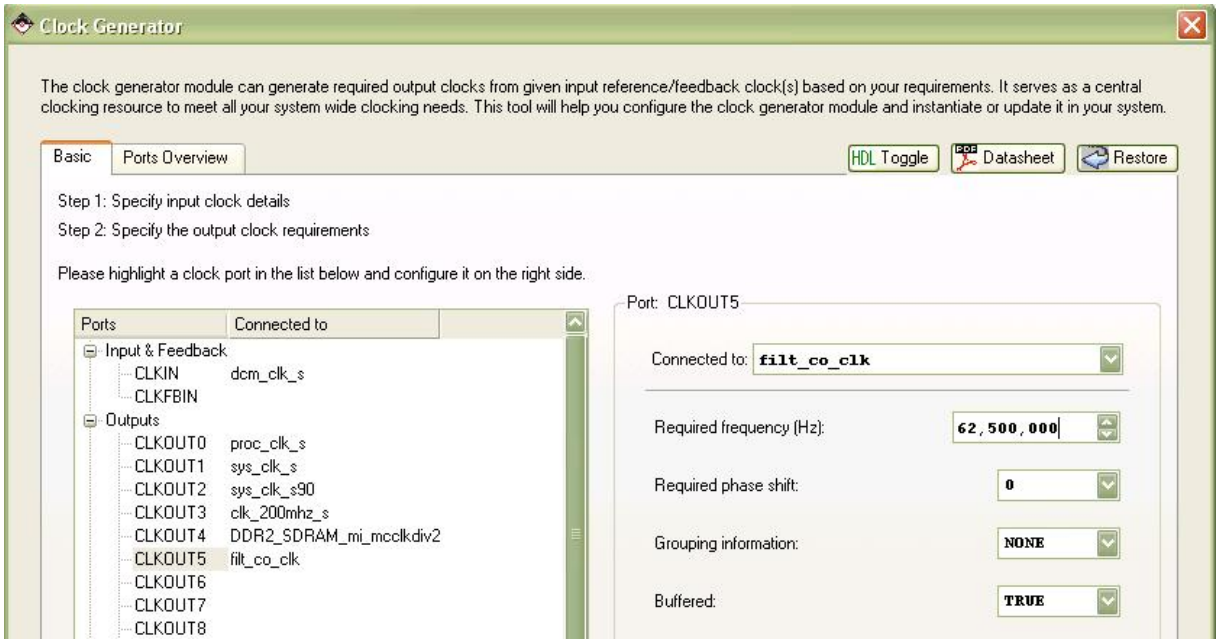


Configuring the Clock

The ComplexFIR hardware requires a separate clock source. For this purpose, we configure the **clock_generator_0** settings by right-clicking and selecting **Configure IP...** as shown:

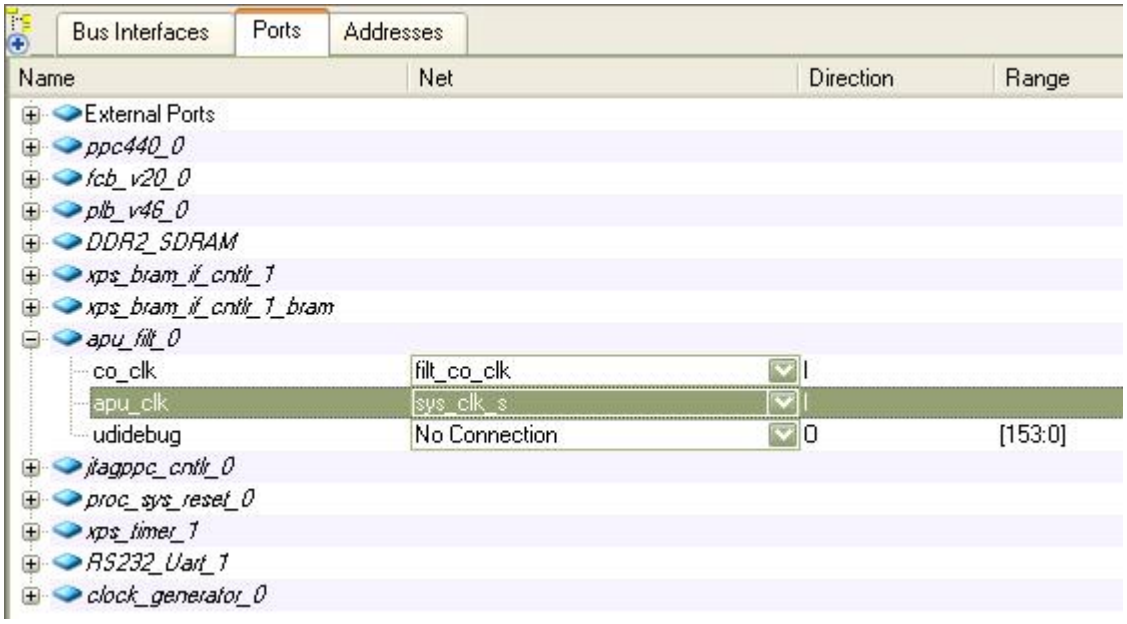


Here we add a new clock output from CLKOUT5 named **filt_co_clk**. The frequency is set to be **62,500,000 Hz**, which is half of the **125,000,000 Hz** system bus frequency.



Click **OK** to save the settings and exit the configure IP dialog.

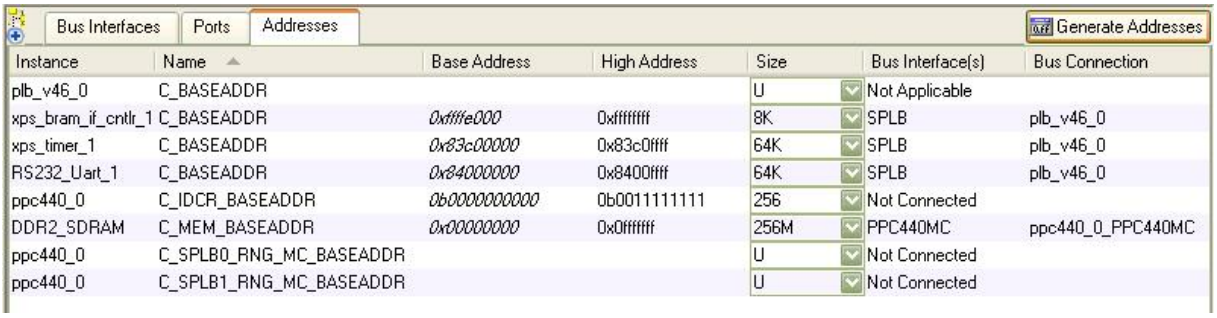
Select the **Ports** tab in the **System Assembly View** and expand **apu_filt_0**. This should reveal ports **co_clk** and **apu_clk**. The **co_clk** has to be connected to the **filt_co_clk** clock that we configured in the previous steps. The **apu_clk** should be connected to **sys_clk_s**, as shown below:



*Note: if **co_clk** is missing from the **apu_filt_0** section, then will need to return to step 3 of this tutorial and specify the **Dual Clock** option in the **CoDeveloper Generate Options** page.*

Generate the Addresses

Now you will need to set the addresses for each of the peripherals specified for the platform. This can be done simply by clicking on the **Generate Addresses** button in the **Addresses** tab and . The addresses will be assigned for you automatically:



You have now exported all necessary hardware files from **CoDeveloper** to the Xilinx tools environment and have configured your new platform. The next step will be to compile the HDL files and generate downloadable FPGA bitstream.

See Also

Generating the FPGA Bitmap

1.1.9 Generating the FPGA Bitmap

ComplexFIR Filter Tutorial for PowerPC, Step 9

At this point, if you have followed the tutorial steps carefully you have successfully:

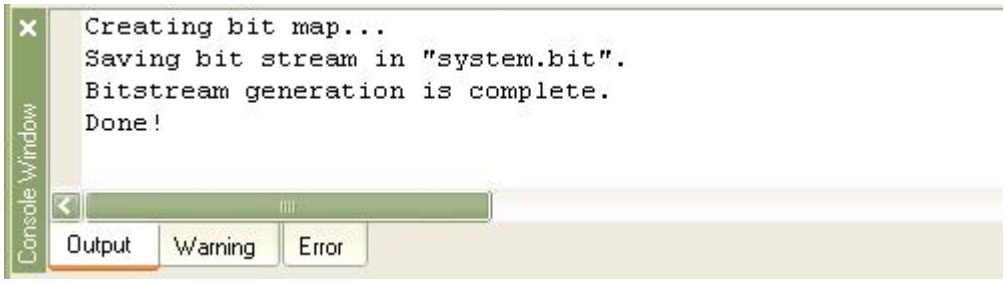
- Generated hardware and software files from the **CoDeveloper** environment.
- Created a new **Xilinx Platform Studio** project and created a new **PowerPC**-based platform.
- Imported your **CoDeveloper**-generated files to the **Xilinx Platform Studio** environment.
- Connected and configured the **Impulse C** hardware process to the **PowerPC** processor via the **FCB** bus.

You are now ready to generate the bitmap.

From within **Xilinx Platform Studio**, select the menu **Hardware** -> **Generate Bitstream**.

Note: this process may require 10 minutes or more to complete, depending on the speed and memory size of your development system.

After the generation process is done, the message in the **Output Console Window** will be like shown below:



Now we can move on to add our own software application.

See Also

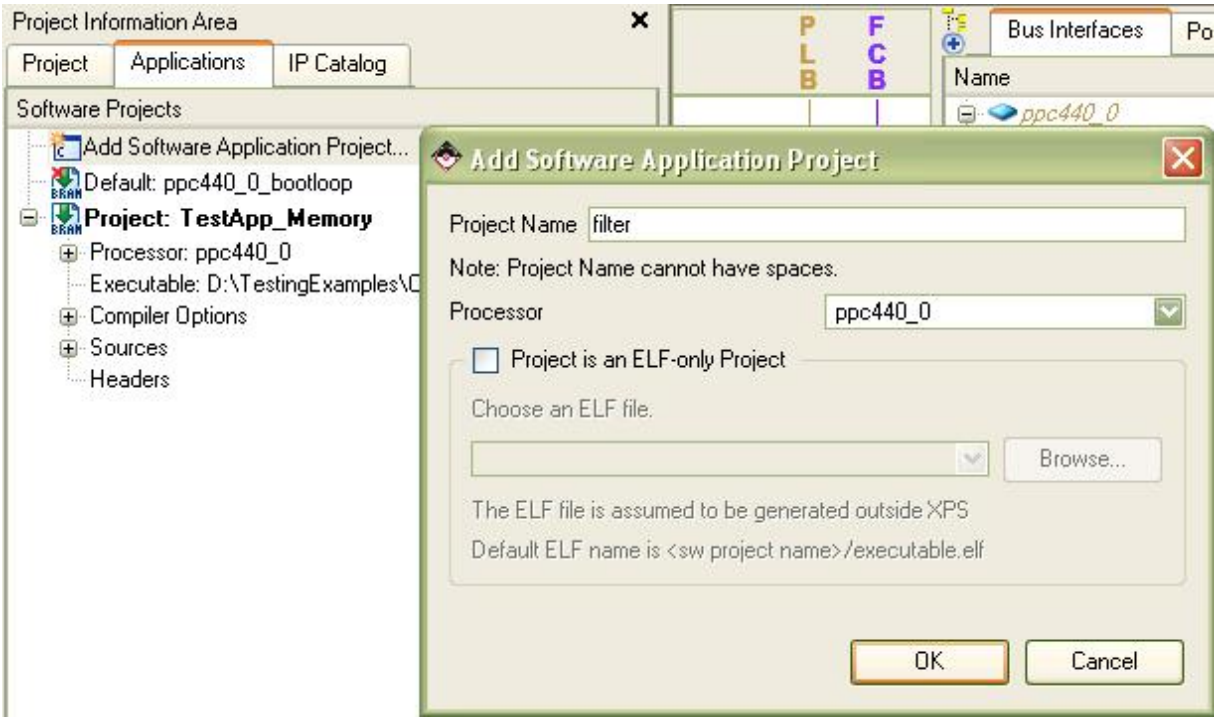
Importing the Application Software

1.1.10 Importing the Application Software

ComplexFIR Filter Tutorial for PowerPC, Step 10

You will now import the relevant software source files to your new **Platform Studio** project.

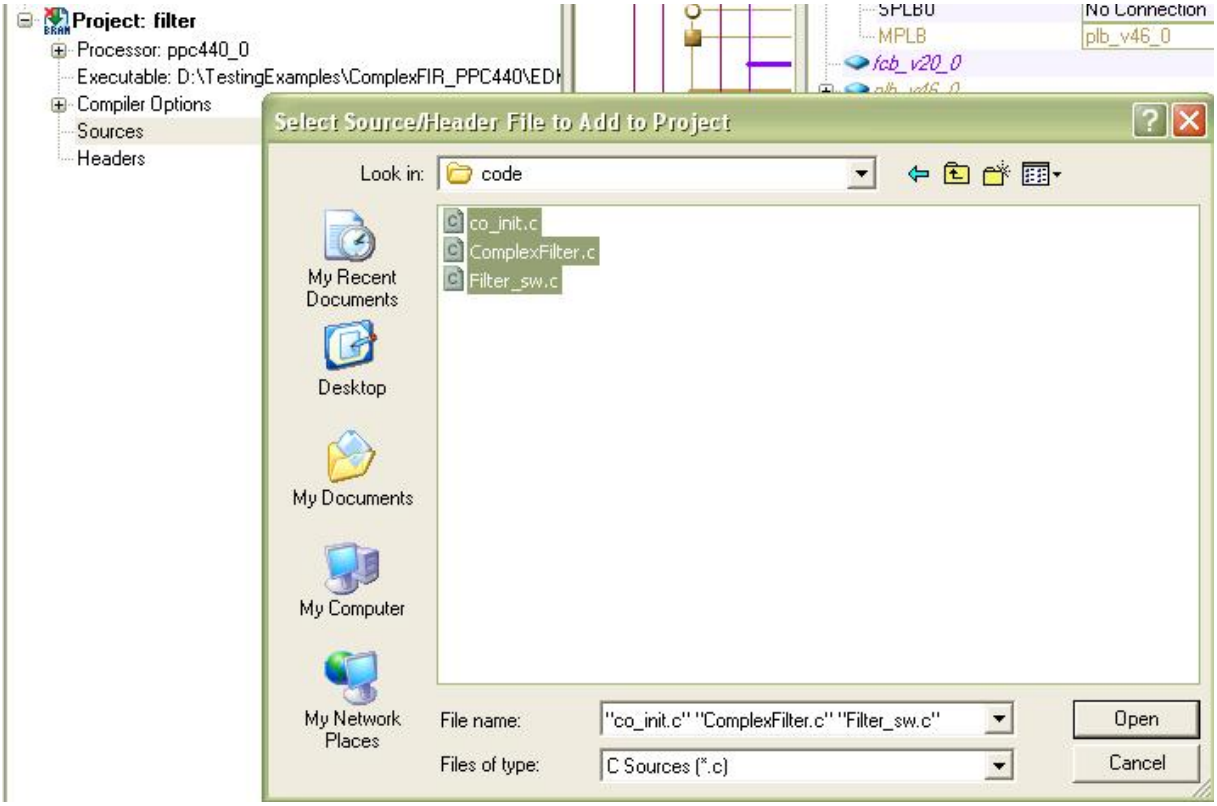
On the **Applications** tab of the **Project Information Area**, create a new software project by double-clicking the **Add Software Application Project...** item. An **Add Software Application Project** dialog will appear. Type in the project name: **filter**.



Click **OK** to continue.

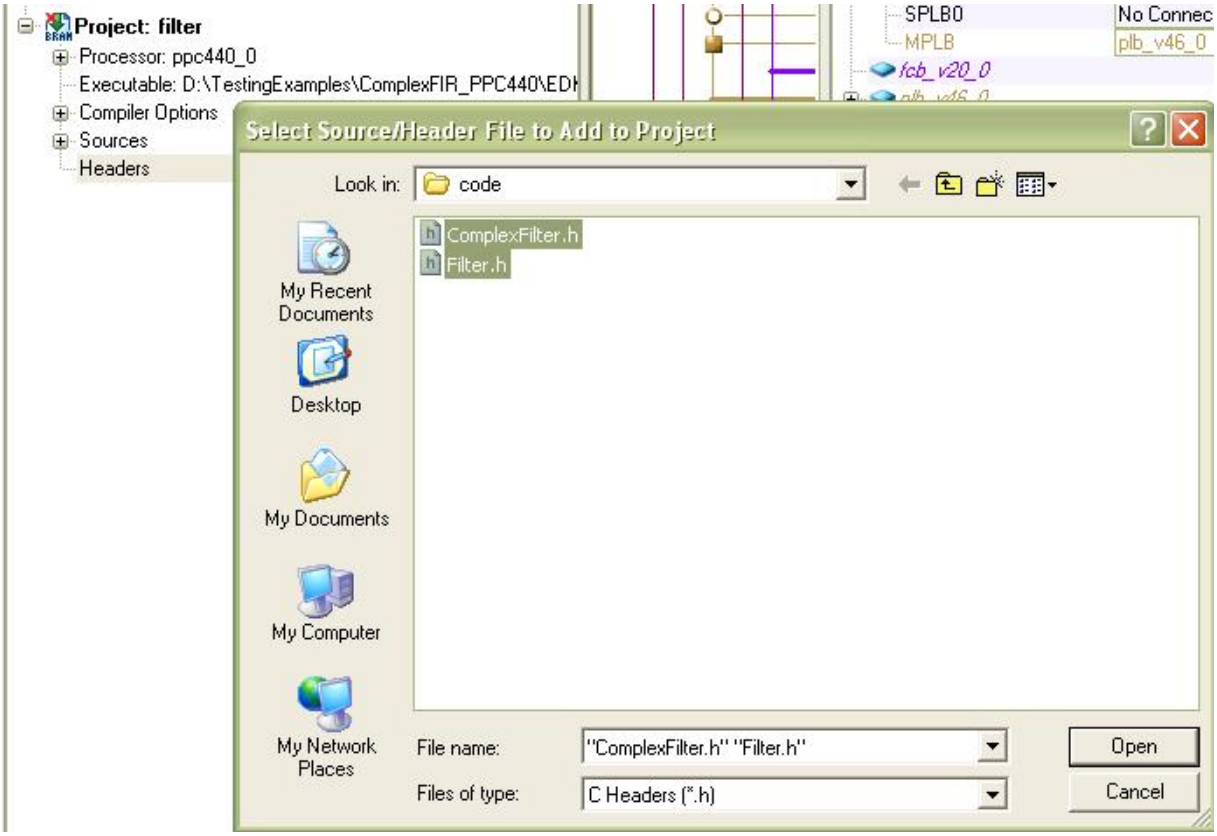
A new **Project: filter** appears in the **Software Projects** list. Double-click the **Sources** item under the **Project: filter** to add source files.

In the **Select Source/Header File to Add to Project** dialog, enter the **code** subdirectory and select all the three C files are shown below:



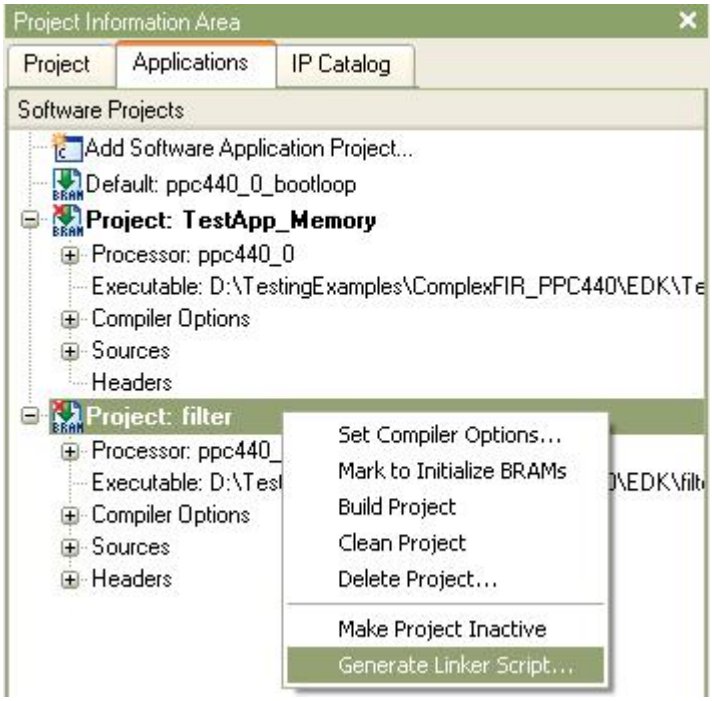
Click **Open** to add the source files shown to your project. These files comprise the software application that will run on the PowerPC CPU.

Next, double-click **Headers** item under the **Project: filter** to add header files. A file selection dialog appears. Select both the header files in the **code** directory shown below.

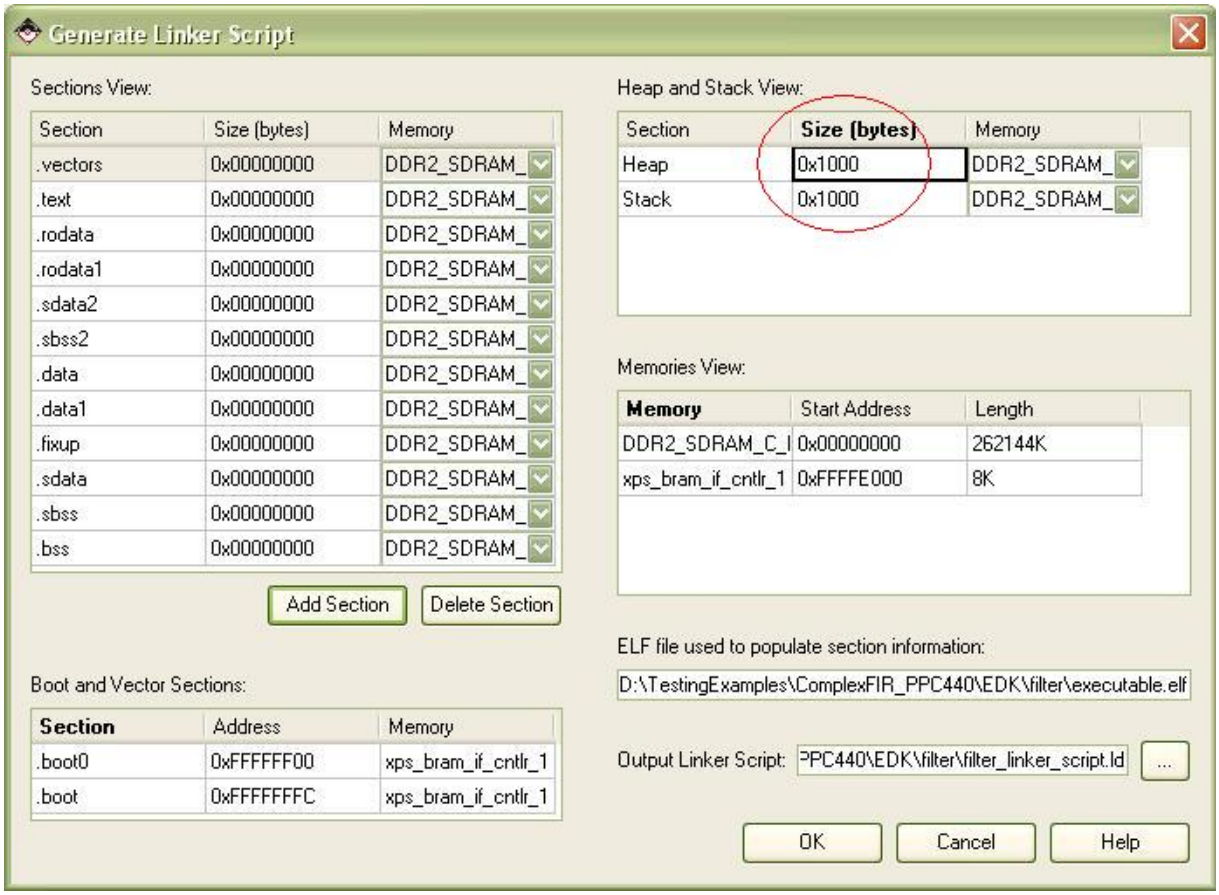


Click **Open** to add the header files to your project.

The on-chip memory is not enough for the project. The instruction and data sessions need to be put into external **DDR2 SDRAM**, and also for the heap and stack. To do this, right-click the **Project: filter**, and select **Generate Linker Script...**

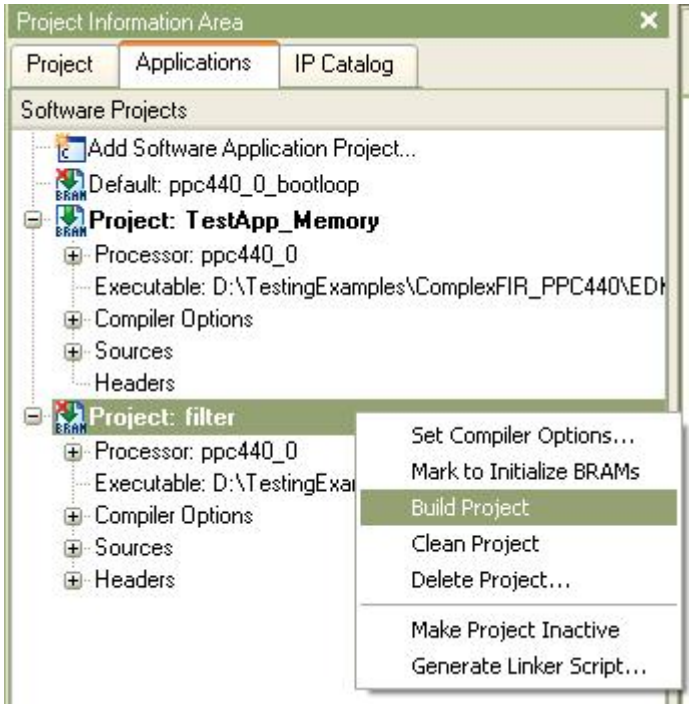


The **Generate Linker Script** dialog will appear. Make sure that all the sections in the **Sections View** are using **DDR2_SDRAM** as memory. For the **Heap and Stack**, enlarge the size to **0x1000 bytes**, and also use **DDR2_SDRAM** as memory.

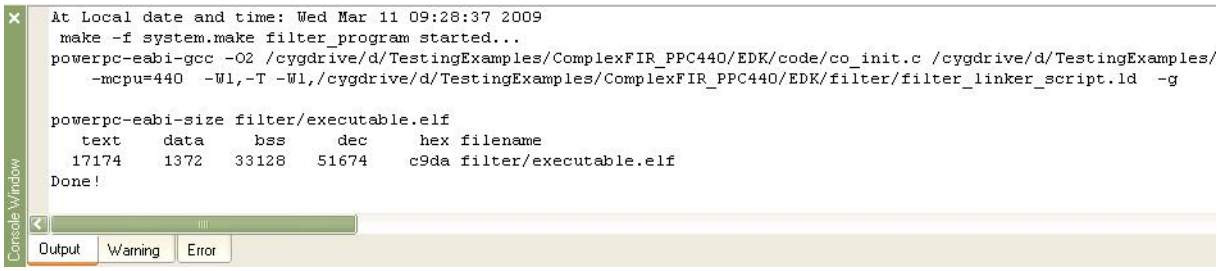


Click **OK** to generate the **Linker Script** file.

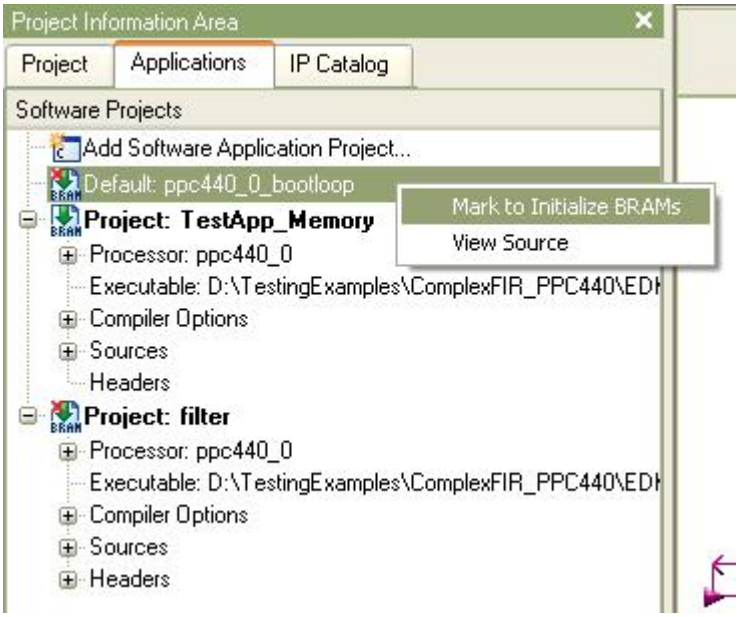
Next, right-click the **Project: filter** and select **Build Project**.



The following messages shown in the **Console Window Output** indicate the software project is built.



Now you will need to change the BRAM initialization application, which is currently the **TestApp_Memory** project. Right-click the **Default: PowerPC_0_bootloop** and select **Mark to Initialize BRAMs**. This will let the bootloop reside in the **BRAMs**.



Next, you will run the application.

See Also

Running the Application

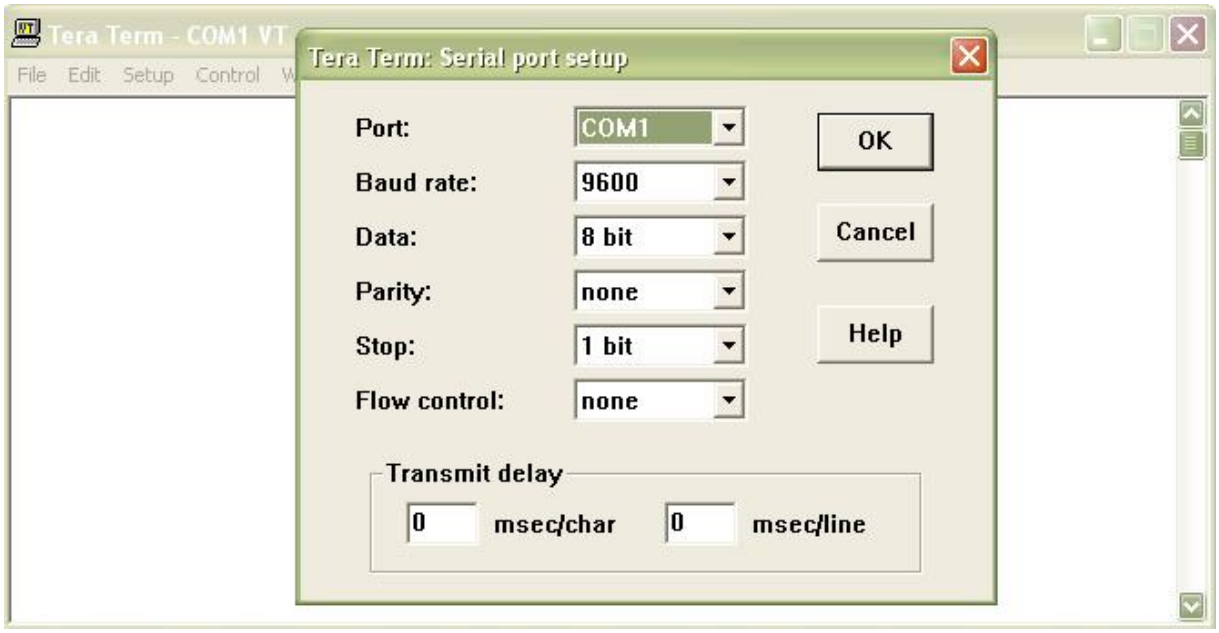
1.1.11 Running the Application

ComplexFIR Filter Tutorial for PowerPC, Step 11

Now let's run the application on the development board.

Connect the serial port of your development machine to that of your development board via a **RS232** cable. Make sure the **JTAG** download cable is connected on the development board. Also ensure that the board is configured to be programmed. Turn on the power to the board.

Open **Tera Term** or **Windows HyperTerminal** application to display the UART output message. Use the same communication settings you chose when defining the **RS232_Uart_1** peripheral in **Base System Builder** (**9600 baud, 8-N-1**). Turn off **flow control**, if available.

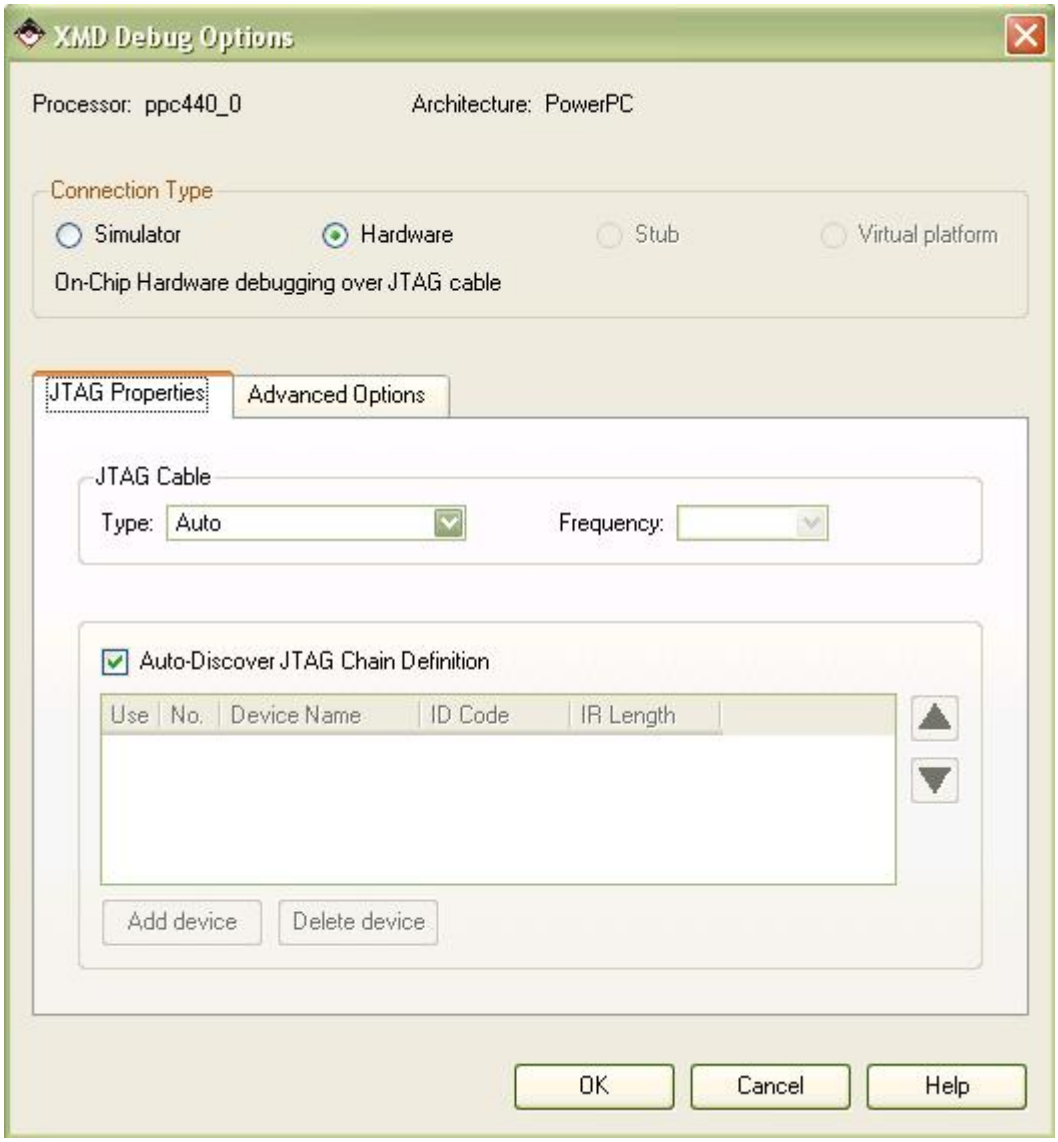


Now, download the bitstream to the device by selecting **Device Configuration** -> **Download Bitstream**.

When downloading is done, the **Console Window Output** will be like this:

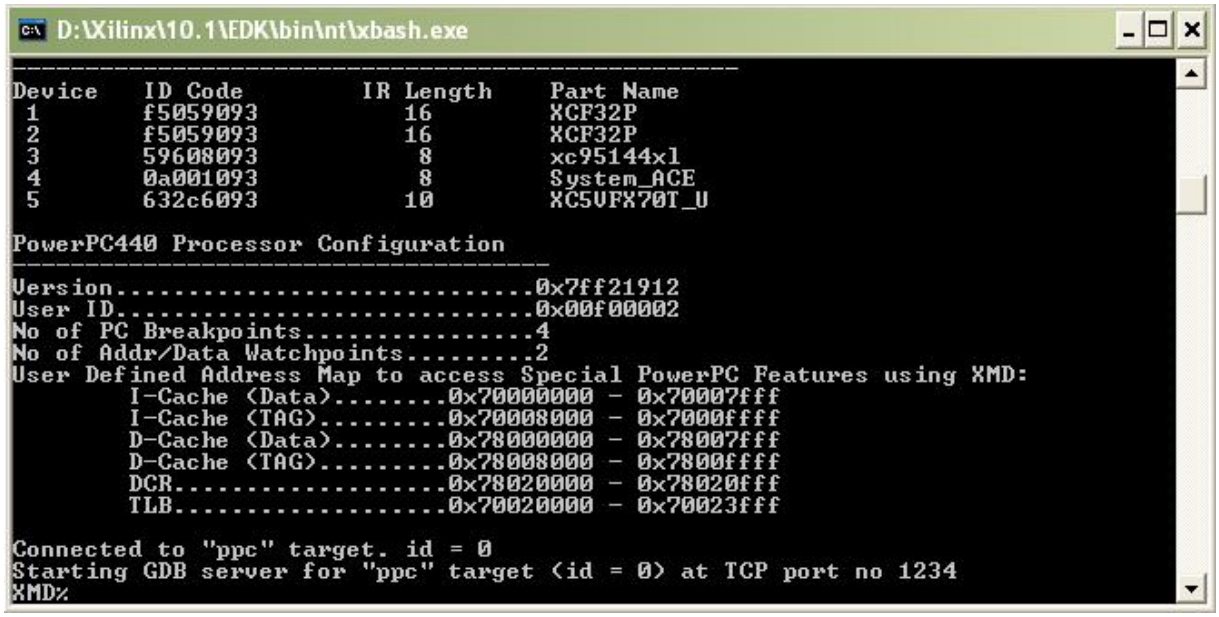
Select from menu **Debug** -> **Launch XMD...**

The **XMD Debug Options** dialog will appear for the first time opening **XMD**.



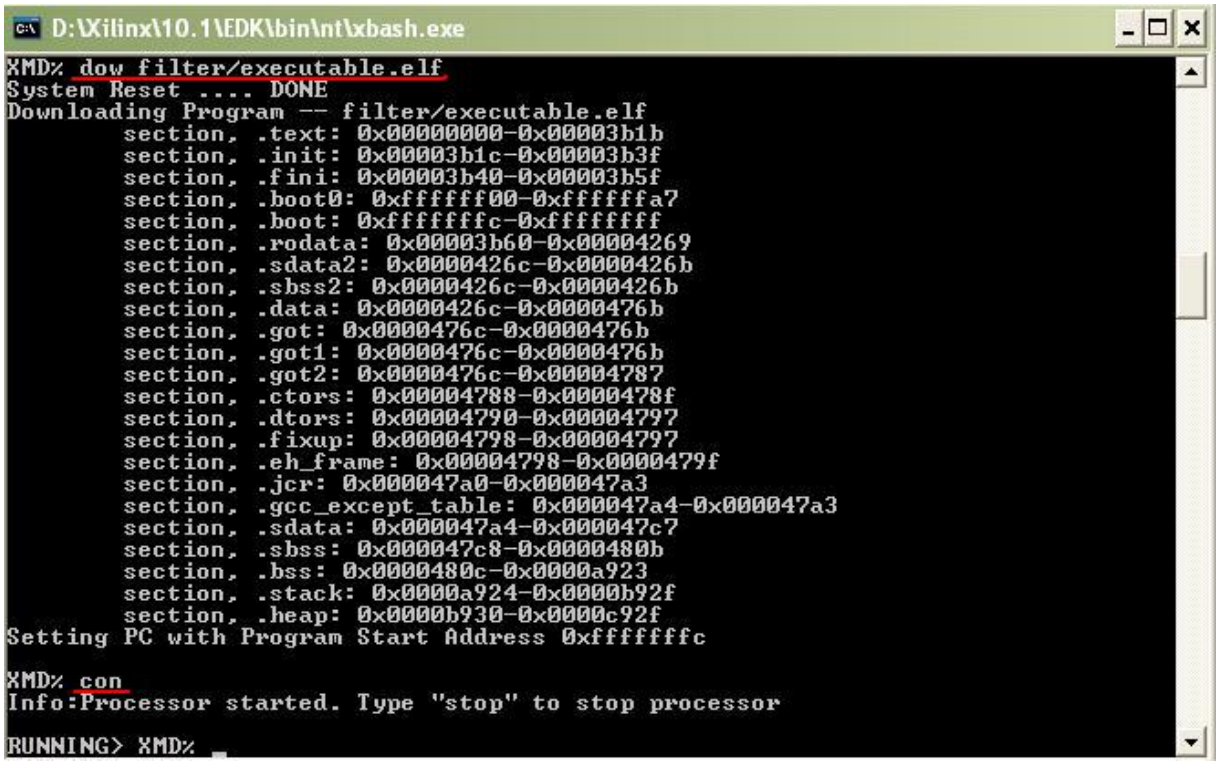
Click **OK** to accept the default settings.

A Cygwin bash shell will come up. It runs a script, connecting to the **PowerPC** processor and the debugger inside the FPGA, as shown below:

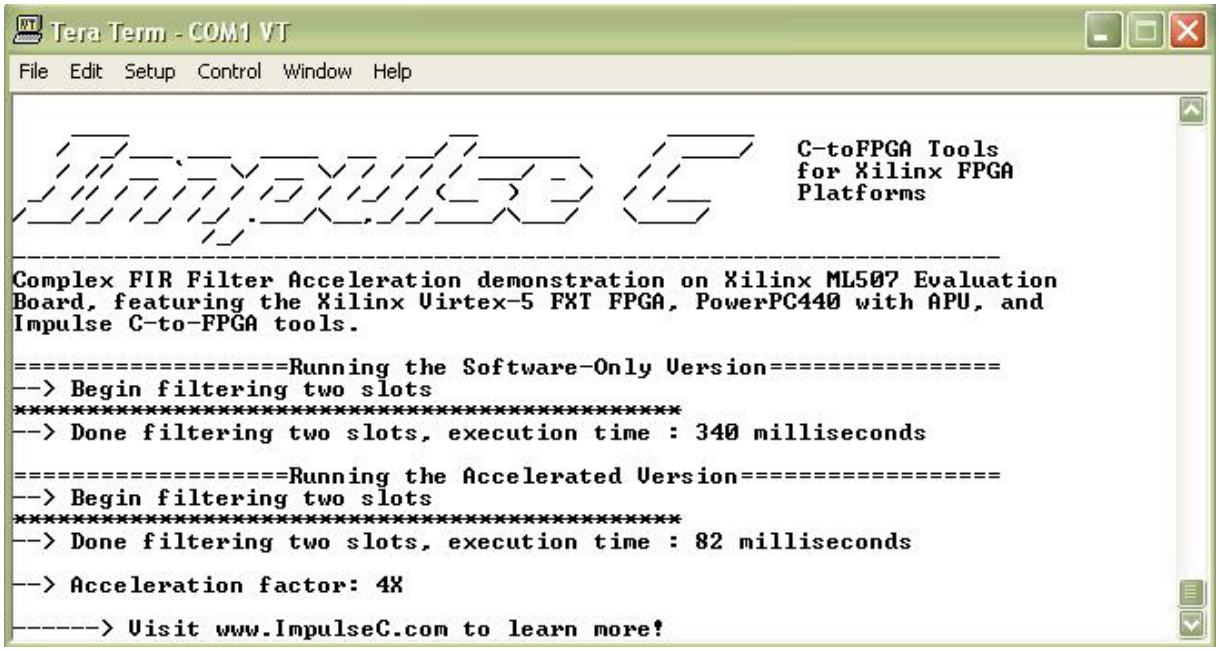


Now, we can download the **filter** project **ELF** file to the target board and run the application with the following **XMD** command:

dow filter/executable.elf
con



Watch Tera Term window again. You should see the messages generated by the software process indicating that the test data has been successfully filtered. The execution with hardware acceleration is 4 times faster than software only running on **PowerPC** microprocessor.



