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

*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

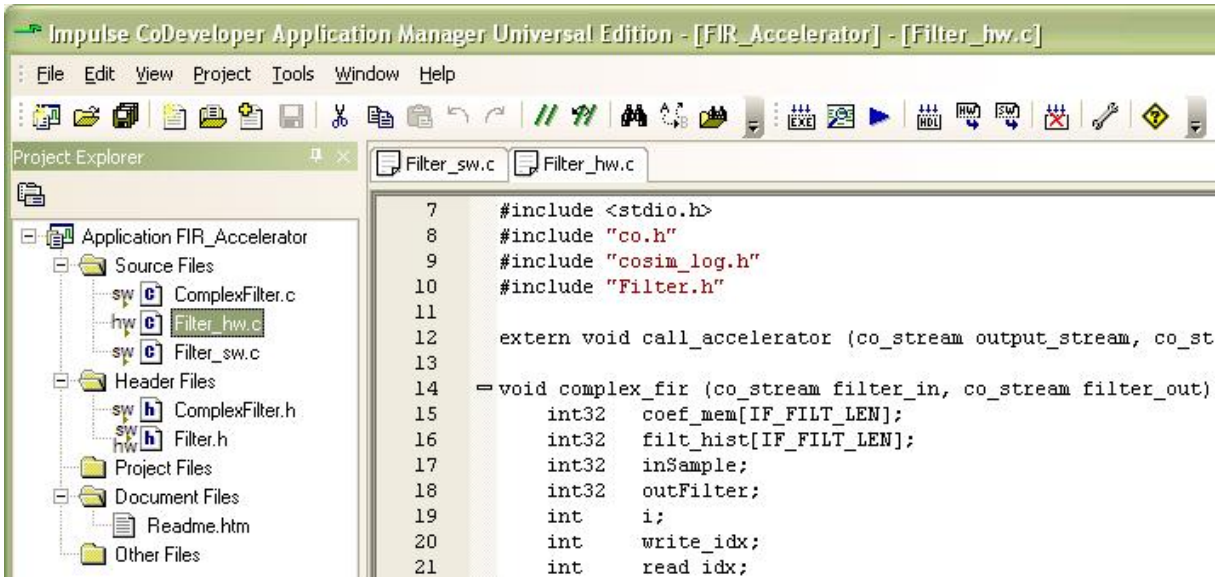
ComplexFIR Filter Tutorial for Nios II, Step 1

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:



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

Step 2: Understanding the Complex FIR Filter Example

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 coeffients 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

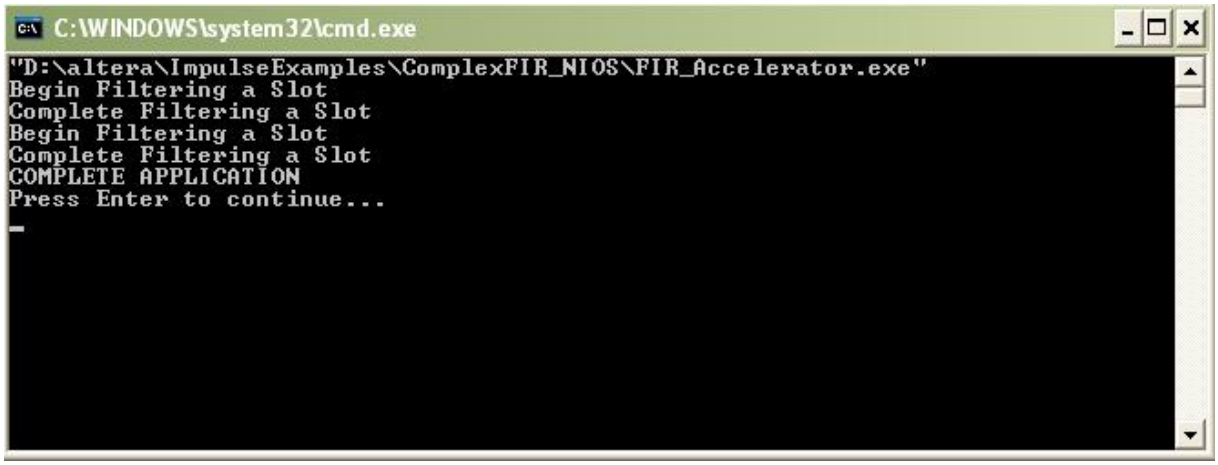
1.3 Compiling the Complex FIR Filter for Simulation

CompleFIR Filter Tutorial for Nios II, Step 3

Simulating the CompleFIR Application

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

1. 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.
2. You now have a Windows executable representing the ComplexFIR application implemented as a desktop (console) software application. Run this executable by selecting **Project** -> **Launch Simulation Executable**. A command window will open and the simulation executable will run as shown below:



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

```
#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

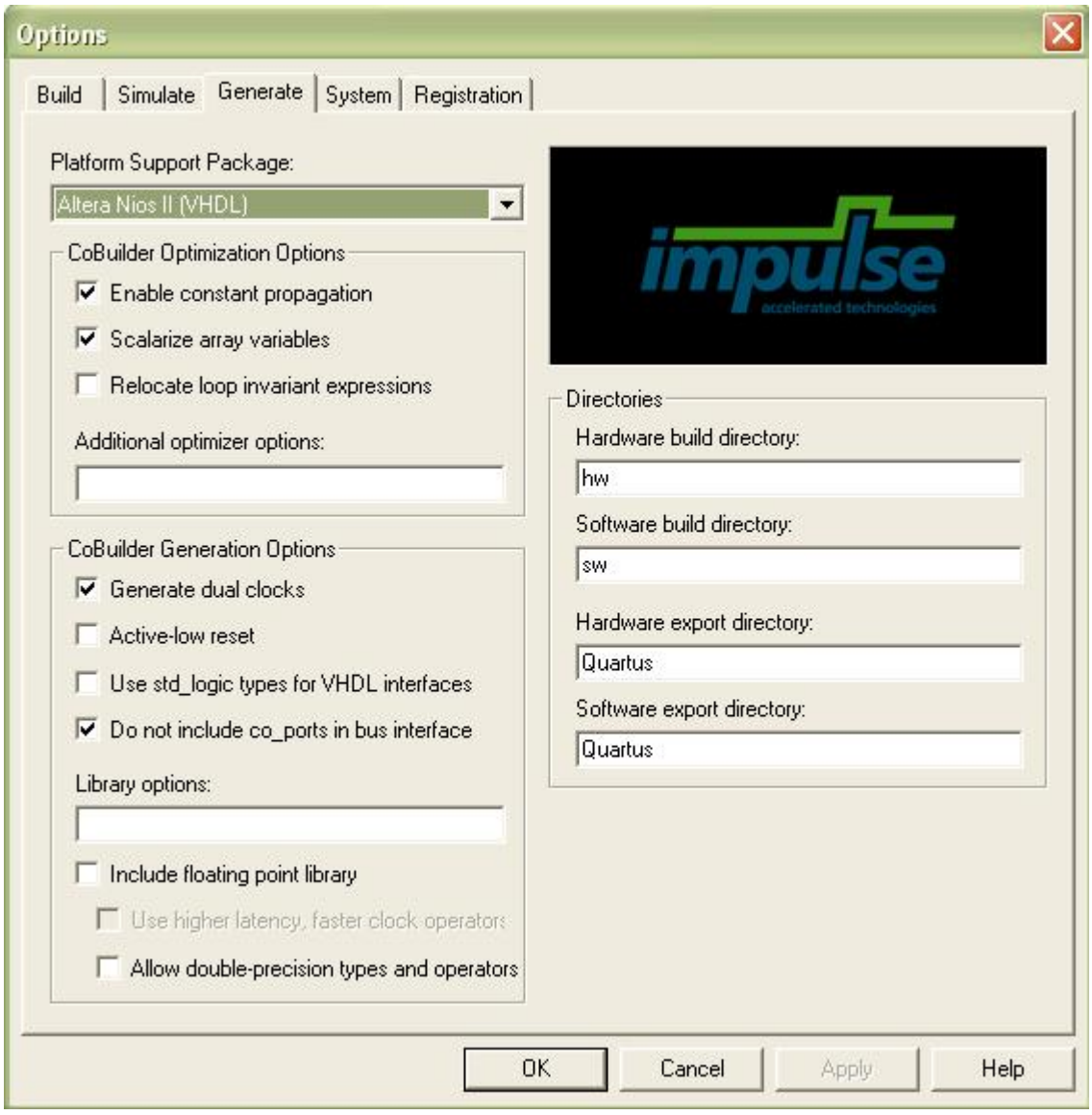
Tutorial 2: Complex FIR Filter on the Nios II platform

1.4 Building the Complex FIR Filter Example

CompleFIR Filter Tutorial for Nios II, Step 4

Specifying the Platform Support Package

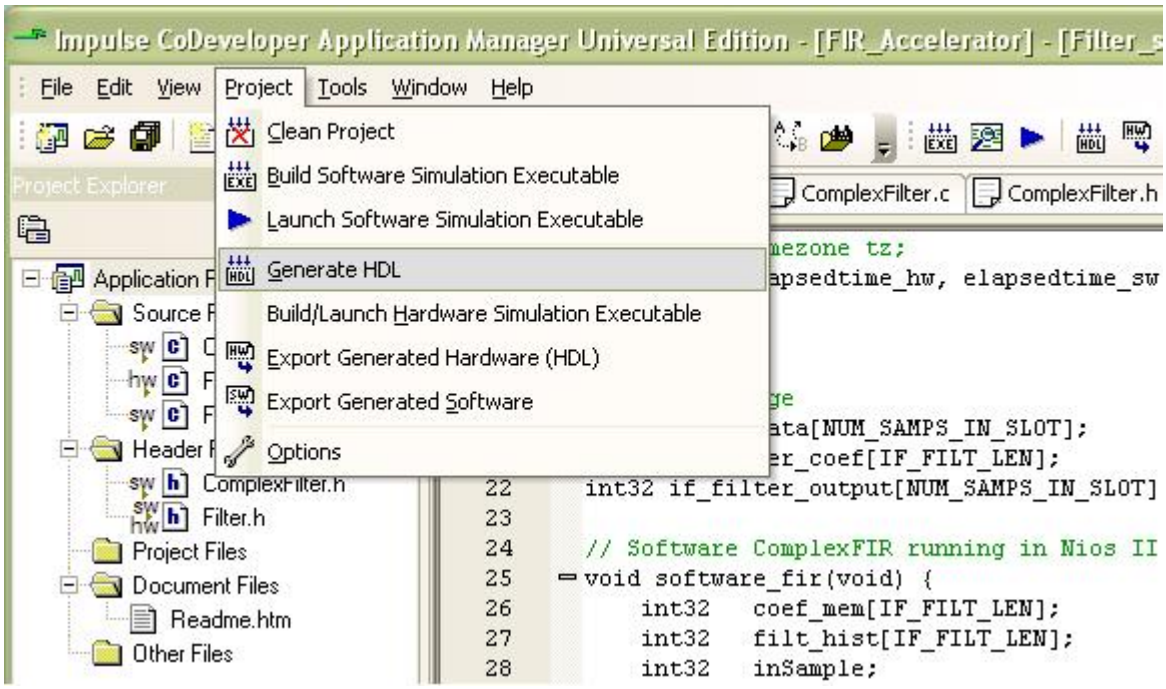
The next step, prior to compiling and generating the HDL and related output files, is to select a platform target. To specify a platform target, open the Generate Options dialog as shown below (**Project** -> **Options**, **Generate** tab):



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:



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

Tutorial 2: Complex FIR Filter on the Nios II platform

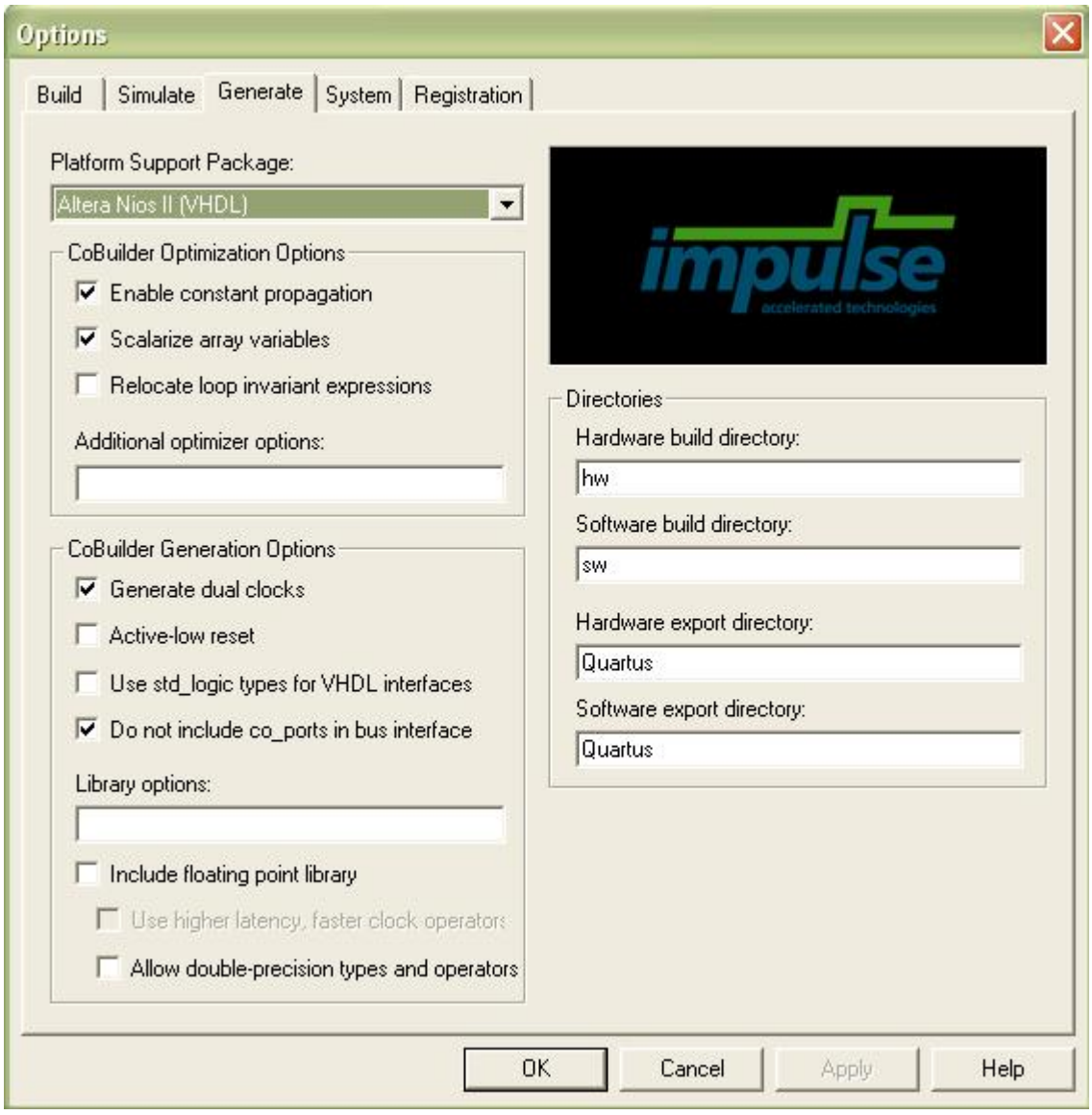
1.5 Exporting Files from CoDeveloper

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:



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.

