

# **The Nios II Embedded “Hello World” Lab: For The MAX 10 Development Kit**

March 2016

Version 1.5

## Revision History

1.0	5/1/2015	L. Landis	Initial Release
1.1	6/2/2015	L. Landis	Added BeMicro
1.2	11/30/2015	I. Rush	Added CVE DevKit
1.3	12/2/2015	S Meer	Consolidated Sections
1.4	12/4/2015	I.Rush	Updated Pinout Table
1.5	3/17/2016	K. Kita	Separated Lab by Board

## Contents

Overview .....	3
Lab Notes .....	4
Quartus Installation .....	4
Design Flow .....	5
Objective of hardware design for the “Hello World” lab.....	6
HARDWARE DESIGN .....	7
Initial Setup .....	7
Get started with Quartus .....	9
Building your Qsys based processor system .....	12
Building the top level design.....	24
Adding the Nios II system into your design .....	30
Adjusting	
SOFTWARE DESIGN .....	33
Creating the Software for the “Hello World” design .....	33
MAX10 FPGA Development Kit Cable Connections and Switch Settings.....	40
Lab Summary .....	45

## Overview

This lab teaches you how to create an embedded system implemented in programmable logic using Altera’s “soft” Nios II processor. A soft processor such as the Nios II, available in Altera’s FPGA families, have built in programmable logic fabric and can be easily modified to suit an application’s requirements. Altera’s “SoC FPGA” families are hard processor built from “hard” standard cells that cannot be changed without redesigning the chip. The Nios II processor is supported by a rich set of peripherals and “IP” blocks built that can be configured and connected to the processor using Altera’s QSys tool within the Quartus II design tool set. Altera also distributes the Nios II Software Build Tools (SBT) for Eclipse (for software development) within the Quartus development suite.

The lab is organized to run on a number of Altera development kits. The links to the other kits can be found in the [Design Store](#) as a Design Example and type in “hello” in the search bar. This lab will show you how to install the Max 10 Development kit pin settings, design the processor-based hardware system, download it to the MAX 10 Development Kit, and run a simple “Hello World” software program which displays text on your terminal. The initial section of

the lab is split into a Hardware Section and Software Section. You can skip the hardware section and move directly to the software section should you choose.

Refer to Appendix A in the Design Example for development kit specific pinout names and configuration of on-chip memory.

## Lab Notes

The lab will require you to choose files, components, and other objects; they must be spelled exactly as directed. This is necessary for consistency and to ensure that each step works properly in the lab, when creating your own systems you can choose your own names as long as you use them consistently in your project. The directory paths shown in the figures are for Linux (forward slash directory delimiter). If you are using Windows, the paths will be shown with backslash directory delimiters.

## Quartus Installation

Quartus is Altera's design tool suite. It serves a number of functions:

1. Design creation through the use of HDL languages or schematics
2. System creation through the Qsys graphical interface
3. Generation and editing of constraints: timing, pin locations, physical location on die, IO voltage levels
4. Synthesis of high level language into an FPGA netlist ("mapping" in FPGA terminology)
5. FPGA place and route ("fitting" in FPGA terminology)
  - a. Generation of design image (used to program FPGA, "assembly" in FPGA terminology)
6. Timing Analysis
7. Programming/download of design image into FPGA hardware
8. Debugging by insertion of debug logic (in-chip logic analyzer)
9. Interfaces to 3rd party tools such as simulators
10. Launching of Software Build Tools (Eclipse) for Nios II

To download Quartus, follow these instructions:

Visit this site: <http://dl.altera.com/?edition=web> to download version 15.1 of Quartus II.

Select version 15.1 and your PC's operating system.

For the smallest installation, and quickest download time, enter only the technology families you are using based on your development board.

Download Center

Get the complete suite of Altera design tools

Quartus® Prime

Design Software

Design Software

Embedded Software

Archives

Licensing

Programming Software

Drivers

Board System Design

Board Layout and Test

Legacy Software

Quartus Prime Lite Edition

Release date: November, 2015

Latest Release: v15.1

Select release: 15.1

Operating System

Windows

Linux

Download Method

Akamai DLM3 Download Manager

Direct Download

✓

The Quartus Prime software version 15.1 supports the following device families: Arria II, Cyclone IV, Cyclone V, MAX II, MAX V, and MAX 10 FPGA. [More](#)

Combined Files

Individual Files

DVD Files

Additional Software

Download and install instructions: [More](#)

[Read Altera Software v15.1 Installation FAQ](#)

[Quick Start Guide](#)

✓ Select All

✓ Quartus Prime Lite Edition (Free)

✓ Quartus Prime (includes Nios II EDS)

Size: 1.4 GB MD5: 2F68BB58C8E484957117E344FA72AE8F

✓ ModelSim-Altera Edition (includes Starter Edition)

Size: 1.1 GB MD5: BA57EE2A66CB560AE8429221A7F7C616

✓ Devices

You must install device support for at least one device family to use the Quartus Prime software.

✓ Arria II device support

Size: 497.7 MB MD5: 48577BD12E186361C7DE923E4CD19074

✓ Cyclone IV device support

Size: 463.5 MB MD5: FD95042C8C58782FF6C25C25EA83CA2E

✓ Cyclone V device support

Size: 1.1 GB MD5: 7F108A307455ACDC3CF6DA21B1FBF211

✓ MAX II, MAX V device support

Size: 11.3 MB MD5: DEACB97D4A14A952521B6F1DFBCB958F

✓ MAX 10 FPGA device support

Size: 338.5 MB MD5: C132D3685C78B3706B36E2C23A0F8209

Download Selected Files

Note: The Quartus Prime software is a full-featured EDA product. Depending on your download speed,

Figure 1: Quartus download page

Follow the download instructions provided from the web page. No license is required to run the Quartus Lite software.

## Design Flow

Unlike system development with hard processors, development with soft processors enables you to optimize the processor system to your application requirements and use the FPGA to

5

add the performance and interfaces required by your system. This means that you need to know how to modify the processor system hardware; this may sound challenging but thanks to the Qsys graphical system design tool this is actually a relatively easy thing to do as we will demonstrate in this lab.

The Qsys design flow diagram below illustrates how an overall system is integrated using the combination of the Qsys system integration tool, Quartus for mapping (FPGA terminology for synthesis), fitting (FPGA terminology for place and route), and the NIOS Software Build Tool (SBT) for software development.

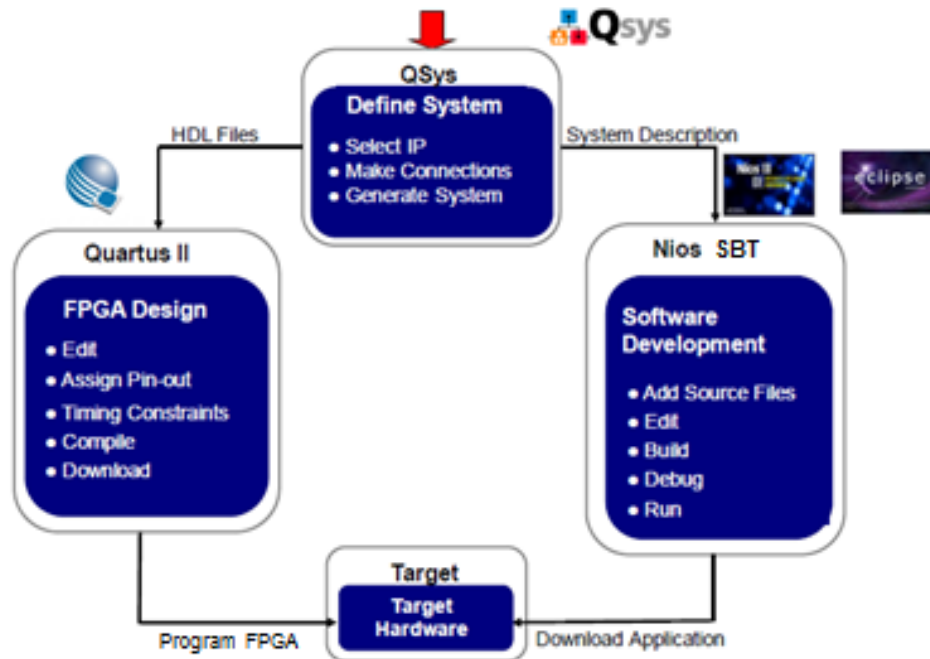


Figure 2: Qsys Development Flow

The above diagram depicts the typical flow for Nios II system design. Hardware System definition is performed using Qsys; the resultant HDL files from the Qsys system are used by the Quartus II FPGA design software to map, fit and download the hardware image into the FPGA device. Quartus II also generates information that describes the configuration of the system designed in Qsys so that the Nios II SBT can be configured to create a software library that matches the hardware system and contains all the correct peripheral drivers.

## Objective of hardware design for the “Hello World” lab

For the simplest example, ‘a hello world lab’, the processor will load a program that prints “Hello World” to the screen. This requires a working processor to execute the code, on-chip memory to

store the software executable, and a JTAG UART peripheral to send the “Hello World” text to a terminal. To make the lab a little bit more interesting and hardware-centric, we will utilize the push button switches and LEDs to allow interaction with the development kit. We will show two different ways to connect the push button to the LED, one through a direct connection in the Verilog code using an assign statement, and one using connections to memory that the processor can access. Note that in the appendix section, there are a number of more advanced modules including how to connect to off-chip memory, utilizing interrupts and displaying hello world on an LCD character display.

The lab hardware is constructed with the components shown below. Altera utilizes the Qsys network on chip interconnect to connect the master and slave devices together. To get a clear understanding of how quickly one can build an Embedded System using Qsys and the Quartus Design Software you will build the Nios II system entirely from scratch.

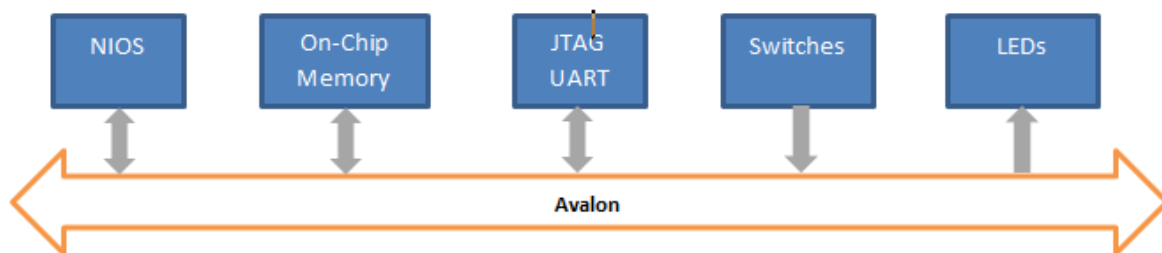


Figure 3: Nios II based system used in this lab

## HARDWARE DESIGN

### Initial Setup

Should you want to skip the hardware design section, continue in the section called SOFTWARE DESIGN. The screenshots in the hardware design section are based on the **Max 10 FPGA Development Kit**.

Altera provides a starting point design to get the FPGA device pinouts associated with the development kits layout and your design via what is called the Baseline design. Navigate to Altera’s design store for the kit that you are using: <https://cloud.altera.com/devstore>

Click on Design Examples.

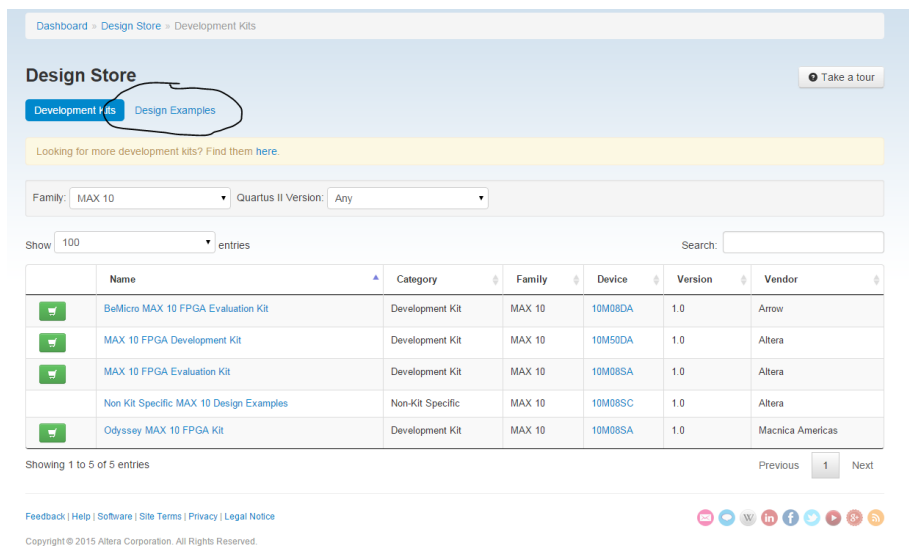


Figure 4: Design Store

Once in Design Examples, filter by respective development kit and Quartus version number.

