I

Table of Contents

	Foreword	1
Part I	Quick Start Tutorials	3
1	Tutorial 1: Complex FIR Filter on Virtex-5 Platform (EDK 10.1)	3
	Loading the Complex FIR Filter Application	4
	Understanding the Complex FIR Filter Application	5
	Compiling the Application for Simulation	7
	Building the Application for the Target Platform	8
	Exporting Files from CoDeveloper	10
	Creating a Platform Using Xilinx Tools	11
	Configuring the New Platform	16
	Importing the Generated Hardware	29
	Generating the FPGA Bitmap	36
	Importing the Application Software	36
	Running the Application	43
2	Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)	47
	Loading the Sample Application	48
	Understanding the Image Filter DMA Application	49
	Compiling the Application for Simulation	51
	Building the Application for Hardware	53
	Exporting the Hardware and Software Files	54
	Creating the ML403 Test Platform	56
	Adding the ImageFilterDMA Hardware	69
	Adding the Software Application Files	77
	Building and Downloading the Application	81
3	Tutorial 3: Fractal Image Generation using APU on the Virtex-4 Platform (EDK 10.1)	87
	Loading the Sample Application	89
	Understanding the Mandelbrot Application	90
	Compiling the Application for Simulation	91
	Building the Application for Hardware	93
	Exporting the Hardware and Software Files	95
	Copying the TFT Display Core Files	97
	Creating the ML403 Test Platform	98
	Adding the Mandelbrot Hardware	111
	Adding the Software Application Files	135
	Building and Downloading the Application	139
	la dese	•

Index

0

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



1 Quick Start Tutorials



Overview

3

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

<u>Tutorial 1: Complex FIR Filter on Virtex-5 Platform (EDK 10.1)</u> <u>Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)</u> 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

Loading the Complex FIR Application Understanding the Complex FIR Application Compiling the Application for Simulation Building the Application for the Target Platform Creating the Platform Using the Xilinx Tools Configuring the New Platform Exporting Files from CoDeveloper Importing the Generated Hardware Generating the FPGA Bitmap Importing the Application Software Running the Application

See Also

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

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 <u>http://impulsec.com/ReadyToRun/</u>. Opening the project will result in the display of a window similar to the following:

5

🗝 Impulse CoDeveloper Applicati	on Manag	er Universal Edition - [FIR_Accelerator] - [Filter_hw.c]				
Eile Edit View Project Tools Wind	dow <u>H</u> elp					
🛺 😅 🕼 🖹 🚇 🎦 🔜 👗 I	1 1 1) / / / / 44 😘 🍅 🍃 : 茜 図 🕨 茜 🔍 🔍 😾 🥜 🍃				
Project Explorer 🛛 📮 🗙	Filter :	sw.c Filter hw.c				
A						
	7	#include <stdio.h></stdio.h>				
크 @의 Application FIR_Accelerator	8	#include "co.h"				
🖻 🔄 Source Files	ource Files 9 #include "cosim log.n"					
sw C ComplexFilter.c	10 #include "Filter.h"					
hw C Filter hw.c	11					
sw C Filter swic	12	extern void call_accelerator (co_stream output_stream, co_stream input_stream);				
	13					
	14	<pre>void complex_fir (co_stream filter_in, co_stream filter_out) {</pre>				
SW h ComplexFilter.h	15	int32 coef_mem[IF_FILT_LEN];				
hiter.h	16	int32 filt_hist[IF_FILT_LEN];				
- Project Files	17	int32 inSample;				
🖻 🔄 Document Files	18	int32 outFilter;				
Beadme htm	19	int i;				
Cher Files	20	int write_idx;				
	21	int read_idx;				
	22	<pre>int t_read_idx, t_read_idx2;</pre>				
	23	int32 t_coef, t_hist;				
	24	<pre>intl6 coef_real, coef_imag;</pre>				
	25	intl6 hist_real, hist_imag;				
	26	int32 pl_real, p2_real;				
	27	int32 pl imag, p2 imag;				
	28	int32 res real, res imag;				
	29	int32 accum real, accum imag;				
	30	IF SIM (int samplesread = $0;$)				
	31	IF SIM (int sampleswritten = 0;)				
	32	(and = main - three califies - here do and the second calification - the sec				
	33	IF SIM (cosim logwindow log;)				
	34	IF SIM (log = cosim logwindow create ("complex fir");)				
	35	and the local second contract the second sec				
	36	co array config(coef mem,co kind,"async");				
	37	co array config(filt hist, co kind, "async");				
	38					

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:

 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:

Build
====== Building target 'build_exe' in file_Makefile =======
"C:/Impulse/CoDeveloper3_30/MinGW/bin/gcc"-g"-IC:\Impulse\CoDeveloper3_30\Include" "IC:/Impulse/CoDeveloper3_30/StageMaster/include" -DW/IN32
"-IC:/Impulse/CoDeveloper3_30/MinGW/include" -o ComplexFilter.o -c ComplexFilter.c
"C://mpulse/CoDeveloper3_30/MinGW/bin/gcc" - g "+C:\/mpulse\CoDeveloper3_30\/Include" ''+C://mpulse/CoDeveloper3_30/StageMaster/include" -DWIN32
"1C:/Impulse/CoDeveloper3_30/MinGW/include" -o Filter_hw.o -c Filter_hw.c
"C:/Impulse/CoDeveloper3_30/MinGW/bin/gcc" -g "-IC:\Impulse\CoDeveloper3_30\Include" '-IC:/Impulse/CoDeveloper3_30/StageMaster/include" -DWIN32
"+IC:/Impulse/CoDeveloper3_30/MinGW/include" -o Filter_sw.o -c Filter_sw.c
"C:/Impulse/CoDeveloper3_30/MinGW/bin/gcc" -g ComplexFilter.o Filter_hw.o Filter_sw.o "C:\Impulse\CoDeveloper3_30\Libraries/ImpulseC.lib" -o FIR_Accelerator.exe

======= Build of target "build_exe" complete =======

 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:

