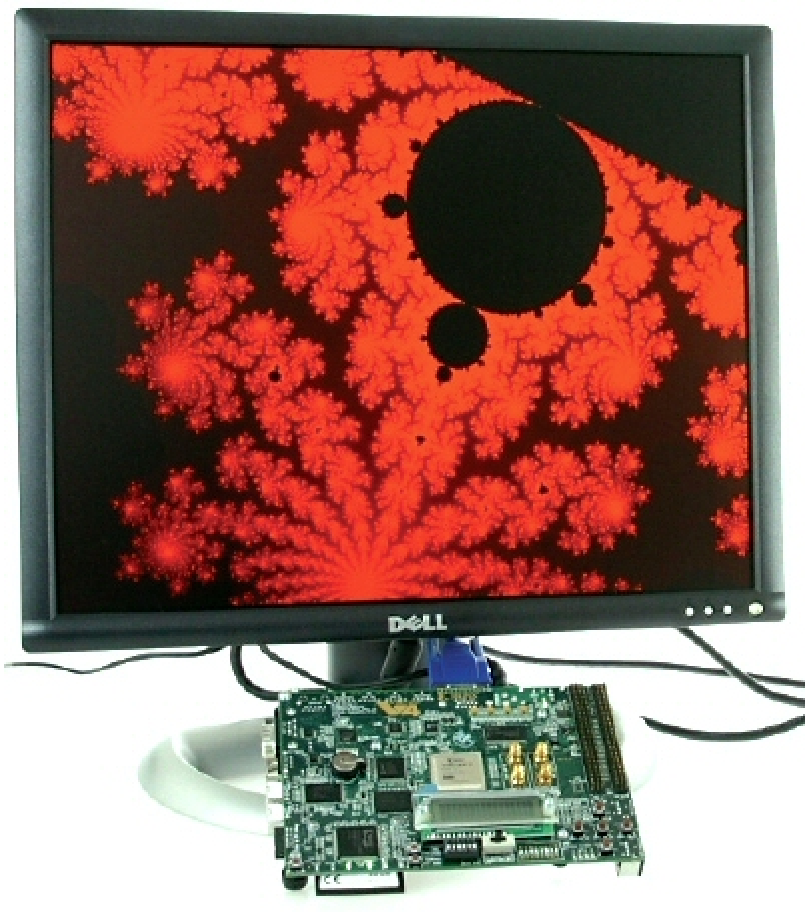
1 Tutorial: Fractal Image Generation using APU on the Virtex-4 FX Platform (EDK 10.1)

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 downloadable bitmap.
8. Download the application to the **ML403** board using a JTAG programming cable.

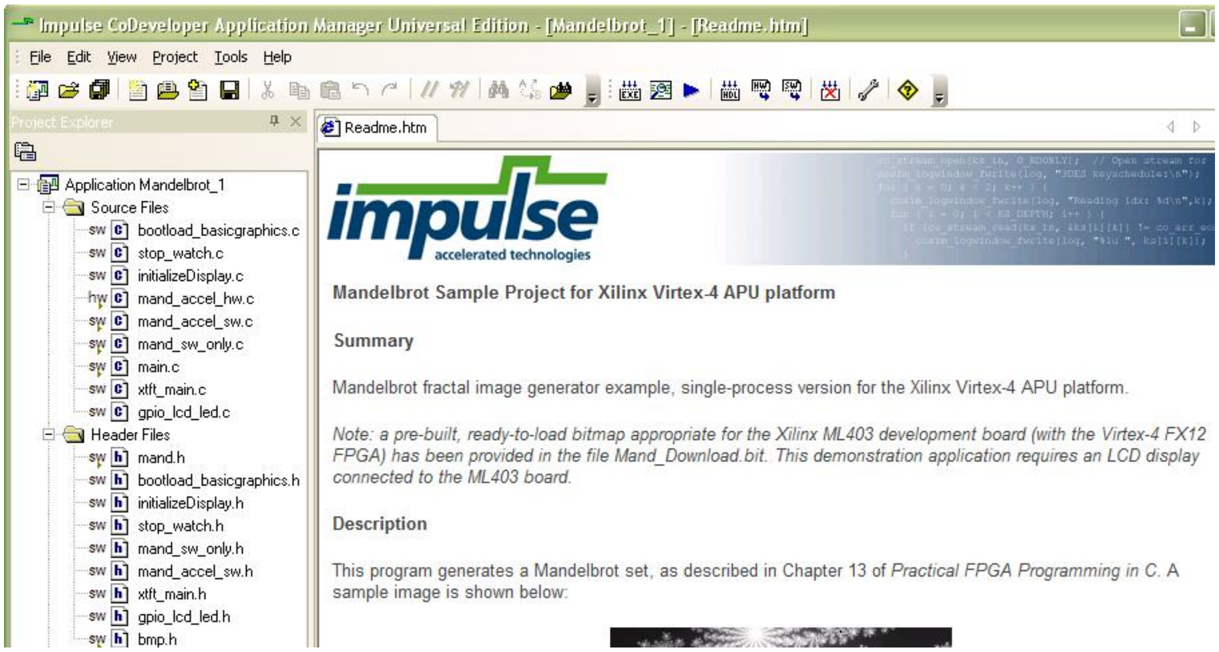
Detailed Steps

1.1 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\Embedded\Mandelbrot_Virtex4FX\ 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:


Impulse CoDeveloper Application Manager Universal Edition - [Mandelbrot_1] - [Readme.htm]
File Edit View Project Tools Help
Project Explorer
Readme.htm

Application Mandelbrot_1
Source Files
sw bootload_basicgraphics.c
sw stop_watch.c
sw initializeDisplay.c
hw mand_accel_hw.c
sw mand_accel_sw.c
sw mand_sw_only.c
sw main.c
sw xtft_main.c
sw gpio_lcd_led.c
Header Files
sw mand.h
sw bootload_basicgraphics.h
sw initializeDisplay.h
sw stop_watch.h
sw mand_sw_only.h
sw mand_accel_sw.h
sw xtft_main.h
sw gpio_lcd_led.h
sw bmp.h

Mandelbrot Sample Project for Xilinx Virtex-4 APU platform

Summary

Mandelbrot fractal image generator example, single-process version for the Xilinx Virtex-4 APU platform.

Note: a pre-built, ready-to-load bitmap appropriate for the Xilinx ML403 development board (with the Virtex-4 FX12 FPGA) has been provided in the file Mand_Download.bit. This demonstration application requires an LCD display connected to the ML403 board.

Description

This program generates a Mandelbrot set, as described in Chapter 13 of *Practical FPGA Programming in C*. A sample image is shown below:

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

1.2 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 FPGA**s. 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 **PLB** 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

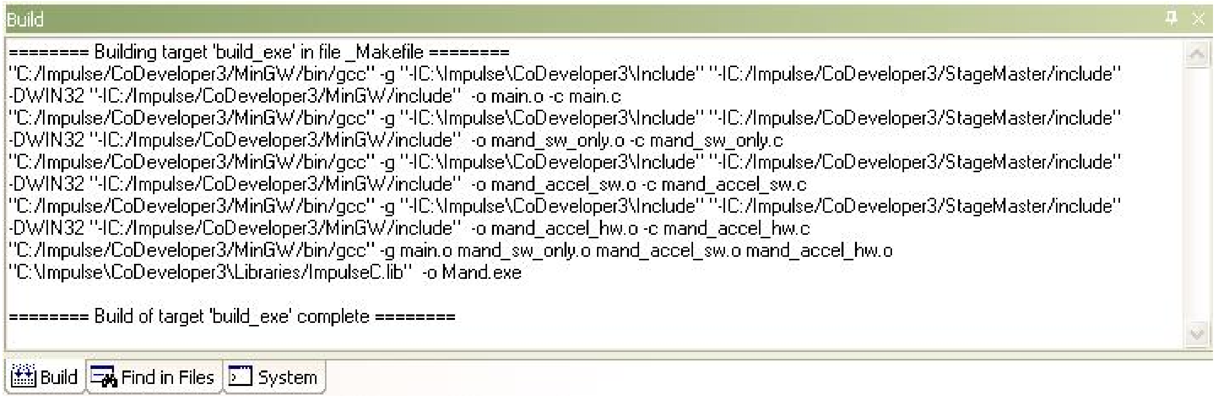
Compiling the Application for Simulation

1.3 Compiling the Application for Simulation

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 3

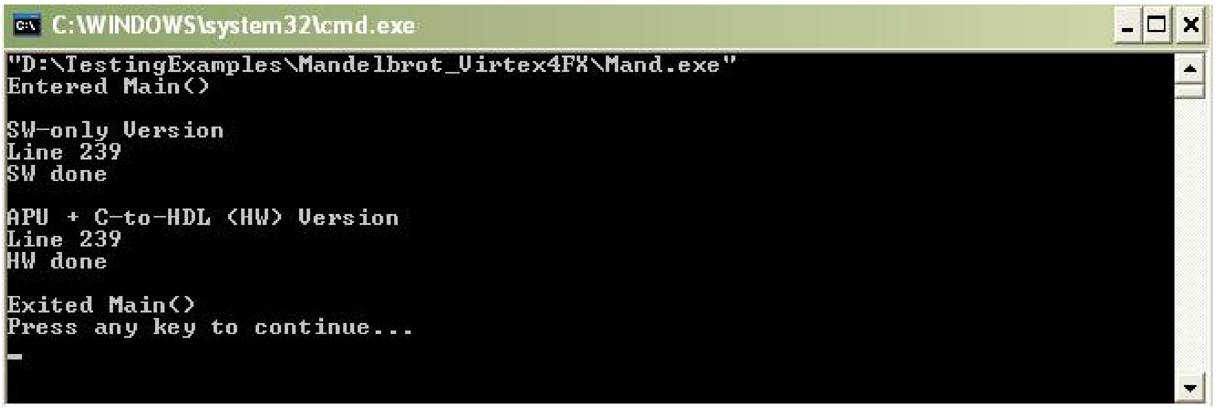
The software test bench provided with this example (in **mand_sw.c**) has been written in such a way that it can be compiled either to an FPGA as hardware (using fixed point math operations) or be compiled for desktop simulation, using either fixed or floating point math operations. This makes it possible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



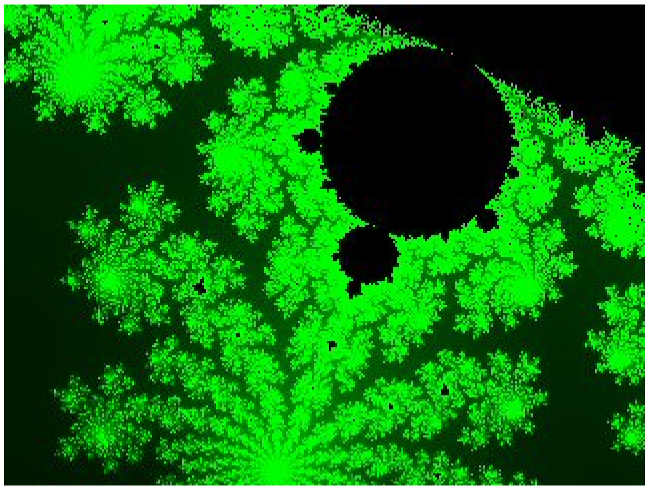
```
======== Building target 'build_exe' in file _Makefile ========
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include"
-DWIN32 "-IC:/Impulse/CoDeveloper3/MinGW/include"  -o main.o -c main.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include"
-DWIN32 "-IC:/Impulse/CoDeveloper3/MinGW/include"  -o mand_sw_only.o -c mand_sw_only.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include"
-DWIN32 "-IC:/Impulse/CoDeveloper3/MinGW/include"  -o mand_accel_sw.o -c mand_accel_sw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3\Include" "-IC:/Impulse/CoDeveloper3/StageMaster/include"
-DWIN32 "-IC:/Impulse/CoDeveloper3/MinGW/include"  -o mand_accel_hw.o -c mand_accel_hw.c
"C:/Impulse/CoDeveloper3/MinGW/bin/gcc" -g main.o mand_sw_only.o mand_accel_sw.o mand_accel_hw.o
"C:\Impulse\CoDeveloper3\Libraries/ImpulseC.lib"  -o Mand.exe

======== Build of target 'build_exe' complete ========
```

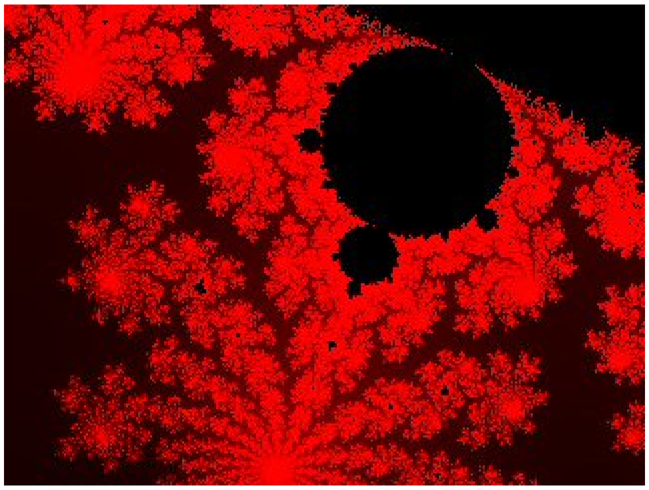
You now have a Windows executable representing the application implemented as a desktop (console) software application. You can run this executable by selecting **Project** -> **Launch Simulation Executable**. A command window will open and the simulation executable will run as shown below:



```
"D:\TestingExamples\Mandelbrot_Virtex4FX\Mand.exe"
Entered Main()

SW-only Version
Line 239
SW done

APU + C-to-HDL (HW) Version
Line 239
HW done

Exited Main()
Press any key to continue...
_
```

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** (640 x 480):





See Also

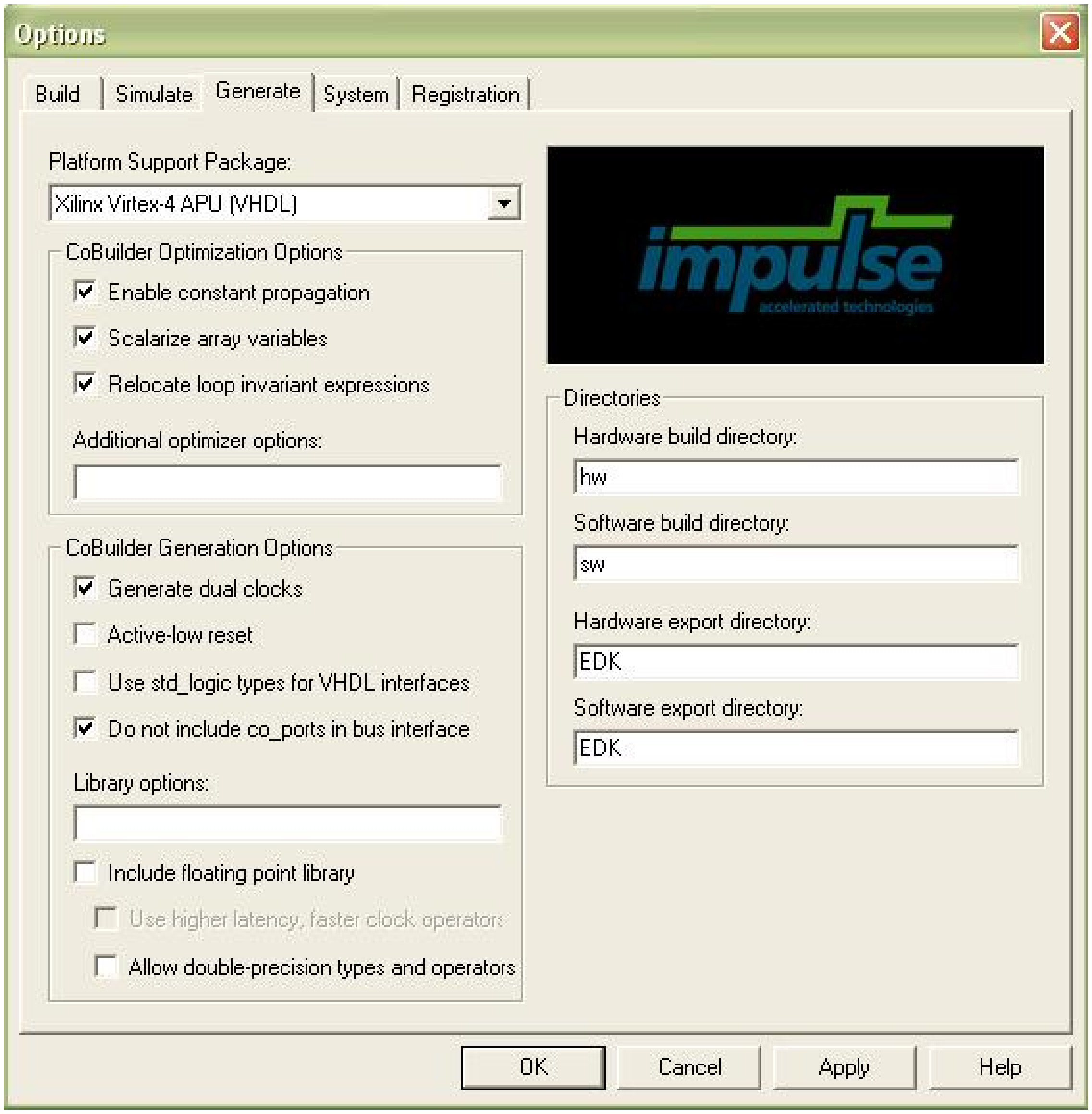
Building the Application for Hardware

1.4 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:



Platform Support Package: Xilinx Virtex-4 APU (VHDL)

CoBuilder Optimization Options
- [✓] Enable constant propagation
- [✓] Scalarize array variables
- [✓] Relocate loop invariant expressions

Additional optimizer options:

CoBuilder Generation Options
- [✓] Generate dual clocks
- [ ] Active-low reset
- [ ] Use std_logic types for VHDL interfaces
- [✓] Do not include co_ports in bus interface

Library options:

- [ ] Include floating point library
  - [ ] Use higher latency, faster clock operators
  - [ ] Allow double-precision types and operators

Directories
Hardware build directory: hw
Software build directory: sw
Hardware export directory: EDK
Software export directory: EDK

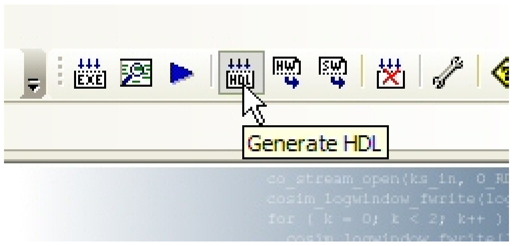
Specify **Xilinx Virtex-4 APU** as shown. Also specify **hw** and **sw** for the hardware and software directories as shown, and specify **EDK** for the hardware and software export directories. (**EDK** is the directory in which you will be creating a **Xilinx Platform Studio** project.)

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.)

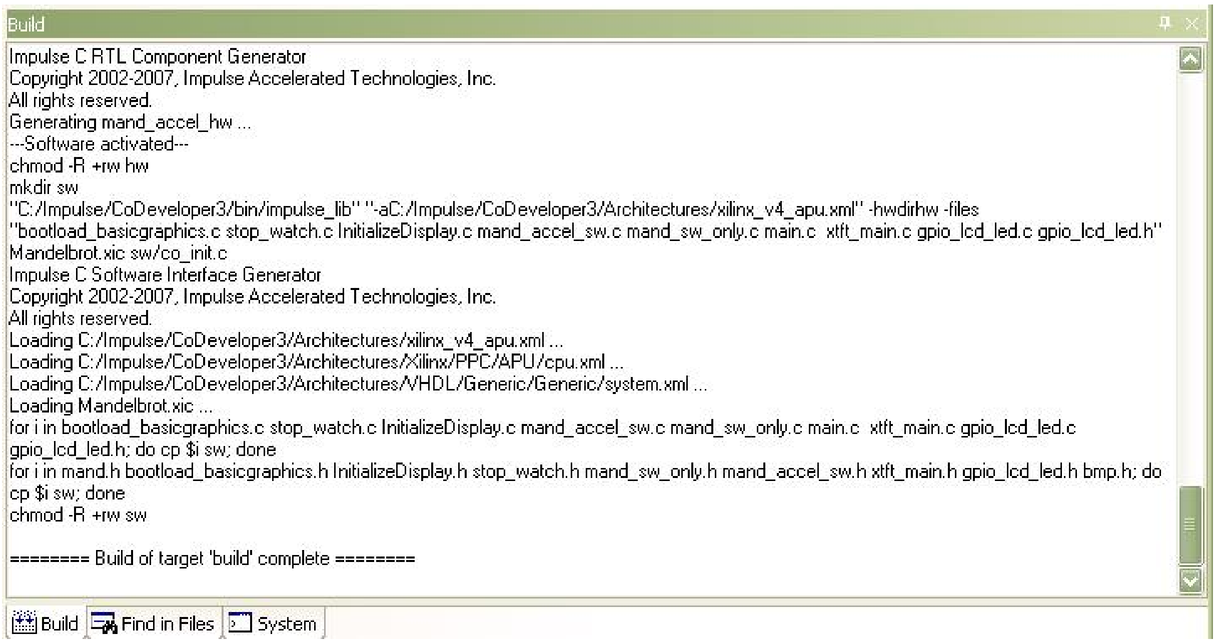
Click **OK** to save the options and exit the dialog.

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:



```
Impulse C RTL Component Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating mand_accel_hw ...
---Software activated---
chmod -R +rw hw
mkdir sw
"C:/Impulse/CoDeveloper3/bin/impulse_lib" "-aC:/Impulse/CoDeveloper3/Architectures/xilinx_v4_apu.xml" -hwdirhw -files
"bootload_basicgraphics.c stop_watch.c InitializeDisplay.c mand_accel_sw.c mand_sw_only.c main.c  xtft_main.c gpio_lcd_led.c gpio_lcd_led.h"
Mandelbrot.xic sw/co_init.c
Impulse C Software Interface Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3/Architectures/xilinx_v4_apu.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/Xilinx/PPC/APU/cpu.xml ...
Loading C:/Impulse/CoDeveloper3/Architectures/VHDL/Generic/Generic/system.xml ...
Loading Mandelbrot.xic ...
for i in bootload_basicgraphics.c stop_watch.c InitializeDisplay.c mand_accel_sw.c mand_sw_only.c main.c  xtft_main.c gpio_lcd_led.c
gpio_lcd_led.h; do cp $i sw; done
for i in mand.h bootload_basicgraphics.h InitializeDisplay.h stop_watch.h mand_sw_only.h mand_accel_sw.h xtft_main.h gpio_lcd_led.h bmp.h; do
cp $i sw; done
chmod -R +rw sw

======== Build of target 'build' complete ========
```

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

1.5 Exporting the Hardware and Software Files

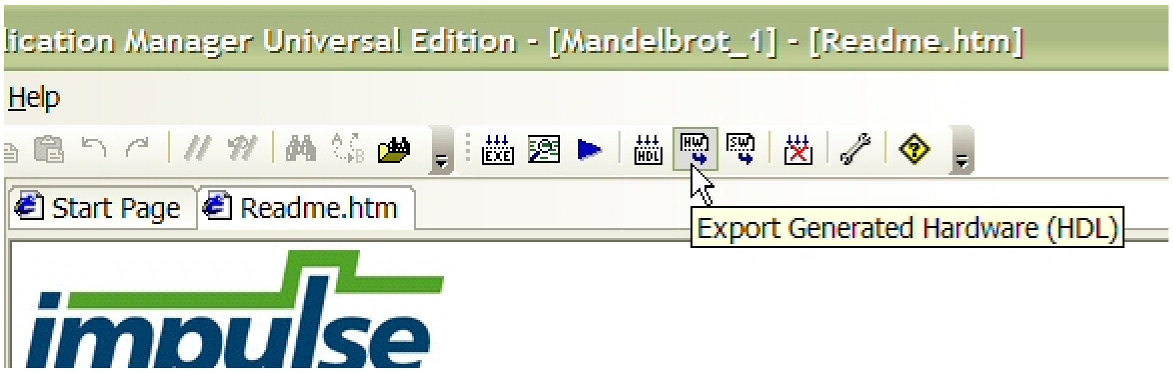
Mandelbrot Extended Tutorial for Virtex-4 FX, Step 5

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.

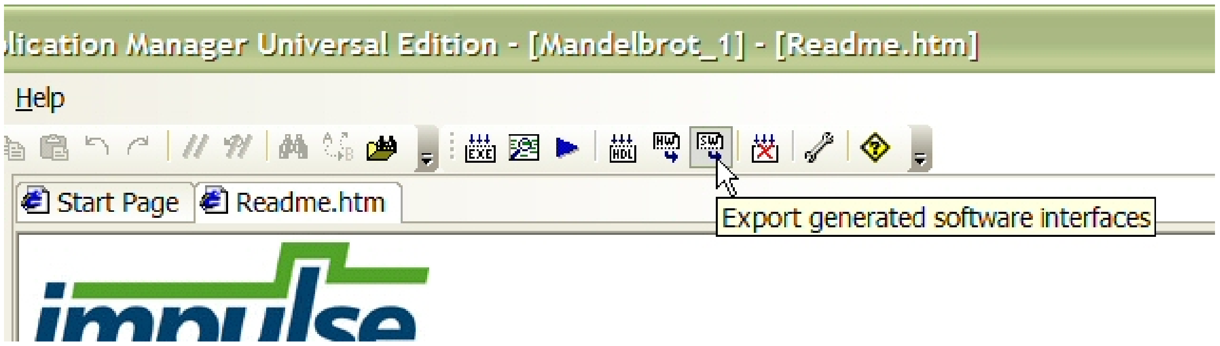
Within these target directories (in this case **EDK**), the specific destination for each file previously generated is determined from the Platform Support Package architecture library files. It is therefore important that the correct Platform Support Package (in this case **Xilinx Virtex-4 APU**) is 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


lication Manager Universal Edition - [Mandelbrot_1] - [Readme.htm]
Help
Start Page
Readme.htm
Export Generated Hardware (HDL)
impulse

Export the Software Files


lication Manager Universal Edition - [Mandelbrot_1] - [Readme.htm]
Help
Start Page
Readme.htm
Export generated software interfaces
impulse

*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

Creating the ML403 Test Platform

1.6 Creating the ML403 Test Platform

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 7

At this point you have:

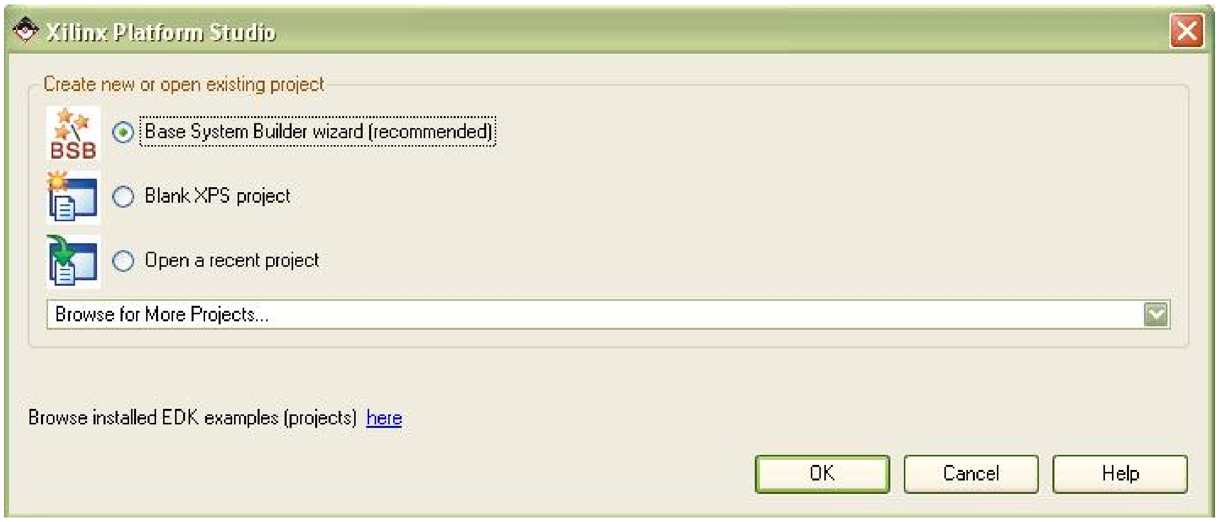
- Created hardware for the **Mandelbrot** accelerator.
- Exported the generated hardware to the **EDK** subdirectory as a pcore.
- Exported the **PowerPC** software application files to the **EDK** subirectory.

