

**Introduction to High-Level Synthesis**

To learn more, visit <http://fpgauniversity.intel.com>

© Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services. Other names and brands may be claimed as the property of others.

# Contents

[Contents 2](#_Toc31883111)

[1 Introduction 3](#_Toc31883112)

[2 Assumptions 3](#_Toc31883113)

[3 Lab 1: HLS Implementation of a 32-bit Square Root Component 3](#_Toc31883114)

[Summary 3](#_Toc31883115)

[Lab Instruction 5](#_Toc31883116)

[3.1 Windows Command Prompt Environment Setup 5](#_Toc31883117)

[3.2 HLS Emulation of floorsqrt Function 7](#_Toc31883118)

[3.3 HLS Cosimulation of floorsqrt Function 8](#_Toc31883119)

[3.4 Opening Compilation Reports (RTL and HLS) 12](#_Toc31883120)

[3.5 Unarchiving .qar Files in Quartus Prime Command Line 15](#_Toc31883121)

[3.6 Appending new HLS Component to existing Embedded Nios System 16](#_Toc31883122)

[3.6.1 Setting up USB Blaster Connection 19](#_Toc31883123)

[3.7 Software Design with new HLS Component System 20](#_Toc31883124)

[4 Document Revision History 26](#_Toc31883125)

# Introduction

This workshop demonstrates the flow of a basic HLS program for adding a new component to a Platform Designer system. The HLS compiler will synthesize a floorsqrt component described in HLS to an RTL design which will be connected as an Avalon Slave device to an existing embedded system in Platform Designer. The last section of this lab will compare the compilation reports generated by Quartus and HLS and demonstrate interaction with a FPGA development kit using the Nios II Software tool in the Eclipse environment.

At the end of this guide, the user will be able to run and simulate an Embedded Nios II project with the HLS compiler and Platform Designer tool on Quartus Lite. Flows using the GUI and the command line will be demonstrated.

# Assumptions

This user guide assumes you have:

* Quartus Prime Lite Edition 18.1
* ModelSim
* Windows Command Prompt
* Windows Visual Studios 2010 Professional (Trial Version)
* Understanding of Quartus® Prime Lite development tools
* Platform Designer and Nios II Processor
* Nios II Software Build Tools
* hls\_quickstart\_source\_files.zip file

# HLS Implementation of a 32-bit Square Root Component

## Summary

The lab hardware is constructed with the components shown below. Intel utilizes the Platform Designer network-on-chip interconnect to connect the master and slave devices together.

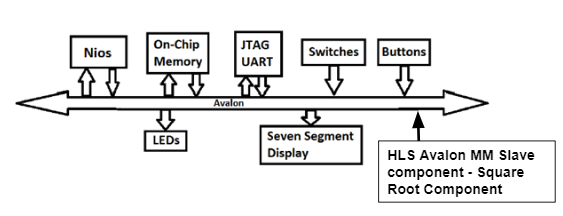


Figure 1: Nios II Based System

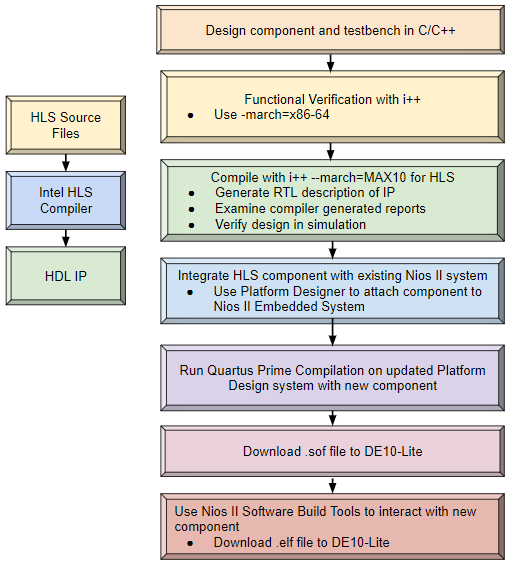


Figure 2: HLS Design Flow

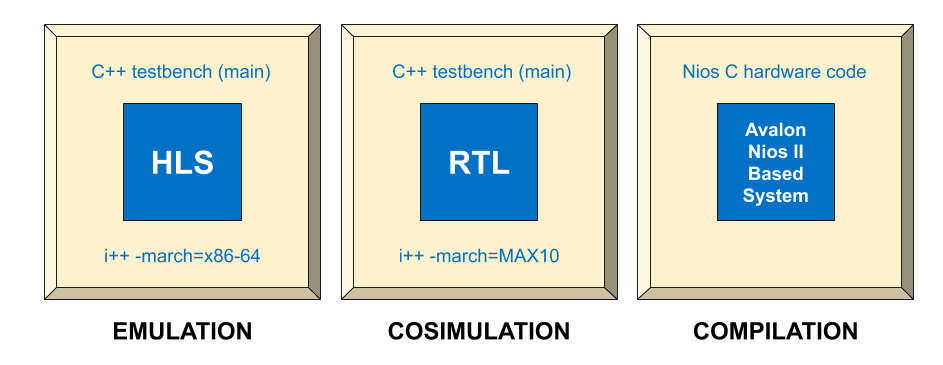


Figure 3: Breakdown of Emulation, Cosimulation, and Compilation Processes

The HLS procedure contains three main steps to integrate a C/C++ algorithm down into RTL. The first step is called ***emulation***, which is a functional verification of the C/C++ source code that was written. This compiler is essentially the exact same as the standard g++ compiler used in the C/C++ programming language; this will not generate any RTL. The next step is called ***cosimulation***, which generates both RTL & compiler reports for a targeted FPGA device, allowing you to verify your design through waveform simulation. The final step involves running an Intel Quartus Prime compile on all the generated RTL files to obtain a more accurate report on resource utilization and the clock’s fmax. This is essentially the same as step 2 with one additional part.

This section will walk you through this HLS flow by first performing emulation, followed by cosimulation, and finally a Quartus compile.

## Lab Instruction

## Windows Command Prompt Environment Setup

The following steps describe how to set up Windows Command Prompt needed for this lab which uses Intel® Quartus Prime software and HLS.

* Extract the downloaded hls\_quickstart\_source\_files.zip file in File Explorer and save the extracted folder in the Downloads folder.
* Open a Visual Studio x64 Win 64 Command Prompt (2010) terminal window by typing Visual Studio x64 in the Windows search bar.

