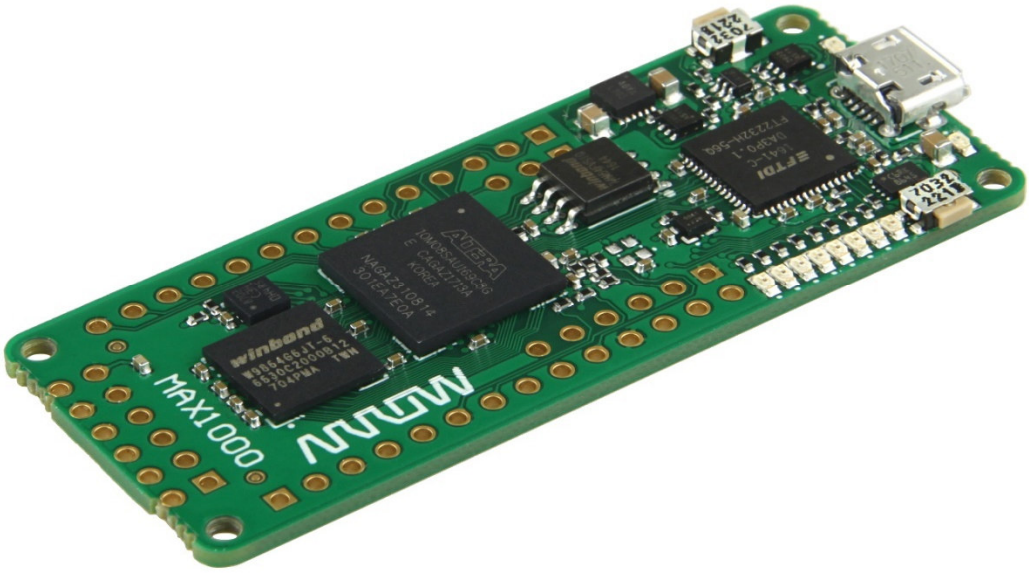
# MAX1000

# Nios II Soft Core Lab




**Please read the legal disclaimer at the end of this document.**

**Revision 1.0**

## Contents

# 1. Introduction

This tutorial provides comprehensive information to help you understand how to create a software project for a Nios II processor system in an Intel FPGA and run the software project on your MAX1000 board. The Nios II processor core is a soft intellectual property (IP) processor that you download (along with other hardware components that comprise the Nios II system) onto an Intel FPGA. This tutorial introduces you to the basic software development flow for the Nios II processor.

**Lab Overview:** This lab teaches you how to create an embedded system implemented in programmable logic. You will build a processor-based hardware system and run software on it. As the lab progresses, you will see how quick and easy it is to build entire systems using Quartus Qsys tools to configure and integrate pre-verified IP blocks.

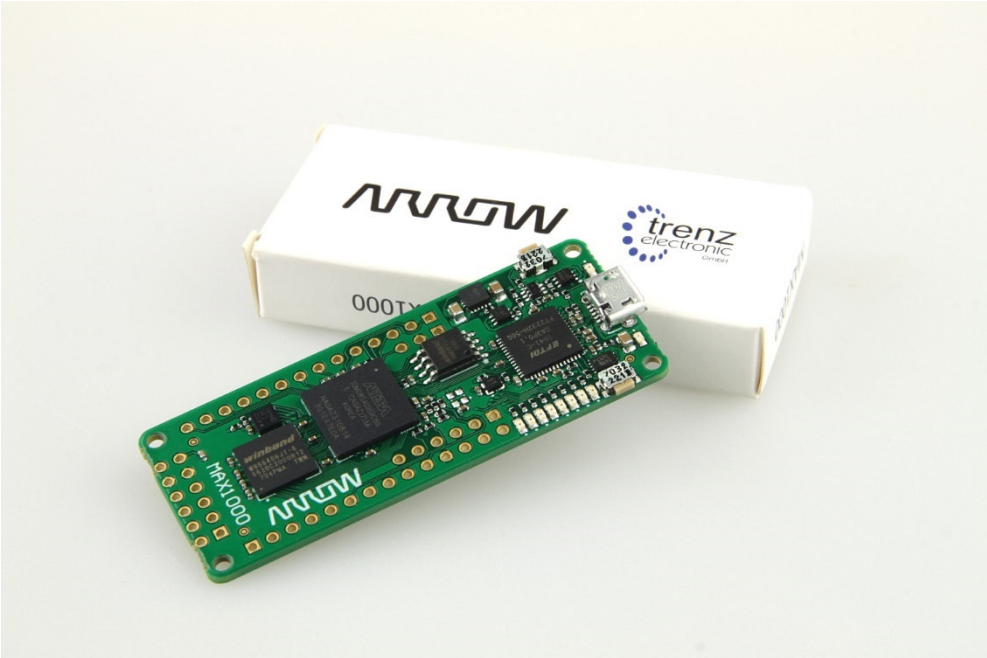
**Project Details:** The lab will guide you through creating an embedded system using Qsys. This system will be able to retrieve data from the on-board accelerometer of the MAX1000. Depending on the data received by the Nios II processor, the LEDs will react to the Y-axis.

**Lab Notes:** Many of the names that the lab asks you to choose for files, components, and other objects in this exercise must be spelled exactly as directed. This nomenclature is necessary because the pre-written software application includes variables that use the names of the hardware peripherals. Naming the components differently can cause the software application to fail. There are also other similar dependencies within the project that require you to enter the correct names.

## 2. Getting Started

The first objective is to ensure that you have all the necessary hardware items and software installed so that the lab can be completed successfully. Below is a list of items required to complete this lab:

- MAX1000 Board
- USB Cable
- Lab Files: https://wiki.trenz-electronic.de/display/PD/MAX1000
    - max1000_nios_template: Template files required to complete the project. Includes: nios_lab_top.vhd, nios_lab_top.qsf, nios_lab_top.sdc, RESET_GEN.vhd
    - max1000_nios_completed: Completed archived project with the Software files for Nios II Eclipse IDE.
- Quartus Prime 17.0 Lite was used for this lab. Previous versions should work (If no Quartus Prime is installed, refer to MAX1000 User Guide for instructions)
- Installed Arrow USB Drivers (If not, refer to MAX1000 User Guide for instructions)
- Personal computer or laptop running 64-bit Linux / Windows 7 or later with at least an Intel i3 core (or equivalent), 4GB RAM and 12 GB of free hard disk space
- A desire to learn!

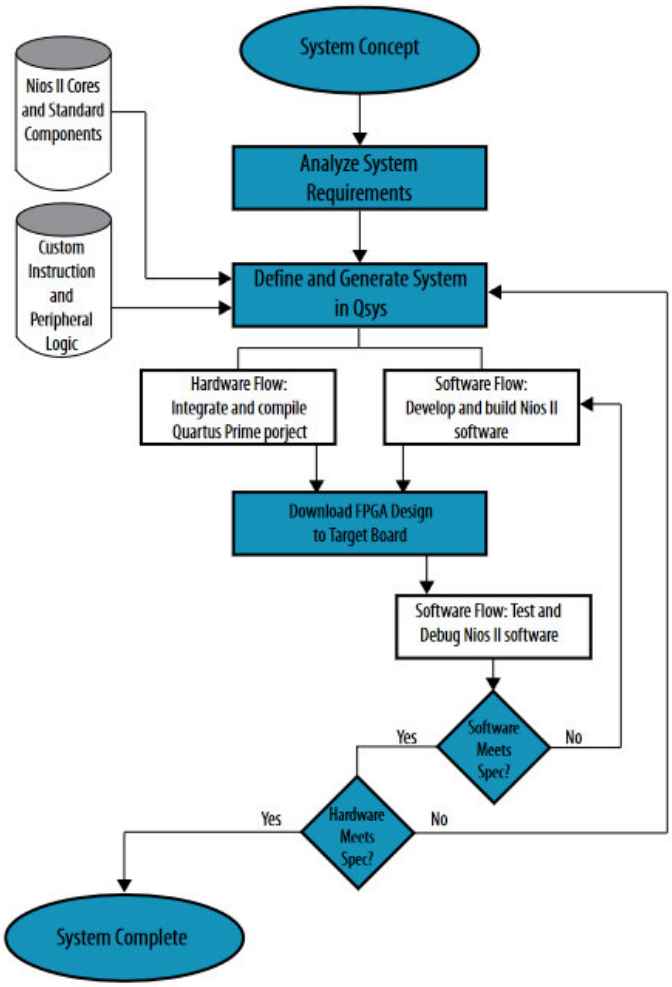


## 3. Examine the System Design

**Overview:** In this section, you will examine the design flow used in modern Intel FPGA designs.

**Examine the Design Tool Flow**

Developing software for an embedded system on a programmable chip requires an understanding of the design flow between the Qsys system integration tool and the Nios II Embedded Development Suite (EDS). Typically, designs begin with requirements and become inputs to system definitions. System definition is the first step in the design flow process. For this workshop, the design will be built and then the FPGA image will be downloaded into the board. The objective of the module is to review the development tools that will be used.




The above diagram shows the typical design flow for the system design. The system definition is done with Qsys. The Nios II IDE uses the system description to create a new project for the software application. The output of the FPGA design is a FPGA image that is used to configure the FPGA. The output of the software flow is an executable which runs on the Nios II processor.

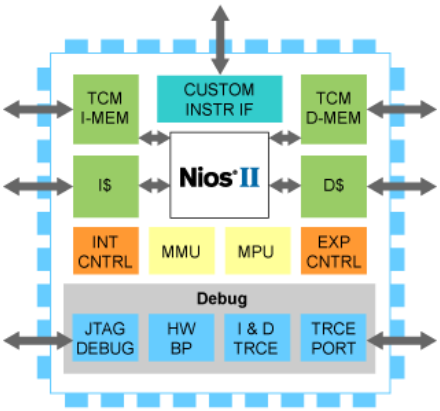
# 4. Nios II Soft Core

The Nios II processor delivers unprecedented flexibility for your cost-sensitive, real-time, safety-critical, ASIC-optimized, and applications processing needs. The Nios II processor supports all Intel® FPGA and SoC families.

Two different versions available:

- NIOS II / f : License Fee, optimized for performance
- NIOS II / e : Royalty Free, optimized for low resource consumption

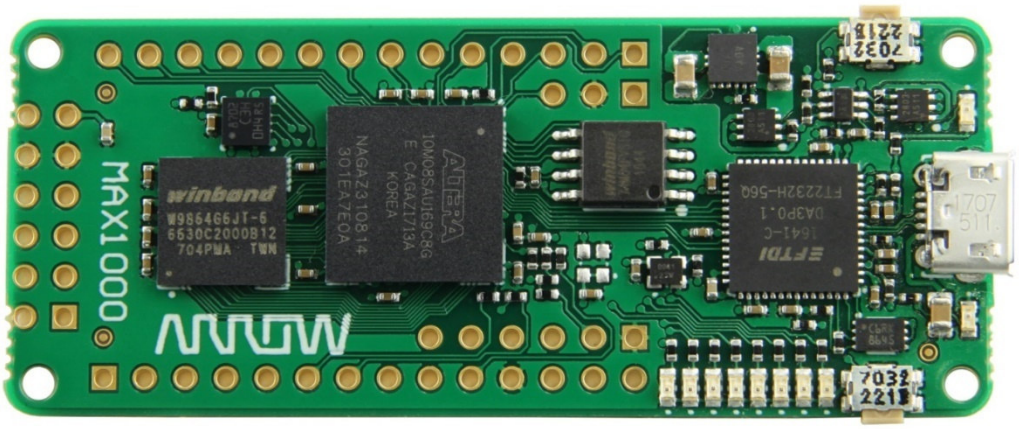
There is a variety configuration options to choose from depending on the application's needs.




Nios II soft core supports a variety of ecosystems, with more information found at:
https://www.altera.com/products/processors/ecosystem.html

## 5. Examine the MAX1000 Development Platform

There are plenty of components on the MAX1000 board that can be used including the LEDs, push buttons, accelerometer, external flash/SDRAM, and headers for connecting various other components through PMOD and Arduino MKR connections.




The completed system from completing the lab, will include many components including the Nios II soft processor, JTAG/UART, on-chip memory, PLL, and a SPI interface. The system that will be created in Qsys will use a library of re-usable IP blocks. Interconnect between components is automatically done by Qsys. The system interconnect manages the dynamics bus-width matching, interrupt priorities, arbitration and address mapping. The processor that is used, Nios II, is a full featured processor that can even run operating systems such as Linux and etc. The following sections of this document will guide you through the process of building a basic embedded system.

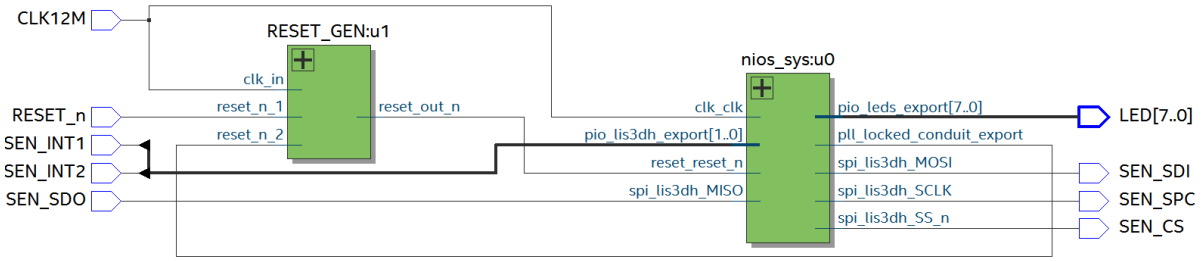
## 6. Implementing Nios II soft core in MAX1000

In this module, you will create a Quartus Prime project for your embedded system design and create the software project to run on the Nios II processor.

We will be using Qsys to add and interconnect different components. The following components will be included in our system:

- Clock source
- PLL
- Nios II Processor
- On-Chip Memory
- Parallel I/O (LED output)
- Parallel I/O (Accelerometer interrupt input)
- SPI (3-Wire Serial)
- System ID Peripheral

The complete system would look like this:



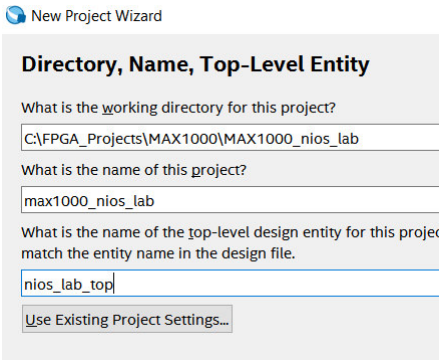

## 6.1 Create a New Quartus Prime Project

6.1.1 Create a new project using the New Project Wizard. Click **File → New Project Wizard**.

6.1.2 Configure the New Project Wizard directory, name and top-level entity information:

- Specify the location of the lab files on your PC.
  In this case, it was: C:\FPGA_Projects\MAX1000\MAX1000_nios_lab
- Specify the name of the project: **max1000_nios_lab**
- Specify the name of the top-level entity: **nios_lab_top**

*Note: It is a common naming convention to include the word "top" in the top-level design entity to make it clear and obvious which entity is at the top of the hierarchy.*




6.1.3 Click Next.

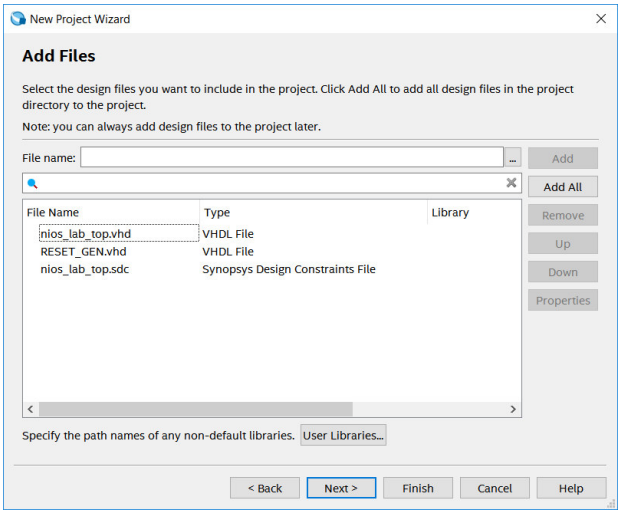
6.1.4 On the Project Type page, select "Empty Project" and click Next.

6.1.5 Add source files to the project

Click on the ... button and browse into the lab files folder where you will locate the three provided design files: nios_lab_top.vhdl, nios_lab_top.sdc and RESET_GEN.vhdl. Select all of them and add them to the project directory.

*Note: To see the sdc file, change file type filter to "All Files" (*.*).*

Do not forget to click the Add button to add the files to the project directory.

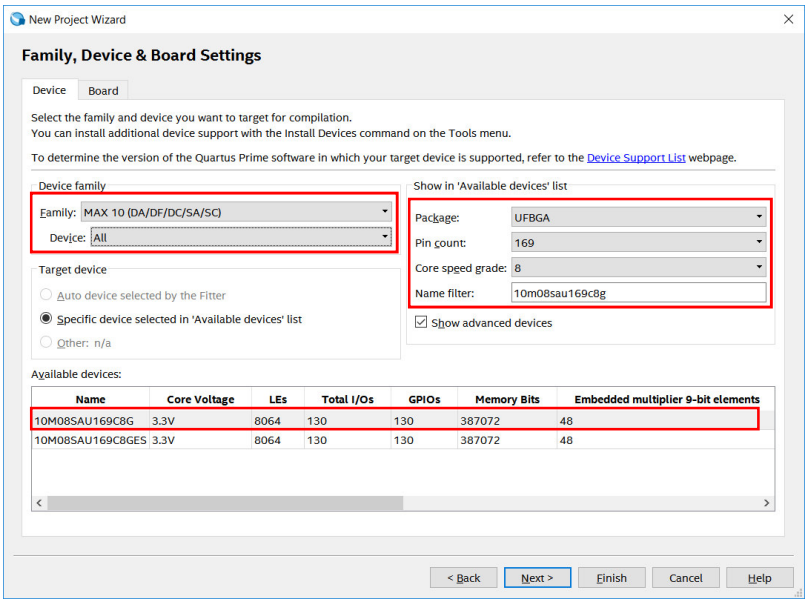



6.1.6 Click Next.

### 6.1.7 Specify Family and Device Settings

Rather than using the pull down menus to select the correct family, enter the part number in the Name Filter text box.

The part number is **10M08SAU169C8G**.




### 6.1.8 After making your selection, look at the kit and confirm that the part number marked on your device matches your selection. Click Finish.

## 6.2 Build the Hardware Design

**Overview:** In this module, you will use Qsys system integration tool to design your hardware system. You will add standard and custom components, make interface connections, assign clocks, set arbitrary levels of interrupts, and generate HDL for the system.