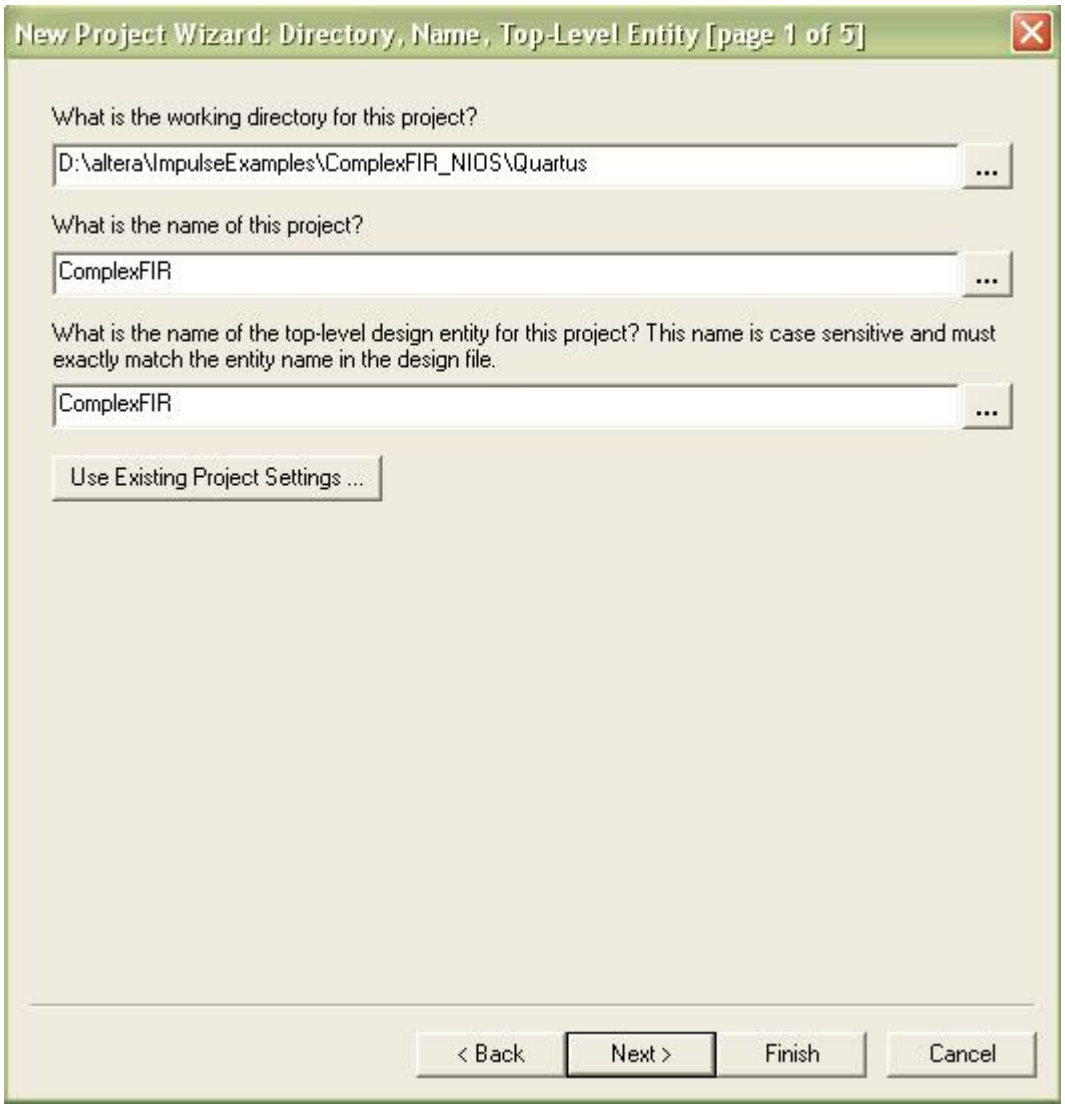
See Also

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:

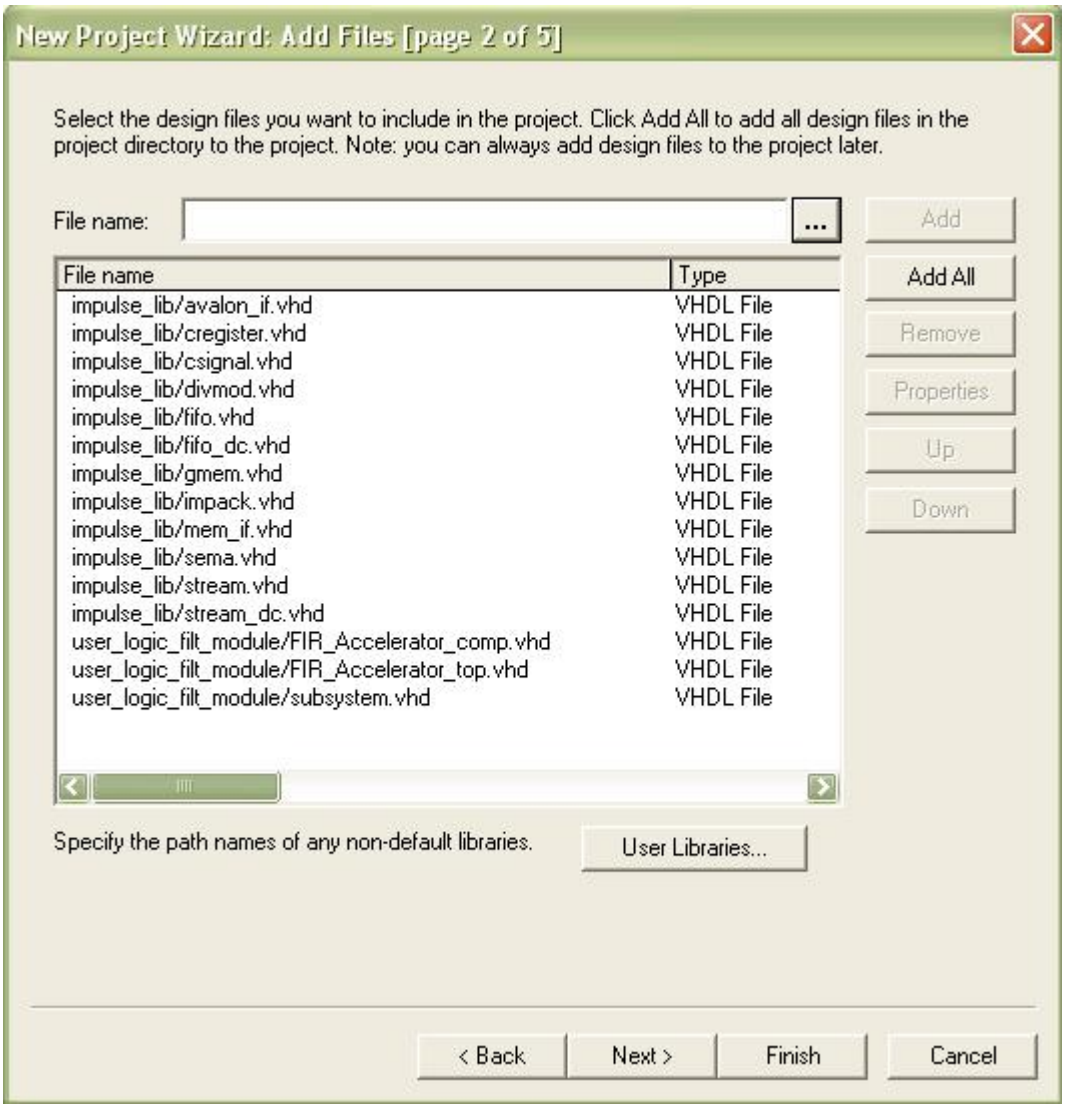


Click **Next** to move to the next page.

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:

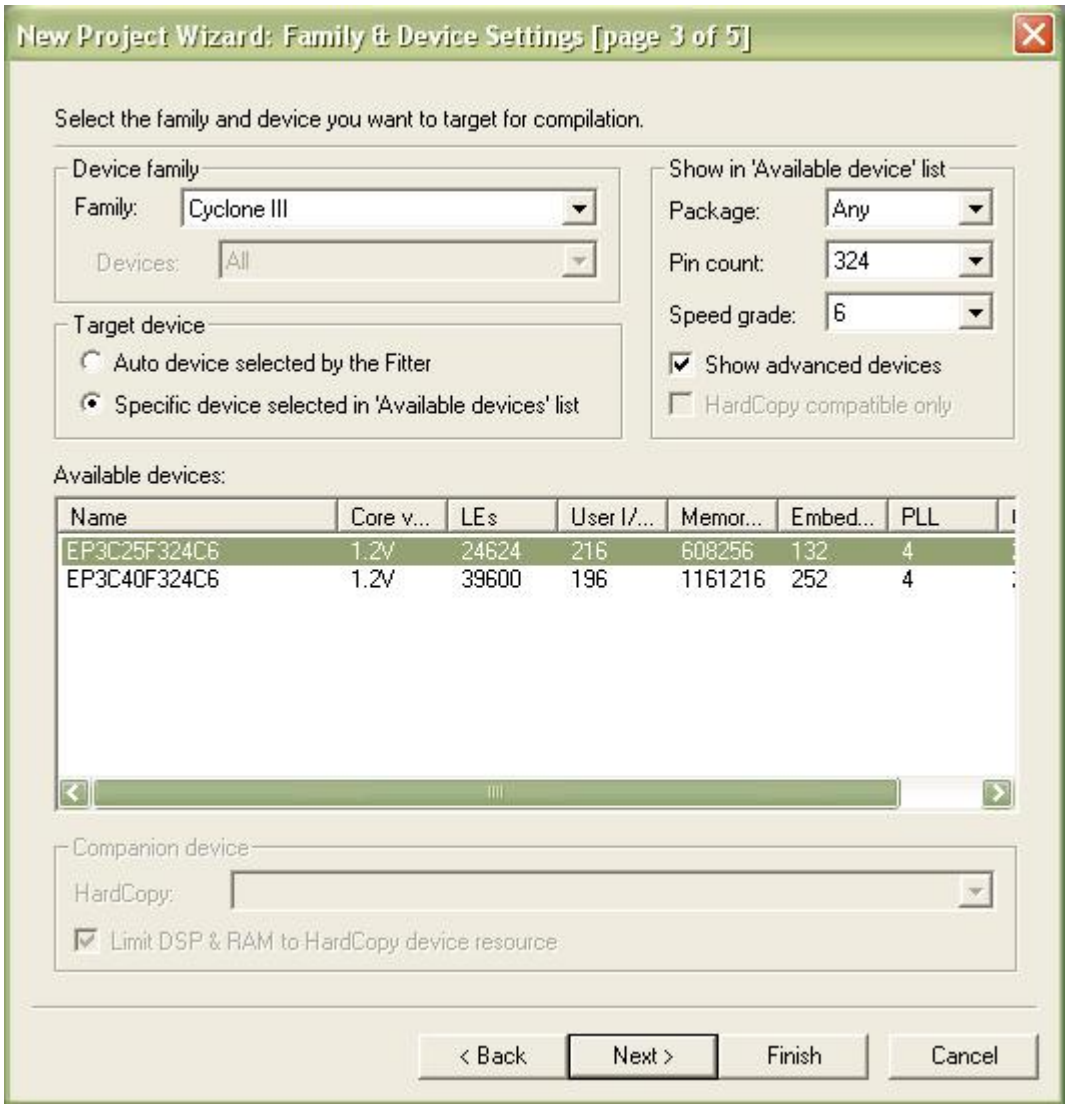
1. Block diagram file (not generated by CoBuilder): **ComplexFIR.bdf**
1. Core logic files in the **user_logic_filt_module** subdirectory: **subsystem.vhd**, **FIR_Accelerator_top.vhd** and **FIR_Accelerator_comp.vhd**
2. All .vhd files in the **impulse_lib** project subdirectory

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):



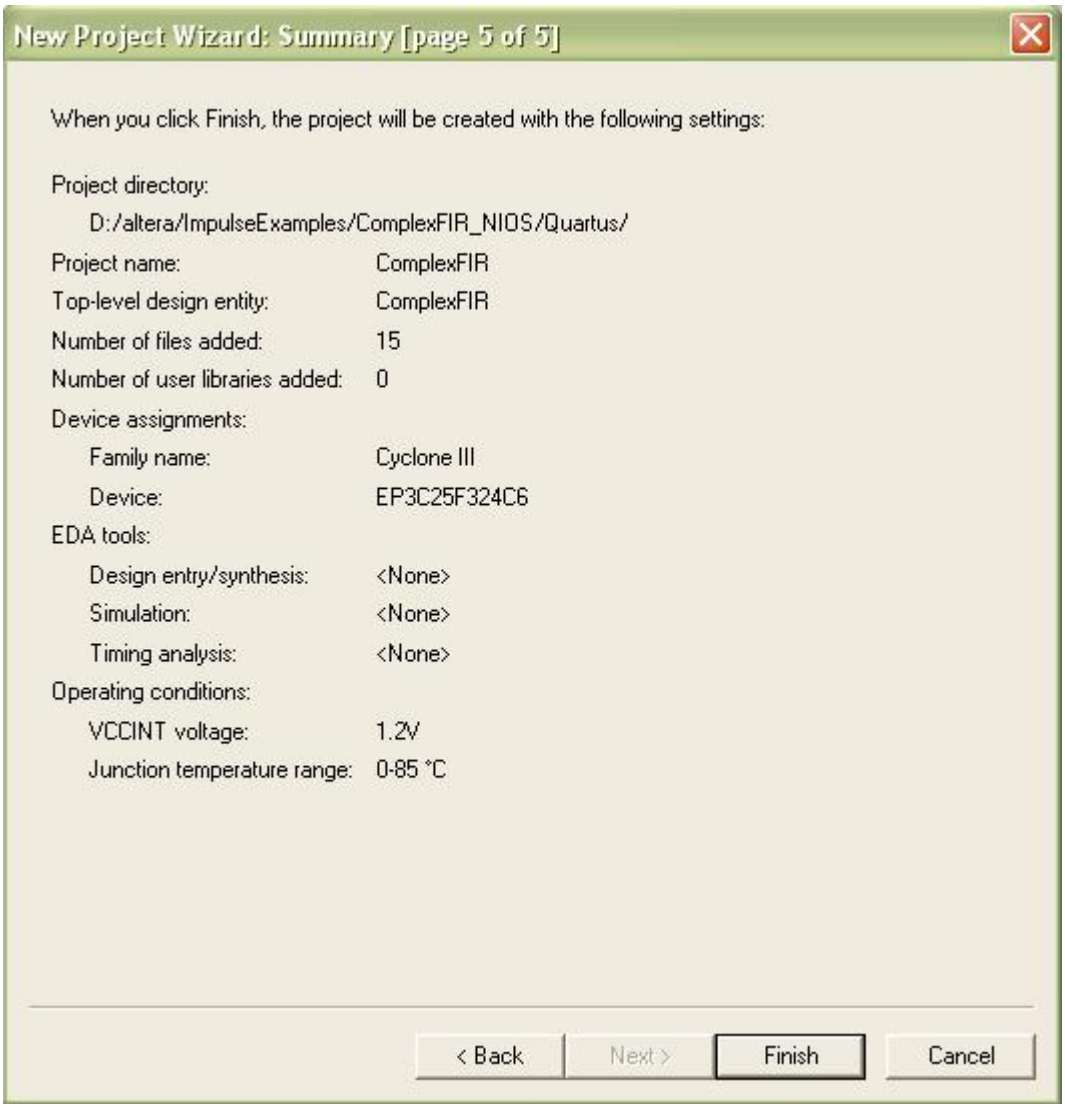
Click **Next** to proceed.

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**.