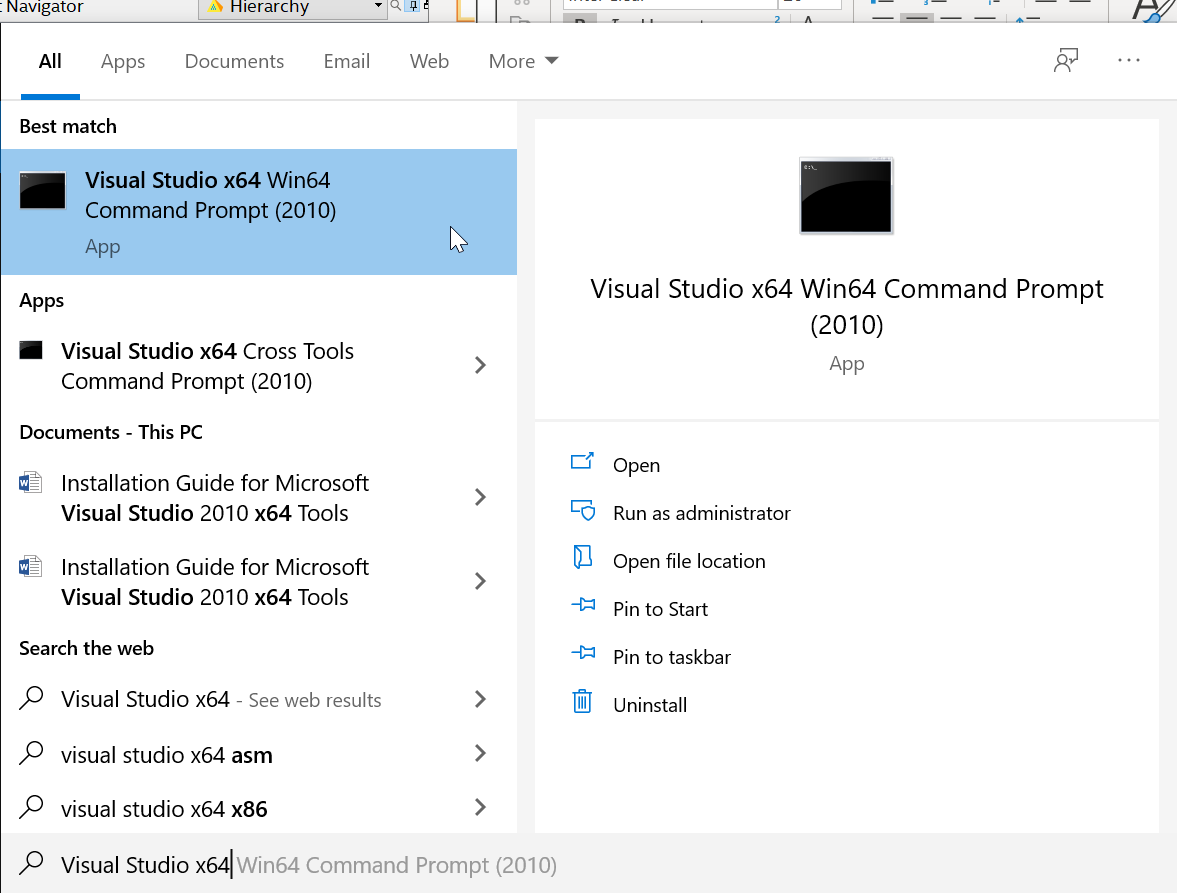


Figure 4: Visual Studio x64 Win64 Command Prompt (2010) Application

* Change your current working directory to the extracted folder located in Downloads. Note: the username will be different for your computer. To copy and paste all commands listed in this guide, refer to the **commands.txt** file included hls\_quickstart\_source\_files folder.

cd C:\Users\**username**\Downloads\hls\_quickstart\_source\_files

* Copy and paste the following command into the terminal. This will initialize the Command Prompt environment and output the correct pathways.

setup.bat

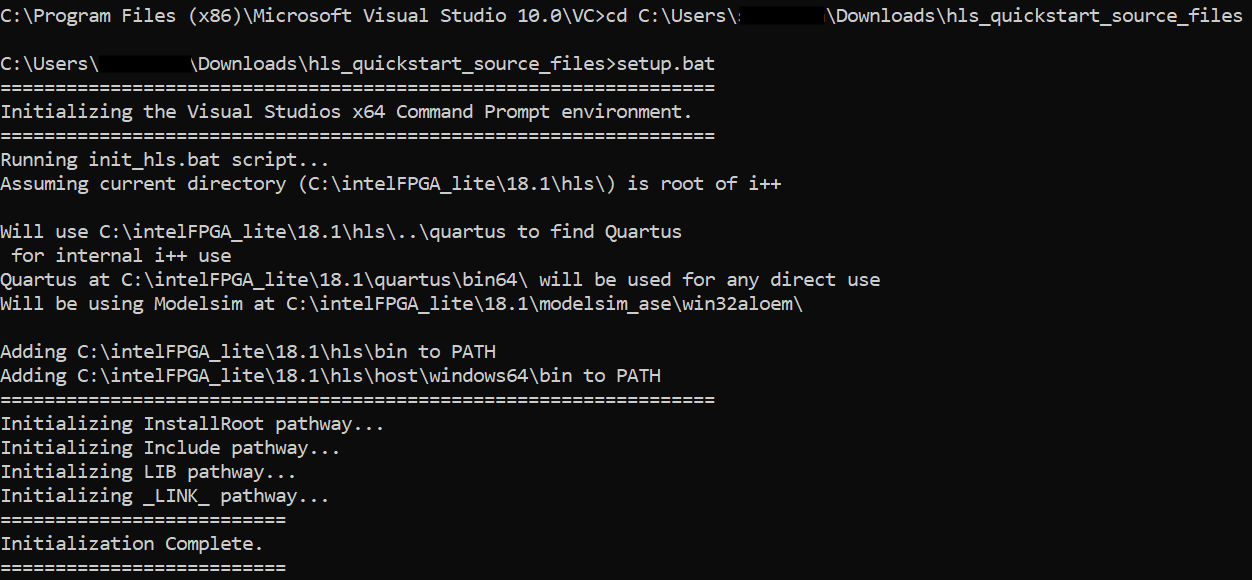


Figure 5: Initializing HLS Command Prompt Environment

You now have all the necessary resources and are ready to begin the exercises!

## HLS Emulation of floorsqrt Function

This section will walk you through the HLS flow for a component that mimics the floorsqrt() function in a C++ library. Given an integer x, the function will find the square root of it. If x is not a perfect square, the function will return floor(√x) which truncates a double downward to the nearest integer.

Emulation will be performed first, then cosimulation, and finally a Quartus compilation.

* View the C++ implementation of the floorsqrt() function in Microsoft Visual Studio by copying and pasting floorsqrt.cpp in the terminal. Press Enter and then select Microsoft Visual Studio to open the file.
* Close the Microsoft Visual Studio window.
* Run an emulation compile on floorsqrt.cpp program by executing the following command in Command Prompt. This command performs the same compilation as the general g++ compiler used in C++ but using the i++ Intel® HLS Compiler instead.

i++ -march=x86-64 floorsqrt.cpp -o emulation

Note: In **cosimulation**, there are HLS-unique directives such as ‘component’ (used to synthesize a C++ function down to RTL) and ‘#pragma max concurrency N’ (used to pipeline loop instructions). In **emulation**, these directives are simply ignored in compilation because it is set to mimic the g++ compilation flow.

* Run the executable named ‘emulation’ generated from the command performed in the prior step. Type the following:

emulation.exe

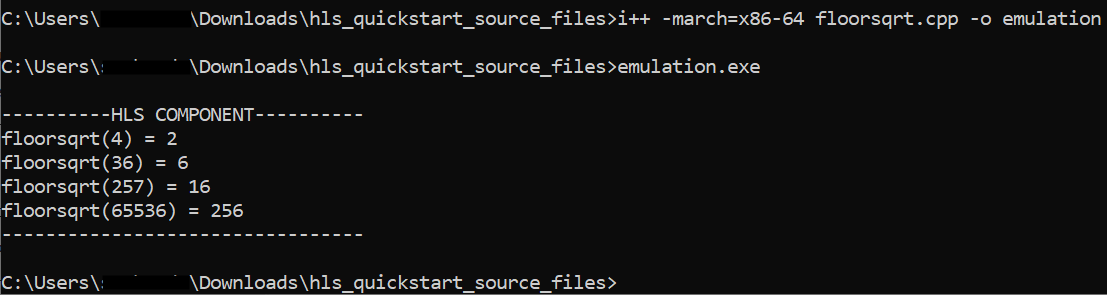


Figure 6: Terminal output after running 'emulation' executable

The output produced from the emulation is nothing more than a functional verification of the C++ code. From the output above, we can evaluate that the algorithm produces the correct output for the chosen inputted values. The values chosen are within the size limit of a 32-bit integer.