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:


Xilinx Platform Studio

Create new or open existing project

(● Base System Builder wizard (recommended)

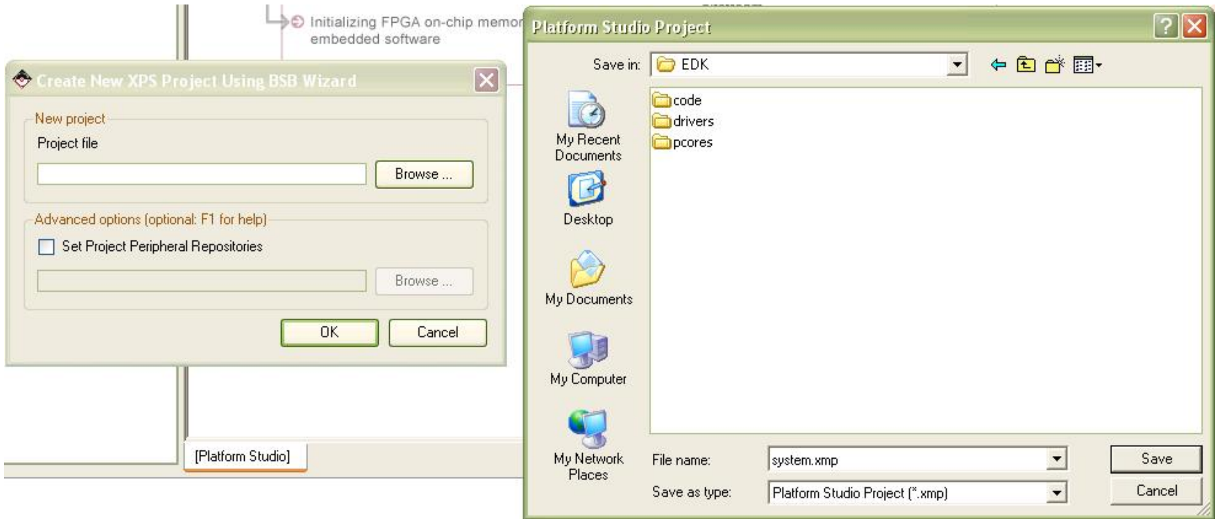
○ Blank XPS project

○ Open a recent project

Browse for More Projects...

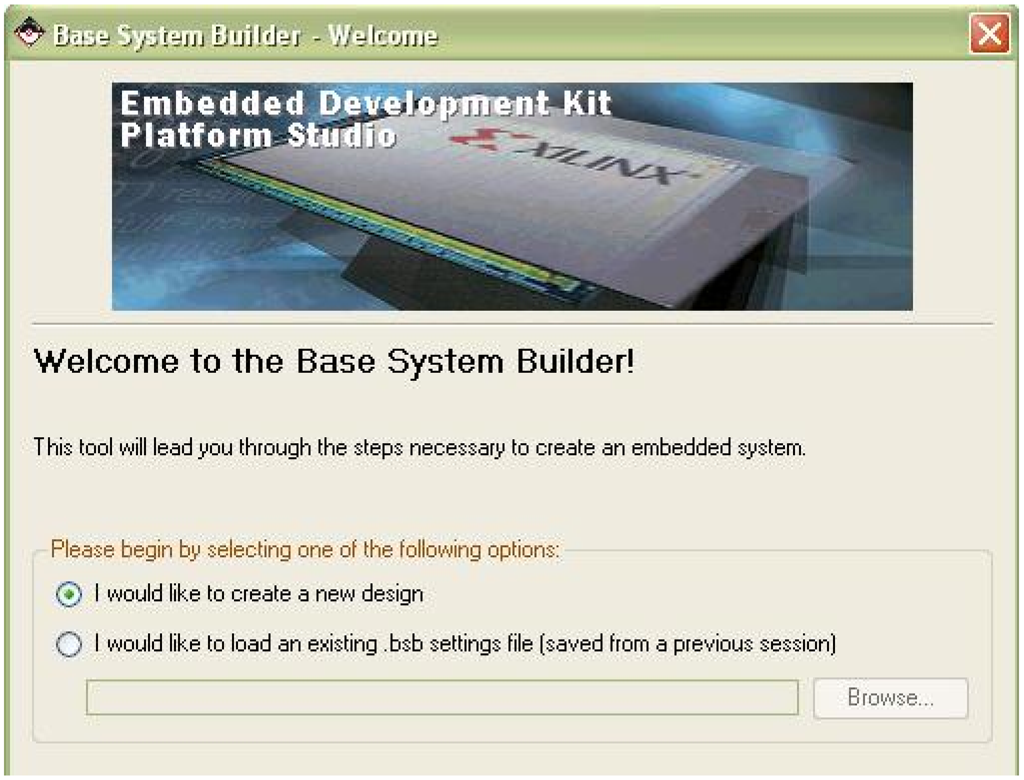
Browse installed EDK examples (projects) here

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:



Welcome to the Base System Builder!

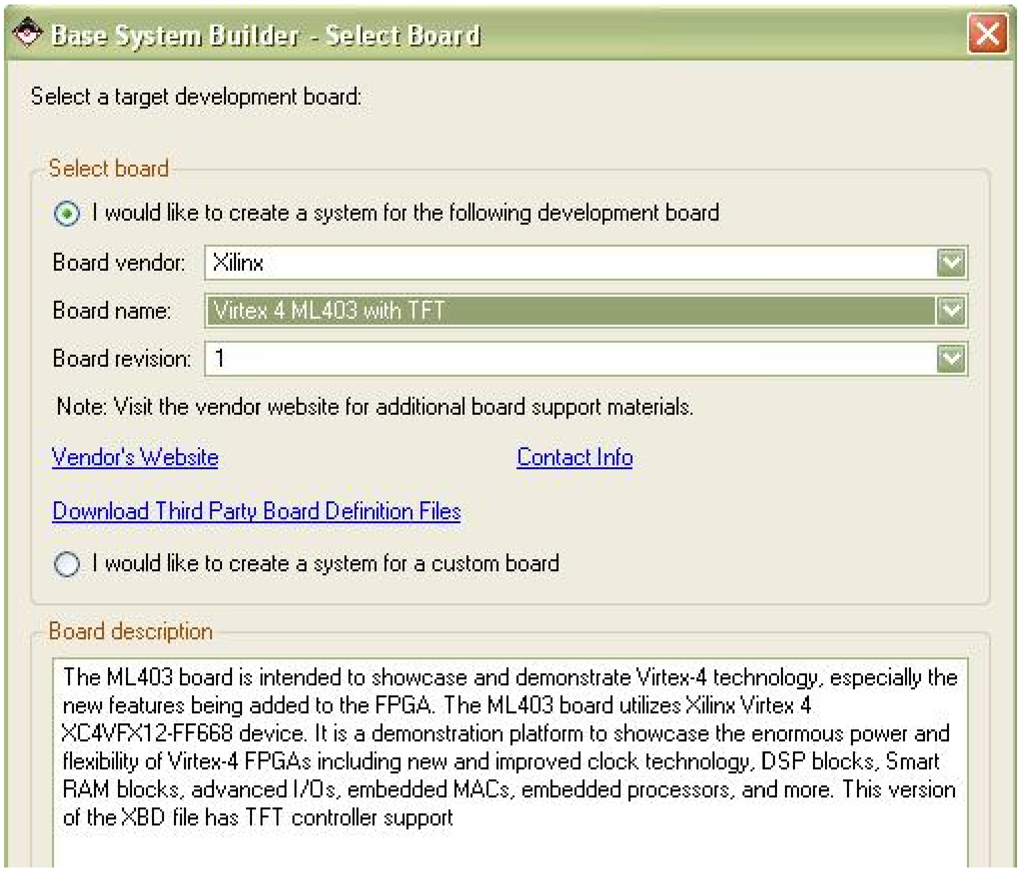
This tool will lead you through the steps necessary to create an embedded system.

Please begin by selecting one of the following options:

[✓ I would like to create a new design]

I would like to load an existing .bsb settings file (saved from a previous session)

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 with TFT** as shown:


Base System Builder - Select Board

Select a target development board:

Select board

● I would like to create a system for the following development board

Board vendor: Xilinx

Board name: Virtex 4 ML403 with TFT

Board revision: 1

Note: Visit the vendor website for additional board support materials.

Vendor's Website   Contact Info

Download Third Party Board Definition Files

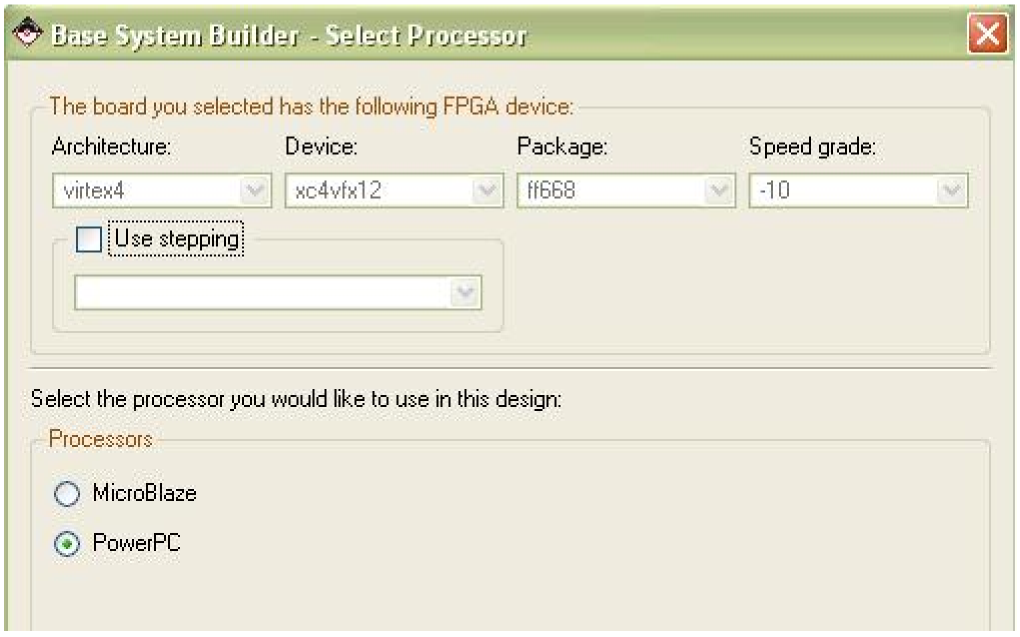
○ I would like to create a system for a custom board

Board description

The ML403 board is intended to showcase and demonstrate Virtex-4 technology, especially the new features being added to the FPGA. The ML403 board utilizes Xilinx Virtex 4 XC4VFX12-FF668 device. It is a demonstration platform to showcase the enormous power and flexibility of Virtex-4 FPGAs including new and improved clock technology, DSP blocks, Smart RAM blocks, advanced I/Os, embedded MACs, embedded processors, and more. This version of the XBD file has TFT controller support

Click the **Next** button to proceed to the next wizard page.

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


Base System Builder - Select Processor

The board you selected has the following FPGA device:

Architecture:	Device:	Package:	Speed grade:
virtex4	xc4vfx12	ff668	-10

☐ Use stepping

Select the processor you would like to use in this design:

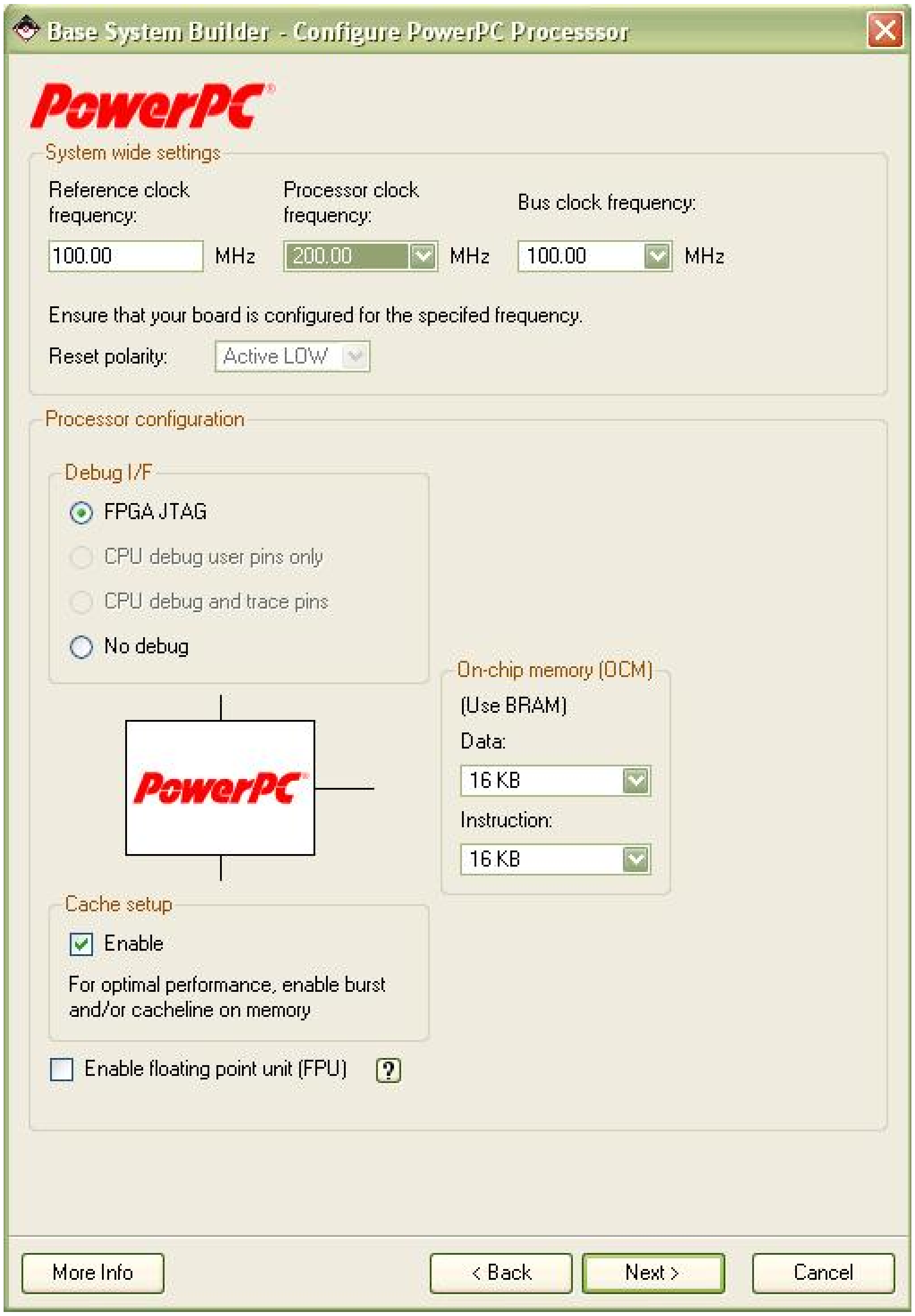
Processors

○ MicroBlaze

● PowerPC

On the **Configure PowerPC Processor** 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



Base System Builder - Configure PowerPC Processor

System wide settings

Reference clock frequency: 100.00 MHz
Processor clock frequency: 200.00 MHz
Bus clock frequency: 100.00 MHz

Ensure that your board is configured for the specifed frequency.

Reset polarity: Active LOW

Processor configuration

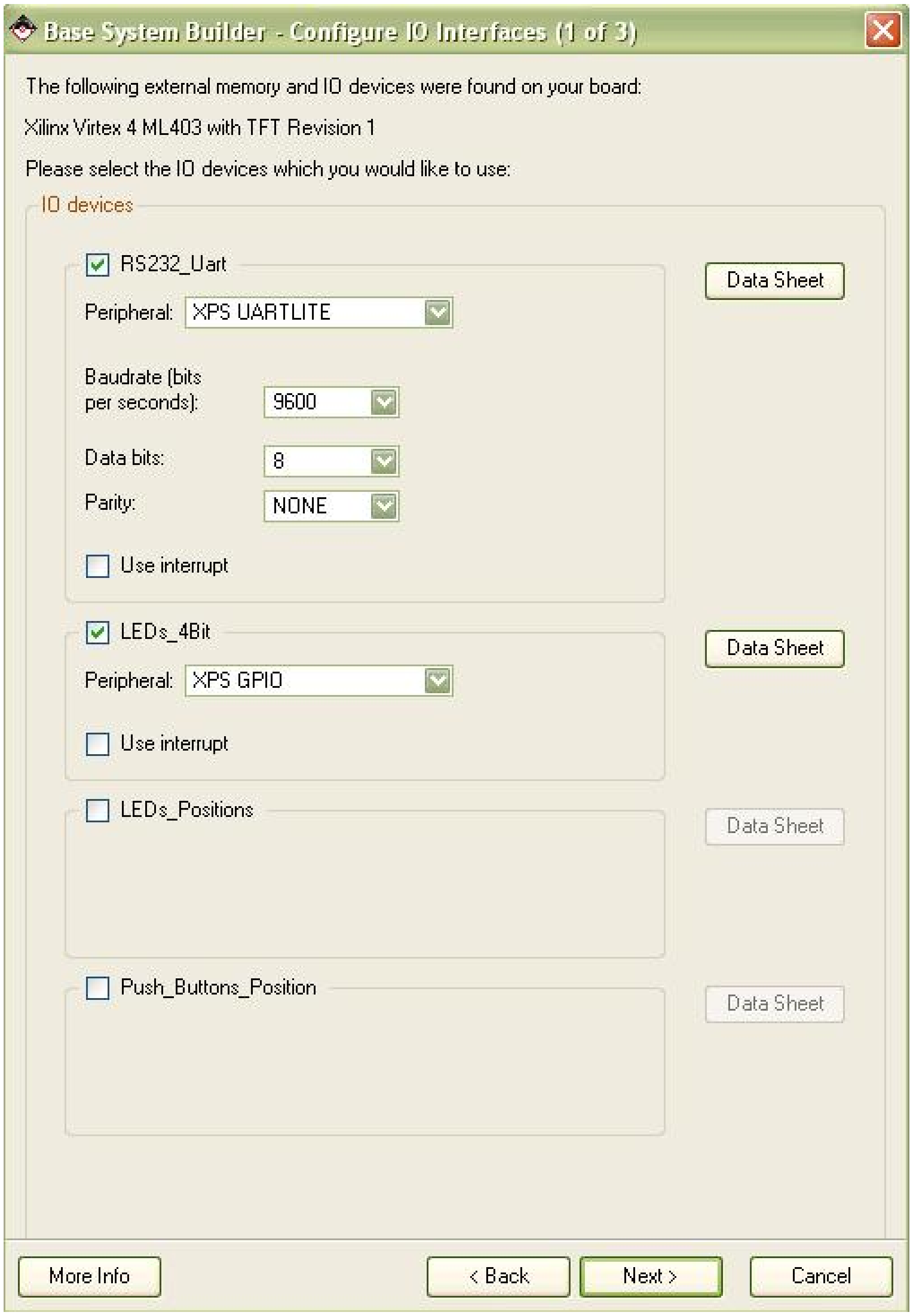
Debug I/F
[✓ FPGA JTAG]
CPU debug user pins only
CPU debug and trace pins
No debug

On-chip memory (OCM)
(Use BRAM)
Data: 16 KB
Instruction: 16 KB

Cache setup
[✓ Enable]
For optimal performance, enable burst and/or cacheline on memory

Enable floating point unit (FPU)

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:



The following external memory and IO devices were found on your board:

Xilinx Virtex 4 ML403 with TFT Revision 1

Please select the IO devices which you would like to use:

IO devices

[✓] RS232_Uart
Peripheral: XPS UARTLITE
Baudrate (bits per seconds): 9600
Data bits: 8
Parity: NONE
[ ] Use interrupt

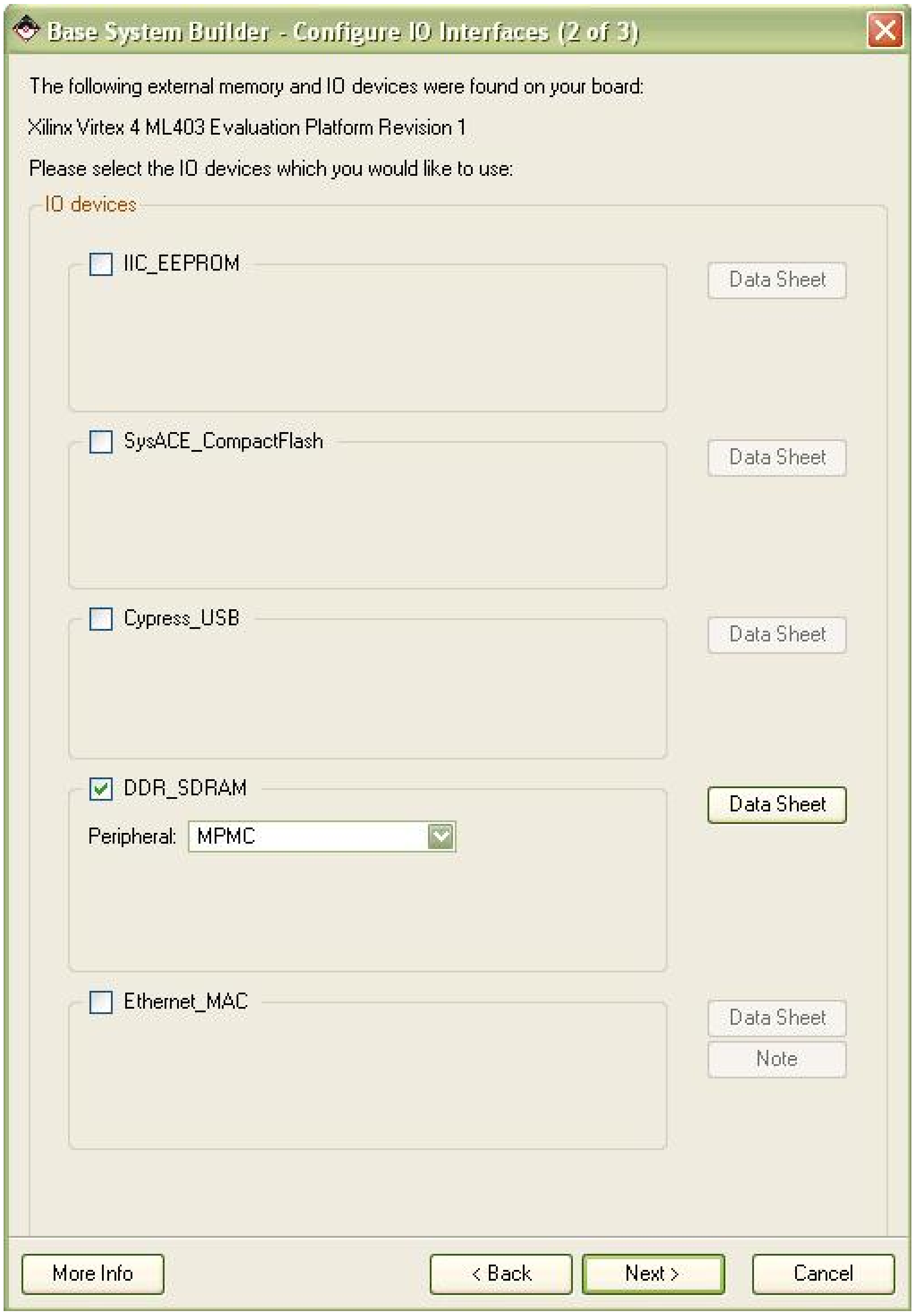
[✓] LEDs_4Bit
Peripheral: XPS GPIO
[ ] Use interrupt

[ ] LEDs_Positions

[ ] Push_Buttons_Position

Click **Next.**

On the next wizard page, select only the **DDR_SDRAM** peripheral:



The following external memory and IO devices were found on your board:

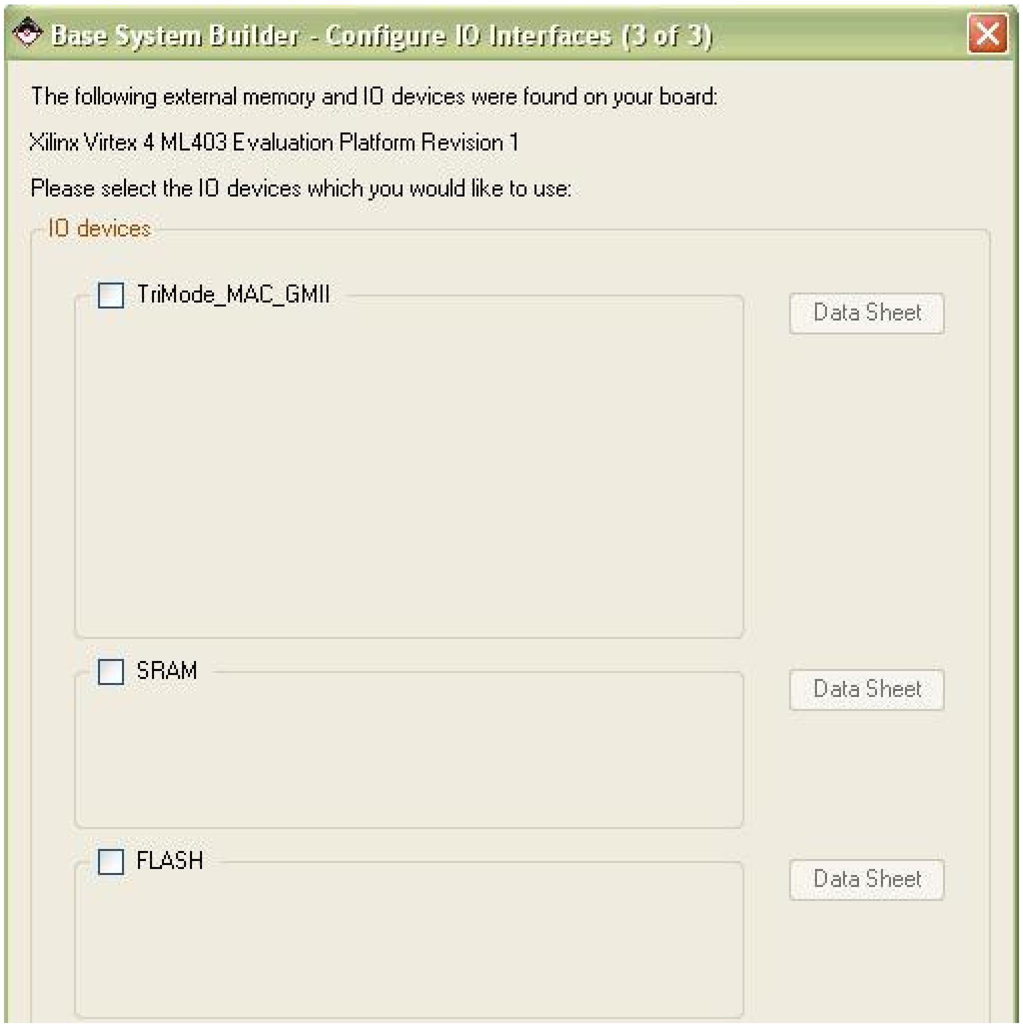
Xilinx Virtex 4 ML403 Evaluation Platform Revision 1

Please select the IO devices which you would like to use:

- [ ] IIC_EEPROM
- [ ] SysACE_CompactFlash
- [ ] Cypress_USB
- [✓] DDR_SDRAM — Peripheral: MPMC
- [ ] Ethernet_MAC

Click **Next.**

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


Base System Builder - Configure IO Interfaces (3 of 3)

The following external memory and IO devices were found on your board:

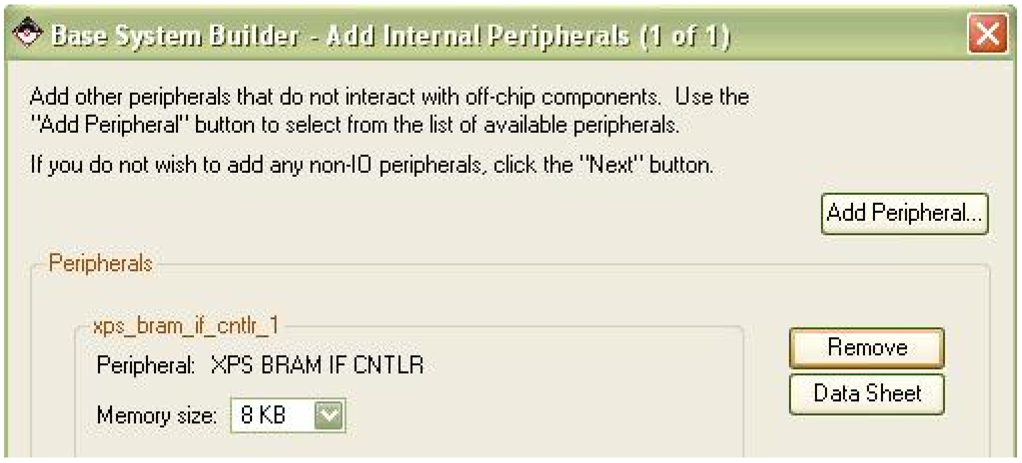
Xilinx Virtex 4 ML403 Evaluation Platform Revision 1

Please select the IO devices which you would like to use:

IO devices

Click **Next**.