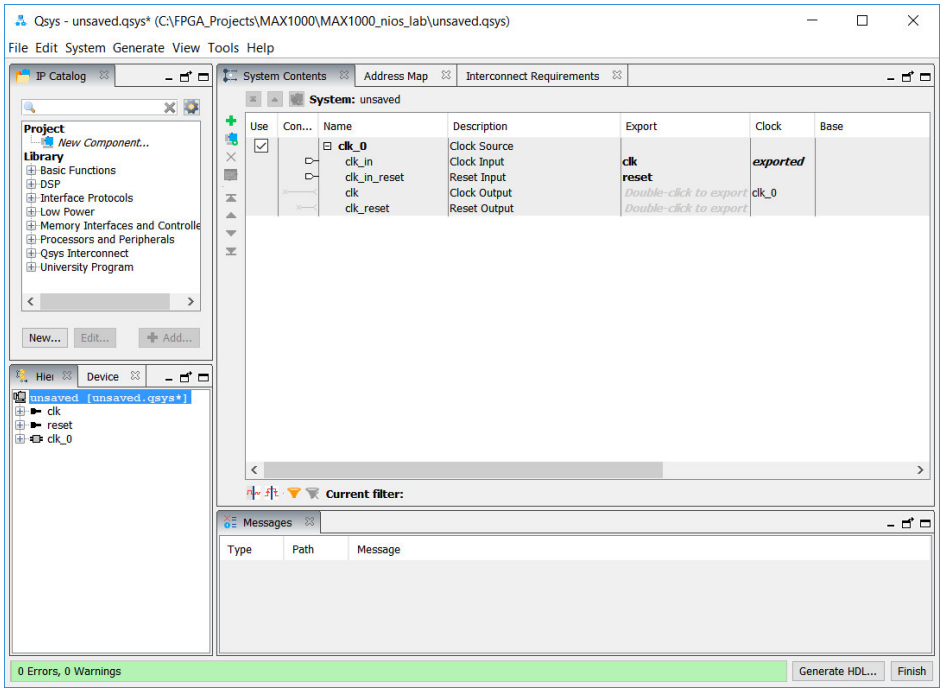
### 6.2.1 Launch Qsys

Qsys is a high level system integration tool that allows you to quickly build a system using Altera's IP blocks as well as custom components. The tool automatically creates interconnect logic between the components and allows for easy design use.

A Qsys is made up of several components and the automatically generated, high performance interconnect between them. Qsys allows you to connect components on an interface level, rather by signal by signal level. Qsys understands the different types of interfaces and will only allow connections between interfaces of same type (i.e. a data master connects to a data slave, clock source to clock sink, etc...).

6.2.1.1 Open Qsys: from the Quartus Prime window: **Tools → Qsys**.

6.2.1.2 In the new Qsys window, you should see a single Clock source component named clk_0 in the System Components Tab. This tab shows all the components currently in your system.



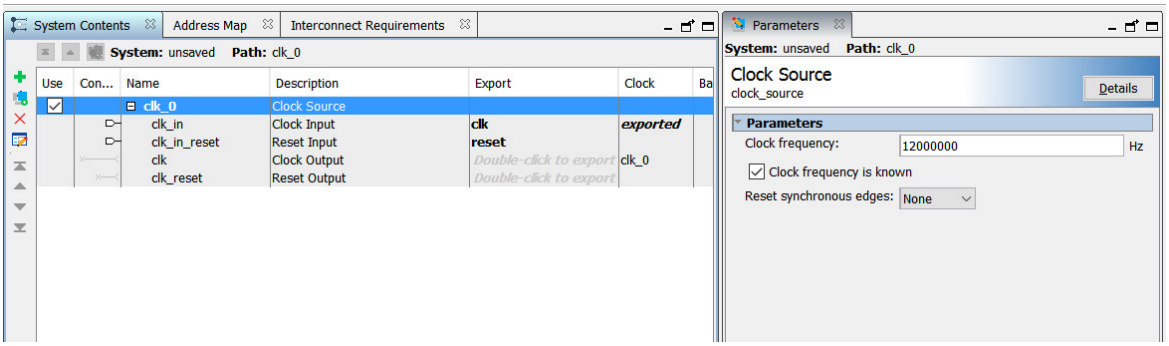

### 6.2.2 Configure the Clock

In this section, you will configure the clock input to your Qsys system. This clock will be fed to a PLL to provide addition frequencies.

6.2.2.1 Double-click on the Clock Source component named clk_0. This will open the Parameter editor window on the right side, which should look very familiar to the traditional Megawizard windows.

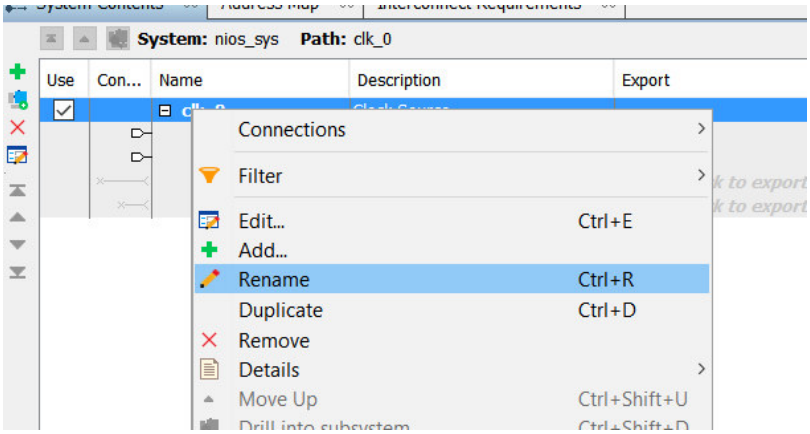
6.2.2.2 Change the clock frequency parameter to **12MHz** (12000000 Hz).

Ensure that the “Clock frequency is known” parameter is enabled.




Click the "X" on the Parameter tab to close the parameter window.

6.2.2.3 To rename the clock, right-click on the clock and select “Rename” or press CTRL+R. Rename the clock to “clk12mhz” and press Enter.

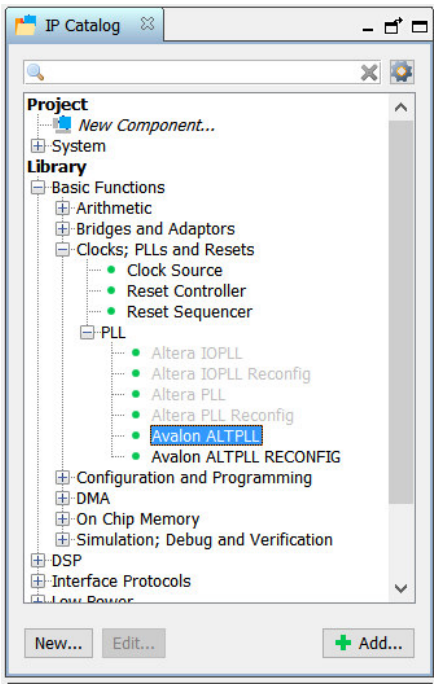



6.2.2.4 Save the Qsys system. Click **File → Save** As and name your qsys system **nios_sys.qsys**. This is the entity name by which you will be instantiating your Qsys system in the top level file. Click Save.

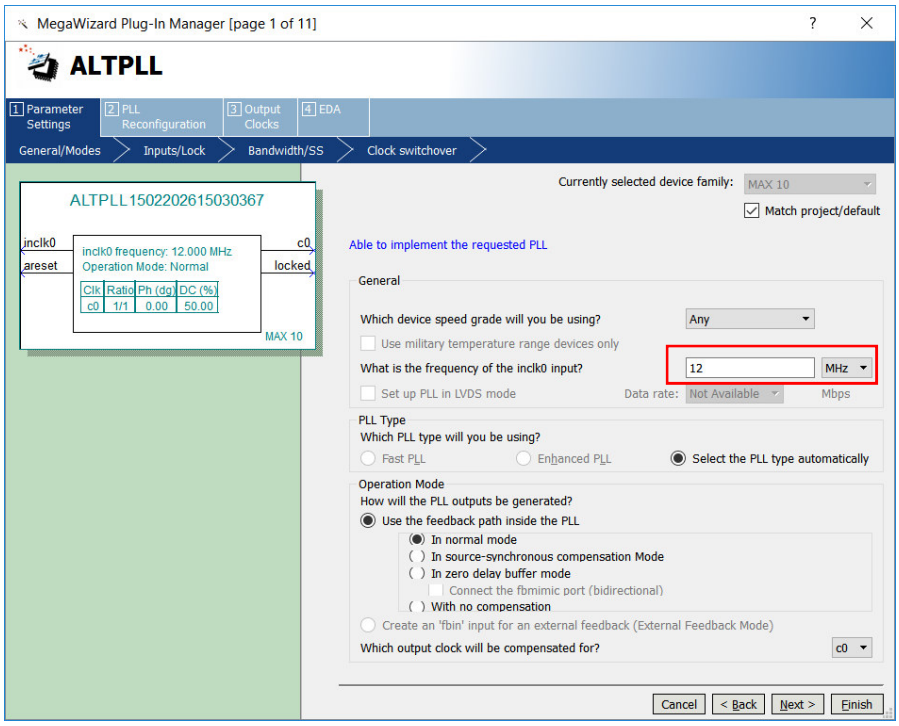
6.2.3 Add an Avalon ALTPLL for the processor and peripherals

The Avalon ALTPLL peripheral instantiates the PLL that will generate the clock for our system.

6.2.3.1 From the IP Catalog panel on the left side of the Qsys window, expand the menus for the **Basic Functions → Clocks; PLLs and Resets → PLL** and select the "Avalon ALTPLL".

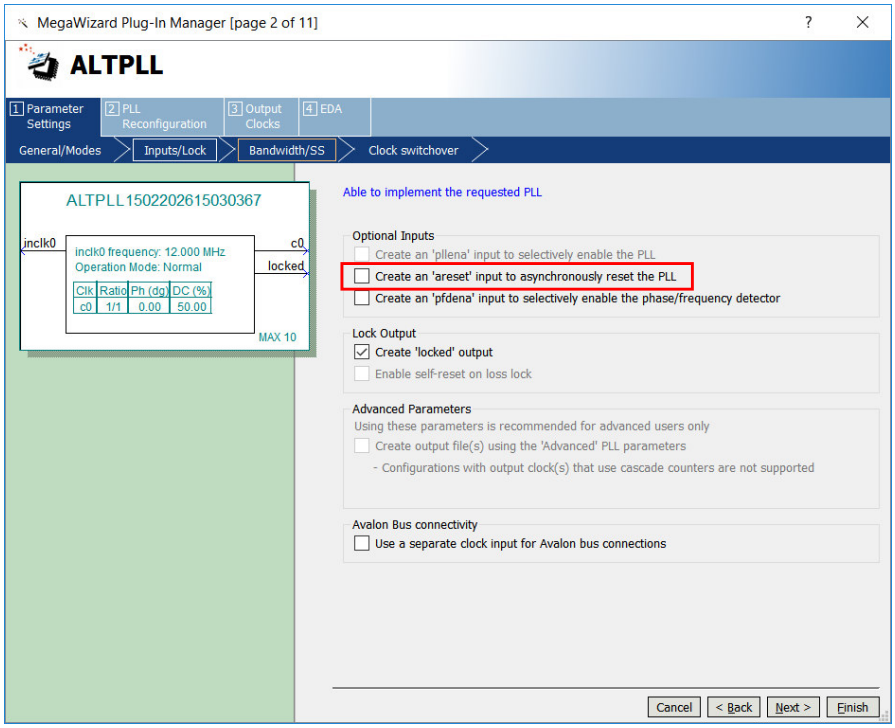



6.2.3.2 Under "General/Modes" tab (Page 1) of PLL MegaWizard change the frequency of the clock input to **12 MHz**. This source is provided by the oscillator on the MAX1000 board.



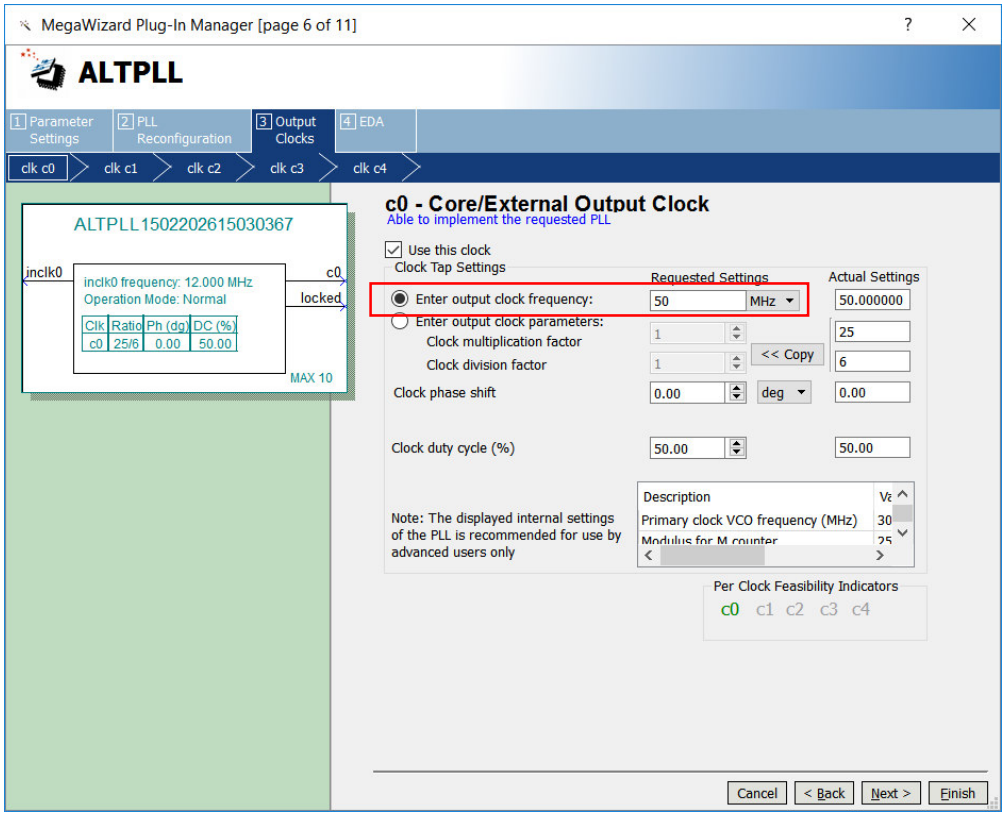

Click Next to move to the next tab of the MegaWizard.

6.2.3.3 “Inputs/Lock” tab (Page 2/11): Uncheck “Create an ‘areset’ input to asynchronously reset the PLL” option.
Accept all other defaults.




6.2.3.4 Pages 3-5: Accept all defaults and click next until you reach the Output Clocks tab.

6.2.3.5 On “c0 Core/External Output” (Page 6): Click “Enter output clock frequency”. Configure c0 as 50 MHz output. To do that, select “Enter output frequency” and enter **50 MHz**. This clock will be used as the processor system clock, clocking the Nios II processor various peripherals of the system. Click Next.




6.2.3.6 Click Finish. This will take you to the summary tab.

Click Finish again to close the Avalon ALTPLL MegaWizard

6.2.3.7 A component entitled “altpll_0” should appear under Module Name. Rename the Avalon ALT PLL component to “pll”. (You can right click to bring up a menu with a rename option.)

Some errors and warnings will appear in the bottom console indicating that various ports are not connected. Ignore these for now. We will address these connections in the upcoming steps.

### 6.2.4 Connect the incoming clock and reset to the PLL

Qsys needs to know what clock and reset sources to use as the input to the PLL component. The clock and reset sources can come from an external source or from another component within the Qsys system. In our case, we will be connecting them to an external clock and reset.

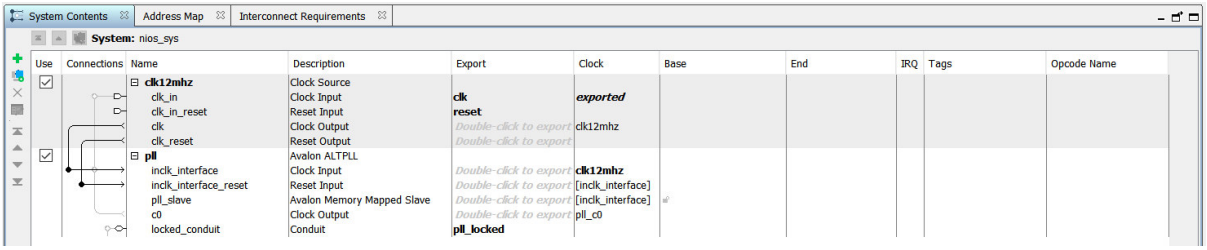
Click on the "System Contents" tab to return to the view of the components in our system. At this point, there are two components, a "Clock Source" component that was in the system by default when Qsys first launched and the "Avalon ALTPLL" component that we added in the first step. The Clock Source component is a Qsys component which brings in a clock and reset source from outside of the Qsys system. We will connect its nodes to the corresponding nodes of the Avalon PLL component.

6.2.4.1 In the "Connections" column, hover over the connections and you will then be able to fill in dots to make the connections.

6.2.4.2 Connect the “clk” Clock Output port of the Clock Source <clk12mhz> to the “inclk_interface” of the <pll>. Similarly connect the “clk_reset” reset output port of the Clock Source <clk12mhz> to the “inclk_interface_reset” of the <pll> component.

6.2.4.3 Click on the "Double-click to export" field next to Conduit and name it “pll_locked”. We will be using this as one of the inputs of the external reset component.

Your resulting connections should look as follows:

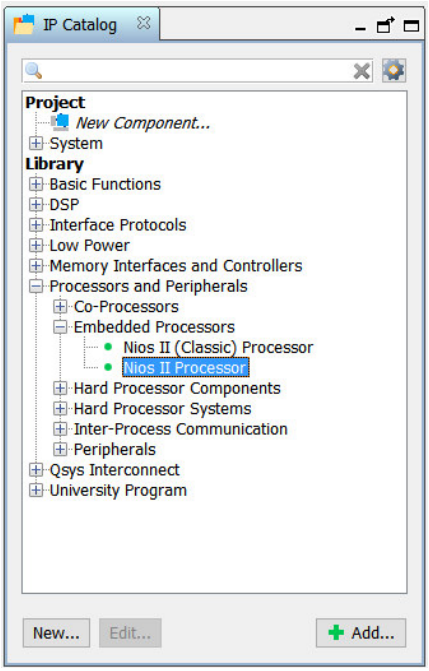



6.2.4.4 Click on **File → Save** and save your work periodically as you continue through the design.

### 6.2.5 Add a Nios II Processor

A CPU is needed to run the software applications.

6.2.5.1 From the IP Catalog panel on the left side of the Qsys window, expand the menus for the **Processors and Peripherals → Embedded Processors** and select the Nios II Processor.




*(Note: MAX10 devices do not support the Nios II (Classic) Processor. However, all code developed on the classic version is fully forward compatible.)*

6.2.5.2 Double-click on the name or click “Add..” to add the component to the system. The Nios II parameter editor window will open.

6.2.5.3 In the Main tab, ensure that the “Nios II /e" option is selected.

6.2.5.4 The settings in the Vectors tab will be set in a later step so skip that for now.

Note that until these settings are applied, the following errors in the Qsys window are expected:

- Error: nios2_gen2_0: Reset slave is not specified. Please select the reset slave.
- Error: nios2_gen2_0: Exception slave is not specified. Please select the exception slave.