Note: We also have no knowledge of how many clock cycles a given state remains in after emulation compilation. But because of its fast compilation time, emulation provides a quick turn-around in functional verification of code compared to RTL design.

## HLS Cosimulation of floorsqrt Function

The following is generated when executing a cosimulation compilation with the HLS compiler in Standard edition:

* An executable which will run the testbench (the main() function in C++). Any calls to functions from the testbench will be part of the simulation.
* All files necessary to include IP in an Intel® Quartus software project (i.e. .qsys, .ip, .v, etc.)
* Component hardware implementation report (estimation)
* Intel® Quartus software project (.qpf) to compile all IP and generate a more accurate report
* A memory map of the arguments in a header file <results>.prj/components/<component\_name>\_csr.h. This file provides the C macros for a master to interact with a slave and was created because the argument label hls\_avalon\_slave\_register\_argument was used.
* Run a cosimulation compile on floorsqrt.cpp by executing the following command in the Command Prompt terminal:

i++ -ghdl -march=MAX10 floorsqrt.cpp -o cosimulation

Note: Compilation time takes ~3 minutes for this design.

This is when HLS directives placed within the C++ code take an effect. The **component** identifier placed in front of function floorsqrt specifies to the compiler to generate RTL for this function and simulate any outputs from it. The **hls\_avalon\_slave\_component** specifies that the floorsqrt function is an Avalon Slave Component to be added to an existing Avalon system.

**#pragma max concurrency 1** specifies to iterate through the loop without pipelining and showcases common usage of a popular HLS directive.

This compilation will generate RTL for any functions marked as a component, a Quartus project, and a compilation report.

* Run the executable Cosimulation generated from the previous command. This executable runs the main() function and serves as the testbench in simulation.

cosimulation.exe

Note: Execution will generate a .wlf waveform file and output the same results from emulation in the terminal window.

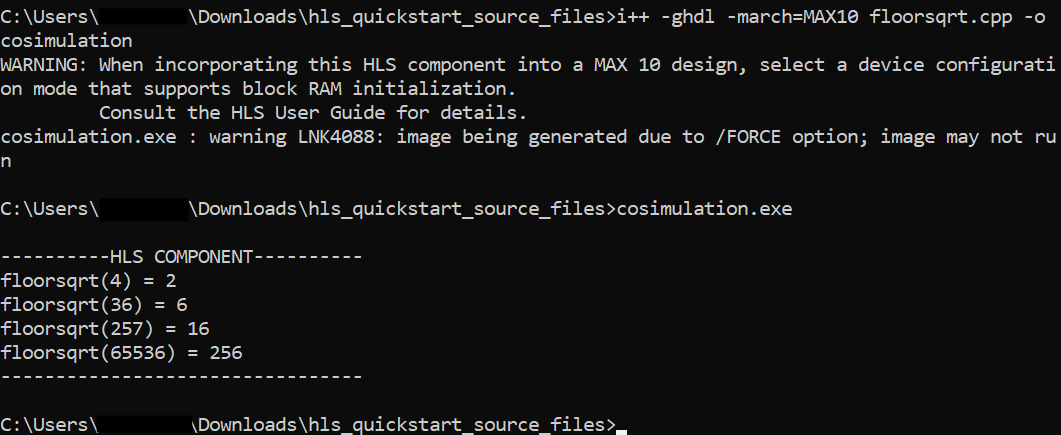


Figure 7: Cosimulation Output Results

* Launch the *vsim.wlf* file generated from the previous step through ModelSim-Altera.

vsim cosimulation.prj/verification/vsim.wlf &

* In the Transcript window located at the bottom of the ModelSim session, type in the following commands to run the simulation script. This will be used to load and format the Wave window with the required signals, radices, cursors, and zoom level intact.

do wave.do

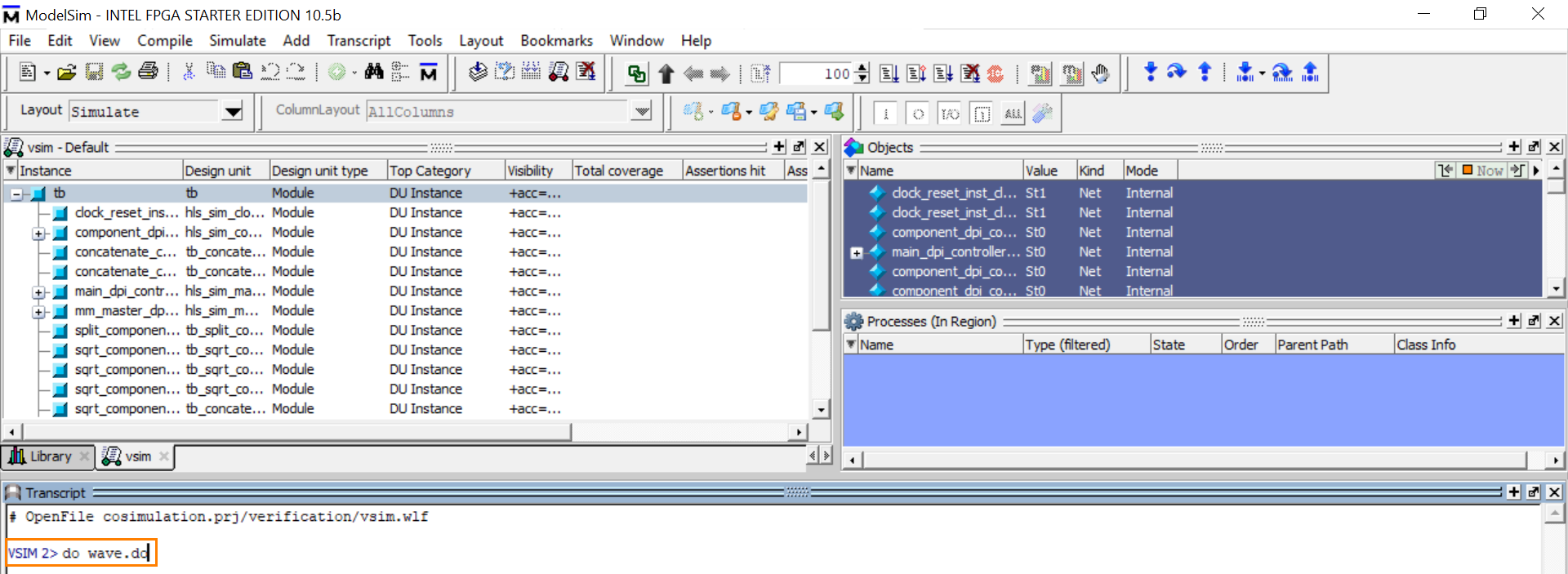


Figure 8: Run wave.do ModelSim Simulation

* Expand the *Wave* window which opened from the previous command. 
* Press F to view the simulation from start to finish. You can zoom in/out with the I/O keys of your keyboard.

