1 Tutorial 2: Complex FIR Filter on the Nios II platform, Cyclone III FPGA



Overview

This tutorial demonstrates how to to use external **FLASH** memory and **SSRAMs** on the platform. The example presented in this tutorial is a **Complex FIR** filter.

The purpose of this tutorial is to take you through the entire process of generating hardware and software interfaces (as was done in Tutorial 1) and importing the relevant files into the Altera environment. The tutorial will also describe how to create the platform and downloadable FPGA bitmap, then run the application software on the platform, all using the Altera tools.

The hardware platform used in this tutorial is Altera Cyclone III Evaluation Kit, featuring Altera Cyclone III EP3C25 FPGA, and a touch-screen LCD display.

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

Steps

Loading the Complex FIR Filter Example Understanding the Complex FIR Filter Example Compiling the Complex FIR Filter for Simulation Building the Complex FIR Filter Example Exporting Files from CoDeveloper Creating the Quartus Project Creating the New Platform Configuring the New Platform Generating the System Generating the FPGA Bitmap Running the Application on the Platform

Note: This tutorial assumes you have purchased or are evaluating the **CoDeveloper Platform Support Package** for **Altera Nios II**, and that you have installed and have valid licenses for the **Altera Quartus II**, **SOPC Builder**, and **Nios II IDE** products.

1.1 Loading the Complex FIR Filter Example

ComplexFIR Filter Tutorial for Nios II, Step 1

To begin, start the CoDeveloper Application Manager by selecting Application Manager from the Start -

> All Programs -> Impulse Accelerated Technologies -> CoDeveloper program group.

Note: this tutorial assumes that you have already read and understand the basic **Hello World** tutorial presented in the CoDeveloper User's Guide, and Tutorial 1: Hello World on the Nios II platform.

Open the Altera Nios II Complex FIR filter sample project by selecting Open Project from the File menu, or by clicking the Open Project toolbar button. Navigate to the .\Examples\Embedded\ComplexFIR_NIOS directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) Opening the project will display a window similar to the following:

📲 Impulse CoDeveloper Applicati	ion Manag	er Universal Edition - [FIR_Accelerator] - [Filter_hw.c]
Eile Edit View Project Tools Win	dow <u>H</u> elp	
i 🚰 😅 🕼 😭 🚇 🎦 🔜 🐰	b 🖻 "	ヽ ⌒ ノ/ タ/ 桷 🏰 👛 🍃 磁 図 🕨 磁 🔍 🕲 🖉 🥔 🍃
Project Explorer 🛛 🔍 🗙	Filter_s	sw.c 🕞 Filter_hw.c
B	7	#include <stdio.h></stdio.h>
🖃 🔂 Application FIR_Accelerator	8	#include "co.h"
🖻 🔄 Source Files	9	#include "cosim_log.h"
sw C ComplexFilter c	10	#include "Filter.h"
hw (c) Filter hw c	11	
	12	extern void call_accelerator (co_stream output_stream, co_st
Filler_sw.c	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;
	19	int i;
Other Files	20	int write_idx;
	21	int read idx;

Files included in the **ComplexFIR** 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.

Quartus subdirectory - Files in the Quartus subdirectory are used later in this tutorial to simplify the creation of the hardware platform.

See Also

2

Step 2: <u>Understanding the Complex FIR Filter Example</u>

Tutorial 2: Complex FIR Filter on the Nios II platform

1.2 Understanding the Complex FIR Filter Example

Complex FIR Filter Tutorial for MicroBlaze, 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 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);
    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 MicroBlaze embedded processor, controlling the stream flow and printing results.