7

```
#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:

Options	\mathbf{X}
Build Simulate Generate System Registration Platform Support Package:	Directories Hardware build directory:
CoBuilder Generation Options Generate dual clocks Active-low reset Use std_logic types for VHDL interfaces V Do not include co_ports in bus interface	hw Software build directory: sw Hardware export directory: EDK Software export directory: EDK
Include floating point library Include floating point library I Use higher latency, faster clock operators Allow double-precision types and operators	Cancel Apply Help

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:

Build Max. Unit Delay: 0 Block #11: Stages: 1 Max. Unit Delay: 0 | Operators: | 10 Adder(s)/Subtractor(s) (32 bit) 4 Multiplier(s) (16 bit) 1 Comparator(s) (3 bit) 5 Comparator(s) (32 bit) | Total Stages: 20 | Max. Unit Delay: 33 | Estimated DSPs: 4 Writing output ... done "C:/Impulse/CoDeveloper3_30/bin/impulse_arch" "-aC:/Impulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml" -dc -no_port_bus_connect -swdirsw -files "FIR_Accelerator_comp.vhd FIR_Accelerator_top.vhd "FIR_Accelerator.xic hw/FIR_Accelerator_top.vhd Impulse C HDL Design Generator Copyright 2002-2007, Impulse Accelerated Technologies, Inc. All rights reserved.

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

"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 Exporte Copyright 2002-2007, Impulse Accelerated Technologies, Inc. All rights reserved. Loading C:/Inpulse/CoDeveloper3_30/Architectures/xilinx_v5_apu.xml ... Loading C:/Inpulse/CoDeveloper3_30/Architectures/Xilinx/V5/PPC/APU/cpu.xml ... Loading FIR_Accelerator.xic . ===== Build of target 'export_hardware' complete =======

Export the Software Files

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:

358			
-) Blank XPS project		
8 0) Open a recent project		
Browse fo	or More Projects		

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):

🗢 Create New XPS Project Using BSB Wizard	Platform Studi	o Projeci		? 🔀
New project Project file	Save in: My Recent Documents Desktop My Documents My Computer	Code		
	My Network Places	File name: Save as type:	system.xmp Platform Studio Project (".xmp)	Save Cancel

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

Base System Builder - Welcome				X
Embedded Develop Platform Studio	oment-Ki	nat.	-	
Welcome to the Base Syst	em Build	ər!		
This tool will lead you through the steps nece	essary to create	an embedded	system.	
Please begin by selecting one of the follow	ing options:			
I would like to create a new design				
I would like to load an existing .bsb se	ttings file (saved	from a previo	us sessior	1)
				Browse
More Info	< Back	Ne>	(t >	Cancel

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

	i priligati - Palaci posti d	
elect a target de	velopment board:	
Select board		
 I would like 	to create a system for the following development board	
Board vendor:	Xilinx	
Board name:	Virtex 5 ML507 Evaluation Platform	~
Board revision:	A	~
Note: Visit the v	vendor website for additional board support materials.	
Vendor's Websi	ite Contact Info	
Download Third	Party Board Definition Files	
🔾 l would like	to create a system for a custom board	
and 2 RS232 s	serial ports.	

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

Architecture:	Device:	Package:	Speed grade:	
virtex5	wc5vfx70t g	Mf1136	1	
Processors O MicroBlaze O PowerPC	you would like to use	in mis design.		
Processor descripti The PowerPC(R) embedded-enviro IP-Immersion tech cores for peripher	ion 440 processor core is nment architecture. It nnology and supported rals and utilities.	a 32-bit implementation (is integrated into the Virt by CoreConnect bus infi	of a RISC PowerPC pro ex-5 FXT devices using rastructure and extensiv	cessor i the re IP
Processor descripti The PowerPC(R) embedded-enviro IP-Immersion tech cores for peripher	on 440 processor core is nment architecture. It nology and supported als and utilities.	a 32-bit implementation (is integrated into the Virt by CoreConnect bus inf	of a RISC PowerPC pro ex-5 FXT devices using rastructure and extensiv	cessor I the re IP

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)

equency:	frequency:		Bus clock fre	quency:	
00.00 MHz	125.00	MHz	125.00	MHz	
leset polarity:	tive LOW				
rocessor configuration					
Debug I/F					
● FPGA JTAG					
O CPU debug user	pins only				
CPU debug and t	trace pins				
🔘 No debug					
Power	PC				
Lache setup					
For optimal performan	ica anabla burst				
and/or cacheline on r	memory				
Enable floating poin	ntunit(FPL) (วิ				
and a server in the server of	(((((((((((((((((((

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.

Tevices		
🔽 RS232_U	art_1	Data Sheet
Peripheral: 🔀	PS UARTLITE	
Baudrate (bits		
per seconds):	9600	
Data bits:	8	
Parity:	NONE	
Use interr	ıpt	
RS232_U	art_2	
-		Data Sheet
LEDs_88		Data Sheet

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

🗢 Base System Builder - Configure	10 Interfaces (2 of 4)
The following external memory and IO device Xilinx Virtex 5 ML507 Evaluation Platform Rev Please select the IO devices which you woul	rs were found on your board: vision A d like to use:
10 devices	
LEDs_Positions	Data Sheet
Push_Buttons_5Bit	Data Sheet
DIP_Switches_8Bit	Data Sheet
	Data Sheet
SRAM	Data Sheet
More Info	< Back Next > Cancel

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

🗢 Base System Builder - Configure 10 Interface	is (3 of 4) 🛛 🔀
The following external memory and 10 devices were found on Xilinx Virtex 5 ML507 Evaluation Platform Revision A Please select the 10 devices which you would like to use:	n your board:
10 devices	
FLASH	Data Sheet
PCIe_Bridge	Data Sheet Note
Hard_Ethernet_MAC	Data Sheet
More Info Sack	Next > Cancel

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

• Base System	Builder - Configure	e 10 Interfaces (4 of 4)	Σ
The following exter	nal memory and 10 devic	es were found on your board:	
Xilinx Virtex 5 ML50)7 Evaluation Platform R	evision A	
Please select the I) devices which you wo	uld like to use:	
-IU devices			
DDR2_	SDRAM		Data Sheet
Peripheral:	PPC440MC DDR2		
- 🗌 SysACE	_CompactFlash		Data Sheet
More Info		< Back Ne:	xt > Cancel

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:

🗢 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.	
In you do not wish to dad dily normo perpinendis, click the mexic battory.	Add Peripheral
Peripherals	
xps_bram_if_cntlr_1	Pamaua
Peripheral: XPS BRAM IF CNTLR	Data Sheet
Memory size: 8 KB	
🗢 Add Peripheral 🔹 🤶	
	Ĩ.
Select the peripheral you want to add:	
×PS TIMER [♥]	
	ak l
OK Cancel	
More Info Kext >	Cancel

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.

🗢 Base System Builder - Add Intern	al Peripherals (1 of 1)	×
Add other peripherals that do not interact with "Add Peripheral" button to select from the list If you do not wish to add any non-IO peripher	n off-chip components. Use the of available peripherals. als, click the "Next" button.	
D. C.L. J.		Add Peripheral
- Peripherals		
xps_bram_if_cntlr_1		Remove
Memory size: 8 KB		Data Sheet
xps timer 1		
Peripheral: XPS TIMER		Remove
Counter bit width: 32		Data Sheet
Timer mode		
 Two timers are present 		
 One timer is present 		
Use interrupt		
More Info	< Back Next >	Cancel

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

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

u nave enabled (he cache feature on t	the PowerPC processor.	
Cache setup			
Size of instruction	and data cache (car	n not be changed on PPC):	
Instruction C	ache (ICache) Size:	32 KB	
Data Cache	(DCache) Size:	32 KB	
Select the memor	y peripherals you wou	Ild like to cache:	
ICache:	DCache:	Cacheable Memories:	
		DDR2_SDRAM	
		xps_bram_if_cntlr_1	
Note: PowerPC44	0 caches are enabled	d by calling XCache_EnableICache and	
Note: PowerPC44	0 caches are enable	d by calling XCache_EnablelCache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	0 caches are enable	d by calling XCache_EnablelCache and	
<cache_enabled< td=""><td>Cache for instruction</td><td>and data respectively in your applications.</td><td></td></cache_enabled<>	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	0 caches are enable	d by calling XCache_EnableICache and	
<cache_enabled< td=""><td>Cache for instruction</td><td>and data respectively in your applications.</td><td></td></cache_enabled<>	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	0 caches are enable	d by calling XCache_EnableICache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
<cache_enabled< td=""><td>Cache for instruction</td><td>and data respectively in your applications.</td><td></td></cache_enabled<>	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
<cache_enabled< td=""><td>Cache for instruction</td><td>and data respectively in your applications.</td><td></td></cache_enabled<>	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
{Cache_EnableD	(Cache for instruction	and data respectively in your applications.	
Note: PowerPC44	10 caches are enable	d by calling XCache_EnableICache and	
KCache_EnableD	Cache for instruction	and data respectively in your applications.	

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

🗢 Base System	ı Builder - Sofiware Setup 🛛 🔀
- Devices to use	as standard input, standard output, and boot memory
STDIN:	RS232 Uart 1
STDOUT:	RS232 Uart 1
Boot Memory:	xps_bram_if_cntlr_1
Select the sam	tion selection ple C application that you would like to have generated. Each application will
include a linker	script.
Memory te: Illustrate sy	st stem aliveness and perform a basic read/write test to each memory in your system
Peripheral	selftest
Perform a si	mple self-test for each peripheral in your system.
More Info	< Back Next > Cancel

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

belect the men	nory devices which will be ased to hold the	rollowing program sections:
nstruction:	xps_bram_if_cntlr_1	
)ata:	xps_bram_if_cntlr_1	
Stack/Heap:	xps_bram_if_cntlr_1	
nterrupt Vec:	No interrupt	×
VARNING f you have pla ise a debugge	ced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memory	program in an external memory, you m y before you can run this program.
<mark>VARNING</mark> f you have pla use a debugge	ced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memory	program in an external memory, you m y before you can run this program.
VARNING f you have pla ise a debugge	iced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memor	program in an external memory, you m y before you can run this program.
VARNING f you have pla use a debugge	ced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memor	program in an external memory, you m y before you can run this program.
VARNING f you have pla use a debugge	ced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memor	orogram in an external memory, you m y before you can run this program.
VARNING f you have pla use a debugge	ced the Instruction or Data section of this p r, bootloader, or ACE file to initialize memory	program in an external memory, you m y before you can run this program.

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:

27

Processor: ppc440_ Processor clock free Bus clock frequency On Chip Memory : 1 Total Off Chip Memo	0 quency: 125.00 MHz x: 125.00 MHz 3 KB ary : 256 MB		
The address maps be editing features of XP	elow have been automatic 'S.	ally assigned. You c	an modify them using the
PLB Bus : PLB_V	46 Inst. name: plb_v	46_0 Attached	Components:
Core Name	Instance Name	Base Addr	High Addr
xps_bram_if_cntlr	xps_bram_if_cntlr_1	0xFFFFE000	OxFFFFFFF
xps_uartlite	RS232_Uart_1	0x84000000	0x8400FFFF
xps_timer	xps_timer_1	0x83C00000	0x83C0FFFF
PPC440MC Bus :	ppc440 0 PPC440M	C Attached Com	ponents:
Core Name	Instance Name	Base Addr	High Addr
ppc440mc_ddr2	DDR2_SDRAM	0x00000000	OxOffffff

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:

🗢 Base System Builder - Finisl	1	×
	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\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP D:\TestingExamples\ComplexFIR_PP	C440\EDK\system.mhs C440\EDK\data\system.ucf C440\EDK\etc\fast_runtime.opt C440\EDK\etc\download.cmd C440\EDK\system.mss C440\EDK\restApp_Memory\src\TestApp_Memory.c C440\EDK\TestApp_Memory\src\TestApp_Memory_LinkS C440\EDK\system.smp	
The settings file contains all the us loaded in a future wizard session.	er's selections and inputs in the wizard session. It can be	
More Info	< Back Finish Cancel	

Click **Finish** to close the wizard.

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

Xilinx Platform Studio - D:/TestingExamples/ComplexFIR_Platform	PC440/EDK/system.xmp - [Sys	tem Assembly \	/iew1]	
File Edit View Project Hardware Software Device Configuration De	bug Simulation Window Help			
1 🗗 🕅 🖬 🖉 🛰 🔏 🖻 🕼 🕺 1 🖬 🖬 👘	📓 🎨 🗋 🆻 🕼 💷	uba 📥 🛐 🏫	fiff 🎥 🕱 💥	i 🗈 🖻 🕅
Project Information Area ×	Bus Interfaces Ports	Addresses		
Project Applications IP Catalog	Name	Bus Connection	IP Type	IP Version
Platform	— 🕀 🗢 ppc440_0		ppc440_virtex5	1.01.a
🕞 Project Files	■ ⊕ <> plb_v46_0		plb_v46	1.03.a
MHS File: system.mhs	– 🕞 🥌 DDR2_SDRAM		ppc440mc_ddr2	2.00.a
MSS File: system.mss 🛛 🧹 🤚	— 🕀 🗢 xps_bram_it_cntlr_1		xps_bram_if_cntlr	1.00.a
	— 🕀 🧼 🗢 xps_bram_if_cntlr_1_bran.		bram_block	1.00.a
iMPACT Command File: etc/download.cmd	— 🕀 🥌 jtagppc_cntlr_0		jtagppc_cntlr	2.01.c
Implementation Options File: etc/fast_runtime.opt	- 🕀 🧼 proc_sys_reset_0		proc_sys_reset	2.00.a
Bitgen Options File: etc/bitgen.ut	— 🕀 🥌 xps_timer_1		xps_timer	1.00.a
😑 Project Options 🛛 🚽	- BS232_Uart_1		xps_uartlite	1.00.a