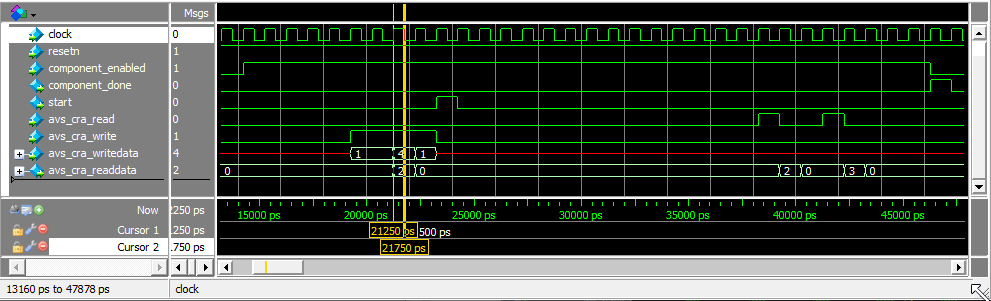


Figure 9: Clock Cycle (500ps)

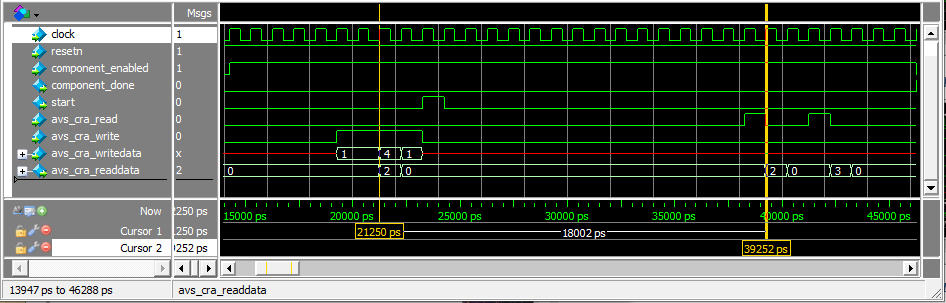


Figure 10: Floorsqrt Calculation Clock Cycles

Upon closer inspection, you will notice a couple of important things.

1. The read-in argument for the floorsqrt() function enters through avs\_cra\_writedata and the output of the floorsqrt() function is returned through avs\_cra\_readdata.
2. In Figure 6 above at the yellow line, the function is executed as floorsqrt(4) = 2. At time 21250 ps, the radicand argument 4 is written. It takes 18000 ps for the squareroot component to calculate and read the answer 2. To determine the number of clock cycles required to complete the square root calculation, we can divide the component’s total calculation time (18000 ps) and by the clock period (500 ps). 18000 ps / 500 ps = 36; therefore, it takes 36 cycles for the square root component to calculate the square root of 4.
3. Notice that it takes even longer to calculate the square root of 36. At 57250 ps, 36 is read into the component. At 90250 ps, the value 6 is outputted. Therefore, it takes 90250 ps – 57250 ps = 33000 ps or 66 clock cycles. This is **1.83 times** longer than calculating floorsqrt(4).
4. For floorsqrt(65536), it takes 237250 ps – 184250 ps = 53000 ps or 106 clock cycles. This is **2.94 times** longer than calculating floorsqrt(4).
5. For floorsqrt(257), it takes 166250 ps – 108250 ps = 58000 ps or 116 cycles. This is **3 times** the number of clock cycles required to calculate floorsqrt(4). Notice that even though 257 is smaller than 65536, it takes longer to calculate 257 is not a square number unlike 4, 36, and 65536.
6. The return data is undefined for 19250 ps or 38.5 clock cycles at the beginning of the simulation. The undefined output is mainly due to the way the compiler schedules the busy loop and state assignments.

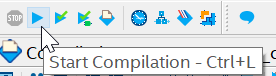
The increased latency between calculation times and during calculations is very apparent, however, this decrease in calculation speed is expected due to the nature of designing components at a higher level of abstraction. Generally, when using a higher-level design tool such as HLS, the designer has less control at the gate level given that more information is abstracted away. Therefore, while we can use simplified syntax with C++ code, speed comes at a cost at the gate level.

## Opening Compilation Reports (RTL and HLS)

This section observes the RTL Compilation Report generated after a Quartus compilation.

* Close the waveform generator window and return to the Visual Studio Command Prompt window.
* Press **Ctrl+C** to close the current job. In File Explorer, navigate to the directory below.

C:\Users\**username**\Downloads\hls\_quickstart\_source\_files\cosimulation.prj\quartus\quartus\_compile.qpf

* Double-click on the **quartus\_compile.qpf** file. This will open the current HLS project generated after cosimulation.
* Once the project is open, compile the design by clicking the blue compilation button.  Compilation will take ~2-3 minutes.
* Open the Compilation Report by typing **Ctrl + R**.
* Observe the number of registers used from the Verilog-compiled RTL.

This section observes the HLS Compilation Report generated after HLS cosimulation compilation.

* Return to the Visual Studios terminal window. Navigate to the directory containing the HLS report generated from cosimulation. Type the following:

cd ../reports/

* Type the following. This command will copy the full path to this directory to your clipboard:

cd | clip

* Open a web browser such as Google Chrome or Internet Explorer.
* Paste the path returned from the clip command into the address line. Type the following after pasting the pathway into the command line:

report.html

The address should look something like this, but the username revised to match your unique username:

C:\Users\**username**\Downloads\hls\_quickstart\_source\_files\cosimulation.prj\reports\report.html



Figure 11: Address Line in Search Browser

The following window should appear:

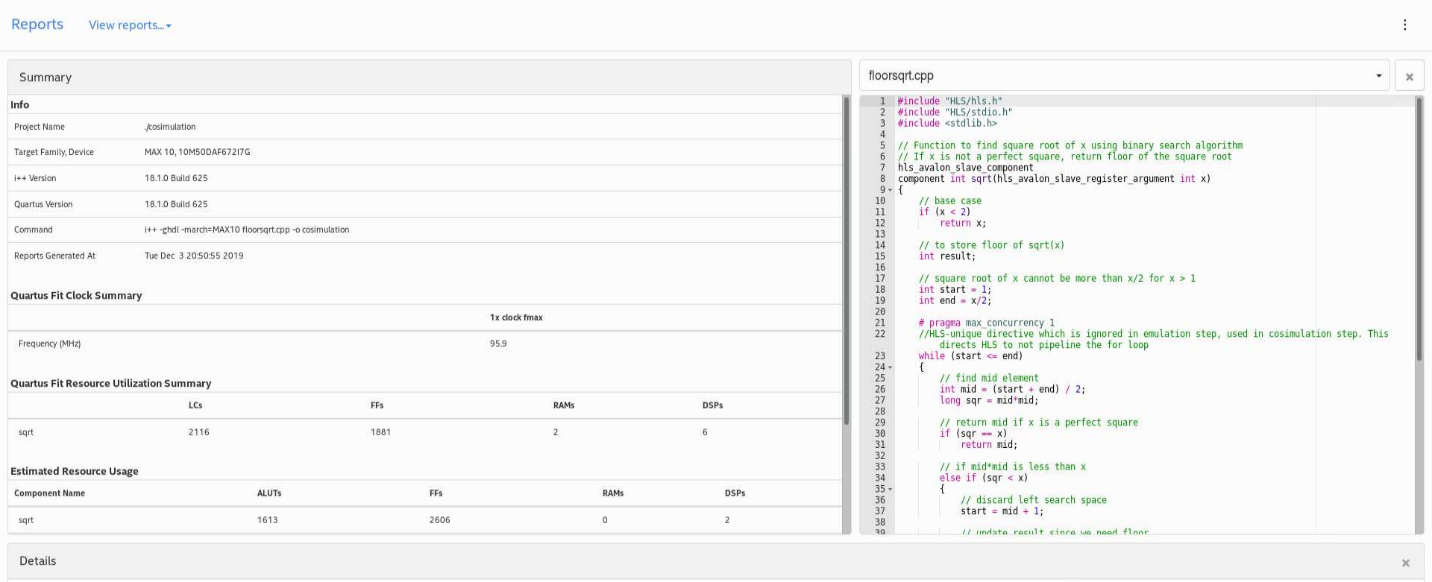


Figure 12: HLS Generated Report

The high resource utilization is expected due to the overhead in the HLS tool. Generally, the higher level a design tool is, the less control and greater abstraction the designer has at the gate level. In terms of our floorsqrt function design, this overhead is most easily seen in the durations given to enable and disable the component and the number of registers used. In RTL, the clock cycles can be defined to a given state because the counter is based on the positive edge of the input clock. In C++, staying in a state for a definitive number of clock cycles is not as simple because instructions are unable to execute based on the rising or falling edge of a clock cycle, unlike in RTL.

