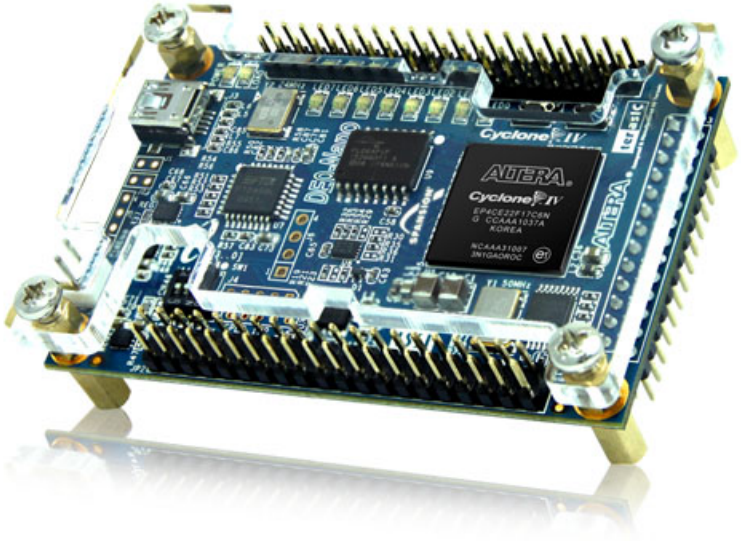
Complex FIR Filter Acceleration for the Nios II on the DE0 Nano Platform (Quartus II 12)

Table of Contents

1 Tutorial: Complex FIR Filter Acceleration for the Nios II on the DE0 Nano Platform (Quartus II 12)

Overview

This tutorial will demonstrate how to create, simulate, build, and run the example **Complex FIR** application targeting the **Terasic DE0 Nano Cyclone IV Development and Education Platform,** including the use of data streams connected to the **Nios II soft-CPU** via **Avalon-MM** interface and using the **Quartus II v12.0** tools.



This tutorial will require approximately one hour to complete, including software run times. To complete the application, you will need access to a **Terasic DE0 Nano** development board shown above. This tutorial shows how to use both the **CoDeveloper IDE** and **plugin** for **Microsoft Visual Studio C++ 2010 Express** to do desktop software simulation through generating and exporting hardware for integration using **Quartus II**. The use of the **plugin** is optional, however comes with the advantage of the integrated **Visual Studio C++** debugger.

General Steps

This tutorial will take you through the entire process of creating a hardware-accelerated system in the **Cyclone IV FPGA** using the Impulse and Altera tools following these general steps:

1. Describe and simulate the application using C language and either the Impulse **CoDeveloper IDE** or **Microsoft Visual Studio C++.**
2. Automatically generate hardware, in the form of VHDL source files, for the hardware accelerator portion of the application.
3. Export the generated files to a **Quartus** project directory which contains a pre-configured set of **Quartus II** and **Qsys** projects based on the "**DE0_Nano_SOPC_DEMO**" supplied with the **DE0**

Nano board.
4. Attach the hardware accelerator generated in step 2 to the **Nios II CPU** via **Avalon-MM** interfaces.
5. Start a new **Nios II EDS** project adding all necessary software files representing the application to be run on the **Nios II**.
6. Run synthesis and place-and-route to generate a binary for download to the FPGA.
7. Download the **FPGA** binary to the **DE0 Nano** board using its **USB** programming cable.
8. Download and run the application on the **Nios II** using **EDS.**

Steps

*Note: This tutorial assumes you have purchased or are evaluating the **CoDeveloper Platform Support Package** for **Altera Qsys**, and that you have installed and have valid licenses for the **Altera Quartus II v12.0**, **Qsys** and **Nios II EDS** products. The **Quartus II Web or Subscription Editions** will include all necessary tools in the single download.*

*Note: Though not required, this tutorial also shows how to make use of the **CoDeveloper plugin** for **Microsoft Visual Studio C++ 2010 IDE**. The **plugin** may be used with any of the purchased or free "**Express**" editions. **Microsoft Visual Studio C++ 2010 Express** may be downloaded free (requires registration for an activation code) from **Microsoft's** web site from:*
http://www.microsoft.com/visualstudio/eng/downloads

1.1 Loading the Complex FIR Filter Example

ComplexFIR Filter Tutorial, Step 1

The base project files for this tutorial are contained within the Zip file "DE0_Nano_ComplexFIR_Tutorial_v12.zip". Unzip the supplied Zip file into the desired working directory ensuring the path does not include the space (' ') character. Once unzipped the following directory structure will be present:

DE0_Nano_ComplexFIR_Tutorial_v12\ - Contains source code and CoDeveloper project files

DE0_Nano_ComplexFIR_Tutorial_v12\MSVC - Contains **Microsoft Visual Studio C++** project files

DE0_Nano_ComplexFIR_Tutorial_v12\DE0_Nano_QsysBase - contains a pre-configured set of **Quartus II** and **Qsys** projects based on the "**DE0_Nano_SOPC_DEMO**" supplied with the **DE0 Nano** board

Files included in the **ComplexFIR** project include the 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 the single hardware process.

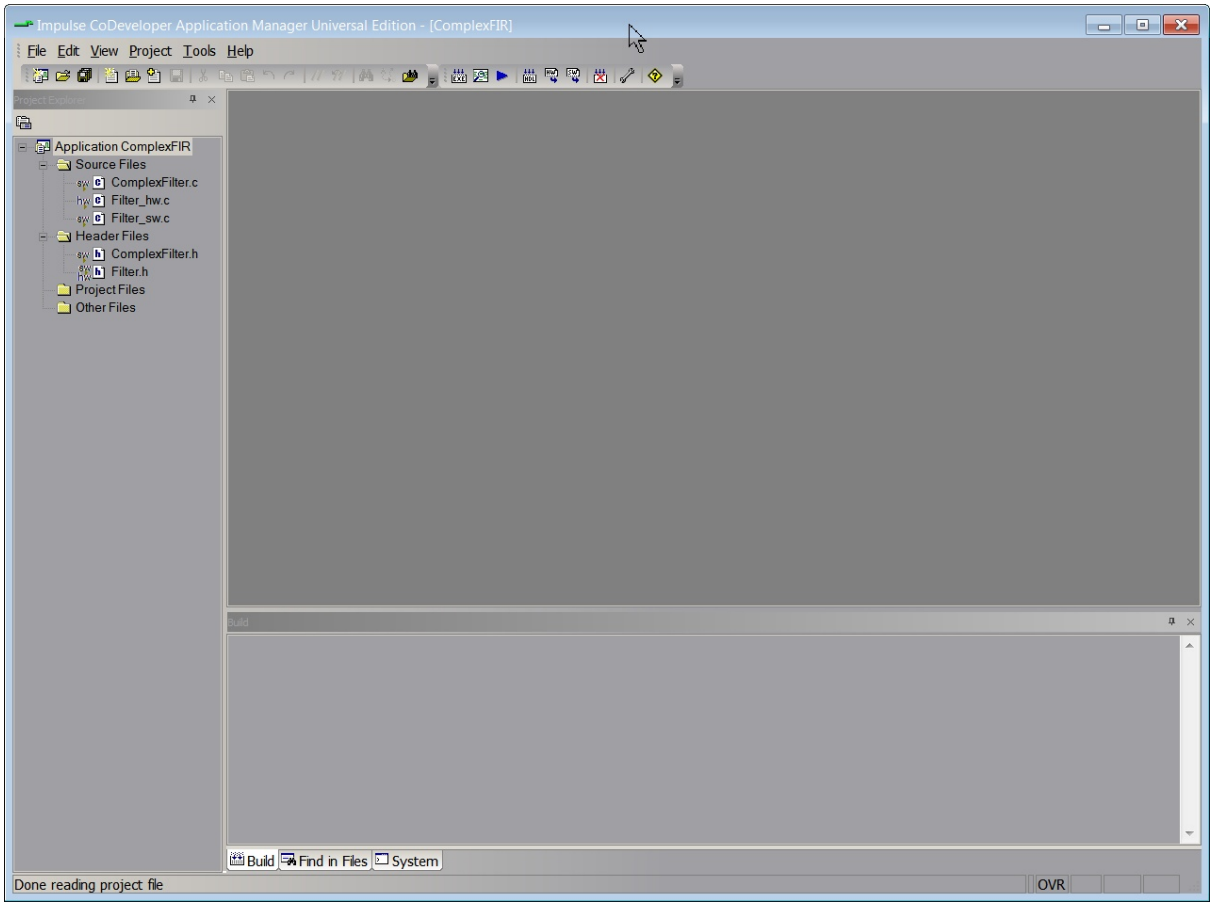
How to Do This Step:

Using the CoDeveloper IDE
Using the Visual Studio C++ Plugin

Using the CoDeveloper IDE:

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

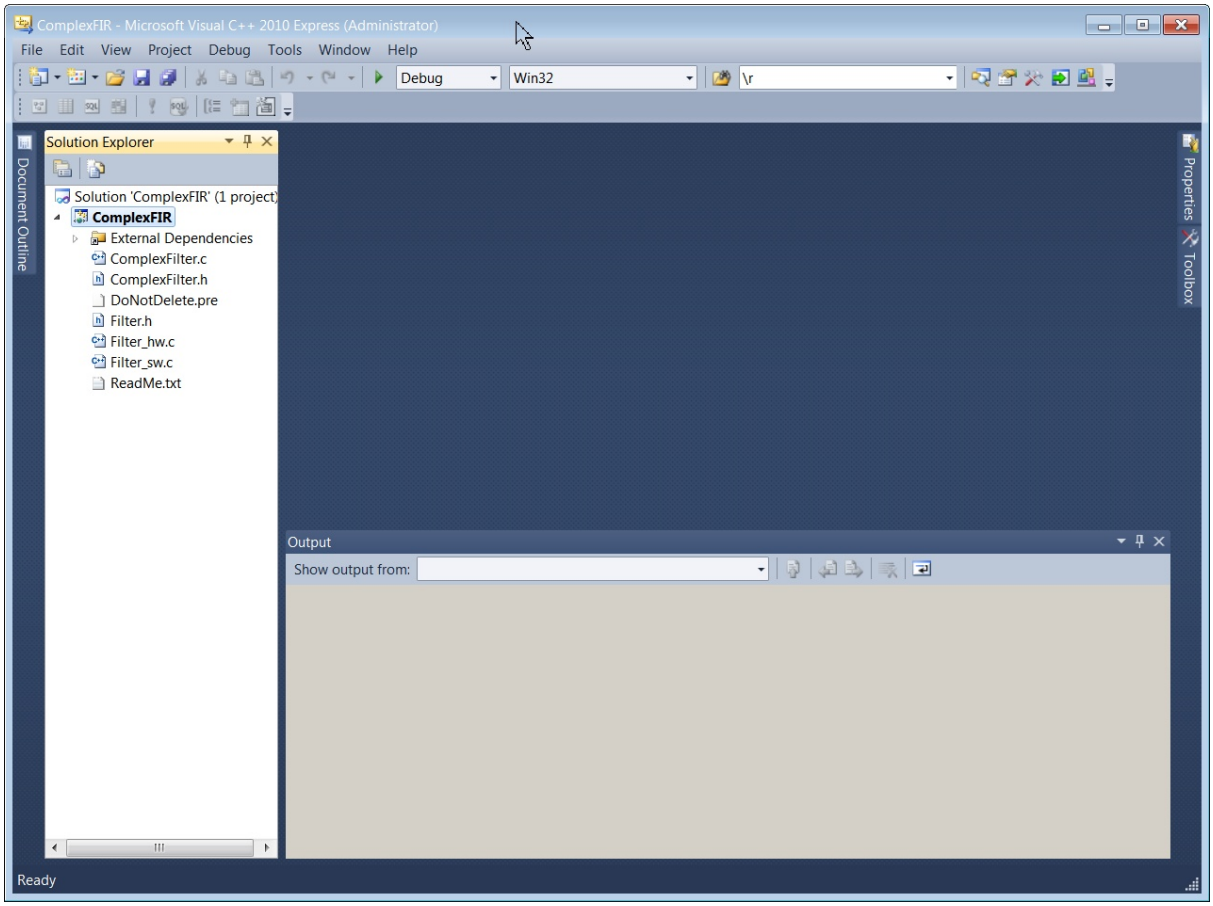
Open the **Altera Nios II Complex FIR filter** sample project by selecting Open Project from the File menu, or by clicking the Open Project toolbar button. Navigate to the location of the DE0_Nano_ComplexFIR_Tutorial_v12 directory which was just unzipped and select the CoDeveloper project named "ComplexFIR.icProj". Opening the project will display a window similar to the following:



Using the Visual Studio C++ Plugin:

To begin, start the **Microsoft Visual C++ 2010** by selecting Application Manager from the **Start** -> **All Programs** -> **Microsoft Visual Studio 2010** program group.

Open the **Altera Nios II Complex FIR filter** sample project by selecting **File -> Open -> Project/ Solution**. or by clicking the **Open Project** link on the **Start Page**. Navigate to the location of the DE0_Nano_ComplexFIR_Tutorial_v12\MSVC directory which was just unzipped and select the project named "ComplexFIR.vcxproj". Opening the project will display a window similar to the following:



See Also

Step 2: Understanding the Complex FIR Filter Example

Complex FIR Filter on the Nios II platform

1.2 Understanding the Complex FIR Filter Example

Complex FIR Filter Tutorial, 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, 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 with one result per input which is 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 both in desktop software simulation and on the target **NIOS II** processor, generating test data and verifying the results against the legacy C algorithm from which **complex_fir** was adapted.

The configuration that describes the how the processes are connected 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");
}
```

This configuration function describes the connectivity between instances of each previously defined process along with partitioning those that are to be implemented in hardware vs. software. Only one process in this example (**filter_proc**) will be mapped into 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 the single instance **filter_proc**). This also instructs the **Platform Support Package** to provide all the necessary hardware interfaces and drivers to make the connections between hardware and software.

The **ComplexFilter.c** contains routines that generates a set of complex FIR coefficients and a group of input data being processed.

The **Filter_sw.c** will run in desktop software simulation as a software test bench as well as be exported for running as the application on the NIOS II embedded processor to again control the stream flow and print results.

See Also

Step 3: Compiling the Complex FIR Filter for Simulation

Complex FIR Filter on the Nios II platform

1.3 Compiling and Simulating the ComplexFIR Application

CompleFIR Filter Tutorial, Step 3

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 desktop simulation purposes. There are also additional printf's commented out in the source code which will display the actual data being transferred.

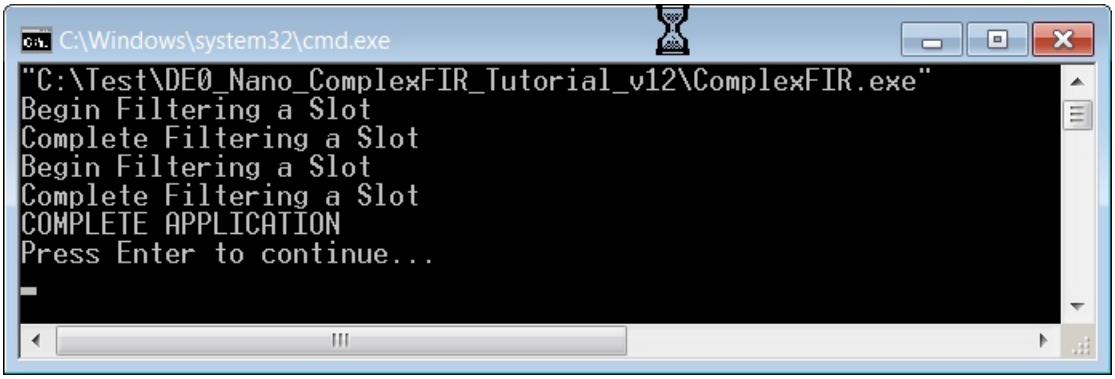
How to Do This Step:

Using the CoDeveloper IDE
Using the Visual Studio C++ Plugin

Using the CoDeveloper IDE:

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

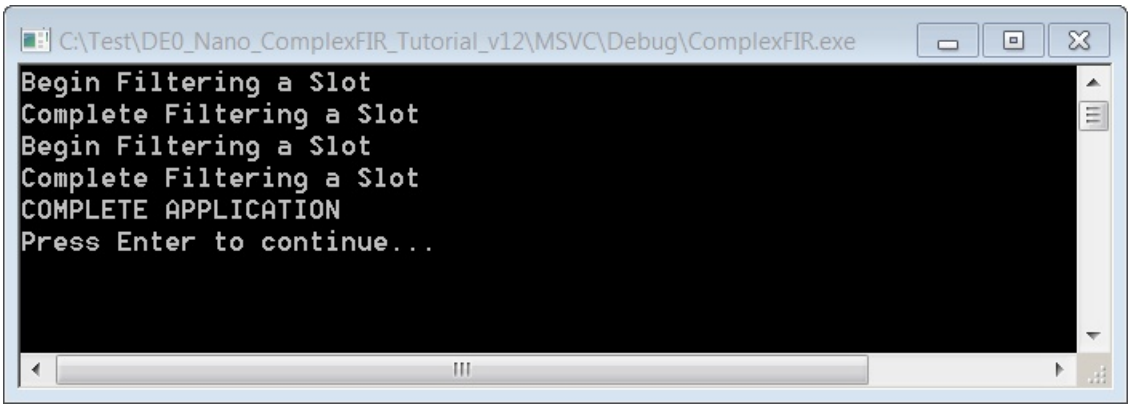
1. Select **Project** -> **Build Software Simulation Executable** (or click the **Build Software Simulation Executable** button) to build the **ComplexFIR.exe** executable. The **Build** tab will display compile and link messages.