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:



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

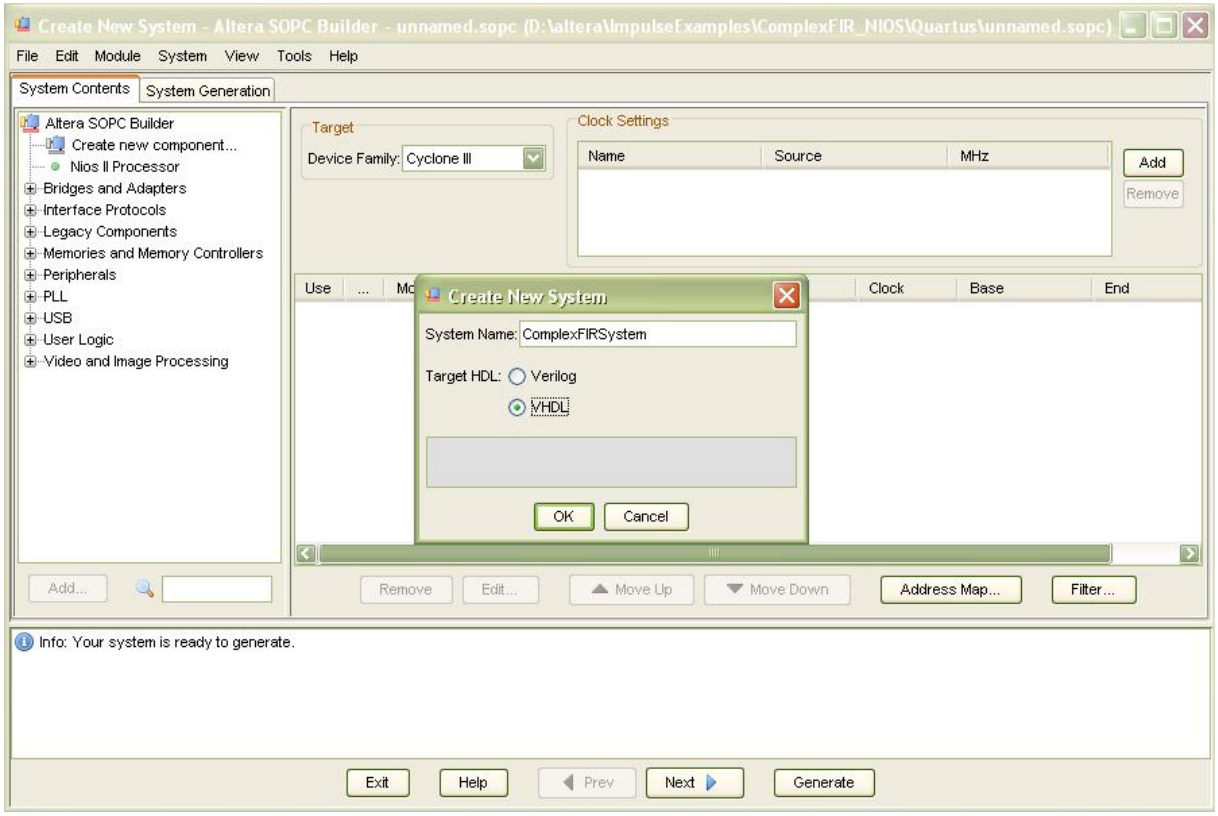
1.7 Creating the New Platform

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:



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

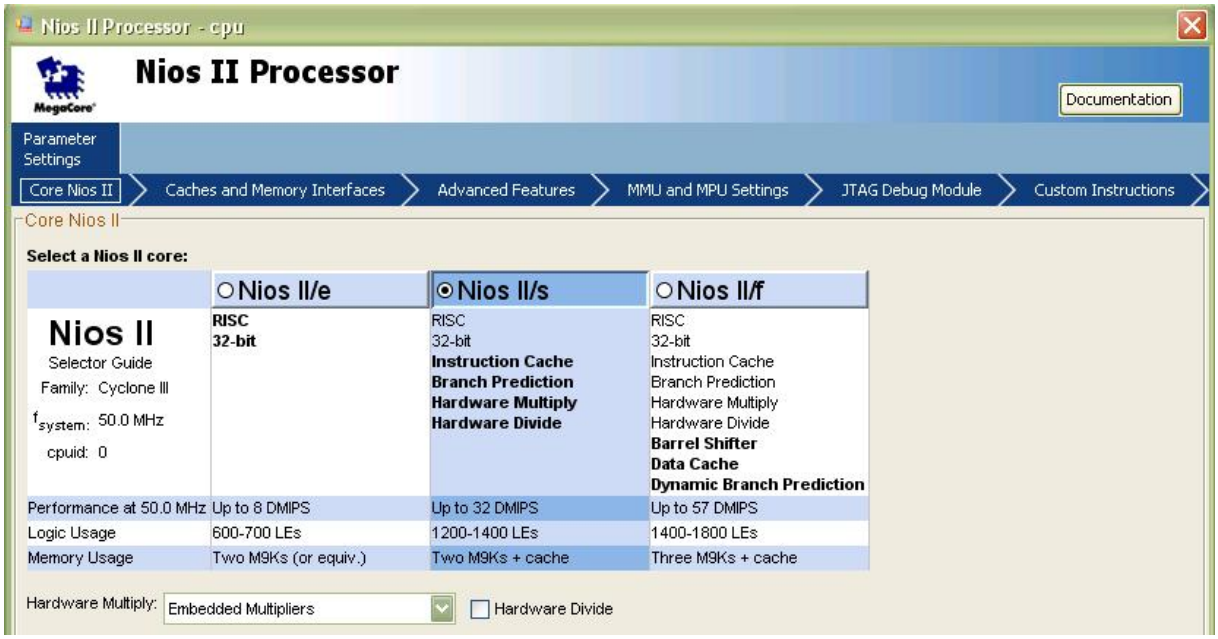
Tutorial 2: Complex FIR Filter on the Nios II platform

1.8 Configuring the New Platform

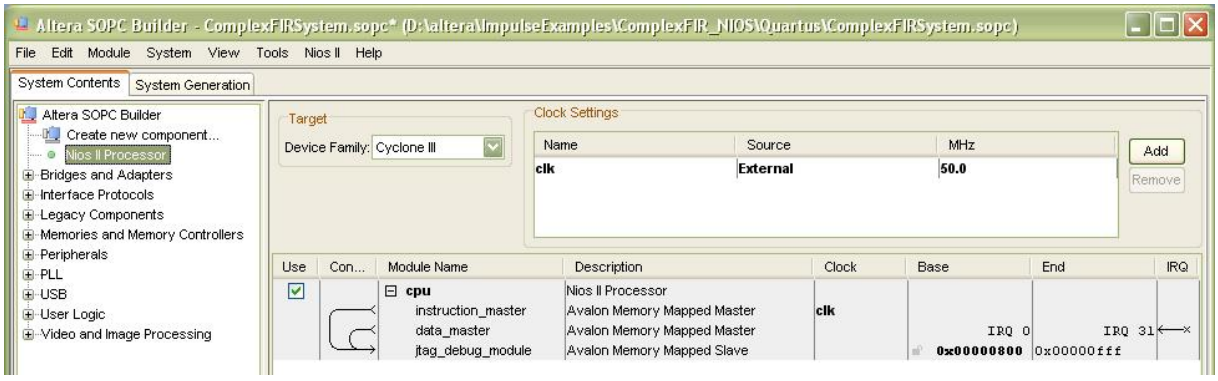
ComplexFIR Filter Tutorial for Nios II, Step 8

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:



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.



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

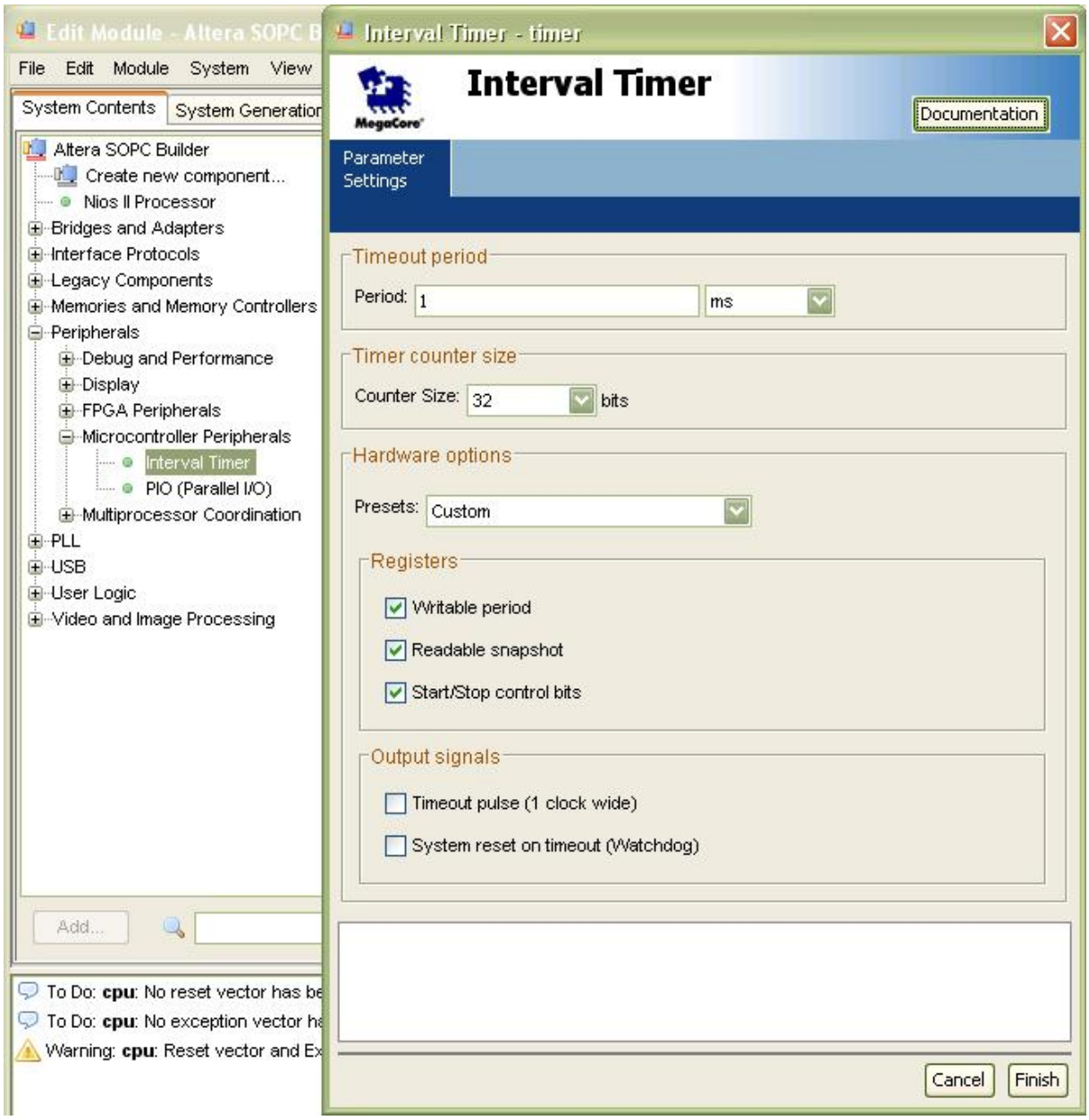
The relevant information begins with the section titled **Timer** (page 2-9) and ends with the section titled **External RAM Bus** (**Avalon Tri-State Bridge**) (page 2-13).

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.



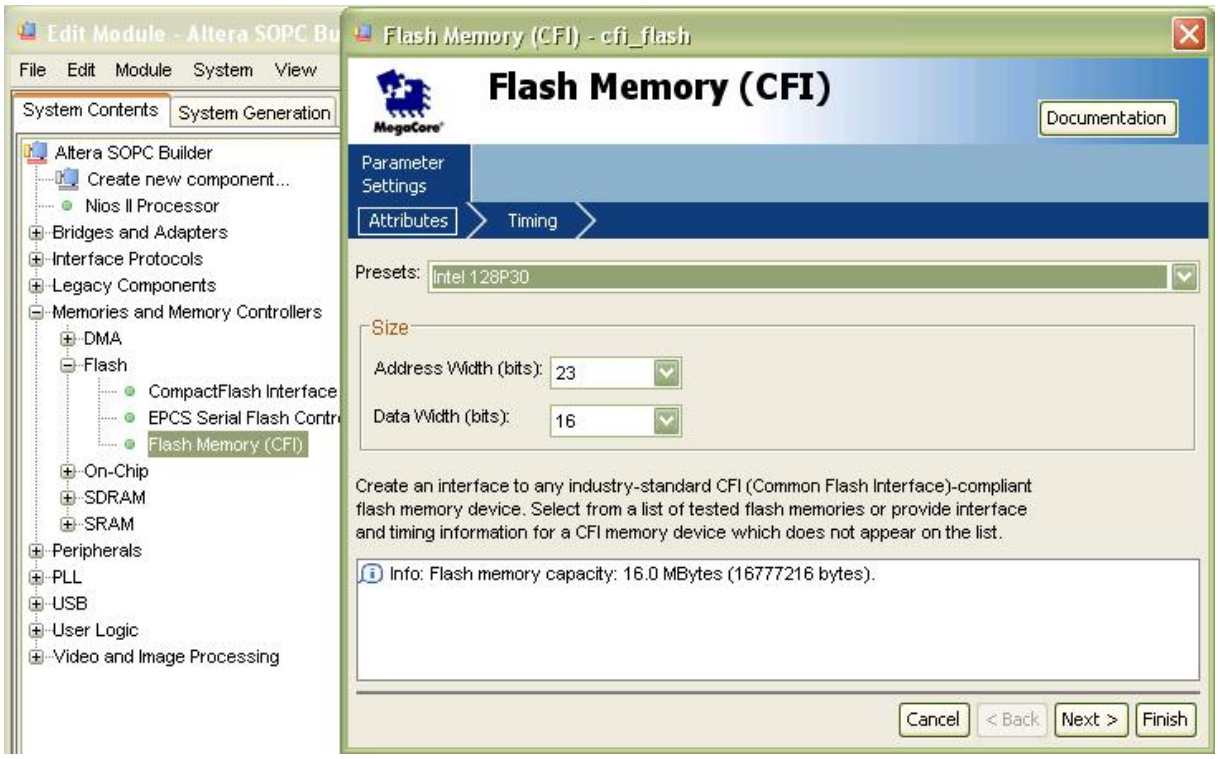
Leave the options at their default settings, and click **Finish** to add the timer to your system. You are returned to the **Altera SOPC Builder** window.

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**.



Design Entry 2

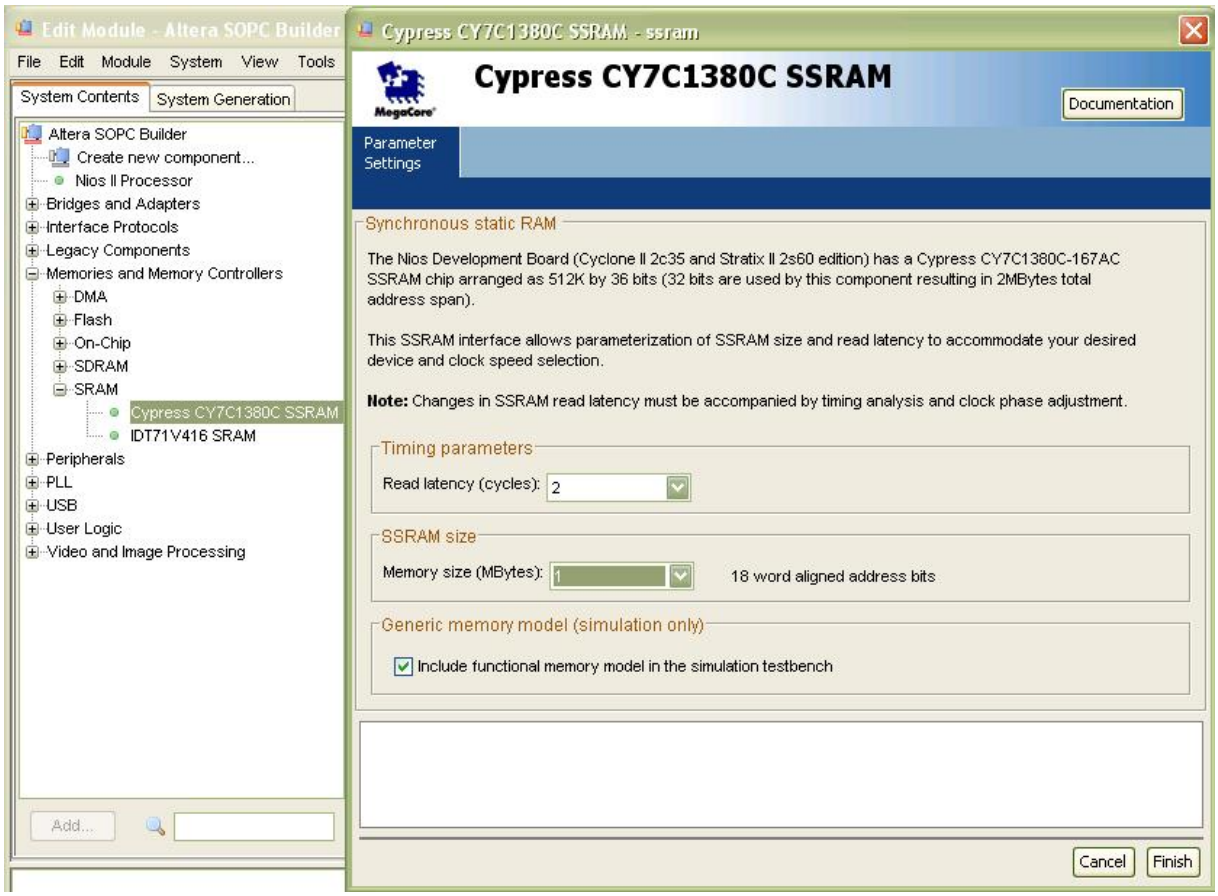
Click **Finish**. You are returned to the **Altera SOPC Builder** window.