- Device: xc5vfx70tff1136-1	clock_generator_0		clock_generator	2.01.a
– Netlist: TopLevel – Implementation: XPS (Xflow)				

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:

🗢 Xilinx Platform Studi	io - D:/TestingEx	amples/Compl	exFIR_PPC440/E	DK/system.xi
File Edit View Project	Hardware Software	Device Configur	ation Debug Simul	ation Window
8 🗗 🕅 🗗 8 🗠 💥	6 G M E) 🗗 🛃 🔽 [2 848 🗟 🍀	🗋 🖻 🖥 🖁
Project Information Area		×	P	🚦 🛛 Bus Ini
Project Applications If	^o Catalog		L	Name
••				- + onc4
Description		IP Version		. ⊕ ⇒ plb v
😑 🗶 EDK Install D:\Xilinx\	10.1\EDK\hw\			🗉 🗢 DDR.
🕀 Analog				- 🕒 🗢 xps_1
Arithmetic				- 🕀 🗢 xps_l
🖨 Bus and Bridge			K	- 🕀 🤝 įtagpį
PLBV46 to PLI	BV46 Bridge	1.01.a	K	– 🕀 🥯 proc_
PLBV46 to DC	R Bridge	1.00.a	• •	- ⊕ > xps_1
🚽 🔶 🔶 🕂 🕂 🕂 🕂	al Bus (PLB) 4.6	1.03.a	<u>ه </u>	\blacksquare \blacksquare \blacksquare $H523$
🚽 🔶 PLB v 46 to FSI	. Bridge	1.00.a		- COCK
🚽 🔶 🔶 🔶	Bus (LMB) 1.0	1.00.a		
🚽 🛧 Fast Simplex L	ink (FSL) Bus	2.11.a		
	essor Bus V2.0(FCB)	100 a		
📩 📩 Device Contro	Register (DCR) Bus :	2. Add IP		10.00
🗉 Clock, Reset and Inter	rupt	View MPD		
Communication High-S	peed	View IP Mo	difications (Change L	(100
Communication Low-S	peed		Jahachaat	og,
DMA and Timer		VIEW PDF L	Vacasheet	

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

PF	Bus Interfaces	Ports	Addresses		
B B	Name		Bus Connection	IP Type	IP Version
	□ □ □ ppc440_0			ppc440_virtex5	1.01.a
	- RESETPPC		ppc_reset_bus	V	
	- JTAGPPC		jtagppc_cntlr_0_0	~	
	MFCM		ppc440_0_MFCM		
	- MFCB		No Connection	$\mathbf{\nabla}$	
$ \Psi$	SDCR		No Connection	▼	
	MDCR		No Connection	~	
6	- PPC440MC		ppc440 0 PPC440	VIC	
0	- SPLB1		No Connection	$\overline{\mathbf{v}}$	
l l o	- SPLB0		No Connection	~	
The second secon	- MPLB		plb v46 0	~	
	fcb v20 0			fcb_v20	1.00.a
	■ 🕀 🧼 plb_v46_0			plb_v46	1.03.a
	- 🕀 🧼 DDR2 SDRAM			ppc440mc ddr2	2.00.a
(- 🕀 🗢 xps bram if cnth	1		xps bram if onth	1.00.a
	– 🕀 🧼 xps_bram_if_cntli	1_bran		bram_block	1.00.a
	😑 🥏 apu_filt_0			apu_filt	1.00.a
6	- SFCB2		No Connection	v	
	– 🕒 🥯 jtagppc_cntli_0			jtagppc_cntlr	2.01.c
K	– 🕀 🧼 proc_sys_reset_b	7		proc_sys_reset	2.00.a
	- 🕀 🧼 xps_timer_1			xps_timer	1.00.a
5	- 🕀 🗢 RS232_Uart_1			xps_uartlite	1.00.a
1.00	clock generator	0		clock generator	2.01.a

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:

LC			LIDT	
BB	Name	Bus Connection	IP Type	IP Version
	😑 🗢 ppc440_0		ppc440_virtex5	1.01.a
		ppc_reset_bus		
	JTAGPPC	jtagppc_cntlr_0_0		
	MFCM	ppc440_0_MFCM		
		fcb_v20_0		
	SDCR	No Connection	✓	
	- MDCR	No Connection		
K		ppc440_0_PPC440M	AC.	
0		No Connection		
0		No Connection		
	MPLB	plb_v46_0		
			fcb_v20	1.00.a
	— 🕀 🗢 plb_v46_0		plb_v46	1.03.a
	— 🕀 🗢 DDR2_SDRAM		ppc440mc_ddr2	2.00.a
	—— 🗍 🕀 🥌 xps_bram_il_cntlr_1		xps_bram_if_cntlr	1.00.a
	—— 🕀 🗢 xps_bram_if_cntlr_1	bram.	bram_block	1.00.a
	🖨 🥯 apu_filt_0		apu_filt	1.00.a
	— SFCB2	fcb_v20_0		
K K	— 🕀 🗢 jtagppc_cntlr_0		jtagppc_cntlr	2.01.c
<	— 🕀 🗢 proc_sys_reset_0		proc_sys_reset	2.00.a
	—— 🗍 🕀 🥌 xps_timer_1		xps_timer	1.00.a
6	—— 🕒 🛥 RS232_Uart_1		xps_uartlite	1.00.a
	clock_generator_0		clock generator	201 a

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.

•	Bus Interfaces	Ports	Addresses	
Nam	e		Net	Direction
÷.<	External Ports			
÷.<	> ppc440_0			
÷.<	> fcb_v20_0			
	SYS_RST		sys_bus_reset	V
-	FCB_CLK		No Connection	
	> plb_v46_0 > DDR2_SDRAM > xps_bram_ii_cr > xps_bram_ii_cr	1 ntlr_1 ntlr_1_brai	fpga_0_DDR2_SDRA fpga_0_RS232_Uart_ proc_clk_s % sys_bus_reset	M_DDR2_WEA
	 apu_fill_0 jtagppc_cntlr_0 proc_sys_resel xps_timer_1 RS232_Uart_1 clock_generation) <u>-</u> 0 0	sys_clk_s sys_clk_s90 sys_periph_reset dcm_clk_s fpga_0_RS232_Uart_ sys_rst_s	.1_RX

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:

± ⇒ itagppc_cntlr_0	
+ > proc_sys_reset_t	2
🗄 🥌 clock_generator_	0
	Configure IP
	View MPD
	View IP Modifications (Change Log)
	View PDF Datasheet
	Browse HDL Sources
	Driver: generic_v1_00_a
	Delete Instance
	Filter Ports 🕨
	Hide Selection

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.

e clock gener	ator rator module	can generate required output clocks from give	en input reference/feedback clock(s) based on your requ	uirements. It serves as a central
cking resources	ce to meet a s Overview	II your system wide clocking needs. This tool w	ill help you configure the clock generator module and ins (HDL Toggle	stantiate or update it in your syster
Step 1: Speci Step 2: Speci	ify input cloc ify the outpu	:k details It clock requirements		
Please highlig	jht a clock p	out in the list below and configure it on the righ	t side. Port: CLKOUT5	
E Input &	: Feedback .KIN .KFBIN	dom_clk_s	Connected to: filt_co_clk	
⊡ Output	ts _KOUTO _KOUT1	proc_clk_s	Required frequency (Hz):	62,500,000
	KOUT2	svs_ck_s90 ck_200mhz_s DDR2_SDRAM_mi_modk_tiv2	Required phase shift:	0
	.KOUT5 .KOUT6	DDH2_3DHAM_MI_MCCKdV2 filt_co_clk	Grouping information:	NONE
	.KOUT7 .KOUT8		Buffered:	TRUE

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:

35



Note: if **co_clk** is missing from the **apu_filt_0** section, then will need to return to <u>step 3</u> 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:

😽 🛛 Bus Interfac	es Ports Addresses					Generate Addresses
Instance	Name 🔺	Base Address	High Address	Size	Bus Interface(s)	Bus Connection
plb_v46_0	C_BASEADDR			U	Not Applicable	
xps_bram_if_cntlr_	1 C_BASEADDR	Oxffffe000	Oxfffffff	8K	SPLB	plb_v46_0
xps_timer_1	C_BASEADDR	0x83c00000	0x83c0ffff	64K	SPLB	plb_v46_0
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400ffff	64K	SPLB	plb_v46_0
ppc440_0	C_IDCR_BASEADDR	06000000000	ОЬОО11111111	256	Not Connected	
DDR2_SDRAM	C_MEM_BASEADDR	0x0000000	OxOfffffff	256M	PPC440MC	ppc440_0_PPC440MC
ppc440_0	C_SPLB0_RNG_MC_BASEADDR			U	Not Connected	
ppc440_0	C_SPLB1_RNG_MC_BASEADDR			U	Not Connected	

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:

× wob	Crea Savi Bits Done	ting bit ng bit s tream ge !	map tream i neratio	n "sys n is c	stem.bi complet	it". ;e.		
Console Wim	< Output	Warning	Error		Ú.			

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 doubleclicking the **Add Software Application Project...** item. An **Add Software Application Project** dialog will appear. Type in the project name: **filter**. 37



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:

Project: filter Processor: ppc440_0 Executable: D:\Testing	jExamples\ComplexF	IR_PPC440\EDł		>h MI > <i>fct</i>	PLBU PLB 0_ <i>v20_0</i>	No Lonnection plb_v46_0
Compiler Options	Salard Same		dd to Desisat			50
- Sources	Dalact 2011cav	ulaggar Lilla to V	ad to subject			
- Headers	Look in:	Code		•	🗢 🗈 💣 🗐•	
	My Recent Documents	ComplexFilter.c				
	Desktop					
	My Documents					
	My Computer					
	My Network	File name:	"co_init.c" "ComplexFilter	.c'' ''Filter_:	sw.c''	Open
	1 10005	Files of type:	C Sources (*.c)		<u>.</u>	Cancel

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.

Project: filter Processor: ppc440 Executable: D:\Te)_0 stingExamples\Comp	olexFIR_PPC440\E	Dł				No Connec plb_v46_0
⊕ Compiler Options ⊕ Sources	Select Source/	Header File to	Add to Pro	ject			? 🛛
Headers	Look in:	Code		125	•	+ 🗈 📸 📰 -	
	My Recent Documents Desktop My Documents My Computer	ComplexFilter	.h				
	My Network Places	File name: Files of type:	"ComplexF	ilter.h" "Filter.h" : (*.h)		-	Open Cancel

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.

41

ections View:			Heap and Stack View	N:	
Section	Size (bytes)	Memory	Section	Size (bytes)	Memory
vectors	0x00000000	DDR2_SDRAM_	Heap	0x1000	DDR2_SDRAM_
text	0x00000000	DDR2_SDRAM_	Stack	0x1000	DDR2_SDRAM_
rodata	0x00000000	DDR2_SDRAM_		\sim	
rodata1	0x00000000	DDR2_SDRAM_			
sdata2	0x00000000	DDR2_SDRAM_			
sbss2	0x00000000	DDR2_SDRAM_			
data	0x00000000	DDR2_SDRAM_	Memories View:		
data1	0x00000000	DDR2_SDRAM_	Memory	Start Address	Length
fixup	0x00000000	DDR2_SDRAM_	DDR2_SDRAM_C_I	0x00000000	262144K
sdata	0x00000000	DDR2_SDRAM_	xps_bram_if_cntlr_1	0xFFFFE000	8K
sbss	0x00000000	DDR2_SDRAM_			
bss	0x00000000	DDR2_SDRAM_			
oot and Vecto	Add Se	ction Delete Section	ELF file used to popu D:\TestingExamples	ilate section informa	ation: 440\EDK\filter\executable
Section	Address	Memory			
boot0	0xFFFFFF00	xps_bram_if_cntlr_1	Output Linker Script:	PPC440\EDK\filte	er\filter_linker_script.ld
boot	0xFFFFFFFC	xps bram if onthr 1			

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.

File Edit Setup Control W	era Term: Serial por	t setup		
	Port:	СОМ1	ОК	
	Baud rate:	9600 🔹		
	Data:	8 bit 💌	Cancel	
	Parity:	none 💌		
	Stop:	1 bit 🔹	Help	
	Flow control:	none 💌		
	Transmit dela	y c/char 0 ms	sec/line	

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.

🔷 XMD Debug Opi	tions				X
Processor: ppc440_0	۵	vrchitecture: I	PowerPC		
Connection Type					
O Simulator	💿 Hardw	are	🚫 Stub		Virtual platform
On-Chip Hardware o	debugging over JTA	(G cable			
JTAG Properties	Advanced Options	e)			
JTAG Cable-					
Type: Auto			Frequency:	\sim	
Auto Direct	war ITAG Chain Da	ofinition			
	levice Name	ID Code	IB Length	1	
USC INU: L	evice indilie	ID CODE	in r cengui		
Add device	Delete device	3			
		_			
			οκ	Cancel	Help

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:

🗪 D:\Xi	linx\10.1\EDK\bir	1\nt\xbash.exe		- 🗆 🗙
Device 1 2 3 4 5	ID Code f5059093 f5059093 59608093 Øa001093 632c6093	IR Length 16 16 8 8 10	Part Name XCF32P XCF32P xc95144x1 System_ACE XC5UFX70T_U	
PowerPC	440 Processor	Configuration		
Version User ID No of P No of A User De	C Breakpoints ddr/Data Watc fined Address I-Cache (Data I-Cache (TAG D-Cache (Data D-Cache (TAG DCR	hpoints Map to access a)	.0x7ff21912 .0x00f00002 .4 .2 Special PowerPC Features using XMD: 00000 - 0x70007fff 08000 - 0x7000ffff 08000 - 0x78007fff 08000 - 0x78007fff 20000 - 0x78020fff 20000 - 0x70023fff	
Connect Startin XMD%	ed to "ppc" ta g GDB server H	arget. id = 0 for "ppc" targe	t (id = 0) at TCP port no 1234	-

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

D:\Xilinx\10.1\EDK\bin\nt\xbash.exe	- 🗆 ×
XMD% dow_filter/executable.elf	
System Reset DONE	I
Downloading Program filter/executable.elf	
section, .text: 0x0000000-0x00003b1b	
section, .init: 0x00003b1c-0x00003b3f	
section, .fini: 0x00003b40-0x00003b5f	
section, .boot0: 0xfffff00-0xffffffa7	
section, .boot: 0xfffffffc-0xffffffff	
section, .rodata: 0x00003b60-0x00004269	
section, .sdata2: 0x0000426c-0x0000426b	
section, .sbss2: 0x0000426c-0x0000426b	
section, .data: 0x0000426c-0x0000476b	
section, .got: 0x0000476c-0x0000476b	
section, .got1: $0 \times 0000476c - 0 \times 0000476b$	
section, .got2: 0x0000476c-0x00004787	
section, .ctors: $0 \times 00004788 = 0 \times 0000478f$	
section, .dtors: $0 \times 00004790 - 0 \times 00004797$	
section, fixup: 0x00004798-0x00004797	
section, .eh_frame:_0x00004798-0x0000479f	
section, .jcr: 0x00004?a0-0x00004?a3	
section, .gcc_except_table: 0x000047a4-0x000047a3	
section, .sdata:_0x00004?a4-0x00004?c?	
section, .sbss: 0x00004?c8-0x0000480b	
section, .bss: 0x0000480c-0x0000a923	
section, stack: 0x0000a924-0x0000b92f	
section, heap: 0x0000by30-0x0000cy2f	
Setting PC with Program Start Address Øxffffffffc	
XMDy con	
Info:Processor started. Type "stop" to stop processor	
	-

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

Loading the Sample Application Understanding the ImageFilterDMA Application Compiling the Application for Simulation Building the Application for Hardware Exporting the Hardware and Software Files Creating the ML403 Test Platform Adding the ImageFilterDMA Hardware Adding the Software Application Files Building and Downloading the Application

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:

49

🗝 Impulse CoDeveloper Applicatio	n Manag	er Universal Edition - [ImageFilterDMA] - [img_hw.c]
Eile Edit View Project Tools Windo	w <u>H</u> elp	
i 🛺 😅 🕼 🏠 🚇 😫 🖬 👗 🎙	3 🛱 '	つ ベニル ダー構 🧐 🍅 💂 🛗 図 🕨 🛗 🧠 쯶 😾 🥒 🗇 💂
Project Explorer 🛛 🕈 🗙	, img_sv	ı.c 🕞 img_hw.c
R i	10	
	13	<pre>#include <stalo.n> #include "co h"</stalo.n></pre>
	15	#include "cosim log h"
	16	#include "img b"
sw C mg_sw.c	17	#Include Img.II
testcpu.c	18	extern void call forma(co memory imomem, co signal start, co signal end):
C testfpga.c	19	
hw C img_hw.c	20	void to stream(co signal go, co memory imgmem, co stream output stream)
🖻 🔄 Header Files	21	
h img.h	22	intl6 i, j;
sw h test.h	23	uint32 offset, data, d0, d1;
Project Files	24	uint32 row[IMG_WIDTH / 2];
E 🖨 Document Files	25	IF_SIM(cosim_logwindow log;)
Beadme htm	26	
Other Files	27	<pre>IF_SIM(log = cosim_logwindow_create("to_stream");)</pre>
Culei riles	28	
	29	co_signal_wait(go, &data);
	30	
	31	co_stream_open(output_stream, U_WRUNLY, INT_TYPE(32));
	34	offset = 0;
	33	For (i = 0, i < IMC HEIGHT: i++) (
	35	comemory readblock(imamem offset row IMG WIDTH # sizeof(int)6)).
	36	for $(i = 0; i < (IMG WIDTH / 2); i++) /$
	37	#nrama CO PIPELINE

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, "PEO");
co_process_config(prep_proc, co_loc, "PEO");
co_process_config(filter_proc, co_loc, "PEO");
co_process_config(writer, co_loc, "PEO");
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:

C:\WINDOWS\system32\cmd.exe	C:\WINDOWS\system32\cmd.exe
"D:\TestingExamples\ImageFilterDMA\ImageFilterDMA.exe" Edge Detect Demo	Running hardware-accelerated filtering
Running software-only filtering Image in:	
.,00,. .000000. .0000000. .000000. 	.,00,. .000000. .0000000. .0000000,. .,,0,0,
Running Done Image out:	Running Done Image out: #0 0.00Goo. 0.% 00 0.% 00 0.0 0.0 0.0 0.0 0.0 0.0 0.

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

53

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:

Kilinx Virtex-4 PLB v4.6 (VHDL) CoBuilder Optimization Options Enable constant propagation Scalarize array variables Relocate loop invariant expressions	Directories
Additional optimizer options:	Hardware build directory:
	Software build directory:
CoBuilder Generation Uptions	SW
	Hardware export directory:
Use std. logic tupes for VHDL interfaces	EDK
Do not include co, ports in hus interface	Software export directory:
Library options:	EDK
 Include floating point library Use higher latency, faster clock operators Allow double-precision types and operators 	

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

🗞 📮
d

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:

: System Builder wizard (rec	ommended)j			
K AFS project				
n a recent project				
e Projects				
	n a recent project e Projects	n a recent project e Projects	n a recent project 9 Projects	n a recent project 9 Projects

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:

Dinitializing FPGA on-chip memor	Platform Stud	io Project			?×
♦ Create New XPS Project Usine BSB Wizard	Save in	EDK	•	🗢 🖻 💣 📰 •	
New project		Code drivers			
Project file	My Recent Documents	pcores			
Browse					
Advanced options (optional: F1 for help)	Desktop				
Set Project Peripheral Repositories					
Dowse	My Documents				
OK Cancel					
	My Computer				
[Platform Studio]	My Network Places	File name:	system.xmp	•	Save
	, ,0005	Save as type:	Platform Studio Project (*.xmp)	•	Cancel

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:

Velcome to the Base System Builder! is 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)		Embedded Development Kit Platform Studio
Velcome to the Base System Builder! is 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)		
his 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)	Vela	ome to the Base System Builderl
his 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)	TOIL	ome to the base of stem builder:
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) 		
 I would like to create a new design I would like to load an existing .bsb settings file (saved from a previous session) 	'his too	I will lead you through the steps necessary to create an embedded system.
I would like to load an existing .bsb settings file (saved from a previous session)	his too Pleas	I will lead you through the steps necessary to create an embedded system.
	Pleas	I will lead you through the steps necessary to create an embedded system. begin by selecting one of the following options: would like to create a new design

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:

Base System	Builder - Select Board	
Select a target de	velopment board:	
Select board		
💿 l would like	to create a system for the following development board	
Board vendor:	Xilinx	
Board name:	Virtex 4 ML403 Evaluation Platform	
Board revision:	1	
Note: Visit the v	vendor website for additional board support materials.	
Vendor's Websi	te Contact Info	
Download Third	Party Board Definition Files	
O I would like	to create a system for a custom board	

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:

Architecture:	Device:	Package:	Speed grade	9:
virtex4	xc4vfx12	M (ff668	-10	3
– 🔲 Use steppin	g			
		2 C		
lect the processor	you would like to use ir	n this design:		
lect the processor	you would like to use ir	n this design:		
ect the processor	you would like to use ir	n this design:		

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

requency:		frequency:		Bus clock freque	ency:	
100.00	MHz	100.00	MHz	50.00	MHz MHz	
Ensure that your l Reset polarity:	ooard is o Active	onfigured for the : • LOW	specifed f	requency.		
Processor configu	iration					
CPU debug	g user pir g and trai	is only ce pins				
	1		Use (Use Data	hip memory (OCM) BRAM) :		
Por			Instru	uction:		
Cache setup ✓ Enable For optimal perf and/or cachelir	ormance ne on me	, enable burst mory				
Enable floatin	ng point u	nit (FPU) <u>?</u>				

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:

🗢 Base System Builder - Configure 10 Interfaces (1 of 3)	X
The following external memory and IO devices were found on your board:	
Xilinx Virtex 4 ML4U3 with TFT Revision 1	
Please select the IO devices which you would like to use:	
- IU devices	
RS232_Uart	Data Sheet
Peripheral: XPS UARTLITE	
Baudrate (bits per seconds): 9600	
Data bits: 8	
Parity: NONE	
Use interrupt	
LEDs_4Bit	Data Sheet
Peripheral: XPS GPI0	
Use interrupt	
LEDs_Positions	Data Sheet
Push Buttons Position	
	Data Sheet
More Info Kext >	Cancel

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

🗢 Base System Builder - Configure 10	Interfaces (2 of 3)
The following external memory and IO devices w Xilinx Virtex 4 ML403 Evaluation Platform Revisio Please select the IO devices which you would lik	ere found on your board: on 1 :e to use:
- 10 devices	
	Data Sheet
SysACE_CompactFlash	Data Sheet
Cypress_USB	Data Sheet
DDR_SDRAM Peripheral: MPMC	Data Sheet
Ethernet_MAC	Data Sheet Note
More Info	< Back Next > Cancel

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

🗢 Base System Builder - Configure 10 Interfaces ((3 of 3) 🛛 🔀
The following external memory and IO devices were found on y Xilinx Virtex 4 ML403 Evaluation Platform Revision 1	our board:
Please select the IO devices which you would like to use:	
TriMode_MAC_GMII	Data Sheet
SRAM	Data Sheet
FLASH	Data Sheet

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

dd other peripherals that do not interact with off-chip compo Add Peripheral'' button to select from the list of available peri	nents. Use the ipherals.
you do not wish to add any non-IO peripherals, click the "N	ext" button.
	Add Peripheral.
Peripherals	
Peripherals	
Peripherals xps_bram_if_cntlr_1 Peripheral: XPS_BBAM IF_CNTLB	Remove

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

🗢 Add Peripheral	? 🗙
Select the peripheral you want to add:	
XPS TIMER	
OK Cancel	

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

d Peripheral" button to select from the list of available periph	nerals.
u do not wish to add any non-IO peripherals, click the "Nex	t" button.
	Add Periphera
ripherals	
_xps_bram_if_cntlr_1	
Peripheral: XPS BRAM IF CNTLR	Remove
Memory size: 16 KB 🔛	Data Sheet
xps_timer_1	
Peripheral: XPS TIMER	- Remove
Counter bit width: 32	Data Sheet
Timer mode	
 Two timers are present 	
 One timer is present 	

Click Next to continue.

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

u have enabled th	ne cache feature on l	the PowerPC processo	or.
ache setup			
Size of instruction	and data cache (car	n not be changed on F	PC):
Instruction C	ache (ICache) Size:	16 KB	~
Data Cache	(DCache) Size:	16 KB	*
Select the memory	v peripherals you wou	Id like to cache:	
ICache:	DCache:	C	acheable Memories:
		D	DR_SDRAM
		×p	os_bram_if_cntlr_1

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:

STDIN:	RS232_Uart	
STDOUT:	RS232_Uart	
Boot Memory:	xps_bram_if_cntlr_1	
Sample applica	stion selection	
Sample applica Select the sam include a linker	a <mark>tion selection</mark> aple C application that you would like to r script.	have generated. Each application will
Sample applica Select the sam include a linker	a <mark>tion selection</mark> uple C application that you would like to r script.	have generated. Each application will
Sample applica Select the sam include a linker Memory te Illustrate sy	a <mark>tion selection</mark> uple C application that you would like to r script. est ustem aliveness and perform a basic rea	have generated. Each application will d/write test to each memory in your system
Sample applica Select the sam include a linker Memory te Illustrate sy V Peripheral	a <mark>tion selection</mark> nple C application that you would like to r script. est setem aliveness and perform a basic rea selftest	have generated. Each application will d/write test to each memory in your systen

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:

he Peripheral S selftest functior	elftest application includes a simple self test n exists in the driver the peripheral).	t for each periperhal in your system (if suc
PeripheralTest		
Select the mer	nory devices which will be used to hold the	following program sections:
Select the mer	nory devices which will be used to hold the xps_bram_if_cntlr_1	following program sections:
Select the mer Instruction: Data:	nory devices which will be used to hold the xps_bram_if_cntlr_1 xps_bram_if_cntlr_1	following program sections:

Click Next.

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

low is a summary of t rrect, hit <generate> nerwise return to the</generate>	he system you have creat to enter the information in previous page to make co	ed. Please review th to the XPS data basi prrections.	e information below. If it is e and generate the system file
Processor: ppc405_ Processor clock free Bus clock frequency On Chip Memory : 1 Total Off Chip Memo - DDR_SDRAM =	0 quency: 100.00 MHz y: 50.00 MHz 6 KB 6 KB ory : 64 MB 64 MB		
Fhe address maps be editing features of XF	elow have been automatic 'S.	ally assigned. You c	an modify them using the
The address maps be aditing features of XP PLB Bus : PLB_V	elow have been automatic PS. 746 Inst. name: plb 4	ally assigned. You o	an modify them using the ents :
Fhe address maps be editing features of XF PLB Bus : PLB_V Core Name	elow have been automatic 'S. '46 Inst. name: plb Instance Name	ally assigned. You c Attached Compor Base Addr	an modify them using the nents: High Addr
The address maps be aditing features of XF PLB Bus : PLB_V Core Name xps_bram_if_cntlr	viow have been automatic S. 746 Inst. name: plb a Instance Name xps_bram_if_cntlr_1	ally assigned. You o Attached Compor Base Addr 0xFFFFC000	an modify them using the nents: High Addr 0xFFFFFFFF
The address maps be aditing features of XF PLB Bus : PLB_V Core Name xps_bram_if_cnttr xps_uartlite	 elow have been automatic 'S. '46 Inst. name: plb Instance Name xps_bram_if_cntlr_1 RS232_Uart 	Attached Compor Base Addr 0xFFFFC000 0x84000000	an modify them using the nents: High Addr 0xFFFFFFF 0x8400FFFF
The address maps be editing features of XF PLB Bus : PLB_V Core Name xps_bram_if_cnttr xps_uartlite xps_gpio	 elow have been automatic '46 Inst. name: plb / Instance Name xps_bram_if_cntlr_1 RS232_Uart LEDs_4Bit 	Attached Compor Base Addr 0xFFFFC000 0x84000000 0x81400000	ents: High Addr 0xFFFFFFF 0x8400FFFF 0x8140FFFF

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.

🗢 Base System Builder - Finish	1	×
	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\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\ D:\TestingExamples\ImageFilterDMA\	XEDK\system.mhs XEDK\data\system.ucf XEDK\etc\fast_runtime.opt XEDK\etc\download.cmd XEDK\system.mss XEDK\TestApp_Peripheral\src\TestApp_Peripheral.c XEDK\TestApp_Peripheral\src\xgpio_tapp_example.c XEDK\TestApp_Peripheral\src\gpio_header.h XEDK\TestApp_Peripheral\src\xtmrctr_selftest_example.c	
Save settings file:		
D:\TestingExamples\ImageFilterDN The settings file contains all the use loaded in a future wizard session.	MA\EDK\system.bsb er's selections and inputs in the wizard session. It can be	
More Info	K Back Finish Cancel	

Click Finish to exit the Wizard.

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

Xilinx Platform Studio - D:/TestingExamples/Imagel	FilterDMA/EDK/system.xr	np - <u>[</u> System As	sembly View1]		
File Edit View Project Hardware Software Device Configu	ration Debug Simulation Wir	ndow Help			
	🔍 Bas 🔝 🎨 🗅 🆻	🖥 🔓 🛛 🖉 🖷	a 📥 🛐 🏫 🗄 🖬 II	給 🛛 🕱 🖗 🖻	K K
Project Information Area 🛛 🗙 P	P P	Interfaces Port	s Addresses		
Project Applications IP Catalog	B B Name		Bus Connection	IP Type	IP Version
Platform		c405_0		ppc405_virtex4	2.01.a
🖨 Project Files				plb_v46	1.03.a
MHS File: system.mhs		c405_0_dplb1		plb_v46	1.03.a
MSS File: system.mss		■ 🔁 🥌 ppc405_0_iplb1		plb_v46	1.03.a
UCF File: data/system.ucf		DR_SDRAM		mpmc	4.02.a
- iMPACT Command File: etc/download.cmd 🔰 🧲 🖕	o-o-o-o	s_bram_if_cntlr_1		xps_bram_if_cntlr	1.00.a
Implementation Options File: etc/fast_runtime	🗕 🗕 🗐 🕀 🗢 plt	bram_if_cnth_1_b	van.	bram_block	1.00.a
Bitgen Options File: etc/bitgen.ut	🗲 🗲 🕀 🛥 jia	gppc_0		jtagppc_cntlr	2.01.Ь
🖨 Project Options 🤍 🤆		oc_sys_reset_0		proc_sys_reset	2.00.a
- Device: xc4vfx12ff668-10	o-o-o-o-	Ds_48it		xps_gpio	1.00.a
- Netlist: TopLevel	o-o-o-o	s_timer_1		xps_timer	1.00.a
Implementation: XPS (Xflow)	o-o-o-o-	232_Uart		xps_uartlite	1.00.a
HDL: VHDL Sim Model: BEHAVIORAL		ick_generator_0		clock_generator	2.01.a

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:

File Edit View Project Hardware			Soft	ware	Device Configuration Debug	:
			Isaunch Platform Studio SDK Image: Software Platform Settings			
Project Information Area						
Project	roject Applications IP Catalog		Assign Default Drivers			
Software F	Projects		Lip@	Gene	rate Libraries and BSPs	
E Ad E Pr E Pr E Co E So E He	fault: ppc405_0_ oject: TestApp ocessor: ppc405 ecutable: D:\Tes ompiler Options ources eaders	bootloop Peripheral 0 stingExamples		Add 9 Build Get P Gene Clear Clear	Software Application Project All User Applications Program Size rate Linker Script Libraries Programs	2018-201

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:

File E	dit View Proje	t Hardware	Software	Device Configuration	Debug
(†) (†	📑 🗄 🛤 🖓	XGG	00 🗄 🖻	INIT Update Bitstream	n
Project Inf	ormation Area	_	×	🔠 Download Bitstre	am
Project	Applications	IP Catalog		📑 Program Flash M	emory
Software	Projects		-		
Ad	ld Software Appli	cation Project		11777	11

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:

🗢 Xilinx Platform Studio - D:/Te	estingExamples/Imag
