1 Tutorial 2: Complex FIR on EDK 10.1i



Overview

This detailed tutorial will demonstrate how to use **Impulse C** to create, compile and optimize a digital signal processing (**DSP**) example for the **MicroBlaze** platform. We will also show how to make use of the **Fast Simplex Link (FSL)** bus provided in the **MicroBlaze** platform.

The goal of this application will be to compile the algorithm (a **Complex FIR Filter** function) on hardware on the FPGA. The **MicroBlaze** 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 Virtex-4 ML401 Evaluation Platform. The board features is a Virtex-4 FPGA with a MicroBlaze soft processor. 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**. 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 **MicroBlaze** 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 1

1.1 Loading the Complex FIR Application

Complex FIR Filter Tutorial for MicroBlaze, Step 1

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

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

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



Files included in the **Complex FIR Filter** 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.

Header files ComplexFilter.h and Filter.h - function prototypes and definitions.

See Also

Understanding the Complex FIR Application

1.2 Understanding the Complex FIR Application

Complex FIR Filter Tutorial for MicroBlaze, Step 2

Before compiling the **Complex FIR Filter** 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 Filter 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 **MicroBlaze** 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);
             = co_process_create ("cpu_proc",
                                                (co_function)
  cpu_proc
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, "PEO");
}
```

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 Filter** coefficients and also a group of input data being processed.

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

See Also

4

Compiling the Application for Simulation

1.3 Compiling the Application for Simulation

Complex FIR Filter Tutorial for MicroBlaze, 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 Software Simulation Executable (or click the Build Software Simulation Executable button) to build the FIR_Accelerator.exe executable. A command window will open, displaying the compile and link messages as shown below:

Build	4 ×
====================================	
Build Image: Find in Files Image: System	2

 You now have a Windows executable representing the Complex FIR Filter 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:

C:\WINDOWS\system32\cmd.exe	- 🗆 🗙
"C:\MeiWork\Microblaze7.00\ComplexFIR\FIR_Accelerator.exe"	
Complete Filtering a Slot	
Begin Filtering a Slot Complete Filtering a Slot	
COMPLETE APPLICATION	
rress Enter to contline	
	-
4	▶ /h

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:

```
#if defined(MICROBLAZE)
   xil_printf ("COMPLETE APPLICATION\r\n");
   return 0;
#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 **MicroBlaze** processor, or to the host development system for simulation purposes.

See Also

Building the Application for the Target Platform

1.4 Building the Application for the Target Platform

Complex FIR Filter Tutorial for MicroBlaze, 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, open the Generate tab of the Options dialog as shown below:

Build Simulate Generate System Registration Platform Support Package: Xilinx MicroBlaze FSL (VHDL) CoBuilder Optimization Options	impulse
 Enable constant propagation Scalarize array variables Relocate loop invariant expressions Additional 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:	Software build directory: sw Hardware export directory: EDK Software export directory: EDK
 Include floating point library Use higher latency, faster clock operators Allow double-precision types and operators 	

Specify Xilinx MicroBlaze FSL (VHDL). Also specify hw and sw for the hardware and software directories as shown, and specify EDK for the hardware and software export directories. Also ensure that the Generate dual clocks option is checked.

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 select the **Generate HDL** toolbar button as shown below:

<u>ج</u>	 ► ttt	ଲେଆ ଲିଲା	× 4	•	Ŧ
	G	ienerate H	DL		-
hw.c					

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



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

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

See Also

Exporting Files from CoDeveloper

1.5 Exporting Files from CoDeveloper

Complex FIR Filter Tutorial for MicroBlaze, Step 5

Recall that in <u>Step 4</u> 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 MicroBlaze FSL**) 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

8



Export the Software Files



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

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

See Also

Creating the Platform Using the Xilinx Tools

1.6 Creating a Platform Using Xilinx Tools

Complex FIR Filter Tutorial for MicroBlaze, Step 6

As you learned in the previous Hello World tutorial, **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 **MicroBlaze** processor). This section will walk you through the file export/import process for this example, using the EDK System Builder (Platform Studio) project.