2. You now have a Windows executable representing the ComplexFIR application 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:



Using the Visual Studio C++ Plugin:

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

1. Select **Debug** -> **Build Solution** to build the **ComplexFIR.exe** executable. The **Output** area will display 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 **Debug** -> **Start Debugging**. A command window will open and the simulation executable will run as shown below:



Using the integrated Visual Studio C++ debugger, breakpoints may be set in each of the processes, including the hardware process **complex_fir**, for further inspection and debug. Each process will appear as a separate thread during debug.

See Also

Step 4: Building the Complex FIR Filter Example

Complex FIR Filter on the Nios II platform

1.4 Building the Complex FIR Filter Example

CompleFIR Filter Tutorial, Step 4

The next step, prior to compiling and generating the HDL and related output files, is to select a platform target and generate the HDL and associated interfaces. 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 (**ComplexFIR_comp.vhd**, **ComplexFIR_top.vhd**) representing the hardware process and the generated hardware interfaces for streams.
- A **lib** subdirectory containing required VHDL library elements.
- A **HAL** subdirectory containing software driver files required by the Altera Nios II software tool.
- A **export** subdirectory containing hardware component files required by the Altera Qsys tool.

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 (**filt_module_co_init.c**) representing the hardware initialization function. This file will also be compiled to the embedded processor.

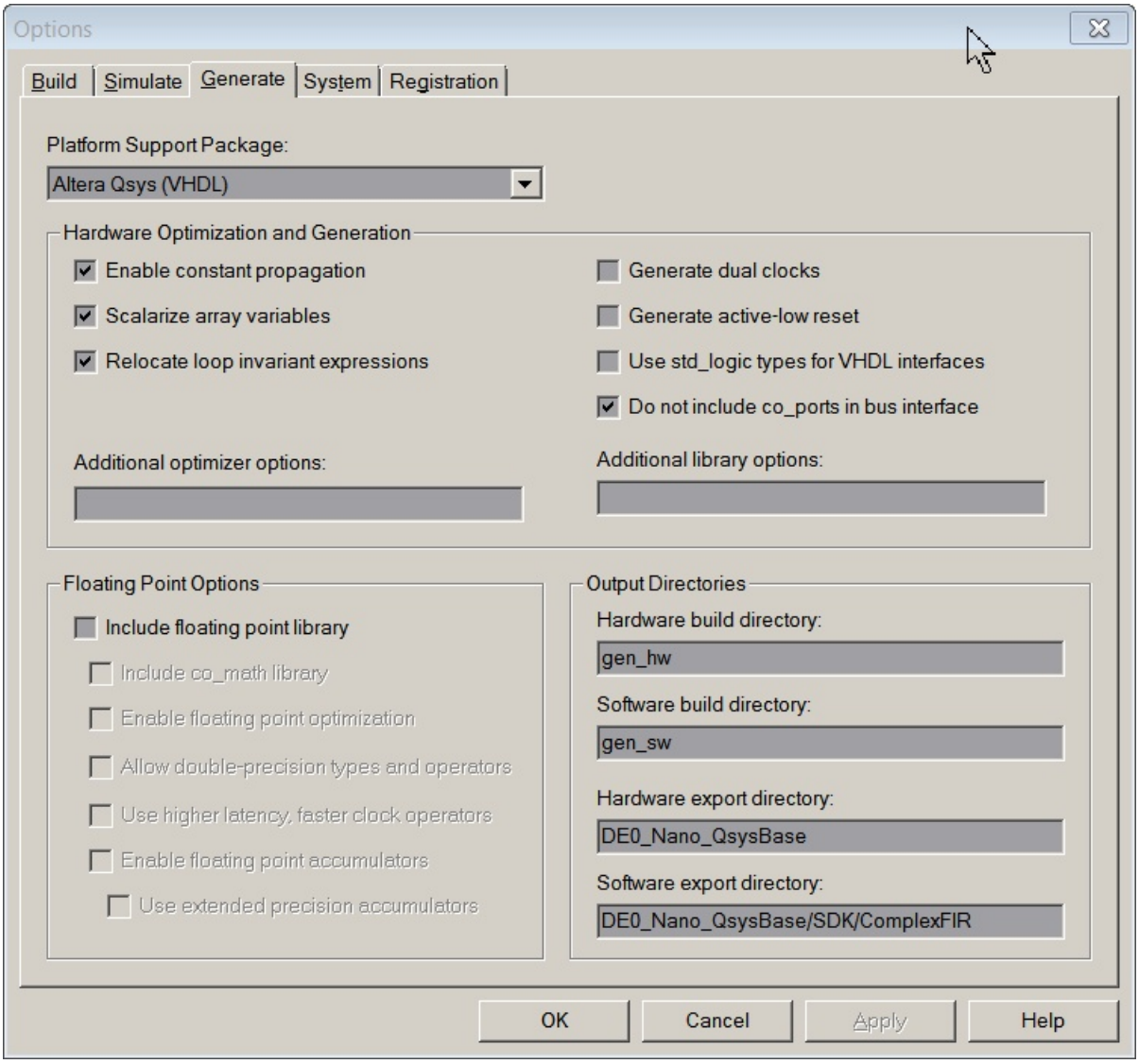
How to Do This Step:

Using the CoDeveloper IDE
Using the Visual Studio C++ Plugin

Using the CoDeveloper IDE:

Specifying the Platform Support Package

To specify a platform target, open the **Generate Options** dialog as shown below (**Project** -> **Options**, **Generate** tab):



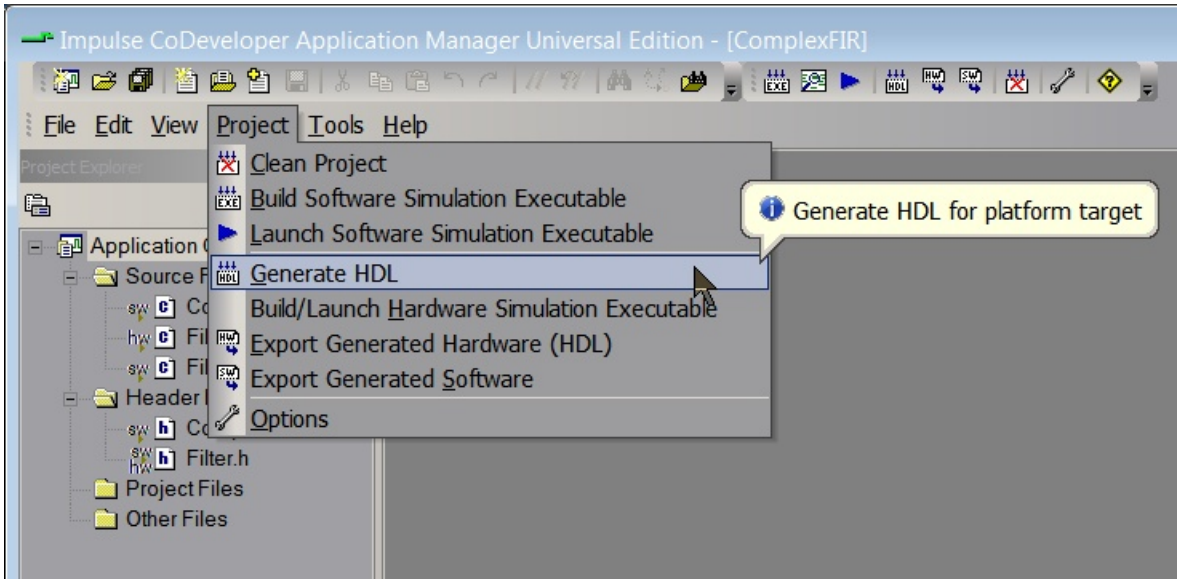
Ensure the following settings are set correctly:

- **Platform Support Package** = **Altera Qsys (VHDL)**
- **Hardware build directory** = **hw**
- **Software build directory** = **sw**
- **Hardware export directory** = **DE0_Nano_QsysBase**
- **Software export directory** = **DE0_Nano_QsysBase/SDK/ComplexFIR**

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:

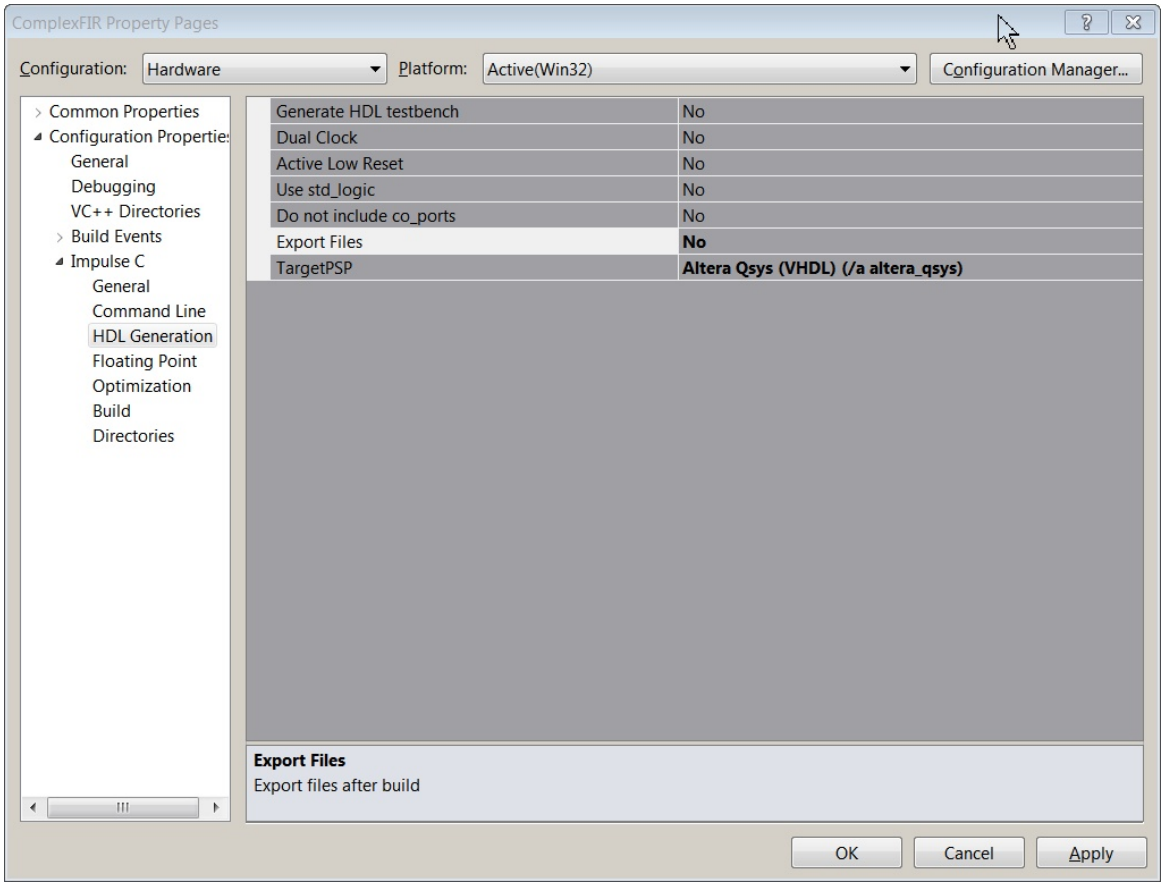


A series of processing steps will run with output shown in the **Build** window. No errors must be present before continuing.

Using the Visual Studio C++ Plugin:

Specifying the Platform Support Package

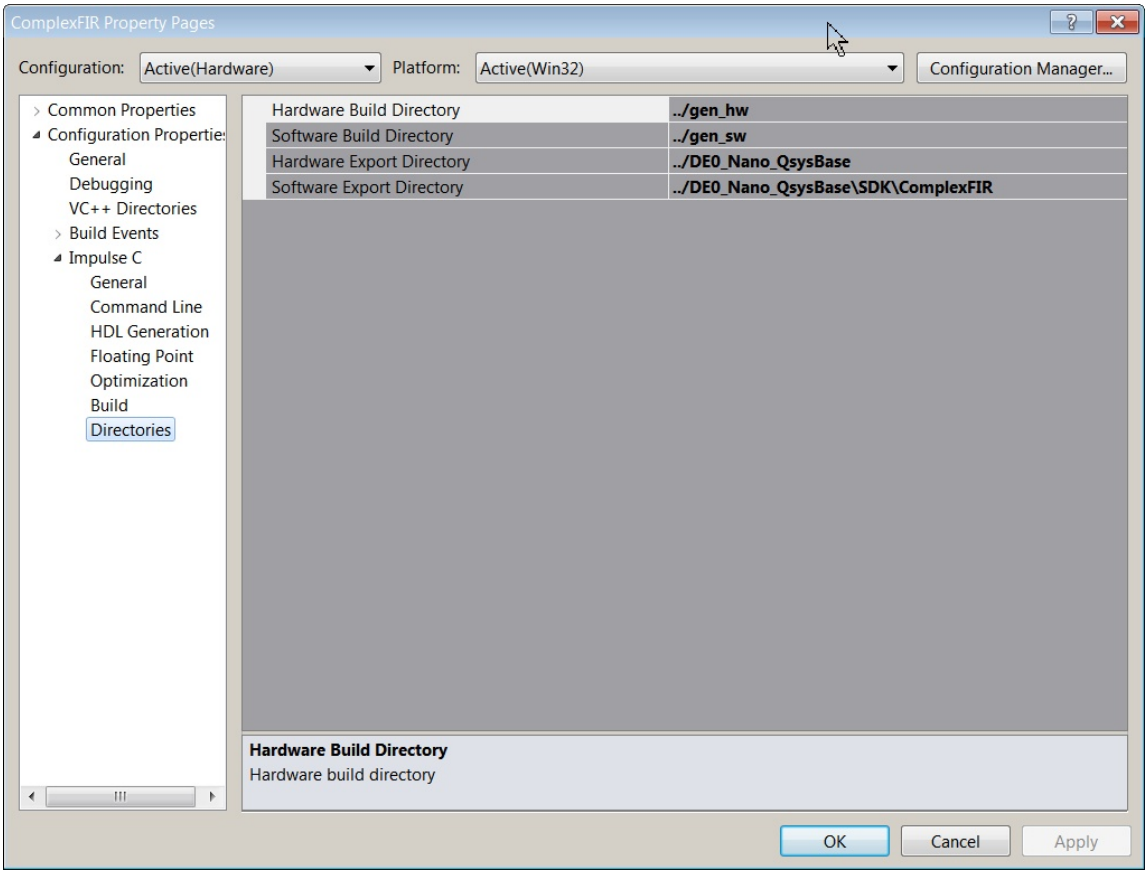
To specify a platform target, open the **Property Pages** dialog as shown below (**Project** -> **ComplesFIR Properties...**, **Impulse C -> HDL Generation,** select **Configuration** = **Hardware**):



Ensure the following settings are set correctly:

- **Platform Support Package** = **Altera Qsys (VHDL)**

To specify directories, now select **Impulse C** -> **Directories** as shown below:



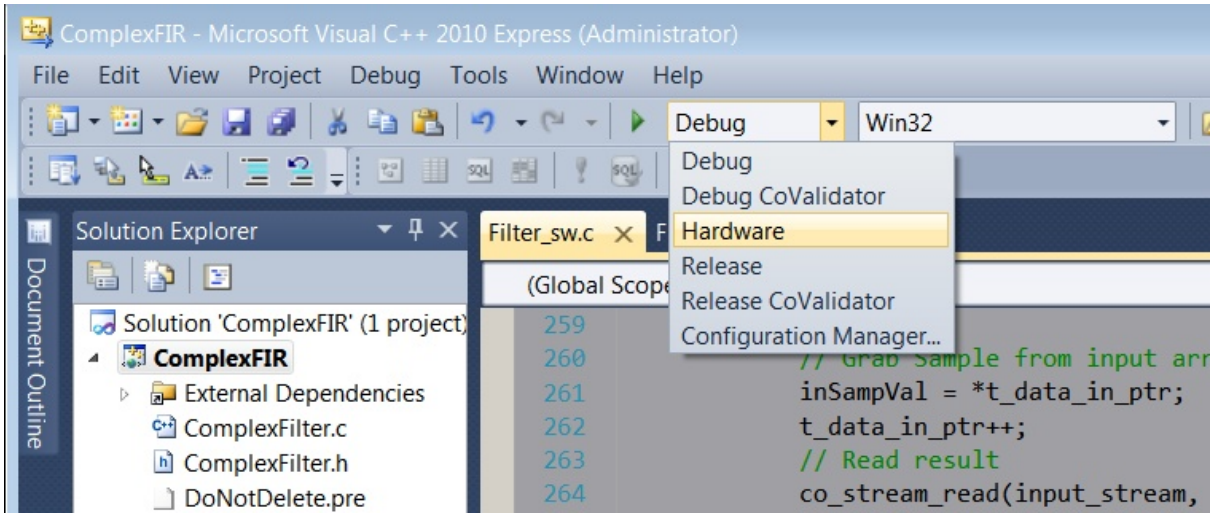
Ensure the following settings are set correctly:

- **Hardware build directory** = **hw**
- **Software build directory** = **sw**
- **Hardware export directory** = **../DE0_Nano_QsysBase**
- **Software export directory** = **../DE0_Nano_QsysBase/SDK/ComplexFIR**

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 the Hardware Solution Configuration as shown:



Now generate hardware by building the **Hardware** configuration via **Project** -> **Build Solution**. A series of processing steps will run with output shown in the **Output** window. No errors must be present before continuing.

See Also

Step 5: Exporting Files from CoDeveloper

Complex FIR Filter on the Nios II platform

1.5 Exporting Files from CoDeveloper

ComplexFIR Filter Tutorial, 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 use the export features of CoDeveloper to integrate the files into the Altera tools semi-automatically. This section exports the necessary files into the supplied Quartus II project directory DE0_Nano_QsysBase as an example.

Note: you must have the Altera Quartus II (version 12.0 or later), Qsys, and Nios Software Development software installed in order to proceed with this and subsequent steps.

Recall that in Step 4 you specified the **Platform Support Package** and directories in DE0_Nano_QsysBase as the export target for hardware and software. These export directories specify where the generated hardware and software files are to be copied when the **Export Software** and **Export Hardware** features of CoDeveloper are invoked. Within these target directories, the specific destination 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 Qsys**) is selected prior to starting the export process.

After this step is complete:

- A complete Qsys component, including HAL software driver, will appear in the directory: DE0_Nano_QsysBase\ip\filt_module
- Supporting HDL libraries will appear in the directory: DE0_Nano_QsysBase\ip\impulse_lib
- All exported software application files ready for adding to a Nios II project will appear in in the directory: DE0_Nano_QsysBase\SDK\ComplexFIR\export_sw\filt

How to Do This Step:

Using the CoDeveloper IDE
Using the Visual Studio C++ Plugin

Using the CoDeveloper IDE:

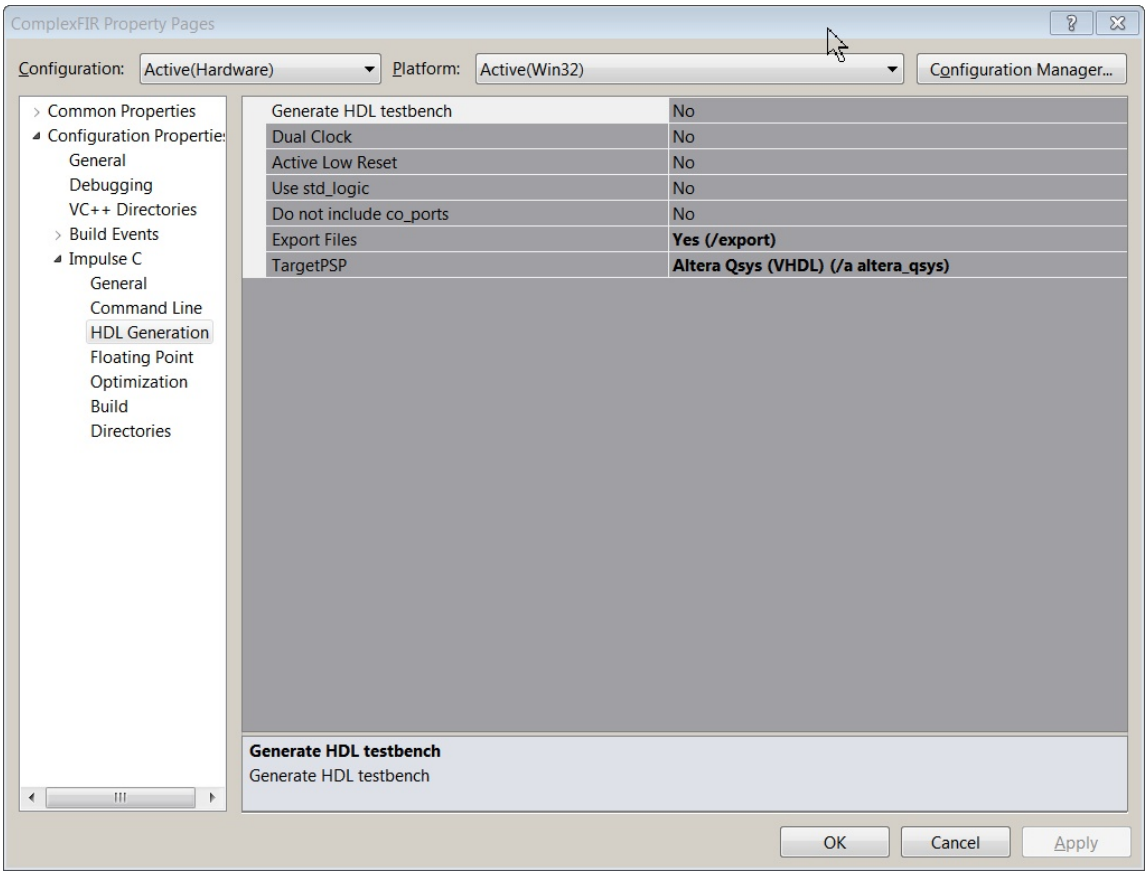
To export the files from the build directories (in this case **hw** and **sw**) to the export directories, select **Project** -> **Export Generated Hardware (HDL)** and **Project** -> **Export Generated Software**, or select the **Export Generated Hardware** and **Export Generated Software** buttons 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 II** project directory.

Using the Visual Studio C++ Plugin:

Exporting hardware and software is done as an option when building the Hardware solution configuration. To export the files from the build directories (in this case **hw** and **sw**) to the export directories, first ensure that the **Export Files** option under **HDL Generation** is set to "**Yes /export**" as shown below:



Now generate hardware by building the **Hardware** configuration via **Project** -> **Build Solution**. A series of processing steps will run with output shown in the **Output** window. No errors must be present before continuing.

See Also

Step 6: Configuring the Quartus Project

Complex FIR Filter on the Nios II platform

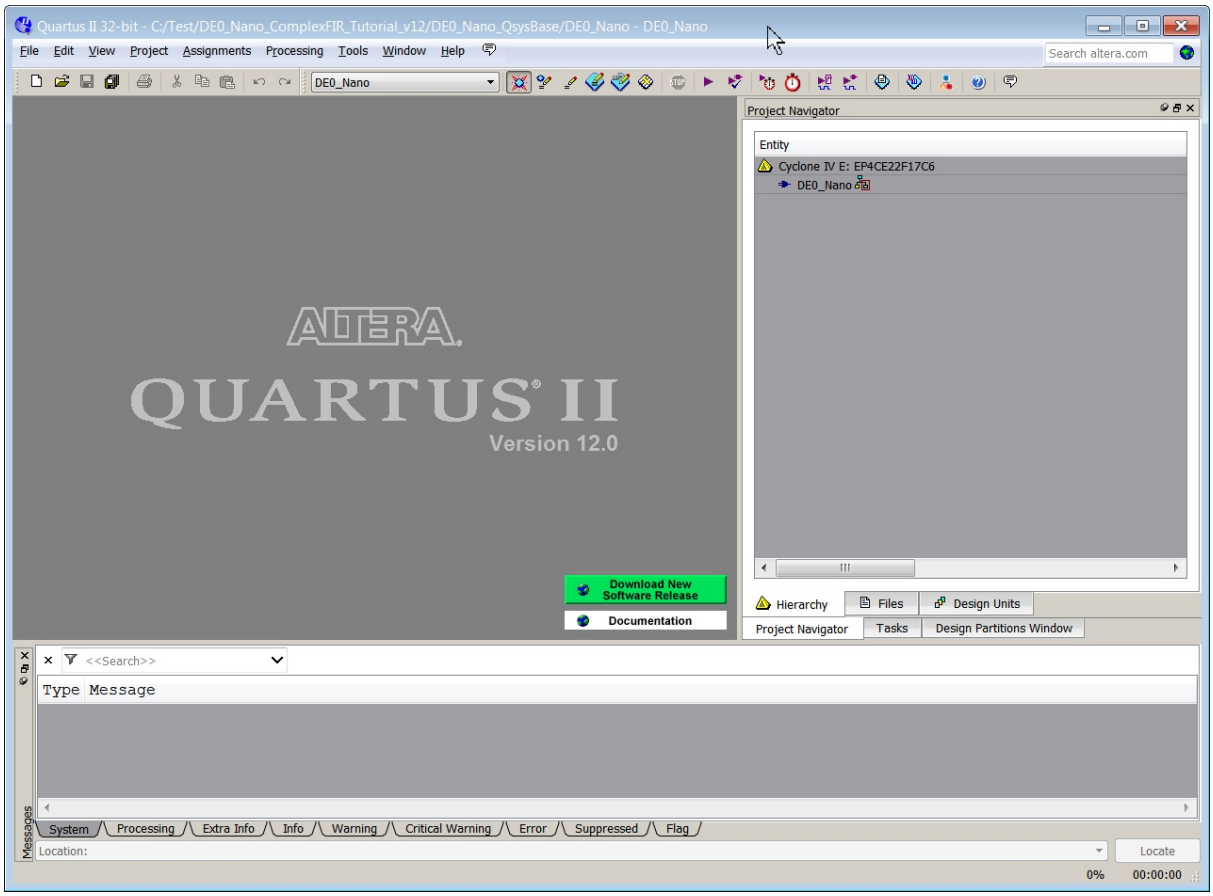
1.6 Configuring the New Platform

ComplexFIR Filter Tutorial, Step 6

The following instructions will lead you through the process of adding the ComplexFIR hardware to the pre-configured set of **Quartus II** and **Qsys** projects based on the "**DE0_Nano_SOPC_DEMO**" supplied with the **DE0 Nano** board located in the DE0_Nano_ComplexFIR_Tutorial_v12\DE0_Nano_QsysBase directory.

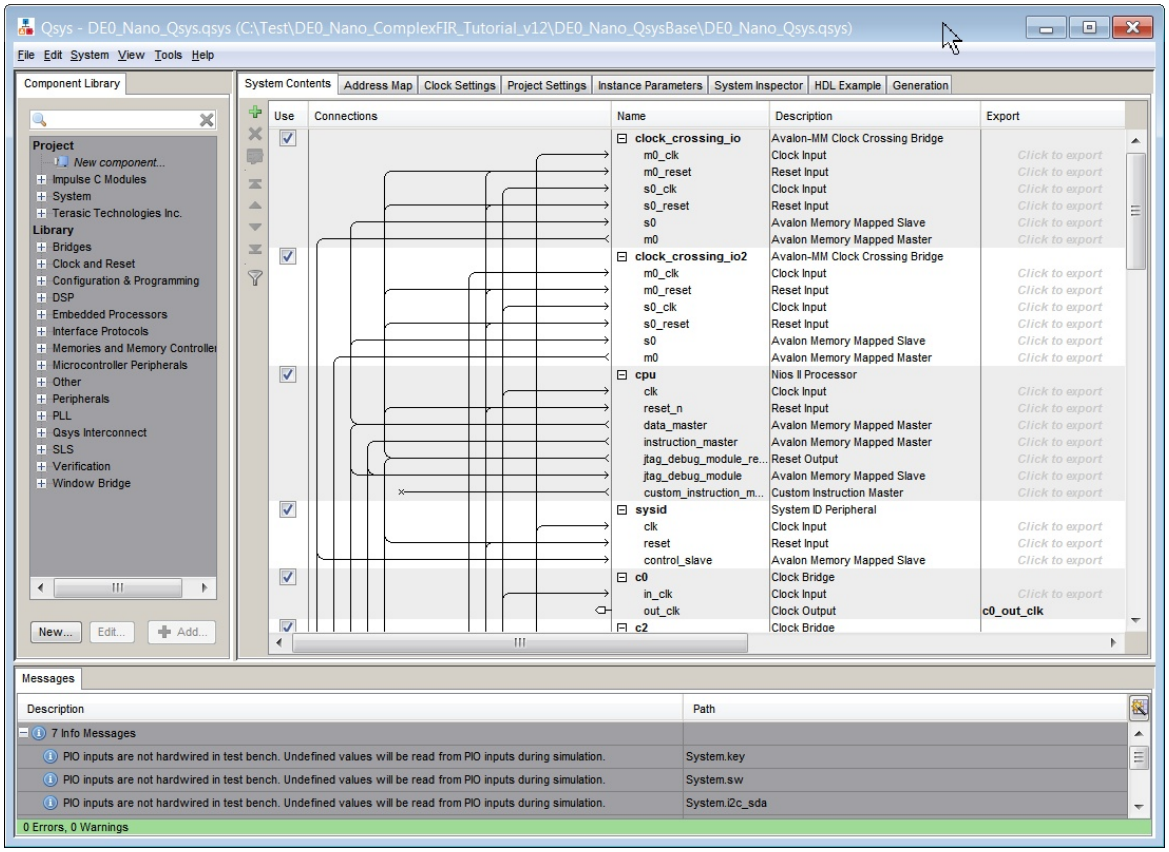
Open the Quartus II Project

Start first by selecting **Quartus II 12.0** (32-bit or 64-bit) from the **Start** -> **All Programs** -> **Altera 12.0** -> **Quartus II 12.0** programs group. Open the project by clicking on the "**Open Existing Project**" button, navigating to your DE0_Nano_ComplexFIR_Tutorial_v12\DE0_Nano_QsysBase directory to select "**DE0_Nano.qpf**", and then clicking the **Open** button. The project is now open and will appear similar to below:

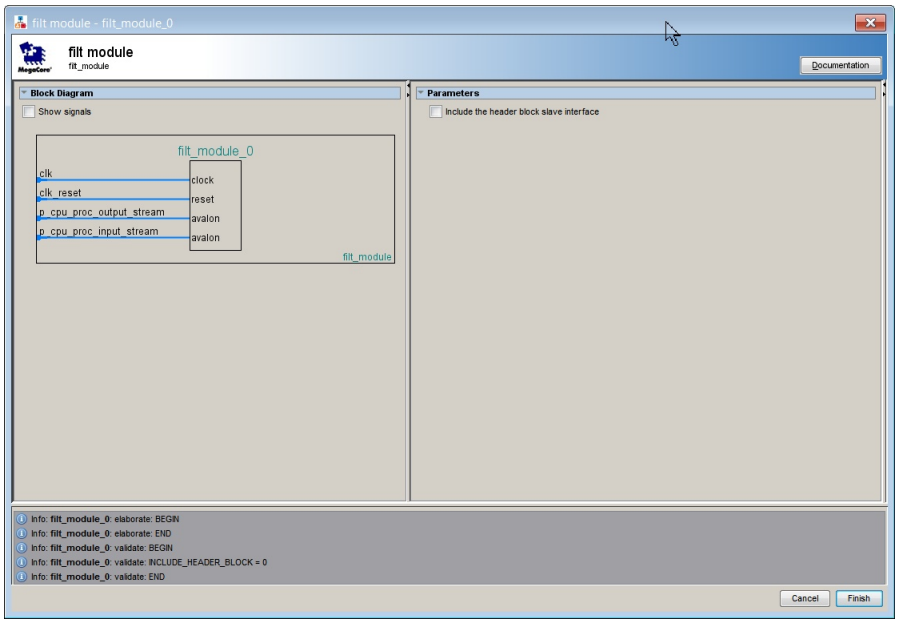


Adding the Hardware Process Module "filt_module" to the Qsys project

Open the Qsys project from within Quartus II by selecting Tools -> Qsys. When the **Open** dialogue appears, select the **DE0_Nano_Qsys.qsys** project and click the **Open** button. Qsys will now be open and appear as below:

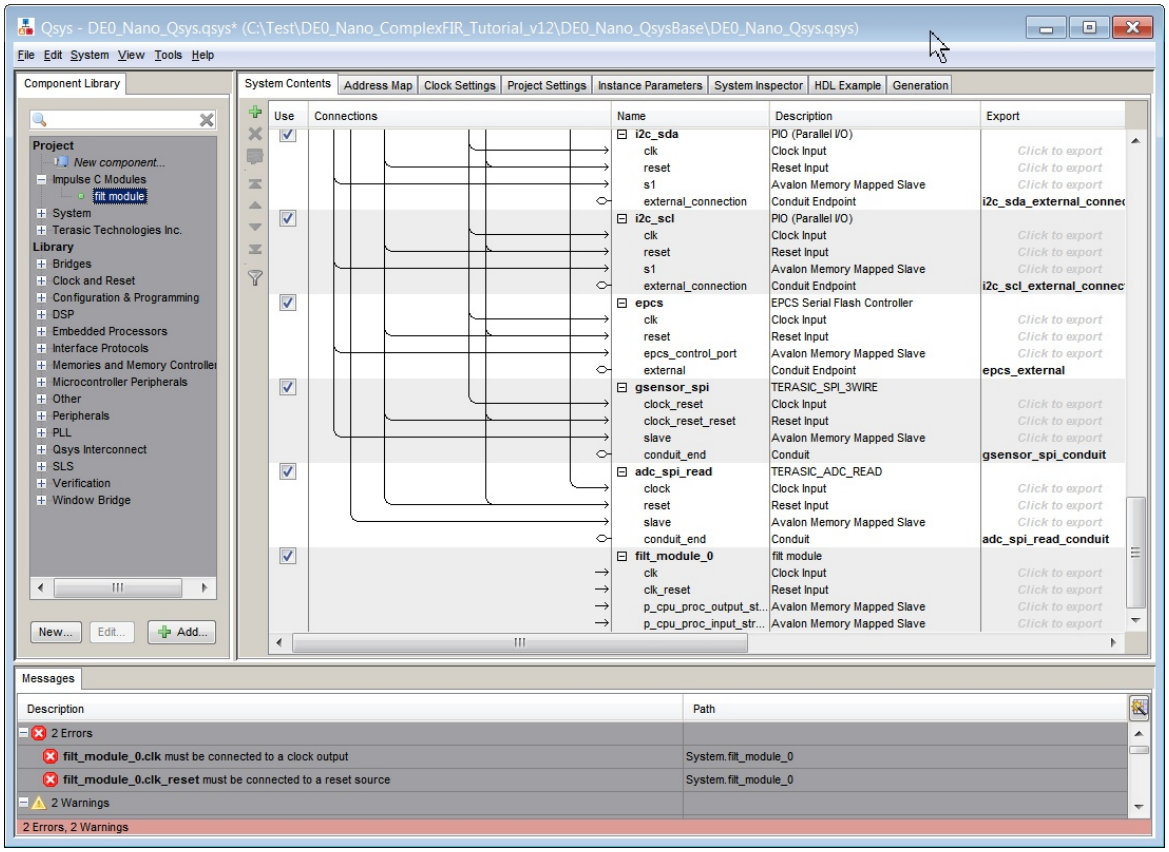


Now to add the module that implements the **ComplexFIR** hardware process. Expand the **Impulse C Modules** on the left under **Project** and double click the **filt module**. The **filt module** configuration window will appear as shown below:



Click the **Finish** button to continue.

Scrolling to the bottom of the **Qsys** project window will show the newly added **filt_module_0** with ports unconnected as shown below:



Connecting the Hardware Process Module "filt_module"

Connect each of the ports of **filt_module_0** by right-clicking the port name, selecting **filte_module_0.<port name>**, then selecting the connection from those listed following the mapping below:

- For **filt_module_0.clk**, select **altpll_sys.c0**
- For **filt_module_0.clk_reset**, select **clk_50.clk_reset**
- For **filt_module_0.p_cpu_proc_output_stream**, select **cpu.data_master**
- For **filt_module_0.p_cpu_proc_input_stream**, select **cpu.data_master**

The **filt_module_0** is now connected to a 100MHz clock and reset with both stream Avalon-MM Slave ports connected to the Nios II CPU.

Assign Addresses

We can see that as we add modules, error messages appear in the console window showing address conflicts. Here, we let **Qsys** re-assign addresses for all the memory-mapped modules to avoid address overlaps. From the **Qsys** menu, select **System** -> **Assign Base Addresses**.

Save Qsys System

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

Your Nios II platform is ready for system generation. No errors must be present in the **Messages** window at this time before continuing.