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**:



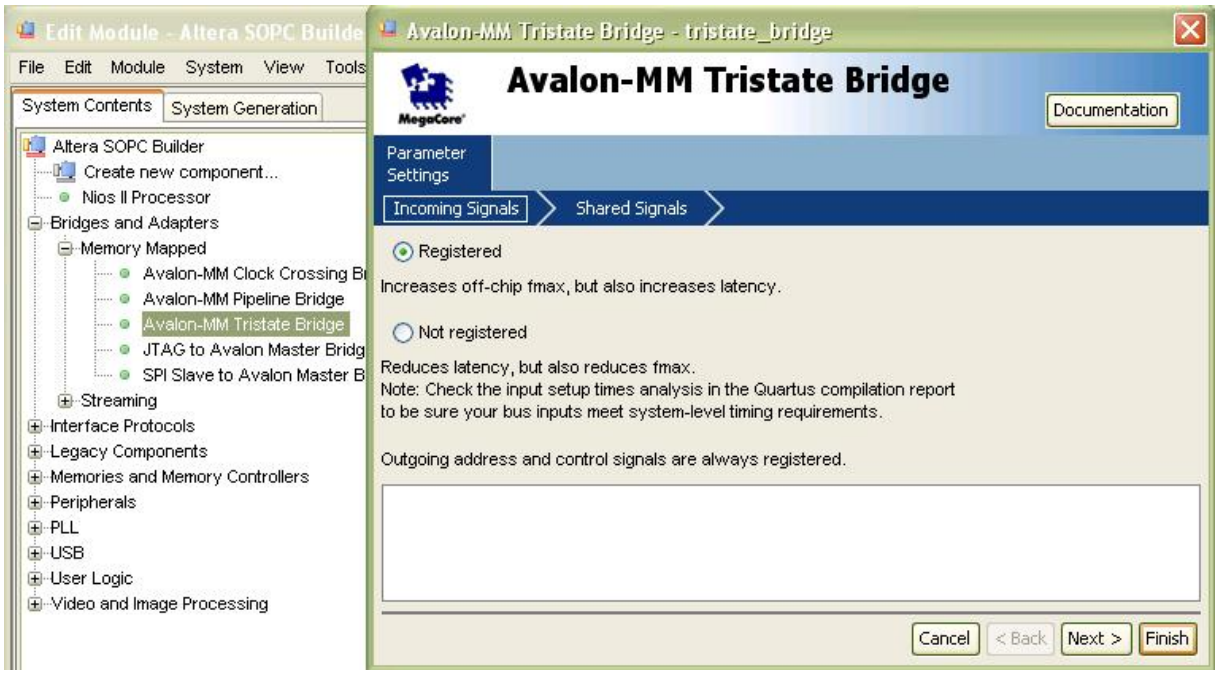
Click **Finish**. You are returned to the **Altera SOPC Builder** window.

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.

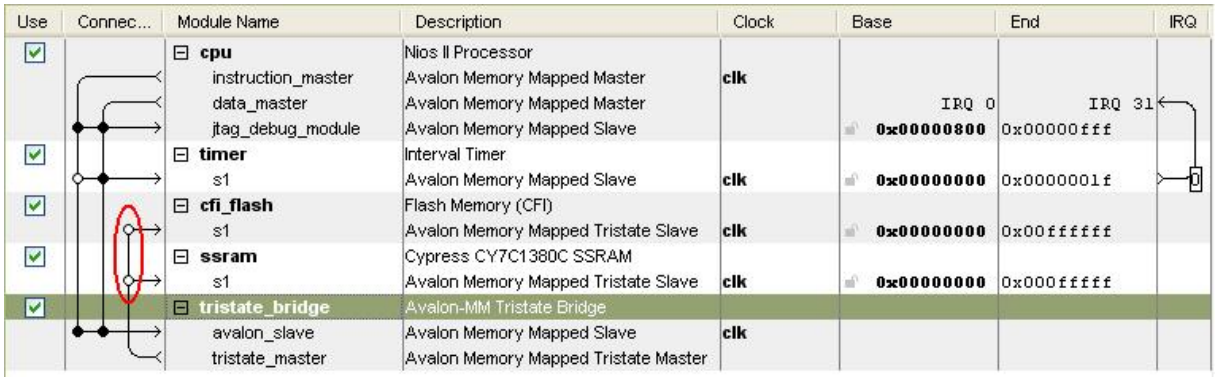


Design Entry 2

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.



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.

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:

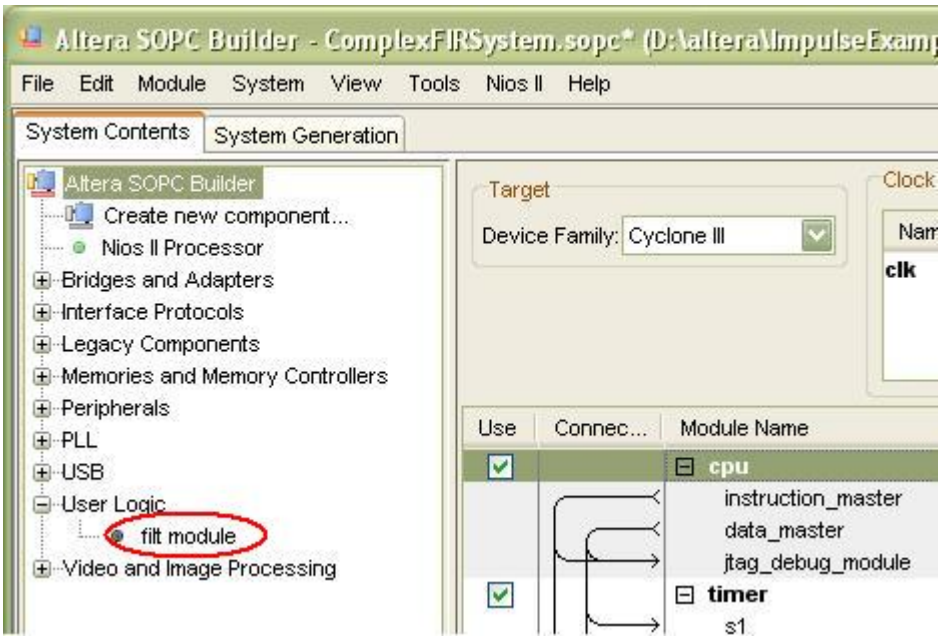
Locate **Interface Protocols** -> **Serial** -> **JTAG UART**, and double-click to add. The **JTAG UART - jtag_uart** wizard displays.

Leave all options at their default settings.

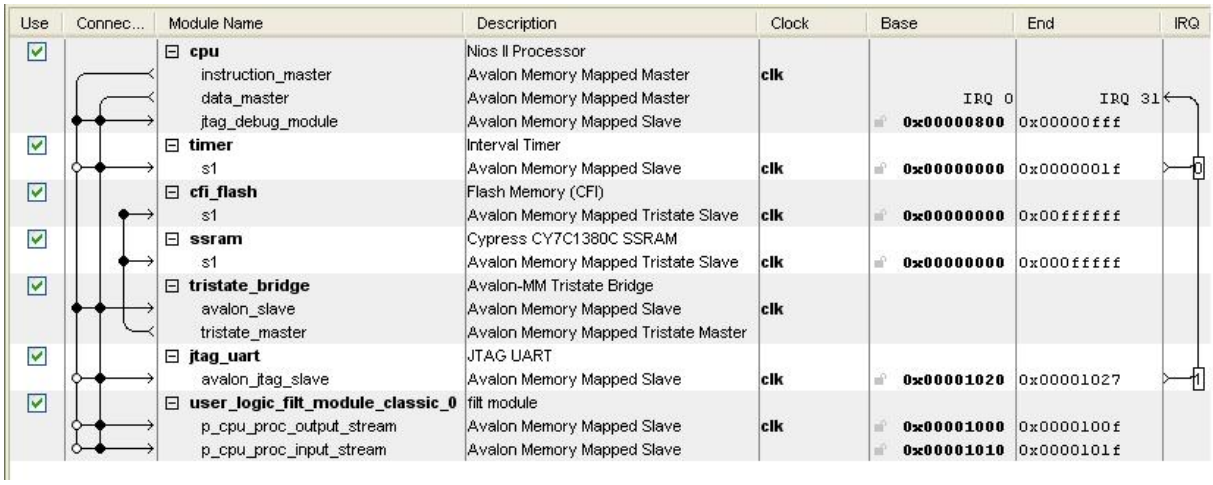
Click **Finish**. You are returned to the **Altera SOPC Builder** window.

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**:



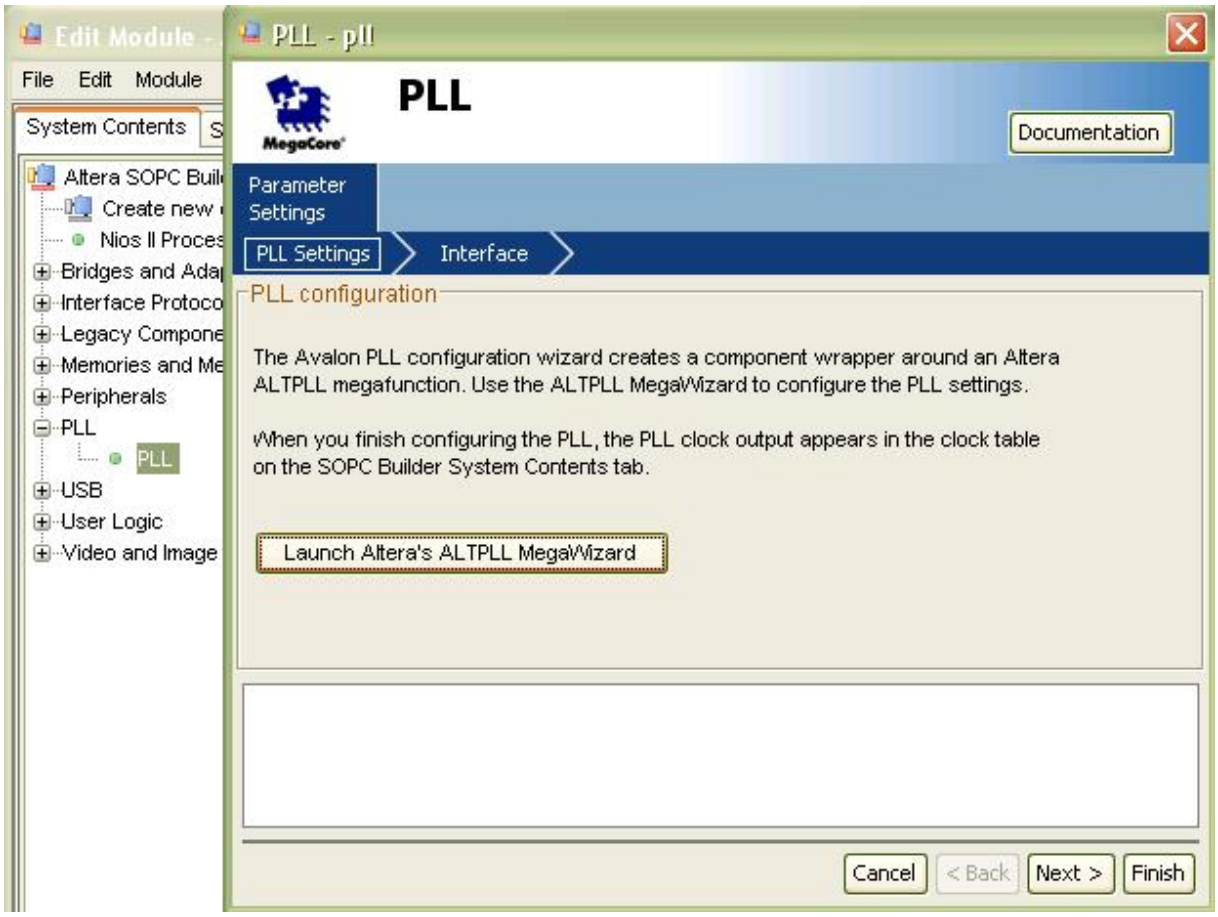
The **ComplexFIR** hardware process, **user_logic_filt_module_classic_0** module, will be connected to the shared Avalon data bus automatically. The system module listing will appear as shown below:



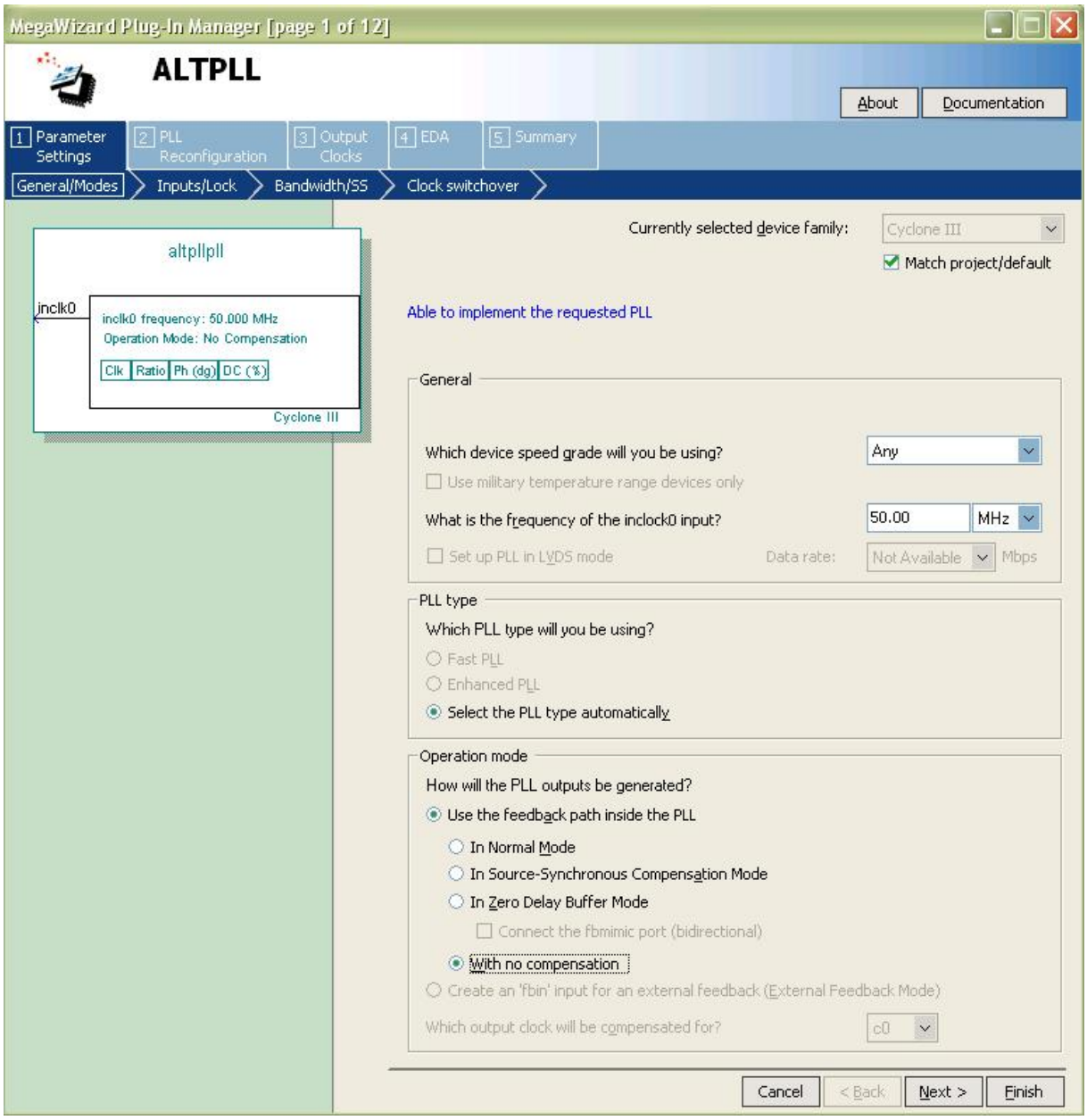
PLL

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**:

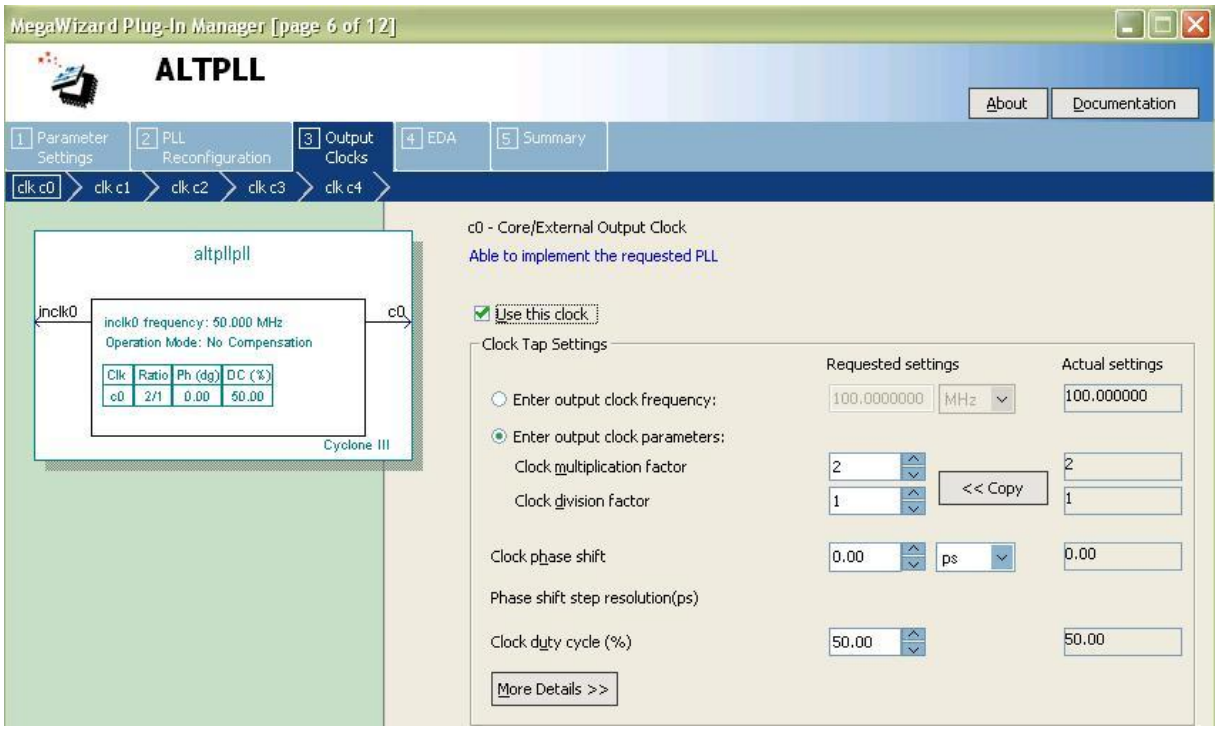


Click **Launch Altera's ALTPLL MegaWizard** to continue. The **MegaWizard Plug-In Manager** will appear as shown below:

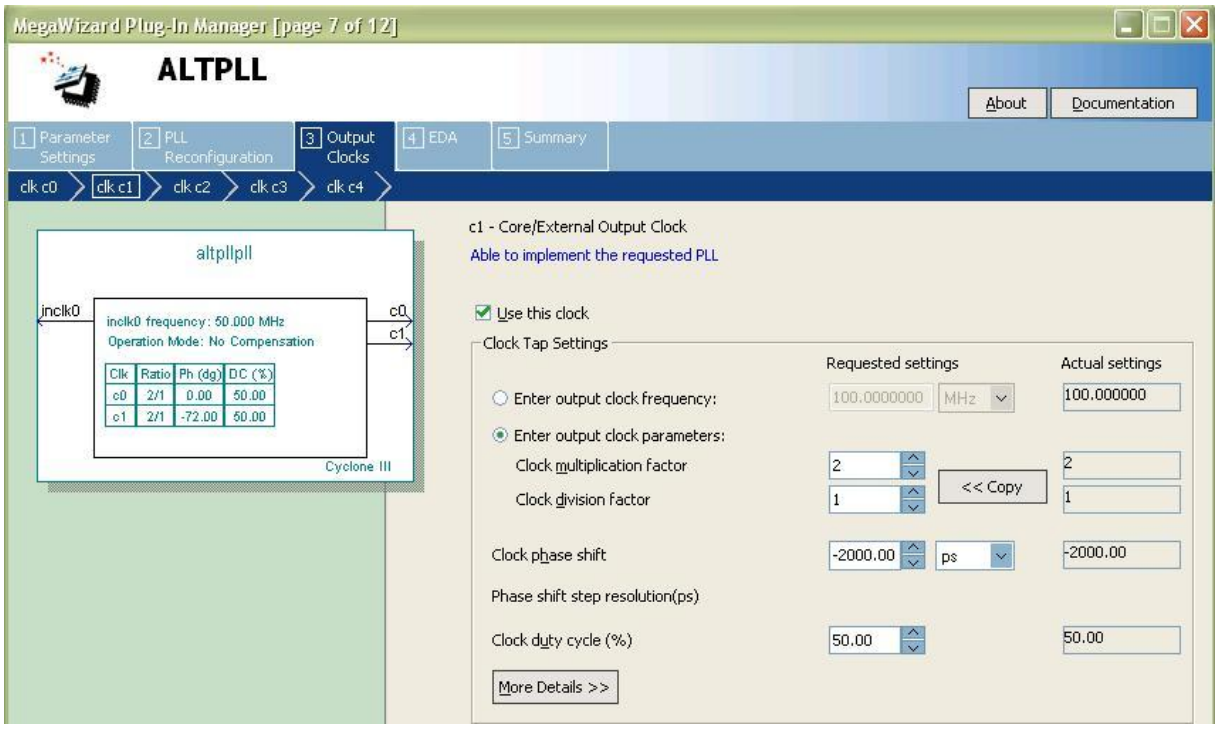


In the **Operation mode** box, check **With no compensation** item. Click **Next** to continue.