File Edit View Project Hardware	Software Device Config
8 🖬 📷 8 🛤 😅 🗶 🖻 🛱	AA 8 🗟 🗗 🔂 🔽
8800X?	
Project Information Area	
Project Applications IP Catalog	
1 2€	
Description	IP Version
😑 🐒 EDK Install D:\Xilinx\10.1\EDK\	\hw\
 ⊕_Analog	
Bus and Bridge	
Clock, Reset and Interrupt	
Communication High-Speed	
Communication Low-Speed	
DMA and Timer	
🕀 Debug	
FPGA Reconfiguration	
🕀 General Purpose IO	
Interprocessor Communication	
Memory and Memory Controller	
⊕ PCI	
Peripheral Controller	
Processor	
💼 Utility	
Project Local poores D:\TestingExa	amples
🖨 USER	
🔤 💁 plb_img_arch	1.00.a
	Add IP
	View MPD

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:

Project Information Area	×	PPP
Project Applications IP Catalog		
2 ●		
Description	IP Version	
🖨 🗶 EDK Install D:\Xilinx\10.1\EDK\hw\		
🕀 Analog		
Arithmetic		
🕀 Bus and Bridge		∳ ∳ Ò-
Elock, Reset and Interrupt		6-0-0-0-
🕀 Communication High-Speed		
Communication Low-Speed		
🖨 DMA and Timer		
─★ XPS Timer/Counter	1.00.a	
🚽 🛧 🔶 🔶 🚽 🔶	1.00.Ь	0-0-0-
- 🖳 🔆 XPS Central DMA Controller	2.00.Ь	
🚽 🚽 Fixed Interval Timer	Add IP	1
Debug FPGA Reconfiguration General Purpose I0 Interprocessor Communication	View MPD View IP Modifical View PDF Datash	tions (Change Log) neet

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:
PPPP	Bus Interfaces	Ports	Addresses		
BBBB	Name		Bus Connection	IP Type	IP Version
	⊕ <> ppc405_0			ppc405_virtex4	2.01.a
	⊕. <i>◆ plb</i>			plb_v46	1.03.a
	⊕ <> plb_v46_0			plb_v46	1.03.a
	⊕ <> ppc405_0_dpl	57		plb_v46	1.03.a
	⊕ <> ppc405_0_iplb	7		plb_v46	1.03.a
0 00	🔁 🥏 DDR_SDRAM				4.02.a
K 0000	🕀 🧼 xps_bram_il_ci	nth_1	Configure I	P	
	 ⇒ plb_bram_i/_cr. ⇒ jlagppc_0 ⇒ plb_img_arch_i ⇒ proc_sys_reset ⇒ xps_central_dn 	ntir_1_bran 0 <u>-</u> 0 ma_0	 View MPD View IP Mod View PDF D Browse HDL 	lifications (Change Log) atasheet Sources	1.00.a 2.01.b 1.00.a 2.00.a 2.00.b
0-	\oplus \checkmark LEDS_4BM		Driver: mpn	ic_v2_00_a →	00.a
ŏ-ŏ-ŏ-ŏ	⊕		Delete Insta	ance	.00.a
		0_10	Filter Bus In	terfaces 🕨	2.01.a
			Hide Selecti	on	

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:

PPPP	Bus Interfaces	Ports Addresses		
BBBB	Name	Bus Connection	IP Type	IP Version
	⊕ <> ppc405_0		ppc405_virtex4	2.01.a
	🕒 🗢 plb		plb_v46	1.03.a
	. ⊕ <> plb_v46_0		plb_v46	1.03.a
	⊕ <> ppc405_0_dplb	7	plb_v46	1.03.a
	⊕ <> ppc405_0_iplb1		plb_v46	1.03.a
	😑 🤝 DDR_SDRAM		mpmc	4.02.a
0-0-0-0-0-0-	-SPLB2	No Connection		
0-0-0-0-0-	- SPLB1	ppc405_0_dplb1	V	
	SPLB0	ppc405_0_iplb1	~	
K 0000	🕞 🕀 🗢 xps_bram_i/_cn	6k_7	xps_bram_if_cntlr	1.00.a
	🕀 🧼 🗩 plb_bram_if_cnt	ilr_1_bram	bram_block	1.00.a
	🕀 🍚 jtagppc_0		jtagppc_cntlr	2.01.Ь
	📔 🖨 🧼 plb_img_arch_0	122	plb_img_arch	1.00.a
<u> </u>	MPLB	No Connection		
0-0-0-0-0-0-	SPLB	No Connection	\sim	
K K	🗄 🗢 proc_sys_reset_	.0	proc_sys_reset	2.00.a
C VAL C COL	📄 🧼 xps_central_dm	a_0	xps_central_dma	2.00.Ь
<u> </u>	- SPLB	No Connection		
<u> </u>	MPLB	No Connection	\checkmark	
0-0-0- 0 -	🕒 🕀 🗢 LED 🖕 48 it		xps_gpio	1.00.a
	🕕 🕀 🧼 xps_timer_1		xps_timer	1.00.a
0-0-0- 0 -	🕀 🧼 🧇 RS232_Uart 👘		xps_uartlite	1.00.a
		<u>r</u> 0	clock_generator	2.01.a

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:

Đ	Bus Interfaces	Ports	Addresses	
Name			Net	Direction
	 External Ports ppc405_0 plb plb_v46_0 Bus_Error_Det 	t	No Connection	0
-	-SYS_Rst		No Connection	💌 I
	PLB_Clk		No Connection	▼
	 ppc405_0_dpll ppc405_0_iplb DDR_SDRAM xps_bram_if_cr plb_bram_if_cn jlagppc_0 	57 7 ntl <u>r_</u> 1 ntl <u>r_1_bran</u>	fpga_0_DDR_SDRAM_DI fpga_0_DDR_SDRAM_DI fpga_0_RS232_Uart_TX proc_clk_s sys_bus_reset sys_clk_s	DR_RAS_
	 plb_img_arch_i proc_sys_reset xps_central_dn LEDs_4Bit xps_timer_1 	0 <u>-</u> 0 na_0	sys_periph_reset dcm_clk_s fpga_0_RS232_Uart_RX sys_rst_s	
	RS232_Uart clock_generate	o <u>r_</u> 0		

And connect the SYS_Rst signal of plb_v46_0 to sys_bus_reset:

Ð	Bus Interfaces	Ports	Addresses	
Na	me		Net	Direction
	 External Ports ppc405_0 plb plb_v46_0 			
	Bus_Error_Det		No Lonnection	U
	-SYS_Hst		No Connection	
	 → ppc405_0_dplb → ppc405_0_iplb → DDR_SDRAM → xps_bram_if_cr 	57 7 Nd <u>r</u> 7	fpga_0_DDR_SDRAM_D fpga_0_DDR_SDRAM_D fpga_0_RS232_Uart_TX proc_clk_s sys_bus_reset	
	 plb_bram_if_cn jtagppc_0 plb_img_arch_t proc_sys_reset xps_central_dn 	tlf_1_bran) _0 va_0	 sys_clk_s sys_periph_reset dcm_clk_s fpga_0_RS232_Uart_RX sys_rst_s 	
	LEDs_4Bit xps_timet_1 RS232_Uast clock_generated	n_0		

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.

Bus Interfaces Ports Addresses Generate Addresses Base Address Size **High Address Bus Connection** Instance Name Bus Interface(s) Охсь600000 64K plb_img_arch_0 C_BASEADDR 0xcb60ffff SPLB plb plb C_BASEADDR U Not Applicable U plb_v46_0 C_BASEADDR Not Applicable ppc405_0_dplb1 C_BASEADDR U Not Applicable ppc405_0_iplb1 C_BASEADDR U Not Applicable Oxfiffc000 Oxffffffff 16K SPLB xps_bram_if_cntlr_1 C_BASEADDR plb Not Connect U LEDs_4Bit 0x81400000 0x8140ffff U SPLB plb C_BASEADDR SPLB C_BASEADDR 0x83c00000 0x83c0ffff 64 plb xps_timer_1 128 SPLB RS232_Uart 0x84000000 0x8400ffff plb C_BASEADDR 256 ppc405_0 C_IDCR_BASEADDR 060100000000 ОЬО111111111 Not Connected 512 SPLB0:SPLB1:SPLB2 DDR_SDRAM C_MPMC_BASEADDR 0x00000000 0x03ffffff 1K

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:

Bus Interfaces	Ports Addresses					Generate Addresses
Instance	Name 🔺	Base Address	High Address	Size	Bus Interface(s)	Bus Connection
plb_img_arch_0	C_BASEADDR	Охсь600000	0xcb60ffff	64K	SPLB	plb
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			Ú	Not Applicable	
xps_bram_if_cntlr_1	C_BASEADDR	Oxffffc000	Oxfffffff	16K	SPLB	plb
xps_central_dma_0	C_BASEADDR	0x00000000	0x0000003F	64	Not Connected	
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
ррс405_0	C_IDCR_BASEADDR	06010000000	0Ь0111111111	256	Not Connected	
DDR_SDRAM	C_MPMC_BASEADDR	0x0000000	0x03ffffff	64M	SPLB0:SPLB1:SP	LB2

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.

1 IS (P) }	88	Gener	ato Notlict	
1. A.			are menist	
rmation Area	12	Gener	ate Bitstre	am
Applications [38	Creat	e or Import	: Peripheral
rojects		cessor		
Software Applicat		Check	and View (Core Licenses
ault: ppc405_0_bc	33	Clean	Netlist	
ject: TestApp_1 cessor.ppc405_0	2	Clean	Bits	
ecutable: D:\Testir	13	Clean	Hardware	
	Applications [Applications] Software Applicat ault: ppc405_0_bc ject: TestApp_I cessor: ppc405_0 coutable: D:\Testir	mation Area Image: Constraint of the sector of the sec	mation Area Image: Configure Applications Applications Image: Configure Applications Software Applicat Check ault: ppc405_0_bc Image: Clean ject: TestApp_I Image: Clean cessor: ppc405_0 Image: Clean cutable: D:\Testin Image: Clean	Applications Image: Configure Copro Applications Image: Configure Copro Software Applicat Check and View ault: ppc405_0_bc Image: Clean Netlist ject: TestApp_ Image: Clean Bits cessor: ppc405_0 Image: Clean Hardware

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

🗙 wopuj	Crea Savi Bits Done	ting bit ng bit s tream ge !	map tream in neration	"system.bit". is complete.	
onsole W	<	a and a second s			

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:

Xillino File Edite File Edite File Edite	CPlatform Sti dit View Projec dit I im Projec	idio - D:/Tes tt Hardware : X 🖻 🖻 1	tingExamples/Imag Software Device Config 10 # 🖻 🗗 🔂 🏹	eFilterDMA guration Debu	EDK/system.comp g Simulation Windo	- [System Assemble w Help
Project Software	Applications Projects	IP Catalog	^			Bus Interfaces Name ■ ⊕ ∽ ppc405 0
E AG	efault: ppc405_0_ oject: TestApp ocessor: ppc405, eccutable: D:\Te ompiler Options ources eaders	bootloop Peripheral _0 stingExamples\Ir	Add Software Project Name Imag Note: Project Name Processor Project is an Choose an ELF fi The ELE file is as	Application geFilterDMA cannot have sp ELF-only Projection	Project paces. ppc405_0 t	Browse
			Default ELF name	e is <sw project<="" td=""><td>name>/executable.elf</td><td>Cancel</td></sw>	name>/executable.elf	Cancel

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:

Project Inf	ormation Area	
Project	Applications	IP Catalog
Software I	Projects	
Ad	ld Software Appli	cation Project
De	efault: ppc405_0	_bootloop
Pr Pr	oject: TestAp	p_Peripheral
⊕ Pr	rocessor: ppc405	<u>i</u> 0
Ex	kecutable: D:\Te	stingExamples\ImageFilterDMA\EDK\TestApp_Peripl
🕀 Co	ompiler Options	
⊕ Se	ources	
⊕ H	eaders	
	oject: ImageF	lterDMA
🕀 Pr	rocessor: ppc405	<u>_</u> 0
E:	kecutable: D:\Te	stingExamples\ImageFilterDMA\EDK\ImageFilterDMA
E Co	ompiler Options	
S	ources	Add Existing Files
HI	eaders	Add New File
		Had Now Flicth

Select Source/H	leader File to	Add to Project			? 🗙
Look in:	Code		•	← 🗈 💣 💷 -	
My Recent Documents Desktop	co_init.c img_sw.c				
My Documents					
My Computer					
My Network Places	File name:	"co_init.c" "img_st	w.c''	. [Open
1 1003	Files of type:	C Sources (*.c)		•	Cancel

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:

oftware P	rojects		
2 A 11			
Calcolor Ca	Software Appli ault: ppc405_0_ ject: TestApp cessor: ppc405 acutable: D:\Te mpiler Options urces	cation Project _bootloop p_Peripheral i_0 stingExamples\Image	FilterDMA\EDK\TestApp_Per
⊕ He	aders	16D144	
Pro ■ Pro Exe ■ Cor ■ Soi	Generated He soutable: D:\Te mpiler Options urces	itterDMA i_0 ader: ppc405_0/inclu stingExamples\Image	Set Compiler Options Mark to Initialize BRAMs Build Project Clean Project Delete Project
0.44	D:\TestingExa D:\TestingExa	mples/ImageFilterDM mples/ImageFilterDM	Make Project Inactive Generate Linker Script

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:

Project	Applications	IP Catalog	
Software I	Projects		
C Ad C Ad Pr Pr E C ⊕ Cı ⊕ Sı	Id Software Appli efault: ppc405_0_ oject: TestApj occessor: ppc405 kecutable: D:\Te ompiler Options ources	cation Project _bootloop p_ Peripheral 5_0 stingExamples\Imag	eFilterDMA\EDK\TestApp_Periphe
. ⊕ H ⊐. ₩ D.	eaders oiget: ImageEi	iterDMA	
Project: ImageFilterDMA Processor: ppc405_0 Generated Header: ppc405_0/inclu Executable: D:\TestingExamples\Image Compiler Options Sources		Set Compiler Options Mark to Initialize BRAMs Build Project Clean Project Delete Project	
	 D:\TestingExa D:\TestingExa 	mples\ImageFilterDN mples\ImageFilterDN	, Make Project Inactive
⊟ H	eaders code\img.h		Generate Linker Script,

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

81

Sections View:			Heap and Stack View	N:	
Section	Size (bytes)	Memory	Section	Size (bytes)	Memory
vectors.	0x00000000	DDR_SDRAM_C	Heap	0x2000000	DDR_SDRAM_C
.text	0x00000000	DDR_SDRAM_C	Stack	0x1000000	DDR_SDRAM_C
.rodata	0x00000000	DDR_SDRAM_C			
.rodata1	0x00000000	DDR_SDRAM_C			
.sdata2	0x00000000	DDR_SDRAM_C			
.sbss2	0x00000000	DDR_SDRAM_C			
data	0x00000000	DDR_SDRAM_C	Memories View:		
.data1	0x00000000	DDR_SDRAM_C	Memory	Start Address	Length
fixup	0x00000000	DDR_SDRAM_C	DDR_SDRAM_C_M	0x00000000	65536K
.sdata	0x00000000	DDR_SDRAM_C	xps_bram_if_cntlr_1	0xFFFFC000	16K
.sbss	0x00000000	DDR_SDRAM_C			
.bss	0x00000000	DDR_SDRAM_C			
Soot and Vecto	Add Se	ction Delete Section	ELF file used to popu stingExamples\Image	llate section inform FilterDMA\EDK\Ir	ation: nageFilterDMA\exec
Section	Address	Memory			
.boot0	0xFFFFFFEC	xps_bram_if_cntlr_1	Output Linker Script:	erDMA\ImageFilt	erDMA_linker_script.I
i	0-FEFEFEC	upe bram if onth 1			

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:

📕 Tera Term - COM1 VT				
File Edit Setup Control Win	Tera Term: Serial po	rt setup]
	Port: Baud rate: Data: Parity: Ston:	COM1 9600 8 bit none 1 bit	OK Cancel Help	
	Flow control: Transmit dela	none v ay c/char 0	msec/line	

Click OK to accept the settings.

Select Download Bitstream as shown below:

🖹 File Ed	dit View Projec	t Hardware	Software	Device Configuration	Debug
67 🕅	📬 🗄 너 🎯	X B B	(∆ :: ≥	រំដំរី។ Update Bitstream	15
	<u> </u>	2 4 %	2 34	😫 Download Bitstrea	am
Project Inf	ormation Area	-		📑 Program Flash Me	emory
Project	Applications	IP Catalog			
Software I	Projects	28. 			
hor 3	d Software Appli	cation Project	2		

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

Configuration	Debug	Simulation Windo	w Help
) 🖸 🖹 🔡	De 🏂 XM	bug Configuration D Debug Options	
	🐹 La	unch XMD	
-	💥 La	unch Software Debu	ugger
		Instance	Name

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.

🐝 D:\Xi	linx\10.1\EDK\bii	n\nt\xbash.exe		- 🗆 ×
Device 1 2 3 4	ID Code Øa001093 f5059093 Ø1e58093 49608093	IR Length 8 16 10 8	Part Name System_ACE XCF32P XC4UFX12 xc95144x1	
PowerPC	405 Processor	Configuration		
Version User ID No of P No of & No of W User De	C Breakpoints ead Addr/Data rite Addr/Dat fined Address I-Cache (Dat I-Cache (TAG D-Cache (TAG D-Cache (TAG DCR. TLB.	Watchpoints. a Watchpoints. Map to access a>0x700 a>0x780 0x780 0x780 0x780 0x780	.0x20011430 .0x00000000 .4 .1 .1 Special PowerPC Features using XMD: 00000 - 0x70003fff 04000 - 0x78003fff 04000 - 0x78003fff 04000 - 0x78007fff 04000 - 0x78004fff 04000 - 0x78004fff	
Connecto Starting XMD% _	ed to "ppc" t g GDB server :	arget. id = 0 for "ppc" targe	t (id = 0) at TCP port no 1234	•

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

dow ImageFilterDMA/executable.elf con

85

D:\Xilinx\10.1\EDK\bin\nt\xbash.exe	- 🗆 🗙
XMD% dow ImageFilterDMA/executable.elf	-
System Reset DUNE	
Downloading rrogram imagerilterprin/executable.elf	
Section, .text. 0x00000000000x0x000000005	
Section, finit, 0.00003037	
section, bouts exfifting exfifting	
section, models 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,	
section, route $0.0000310 - 0.000001233$	
section, success 2^{-1} $0 \times 000004238 - 0 \times 00004237$	
section, data: 0x00004238-0x00005bbb	
section, and: 0x00005 hbc-0x00005 hbb	
section, .got1: 0x00005bbc-0x00005bbb	
sectiongot2: 0x00005bbc-0x00005bd7	