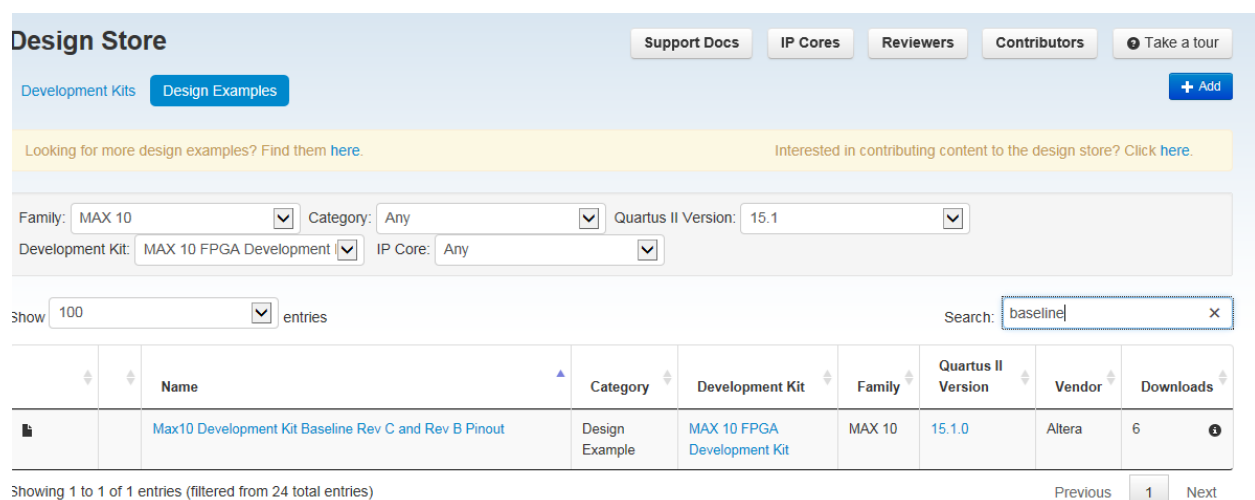


Figure 5: Design Examples under Design Store (note that this list changes over time and might not look the same as this picture)

Select the MAX10 FPGA Development Kit Baseline Design or other supported development kit you are using for this lab.



Dashboard » Design Store » Design Examples » Max10 Development Kit Baseline Rev C and Rev B Pinout

## Design Store

Support Docs IP Cores Reviewers Contributors Take a tour

### Max10 Development Kit Baseline Rev C and Rev B Pinout

14.1.0 15.0.0 15.1.0 [Edit](#)

Category	Design Example
Name	Max10 Development Kit Baseline Rev C and Rev B Pinout
Description	This design contains device pinout only and can be used as a starting point for designing with your MAX10 FPGA Development Kit. You can change the pin names as needed in the Verilog HDL code and the .qsf files (or with the Assignment Editor). Pin locations are locked down on the board. Read through the readme file to convert the Rev C baseline to Rev B baseline
Operating System	None
Version	1.0
Family	MAX 10
Device	10M50DA
Documentation	<a href="#">Add new documentation</a>
Development Kit	MAX 10 FPGA Development Kit
Installation Package	<a href="#">Download</a>

Figure 6: MAX 10 Development Baseline Design Example

Select the Download button and save the baseline.par design locally to your lab working directory (call the directory devkit\_hello\_world).

## Get started with Quartus

Now you are ready to get started designing hardware! Launch Quartus by double clicking the Quartus icon.

Next you will launch the New Project Wizard from Quartus from the main panel or alternatively File → New Project Wizard.

Home

## Start Designing

**New Project Wizard** **Open Project**

Recent Projects

- fred.qpf** (C:/Users/Ilandis/Documents/an490\_design\_example/fred.qpf)
- Hello\_World.qpf** (C:/Users/Ilandis/Documents/Hello\_World\_Lab/Hello\_World.qpf)
- Hello\_World\_v1.qpf** (C:/Users/Ilandis/Documents/Hello\_World\_Lab/Hello\_World\_v1.qpf)
- foo.qpf** (C:/Users/Ilandis/Documents/an488\_design\_example/foo.qpf)

Figure 7: Quartus Main Panel

Fill in the New Project Wizard first panel with your devkit\_hello\_world directory and project which we will also call hello\_world\_lab.

**New Project Wizard <@sj-swcf5690-005>**

**Directory, Name, Top-Level Entity**

What is the working directory for this project?

/home/lldis/TEMP/devkit\_hello\_world

What is the name of this project?

hello\_world\_lab

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

hello\_world\_lab

Use Existing Project Settings...

< Back   Next >   Finish   Cancel   Help

Figure 8: New Project Wizard first panel

Click next and select project template and click next.

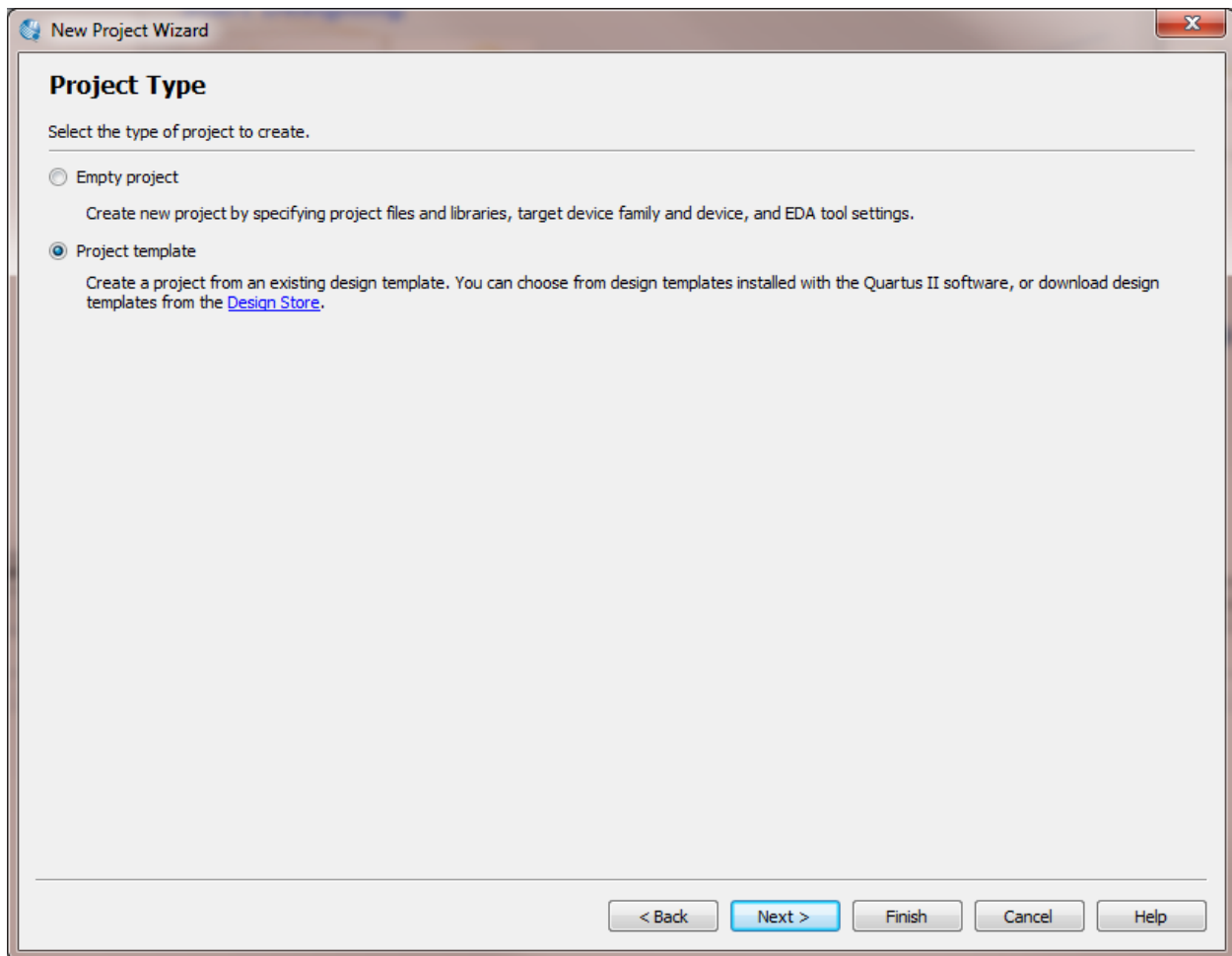


Figure 9: New project wizard second panel



Figure 10: Design Template Install (note the name of the “.par” file will change based on the name of the development kit you are using).

Once you hit ok, Quartus loads this starting point design that contains the pinout for the specific hardware device based on the Development Kit. Note that only a handful of pins are needed for the lab, but you can rely on the settings utilized in the Baseline project to make sure the right pin locations and voltage settings are correct for your project.

## Building your Qsys based processor system

The Figure 11 diagram illustrates what you are designing in the Qsys environment. This system has a single master, the Nios II processor, and 4 slave devices. Building the Qsys system is a highly efficient way of designing systems with or without a processor.

Launch Qsys from Quartus: Tools → Qsys. The initial screen looks something like this:

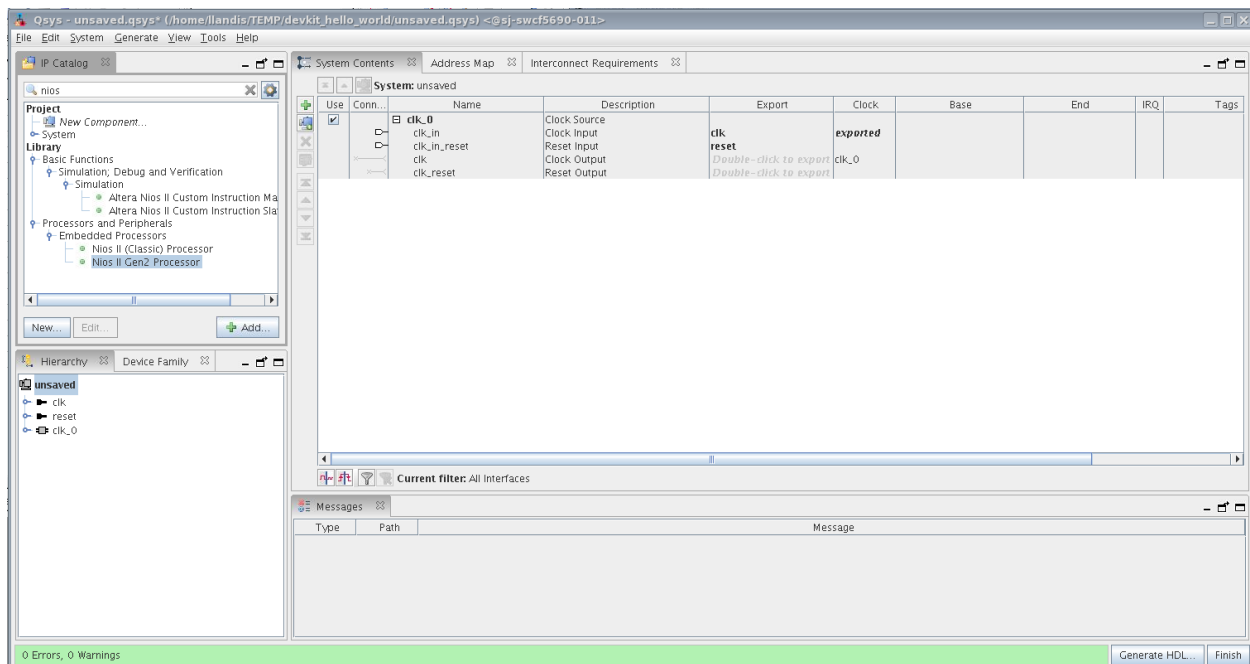


Figure 11: Qsys main panel

Next, we will add the various components of the system and make the connections between them. By default Qsys inserts a clock module. We will connect to this later on in the lab.

Below the IP catalog tab, you can search for the various components you want to add to your Qsys based system. Enter Nios in the search tab and select the Nios II processor from the library.

A configuration window will appear, in this select the Nios II/e processor. This version of the Nios II processor is resource optimized and will work well for the Hello World Lab implementation.