See Also

Step 3: Compiling the Complex FIR Filter for Simulation

Tutorial 2: Complex FIR Filter on the Nios II platform

1.3 Compiling the Complex FIR Filter for Simulation

CompleFIR Filter Tutorial for Nios II, Step 3

Simulating the CompleFIR Application

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.
- 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:

```
#if defined(IMPULSE_C_TARGET)
    // Print Acceleration Numbers
    printf ("\r\n--> Acceleration factor: %.2fX\r\n\n",
elapsedtime_sw/elapsedtime_hw);
    printf ("-----> Visit www.ImpulseC.com to learn more!\r\n\n\n");
#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 Nios II processor, or to the host development system for simulation purposes.

See Also

Step 4: Building the Complex FIR Filter Example

Tutorial 2: Complex FIR Filter on the Nios II platform

1.4 Building the Complex FIR Filter Example

CompleFIR Filter Tutorial for Nios II, Step 4

Specifying the Platform Support Package

The next step, prior to compiling and generating the HDL and related output files, is to select a platform target. To specify a platform target, open the Generate Options dialog as shown below (**Project** -> **Options**, **Generate** tab):

5

Build Simulate Generate System Registration Platform Support Package: Altera Nios II (VHDL) Image: CoBuilder Optimization Options Image: CoBuilder Optimization Options Image: CoBuilder Optimization Options Image: Optimization Options Image: CoBuilder Optimization Options	impulse eccelerated technologies
 Scalarize array variables Relocate loop invariant expressions Additional optimizer options: 	Directories Hardware build directory: hw
CoBuilder Generation Options Generate dual clocks Active-low reset Use std_logic types for VHDL interfaces On not include co_ports in bus interface Library options:	Software build directory: sw Hardware export directory: Quartus Software export directory: Quartus
 Include floating point library Use higher latency, faster clock operators Allow double-precision types and operators 	

Specify Altera Nios II (VHDL) as shown. Also specify hw and sw for the hardware and software directories as shown, and specify Quartus for the hardware and software export directories. 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 on the **Generate HDL** button:

🖛 Impulse CoDeveloper Applicat	ion Manager Universal Edi	tion - [FIR_Accelerator] - [Filter_s
Eile Edit View Project Tools Win	ndow <u>H</u> elp	
Image:	imulation Executable e Simulation Executable	🐝 🇀 💂 i 🛗 🕺 🕨 🛗 🕎 D ComplexFilter.c
Application F Generate HDL Generate HDL Source F Source F Swic C hw C F sw C F Swic F Header F Options	rdware Simulation Executable ed Hardware (HDL) ed <u>S</u> oftware	<pre>nezone tz; apsedtime_hw, elapsedtime_sw pe ata[NUM_SAMPS_IN_SLOT]; er_coef[IF_FUT_LFN];</pre>
sw h ComplexFilter.h Filter.h Project Files ComplexFilter.h Project Files Readme.htm	22 int32 if_fil 23	<pre>Lter_output[NUM_SAMPS_IN_SLOT] ComplexFIR running in Nios II re_fir(void) { coef_mem[IF_FILT_LEN]; filt_hist[IF_FILT_LEN]; inSample;</pre>

A series of processing steps will run in a command window. When processing is complete you will have a number of resulting files created in the **hw** and **sw** subdirectories of your project directory. Take a moment to review these generated files. They include:

Hardware directory ("hw")

- Generated VHDL source files (FIR_Accelerator_comp.vhd, FIR_Accelerator_top.vhd and subsystem.vhd) representing the hardware process and the generated hardware stream and memory interfaces.
- A lib subdirectory containing required VHDL library elements.
- A class subdirectory containing generated files required by the Altera SOPC Builder tools.

Software directory ("sw")

- C source and header files extracted from the project that are required for compilation to the embedded processor (in this case Filter_sw.c, Filter.h, ComplexFilter.c and ComplexFilter.h).
- A generated C file (**co_init.c**) representing the hardware initialization function. This file will also be compiled to the embedded processor.
- A **class** subdirectory containing additional software libraries to be compiled as part of the embedded software application. These libraries implement the software side of the hardware/software interface.

If you are an experienced Altera tools user you may copy these files manually to your Altera project area and, if needed, modify them to suit your needs. In the next step, however, we will show how to use the hardware and software export features of CoDeveloper to move these files into your Altera project automatically.

See Also

Step 5: Exporting Files from CoDeveloper

7

Tutorial 2: Complex FIR Filter on the Nios II platform

1.5 Exporting Files from CoDeveloper

ComplexFIR Filter Tutorial for Nios II, Step 5

As you saw in 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. You can, if you wish, copy these files manually and integrate them into your existing Altera projects. Alternatively, you can use the export features of CoDeveloper to integrate the files into the Altera tools semi-automatically. This section will walk you through the process, using a new Quartus project as an example.

Note: you must have the Altera Quartus II (version 7.1 or later) and SOPC Builder software installed in order to proceed with this and subsequent steps.

Recall that in Step 4 you specified the directory **Quartus** as the export target for hardware and software:

Altera Nios II (VHDL)	impulse
Scalarize array variables	accelerated technologies
Relocate loop invariant expressions	_ Directories
Additional optimizer options:	Hardware build directory:
	hw
CoBuilder Generation Options	Software build directory:
Generate dual clocks	sw
	Hardware export directory:
	Quartus
Use std_logic types for VHDL interfaces	Software export directory:
Do not include co_ports in bus interface	Quartus
Library options:	
I Include floating point library	

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 we have specified both directories as "Quartus"), the specific destination (which may be a subdirectory under **Quartus**) for each file is determined from the **Platform Support Package** architecture library files. It is therefore important that the correct **Platform Support Package** (in this case **Altera Nios II**) 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 Quartus 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.

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 Quartus project directory.

See Also

Step 6: Creating the Quartus Project

Tutorial 2: Complex FIR Filter on the Nios II platform

1.6 Creating the Quartus Project

ComplexFIR Filter Tutorial for Nios II, Step 6

Now we'll move into the Altera tool environment. Begin by launching Altera Quartus II (from the Windows Start -> Altera menu). Open a new project by selecting File -> New Project Wizard. In the field prompting you for the new project's working directory, use the browse button and find the directory (Quartus) to which you exported the hardware and software files in the previous step.

Select the **Quartus** directory and click **Open**. On page one of the **New Project Wizard** dialog, enter **ComplexFIR** in both the project name and top-level design entity fields as shown:

D. lakera impoloophampio	s\ComplexFIR_NIOS\C	Juartus		
What is the name of this pr	plect			
ComplexFIN				
What is the name of the top	p-level design entity for	this project? This r	ame is case sensiti	ve and must
CompleyEIR	ne in the design ne.			