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) and creating a new project. The Xilinx Platform Studio dialog appears as shown below:

SSB 💿 Ba	se System Builder wiz	ard (recommended)		
📩 🔿 Bla	ank XPS project			
	en a recent project			
Browse for Mo	ore 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 **Open** to create a project file called **system.xmp** (you can specify a different project name if desired):

Platform Studie	o Project				? 🔀
Save in:	EDK		•	+ 🗈 💣 📰 +	
My Recent Documents					
My Documents					
My Computer					
Mu Network	File name:	sustem ymp		.	Save
Places	Save as type:	Platform Studio Projec	t (*.xmp)		Cancel

Now click **OK** in the **Create New XPS Project Using BSB Wizard** dialog. The **Base System Builder** - **Welcome** page will appear. 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-4 ML401 Evaluation Platform Board Revision: 1

Select board I would like	to create a system for the following development board	
Board vendor:	Xilinx	
Board name:	Virtex 4 ML401 Evaluation Platform	V
Board revision:	1	~
Note: Visit the	endor website for additional board support materials.	
Vendor's Webs	e <u>Contact Info</u>	
Vendor's Webs Download Third	e <u>Contact Info</u> Party Board Definition Files	
Vendor's Webs Download Third O I would like	e <u>Contact Info</u> Party Board Definition Files to create a system for a custom board	
<u>Vendor's Webs</u> <u>Download Thirc</u> O I would like Board descriptic	e <u>Contact Info</u> Party Board Definition Files to create a system for a custom board	

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

Pro	t the processor you would like to use in this design:
~	uter minera
0	MICroblaze
	PowerPC
	Not supported by this device

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 **MicroBlaze** processor and create the necessary I/O interfaces for our sample application.

See Also

Configuring the New Platform

1.7 Configuring the New Platform

Complex FIR Filter Tutorial for MicroBlaze, Step 7

Now that you have created a basic **MicroBlaze** 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: 100 MHz Processor-Bus Clock Frequency: 100 MHz

Processor Configuration On-chip H/W debug module (default setting) Local memory - Data and Instruction : 8 KB Cache setup: Enable

AicroBlaze	
System wide settings	
Reference clock frequency:	-rocessor-Bus clock frequency:
100.00 MHz	100.00 MHz
Ensure that your board is configur	ed for the specifed frequency.
Reset polarity: Active LOW	
XMD with S/W debug stub No debug	
	Data and Instruction:
MicroBlaze	(Use BRAM) 8 KB
Cache setup	

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 Peripheral: XPS UARTLITE Baudrate: 9600 Data Bits: 8 Parity: NONE Use Interrupt: disabled

Base System Builder - Configure IO Interfaces (1 of 3)	×
The following external memory and IO devices were found on your board:	
Xilinx Virtex 4 ML401 Evaluation Platform Revision 1	
Please select the IO devices which you would like to use:	
ID devices	
RS232_Uart	Data Sheet
Peripheral: XPS UARTLITE	Data Sheet
Baudrate (bits	
per seconds): 9600	
Data bits: 8	
Parity: NONE	
Use interrupt	
LEDs_4Bit	
	Data Sheet
LEDs_Positions	Data Sheet
Push_Buttons_Position	Data Sheet
More Info Kext >	Cancel

Click **Next** to continue. Disable all the I/O interfaces except the **DDR2_SDRAM**:

I/O Device: DDR_SDRAM Peripheral: MPMC

🗢 Base System Builder - Configure	e IO Interfaces (2 of 3)	
The following external memory and IO devic Xilinx Virtex 4 ML401 Evaluation Platform Re Please select the IO devices which you wor	es were found on your board: evision 1 uld like to use:	
-10 devices		
DIP_Switches_8Bit		Data Sheet
		Data Sheet
		Data Sheet
DDR_SDRAM		
Peripheral: MPMC		Data Sheet
Ethernet_MAC		Data Sheet
		Note
More Info	K Back Nex	kt > Cancel

Disable all the I/O interfaces on this page:

🗢 Base System Builder - Configure 10 Interfaces (3 of 3	n) 🔀
The following external memory and IO devices were found on your boar Xilinx Virtex 4 ML401 Evaluation Platform Revision 1 Please select the IO devices which you would like to use: IO devices	d:
Soft_TEMAC	Data Sheet
SRAM	Data Sheet
FLASH	Data Sheet

Click **Next**. In the **Add Internal Peripherals** page, click the **Add Peripheral** and select the **XPS_TIMER** peripheral as shown below:

🕏 Base System		×
Add other periph "Add Peripheral"	erals that do not interact with off-chip components. Use the button to select from the list of available peripherals.	
lf you do not wisl	h to add any non-IO peripherals, click the "Next" button.	Add Peripheral
- Peripherals	🗢 Add Peripheral 🔹 🔀	
	Select the peripheral you want to add:	
	XPS TIMER	
	OK Cancel	

Choose to use only one timer, and no interrupt.