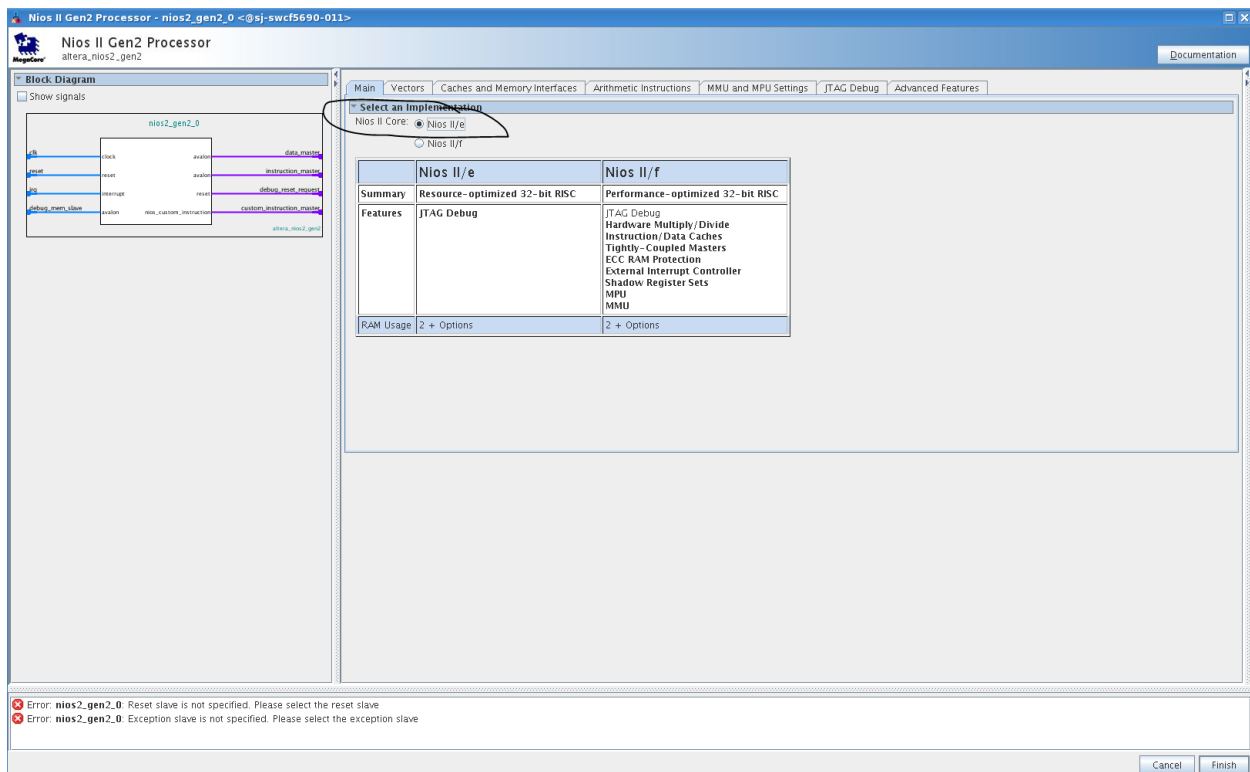


Figure 12: Nios II Gen 2 Configuration panel

Click finish and you will see the Nios IIe processor in your connection diagram. For now don't worry about the system errors reported, we will address them soon.

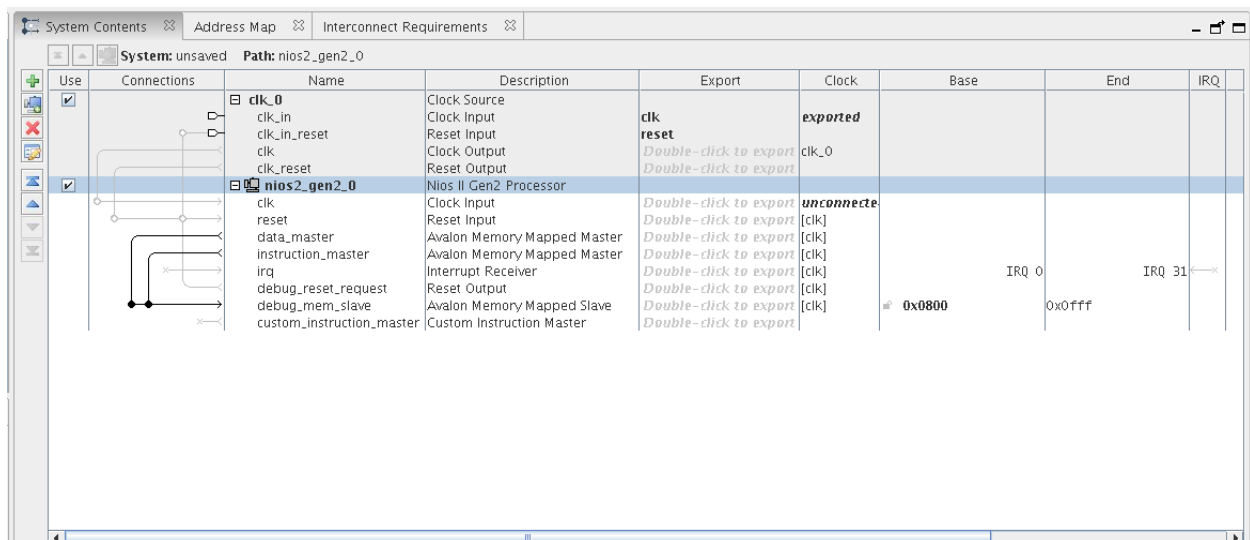


Figure 13: Qsys System Contents panel

Qsys has a very elegant and efficient way of making connections by clicking on the nodes on 'wires' in the connections panel on the 2<sup>nd</sup> column from the left. You can add the connections as

you add components, but it's often easier to make all the connections once you have finished adding the various blocks. With the Nios II processor added, you still need to add the On Chip Memory, JTAG UART, SWITCHES and LED to your system defined in Figure 3: Nios II based system.

Search for memory in the IP catalog. You will see many options for memory. It might be easiest to detach the IP Catalog from the main panel by clicking on the detach window icon.



Figure 14: Detach window icon

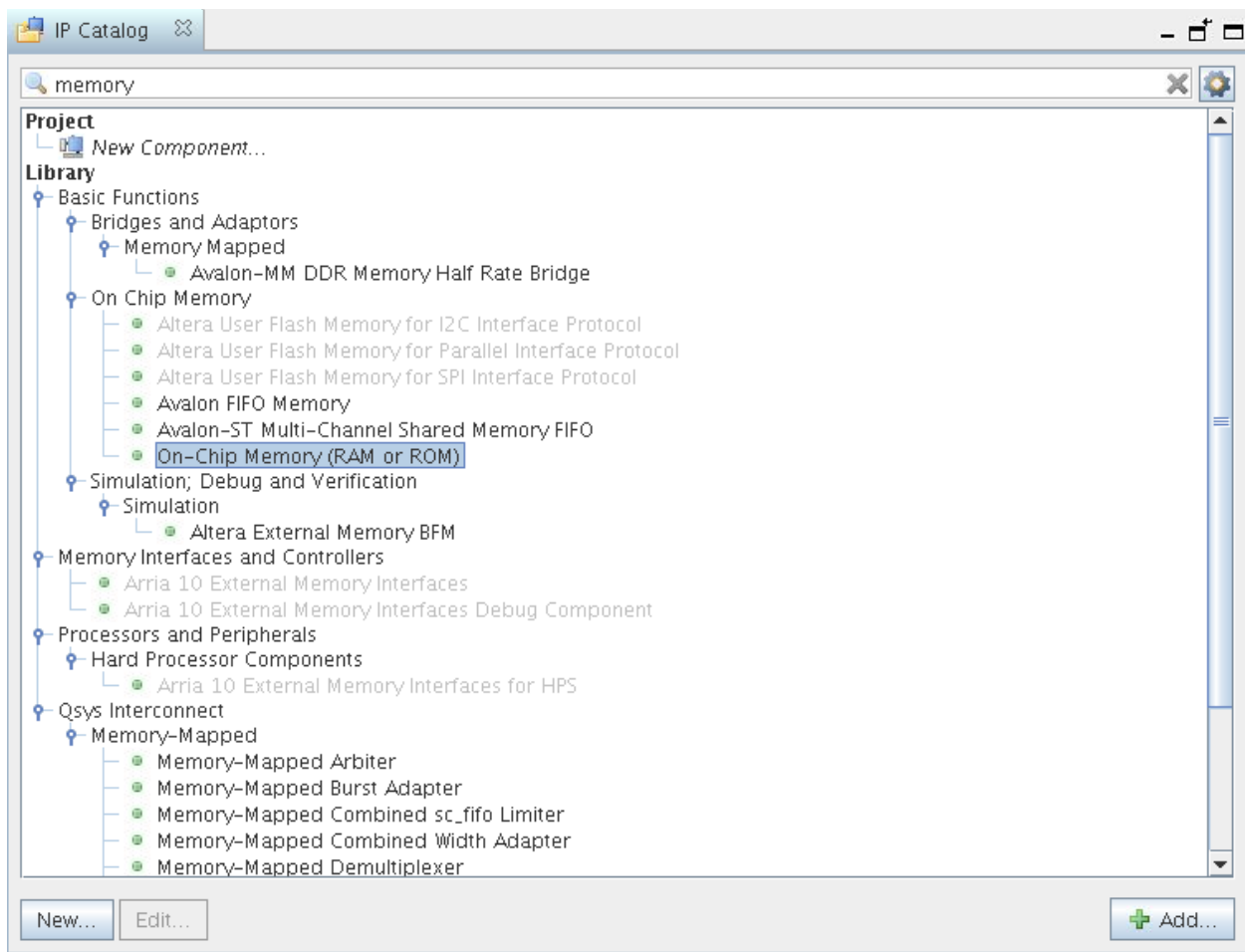





Figure 15: IP catalog search for on chip memory

Locate the On-Chip Memory (RAM or ROM) component and click Add. You can use all of the default settings except that you need to change the memory size from 4096 to 16384. This will ensure that you have a plenty of space for your software program. Uncheck initialize memory content. This feature includes the software executable in the hardware image. For this lab, you will initialize the software executable from Eclipse.

Parameters   

System: nios\_setup\_v2 Path: onchip\_memory

### On-Chip Memory (RAM or ROM)

altera\_avalon\_onchip\_memory2 Details

**Memory type**

Type: RAM (Writable) ▼

☐ Dual-port access

☐ Single clock operation

Read During Write Mode: DONT\_CARE ▼

Block type: AUTO ▼

**Size**

Data width: 32 ▼

Total memory size: 16384 bytes

☐ Minimize memory block usage (may impact fmax)

**Read latency**

Slave s1 Latency: 1 ▼

Slave s2 Latency: 1 ▼

**ROM/RAM Memory Protection**

Reset Request: Enabled ▼

**ECC Parameter**


Extend the data width to support ECC bits: Disabled ▼

**Memory initialization**

☐ Initialize memory content

☐ Enable non-default initialization file

Type the filename (e.g. my\_ram.hex) or select the hex file using the file browser button.

User created initialization file: onchip\_mem.hex 

☐ Enable In-System Memory Content Editor feature

Instance ID: NONE

**This memory is not initialized during device programming.**

Figure 16: On chip memory configuration panel

Click finish and you will now see a total 3 components in your Qsys system: clock, Nios II processor and on-chip memory.

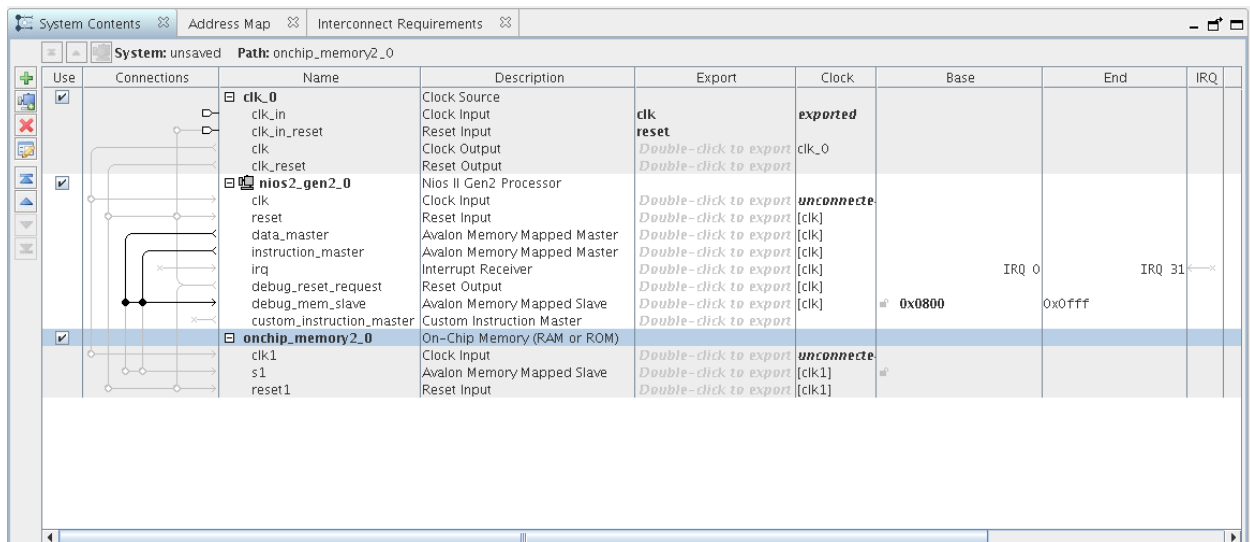


Figure 17: System contents with NIOSII and on chip memory

The next component you will add is the JTAG UART. Search for JTAG in the IP catalog, locate the JTAG UART and double click or add that component. Keep the default settings and click finish.