6.2.5.5 The settings in the other tabs are left as their defaults but feel free to explore the parameter editor and see what settings can be applied to the Nios II. Click Finish.

Note: There will be errors related to clocks as well. This will be resolved in a few steps.

6.2.5.6 Rename the Nios II to “nios”.

### 6.2.6 Configure clock source for the Nios II processor

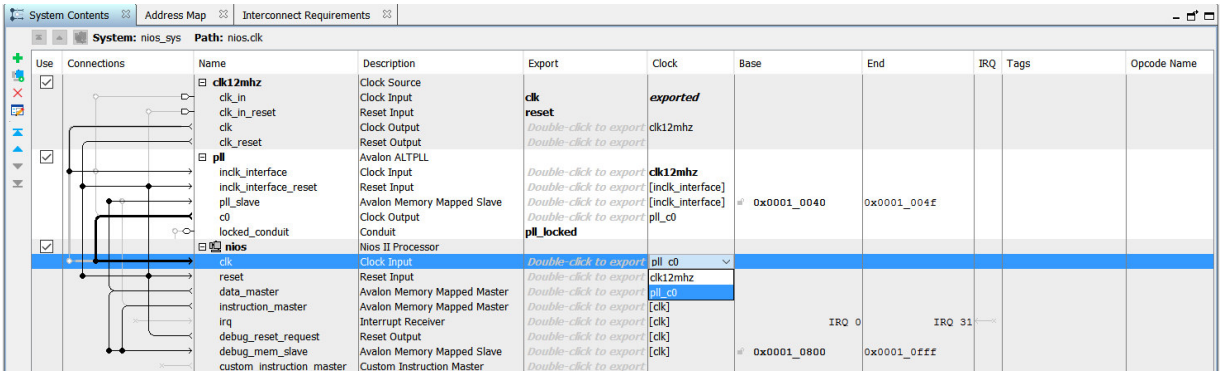
At this point, there are 3 components in the system.

6.2.6.1 From the drop-down list in the Clock column, select and choose “pll_c0”. Note that we made this connection with the connection dots in an earlier step.

6.2.6.2 Connect “reset” of the <nios> component to “clk_reset” of the <clk12mhz> component.

6.2.6.3 Also connect “data_master” of the <nios> component to “pll_slave” of the <pll> component.

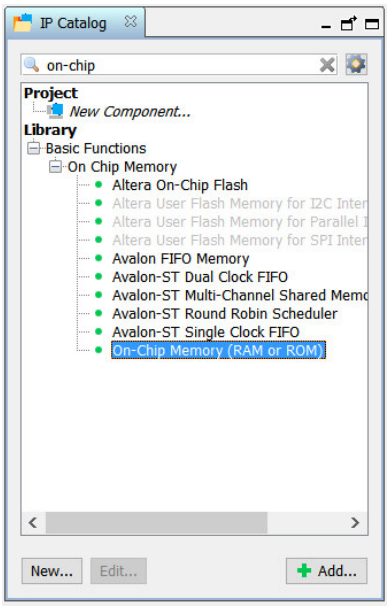
Your system should look as follows:




### 6.2.7 Add On-Chip Memory

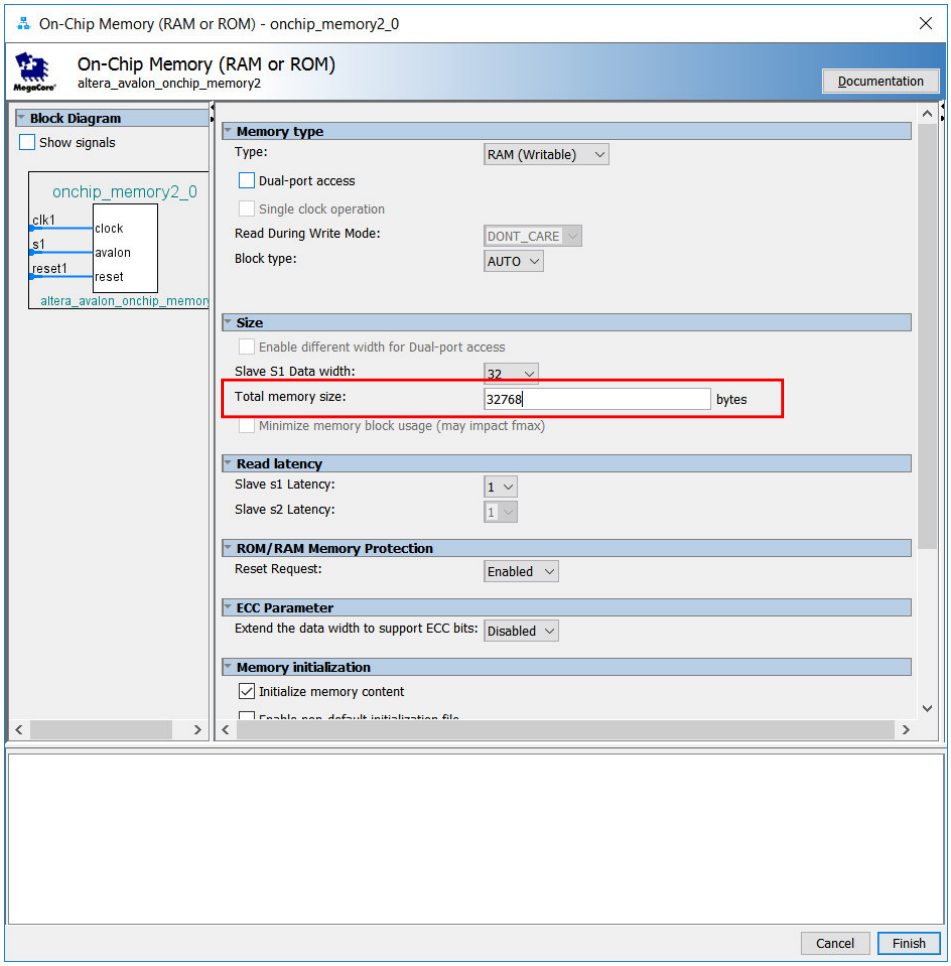
Intel FPGAs provide internal on-chip memory blocks that can be used to build up an internal RAM (or ROM) block of memory. In this lab, this provides Nios II with access to very low-latency, high speed memory for executable code and variable storage.

6.2.7.1 In the IP Catalog panel, type “on-chip” in the search bar. You should see the On-Chip Memory (RAM or ROM) appear under **Basic Functions → On Chip Memory**.



6.2.7.2 Double click the component or select it and click "Add..." to add it to the system. The On-Chip Memory parameter editor will open.

6.2.7.3 Change the total memory size parameter to **32768** bytes or type 32k and the field will update.

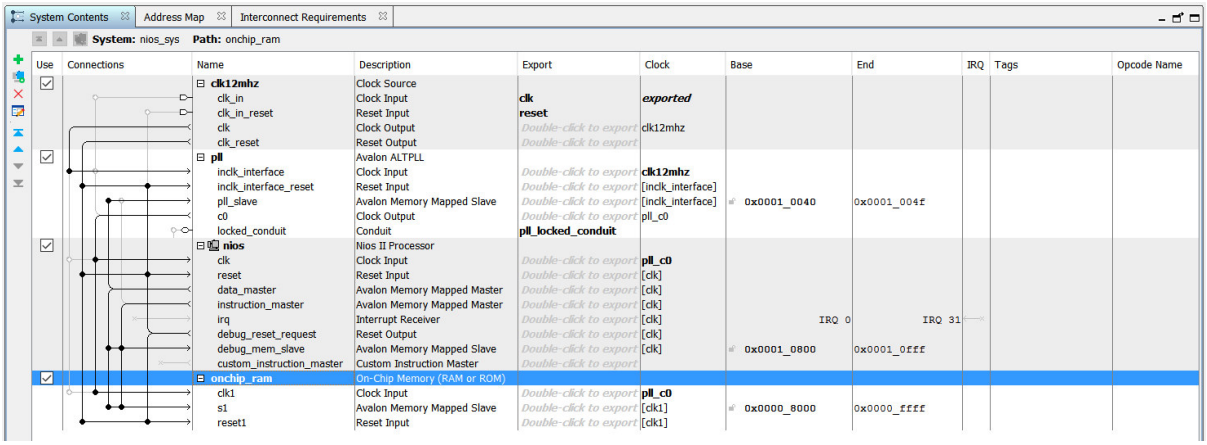



6.2.7.4 Accept the defaults for the remaining fields and click Finish to add the component to the system. Don’t worry about the errors, they will be resolved later in the lab.

6.2.7.5 Rename the component to “onchip_ram”.

6.2.7.6 Using the Clock column, change the clock input of the “onchip_ram” to pll_c0 clock source.

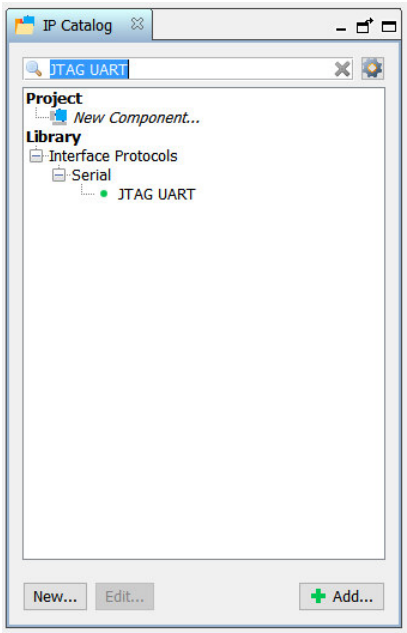
6.2.7.7 Using the Connections column, connect the “s1” Avalon Memory Mapped Slave interface of the <onchip_ram> to the <nios> instruction_master and data_master.




## 6.2.8 Add the JTAG UART Peripheral

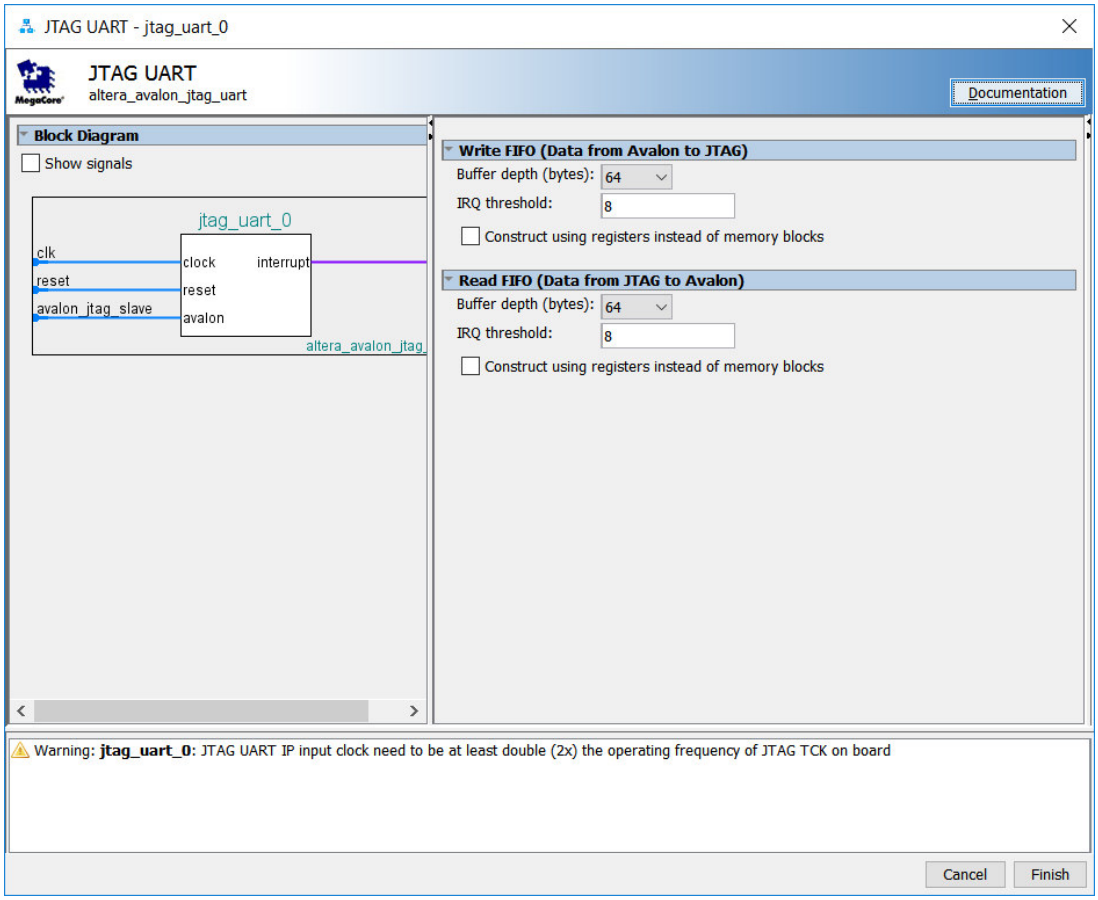
Many software developers like to have access to a debug serial port from the target to leverage <printf> debugging, input control, log status information, etc. The JTAG UART connects to Nios II processor to the debugger console in the Nios II IDE for easy debug and development using a console interface.

6.2.8.1 In the IP Catalog search bar, type JTAG UART. You should see the JTAG UART peripheral appear under **Interface Protocols → Serial.**




6.2.8.2 Double-click the component or select it and click "Add..." to add it to the system. The JTAG UART parameter editor will open.

6.2.8.3 Verify that the parameters for the Write and Read FIFO are the same as below.




6.2.8.4 Click Finish to add the component to the system. Don't worry about the errors; they will be addressed later.

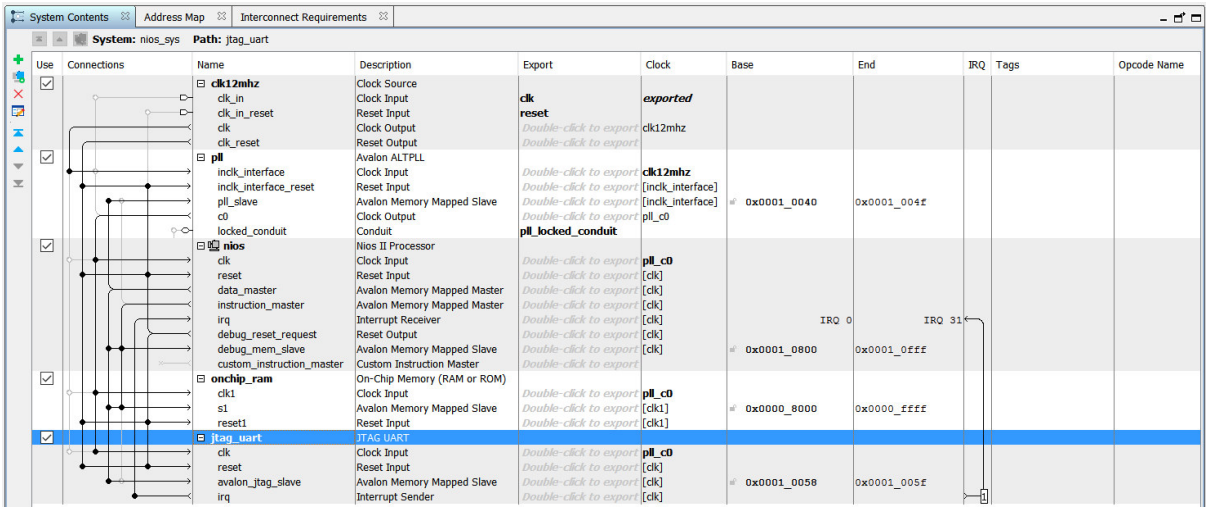
6.2.8.5 Rename the component to "jtag_uart".

6.2.8.6 Connect the Avalon_jtag_slave_port of the <jtag_uart> to the data_master of the <nios>.

6.2.8.7 In the clock column, select pll_c0 as the Clock Input.

6.2.8.8 Connect the irq port of the <jtag_uart> to the irq of the <nios> processor.

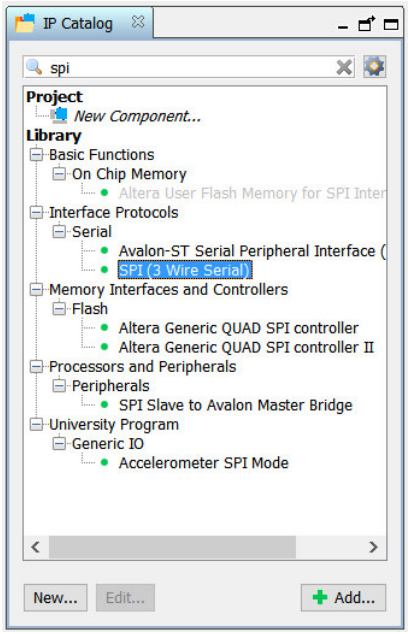
At this point, there are 5 components in the system and should look as follows:




### 6.2.9 Add SPI Interface

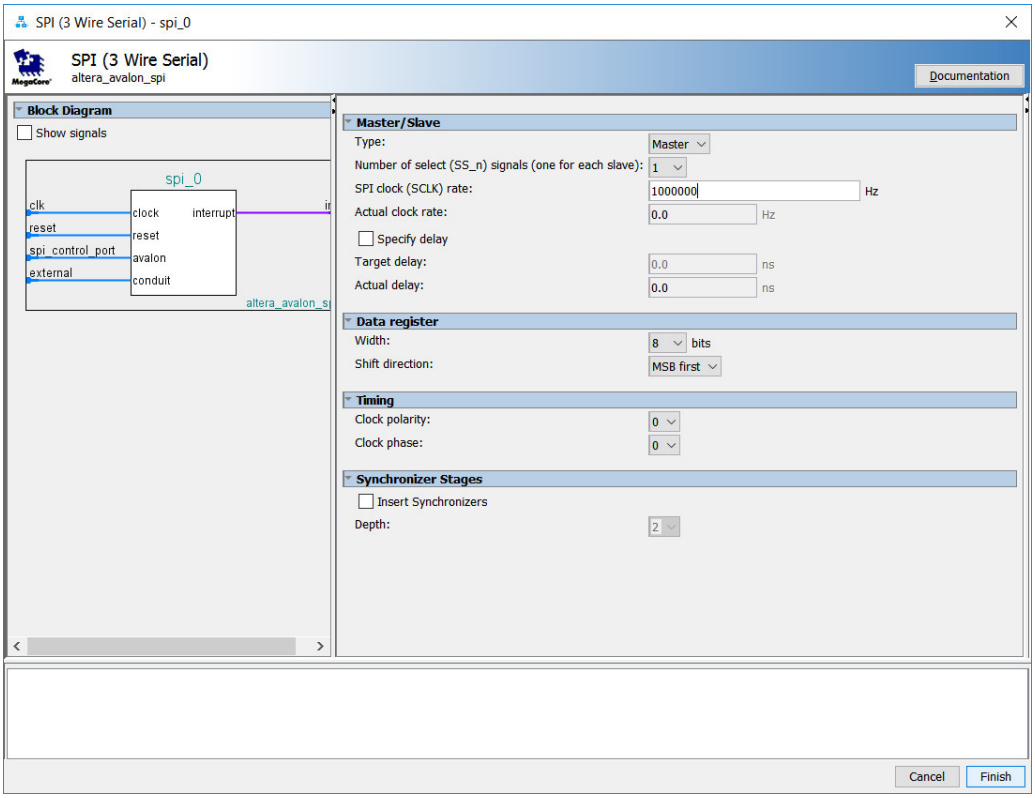
To make a connection with the on-board accelerometer of the MAX1000, an SPI interface connection is needed.