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
<pre>xps_timer_1 Peripheral: XPS TIMER Counter bit width: 32 Timer mode Timer mode Two timers are present One timer is present Use interrupt Use interrupt</pre>	Remove Data Sheet

Choose the cache settings as follows:



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

	BS232 Llart
51 D II 4.	
STDOUT:	RS232_Uart
Boot Memory:	imb ontr
ample applica	
ample applica	ation selection uple C application that you would like to have generated. Each application will
ample applica Select the sam nclude a linke	ation selection application that you would like to have generated. Each application will r script.
ample applica Select the sam nclude a linker	ation selection apple C application that you would like to have generated. Each application will r script.

Click **Next** to reveal the **MemoryTest** software settings as shown below. All the program sections are on the onchip memory.

your memory devices. MemoryTest Select the memory devices which will be used to hold the following program sections: Instruction:	memory devices. oryTest ct the memory devices which will be used to hold the following program sections: uction: ilmb_cntlr dlmb_cntlr	he simple Mem	ory Test application will illustrate system ali	veness and perform a basic read/write tes
MemoryTest Select the memory devices which will be used to hold the following program sections:	oryTest ct the memory devices which will be used to hold the following program sections: uction: ilmb_cntlr dlmb_cntlr	your memory	devices.	
Select the memory devices which will be used to hold the following program sections:	ct the memory devices which will be used to hold the following program sections: uction: ilmb_cntlr	MemoryTest		
Instruction: limb.onth	action: ilmb_cntlr 💽			
Instruction: import	uction: limb_cntlr dlmb_cntlr	Select the me	mory devices which will be used to hold the	e following program sections:
	: dimb_cntir 💽	Select the me	mory devices which will be used to hold the	e following program sections:
Data: dlmb_cntlr 🔤		Select the me Instruction:	mory devices which will be used to hold the ilmb_cnttr	e following program sections:

Click **Next** to view the **Peripheral Test** software settings. Make sure the **Instruction**, **Data** and **Stack/Heap** sections are mapped to the external **DDR_SDRAM** as shown below:

The Peripheral S a selftest functio	elftest application includes a simple self test fo n exists in the driver the peripheral).	or each periperhal in your system (if such
PeripheralTest		
Select the me	mory devices which will be used to hold the fo	llowing program sections:
Instruction:	DDR_SDRAM	
Instruction: Data:	DDR_SDRAM DDR_SDRAM	

Click Next to continue.

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



Click Generate to generate the system and project files, then Finish to close the wizard.

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

Tilinx Platform Studio - D:/TestingExamples/C	omplexFIR_ML401/EDK/sy	stem.xmp - [System Assembly View1]		
File Edit View Project Hardware Software Device C	onfiguration Debug Simulation	Window Help		
1 🗗 🕅 🗗 1 🛤 🕬 🗶 🖻 🖨 🕺 1 🗟 🕫 🚭	🗩 🛛 🗄 👪 🔛 🏀 🗄 🗋	😢 📾 😓 🛛 🗢 🗠 😒 🕼 😫	I 🛛 💥 I 🖻 🕅	ε ε β
Project Information Area 🗙 🗙	PLL	Bus Interfaces Ports Addresses		
Project Applications IP Catalog	B B B	Name Bus Connection	IP Type	IP Version
Platform		⊕ microblaze_0	microblaze	7.10.d
😑 Project Files		🧼 dlmb	Imb_v10	1.00.a
- MHS File: system.mhs		ilmb	Imb_v10	1.00.a
MSS File: system.mss		🕀 🗢 mb_plb	plb_v46	1.03.a
	0 60	😟 🧼 🗢 dlmb_cntlr	Imb_bram_if_cntlr	2.10.a
- iMPACT Command File: etc/download.cmd	K	🕀 🗢 ilmb_cntlr	Imb_bram_if_cntlr	2.10.a
Implementation Options File: etc/fast_runtime.opt		🗄 🗢 DDR_SDRAM	mpmc	4.03.a
Bitgen Options File: etc/bitgen.ut		🕀 🥌 Imb_bram	bram_block	1.00.a
Project Options	K	🕀 🥌 debug_module	mdm	1.00,d
Device: xc4vlx25ff668-10		🕀 🥌 xps_timer_1	xps_timer	1.00,a
- Netlist: TopLevel		🕀 🥌 RS232_Uart	xps_uartlite	1.00.a
Implementation: XPS (Xflow)		clock_generator_0	clock_generator	2.01.a
HDL: VHDL Sim Model: BEHAVIORAL		→	proc_sys_reset	2.00.a

See Also

Importing the Generated Hardware

1.8 Importing the Generated Hardware

Complex FIR Filter Tutorial for MicroBlaze, 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.