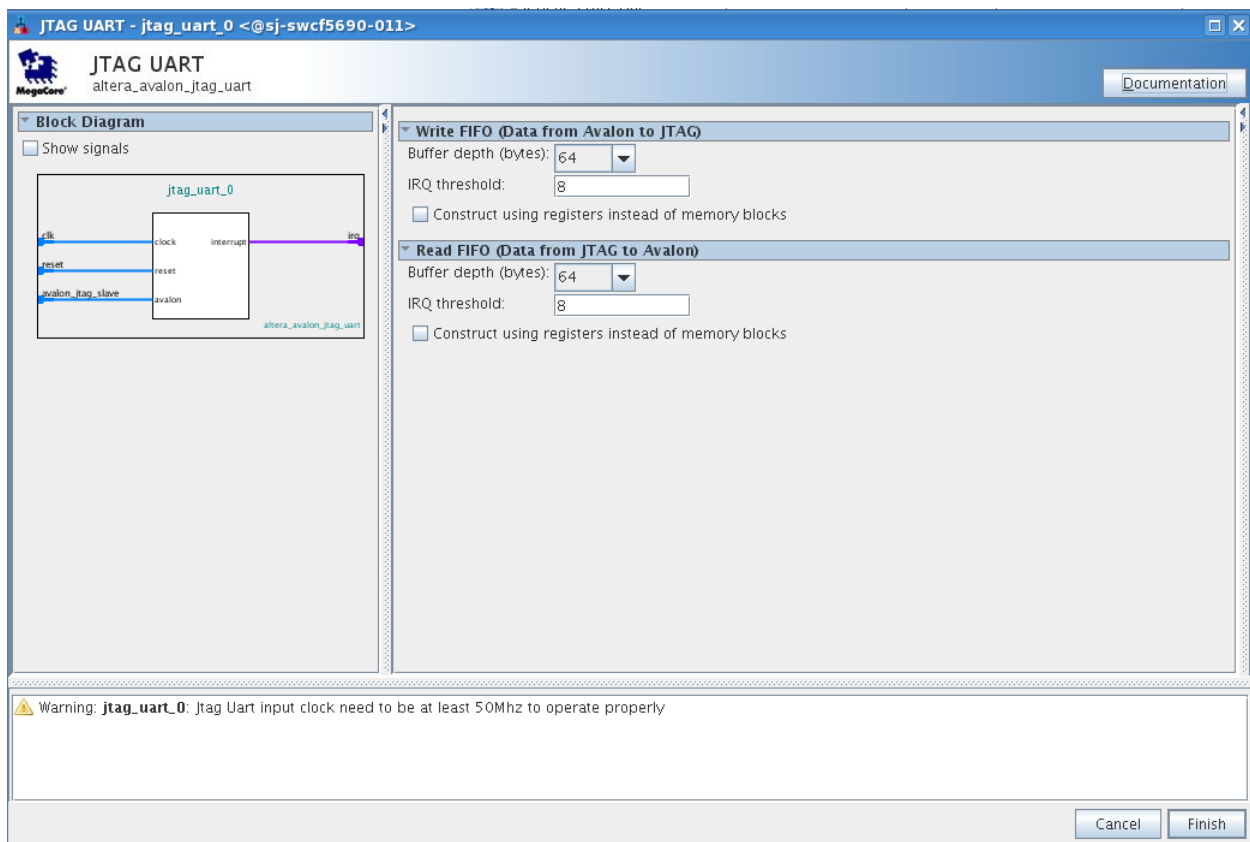


Figure 18: JTAG UART configuration panel



The next two components SWITCH and LED are actually configured instances of general purpose parallel IO components in the IP catalog. Search for parallel IO (PIO) and select this block. By using the PIO block for the switch and LED, you will be able to map the values of the SWITCH and LED to address space and your C code will read and write these components. For the switch block, you will set this up as a 1 bit input interface using the settings shown below.

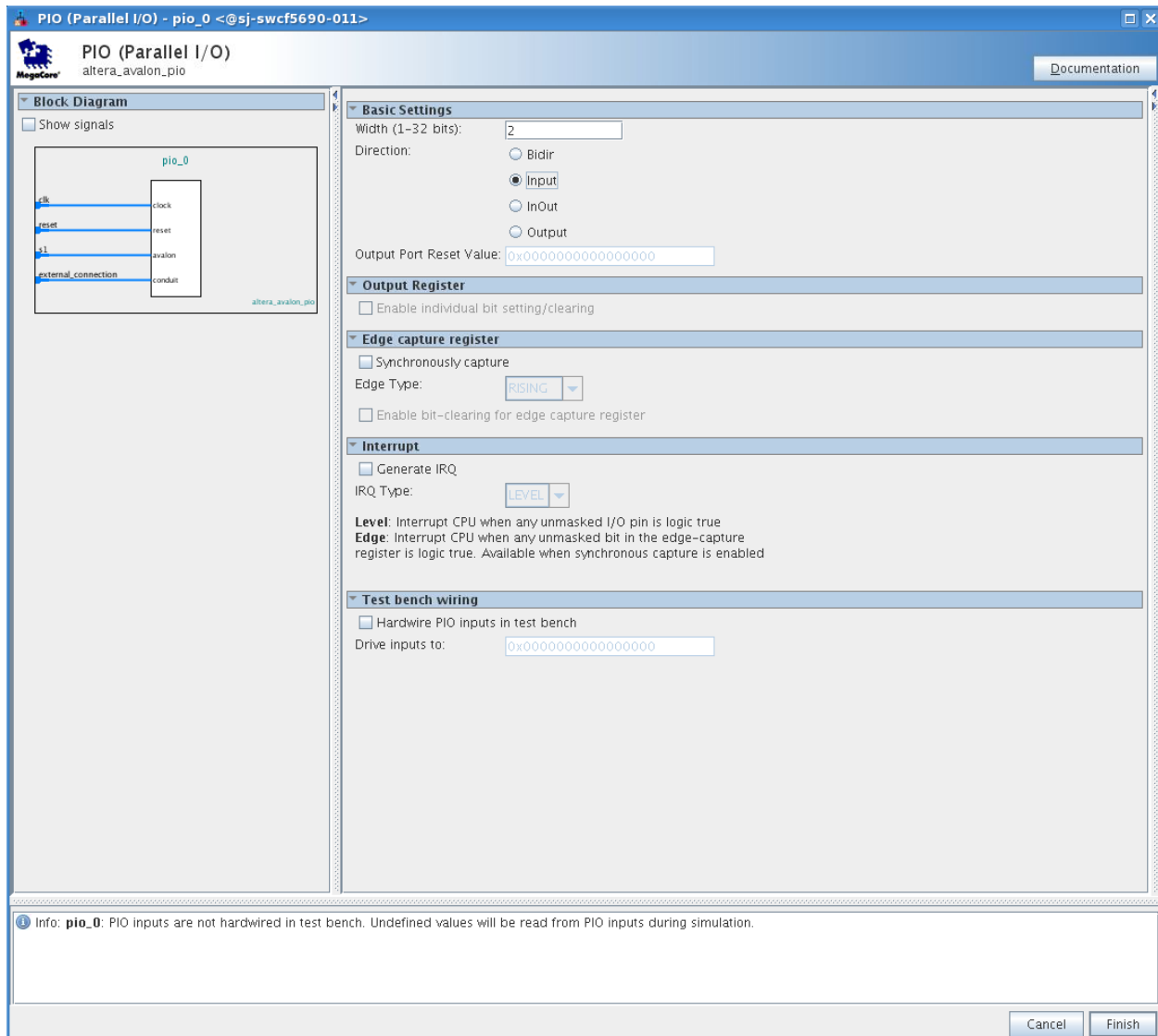


Figure 19: Parallel IO configuration panel

Next, you will add a second PIO block. Double click on the PIO component as you did for the SWITCH. This time you will configure this component as the LED which is a 1 bit output.

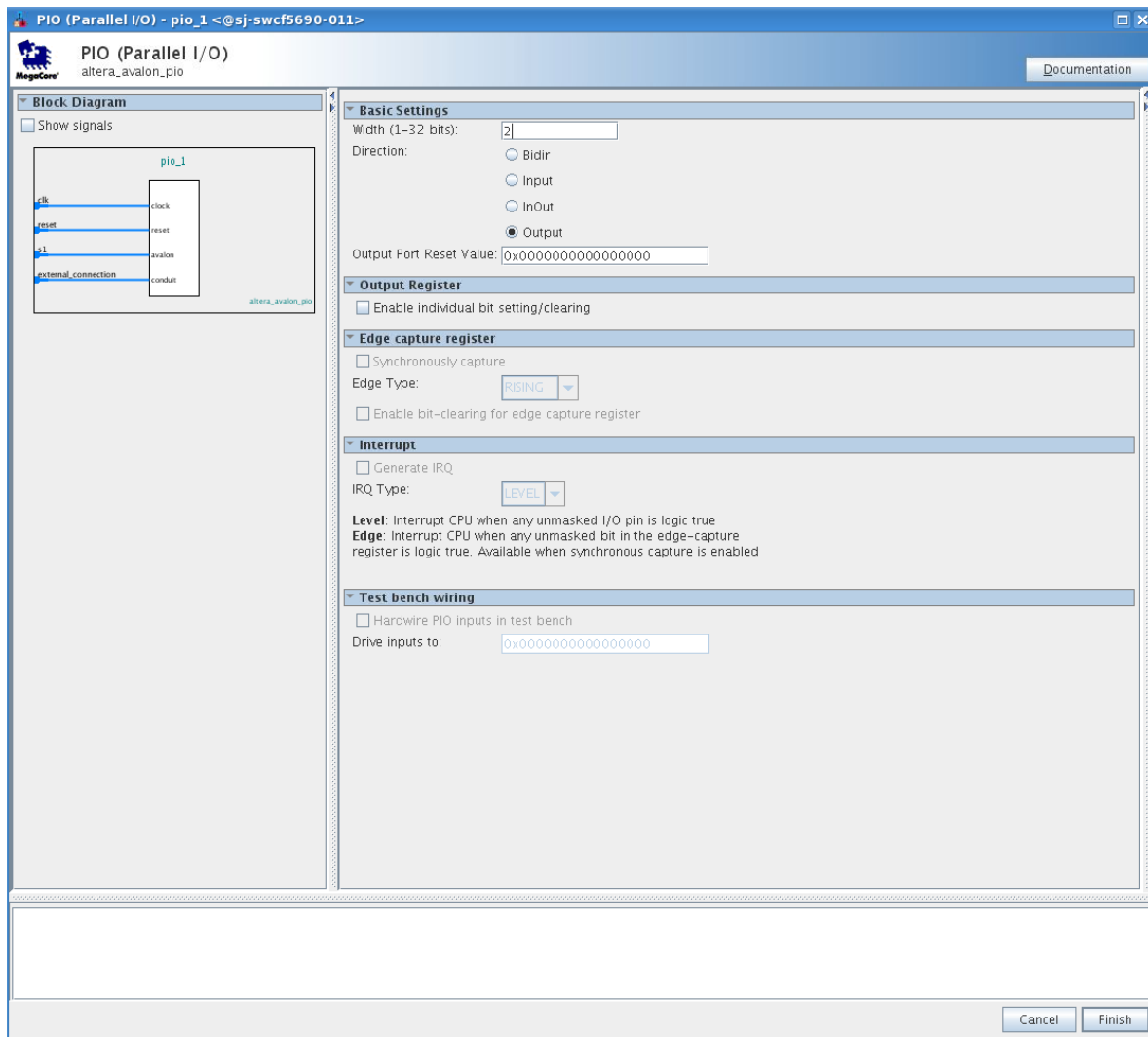


Figure 20: Parallel IO configuration panel for LED outputs

Click finish. You have completed adding the 6 components that make up your Qsys system. Next you will rename the components in the design with names that are easy to remember.

In the system contents tab, right click on the nios2\_gen\_2\_0 component, select rename and type in nios2e, similarly rename the rest of the components: onchip\_memory, uart, switch and led. This will make these components names easy to remember and reference in future steps.

System Contents    Address Map    Interconnect Requirements <span style="float: right;">1</span>										
System: unsaved    Path: onchip_memory										
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode N
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source	<b>clk</b>	<b>exported</b>					
		clk_in	Clock Input	Double-click to export	clk_0					
		clk_in_reset	Reset Input	Double-click to export						
		clk	Clock Output	Double-click to export						
		clk_reset	Reset Output	Double-click to export						
<input checked="" type="checkbox"/>		<b>nios2e</b>	Nios II Gen2 Processor	Double-click to export	<b>unconnecte</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]					
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]					
		irq	Interrupt Receiver	Double-click to export	[clk]					
		debug_reset_request	Reset Output	Double-click to export	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0800	0x0fff	IRQ 0	IRQ 31	
		custom_instruction_master	Custom Instruction Master	Double-click to export	[clk]					
<input checked="" type="checkbox"/>		<b>onchip_memory</b>	On-Chip Memory (RAM or ROM)	Double-click to export	<b>unconnecte</b>					
		clk1	Clock Input	Double-click to export	[clk1]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
		reset1	Reset Input	Double-click to export	[clk1]					
<input checked="" type="checkbox"/>		<b>uart</b>	JTAG UART	Double-click to export	<b>unconnecte</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		irq	Interrupt Sender	Double-click to export	[clk]					
<input checked="" type="checkbox"/>		<b>switch</b>	PIO (Parallel I/O)	Double-click to export	<b>unconnecte</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		external_connection	Conduit	Double-click to export						
<input checked="" type="checkbox"/>		<b>led</b>	PIO (Parallel I/O)	Double-click to export	<b>unconnecte</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		external_connection	Conduit	Double-click to export						

Figure 21: System Content connections starting panel

The next step consists of making the appropriate connections between the components within Qsys.