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**.

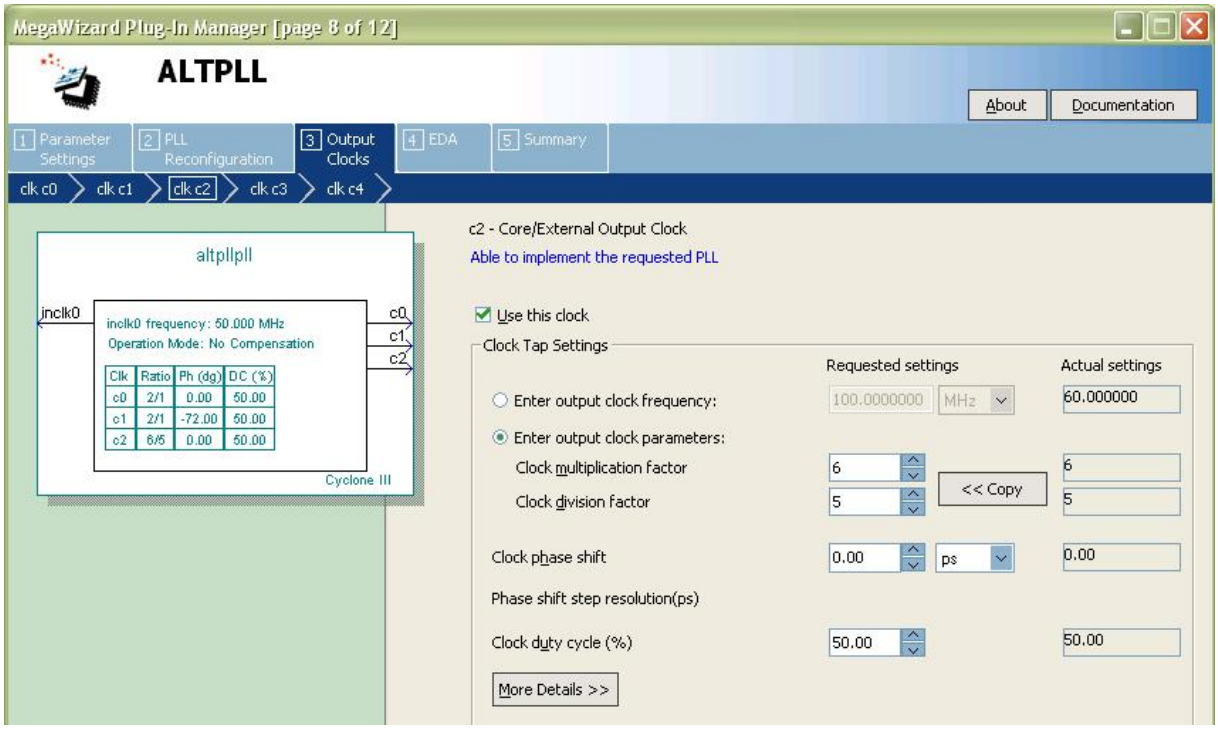


Click **Next** to continue. In **page 7** of the wizard, check **Use this clock** to activate clock **c1**. Enter **2** in the **Clock multiplication factor** box. In the **Clock phase shift** box, enter **-2000.00 ps**. This will create a clock **c1** of **100 MHz**.

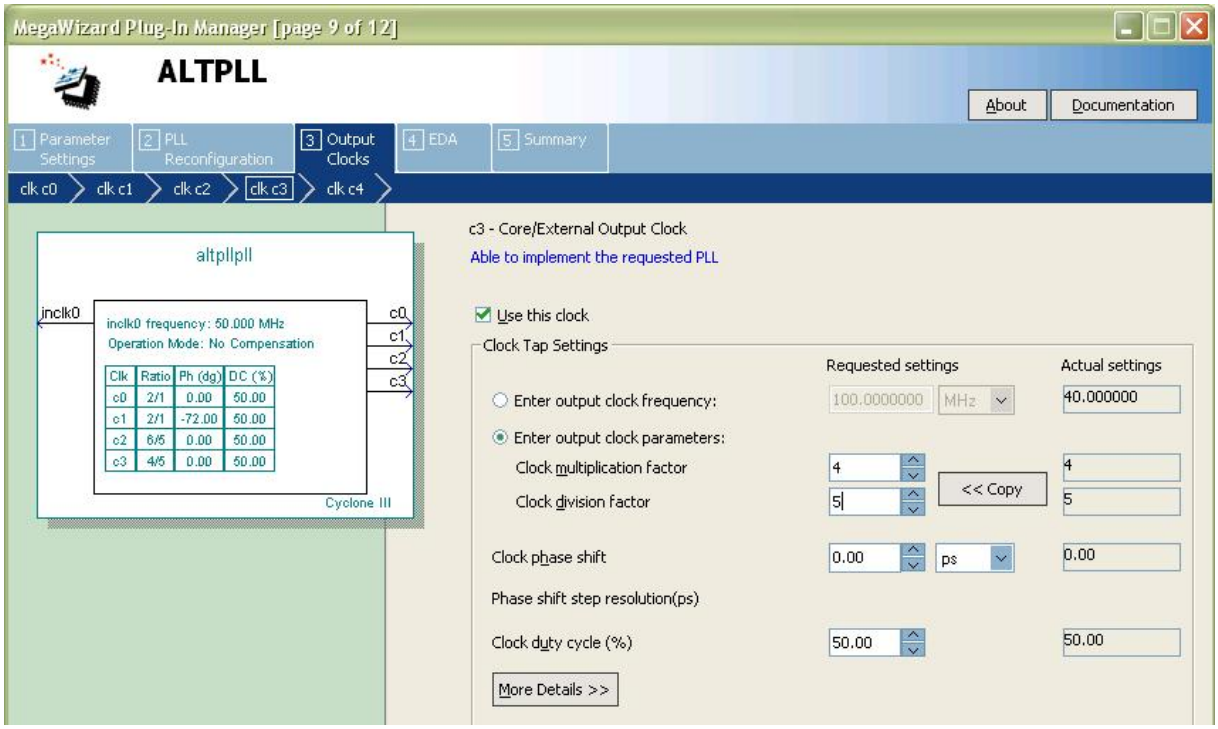


Click **Next** to continue. In **page 8** of the wizard, check **Use this clock** to activate clock **c2**. Enter **6** in the **Clock multiplication factor** box, and **5** in the **Clock division factor** box. This will create a clock

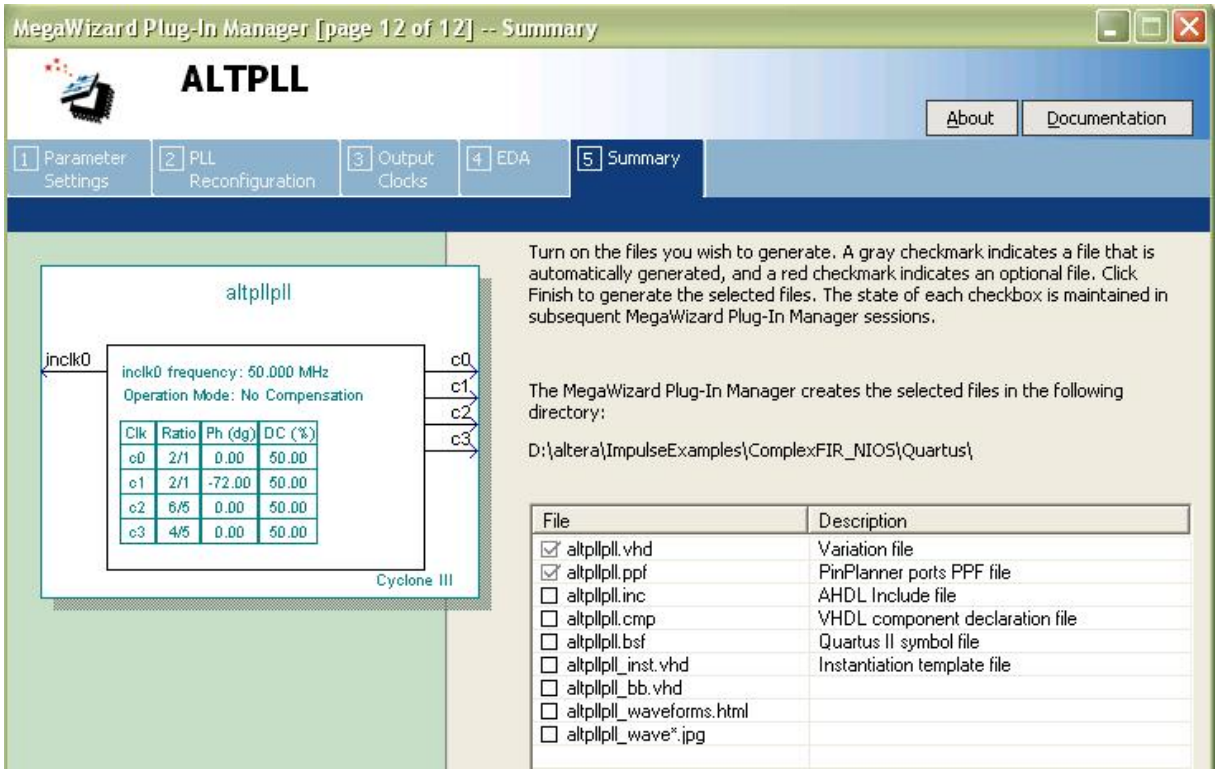
c2 of **60 MHz**.



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**.



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



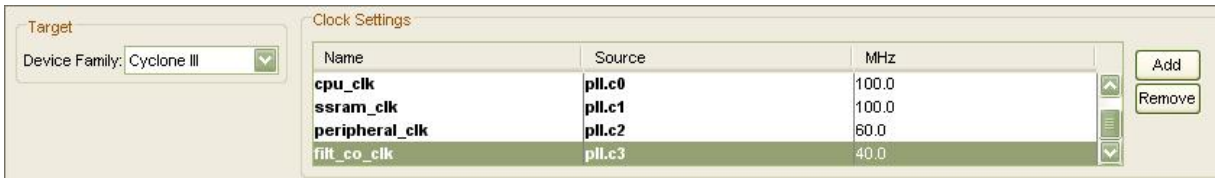
Click **Finish** again to accept the PLL settings, and then click the **Finish** button on the **PLL - pll** dialog box. A **pll** module will be added to the system.