Congratulations! You have successfully completed this tutorial and run the generated hardware on the development board.

See Also

Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)

1.2 Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)

Overview

This tutorial will demonstrate how to create, simulate and build an application targeting the Xilinx Virtex-4 FX platform, including the use of data streams and the shared memory interface. It includes all steps necessary to create a new platform using the Xilinx EDK 10.1 tools.

This tutorial will require approximately one hour to complete, including software run times. To complete the application, you will need access to a Xilinx ML403 development board (or equivalent board equipped with a Xilinx Virtex-4 FX device).

General Steps

This tutorial will take you through the entire process of creating a hardware-accelerated system in the Virtex-4 FX FPGA using the Impulse and Xilinx tools. This is an advanced tutorial with many detailed steps, but can be summarized as the following general steps:

1. Describe and simulate the application using C language and the Impulse CoDeveloper tools.
2. Automatically generate hardware, in the form of VHDL source files, for the hardware accelerator portion of the application.
3. Export the generated files to an EDK project directory.
4. Build a new EDK project describing the PowerPC and all required peripherals.
5. Attach the hardware accelerator generated in step 2 to the PowerPC via the PLB interface.
6. Add all needed software files representing the application to be run on the PowerPC.
7. Run synthesis and place-and-route to generate a downloadable bitmap.
8. Download the application to the ML403 board using a JTAG programming cable.

Detailed Steps

See Also

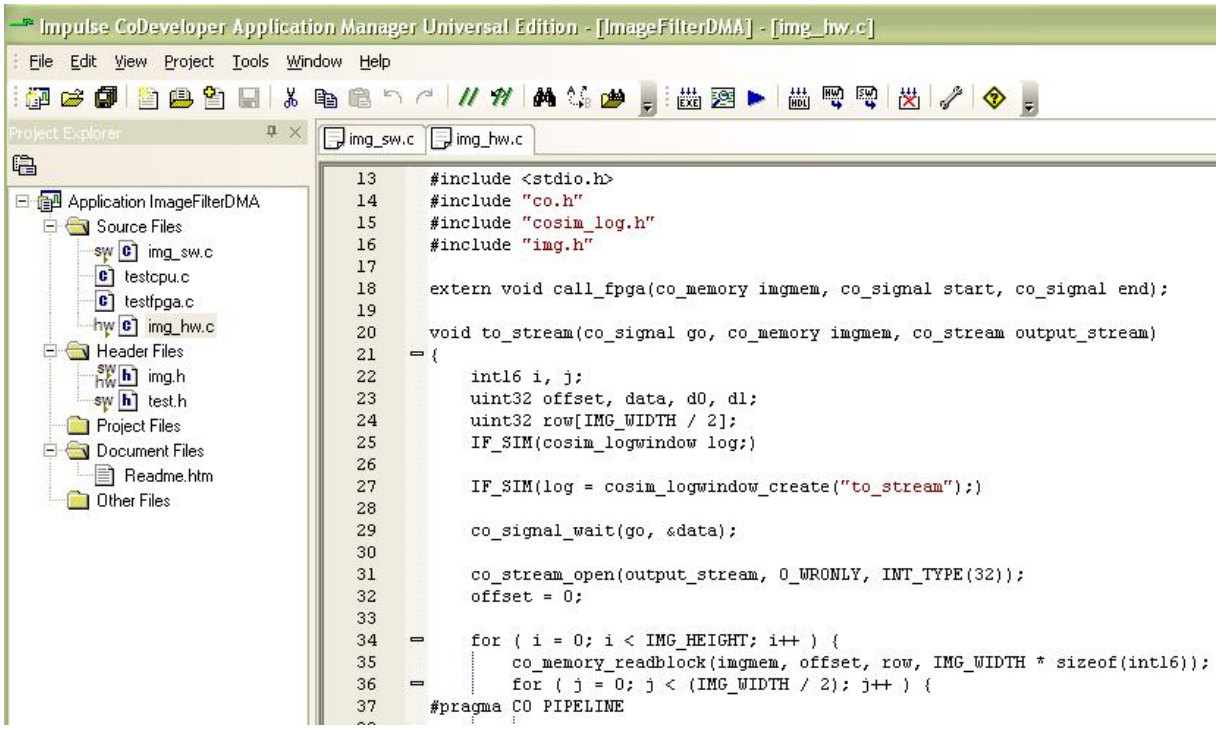
Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)
Tutorial 3: Fractal Image Generation on the Virtex-4 Platform (EDK 10.1)

1.2.1 Loading the Sample Application

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 1

To begin, start the CoDeveloper Application Manager by selecting Application Manager from the Start -> Programs -> Impulse Accelerated Technologies -> CoDeveloper program group.

Open the Xilinx Virtex-4 FX Mandelbrot sample project by selecting Open Project from the File menu, or by clicking the Open Project toolbar button. Navigate to the .\Examples\Xilinx\Virtex4\Mandelbrot\ directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) Opening the project will result in the display of a window similar to the following:



Files included in the Mandelbrot project include:

Source file img_hw.c - This source file includes the Image Filter DMA process, and also includes the application's configuration function.

Source file img_sw.c - This source file includes the test application that runs on the target PowerPC processor. The test application includes a **main()** function, and a call_fpga function to access the external DDR SDRAM. As written, this test application can be compiled either on the PowerPC processor or as a desktop simulation executable.

Source file img.h - This header file defines the size of the test image.

Source file test.h - This header file include a test image.

See Also

Understanding the ImageFilterDMA Application

1.2.2 Understanding the Image Filter DMA Application

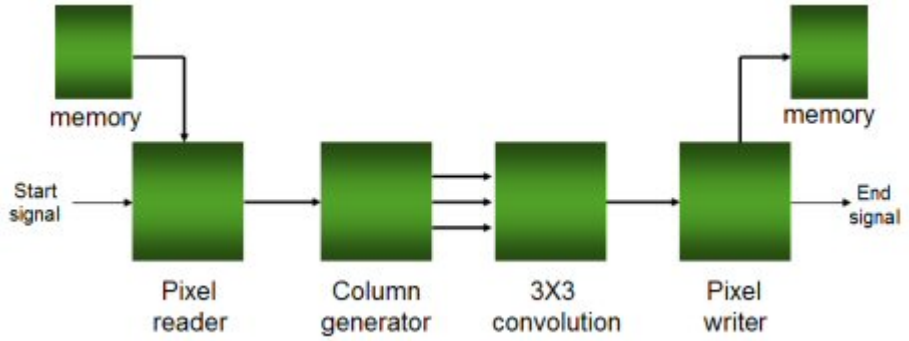
Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 2

This tutorial example demonstrates a number of important concepts of Impulse C programming for Xilinx Virtex-4 FPGA platforms. The most important of these concepts is the use of shared memory. In platforms based on the PowerPC processor, it is often most efficient to move large blocks of data between software and hardware elements of the system using direct memory access (DMA)

techniques, rather than making use of streams. Which method you use (memories or streams) will depend on the nature of the application, so you may wish to try both methods and compare relative performance.

In this example we will create a simple image filter, which operates on incoming image data (pixel values) to generate a converted image. The specific image processing algorithm that we have chosen for this example is an image convolution algorithm, which is a critical step in many image processing algorithms and is representative of other such image processing filters.

The specific convolution performed in this test case is an edge-detection function, in which a 3x3 pixel "window" is assembled and processed for each pixel in the source image. The algorithm is represented by two pipelined hardware processes that decompose the image data into three rows of image data and process those rows to calculate a resulting value from a 3x3 pixel window. Two additional hardware processes are used to read and write image data from shared memory and present the data to the image processing algorithm as a stream.



These four processes and the corresponding stream, memory and signal declarations are described using Impulse C and interconnected using the configuration function shown below:

```
void config_img(void *arg)
{
  int error;
  co_signal startsig, donesig;
  co_memory shrmem;
  co_stream istream, row0, row1, row2, ostream;
  co_process reader, writer;
  co_process cpu_proc, prep_proc, filter_proc;

  startsig = co_signal_create("start");
  donesig = co_signal_create("done");
  shrmem = co_memory_create("image", "heap0", IMG_WIDTH * IMG_HEIGHT *
  sizeof(uint16));
  istream = co_stream_create("istream", INT_TYPE(32), IMG_HEIGHT/2);
  row0 = co_stream_create("row0", INT_TYPE(32), 4);
  row1 = co_stream_create("row1", INT_TYPE(32), 4);
  row2 = co_stream_create("row2", INT_TYPE(32), 4);
  ostream = co_stream_create("ostream", INT_TYPE(32), IMG_HEIGHT/2);

  cpu_proc = co_process_create("cpu_proc", (co_function)call_fpga, 3, shrmem,
  startsig, donesig);
  reader = co_process_create("reader", (co_function)to_stream, 3, startsig,
  shrmem, istream);
  prep_proc = co_process_create("prep_proc", (co_function)prep_run, 4, istream,
  row0, row1, row2);
  filter_proc = co_process_create("filter", (co_function)filter_run, 4, row0,
  row1, row2, ostream);
  writer = co_process_create("writer", (co_function)from_stream, 3, ostream,
```

```
    shrmem, donesig);

    co_process_config(reader, co_loc, "PE0");
    co_process_config(prep_proc, co_loc, "PE0");
    co_process_config(filter_proc, co_loc, "PE0");
    co_process_config(writer, co_loc, "PE0");

    IF_SIM(error = cosim_logwindow_init();)
  }
```

Note that a fifth process (call_fpga) is included that represents the controlling software application that will run on the embedded PowerPC processor.

See Also

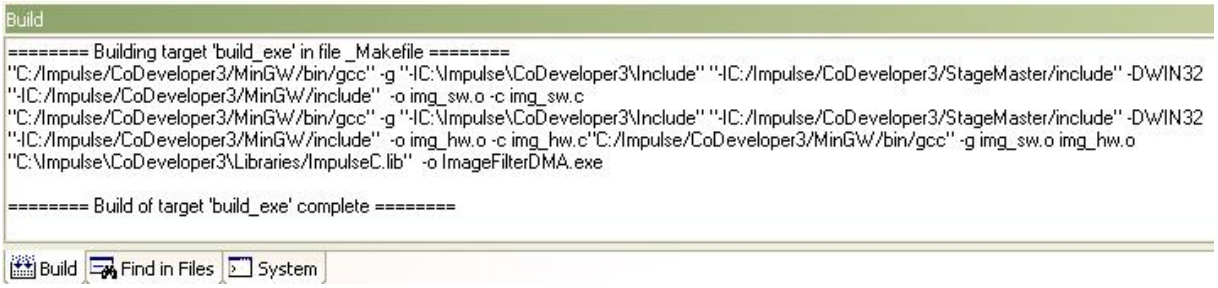
Compiling the Application for Simulation

1.2.3 Compiling the Application for Simulation

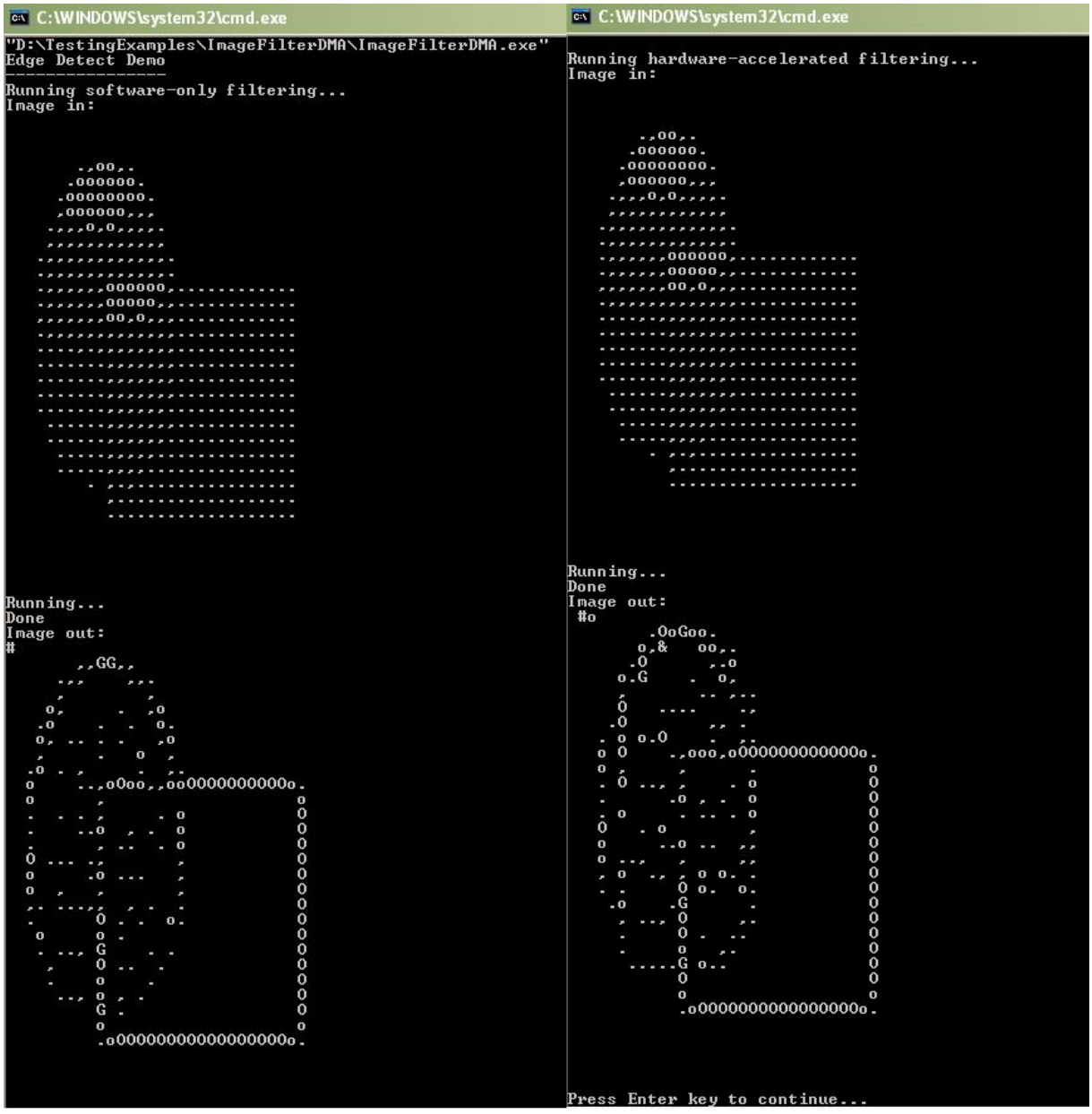
Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 3

The software test bench provided with this example (in **img_sw.c**) has been written in such a way that it can be compiled either to an FPGA as hardware (using fixed point math operations) or be compiled for desktop simulation, using either fixed or floating point math operations. This makes it possible to compile and simulate the application for the purpose of functional verification.

Select Project -> Build Simulation Executable (or click the Build Simulation Executable button) to build the Mand.exe executable. The CoDeveloper transcript window will display the compile and link messages as shown below:



You now have a Windows executable representing the application implemented as a desktop (console) software application. You can run this executable by selecting Project -> Launch Simulation Executable. A command window will open and the simulation executable will run as shown below:



The left-hand side is the output from software only filtering, and the right-hand side is from the hardware accelerated filtering.

See Also

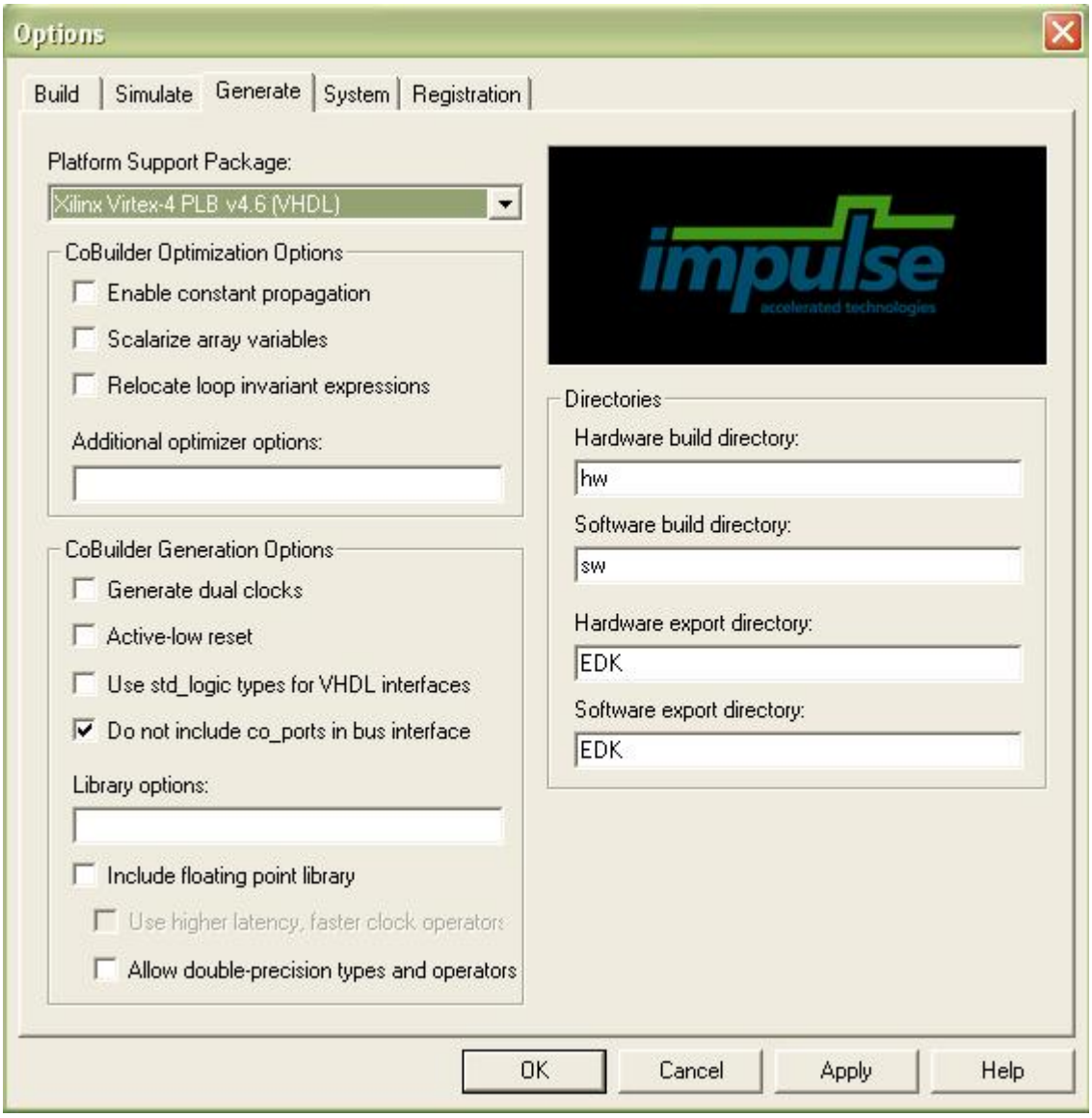
Building the Application for Hardware

1.2.4 Building the Application for Hardware

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 4

Specifying the Platform Support Package

To specify a platform target, open the Generate Options dialog as shown below:



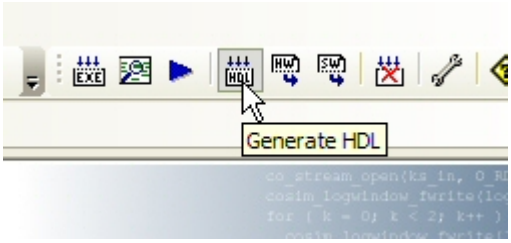
Specify *Xilinx Virtex-4 PLB v4.6* as shown. Also specify "hw" and "sw" for the hardware and software directories as shown, and specify "EDK" for the hardware and software export directories. ("EDK" is the directory in which you will be creating a Xilinx Platform Studio project.)

Click OK to save the options and exit the dialog.

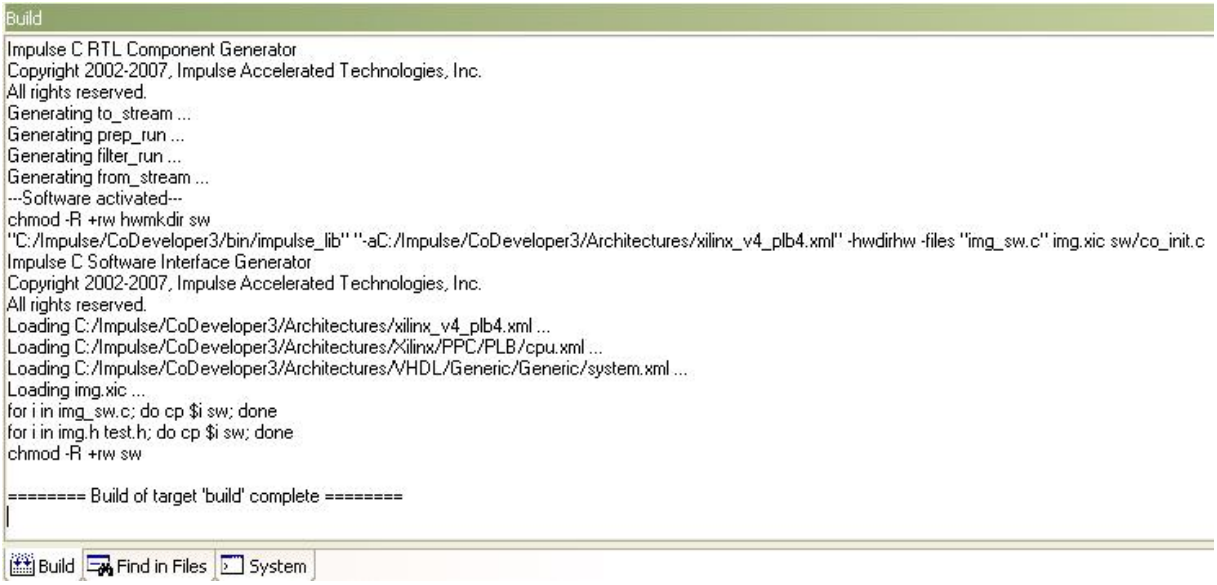
Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces

and library files, select Generate HDL from the Project menu, or click the Generate HDL button as shown:



A series of processing steps will run in a command window as shown below:



Note: the processing of this example may require a minute or more to complete, depending on the performance of your system.

When processing has completed you will have a number of resulting files created in the **hw** and **sw** subdirectories of your project directory. These files are ready to be exported into a Xilinx Platform Studio project directory.

See Also

Exporting the Hardware and Software Files

1.2.5 Exporting the Hardware and Software Files

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 5

Recall that in the previous step you specified the directory "EDK" as the export target for hardware and software. These export directories specify where the generated hardware and software processes are to be copied when the Export Software and Export Hardware features of CoDeveloper are invoked. Within these target directories (in this case "EDK"), the specific destination for each file previously generated is determined from the Platform Support Package architecture library files. It is therefore important that the correct Platform Support Package (in this case Xilinx Virtex-4 APU) is selected prior to starting the export process.

To export the files from the build directories (in this case "hw" and "sw") to the export directories (in this case the "EDK" directory), select Project -> Export Generated Hardware (HDL) and Project -> Export Generated Software, or select the Export Generated Hardware and Export Generated Software buttons from the toolbar.

Export the Hardware Files

Export the Software Files

Note: you must select BOTH Export Software and Export Hardware before going onto the next step.

You have now exported all necessary files from CoDeveloper for use in the Xilinx tools environment. By opening a Windows Explorer window, you can see how the hardware and software files have been copied into subdirectories of your EDK directory. In particular, notice that CoDeveloper has created a "pcores/plb_img_arch_v1_00_a" directory containing the generated HDL and other related files. This generated directory structure will allow you to import the generated core directly into the Platform Studio tools.

See Also

Creating the ML403 Test Platform

1.2.6 Creating the ML403 Test Platform

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 6

At this point you have:

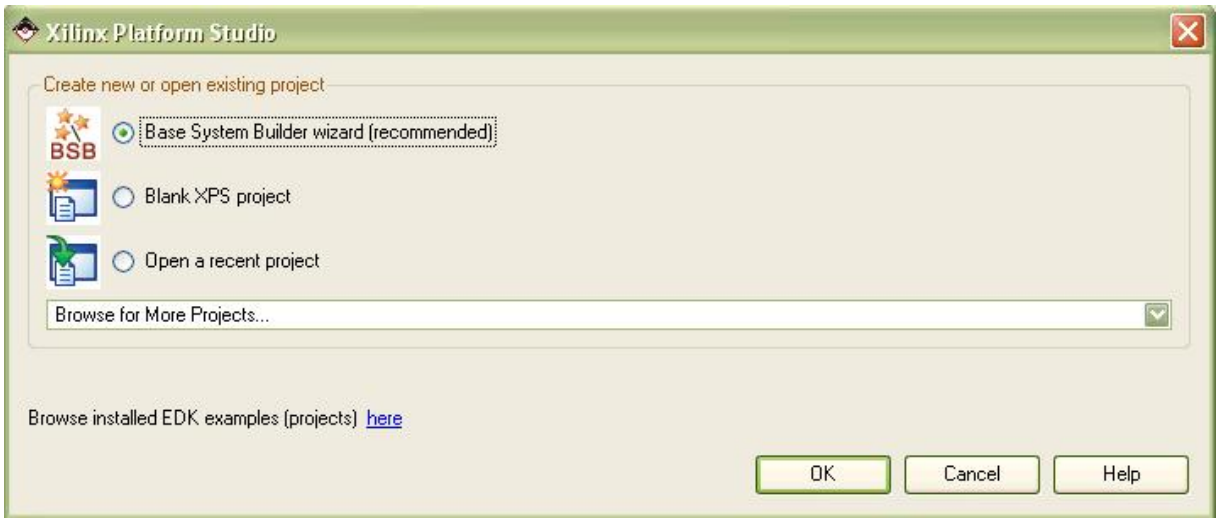
- Created hardware for the Mandelbrot accelerator.
- Exported the generated hardware to the EDK subdirectory as a pcore.
- Exported the PowerPC software application files to the EDK subirectory.

In this tutorial section, you will be making use of the Platform Studio tools, including the Base System Builder Wizard, to define and build a new PowerPC-based platform targeting the Xilinx ML403 development board. You will first create a test platform allowing you to download and verify your PowerPC and its standard peripherals. After successfully creating and testing the basic platform, you will add the necessary hardware and software files to build, download and test the Mandelbrot sample application.

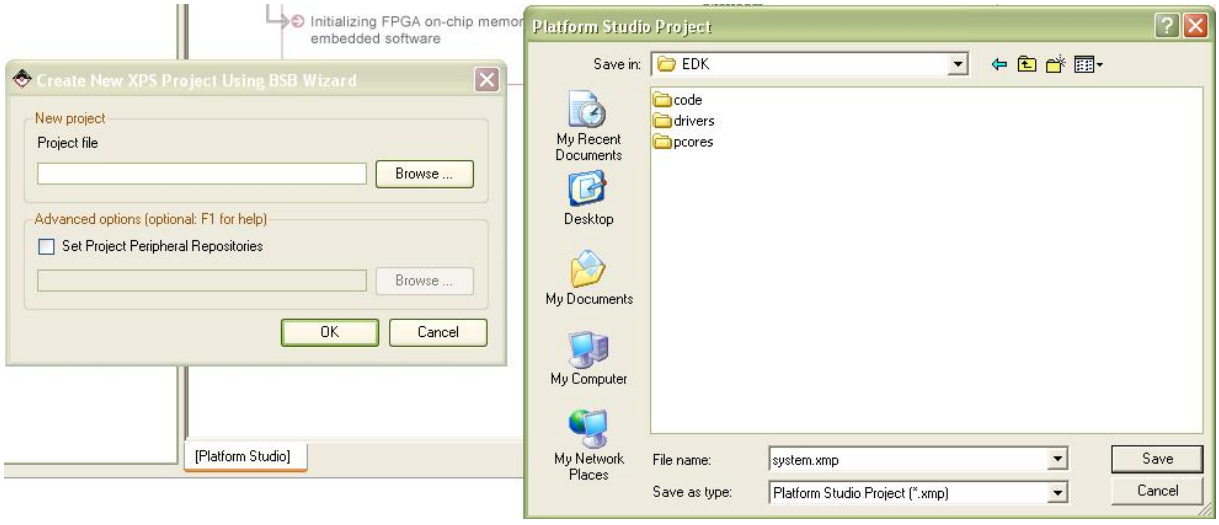
Note: If you are using a different Virtex-4 FPGA development board, you will need to obtain an associated .XBD file from your board vendor, as described in the introduction to this tutorial.

Using Base System Builder to Create the Platform

To begin, start the Xilinx Platform Studio tools and select the Base System Builder Wizard as shown below:



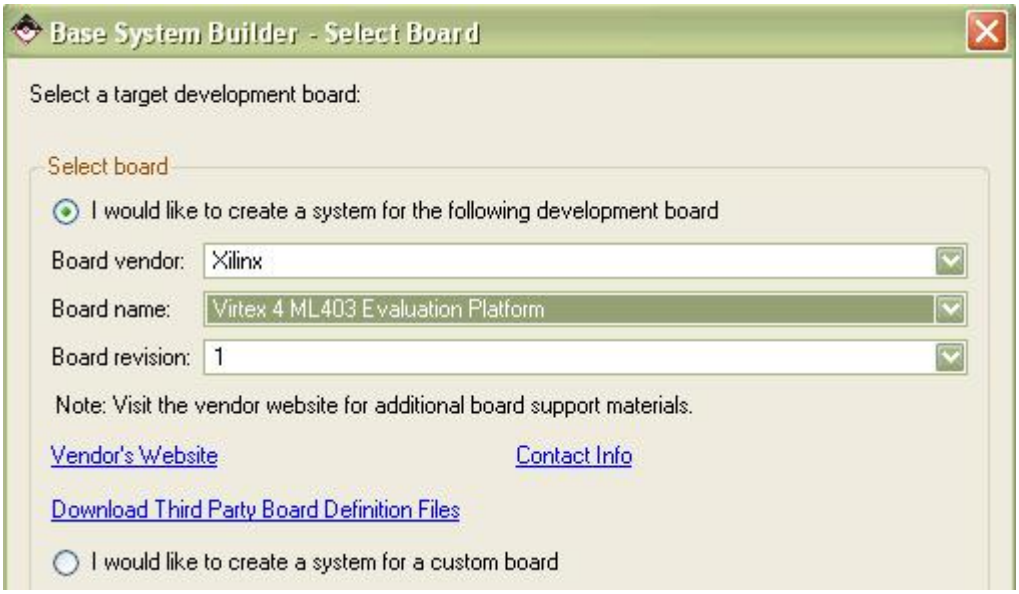
Click the OK button to proceed. When asked for a project name and location, specify the EDK subdirectory of your project, and accept the default project name (system.xmp) as shown below:



Press the OK button to continue.

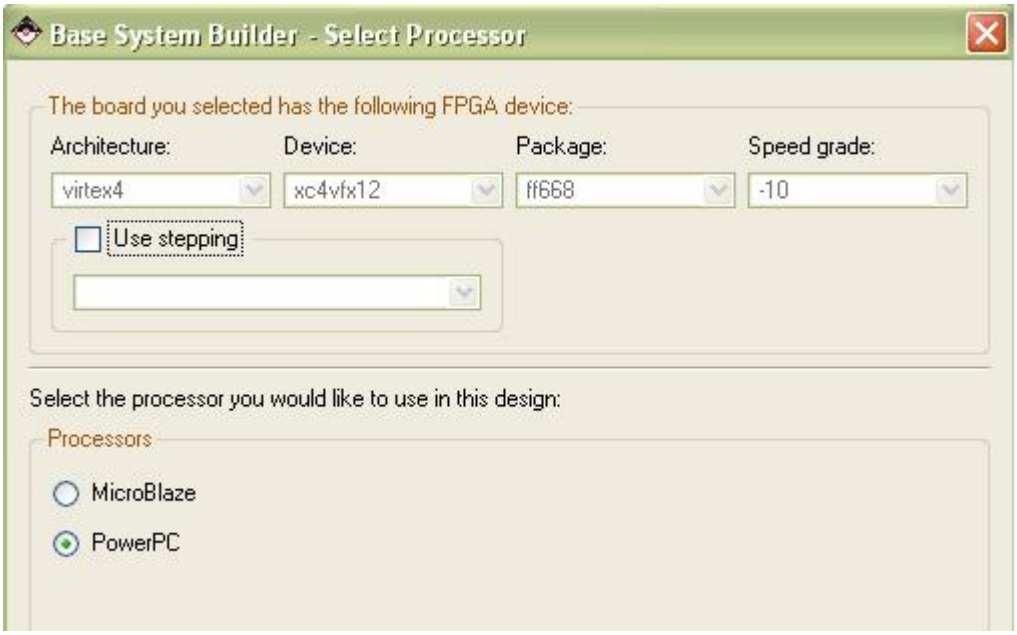
You will now be presented with the Base System Builder Wizard. Select the "I would like to create a new design" option, then click Next to continue:

Next, select your target board using the "Board vendor" and "Board name" drop-down lists. To use the Xilinx ML403 board with attached LCD display, choose the "Virtex 4 ML403"" board as shown:



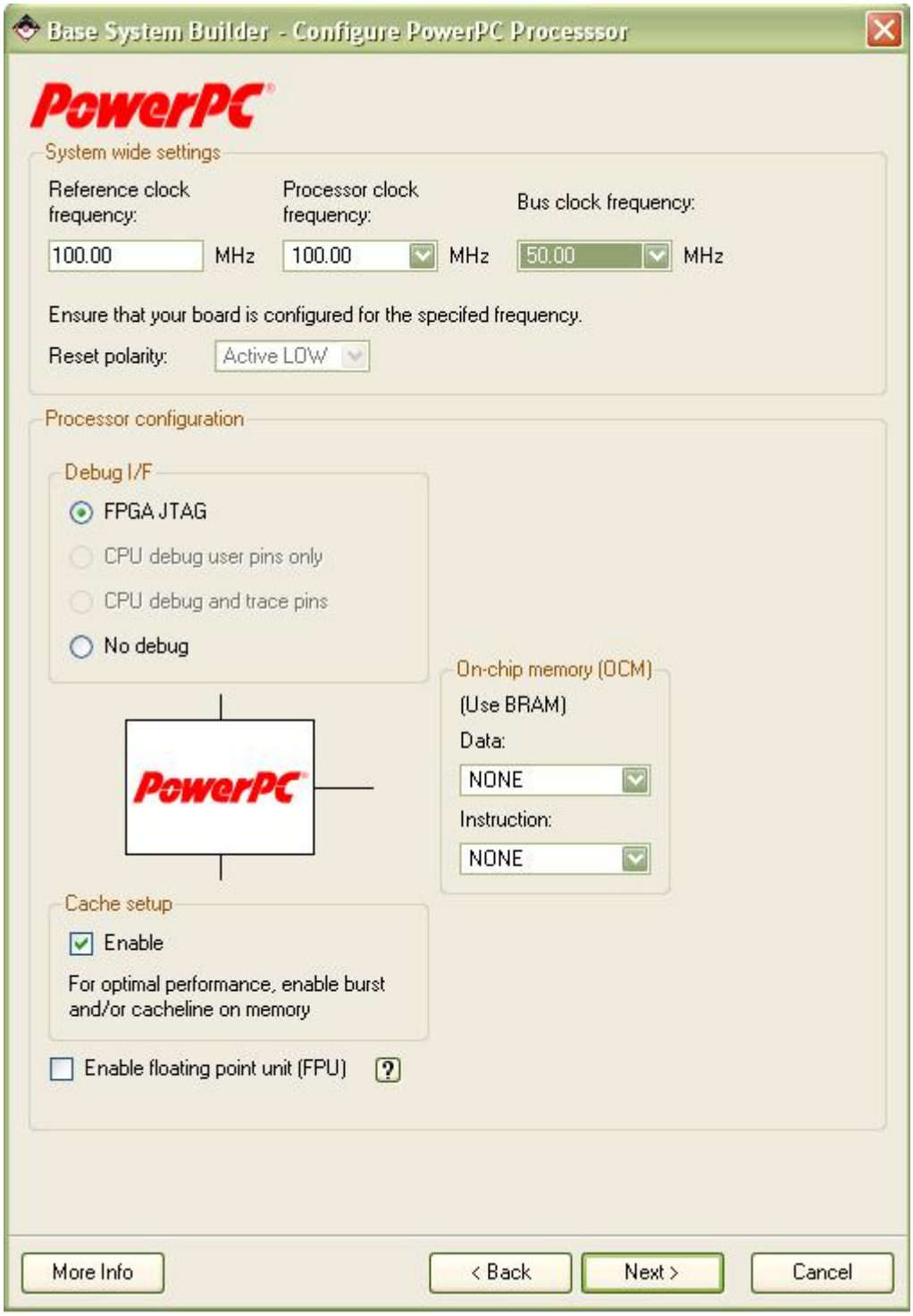
Click the Next button to proceed to the next Wizard page.

On the Select Processor page, be sure PowerPC is selected as the target processor, then click Next:



On the Configure PowerPC page, specify the following options:

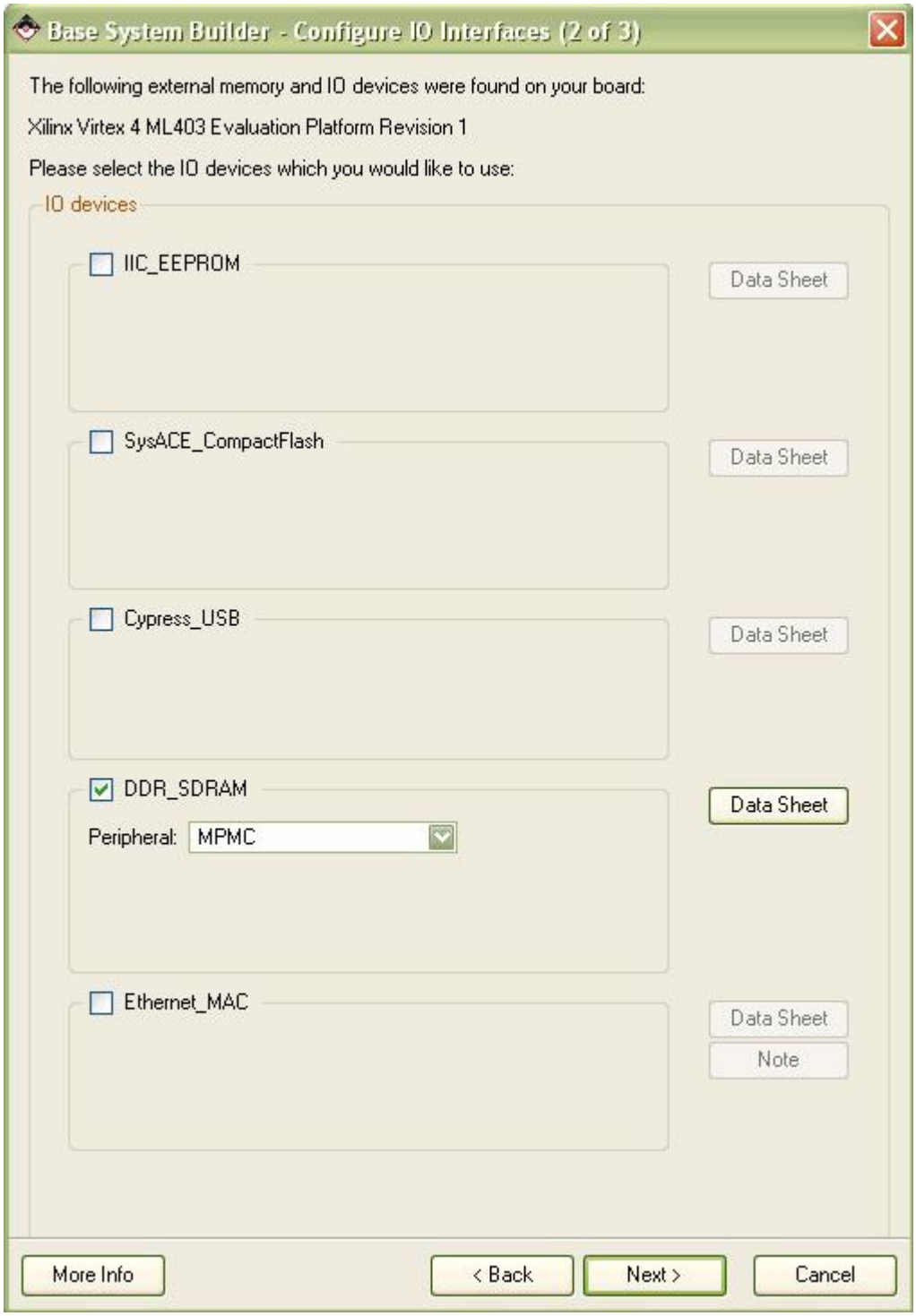
Processor clock frequency: 100 MHz
Bus clock frequency: 50 MHz
Debug I/O: JTAG
Cache setup: Enable
On-chip memory (OCM): NONE



Click Next to continue. You will now be presented with a series of pages for configuring various I/O interfaces. Select the RS232_Uart and LEDs_4Bit peripherals as shown, but do not select the LEDs_Positions and the Push_Button_Position peripheral:

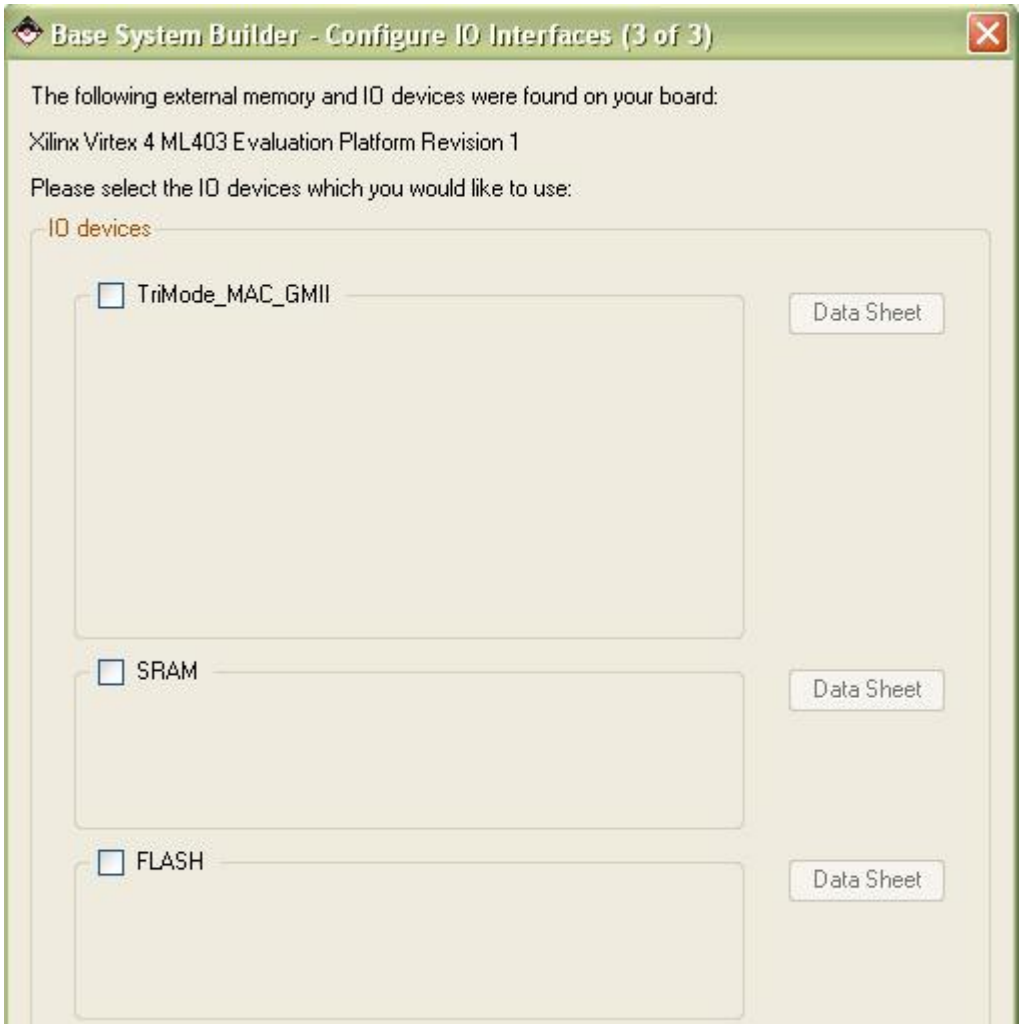
Click Next.

On the next Wizard page, select only the DDR_SDRAM peripheral:



Click Next.