section, .ctors: 0x00005bd8-0x00005bdf	
section, .dtors: 0x00005be0-0x00005be7	
section, .fixup: 0x00005be8-0x00005be7	
section, .eh_frame: 0x00005be8-0x00005bef	
section, .jcr: 0x00005bf0-0x00005bf3	
section, .gcc_except_table: 0x00005bf4-0x00005bf3	
section, .sdata:_0x00005bf4-0x00005c17	
section, .sbss:_0x00005c18_0x00005c4f	
section, bss: 0x00005c50-0x00005ca?	
section, .stack: 0x00005ca8-0x01005caf	
section, heap: 0x01005cb0-0x03005caf	
Setting PG with Program Start Hddress Øxfffffffc	
XMD% con	
Info-rrocessor started. Type "stop" to stop processor	
RUNNING> XMD%	+

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

	lit	Set	up	(Cor	htr	bl	W	'ind	ow		He	lp						
Runni	ng	ha	r	łw.	ar	·e·	-a	cc	el	eı	ra	te	ed		Fi:	lte	er	ing	
Image	i	1:																	
		.,0	0,																
	.00)00)00		00	ċ.	13													
	, 00	000	000	,	• •	3													
;				.,	;;	;													
	•••		••			•													
			00	00	00	Ó.								•					
			00)),(οο ο,				0202 0202										
••		• • •	• • •	• •	.,			693	191	393									
				.,	;;	2													
			•••	• •	•••	•				1		đ		•					
1.000	• • •			.,	;;	;													
			:		• •	-								:					
	•••	• • •			.,				191	393									
	•		;;	:;	22									1					
									111	88		•		•					
Ruppi																			
Runni Done Image	ng ol		8																
Runni Done Image # o	ng ol	 .t:	00	0,	0.	0													
Runni Done Image # o	ng ou ō	 	00), i	0.	0,													
Runni Done Image # o	ng ou 0	 it: ó,	00), 0	0.	0													
Runni Done Image # o	ng ou 0 0	ut: 0, &	00	0,0	0.	0													
Runni Done Image # o o	ng. ol	ut: 0, &	00), 0 0 	o.,;	0													
Runni Done Image # o o o o o o	ng. 00 00 0		00			0		, , ,	0,	00	00	00	00	00),,				
Runn i Done I mage # o o o o	ng ou 0 0 0 0 0	ut: 0, &	00	0,00	0. ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;	0		, o	0,	00	00	00	00	00),	•			
Runni Done Image # o o o o o o		ut: Ó, &	00	0,00				· · o · · · · · · · · · · · · · · · · ·	.o.	00	00	00	>0						
Runn i Done I mage # 0		0.	00						0,	0(00	00	>0						
Runni Done Image # o			00						0	00	00	00	00						
Runni Done Image # 0		0.	0(, , , ,						0,	00	00	00	>0						
Runn i Done Image # 0		(1) (1) (2) (2) (3) (4) (4) (4) (4) (4) (4) (4) (4) (4) (4	00						0,	0(00	00	>0						
Runn i. Done Image # 0			00						0,	00	00	00	00						
Runn i Done Image # 0		ut: ó, &							0,	00	00	00	00						
Runn i Done Image # 0									0, , , , , , , , , , , , , , , , , , ,	00	00	00	>0						
Runni Done Image # 0 		1t: 0, 8, - - - - - - - -							0	0(00	00	00						
Runn i Done Image # 0		0.	0				· · · · · · · · · · · · · · · · · · ·		0,	00	00	00	000						

© 2009 ... Your company

execution time : 13 milliseconds --> Acceleration factor: 103X

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

Loading the Sample Application Understanding the Mandelbrot Application Compiling the Application for Simulation Building the Application for Hardware Exporting the Hardware and Software Files Copying the TFT display core files Creating the ML403 Test Platform Adding the Mandelbrot Hardware Adding the Software Application Files Building and Downloading the Application

See Also

<u>Tutorial 1: Complex FIR Filter on Virtex-5 Platform (EDK 10.1)</u> <u>Tutorial 2: Image Filter DMA Using Shared Memory on the Virtex-4 Platform (EDK 10.1)</u>

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:

Build	4 ×
======================================	~
======= Build of target 'build_exe' complete =======	~
🔛 Build 🖼 Find in Files 🔄 System	

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:

CoBuilder Optimization Options CoBuilder Optimization Options CoBuilder Optimization Options CoBuilder Optimization Options CoBuilder Optimizer array variables CoBuilder Optimizer options: CoBuilder Optimizer options:	Directories Hardware build directory:
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:	hw Software build directory: sw Hardware export directory: EDK Software export directory: EDK
Library options:	

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

lication Manager Universal Edition - [Mandelbr	rot_1] - [Readme.htm]
Help	
🗈 🛍 ち ペール 🎢 🛤 🌿 🍅 📕 🛗 🖻 🕨 🛗	🗑 🗠 💥 🛷 💂
Start Page Readme.htm	Export Generated Hardware (HDL)
Export the Software Files	
ication Manager Universal Edition - [Mandelbrot_	1] - [Readme.htm]
<u>Help</u>	
a 🖻 5 7 // 9/ A 🕼 🍅 📙 🛗 🖻 🕨 🛗 🖷 🦉	2 🔀 🥜 📀 💂
E 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

Copying the TFT Display Core Files

97

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:

eate new or open existing project	
Base System Builder wizard (recommended) Base System Builder wizard (recommended)	
🔄 🔿 Blank XPS project	
🔽 🔿 Open a recent project	
Browse for More Projects	
srowse for More Projects	
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:

Dinitializing FPGA on-chip memo	Platform Stud	io Project			? 🗙
Create New XPS Project Using BSB Wizard	Save in:	EDK	•	-	
New project		Code drivers			
Project file	My Recent Documents	pcores			
Browse					
Advanced options (optional: F1 for help)	Desktop				
Set Project Peripheral Repositories					
Dibwse	My Documents				
OK Cancel					
	My Computer				
[Platform Studio]	My Network Places	File name:	system.xmp	.	Save
		Save as type:	Platform Studio Project (*.xmp)	•	Cancel

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:

Velcome to the Base System Builder! is 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)		Embedded Development Kit Platform Studio
Velcome to the Base System Builder! is 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)		
his 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)	Vela	ome to the Base System Builderl
his 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)	TOIL	ome to the base of stem builder:
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) 		
 I would like to create a new design I would like to load an existing .bsb settings file (saved from a previous session) 	'his too	I will lead you through the steps necessary to create an embedded system.
I would like to load an existing .bsb settings file (saved from a previous session)	his too Pleas	I will lead you through the steps necessary to create an embedded system.
	Pleas	I will lead you through the steps necessary to create an embedded system. begin by selecting one of the following options: would like to create a new design

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:

🗢 Base System	Builder - Select Board	×
Select a target de	velopment board:	
Select board		
💿 l would like	to create a system for the following development board	
Board vendor:	Xilinx	
Board name:	Virtex 4 ML403 Evaluation Platform	
Board revision:	1	
Note: Visit the v	vendor website for additional board support materials.	
Vendor's Websi	ite <u>Contact Info</u>	
Download Third	Party Board Definition Files	
O I would like	to create a system for a custom board	

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:

Architecture:	Device:	Package:	Speed grade:	
virtex4	xc4vfx12	💉 (ff668	-10	3
– 🔲 Use steppin	g			
	T .:			
		×		
		×		
	neu mentet like te me is	Vkia dosiga:		
lect the processor	you would like to use ir	this design:		
lect the processor	you would like to use ir	this design:		
lect the processor	you would like to use ir	this design:		
ect the processor rocessors	you would like to use ir	this design:		

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

requency:		Processor clo frequency:	ck	Bus clock f	equency:	
100.00	MHz	200.00	MHz	100.00	MHz	
insure that your bo Reset polarity:	ard is c Active	onfigured for th	e specifed fr	equency.		
rocessor configura	ation					
CPU debug CPU debug CPU debug No debug Cache setup For optimal perfo	user pin: and trac	e pins	On-cl (Use Data 16 k Instru	hip memory (C BRAM) : (B action: NE	CM)	

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:

ne following external memory and IO devices were found on your l	board:
ease select the IU devices which you would like to use:	
IU devices	
RS232_Uart	
Peripheral: XPS UARTLITE	Data Sheet
Baudrate (bits per seconds): 9600	
Data bits: 8	
Parity: NONE	
LEDs_4Bit	Data Sheet
Peripheral: XPS GPI0	
Use interrupt	
LEDs_Positions	Data Sheet
Push_Buttons_Position	Data Sheet
More Info	Next > Cancel

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

🗢 Base System Builder - Configure	IO Interfaces (2 of 3)	×
The following external memory and IO device Xilinx Virtex 4 ML403 Evaluation Platform Re Please select the IO devices which you wou	es were found on your board: vision 1 Id like to use:	
10 devices		
		Data Sheet
SysACE_CompactFlash		Data Sheet
Cypress_USB		Data Sheet
DDR_SDRAM Peripheral: MPMC		Data Sheet
Ethernet_MAC		Data Sheet Note
More Info	K Back Next	> Cancel

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

🗢 Base System Builder - Configure 10 Interfaces (t of 3) 🛛 🔀
The following external memory and IO devices were found on yo Xilinx Virtex 4 ML403 Evaluation Platform Revision 1 Please select the IO devices which you would like to use: 10 devices	ur board:
TriMode_MAC_GMII	Data Sheet
SRAM	Data Sheet
FLASH	Data Sheet

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.	
	Add Peripheral
Peripherals	
xps_bram_if_cntlr_1	
Peripheral: XPS BRAM IF CNTLR	Remove
Memory size: 8 KB	Data Sheet

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

ache setup			
ize of instruction	and data cache ícar	n not be changed on PPC	
Instruction C	ache (ICache) Size:	16 KB	×
Data Cache	(DCache) Size:	16 KB	*
elect the memory	peripherals you wou	ıld like to cache:	
ICache:	DCache:	Cach	eable 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:

evices to use	as standard input, standard output, and boot memory
TDIN:	RS232_Uart
TDOUT:	RS232_Uart
loot Memory:	ppc405_0_iocm_cntlr
ample applica	tion selection
ample applica	ition selection
ample applica ielect the sam include a linke	i <mark>tion selection</mark> iple C application that you would like to have generated. Each application will r script.
ample applica ielect the sam include a linker	<mark>ition selection</mark> iple C application that you would like to have generated. Each application will r script. st
ample applica jelect the sam nclude a linker Memory te Illustrate sy	i <mark>tion selection</mark> iple C application that you would like to have generated. Each application will r script. ist istem aliveness and perform a basic read/write test to each memory in your syster



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:

he Peripheral S selftest function	elftest application includes a simple self test n exists in the driver the peripheral).	for each periperhal in your system (if su
PeripheralTest		
Select the mer	nory devices which will be used to hold the l	following program sections:
Instruction:	ppc405_0_iocm_cntlr	
Instruction: Data:	ppc405_0_iocm_cntlr ppc405_0_docm_cntlr	
Instruction: Data: Stack/Heap:	ppc405_0_iocm_cntlr ppc405_0_docm_cntlr ppc405_0_docm_cntlr	

Click Next.

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

rect, hit <generate: nerwise return to the</generate: 	> to enter the information into previous page to make corr	o the XPS data basi rections.	e and generate the system file
Processor: ppc405 Processor clock fre Bus clock frequend On Chip Memory : Total Off Chip Mem - MPMC = 64 MB	_0 equency: 200.00 MHz cy: 100.00 MHz 32 KB lory : 64 MB		
he address maps b diting features of X	elow have been automatica PS. V46 Inst. name: nlb. 4	lly assigned. You o	an modify them using the
Core Name	Instance Name	Base Addr	High Addr
Core Name (ps_uartlite	Instance Name RS232_Uart	Base Addr 0x84000000	High Addr 0x8400FFFF
Core Name (ps_uartlite (ps_gpio	Instance Name RS232_Uart LEDs_4Bit	Base Addr 0x84000000 0x81400000	High Addr 0x8400FFFF 0x8140FFFF
Core Name (ps_uartlite (ps_gpio (ps_gpio	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0	Base Addr 0x84000000 0x81400000 0x81420000	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF
Core Name (ps_uartlite (ps_gpio (ps_gpio Processor OCM:	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0	Base Addr 0x84000000 0x81400000 0x81420000	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF
Core Name (ps_uartlite (ps_gpio (ps_gpio Processor OCM: Core Name	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0 Instance Name	Base Addr 0x84000000 0x81400000 0x81420000 Base Addr	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF High Addr
Core Name (ps_uartlite (ps_gpio (ps_gpio Processor OCM: Core Name sbram_if_cntlr	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0 Instance Name ppc405_0_iocm_cnttr	Base Addr 0x84000000 0x81400000 0x81420000 Base Addr 0xFFFFC000	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF High Addr 0xFFFFFFFF
Core Name (ps_uartlite (ps_gpio (ps_gpio Processor OCM: Core Name sbram_if_cntlr Processor OCM:	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0 Instance Name ppc405_0_iocm_cnttr	Base Addr 0x84000000 0x81400000 0x81420000 Base Addr 0xFFFFC000	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF High Addr 0xFFFFFFFF
Core Name <ps_uartlite <ps_gpio <ps_gpio Processor OCM: Core Name sbram_if_cntlr Processor OCM: Core Name</ps_gpio </ps_gpio </ps_uartlite 	Instance Name RS232_Uart LEDs_4Bit LCD_7Bit_GPI0 Instance Name ppc405_0_iocm_cntlr Instance Name	Base Addr 0x84000000 0x81400000 0x81420000 Base Addr 0xFFFFC000 Base Addr	High Addr 0x8400FFFF 0x8140FFFF 0x8142FFFF High Addr 0xFFFFFFFF High Addr

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:

×	Libraries generated in D:\TestingExamples\Mandelbrot_Virtex4FX\EDK\ppc405_0\lib\ directory