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**

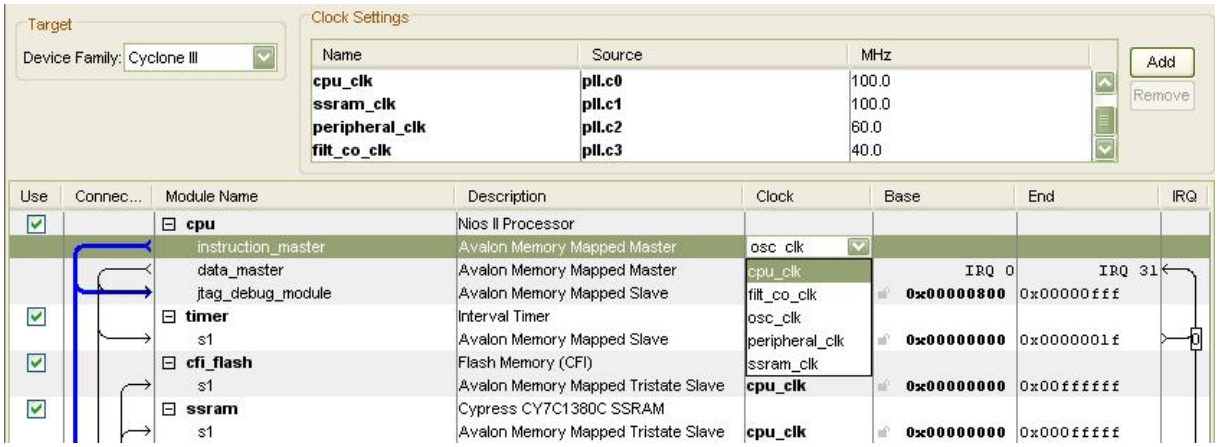
The **Clock Settings** in the **SOPC Builder** will appears as shown below:



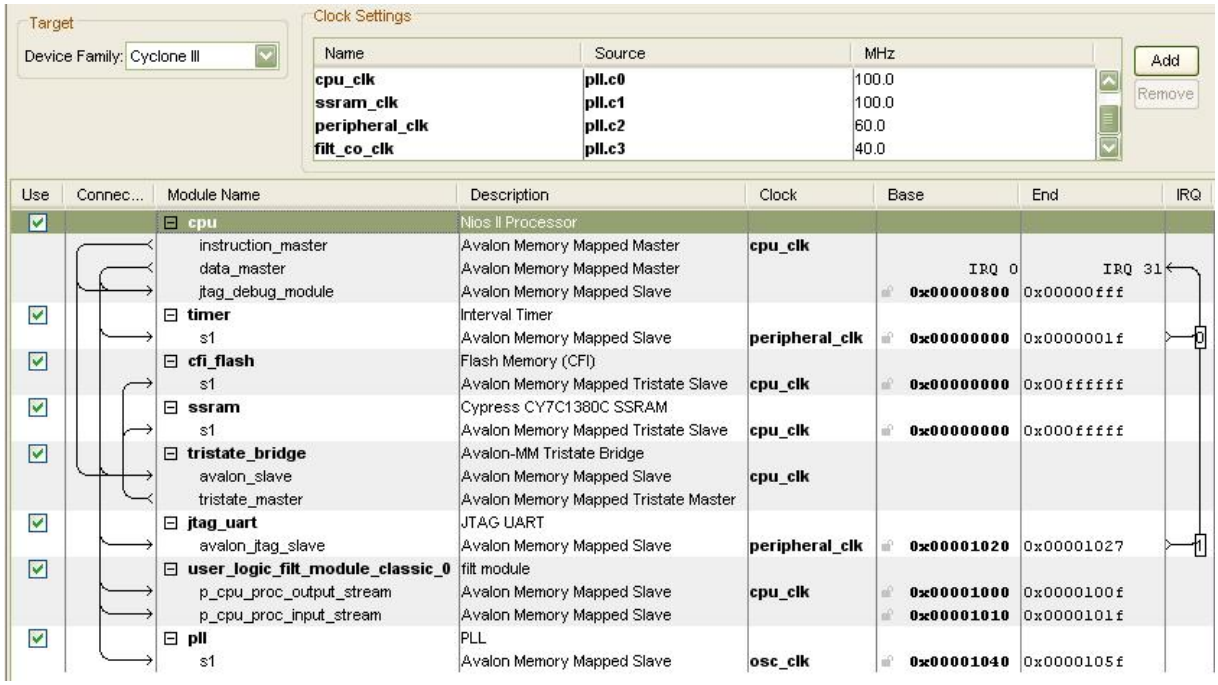
Next, change the **Clocks** for each module to the following settings:

- **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:

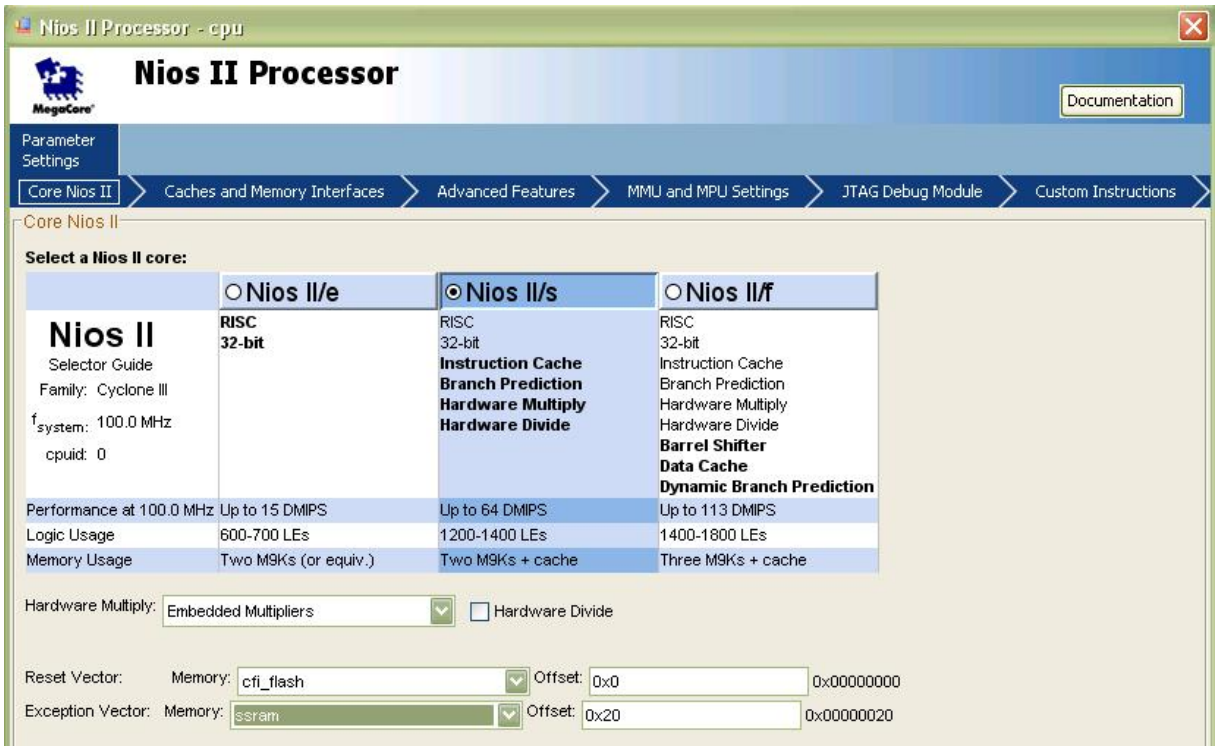


After this is done, the modules and their associated names and bus connections in the **SOPC Builder** should appear as below:



Setting "More 'cpu' Settings"

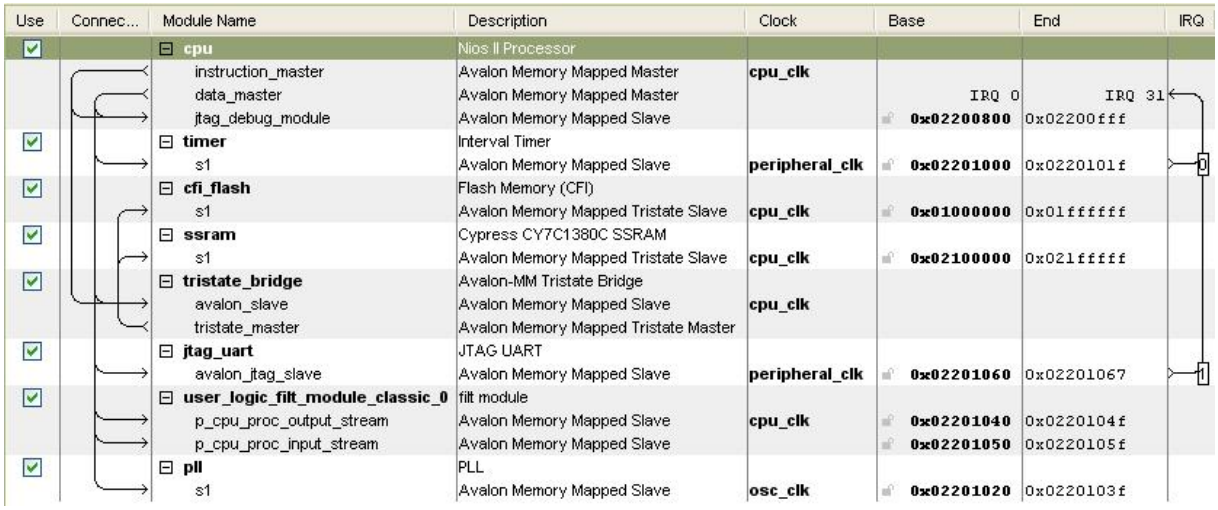
Now double-click the **cpu** module to edit the **Nios II Processor - cpu** settings. Change the **Reset Vector Memory** to **cfi_flash**, and the **Exception Vector Memory** to **ssram** as shown:



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:



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

Your new Nios II platform is ready for system generation.

See Also

Step 9: Generating the System

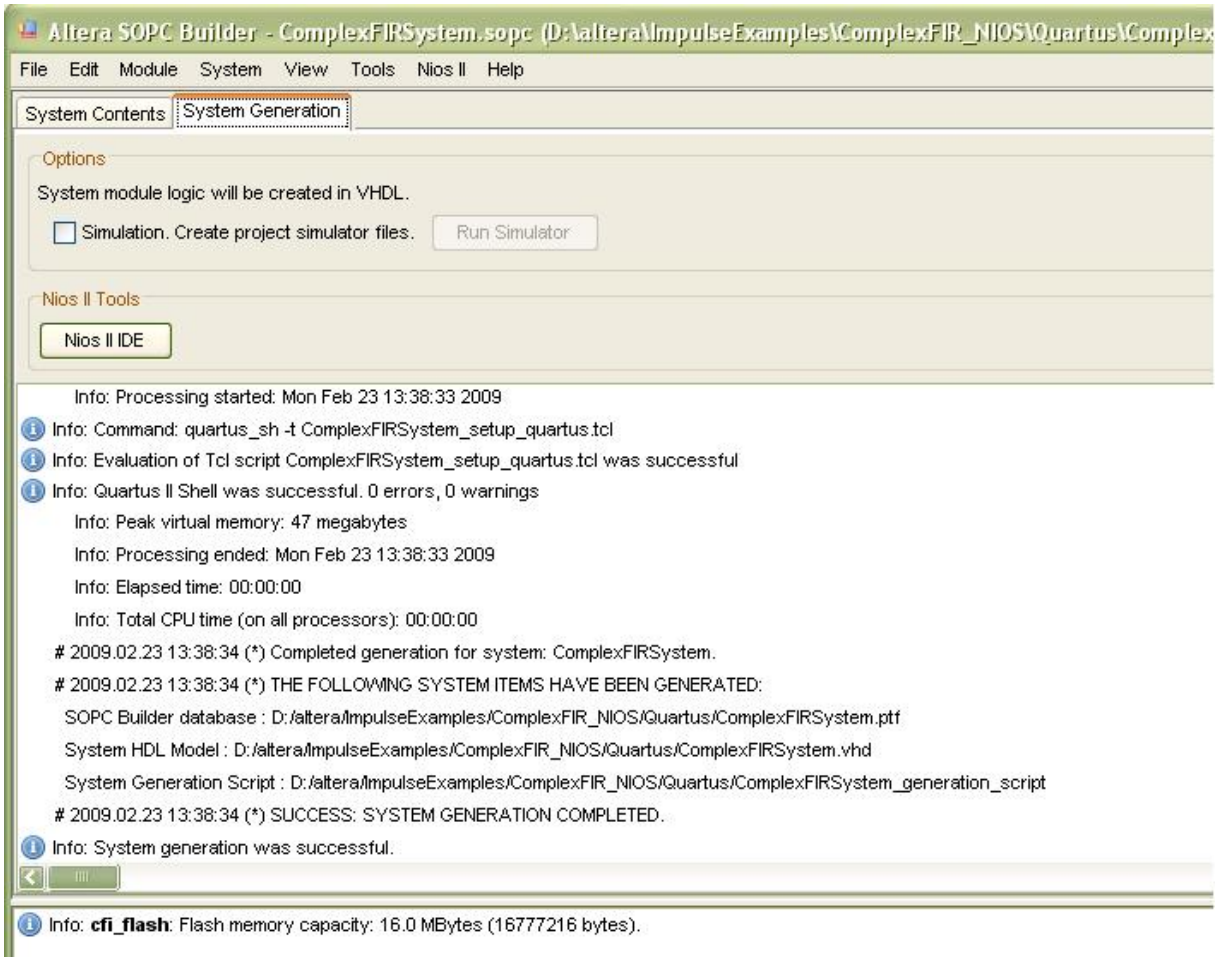
Tutorial 2: Complex FIR Filter on the Nios II platform

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**.

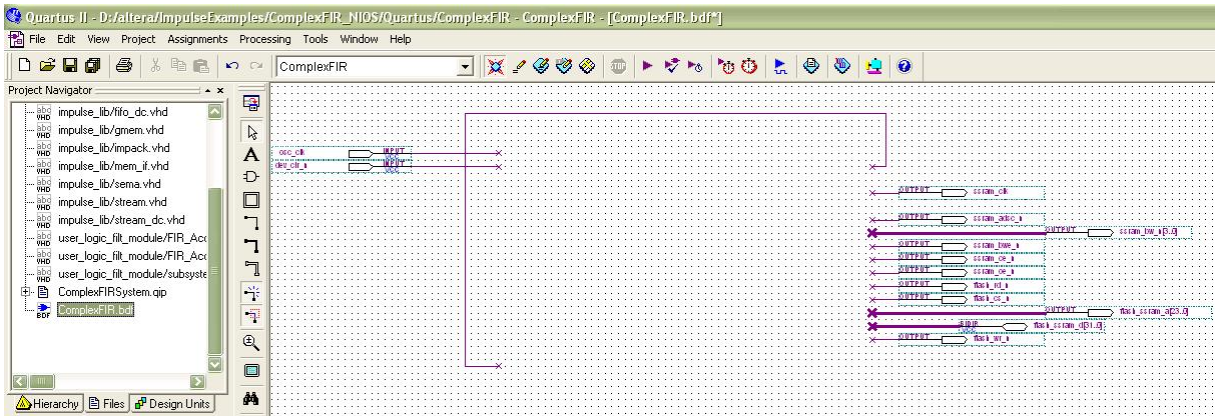
Click **Generate** on the bottom of the **SOPC Builder** window to generate the system. The **SOPC Builder** will automatically switch to the **System Generation** tab and display generation information.Make sure the **Simulation** option is unchecked to save time. This process may take several minutes.



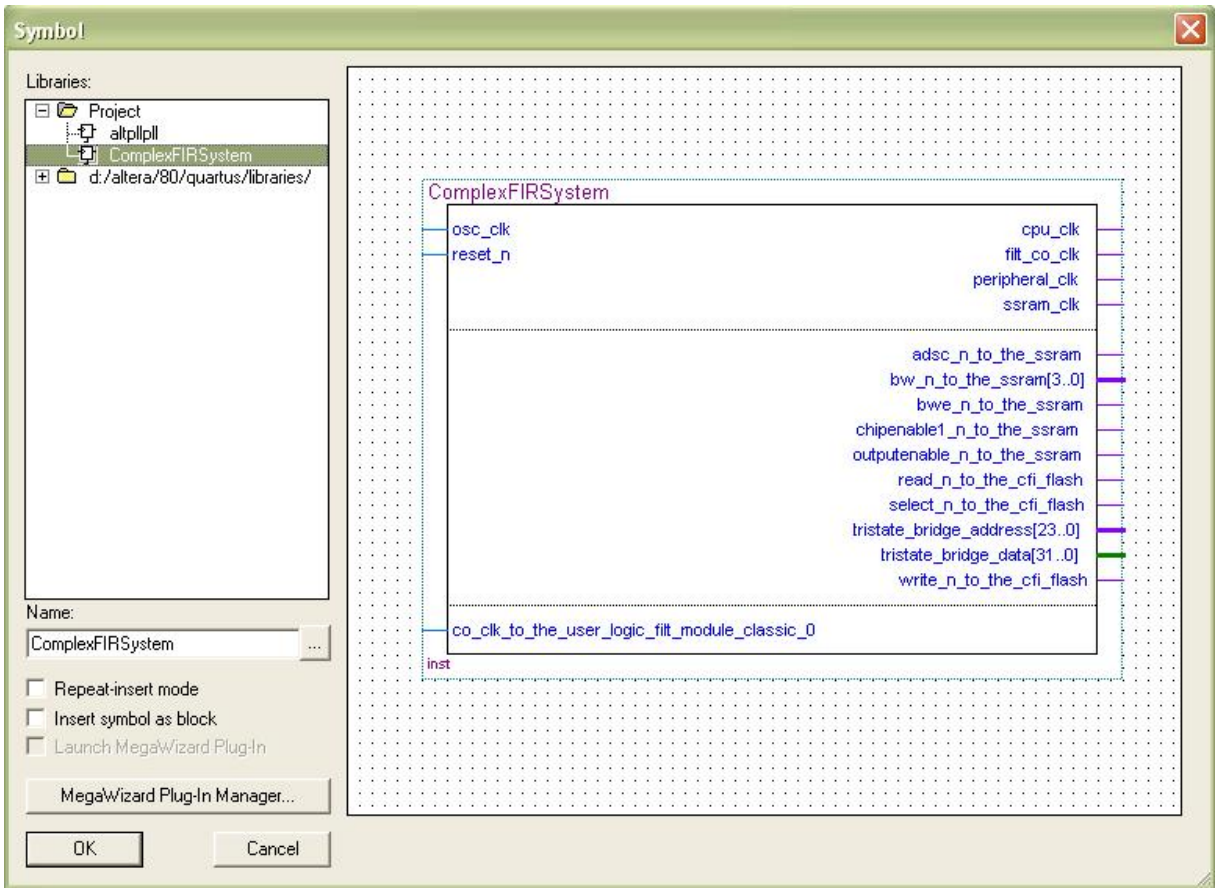
When generation is complete you may exit **SOPC Builder** and return to **Quartus**.

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.