* View the component in **Component Viewer**. Under the drop-down menu **View reports…**, select **Component Viewer**.

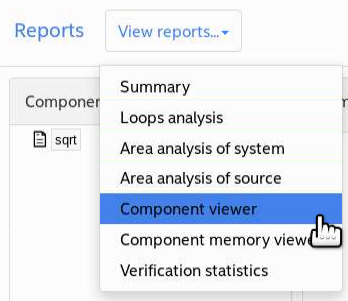


Figure 13: Component Viewer

* Click the component **sqrt** under the component list and the sqrt component will appear in the viewer window.

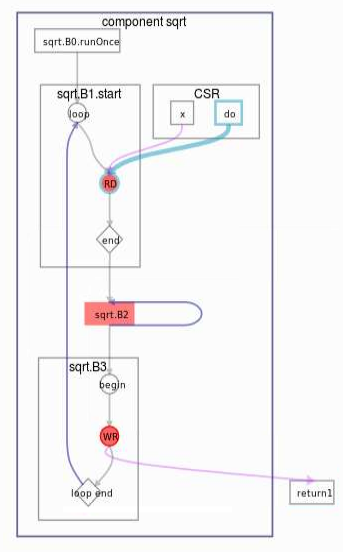


Figure 14: sqrt Component in Component Viewer Report

* Hover over the different flow stops in the sqrt system to see latency times and bit widths of input arguments.

The next sections involve adding this newly generated HLS component to an existing Max 10 Embedded Nios II system in Platform Designer.

## Unarchiving .qar Files in Quartus Prime Command Line

This section will cover how to unarchive .qar files in the Quartus Shell command line.

* Change your current open directory to the hls\_quickstart\_source\_files folder.

cd ../../

* Enter the following in the terminal to restore a project archive in the Quartus Prime Shell:

quartus\_sh --restore -output floorsqrt\_restored floorsqrt.qar

* The Quartus Prime Shell should state that the job was successfully completed. All source and design files will be saved in the destination folder named floorsqrt\_restored.

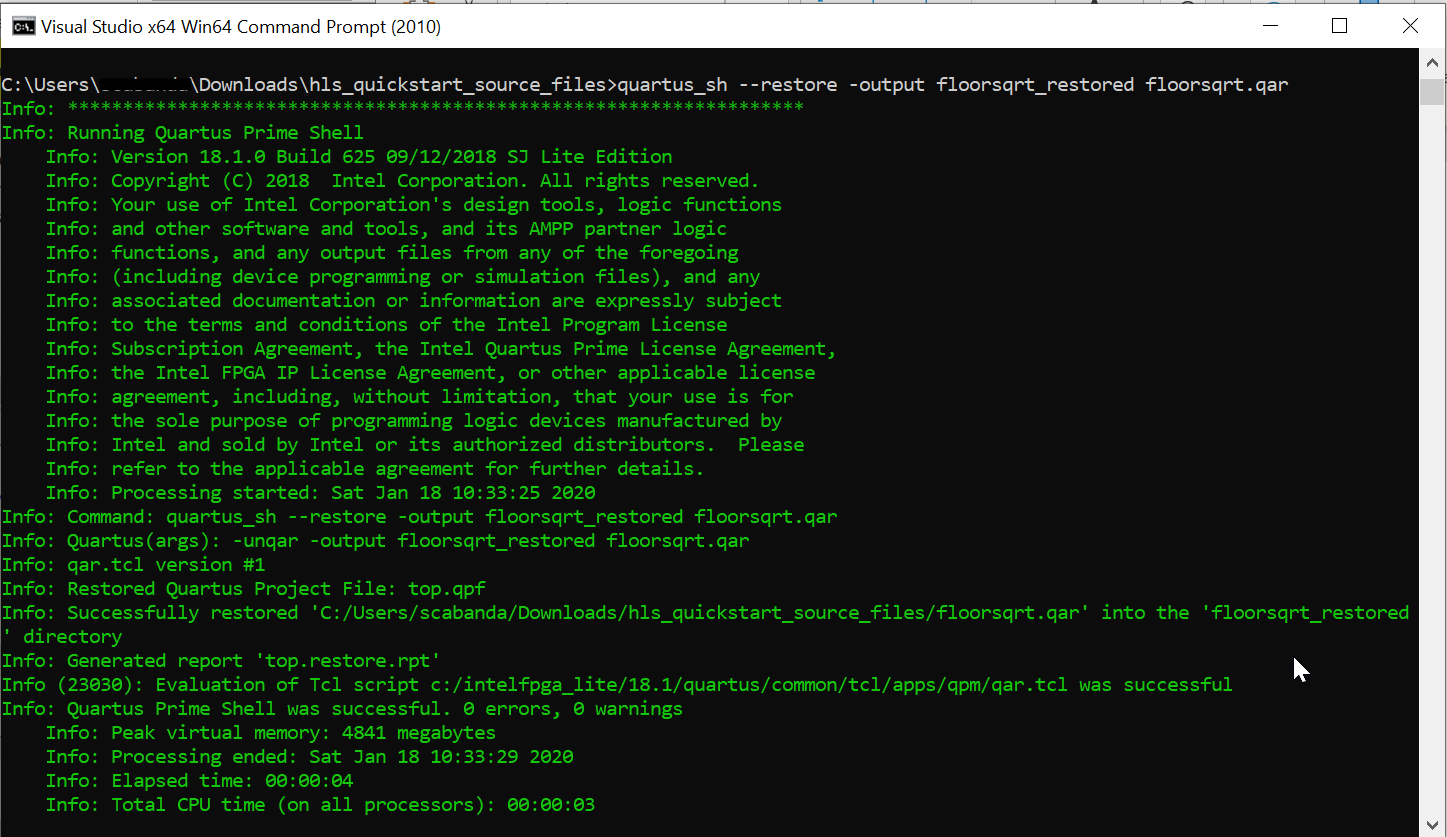


Figure 15: Successful Restoration of Quartus Archive File

## Appending new HLS Component to existing Embedded Nios System

We will now add the newly created floorsqrt component to an existing Embedded Nios II Processor system in Platform Designer. To use the HLS compiler-generated IP in a Platform Designer system, you must first add the directory to the IP search path or the IP catalog.

* Navigate to the floorsqrt\_restored folder.

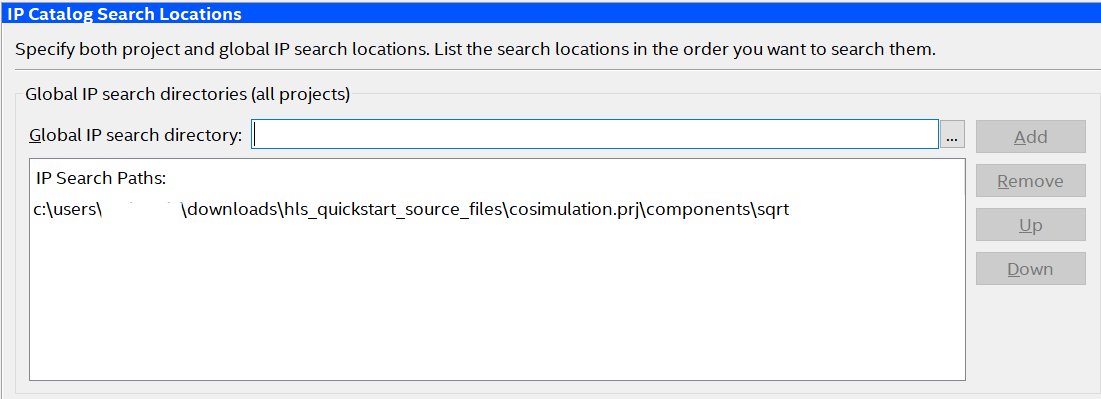
cd floorsqrt\_restored

* Open the Quartus project in the Quartus GUI.

quartus top.qpf &

* In Intel® Quartus Prime, click **Tools** > **Options**.
* In the Category pane, expand **IP Settings**, and click **IP Catalog Search Locations**.
* In the **Global IP Search Directory** dialog box, click the **…** symbol to add the path to the directory that contains the .qsys file to IP Search Paths.
* To find the sqrt component, specify the path like the following. Be sure to click **Add** when finished.

hls\_quickstart\_source\_files\cosimulation.prj\components\sqrt

Figure : IP Catalog Search Paths

* Press **OK** and close the Options window.
* In the Project Navigator Panel, select **Files** in the drop-down menu and double-click on the **nios\_setup\_v2.qsys** file. This will open the existing Nios Processor system in the Platform Designer tool.