Use Existing Project Setti	nas			

Click **Next** to move to the next page.

Now you will import the VHDL files generated by CoDeveloper, as well as a block diagram file included with this tutorial example, to your Quartus project. In the Add Files page (page 2), add the files in the following order:

- 1. Block diagram file (not generated by CoBuilder): ComplexFIR.bdf
- 1. Core logic files in the user_logic_filt_module subdirectory: subsystem.vhd, FIR_Accelerator_top.vhd and FIR_Accelerator_comp.vhd
- 2. All .vhd files in the impulse_lib project subdirectory

The files should be listed in the opposite order from which they were added (i.e., the **impulse_lib** files should be at the top of the list):

ile name:		Add
File name	Туре	Add All
impulse_lib/avalon_if.vhd	VHDL File	
impulse_lib/cregister.vhd	VHDL File	Hemove
impulse_lib/csignal.vng impulse_lib/divmod.vbd	VHDL File	Properties
impulse_lib/fifo.vhd	VHDL File	Tiopenes
impulse_lib/fifo_dc.vhd	VHDL File	Up
impulse_lib/gmem.vhd	VHDL File	
impulse_lib/impack.vhd	VHDL File	Down
impulse_lib/sema.vhd	VHDL FIIe VHDL File	· · ·
impulse_lib/stream.vhd	VHDL File	
impulse_lib/stream_dc.vhd	VHDL File	
user_logic_filt_module/FIR_Accelerator_comp.vhc	I VHDL File	
user_logic_filt_module/FIR_Accelerator_top.vhd	VHDL File	
user_logic_tilt_module/subsystem.vhd	VHDL File	
	>	
pecify the path names of any non-default libraries.	User Libraries	

Click Next to proceed.

In the **Family and Device Settings** page (page 3) select the device you will be targeting. For this example we will choose **Cyclone III**, **EP3C25F324C6** device, with 324 pins and speed grade 6. This is the FPGA used in the **Cyclone III FPGA Starter Board**.

Device family				- Show in 'Available device' list-			
Family: Cyclone I	11		-	Package:	Any		-
Devices: All Target device C Auto device selected by the Fitter C Specific device selected in 'Available devices' list			-	Pin count:	324	324	
				Speed grade	e: 6		•
			st	Show advanced devices		levices ible only	
vailable devices: Name	Core v	I I F S	llser1/	Memor	Embed	PU	
P3C25E324C6	1.2V	24624	216	608256	132	4	
273640732468	1.20	33000	136	1101210	232		
Companion device-		111					
Companion device —		111					*

Click Next again. Skip the EDA Tool Settings page (page 4) by again clicking Next.

You will see a **Summary** page listing the options you have chosen as shown below:

D:/altera/ImpulseEvamples/I	ComplexFIB_NIOS/Quartus/
Project name:	Complex III Rest qualities
Top-level design entitu:	ComplexFIB
Number of files added:	15
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone III
Device:	EP3C25F324C6
EDA tools:	
Desian entrv/synthesis:	<none></none>
Simulation:	<none></none>
Timing analysis:	<none></none>
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 °C

Click Finish to exit the Wizard and return to Quartus.

The next steps will demonstrate how to create and configure a hardware system with a Nios II processor and the necessary I/O interfaces for our sample application.

See Also

Step 7: Creating the New Platform

Tutorial 2: Complex FIR Filter on the Nios II platform

1.7 Creating the New Platform

ComplexFIR Filter Tutorial for Nios II, Step 7

Now that you have created a Quartus project using the wizard, you will need to specify additional information about your platform in order to support the requirements of your software/hardware application. These steps include the creation of a hardware system with a Nios II processor and the necessary I/O elements.

You will use **SOPC Builder** to create a hardware system containing an Altera Nios II embedded processor, the FPGA module created for the ComplexFIR hardware process by CoBuilder, and several necessary peripherals. To do this, select **Tools -> SOPC Builder** to start **SOPC Builder**.

In the **Create New System** dialog that appears, enter **ComplexFIRSystem** as the **System Name**, and specify **VHDL** for the **Target HDL** language:

😃 Create New System - Altera SC					d.sopc) 📘 🗖 🗙
File Edit Module System View T	ools Help				
System Contents System Generation					
Altera SOPC Builder	Target	Clock Settings			
Inios Il Processor	Device Family: Cyclone III	Name	Source	MHz	Add
⊕ Bridges and Adapters					Remove
Legacy Components					
Memories and Memory Controllers					
± −PLL	Use 📖 Mo 💶 Create New	System	X a	lock Base	End
⊕-USB	System Name: Cor	nplexElRSvstem			
u → Oser Logic					
	Target HDL: OV	erilog			
		HDL			
		OK Cancel			
	<		1111		
Add	Remove Edit	Move Up	Move Down	Address Map	Filter
Info: Your system is ready to generate	ə.				
	Exit Help	Prev Next	Generate		

Click OK to continue.

Note: the System Name that you specify in this step must be a valid VHDL identifier. Specifically, it must not begin with a numeric character or include spaces or other non-alphanumeric characters other than the underscore character (_).

See Also

Step 8: Configuring the New Platform

Tutorial 2: Complex FIR Filter on the Nios II platform

1.8 Configuring the New Platform

ComplexFIR Filter Tutorial for Nios II, Step 8

The following instructions will lead you through the process of creating your Nios II-based platform using **SOPC Builder**. This process requires many steps, but will only need to be done once for each new project that you create.

We'll begin by adding the largest component of the **ComplexFIR** system, the Nios II processor. From the **System Contents** tab (on the left side of the **SOPC Builder** window), double-click **Nios II Processor** under **Altera SOPC Builder**. The **Nios II Processor - cpu** configuration Wizard will appear. Select the **Nios II/s** core as shown below:

👊 Nios II Processor	- срц				×
Mios MegaCore	s II Processor				Documentation
Parameter Settings Core Nios II Cach	hes and Memory Interfaces	Advanced Features	MMU and MPU Settings > JTA	AG Debug Module >	Custom Instructions
Select a Nios II core:					
	ONios II/e	Nios II/s	○Nios II/f	1	
Nios II Selector Guide Family: Cyclone III f _{system:} 50.0 MHz cpuid: 0	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction		
Performance at 50.0 MH	tz Up to 8 DMIPS	Up to 32 DMIPS	Up to 57 DMIPS		
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs		
Memory Usage	Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache		
Hardware Multiply: Emb	oedded Multipliers	Hardware Divide			

Click **Finish** to add the Nios II CPU to the system and return to **SOPC Builde**r. A module called **cpu** appears in the SOPC window.

Haltera SOPC Builder - Comple File Edit Module System View 1	xFIRSystem.sopc* (D:\altera\lmp fools NiosII Help	ulseExamples\Compl	exFIR_NIOS\Quartus\Com	plexFIRSystem.sopc) [
System Contents System Generation						
Marca SOPC Builder	Target	Clock Settings				
Create new component	Device Family: Cyclone III	Name	Source	MHz	ſ	Add
Bridges and Adapters		cik	External	50.0		Remove
Interface Protocols						temore
E-Legacy Components						
Peripherals						
	Use Con Module Name	Description	Clock	Base	End	IRQ
	C cpu instruction_mast data_master itag_debug_mod	Nios II Processoi er Avalon Memory Avalon Memory ule Avalon Memory	r Mapped Master clk Mapped Master Mapped Slave	IRQ • 0x0000080	0 IRQ 0 0x00000fff	31 ← →×

Next, you must add the necessary peripherals to the new Nios II system. If you are not familiar with how to do this in **SOPC Builder**, you may wish to review the information provided in your **Nios II Development Kit** documents, and in particular the tutorials provided by Altera. Refer to the

instructions provided by Altera in the following file:

http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf

The relevant information begins with the section titled **Timer** (page 2-9) and ends with the section titled **External RAM Bus** (Avalon Tri-State Bridge) (page 2-13).

Using the methods described in the Altera documentation (and summarized below), you will need to add the following components:

Timer

To add the **timer** peripheral, perform the following steps:

Select Interval Timer under Peripherals -> Microcontroller Peripherals, and click Add. The Interval Timer - timer wizard appears.

🕮 Edit Module - Altera SOPC B	🚇 Interval Timer - timer 🛛 🔀
File Edit Module System View	🔹 Interval Timer
System Contents System Generation	Megacore Documentation
Attera SOPC Builder Create new component Nios II Processor Create new components Pridges and Adapters Create Protocols Create	Parameter Settings Timeout period Period: 1 Image: Setting Seting Setting Setting Seting Setting Setting Seting Setting Setting
Add	
To Do: enur No reset vector has he	
To Do. cpu . No reset vector has be	
A Warning: chu: Reset vector and Ev	
, romming, open, reson votion and LA	Cancel Finish

Leave the options at their default settings, and click **Finish** to add the timer to your system. You are returned to the **Altera SOPC Builder** window.

External Flash Memory Interface

To add the **external flash** peripheral, perform the following steps:

Select Flash Memory (CFI) under Memories and Memory Controllers -> Flash, and click Add. The Flash Memory (CFI) - cfi_flash wizard appears.

Make sure Intel 128P30 is selected in the Presets drop-down box. In the Size box, change the Address Width to 23 bits, and Data Width to 16 bits.

🛍 Edit Module - Altera SOPC Bu	🖷 Flash Memory (CFI) - cfi_flash 🛛 🔀
File Edit Module System View	💏 Flash Memory (CFI)
System Contents System Generation	MegoCore" Documentation
Attera SOPC Builder Create new component Nios II Processor	Parameter Settings Attributes Timing
 Bridges and Adapters Interface Protocols Legacy Components 	Presets: Intel 128P30
E - Memories and Memory Controllers ⊕-DMA	Size
Flash CompactFlash Interface Gers Serial Flash Contro Flash Memory (CFI)	Address Width (bits): 23
⊕ On-Chip ⊕ SDRAM ⊕ SRAM ⊕ Peripherals	Create an interface to any industry-standard CFI (Common Flash Interface)-compliant flash memory device. Select from a list of tested flash memories or provide interface and timing information for a CFI memory device which does not appear on the list.
 ➡ PLL ■ USB ■ User Logic ■ Video and Image Processing 	Info: Flash memory capacity: 16.0 MBytes (16777216 bytes).
	Cancel < Back Next > Finish

Design Entry **2**

Click Finish. You are returned to the Altera SOPC Builder window.

External SRAM Interface

To add the **external SRAM** peripheral, perform the following steps:

Select Cypress CY7C1380C SSRAM under Memory -> SRAM, and click Add. The Cypress CY7C1380C SSRAM - ssram wizard displays.

Make sure the memory size is set to 1 MBytes:

🛍 Edit Module - Altera SOPC Builder	🗳 Cypress CY7C1380C SSRAM - ssram
File Edit Module System View Tools	Cypress CY7C1380C SSRAM
System Contents System Generation	MegeCore
Altera SOPC Builder Altera SOPC Builder Nios II Processor Bridges and Adapters Hinterface Protocols Legacy Components	Parameter Settings Synchronous static RAM The Nios Development Board (Cyclone II 2c35 and Stratix II 2s60 edition) has a Cypress CY7C1380C-167AC
	SSRAM chip arranged as 512K by 36 bits (32 bits are used by this component resulting in 2MBytes total address span).
⊕ On-Chip ⊕ SDRAM ⊜ SRAM	This SSRAM interface allows parameterization of SSRAM size and read latency to accommodate your desired device and clock speed selection.
Cypress CY7C1380C SSRAM DT71V416 SRAM DT71V416 SRAM Peripherals PLL ⊕-USB	Timing parameters Read latency (cycles): 2
⊕-User Logic ⊕-Video and Image Processing	SSRAM size Memory size (MBytes): 18 word aligned address bits
	Generic memory model (simulation only) Include functional memory model in the simulation testbench
Add	Cancel Finish

Click Finish. You are returned to the Altera SOPC Builder window.

External RAM Bus (Avalon Tristate Bridge)

For the Nios II system to communicate with memory external to the FPGA on the development board, you must add a bridge between the Avalon bus and the bus or buses to which the external memory is connected.

To add the Avalon tristate bridge, perform the following steps:

Select Avalon-MM Tristate Bridge under Bridges and Adapters -> Memory Mapped, and click Add. The Avalon-MM Tristate Bridge - tristate_bridge wizard displays. See that the Registered option is turned on by default.

🛍 Edit Module - Altera SOPC Builde	🐸 Avalon-MM Tristate Bridge - tristate_bridge	
File Edit Module System View Tools System Contents System Generation	Avalon-MM Tristate Bridge	entation
Attera SOPC Builder Create new component Nios II Processor Bridges and Adapters Avalon-MM Clock Crossing Bi Avalon-MM Pipeline Bridge Avalon-MM Pipeline Bridge JTAG to Avalon Master Bridge SPI Slave to Avalon Master B SPI Slave to Avalon Master B Breaming Interface Protocols Legacy Components Memories and Memory Controllers Peripherals PULL User Logic Video and Image Processing	Parameter Settings Shared Signals Incoming Signals Shared Signals Increases off-chip fmax, but also increases latency. Increases off-chip fmax, but also increases latency. Not registered Reduces latency, but also reduces fmax. Note: Check the input setup times analysis in the Quartus compilation report to be sure your bus inputs meet system-level timing requirements. Outgoing address and control signals are always registered. Cancel < Back	:> Finish

Design Entry 2

Click Finish. You are returned to the Altera SOPC Builder window.

Connect the External Memories to the Tristate Bridge

The external memories **cfi_flash** and **ssram** modules must be connected to the **tristate_bridge**. Click both the open circles inside the red oval to make the connections. The open circles will turn black to indicate a bus connection.

Use	Connec	Module Name	Description	Clock	Base	End	IRQ
✓		C cpu instruction_master data_master jtag_debug_module	Nios II Processor Avalon Memory Mapped Master Avalon Memory Mapped Master Avalon Memory Mapped Slave	cik	IRQ 0x0000800	0 IRQ 31 0x00000fff	-
V	$\downarrow \rightarrow$	E timer s1	Interval Timer Avalon Memory Mapped Slave	cik	₽ 0x0000000	0x0000001f)
V	A.	E cfi_flash s1	Flash Memory (CFI) Avalon Memory Mapped Tristate Slave	cik	₽ 0x0000000	0x00ffffff	
V		E ssram	Cypress CY7C1380C SSRAM Avalon Memory Mapped Tristate Slave	cik	₽ 0x0000000	0x000fffff	
	Y .	🗆 tristate_bridge	Avalon-MM Tristate Bridge				
	►►	avalon_slave tristate_master	Avalon Memory Mapped Slave Avalon Memory Mapped Tristate Master	cik			

Double-click the **tristate_bridge** module to edit the its **Parameter Settings**. In the **Shared Signals** tab, Check **address** under both **ssram.s1** and **cfi_flash.s1**. This will allow the external ssram and flash module to share the address bus in the generated system.

🕌 Avalon-J	MM. Tristate Bridge - tristate_bridge	×
MegaCore'	Avalon-MM Tristate Bridge	Documentation
Parameter Settings Incoming Sid	Inals Shared Signals	
Specify which Note: The dat	n signals should share external system connections. a signal is always shared, and the chipselect_n signal is never sha	red.
ssram.s1 address adsc_n bw_n bwe_n outputen	able_n	
cfi_flash.s1 ✓ address ☐ read_n ☐ write_n		
	Cancel < t	Back Next > Finish

Click Finish to accept the changes.

JTAG UART Interface