Adding the ComplexFIR Hardware IP Core

Next, add the module representing the **ComplexFIR Filter** hardware process to your development system. Select the **Project Local pcores** -> **USER** in the **IP Catalog** tab on the left. Right-click **fsl_filt** and select **Add IP** as shown.

Villas Distance Studie Duffertion Fr	In Item In Item
Silin File View Decisity Mandaura Software	amplearcomplexi
File Edic view Project Hardware Software	: Device Conriguratio
📅 🗗 🖬 🛤 🖉 💥 🖻 🕼 II 🖻	00 😼 🖸 🚺
Project Information Area	ХР
Project Applications IP Catalog	L
Description IP Version	
😑 🗶 EDK Install D: Wilinx\10.1\E	
🕀 Analog	
🕀 Bus and Bridge	
Clock, Reset and Interrupt	K
🕀 Communication High-Speed	
😥 Communication Low-Speed	
😟 DMA and Timer	← ∳───
🕀 Debug	
🕀 General Purpose IO	
Interprocessor Communication ■	
Memory and Memory Controller	
Peripheral Controller	
🖬 Utility	
Project Local poores D:\Testing	
G-USER	
	Add IP
	View MPD
L	

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

Adding FSL Busses

Next you will need to set some parameters related to this hardware process, setting up the communication with the FSL bus. In the IP Catalog tab, select the Fast Simplex Link (FSL) Bus IP core. Right-click it and select Add IP as shown:

Project Information Area		×
Project Applications IP Catalog		
1 <mark>2</mark> 📀		
Description	IP Version	IP Type
🖨 🗶 EDK Install D:\Xilinx\10.1\EDK\hw\		
⊕ Analog		
🕀 Arithmetic		
😑 Bus and Bridge		
	1.01.a	plbv46_plbv46_brid
- 🛧 PLBV46 to DCR Bridge	1.00.a	plbv46_dcr_bridge
🚽 🚽 🕂 🕂 🕂 🕂 🕂 🕂 🕂	1.03.a	plb_v46
- 🛨 PLBv46 to FSL Bridge	1.00.a	plb2fsl_bridge
🚽 🛨 📩 🗠 🛶 🗠 🛶	1.00.a	lmb_v10
🚽 📩 🚽 📩 🚽 🕂 🔶	(ОСМ) 2.00.Ь	isocm_v10
	211 a	Fell v20
🚽 🛧 Fabric Co-processor Bus (FCB)	Add IP	
- 🛧 FCB to FSL Bridge	View MPD	
	View IP Modifications	(Change Log)
🚽 🚽 Device Control Register (DCR)	View PDF Datasheet	(
Elock, Reset and Interrupt		

This will need to be done two times, because we will need two **Fast Simplex Links** to connect the **MicroBlaze** processor and **fsl_filt** core together. When you have added two of the **FSL** cores, your project should look like this:

P L F F L	Bus Interfaces	Ports	Addresses		
B BLL B	Name	Bus	Connection	IP Type	IP Version
	🕀 🥯 microblaze_0			microblaze	7.10.d
				fsl_v20	2.11.a
				fsl_v20	2.11.a
	🧼 dlmb			Imb_v10	1.00.a
	🧼 ilmb 👘			lmb_v10	1.00.a
	🕀 🥯 mb_plb			plb_v46	1.03.a
0 6	🕀 🥯 dimb_cntir			Imb_bram_if_cntlr	2.10.a
C	🕀 🥯 ilmb_cnth			Imb_bram_if_cntlr	2.10.a
	🕀 🧼 DDR_SDRAM			mpmc	4.03.a
	🕀 🥯 lmb_bram			bram_block	1.00.a
<u> </u>	🕀 🥯 fsl_fill_0			fsl_filt	1.00.a
	🕀 🤝 debu <u>q_</u> module			mdm	1.00.d
•	⊕			xps_timer	1.00.a
.	⊕ <> RS232_Uart			xps_uartlite	1.00.a
		0		clock_generator	2.01.a
		0		proc_sys_reset	2.00.a

Making FSL Connections

The **microblaze_0** module needs to be configured in order to link to two **FSL** links. Right-click on **microblaze_0** and select **Configure IP** as shown:



Go to the Bus Interfaces tab and change Number of FSL Links to 2 as shown:

nicroblaze_0 ; microblaze_v7_10_a	
MU Debug Interrupt and Reset PVR Buses > HDL To	ggle 🏂 Datasheet 🏹 Restore
1998, 1 	
Select Processor Local Bus (PLB) interface	
Number of FSL Links	2

Click OK. Now we just need to connect the microblaze_0 to the fsl_filt_0 with the two new FSL links.