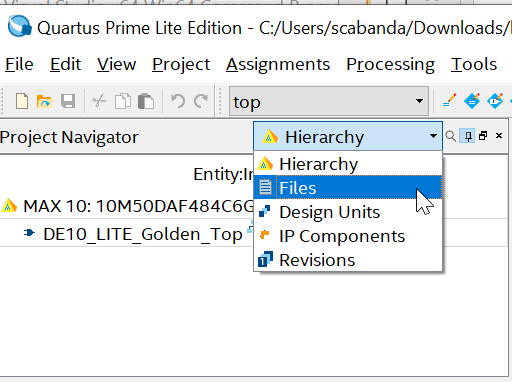


Figure 17: Project Navigator Drop-down Menu

* In the **IP Catalog**, double-click the **HLS** library directory and add the **sqrt** component to the Platform Designer system by selecting it and clicking **+ Add …** followed by **Finish**.

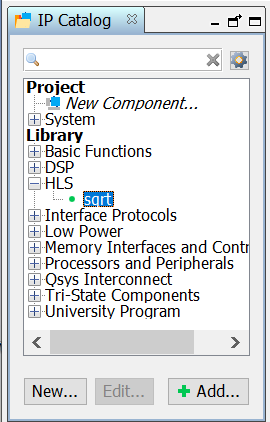


Figure 18: HLS Component in Platform Designer IP Catalog

* Connect the new squareroot component as shown in the figure below.

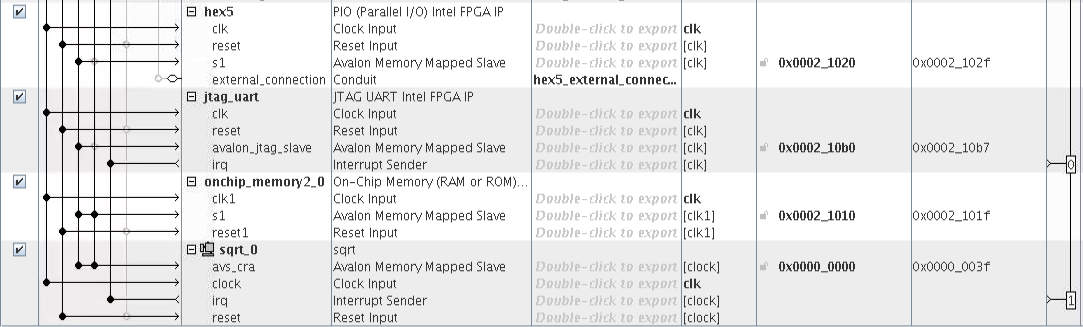


Figure 19: Connections for New Squareroot Component

* Under the **System** drop-down menu, click **Assign Base Address.**
* Click **Generate HDL**, keep default settings and then click **Generate**. Allow Platform Designer to finish before closing the window.
* Close the window, wait for Platform Designer to finish generating, thenclick **Finish** to close Platform Designer.
* Recompile the system by pressing **Ctrl + L** on the keyboard or clicking the blue triangle compilation button.

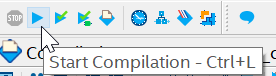


Figure 20: Start Compilation Button

### Setting up USB Blaster Connection

* To download your completed FPGA design to an attached DE10-Lite development kit, start by connecting the USB Blaster cable between your PC USB port and the USB Blaster port on your development kit. Upon plugging in your device, you should see flashing LEDs and 7-segment displays counting in hexadecimal, or other lit up LEDs and 7-segments depending on previous projects that have been downloaded to the local development kit.
* To use the USB Blaster to program your local device, you need to install the USB Blaster driver. Search for **Device Manager** in the Windows Search bar.
* Navigate to the **Other Devices** section of the Device Manager and expand the section below.

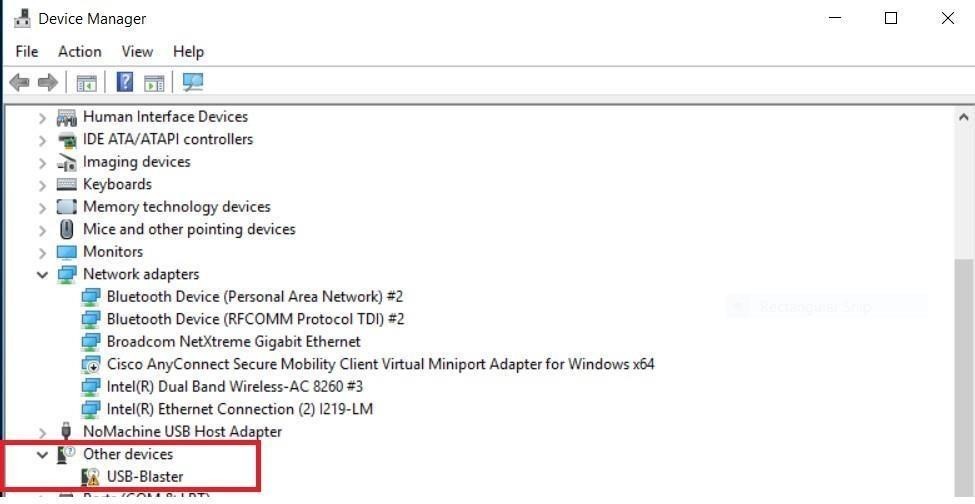


Figure 21: Device Manager Settings for USB-Blaster

* Return to the Quartus Prime Lite window. Under the Tools drop-down menu, open Programmer. The Programmer window should pop-up.