On the page that follows, do not select any of the peripherals:

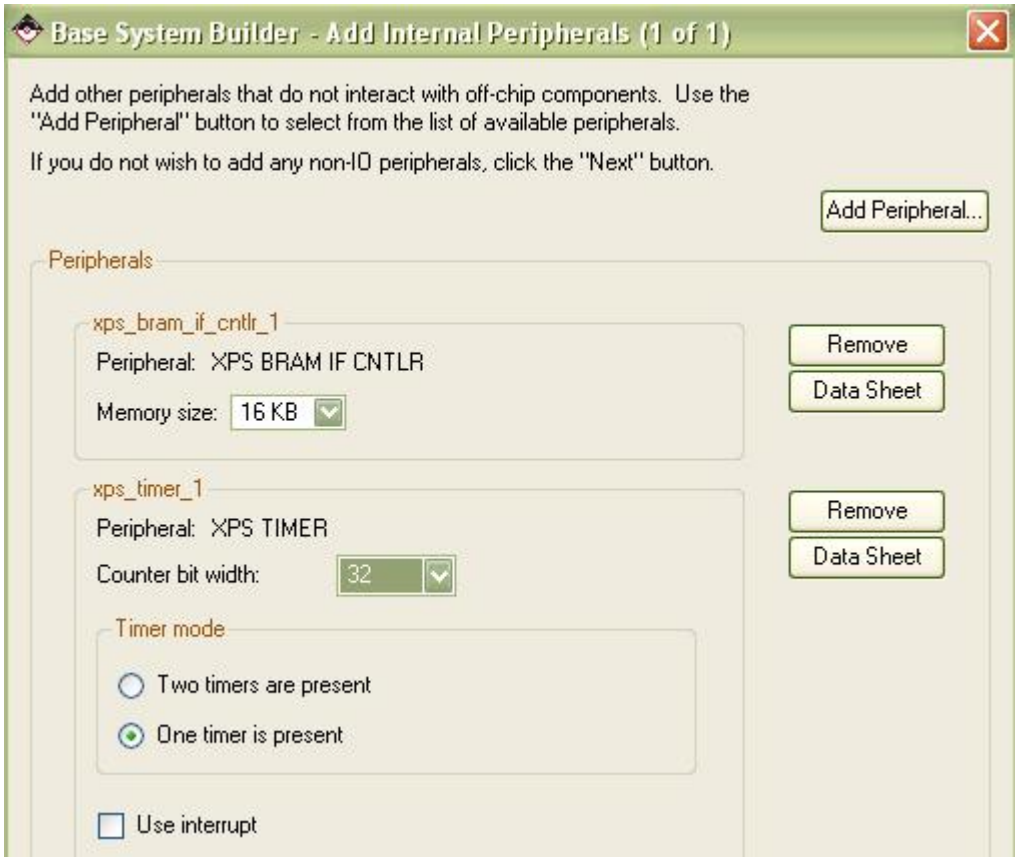


Click Next.

On the Add Internal Peripherals page, change the memory size of the plb_bram_if_cntlr_1 to 16 KB as shown:

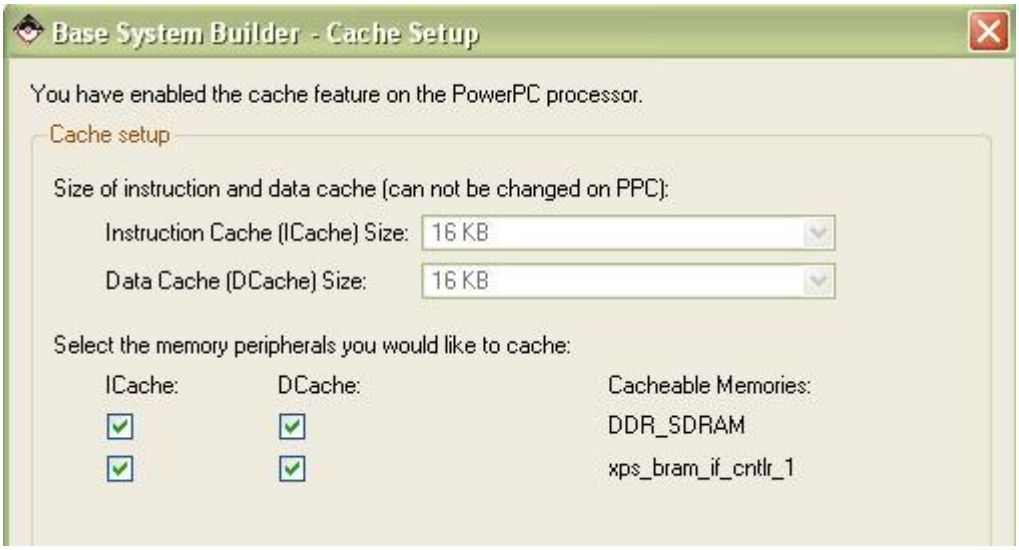
Click Add Peripheral button to open the Add Peripheral dialogue. Choose XPS TIMER from the peripheral list as shown below:

Click OK.The xps_timer_1 is added to the peripheral list. Configure the timer as the following options:
Counter bit width: 32
Timer mode: One timer is present
Use interrupt: no



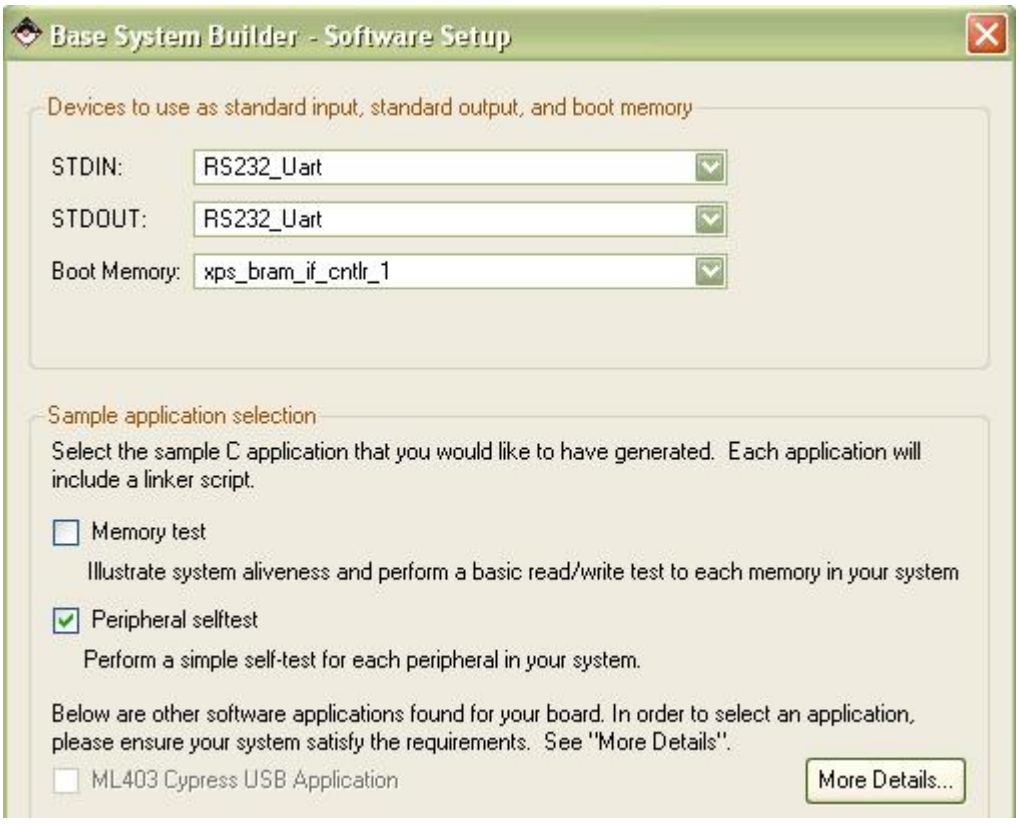
Click Next to continue.

On the Cache Setup page, enable both cache selections as shown:



Click Next.

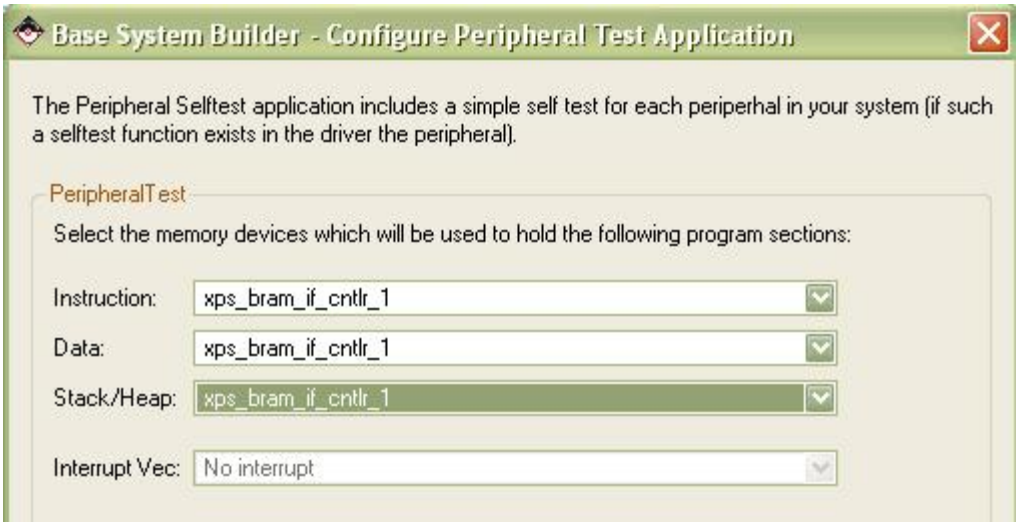
The Wizard will now ask if you want to create memory and peripheral test applications. Select the "Peripheral selftest" application, but do not select the "Memory test" application:



Click Next.

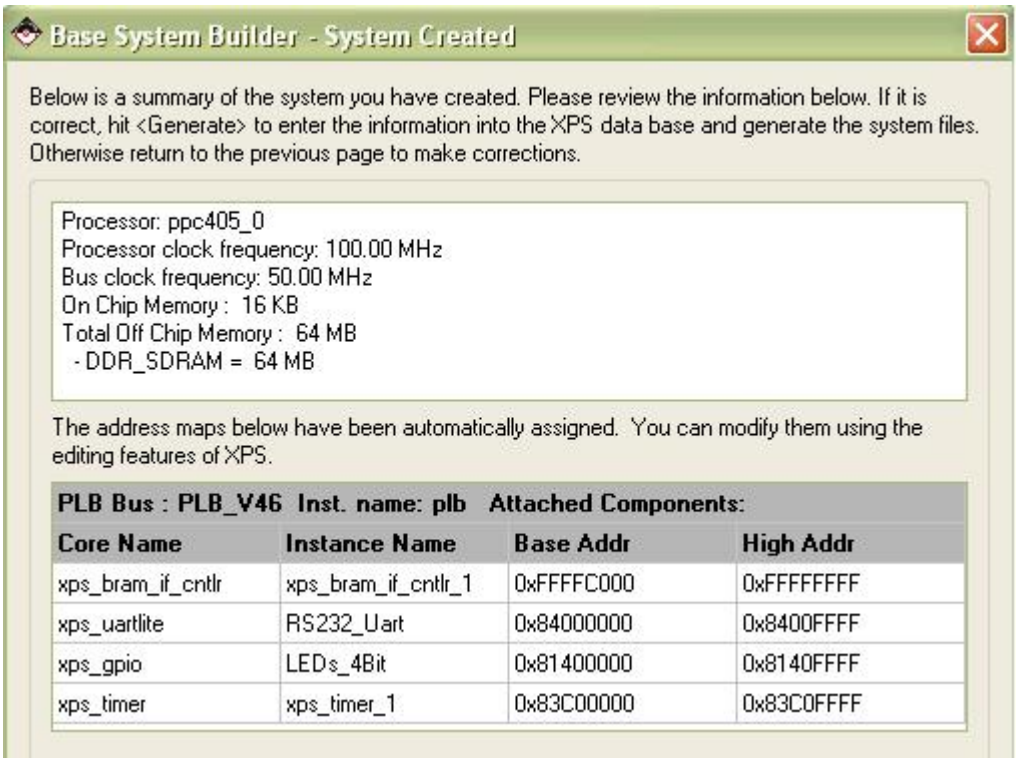
You will now be prompted for memory locations for Instruction, Data and Stack/Heap for the

PeripheralTest application. Select xps_bram_if_cntlr_1 for the Instruction field, the Data and Stack/Heap fields as shown below:

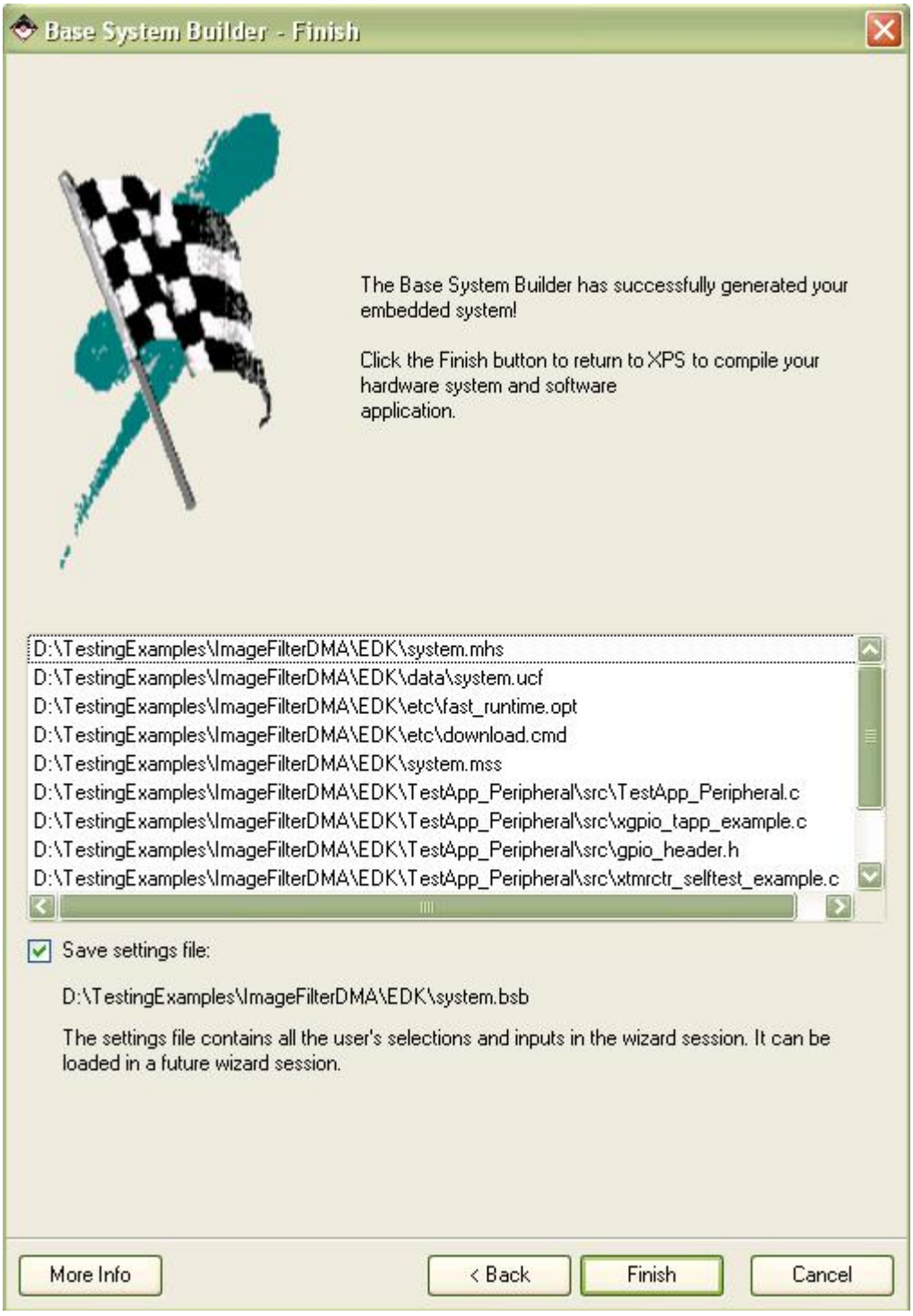


Click Next.

The Wizard will now display a summary of your platform selections:

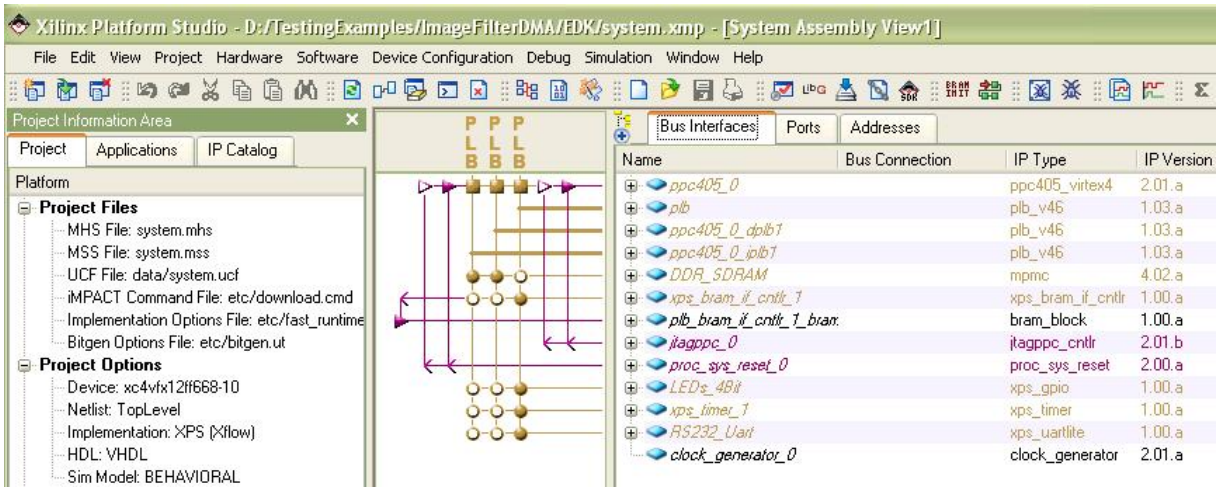


Click the Generate button to generate the platform with the specified configurations. After the platform has been generated, the Wizard will display a final page, and will give you the option of saving the platform settings to a .BSB file. This file can be used when creating new platforms with similar settings.



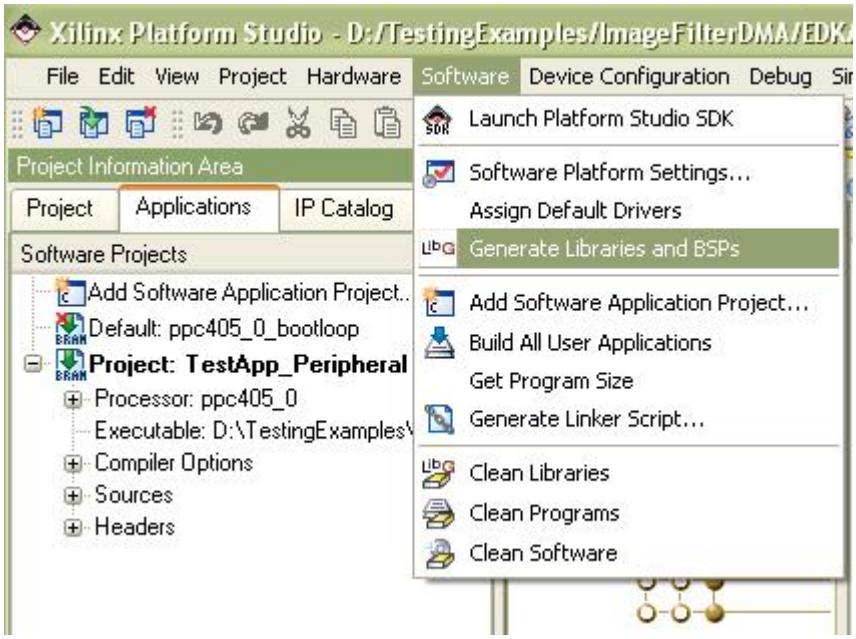
Click Finish to exit the Wizard.

The Platform Studio interface will now appear similar to the following:

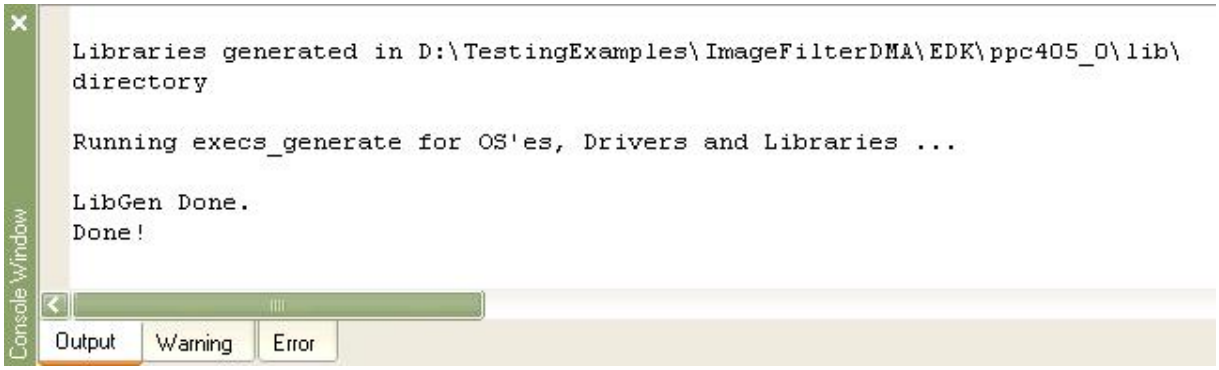


Building and Running the Peripheral Test

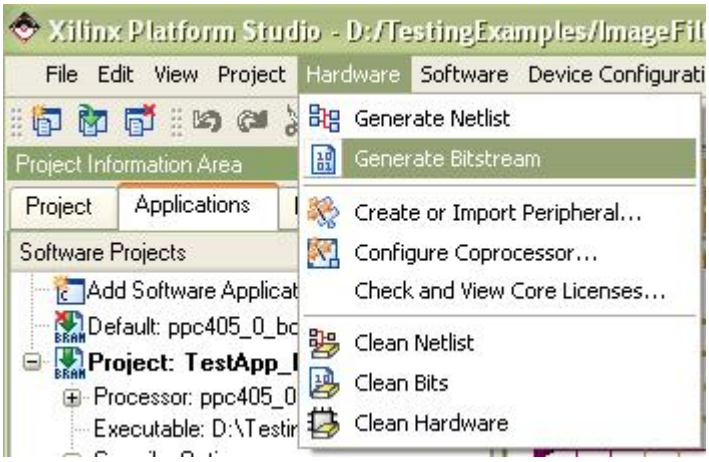
Before creating and building the ImageFilterDMA sample application, it is a good idea to do a quick test of the platform, using the Peripheral Selftest test application created by Base System Builder. To build the test application, you must first generate the PowerPC libraries, peripheral drivers, and other files needed for the software portion of the application. To do this, select the Generate Libraries and BSPs command from the Software menu as shown below:



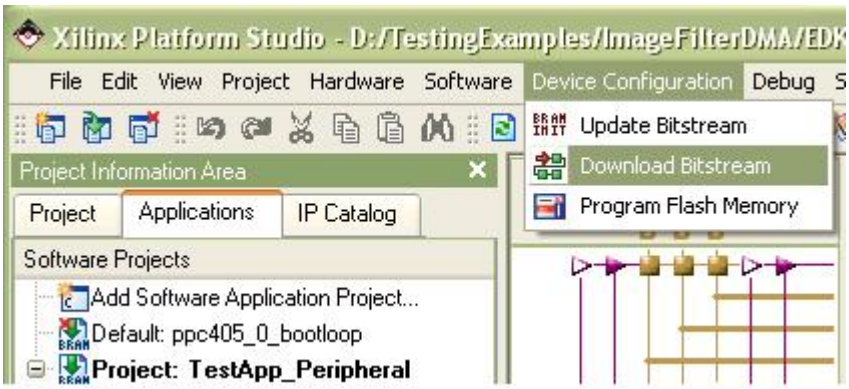
When the libraries have been built, Platform Studio will display a message similar to the following:



Next, select the Generate Bitstream command from the Hardware menu as shown below. This command starts the synthesis and place-and-route process, resulting in a downloadable .BIT file. This will take a few minutes, depending on the speed of your computer.



After the bitstream generation has completed, make sure your JTAG cable is plugged in properly and the ML403 board is powered up. Select Download Bitstream from the Device Configuration menu as shown below:



When the FPGA has been successfully programmed, you will see a "Programming Complete" message in the Platform Studio transcript, and you will see a small row of LEDs located light up in sequence on the lower right corner of board.

You have now verified the complete design flow and all needed hardware connections, from Platform Studio and Base System Builder to the ML403 board. In the next tutorial section, you will replace this test application with a new application representing the Image Filter DMA.

See Also

Adding the ImageFilterDMA Hardware

1.2.7 Adding the ImageFilterDMA Hardware

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 7

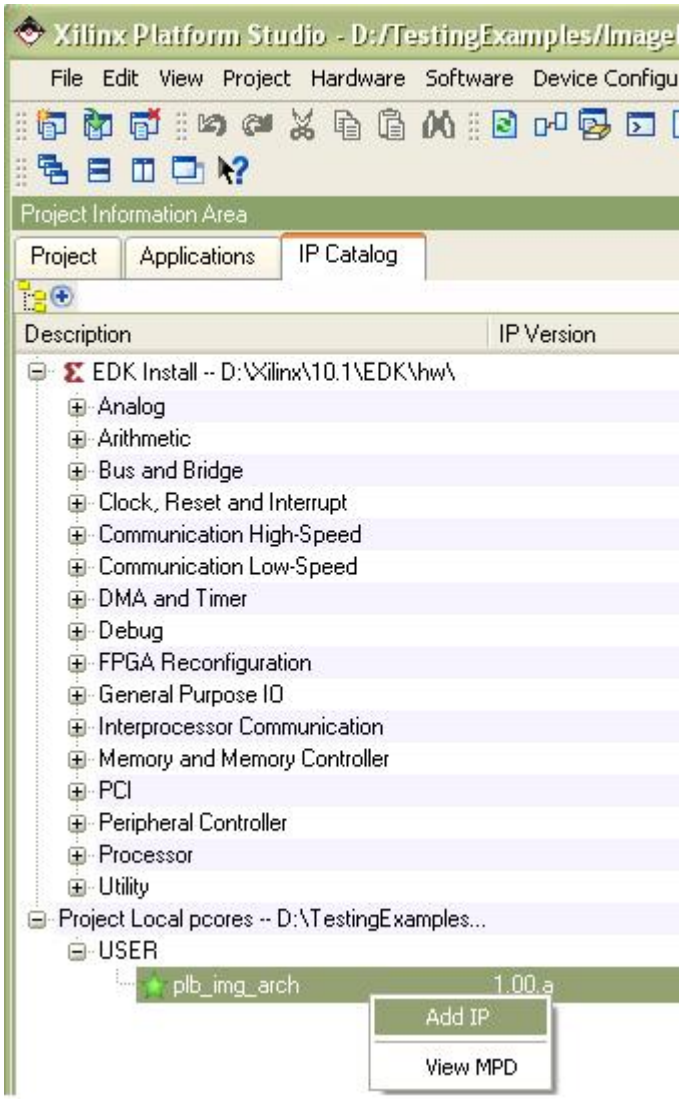
In the previous step you used Xilinx Platform Studio and the Base System Builder to create a test application, ready to download and run on the ML403 board. This test was important because it established that all required peripherals, memories, etc. had been properly assembled, forming a base platform on which the Mandelbrot example can be implemented.

In the steps that remain, we will modify the base platform to:

- Add the ImageFilterDMA accelerator
- Make bus and port connections of the components
- Add the ImageFilterDMA software application files
- Build the platform, including synthesizing the new cores
- Download and run the ImageFilterDMA application on the target board

Adding the ImageFilterDMA Core

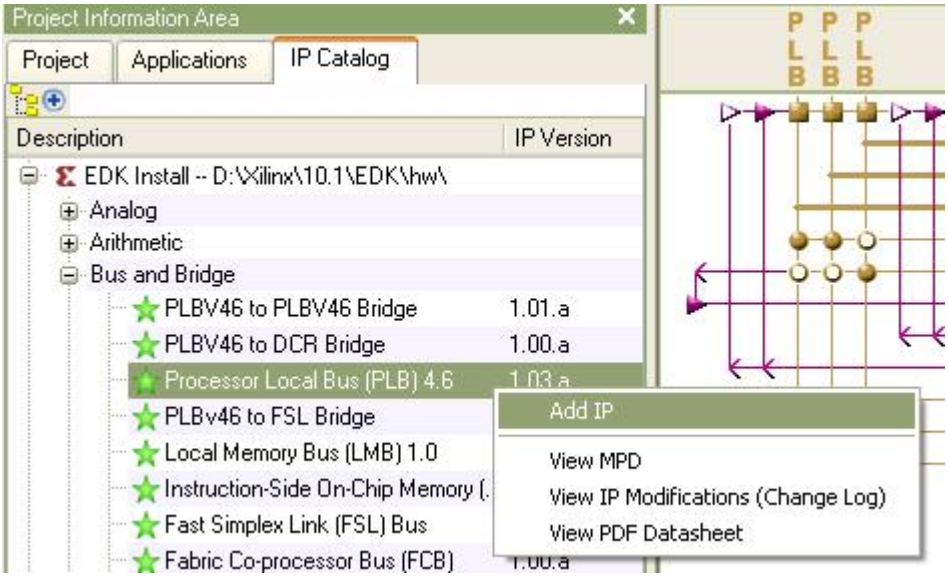
To add the ImageFilterDMA accelerator core as a peripheral, select the IP Catalog tab and look for the category titled "Project Local pcores". Under USER directory you will find the core that was created (copied to) the EDK/pcores directory of your project. Add the plb_img_arch core by clicking the right mouse button as shown below:



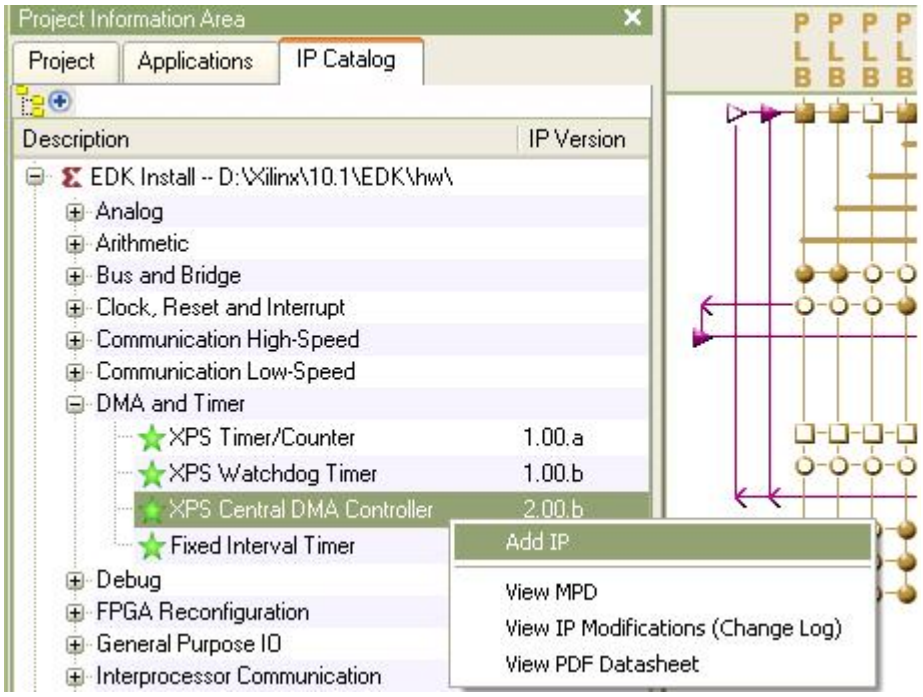
This will add the core to the project as a peripheral.

Adding IP Cores

Next, add a Processor Local Bus (PLB) 4.6 as shown below:

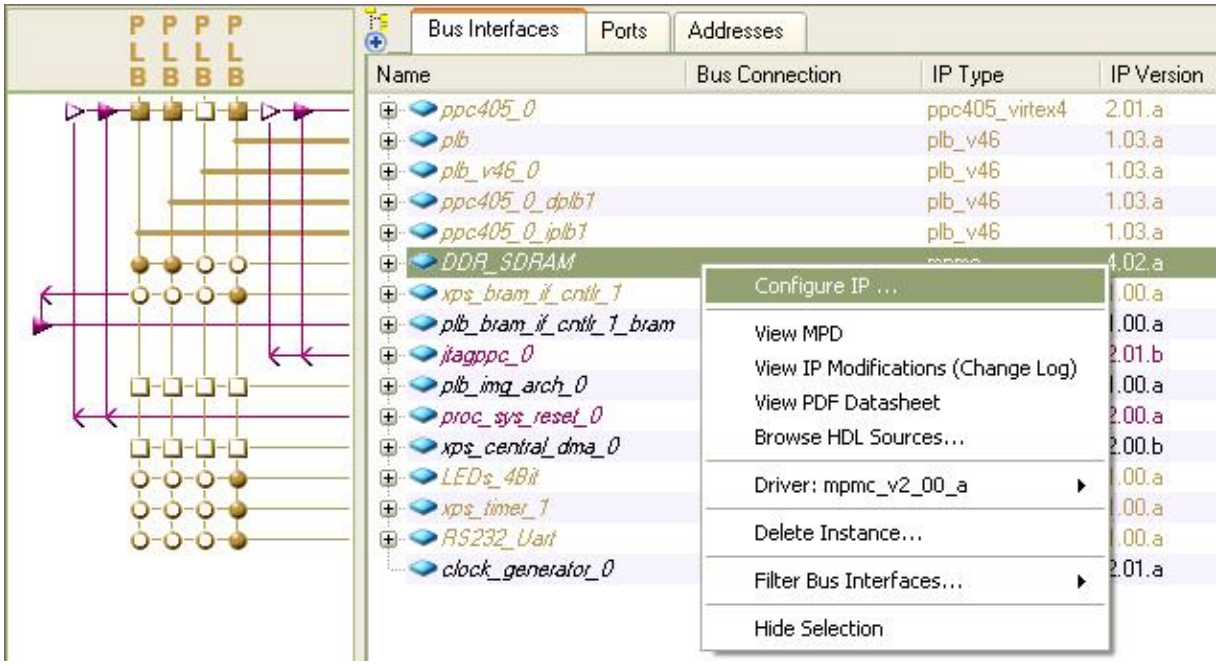


A XPS Central DMA Controller (MPLB device) is needed on the PLB to keep the EDK from optimizing out some arbitrating mechanism. Add the IP core as shown below:

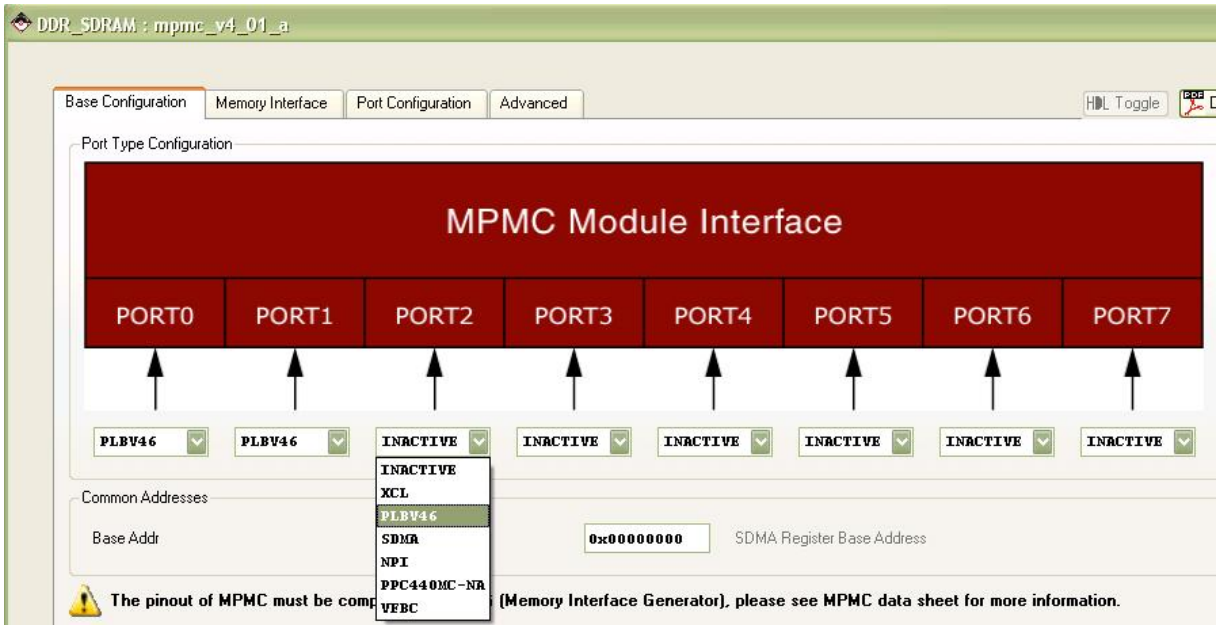


Configure the DDR_SDRAM Controller

The MPLB of plb_img_arch_0 needs to connect to the DDR_SDRAM through a separate SPLB port. To do this, open the Configure IP Dialogue of the DDR_SDRAM:

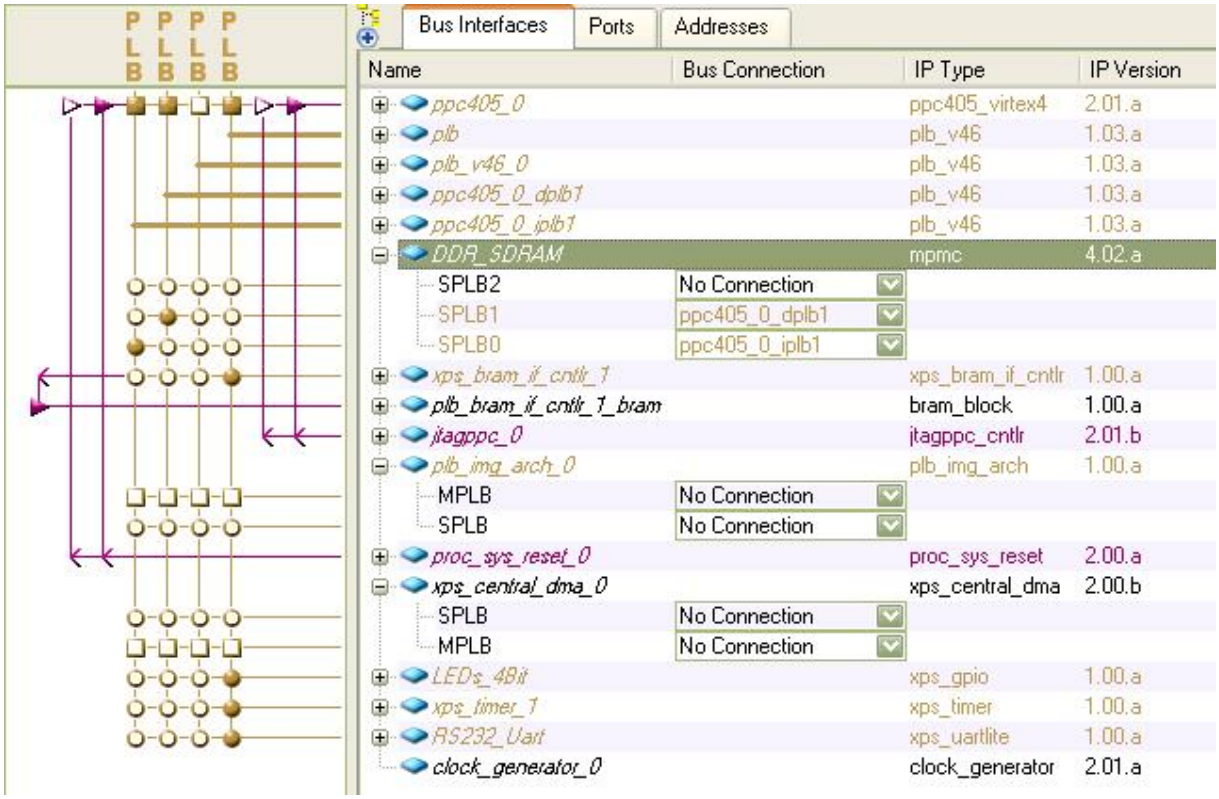


Change the Port Type Configuration of Port 2 from INACTIVE to PLBV46 as shown below. This will add another PLBV46 port to the DDR_SDRAM.



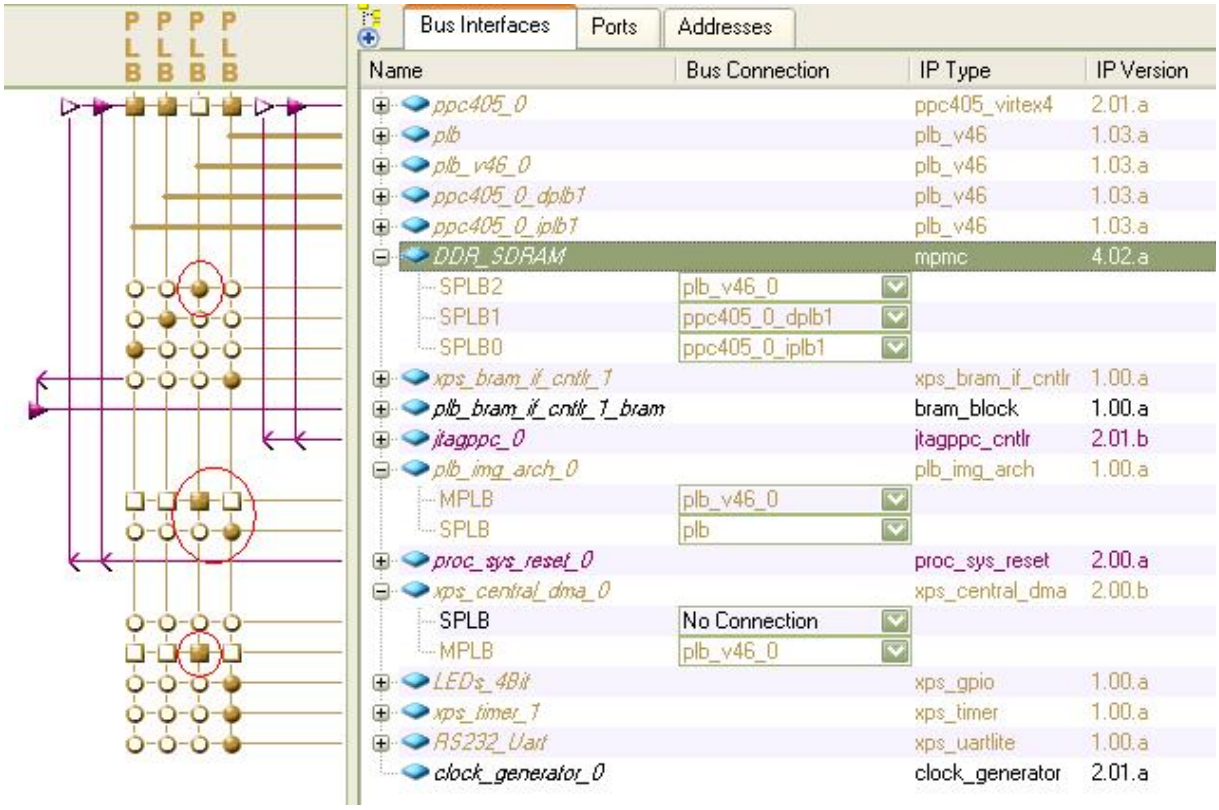
Connect Bus Interfaces

Now, the Bus Interfaces view of the system should look like this:



Connect the MPLB of the plb_img_arch_0, and the MPLB of the xps_central_dma_0 to the newly added PLB (plb_v46_0), also connect the SPLB2 of the DDR_SDRAM to the same plb_v46_0.