The following connections should be made:

microblaze_0 MFSL0 connects to fsl_v20_0, and then to fsl_filt_0 SFSL0. microblaze_0 SFSL1 connects to fsl_v20_1, and then to fsl_filt_0 MFSL1.

Expand the **microblaze_0** and the **fsl_filt_0** modules. Make connections by clicking the boxes as indicated in the two red circles shown below:



Connecting Clock and Reset Ports

Next, you need to configure the clock and reset signals for each FSL IP Core. Click on the **Ports** filter in the **System Assembly View** and expand fsl_v20_0 and fsl_v20_1 modules. For each FSL bus, set FSL_Clk to sys_clk_s and set SYS_Rst to sys_bus_reset as shown:

)		1000000000				
Nar	me		Net		Direction	
•	External Ports					
	microblaze_0					
.	>fsl_v20_0					
	- FSL_Control_I	RQ	NoC	Connection	0	
	- FSL_Has_Dat	а	NoC	onnection	0	
	- FSL_Full		NoC	Connection	0	
	- FSL_S_Clk		NoC	onnection		
	-FSL_M_Clk		NoC	onnection		
			sys_l	ous_reset		
	FSL_Clk		sys_(olk_s		
9.	✓ fsl_v20_1					
	-FSL_Control_I	RQ	NoC	ionnection	0	
	FSL_Has_Dat	а	No C	onnection	0	
	-FSL_Full		NoC	ionnection	0	
	FSL_S_Clk		NoC	ionnection		
	-FSL_M_Clk		NoC	ionnection		
			sys_l	ous_reset		
	-FSL_Clk		No C	Connection	V	
🕀 🗢 dimb			fpga	fpga_0_DDR_SDRAM_DDR_RAS_		
🕀 🗢 ilmb			fpga	fpga_0_DDR_SDRAM_DDR_WE_r		
🕀 🗢 mb_plb			fpga	_0_RS232_Uart_TX		
Đ-	🗢 dlimb_cntlr		mb_	mb_reset		
	Imb_cntlr 🧼		sys_	bus_reset		
Đ.	DDR_SDRAM		sys_	clk_s		
•	✓ Imb_bram		sys_	periph_reset	0.0.0.0.0.2	
Đ.	✓ fsl_filt_0		dem	_clk_s		
	debu <u>q</u> module		fpga	_0_RS232_Uart_RX		
•	<pre>>xps_timer_1</pre>		sys_	rst_s		
	RS232_Uart		191			
	clock_generate	N O V				

Configuring the Clock

The **ComplexFIR** hardware requires a different clock source. For this purpose, we configure the **clock_generator_0** by selecting the **Configure IP** as shown below:

±			
∎			
Ð- 😎 clock_generator_0 Ð- 🥯 proc_sys_reset_0	Configure IP		
⊎ proc_sys_rese[_U	View MPD View IP Modifications (Change L View PDF Datasheet Browse HDL Sources	og)	
	Driver: generic_v1_00_a	्।	
	Delete Instance		
	Filter Ports		
	Hide Selection		

The Clock Generator dialog appears. We add a new clock output **CLKOUT3** named **pcore_co_clk**. The frequency is set to be **50,000,000 Hz**, which is half of the **100,000,000 Hz** system bus frequency. Make sure the **Buffered** value is **TRUE**.

lock Ger	nerator			
e clock g ource to i	enerator module meet all your sys	can generate required output clocks from given tem wide clocking needs. This tool will help you o	input reference/feedback clock(s) based on your requir configure the clock generator module and instantiate or	ements. It serves as a central clocking update it in your system.
lasic	Ports Overview		HDL To	ggle 🛛 🎏 Datasheet 🛛 🏹 Restor
Step 1: Sj	pecify input cloa	k details		
Step 2: Sj	pecify the outpu	t clock requirements		
Please hig	ghlight a clock p	ort in the list below and configure it on the right si	ide.	
Derte	24 - 24 - 4	Connected to	Port: CLKOUT3	
	out & Feedback 	dom_cik_s	Connected to: pcore_co_c1k	
⊜ .0ι	LLL DIN LIPUTS	sys_clk_s	Required frequency (Hz):	50,000,000
		clk_200mhz_s pcore_co_clk	Required phase shift:	0
	CLKOUT4 CLKOUT5		Grouping information:	NONE
			Buffered:	TRUE

Click OK to exit the Clock Generator dialog.

Select the **Ports** filter in the **System Assembly View** and expand **fsl_filt_0**. This should reveal ports **co_clk** and **FSL_Rst**. The **co_clk** has to be connected to the **pcore_co_clk** clock that we configured in the previous steps. The **FSL_Rst** should be tied to **sys_bus_reset**.