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


Base System Builder - Add Internal Peripherals (1 of 1)

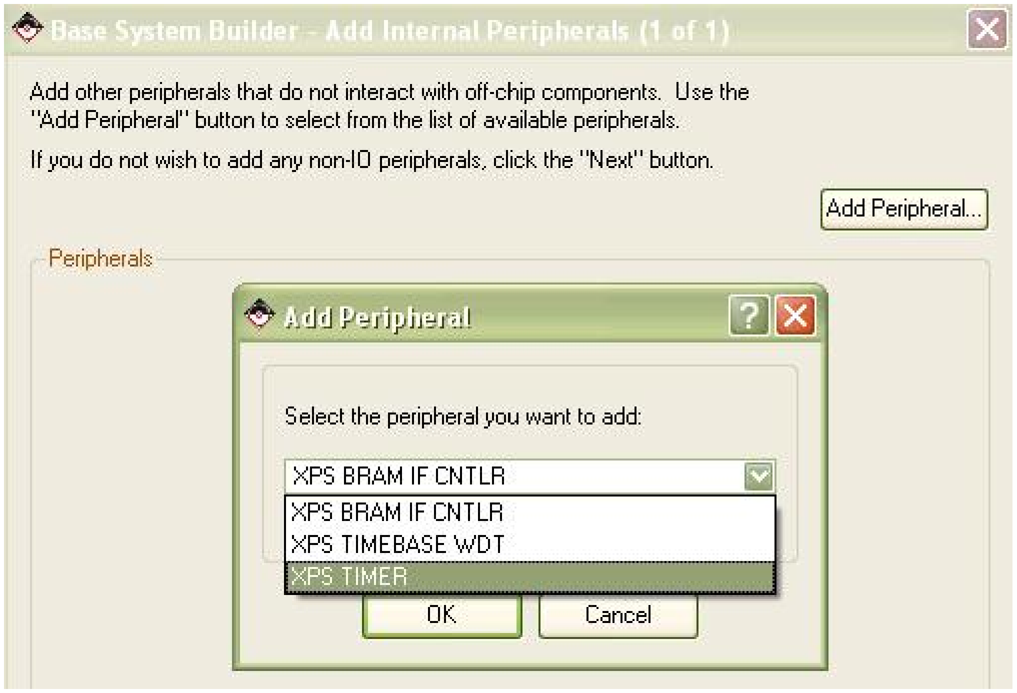
Add other peripherals that do not interact with off-chip components. Use the "Add Peripheral" button to select from the list of available peripherals.

If you do not wish to add any non-IO peripherals, click the "Next" button.

Peripheral: XPS BRAM IF CNTLR

Memory size: 8 KB

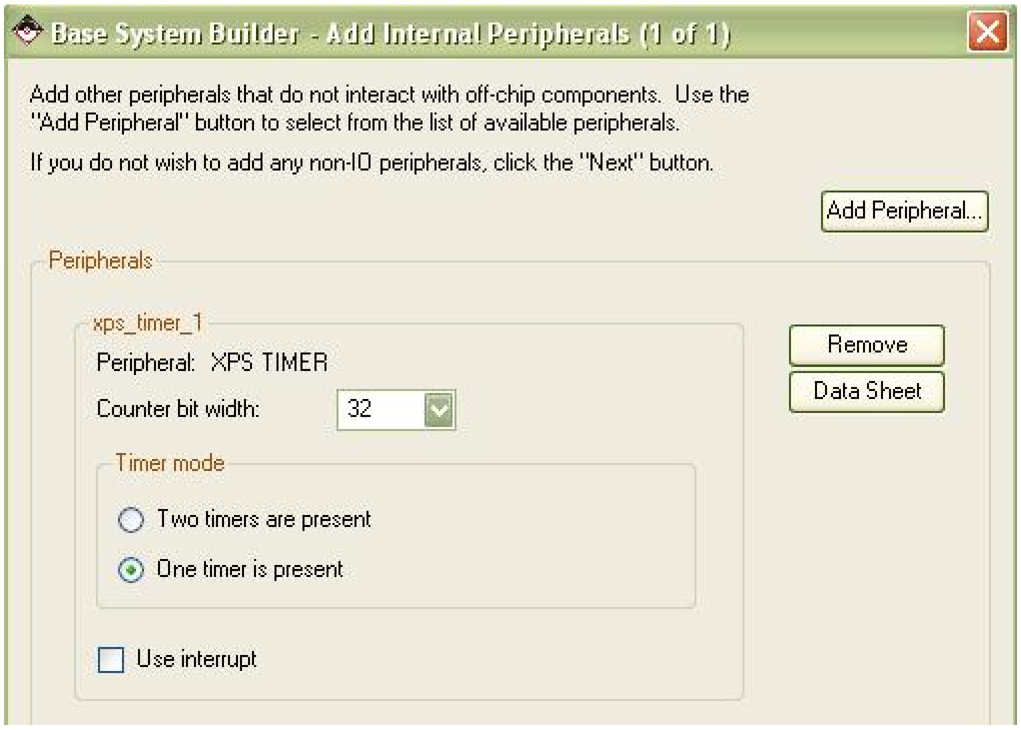
Click **Add Peripherals...** to add an **XPS TIMER** to the design.


Base System Builder - Add Internal Peripherals (1 of 1)

Add other peripherals that do not interact with off-chip components. Use the "Add Peripheral" button to select from the list of available peripherals.

If you do not wish to add any non-IO peripherals, click the "Next" button.

Set up the **xps_timer_1** as shown below:


Base System Builder - Add Internal Peripherals (1 of 1)

Add other peripherals that do not interact with off-chip components. Use the "Add Peripheral" button to select from the list of available peripherals.

If you do not wish to add any non-IO peripherals, click the "Next" button.

xps_timer_1

Peripheral: XPS TIMER

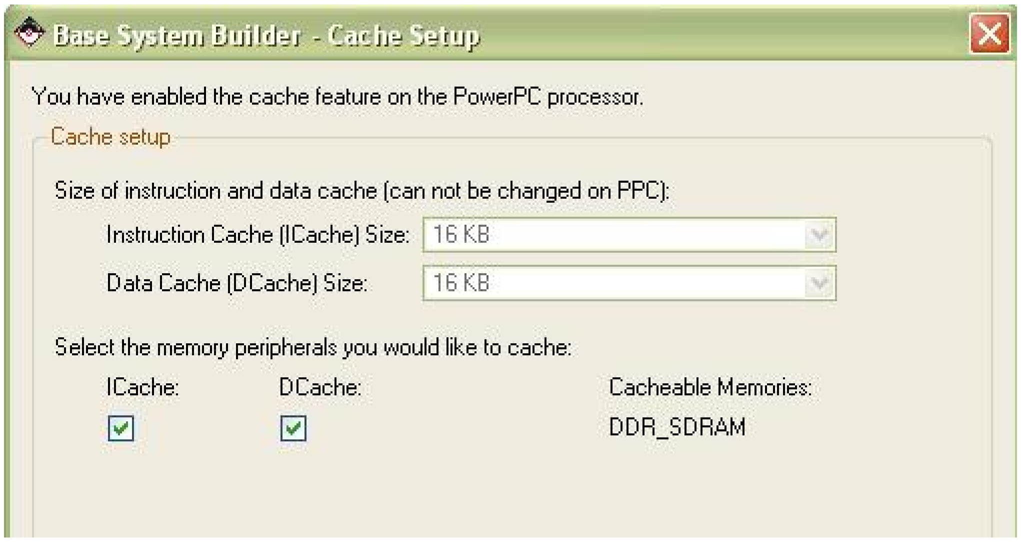
Counter bit width: 32

Timer mode: Two timers are present [✓ One timer is present]

Use interrupt: ☐

Click **Next.**

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



You have enabled the cache feature on the PowerPC processor.

Cache setup

Size of instruction and data cache (can not be changed on PPC):

Instruction Cache (ICache) Size: 16 KB

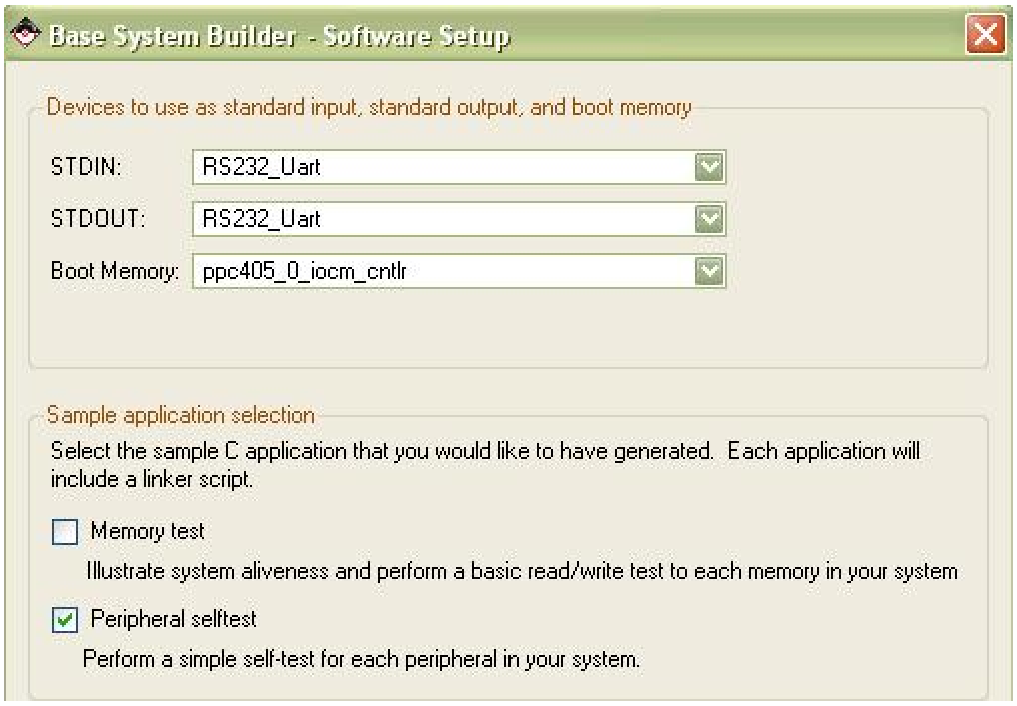
Data Cache (DCache) Size: 16 KB

Select the memory peripherals you would like to cache:

ICache:	DCache:	Cacheable Memories:
✓	✓	DDR_SDRAM

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:



Devices to use as standard input, standard output, and boot memory

STDIN: RS232_Uart

STDOUT: RS232_Uart

Boot Memory: ppc405_0_iocm_cntlr

Sample application selection

Select the sample C application that you would like to have generated. Each application will include a linker script.

☐ Memory test

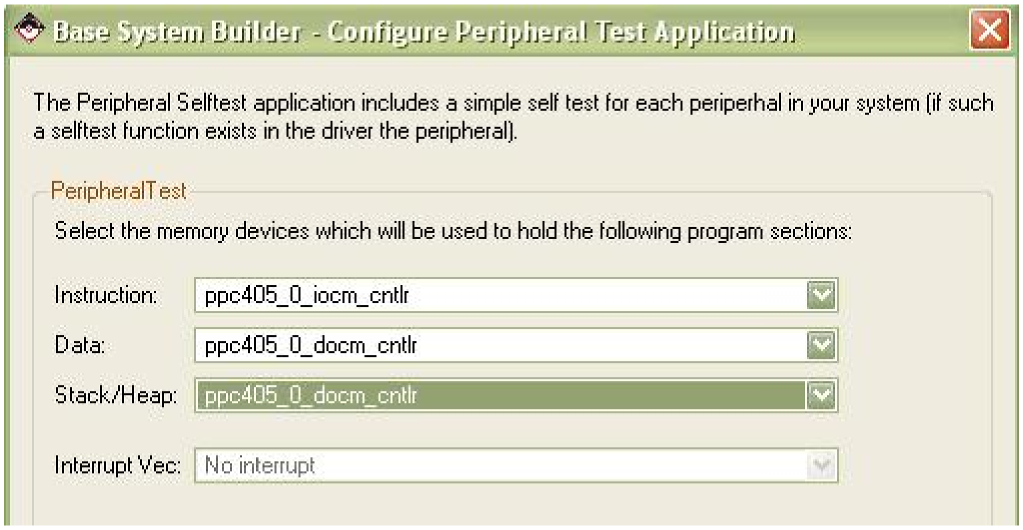
Illustrate system aliveness and perform a basic read/write test to each memory in your system

☑ Peripheral selftest

Perform a simple self-test for each peripheral in your system.

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:


Base System Builder - Configure Peripheral Test Application

The Peripheral Selftest application includes a simple self test for each periperhal in your system (if such a selftest function exists in the driver the peripheral).

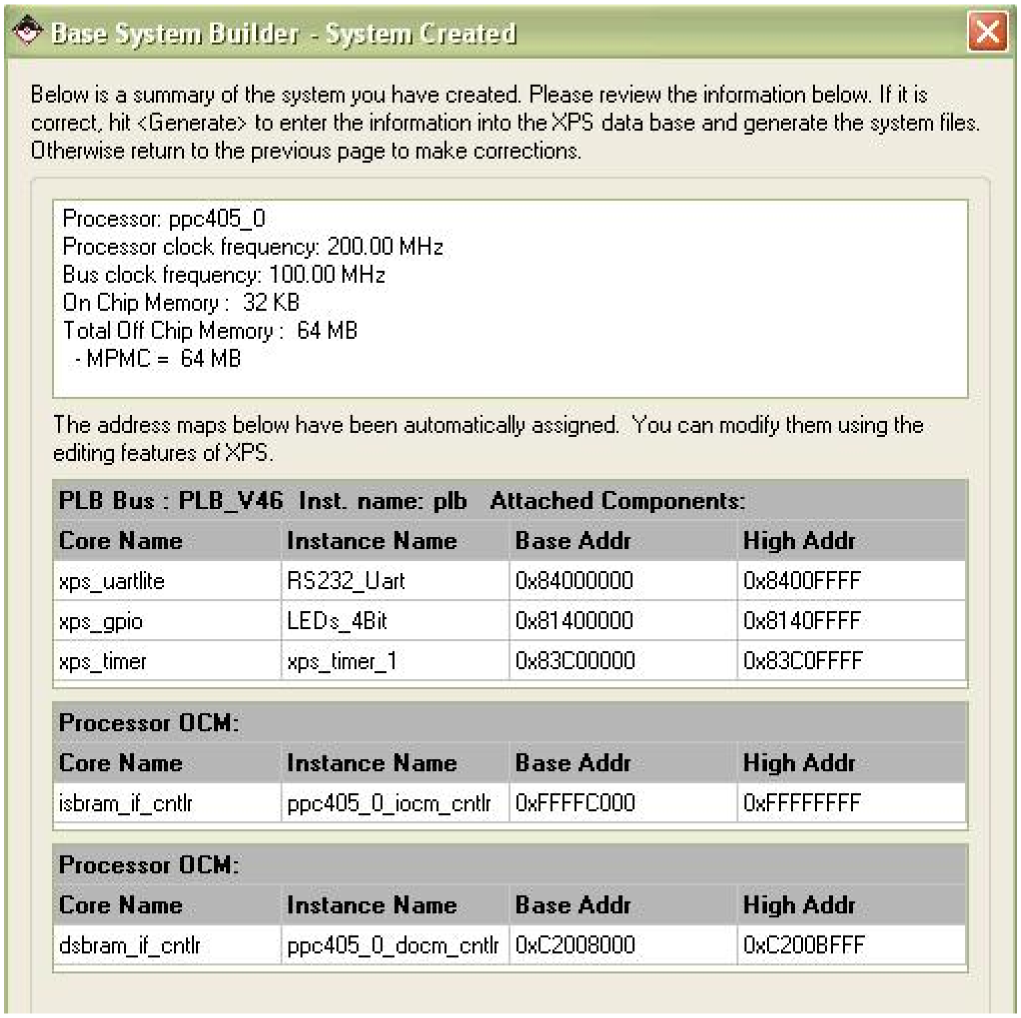
PeripheralTest

Select the memory devices which will be used to hold the following program sections:

Instruction:	ppc405_0_iocm_cntlr
Data:	ppc405_0_docm_cntlr
Stack/Heap:	ppc405_0_docm_cntlr
Interrupt Vec:	No interrupt

Click **Next**.

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



Below is a summary of the system you have created. Please review the information below. If it is correct, hit <Generate> to enter the information into the XPS data base and generate the system files. Otherwise return to the previous page to make corrections.

```
Processor: ppc405_0
Processor clock frequency: 200.00 MHz
Bus clock frequency: 100.00 MHz
On Chip Memory :  32 KB
Total Off Chip Memory :  64 MB
  - MPMC =  64 MB
```

The address maps below have been automatically assigned. You can modify them using the editing features of XPS.

PLB Bus : PLB_V46 Inst. name: plb Attached Components:

Core Name	Instance Name	Base Addr	High Addr
xps_uartlite	RS232_Uart	0x84000000	0x8400FFFF
xps_gpio	LEDs_4Bit	0x81400000	0x8140FFFF
xps_timer	xps_timer_1	0x83C00000	0x83C0FFFF

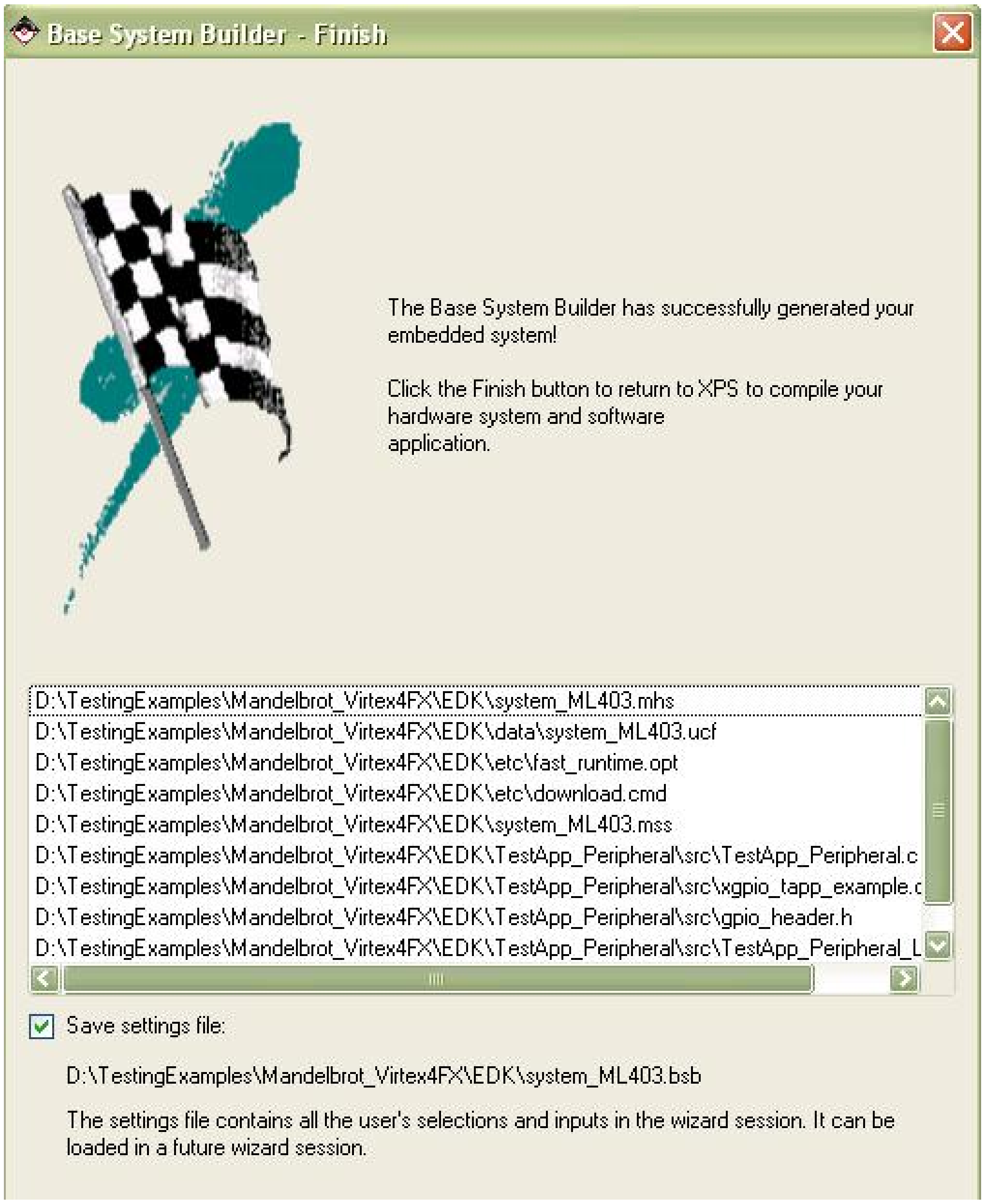
Processor OCM:

Core Name	Instance Name	Base Addr	High Addr
isbram_if_cntlr	ppc405_0_iocm_cntlr	0xFFFFC000	0xFFFFFFFF

Processor OCM:

Core Name	Instance Name	Base Addr	High Addr
dsbram_if_cntlr	ppc405_0_docm_cntlr	0xC2008000	0xC200BFFF

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.



The Base System Builder has successfully generated your embedded system!

Click the Finish button to return to XPS to compile your hardware system and software application.

```
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\system_ML403.mhs
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\data\system_ML403.ucf
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\etc\fast_runtime.opt
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\etc\download.cmd
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\system_ML403.mss
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\TestApp_Peripheral\src\TestApp_Peripheral.c
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\TestApp_Peripheral\src\xgpio_tapp_example.c
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\TestApp_Peripheral\src\gpio_header.h
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\TestApp_Peripheral\src\TestApp_Peripheral_L
```

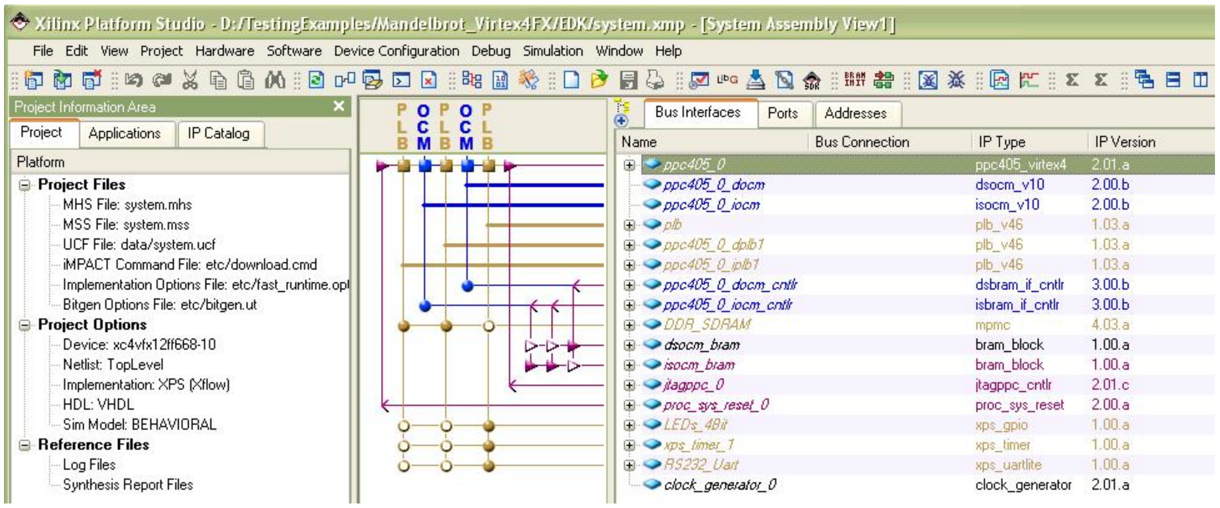
☑ Save settings file:

D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\system_ML403.bsb

The settings file contains all the user's selections and inputs in the wizard session. It can be loaded in a future wizard session.

Click **Finish** to exit the wizard.