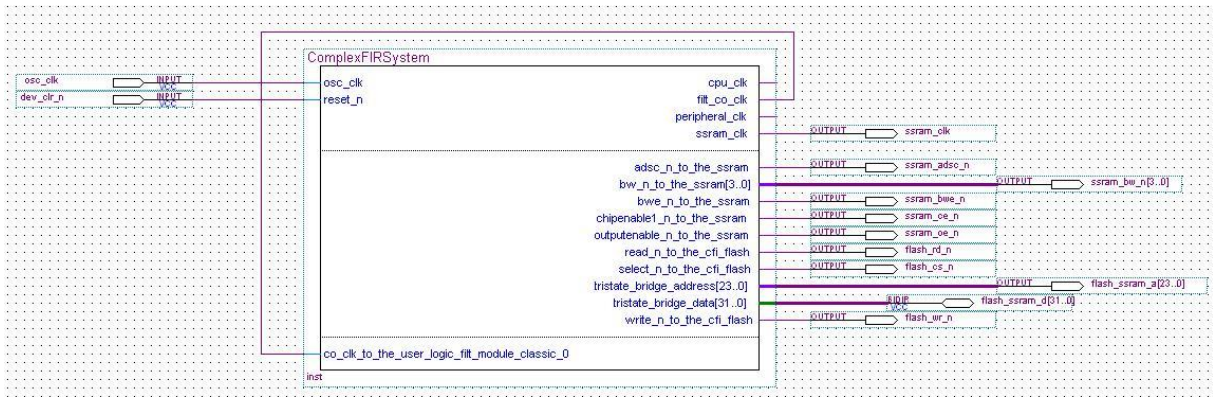
To begin, open the block diagram file **ComplexFIR.bdf** by selecting the **Files** tab in the **Project Navigator** window and double-clicking **ComplexFIR.bdf**. The block diagram file contains input and output pins to be connected to the **ComplexFIRSystem** symbol as shown below:



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:

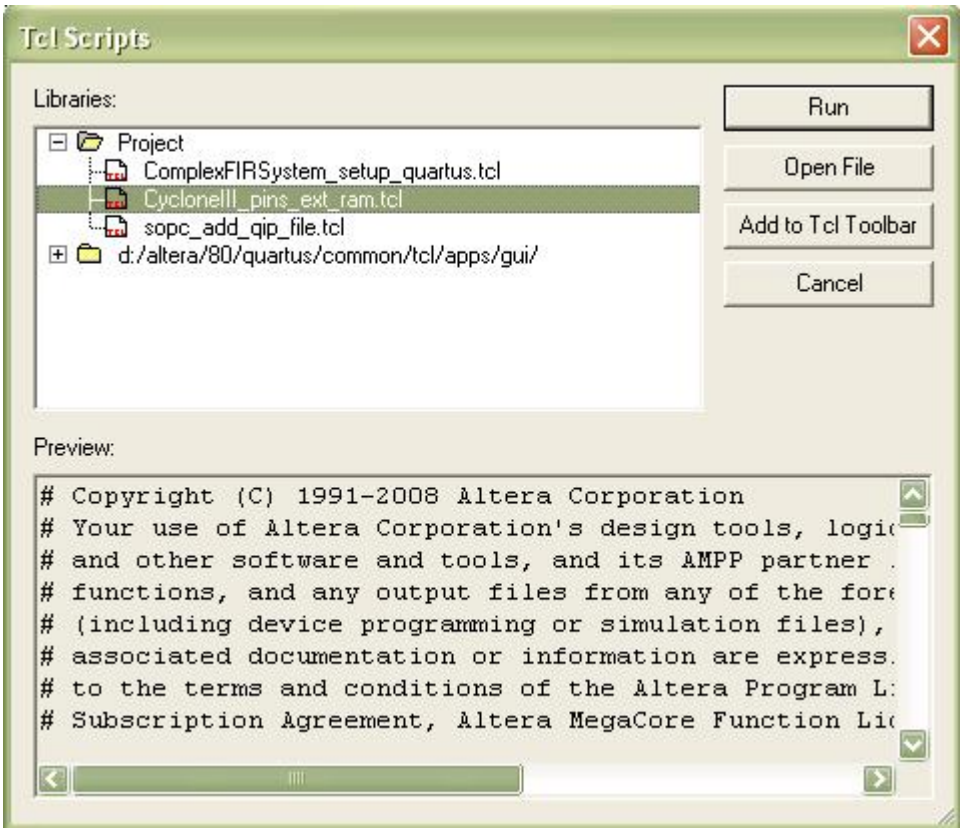


Click **OK**. A symbol outline appears attached to the mouse pointer. Align the outline with the pins on the block diagram and click once to place the symbol as shown:



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:



Select **CycloneIII_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

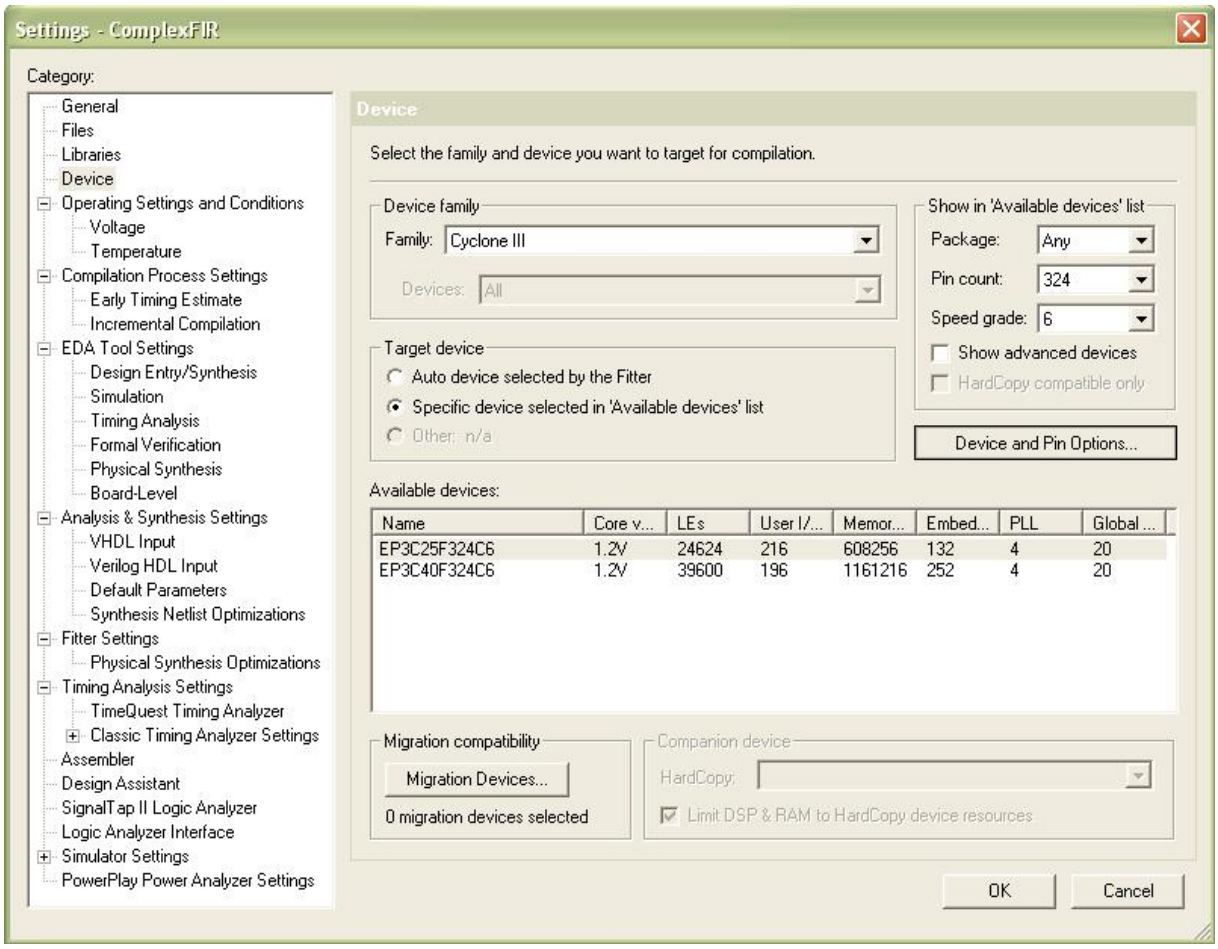
platform.

- 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.

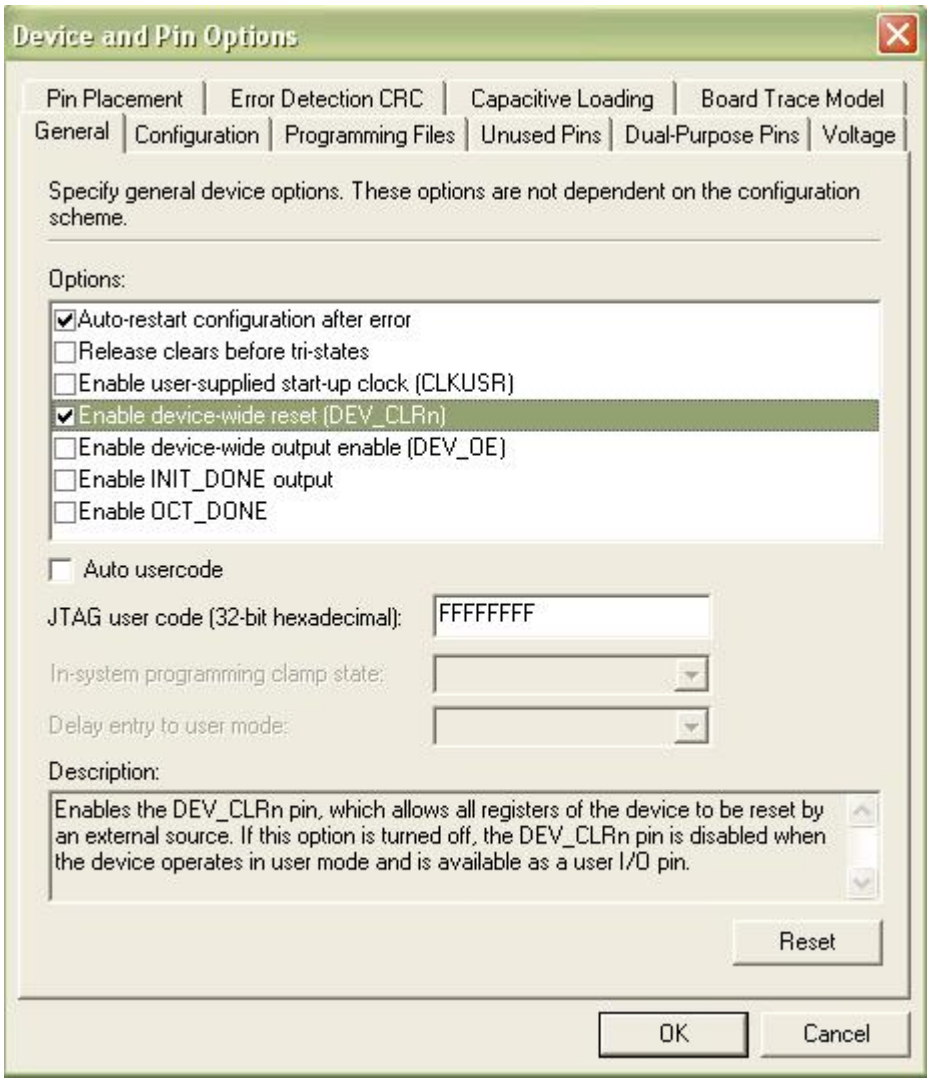
Pin Settings

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

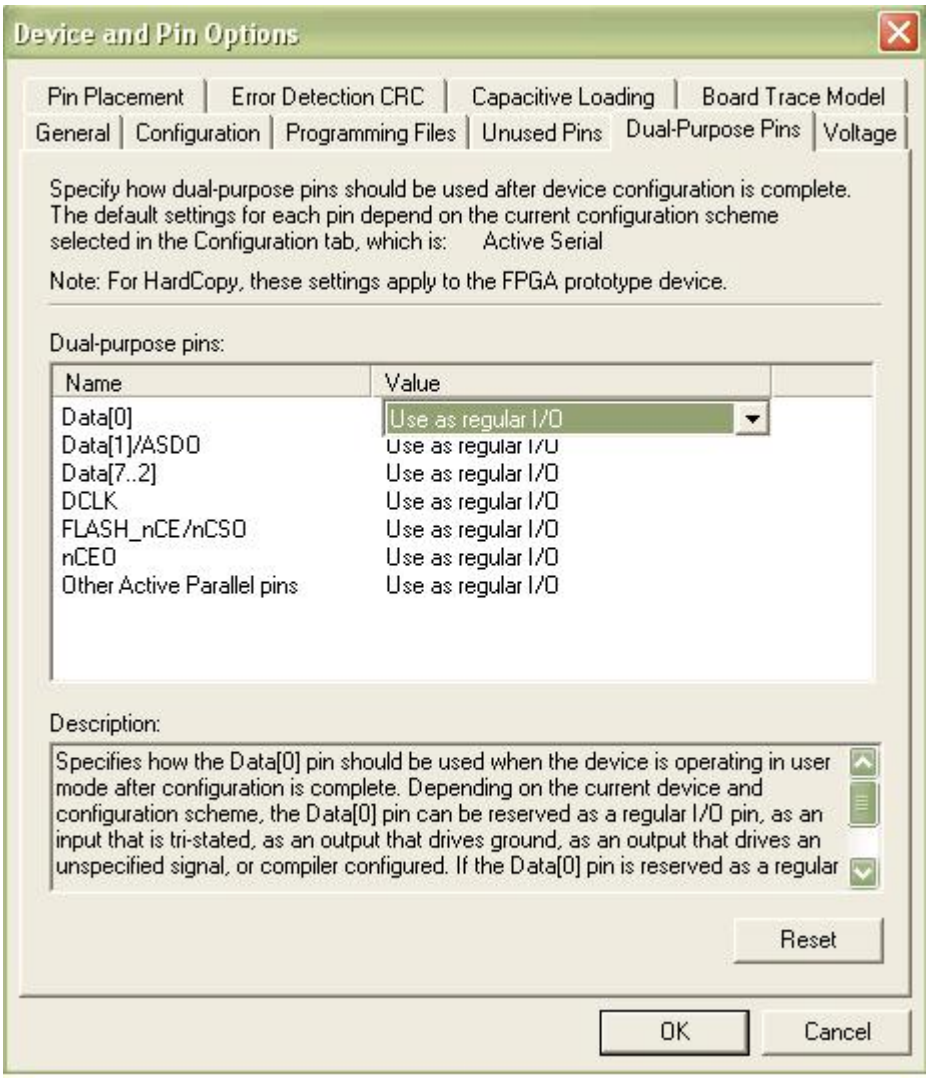


Click the **Device & Pin Options...** button to open the **Device & Pin Options** dialog:

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.



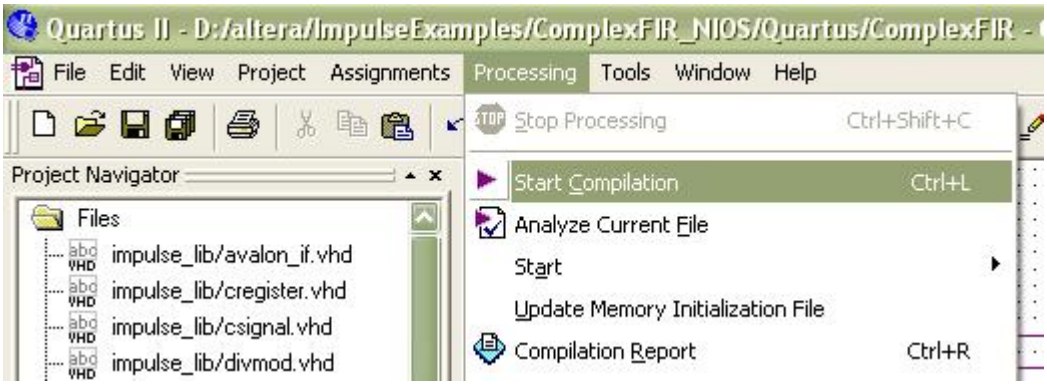
Next, select the **Dual-Purpose Pins** tab and specify **Use as regular I/O** for all dual-purpose pins listed:



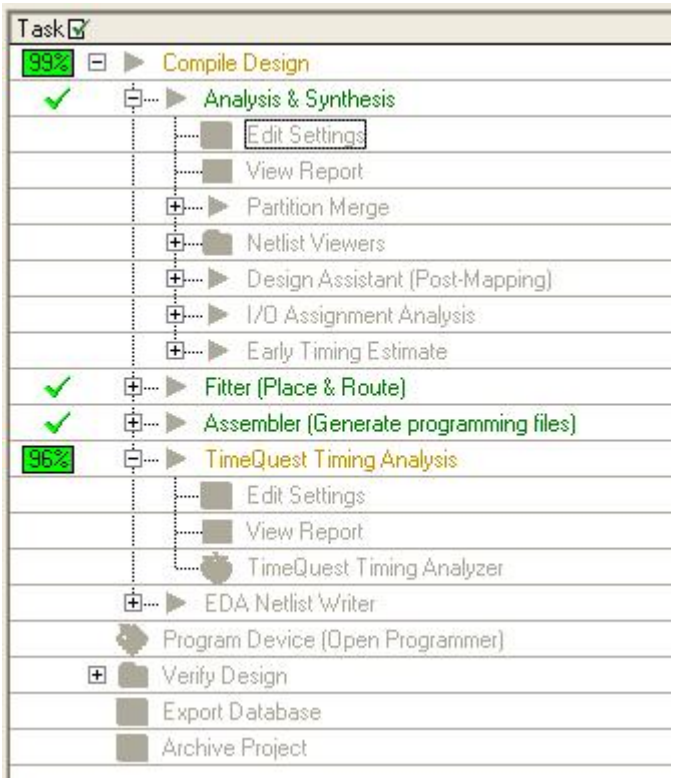
Click **OK** to save the changes.