	Running execs_generate for OS'es, Drivers and Libraries
,wopui	LibGen Done. Done!
sole V	
Co	Output Warning Error

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.

File E	dit View Project	Har	dware	Software	Device Configuration
i 🗗 🗑 📑 i 🖄 🍽 🎖			Gener	ate Netlist	
Project Information Area		1	Gener	ate Bitstre	am 🖡
Project	ect Applications		Creat	e or Import	Peripheral
Software Projects			Config	gure Copro	cessor
			Check	and View (Core Licenses
De	fault: ppc405_0_bo	33	Clean	Netlist	
Processor: ppc405_0 Executable: D:\Testir			Clean	Bits	
			Clean	Hardware	ŀ

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:

Project Information Area	
	tstream
Project Applications IP Catalog	sh Memory
Software Projects	

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:

5	Bus Interfaces	Ports	Addresses		
Name		Bus Connection	IP Type	IP Version	
÷.	⇒ррс405_0			ppc405_virtex4	2.01.a
	ppc405_0_doc	m		dsocm_v10	2.00.Б
			isocm_v10	2.00.Ь	
			plb_v46	1.02.a	
				plb_v46	1.02.a
	⇒ppc405_0_iplb	7		plb_v46	1.02.a
Đ.	ppc405_0_doc	m_cnth		dsbram_if_cntlr	3.00.Б
	ppc405_0_iocr.	n_cnth		isbram_if_cntlr	3.00.Б
	DDR_SDRAM			mpmc	4.01.a
÷	🧼 dsocm_bram			bram_block	1.00.a
÷	🧼 isocm_bram			bram_block	1.00.a
 → jiagppc_0 → proc_sys_reset_0 → → LCD_7Bit_GPI0 → → LEDs_4Bit → RS232_Uart 			jtagppc_cntlr	2.01.a	
			proc_sys_reset	2.00.a	
			xps_gpio	1.00.a	
			xps_gpio	1.00.a	
			xps_uartlite	1.00.a	
	🧼 clock_generato	0_10	Configure IP .		2.01.a
-			View MPD View IP Modific Browse HDL So	cations (Change Log) burces	
			Driver: generio	:_v1_00_a 🔹 🕨	
-			Delete Instanc	e	
			Filter Bus Inter	rfaces 🕨	
			Hide Selection	8	-

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

asic Ports O	/erview	HDL	Toggle 🧏 Datasheet 📿 Restor
itep 1: Specify in	put clock details		
tep 2: Specify t	e output clock requirements		
Please highlight a clock port in the list below and configure its requirement		ts on the right side	
rease highlight a clock port in the list below and conlighte its requirements		Clock requirement: CLKOUT3	
Ports	Connected to		
- Input & Fe	edback dcm_clk_s 3IN	Connected to: ccore_co_c1k	
Outputs CLK0	JTO proc_clk_s	Required frequency (Hz):	40,000,000
CLKO CLKO CLKO	JII sys_clk_s JT2 DDR_SDRAM_mpmc_clk_90_s JT3 pcore_co_clk	Required phase shift:	0
- CLKO	JT4 JT5	Grouping information:	NONE
- CLKO			

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

ck generator	module can generate required output clocks from giv	ven input reference/feedback clock(s) based on your requir	ements. It serves as a central clocking
Ports Ov	erview	HD	L Toggle 🏾 🎏 Datasheet 🛛 🏹 Re
 1: Specify inp 2: Specify the 	out clock details e output clock requirements		
ise highlight a	clock port in the list below and configure its requirem	ents on the right side.	
Ports	Connected to	Llock requirement: LLNOUT4	
∃ Input & Fee CLKIN	dback dcm_clk_s N	Connected to: tft_25mhz_c1k	
Outputs CLKOL	ITO proc_clk_s	Required frequency (Hz):	25,000,000
	IT2 DDR_SDRAM_mpmc_clk_90_s IT3 pcore_co_clk	Required phase shift:	0
	IT4 ttt_25mhz_clk IT5	Grouping information:	NONE
LLKUL	пр	D. ((_)	TIDITE T

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.

Project Inl	ormation Area		×
Project	Applications	IP Catalog	
Platform			
Proje	ct Files HS File: system.r SS File: system.r CF File: data/sys	nhs nss tem.ucf	
-iN -In B ⊡ Proje	IPACT Command nplementation Op itgen Options File ct Options	l File: etc/download.c tions File: etc/fast_ru : etc/bitgen.ut	:md intime.opt

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
                                        LOC = G8; \# VGA_B6
NET plbv46_tft_cntlr_0_TFT_LCD_B_pin<4>
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 O DDR SDRAM DDR DQ<31> IOSTANDARD = SSTL2 II;
178
     Net fpga O DDR SDRAM DDR Clk pin LOC=A10;
     Net fpga_0_DDR_SDRAM_DDR_C1k_pin_IOSTANDARD = DIFF_SSTL2_II;
179
     Net fpga O DDR SDRAM DDR Clk n pin LOC=B10;
180
181
     Net fpga O DDR SDRAM DDR Clk n pin IOSTANDARD = DIFF SSTL2 II;
182
     #### Module plbv46 tft cntlr constraints
183
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
187NET plbv46 tft cntlr 0 TFT LCD B pin<3>LOC = B7; # VGA B5188NET plbv46 tft cntlr 0 TFT LCD B pin<4>LOC = G8; # VGA B6189NET plbv46 tft cntlr 0 TFT LCD B pin<5>LOC = F8; # VGA B7
190 NET plbv46 tft cntlr O TFT LCD B pin<*> SLEW = FAST | DRIVE = 8;
191
     192
193 NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<2> LOC = D3; # VGA_G4
194 NET plbv46 tft cntlr O TFT LCD G pin<3> LOC = H7; # VGA G5
195NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<4>LOC = H8; # VGA_G6196NET plbv46_tft_cntlr_0_TFT_LCD_G_pin<5>LOC = C1; # VGA_G7
197 NET plbv46 tft cntlr O TFT LCD G pin<*> SLEW = FAST | DRIVE = 8;
198
199
     NET plbv46 tft cntlr O 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
202NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<4>LOC = E5; #VGA_R6203NET plbv46_tft_cntlr_0_TFT_LCD_R_pin<5>LOC = E6; #VGA_R7
204 NET plbv46 tft cntlr O 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 O TFT LCD HSYNC pin LOC = C10;
213 NET plbv46 tft cntlr O 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:

🗢 Xilinx Platform Studio - D:/TestingExa	mples/Mandelbro			
File Edit View Project Hardware Software	Device Configuration			
	0-0 🛃 🖸 📓			
Project Information Area	×			
Project Applications IP Catalog				
Description	IP Versio			
EDK Install D:\Xilinx\10.1\EDK\bw\	line same			
Analog				
Arithmetic				
⊕ Bus and Bridge ■				
Clock, Reset and Interrupt				
Communication High-Speed				
Communication Low-Speed				
DMA and Timer				
🕀 Debug				
⊕ FPGA Reconfiguration				
🚊 General Purpose IO				
Interprocessor Communication				
Memory and Memory Controller				
De PCI				
Peripheral Controller				
Processor				
· ⊕· Utility				
Project Local poores D:\1 estingExamples\Man	delbrot_Vi			
	1.00			
PLBV46_IFI_UNILR	1.00.a			
apu_mand	1.00.a			
A00 1P				
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 category and find the "Fabric Co_processor Bus" item. Add the FCB to your systemby clicking the right mouse button as shown below:

🗢 Xilinx Platform Studio - D:/TestingExamples/	Mandelbroi	_Virte	x4FX/EDK	/syster	n_1
File Edit View Project Hardware Software Device	Configuration	Debug	Simulation	Window	v H
👘 🕅 🗗 🛤 🖉 🔏 🖻 🙆 M 🛛 🖻 🗝 🗟	5 🛛 🔛	318 12	🍀 🗄 🗋	1	5
Project Information Area	×		POP	O P	
Project Applications IP Catalog			LCL	CL	
2 ⊕		104			-
Description	IP Version	1	TTT		Í
😑 🗶 EDK Install D:\Xilinx\10.1\EDK\hw\					-
🕀 Analog					+
⊕ Arithmetic					H
🖨 Bus and Bridge					-
	1.01.a			•	
	1.00.a				1
🚽 🚽 🚽 🚽 🚽 🚽 🚽 🚽 🚽 🚽 🚽 🛶	1.02.a		99	-0-	
- 🛧 PLBv46 to FSL Bridge	1.00.a				
🚽 📩 📩 📩 📩 📩 📩 📩 📩 📩	1.00.a				1
- 🚽 Instruction-Side On-Chip Memory (OCM) Bus 1	.0 2.00.Ь		(1
- 🙀 Fast Simplex Link (FSL) Bus	2.11.a		6-0	-	
🗝 📩 Fabric Co-processor Bus (FCB)			<u> </u>	1	
🛧 FCB to FSL Bridge	Add IP.				
- 🛧 Data-Side On-Chip Memory (OCM) Bus 1.0	View MP	D			Т
🚽 🛨 🛨 🚽 🚽 🚽 🚽	View IP	Modifical	tions (Chang	je Log)	
Clock, Reset and Interrupt	View PD	F Datash	neet		
Communication High-Speed		1			-
Communication Low-Speed					
DMA and Timer					

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):

POPOPF	Bus Interfaces Ports	Addresses		
BMBMB B	Name	Bus Connection	IP Type	IP Version
			ppc405_virtex4	2.01.a
	RESETPPC	ppc_reset_bus	3	
	JTAGPPC	jtagppc_0_0		
	EMACDCR	ppc405_0_EMACDCF	7	
	MFCM	ppc405_0_MFCM		
	MFCB	fcb_v10_0	2	
	ISOCM	ppc405_0_iocm	3	
	- DSOCM	ppc405_0_docm		
	IPLB1	ppc405_0_iplb1		
	DPLB1	ppc405_0_dplb1		
	IPLB0	plb	2	
	DPLB0	plb 💽	2	
	MDCR	No Connection		
			fcb_v10	1.00.a
			dsocm_v10	2.00.b
			isocm_v10	2.00.b
	🗢 plb		plb_v46	1.02.a
			plb_v46	1.02.a
			plb_v46	1.02.a
	€ → ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.b
	⊕ → ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.Ь
	🕀 🗢 DDR_SDRAM		mpmc	4.01.a
Þ-Þ-+	🕀 🧼 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		
	·æ → jtagppc_0		jtagppc_cntlr	2.01.a
**	🕀 🗢 proc_sys_reset_0		proc_sys_reset	2.00.a
ò_Ò_♦	🕀 🗢 LCD_78it_GPIO		xps_gpio	1.00.a
ò_ò_∳	⊕- ◇ LEDs_4Bit		xps_gpio	1.00.a
ò	⊕		xps_uartlite	1.00.a
	clock_generator_0		clock_generator	2.01.a

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:

Xilinx Platform Studio - D:/TestingExamples/	Mandelbrot		
File Edit View Project Hardware Software Device	Configuration	Debug Simulation Winde	
📅 🖥 🖬 🕬 🖉 💥 🖻 🛱 M 🛙 🖻 🕫	5	ka 🔝 🏀 🗄 🗋 🏓 🖡	
Project Information Area	×	POPOP	
Project Applications IP Catalog			
	IP Version	│ ┝ ╞╝╝╝╝	
Contraction Different 10 11 EDK hard			
Analag			
H Analog			
And Intege			
→ PLBV46 to PLBV46 Bridge	1.01.a		
PLBV46 to DCB Bridge	1.00.a		
Processor Local Bus (PLB) 4.6	1.02.a		
PLBv46 to FSL Bridge	1.00.a	 • • •	
	1.00.a		
	.0 2.00.Ь		
Fast Simplex Link (FSL) Bus	2.11.a		
- Fabric Co-processor Bus (FCB)	1.00.a		
	1.00.a		
Data-Side On-Chip Memory (OCM) Bus 1.0	2.00.b		
		<u> </u>	
Clock, Reset and Interrupt	Add IP		
Communication High-Speed	View MPD		
Communication Low-Speed	View IP Modifications (Change Log)		
DMA and Timer	View PDF Da	atasheet	
🕀 Debug	1		

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

🗢 Xilinx Platform Studio - D:/TestingExamp	les/Mandelbrot_Virtex4FX/EDK
File Edit View Project Hardware Software De	evice Configuration Debug Simulation
	0 😼 🖸 😣 🗄 🍀 🗎 🗋
Project Information Area	× POP
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 🛛 🚽	102-
PLBv46 to FSL Bridge	Add IP
- 🛨 Local Memory Bus (LMB) 1.0	View MPD
Instruction-Side On-Chip Memory (OI	View IP Modifications (Change Log)
Fast Simplex Link (FSL) Bus	View PDF Datasheet
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.Б
Device Control Register (DCR) Bus 2.9	1.00.a
⊕ Clock, Reset and Interrupt	

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:

POPOPP F D	Bus Interfaces Ports	Addresses		
BMBMBB B R	Name	Bus Connection	IP Type	IP Version
			ppc405_virtex4 dcr_v29 fcb_v10 dsocm_v10 isocm_v10 plb_v46 plb_v46 plb_v46 plb_v46 plb_v46 dsbram_if_cntlr	2.01.a 1.00.a 1.00.a 2.00.b 2.00.b 1.02.a 1.02.a 1.02.a 1.02.a 1.02.a 3.00.b
	⊕	Configure IP	isbram_if_cntlr	3.00.b 4.01.a 1.00.a
	 ⇒ isocm_bram ⇒ apu_mand_0 SFCB ⇒ jlagppc_0 ⇒ plbv45 dcr_bridge 	View MPD View IP Modificatio View PDF Datashee Browse HDL Source	ns (Change Log) et es	1.00.a 1.00.a 2.01.a ie 1.00.a
	MDCR	Driver: mpmc_v2_(00_a ►	
0-	⇒ srLB ⇒ ⇒ plbv46_tit_cntlr_0	Delete Instance		1.00.a
	SDCR	Filter Bus Interface	es 🕨	
	MPLB ⊕ → proc sys reset 0	Hide Selection	proc sys reset	2.00.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.



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:

Đ	Bus Interfaces	Ports	Addresses	
Na	me	701	Net	Direction
÷	External Ports			
÷	ppc405_0			
÷	→ dcr_v29_0			
÷	icb_v10_0			
÷	ppc405_0_doc	war		
÷	> ppc405_0_ioci	n		
÷	🧼 plb			
÷	> plb_v46_0			
÷	ppc405_0_dpll	57		
÷	ppc405_0_iplb	7		
÷	ppc405_0_doc	m_cnth		
÷	🗢 ррс405_0_іосі	n_cnth		
÷	DDR_SDRAM			
÷	🧼 dsocm_bram			
÷	isocm_bram			
÷	⇒apu_mand_0			
÷	⇒jtagppc_0			
ġ	⇒ plbv46_dcr_bri	dge_0		
-	PLB_dcrRst		No Connection	0
	PLB_dcrClk		No Connection	v 0
Đ	◇ plbv46_tft_cntil	<u>r</u> 0	No Connection	
Đ	<pre>> proc_sys_reset</pre>	0	New Connection	
Đ	⇒LEDs_4Bit		Make External	00.0.0.0.0.0.0.0.0.0.

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.

Filte	rs (Applied)	Add External Port
Class	Freque	All
		Default Connections
	· · · · · · · · · · · · · · · · · · ·	Connected
	V	 Unconnected
		Clocks
	· ·	Resets
		 Interrupts
		• Other
		· Inputs
		• Outputs
		 InOuts

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:

-DCR_ABus	Default Connection	V
- DCR_DBusOut	Default Connection	0
DCR_Ack	Default Connection	0
SYS_dcrClk	Default Connection	V 1
TFT_LCD_B TFT_LCD_G TFT_LCD_R TFT_LCD_DPS TFT_LCD_CLK TFT_LCD_DE TFT_LCD_VSYNC TFT_LCD_HSYNC SYS_tttClk	fpga_0_DDR_SDRAM_DDR_Cl fpga_0_DDR_SDRAM_DDR_R/ fpga_0_DDR_SDRAM_DDR_W fpga_0_RS232_Uart_TX pcore_co_clk plbv46_dcr_bridge_0_PL8_dcrC plbv46_tft_cntlr_0_TFT_LCD_Cl plbv46_tft_cntlr_0_TFT_LCD_H plbv46_tft_cntlr_0_TFT_LCD_V	

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

- MD_error	No Connection	0	
SYS_dcrClk	plbv46_dcr_bridge_0_PLB	_dcrClk 💟 I	
TFT_LCD_B	No Connection	0	[5:0]
TFT_LCD_G	No Connection	o	[5:0]
TFT_LCD_R	New Connection	o	[5:0]
TFT_LCD_DPS	Make External	0	

The resulting port connection should be like this:

MD_error	No Connection	0	
SYS_dcrClk	plbv46_dcr_bridge_0_PLB_dcrClk		
TFT_LCD_B	plbv46_tft_cntlr_0_TFT_LCD_B	0	[5:0]
TFT_LCD_G	plbv46_tft_cntlr_0_TFT_LCD_G	0	[5:0]
-TFT_LCD_R	plbv46_tft_cntlr_0_TFT_LCD_R	0	[5:0]
-TFT_LCD_DPS	No Connection	0	
TFT_LCD_CLK	plbv46_tft_cntlr_0_TFT_LCD_CLK	0	
-TFT_LCD_DE	No Connection	0	
-TFT_LCD_VSYNC	plbv46_tft_cntlr_0_TFT_LCD_VSYNC	0 💟 0	
-TFT_LCD_HSYNC	plbv46_tft_cntlr_0_TFT_LCD_HSYNC	0	

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

🖨 🗢 plbv46_ttt_cntlt_0		
MD_error	No Connection 🛛 🔽 🖸)
SYS_dcrClk	plbv46_dcr_bridge_0_PLB_dcrClk 🛛 🔛 I	
TFT_LCD_B	plbv46_tft_cntlr_0_TFT_LCD_B 🛛 💟 C)
-TFT_LCD_G	plbv46_tft_cntlr_0_TFT_LCD_G 🛛 💟 0)
-TFT_LCD_R	plbv46_tft_cntlr_0_TFT_LCD_R 🛛 💟 C)
- TFT_LCD_DPS	No Connection 🛛 🔽 🖸)
-TFT_LCD_CLK	plbv46_tft_cntlr_0_TFT_LCD_CLK 🛛 💟 C)
- TFT_LCD_DE	No Connection 🛛 🔽 🖸)
- TFT_LCD_VSYNC	plbv46_tft_cntlr_0_TFT_LCD_VSYNC 🛛 🖸)
-TFT_LCD_HSYNC	plbv46_tft_cntlr_0_TFT_LCD_HSYNC 💟 C)
SYS_tftClk	No Connection	
⊕ 🗢 proc_sys_reset_0	plbv46_tft_cntlr_0_TFT_LCD_HSYNC	
🕀 🧼 LEDs_4Bit	plbv46_tft_cntlr_0_TFT_LCD_VSYNC	
🕀 🗢 RS232_Uart	proc_clk_s	
🗄 🥯 clock_generator_0	sys_bus_reset	
	sys_clk_s	
	sys_periph_reset	
	tft_25mhz_clk	
	dcm_clk_s	
	dcm_clk_s fpga_0_RS232_Uart_RX	

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:

- co_clk	No Connection
apu_clk	No Connection 🛛 🐼