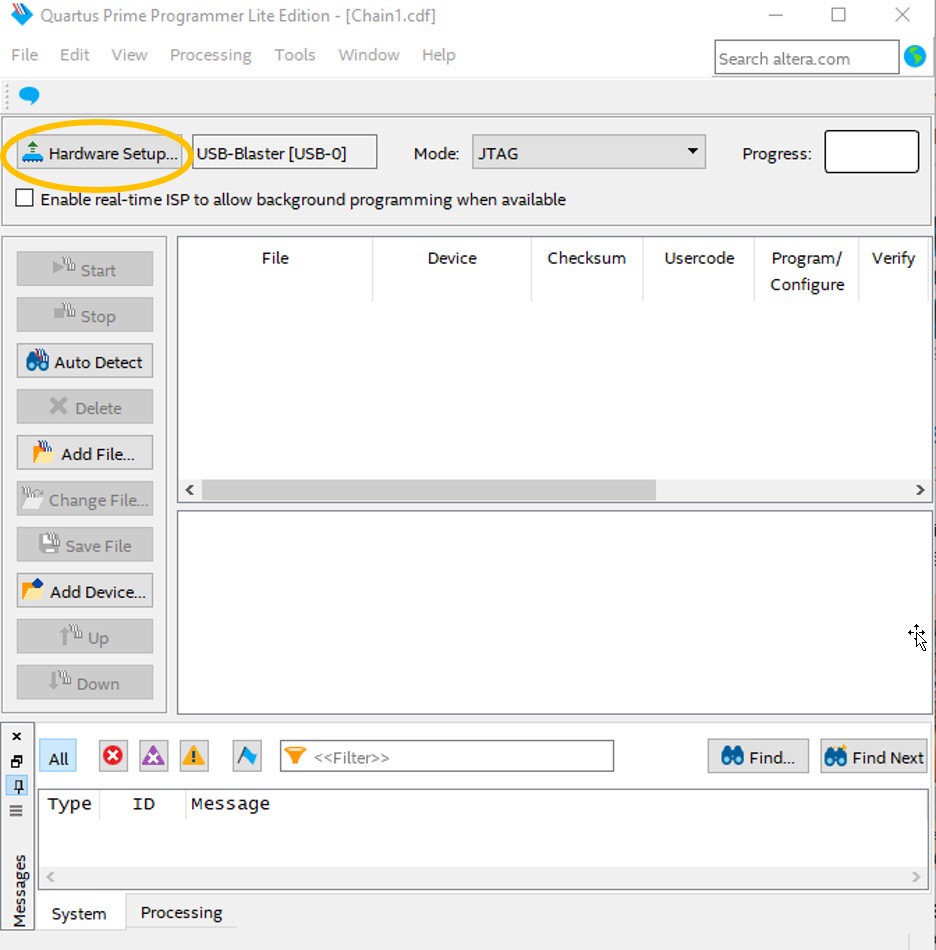


Figure 22: Hardware Setup Window in Quartus Prime Programmer

* Left click on **Hardware Setup…** and then double-click on the USB-Blaster [USB-0] connection. The currently selected hardware should change to “USB-Blaster [USB-0].
* Close the **Hardware Setup** window and select **Add File**.
* Navigate to the hls\_quickstart\_source\_files/floorsqrt\_restored/output\_files directory. Select **top.sof** and click **Open**.
* Click **OK** and press **Start**. The progress bar should turn green and show 100% successful. If it fails the first time, try clicking start a second time.
* Exit programmer (you do not have to save the .cdf file) and reopen the Quartus Prime Lite window.

## Software Design with new HLS Component System

The NIOS Software Build Tools for Eclipse are included as part of Quartus. These tools will help manage creation of the application software and Board Support Package (BSP).

* Launch **Tools** > **Nios II Software Build Tools** for Eclipse. You can use the default location the Eclipse picks for you.

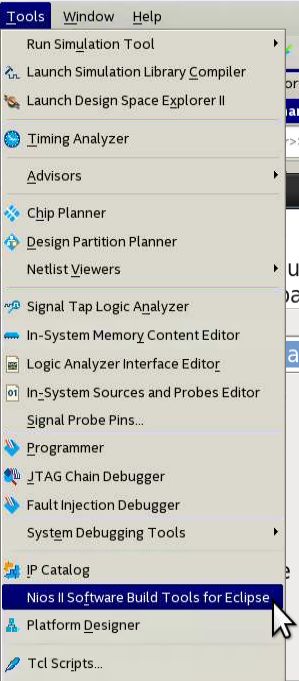


Figure 23: Nios II Software Build Tools for Eclipse

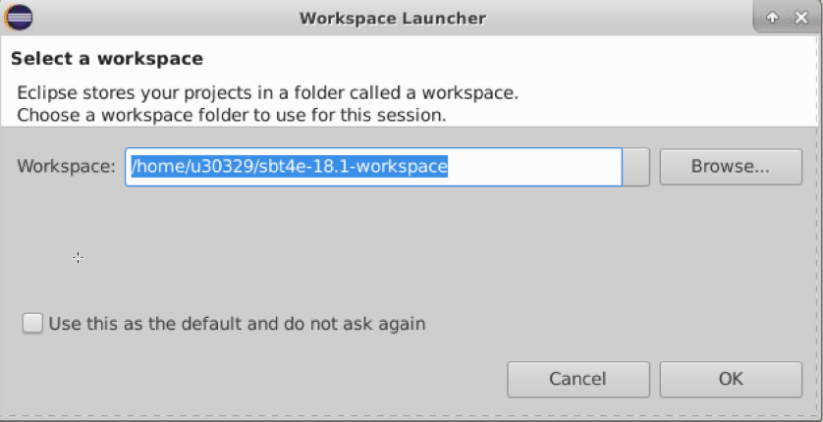


Figure 24: Eclipse Initial Workspace Setup

* Click **OK** in the Workspace Launcher. The Eclipse SBT will launch.
* Under **File** > **New**, select **Nios II Application and BSP from Template**.

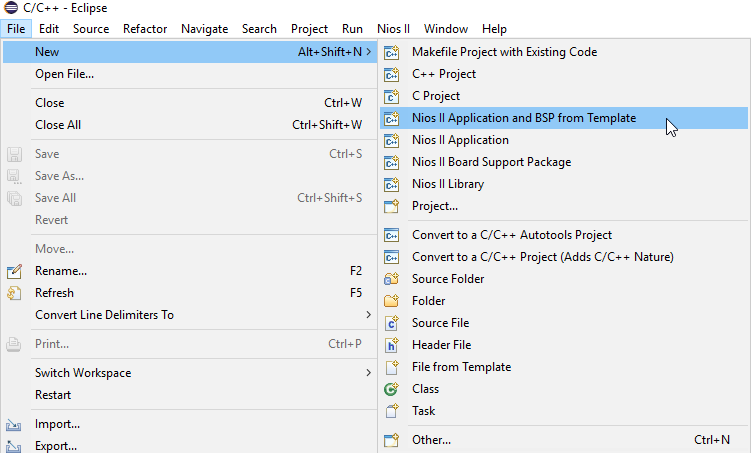


Figure 25: Creating the Initial Project in the Eclipse SBT

The BSP is the “Board Support Package” that contains the drivers for things like translating *printf* C commands to the appropriate instructions to write to the terminal. Within the BSP, there is also an important. Next, you will see a panel that requests information to setup your design.

* Click **…,** navigate to your working directory, and select the **nios\_setup\_v2\_sopcinfo** file. The **.sopcinfo** file informs Eclipse on what your Platform Designer system contains. It should look something like this: C:\Users\username\Downloads\hls\_quickstart\_source\_files\floorsqrt\_restored\nios\_setup\_v2.sopcinfo
* After, selecting the correct .sopcinfo file, Click **OK**.