Click on the clk net coming out of clk\_0. When first selected, it will be gray color. Make connections by clicking on on the small open circles on the lines that intersecting with the 5 other components.

You should see something similar to Figure 22.

System Contents    Address Map    Interconnect Requirements <span style="float: right;">1</span>										
System: unsaved    Path: led.clk										
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode N
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source	<b>clk</b>	<b>exported</b>					
		clk_in	Clock Input	Double-click to export	clk_0					
		clk_in_reset	Reset Input	Double-click to export						
		clk	Clock Output	Double-click to export						
		clk_reset	Reset Output	Double-click to export						
<input checked="" type="checkbox"/>		<b>nios2e</b>	Nios II Gen2 Processor	Double-click to export	<b>clk_0</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]					
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]					
		irq	Interrupt Receiver	Double-click to export	[clk]					
		debug_reset_request	Reset Output	Double-click to export	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0800	0x0fff	IRQ 0	IRQ 31	
		custom_instruction_master	Custom Instruction Master	Double-click to export	[clk]					
<input checked="" type="checkbox"/>		<b>onchip_memory</b>	On-Chip Memory (RAM or ROM)	Double-click to export	<b>clk_0</b>					
		clk1	Clock Input	Double-click to export	[clk1]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
		reset1	Reset Input	Double-click to export	[clk1]					
<input checked="" type="checkbox"/>		<b>uart</b>	JTAG UART	Double-click to export	<b>clk_0</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		irq	Interrupt Sender	Double-click to export	[clk]					
<input checked="" type="checkbox"/>		<b>switch</b>	PIO (Parallel I/O)	Double-click to export	<b>clk_0</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		external_connection	Conduit	Double-click to export						
<input checked="" type="checkbox"/>		<b>led</b>	PIO (Parallel I/O)	Double-click to export	<b>clk_0</b>					
		clk	Clock Input	Double-click to export	[clk]					
		reset	Reset Input	Double-click to export	[clk]					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		external_connection	Conduit	Double-click to export						

Figure 22: System contents after clock connection

Perform the same operation to connect the clk\_reset to the resets on the other components.

Next, connect the nios2e data master to the slaves.

Make the connections between the Nios2e data master and the s1 connection of the onchip memory, avalon\_jtag\_slave on the uart, s1 port on the switch and s1 port of the led component as shown below. Instruction master -> debug\_mem\_slave

System Contents    Address Map    Interconnect Requirements									
System: unsaved    Path: led.s1									
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Ta
<input checked="" type="checkbox"/>		<div>clk_0</div> <div>clk_in</div> <div>clk_in_reset</div> <div>clk</div> <div>clk_reset</div>	Clock Source Clock Input Reset Input Clock Output Reset Output	clk reset <i>Double-click to export</i> <i>Double-click to export</i>	exported clk_0				
<input checked="" type="checkbox"/>		<div>nios2e</div> <div>clk</div> <div>reset</div> <div>data_master</div> <div>instruction_master</div> <div>irq</div> <div>debug_reset_request</div> <div>debug_mem_slave</div> <div>custom_instruction_master</div>	Nios II Gen2 Processor Clock Input Reset Input Avalon Memory Mapped Master Avalon Memory Mapped Master Interrupt Receiver Reset Output Avalon Memory Mapped Slave Custom Instruction Master	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk] [clk] [clk] [clk]		IRQ 0	IRQ 31	
<input checked="" type="checkbox"/>		<div>onchip_memory</div> <div>clk1</div> <div>s1</div> <div>reset1</div>	On-Chip Memory (RAM or ROM) Clock Input Avalon Memory Mapped Slave Reset Input	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk1] [clk1]	# 0x0800 # 0x0000	0x0fff 0x3fff		
<input checked="" type="checkbox"/>		<div>uart</div> <div>clk</div> <div>reset</div> <div>avalon_jtag_slave</div> <div>irq</div>	JTAG UART Clock Input Reset Input Avalon Memory Mapped Slave Interrupt Sender	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk]	# 0x0000	0x0007		
<input checked="" type="checkbox"/>		<div>switch</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000	0x000f		
<input checked="" type="checkbox"/>		<div>led</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x0000	0x000f		

Figure 23: System contents after data master/slave connection

The instruction master signal from the nios2e component does not need to be connected to each slave component as it only needs access to memory that contains the software executable. Make the connection between the nios2e instruction master and the onchip\_memory s1.

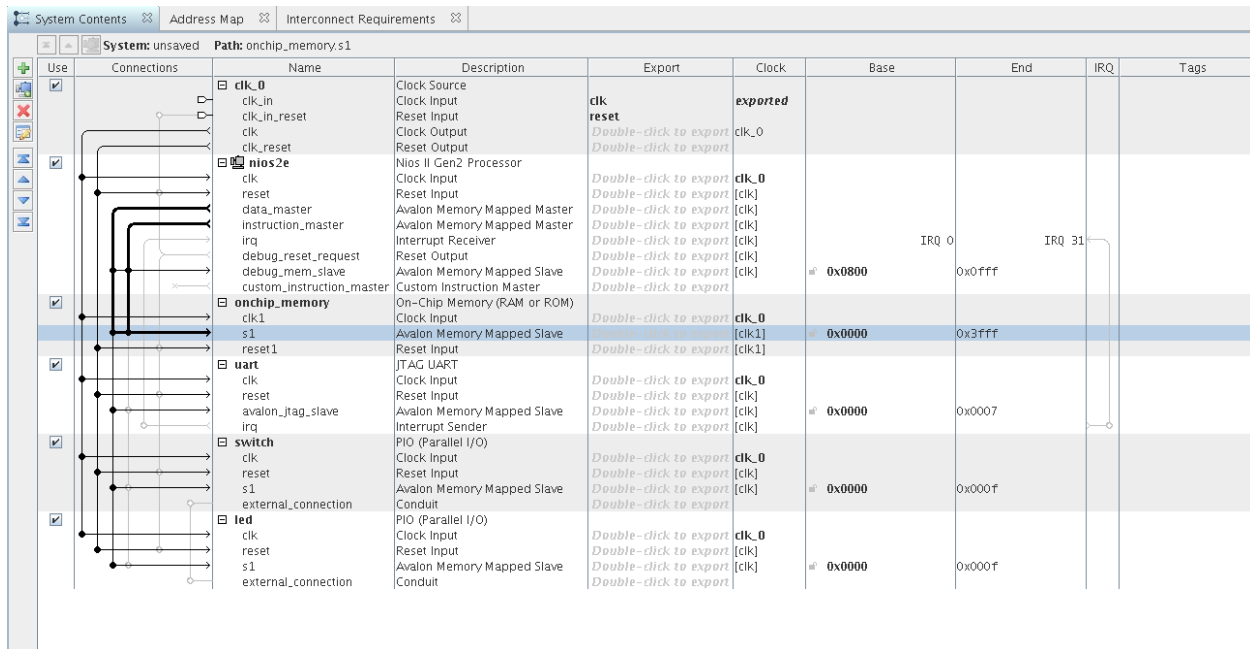


Figure 24: System contents after instruction master/slave connections

The next connections to make are the processor interrupt request (IRQ) signals. The UART can drive interrupts and hence needs to be wired to the nios2e processor interrupt lines. Make this connection as shown in Figure 25. We will use the default setting for the IRQ number.

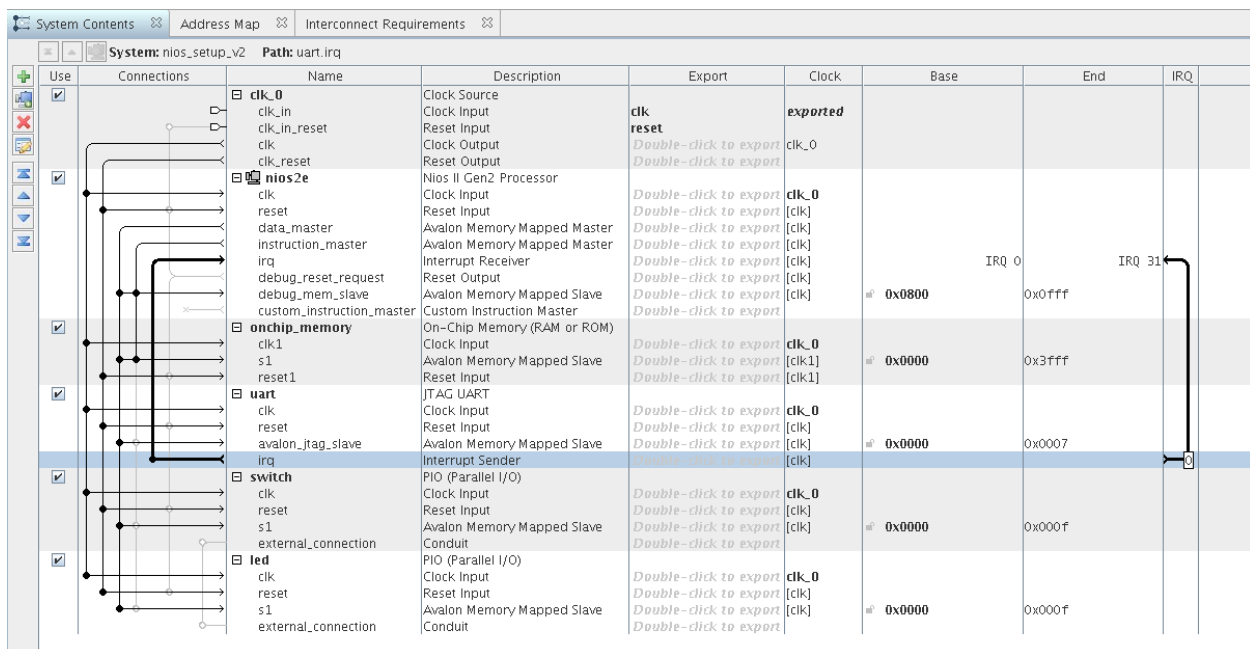


Figure 25: System contents after interrupt connections

You have now completed the internal connections for this Nios II processor based system. The next step is to make the external connections that connect the Qsys based system to the next higher level in the hierarchy of your FPGA design, or to FPGA device pins that connect to the

PCB. Double click on the switch and led conduit items under the export column circled in Figure 26. This will bring these ports out of the Qsys component to connect to the top level design.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	exported			
<input checked="" type="checkbox"/>		clk_in	Clock Input	reset				
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input					
<input checked="" type="checkbox"/>		clk	Clock Output					
<input checked="" type="checkbox"/>		clk_reset	Reset Output					
<input checked="" type="checkbox"/>		nios2e	Nios II Gen2 Processor					
<input checked="" type="checkbox"/>		clk	Clock Input					
<input checked="" type="checkbox"/>		reset	Reset Input					
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped Master					
<input checked="" type="checkbox"/>		instruction_master	Avalon Memory Mapped Master					
<input checked="" type="checkbox"/>		irq	Interrupt Receiver					
<input checked="" type="checkbox"/>		debug_reset_request	Reset Output					
<input checked="" type="checkbox"/>		debug_mem_slave	Avalon Memory Mapped Slave					
<input checked="" type="checkbox"/>		custom_instruction_master	Custom Instruction Master					
<input checked="" type="checkbox"/>		onchip_memory	On-Chip Memory (RAM or ROM)					
<input checked="" type="checkbox"/>		clk1	Clock Input					
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave					
<input checked="" type="checkbox"/>		reset1	Reset Input					
<input checked="" type="checkbox"/>		uart	JTAG UART					
<input checked="" type="checkbox"/>		clk	Clock Input					
<input checked="" type="checkbox"/>		reset	Reset Input					
<input checked="" type="checkbox"/>		avalon_jtag_slave	Avalon Memory Mapped Slave					
<input checked="" type="checkbox"/>		irq	Interrupt Sender					
<input checked="" type="checkbox"/>		switch	PIO (Parallel I/O)					
<input checked="" type="checkbox"/>		clk	Clock Input					
<input checked="" type="checkbox"/>		reset	Reset Input					
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave					
<input checked="" type="checkbox"/>		external_connection	Conduit					
<input checked="" type="checkbox"/>		led	PIO (Parallel I/O)					
<input checked="" type="checkbox"/>		clk	Clock Input					
<input checked="" type="checkbox"/>		reset	Reset Input					
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave					
<input checked="" type="checkbox"/>		external_connection	Conduit					

Figure 26: System contents after exporting PIO switch and LED

Next you will need to generate the base Addresses for your Qsys system. This is achieved by using the command System → Assign Base Addresses.