6.2.9.1 In the IP Catalog panel, type “spi” in the search bar. You should see the SPI (3 Wire Serial) appear under **Interface Protocols → Serial**.




6.2.9.2 Double-click the component or select it and click "Add..." to add it to the system. The SPI (3 Wire Serial) parameter editor will open.

6.2.9.3 Change the SPI Clock rate (SCLK) to **1 MHz** or type 1m and the field will update.




6.2.9.4 Accept the defaults for the remaining fields and click Finish to add the component to the system. Don’t worry about the errors, they will be resolved later in the lab.

6.2.9.5 Rename the component to “spi_lis3dh”.

6.2.9.6 In the clock column, select pll_c0 as the Clock Input.

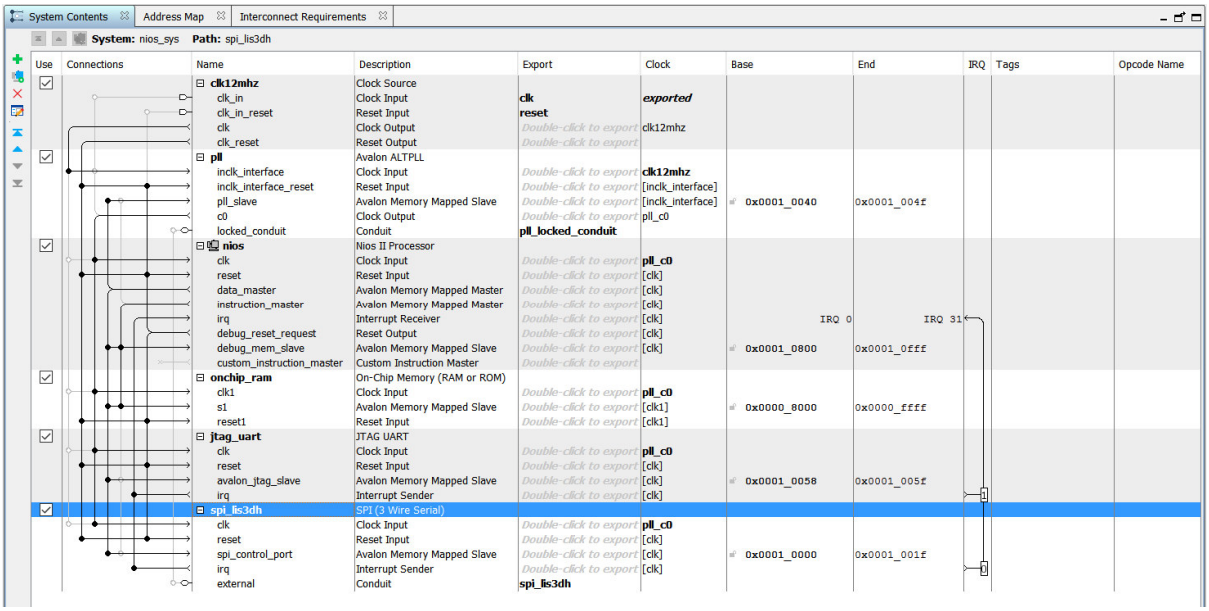
6.2.9.7 Connect the spi_control_port of the <spi_lis3dh> to the data_master of the <nios>.

6.2.9.8 In the clock column, select pll_c0 as the Clock Input.

6.2.9.9 Connect the irq port of the <spi_lis3dh> to the irq of the <nios> processor.

6.2.9.10 Finally, click in the "click to export" field next to the external_connection Conduit and name it “spi_lis3dh”.

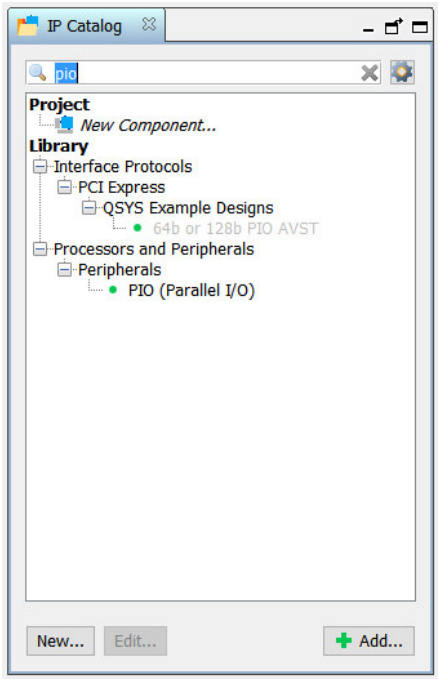
6.2.9.11 At this point, there are 6 components in the system and should look as follows:




## 6.2.10 Add PIO Peripheral for Accelerometer Interrupts

The LIS3DH accelerometer has two interrupt pins. You can use an input PIO peripheral so the processor can detect when those interrupts are triggered

6.2.10.1 In the IP Catalog panel, type “pio” in the search bar. You should see the PIO (Parallel I/O) appear under **Processors and Peripherals → Peripherals**.

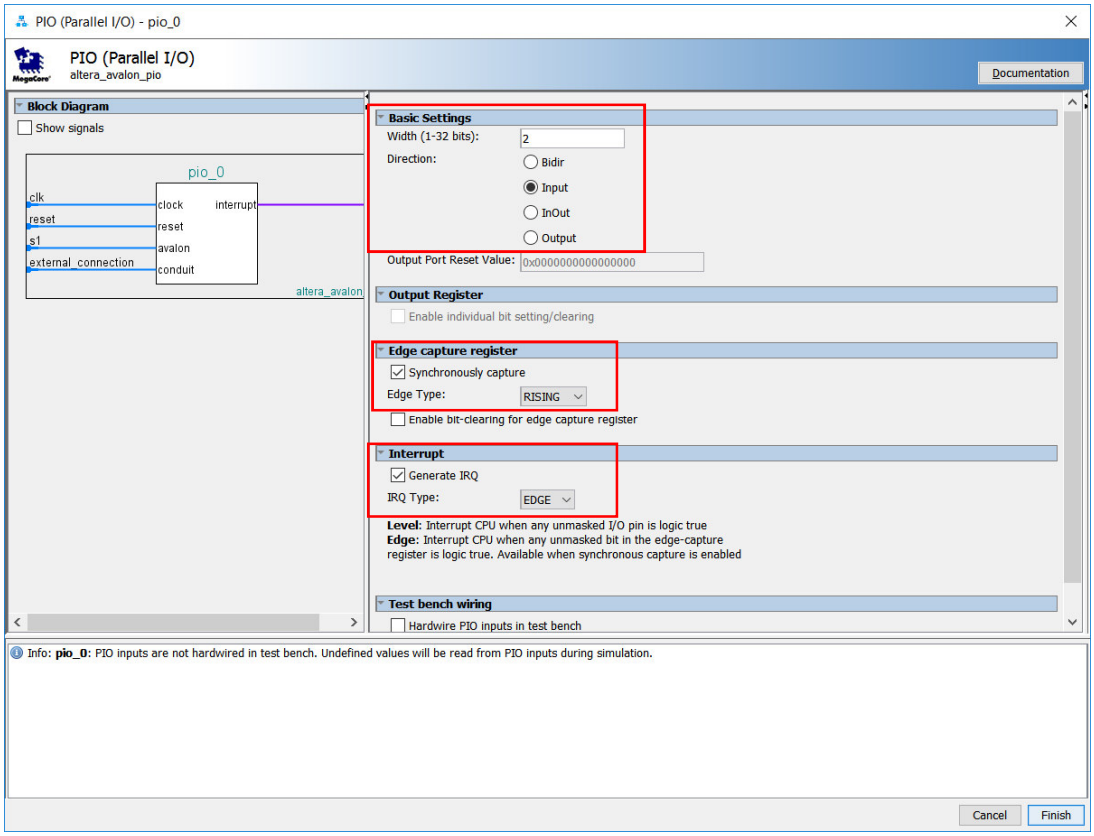



6.2.10.2 Double-click the component or select it and click "Add..." to add it to the system. The PIO (Parallel I/O) parameter editor will open.

6.2.10.3 Set the width to **2** bits. Ensure that the direction is **Input**.

6.2.10.4 Make sure Synchronous Capture is enabled along with **RISING** as the Edge Type, under Edge capture register.

6.2.10.5 Make sure Generate IRQ in enabled along with **EDGE** as the IRQ Type, under Interrupt.




6.2.10.6 Accept the defaults for the remaining fields and click Finish to add the component to the system. Don't worry about the errors, they will be resolved later in the lab.

6.2.10.7 Rename the component to "pio_lis3dh".

6.2.10.8 In the clock column, select pll_c0 as the Clock Input.

6.2.10.9 Connect the s1 of the <pio_lis3dh> to the data_master of the <nios>.

6.2.10.10 Connect the irq port of the <pio_lis3dh> to the irq of the <nios> processor.

6.2.10.11 Finally, click in the "click to export" field next to the external_connection Conduit and name it "pio_lis3dh".

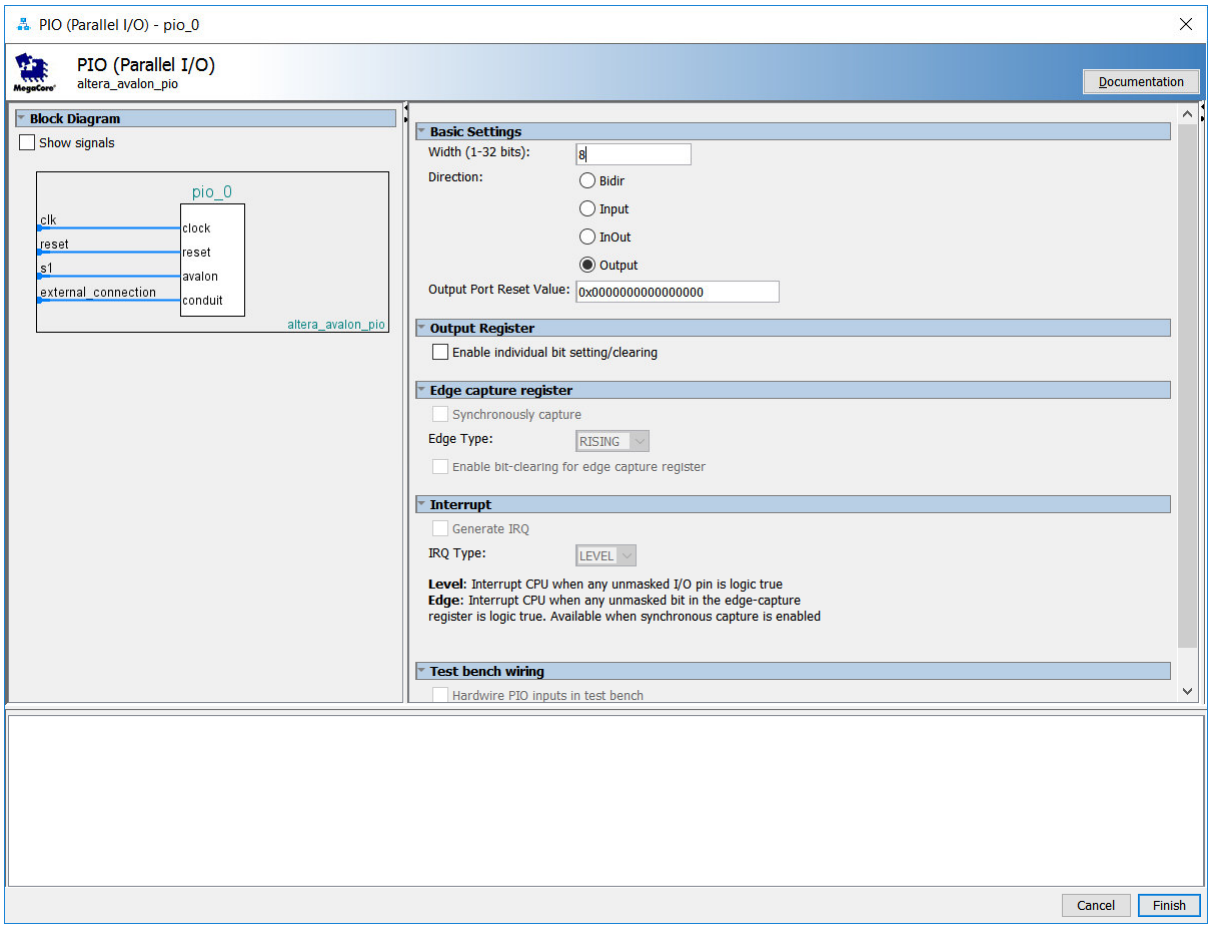
### 6.2.11 Add PIO peripheral for LEDs

The MAX1000 has 8 LEDs on it. You can drive those LEDs with an output PIO peripheral.

6.2.11.1 In the IP Catalog panel, type “pio” in the search bar. You should see the PIO (Parallel I/O) appear under **Processors and Peripherals → Peripherals**.

6.2.11.2 Double-click the component or select it and click "Add..." to add it to the system. The PIO (Parallel I/O) parameter editor will open.

6.2.11.3 Set the width to **8** bits. Ensure that the direction is **Output**.




6.2.11.4 Accept the defaults for the remaining fields and click Finish to add the component to the system. Don’t worry about the errors, they will be resolved later in the lab.

6.2.11.5 Rename the component to “pio_leds”.

6.2.11.6 In the clock column, select pll_c0 as the Clock Input.

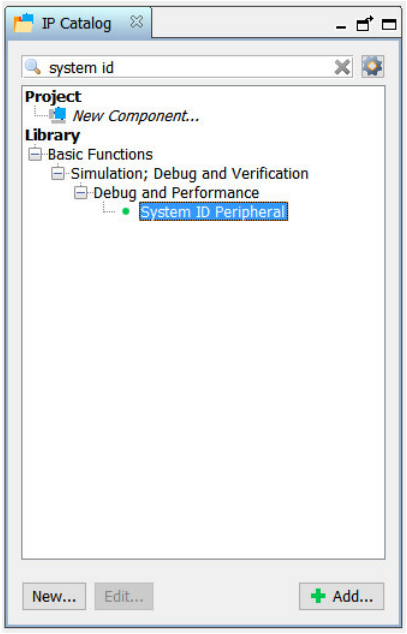
6.2.11.7 Connect the s1 of the <pio_leds> to the data_master of the <nios>.

6.2.11.8 Finally, click in the "click to export" field next to the external_connection Conduit and name it “pio_leds”.

### 6.2.12 Add a System Peripheral ID

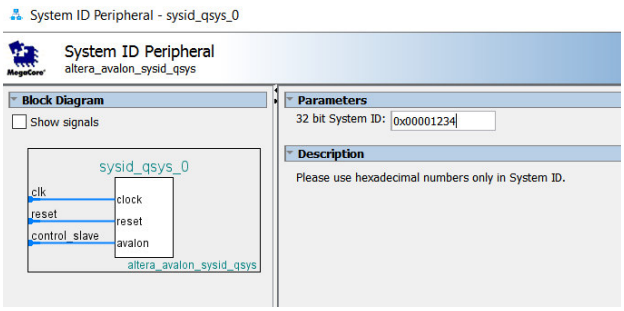
This is a VERY IMPORTANT peripheral to have in your system. It allows the Nios II development tools to validate that the software application is being built for the correct hardware system.

6.2.12.1 In the IP Catalog panel, type “system id” in the search bar. You should see the System Peripheral ID appear under **Basic Functions → Simulation; Debug and Verification → Debug and Performance**.




6.2.12.2 Double-click the component or select it and click "Add..." to add it to the system. The System Peripheral ID parameter editor will open.

6.2.12.3 Edit the 32 bit System ID to any value you like, or use 0x00001234.



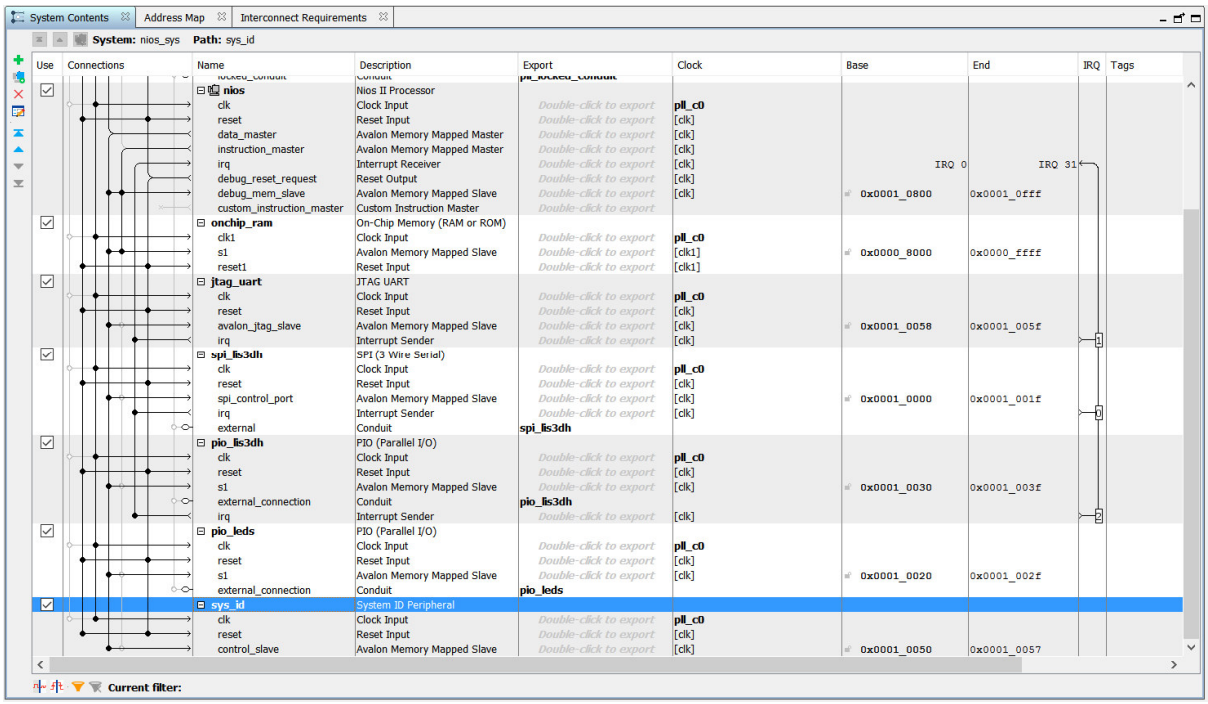

6.2.12.4 Select Finish and ignore the errors.

6.2.12.5 Rename the component to “sys_id”.

6.2.12.6 In the clock column, select pll_c0 as the Clock Input.

6.2.12.7 Connect the control_slave of the <sys_id> to the data_master of the <nios>.

6.2.12.8 At this point, there are 9 components in the system and should look as follows:

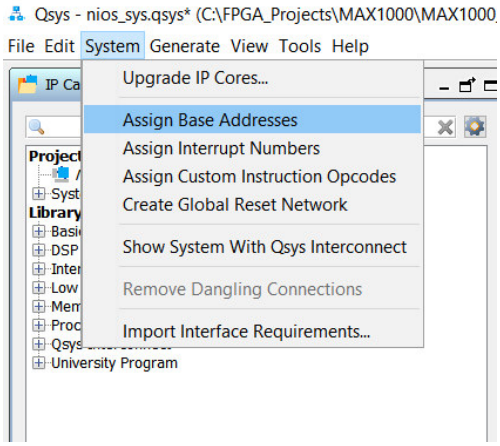


### 6.2.13 Resolve the Errors

At this point, Qsys will report a number of errors referencing unconnected clocks, unconnected resets, and unconnected Avalon interface because some of the components in your Qsys system are not fully connected. Once all the interfaces are connected, these errors will disappear.