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


Xilinx Platform Studio - D:/TestingExamples/Mandelbrot_Virtex4FX/EDK/system.xmp - [System Assembly View1]

File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help

Project Information Area

Project | Applications | IP Catalog

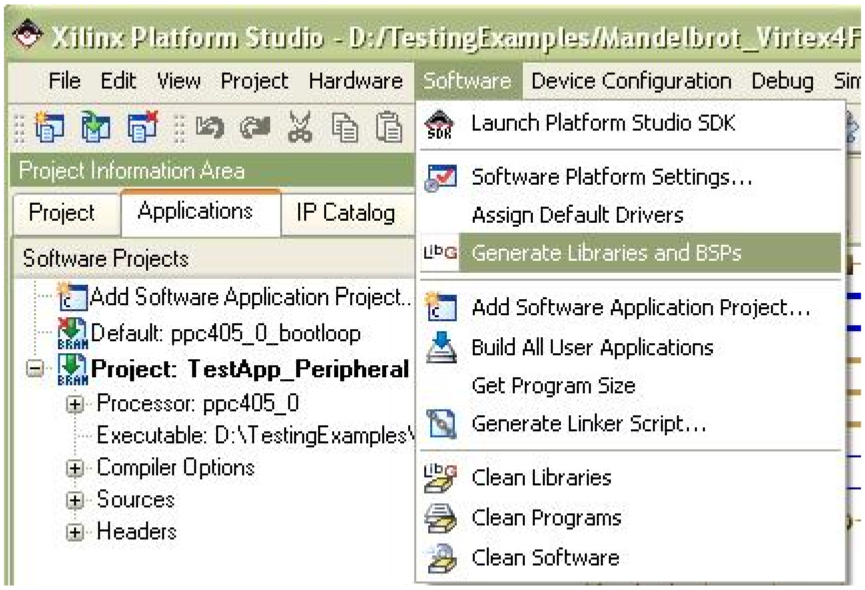
Platform
- Project Files
  - MHS File: system.mhs
  - MSS File: system.mss
  - UCF File: data/system.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.opt
  - Bitgen Options File: etc/bitgen.ut
- Project Options
  - Device: xc4vfx12ff668-10
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Reference Files
  - Log Files
  - Synthesis Report Files

Bus Interfaces | Ports | Addresses

Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.03.a
ppc405_0_dplb1		plb_v46	1.03.a
ppc405_0_iplb1		plb_v46	1.03.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.03.a
dsocm_bram		bram_block	1.00.a
isocm_bram		bram_block	1.00.a
jtagppc_0		jtagppc_cntlr	2.01.c
proc_sys_reset_0		proc_sys_reset	2.00.a
LEDs_4Bit		xps_gpio	1.00.a
xps_timer_1		xps_timer	1.00.a
RS232_Uart		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a

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** 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:


Xilinx Platform Studio - D:/TestingExamples/Mandelbrot_Virtex4F

File Edit View Project Hardware Software Device Configuration Debug Sim

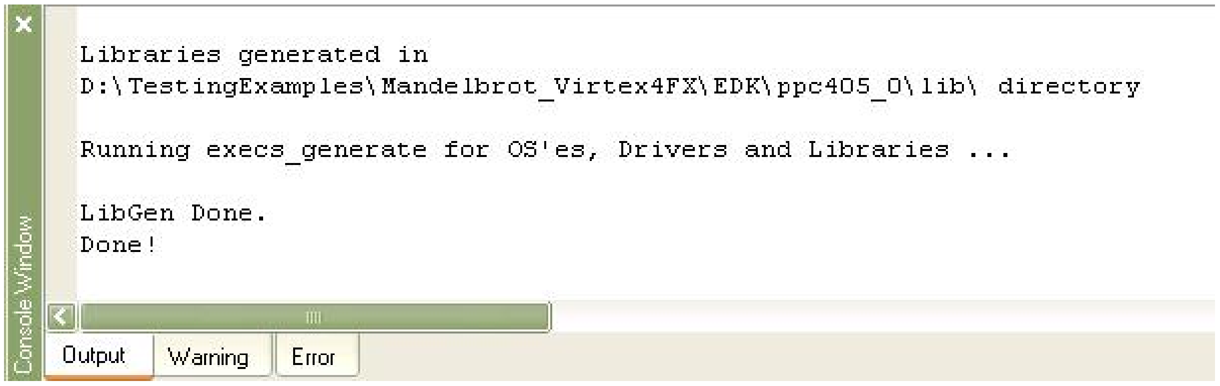
Project Information Area

Project | Applications | IP Catalog

Software Projects
- Add Software Application Project...
- Default: ppc405_0_bootloop
- Project: TestApp_Peripheral
  - Processor: ppc405_0
  - Executable: D:\TestingExamples\
  - Compiler Options
  - Sources
  - Headers

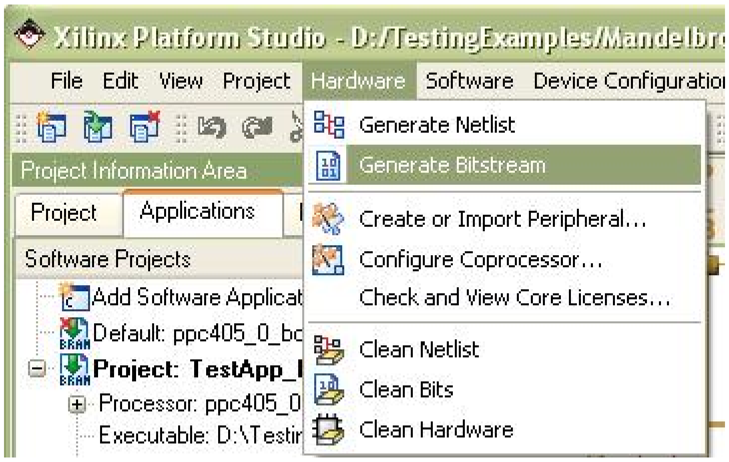
Software menu:
- Launch Platform Studio SDK
- Software Platform Settings...
- Assign Default Drivers
- Generate Libraries and BSPs
- Add Software Application Project...
- Build All User Applications
- Get Program Size
- Generate Linker Script...
- Clean Libraries
- Clean Programs
- Clean Software

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

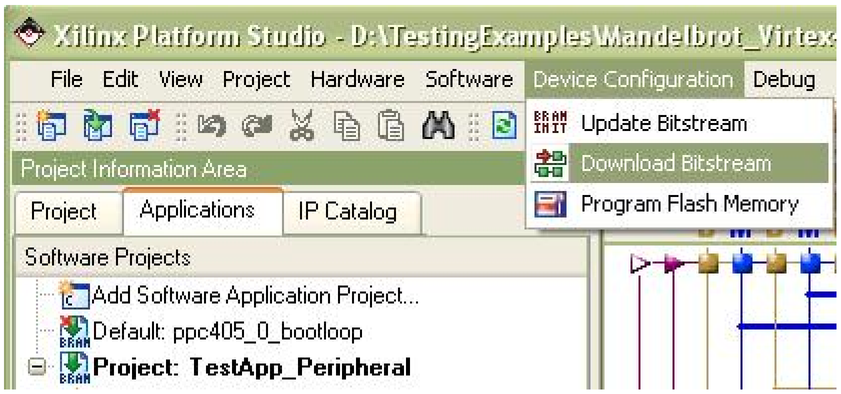


```
Libraries generated in
D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\ppc405_0\lib\ directory

Running execs_generate for OS'es, Drivers and Libraries ...

LibGen Done.
Done!
```

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 at the lower right corner of the board 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

1.7 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.

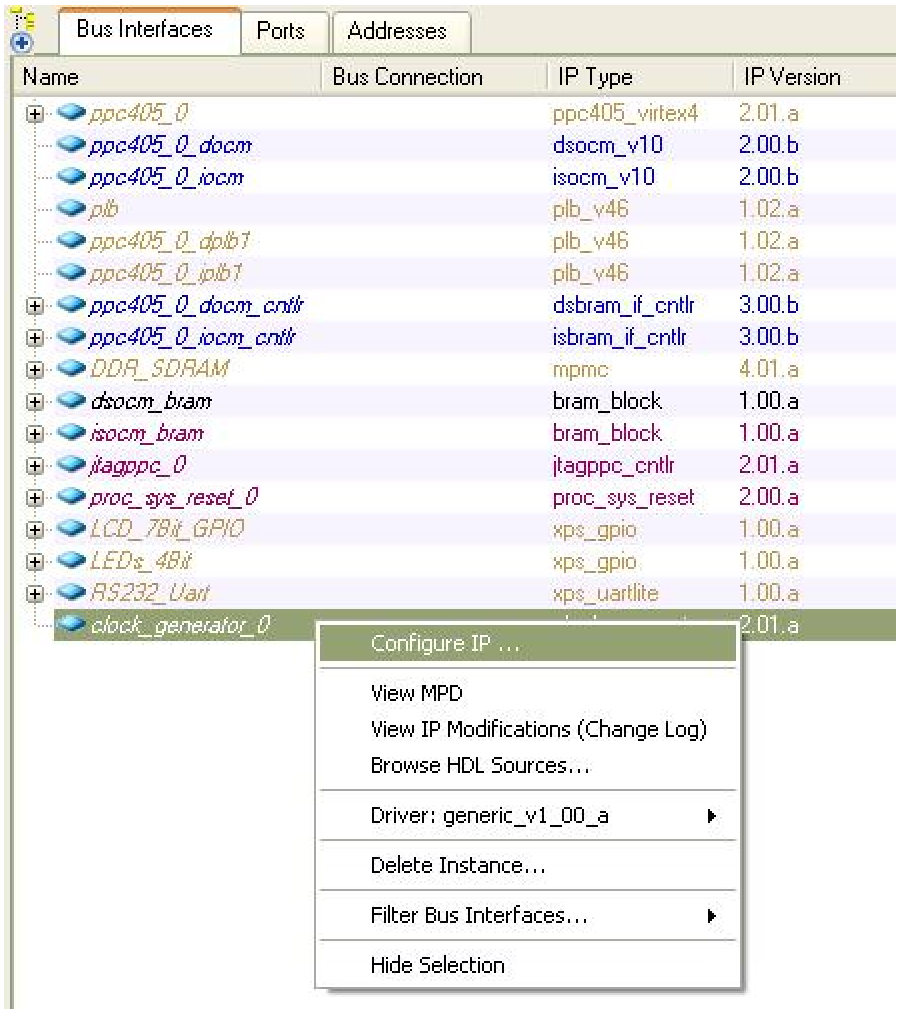
In the steps that remain, we will modify the base platform to:

- 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

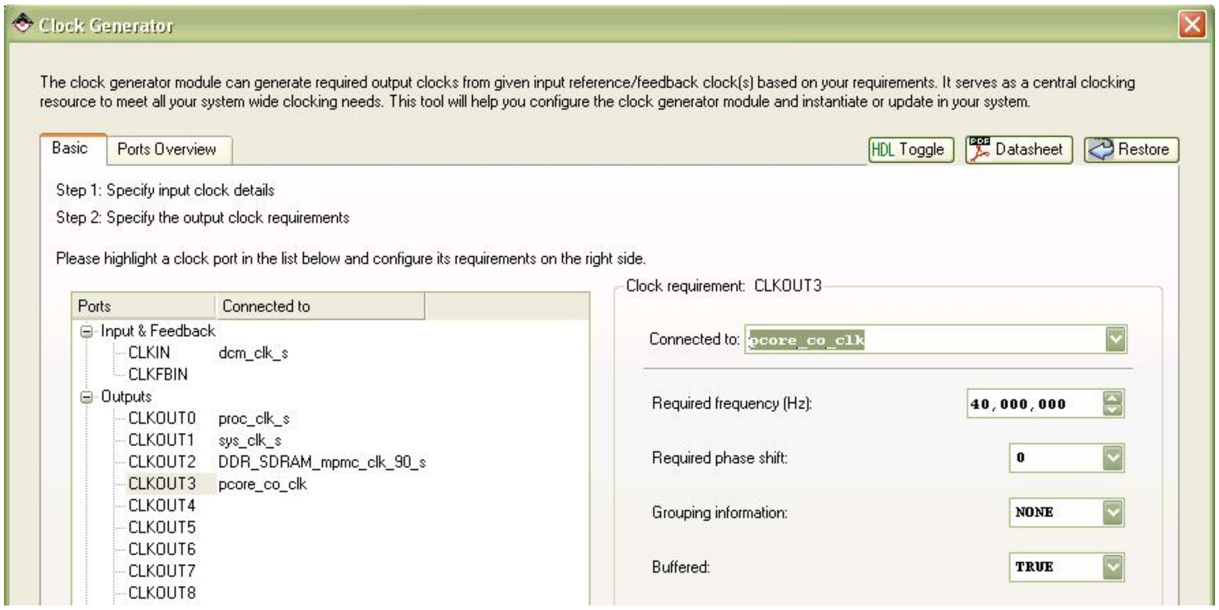
Our fractal image generator application requires three distinct clock sources, one for the **PowerPC** processor, one for the **Fabric Co-processor Bus (FCB)**, and one for the hardware accelerator, which in this example runs at **40 MHz**. The **TFT Controller** needs a **25 MHz** clock.

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



Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.02.a
ppc405_0_dplb1		plb_v46	1.02.a
ppc405_0_iplb1		plb_v46	1.02.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.01.a
dsocm_bram		bram_block	1.00.a
isocm_bram		bram_block	1.00.a
jtagppc_0		jtagppc_cntlr	2.01.a
proc_sys_reset_0		proc_sys_reset	2.00.a
LCD_7Bit_GPIO		xps_gpio	1.00.a
LEDs_4Bit		xps_gpio	1.00.a
RS232_Uart		xps_uartlite	1.00.a
clock_generator_0		[illegible]	2.01.a

Add a new clock output in **CLKOUT3**, type in the name **pcore_co_clk**, and frequency as **40,000,000 Hz** as shown:



The clock generator module can generate required output clocks from given input reference/feedback clock(s) based on your requirements. It serves as a central clocking resource to meet all your system wide clocking needs. This tool will help you configure the clock generator module and instantiate or update in your system.

Step 1: Specify input clock details
Step 2: Specify the output clock requirements

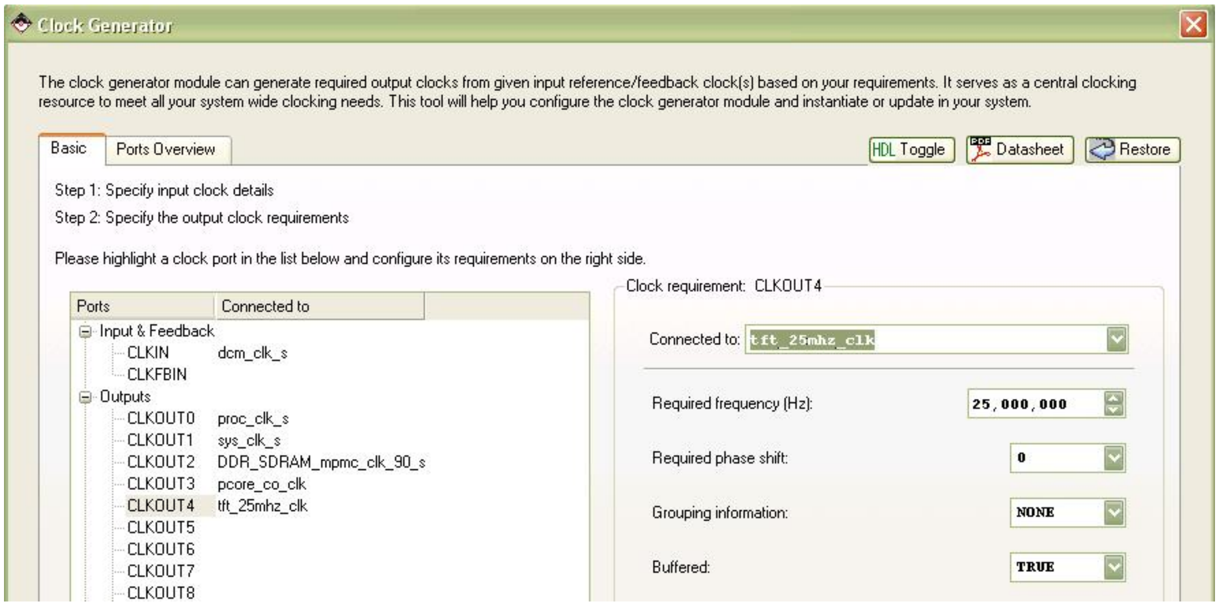
Please highlight a clock port in the list below and configure its requirements on the right side.

Ports	Connected to
Input & Feedback	
CLKIN	dcm_clk_s
CLKFBIN	
Outputs	
CLKOUT0	proc_clk_s
CLKOUT1	sys_clk_s
CLKOUT2	DDR_SDRAM_mpmc_clk_90_s
CLKOUT3	pcore_co_clk
CLKOUT4	
CLKOUT5	
CLKOUT6	
CLKOUT7	
CLKOUT8	

Clock requirement: CLKOUT3

Connected to: pcore_co_clk
Required frequency (Hz): 40,000,000
Required phase shift: 0
Grouping information: NONE
Buffered: TRUE

Add another clock output in **CLKOUT4**, type in the name **tft_25mhz_clk**, and frequency as **25,000,000 Hz** as shown:



The clock generator module can generate required output clocks from given input reference/feedback clock(s) based on your requirements. It serves as a central clocking resource to meet all your system wide clocking needs. This tool will help you configure the clock generator module and instantiate or update in your system.

Step 1: Specify input clock details
Step 2: Specify the output clock requirements

Please highlight a clock port in the list below and configure its requirements on the right side.

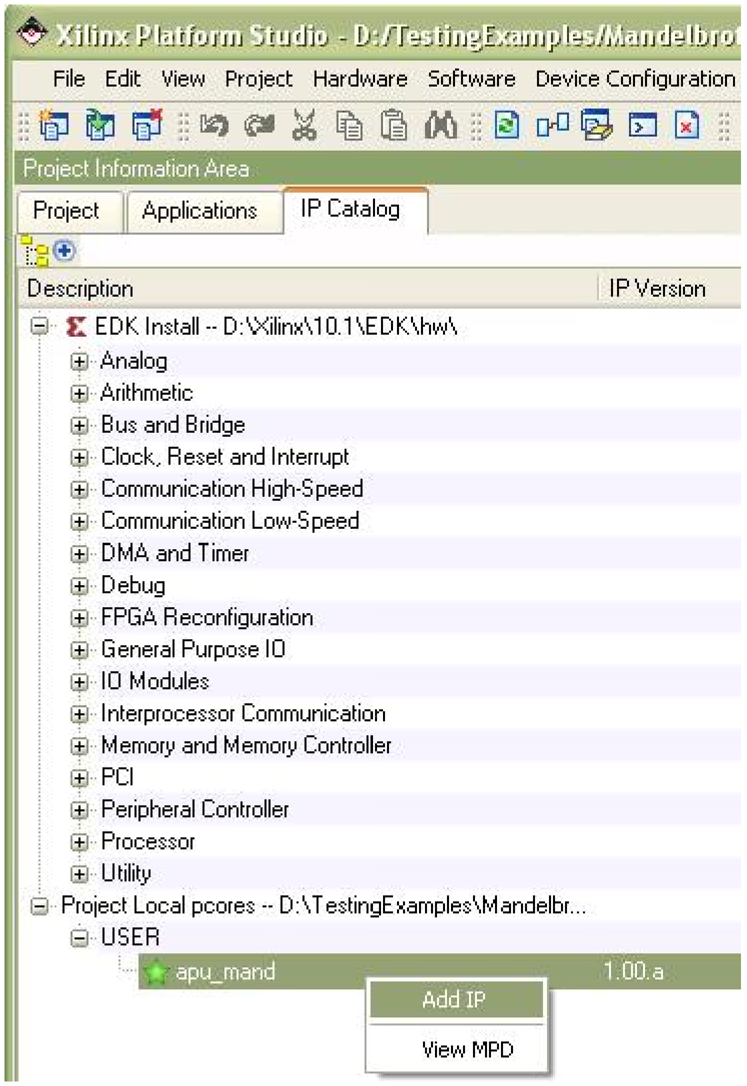
Ports	Connected to
Input & Feedback	
CLKIN	dcm_clk_s
CLKFBIN	
Outputs	
CLKOUT0	proc_clk_s
CLKOUT1	sys_clk_s
CLKOUT2	DDR_SDRAM_mpmc_clk_90_s
CLKOUT3	pcore_co_clk
CLKOUT4	tft_25mhz_clk
CLKOUT5	
CLKOUT6	
CLKOUT7	
CLKOUT8	

Clock requirement: CLKOUT4

Connected to: tft_25mhz_clk
Required frequency (Hz): 25,000,000
Required phase shift: 0
Grouping information: NONE
Buffered: TRUE

Adding the Mandelbrot Accelerator Core

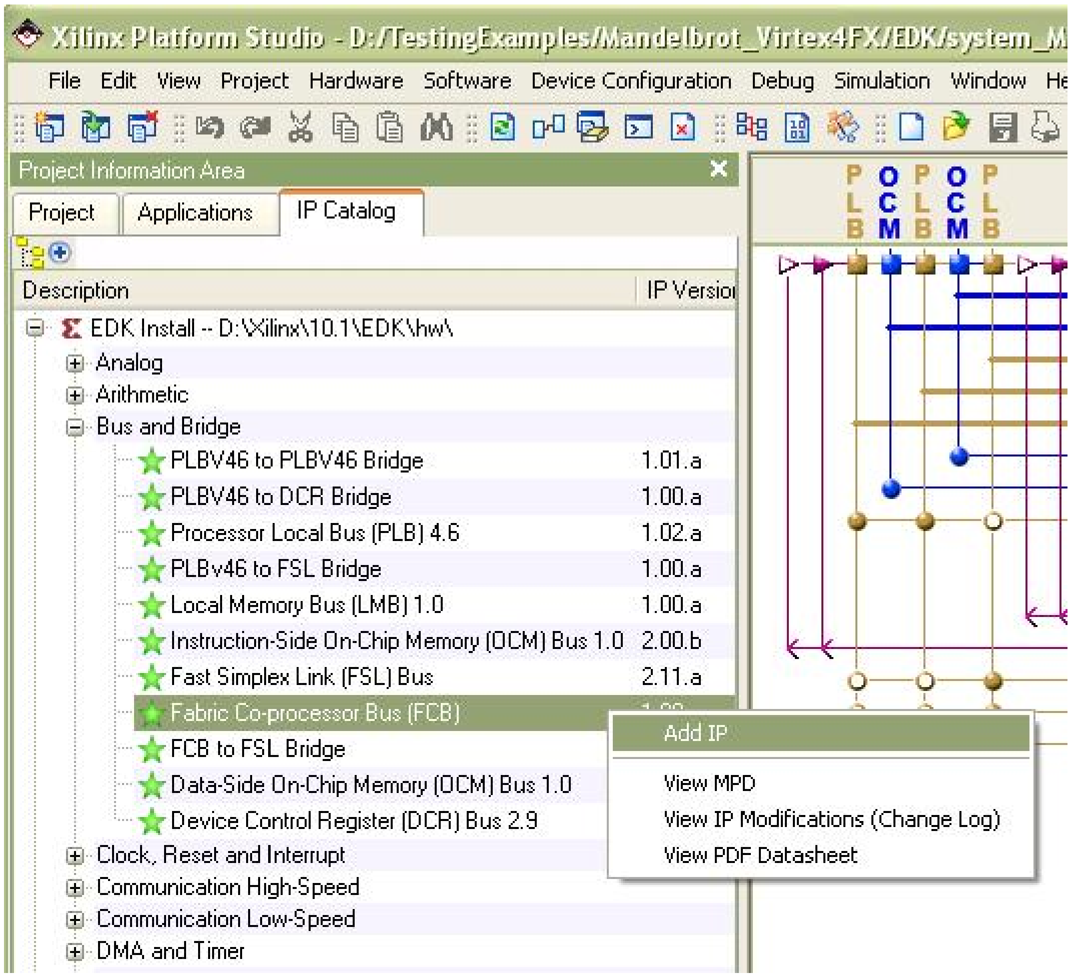
To add the Mandelbrot fractal image generator core as a peripheral, select the IP Catalog tab and look for the category titled **Project Local pcores**. Under **USER** directory you will find the two cores that were created (copied to) the **EDK/pcores** directory of your project. Add the **apu_mand**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


Xilinx Platform Studio - D:/TestingExamples/Mandelbro
File Edit View Project Hardware Software Device Configuration
Project Information Area
Project Applications IP Catalog
Description IP Version
EDK Install -- D:\Xilinx\10.1\EDK\hw\
Analog
Arithmetic
Bus and Bridge
Clock, Reset and Interrupt
Communication High-Speed
Communication Low-Speed
DMA and Timer
Debug
FPGA Reconfiguration
General Purpose IO
IO Modules
Interprocessor Communication
Memory and Memory Controller
PCI
Peripheral Controller
Processor
Utility
Project Local pcores -- D:\TestingExamples\Mandelbr...
USER
apu_mand 1.00.a
Add IP
View MPD

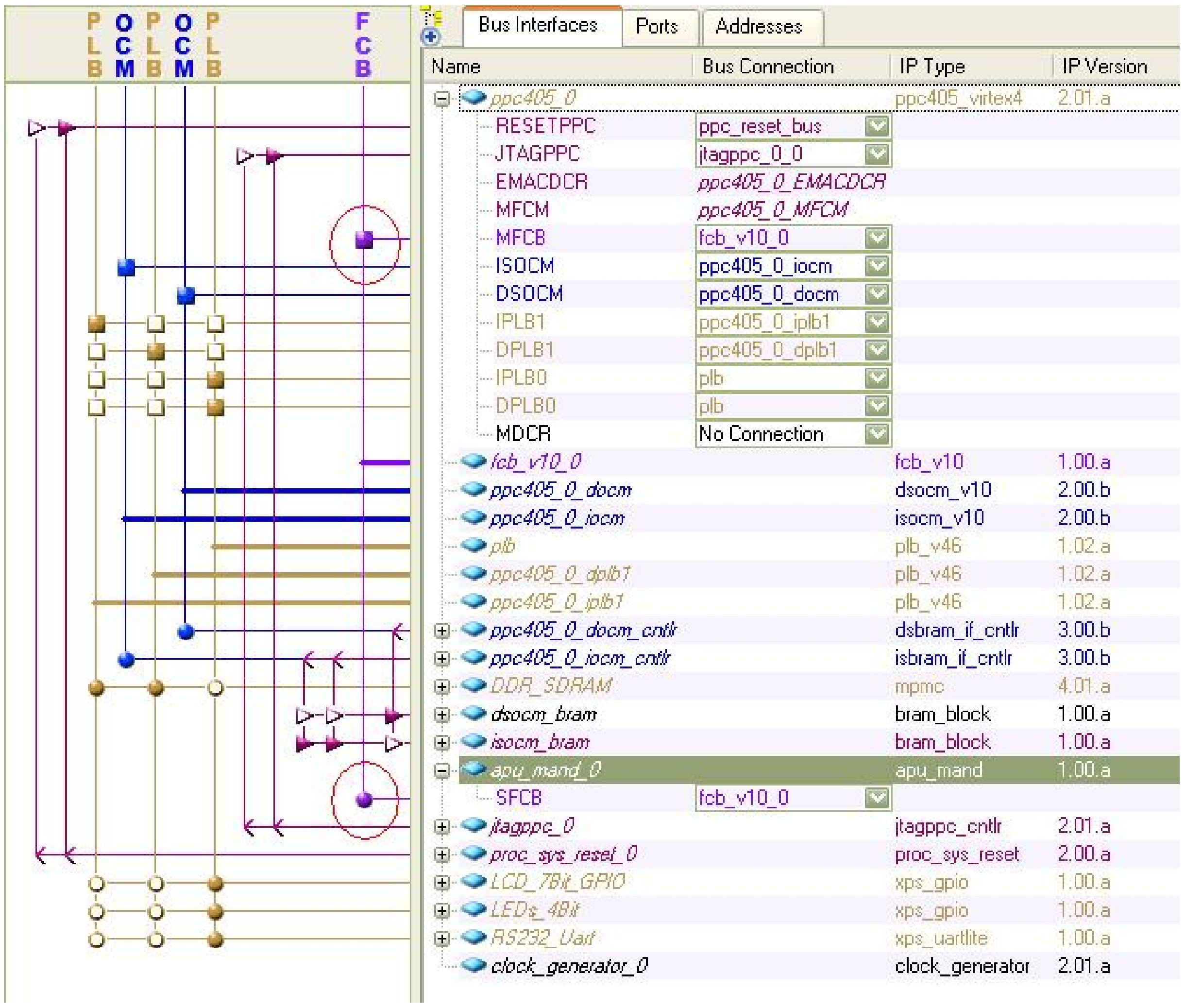
This will add the core to the project as a peripheral.