Save your Qsys system by using File → Save As and pick a name for the Qsys system that you will remember. Note that the lab figures call it nios\_setup\_v2 so to avoid confusion you should name your .qsys file the same. The information is saved in what is called a .qsys file. Although you are not entirely finished, it's good practice to save edits along the way.

You should see 2 error messages in the Message Console of Qsys. They are shown in Figure 27.

Type	Path	Message
2 Errors		
	nios_setup_v2.nios2e	Reset slave is not specified. Please select the reset slave
	nios_setup_v2.nios2e	Exception slave is not specified. Please select the exception slave
1 Info Message		
	nios_setup_v2.switch	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 27: Error message prior to assign memory location to execute from

These error messages have to do with the fact that nios2e processor doesn't know where the software code that handles resets and exceptions is located. This is fairly straightforward to fix.

Double click on the nios2e component and set the reset vector memory and exception vector memory both to onchip\_memory.s1. This will set the system to execute from onchip memory at these respective locations upon reset or interrupt. The 2 errors that were shown in Figure 27 should now be resolved.

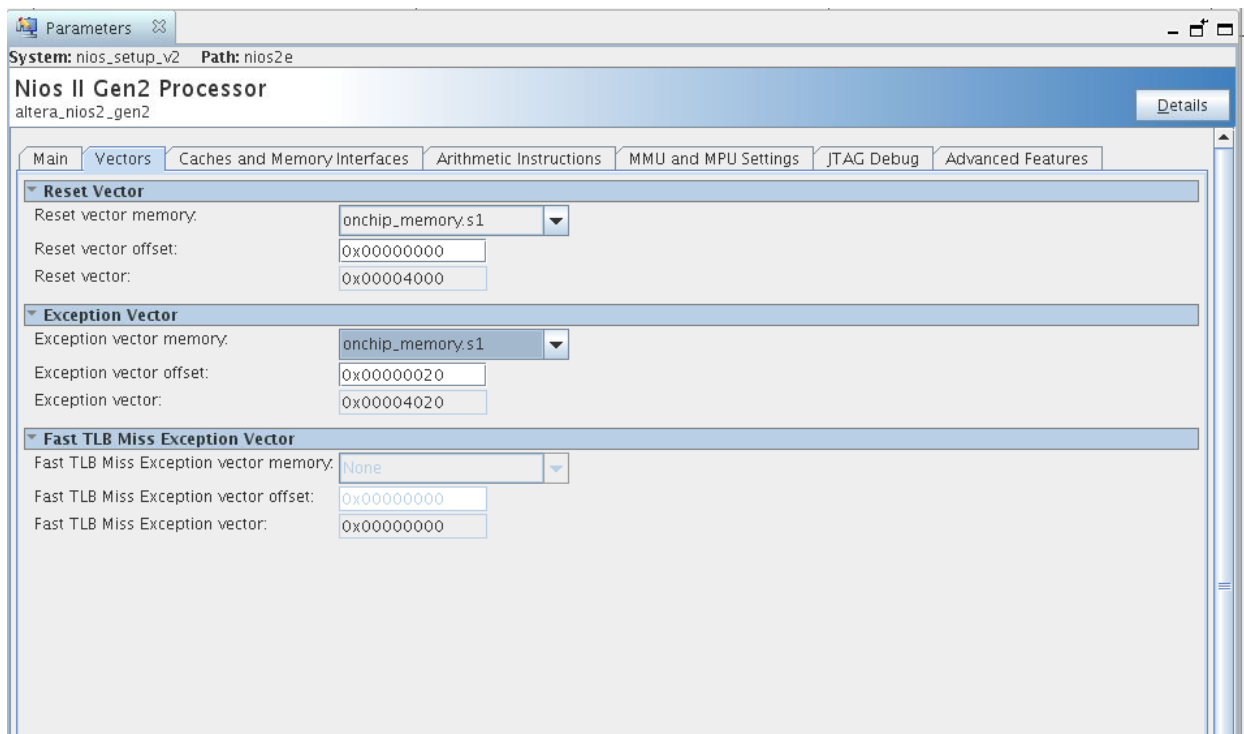


Figure 28: Assign vectors in the NIOS2E panel

Save your design once again. Note that by saving, you still have not generated the files that you need for Quartus II compilation or with the Eclipse SBT. The step to complete this is to click on the button on the lower right of Qsys.

Click on the button 'Generate HDL'.  
Click Generate on the panel that appears.

Congratulations, this completes the Qsys section of the lab.

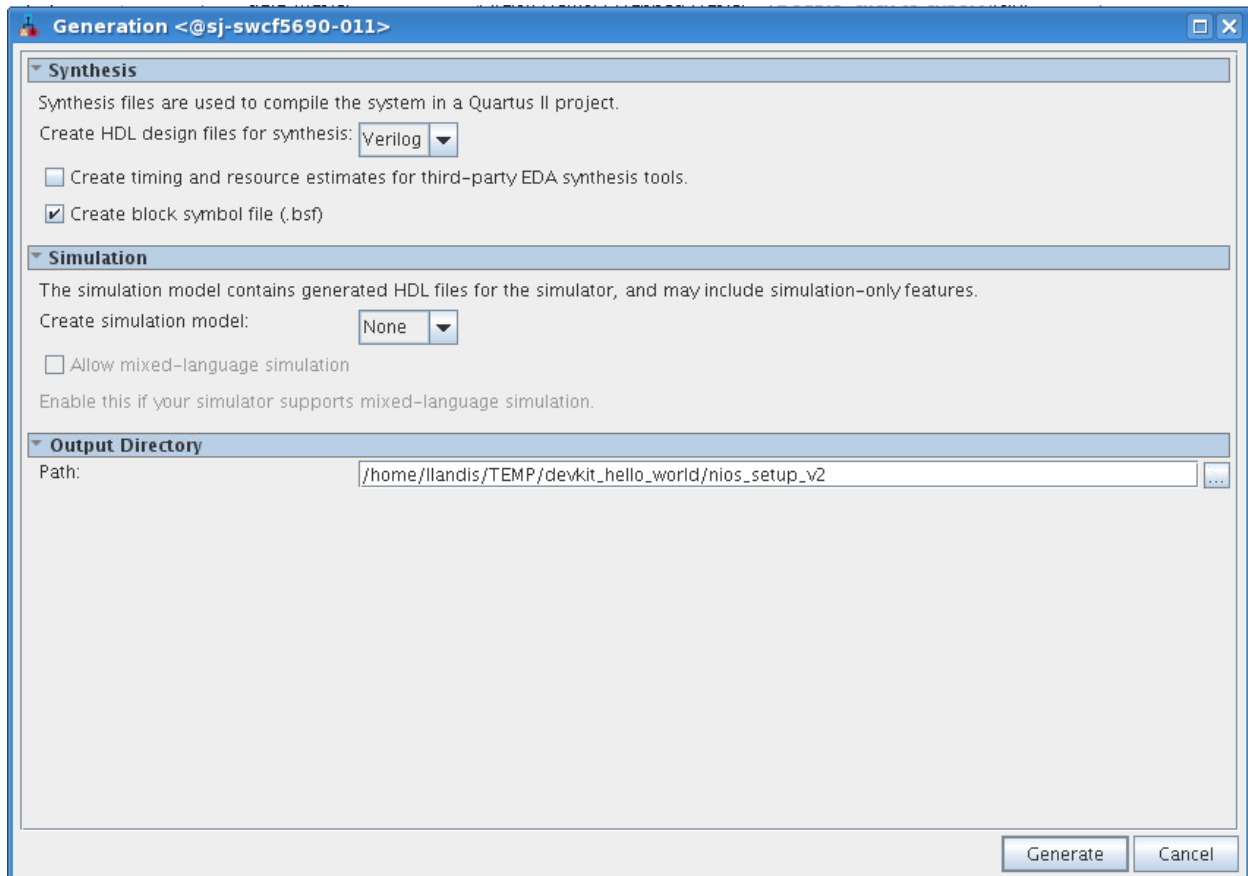


Figure 29: Generating the Qsys system HDL files

## Building the top level design

The next step will take a little bit of knowledge in Verilog. Should you want to use a schematic capture graphical editor, jump to the **Error! Reference source not found.** If you are familiar with VHDL, you can make the same connections in VHDL, but you will have to change the design to VHDL on your own. For ease of following along the lab document, we recommend continuing the lab in Verilog. During the early steps using the project wizard, you loaded the baseline design, and have a baseline.v preloaded in the Quartus project. We will take a look at this starting point baseline.v file and strip out the unnecessary signals, while only leaving the signals that are needed to run the Hello World design. It is important to note that each development kit used has its own names for clocks, switches, pins and so you will need to use the right names according to the development kit naming convention.

Quartus should be open, bring that to the front of your screen. Make sure the hierarchy tab is highlighted and double click the baseline design. Note that for this design there is a clock, reset, push button inputs, LED outputs, and a JTAG UART. The JTAG UART pins are hard wired into the FPGA so you don't need to add them in your Verilog source file. The 4 pins: TCLK, TDI, TMS and TDO that constitute a 4 wire JTAG interface are at a fixed location in your FPGA and they don't need to be added to your Verilog source file. Only pins that are synthesized from your RTL source code need to be specified. The baseline design includes all non hard-wired device pins and you will need to delete extra pins and include the following pins in the port list:



CLK\_50\_MAX10, CPU\_RESETh, USER\_PB, USER\_LED. Delete all other pins from the port list. The original baseline.v is shown in Figure 31. Make the changes including changing the module from baseline to hello\_world and save the file as hello\_world.v.

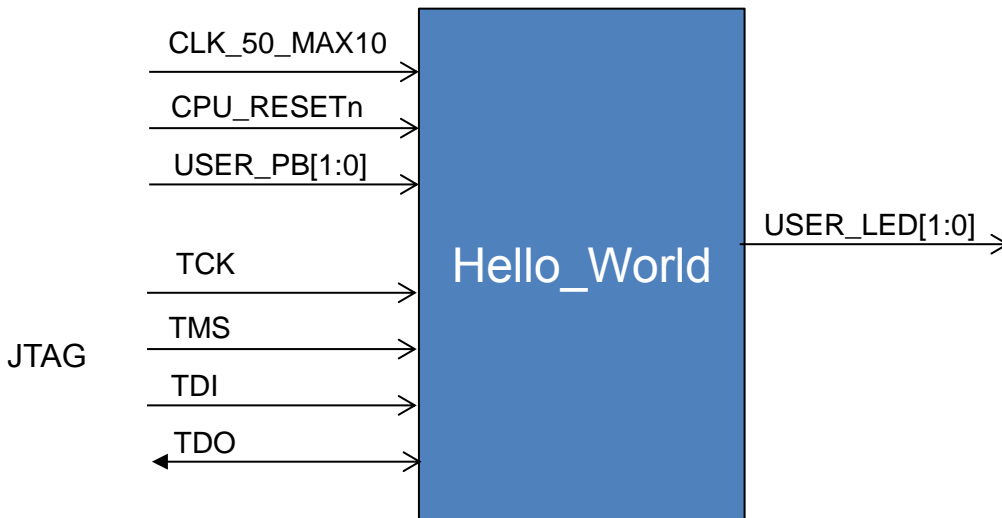


Figure 30: Block diagram of hello\_world design for the MAX 10 FPGA Development Kit.

**Important Note:** The pin names throughout this section reflect the names for the MAX 10 FPGA Development Kit. Refer to the table below for the naming convention for the other kits supported by this lab.