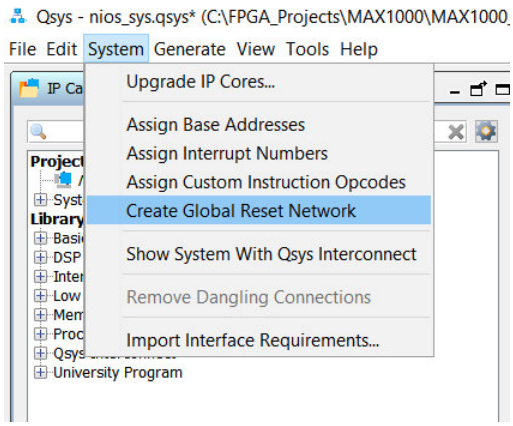
#### 6.2.13.1 Assign Base Addresses

When the peripherals were added to the system, they we all given the default base address of 0x0000000, so the components now have overlapping addresses. Qsys will report this as an error. You can manually enter the base addresses in the Base column, or you can let Qsys automatically assign them. Automatically assign them by selecting: **System → Assign Base Addresses**.




#### 6.2.13.2 Create Global Reset Network

In some cases, the reset ports of the components may not have been connected. You can manually connect these ports, or you can let Qsys automatically connect them. Automatically connect them by selecting: **System → Create Global Reset Network**.

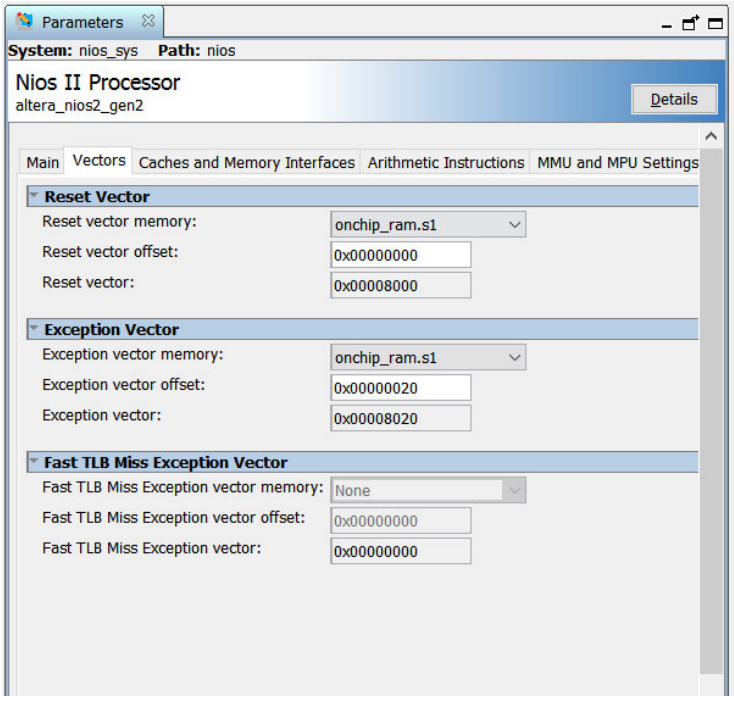



#### 6.2.13.3 Define the Nios II Reset and Exception Vectors

Like any processor, the Nios II requires memory locations to jump to in the event of a processor reset or exception within the execution of its code. The reset vector is the memory location to which the processor jumps on processor reset and the exception vector is the memory location to which the processor jumps on an exception. These are typically in non-volatile memory and can be at the same memory location.

6.2.13.3.1 To set these vectors, double-click on the Nios II component <nios>. The Nios II parameter editor will reopen.

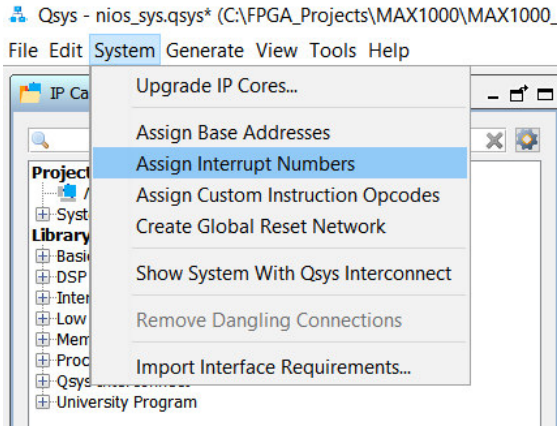
6.2.13.3.2 Click on the Vectors tab and set both the reset vector memory and exception vector memory to be **onchip_ram.s1** from the pull-downs. The offset and vector values may be different that the image below but will be corrected in a following step.



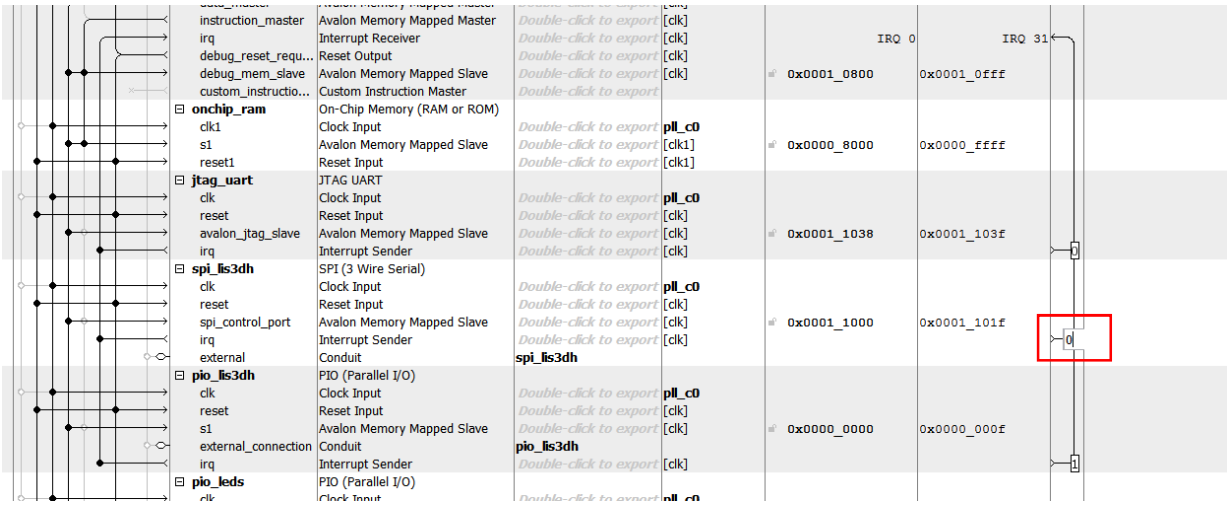

6.2.13.4 Review message window for remains errors.
At this point there should be no remaining errors in the message window. If there are, please refer again to the previous steps to resolve them.

### 6.2.14 Set Interrupt Priorities

The Nios II processor can process up to 32 independent interrupts (IRQs) from various parts of a system that can be assigned unique priorities. This system only has 3 interrupts and the priorities will need to be set manually depending on the user's needs although it can be done automatically by selecting **System → Assign Interrupt Numbers** from the Qsys menu as below.



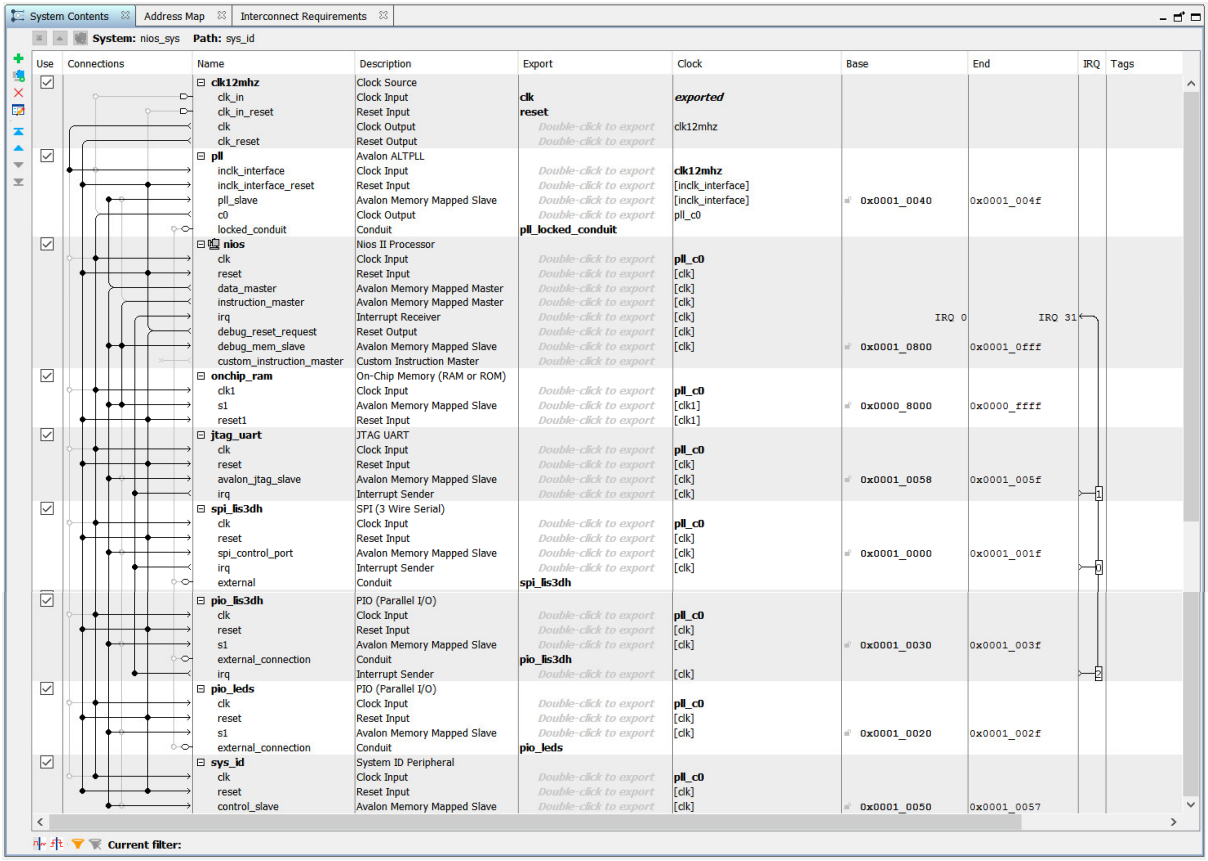

You can also manually set an IRQ priority in Qsys by double clicking the number in the IRQ column of the System Contents tab and entering the priority (priority 0 is the highest priority. For example, double click the number in the IRQ column to the right to the "irq" signal in the <spi_lis3dh> component and type 0. This will give the <spi_lis3dh> component's interrupt the highest priority.



### 6.2.15 Check the full system

Below is a screenshot of the full Qsys system with all connections visible.

6.2.15.1 Confirm that your Qsys matches the screenshot below.

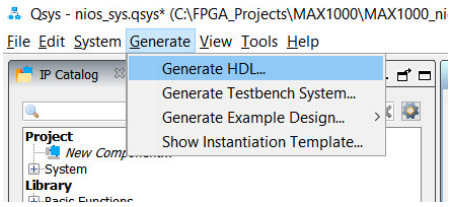


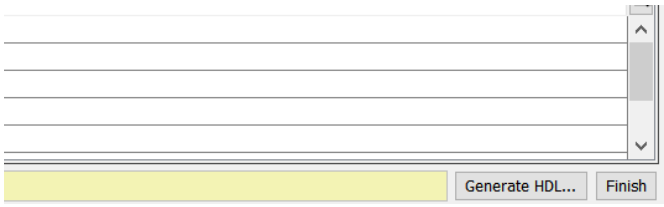

6.2.15.2 Make sure there are no errors messages in the Messages tab.

### 6.2.16 Generate the Qsys System

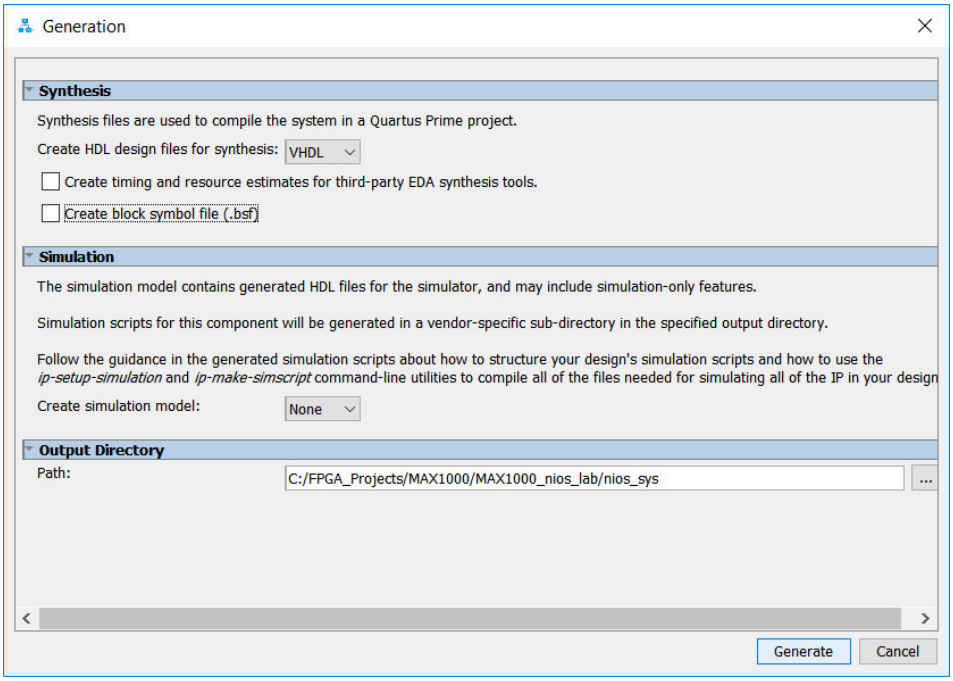
One of the great parts about Qsys is that it generates HDL (hardware description language) code from the created system so that the internals can be investigated for a better understanding. The next step is to generate the HDL from the system.

6.2.16.1 Select **Generate → Generate HDL…** from the Qsys menu or alternately click the Generate HDL… button on the bottom right of the Qsys window.



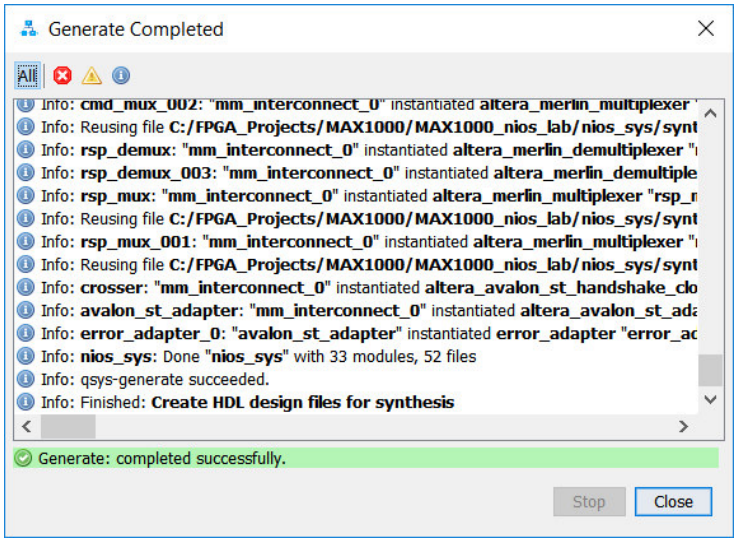




6.2.16.2 The Generate window will appear. Select "VHDL" as the synthesis language and "None" from the simulation model dropdown (Verilog can be used but the top-level file in this lab is in VHDL). Unselect “Create block symbol file(.bsf)” since this will not needed for this lab.The generated HDL files will appear in the directory pointed to by the file path specified under the Output Directory section. Leave this as the default.



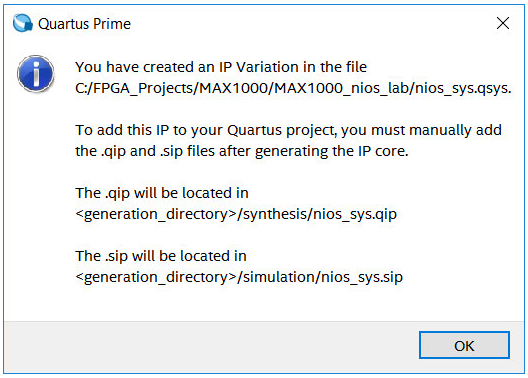
6.2.16.3 Click Generate.

Qsys will generate the necessary HDL for synthesis. When the generate process completes, click Close.




### 6.2.17 Add the Qsys System to the Quartus Project

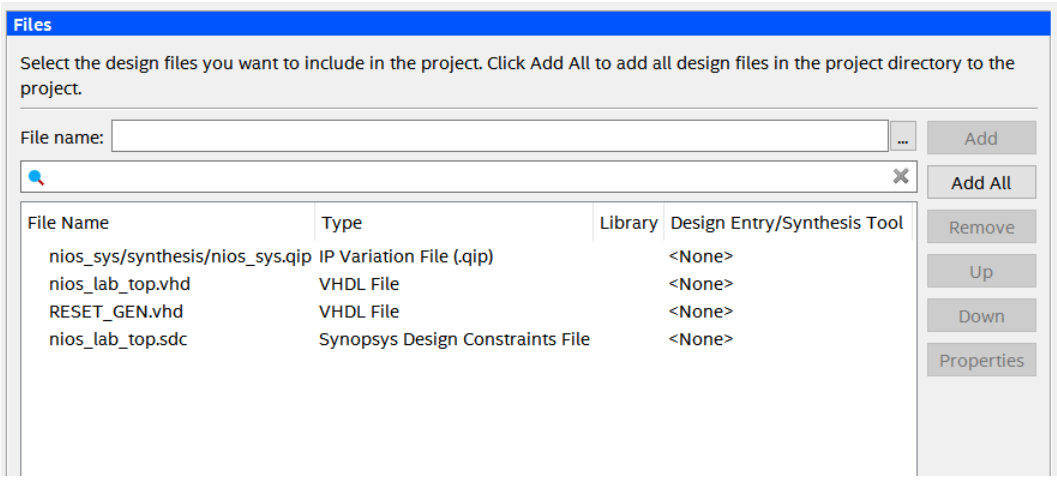
The system created in Qsys now needs to be added to your Quartus project so that it can be instantiated in the top-level design file. You can think of the Qsys system as a module or component as you would in any other FPGA design. Qsys generates IP "pointer" files for both synthesis (.qip) and simulation (.sip) that will point Quartus to all the necessary design files needed to synthesize or simulate the Qsys system. Press OK to close as the .qip file will be added to the project in the following steps.