🕀 🗢 DDR_SDRAM	
🕀 🧼 Imb_bram	
🖨 🗢 fs <u>l_</u> fill_0	
- co_clk	pcore_co_clk 🛛 🐼 I
-FSL_Rst	No Connection 🛛 🔽
 → debug_module → xps_timer_1 → RS232_Uart → clock_generator_0 	fpga_0_DDR_SDRAM_DDR_WE_r fpga_0_RS232_Uart_TX mb_reset pcore_co_clk
	sys_bus_reset sys_clk_s sys_periph_reset dcm_clk_s fpga_0_RS232_Uart_RX sys_rst_s

Note: if **co_clk** is missing from the **fsl_filt_0** section, then will need to return to <u>step 4</u> of this tutorial and specify the **Dual Clock** option in the **CoDeveloper Generate Options** page.

Specify 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 selecting the **Addresses** tab and clicking on the **Generate Addresses** button. The addresses will be assigned for you automatically:

😽 🛛 Bus Interfa	ces Ports	Addresses					🚟 Generate A	Addresses
Instance	Name	*	Base Address	High Address	Size	Bus Interface(s)	Bus Connection	Lock
dimb_cntir	C_BASEA	DDR	0x00000000	0x00001fff	8K	SLMB	dlmb	
ilmb_cntlr	C_BASEA	DDR	0x00000000	0x00001fff	8K	SLMB	ilmb	
debug_module	C_BASEA	DDR	0x84400000	0x8440ffff	64K	SPLB	mb_plb	
mb_plb	C_BASEA	DDR			U	Not Applicable		
xps_timer_1	C_BASEA	DDR	0x83c00000	0x83c0ffff	64K	SPLB	mb_plb	
RS232_Uart	C_BASEA	DDR	0x84000000	0x8400ffff	64K	SPLB	mb_plb	
DDR_SDRAM	C_MPMC	_BASEADDR	0x8c000000	Ox8fffffff	64M	XCL0:XCL1:SPLB2		

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 generate FPGA bitstream.

See Also

Generating the FPGA Bitmap

1.9 Generating the FPGA Bitmap

Complex FIR Filter Tutorial for MicroBlaze, 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 MicroBlaze-based platform.
- Imported your CoDeveloper-generated files to the Xilinx Platform Studio environment.
- Connected and configured the Impulse C hardware process to the MicroBlaze processor via the FSL bus.

You are now ready to generate the bitmap.

First, from within Platform Studio select the menu item 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 bitstream is generated, the Output Console Window displays the following message:



Now we can move on to add software application.

See Also

Importing the Application Software

1.10 Importing the Application Software

Complex FIR Filter Tutorial for MicroBlaze, Step 10

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

On the **Applications** tab of the **Project Information Area**, create a new software project by doubleclicking **Add Software Application Project...**

Type in the project name: ComplexFIR.

Click OK to exit.

Note: Project Name cann	ot have spaces.
rocessor	microblaze_0
Project is an ELF-o	nly Project
Choose an ELF file.	
	Browse
The ELF file is assumed	to be generated outside XPS
Default ELE name is Ze	w project name>/executable.elf

A new project **ComplexFIR** is added to the project list. Right-click **Sources** under **Project: ComplexFIR** and select **Add Existing Files**. A file selection dialog appears. Enter the **code** directory, and select all the **C** files are shown below:

30

😑 🌺 Project: ComplexFIR 🕠		II ~ ~ ~ ~				
Processor: microblaze_0	Select Source/H	leader File to	Add to Project			? 🗙
Executable: D:\TestingEx ⊕ Compiler Options	Look in:	Code		•	- 🗈 💣 📰 -	
- Sources		Contraction in the second				
E Headers	3	ComplexEilter	c.			
	My Recent Documents	Filter_sw.c				
	B					
	Desktop					
	Mu Documents					
	My Computer					
	My Network Places	File name:	"co_init.c" "ComplexFilte	r.c'' "Filter_sw.	.c'' 💌	Open
		Files of type:	C Sources (*.c)		-	Cancel

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

Next, right-click **Headers** and select **Add Existing Files**. A file selection dialog appears. Enter the **code** directory and select all three header files shown below. Click **Open** to add the files shown to your project.



After you are done with adding files to the **ComplexFIR** project, right-click **Project: ComplexFIR** and select **Build Project**.