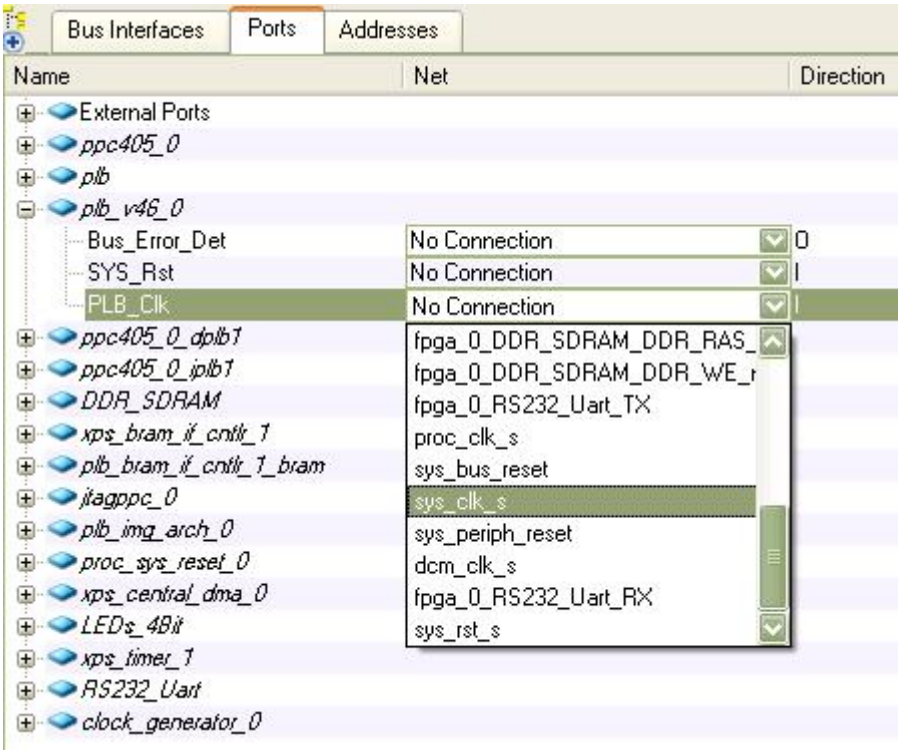
Connect the SPLB of the plb_img_arch_0 to the shared PLB (plb) as shown below (as indicated in red circles):



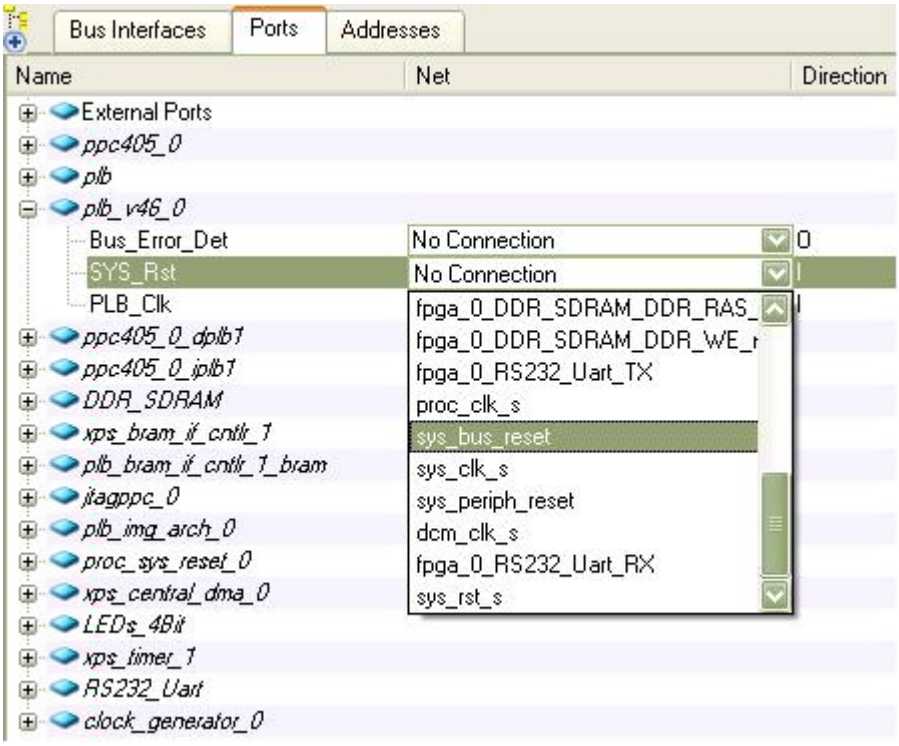
Connecting the Peripheral Clock and Reset Signals

To do this, switch to the Ports Tab in the System Assembly View Window.

Connect the PLB_Clk signal of the plb_v46_0 peripheral to sys_clk_s:



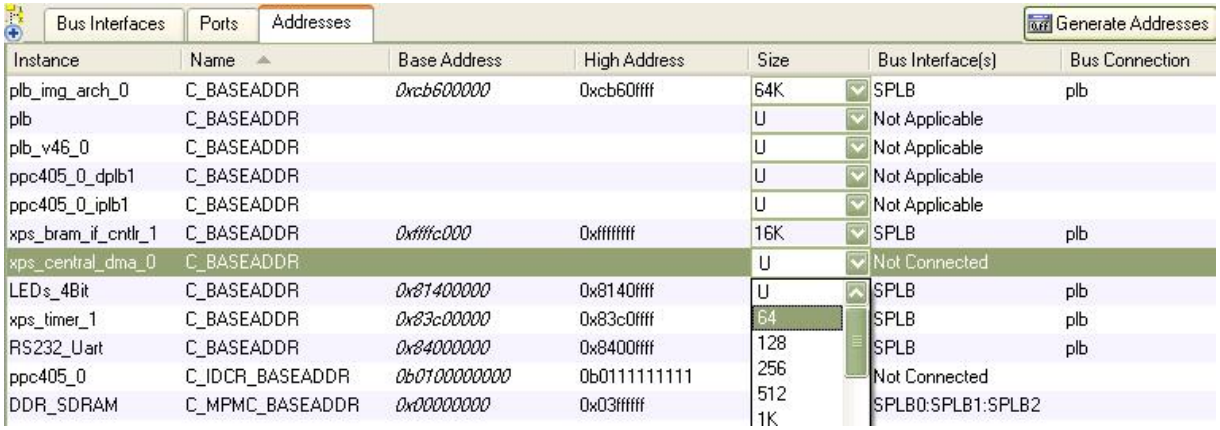
And connect the SYS_Rst signal of plb_v46_0 to sys_bus_reset:



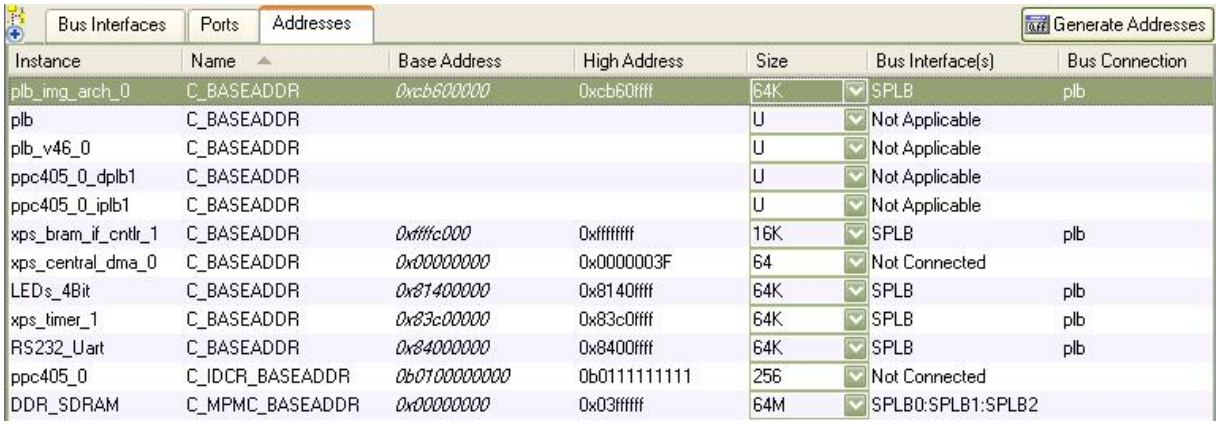
Generate Addresses

Next step is to generate addresses for the memory related modules in EDK. Switch to the Addresses Tab of the System Assembly View Window.

Change the size of the xps_central_dma_0 from U (undefined) to 64 as shown below:



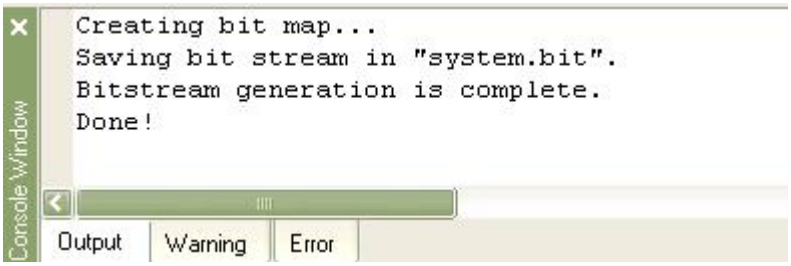
Click the Generate Addresses button on the upper right corner to let EDK assign addresses for the modules as shown below:



Generate Bitstream

Now, build the hardware by choosing the Hardware -> Generate Bitstream menu. The synthesis, place-and-route and bitstream generation process will take a few minutes to complete depending on your PC.

The process is done. A file "system.bit" is created.



The hardware side of the application, including the ImageFilterDMA accelerator and the PLB interface, is now ready for use. In the next tutorial section you will set up the software side of the application.

See Also

Adding the Software Application Files

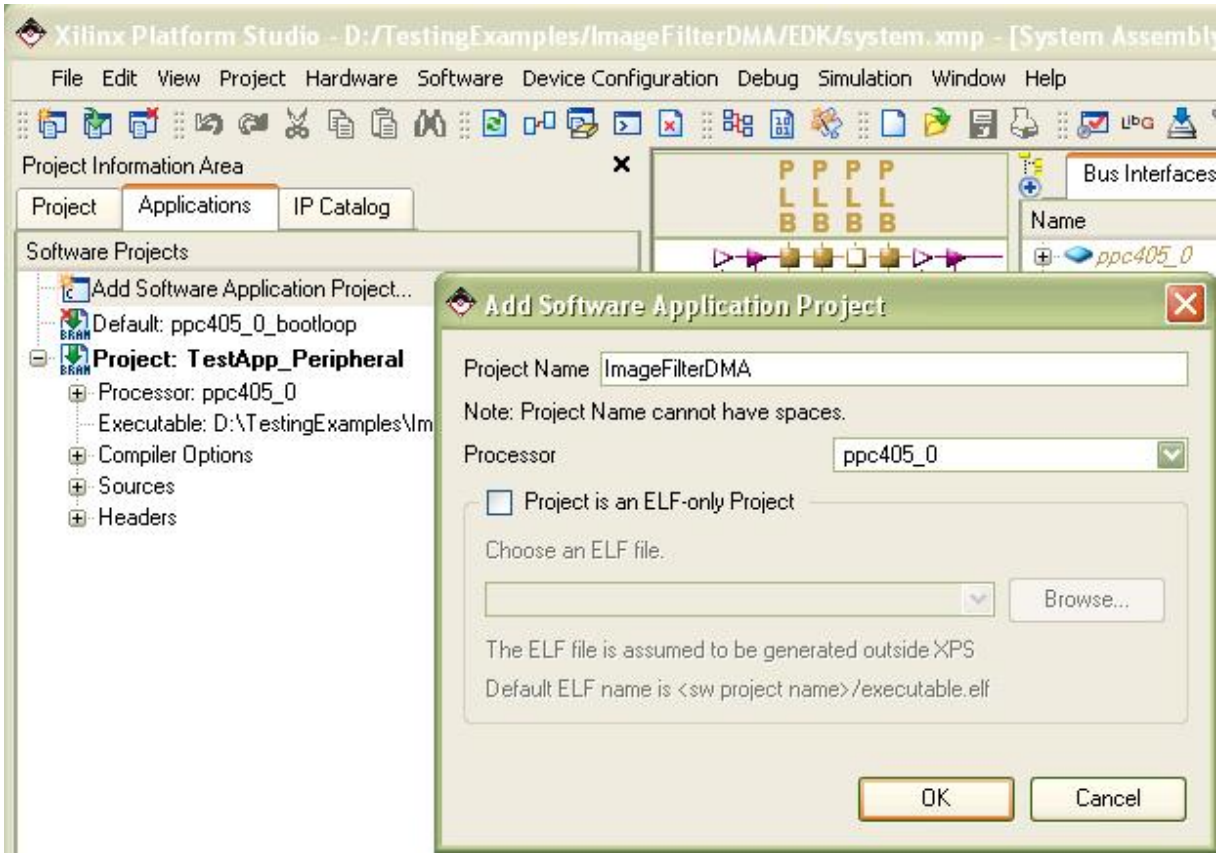
1.2.8 Adding the Software Application Files

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 8

The hardware configuration, including all required peripheral settings and connections, is now complete. The next step is to add the Mandelbrot sample application.

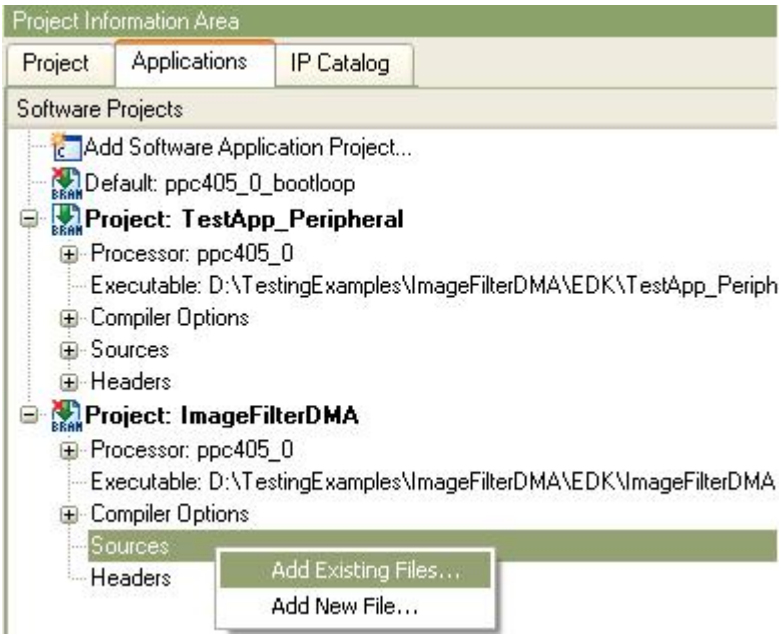
Create Image Filter DMA Software Application

Select the Applications tab of the project, double-click the Add Software Application Project to show a dialogue. Type in the Project Name as "ImageFilterDMA" and click OK as shown below:

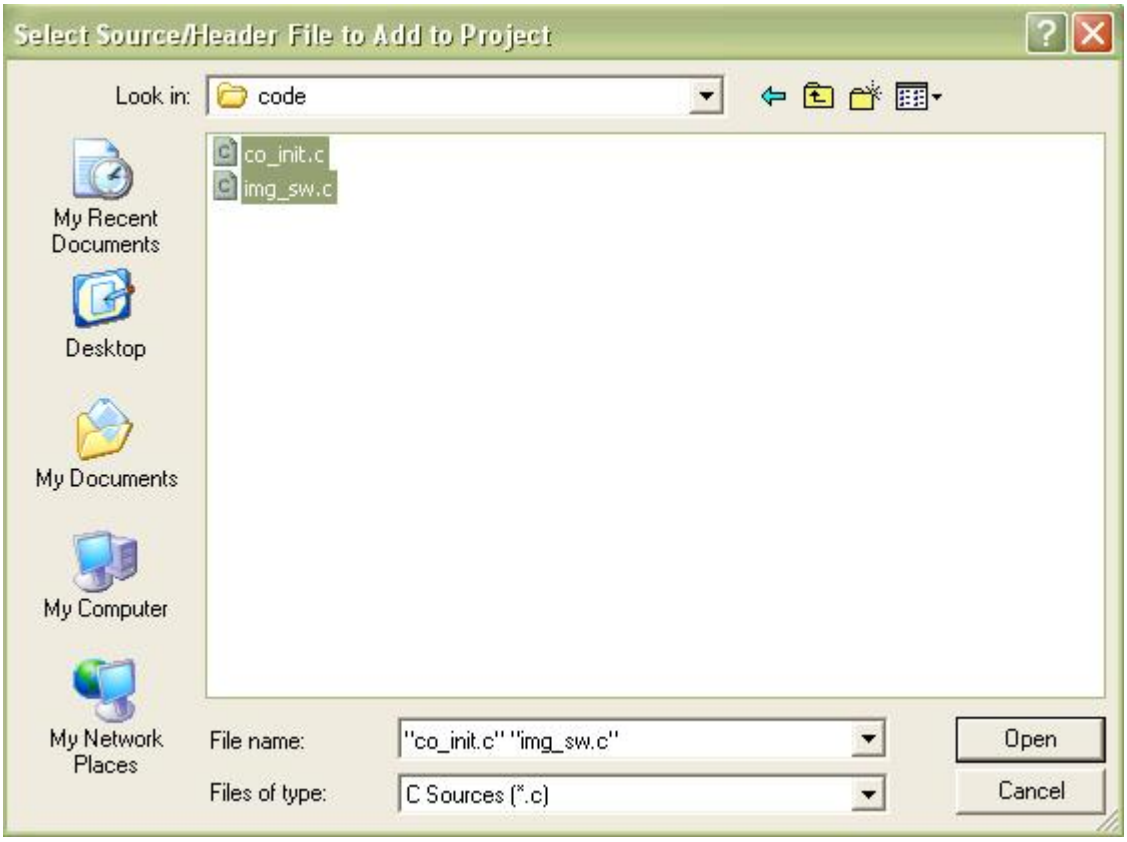


Adding the Image Filter DMA Application Source Files

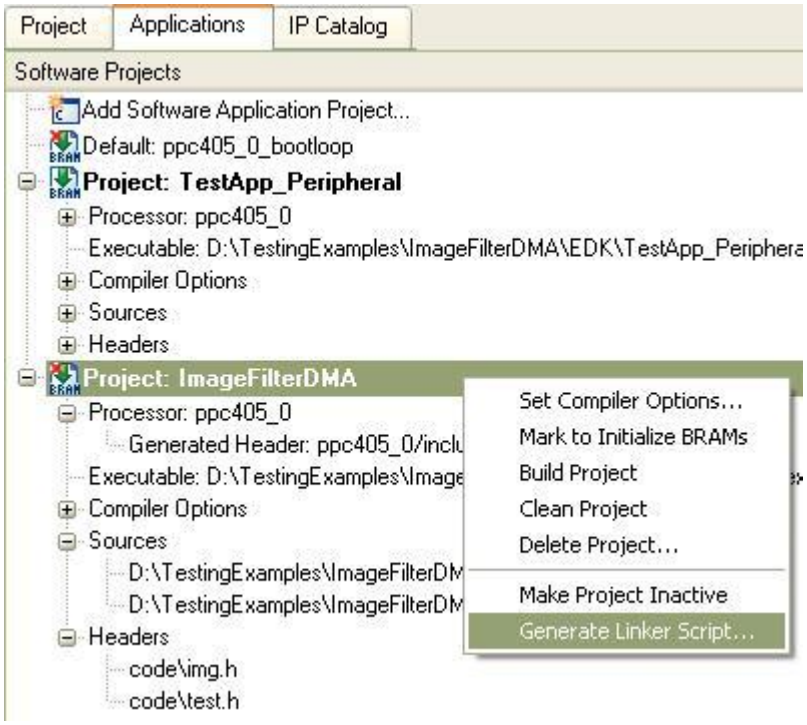
To add source C files to the project, open the Add Existing Files Dialogue from the Sources category by using right mouse button as shown below:



Select all files from the code subdirectory of your project as shown below:

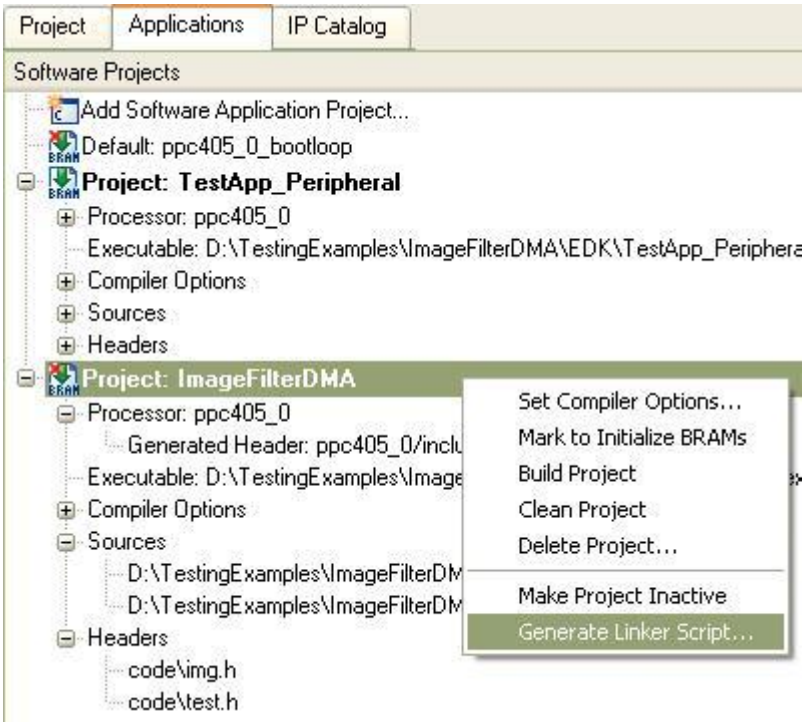


Next, add header files to your project similar to above as shown below:

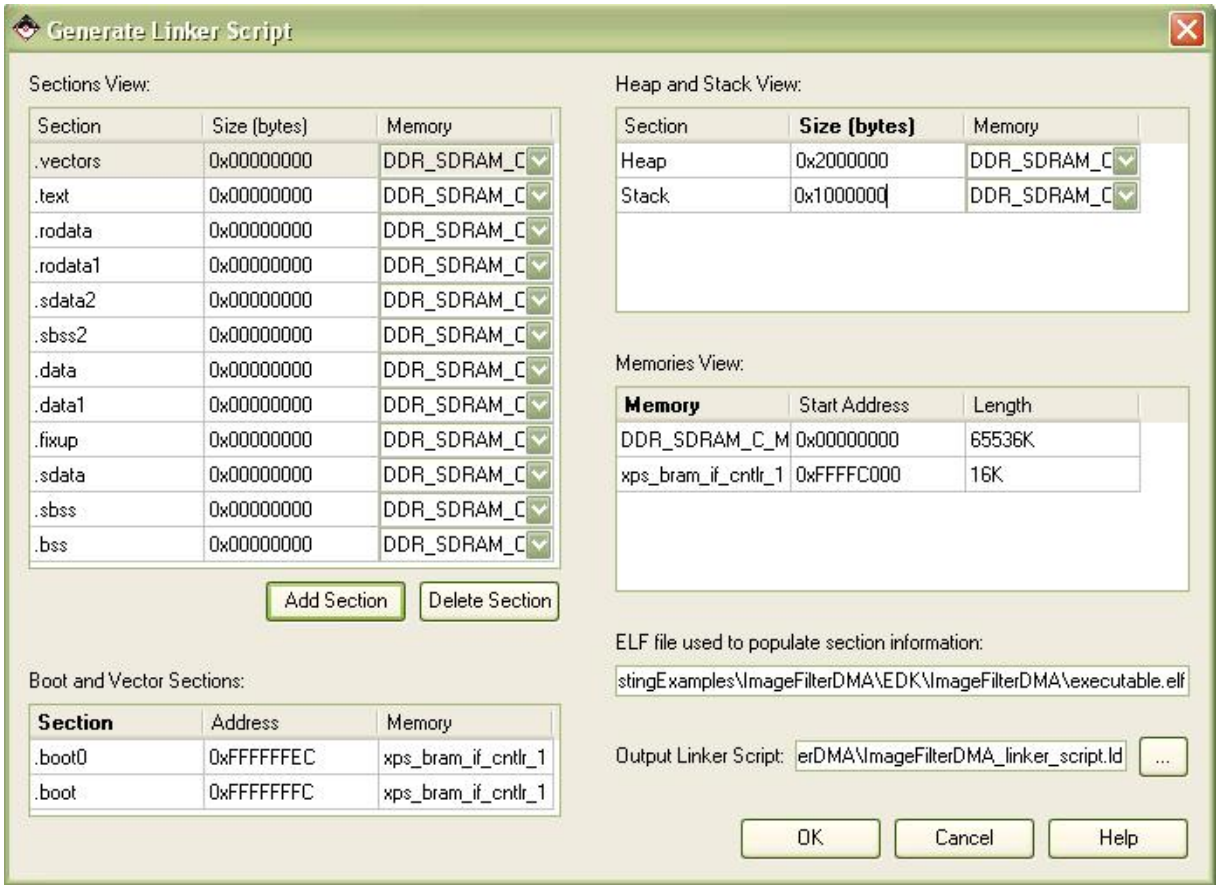


Setting Compiler Options

Now you will need to set compiler options for the project. To set the compiler options, right-click on the project title and select Generate Linker Script from the menu as shown below:



A Generate Linker Script dialogue appears. Change the Heap Size and Stack Size to 0x2000000 and 0x1000000, respectively:



Click OK to close the Generate Linker Script dialog.

The software application is now ready to compile for the PowerPC processor.

See Also

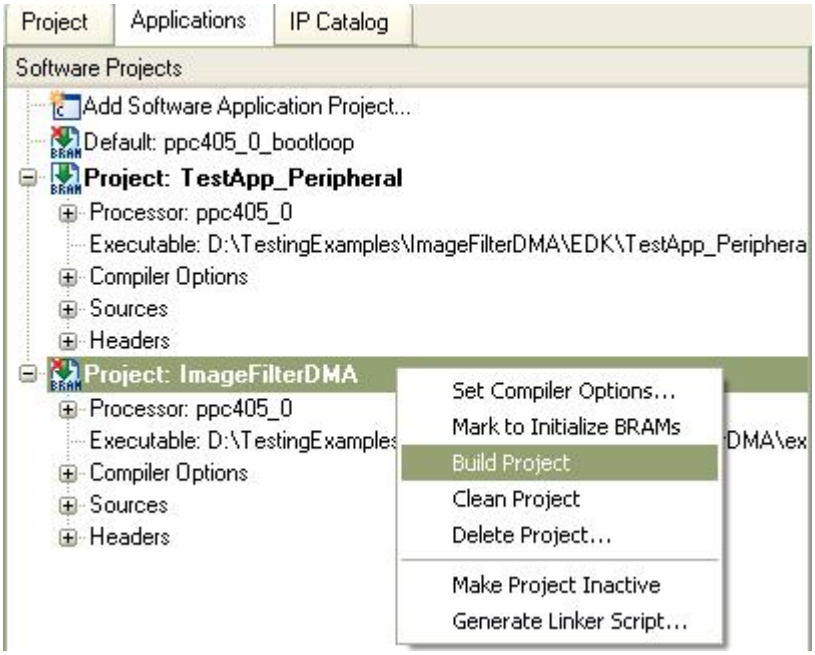
Building and Downloading the Application

1.2.9 Building and Downloading the Application

Image Filter DMA Accelerator Tutorial for Virtex-4 FX, Step 9

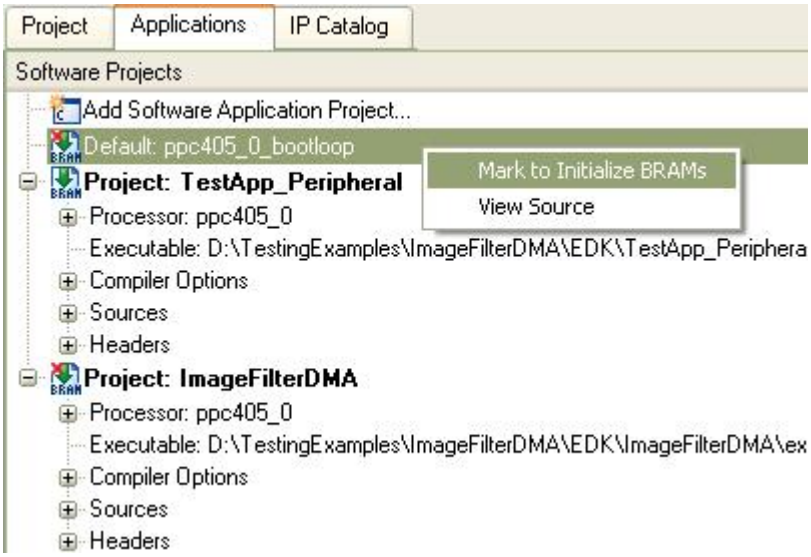
The Mandelbrot application is now ready to build, download and execute on the target ML403 board.

First, compile the software application to create a PowerPC executable. Do this by selecting Build Project from the Project: mand entry as shown below:



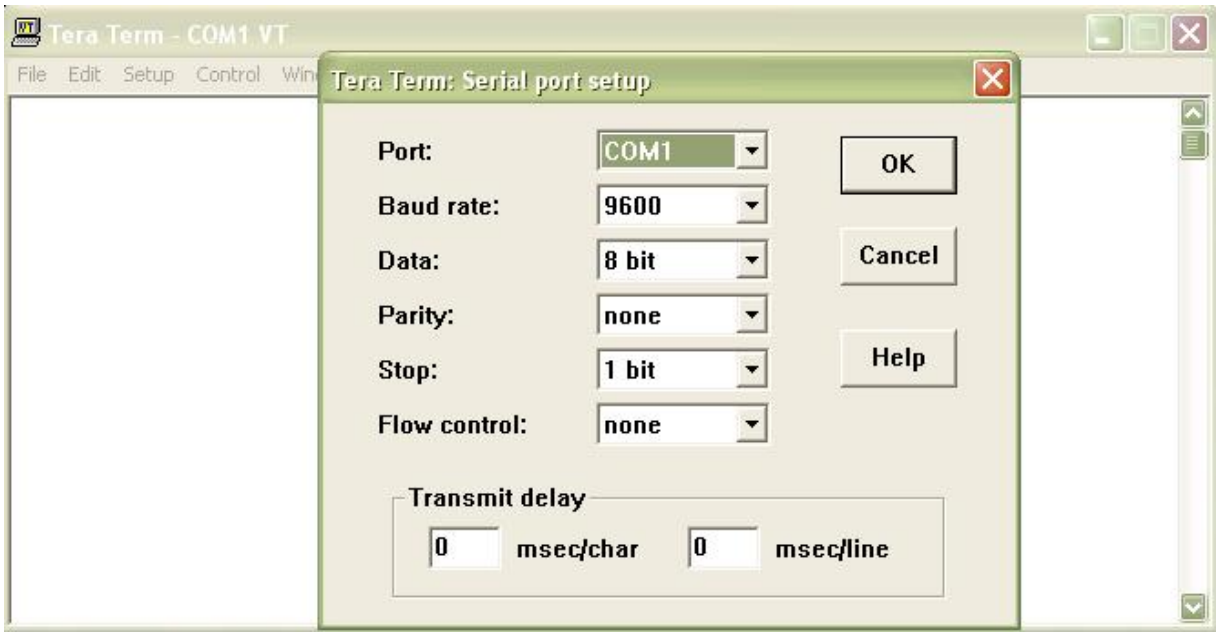
The size of the generated executable is shown below. It will be included in the FPGA bitstream.

Next, mark the ppc405_bootloop to initialize BRAMs by using the right mouse button. This will put a loop in the starting address of the on-chip memory.



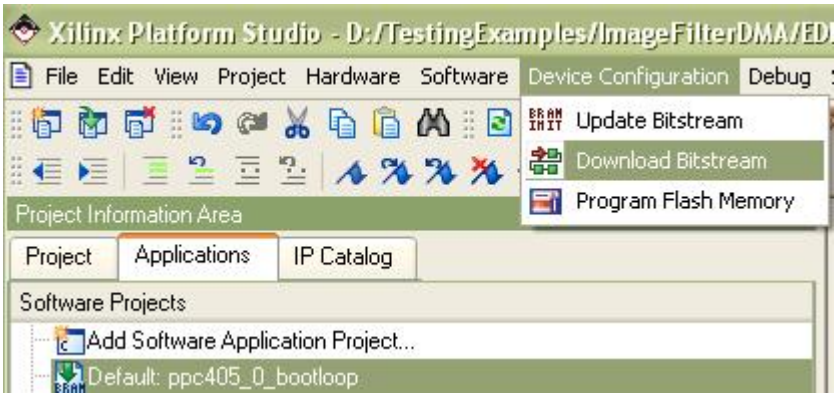
Now, it is time to download the bitstream to the ML403 board. Make sure the JTAG cable is properly connected and that the ML403 board is powered on. Also make sure the crossover RS-232 serial cable is connected properly between the ML403 and your PC.

Open the TeraTerm application to receive the UART output. The serial port is set to be 9600-8-N-1, no flow control, as shown below:



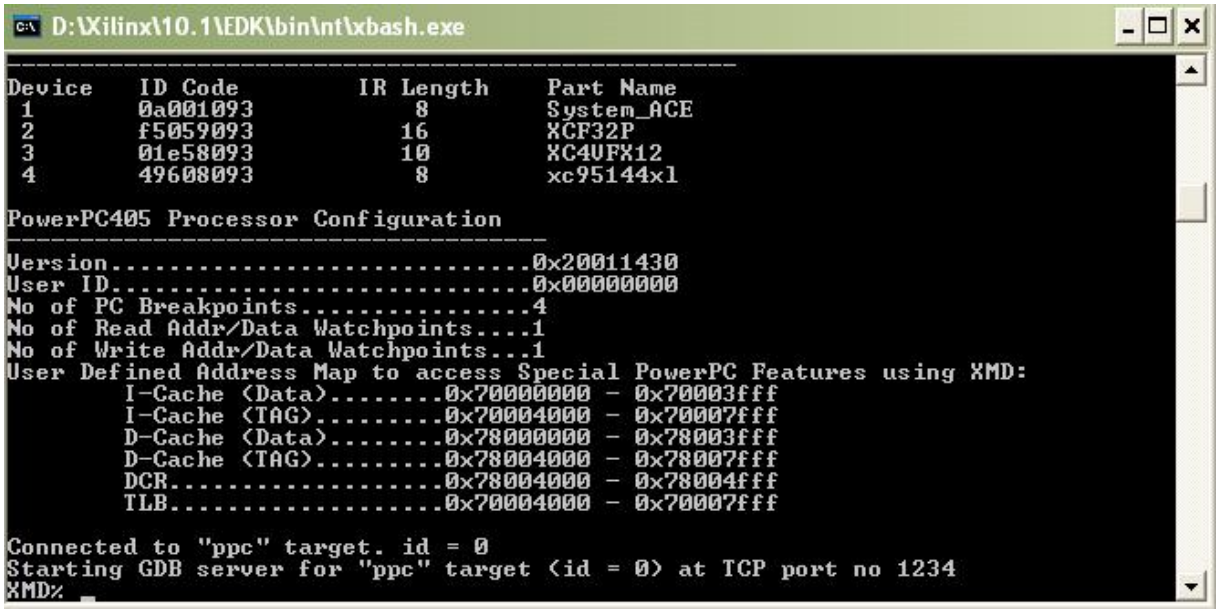
Click OK to accept the settings.

Select Download Bitstream as shown below:



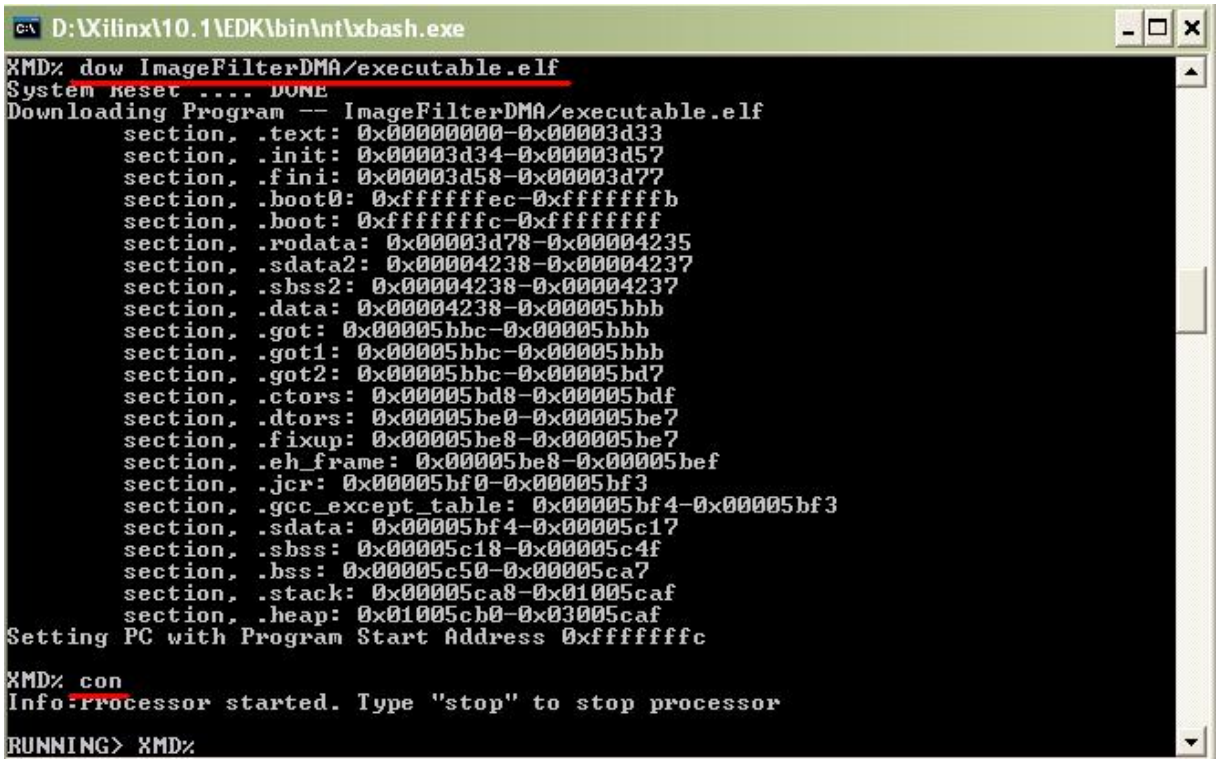
Next, launch Xilinx Microprocessor Debugger (XMD) from the menu as shown below:

If this is the first time you have launched XMD for this EDK project, a couple of dialogue windows will pop up. Just click OK, then the XMD terminal will appear.

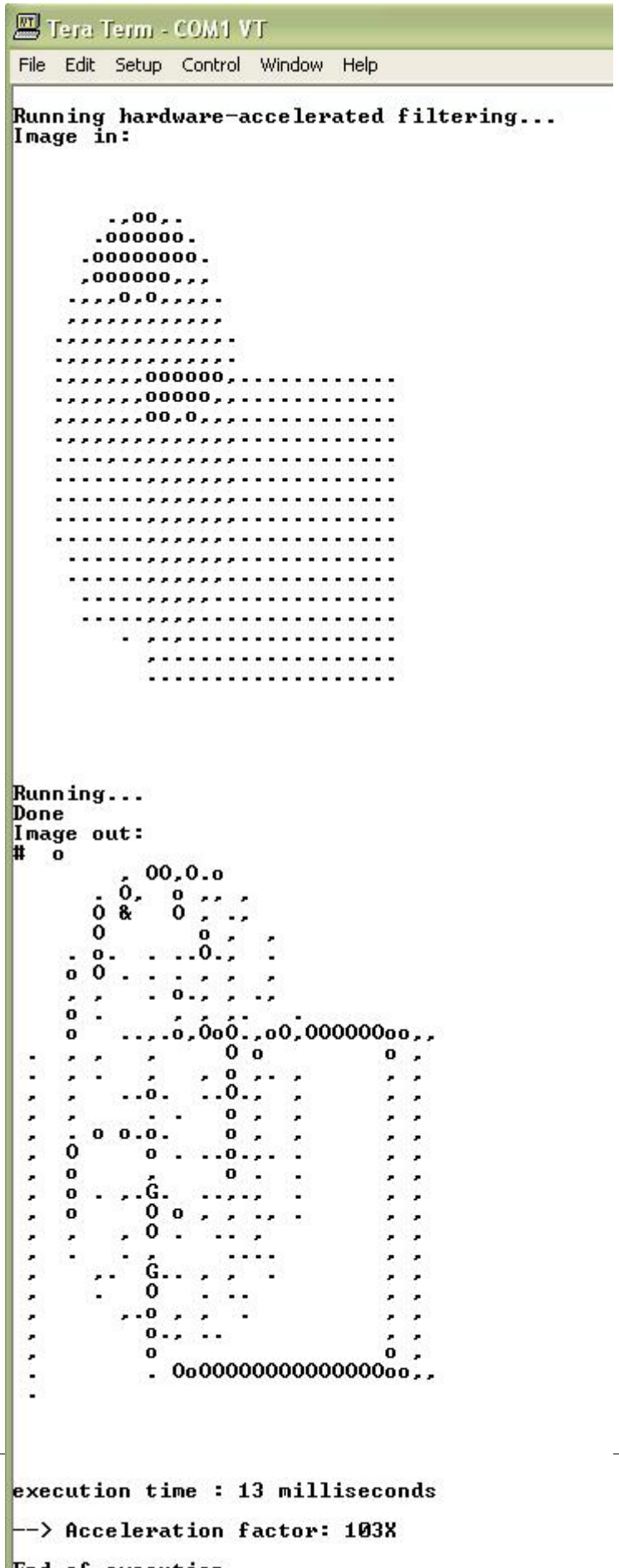


Download the ImageFilterDMA ELF file to the DDR_SDRAM, and then start running the program using the following commands:

dow ImageFilterDMA/executable.elf
con



After downloading has completed, the application will start running, resulting in an output image in the TeraTerm window similar to the following:


Tera Term - COM1 VT
File Edit Setup Control Window Help
Running hardware-accelerated filtering...
Image in:
Running...
Done
Image out:
execution time : 13 milliseconds
--> Acceleration factor: 103X
End of execution.

The execution times of software only filtering and hardware accelerated filtering are measured and compared, and the acceleration factor is 103.

Congratulations! You have completed this advanced tutorial.

See Also

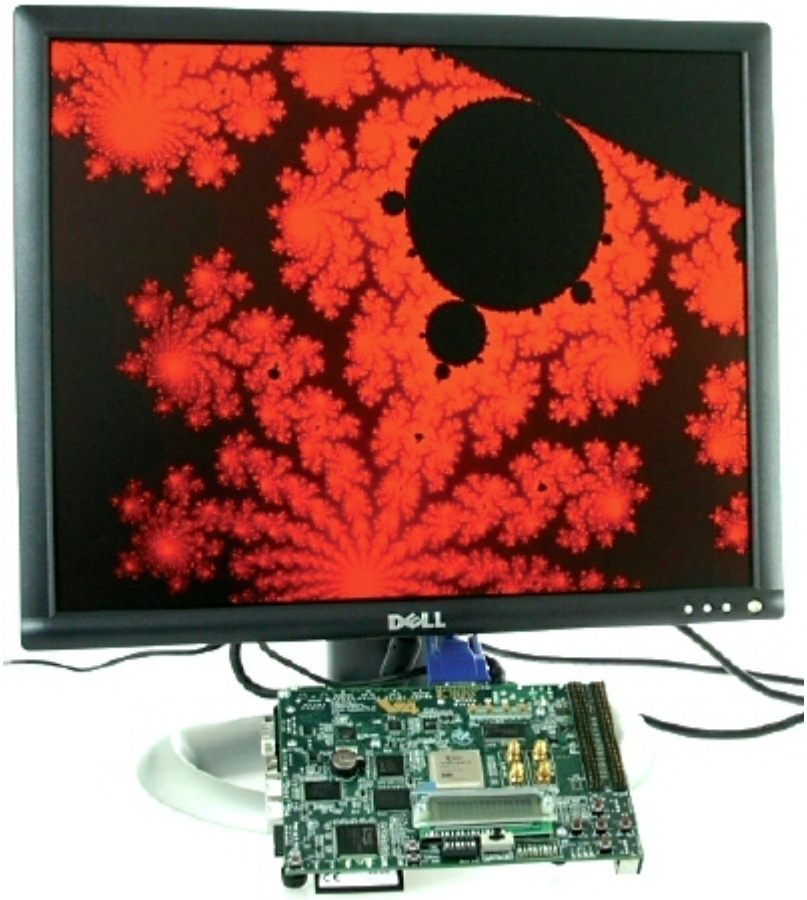
Quick Start Tutorials

1.3 Tutorial 3: Fractal Image Generation using APU on the Virtex-4 Platform (EDK 10.1)

Overview

This tutorial will demonstrate how to create, simulate and build an application targeting the Xilinx Virtex-4 FX platform, including the use of data streams and the Auxiliary Peripheral Unit (APU) interface. It includes all steps necessary to create a new platform using the Xilinx EDK 10.1 tools.

This example is described in Chapter 13 of *Practical FPGA Programming in C*.



This tutorial will require approximately one hour to complete, including software run times. To complete the application, you will need access to a Xilinx ML403 development board (or equivalent board equipped with a Xilinx Virtex-4 FX device), and a VGA monitor as shown above.

You should also download and read the following Xilinx Application Note APP901:

Accelerating Software Applications Using the APU Controller and C-to-HDL Tools.