6.2.17.1 Open the project files manager: **Project → Add/Remove Files** in the Project from the Quartus Prime menu.

6.2.17.2 Browse through the synthesis directories:
(it should be **<project_directory>/nios_sys/synthesis/**) and select nios_sys.qip.

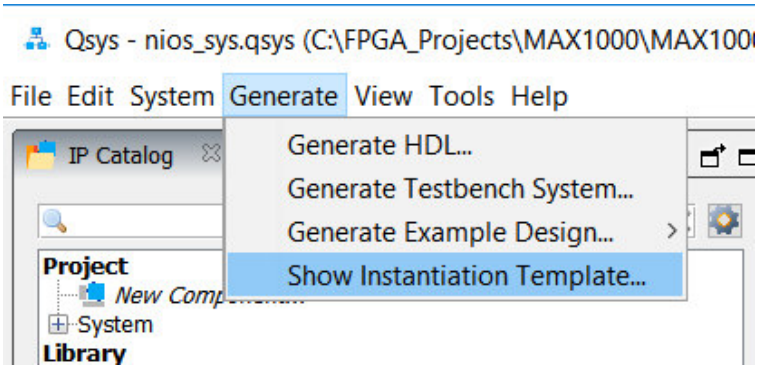
6.2.17.3 Click "Add" to add the .qip file to the project. Click "Apply" and "OK".



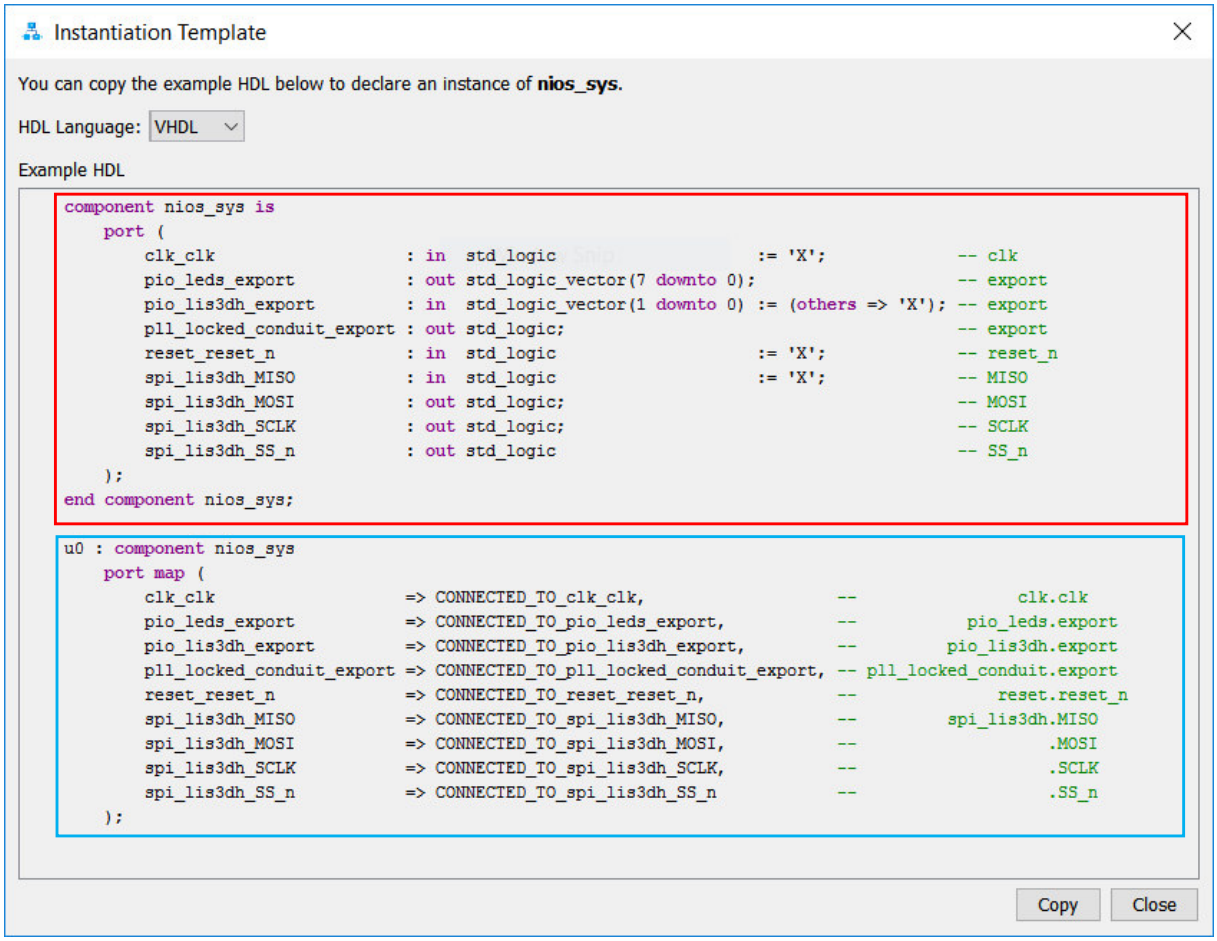

### 6.2.18 Instantiate the Embedded System Component in the top-level entity

Having done the above steps, we will need to instantiate our Qsys component in our top-level entity.

6.2.18.1 In the Qsys window, select **Generate → Show Instantiation Template…**



6.2.18.2 Select VHDL as the HDL Language. The following template will be shown which can be easily copied into your project, saving you valuable time.




6.2.18.3 There are two parts that would need be copied to the top-level entity “nios_lab_top”.

- nios_sys Component Declaration (highlighted in red)
  Copy this section and paste in the architecture section of “nios_lab_top” before the word begin. There should be a commented area indicating where exactly.
- nios_sys Component Instantiation (highlighted in blue)
  Copy this section and paste in the architecture section of “nios_lab_top” after the word begin. There should be a commented area indicating where exactly.

Top-level entity Qsys component declaration areas:

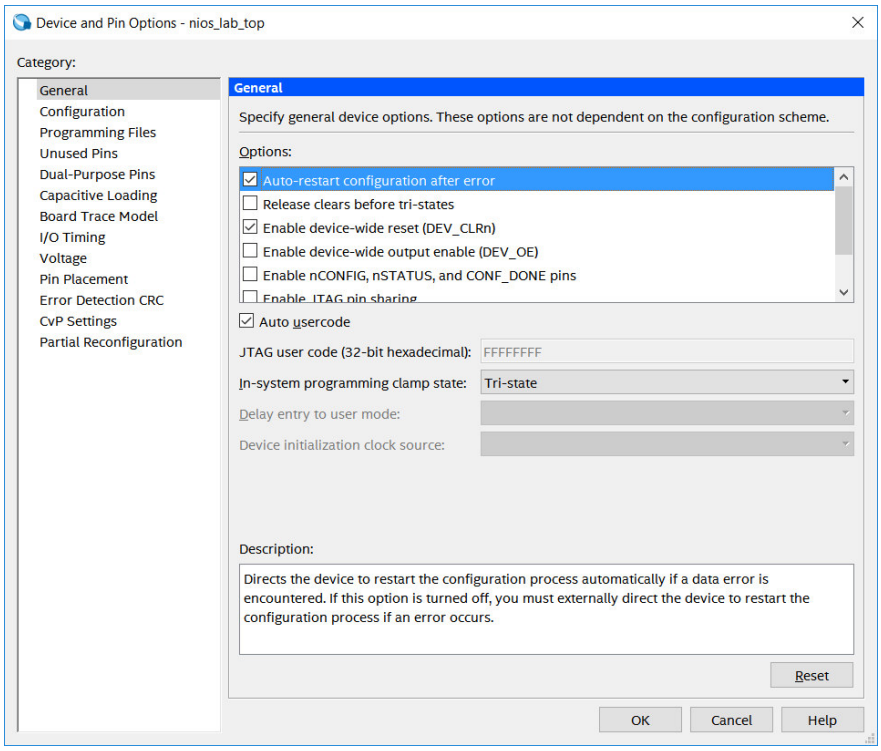
```
-- Component Declaration
   component RESET_GEN
      port (
            clk_in          : in  std_logic;
            reset_n_1       : in  std_logic;
            reset_n_2       : in  std_logic;
            reset_out_n     : out std_logic
      );
   end component;

------ INSERT nios_sys component declaration here ------
-------------------------------------------------------


-------------------------------------------------------
------ END of nios_sys component declaration      ------

begin

------ nios_sys connections ---------------------------------------
CONNECTED_TO_clk_clk              <= CLK12M;
LED                               <= CONNECTED_TO_pio_leds_export;
CONNECTED_TO_pio_lis3dh_export    <= (SEN_INT1 & SEN_INT2);
CONNECTED_TO_reset_reset_n        <= reset;
CONNECTED_TO_spi_lis3dh_MISO      <= SEN_SDO;
SEN_SDI                           <= CONNECTED_TO_spi_lis3dh_MOSI;
SEN_SPC                           <= CONNECTED_TO_spi_lis3dh_SCLK;
SEN_CS                            <= CONNECTED_TO_spi_lis3dh_SS_n;
-----------------------------------------------------------------------

-- Component Instantiation

------ INSERT nios_sys component instantiation here ------
-------------------------------------------------------


-------------------------------------------------------
------ END of nios_sys component instantiation    --------

u1 :  component RESET_GEN
      port map (
      clk_in            => CLK12M,
      reset_n_1         => RESET_n,
      reset_n_2         => CONNECTED_TO_pll_locked_conduit_export,
      reset_out_n       => reset
      );
```

### 6.2.19 Compile the Quartus Prime Project

With the hardware design complete, a few device settings need to be changed before the project can be compiled to create a configuration file. Those settings should have been already set in the project files, but would be beneficial to double-check and learn their importance.

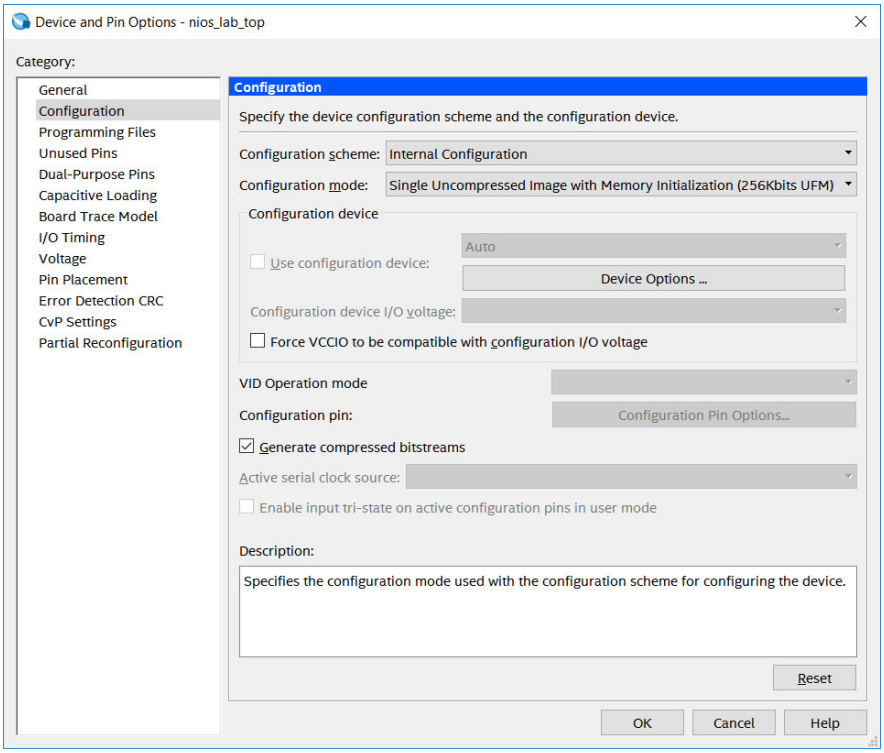
6.2.19.1 Open the device settings window from **Assignments → Device…** and click "Device and Pin Options".

6.2.19.2 In the General Category make sure the settings match the following picture.

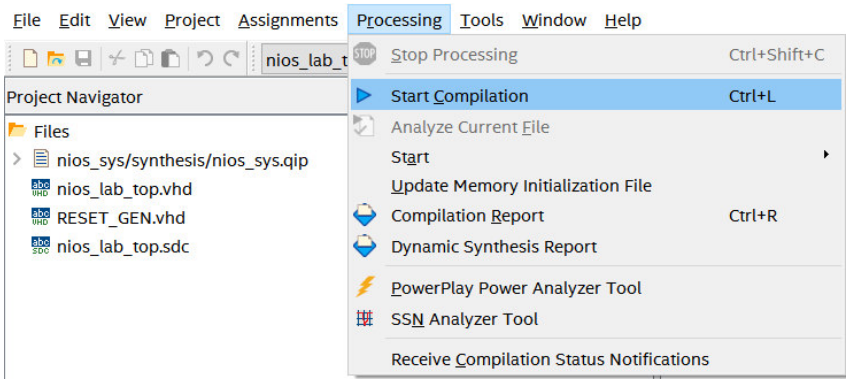



- "Enable DEV_CLRn": option not trivial as that pin is tied to Vcc. If Vcc drops low, this will reset the registers of the device.
- "Enable nCONFIG,nSTATUS,and CONF_DONE pins": nCONFIG is tied to RESET_n. If option is enabled, it will reset the configuration when the RESET_n button is pressed. Disable the option to use RESET_n as an input only and external reset for the embedded system.

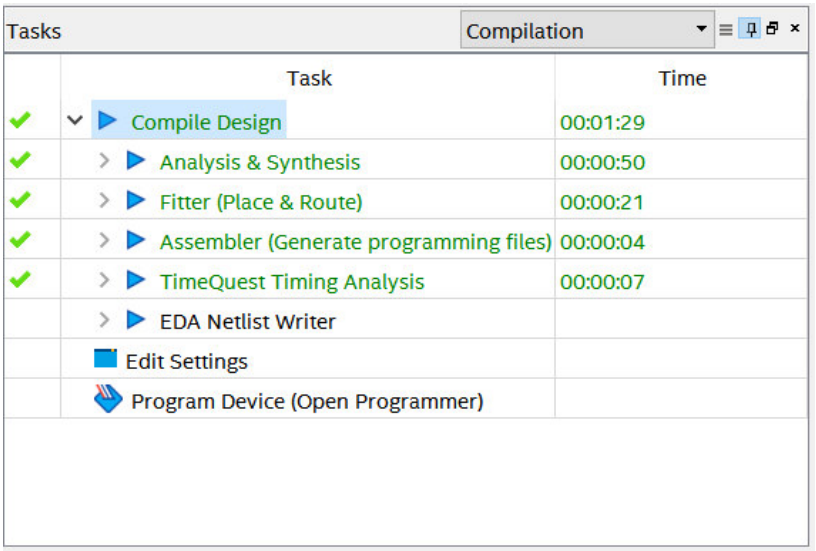
6.2.19.3 Under the Configuration category, check that "Single Uncompressed Image with Memory Initialization (256Kbits UFM) is set as the Configuration mode and ensure the other settings match the following:




6.2.19.4 Start the compilation by selecting **Processing → Start Compilation** or double-click Compile Design in the Tasks window.



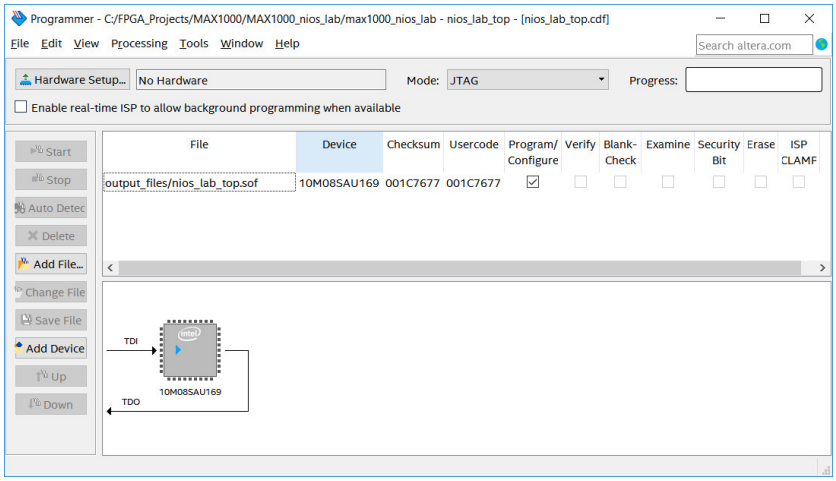
6.2.19.5 After few minutes the compilation should complete without any errors.




### 6.2.20 Download Configuration File to MAX1000

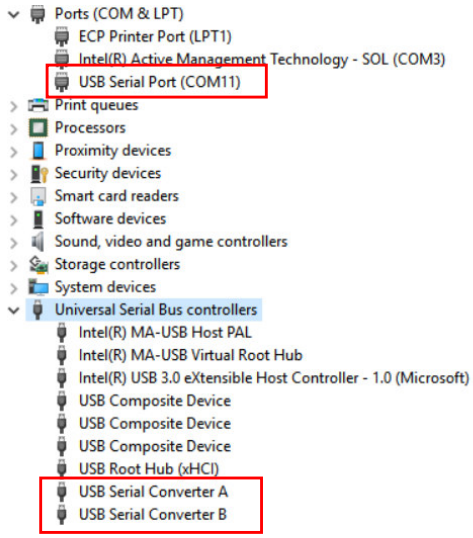
Now that hardware design is complete and has been converted into a configuration file, the MAX100 board needs to be programmed.

6.2.20.1 Open the Quartus Prime Programmer from **Tools → Programmer** or double-click on Program Device (Open Programmer) from the Tasks window. Since the MAX1000 board isn't connected yet, the Programmer should show a blank configuration.