Project Inf	ormation Area		×	1	7
Project	Applications	IP Catalog		2	1
Software	Projects			3	1
Cad California Califo	Add Software Application Project Default: microblaze_0_bootloop Default: microblaze_0_xmdstub Project: TestApp_Memory			4 5 6 7 8	// ##
H Pr	Project: TestApp_Peripheral Project: Complex FIP			10	#
Brent Pr	Project: LomplexFIR Processor: microblaze_0 Generated Header: microblaze_1		Set Compiler Options Mark to Initialize BRAMs		
E	kecutable: D:\Te	stingExamples\Cc	Build Project		
Compiler Options Sources			Clean Project Delete Project		
	D:/TestingExa D:/TestingExa D:/TestingExa	mples/ComplexFII mples/ComplexFII mples/ComplexFII	Make Project Ir Generate Linke	nactive r Script	

You will now see the following messages in the Console Window Output:

At Local date and time: Wed May 14 11:03:05 2008
xbash -q -c "cd /cygdrive/d/TestingExamples/ComplexFIR/EDK/; /usr/bin/make -f Spartan3i.make ComplexFIR program; exit;" started...
mb-gcc -02 /cygdrive/d/TestingExamples/ComplexFIR/EDK/code/co_init.c /cygdrive/d/TestingExamples/ComplexFIR/EDK/code/ComplexFIR/EDK/COMPLEXFI

From this, we can see that the size of the generated **ELF** file is over 50KB, which is larger than the **BRAM** size of 8KB. Therefore, we need to put this application on the external **DDR2_SDRAM** for execution.

To do this, first we select Generate Linker Script option from the Project:ComplexFIR menu:

Project Information Area)	×	1	
Project Application	IP Catalog		2	1
Software Projects			З	1
Add Software A	polication Project		4	1
Default: microbl	eze. O bootloop		5	1
			6	
	aze_U_xmastub		7	- #
🗄 🎇 Project: Test	App_Memory		8	- #
🕀 🎇 Project: Test	App_Peripheral		9	1
😑 🌄 Project: Com	olexFIR	Set Compiler Optiv	10	
 Processor: micr Generated Executable: D: Compiler Option 	oblaze_0 Header: microblaze \TestingExamples\C \s	Mark to Initialize B Build Project Clean Project	/RAMs	
Sources		Delete Project		_
D:/Testing	Examples/ComplexF Examples/ComplexF	Make Project Inac	tive	
D:/Testing	Examples/ComplexF	Generate Linker 5	cript	
 Headers 			19	1

The **Generate Linker Script** interface appears. Configure all the section memory in the **Sections View** field as **DDR_SDRAM** as shown.

In the Heap and Stack View, change heap and stack size to 0x4000 bytes, and change the heap and stack memory to DDR_SDRAM as shown.

Sections View:			Heap and Stack View	W:	
Section	Size (bytes)	Memory	Section	Size (bytes)	Memory
.text	0x00003A50	DDR_SDRAM_	Неар	0x4000	DDR_SDRAM_
.rodata	0x0000087A	DDR_SDRAM_	Stack	0x4000	DDR_SDRAM_
.sbss2	0x00000000	DDR_SDRAM_			
.data	0x00000530	DDR_SDRAM_			
.sbss	0x00000000	DDR SDRAM	Memories View:		
	Contraction of the second second				
.bss	0x0000615C	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M	Start Address 0x00000000 0x8C000000	Eength 8K 65536K
bss loot and Vector Sec	0x0000615C Add Se	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M	Start Address 0x00000000 0x8C000000	Length 8K 65536K
.bss }oot and Vector Sec Section	0x0000615C Add Se tions: Address	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M ELF file used to popu	Start Address 0x00000000 0x8C000000 ulate section inform	Length 8K 65536K ation:
.bss }oot and Vector Sect Section .vectors.reset	0x0000615C Add Se tions: Address 0x0000000	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M ELF file used to popu xamples\ComplexFIF	Start Address 0x00000000 0x8C000000 late section inform R_ML401\EDK\Cc	Length 8K 65536K ation:
bss Foot and Vector Sect Section vectors.reset vectors.sw_exception	0x0000615C Add Se tions: Address 0x0000000 0x0000000	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M ELF file used to popu xamples\ComplexFIF	Start Address 0x00000000 0x8C000000 late section inform R_ML401\EDK\Cc	Length 8K 65536K ation: mplexFIR\executable
bss foot and Vector Sect Section vectors.reset vectors.sw_exceptivectors.interrupt	0x0000615C Add Se tions: Address 0x00000000 0x00000008 0x00000010	DDR_SDRAM	Memory ilmb_cntlr_dlmb_cntl DDR_SDRAM_C_M ELF file used to popu xamples\ComplexFIF Output Linker Script:	Start Address 0x00000000 0x8C000000 alate section inform R_ML401\EDK\Cc exFIR/ComplexFI	Length 8K 65536K ation: mplexFIR\executable R_linker_script.ld

Click **OK** to generate the linker script.