The **JTAG UART** is used for communication between the board and the host machine and for debugging software running on the Nios II processor. To add the **JTAG UART** peripheral, **jtag_uart**, perform the following steps:

Locate Interface Protocols -> Serial -> JTAG UART, and double-click to add. The JTAG UART - jtag_uart wizard displays.

Leave all options at their default settings.

🛍 Edit Module - Altera SOPC Build	🗳 JTAG UART - jtag_uart 🛛 🔀
File Edit Module System View Tool System Contents System Generation	JTAG UART
Altera SOPC Builder Create new component Nios II Processor Bridges and Adapters Interface Protocols ASI Ethernet High Speed PCI Serial Serial JTAG UART SPI (3 Wire Serial) UART (RS-232 Serial Port) Legacy Components Peripherals PLL USB USER Logic Video and Image Processing	Megecer Parameter Settings Configuration Simulation Write FIFO (Data from Avalon to JTAG) Buffer depth (bytes): 64 © Construct using registers instead of memory blocks Read FIFO (Data from JTAG to Avalon) Buffer depth (bytes): 64 Image: Construct using registers instead of memory blocks Read FIFO (Data from JTAG to Avalon) Buffer depth (bytes): 64 Image: Construct using registers instead of memory blocks Construct using registers instead of memory blocks
	Cancel < Back Next > Finish

Click Finish. You are returned to the Altera SOPC Builder window.

Adding the Hardware Process Module "filt_module"

Now add the **filt_module**, which implements the **ComplexFIR** hardware process. Double-click **User Logic** under **Avalon Modules** in the System Contents pane. Select **img_arch module** and click **Add**:



The **ComplexFIR** hardware process, **user_logic_filt_module_classic_0** module, will be connected to the shared Avalon data bus automatically. The system module listing will appear as shown below:

Use	Connec	Module Name	Description	Clock	Base	End	IRQ
		cpu instruction_master data_master jtag_debug_module	Nios II Processor Avalon Memory Mapped Master Avalon Memory Mapped Master Avalon Memory Mapped Slave	cik	IRQ 0	IRQ 31 0x00000fff	1 L
~	$\downarrow \rightarrow$	E timer s1	Interval Timer Avalon Memory Mapped Slave	clk	0x0000000	0x0000001f	
	_● →	E cfi_flash s1	Flash Memory (CFI) Avalon Memory Mapped Tristate Slave	clk	■ 0x0000000	0x00ffffff	
~		s1	Cypress CY7C1380C SSRAM Avalon Memory Mapped Tristate Slave	clk	■ 0x00000000	0x000fffff	
>	┝╋┶	 tristate_bridge avalon_slave tristate_master 	Avalon-MM Tristate Bridge Avalon Memory Mapped Slave Avalon Memory Mapped Tristate Master	cik			
>	$\downarrow \rightarrow$	itag_uart avalon_itag_slave	JTAG UART Avalon Memory Mapped Slave	clk	■ 0x00001020	0x00001027	F
•		user_logic_filt_module_classic_0 p_cpu_proc_output_stream p_cpu_proc_input_stream	filt module Avalon Memory Mapped Slave Avalon Memory Mapped Slave	cik		0x0000100f 0x0000101f	

PLL

We need to generate different clocks for various modules. The CPU can run at a fast clock of 100 MHz, while peripherals need slower clock sources. The hardware process filt_module needs to run at a slow clock of 40 MHz. Adding a PLL module will serve this purpose.

To add a PLL module, simple select it under PLL, and click Add:

🚇 Edit Module -	📲 PLL - pli 🛛 🔀
File Edit Module System Contents s	PLL Documentation
Altera SOPC Buil Nios II Proces Bridges and Ada Harden Ada Har	Parameter Settings Interface PLL Settings Interface PLL configuration Interface The Avalon PLL configuration wizard creates a component wrapper around an Altera ALTPLL megafunction. Use the ALTPLL MegaVVizard to configure the PLL settings. When you finish configuring the PLL, the PLL clock output appears in the clock table on the SOPC Builder System Contents tab. Launch Altera's ALTPLL MegaVVizard
	Cancel < Back Next > Finish

Click Launch Altera's ALTPLL MegaWizard to continue. The MegaWizard Plug-In Manager will appear as shown below:



MegaWizard Plug-In Manager [page	l of 12]	
		About Documentation
Parameter 2 PLL 3	Dutput 4 EDA 5 Summary	
General/Modes Inputs/Lock Bandwi	dth/S5 🔪 Clock switchover 🔪	
	Currently selected device family	u Cudana III
altplipli		Match project/default
		Match project/dei adic
inclk0 inclk0 frequency: 50.000 MHz	Able to implement the requested PLL	
Operation Mode: No Compensation		
Clk Ratio Ph (dg) DC (%)	General	
Cyclone		
	Which device speed grade will you be using?	Any 🔽
	Use military temperature range devices only	
	What is the frequency of the inclock0 input?	50.00 MHz 🖌
	Set up PLL in L <u>V</u> DS mode Data rate:	Not Available 🔽 Mbps
	PLL type	
	Which PLL type will you be using?	
	⊖ Fast P <u>L</u> L	
	C Enhanced PLL	
	Select the PLL type automatically	
	Operation mode	
	How will the PLL outputs be generated?	
	 Use the feedback path inside the PLL 	
	O In Normal Mode	
	In Source-synchronous Compensation Mode In Zero Delay Buffer Mode	
	Connect the fbmimic port (bidirectional)	
	With no compensation	
	Create an 'fbin' input for an external feedback (External Fe	eedback Mode)
	Which output clock will be compensated for?	c0 💌
	Cancel	

In the **Operation mode** box, check **With no compensation** item. Click **Next** to continue.

Switch to the **Output Clocks** tab of the wizard. In **page 6** of the wizard, check **Use this clock** to activate clock **c0**. Enter **2** in the **Clock multiplication factor** box. This will create a clock **c0** of **100 MHz**.

ALTPLL ALTPLL Parameter Settings Plus Aconfiguration Clocks Clo	About	Documentation
Settings Reconfiguration Clocks		
Clic Ratio Ph (dg) DC (%) Cyclone III Cyclone III Colock multiplication factor Clock givision factor	Requested settings 100.0000000 MHz 2 1 Copy	Actual settings 100.000000 2 1
Phase shift step resolution(ps) Clock d <u>u</u> ty cycle (%)	50.00 × ps ×	50.00

Click Next to continue. In page 7 of the wizard, check Use this clock to activate clock c1. Enter 2 in the Clock multiplication factor box. In the Clock phase shift box, enter -2000.00 ps. This will create a clock c1 of 100 MHz.

MegaWizard Plug-In Manager [page 7 of 12]			
ALTPLL		About	Documentation
Parameter 2 PLL 3 Output 4 ED/ Settings Reconfiguration Clocks clk c0 clk c1 clk c2 clk c3	A 5 Summary		
altplipli inclk0 inclk0 frequency: 50.000 MHz Operation Mode: No Compensation c1 Clk Ratio Ph (dg) DC (%) c0 2/1 0.00 60.00	c1 - Core/External Output Clock Able to implement the requested PLL ✓ Use this clock Clock Tap Settings O Enter output clock frequency: • Enter output clock parameters: Clock multiplication factor	Requested settings	Actual settings
	Clock glvision factor Clock phase shift	-2000.00 ps	-2000.00
	Clock duty cycle (%)	50.00	50.00

Click Next to continue. In page 8 of the wizard, check Use this clock to activate clock c2. Enter 6 in the Clock multiplication factor box, and 5 in the Clock division factor box. This will create a clock

c2 of 60 MHz.

MegaWizard Plug-In Manager [page 8 of 1	2]		
		About	Documentation
Parameter PLL Settings PL Reconfiguration Clocks	4 EDA 5 Summary		
altpilpil incik0 incik0 frequency: 50.000 MHz	c2 - Core/External Output Clock Able to implement the requested PLL		
Operation wode: No Compensation CIk Ratio Ph (dg) DC (%) o0 2/1 0.00 50.00 o1 2/1 -72.00 50.00 o2 6/6 0.00 50.00	Clock Tap Settings Enter output clock frequency: Enter output clock parameters: Clock <u>multiplication factor</u> Clock <u>division factor</u>	Requested settings 100.0000000 MHz 6 5 C <copy< td=""><td>Actual settings 60.000000 6 5</td></copy<>	Actual settings 60.000000 6 5
	Clock p <u>h</u> ase shift Phase shift step resolution(ps)	v 3q	0.00
	Clock duty cycle (%)	50,00	50.00