*Note: With the newer releases of Quartus, the programming file .sof might already been added in the Programmer by default.*

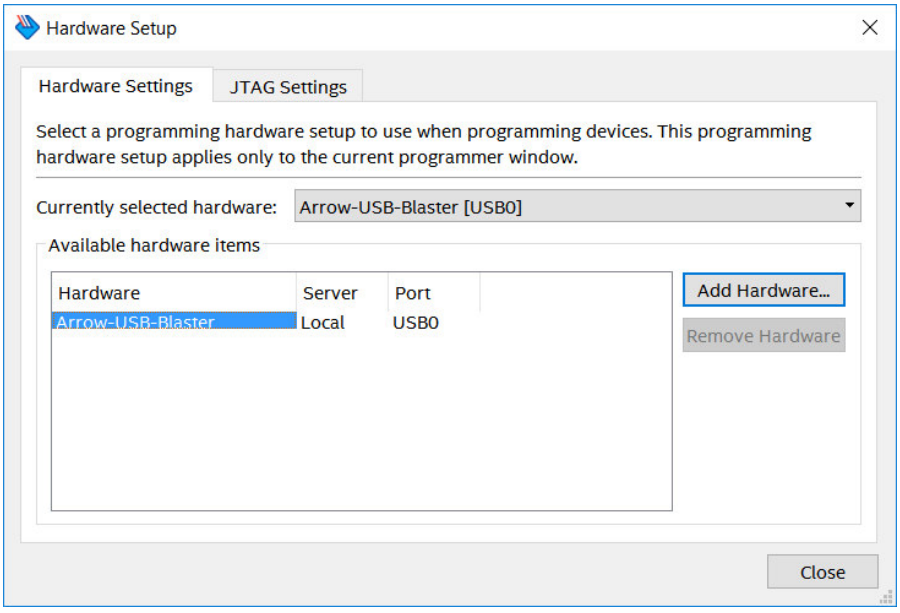
6.2.20.2 Connect your MAX1000 board to your PC using a USB cable. Since the Arrow USB Blaster should be already installed, the Window's Device Manager should display the following entries (port number may differ depending on your PC):




If not, please refer to MAX1000 User Guide for information on how to install the drivers properly.

You should see the green LED power on, indicating 3.3V applied voltage.

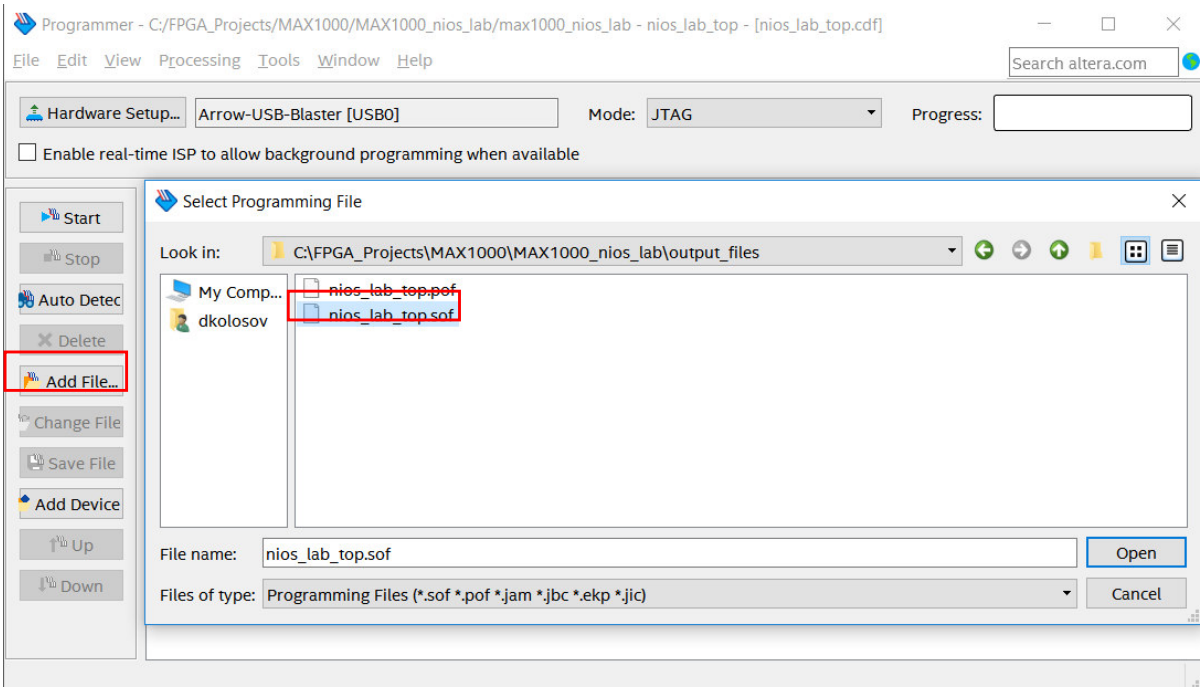
6.2.20.3 In the Programmer window, click "Hardware Setup.." and double-click the "Arrow-USB-BLASTER" entry in the Hardware panel. The "Currently selected hardware" drop-down box should display Arrow-USB-Blaster [USB0]. Depending on your PC, the port number may differ.



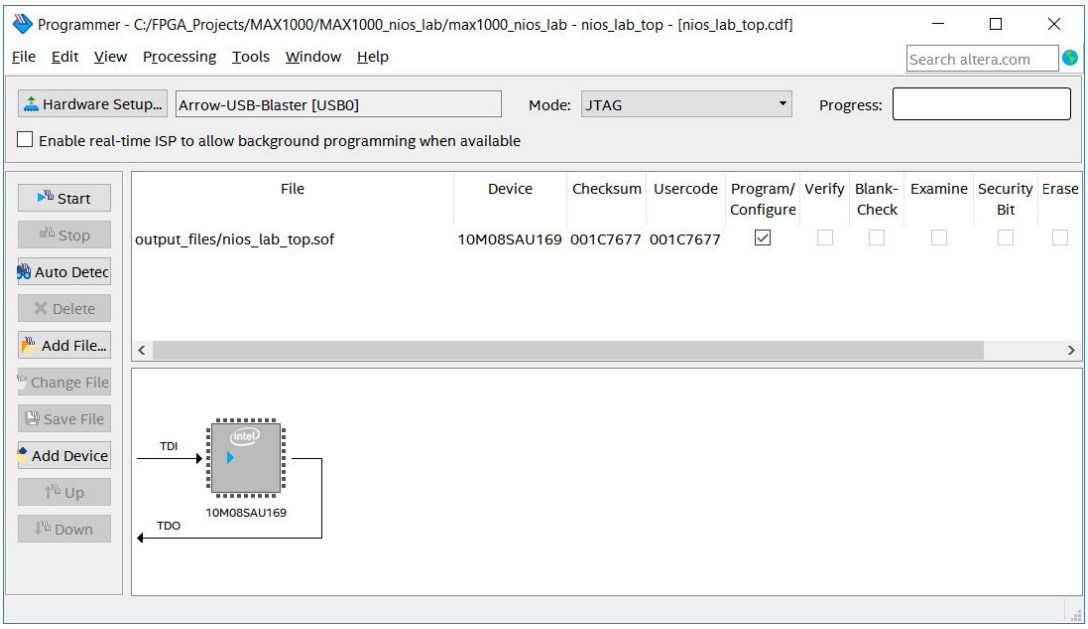

6.2.20.4 Click Close.

6.2.20.5 If the configuration file has been added by default, you can skip this step.

To add the configuration file, click “Add File..” and navigate through <project_directory>/output_files/ in you compilation directory. Open “nios_lab_top.sof” file.




6.2.20.6 Make sure the Programmer shows the correct file and the correct part in the JTAG chain as shown below:



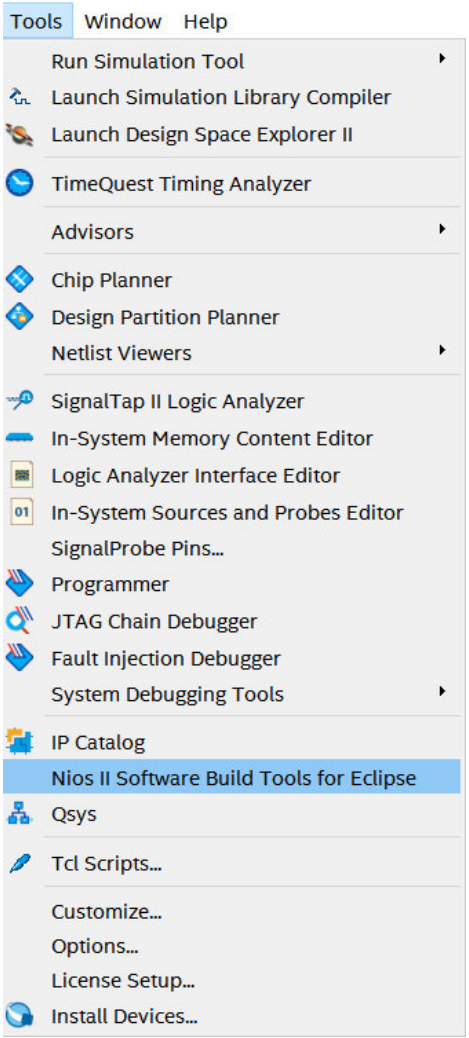
6.2.20.7 Make sure the Program/Configure checkbox is checked and click Start to program the MAX1000. You should see the CONF_D (red LED) toggle briefly to indicate that the configuration is complete and the Progress bar should reach 100% (Successful).

## 6.3 Build the Software Design

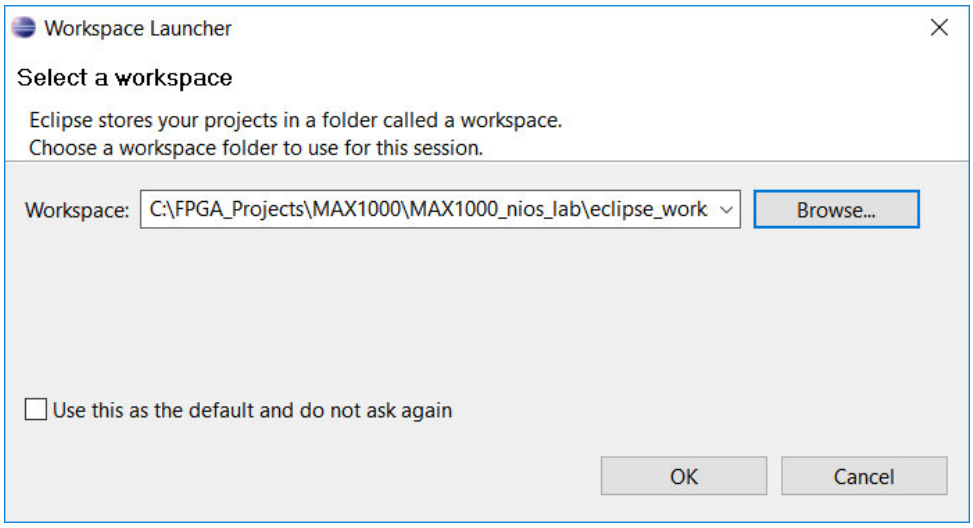
**Overview:** In this section, you will use the Nios II Software Build Tools (SBT) for Eclipse to quickly create a board support package (BSP) and a C software application to run on the Nios II processor you implemented in the previous step. The software has already been provided for you in the lab files.

### 6.3.1 Start the Nios II Software Build Tools for Eclipse

6.3.1.1 From the main Quartus Prime window, start SBT from **Tools → Nios II Software Build Tools for Eclipse**.



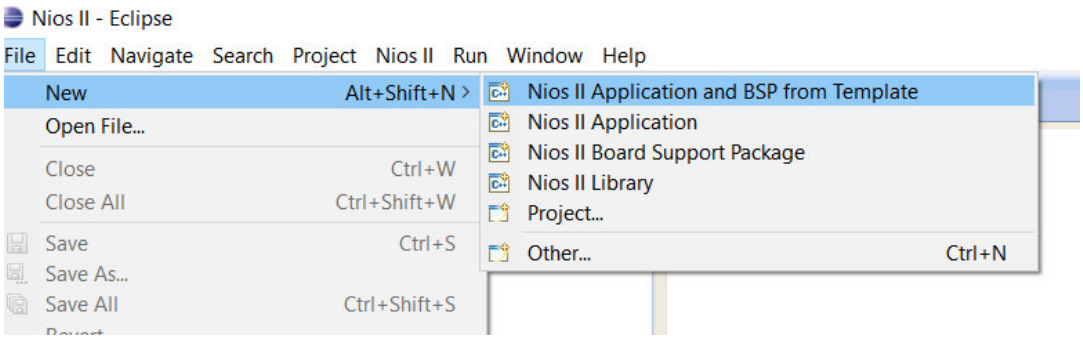
6.3.1.2 The Eclipse Workspace Launcher will open. Click “Browse…” and create a folder titled eclipse_workstation in your lab directory to use in the software directory for the project. Click “OK”.




### 6.3.2 Create a New Software Project

Now that Eclipse has a workstation, a new software application project and BSP can be created for your hardware system.

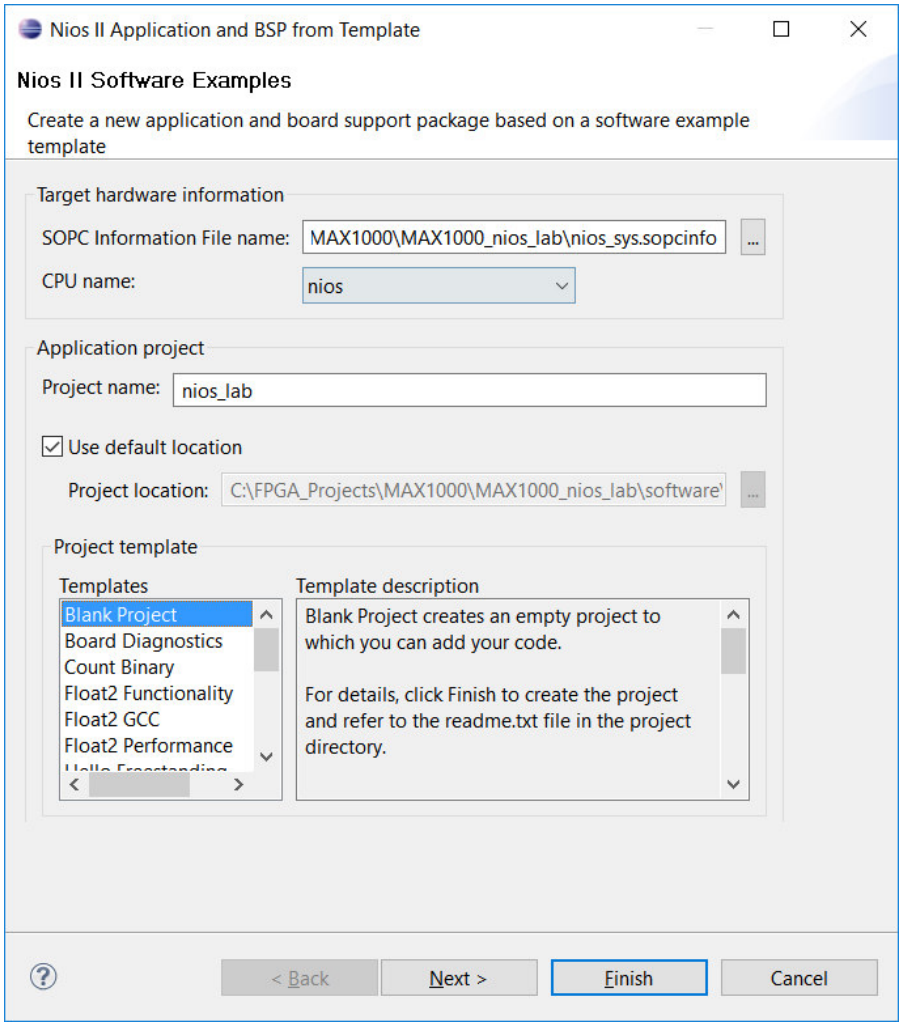
6.3.2.1 Once Eclipse opens the workbench in the Nios II prospective, select **File → New → New II Application and BSP from Template** as shown below. This is an easy way to create a BSP and application together in a few easy steps.



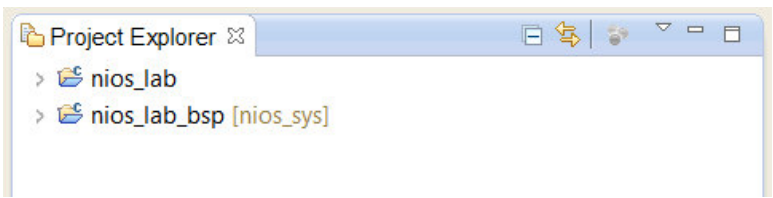

The BSP uses the Qsys- generated .sopcinfo file to import the necessary settings from the hardware project to the software project so that your application can run on the Nios II processor. It allows Eclipse to build the system library drivers and generate system-specific macros for the custom Qsys system with the Nios II processor.

6.3.2.2 Click "…" to select the nios_sys.sopcinfo from your project directory and name the project nios_lab. Make sure you select Blank Project from the Templates section as the software sources will be added in a later step. Make sure the settings match the screenshot below and select "Finish".

*Note: Your file path may differ from the one shown below.*




6.3.2.3 Eclipse will create two directories in the workspace; one for the application project and one for the BSP. The application directory (nios_lab) is currently empty while the BSP directory (nios_lab_bsp) contains software drivers, a system.h header file, initialization source code and other software infrastructure.

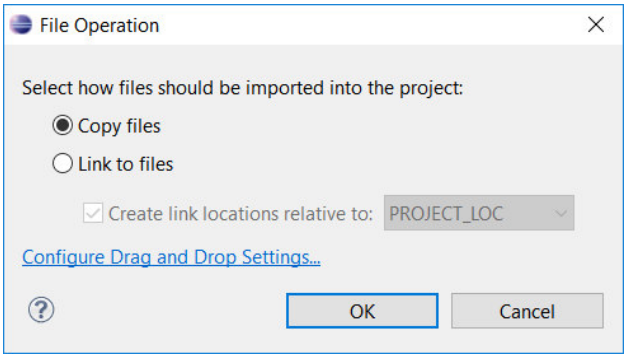


### 6.3.3 Add Source Code to the Project

The C source file have been provided for you in this lab. All that needs to be done is to copy it to your workspace.

6.3.3.1 From Windows Explorer, navigate to your main project directory. There you will find a file named main.c which you will need to copy to this project.

6.3.3.2 Select the main.c file and drag it into the nios_lab directory in Eclipse. Select the "Copy files" option in the pop-up and click "OK".




*Note: Since we are copying the files instead of linking to them, any changes that you would want to make to the source files need to be made to the versions inside the nios_lab directory. Otherwise, the changes will not be compiled.*

You should now see the new file appear under the nios_lab project in the Project Explorer.

6.3.3.3 In some cases, the familiar windows "do not enter" symbol appears indicating you cannot add files using the previous method. In this case, you can copy files using Windows Explorer. Copy the source file from the project directory into the nios_lab folder. In Eclipse, you need to right-click your nios_lab project and click Refresh.