General Steps

This tutorial will take you through the entire process of creating a hardware-accelerated system in the Virtex-4 FX FPGA using the Impulse and Xilinx tools. This is an advanced tutorial with many detailed steps, but can be summarized as the following general steps:

1. Describe and simulate the application using C language and the Impulse CoDeveloper tools.
2. Automatically generate hardware, in the form of VHDL source files, for the hardware accelerator portion of the application.
3. Export the generated files to an EDK project directory.
4. Build a new EDK project describing the PowerPC and all required peripherals, including the TFT display peripheral.
5. Attach the hardware accelerator generated in step 2 to the PowerPC via the APU interface.
6. Add all needed software files representing the application to be run on the PowerPC.
7. Run synthesis and place-and-route to generate a downloable bitmap.
8. Download the application to the ML403 board using a JTAG programming cable.

Detailed Steps

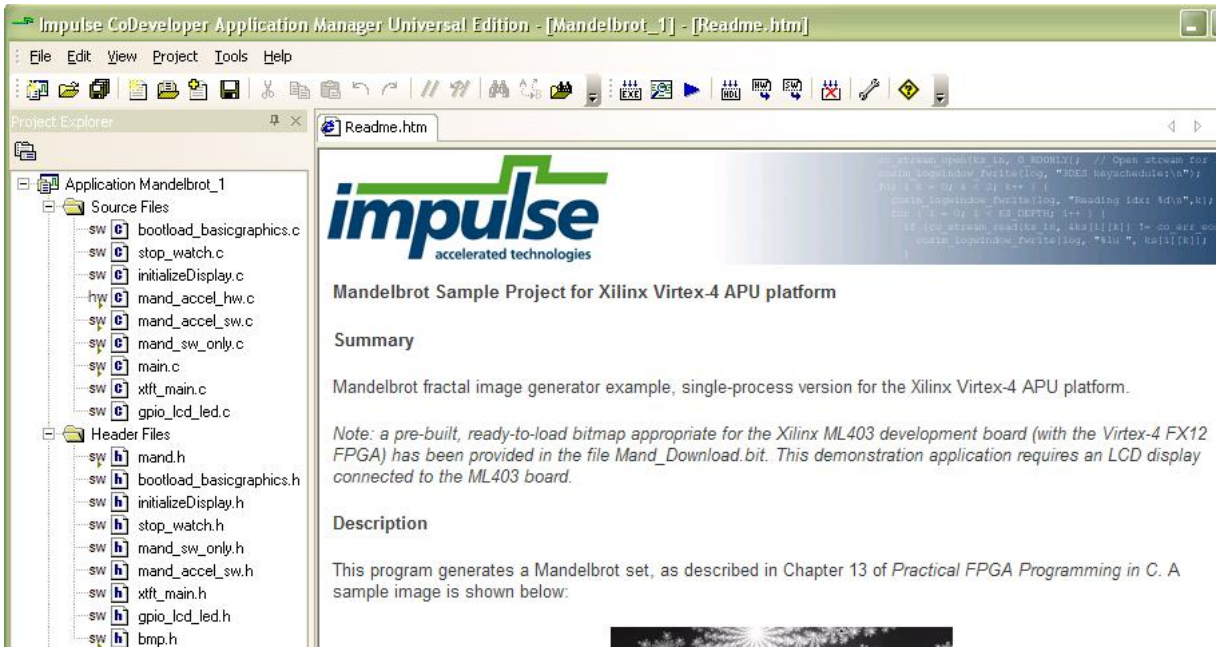
See Also

1.3.1 Loading the Sample Application

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 1

To begin, start the CoDeveloper Application Manager by selecting Application Manager from the Start -> Programs -> Impulse Accelerated Technologies -> CoDeveloper program group.

Open the Xilinx Virtex-4 FX Mandelbrot sample project by selecting Open Project from the File menu, or by clicking the Open Project toolbar button. Navigate to the .\Examples\Xilinx\Virtex4\Mandelbrot\ directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) Opening the project will result in the display of a window similar to the following:



Files included in the Mandelbrot project include:

Source file mand_accel_hw.c - This source files includes the Mandelbrot fractal image generator process, and also includes the application's configuration function.

Source file mand_accel_sw.c - This source file includes the test application that runs on the target PowerPC processor. The test application includes a **main()** function, and a consumer/producer function. As written, this test application can be compiled either on the PowerPC processor or as a desktop simulation executable.

Source file mand.h - This source files includes global definitions, including the image size and precision. This file also includes macros used for fixed-point math operations.

Other .C and .H source files - The remainder of the application source files are used for displaying the results of the application (the generated fractal image) on an LCD display, and for creating a timer used to compare performance.

See Also

Understanding the Mandelbrot Application

1.3.2 Understanding the Mandelbrot Application

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 2

Fractal texturing is a technique used in image rendering to create imagery with an organic appearance. The Mandelbrot image generation algorithm is one example of fractal texturing. This sample application is a fractal image generator that calculates and displays an image such as the one shown below:

To generate this image, the algorithm examines all points in a subregion of a complex plane that has both real and imaginary parts between -2 and +2. The maximum number of iterations to determine if a given point converges is defined by MAX_ITERATIONS, which is defined in source file **mand.h**. You can increase this value for more precision in the generated output.

The generator is implemented as a single Impulse C process. The process accepts configuration data defining the image subregion on a single input stream, and generates the resulting image as a stream of pixels on the output stream. The provided software test bench is compatible with the PPC405 processor in the Virtex-4 FX, and communicates with the hardware process via the APU (Auxiliary Peripheral Unit) interface.

The Virtex-4 APU Controller

The APU controller provides a flexible and high-bandwidth data transfer mechanism between the FPGA fabric (via the Fabric Control Modules, or FCMs) and the embedded PowerPC processor on Virtex-4 FX FPGAs. The APU interface is connected directly to the instruction pipeline and to one or more FCMs. The advantage of this approach is that the typical latency associated with arbitration on a peripheral bus (such as PLB or OPB) is absent.

The Virtex-4 APU controller performs two main functions:

- The APU provides a synchronization mechanism between the PowerPC processor and the FCM, which may be running at a lower clock rate.
- The APU decodes instructions or allows the FCM to decode instructions. Execution, however, is always carried out by the FCM.

When the instruction is due for decoding, it is presented to both the PowerPC processor and APU controller. If the instruction is not recognized as a CPU instruction, the PowerPC processor looks for a response from the APU controller to signal a valid instruction. If valid, the required operands are fetched and passed to the APU for processing. Instructions directed towards the FCM can be either predefined in the Instruction Set Architecture (ISA), such as floating-point instructions, or can be user-defined instructions. The CoDeveloper toolset creates hardware cores designed to interface with the APU interface for easy integration into FPGA systems using XPS. In this example, CoDeveloper uses the load/store instructions (predefined by the ISA) to transfer data between the PowerPC data memory system and the FCM.

See Also

Compiling the Application for Simulation

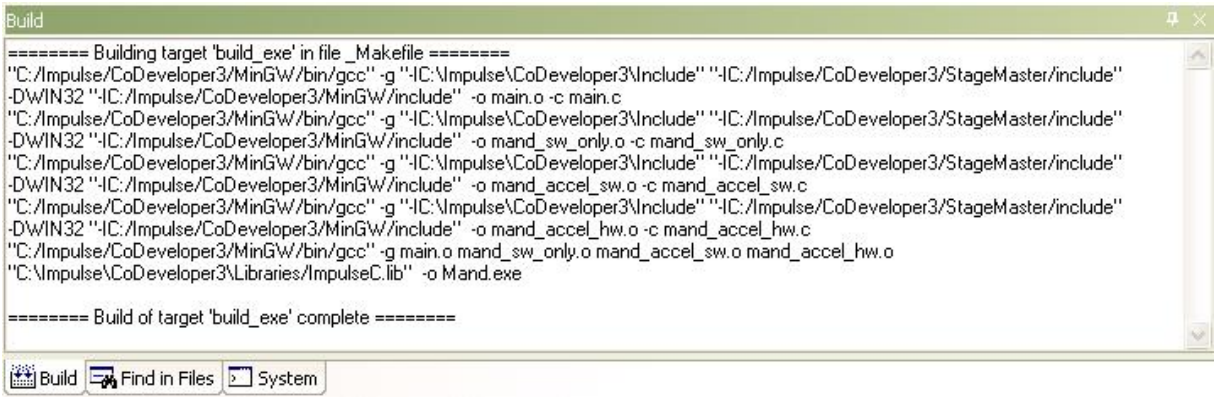
1.3.3 Compiling the Application for Simulation

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 3

The software test bench provided with this example (in **mand_sw.c**) has been written in such a way that it can be compiled either to an FPGA as hardware (using fixed point math operations) or be compiled for desktop simulation, using either fixed or floating point math operations. This makes it possible to compile and simulate the application for the purpose of functional verification.

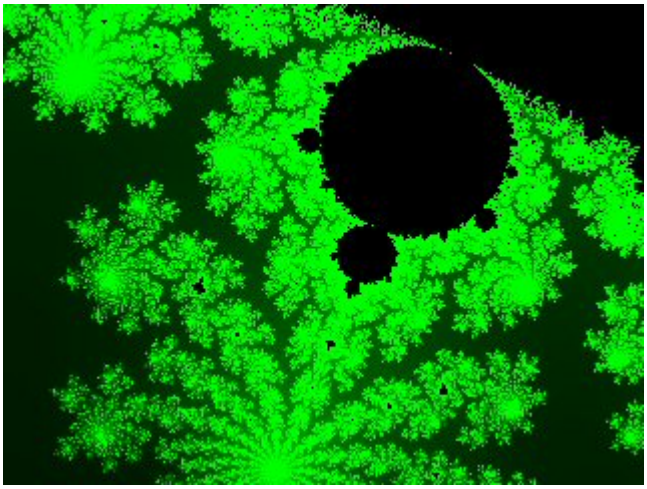
Select Project -> Build Simulation Executable (or click the Build Simulation Executable button) to

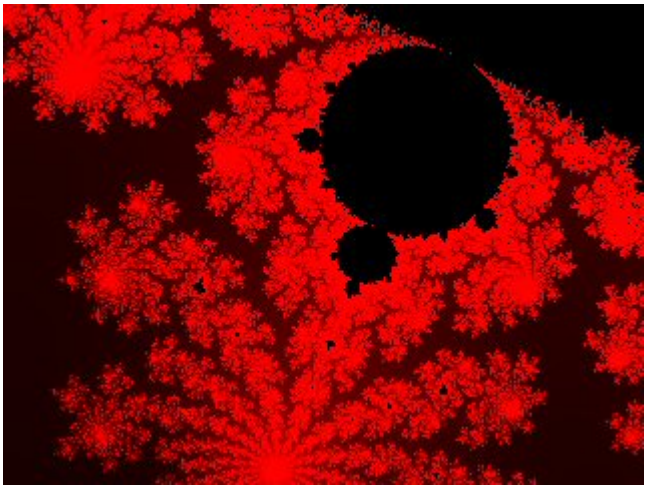
build the Mand.exe executable. The CoDeveloper transcript window will display the compile and link messages as shown below:



You now have a Windows executable representing the application implemented as a desktop (console) software application. You can run this executable by selecting Project -> Launch Simulation Executable. A command window will open and the simulation executable will run as shown below:

When complete, two BMP format files will be created in the project directory that represent the generated hardware and software images, which have been sized for eventual display on the output LCD:





See Also

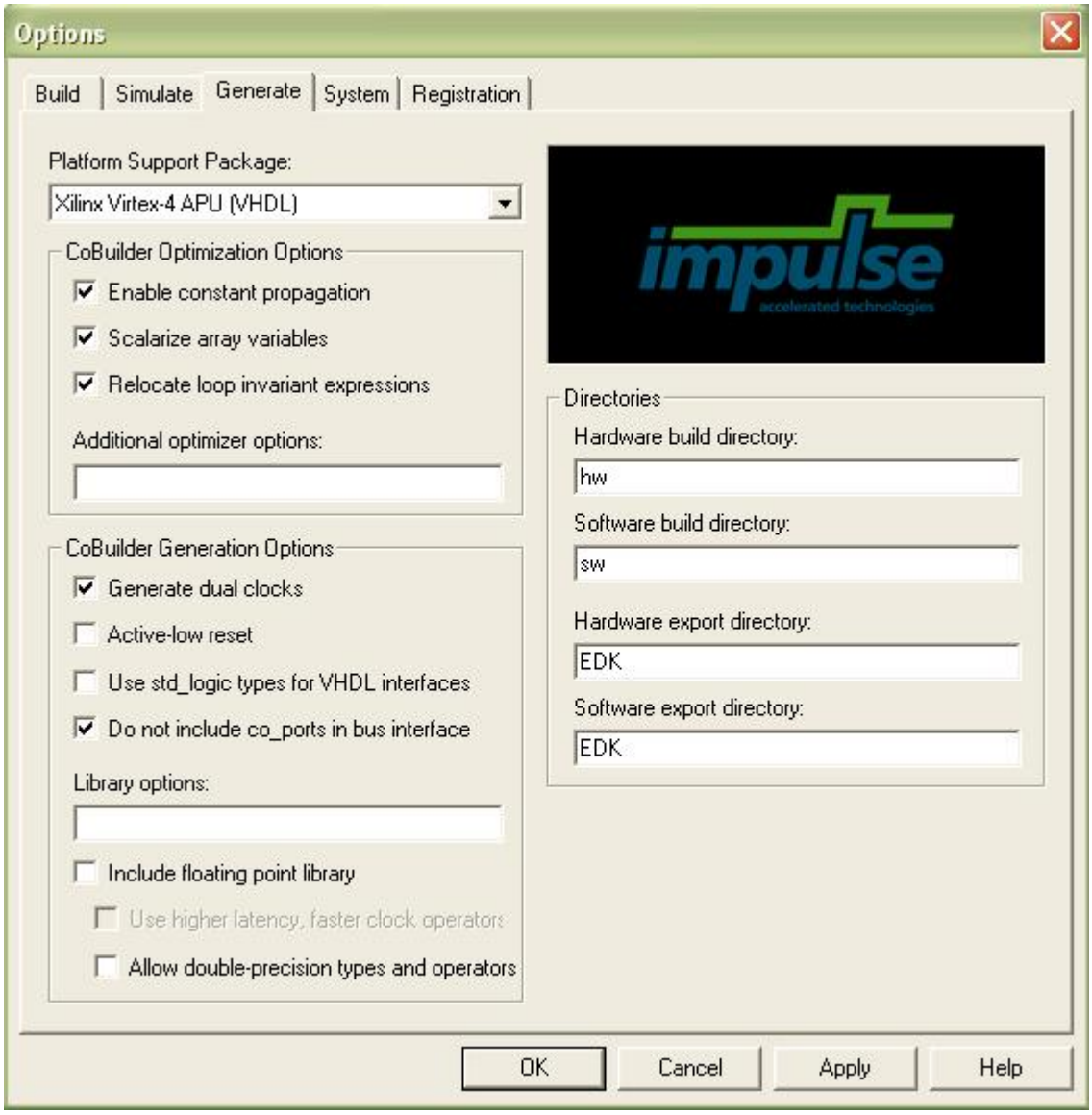
Building the Application for Hardware

1.3.4 Building the Application for Hardware

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 4

Specifying the Platform Support Package

To specify a platform target, open the Generate Options dialog as shown below:



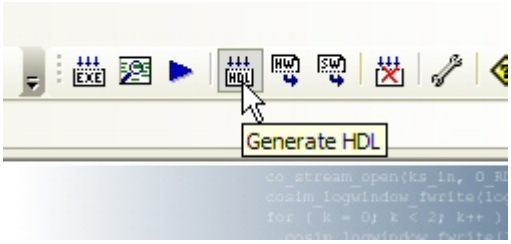
Specify *Xilinx Virtex-4 APU* as shown. Also specify "hw" and "sw" for the hardware and software directories as shown, and specify "EDK" for the hardware and software export directories. ("EDK" is the directory in which you will be creating a Xilinx Platform Studio project.)

Also ensure that the Generate Dual Clocks option is selected as shown. (The Generate Dual Clocks option is important because you will be clocking the PowerPC processor at a different rate than the generated FPGA logic.)

Click OK to save the options and exit the dialog.

Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces and library files, select Generate HDL from the Project menu, or click the Generate HDL button as shown:



A series of processing steps will run in a command window as shown below:

Note: the processing of this example may require a minute or more to complete, depending on the performance of your system.

When processing has completed you will have a number of resulting files created in the **hw** and **sw** subdirectories of your project directory. These files are ready to be exported into a Xilinx Platform Studio project directory.

See Also

Exporting the Hardware and Software Files

1.3.5 Exporting the Hardware and Software Files

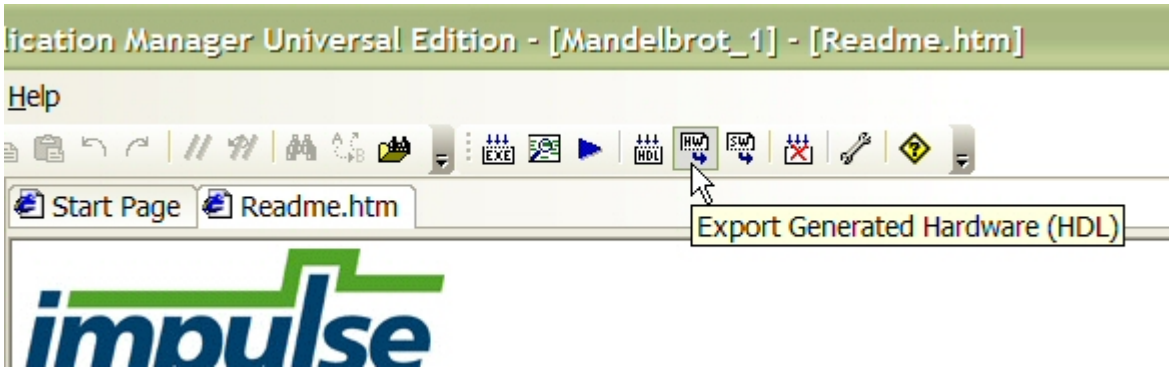
Mandelbrot Extended Tutorial for Virtex-4 FX, Step 5

Recall that in the previous step you specified the directory "EDK" as the export target for hardware

and software. These export directories specify where the generated hardware and software processes are to be copied when the Export Software and Export Hardware features of CoDeveloper are invoked. Within these target directories (in this case "EDK"), the specific destination for each file previously generated is determined from the Platform Support Package architecture library files. It is therefore important that the correct Platform Support Package (in this case Xilinx Virtex-4 APU) is selected prior to starting the export process.

To export the files from the build directories (in this case "hw" and "sw") to the export directories (in this case the "EDK" directory), select Project -> Export Generated Hardware (HDL) and Project -> Export Generated Software, or select the Export Generated Hardware and Export Generated Software buttons from the toolbar.

Export the Hardware Files



Export the Software Files

Note: you must select BOTH Export Software and Export Hardware before going onto the next step.

You have now exported all necessary files from CoDeveloper for use in the Xilinx tools environment. By opening a Windows Explorer window, you can see how the hardware and software files have been copied into subdirectories of your EDK directory. In particular, notice that CoDeveloper has created a "pcores/apu_mand_v1_00_a" directory containing the generated HDL and other related files. This generated directory structure will allow you to import the generated core directly into the Platform Studio tools.

See Also

Copying the TFT Display Core Files

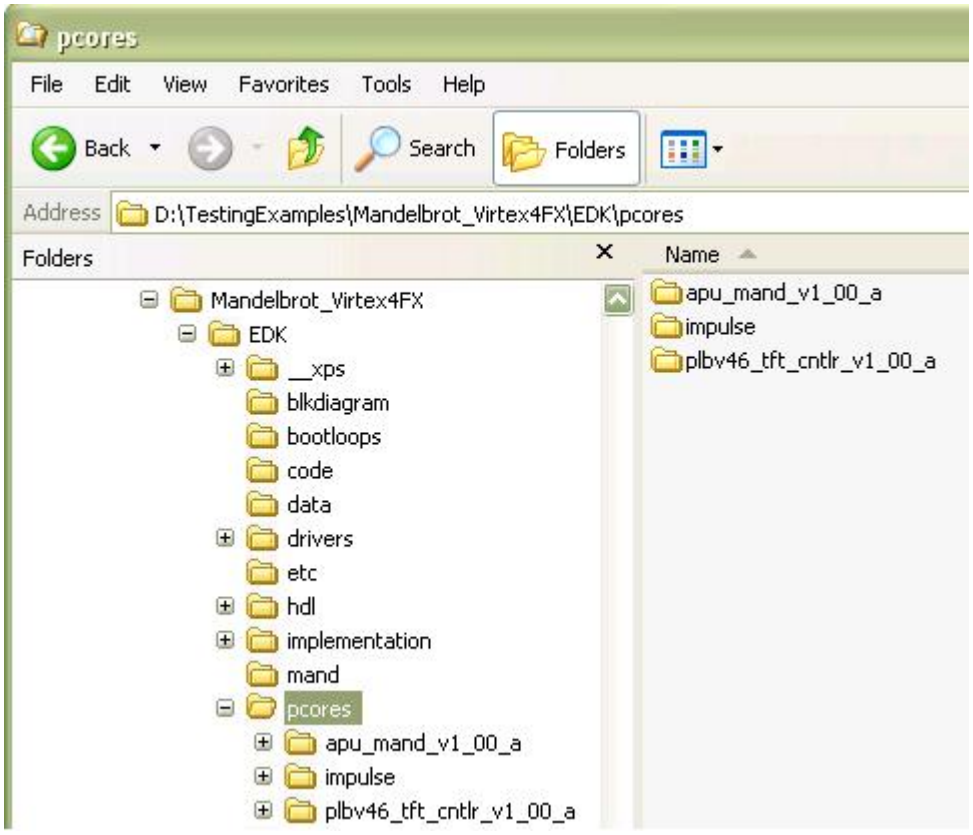
1.3.6 Copying the TFT Display Core Files

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 6

As described in the previous step, the CoDeveloper tools are capable of generating all required files for "pcore" components usable within the Xilinx Platform Studio tools. These pcore components represent processor peripherals and other components that can be assembled, using the Platform Studio tools, to create a complete system.

In this example, we will also make use of another pcore for driving the TFT display. This pcore has been provided by Xilinx, but is not part of the standard Platform Studio (EDK) installation. For your convenience, the required TFT pcore has been included as a ZIP file with the Mandelbrot sample project.

To add the TFT pcore to our EDK project directory, unzip the supplied file (located in the ../Mandelbrot/EDK directory), resulting in the following directory structure in your EDK project subdirectory:



Note: The included ZIP file may include other directories, including a pre-built Mandelbrot accelerator pcore. These additional files can be ignored. In particular, you should take care not to overwrite the pcores/apu_mand_v1_00_a directory created in the previous step.

See Also

Creating the ML403 Test Platform

1.3.7 Creating the ML403 Test Platform

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 7

At this point you have:

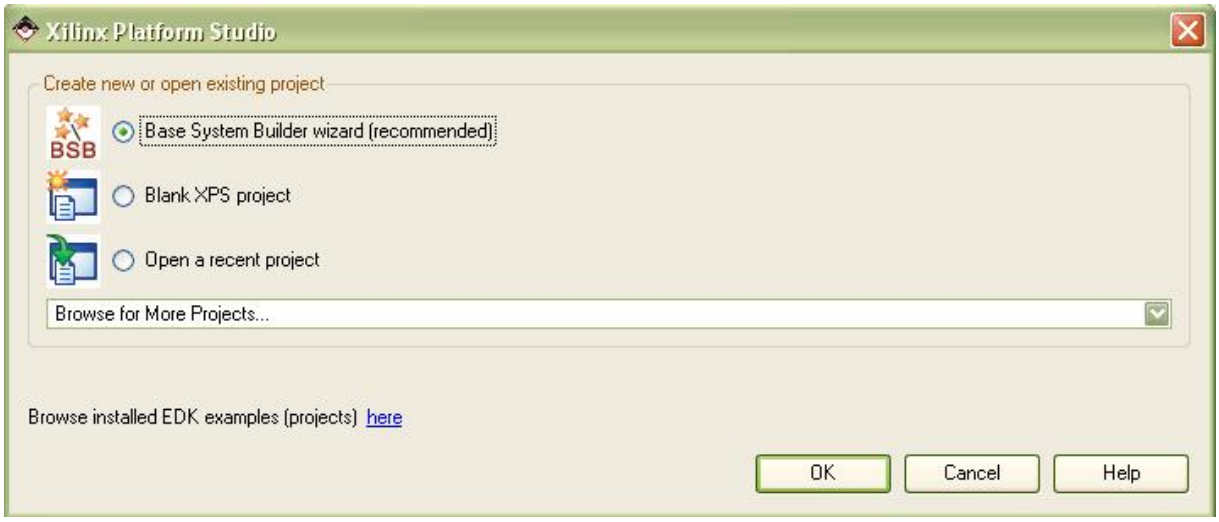
- Created hardware for the Mandelbrot accelerator.
- Exported the generated hardware to the EDK subdirectory as a pcore.
- Exported the PowerPC software application files to the EDK subirectory.
- Created/copied an additional pcore representing the TFT display interface.

In this tutorial section, you will be making use of the Platform Studio tools, including the Base System Builder Wizard, to define and build a new PowerPC-based platform targeting the Xilinx ML403 development board. You will first create a test platform allowing you to download and verify your PowerPC and its standard peripherals. After successfully creating and testing the basic platform, you will add the necessary hardware and software files to build, download and test the Mandelbrot sample application.

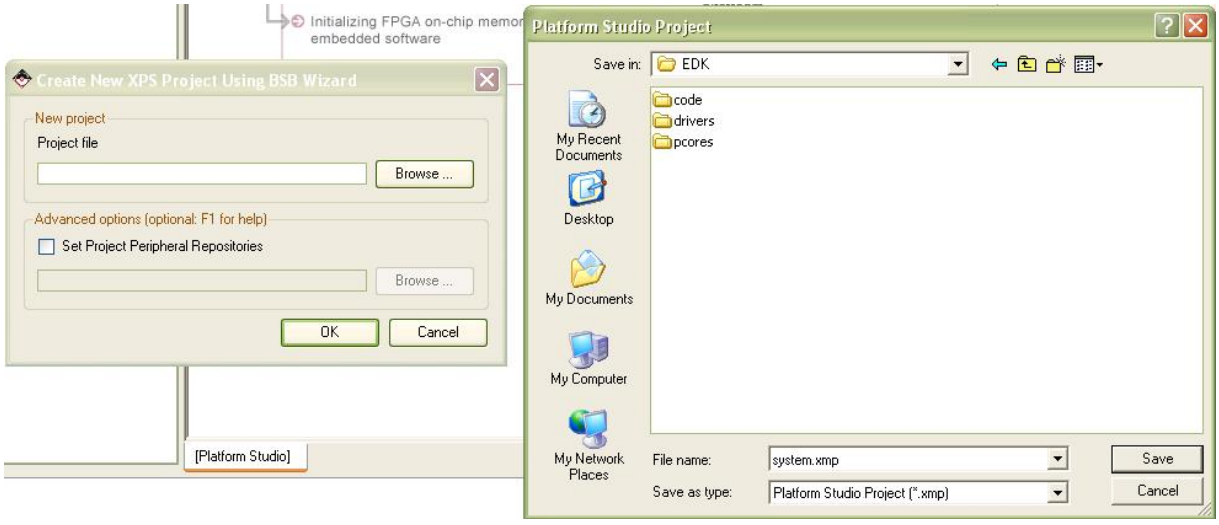
Note: If you are using a different Virtex-4 FPGA development board, you will need to obtain an associated .XBD file from your board vendor, as described in the introduction to this tutorial.

Using Base System Builder to Create the Platform

To begin, start the Xilinx Platform Studio tools and select the Base System Builder Wizard as shown below:



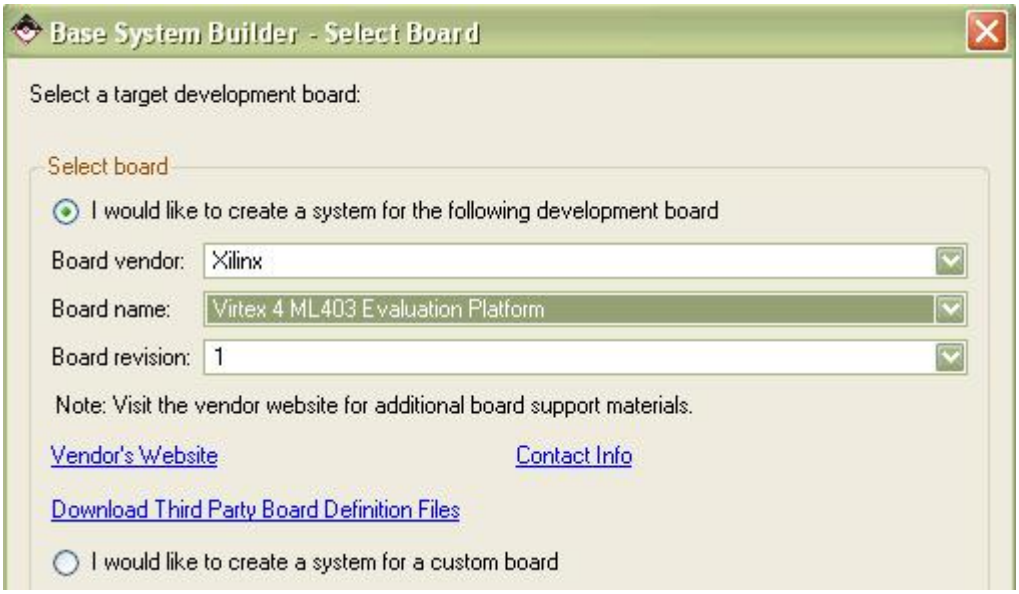
Click the OK button to proceed. When asked for a project name and location, specify the EDK subdirectory of your project, and accept the default project name (system.xmp) as shown below:



Press the OK button to continue.

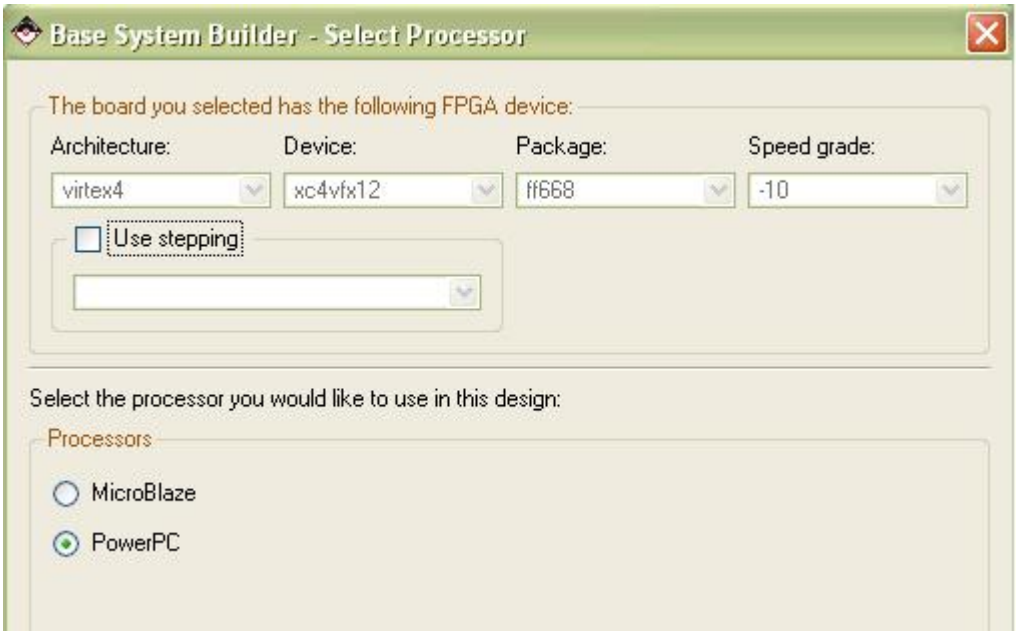
You will now be presented with the Base System Builder Wizard. Select the "I would like to create a new design" option, then click Next to continue:

Next, select your target board using the "Board vendor" and "Board name" drop-down lists. To use the Xilinx ML403 board with attached LCD display, choose the "Virtex 4 ML403"" board as shown:



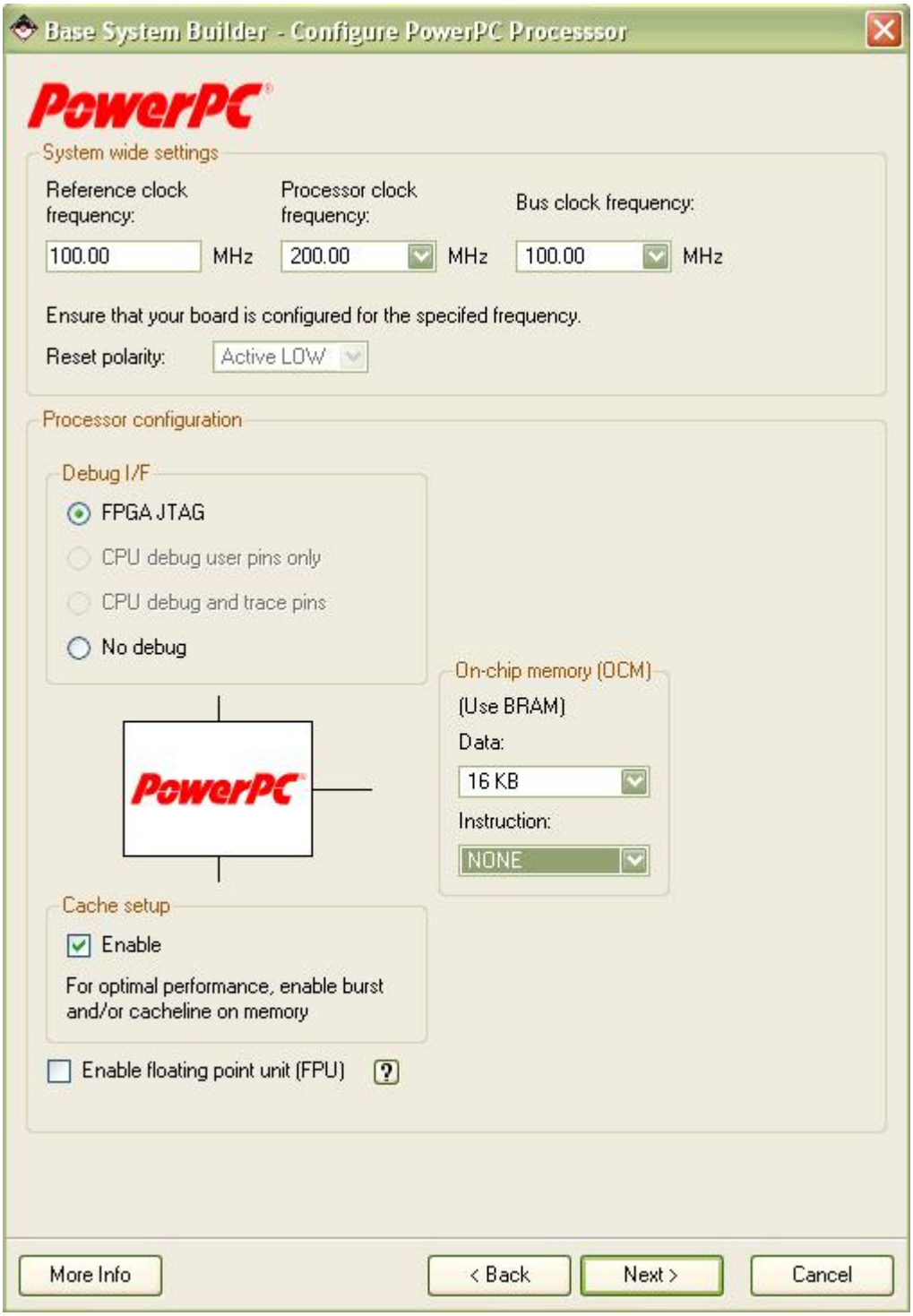
Click the Next button to proceed to the next Wizard page.

On the Select Processor page, be sure PowerPC is selected as the target processor, then click Next:



On the Configure PowerPC page, specify the following options:

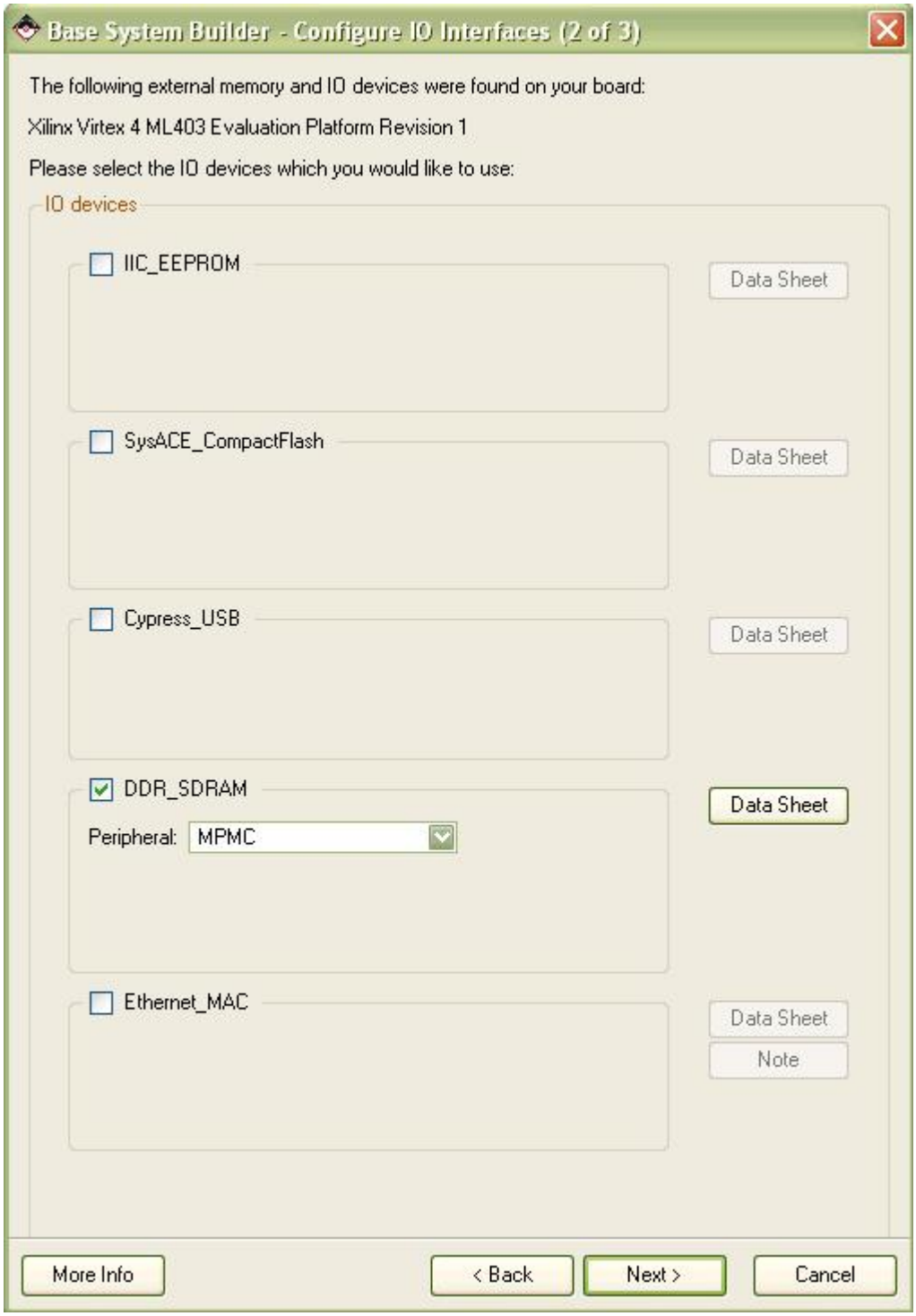
Processor clock frequency: 200 MHz
Debug I/O: JTAG
Cache setup: Enable
On-chip memory: 16 KB each for data and instruction



Click Next to continue. You will now be presented with a series of pages for configuring various I/O interfaces. Select the RS232_Uart and LEDs_4Bit peripherals as shown, but do not select the LEDs_Positions and the Push_Button_Position peripheral:

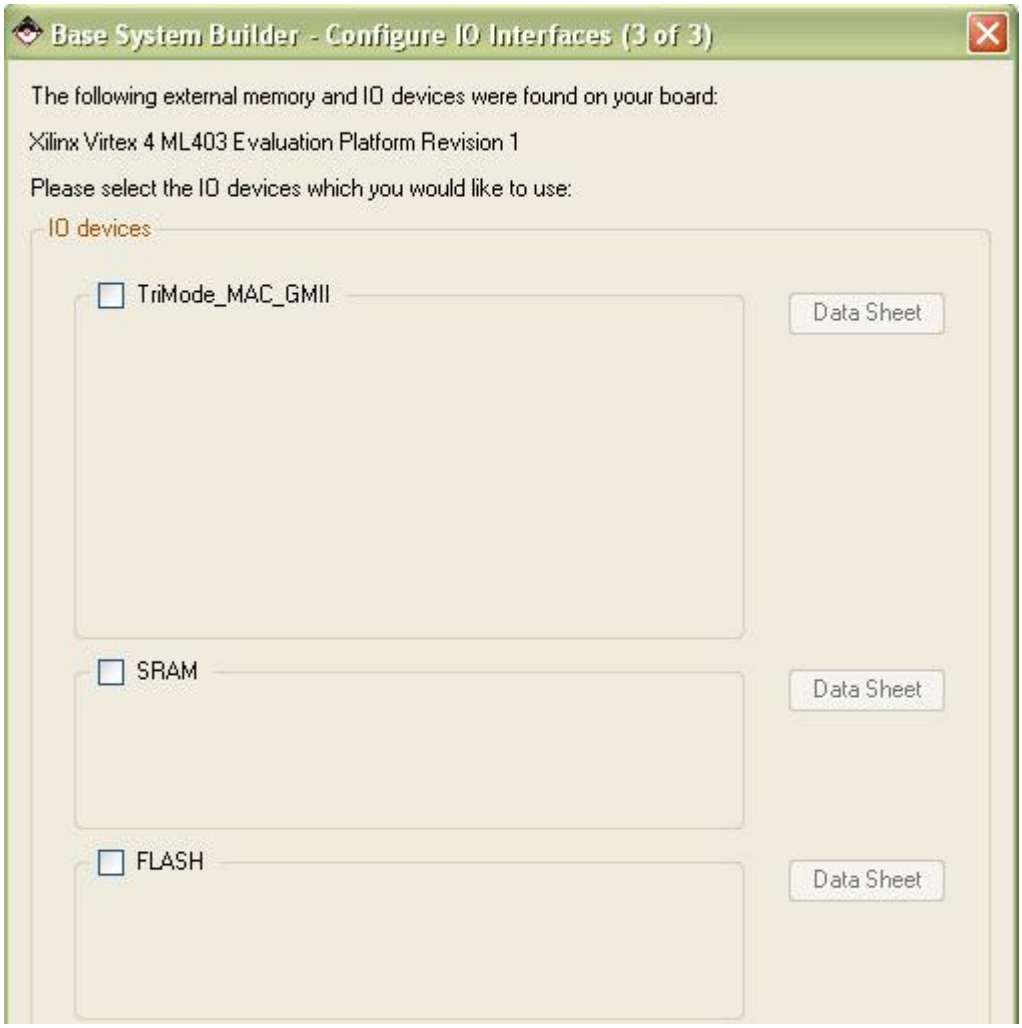
Click Next.

On the next Wizard page, select only the DDR_SDRAM peripheral:



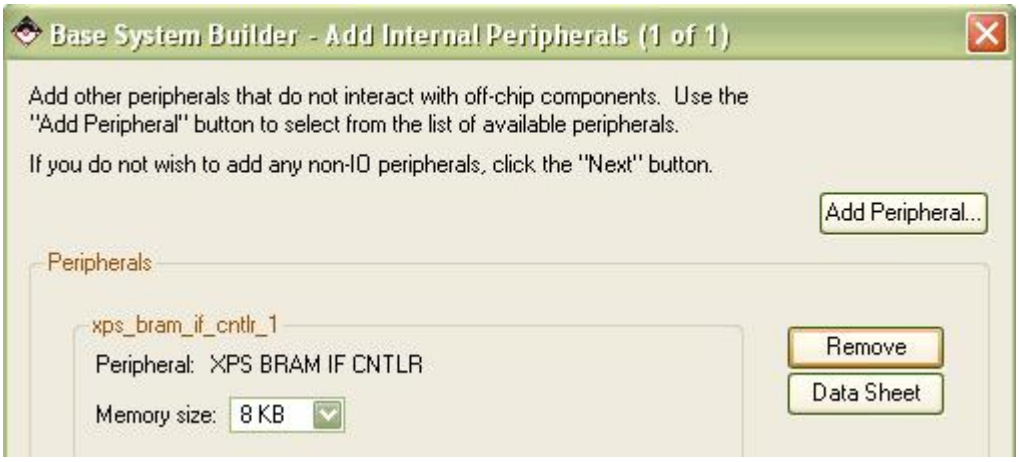
Click Next.