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 and Bridge** category and find the **Fabric Co-processor Bus** item. Add the **FCB** to your system by by right-clicking and selecting **Add IP** as shown below:


Xilinx Platform Studio - D:/TestingExamples/Mandelbrot_Virtex4FX/EDK/system_M
File Edit View Project Hardware Software Device Configuration Debug Simulation Window He
Project Information Area
Project Applications IP Catalog
Description IP Versio
EDK Install -- D:\Xilinx\10.1\EDK\hw\
Analog
Arithmetic
Bus and Bridge
PLBV46 to PLBV46 Bridge 1.01.a
PLBV46 to DCR Bridge 1.00.a
Processor Local Bus (PLB) 4.6 1.02.a
PLBv46 to FSL Bridge 1.00.a
Local Memory Bus (LMB) 1.0 1.00.a
Instruction-Side On-Chip Memory (OCM) Bus 1.0 2.00.b
Fast Simplex Link (FSL) Bus 2.11.a
Fabric Co-processor Bus (FCB)
FCB to FSL Bridge
Data-Side On-Chip Memory (OCM) Bus 1.0
Device Control Register (DCR) Bus 2.9
Clock, Reset and Interrupt
Communication High-Speed
Communication Low-Speed
DMA and Timer
Add IP
View MPD
View IP Modifications (Change Log)
View PDF Datasheet
PLB OCM PLB OCM PLB

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):



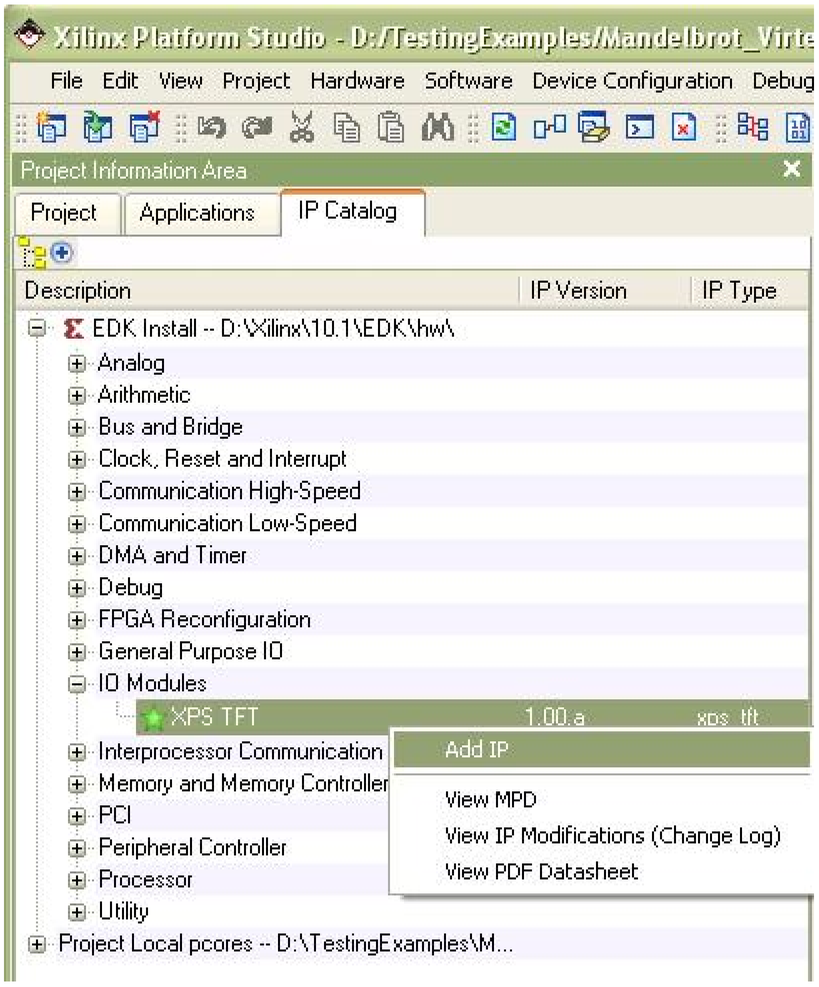
Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
RESETPPC	ppc_reset_bus		
JTAGPPC	jtagppc_0_0		
EMACDCR	ppc405_0_EMACDCR		
MFCM	ppc405_0_MFCM		
MFCB	fcb_v10_0		
ISOCM	ppc405_0_iocm		
DSOCM	ppc405_0_docm		
IPLB1	ppc405_0_iplb1		
DPLB1	ppc405_0_dplb1		
IPLB0	plb		
DPLB0	plb		
MDCR	No Connection		
fcb_v10_0		fcb_v10	1.00.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.02.a
ppc405_0_dplb1		plb_v46	1.02.a
ppc405_0_iplb1		plb_v46	1.02.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.01.a
dsocm_bram		bram_block	1.00.a
isocm_bram		bram_block	1.00.a
apu_mand_0		apu_mand	1.00.a
SFCB	fcb_v10_0		
jtagppc_0		jtagppc_cntlr	2.01.a
proc_sys_reset_0		proc_sys_reset	2.00.a
LCD_7Bit_GPIO		xps_gpio	1.00.a
LEDs_4Bit		xps_gpio	1.00.a
RS232_Uart		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a

The **apu_mand_0** peripheral is now connected to the **PowerPC APU** via the **FCB.**

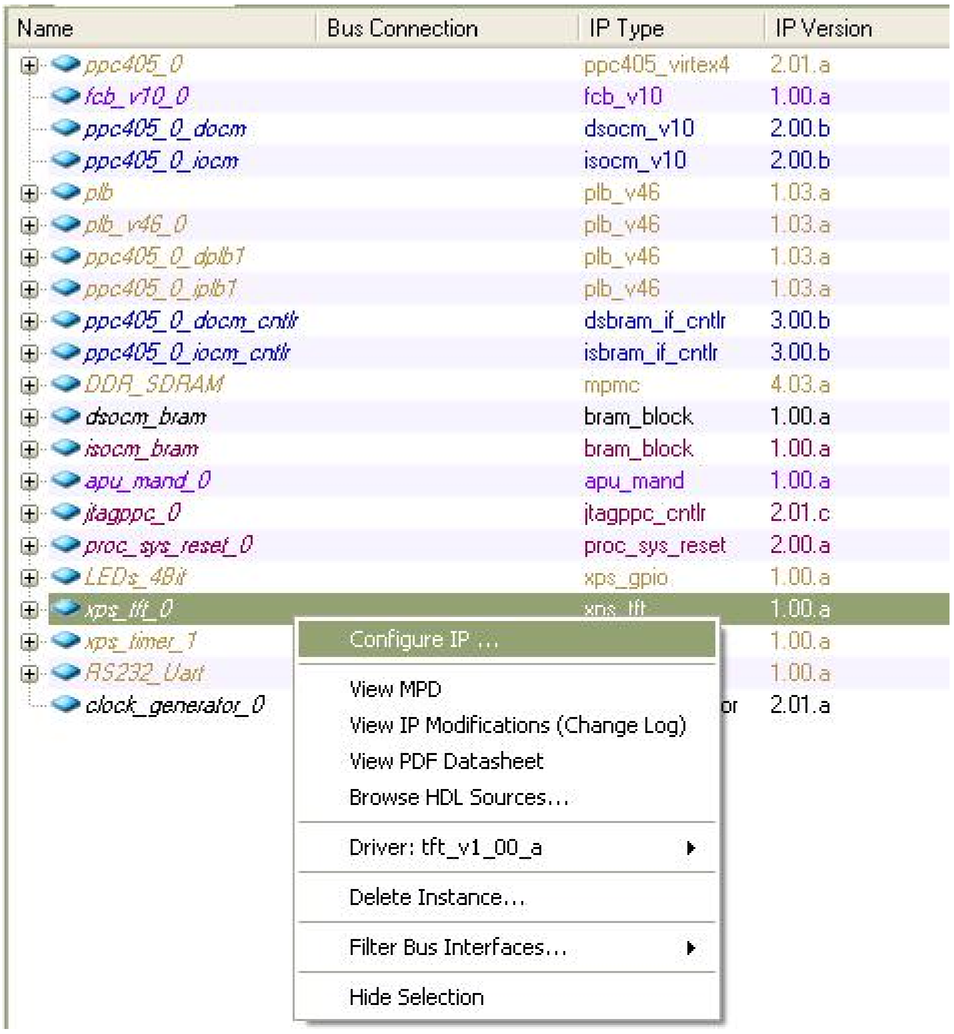
Adding the XPS_TFT IP Core

The **XPS_TFT IP Core** controls the **Thin Film Transistor (TFT) LCD Display**, which gives us the graphical output of the computation results.

The **XPS_TFT IP Core** is located under the **IO Modules** category of the **IP Catalog**. Add it by by right-clicking and selecting **Add IP** as shown below:

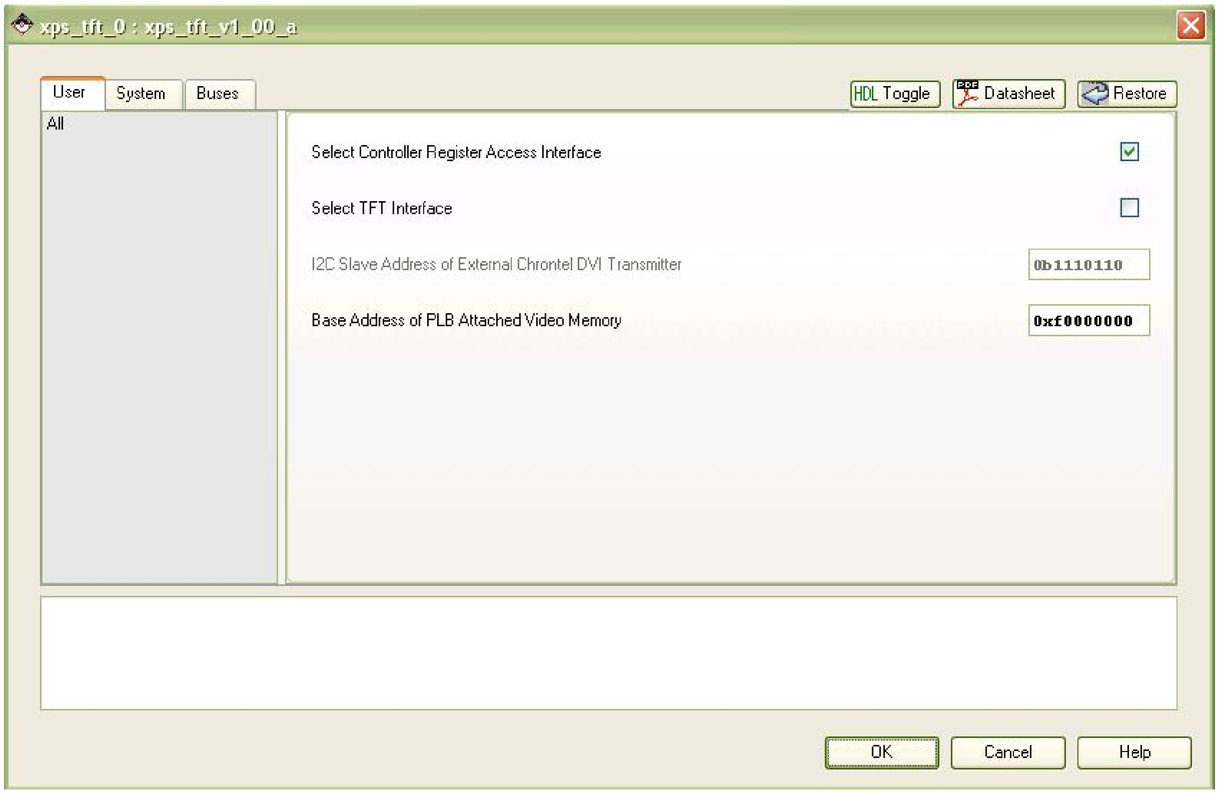

Xilinx Platform Studio - D:/TestingExamples/Mandelbrot_Virte
File Edit View Project Hardware Software Device Configuration Debug
Project Information Area
Project
Applications
IP Catalog
Description
IP Version
IP Type
EDK Install -- D:\Xilinx\10.1\EDK\hw\
Analog
Arithmetic
Bus and Bridge
Clock, Reset and Interrupt
Communication High-Speed
Communication Low-Speed
DMA and Timer
Debug
FPGA Reconfiguration
General Purpose IO
IO Modules
XPS TFT
1.00.a
xps_tft
Interprocessor Communication
Memory and Memory Controller
PCI
Peripheral Controller
Processor
Utility
Project Local pcores -- D:\TestingExamples\M...
Add IP
View MPD
View IP Modifications (Change Log)
View PDF Datasheet

Next, edit the settings of the newly added **xps_tft_0** module by right-clicking and selecting **Configure IP ...**



Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
fcb_v10_0		fcb_v10	1.00.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.03.a
plb_v46_0		plb_v46	1.03.a
ppc405_0_dplb1		plb_v46	1.03.a
ppc405_0_iplb1		plb_v46	1.03.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.03.a
dsocm_bram		bram_block	1.00.a
isocm_bram		bram_block	1.00.a
apu_mand_0		apu_mand	1.00.a
jtagppc_0		jtagppc_cntlr	2.01.c
proc_sys_reset_0		proc_sys_reset	2.00.a
LEDs_4Bit		xps_gpio	1.00.a
xps_tft_0		xps_tft	1.00.a
xps_timer_1			1.00.a
RS232_Uart			1.00.a
clock_generator_0		or	2.01.a

In order to expose the **VGA** interface, unselect the **Select TFT Interface** option.



Select Controller Register Access Interface ✓

Select TFT Interface

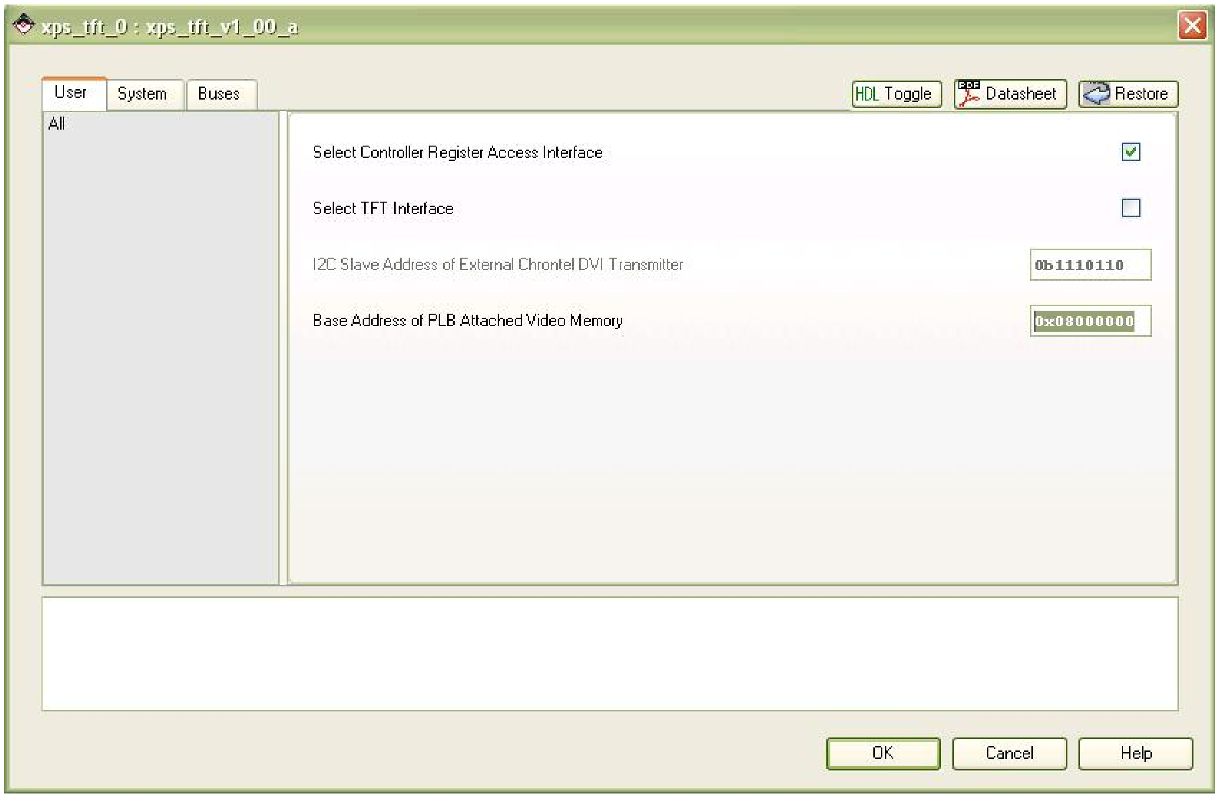
I2C Slave Address of External Chrontel DVI Transmitter 0b1110110

Base Address of PLB Attached Video Memory 0xf0000000

Modifying the TFT Base Address Parameter

The **Base Address of PLB Attached Video Memory** parameter sets the starting address of the **TFT** image memory. This address is the key point to have the **TFT LCD** display properly, and you will need to set the corresponding value in the software code. (See **xtft_main.c**, line 174.)

Change the **Base Address of PLB Attached Video Memory** to **0x08000000** as shown below:



Select Controller Register Access Interface [✓]

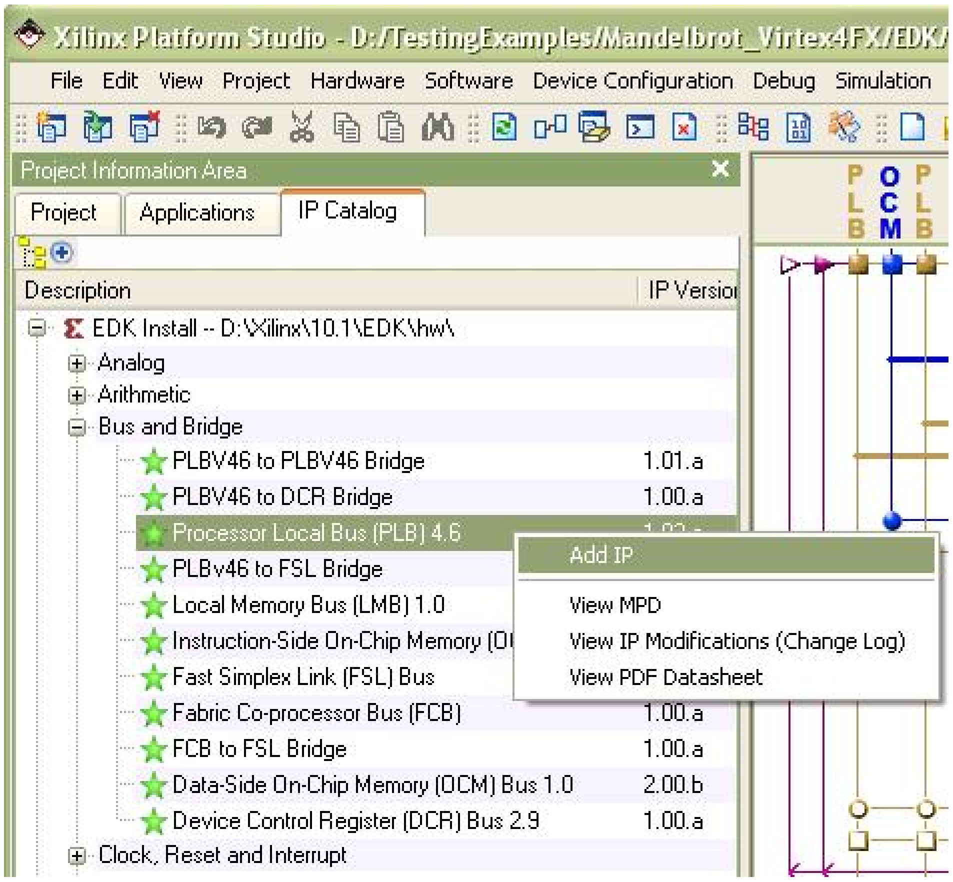
Select TFT Interface [ ]

I2C Slave Address of External Chrontel DVI Transmitter 0b1110110

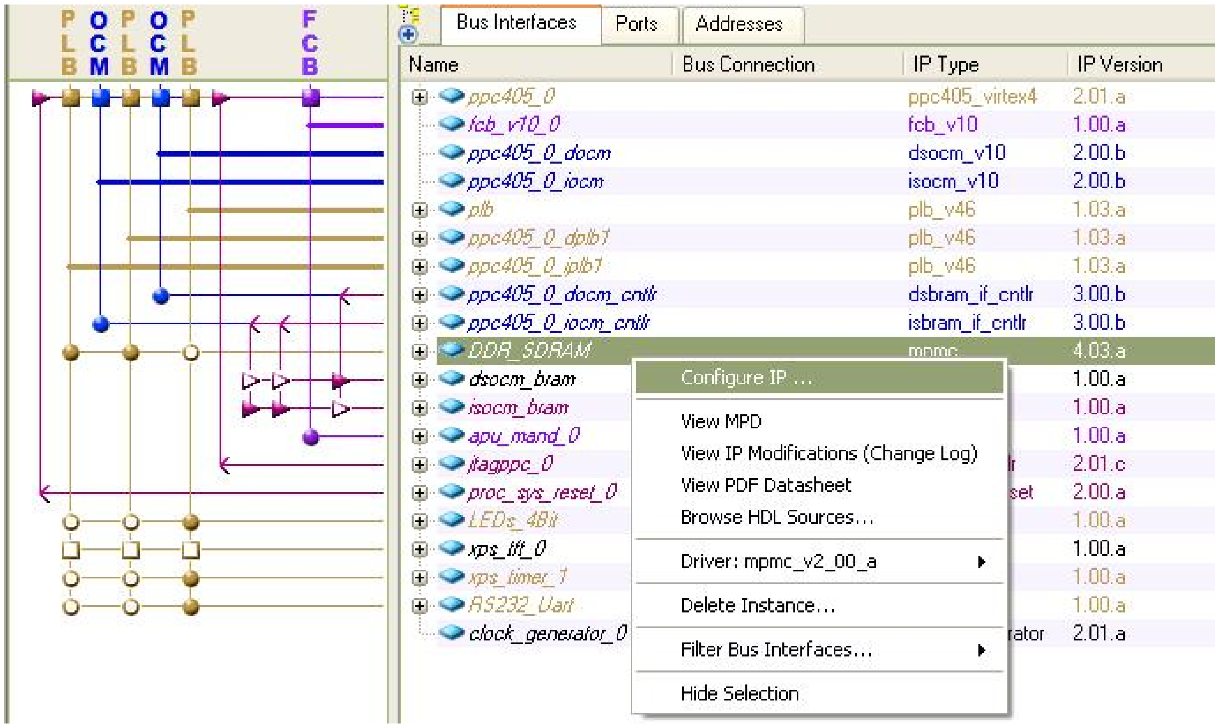
Base Address of PLB Attached Video Memory 0x08000000

Clink **OK** to save the change.

The **XPS_TFT** has a **Master PLB** and a **Slave PLB** bus interface. We need to add an additional **PLB** bus to connect the **XPS_TFT** to the **DDR_SDRAM**. Add a **Processor Local Bus (PLB)** as shown below:

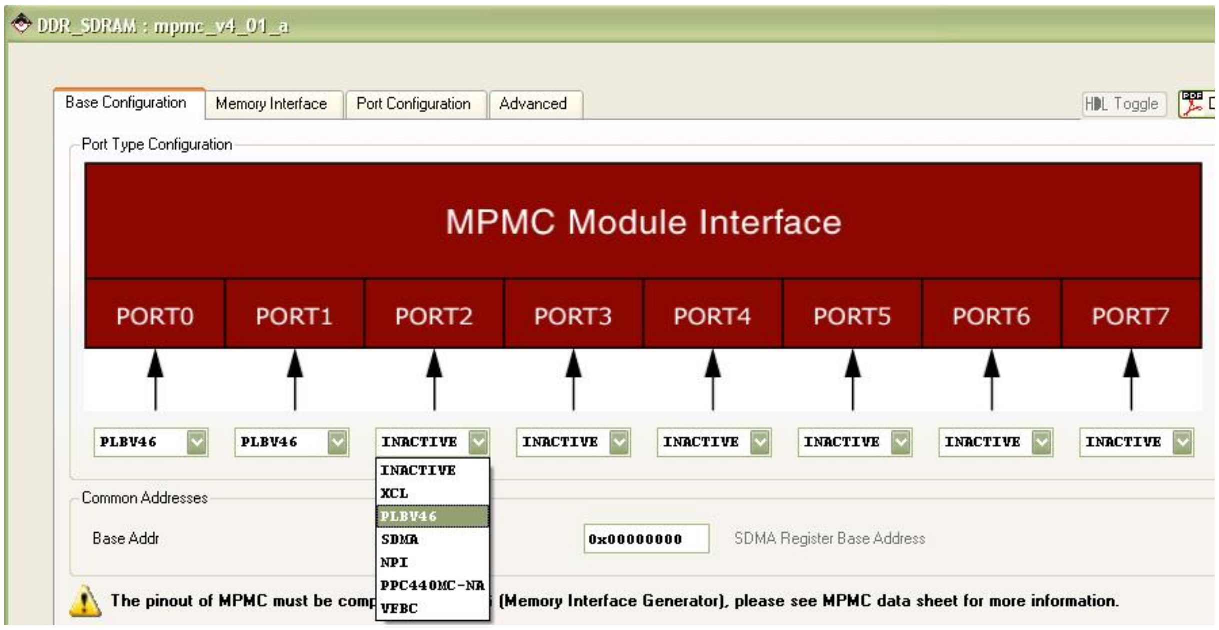

Xilinx Platform Studio - D:/TestingExamples/Mandelbrot_Virtex4FX/EDK/
File Edit View Project Hardware Software Device Configuration Debug Simulation
Project Information Area
Project
Applications
IP Catalog
Description
IP Version
EDK Install -- D:\Xilinx\10.1\EDK\hw\
Analog
Arithmetic
Bus and Bridge
PLBV46 to PLBV46 Bridge 1.01.a
PLBV46 to DCR Bridge 1.00.a
Processor Local Bus (PLB) 4.6
PLBv46 to FSL Bridge
Local Memory Bus (LMB) 1.0
Instruction-Side On-Chip Memory (O
Fast Simplex Link (FSL) Bus
Fabric Co-processor Bus (FCB) 1.00.a
FCB to FSL Bridge 1.00.a
Data-Side On-Chip Memory (OCM) Bus 1.0 2.00.b
Device Control Register (DCR) Bus 2.9 1.00.a
Clock, Reset and Interrupt
Add IP
View MPD
View IP Modifications (Change Log)
View PDF Datasheet
PLB
OCM
PLB

The **MPLB** of **xps_tft_0** needs to be connected to the **DDR_SDRAM** through a separate **SPLB** port. To do this, open the **Configure IP** dialog of the **DDR_SDRAM**:



Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
fcb_v10_0		fcb_v10	1.00.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.03.a
ppc405_0_dplb1		plb_v46	1.03.a
ppc405_0_iplb1		plb_v46	1.03.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.03.a
docm_bram			1.00.a
isocm_bram			1.00.a
apu_mand_0			1.00.a
jtagppc_0			2.01.c
proc_sys_reset_0			2.00.a
LEDs_4Bit			1.00.a
xps_tft_0			1.00.a
xps_timer_1			1.00.a
RS232_Uart			1.00.a
clock_generator_0			2.01.a

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**.