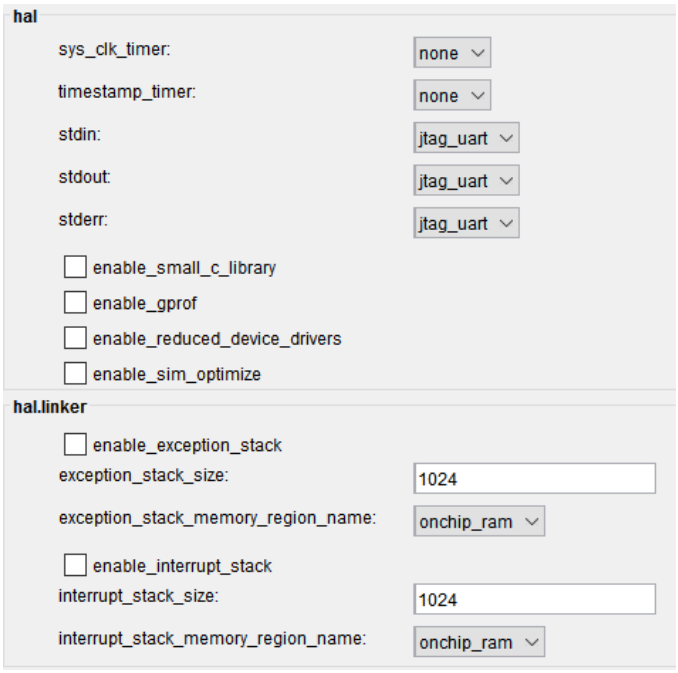
6.3.3.4 Using this method, the C-source files added to the project may not be automatically added to the Makefile. You will notice a white dot or green dot besides the source file. For the C-files, you need to make this dot green by right-clicking each .c file and selecting: Add to Nios II Build.

### 6.3.4 Configure the Board Support Package

The Board Support Package specifies the properties of the software system and needs to be configured for the software to execute correctly. Those properties include setting the stdin, stdout and stderr interfaces, memory allocation for the heap and stack, drivers, and wether an operating system will be used.

6.3.4.1 Right-click on the nios_lab_bsp project and select Nios II → BSP Editor… from the pop-up window.

6.3.4.2 The Nios II BSP Editor will open. In the Common settings under the Main tab, ensure the settings are configured as below.




Notice that since there is no operating system in this lab, the stdout, stdin, and stderr messages are reported through the JTAG UART which you will be able to see in the Nios II Console in Eclipse. On-chip memory will be used processor code storage, data storage, the exception, and interrupt stack.

Feel free to explore the BSP editor. The Drivers tab gives the user control over what drivers are built into the BSP. The Linker Script tab provides a mechanism to adjust what memory regions are utilized for certain purposes. We only have one memory in this system but for systems with multiple memory locations (i.e. DDR3, flash, and on-chip ram), this is particularly useful.

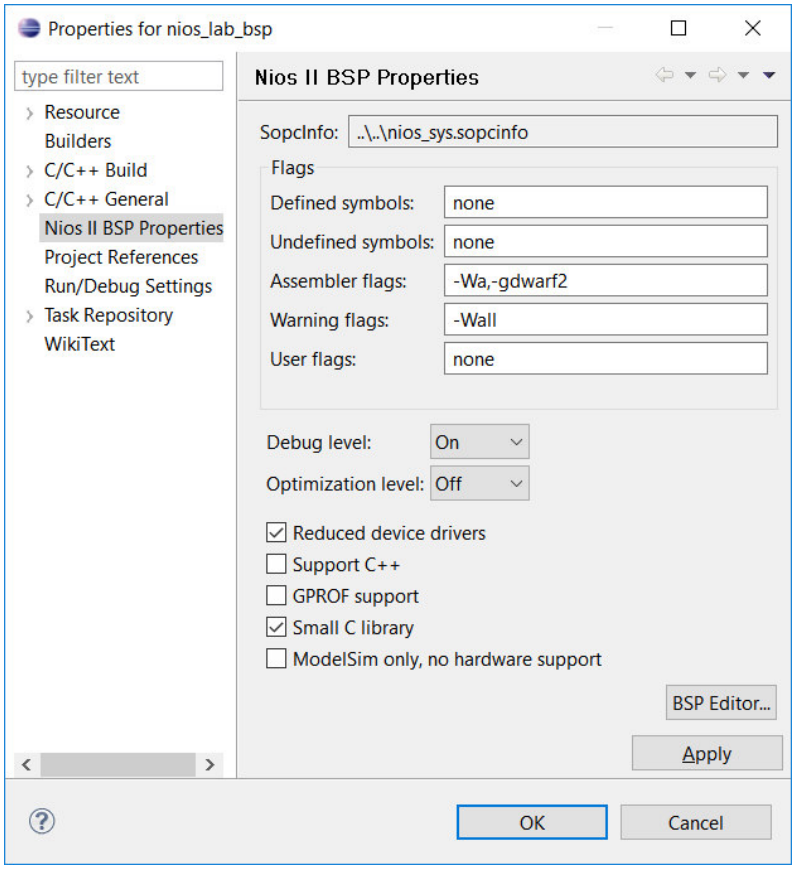
6.3.4.3 Click “Generate” button to update the BSP and select “Exit” to close it once the process is complete.

6.3.4.4 There are a few more BSP settings to edit. Right-click on the nios_lab_bsp project and select Properties from the pop-up menu.

6.3.4.5 In the Properties window, select the Nios II BSP Properties tab. It may take a moment to load the settings.

6.3.4.6 To keep the software footprint small so it fits our device, enable “Reduced device drivers” and “Small C library” options. As there is no C++ code, disable the “Support C++” option.

The BSP Properties should match the following:

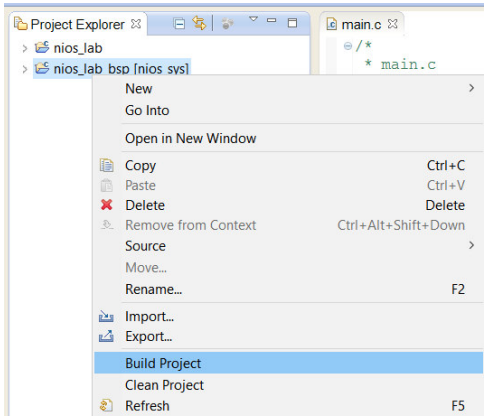



6.3.4.7 Select “Apply” and then click “OK” to exit the Properties Window.

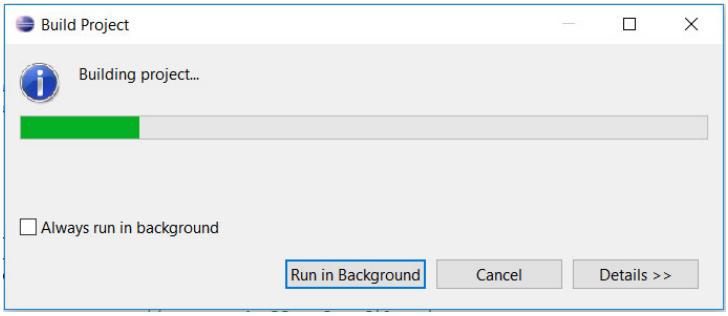
### 6.3.5 Build the Software

With all of the appropriate settings configured, you can now build the BSP and software project using the next two steps to produce an executable and linked format (.elf) file to run on the MAX1000 board.

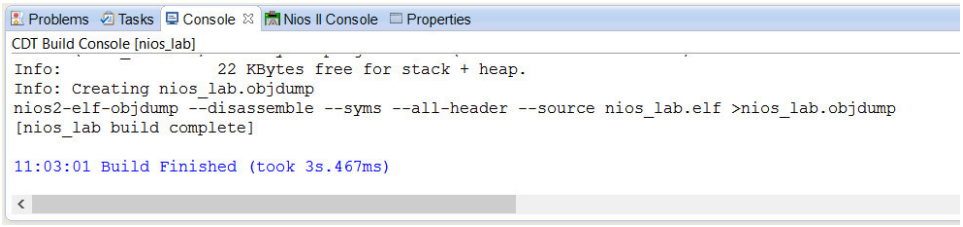
6.3.5.1 Right-click on the nios_lab_bsp project and select Build Project from the pop-up menu to build the BSP.



You can have the build process run in the background by from the pop-up window if you wish. You can observe the process commands in the Console window.




6.3.5.2 Repeat the procedure for the application. Right-click the nios_lab project and select Build Project from the pop-up menu.




6.3.6 Run the Application on the MAX1000 Board

**Overview:** Now that you have an executable, you can download the application to the on-chip memory in the MAX10 and the Nios II processor will execute.

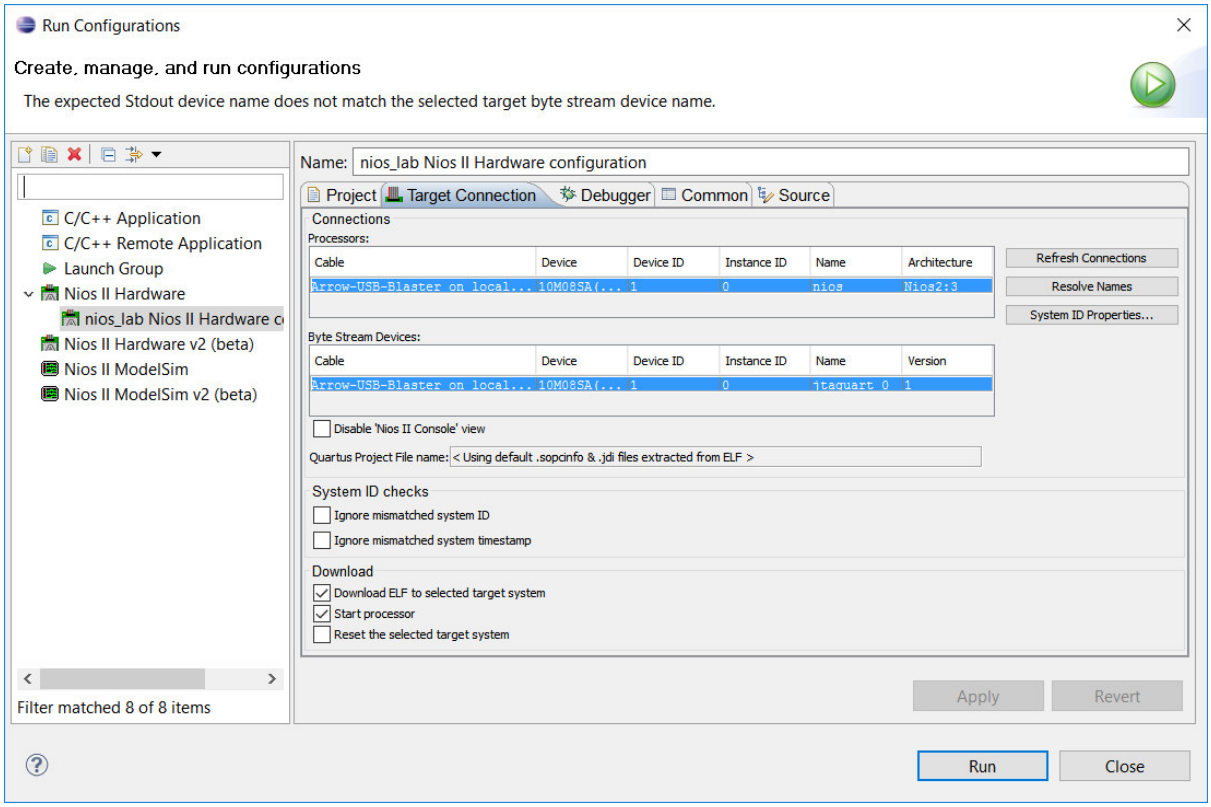
6.3.7 Download the Executable to the MAX1000

First, a target configuration will need to be established with the MAX1000 board so that Eclipse can download the code and communicate with the board.

6.3.7.1 Right-click on the nios_lab software project and select **Run As → Nios II Hardware**.

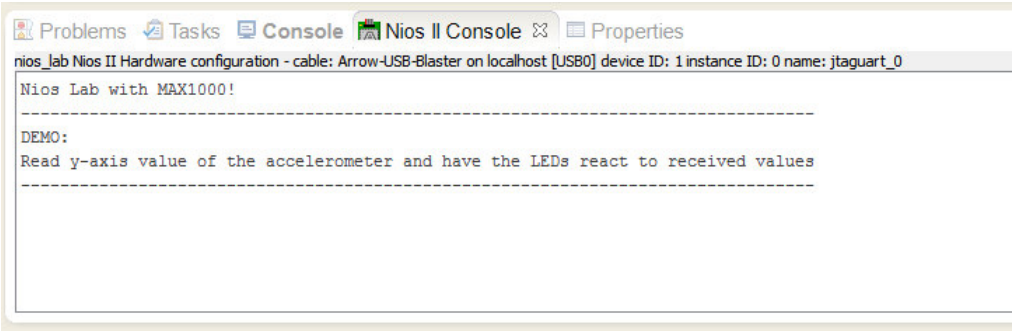
This will rebuild the software project to create an up-to-date executable and download the code into the memory of the MAX10. The debugger then resets the Nios II and it begins executing the code.

*Note: If a Run Configuration dialogue appears, you may need to click the Target Connection tab and scroll to the right. Click "Refresh Connections" and the appropriate connection to the MAX100 should appear as below. Then click "Run".*




In the “System ID Properties” you can verify if the target device has the matching Device ID that we previously set Qsys for System Device ID.

6.3.7.2 After a few seconds, the Nios II Console should open at the bottom of the Eclipse:




After this message, the software downloaded to MAX1000 will obtain the y-axis data from its on-board accelerometer and toggle it’s LEDs accordingly to the tilt level. Every 10ms the y-axis value will be sent to Nios II Console window.

#### 6.3.7.3 Next Steps

Having successfully downloaded a configuration hardware image containing a Nios II processor to the FPGA and a software executable, you can now experiment with your own application ideas. The design flow would be the same but you are able to add more components/peripherals to the embedded system in Qsys and expand the system's capabilities.

There are various methods to try other applications. Here are some optional steps:

1) Edit existing C-source file.
2) Create new Qsys with additional components along with a new software project following the steps used in this lab.

#### 6.3.7.4 Non-volatile Configuration

In this lab, a volatile configuration(.sof) was created for the MAX10, meaning that on power off all configurations are erased. If you wish to create a non-volatile configuration file that includes the hardware and software files in one file for the MAX10 please refer to AN730 for details.

Link: https://www.altera.com/en_US/pdfs/literature/an/an730.pdf

**CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED THE NIOS LAB!**

# 7. Revisions

| Version | Change Log | Date of Change |
|---|---|---|
| V1.0 | Initial Version | 17/08/2017 |

# 8. Legal Disclaimer

**ARROW ELECTRONICS**

**EVALUATION BOARD LICENSE AGREEMENT**

By using this evaluation board or kit (together with all related software, firmware, components, and documentation provided by Arrow, "Evaluation Board"), You ("You") are agreeing to be bound by the terms and conditions of this Evaluation Board License Agreement ("Agreement"). Do not use the Evaluation Board until You have read and agreed to this Agreement. Your use of the Evaluation Board constitutes Your acceptance of this Agreement.

**PURPOSE**

The purpose of this evaluation board is solely intended for evaluation purposes. Any use of the Board beyond these purposes is on your own risk. Furthermore, according the applicable law, the offering Arrow entity explicitly does not warrant, guarantee or provide any remedies to you with regard to the board.

**LICENSE**

Arrow grants You a non-exclusive, limited right to use the enclosed Evaluation Board offering limited features only for Your evaluation and testing purposes in a research and development setting. Usage in a live environment is prohibited. The Evaluation Board shall not be, in any case, directly or indirectly assembled as a part in any production of Yours as it is solely developed to serve evaluation purposes and has no direct function and is not a finished product.

**EVALUATION BOARD STATUS**

The Evaluation Board offers limited features allowing You only to evaluate and test purposes. The Evaluation Board is not intended for consumer or household use. You are not authorized to use the Evaluation Board in any production system, and it may not be offered for sale or lease, or sold, leased or otherwise distributed for commercial purposes.

**OWNERSHIP AND COPYRIGHT**

Title to the Evaluation Board remains with Arrow and/or its licensors. This Agreement does not involve any transfer of intellectual property rights ("IPR) for evaluation board. You may not remove any copyright or other proprietary rights notices without prior written authorization from Arrow or it licensors.

**RESTRICTIONS AND WARNINGS**

Before You handle or use the Evaluation Board, You shall comply with all such warnings and other instructions and employ reasonable safety precautions in using the Evaluation Board. Failure to do so may result in death, personal injury, or property damage.

You shall not use the Evaluation Board in any safety critical or functional safety testing, including but not limited to testing of life supporting, military or nuclear applications. Arrow expressly disclaims any responsibility for such usage which shall be made at Your sole risk.

**WARRANTY**

Arrow warrants that it has the right to provide the evaluation board to you. This warranty is provided by Arrow in lieu of all other warranties, written or oral, statutory, express or implied, including any warranty as to merchantability, non-infringement, fitness for any particular purpose, or uninterrupted or error-free operation, all of which are expressly disclaimed. The evaluation board is provided "as is" without any other rights or warranties, directly or indirectly.

You warrant to Arrow that the evaluation board is used only by electronics experts who understand the dangers of handling and using such items, you assume all responsibility and liability for any improper or unsafe handling or use of the evaluation board by you, your employees, affiliates, contractors, and designees.

**LIMITATION OF LIABILITIES**

In no event shall Arrow be liable to you, whether in contract, tort (including negligence), strict liability, or any other legal theory, for any direct, indirect, special, consequential, incidental, punitive, or exemplary damages with respect to any matters relating to this agreement. In no event shall arrow's liability arising out of this agreement in the aggregate exceed the amount paid by you under this agreement for the purchase of the evaluation board.

**IDENTIFICATION**

You shall, at Your expense, defend Arrow and its Affiliates and Licensors against a claim or action brought by a third party for infringement or misappropriation of any patent, copyright, trade secret or other intellectual property right of a third party to the extent resulting from (1) Your combination of the Evaluation Board with any other component, system, software, or firmware, (2) Your modification of the Evaluation Board, or (3) Your use of the Evaluation Board in a manner not permitted under this Agreement. You shall indemnify Arrow and its Affiliates and Licensors against and pay any resulting costs and damages finally awarded against Arrow and its Affiliates and Licensors or agreed to in any settlement, provided that You have sole control of the defense and settlement of the claim or action, and Arrow cooperates in the defense and furnishes all related evidence under its control at Your expense. Arrow will be entitled to participate in the defense of such claim or action and to employ counsel at its own expense.

**RECYCLING**

The Evaluation Board is not to be disposed as an urban waste. At the end of its life cycle, differentiated waste collection must be followed, as stated in the directive 2002/96/EC. In all the countries belonging to the European Union (EU Dir. 2002/96/EC) and those following differentiated recycling, the Evaluation Board is subject to differentiated recycling at the end of its life cycle, therefore: It is forbidden to dispose the Evaluation Board as an undifferentiated waste or with other domestic wastes. Consult the local authorities for more information on the proper disposal channels. An incorrect Evaluation Board disposal may cause damage to the environment and is punishable by the law.