On the page that follows, do not select any of the peripherals:



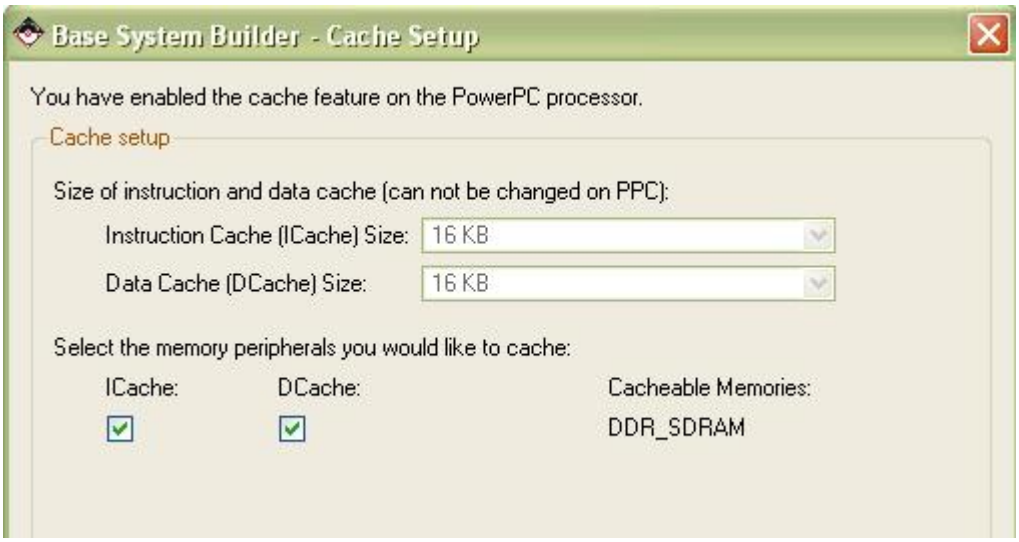
Click Next.

On the Add Internal Peripherals page, remove the plb_bram_if_cntlr_1 as shown:



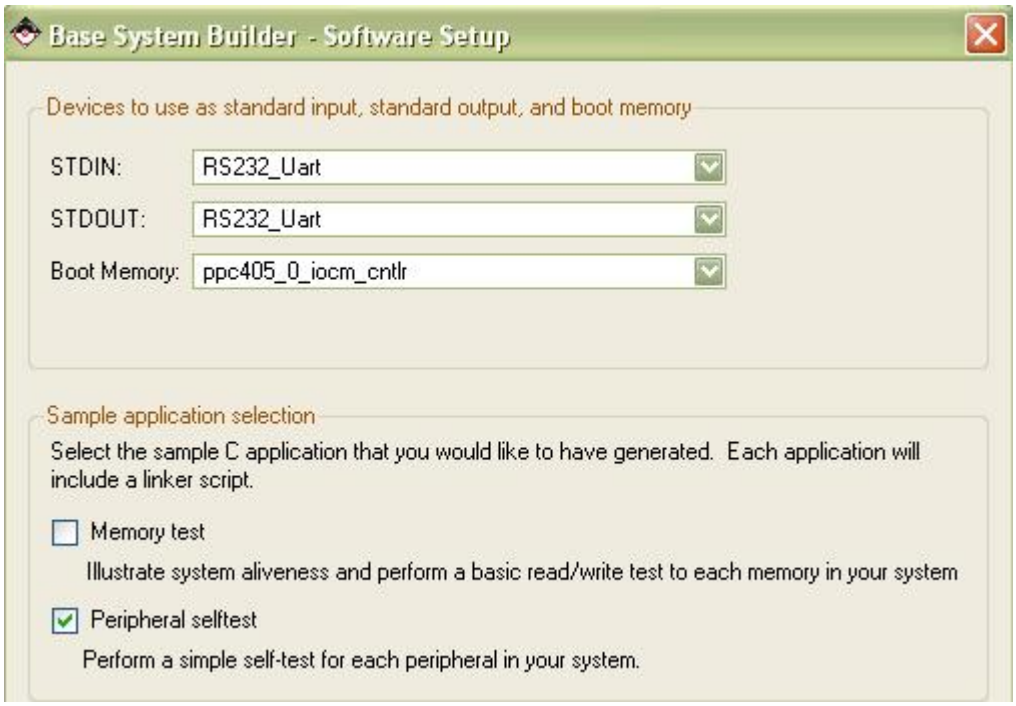
Click Next.

On the Cache Setup page, enable both cache selections as shown:



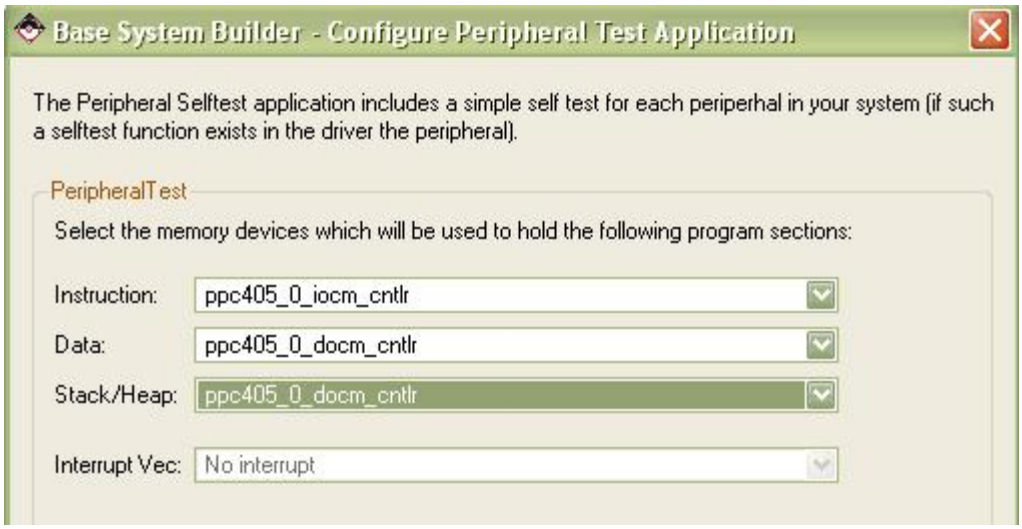
Click Next.

The Wizard will now ask if you want to create memory and peripheral test applications. Select the "Peripheral selftest" application, but do not select the "Memory test" application:



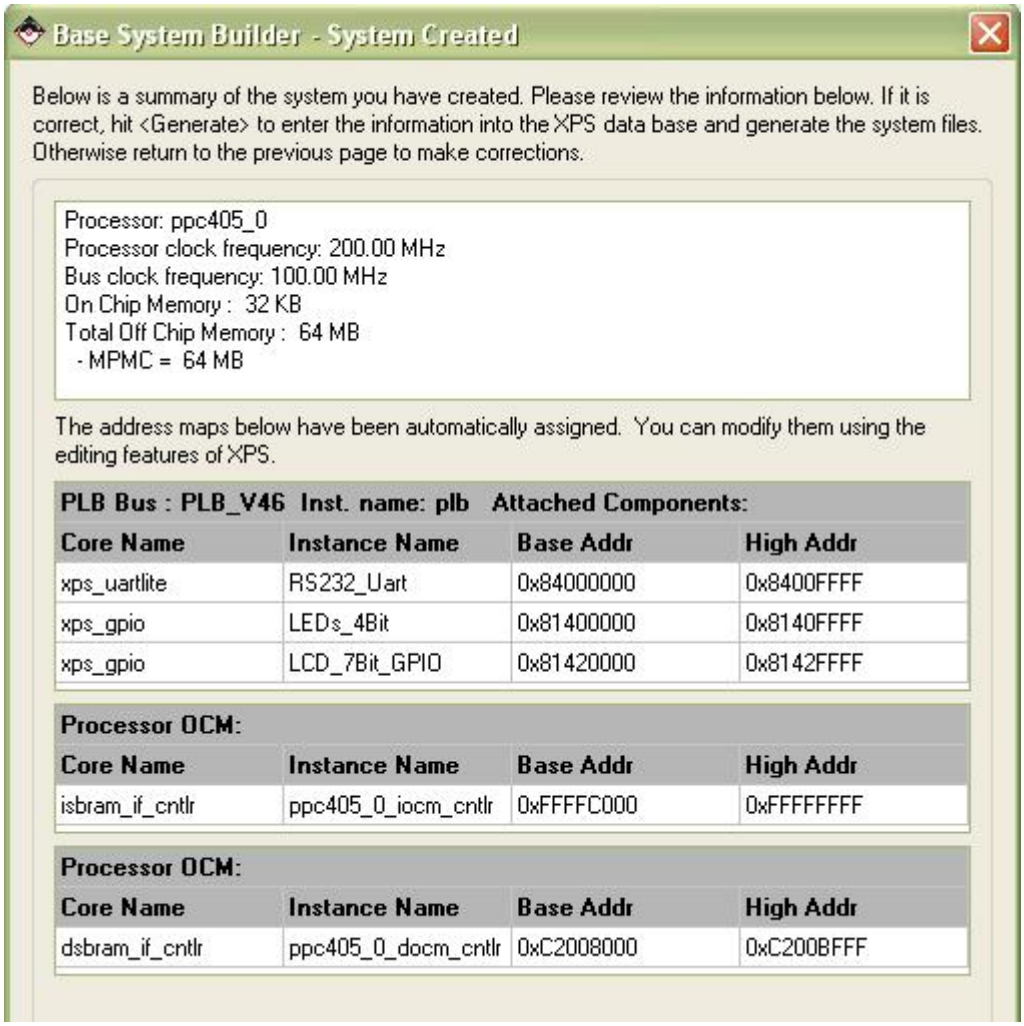
Click Next.

You will now be prompted for memory locations for Instruction, Data and Stack/Heap for the PeripheralTest application. Select ppc405_0_iocm_cntlr for the Instruction field, and ppc405_0_docm_cntlr for the Data and Stack/Heap fields as shown below:

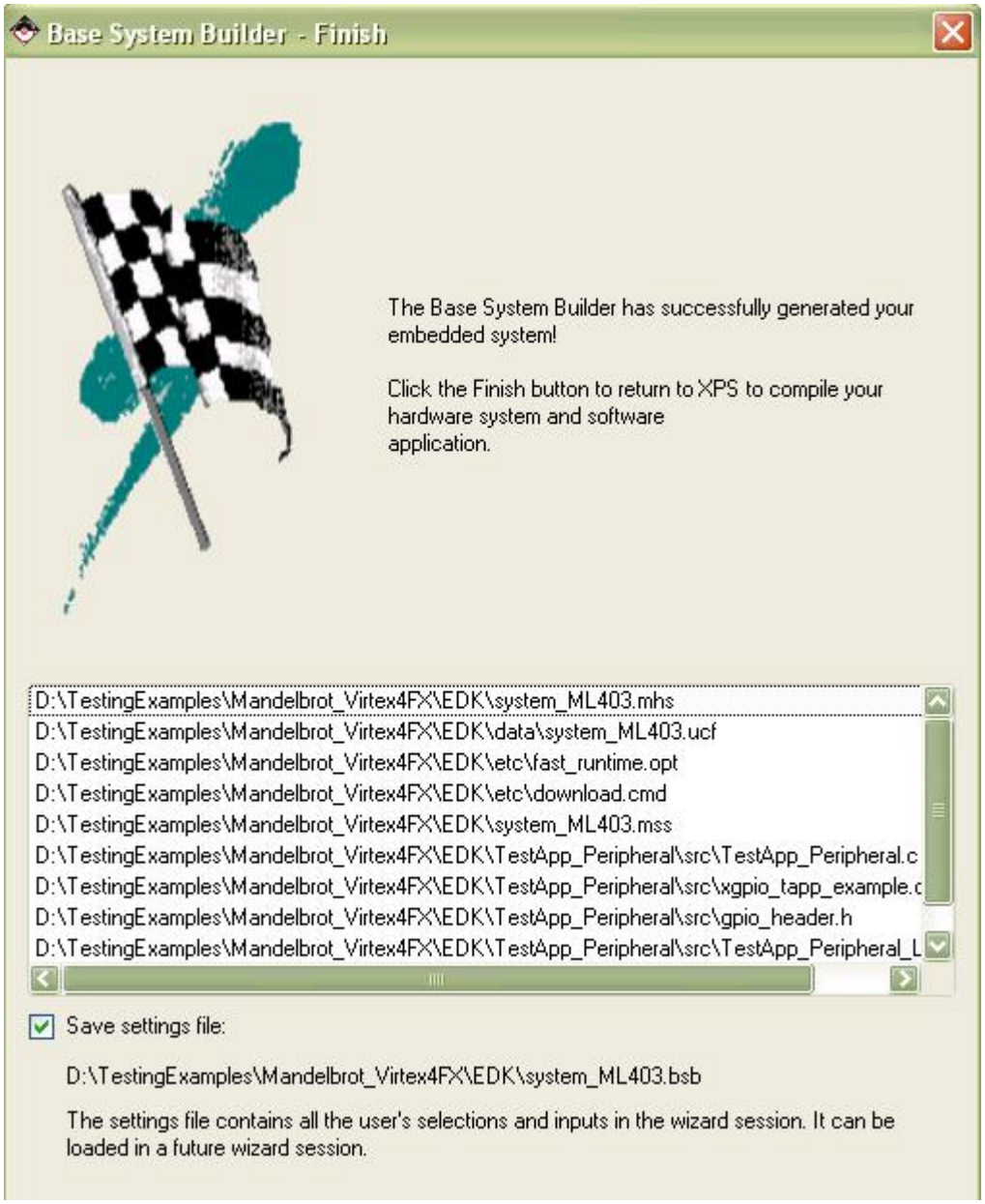


Click Next.

The Wizard will now display a summary of your platform selections:

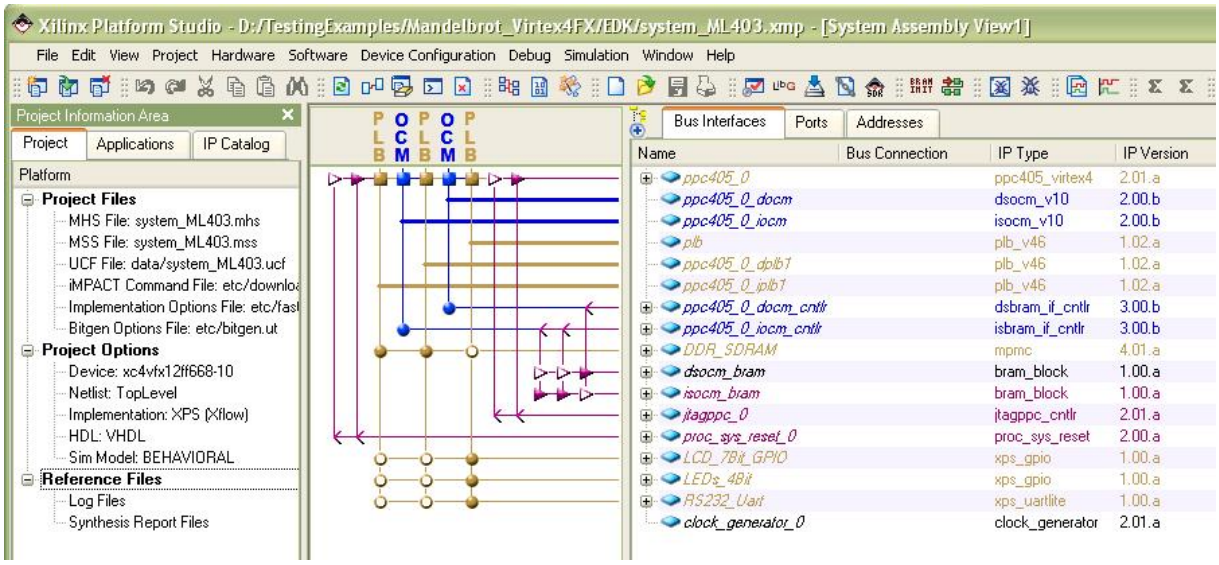


Click the Generate button to generate the platform with the specified configurations. After the platform has been generated, the Wizard will display a final page, and will give you the option of saving the platform settings to a .BSB file. This file can be used when creating new platforms with similar settings.



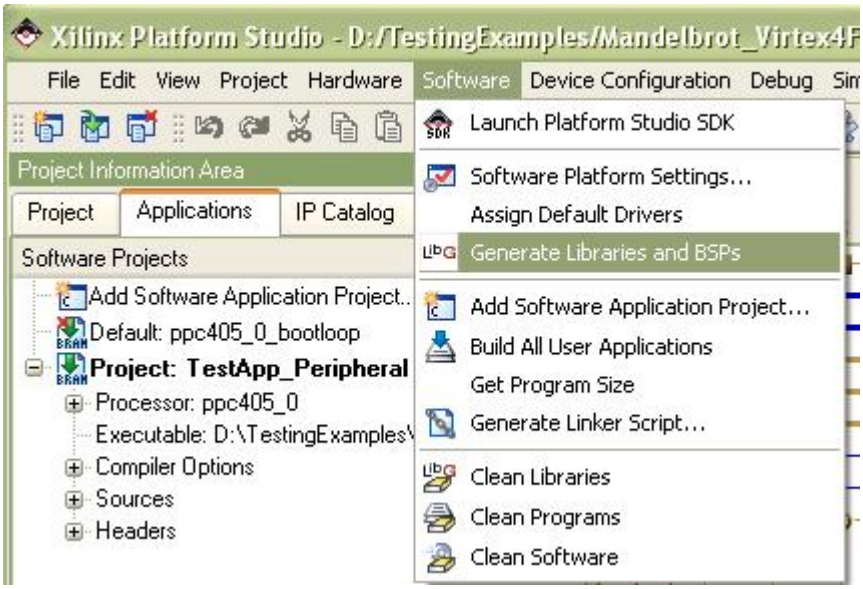
Click Finish to exit the Wizard.

The Platform Studio interface will now appear similar to the following:

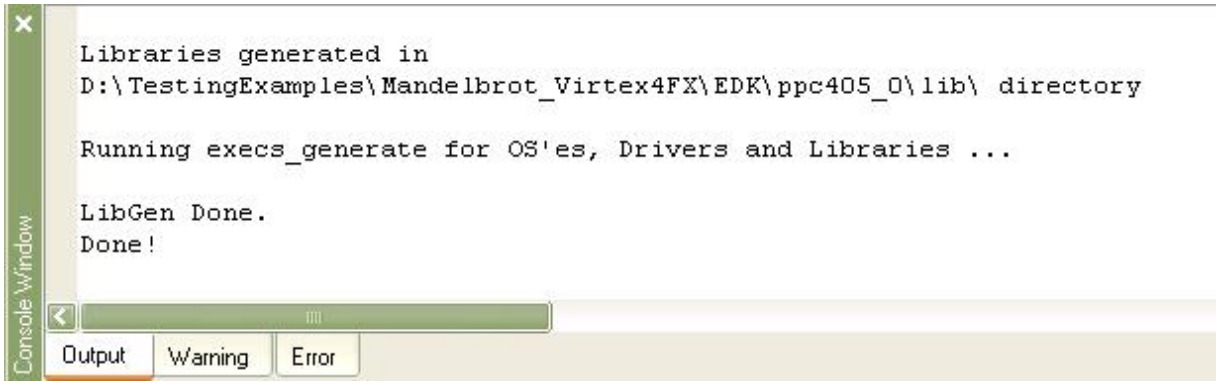


Building and Running the Peripheral Test

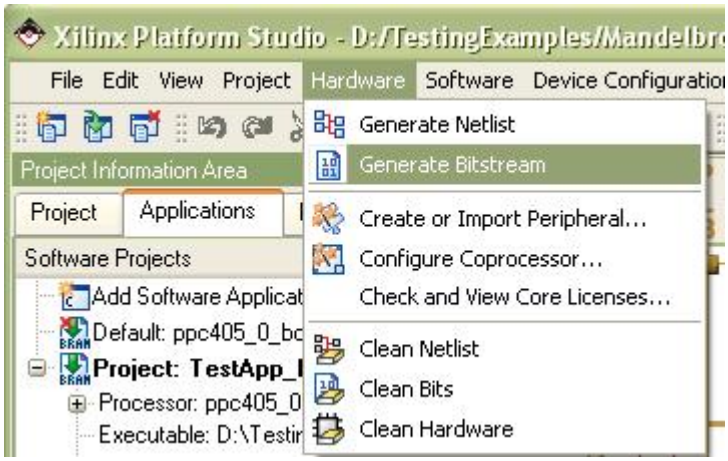
Before creating and building the Mandelbrot sample application, it is a good idea to do a quick test of the platform, using the Peripheral Selftest test application created by Base System Builder. To build the test application, you must first generate the PowerPC libraries, peripheral drivers, and other files needed for the software portion of the application. To do this, select the Generate Libraries and BSPs command from the Software menu as shown below:



When the libraries have been built, Platform Studio will display a message similar to the following:



Next, select the Generate Bitstream command from the Hardware menu. This command starts the synthesis and place-and-route process, resulting in a downloadable .BIT file.



After the bitstream generation has completed, make sure your JTAG cable is plugged in properly and the ML403 board is powered up. Select Download Bitstream from the Device Configuration menu as shown below:

When the FPGA has been successfully programmed, you will see a "Programming Complete"

message in the Platform Studio transcript, and you will see a small row of LEDs located light up in sequence on the lower right corner of board.

You have now verified the complete design flow and all needed hardware connections, from Platform Studio and Base System Builder to the ML403 board. In the next tutorial section, you will replace this test application with a new application representing the Mandelbrot fractal image generator.

See Also

Adding the Mandelbrot Hardware

1.3.8 Adding the Mandelbrot Hardware

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 8

In the previous step you used Xilinx Platform Studio and the Base System Builder to create a test application, ready to download and run on the ML403 board. This test was important because it established that all required peripherals, memories, etc. had been properly assembled, forming a base platform on which the Mandelbrot example can be implemented.

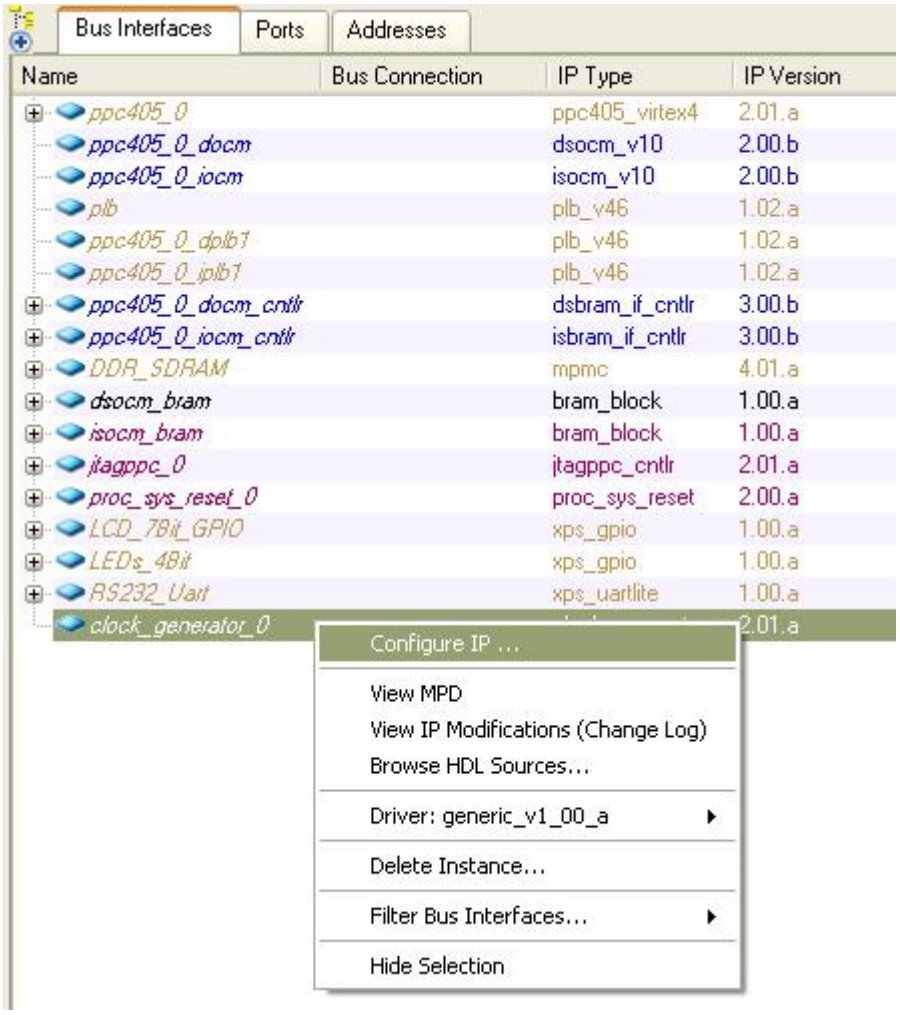
In the steps that remain, we will modify the base platform to:

- Configure clock generator component
- Configure the TFT display
- Add the Mandelbrot APU accelerator
- Add the Mandelbrot software application files
- Build the platform, including synthesizing the new cores
- Download and run the Mandelbrot application on the target board

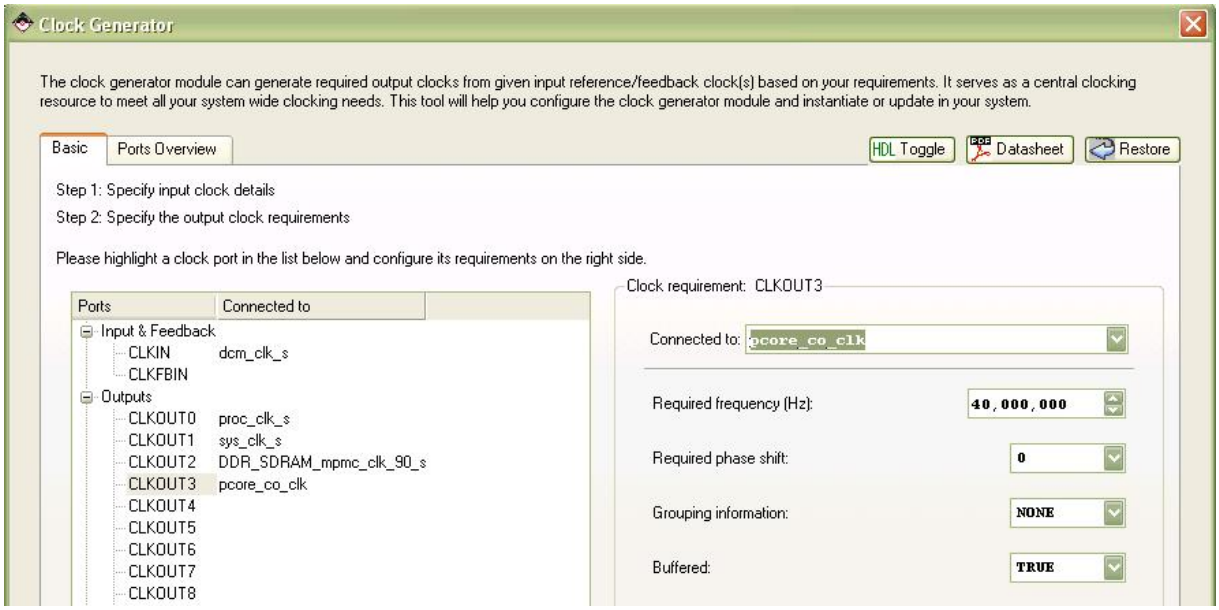
Configuring the Clock Generator

Our fractal image generator application requires three distinct clock sources, one for the PowerPC processor, one for the fabric control bus (FCB), and one for the hardware accelerator, which in this example runs at 40MHz. The TFT Controller needs a 25 MHz clock.

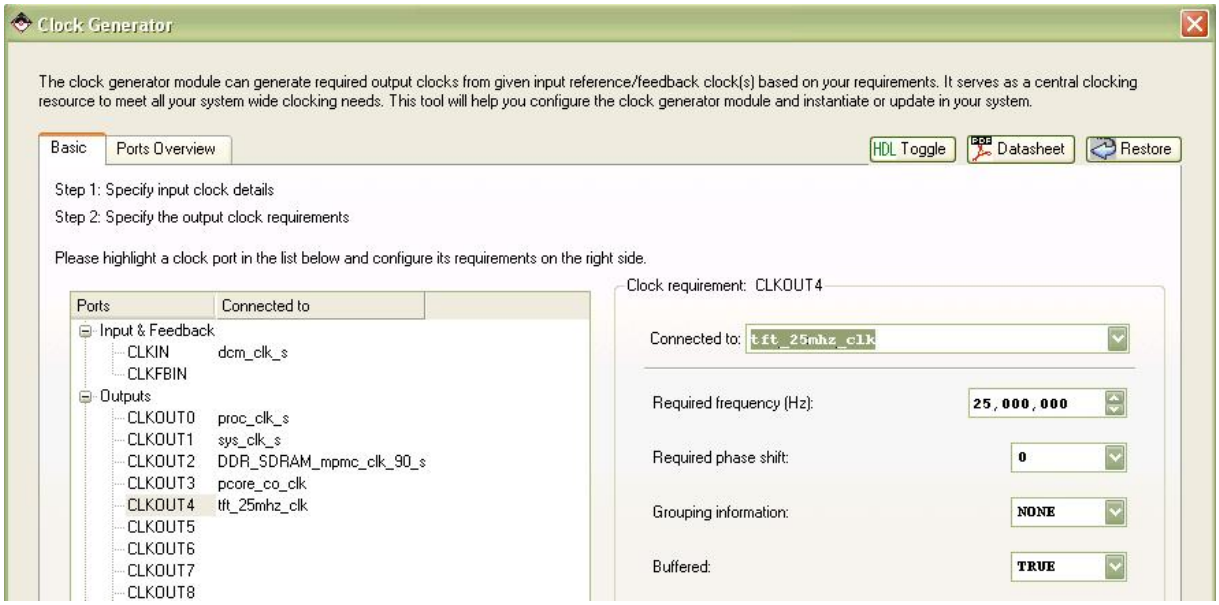
To configure the Clock Generator, right-click the clock_generator_0 to open the Configure IP option as shown below:



Add a new clock output in CLKOUT3, type in the name "pcore_co_clk", and frequency as "40000000" Hz as shown:

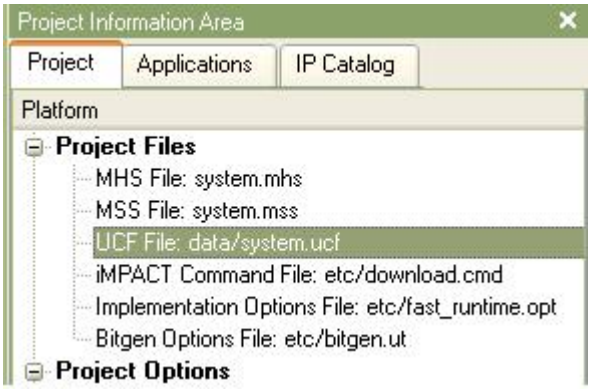


Add another clock output in CLKOUT4, type in the name "tft_25mhz_clk", and frequency as "25000000" Hz as shown:



Adding PLBV46_TFT_CNTLR Constraints

The PLBv46 TFT Controller is currently not included in the standard Xilinx IP Cores, so the PLBV46_TFT_CNTLR constaints need to be added manually to the .UCF file associated with the project (system.ucf). To edit this file, open the Project tab, and find the .UCF file listed under Project Files as shown below. Double-click on the system.ucf file entry.



Using the editing window that appears, add the following lines shown below to the end of the .UCF file:

```
#### Module plbv46_tft_cntlr constraints

NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<1> LOC = C5;  # VGA_B3
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<2>   LOC = C7;  # VGA_B4
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<3>   LOC = B7;  # VGA_B5
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<4>   LOC = G8;  # VGA_B6
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<5>   LOC = F8;  # VGA_B7
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<*> SLEW = FAST | DRIVE = 8;

NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<1> LOC = E4;  # VGA_G3
NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<2>   LOC = D3;  # VGA_G4
NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<3>   LOC = H7;  # VGA_G5
NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<4>   LOC = H8;  # VGA_G6
NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<5>   LOC = C1;  # VGA_G7
NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<*> SLEW = FAST | DRIVE = 8;

NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<1> LOC = C2; #VGA_R3
NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<2>   LOC = G7; #VGA_R4
NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<3>   LOC = F7; #VGA_R5
NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<4>   LOC = E5; #VGA_R6
NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<5>   LOC = E6; #VGA_R7
NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<*> SLEW = FAST | DRIVE = 8;

NET plbv46_tft_cntlr_0_TFT_LCD_CLK_pin LOC = AF8;
NET plbv46_tft_cntlr_0_TFT_LCD_CLK_pin IOSTANDARD = LVDCI_33 | SLEW = FAST |
DRIVE = 8;

NET plbv46_tft_cntlr_0_TFT_LCD_VSYNC_pin LOC = A8;
NET plbv46_tft_cntlr_0_TFT_LCD_VSYNC_pin SLEW = FAST | DRIVE = 8;

NET plbv46_tft_cntlr_0_TFT_LCD_HSYNC_pin LOC = C10;
NET plbv46_tft_cntlr_0_TFT_LCD_HSYNC_pin SLEW = FAST | DRIVE = 8;
```

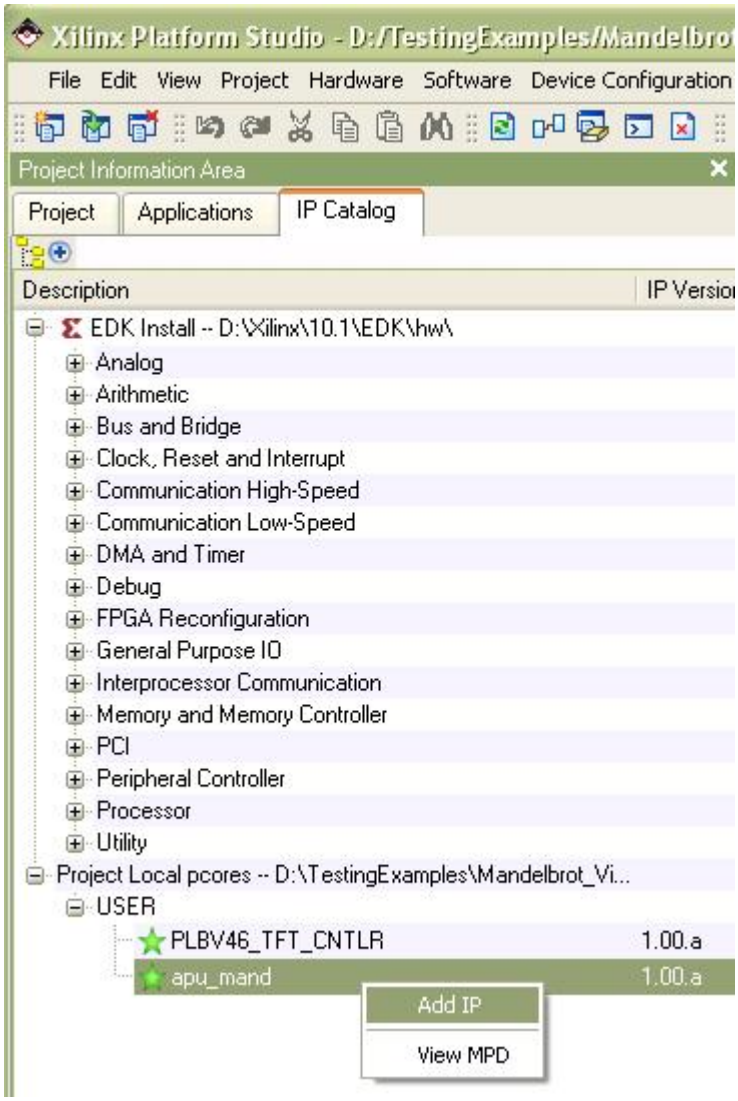
The modified .UCF file should look as shown below:

```
177  Net fpga_0_DDR_SDRAM_DDR_DQ<31> IOSTANDARD = SSTL2_II;
178  Net fpga_0_DDR_SDRAM_DDR_Clk_pin LOC=A10;
179  Net fpga_0_DDR_SDRAM_DDR_Clk_pin IOSTANDARD = DIFF_SSTL2_II;
180  Net fpga_0_DDR_SDRAM_DDR_Clk_n_pin LOC=B10;
181  Net fpga_0_DDR_SDRAM_DDR_Clk_n_pin IOSTANDARD = DIFF_SSTL2_II;
182
183  #### Module plbv46_tft_cntlr constraints
184
185  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<1> LOC = C5;  # VGA_B3
186  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<2>   LOC = C7;  # VGA_B4
187  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<3>   LOC = B7;  # VGA_B5
188  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<4>   LOC = G8;  # VGA_B6
189  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<5>   LOC = F8;  # VGA_B7
190  NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<*> SLEW = FAST | DRIVE = 8;
191
192  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<1> LOC = E4;  # VGA_G3
193  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<2>   LOC = D3;  # VGA_G4
194  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<3>   LOC = H7;  # VGA_G5
195  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<4>   LOC = H8;  # VGA_G6
196  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<5>   LOC = C1;  # VGA_G7
197  NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<*> SLEW = FAST | DRIVE = 8;
198
199  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<1> LOC = C2; #VGA_R3
200  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<2>   LOC = G7; #VGA_R4
201  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<3>   LOC = F7; #VGA_R5
202  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<4>   LOC = E5; #VGA_R6
203  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<5>   LOC = E6; #VGA_R7
204  NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<*> SLEW = FAST | DRIVE = 8;
205
206  NET plbv46_tft_cntlr_0_TFT_LCD_CLK_pin LOC = AF8;
207  NET plbv46_tft_cntlr_0_TFT_LCD_CLK_pin IOSTANDARD = LVDCI_33 | SLEW = FAST | DRIVE = 8;
208
209  NET plbv46_tft_cntlr_0_TFT_LCD_VSYNC_pin LOC = A8;
210  NET plbv46_tft_cntlr_0_TFT_LCD_VSYNC_pin SLEW = FAST | DRIVE = 8;
211
212  NET plbv46_tft_cntlr_0_TFT_LCD_HSYNC_pin LOC = C10;
213  NET plbv46_tft_cntlr_0_TFT_LCD_HSYNC_pin SLEW = FAST | DRIVE = 8;
```

Save the .UCF file using the File Save menu command, then close the editing window.

Adding the Mandelbrot Accelerator Core

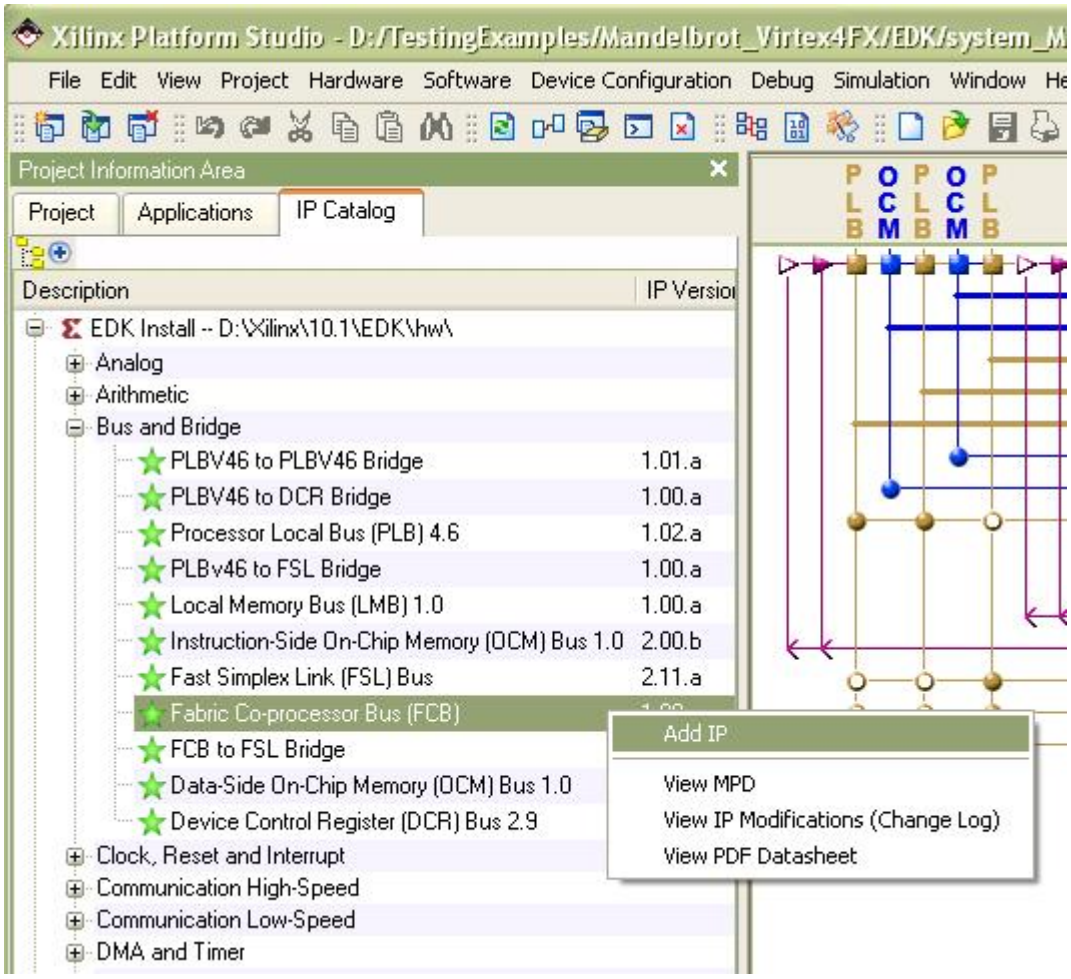
To add the Mandelbrot fractal image generator core as a peripheral, select the IP Catalog tab and look for the category titled "Project Local pcores". Under USER directory you will find the two cores that were created (copied to) the EDK/pcores directory of your project. Add the apu_mand core by clicking the right mouse button as shown below:



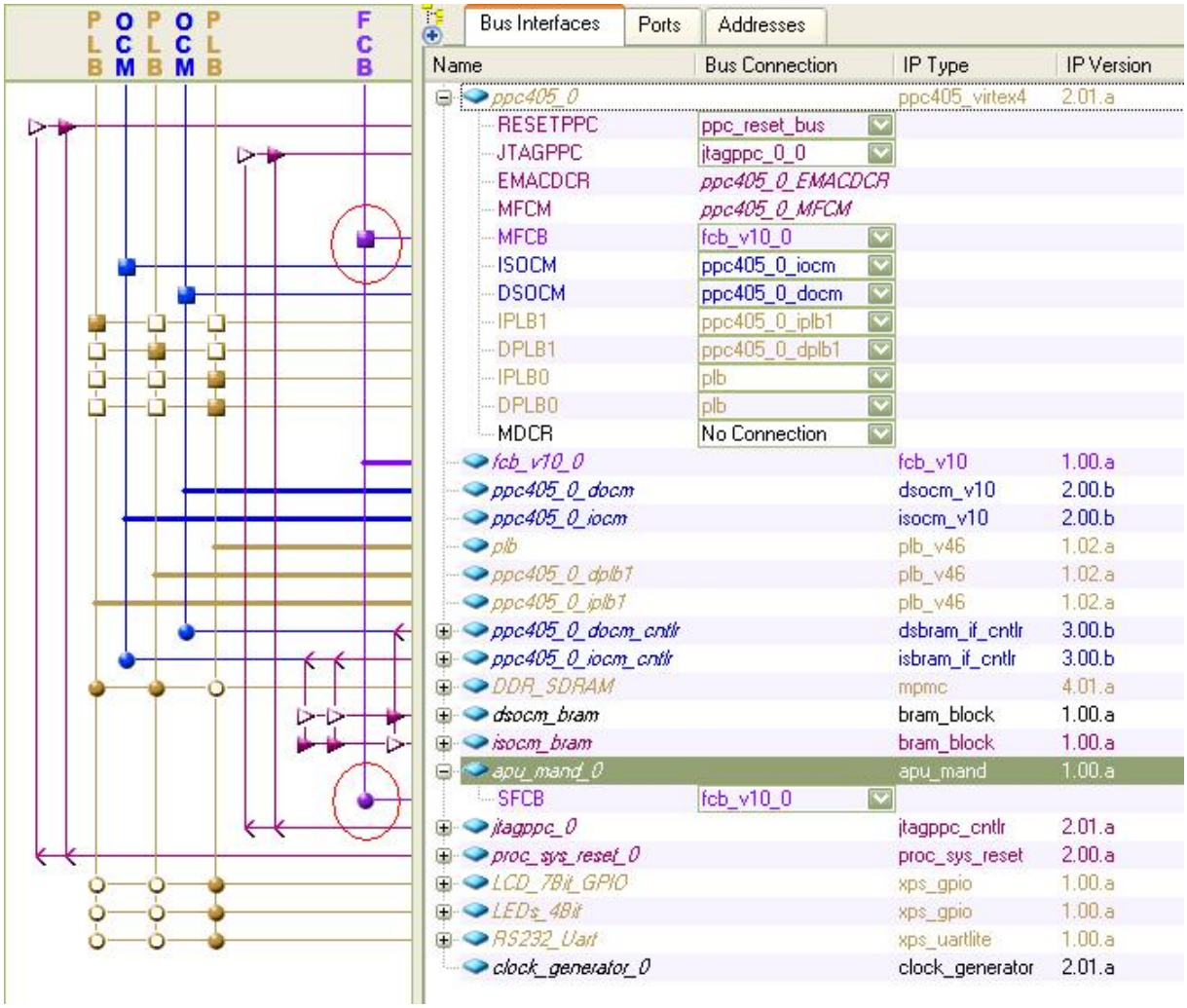
This will add the core to the project as a peripheral.