Now you will need to rebuild the project to reflect the changes in section mapping.

Now you will need to change the **BRAM** initialization application. Right-click the **Default: microblaze_0_bootloop** project and select **Mark to Initialize BRAMs**. This will let the bootloop reside in the **BRAMs**.



After this, you should redo the **Update Bitstream** to initialize the **BRAMs**.

Exa	mple	es/ComplexFlf	U/EDK/S	parte
Jare	Devi	ice Configuration	Debug	Simul
	BRAM	Update Bitstrean	1	100
		Download Bitstre	am	
		Program Flash M	emory	eat

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

Next, you will run the application from XMD.

See Also

Running the Application

1.11 Running the Application

Complex FIR Filter Tutorial for MicroBlaze, Step 11

Setting up Terminal Window and Connecting Cables

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

Tera Term - COM1 File Edit Setup Contro	port setup		
Port:	СОМ1	ОК	
Baud rate:	9600 🗾		
Data:	8 bit 💌	Cancel	
Parity:	none 💌		
Stop:	1 bit 🔹	Help	
Flow control:	none 💌		
Transmit de	elay sec/char 0 m	nsec/line	1997

Connect the serial port of your development machine to the **RS232** interface on your development board. Make sure the download (**JTAG**) cables are connected on the development board. Also ensure that the board is configured to be programmed. Turn on the power to the board.

Running Application from XMD

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

Select menu Debug -> Launch XMD...

An **XMD Debug Options** dialogue will come up for the first time opening **XMD**. Just click **OK** to continue.

A **Cygwin** bash shell will come up. It runs a script, connecting to the **MicroBlaze** processor and the debugger inside the FPGA. We can learn the base address of the **DDR_SDRAM** is **0x8c000000**.

D:\Xilinx\10.1\EDK\bin\nt\xbash.exe	- 🗆 ×
Version 7.10.d Optimization Performance Interconnect PLBv46 MMU Type No_MMU No of PC Breakpoints 1 No of Read Addr/Data Watchpoints 0 No of Write Addr/Data Watchpoints 0 Instruction Cache Support on Instruction Cache Base Address 0x8c00000 Instruction Cache High Address 0x8ffffff Data Cache Base Address 0x8c00000 Data Cache Base Address 0x8ffffff Exceptions Support off Hard Divider Support off Hard Multiplier Support off Mard Multiplier Support off Mard Suport off <t< td=""><td></td></t<>	
Connected to "mb" target. id = 0 Starting GDB server for "mb" target (id = 0) at TCP port no 1234 XMD% _	•

Now we can download the **ComplexFIR** project **ELF** file to the target board using **XMD** command **dow** as shown below.

dow ComplexFIR/executable.elf con

D:\Xilinx\10.1\EDK\bin\nt\xbash.exe	- 🗆 🗙
<pre>XMD% dow ComplexFIR/executable.elf System Reset DONE Downloading Program ComplexFIR/executable.elf section, .vectors.reset: 0x0000000-0x00000007 section, .vectors.interrupt: 0x00000010-0x00000017 section, .vectors.interrupt: 0x00000020-0x00000027 section, .vectors.hw_exception: 0x00000020-0x00000027 section, .text: 0x8c000000-0x8c003d1f section, .init: 0x8c003d20-0x8c003d43 section, .fini: 0x8c003d20-0x8c003d5f section, .fini: 0x8c003d44-0x8c003d5f section, .rodata: 0x8c003d60-0x8c004b0b section, .data: 0x8c004b0c-0x8c004b0b section, .data: 0x8c004b0c-0x8c004b13 section, .dtors: 0x8c004b1c-0x8c004b1b section, .dtors: 0x8c004b1c-0x8c004b1b section, .jcr: 0x8c004b1c-0x8c004b1f section, .jcr: 0x8c004b20-0x8c004b23 section, .bss: 0x8c004b20-0x8c004b27 section, .bss: 0x8c004b28-0x8c00ac83 section, .heap: 0x8c00ac84-0x8c00ec87 section, .stack: 0x8c00ac84-0x8c00ec87 section, .stack: 0x8c00ac84-0x8c00ec87 section, .stack: 0x8c00ac84-0x8c00ec87</pre>	
Setting PC with Program Start Address 0x00000000	
XMD% con Info:Processor started. Type "stop" to stop processor	
RUNNING> XMDx	-

Now 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 **44** times faster than software only running on **MicroBlaze** microprocessor.



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

See Also

Tutorial 2: Complex FIR on EDK 10.1i