MAX 10 FPGA Development Kit	BeMicro MAX 10	Cyclone V E FPGA Development Kit	DECA MAX 10 Development Kit
CLK50_50_MAX10	CLK50_50_MAX10	CLKIN_50_FPGA_TOP	MAX10_CLK1_50
CPU_RESETh	CPU_RESETh	RESET_EXPN	RESET_EXPN
USER_PB	USER_PB	USER_PB	KEY
USER_LED	USER_LED	USER_LED	LED

```

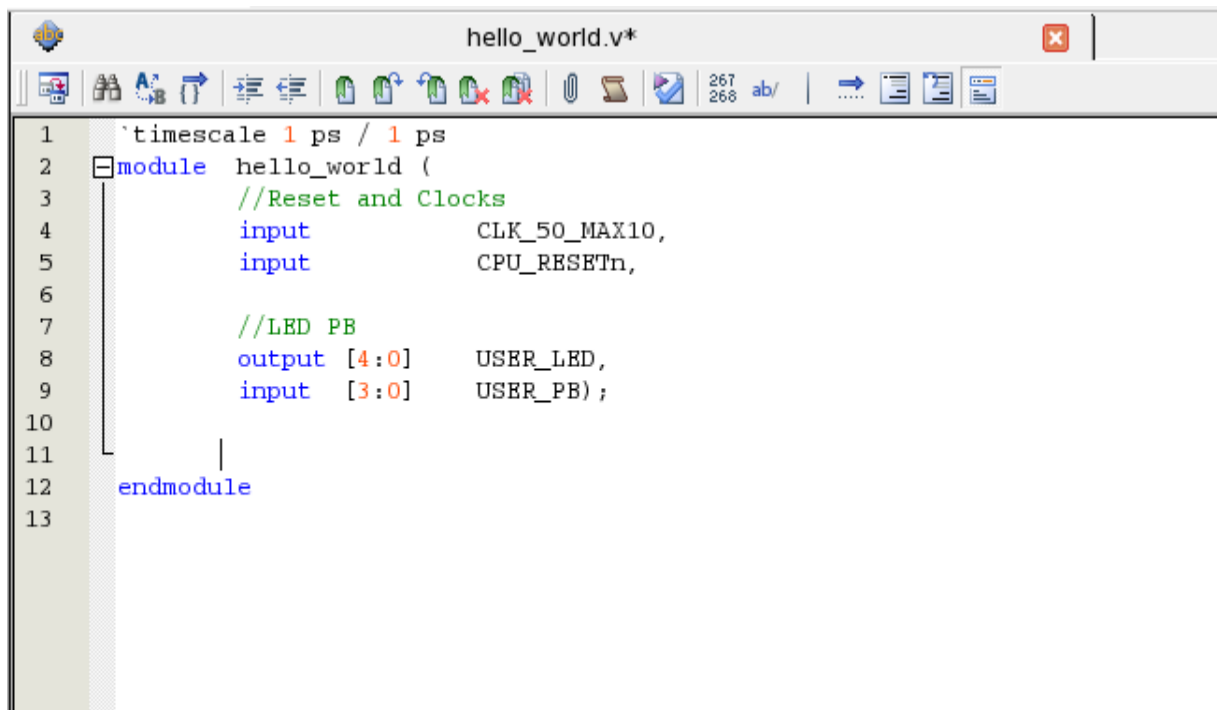
`timescale 1 ps / 1 ps
module baseline (
    //Reset and Clocks
    //      input          clk_ddr3_100_p,
    input          CLK_50_MAX10,
    input          CLK_25_MAX10,
    input          CLK_LVDS_125_p,
    input          CLK_10_ADC,
    input          CPU_RESETh,

    //LED PB DIPSW
    output [4:0]    USER_LED,
    input  [3:0]    USER_PB,
    input  [4:0]    USER_DIPSW,

    //USB
    input          USB_RESETh,
    input          USB_WRn,
    input          USB_RDn,
    input          USB_OEn,
    inout  [1:0]    USB_ADDR,
    inout  [7:0]    USB_DATA,
    output         USB_FULL,
    output         USB_EMPTY,
    input          USB_SCL,
    input          USB_SDA
)

```

Figure 31: Original baseline.v design with MAX 10 FPGA Development Kit signal names



```

1  `timescale 1 ps / 1 ps
2  module hello_world (
3      //Reset and Clocks
4      input          CLK_50_MAX10,
5      input          CPU_RESETh,
6
7      //LED PB
8      output [4:0]    USER_LED,
9      input  [3:0]    USER_PB);
10
11  endmodule
12
13

```

Figure 32: Edited baseline MAX 10 Development Kit design with pins removed. Note save as: hello\_world.v

Next we need to check that the `hello_world.v` file is included in your project. Note that it should be the only file in your project so far. Go to Project → Add/Remove Files in Project. Confirm that `hello_world.v` is listed.

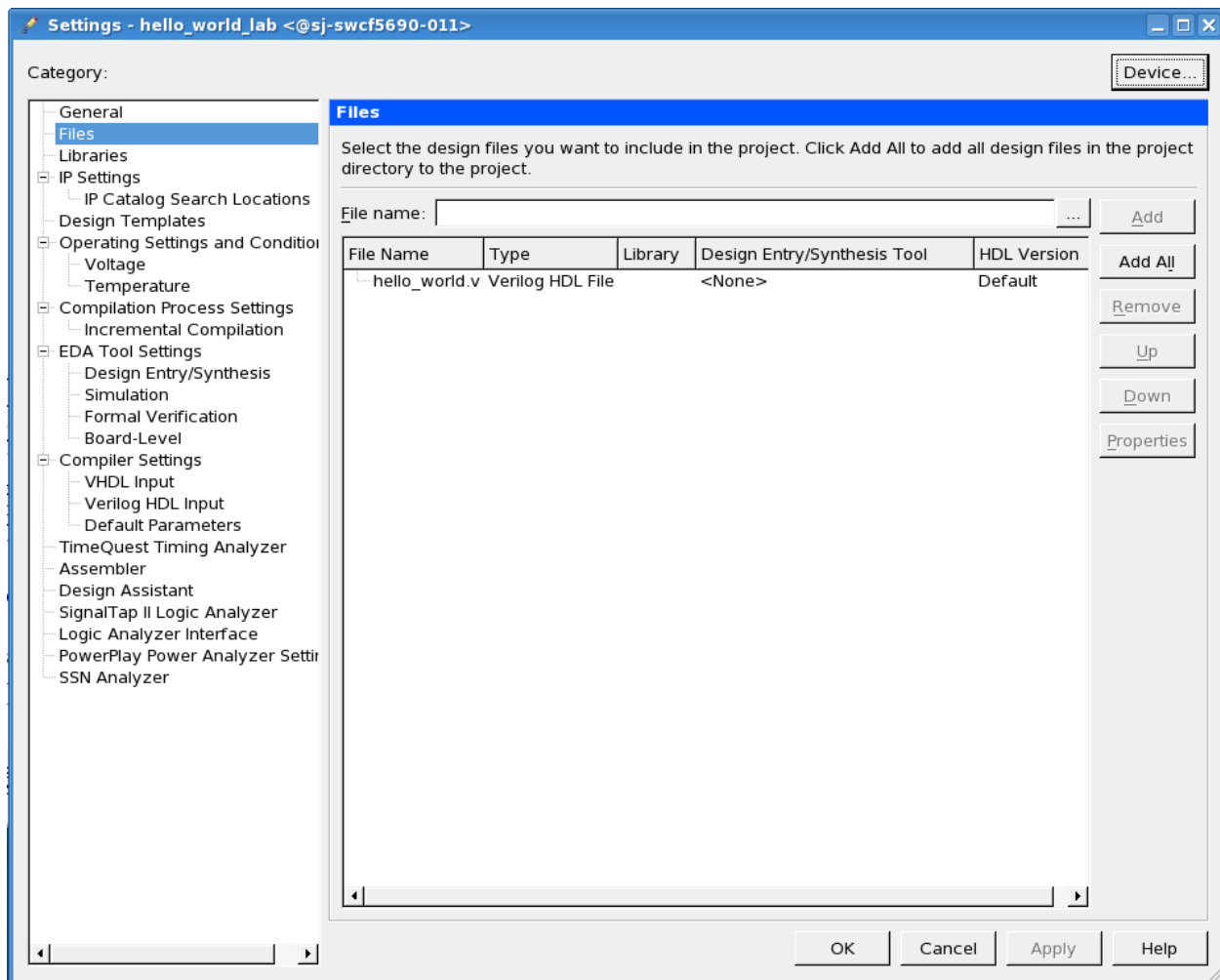


Figure 33: Add/Remove Files pane

Next you need to make the top level entity `hello_world` since its currently set at baseline. In the same window upper left corner, click on General. Change baseline to `hello_world`. You can also change by right clicking on the `hello_world.v` and set as top level entity.

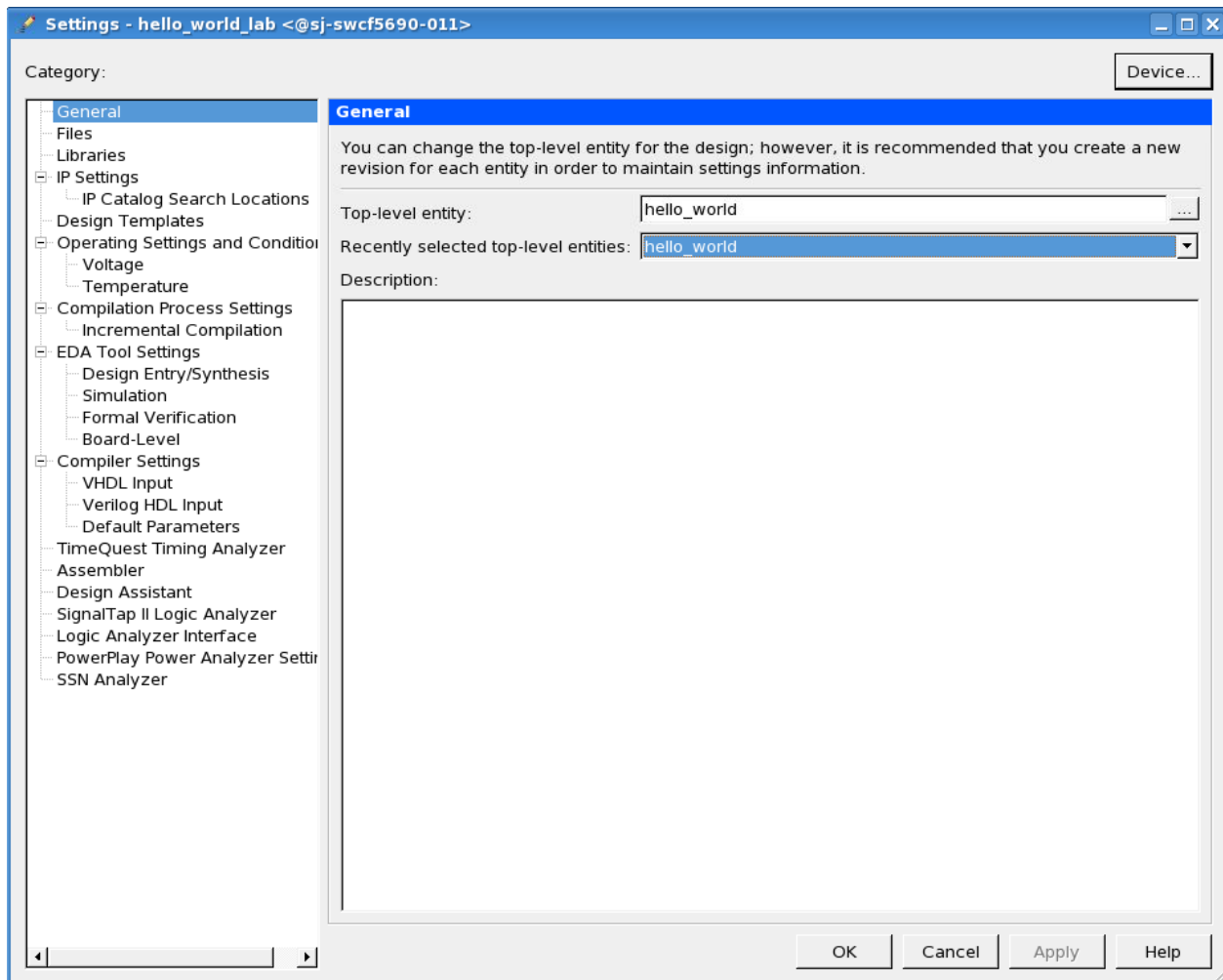


Figure 34: Settings pane

Click OK when complete. Now it is a good idea to make sure your Verilog is syntactically correct. Return to the main Quartus window and select the Tasks pane. Double-Click on the Play (right triangle) for analysis/synthesis. You will get warnings but you should get no errors. If you do get an error, it's likely syntax (eg missing semicolon). Make changes, save, and continue to run analysis/synthesis until the Verilog runs error/free (ignore dangling pin warning for now).