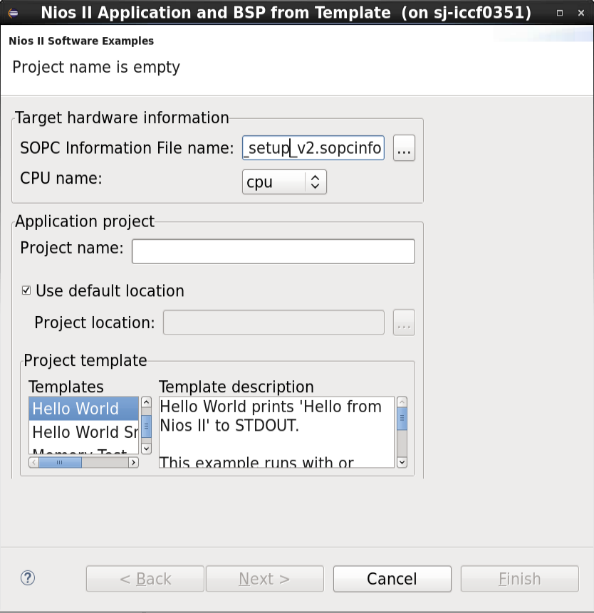


Figure 26: Navigation to the .sopcinfo File

* Fill in the Project name—call it **floorsqrt\_sw**.
* Next, you will be asked to pick a template design. Select the **Hello World Small** application template. This template writes “Hello from Nios II” to the screen.
* Click **Finish.**

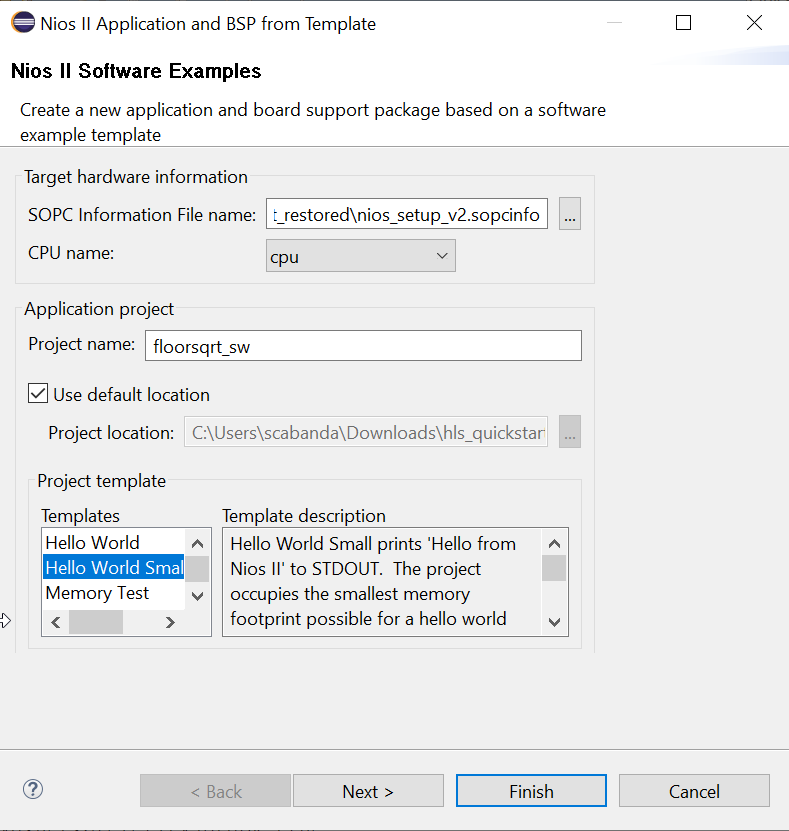


Figure 27: Finished Settings for the Nios II Software Examples Setup Screen

* Two new folders will appear in the Project Explorer window named **floorsqrt\_sw** and floorsqrt\_sw\_bsp. Click on the “**>**” symbol next to the **floorsqrt\_sw** folder.

Now, the **sqrt\_csr.h** file and **floorsqrt\_avalon\_program.c** needs to be added into the floorsqrt folder.

* Right click on the **floorsqrt\_sw** folder and select **Import**.
* Select **General** > **File System** and then browse fortheC:\Users\username\Downloads\hls\_quickstart\_source\_files\cosimulation.prj\components\sqrtdirectory**.**
* Select the **sqrt\_csr.h** which was generated after cosimulation. This header file contains pre-defined C macros that allow us to interact with the component in the **floorsqrt\_avalon\_program.c** program. Click **Finish**.
* Import the **floorsqrt\_avalon\_program.c** into the project. The floorsqrt\_avalon\_program.c is located in the following pathway: C:\Users\username\Downloads\hls\_quickstart\_source\_files\floorsqrt\_avalon\_program.c
* Click **Finish** and then delete the existing hello\_world\_small.c in the Project Explorer window.
* View the **floorsqrt\_avalon\_program.c** in the Eclipse IDE by double clicking on the file.
* Explore the commented header files included in the program: system.h, altera\_avalon\_pio\_regs.h, sqrt\_csr.h.
* Right click on the project **floorsqrt\_sw** and then select **Build Project**. Eclipse will generate a .**elf** file (executable load file). If the **.elf** file does not exist, the project did not build properly. Inspect the problems tab on the bottom of the Eclipse SBT and determine if there are syntax problems, correct, and rerun **Build Project**. Typical problems include missing semicolons and mismatched brackets.
* Next, right click on the project **floorsqrt\_sw** and the select **Run As** > **Nios II Hardware**.
* A window will appear as shown below. Click on the **Target Connection** tab.
  + The connection should indicate the Eclipse has connected to the USB-blaster.
  + If the connection is not identified, you can click **Refresh Connections**.
  + You might need to stretch the window wider to see the Refresh Connections button.
* Click **Run**. If the run button is grayed out but your device shows up under the connections window, you may need to select both **Ignore mismatched system ID** and **Ignore mismatched system timestamp.**

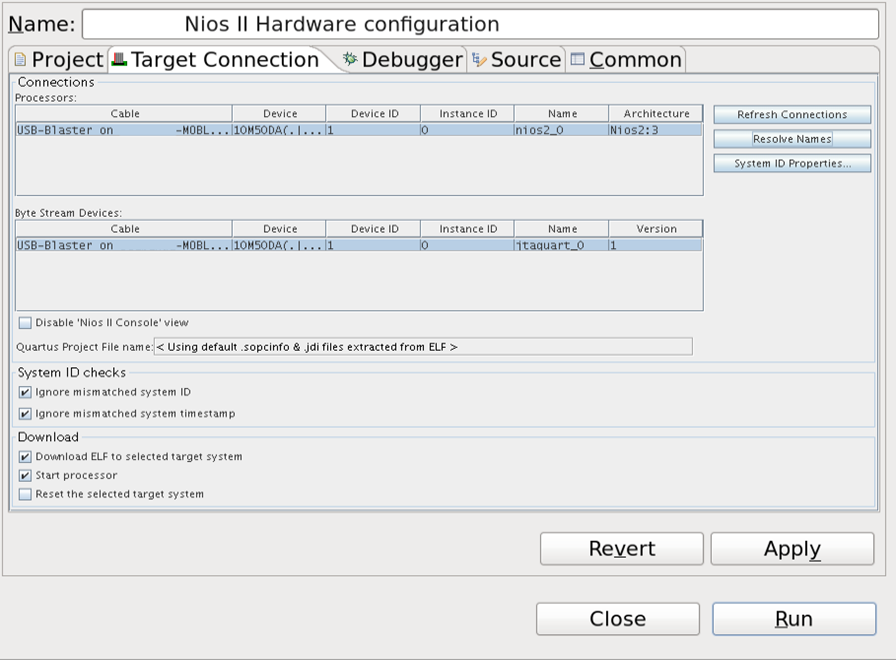


Figure 28: Eclipse SBT Tools after Connection is made to the USB-Blaster

* Now you have hardware and software downloaded into your board. You should observe “Hello from Nios II!” in addition to output of the floorsqrt component with a printed on the Nios II Console tab.

# Document Revision History

List the revision history for the application note.

|  |  |  |
| --- | --- | --- |
| Name | Date | Changes |
| Shawnna Cabanday | 11/26/2019 | Initial Release of guide |
| Shawnna Cabanday | 12/4/2019 | Revision for NoMachine implementation |
| Shawnna Cabanday | 1/19/2020 | Revision for downloaded Quartus Lite implementation |
| Shawnna Cabanday | 1/27/2020 | Minor revisions, added new figures |
| Shawnna Cabanday | 1/28/2020 | Added more information in ModelSim simulation section, minor grammar revisions and mistakes |
| Shawnna Cabanday | 2/6/2020 | Minor grammar revisions, removed hyperlink to location of source\_files.zip, removed section on quartus\_compile.qpf command executable |