 ⇒ jiagppc_0 ⇒ plbv46_dcr_bridge_0 ⇒ plbv46_tfr_cntlr_0 ⇒ proc_sys_reset_0 ⇒ LEDs_48it 	plbv46_tft_cntlr_0_TFT_LCD_HSYNC plbv46_tft_cntlr_0_TFT_LCD_VSYNC proc_clk_s sys_bus_reset sys_clk_s
	sys_periph_reset 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:

co_clk	No Connection
apu_clk ⊕	fpga_0_DDR_SDRAM_DDR_RAS_n fpga_0_DDR_SDRAM_DDR_WE_n fpga_0_RS232_Uart_TX
	pcore_co_clk
 □ → µvc_sys_reset_v ■ → LEDs_4Bit ■ → RS232_Vart 	plbv46_dcr_bridge_0_PLB_dcrClk plbv46_tft_cntlr_0_TFT_LCD_CLK
Gock_generator_0	plbv46_tft_cntlr_0_TFT_LCD_HSYNC 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		
General Content of the second sec		
SYS_RST	No Connection	I
FCB_CLK	No Connection	<u>v</u> i
⊕	plbv46 tft ontlr 0 TFT LCD HSYNC	
🕀 🗢 ppc405_0_iocm	plbv46 tft_cntlr_0_TFT_LCD_VSYNC	-
🗄 🗢 plb	proc clk s	
⊕	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	
C1000		

And connect the SYS_RST signal of fcb_v10_0 to sys_bus_reset:

Name	Net	Direction
🕀 🥯 External Ports		
🕀 🗢 ррс405_0		
🕀 🥌 dcr_v29_0		
🖨 🥌 fcb_v10_0		
-SYS_RST	No Connection	v]I
FCB_CLK	plbv46_tft_cntlr_0_TFT_LCD_HSYNC	
⊕	plbv46 tft_cntlr_0_TFT_LCD_VSYNC	
🕀 🥌 ppc405_0_iocm	proc_clk_s	
🕀 🧼 plb	sys bus reset	
🕀 🗢 plb_v46_0	sys clk s	
⊕	sys periph reset	
🕀 🥌 ppc405_0_iplb1	tft 25mhz clk	
🗄 🧼 ppc405_0_docm_cntlr	dcm clk s	
🕀 🥌 ppc405_0_iocm_cntlr	fpga 0 RS232 Uart RX	
🗄 🥌 DDR_SDRAM	sus rst s	

•	Bus Interfaces	Ports	Addresses		
Na	ime	780		Net	Direction
÷	External Ports				
Đ	> ppc405_0				
ġ	dcr_v29_0				
Đ	<pre>> fcb_v10_0</pre>				
Đ	ppc405_0_doc	m			
۲	ppc405_0_ioci	m			
Đ	Ib 🗢 🗢				
P	> plb_v46_0				
	-Bus_Error_Del	t		No Connection	0
	-SYS_Rst			No Connection	\sim
-	PLB_CIK			sys_clk_s	\sim
	ppc405_0_dpll	67		No Connection	
Ð	ppc405_0_iplb	1		New Connection	
E	ppc405_0_doc	m_cnth		Make External	
E	ppc4U5_U_loci	m_cnth		sys_clk_s	
1				net_vcc	
1	Osocm_bram			net_gnd	
1	socm_pram			DDR_SDRAM_mpmc_clk_90_s	
	<pre>apu_mand_0</pre>				
	pagppc_0	idaa 0		rpga_U_UUR_SURAM_UUR_CAS_n	1000
Đ.		uge_v		IPGa_U_UUK_SUKAM_UUK_LE	

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:

Name	Net	Direction
🖶 🧼 External Ports		
🕀 🧼 ррс405_0		
😑 🥌 dcr_v29_0		
⊕ <> fcb_v10_0		
⊕		
⊕ <> ppc405_0_iocm		
الم 🧼 🕀		
⇒ plb_v46_0		
-Bus_Error_Det	No Connection	0
SYS_Rst	No Connection	V I
PLB_Clk	plbv46_tft_cntlr_0_TFT_LCD_HSYNC	
⊕ <> ppc405_0_dplb1	plbv46_tft_cntlr_0_TFT_LCD_VSYNC	
🕀 🧼 ppc405_0_iplb1	proc_clk_s	
🕀 🥯 ppc405_0_docm_cntlr	sys_bus_reset	
🕀 🧼 ppc405_0_iocm_cntlr	sys_clk_s	
🕀 🥌 DDR_SDRAM	sys_periph_reset	
🕀 🥯 dsocm_bram	tft_25mhz_clk	
🕀 🥯 isocm_bram	dcm_clk_s	
🕀 🧼 apu_mand_0	fpga_0_RS232_Uart_RX	
🕀 🥯 jtagppc_0	sys_rst_s	

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

•	Bus Interfaces	Ports	Addresses		
Nar	me	74	14	Net	Direc
÷.	External Ports				12964545664
	ppc405_0				
ė.	→ dcr_v29_0				
	<pre>> fcb_v10_0</pre>				
	-SYS_RST			sys_bus_reset	V
	FCB_CLK			sys_clk_s	V
	ppc405_0_doc.	m			
	ppc405_0_iocn	7			
	🧼 plb				
	> plb_v46_0				
	-Bus_Error_Det			No Connection	0
	-SYS_Rst			sys_bus_reset	
	PLB_Clk			sys_clk_s	V
	⇒ppc405_0_dplb	7			
	ppc405_0_iplb:	1			
	> ppc405_0_doc.	m_cnth			
Đ.	> ppc405_0_iocn	n_cnth			
	DDR_SDRAM				
	🧼 dsocm_bram				
	🧼 isocm_bram				
	⇒apu_mand_0				
	co_clk			pcore_co_clk	
	apu_clk			sys_clk_s	\sim
Đ	<i>⋘itagppc_0</i>				
	> plbv46_dcr_brid	dge_0			
	-PLB_dcrRst			No Connection	0
	PLB_dcrClk			plbv46_dcr_bridge_0_PLB_dcrClk	0
	plbv46_ttt_cntli	_0			
	MD_error			No Connection	0
	- SYS_dcrClk			plbv46_dcr_bridge_0_PLB_dcrClk	
	- TFT_LCD_B			plbv46_tft_cntlr_0_TFT_LCD_B	
	- TFT_LCD_G			plbv46_tft_cntlr_0_TFT_LCD_G	0
	- TFT_LCD_R			plbv46_tft_cntlr_0_TFT_LCD_R	
	- TFT_LCD_DP	5		No Connection	0
	- TFT_LCD_CLK	<		plbv46_tft_cntlr_0_TFT_LCD_CLK	0
	- TFT_LCD_DE			No Connection	0
	- TFT_LCD_VS	rnc		plbv46_tft_cntlr_0_TFT_LCD_VSYNC	0
	- TFT_LCD_HS	YNC		plbv46_tft_cntlr_0_TFT_LCD_HSYNC	0
	-SYS_tftClk			tft_25mhz_clk	\sim
Đ	proc_sys_reset	_0			
Đ	◆LEDs_4Bit				
Ð	RS232_Uart				
Đ.	clock generato	v O			

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.

	Bus Interfaces Po	orts Addresses		
BMBMBB	Name	Bus Connection	IP Type	IP Version
> > = = = = = >	⊕ ppc405_0 → dcr_v29_0	Configure IP	405_virtex4 v29 v10	2.01.a 1.00.a 1.00.a
		View MPD		2.00.b 2.00.b
	◇ plb ◇ plb_v46_0	View IP Modifications (Change Lo View PDF Datasheet	g) v46 v46	1.02.a 1.02.a
		Browse HDL Sources	v46 	1.02.a 1.02.a
		OS: standalone_v2_00_a	am_it_cntlr	3.00.b 3.00.b
		Delete Instance	block	4.01.a 1.00.a
		Hide Selection	mand	1.00.a

Switch to the APU Tab and change the APU Controller Configuration Register Initial Value to 0b00000000000001 as shown below:

PowerPC	Bus Settings	APU	Buses	HDL Toggle 🔀 Datasheet 🧭 Restore
APU Fea	ature			
APU C	ontroller Configura	ation Regis	ter Initial Value	0500000000000001
UDI Co	onfiguration Regis	ter 1 Initial	Value	100110000011
UDI Co	onfiguration Regis	ter 2 Initial	Value	100110000011
UDI Ca	onfiguration Regis	ter 3 Initial	Value	100111000011

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.

Name	Bus Connection	IP Type	IP Version
⊕ <>ppc405_0		ppc405_virtex4	2.01.a
🗢 dcr_v29_0		dor_v29	1.00.a
		fcb_v10	1.00.a
		dsocm_v10	2.00.Ь
		isocm_v10	2.00.Ь
🧼 plb		plb_v46	1.02.a
plb_v46_0		plb_v46	1.02.a
		plb_v46	1.02.a
ppc405_0_iplb1		plb_v46	1.02.a
🕀 🧼 ppc405_0_docm_cntlr		dsbram_if_cntlr	3.00.Ь
🕀 🧼 ppc405_0_iocm_cntlr		isbram_if_cntlr	3.00.Ь
🕀 🧼 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
🕀 🧼 jtagppc_0		jtagppc_cntlr	2.01.a
🕞 🗢 plbv46_dcr_bridge_0		plbv46_dcr_bridge	1.00.a
🕀 🥌 plbv46_tit_cntli_0	10 with 4000	olbv46_tft_cntlr	1.00.a
⊕ <pre>> proc_sys_reset_0</pre>	Configure IP	ic_sys_reset	2.00.a
⊕. <>LEDs_48it	View MPD	gpio	1.00.a
🕀 🗢 RS232_Uart	Distance LIDI Common	_uartlite	1.00.a
clock_generator_0	Browse HDL Sources	ck_generator	2.01.a
	Driver: generic_v1_00_a	•	
	Delete Instance		

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

Buses	HDL T	loggle 🔀 Datasheet 📿 Restr
C_DCR_BASEADDR		ØÐ 0 0 0 0 0 0 0 0 0 0 0
C_DCR_HIGHADDR		0Ъ000000001
C_DEFAULT_TFT_BASE_ADDR		060001000000
C_DPS_INIT	HDL Param Name : <i>C_DEFAULT_TFT_BASE_ADDR</i>	1
C_ON_INIT	Value : <i>051111000000</i>	1
C_BASEADDR		0xc9800000
C HIGHADDB		0xc980ffff

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.

🕌 🛛 Bus Interfaces	Ports Addresses					
Instance	Name 🔺	Base Address	High Address	Size	Bus Interface(s)	Bus Connection
ppc405_0_docm_cni	thC_BASEADDR	Оха6с08000	0xa6c0bfff	16K	SOCM	ppc405_0_docm
ppc405_0_iocm_cntl	r C_BASEADDR	Oxffffc000	Oxfffffff	16K	ISOCM	ppc405_0_iocm
plbv46_dcr_bridge_0	C_BASEADDR			U	SPLB	plb
plbv46_tft_cntlr_0	C_BASEADDR			U	SPLB	plb
LEDs_4Bit	C_BASEADDR	0x81400000	0x8140ffff	64K	SPLB	plb
RS232_Uart	C_BASEADDR	0x84000000	0x8400ffff	64K	SPLB	plb
plbv46_tft_cntlr_0	C_DCR_BASEADDR	06000000000	060000000000	2	SDCR	dcr_v29_0
ррс405_0	C_IDCR_BASEADDR	06010000000	ОЬО111111111	256	Not Connected	
DDR_SDRAM	C_MPMC_BASEADDR	0x0000000	0x03FFFFFF	64M	SPLB0:SPLB1:SPLB2	2
				1M 2M 4M 8M 16M 32M 64M 128M 256M		

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

🚯 🛛 Bus Interfaces	s Ports	Addresses					🚟 Generate Addresses
Instance	Name	*	Base Address	High Address	Size	Bus Interface(s)	B Generate Addres
ppc405_0_docm_cr	htlr C_BASEA	ADDR	Оха6с08000	0xa6c0bfff	16K	SOCM	ppc4U5_U_docm
ppc405_0_iocm_cnl	tlr C_BASEA	ADDR	Oxffffc000	Oxfffffff	16K	ISOCM	ppc405_0_iocm
plbv46_dcr_bridge_l	0 C_BASEA	ADDR	0x48a00000	0x48a00fff	4K	SPLB	plb
plbv46_tft_cntlr_0	C_BASEA	ADDR	0xc:9800000	0xc980ffff	64K	SPLB	plb
LEDs_4Bit	C_BASEA	ADDR	0x81400000	0x8140ffff	64K	SPLB	plb
RS232_Uart	C_BASEA	ADDR	0x84000000	0x8400ffff	64K	SPLB	plb
plbv46_tft_cntlr_0	C_DCR_E	BASEADDR	06000000000	060000000000000000000000000000000000000	2	SDCR	dcr_v29_0
ppc405_0	C_IDCR_	BASEADDR	06010000000	ОЬО111111111	256	Not Connected	
DDR_SDRAM	C_MPMC	_BASEADDR	0x0000000	0x0FFFFFFF	256M	SPLB0:SPLB1:SPLB	2

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 building 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.

🗢 Xilinx Platform Stud	lio - D:\TestingExamples\Mandel	brot_Virtex4FX\EDK
File Edit View Project	Hardware Software Device Configura	ation Debug Simulation
- 🗗 🗑 📑 - 100 @ 🕽	Btg Generate Netlist	11 848 🔝 🎨 11 🗋
Project Information Area	Generate Bitstream	OPOPP
Project Applications	🌺 Create or Import Peripheral	CLCLL
₽	Configure Coprocessor	
Description	Check and View Core Licenses	
 EDK Install D:\Xilinx ⊕ Analog ⊕ Arithmetic 	👺 Clean Netlist 🕦 Clean Bits	
Bus and Bridge	🛱 Clean Hardware	

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 <u>Xilinx Answers Database</u> 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:

🗢 Xilinx Platform Studio - D:\TestingE	kamplesWandelbrot_Virtex4FX	\EDK_4\system.xmp - [System	
File Edit View Project Hardware Softwar	re Device Configuration Debug Simu	lation Window Help	
8 🗗 🗗 🕬 🕬 🗶 🖻 🕅 8 8	3 60 😼 🖸 😣 🗄 👪 🗎 🇞	🗋 🖻 🖥 🤤 🛛 🖉 🚾 🛓	
Project Information Area	🗢 Add Software Application P	roject	×
Software Projects	Project Name mand		
Add Software Application Project	Note: Project Name cannot have space	es.	
Default: ppc405_0_bootloop	Processor	ppc405_0	
Project: TestApp_Peripheral Processor: ppc405_0 Executable: D:\TestingExamples\Mandel Constants	Choose an ELF file.		
⊕-Sources		Browse	
. Headers	The ELF file is assumed to be gener Default ELF name is <sw nar<="" project="" td=""><td>ated outside XPS me>/executable.elf</td><td>30</td></sw>	ated outside XPS me>/executable.elf	30
		OK Cancel	

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 Source/	leader File to Add to	Project			? 🗙
Look in:	🗁 code		•	← 🗈 😁 💷 →	
My Recent Documents Desktop My Documents My Computer	 bootload_basicgraphic 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 	:S.C			
My Network Places	File name: "bool Files of type: C So	tload_basicgraphics.c" " urces (*.c)	'co_init.	c" "gpio_l 💌	Open Cancel

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:

Select Source/H	leader File to	Add to Project			? 🔀
Look in:	Code		•	+ 🗈 💣 💷 +	
My Recent Documents Desktop My Documents My Computer	bmp.h bootload_bas gpio_lcd_led. InitializeDispla mand.h mand_accel_s mand_sw_onl stop_watch.h xtft_main.h	iicgraphics.h h ay.h sw.h ly.h			
My Network Places	File name:	"bmp.h" "bootload_ba	sicgraphics.h'	"gpio_lcd 💌 🛛 [Open
	Files of type:	C Headers (*.h)		•	Cancel

Setting Compiler Options

Now you will need to set a few compiler options for the project. To set the compiler options, doubleclick 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:

Invironment	Debug and Optimization	Paths and Options	
-Application M	ode ole 🔘 XmdStub (xmdstu	ub_peripheral : not assi	gned)
-Output ELF fil	e		
xamples\Ma	ndelbrot_Virtex4FX\EDK_4	\mand\executable.elf	Browse
- Vuse Del Program St	fault Linker Script		Browse
- Stack Size	0x4000		
Hean 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

1.3.10 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
-W1,-defsym -W1,_START_ADDR=0x02000000 -W1,-defsym -W1,_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:

File E	dit View Proje	ct Hardware	Software	Dev	ice Configuration	Debug	Sim
i 🗗 🕅	📑 🗄 🖍 🖓	XAG	00 8 🖻	BRAM	Update Bitstrean	า	82
Project Information Area			盎	Download Bitstre	am	P	
Project	Applications	IP Catalog	1	=î	Program Flash M	emory	

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

• Configuration	Debug	Simulation Window Help
🚽 🔽 🔜 🗄	De	bug Configuration
P	🌋 XM	1D Debug Options
<u>i</u> č	🌋 La	unch XMD
BN	💥 La	unch Software Debugger

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.

👓 D:\Xi	linx\10.1\EDK\bir	1\nt\xbash.exe		- 🗆 ×
Device 1 2 3 4	ID Code Øa001093 f5059093 21e58093 59608093	IR Length 8 16 10 8	Part Name System_ACE XCF32P XC4UFX12 xc95144x1	
PowerPC	405 Processor	Configuration		
Version User ID No of P No of R No of W ISOCM User De	C Breakpoints ead Addr/Data rite Addr/Data fined Address I-Cache (Data D-Cache (TAG D-Cache (TAG DCR. TLB.	Watchpoints. a Watchpoints. Map to access a)0x700)0x780 a)0x780)0x780 0x780	.0×20011470 .0×00000000 .4 .1 .1 .0×ffffc000 - 0×ffffffff Special PowerPC Features using XMD: 00000 - 0×70003fff 04000 - 0×70007fff 04000 - 0×78003fff 04000 - 0×78004fff 04000 - 0×78004fff 04000 - 0×70007fff	
Connect Startin XMD% _	ed to "ppc" ta g GDB server b	arget. id = 0 for "ppc" targe	t (id = 0) at TCP port no 1234	-

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	A
<pre>XHU2 dow mand/executable.elf System Keset DONE Downloading Program mand/executable.elf section, .text: 0x0200000-0x02000bab section, .init: 0x0200dba8-0x0200dbab section, .bot0: 0xffffffdc-0x0200dbcb section, .bot0: 0xffffffdc-0xfffffff section, .rodata: 0x0200dbd0-0x0200e52e section, .sdata2: 0x0200e530-0x0200e52f section, .sbs2: 0x0200e530-0x0200e52f section, .data: 0x0200e530-0x0200e52f section, .data: 0x0200e530-0x0200e52f section, .got1: 0x0200f640-0x0200f63f section, .got2: 0x0200f640-0x0200f65b section, .dtors: 0x0200f66c-0x0200f66b section, .fixup: 0x0200f66c-0x0200f66b section, .fixup: 0x0200f66c-0x0200f66b section, .jcr: 0x0200f66c-0x0200f66b section, .jcr: 0x0200f66c-0x0200f6bf section, .jcr: 0x0200f66c-0x0200f6c3 section, .got2: 0x0200f66c-0x0200f6bf section, .jcr: 0x0200f66c-0x0200f6c5 section, .sdata: 0x0200f6c4-0x0200f6c5 section, .sdata: 0x0200f6c6-0x0200f6c5 section, .sdata: 0x0200f6c6-0x0200f6c5 section, .sdata: 0x0200f6c6-0x0200f6c5 section, .sdata: 0x0200f6c6-0x0200f6c5 section, .sdata: 0x0200f6c4-0x0200f6c5 section, .sdata: 0x0200f6c4-0x0200f6c5 section, .sdata: 0x0200f6c4-0x0200f6c5 section, .sbss: 0x0200f6c6-0x0200f6c5 section, .sbss: 0x0200f6c6-0x0200f6c5</pre>	
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	
	-

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)