See Also

Step 7: Generating the System

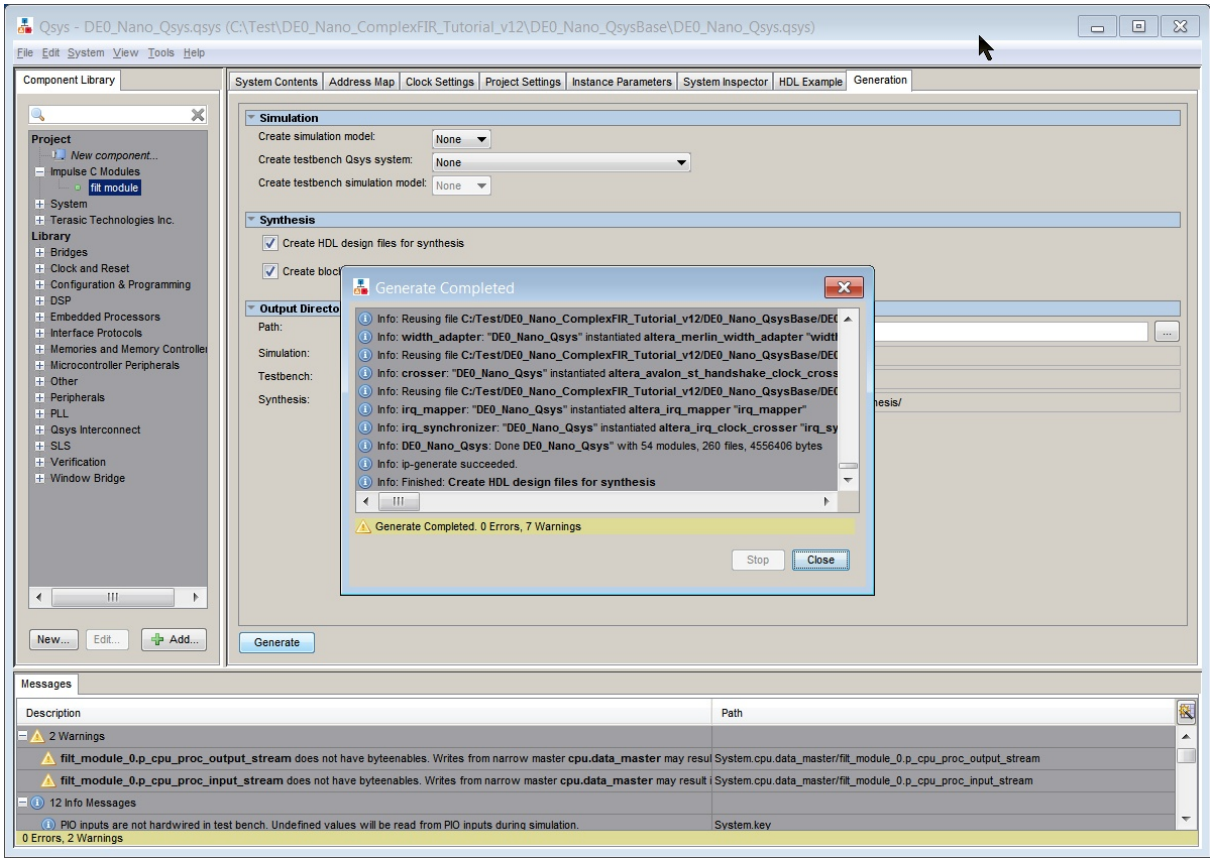
Complex FIR Filter on the Nios II platform

1.7 Generating the System

ComplexFIR Filter Tutorial, Step 7

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

From the **Generation** tab, click **Generate** button at the bottom of the **Qsys** window to generate the system. Make sure the **Create simulation model** option is set to **None** to save time. This process may take several minutes. Upon success, the **Generation** tab should appear similar to below:



When generation is complete you may exit **Qsys** and return to **Quartus**. Note that this step is required each time that you export your design from CoDeveloper.

Your project is now ready for **FPGA** binary 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 8: Generating the FPGA Binary

Complex FIR Filter on the Nios II platform

1.8 Generating the FPGA Binary

ComplexFIR Filter Tutorial, Step 8

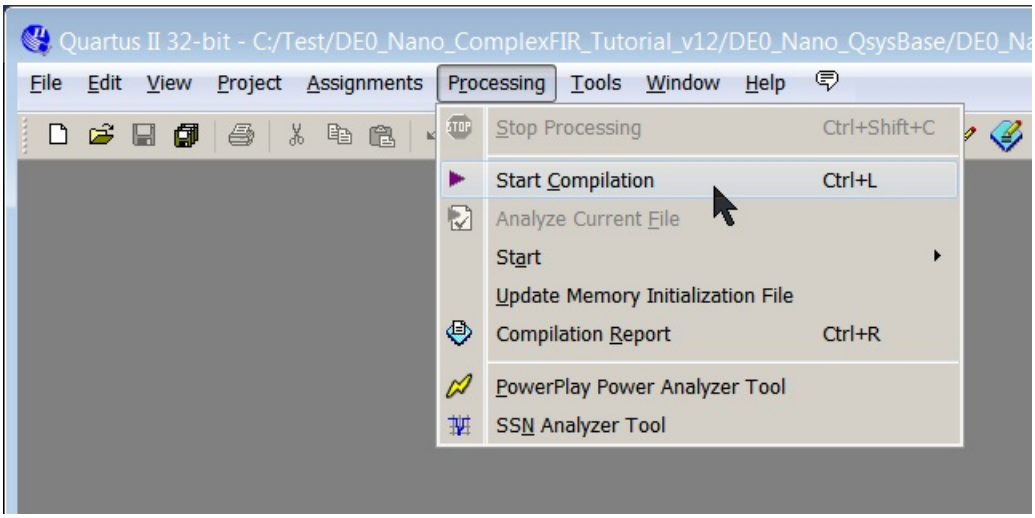
At this point, if you have followed the tutorial steps carefully you have successfully:

- Generated hardware and software files from the **CoDeveloper** environment.
- Opened the **Altera Quartus II** project and used **Qsys** to add the **filt module** to a **Nios II**-based platform.
- Imported your **CoDeveloper**-generated files to the **Altera** tools environment.

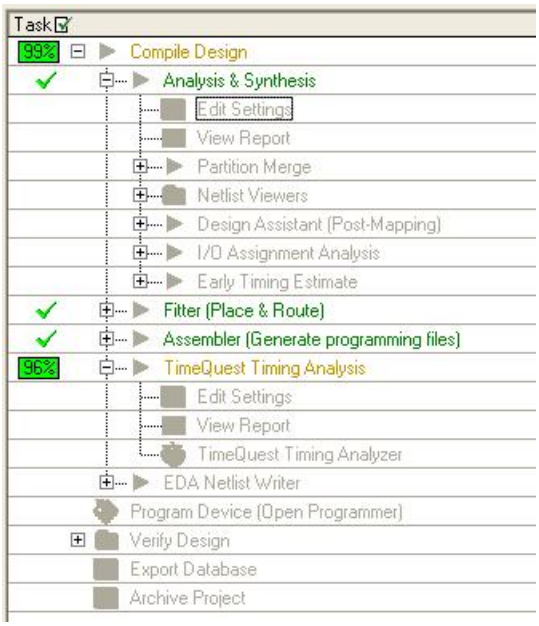
You are now ready to generate the FPGA binary and download the complete application to the target platform. This process is not complicated (at least in terms of your actions at the keyboard) but can be time consuming due to the large amount of processing that is required within the **Altera** tools.

Compiling the System

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



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



Note: this process may require 5 minutes or more to complete, depending on the speed and memory of your development system.

During compilation, **Quartus** will analyze the generated HDL 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 FPGA binary .sof file (in the appropriate Altera device format) ready for downloading to the device.

Downloading the FPGA Binary

Note: Before continuing, ensure that the DE0 Nano board is already connected to the PC via the supplied USB cable.

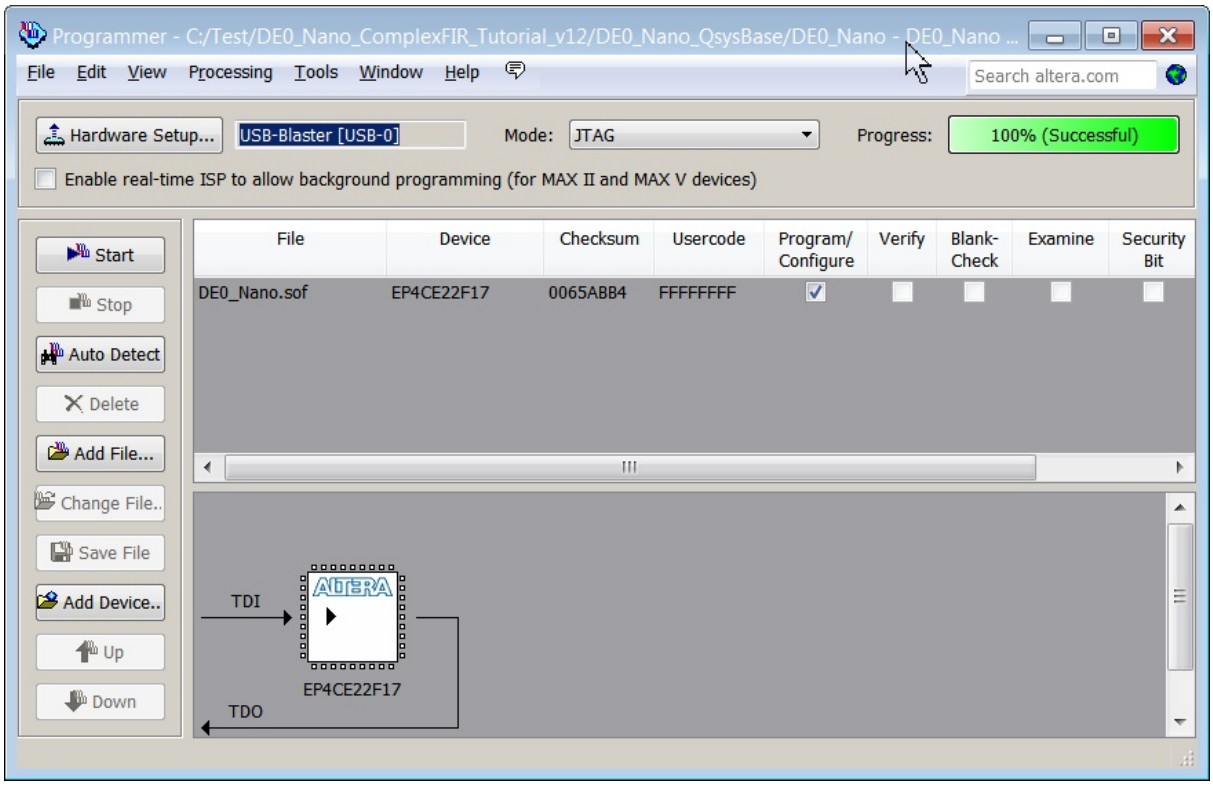
When the FPGA binary has been generated, select **Tools** -> **Programmer** to open a new programming file. Select **File** -> **Save As** and save the chain description file as ComplexFIR.cdf (make sure the "Add file to current project" option is selected).

The programming file **DE0_Nano.sof** should be visible in the programming window. If it is not, select **Add File...** and open **DE0_Nano.sof**.

Enable Program/Configure for **DE0_Nano.sof** and make sure your programming hardware is configured properly. Click **Start** to begin downloading the **DE0_Nano.sof** file to the target device.