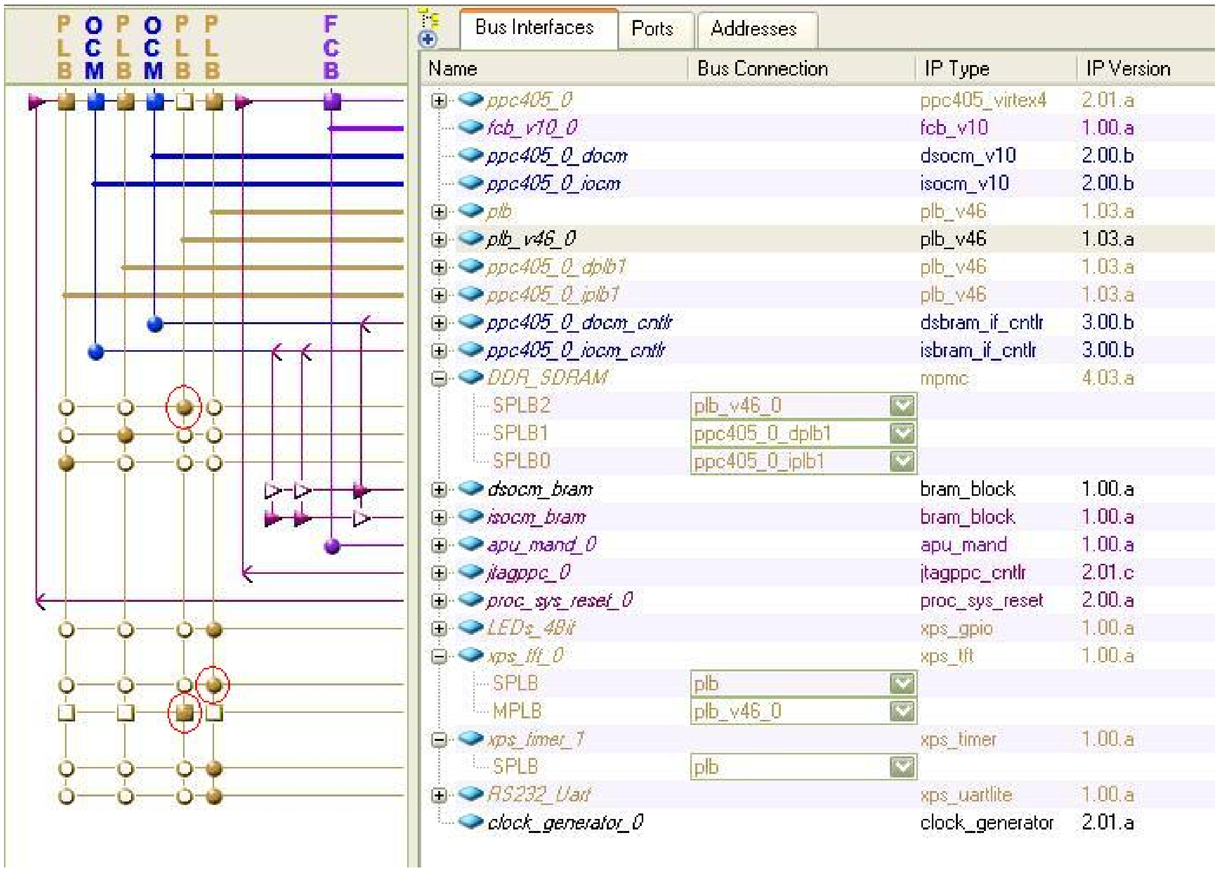


PORT0	PORT1	PORT2	PORT3	PORT4	PORT5	PORT6	PORT7
PLBV46	PLBV46	INACTIVE	INACTIVE	INACTIVE	INACTIVE	INACTIVE	INACTIVE

Base Addr 0x00000000 SDMA Register Base Address

The pinout of MPMC must be comp (Memory Interface Generator), please see MPMC data sheet for more information.

Then, connect the **MPLB** of the **xps_tft_0** to the newly added **PLB**, also conect the **SPLB2** of the **DDR_SDRAM** to the same **PLB**. Connect the **SPLB** of the **xps_tft_0** to the shared **PLB** as shown below (as indicated in red circles):



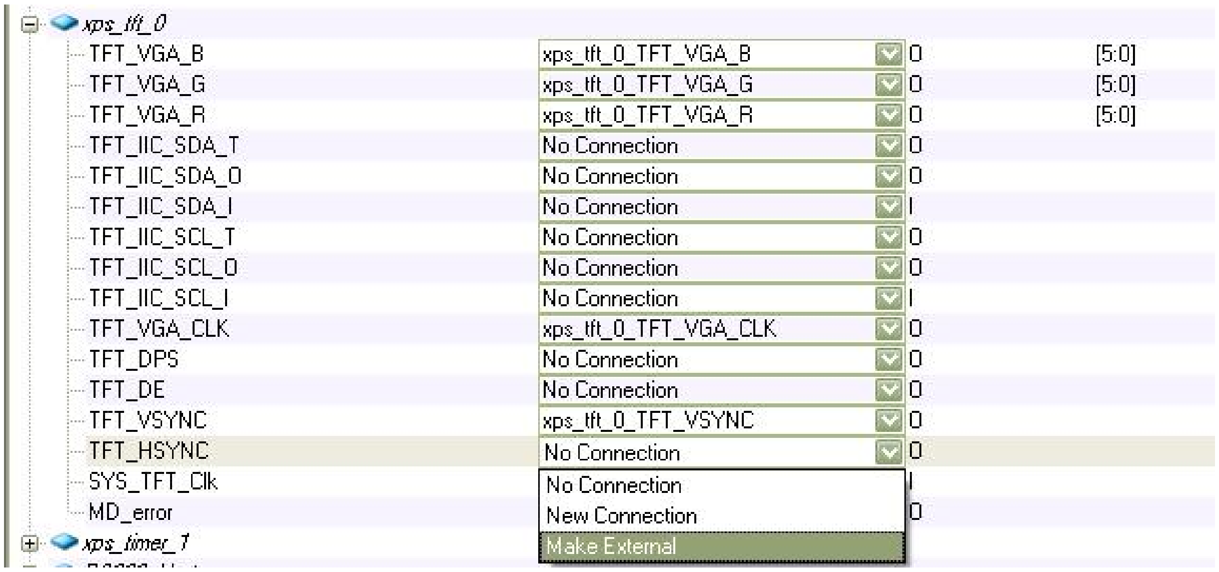
Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	2.01.a
fcb_v10_0		fcb_v10	1.00.a
ppc405_0_docm		dsocm_v10	2.00.b
ppc405_0_iocm		isocm_v10	2.00.b
plb		plb_v46	1.03.a
plb_v46_0		plb_v46	1.03.a
ppc405_0_dplb1		plb_v46	1.03.a
ppc405_0_iplb1		plb_v46	1.03.a
ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.b
DDR_SDRAM		mpmc	4.03.a
SPLB2	plb_v46_0		
SPLB1	ppc405_0_dplb1		
SPLB0	ppc405_0_iplb1		
dsocm_bram		bram_block	1.00.a
isocm_bram		bram_block	1.00.a
apu_mand_0		apu_mand	1.00.a
jtagppc_0		jtagppc_cntlr	2.01.c
proc_sys_reset_0		proc_sys_reset	2.00.a
LEDs_4Bit		xps_gpio	1.00.a
xps_tft_0		xps_tft	1.00.a
SPLB	plb		
MPLB	plb_v46_0		
xps_timer_1		xps_timer	1.00.a
SPLB	plb		
RS232_Uart		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a

Connecting the Peripheral Clock and Reset Signals

To do this, switch to the **Ports** tab in the **System Assembly View** window.

First, connect the following 6 ports of the **xps_tft_0** to outside of the FPGA by select the **Make External** for each port as shown below:

TFT_VGA_B
TFT_VGA_G
TFT_VGA_R
TFT_VGA_CLK
TFT_VSYNC
TFT_HSYNC

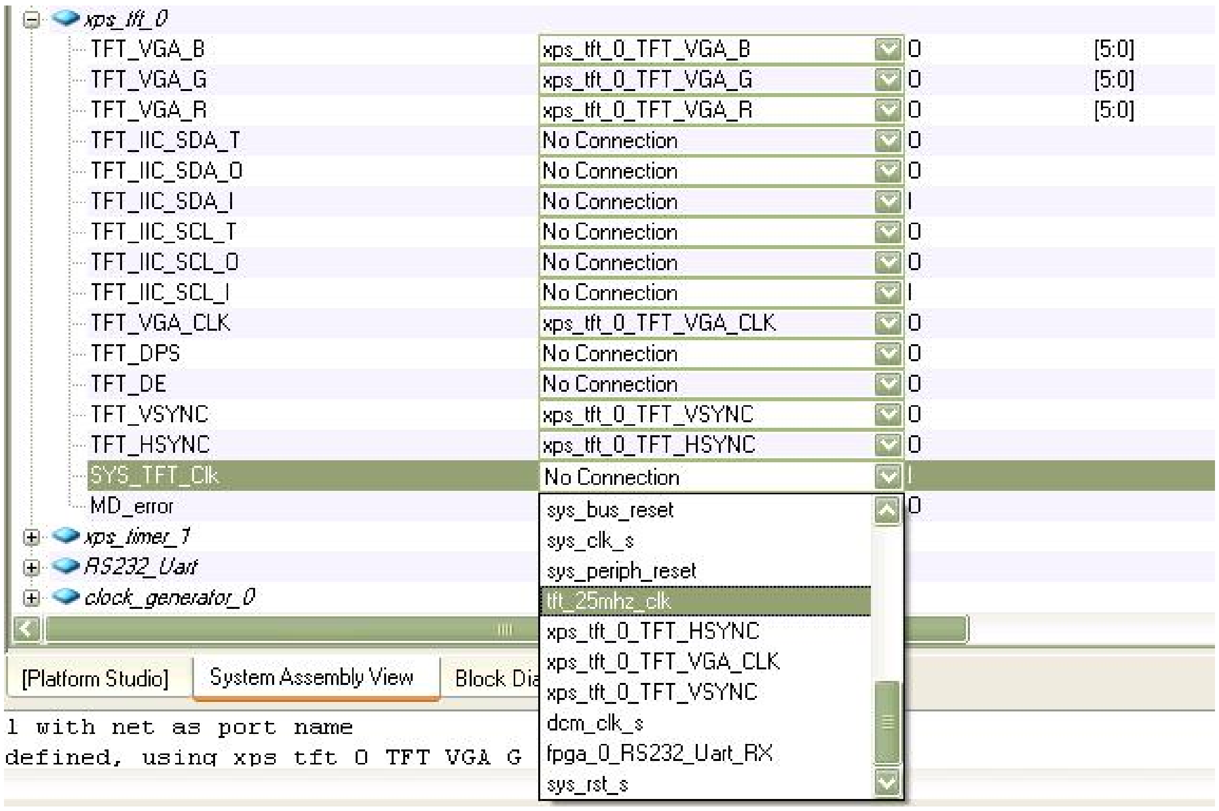


xps_tft_0

TFT_VGA_B	xps_tft_0_TFT_VGA_B	O	[5:0]
TFT_VGA_G	xps_tft_0_TFT_VGA_G	O	[5:0]
TFT_VGA_R	xps_tft_0_TFT_VGA_R	O	[5:0]
TFT_IIC_SDA_T	No Connection	O	
TFT_IIC_SDA_O	No Connection	O	
TFT_IIC_SDA_I	No Connection	I	
TFT_IIC_SCL_T	No Connection	O	
TFT_IIC_SCL_O	No Connection	O	
TFT_IIC_SCL_I	No Connection	I	
TFT_VGA_CLK	xps_tft_0_TFT_VGA_CLK	O	
TFT_DPS	No Connection	O	
TFT_DE	No Connection	O	
TFT_VSYNC	xps_tft_0_TFT_VSYNC	O	
TFT_HSYNC	No Connection	O	
SYS_TFT_Clk		I	
MD_error		O	

xps_timer_1

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



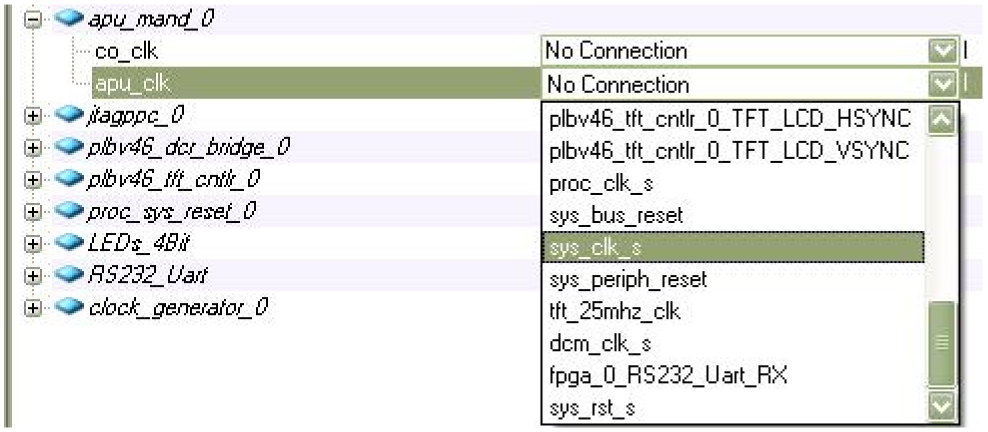
xps_tft_0

TFT_VGA_B	xps_tft_0_TFT_VGA_B	O	[5:0]
TFT_VGA_G	xps_tft_0_TFT_VGA_G	O	[5:0]
TFT_VGA_R	xps_tft_0_TFT_VGA_R	O	[5:0]
TFT_IIC_SDA_T	No Connection	O	
TFT_IIC_SDA_O	No Connection	O	
TFT_IIC_SDA_I	No Connection	I	
TFT_IIC_SCL_T	No Connection	O	
TFT_IIC_SCL_O	No Connection	O	
TFT_IIC_SCL_I	No Connection	I	
TFT_VGA_CLK	xps_tft_0_TFT_VGA_CLK	O	
TFT_DPS	No Connection	O	
TFT_DE	No Connection	O	
TFT_VSYNC	xps_tft_0_TFT_VSYNC	O	
TFT_HSYNC	xps_tft_0_TFT_HSYNC	O	
SYS_TFT_Clk	No Connection	I	
MD_error		O	

xps_timer_1
RS232_Uart
clock_generator_0

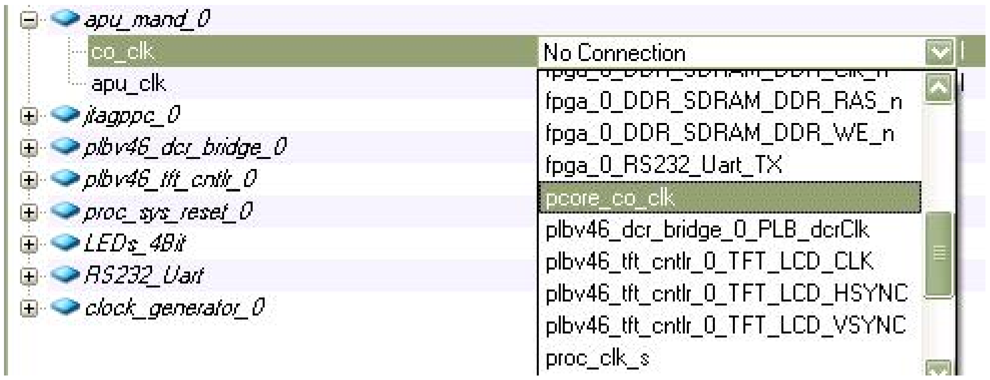
```
l with net as port name
defined, using xps tft 0 TFT VGA G
```

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:



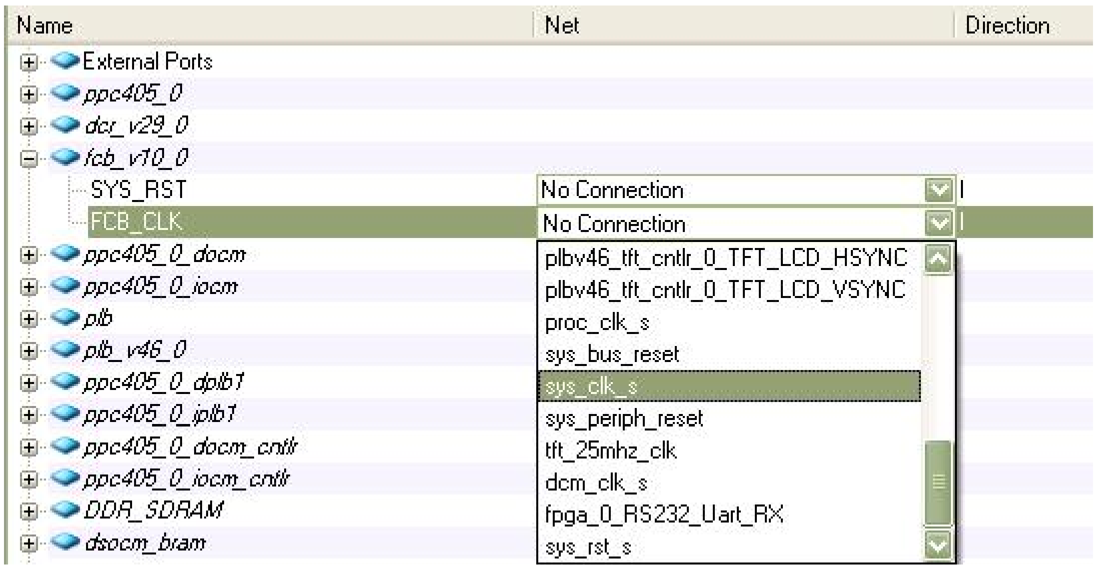
```
⊟ apu_mand_0
    co_clk                  No Connection
    apu_clk                 No Connection
⊞ jtagppc_0                 plbv46_tft_cntlr_0_TFT_LCD_HSYNC
⊞ plbv46_dcr_bridge_0       plbv46_tft_cntlr_0_TFT_LCD_VSYNC
⊞ plbv46_tft_cntlr_0        proc_clk_s
⊞ proc_sys_reset_0          sys_bus_reset
⊞ LEDs_4Bit                 sys_clk_s
⊞ RS232_Uart                sys_periph_reset
⊞ clock_generator_0         tft_25mhz_clk
                            dcm_clk_s
                            fpga_0_RS232_Uart_RX
                            sys_rst_s
```

Now change the **co_clk** entry to **pcore_co_clk** as shown below:



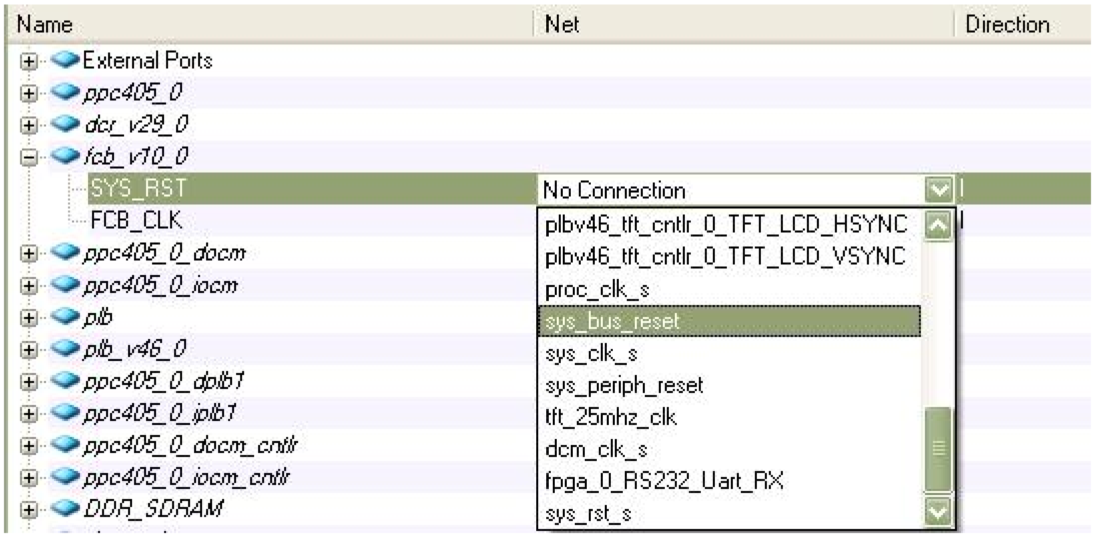
```
⊟ apu_mand_0
    co_clk                  No Connection
    apu_clk                 fpga_0_DDR_SDRAM_DDR_RAS_n
⊞ jtagppc_0                 fpga_0_DDR_SDRAM_DDR_WE_n
⊞ plbv46_dcr_bridge_0       fpga_0_RS232_Uart_TX
⊞ plbv46_tft_cntlr_0        pcore_co_clk
⊞ proc_sys_reset_0          plbv46_dcr_bridge_0_PLB_dcrClk
⊞ LEDs_4Bit                 plbv46_tft_cntlr_0_TFT_LCD_CLK
⊞ RS232_Uart                plbv46_tft_cntlr_0_TFT_LCD_HSYNC
⊞ clock_generator_0         plbv46_tft_cntlr_0_TFT_LCD_VSYNC
                            proc_clk_s
```

Connect the **FCB_CLK** signal of the **fcb_v10_0** peripheral to **sys_clk_s**:



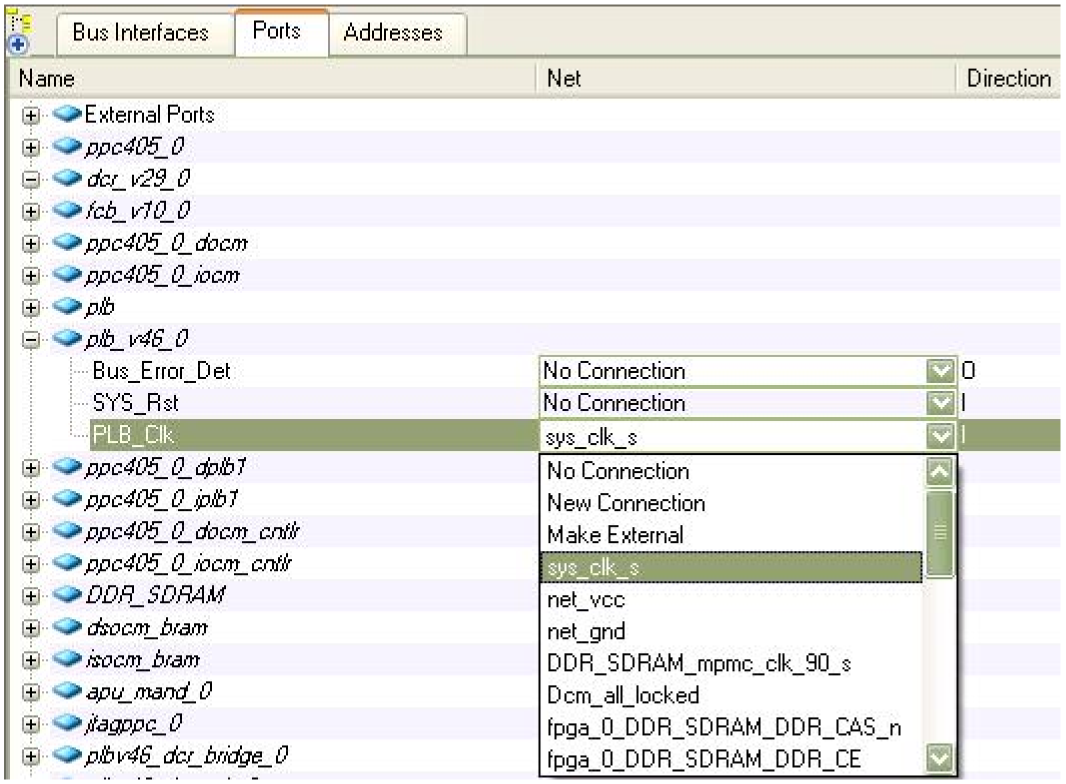
```
Name                        Net                                 Direction
⊞ External Ports
⊞ ppc405_0
⊞ dcr_v29_0
⊟ fcb_v10_0
    SYS_RST                 No Connection                       I
    FCB_CLK                 No Connection                       I
⊞ ppc405_0_docm             plbv46_tft_cntlr_0_TFT_LCD_HSYNC
⊞ ppc405_0_iocm             plbv46_tft_cntlr_0_TFT_LCD_VSYNC
⊞ plb                       proc_clk_s
⊞ plb_v46_0                 sys_bus_reset
⊞ ppc405_0_dplb1            sys_clk_s
⊞ ppc405_0_iplb1            sys_periph_reset
⊞ ppc405_0_docm_cntlr       tft_25mhz_clk
⊞ ppc405_0_iocm_cntlr       dcm_clk_s
⊞ DDR_SDRAM                 fpga_0_RS232_Uart_RX
⊞ dsocm_bram                sys_rst_s
```

And connect the **SYS_RST** signal of **fcb_v10_0** to sys_bus_reset:



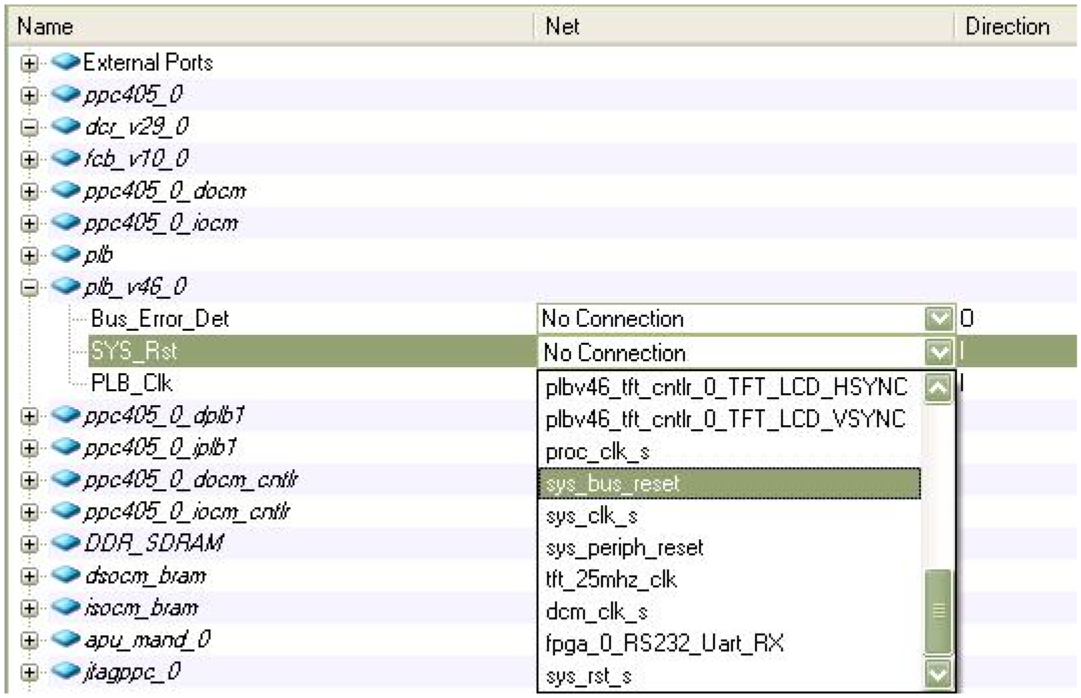
Name	Net	Direction
External Ports		
ppc405_0		
dcr_v29_0		
fcb_v10_0		
SYS_RST	No Connection	I
FCB_CLK		I
ppc405_0_docm		
ppc405_0_iocm		
plb		
plb_v46_0		
ppc405_0_dplb1		
ppc405_0_iplb1		
ppc405_0_docm_cntlr		
ppc405_0_iocm_cntlr		
DDR_SDRAM		

Connect the **PLB_Clk** signal of the **plb_v46_0** peripheral to **sys_clk_s**:



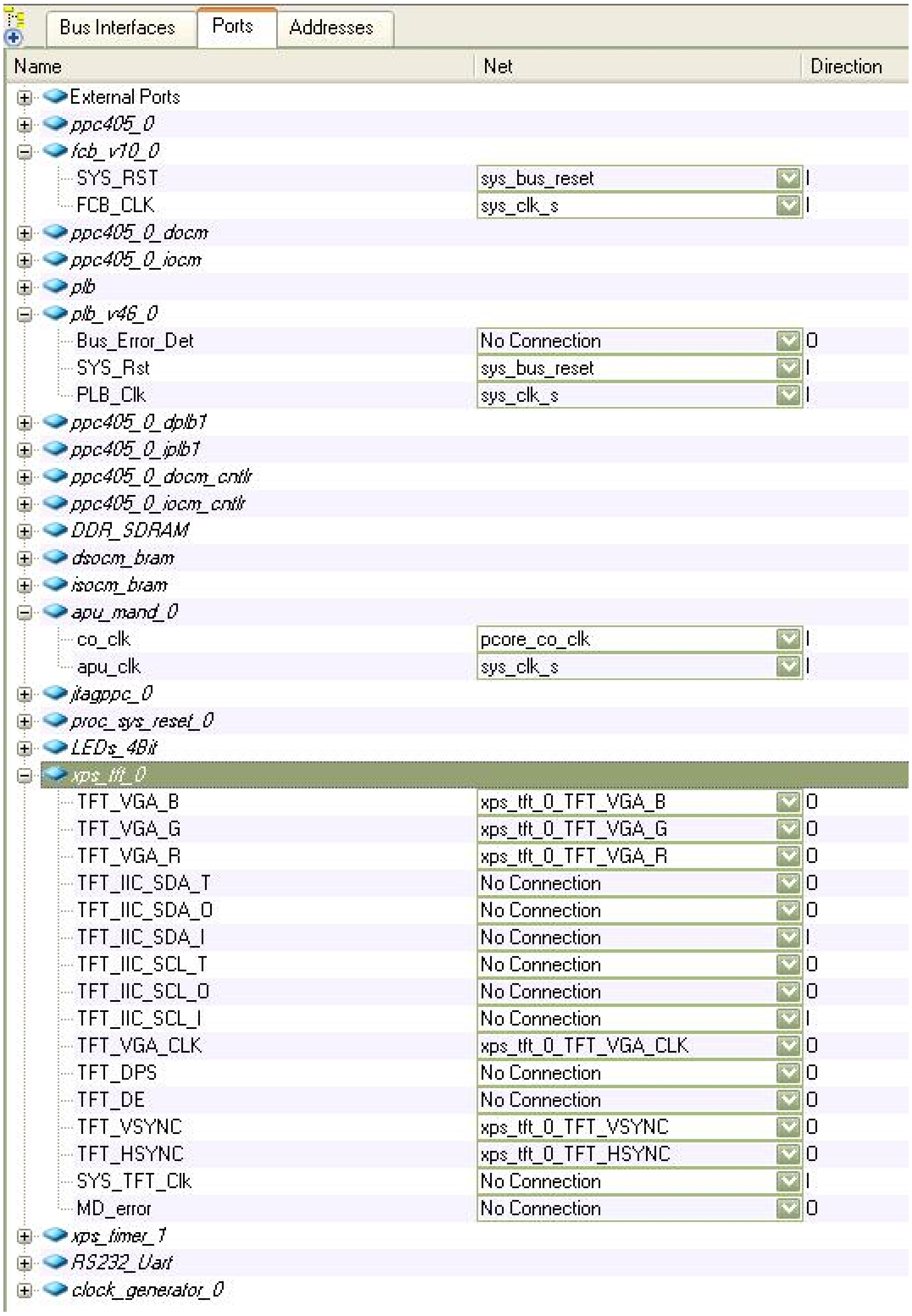
Name	Net	Direction
External Ports		
ppc405_0		
dcr_v29_0		
fcb_v10_0		
ppc405_0_docm		
ppc405_0_iocm		
plb		
plb_v46_0		
Bus_Error_Det	No Connection	O
SYS_Rst	No Connection	I
PLB_Clk	sys_clk_s	I
ppc405_0_dplb1		
ppc405_0_iplb1		
ppc405_0_docm_cntlr		
ppc405_0_iocm_cntlr		
DDR_SDRAM		
dsocm_bram		
isocm_bram		
apu_mand_0		
jtagppc_0		
plbv46_dcr_bridge_0		

And connect the **SYS_Rst** signal of **plb_v46_0** to **sys_bus_reset**:



Name	Net	Direction
External Ports		
ppc405_0		
dcr_v29_0		
fcb_v10_0		
ppc405_0_docm		
ppc405_0_iocm		
plb		
plb_v46_0		
Bus_Error_Det	No Connection	O
SYS_Rst	No Connection	I
PLB_Clk		I
ppc405_0_dplb1		
ppc405_0_iplb1		
ppc405_0_docm_cntlr		
ppc405_0_iocm_cntlr		
DDR_SDRAM		
dsocm_bram		
isocm_bram		
apu_mand_0		
jtagppc_0		

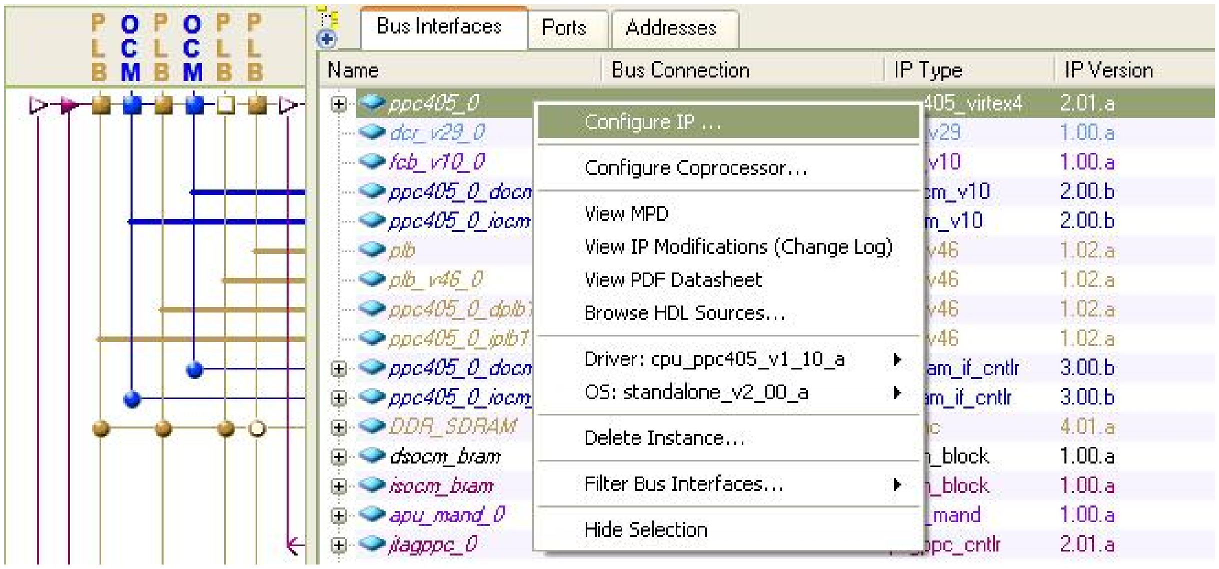
The **Ports** view of your project should now appear similar to the following:



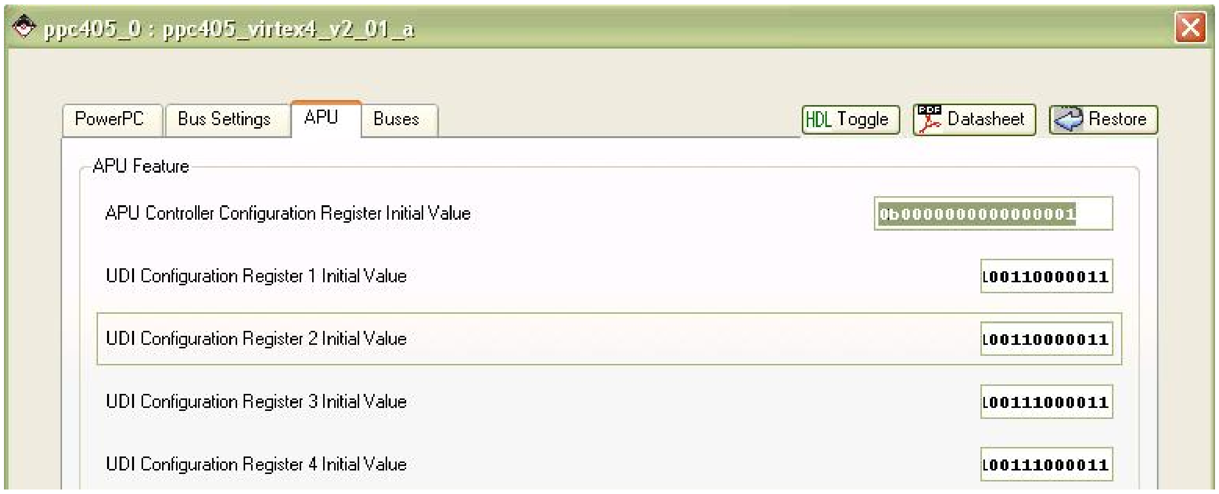
Name	Net	Direction
External Ports		
ppc405_0		
fcb_v10_0		
SYS_RST	sys_bus_reset	I
FCB_CLK	sys_clk_s	I
ppc405_0_docm		
ppc405_0_iocm		
plb		
plb_v46_0		
Bus_Error_Det	No Connection	O
SYS_Rst	sys_bus_reset	I
PLB_Clk	sys_clk_s	I
ppc405_0_dplb1		
ppc405_0_iplb1		
ppc405_0_docm_cntlr		
ppc405_0_iocm_cntlr		
DDR_SDRAM		
dsocm_bram		
isocm_bram		
apu_mand_0		
co_clk	pcore_co_clk	I
apu_clk	sys_clk_s	I
jtagppc_0		
proc_sys_reset_0		
LEDs_4Bit		
xps_tft_0		
TFT_VGA_B	xps_tft_0_TFT_VGA_B	O
TFT_VGA_G	xps_tft_0_TFT_VGA_G	O
TFT_VGA_R	xps_tft_0_TFT_VGA_R	O
TFT_IIC_SDA_T	No Connection	O
TFT_IIC_SDA_O	No Connection	O
TFT_IIC_SDA_I	No Connection	I
TFT_IIC_SCL_T	No Connection	O
TFT_IIC_SCL_O	No Connection	O
TFT_IIC_SCL_I	No Connection	I
TFT_VGA_CLK	xps_tft_0_TFT_VGA_CLK	O
TFT_DPS	No Connection	O
TFT_DE	No Connection	O
TFT_VSYNC	xps_tft_0_TFT_VSYNC	O
TFT_HSYNC	xps_tft_0_TFT_HSYNC	O
SYS_TFT_Clk	No Connection	I
MD_error	No Connection	O
xps_timer_1		
RS232_Uart		
clock_generator_0		

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.



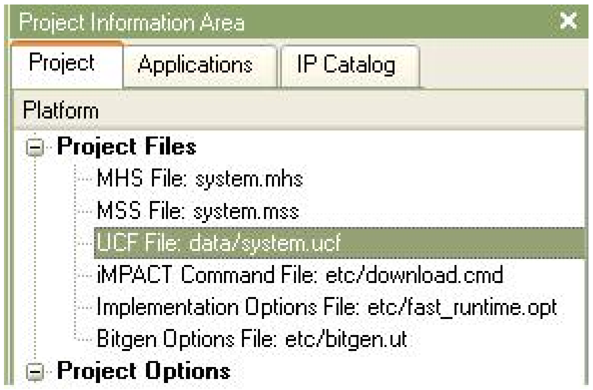
Switch to the **APU** tab and change the **APU Controller Configuration Register Initial Value** to **0b0000000000000001** as shown below:



APU Controller Configuration Register Initial Value	0b0000000000000001
UDI Configuration Register 1 Initial Value	100110000011
UDI Configuration Register 2 Initial Value	100110000011
UDI Configuration Register 3 Initial Value	100111000011
UDI Configuration Register 4 Initial Value	100111000011

Adding XPS_TFT Constraints

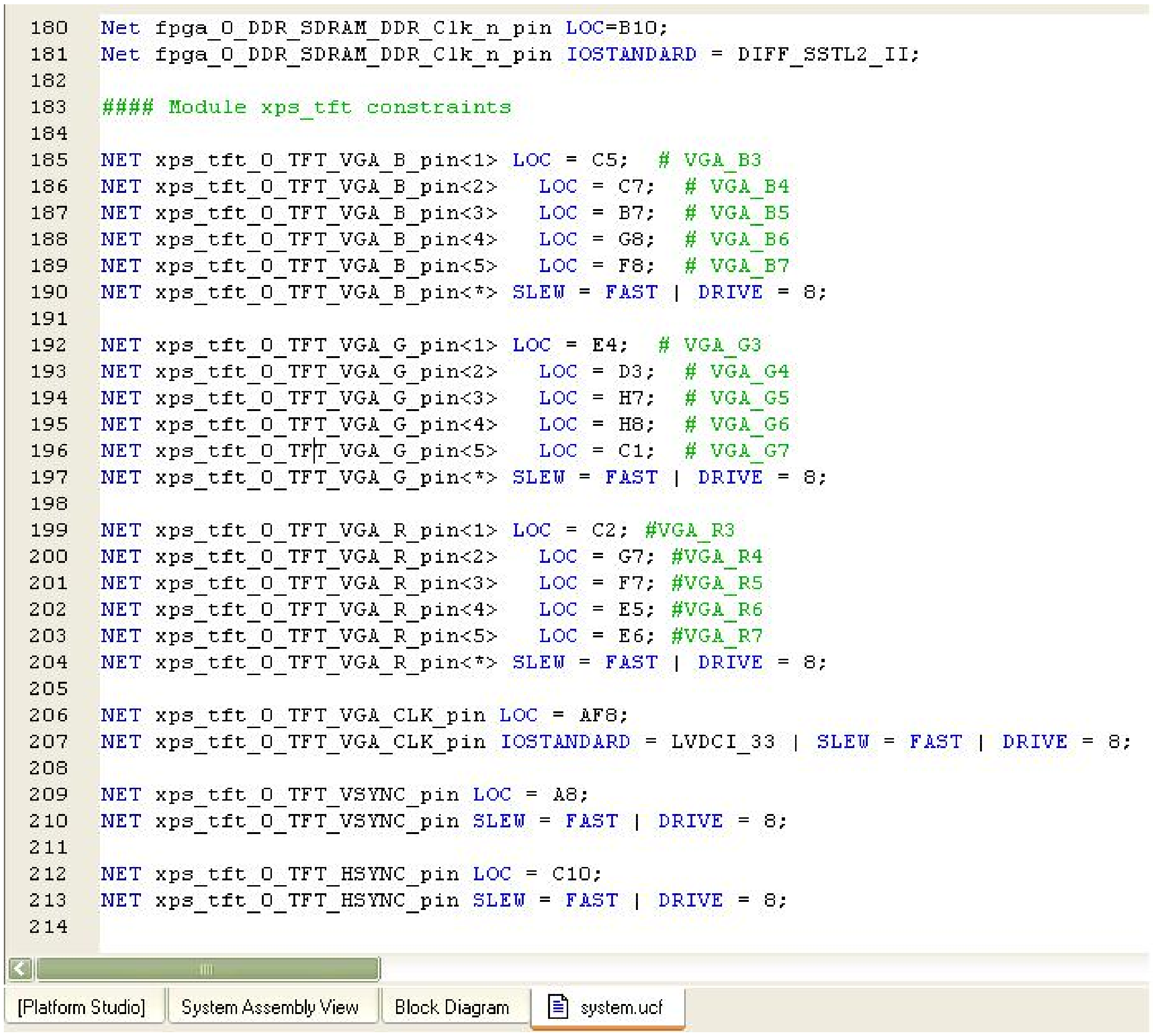
Since the **XPS_TFT** module is not added in the **BSB**, its constaints, such as port locations and types, need to be added manually to the **UCF** file associated with the project (**system.ucf**). To edit this file, in the **Project** tab, find the **UCF** file listed under **Project Files** as shown below. Double-click on the **UCF File: data/system.ucf** entry to open the file.


Project Information Area
Project
Applications
IP Catalog
Platform
Project Files
MHS File: system.mhs
MSS File: system.mss
UCF File: data/system.ucf
iMPACT Command File: etc/download.cmd
Implementation Options File: etc/fast_runtime.opt
Bitgen Options File: etc/bitgen.ut
Project Options

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

```
#### Module xps_tft constraints

NET xps_tft_0_TFT_VGA_B_pin<1> LOC = C5;  # VGA_B3
NET xps_tft_0_TFT_VGA_B_pin<2>      LOC = C7;  # VGA_B4
NET xps_tft_0_TFT_VGA_B_pin<3>      LOC = B7;  # VGA_B5
NET xps_tft_0_TFT_VGA_B_pin<4>      LOC = G8;  # VGA_B6
NET xps_tft_0_TFT_VGA_B_pin<5>      LOC = F8;  # VGA_B7
NET xps_tft_0_TFT_VGA_B_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_G_pin<1> LOC = E4;  # VGA_G3
NET xps_tft_0_TFT_VGA_G_pin<2>      LOC = D3;  # VGA_G4
NET xps_tft_0_TFT_VGA_G_pin<3>      LOC = H7;  # VGA_G5
NET xps_tft_0_TFT_VGA_G_pin<4>      LOC = H8;  # VGA_G6
NET xps_tft_0_TFT_VGA_G_pin<5>      LOC = C1;  # VGA_G7
NET xps_tft_0_TFT_VGA_G_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_R_pin<1> LOC = C2; #VGA_R3
NET xps_tft_0_TFT_VGA_R_pin<2>      LOC = G7; #VGA_R4
NET xps_tft_0_TFT_VGA_R_pin<3>      LOC = F7; #VGA_R5
NET xps_tft_0_TFT_VGA_R_pin<4>      LOC = E5; #VGA_R6
NET xps_tft_0_TFT_VGA_R_pin<5>      LOC = E6; #VGA_R7
NET xps_tft_0_TFT_VGA_R_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_CLK_pin LOC = AF8;
NET xps_tft_0_TFT_VGA_CLK_pin IOSTANDARD = LVDCI_33 | SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VSYNC_pin LOC = A8;
NET xps_tft_0_TFT_VSYNC_pin SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_HSYNC_pin LOC = C10;
NET xps_tft_0_TFT_HSYNC_pin SLEW = FAST | DRIVE = 8;
```

The modified **UCF** file should look as shown below:



```
Net fpga_0_DDR_SDRAM_DDR_Clk_n_pin LOC=B10;
Net fpga_0_DDR_SDRAM_DDR_Clk_n_pin IOSTANDARD = DIFF_SSTL2_II;

#### Module xps_tft constraints

NET xps_tft_0_TFT_VGA_B_pin<1> LOC = C5;  # VGA_B3
NET xps_tft_0_TFT_VGA_B_pin<2>   LOC = C7;  # VGA_B4
NET xps_tft_0_TFT_VGA_B_pin<3>   LOC = B7;  # VGA_B5
NET xps_tft_0_TFT_VGA_B_pin<4>   LOC = G8;  # VGA_B6
NET xps_tft_0_TFT_VGA_B_pin<5>   LOC = F8;  # VGA_B7
NET xps_tft_0_TFT_VGA_B_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_G_pin<1> LOC = E4;  # VGA_G3
NET xps_tft_0_TFT_VGA_G_pin<2>   LOC = D3;  # VGA_G4
NET xps_tft_0_TFT_VGA_G_pin<3>   LOC = H7;  # VGA_G5
NET xps_tft_0_TFT_VGA_G_pin<4>   LOC = H8;  # VGA_G6
NET xps_tft_0_TFT_VGA_G_pin<5>   LOC = C1;  # VGA_G7
NET xps_tft_0_TFT_VGA_G_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_R_pin<1> LOC = C2; #VGA_R3
NET xps_tft_0_TFT_VGA_R_pin<2>   LOC = G7; #VGA_R4
NET xps_tft_0_TFT_VGA_R_pin<3>   LOC = F7; #VGA_R5
NET xps_tft_0_TFT_VGA_R_pin<4>   LOC = E5; #VGA_R6
NET xps_tft_0_TFT_VGA_R_pin<5>   LOC = E6; #VGA_R7
NET xps_tft_0_TFT_VGA_R_pin<*> SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VGA_CLK_pin LOC = AF8;
NET xps_tft_0_TFT_VGA_CLK_pin IOSTANDARD = LVDCI_33 | SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_VSYNC_pin LOC = A8;
NET xps_tft_0_TFT_VSYNC_pin SLEW = FAST | DRIVE = 8;

NET xps_tft_0_TFT_HSYNC_pin LOC = C10;
NET xps_tft_0_TFT_HSYNC_pin SLEW = FAST | DRIVE = 8;
```

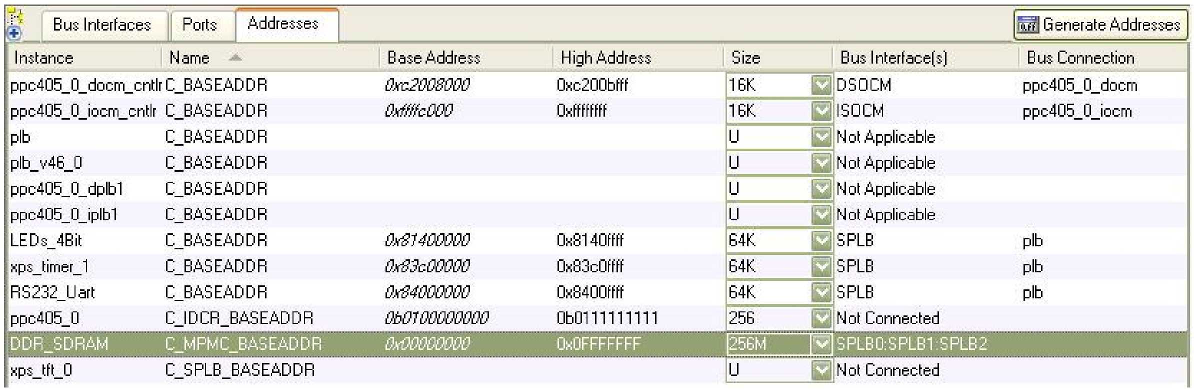
Save the **UCF** file using the menu **File** -> **Save** command, then close the editing window.

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:



Instance	Name	Base Address	High Address	Size	Bus Interface(s)	Bus Connection
ppc405_0_docm_cntlr	C_BASEADDR	0xc2008000	0xc200bfff	16K	DSOCM	ppc405_0_docm
ppc405_0_iocm_cntlr	C_BASEADDR	0xffffc000	0xffffffff	16K	ISOCM	ppc405_0_iocm
plb	C_BASEADDR			U	Not Applicable	
plb_v46_0	C_BASEADDR			U	Not Applicable	
ppc405_0_dplb1	C_BASEADDR			U	Not Applicable	
ppc405_0_iplb1	C_BASEADDR			U	Not Applicable	
LEDs_4Bit	C_BASEADDR	0x81400000	0x8140ffff	64K	SPLB	plb
xps_timer_1	C_BASEADDR	0x83c00000	0x83c0ffff	64K	SPLB	plb
RS232_Uart	C_BASEADDR	0x84000000	0x8400ffff	64K	SPLB	plb
ppc405_0	C_IDCR_BASEADDR	0b0100000000	0b0111111111	256	Not Connected	
DDR_SDRAM	C_MPMC_BASEADDR	0x00000000	0x0FFFFFFF	256M	SPLB0:SPLB1:SPLB2	
xps_tft_0	C_SPLB_BASEADDR			U	Not Connected	

An error message might show up when generating the addresses:

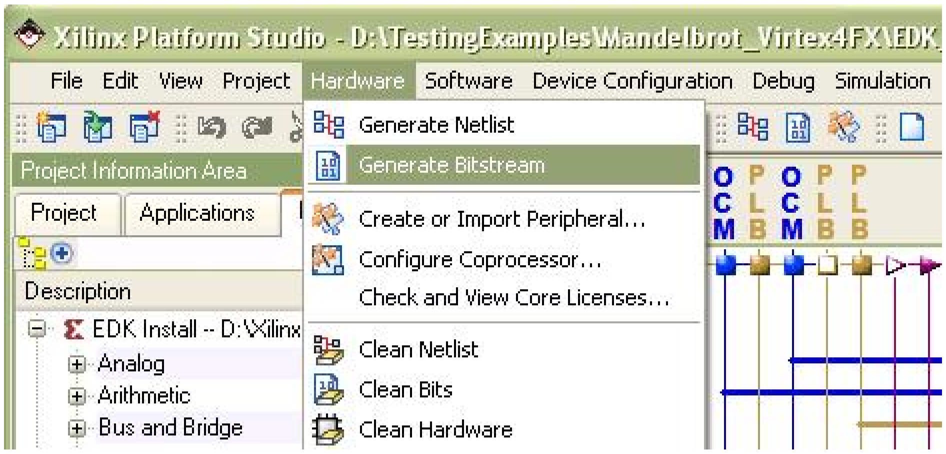
ERROR:MDT - C_IDCR_BASEADDRof ppc405_0 has no high address in MHS

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.

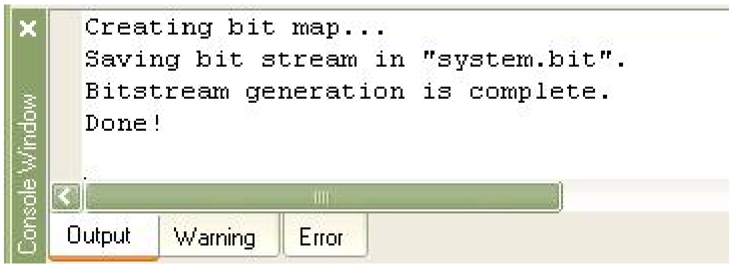


During the building process, an error message might pop up due to a known issue with the ***EDK*** *software:*

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.*

After the process is done, a file called **system.bit** is created.



```
Creating bit map...
Saving bit stream in "system.bit".
Bitstream generation is complete.
Done!
```

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

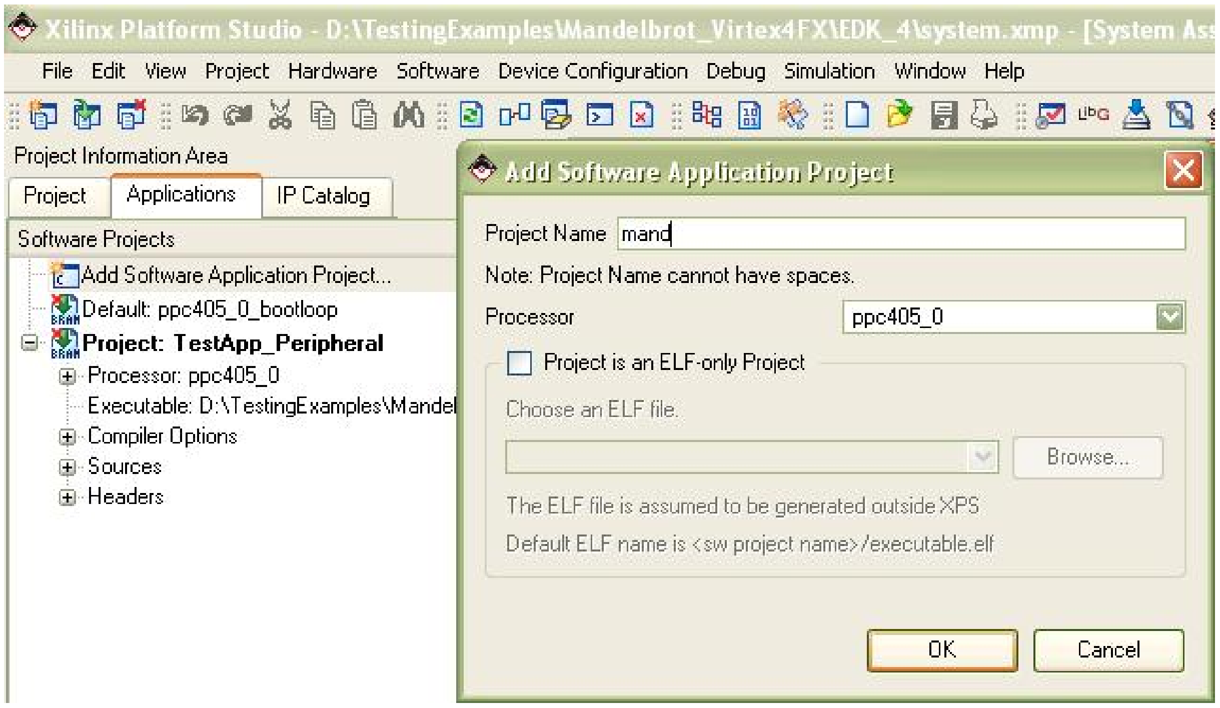
1.8 Adding the Software Application Files

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 **mand** as the **Project Name** as shown below:



Xilinx Platform Studio - D:\TestingExamples\Mandelbrot_Virtex4FX\EDK_4\system.xmp - [System Ass
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project Information Area
Project
Applications
IP Catalog
Software Projects
Add Software Application Project...
Default: ppc405_0_bootloop
Project: TestApp_Peripheral
Processor: ppc405_0
Executable: D:\TestingExamples\Mandel
Compiler Options
Sources
Headers

Add Software Application Project

Project Name: mand

Note: Project Name cannot have spaces.

Processor: ppc405_0

☐ Project is an ELF-only Project

Choose an ELF file.