Click Next to continue. In page 9 of the wizard, check Use this clock to activate clock c3. Enter 4 in the Clock multiplication factor box, and 5 in the Clock division factor box. This will create a clock c2 of 60 MHz.

MegaWizard Plug-In Manager [page 9 of 12]			
ALTPLL 1 Parameter Settings Clocks	EDA 5 Summary	About	Documentation
Cikto Cikto Cikto Cikto incik0 altplipil c0 c1 c1 c1 c1 c2 c3 c1 c2 c3 c4 c0 c3 c4 c0 c1 c2 c3 c3 c3 c3 c3 c3 c3 c3 c4 c3 c3 c4 c3 c3 c4 c3 c3 c4 c4 c3 c3 c3 c4 c	c3 - Core/External Output Clock Able to implement the requested PLL	Requested settings 100.0000000 MHz 4 • 5 • 0.00 • ps • 50.00 •	Actual settings 40.000000 4 5 0.00 50.00

28

MegaWizard Plug-In Manager [page 12 of 12] S	jummary	
ALTPLL 1 Parameter 2 PLL Settings Reconfiguration Clocks 4 E	DA 5 Summary	<u>About</u> <u>D</u> ocumentation
altplipli	Turn on the files you wish to ger automatically generated, and a Finish to generate the selected I subsequent MegaWizard Plug-Ir	erate. A gray checkmark indicates a file that is red checkmark indicates an optional file. Click iles. The state of each checkbox is maintained in Manager sessions.
inclk0 frequency: 50.000 MH2 C0 Operation Mode: No Compensation C2 Clk Ratio Ph (dg) DC (%) C3 o0 2/1 0.00 50.00 o1 2/1 -77 00 50.00	The MegaWizard Plug-In Manage directory: D:\altera\ImpulseExamples\Com	er creates the selected files in the following plexFIR_NIOS\Quartus\
c2 6/5 0.00 50.00	File	Description
Cyclone III	Image: style Image: style	Variation file PinPlanner ports PPF file AHDL Include file VHDL component declaration file Quartus II symbol file Instantiation template file

Now we are done with configuring clocks. Click Finish to view the Summary page of the wizard.

Click **Finish** again to accept the PLL settings, and then click the **Finish** button on the **PLL - pll** dialog box. A **pll** module will be added to the system.

Rename Clocks

In order to better identify the clocks, rename the clocks as follows:

- External -> osc_clk
- pll.c0 -> cpu_clk
- pll.c1 -> ssram_clk
- pll.c2 -> peripheral_clk
- pll.c3 -> filt_co_clk

The Clock Settings in the SOPC Builder will appears as shown below:

Clock Settings			
Name	Source	MHz	Add
cpu_clk	pll.c0	100.0	
ssram_clk	pll.c1	100.0	Remov
peripheral_clk	pll.c2	60.0	
filt_co_clk	pll.c3	40.0	
	Clock Settings Name cpu_clk ssram_clk peripheral_clk filt_co_clk	Clock Settings Name Source cpu_clk pll.c0 ssram_clk pll.c1 peripheral_clk pll.c2 fift_co_clk pll.c3	Name Source MHz cpu_clk pll.c0 100.0 ssram_clk pll.c1 100.0 peripheral_clk pll.c2 60.0 fift_co_clk pll.c3 40.0

Next, change the **Clocks** for each module to the following settings:

- osc_clk: pll
- cpu_clk: cpu, cfi_flash, ssram and tristate_bridge
- peripheral_clk: timer and jtag_uart

To do so, click on the clock name, and choose the right clock source. The example of changing the **Clock** of **cpu** from **osc_clk** to **cpu_clk** is shown below:

Targ	et		Clock Settings							
Devid	e Family: Cyc	sione III 🛛 🔽	Name		Source		MHz			Add
			cpu_clk ssram_clk peripheral_clk filt_co_clk		pli.c0 pli.c1 pli.c2 pli.c3		100.0 100.0 60.0 40.0			Remove
Use	Connec	Module Name		Description		Clock	E	Base	End	IRQ
		🗆 cpu		Nios II Process	or					
	-	instruction_ma	aster	Avalon Memory	/ Mapped Master	osc clk	~			
		data_master		Avalon Memory	/ Mapped Master	cpu_clk		IRQ O	IRQ	31
		jtag_debug_m	odule	Avalon Memory	/ Mapped Slave	filt_co_clk	all.	0x00000800	0x00000fff	
~		S1		Interval Timer Avalon Memory	/ Mapped Slave	osc_clk peripheral_clk	11	0x00000000	0x0000001f	, ⊢¶
		⊡ cfi_flash		Flash Memory I Avalon Memory	(CFI) / Mapped Tristate Slave	ssram_clk	_	0x00000000	0x00ffffff	
V		⊟ ssram s1		Cypress CY7C Avalon Memory	1380C SSRAM / Mapped Tristate Slave	cpu_cik	-	0x00000000	0x000fffff	

After this is done, the modules and their associated names and bus connections in the **SOPC Builder** should appear as below:

Targ	et		Clock Settings							
Devic	e Family: Cycl	one III 🔽	Name		Source		MHz			bbA
			cpu_clk ssram_clk peripheral_clk filt_co_clk		pli.c0 pli.c1 pli.c2 pli.c3	1 1 6 4	100.0 100.0 30.0 (40.0 (Remove
Use	Connec	Module Name		Description		Clock	в	ase	End	IRQ
		🗆 сри		Nios II Processor	ł.		-			1
		instruction_ma data_master jtag_debug_mo	ster	Avaion Memory I Avaion Memory I Avaion Memory I	Mapped Master Mapped Master Mapped Slave	cpu_clk	-	IRQ 0 0x00000800) IRQ 0x00000fff	31
		l⊟ timer s1		Interval Timer Avalon Memory I	Mapped Slave	peripheral_clk	.	0x00000000	0x0000001f	
	$ \rightarrow$	⊡ cfi_flash ঙ1		Flash Memory (C Avalon Memory I	Fl) Mapped Tristate Slave	cpu_clk	ii)	0x00000000	0x00ffffff	
		⊡ ssram s1		Cypress CY7C1: Avalon Memory I	380C SSRAM Mapped Tristate Slave	cpu_clk	iii)	0x00000000	0x000fffff	
	\↓	tristate_bridge avalon_slave tristate maste	e r	Avalon-MM Trista Avalon Memory I Avalon Memory I	ate Bridge Mapped Slave Mapped Tristate Master	cpu_clk				
		□ jtag_uart avalon_jtag_sl	ave	JTAG UART Avalon Memory I	Mapped Slave	peripheral_clk	c	0x00001020	0x00001027	⊢ ⊢ fi
		user_logic_filt p_cpu_proc_o p_cpu_proc_ir	module_classic_0 utput_stream nput_stream	filt module Avalon Memory I Avalon Memory I	Mapped Slave Mapped Slave	cpu_clk	8	0x00001000 0x00001010	0x0000100f 0x0000101f	
		□ pll s1		PLL Avalon Memory I	Mapped Slave	osc_clk	in?	0x00001040	0x0000105f	

Setting "More 'cpu' Settings"

Now double-click the **cpu** module to edit the **Nios II Processor - cpu** settings. Change the **Reset Vector Memory** to **cfi_flash**, and the **Exception Vector Memory** to **ssram** as shown:

Mios Nios	II Processor			Documentation
arameter ettings				
Core Nios II 🔪 Cach	ies and Memory Interfaces $>$	Advanced Features $ ightarrow$ M	IMU and MPU Settings $ > $ JTAG Debug M	1odule > Custom Instructions
ore Nios II				
Select a Nios II core:				
	ONios II/e	Nios II/s	○Nios II/f	
Nios II Selector Guide Family: Cyclone III f _{system:} 100.0 MHz cpuid: 0	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction	
Performance at 100.0 M	Hz Up to 15 DMIPS	Up to 64 DMIPS	Up to 113 DMIPS	
.ogic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs	
Hardware Multiply: Emb	edded Multipliers	Hardware Divide	Inree M9KS + cache	

Click Finish to save the changes.

Assign Addresses

We can see that as we add modules, error messages appear in the console window showing address conflicts. Here, we let the SOPC to re-assign addresses for all the memory-mapped modules to avoid address overlaps. From the **SOPC Builder** menu, select **System** -> **Auto-Assign Base Addresses**. The newly assigned addresses are shown below:

Use	Connec	Module Name	Description	Clock	Base	End	IRQ
		🗖 cpu	Nios II Processor		5		
		instruction_master data_master jtag_debug_module	Avalon Memory Mapped Master Avalon Memory Mapped Master Avalon Memory Mapped Slave	cpu_clk	IRQ (• 0x02200800) IRQ 3. 0x02200fff	ı
 Image: A set of the set of the		🖂 timer	Interval Timer				100
	$ \rightarrow$	s1	Avalon Memory Mapped Slave	peripheral_clk	0x02201000	0x0220101f	⊳—þ
	$ \rightarrow$	E cfi_flash s1	Flash Memory (CFI) Avalon Memory Mapped Tristate Slave	cpu_clk	₽ 0x0100000	0x0lffffff	
 Image: A start of the start of		🖻 ssram	Cypress CY7C1380C SSRAM				
	$ \rightarrow$	s1	Avalon Memory Mapped Tristate Slave	cpu_clk	∅ 0x02100000	0x021fffff	
	ΙΨĘ	tristate_bridge avalon_slave tristate_master	Avalon-MM Tristate Bridge Avalon Memory Mapped Slave Avalon Memory Mapped Tristate Master	cpu_clk			
V		itag_uart avalon_itag_slave	JTAG UART Avalon Memory Mapped Slave	peripheral_clk	■ 0x02201060	0x02201067	⊨¶