Adding Fabric Co-processor Bus

To connect the peripheral to the PowerPC via the APU interface, you will also need to add a Fabric Co_processor Bus (FCB) to the system. To add this core, select the Bus category and find the "Fabric Co_processor Bus" item. Add the FCB to your systemby clicking the right mouse button as shown below:



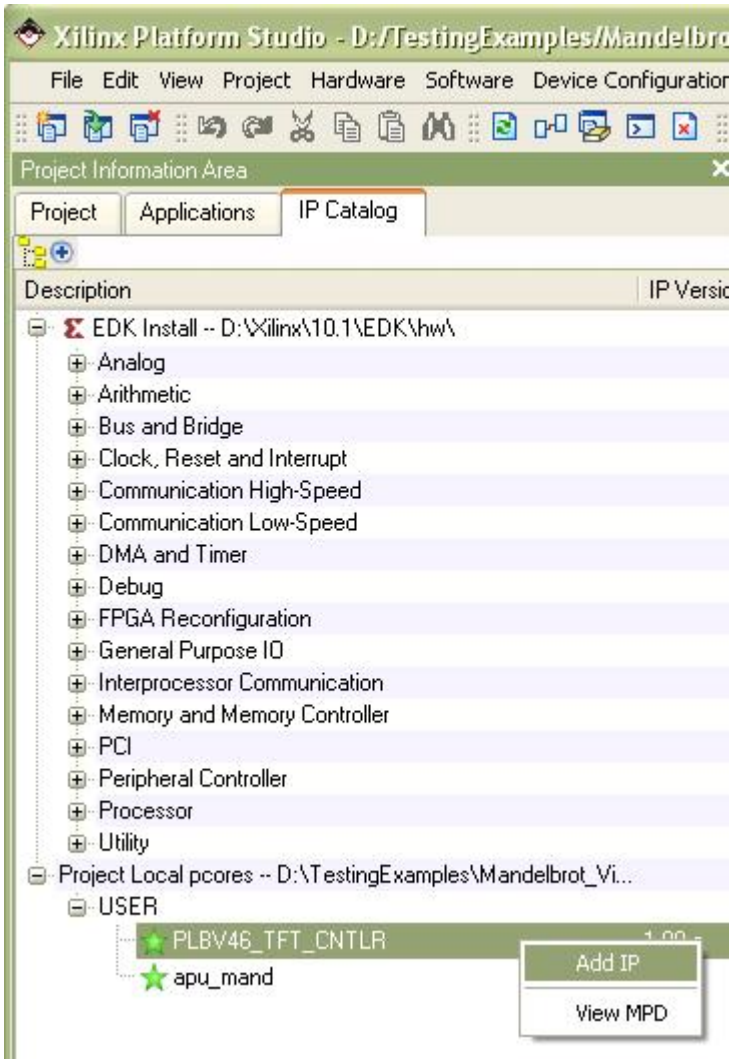
After you have added the FCB, it will appear in the Platform Studio connections window as shown below. Use the mouse pointer to select and connect the MFCB port of the ppc405_0 to the FCB, by clicking on the square connection point, and then connect the apu_mand_0 peripheral (SFCB connection) to the FCB as shown below (and indicated by the the red circle):



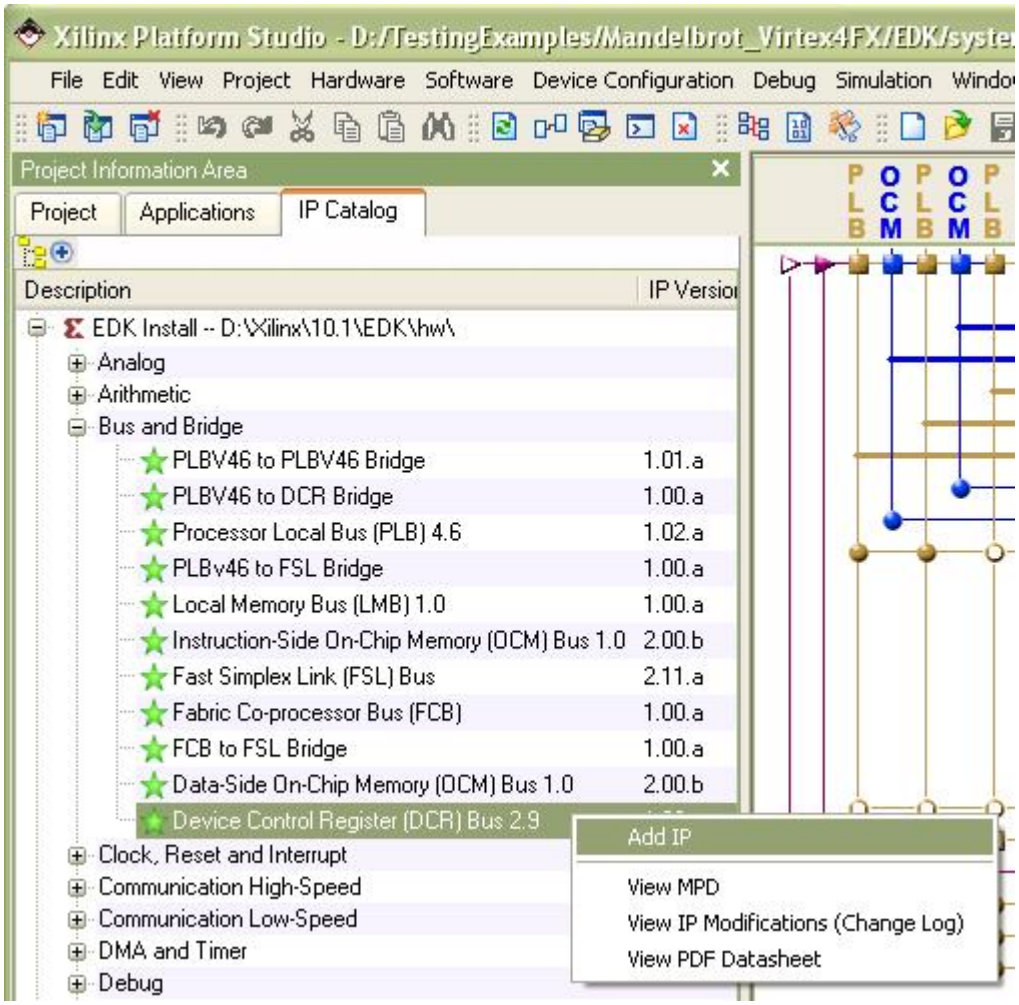
The apu_mand peripheral is now connected via the APU to the PowerPC.

Adding the PLBV46 TFT Controller Core

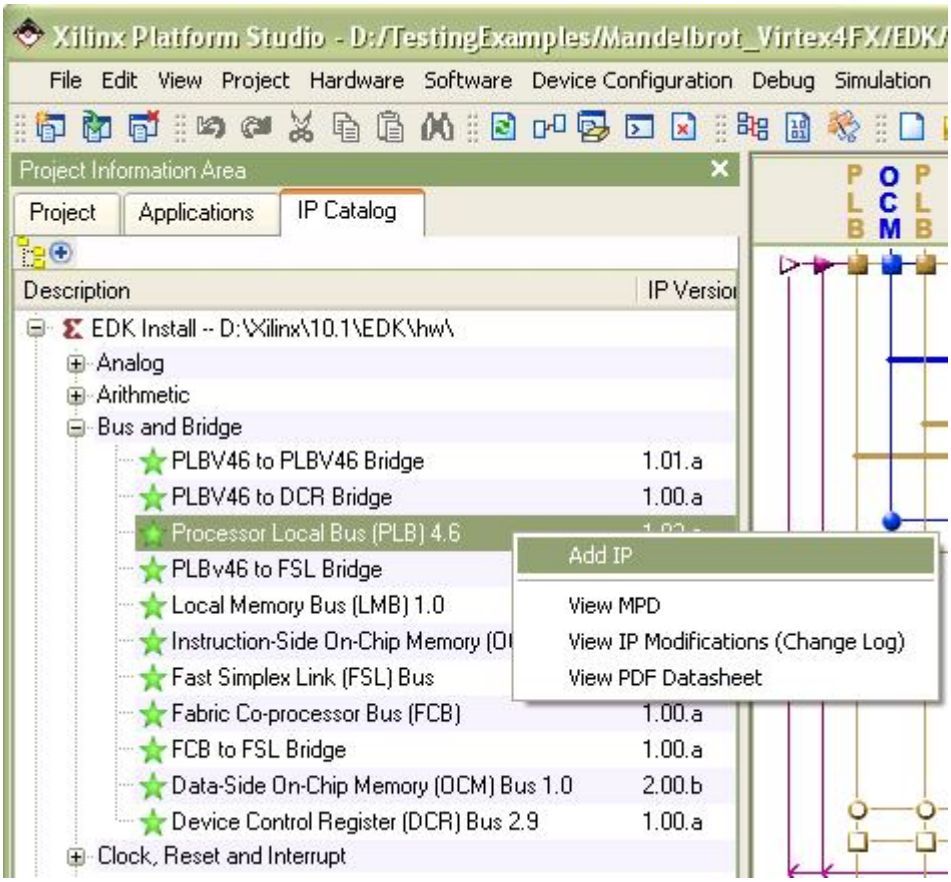
Next, add the PLBV46_TFT_CNTLR by clicking the right mouse button as shown below:



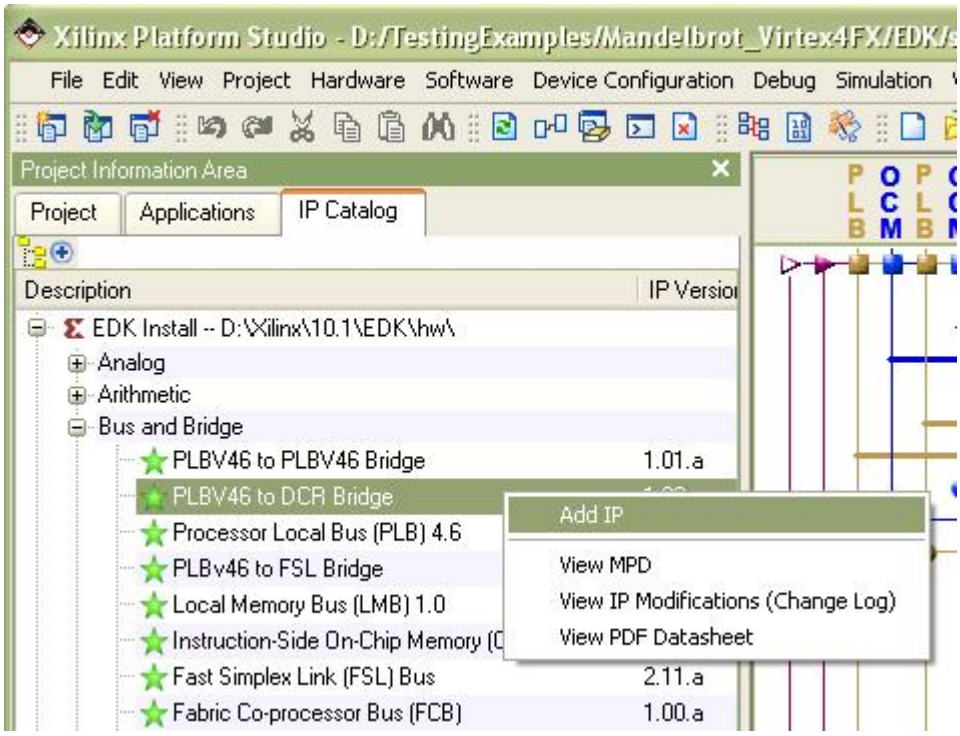
The PLBV46_TFT_CNTLR controls the Thin Film Transistor LCD Display, which gives us the graphical output of the computation results. The PLBV46_TFT_CNTLR has a SDCR bus interface, a MPLB and an SPLB bus interface. We need to add a DCR bus and an additional PLB bus to connect the PLBV46_TFT_CNTLR properly. Choose Device Control Register (DCR) Bus from the Bus and Bridge Category, and add it by right-clicking as shown below:



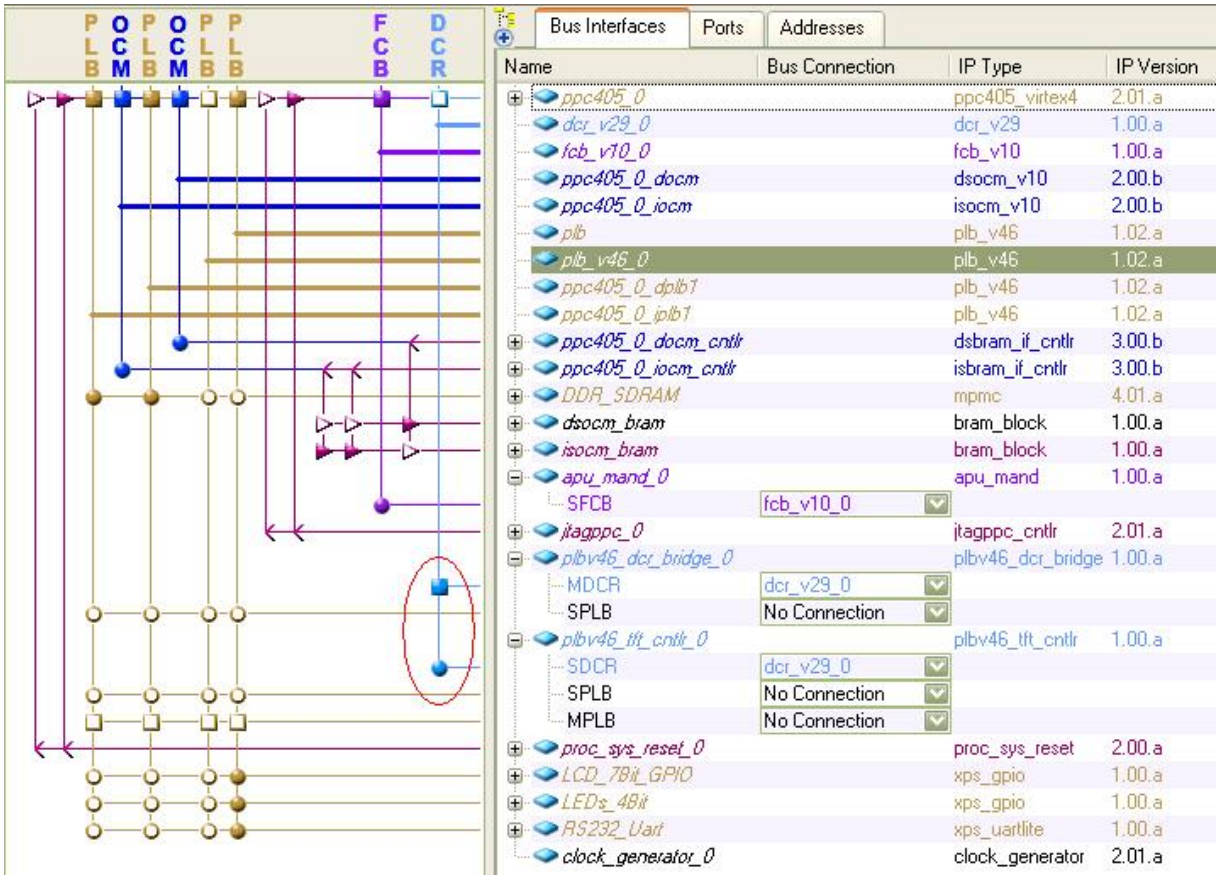
Add a Processor Local Bus (PLB) as shown below:



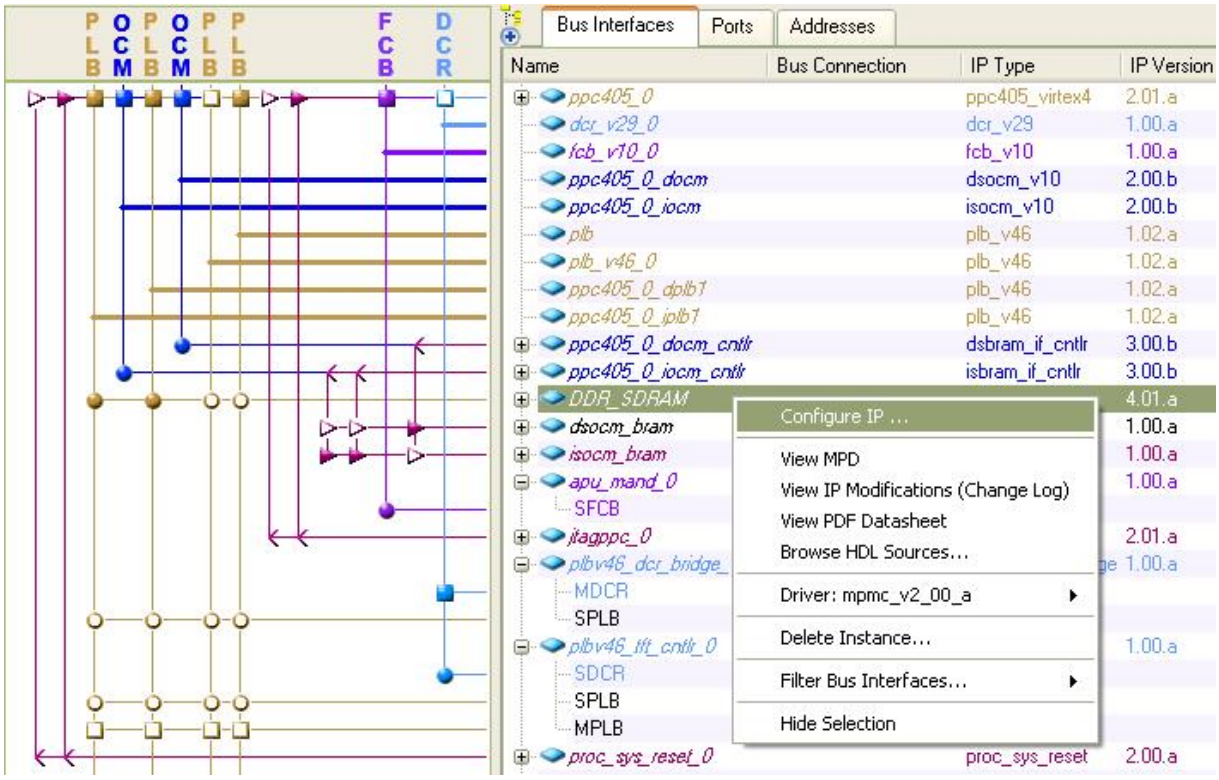
A PLBV46 to DCR Bridge is needed to connect the DCR and the PLB together. Add the bridge as shown below:



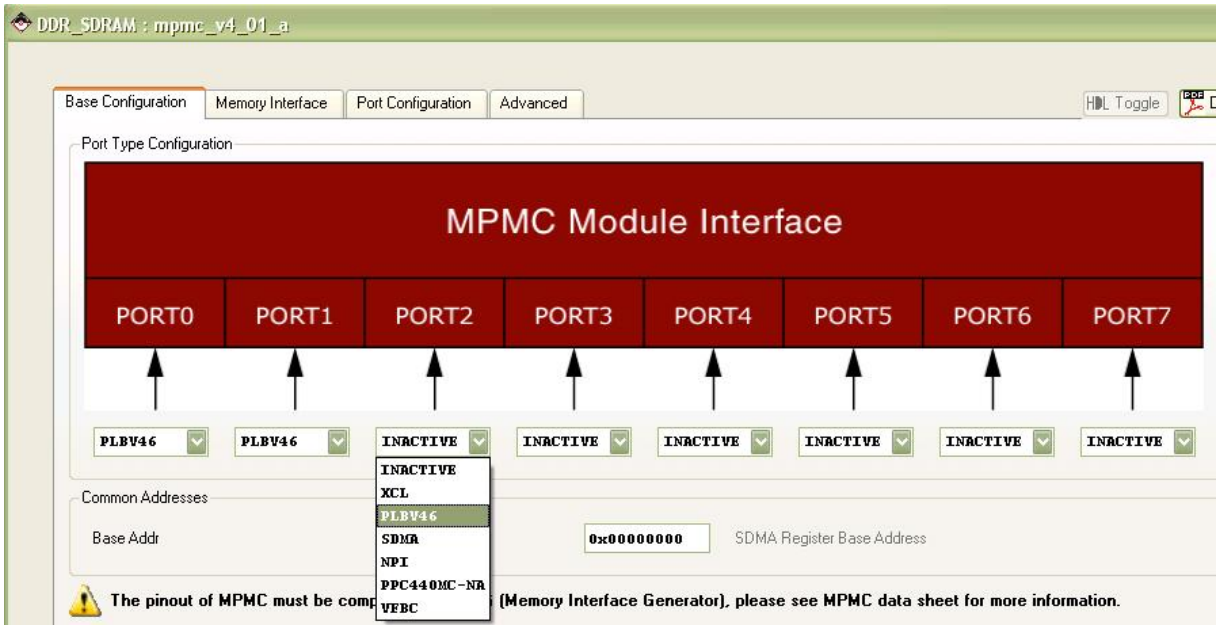
Now, connect the bus interfaces. Connect the MDCR bus of the plbv46_dcr_bridge_0 and the SDCR bus of the plbv46_tft_cntlr_0 by mouse clicking the connection points as shown below (indicated in the red oval):



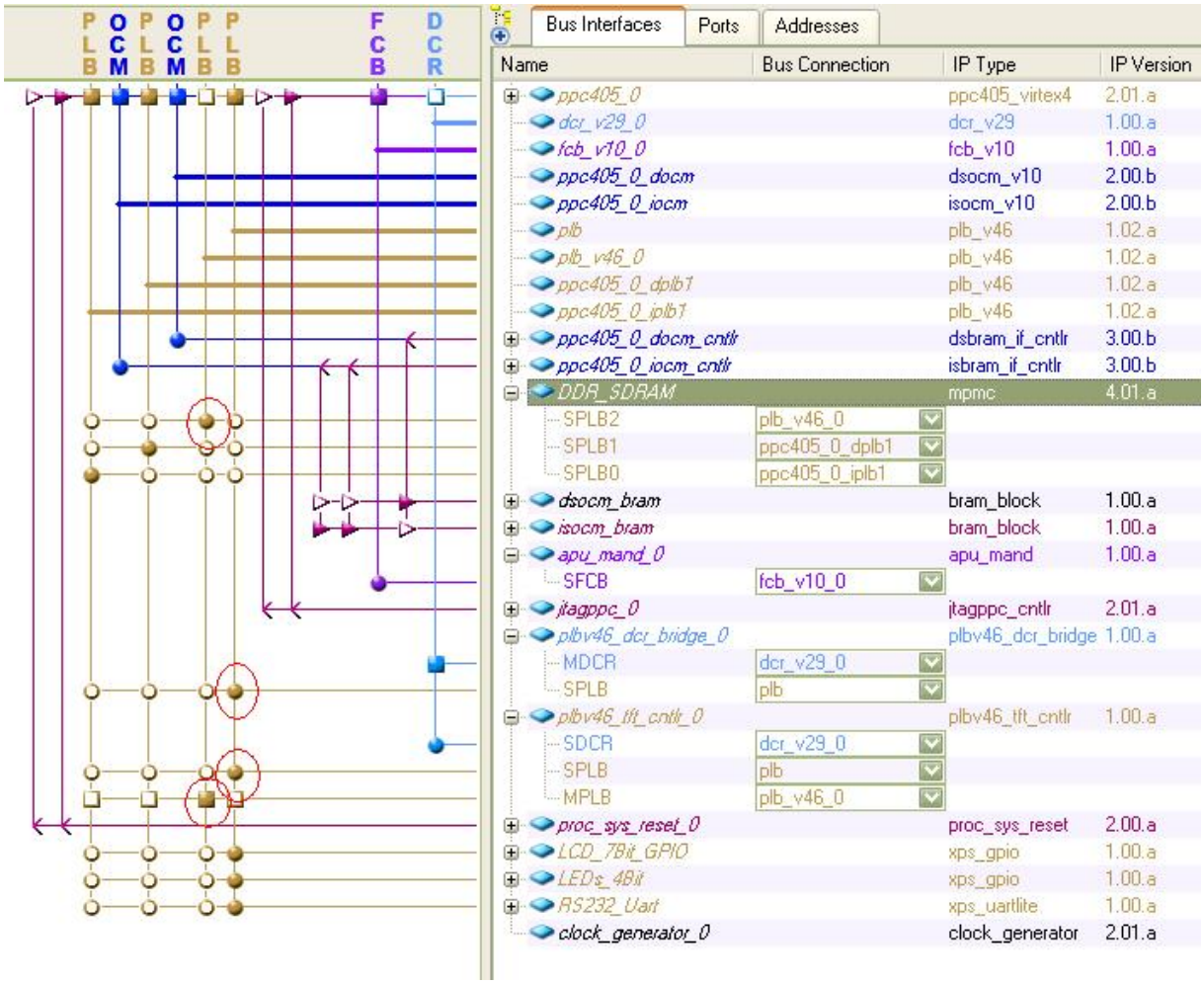
The MPLB of plbv46_tft_cntlr_0 needs to connect to the DDR_SDRAM through a separate SPLB port. To do this, open the Configure IP Dialogue of the DDR_SDRAM:



Change the Port Type Configuration of Port 2 from INACTIVE to PLBV46 as shown below. This will add another PLBV46 port to the DDR_SDRAM.



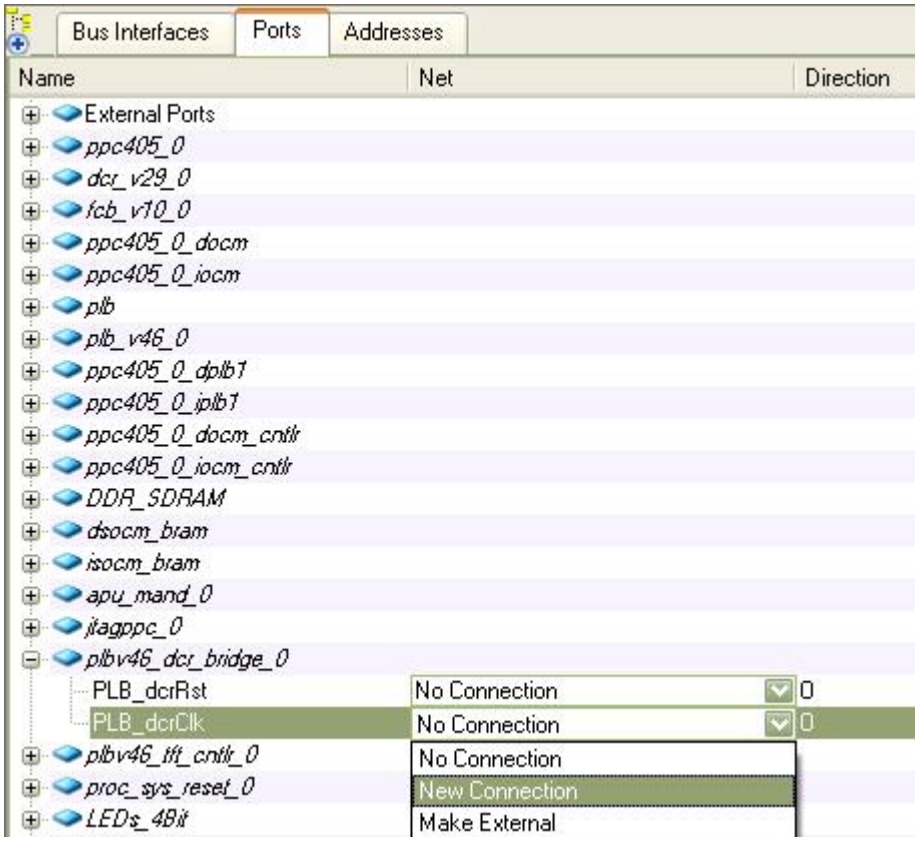
Then, connect the MPLB of the plbv46_tft_cntlr_0 to the newly added PLB, also conect the SPLB2 of the DDR_SDRAM to the same PLB. Connect the SPLB of the plbv46_tft_cntlr_0 and the SPLB of the plbv46_dcr_bridge_0 to the shared PLB as shown below (as indicated in red circles):



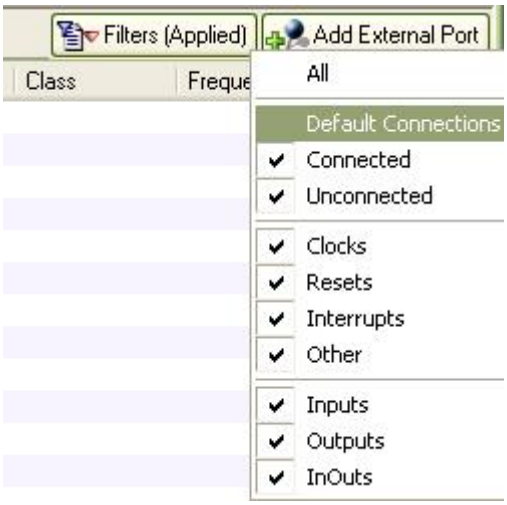
Connecting the Peripheral Clock and Reset Signals

To do this, switch to the Ports Tab in the System Assembly View Window.

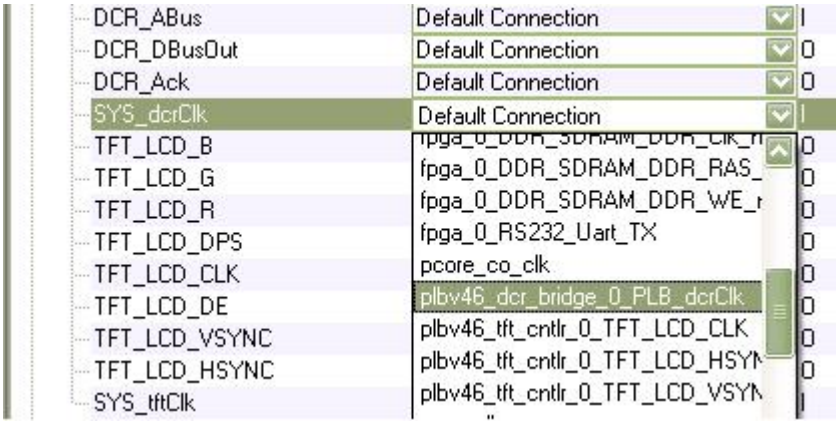
To connect the DCR clock of the PLBv46 DCR Bridge, make a new connection to the PLB_dcrClk as shown below:



Connect this clock to the SYS_dcrClk port of the plbv46_tft_cntlr. To view the port, change the port filter to show default connections.



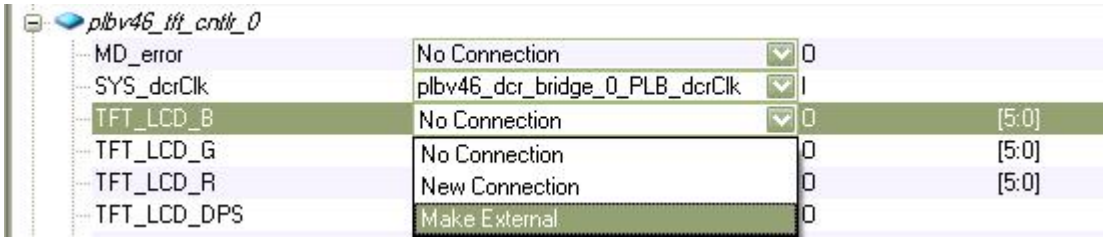
A long list of ports will show up. Find the SYS_dcrClk port under the plbv46_tft_cntlr and connect it to the plbv46_dcr_bridge_0_PLB_dcrClk port as shown below:



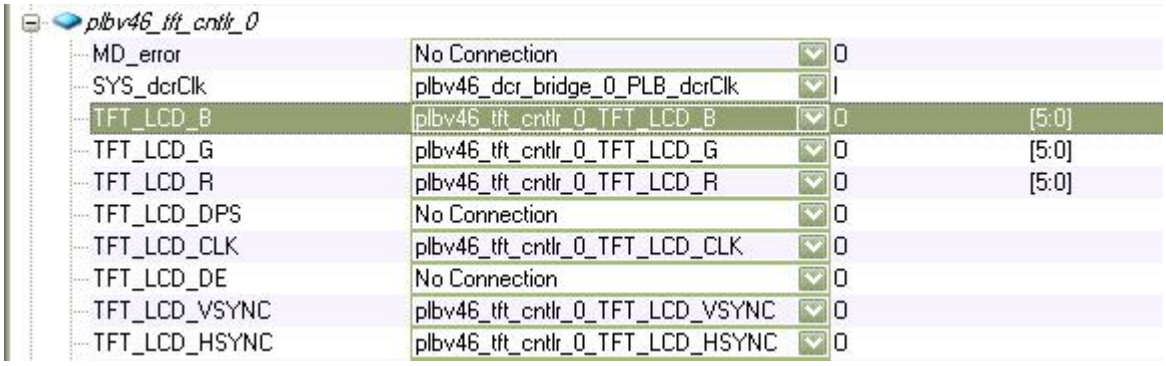
After this step is done, change back the port filter setting to avoid showing default ports.

Next, connect the following 6 ports of the plbv46_tft_cntlr to outside of the FPGA by select the Make External for each port as shown below:

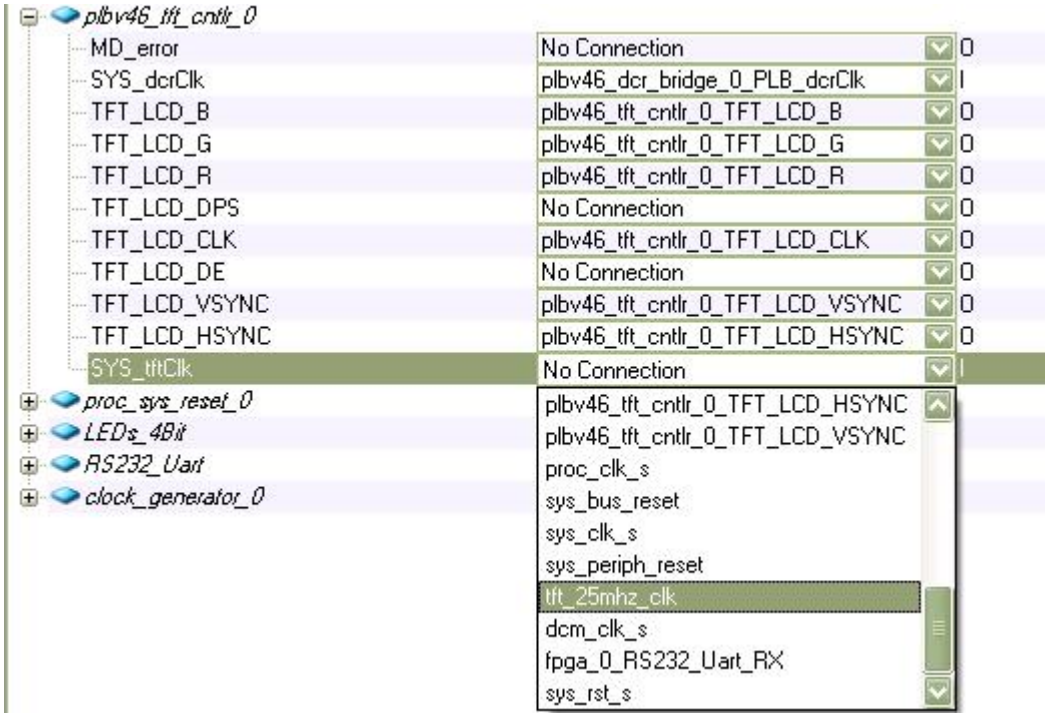
TFT_LCD_B
TFT_LCD_G
TFT_LCD_R
TFT_LCD_CLK
TFT_LCD_HSYNC
TFT_LCD_VSYNC



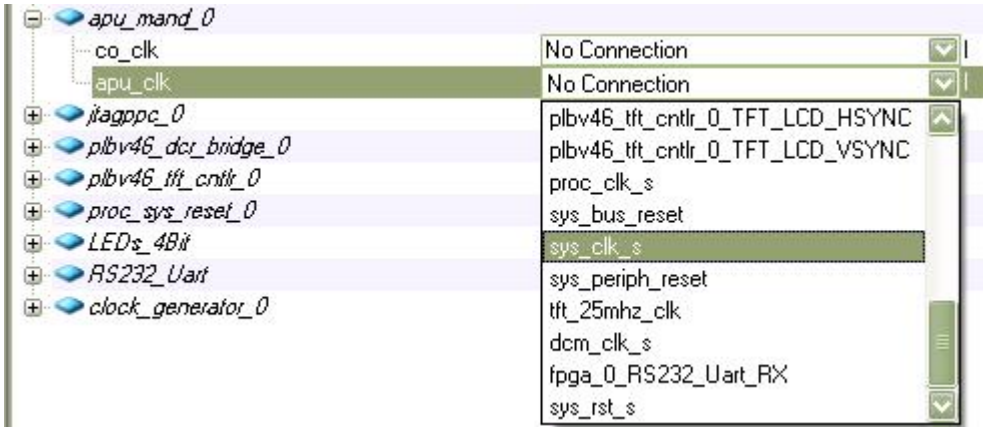
The resulting port connection should be like this:



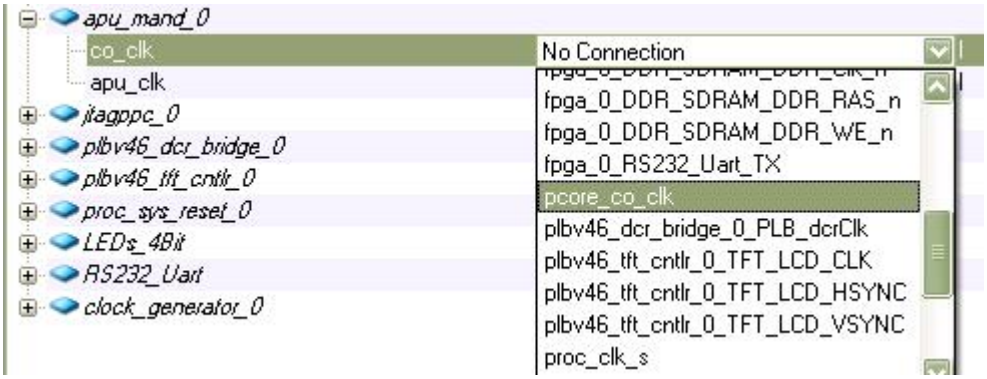
Connect the SYS_tftClk port to the tft_25mhz_clk from the Clock Generator as shown below:



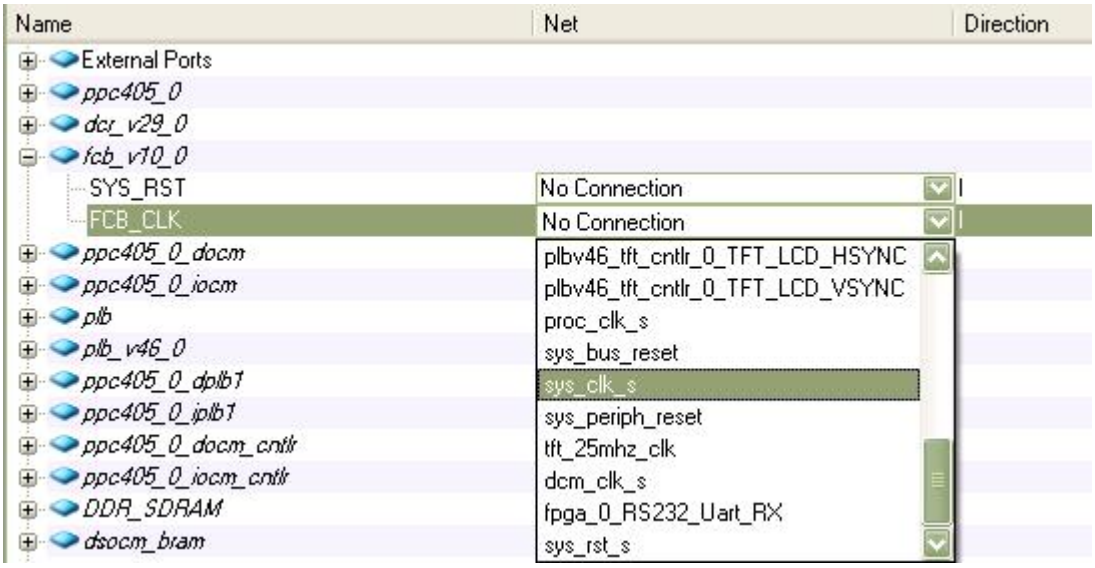
The next step is to connect the two apu_mand_0 clock signals. To do this, change the apu_clk entry to sys_clk_s as shown below:



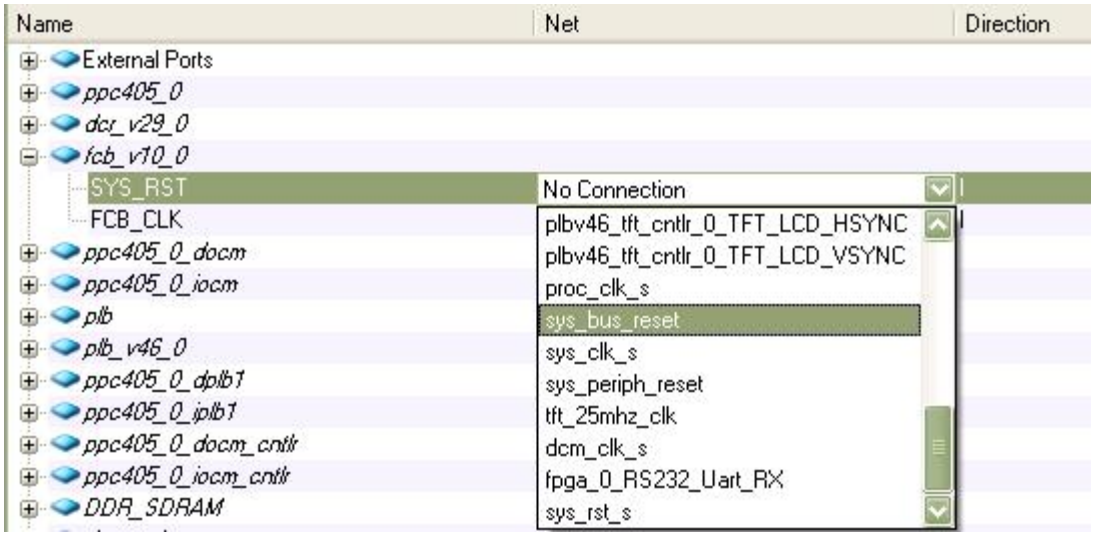
Now change the co_clk entry to pcore_co_clk as shown below:



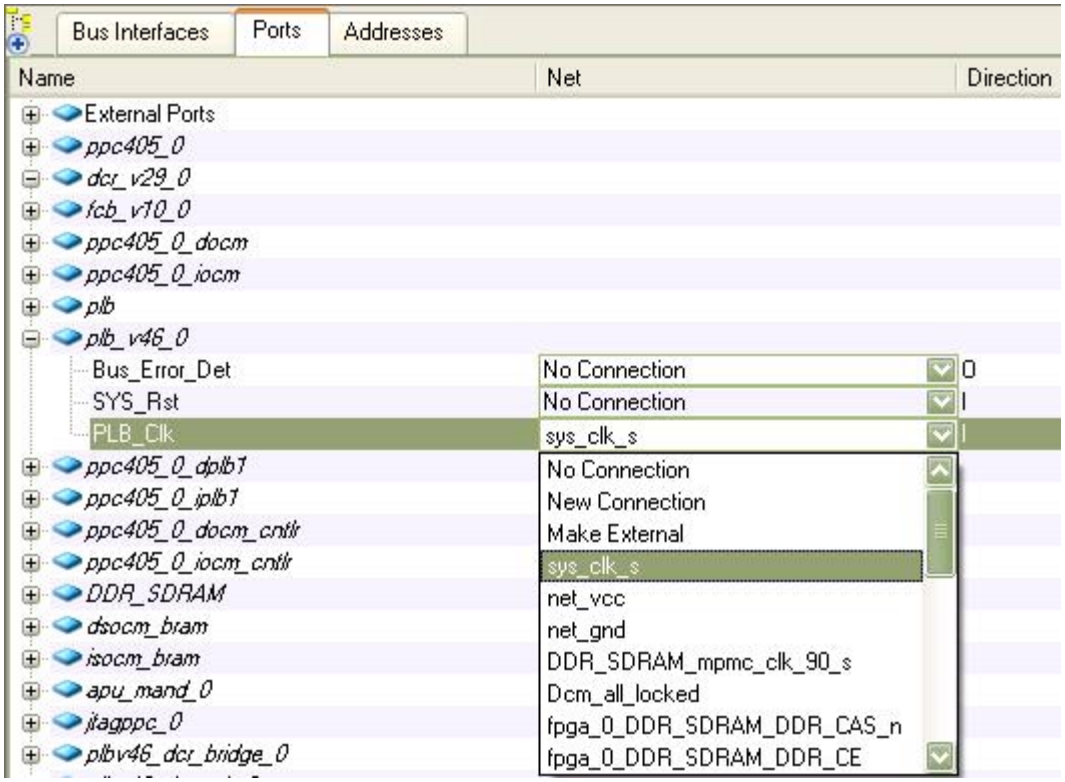
Connect the FCB_CLK signal of the fcb_v10_0 peripheral to sys_clk_s:



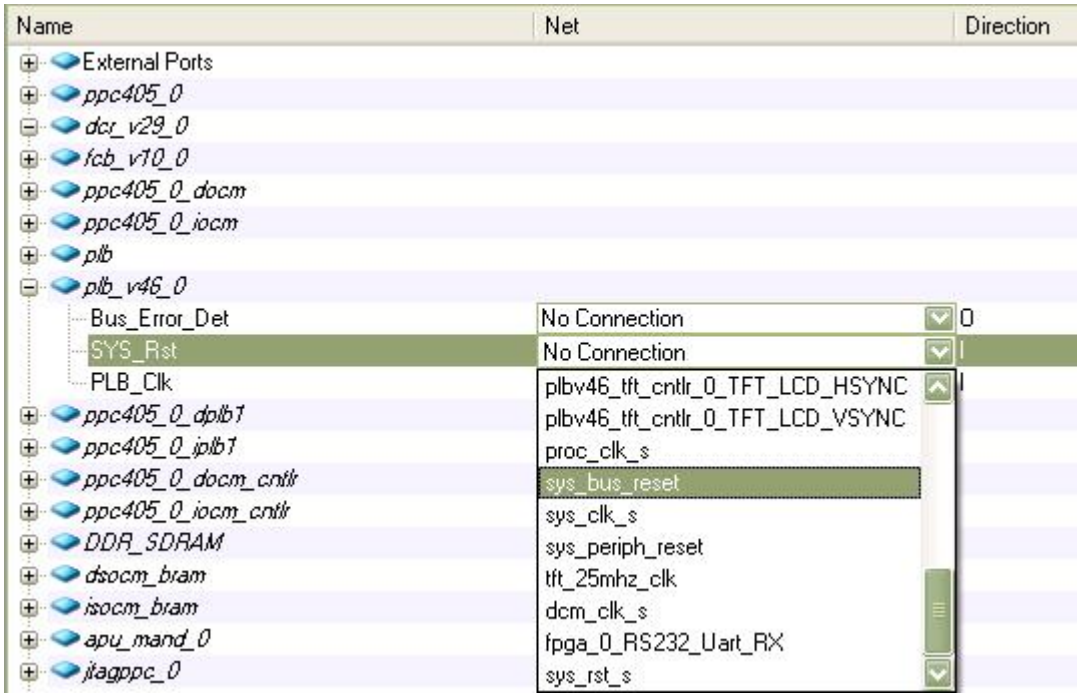
And connect the SYS_RST signal of fcb_v10_0 to sys_bus_reset:



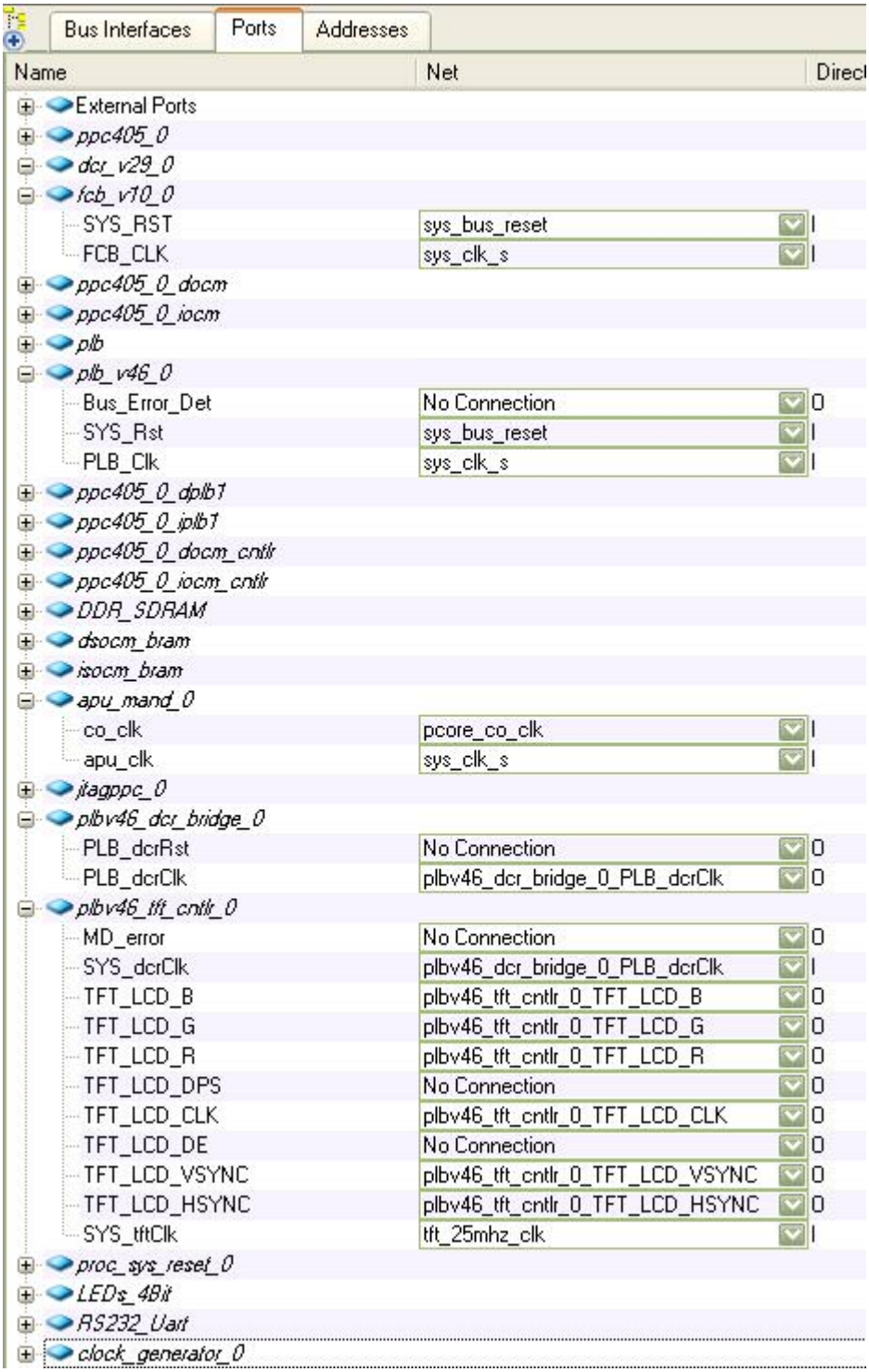
Connect the PLB_Clk signal of the plb_v46_0 peripheral to sys_clk_s:



And connect the SYS_Rst signal of plb_v46_0 to sys_bus_reset:

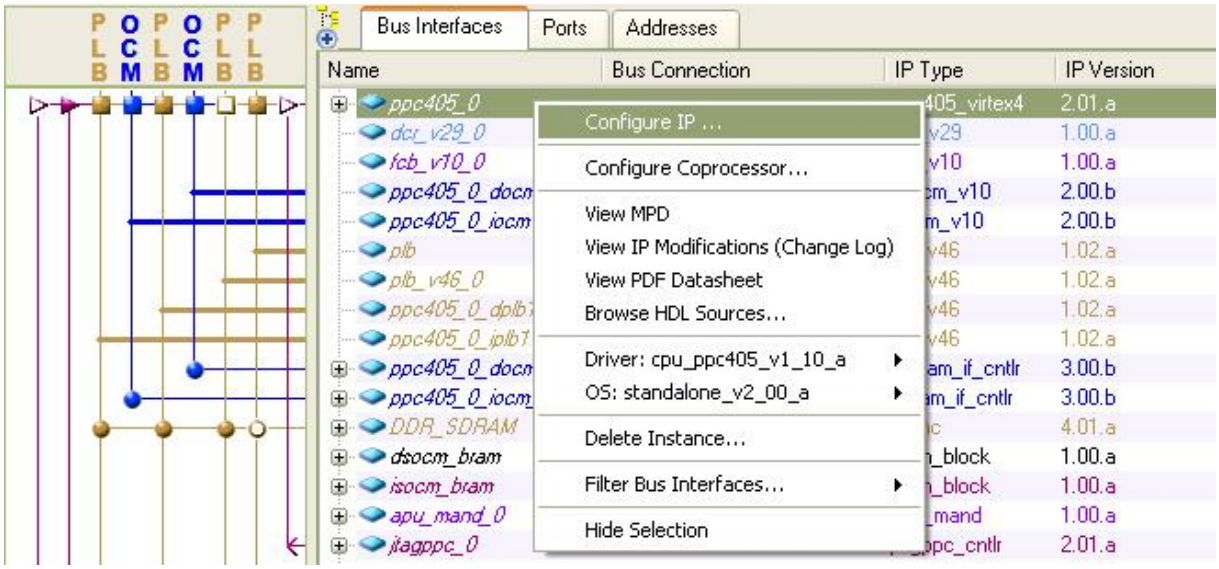


The port view of your project should now appear similar to the following:



Modifying the C_APU_CONTROL Parameter

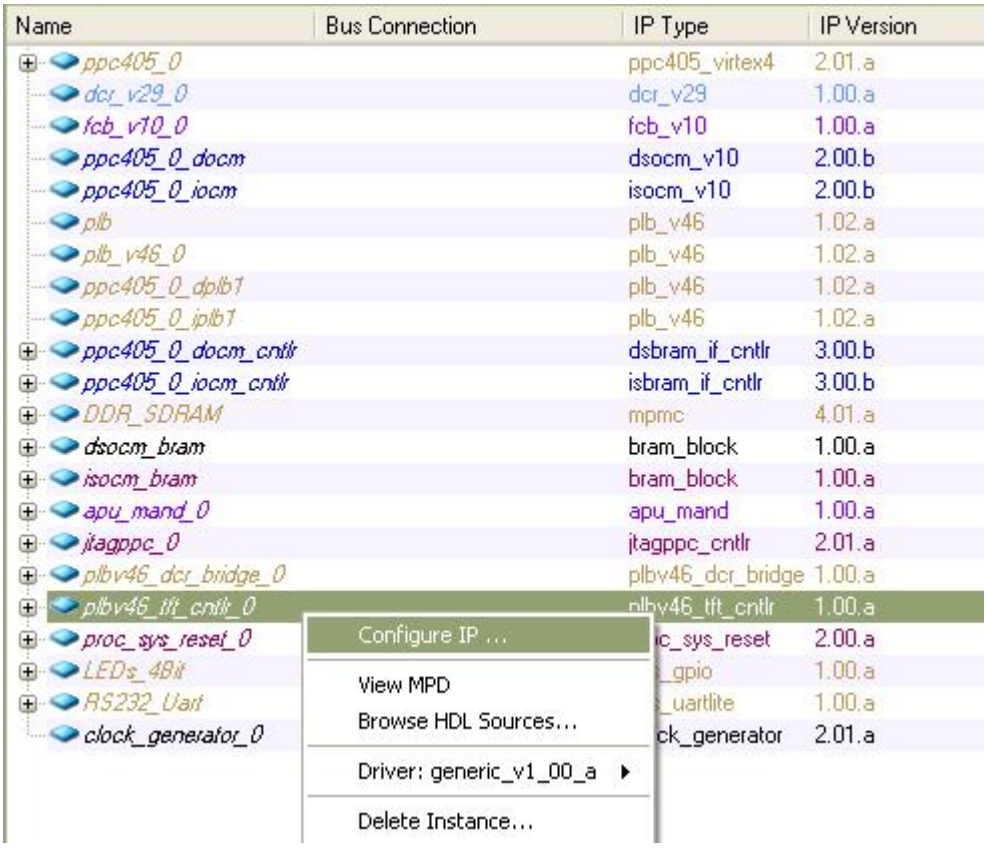
The C_APU_CONTROL parameter is used to enable the APU interface, which in this example is used to transmit data between the PowerPC processor and the hardware accelerator. This parameter can be viewed and edited in the Configure IP Dialogue as shown below.



Switch to the APU Tab and change the APU Controller Configuration Register Initial Value to 0b0000000000000001 as shown below:

Modifying the TFT Base Address Parameter

The C_DEFAULT_TFT_BASE_ADDR parameter is used to set the starting address of the TFT image memory. This is very important to have the TFT LCD display properly, and you will need to set the corresponding value in the software code. This parameter can be viewed and edited in the Configure IP Dialogue as shown below.



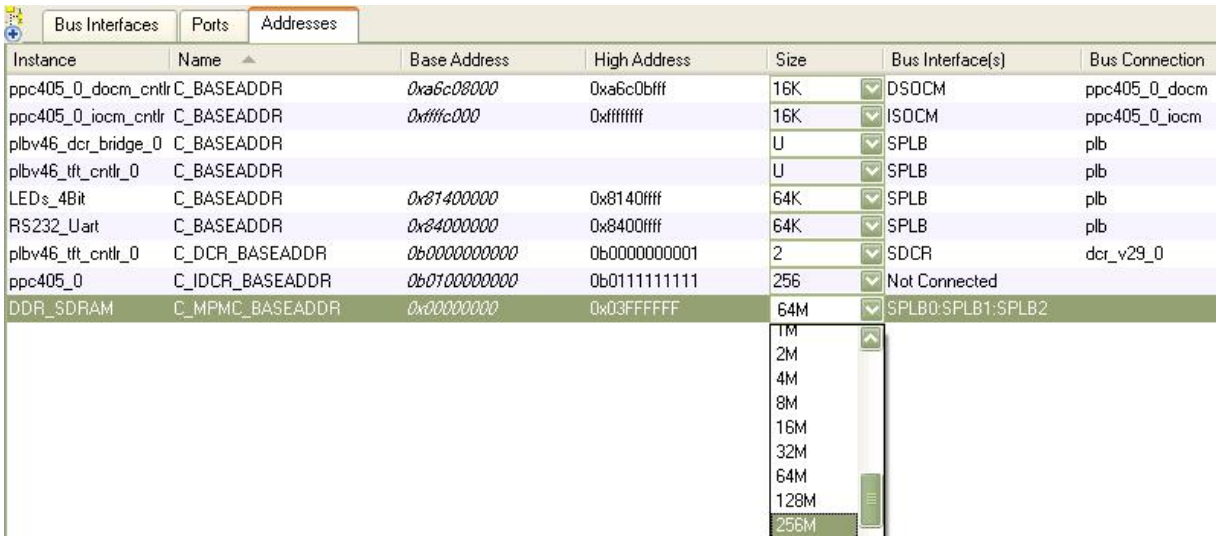
Change the C_DEFAULT_TFT_BASE_ADDR value to 0b00001000000 (11 bits altogether) as shown below:

Clink OK to save the change.

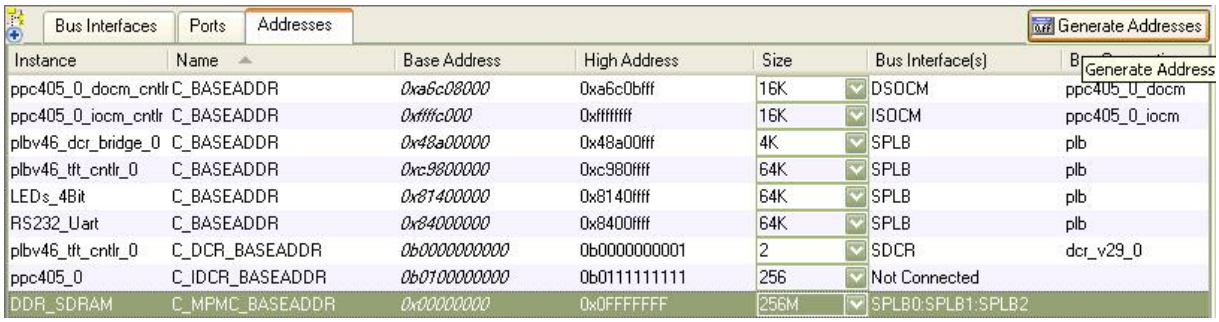
Generate Addresses

Next step is to generate addresses for the memory related modules in EDK. Switch to the Addresses Tab of the System Assembly View Window.

First, change the size of the DDR_SDRAM from 64MB to 256MB. The actual size of the DDR_SDRAM is 64MB. The purpose of mapping it to upper address space is to use the uncached memory space for the TFT image memory.



Next, click the Generate Addresses button on the upper right corner to let EDK assign addresses for the modules as shown below:



An error message might show up when generating the addresses:

ERROR:MDT - C_IDCR_BASEADDRof ppc405_0 has no high address in MHS

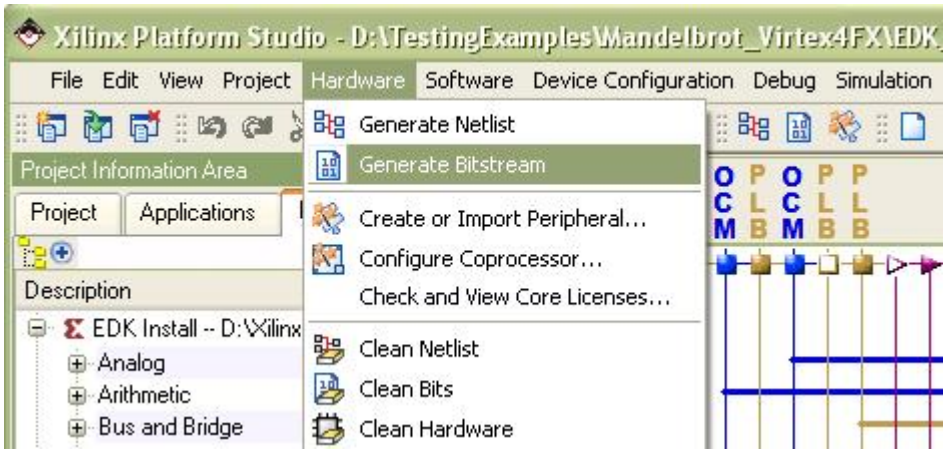
If this happens, add the following line to the ppc405_virtex4 paremeters, in the system.mhs file:

PARAMETER C_IDCR_HIGHADDR = 0b0111111111

Before buiding the hardware, check the system.mhs file to make sure that ppc405_virtex4 comes before all other instances. If not, move it to the top. The instance order might affect the hardware synthesis for some reason.

Now, build the hardware by choosing the Hardware -> Generate Bitstream menu. The synthesis, place-and-route and bitstream generation process will take a few minutes to complete depending on

your PC.

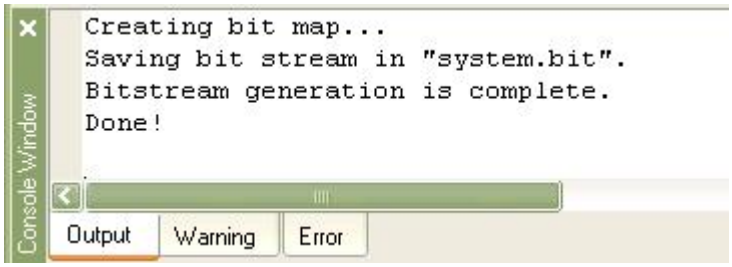


During the building process, an error message might pop up due to a known issue with the EDK software:

FATAL_ERROR: GuiUtilities:Gq_Application.c:590:1.20

If this happens, just close the EDK window, and then re-open it and restart the building process. Clearing the output window frequently may help. Please refer to Xilinx Answers Database for a possible solution.

The process is done. A file "system.bit" is created.



The hardware side of the application, including the APU interface and Mandelbrot fractal image generator core, is now ready for use. In the next tutorial section you will set up the software side of the application.

See Also

Adding the Software Application Files

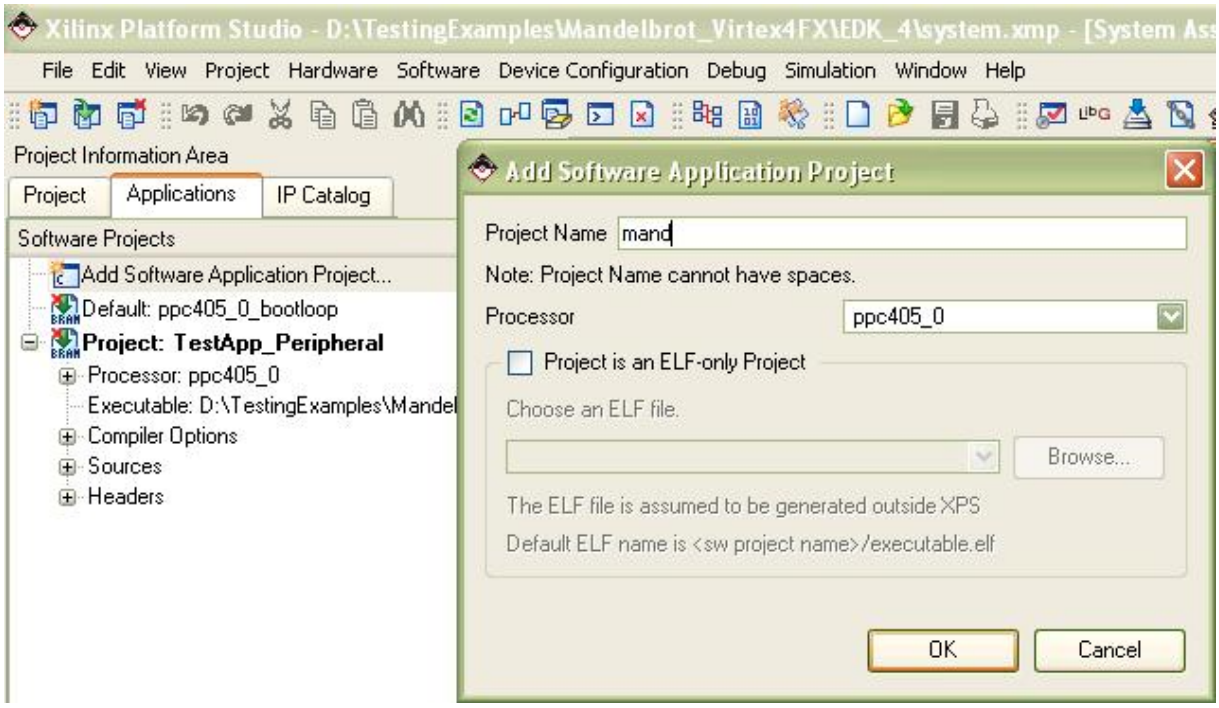
1.3.9 Adding the Software Application Files

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 9

The hardware configuration, including all required peripheral settings and connections, is now complete. The next step is to add the Mandelbrot sample application.

Create Mandelbrot Software Application

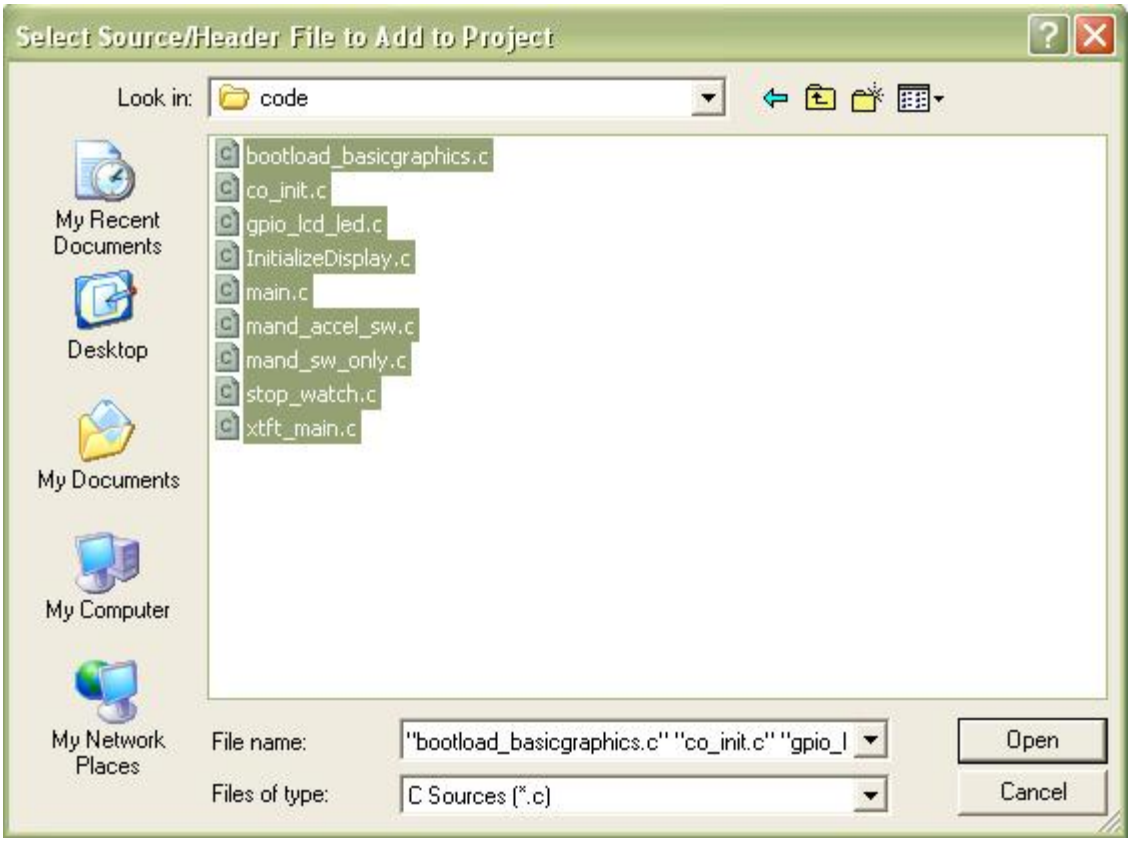
Select the Applications tab of the project, double-click the Add Software Application Project to show a dialogue. Type in the Project Name as "mand" and click OK as shown below:



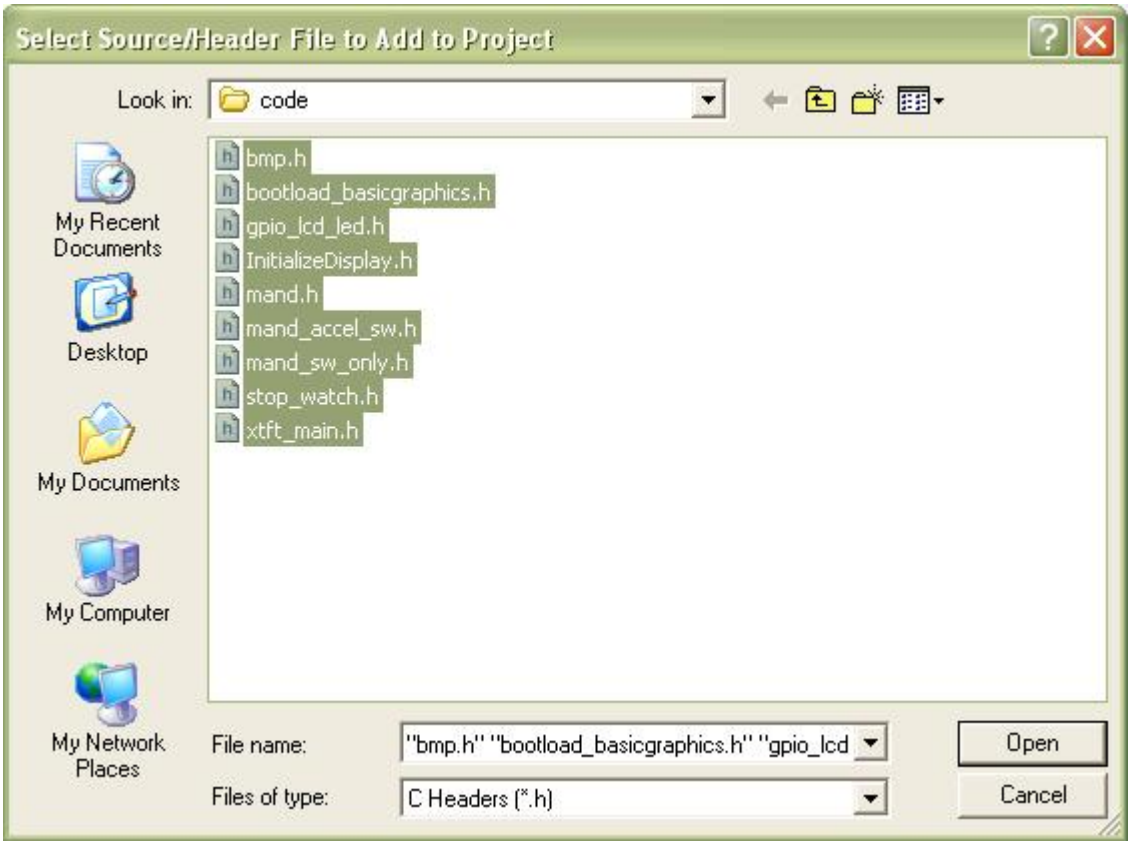
Adding the Mandelbrot Application Source Files

To add source C files to the project, open the Add Existing Files Dialogue from the Sources category by using right mouse button as shown below:

Select all files from the code subdirectory of your project as shown below:

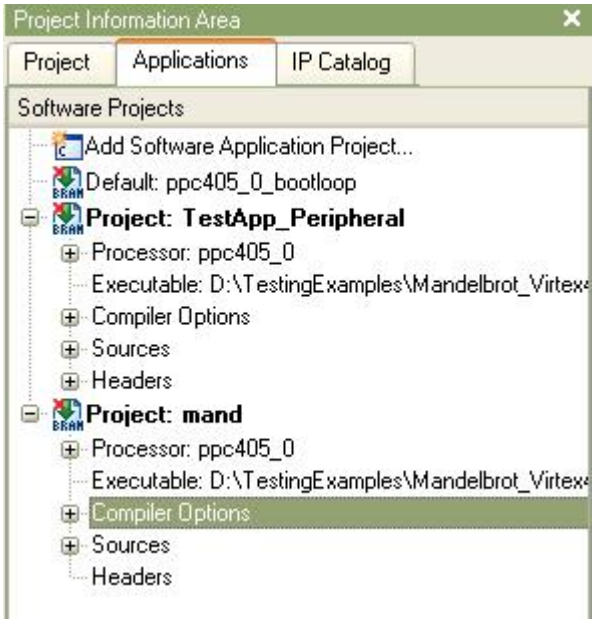


Next, add header files to your project similar to above as shown below:

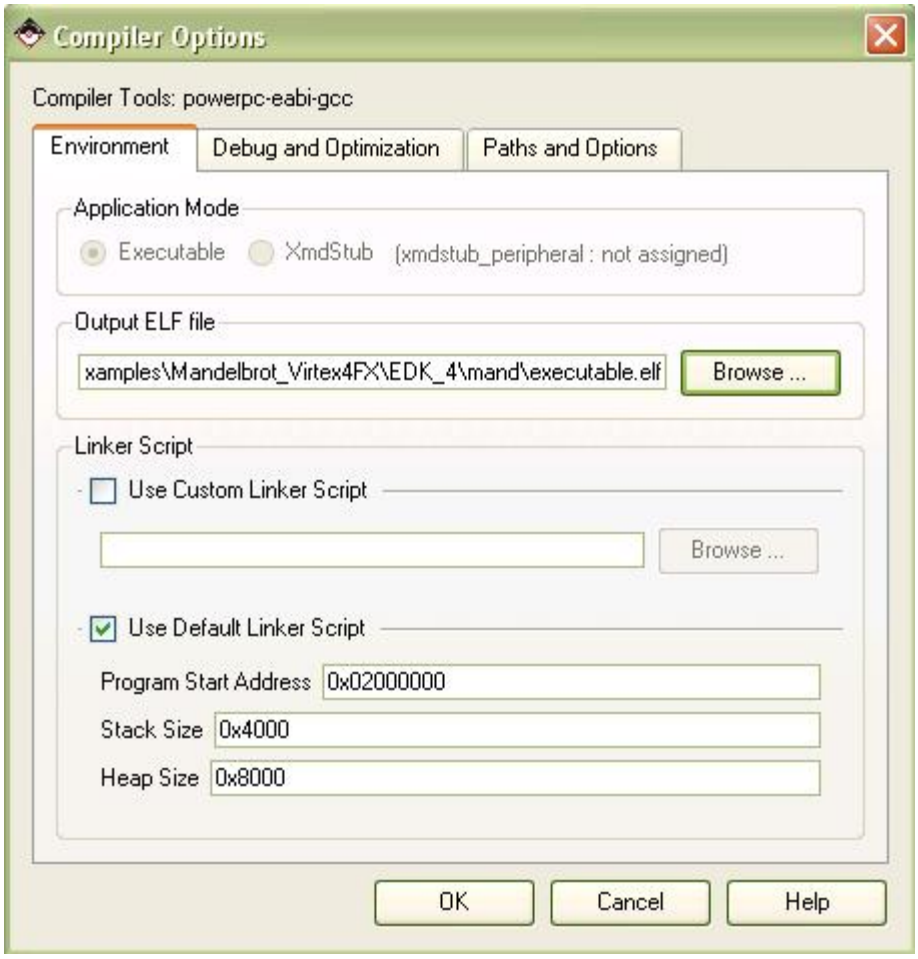


Setting Compiler Options

Now you will need to set a few compiler options for the project. To set the compiler options, double-click on the Compiler Options category in the Software Projects window:



In the Environment tab, select Use Default Linker Script, set the Program Start Address as 0x02000000 to avoid overlap with the TFT image memory location. Then enter Stack Size and Heap Size values of 0x4000 and 0x8000, respectively:



Click OK to close the Compiler Options dialog.

The software application is now ready to compile for the PowerPC processor.

See Also

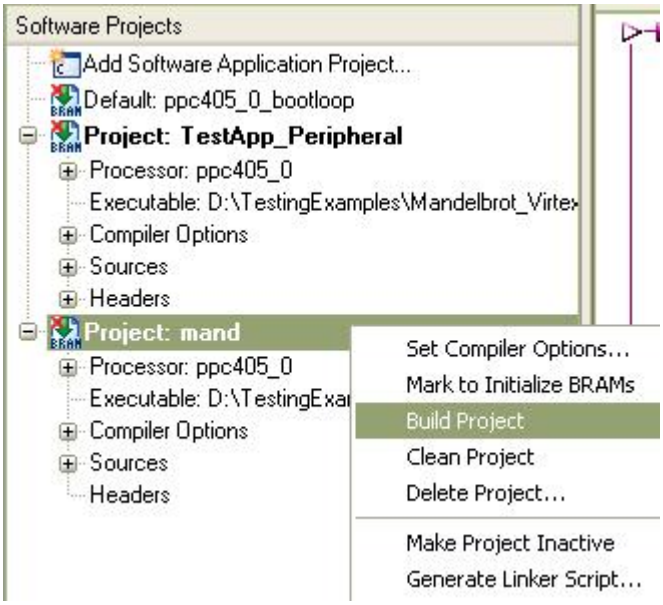
Building and Downloading the Application

1.3.10 Building and Downloading the Application

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 10

The Mandelbrot application is now ready to build, download and execute on the target ML403 board.

First, compile the software application to create a PowerPC executable. Do this by selecting Build Project from the Project: mand entry as shown below:

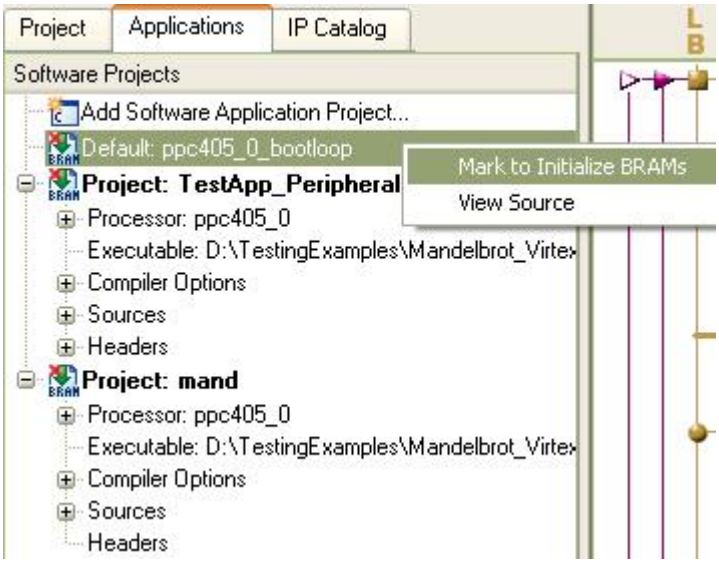


The size of the generated executable is shown below. It will be included in the FPGA bitstream.

```
LibGen Done.
powerpc-eabi-gcc -O2 /cygdrive/d/TestingExamples/Mandelbrot_Vir
/Mandelbrot_Virtex4FX/EDK_4/code/mand_sw_only.c /cygdrive/d/Tes
 -Wl,-defsym -Wl,_START_ADDR=0x02000000 -Wl,-defsym -Wl,_STACK_

powerpc-eabi-size mand/executable.elf
   text    data     bss     dec     hex filename
  58687    4528   49376  112591   1b7cf mand/executable.elf
Done!
```

Next, mark the ppc405_bootloop to initialize BRAMs by using the right mouse button. This will put a loop in the starting address of the on-chip memory.

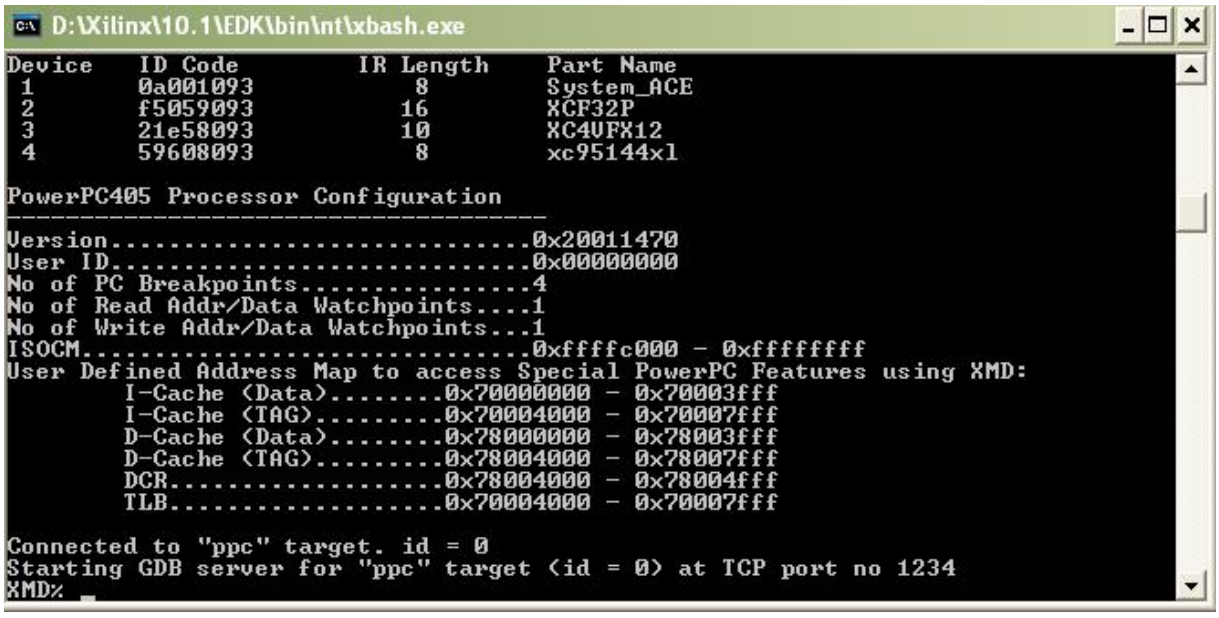


Now, it is time to download the bitstream to the ML403 board. Make sure the JTAG cable is properly connected and that the ML403 board is powered on. Also make sure the VGA display is connected and powered on.

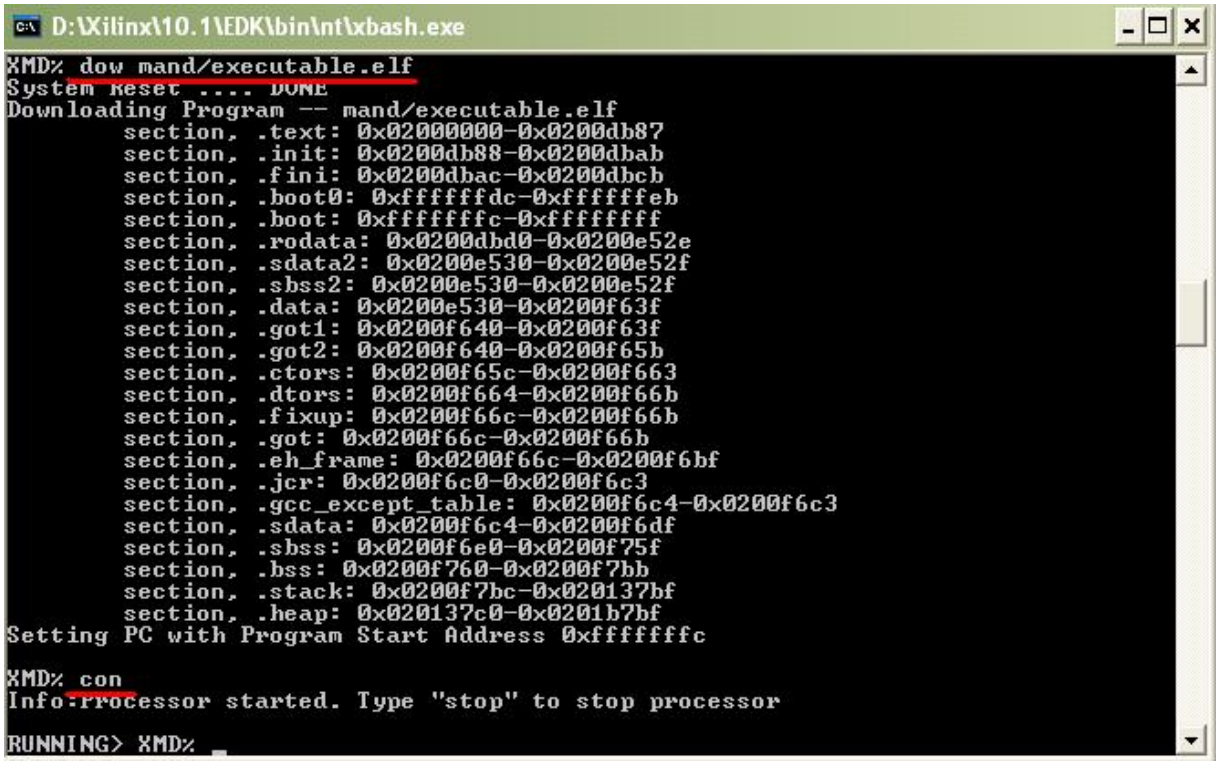
Select Download Bitstream as shown below:

Next, launch Xilinx Microprocessor Debugger (XMD) from the menu as shown below:

If this is the first time you have launched XMD for this EDK project, a couple of dialogue windows will pop up. Just click OK, then the XMD terminal will appear.



Download the Mandelbrot ELF file to the DDR_SDRAM, and then start running the program as follows:



After downloading has completed, the application will start running, resulting in a display similar to the following:

Congratulations! You have completed this advanced tutorial.

See Also

Quick Start Tutorials

Endnotes 2... (after index)

Back Cover