Browse...

The ELF file is assumed to be generated outside XPS

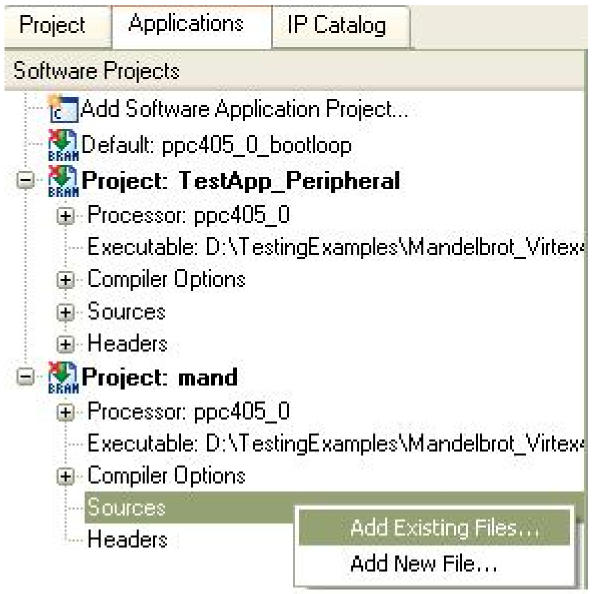
Default ELF name is <sw project name>/executable.elf

OK Cancel

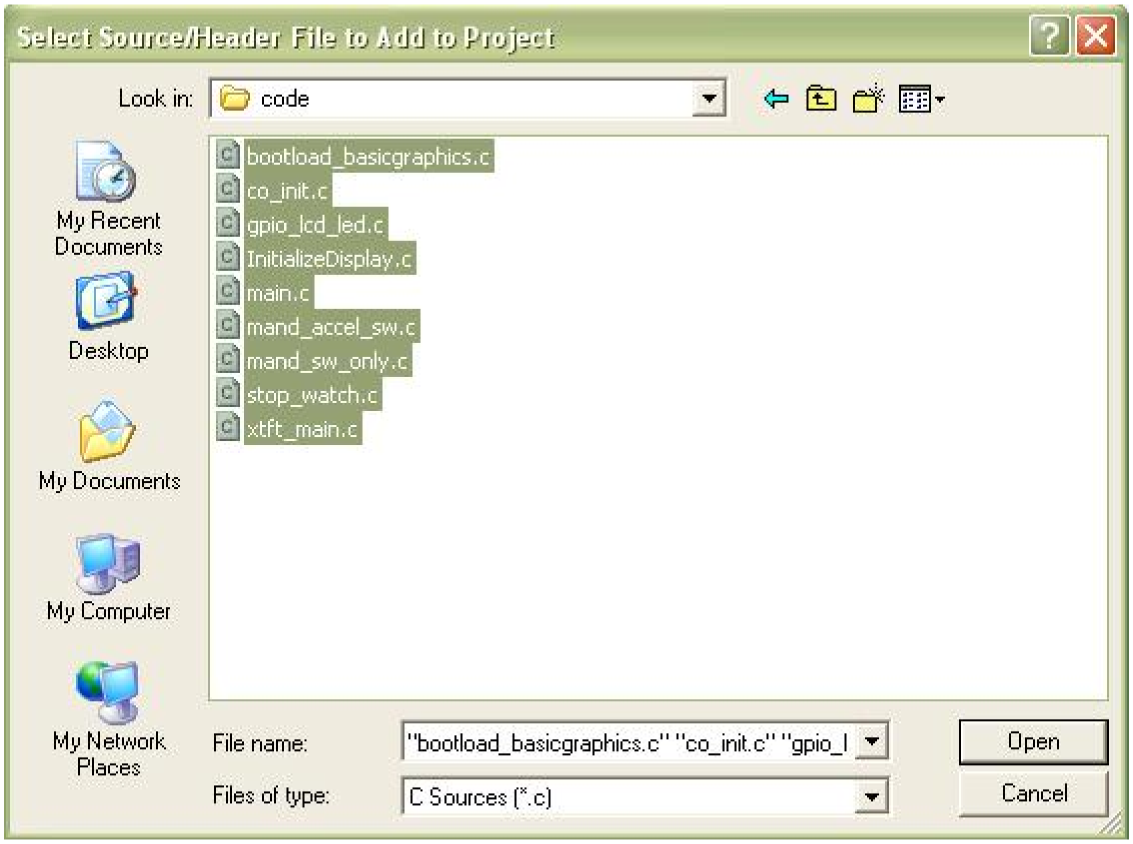
Click **OK** to continue.

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 right-clicking as shown below:



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

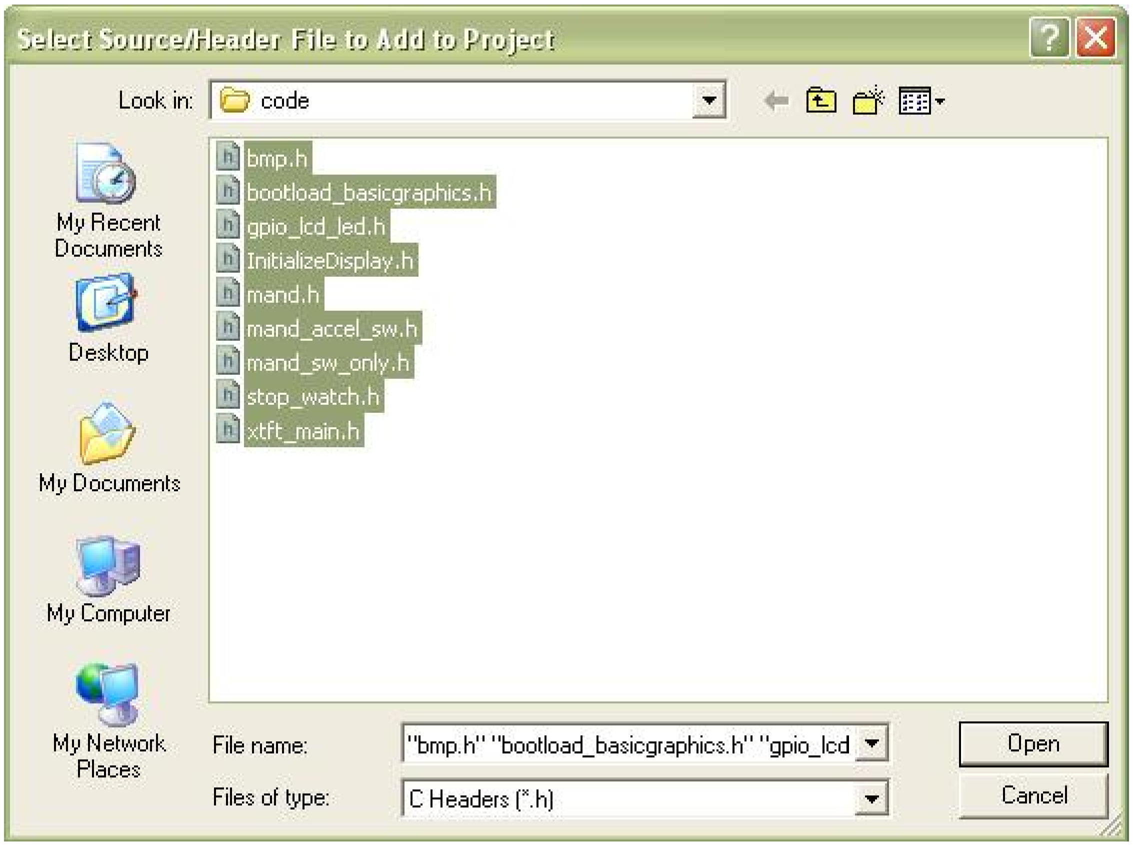


bootload_basicgraphics.c
co_init.c
gpio_lcd_led.c
InitializeDisplay.c
main.c
mand_accel_sw.c
mand_sw_only.c
stop_watch.c
xtft_main.c

File name: "bootload_basicgraphics.c" "co_init.c" "gpio_l

Files of type: C Sources (*.c)

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



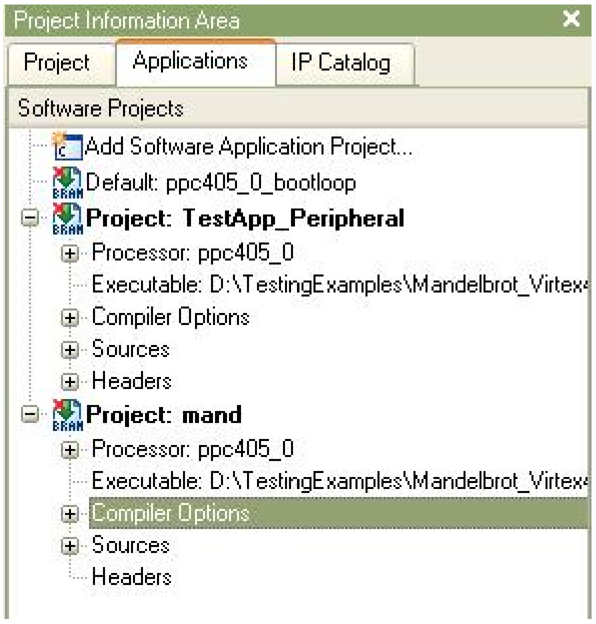
bmp.h
bootload_basicgraphics.h
gpio_lcd_led.h
InitializeDisplay.h
mand.h
mand_accel_sw.h
mand_sw_only.h
stop_watch.h
xtft_main.h

File name: "bmp.h" "bootload_basicgraphics.h" "gpio_lcd

Files of type: C Headers (*.h)

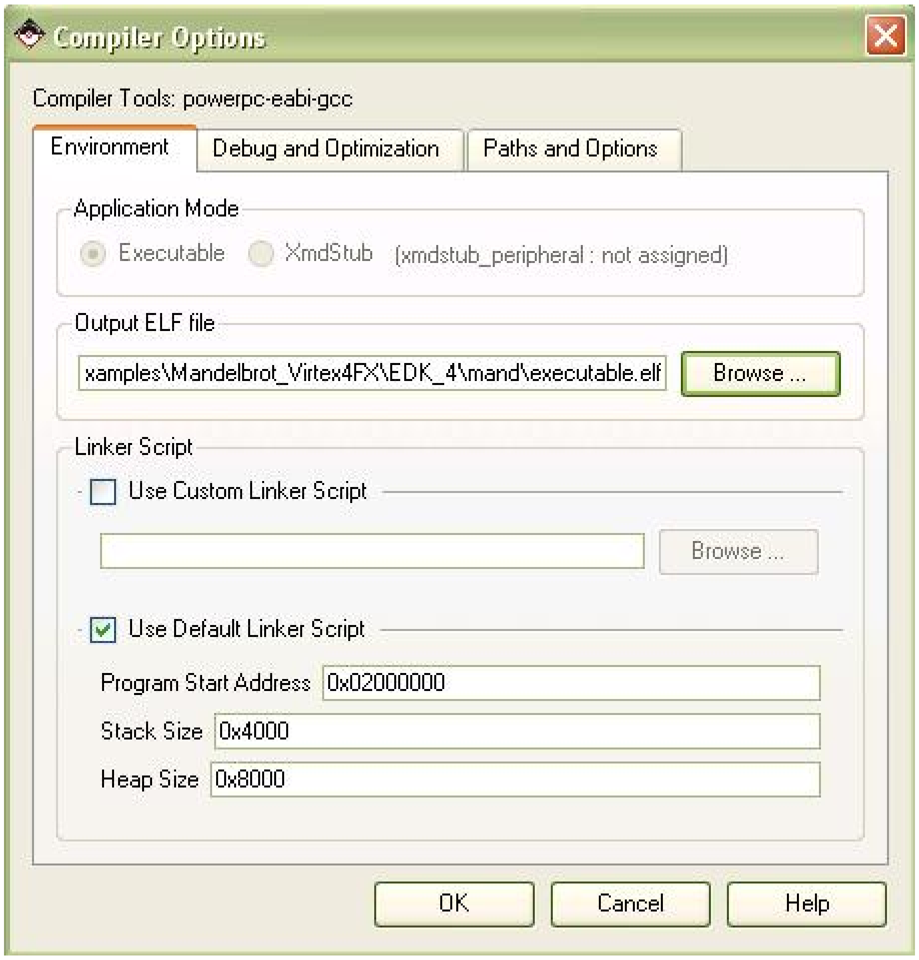
Setting Compiler Options

Now you will need to set a few compiler options for the project. To set the compiler options, double-click on the **Compiler Options** under the **Project: mand**:



Add Software Application Project...
Default: ppc405_0_bootloop
Project: TestApp_Peripheral
Processor: ppc405_0
Executable: D:\TestingExamples\Mandelbrot_Virtex
Compiler Options
Sources
Headers
Project: mand
Processor: ppc405_0
Executable: D:\TestingExamples\Mandelbrot_Virtex
Compiler Options
Sources
Headers

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:



Compiler Tools: powerpc-eabi-gcc

Application Mode
Executable [✓] XmdStub (xmdstub_peripheral : not assigned)

Output ELF file: xamples\Mandelbrot_Virtex4FX\EDK_4\mand\executable.elf

Linker Script
Use Custom Linker Script [ ]
Use Default Linker Script [✓]
Program Start Address 0x02000000
Stack Size 0x4000
Heap Size 0x8000

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

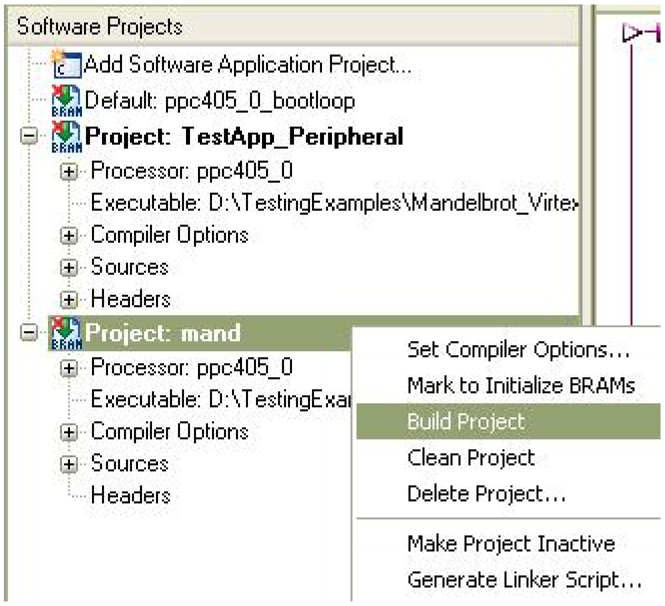
1.9 Building and Downloading the Application

Mandelbrot Extended Tutorial for Virtex-4 FX, Step 10

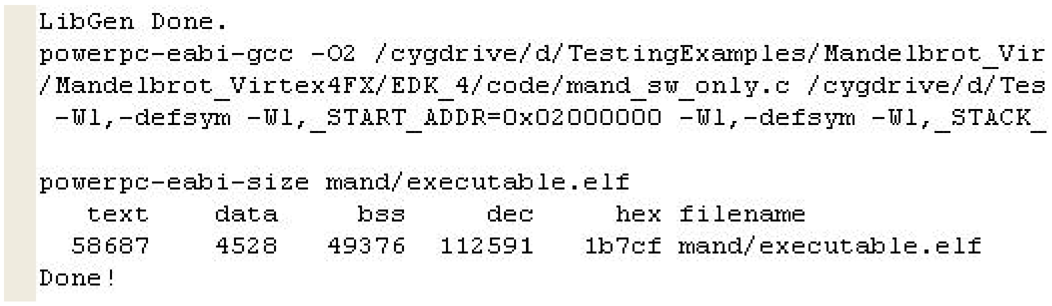
The Mandelbrot application is now ready to build, download and execute on the target **ML403** board.

First, compile the software application to create a **PowerPC** executable. Do this by selecting **Build**

Project from the **Project: mand** entry as shown below:

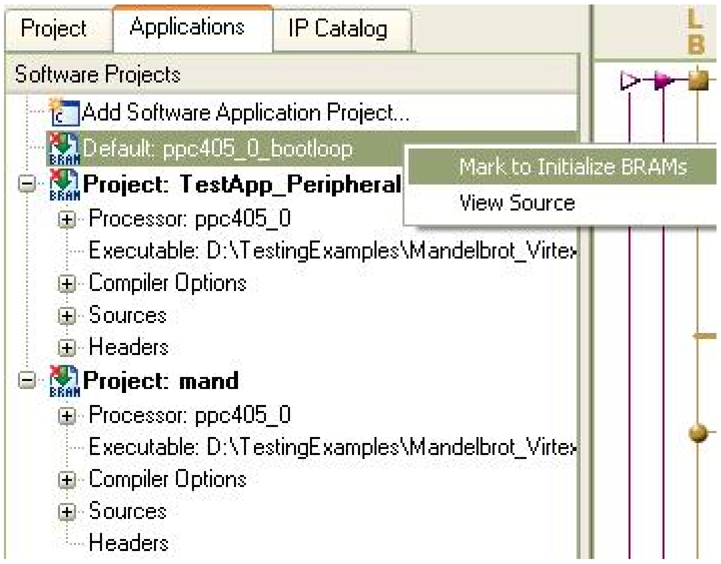


The size of the generated executable is shown below. It will be included in the FPGA bitstream.



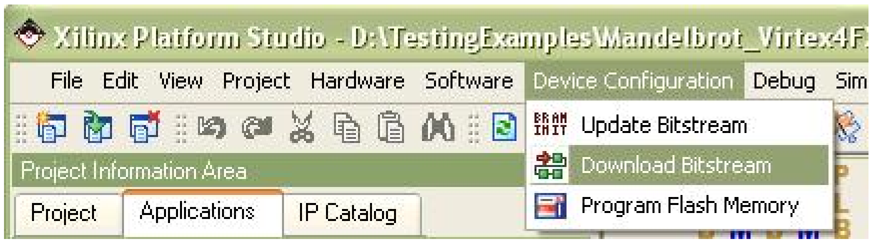
```
LibGen Done.
powerpc-eabi-gcc -O2 /cygdrive/d/TestingExamples/Mandelbrot_Vir
/Mandelbrot_Virtex4FX/EDK_4/code/mand_sw_only.c /cygdrive/d/Tes
 -Wl,-defsym -Wl,_START_ADDR=0x02000000 -Wl,-defsym -Wl,_STACK_

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.

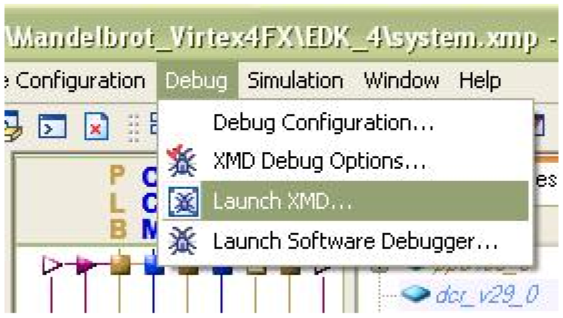


Now, it is time to download the bitstream to the **ML403** board. Make sure the **JTAG** cable is properly connected and that the **ML403** board is powered on. Also make sure the **VGA** display is connected and powered on.

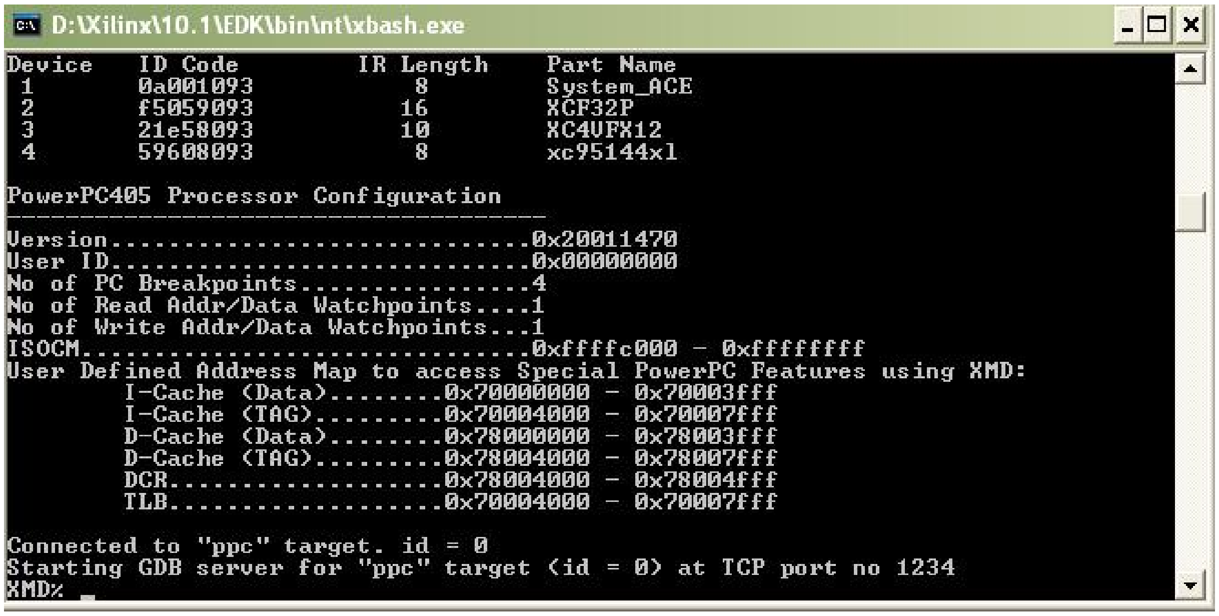
Select **Download Bitstream** as shown below:


Xilinx Platform Studio - D:\TestingExamples\Mandelbrot_Virtex4F
File Edit View Project Hardware Software Device Configuration Debug Sim
Update Bitstream
Download Bitstream
Program Flash Memory
Project Information Area
Project
Applications
IP Catalog

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


Mandelbrot_Virtex4FX\EDK_4\system.xmp
Configuration Debug Simulation Window Help
Debug Configuration...
XMD Debug Options...
Launch XMD...
Launch Software Debugger...
dcr_v29_0

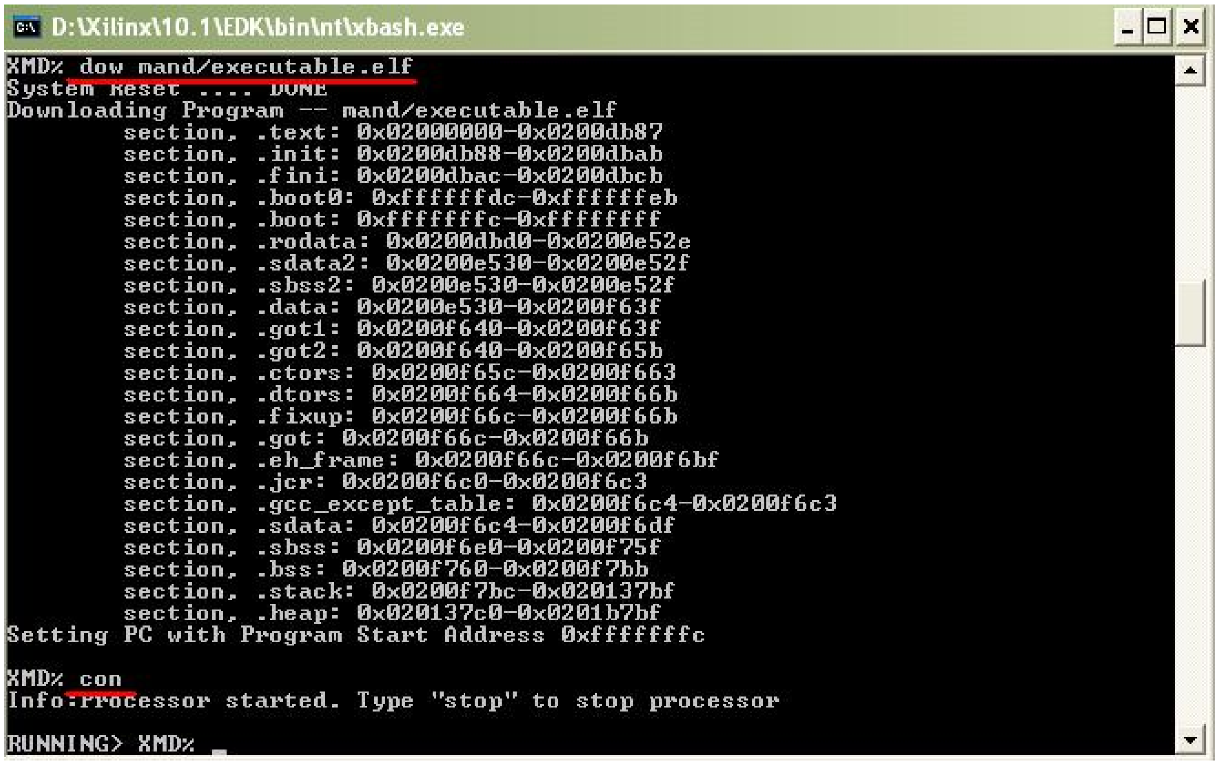
If this is the first time you have launched **XMD** for this **EDK** project, the **XMD Debug Options** dialogue will pop up. Just click **OK** to accept the default settings, then the **XMD** terminal will appear.



D:\Xilinx\10.1\EDK\bin\nt\xbash.exe

```
Device   ID Code        IR Length    Part Name
 1       0a001093           8        System_ACE
 2       f5059093          16        XCF32P
 3       21e58093          10        XC4VFX12
 4       59608093           8        xc95144xl

PowerPC405 Processor Configuration
-------------------------------------
Version.............................0x20011470
User ID.............................0x00000000
No of PC Breakpoints.................4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
ISOCM...............................0xffffc000 - 0xffffffff
User Defined Address Map to access Special PowerPC Features using XMD:
        I-Cache (Data)........0x70000000 - 0x70003fff
        I-Cache (TAG).........0x70004000 - 0x70007fff
        D-Cache (Data)........0x78000000 - 0x78003fff
        D-Cache (TAG).........0x78004000 - 0x78007fff
        DCR...................0x78004000 - 0x78004fff
        TLB...................0x70004000 - 0x70007fff

Connected to "ppc" target. id = 0
Starting GDB server for "ppc" target (id = 0) at TCP port no 1234
XMD% _
```

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



D:\Xilinx\10.1\EDK\bin\nt\xbash.exe

```
XMD% dow mand/executable.elf
System Reset .... DONE
Downloading Program -- mand/executable.elf
        section, .text: 0x02000000-0x0200db87
        section, .init: 0x0200db88-0x0200dbab
        section, .fini: 0x0200dbac-0x0200dbcb
        section, .boot0: 0xffffffdc-0xffffffeb
        section, .boot: 0xfffffffc-0xffffffff
        section, .rodata: 0x0200dbd0-0x0200e52e
        section, .sdata2: 0x0200e530-0x0200e52f
        section, .sbss2: 0x0200e530-0x0200e52f
        section, .data: 0x0200e530-0x0200f63f
        section, .got1: 0x0200f640-0x0200f63f
        section, .got2: 0x0200f640-0x0200f65b
        section, .ctors: 0x0200f65c-0x0200f663
        section, .dtors: 0x0200f664-0x0200f66b
        section, .fixup: 0x0200f66c-0x0200f66b
        section, .got: 0x0200f66c-0x0200f66b
        section, .eh_frame: 0x0200f66c-0x0200f6bf
        section, .jcr: 0x0200f6c0-0x0200f6c3
        section, .gcc_except_table: 0x0200f6c4-0x0200f6c3
        section, .sdata: 0x0200f6c4-0x0200f6df
        section, .sbss: 0x0200f6e0-0x0200f75f
        section, .bss: 0x0200f760-0x0200f7bb
        section, .stack: 0x0200f7bc-0x020137bf
        section, .heap: 0x020137c0-0x0201b7bf
Setting PC with Program Start Address 0xfffffffc

XMD% con
Info:Processor started. Type "stop" to stop processor

RUNNING> XMD% _
```

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

Congratulations! You have completed this advanced tutorial.