		user_logic_filt_module_classic_0 p_cpu_proc_output_stream p_cpu_proc_input_stream	filt module Avalon Memory Mapped Slave Avalon Memory Mapped Slave	cpu_clk		0x0220104f 0x0220105f	
~	$ \; \bigcup \;$	e pli s1	PLL Avalon Memory Mapped Slave	osc_clk	■ 0x02201020	0x0220103f	25

Save the system by selecting File -> Save from the SOPC Builder menu.

Your new Nios II platform is ready for system generation.

See Also

32

Step 9: Generating the System

Tutorial 2: Complex FIR Filter on the Nios II platform

1.9 Generating the System

ComplexFIR Filter Tutorial for Nios II, Step 9

At this point you have set up and configured your new **Nios II**-based platform, including the hardware module generated by **CoDeveloper**, and can now start the system generation process within the **SOPC Builder**.

Click **Generate** on the bottom of the **SOPC Builder** window to generate the system. The **SOPC Builder** will automatically switch to the **System Generation** tab and display generation information.Make sure the **Simulation** option is unchecked to save time. This process may take several minutes.

File Edi	ra SOPC Builder - ComplexFIRSystem.sopc (Dr\altera\Impulseixamples\ComplexFIR_NIOS\Quartus\Compl t Module System View Tools NiosII Help
System	Contents System Generation
Option	8
System	imulation. Create project simulator files. Run Simulator
Nios II	Tools
Nios	s II IDE
Int	io: Processing started: Mon Feb 23 13:38:33 2009
🕕 Info:	Command: quartus_sh -t ComplexFIRSystem_setup_quartus.tcl
Info:	Evaluation of Tcl script ComplexFIRSystem_setup_quartus.tcl was successful
🕕 Info:	Quartus II Shell was successful. 0 errors, 0 warnings
Inf	io: Peak virtual memory: 47 megabytes
Inf	io: Processing ended: Mon Feb 23 13:38:33 2009
Inf	io: Elapsed time: 00:00:00
Inf	io: Total CPU time (on all processors): 00:00:00
# 200	09.02.23 13:38:34 (*) Completed generation for system: ComplexFIRSystem.
# 200	09.02.23 13:38:34 (*) THE FOLLOWING SYSTEM ITEMS HAVE BEEN GENERATED:
SO	PC Builder database : D:/altera/ImpulseExamples/ComplexFIR_NIOS/Quartus/ComplexFIRSystem.ptf
Sys	stem HDL Model : D:/attera/ImpulseExamples/ComplexFIR_NIOS/Quartus/ComplexFIRSystem.vhd
Sys	stem Generation Script : D:/altera/ImpulseExamples/ComplexFIR_NIOS/Quartus/ComplexFIRSystem_generation_script
# 200	09.02.23 13:38:34 (*) SUCCESS: SYSTEM GENERATION COMPLETED.
🕕 Info:	System generation was successful.
₹[

When generation is complete you may exit SOPC Builder and return to Quartus.

Now you will need to use the **block diagram editor** to connect the complete **SOPC Builder**-generated system (which includes the **ComplexFIR** hardware process module, the **Nios II processor**, and peripherals) to the pins on the FPGA.

To begin, open the block diagram file **ComplexFIR.bdf** by selecting the **Files** tab in the **Project Navigator** window and double-clicking **ComplexFIR.bdf**. The block diagram file contains input and output pins to be connected to the **ComplexFIRSystem** symbol as shown below:



Now add the block representing the **SOPC Builder**-generated system. Double-click anywhere in the open block diagram file to bring up the **Symbol** dialog. Open the **Project** folder and select the **ComplexFIRSystem** symbol as shown below:



Click **OK**. A symbol outline appears attached to the mouse pointer. Align the outline with the pins on the block diagram and click once to place the symbol as shown:



Pin Assignment

The next step is to assign pins. Instead of assigning each individual pin (a tedious process), this tutorial includes a Tcl script that does the pin assignments for you. To run the Tcl script, select **Tools** -> **Tcl Scripts...** The following dialog will appear:

Libraries:	Run
Project 	Open File
CyclonellI_pins_ext_ram.tcl	Add to Tcl Toolbar
💼 d:/altera/80/quartus/common/tcl/apps/gui/	Cancel
Preview: # Copyright (C) 1991-2008 Altera Corpor # Your use of Altera Corporation's desi	ation gn tools, logi(
Preview: # Copyright (C) 1991-2008 Altera Corpor # Your use of Altera Corporation's desi # and other software and tools, and its # functions, and any output files from # (including device programming or simu # associated documentation or informati # to the terms and conditions of the Al	ation gn tools, logi AMPP partner . any of the for lation files), on are express. tera Program L:

Select **CyclonellI_pins_ext_ram.tcl** in the **Project** folder and click **Run** to assign the pins in your design.

Your project is now ready for bitmap generation and subsequent downloading.

Tip: you may wish to to save your Altera project at this point and save a copy for later use with other CoBuilder-generated projects.

See Also

Step 10: Generating the FPGA Bitmap

Tutorial 2: Complex FIR Filter on the Nios II platform

1.10 Generating the FPGA Bitmap

ComplexFIR Filter Tutorial for Nios II, Step 10

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 Altera Quartus II project and used SOPC Builder to create a new Nios II-based

platform.

- Imported your CoDeveloper-generated files to the Altera tools environment.
- Completed a block diagram and assigned pins for the selected FPGA device.

You are now ready to generate the bitmap and download the complete application to the target platform. This process is not complicated (at least in terms of your actions at the keyboard) but can be time consuming due to the large amount of processing that is required within the **Altera** tools.

Pin Settings

First, you must apply some compiler settings related to pin assignment. Select Assignments -> Settings... from the Quartus menu, and select the Device Category.

ategory:								
General								
Files Libraries	Select the family and dev	vice you want to	target for c	ompilation.				
 Device Operating Settings and Conditions 	Device family					- Show in 'Available devices' list		
Voltage	Family: Cyclone III				•	Package	An	, –
Compilation Process Settings Early Timing Estimate	Devices: All				Pin count	: 324	1 •	
Incremental Compilation	- Target device					speed gr	ade: 6	.
Design Entry/Synthesis Simulation	Auto device selected by the Fitter Specific device selected in 'Available devices' list					HardCopy compatible only		
Firming Analysis Formal Verification Physical Supplements	C Other: n/a Device and Pin Options							
Board-Level	Available devices:							
Analysis & Synthesis Settings	Name	Core v	LEs	User I/	Memor	Embed	PLL	Global
	EP3C25F324C6	1.2V	24624	216	608256	132	4	20
Verilog HDL Input Verilog HDL Input Default Parameters Synthesis Netlist Optimizations Fitter Settings Physical Synthesis Optimizations Timing Analysis Settings TimeQuest Timing Analyzer	EP3C40F324C6	1.2V	39600	196	1161216	252	4	20
E Classic Timing Analyzer Settings	Migration compatibility		Companion	device ——				
- Assembler - Design Assistant	Migration Devices	1	HardCopy;					Ŧ
- SignalTap II Logic Analyzer - Logic Analyzer Interface	0 migration devices set	ected	🔽 Limit DS	P & RAM to	HardCopy	levice resol	lices	
Simulator Settings								

Click the Device & Pin Options... button to open the Device & Pin Options dialog:

In the **General** tab, check the **Enable device-wide reset (DEV_CLRn)** option. This will allow the system to be reset by an external push button on the board.

Specify general device opt scheme.	ions. These	options are n	ot dependent (on the cor	nfiguration
Options:					
 Auto-restart configuration Release clears before to Enable user-supplied statements 	in after error ri-states art-up clock	(CLKUSR)			
Enable device-wide res	et (DEV_CLI put enable (i	Rn) DEV_OE)			