*Note: If you do not have the full license for **OpenCore Plus** megafunctions, such as when using the **Quartus II Web Edition,** then a message will pop up. Click **OK** to continue. The FPGA binary file will also be named **DE0_Nano_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 FPGA binary will not function correctly.*

The Programmer tool will appear similar to below upon successful download of the FPGA binary:



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

See Also

Step 9: Running the Application on the Platform

Complex FIR Filter on the Nios II platform

1.9 Running the Application on the Platform

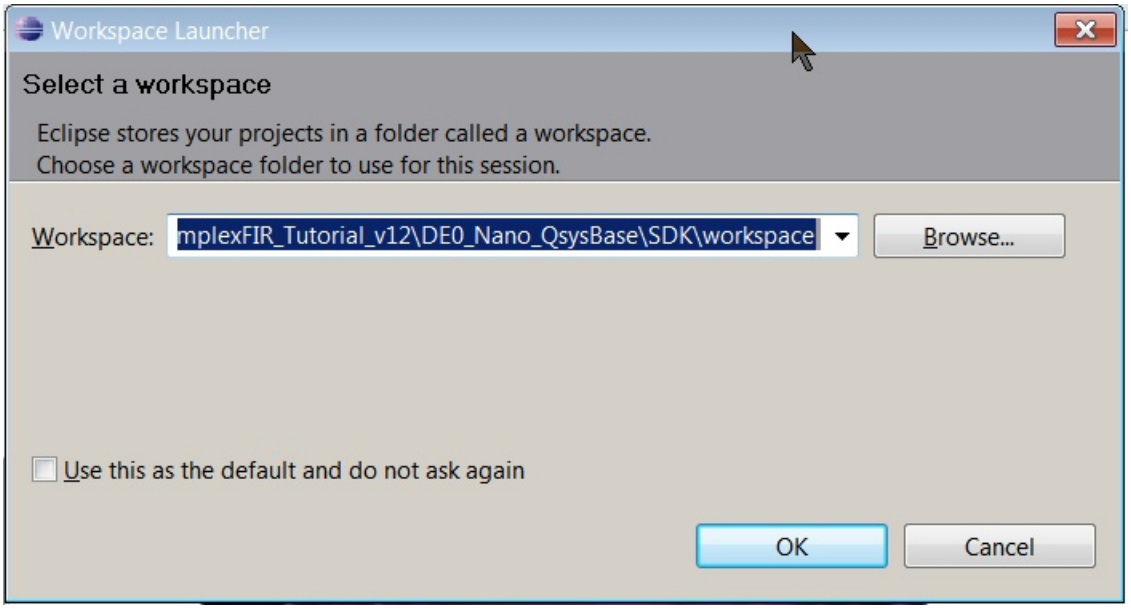
ComplexFIR Filter Tutorial, Step 9

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

Create Software Projects

Begin by starting the **Nios II IDE** from **Quartus II** via **Tools** -> **Nios II Software Build Tools for Eclipse**.

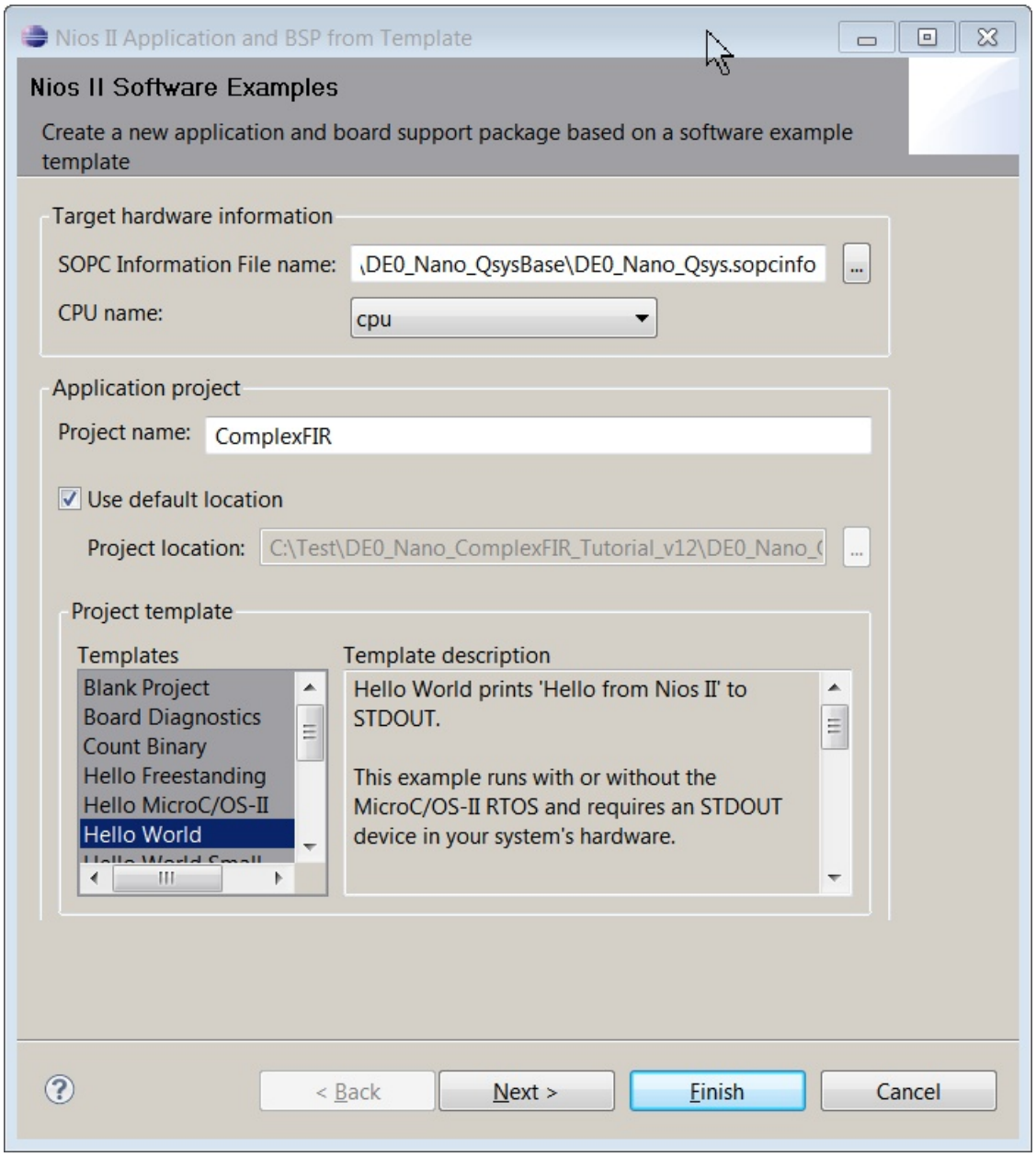
When the **Workspace Launcher** window appears, set the **Workspace** to the directory <your path>\DE0_Nano_ComplexFIR_Tutorial_v12\DE0_Nano_QsysBase\SDK\workspace as shown below:



Create a new project to manage the **ComplexFIR** software files. Select **File** -> **New** -> **Nios II Application and BSP from Template** and a dialog box will appear. Set the following:

- **SOPC Information File name:** <your path>\DE0_Nano_ComplexFIR_Tutorial_v12 \DE0_Nano_QsysBase\DE0_Nano_Qsys.sopcinfo
- **CPU name:** cpu
- **Use default location:** checked
- **Project name:** ComplexFIR
- **Templates:** Hello World

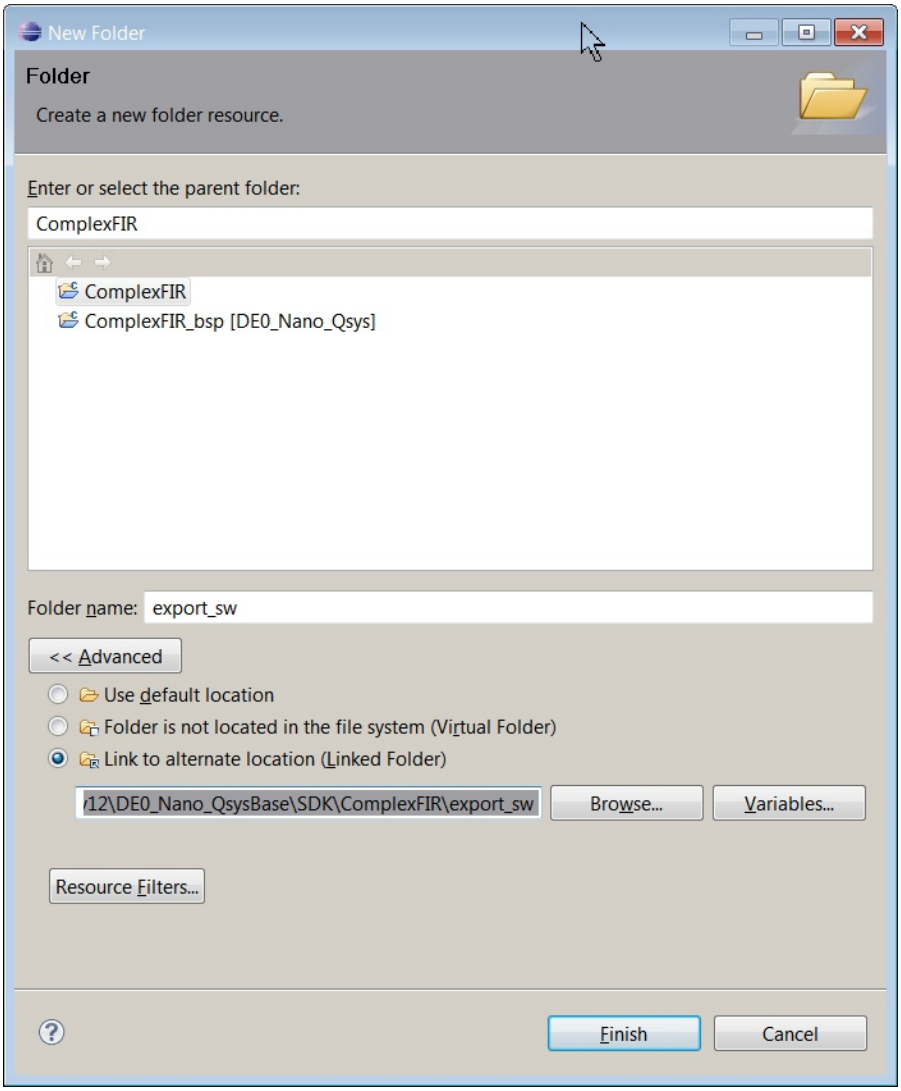
The dialog box will now look as below:



Click **Finish** to create the new project along with its associated BSP. Two new projects **ComplexFIR** and **ComplexFIR_bsp** should appear in the **Project Explorer** window.

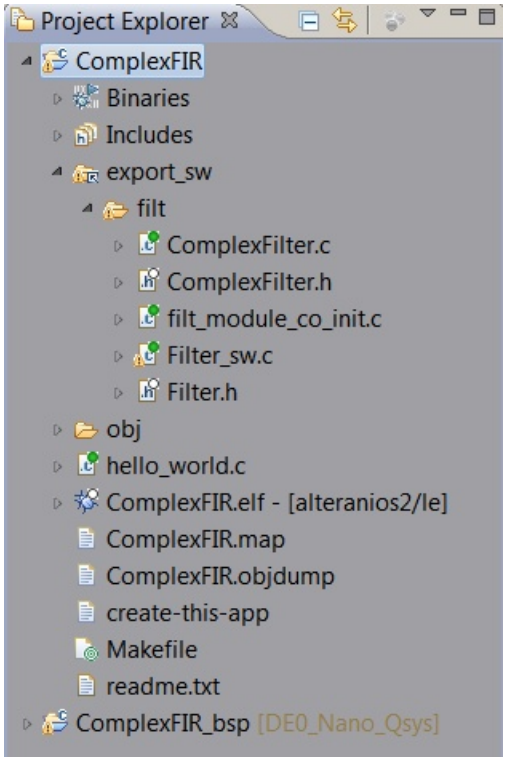
Add Exported Software to ComplexFIR Project

To add the exported software from CoDeveloper, right-click the **ComplesFIR** project in the **Project Explorer** and then select **New -> Folder**, a **New Folder** window will appear. Click the **Advanced** button, select **Link to alternate location (Linked Folder)**, then type the path or click **Browse** to select the <your path>\DE0_Nano_ComplexFIR_Tutorial_v12 \DE0_Nano_QsysBase\SDK\ComplexFIR\export_sw directory already containing exported software files from CoDeveloper. The **New Folder** window should now appear similar to below:



Click the **Finish** button to continue.

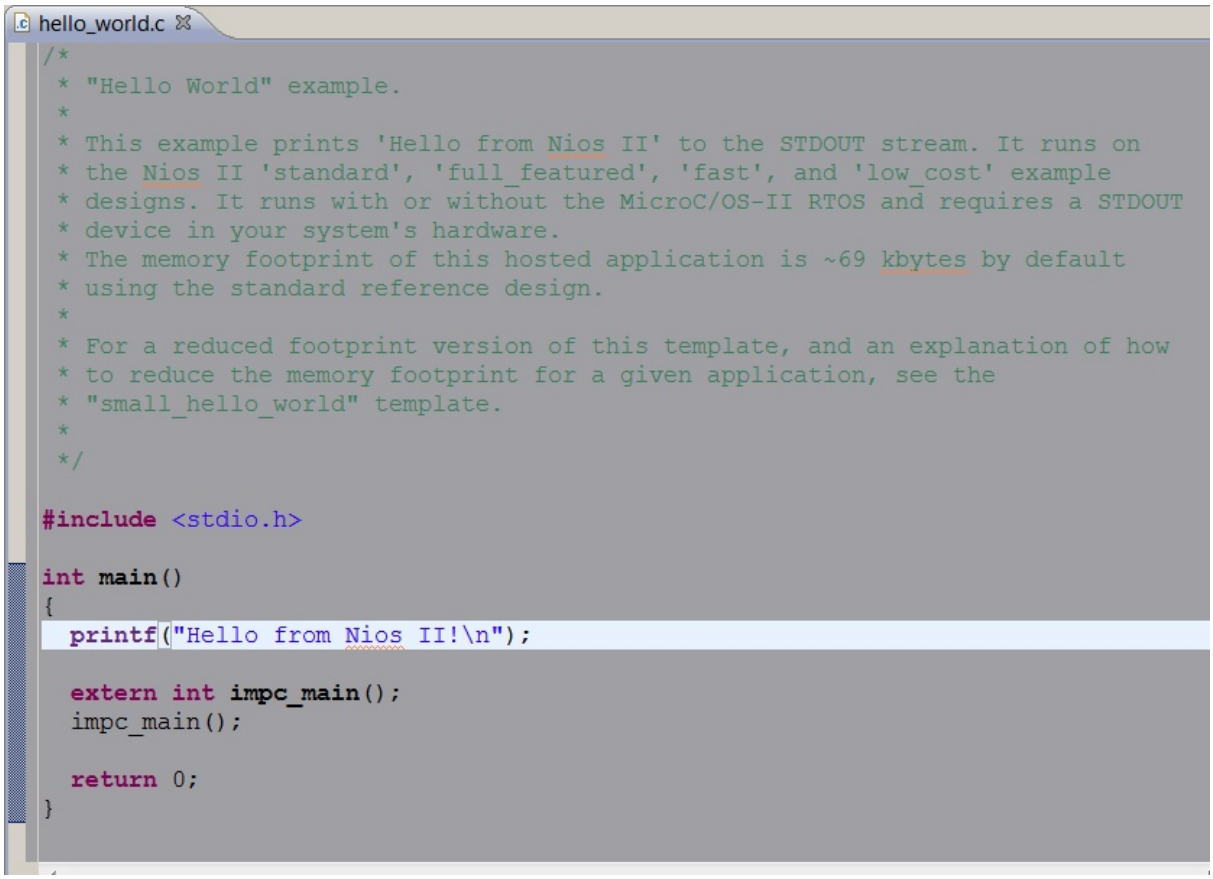
To add the exported source code to be built as part of the **ComplexFIR** project, in the **Project Folders** view, first expand the directory **ComplexFIR** -> **export_sw** -> **filt** to view all the exported source files. For each of the *.c files, right-click and select Add to Nios II Build, a green dot will now appear next to each of the files. Upon completion, the Project Folders view must look as below:



The **ComplexFIR** project was started using a **Hello World** template which must now be modified in order to call the **main** routine in the exported source files. To make this change, open **hello_world.c** by double-clicking the file name in the **Project Explorer** view. Scroll down to the **main()** routine and add the two lines:

```
extern int impc_main();
impc_main();
```

to make it look exactly like what appears below:



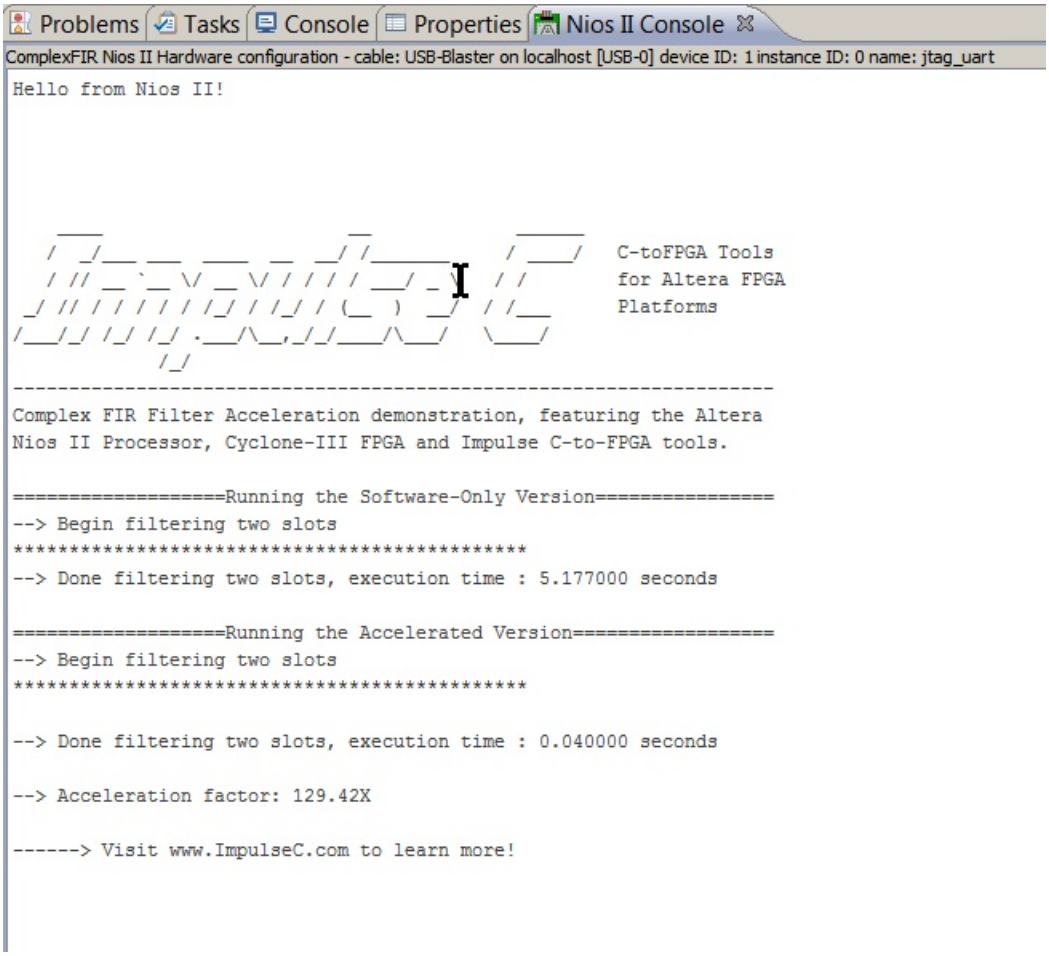
Save the source file via Ctrl+S.

Build and Run ComplexFIR Project

Now build the project by right-clicking the **ComplexFIR** project and selecting **Build Project**. The IDE will build both the **ComplexFIR_bsp** which includes a driver for the Impulse C hardware module created by CoDeveloper, 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 **Nios II Console** window similar to below:



The result tells us that with hardware acceleration, the execution of the ComplexFIR filter is 129.42 times faster than the software-only version running on the Nios II CPU as configured.

Congratulations! You have successfully completed the Complex FIR tutorial.

See Also

Complex FIR Filter on the Nios II platform