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:



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.

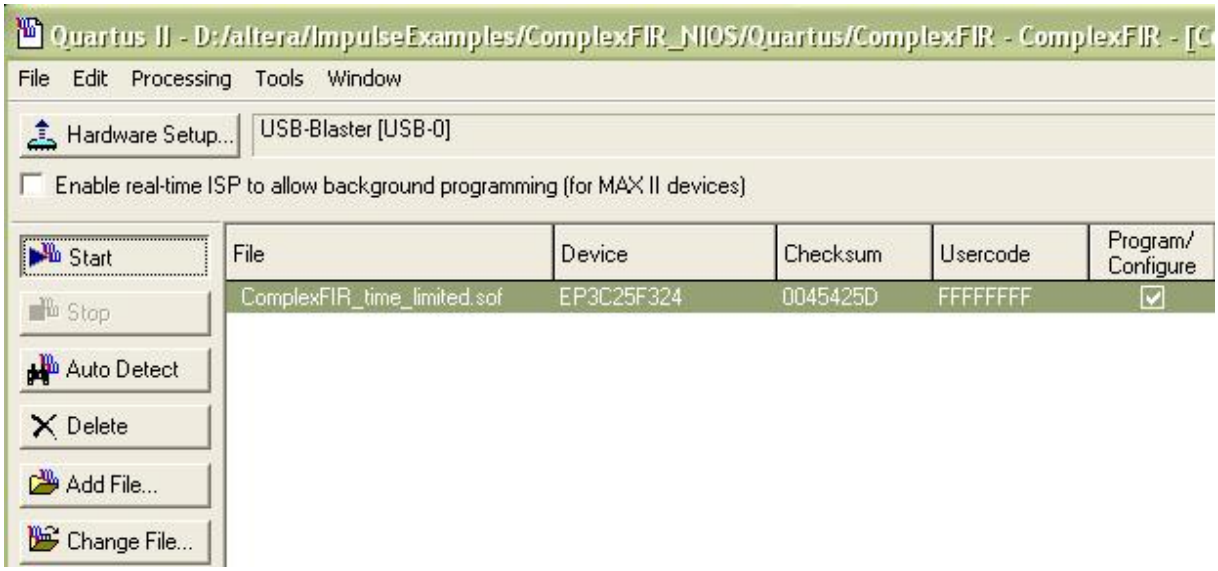
Downloading Bitmap

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.*



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

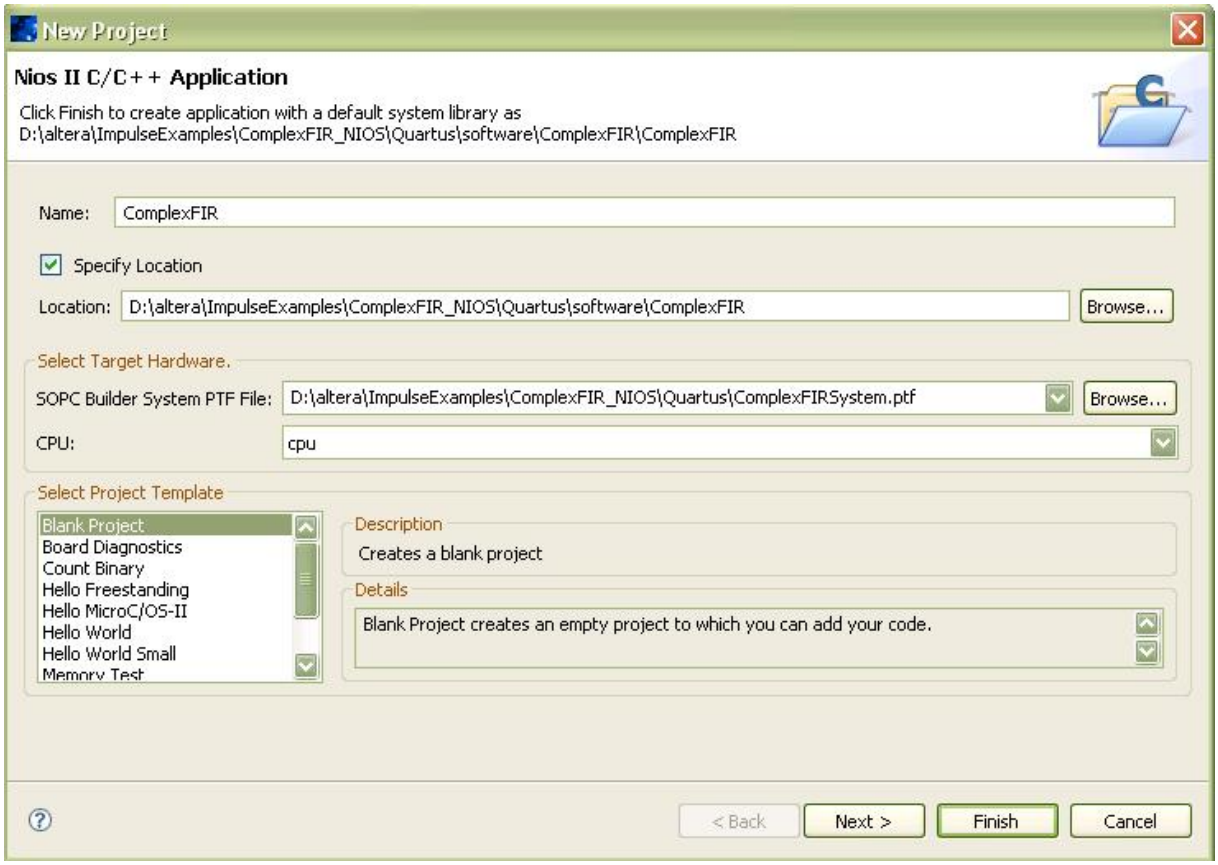
(The project path should point to the software files that were exported by CoDeveloper in Step 5.)

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

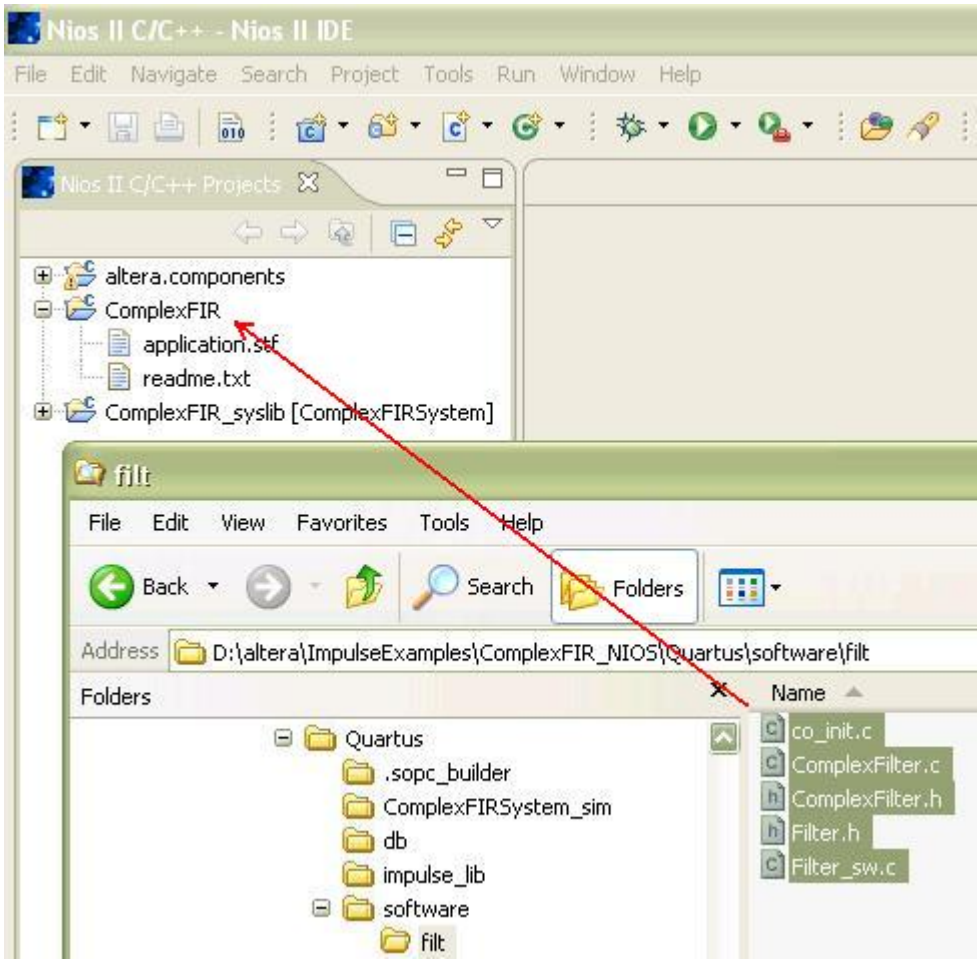
Select Project Template: Blank Project

The **New Project** dialog box will look as follows

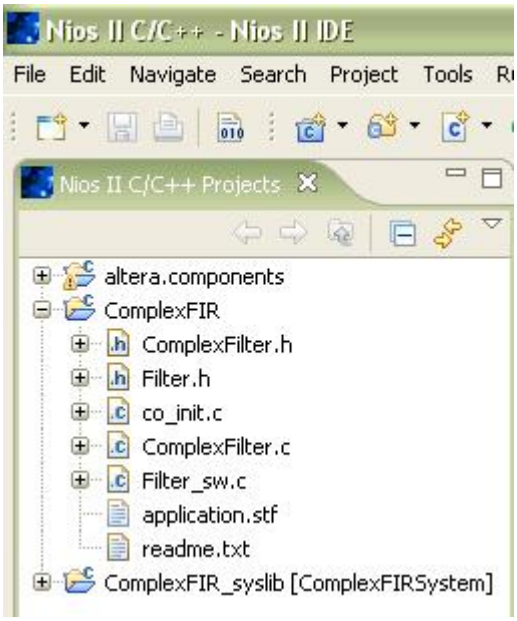


Click **Finish** to create the new project. Two new projects (**ComplexFIR** and **ComplexFIR_syslib**) should appear in the **Nios II C/C++ Projects** window in the **Nios II IDE**, as shown below.

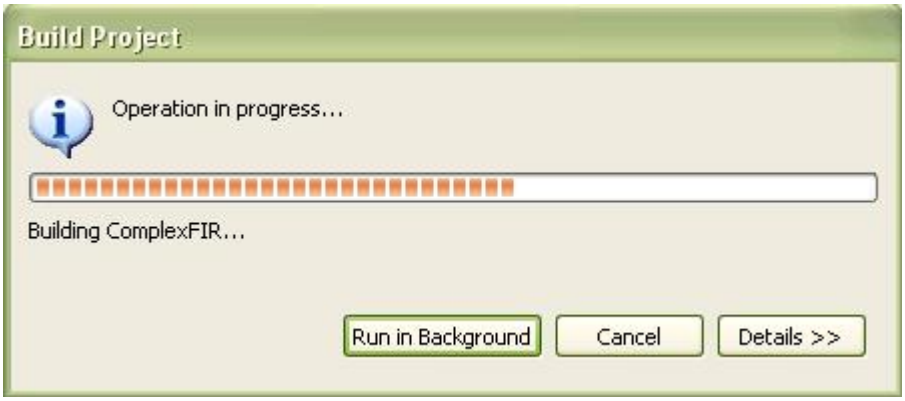
Copy the software files that were exported in Step 4 (**co_init.c**, **ComplexFilter.c**, **Filter_sw.c** and **ComplexFilter.h**, **Filter.h**) to the **ComplexFIR** project as shown below.



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

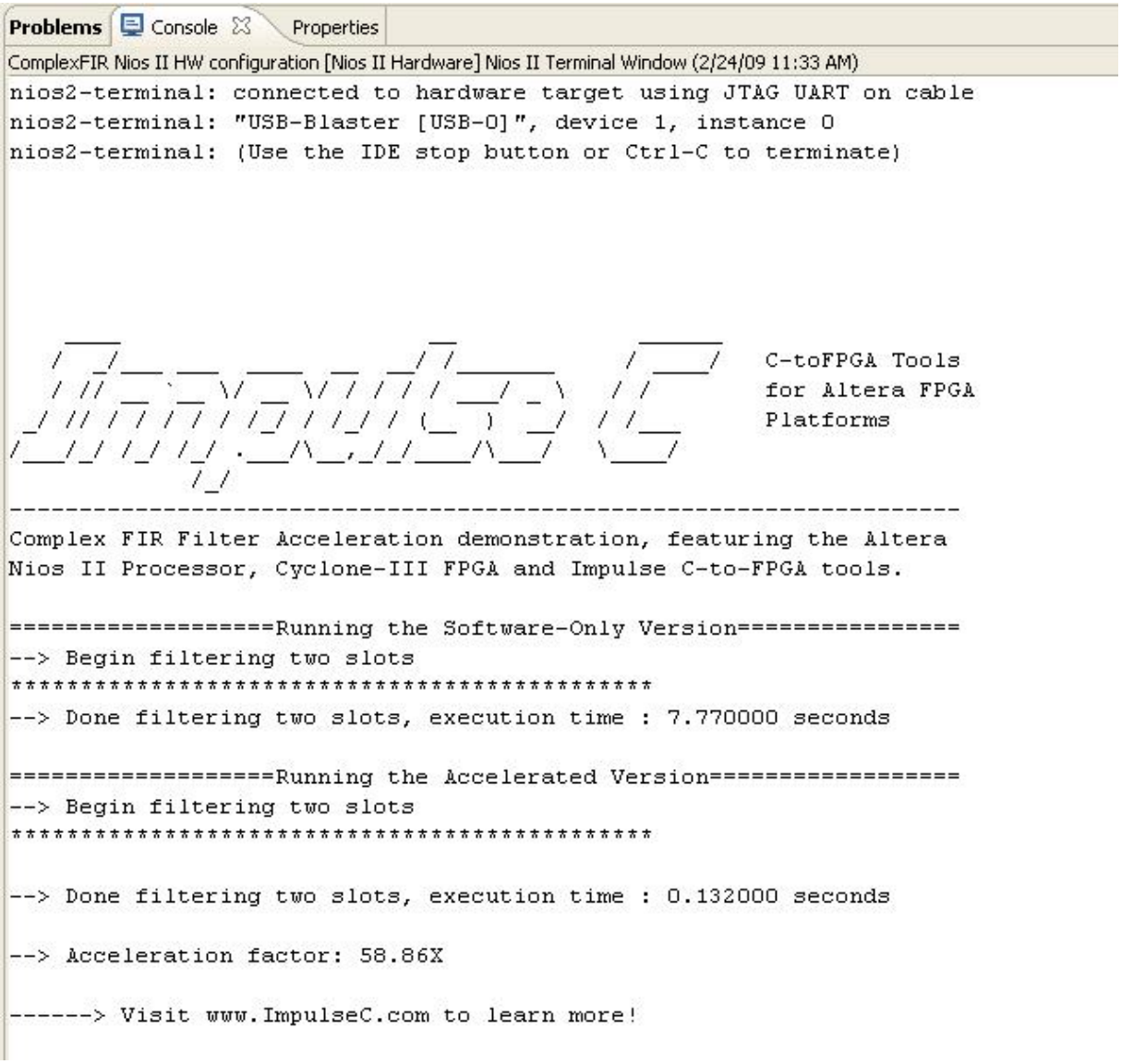


Now build the project by right-clicking the **ComplexFIR** project and selecting **Build Project**. The IDE will build the **ComplexFIR_syslib** system library, which includes a driver for the Impulse C hardware module created by CoBuilder, along with the application software code in the **ComplexFIR**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



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