Enable INIT_DONE out	tput				
F Auto usercode					
JTAG user code (32-bit he:	kadecimal):	FFFFFFF			
In-system programming cla	mp state:		3	-	
Delay entry to user mode:				÷	
Description:					
Enables the DEV_CLRn p an external source. If this	in, which all option is turn	ows all registe ied off, the DB	rs of the devic V_CLRn pin i	e to be re s disabled	set by 🛛 📩 I when

Next, select the **Dual-Purpose Pins** tab and specify **Use as regular I/O** for all dual-purpose pins listed:

The default settings for each pi selected in the Configuration ta Note: For HardCopy, these sett Dual-purpose pins:	n depend on the current configuration scheme b, which is: Active Serial ings apply to the FPGA prototype device.	рюке.
Name	Value	
Data[0] Data[1]/ASD0 Data[72] DCLK FLASH_nCE/nCS0 nCE0 Other Active Parallel pins	Use as regular 1/0 Use as regular 1/0	
Description:		
Specifies how the Data[0] pin mode after configuration is cor configuration scheme, the Dat input that is tri-stated, as an ou unspecified signal, or compiler	should be used when the device is operating in u aplete. Depending on the current device and a[0] pin can be reserved as a regular I/O pin, as a tput that drives ground, as an output that drives a configured. If the Data[0] pin is reserved as a reg	ser 🔼 an 📕 an gular 💟

Click **OK** to save the changes.

Compiling the System

Now you're ready to synthesize, download, and run the application. To generate the bitmap, select **Processing -> Start Compilation** as shown below:

📸 File Edit View Project Assignments	Processing Tools Window Help	
D 🗃 🖬 🌒 🍯 🕺 🖻 🛍 🖍	Etop Processing	Ctrl+Shift+C
Project Navigator 🔜 🔺 🗙	Start Compilation	Ctrl+L
Files	Analyze Current <u>File</u> St <u>a</u> rt Update Memory Initialization File Compilation <u>R</u> eport	• Ctrl+R

38



From the Task window, you can see the compilation progress.

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

During compilation, **Quartus** will analyze the generated VHDL source files, synthesize the necessary logic and create logic that is subsequently placed and routed into the FPGA along with the **Nios II** processor and interface elements that were previously specified. The result will be a bitmap file (in the appropriate Altera format) ready for downloading to the device.

Downloading Bitmap

When the bitstream has been generated, select **Tools** -> **Programmer** to open a new programming file. Select **File** -> **Save As** and save the chain description file as ComplexFIR.cdf (make sure the "Add file to current project" option is selected).

The programming file **ComplexFIR.sof** should be visible in the programming window. If it is not, select **Add File...** and open **ComplexFIR.sof**.

Enable Program/Configure for **ComplexFIR.sof** and make sure your programming hardware (e.g., the ByteBlasterMV cable) is configured properly. Click **Start** to begin downloading the **ComplexFIR.sof** file to the target device.

Note: If you don't have the full license for **OpenCore Plus** megafunctions, then a message will pop up. Click **OK** to continue. The bitmap file with be named **ComplexFIR_time_limited.sof** instead. After the downloading is done, a **OpenCore Plus Status** message box will pop up. Don't click the **Cancel** button. Otherwise the downloaded bitmap will be reset.

🖺 Quartus II - I):/altera/ImpulseExamples/	ComplexFIR_NIO	5/Quartus/Com	plexFIR - Con	iplexFIR - <u>[</u> C
File Edit Process	ing Tools Window				
🚖 Hardware Setu	ip USB-Blaster [USB-0]				
Enable real-time	ISP to allow background programm	ning (for MAX II device	s)		
Mart Start	File	Device	Checksum	Usercode	Program/ Configure
Hu Stop	ComplexFIR_time_limited.sof	EP3C25F324	0045425D	FFFFFFFF	
Auto Detect	1				
Auto Detect					
Add File					
👺 Change File	1				

Now that the hardware is programmed, you are ready to download and run the software application on the platform.

See Also

Step 11: Running the Application on the Platform

Tutorial 2: Complex FIR Filter on the Nios II platform

1.11 Running the Application on the Platform

ComplexFIR Filter Tutorial for Nios II, Step 11

In the previous step, you programmed the FPGA device with the design you created in **Quartus** and **SOPC Builder**. Now you will use **Altera Nios II IDE** to compile the software portion of the project and run it on the development board.

Begin by starting the Nios II IDE (usually available in the Windows Start menu under altera -> Nios II EDS 8.0 -> Nios II 8.0 IDE).

Create a new project to manage the **ComplexFIR** software files. Select **File** -> **New** -> **Nios II C/C++ Application**. A **New Project** dialog box will appear.

Select the project path, target hardware, and project template as follows, using the **Browse...** buttons to locate the appropriate **Path** and **SOPC Builder System PTF File** options:

Name: ComplexFIR

Specity Location: <selected>

Location: D:\altera\ImpulseExamples\ComplexFIR_NIOS\Quartus\software\ComplexFIR

(The project path should point to the software files that were exported by CoDeveloper in Step

5.)

SOPC Builder System PTF File:

D:\altera\ImpulseExamples\ComplexFIR_NIOS\Quartus\ComplexFIRSystem.ptf (This is the system .ptf file generated by SOPC Builder in Step 9.)

CPU: cpu

Select Project Template: Blank Project

The New Project dialog box will look as follows

🚺 New Pr	ojeci		
Nios II C/(Click Finish to D:\altera\Im	C + + Application o create application « pulseExamples\Comp	n with a default system library as plexFIR_NIOS\Quartus\software\ComplexFIR\ComplexFIR	G
Name: [ComplexFIR		
Location:	D:\altera\ImpulseE	Examples\ComplexFIR_NIOS\Quartus\software\ComplexFIR	Browse
Select Tan SOPC Build CPU:	get Hardware. der System PTF File:	D:\altera\ImpulseExamples\ComplexFIR_NIO5\Quartus\ComplexFIR5ystem.ptf	Browse
Blank Pro Board Dia Count Bin Hello Free Hello Mor Hello Wor Hello Wor Memory T	ject remplace ignostics lary estanding oC/OS-II Id Id Small fest	Description Creates a blank project Details Blank Project creates an empty project to which you can add your code.	
0		< Back Next > Finish	Cancel

Click Finish to create the new project. Two new projects (ComplexFIR and ComplexFIR_syslib) should appear in the Nios II C/C++ Projects window in the Nios II IDE, as shown below.

Copy the software files that were exported in Step 4 (**co_init.c**, **ComplexFilter.c**, **Filter_sw.c** and **ComplexFilter.h**, **Filter.h**) to the **ComplexFIR** project as shown below.



The software files will appear under the **ComplexFIR** project as shown below:



Now build the project by right-clicking the **ComplexFIR** project and selecting **Build Project**. The IDE will build the **ComplexFIR_syslib** system library, which includes a driver for the Impulse C hardware module created by CoBuilder, along with the application software code in the **ComplexFIR** project.

Build Project	
Operation in prog	gress
Building ComplexFIR	
	Run in Background Cancel Details >>

Once the software has finished building, you are ready to run the application on the hardware platform. Right-click the **ComplexFIR** project and select **Run As** -> **Nios II Hardware**.

You should see printed output in the Console window as shown below:



The result tells us that with hardware acceleration, the execution of the ComplexFIR filter is 58.86 times faster than the software-only version.

Congratulations! You have successfully completed the Image Filter tutorial.

See Also

Tutorial 2: Complex FIR Filter on the Nios II platform