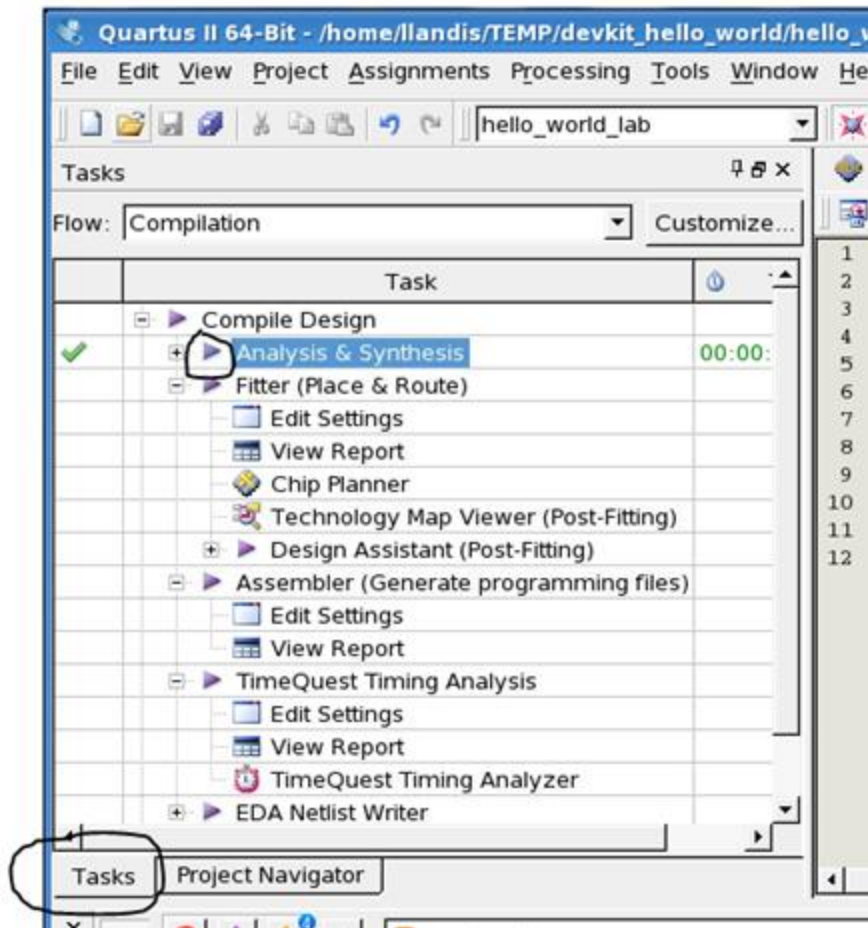


Figure 35: Task pane

The baseline design that you uploaded in the prior steps contain all of the pin settings needed so that the pin locations are consistent with how the MAX 10 device is connected for the PCB on the MAX 10 Development Kit. You can inspect the pin setting locations to understand where they come from. Launch Assignments → Assignment Editor. You will see a list of pins in spreadsheet form that contain pin (package ball to be specific) locations, IO standard and current strength settings. Note that you are not using all the pins in the design, but this is ok – Quartus will ignore pin assignments that are not referenced in your design.

	tatu	From	To	Assignment Name	Value	Enabled	Entity
1	✓		CLK_D...100_n	Location	PIN_N15	Yes	
2	✓		CLK_D...100_p	Location	PIN_N14	Yes	
3	✓		CLK_...AX10	Location	PIN_M9	Yes	
4	✓		CLK_...AX10	Location	PIN_M8	Yes	
5	✓		CLK_10_ADC	Location	PIN_N5	Yes	
6	✓		CLK_L...125_n	Location	PIN_R11	Yes	
7	✓		CLK_L...125_p	Location	PIN_P11	Yes	
8	✓		CPU_RESETn	Location	PIN_D9	Yes	
9	✓		DDR3_A[0]	Location	PIN_V20	Yes	
10	✓		DDR3_A[3]	Location	PIN_U20	Yes	
11	✓		DDR3_A[5]	Location	PIN_F19	Yes	
12	✓		DDR3_A[6]	Location	PIN_E21	Yes	
13	✓		DDR3_A[8]	Location	PIN_D22	Yes	
14	✓		DDR3_A[9]	Location	PIN_E22	Yes	
15	✓		DDR3_A[10]	Location	PIN_Y20	Yes	
16	✓		DDR3_A[11]	Location	PIN_E20	Yes	
17	✓		DDR3_A[12]	Location	PIN_J14	Yes	
18	✓		DDR3_A[13]	Location	PIN_C22	Yes	
19	✓		DDR3_BA[0]	Location	PIN_V22	Yes	
20	✓		DDR3_BA[1]	Location	PIN_N18	Yes	
21	✓		DDR3_BA[2]	Location	PIN_W22	Yes	

Figure 36: Assignment Editor

Note: Some of the pin names in the list above might differ from the names assigned when you loaded the baseline design. In a future section, you will be instructed to change some of the pin names to match what is being used in this specific design.

## Adding the Nios II system into your design

Now that you have the hello\_world entity completed and syntactically correct, you will need to add the Nios II Qsys system into your design. Qsys makes this task quite convenient. Go to File → Open and navigate to the name of the Qsys project you created (the one shown in the lab is called nios\_setup\_v2). You should see a file called nios\_setup\_v2\_inst.v . Open this file and you see how to instantiate (fancy word meaning placing this component in your design) the Qsys system. The contents of this file are shown below:

```

1  nios2e u0 (
2      .clk_clk                (<connected-to-clk_clk>),
3      .reset_reset_n         (<connected-to-reset_reset_n>),
4      .pb_external_connection_export (<connected-to-pb_external_connection_export>),
5      .led_external_connection_export (<connected-to-led_external_connection_export>)
6  );
7

```

You will need to connect the IO ports to the nios\_setup\_v2. Copy the entire contents of the nios\_setup\_v2 file by highlighting and copy (ctrl-c), followed by inserting into the Verilog file hello\_world.v and pasting (ctrl-v). Next we will connect the push button switches to the LEDs in two different ways to demonstrate how the connection can be made through the FPGA fabric, and in the software that we use that the Nios II executes. To simplify knowing which push button is connected through hardware and which one through software, we will invert the hardware connection so that activating push buttons 0 turn *off* LEDs through a hardware connection, while activating push buttons 1 turn *on* the respective LEDs. Take one push buttons ([0]) and connect to LEDs[0] with an inverted assignment (see line 11 in Figure 37). The LEDs [1] will be connected non-inverted in software by connection to the Qsys system. Note that the USER\_LED for the MAX 10 Development Kit (note that the number varies by kit) is defined as 5 bits wide, [4:0], but you have only used 2 LEDs in total: bit 0 is connected through the FPGA fabric, and bit 1 is connected through application software. USER\_LED [4:2] are unconnected and will give a dangling wire warning message when you compile, this is ok. Click the save icon or File → Save.


```

1  `timescale 1 ps / 1 ps
2  module hello_world (
3      //Reset and Clocks
4      input      CLK_50_MAX10,
5      input      CPU_RESEtN,
6
7      //LED PB DIPSW
8      output [4:0] USER_LED,
9      input  [3:0]  USER_PB,
10
11      assign USER_LED[1:0] = ~USER_PB[1:0];
12
13      nios_setup_v2 u0 (
14          .clk_clk                (CLK_50_MAX10),
15          .led_external_connection_export (USER_LED[3:2]), // led_external_connection.export
16          .reset_reset_n         (CPU_RESEtN), // reset.reset
17          .switch_external_connection_export (USER_PB[3:2]) // switch_external_connection.export
18      );
19
20
21  endmodule
22

```

Figure 37: hello\_world.v after making connections to nios system and adding led to push button assignment

You now have completed the creation of the Nios II system using Qsys, instantiating this component into the top level design, and making connections from led to push buttons for testing in your Verilog file called `hello_world.v`. You now add the Nios II system into your project using the Project → Add/Remove Files in Project command. Instead of adding individual Qsys generated Verilog files and settings into the project, you add the NIOS qip file which is located under: `nios_setup_v2/synthesis/nios_setup_v2.qip`. The qip file contains pointers to the location of all the generated source files generated from Qsys and necessary settings required to

compile. You can open this file in a text editor to see its content. Navigate using the  button and select the file. Hit Add followed by OK.

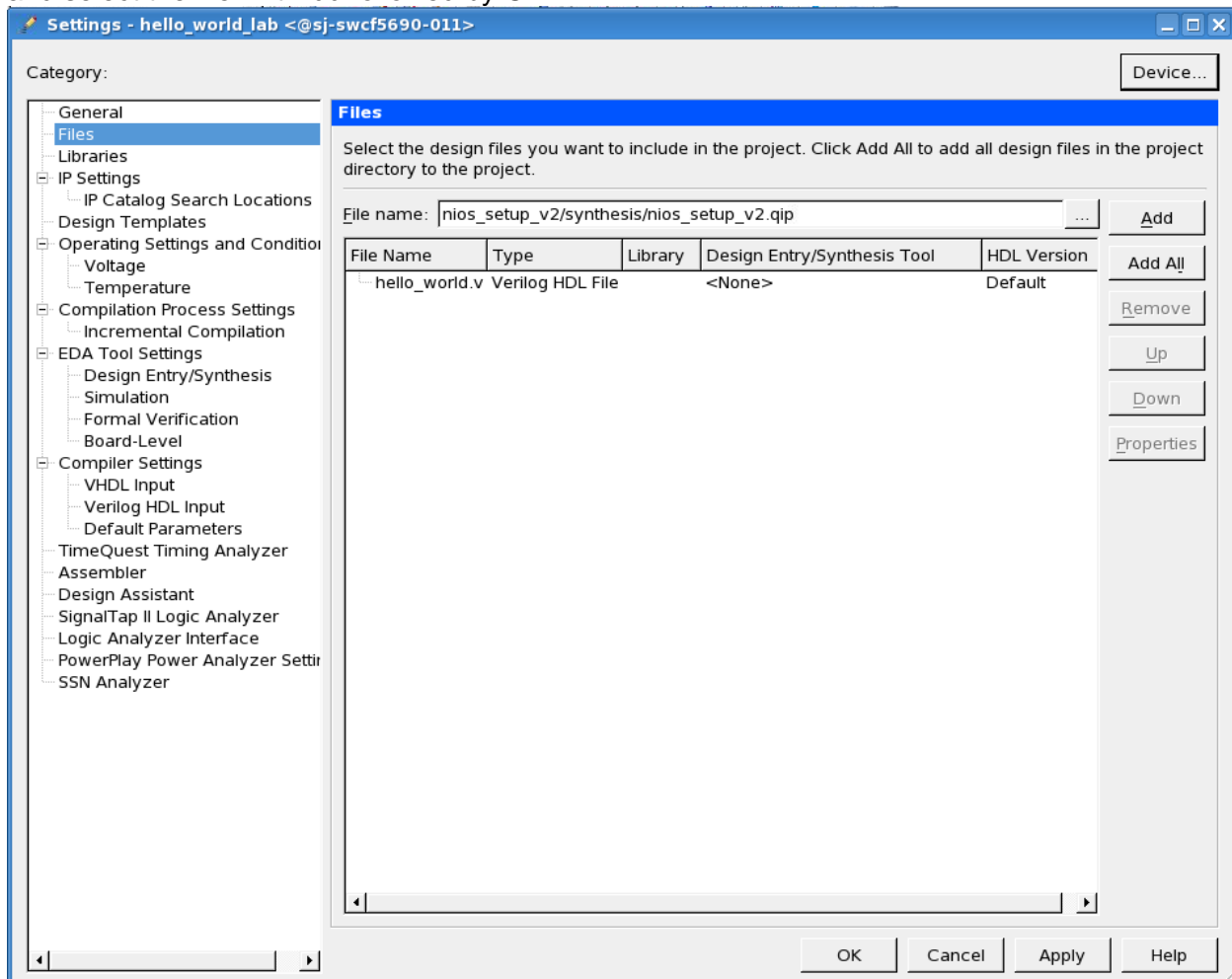


Figure 38: Add/Remove Files from Project - .qip file





Now you can compile your design which will run analysis & synthesis, fitter (place and route in FPGA terminology), Assembler (generate programming image) and TimeQuest (the static timing analyzer). This can be achieved by clicking on the play button as shown in Figure 39.





Figure 39: Compilation button

Note that some warnings and information messages come up in the bottom window.

You can filter by message level. The errors are filtered with the  button, critical warnings with the  button, warnings with the  button and informational messages with the  button. You cannot proceed if you have errors. In this case there are only critical and standard warnings, primarily because we did not add timing constraints to this project. Due to the simplicity of this design and low frequency, it's okay to start without timing constraints. Consult other Altera online training courses for instructions on how to add timing constraints to your design.

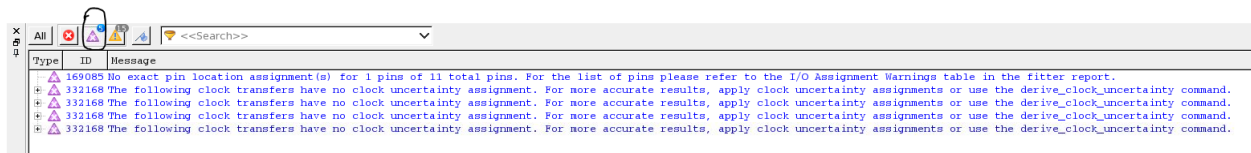


Figure 40: Filter for critical warnings

Congratulations, your FPGA hardware design is now complete.

## Adjusting Pin Names

The pin naming convention for every development kit supported by this lab can be different. For instance some kits name the push buttons USER\_PB and others call it KEY. You will need to check the Appendix A for which pin names you will need to change in the assignment editor to make sure they are properly mapped when running the compilation step. To see if you need to adjust the pinout names, do the following: (LL).

## SOFTWARE DESIGN

### Creating the Software for the “Hello World” design

Should you choose to start directly in the Software Design section and skip the Hardware Design section, consult with your lab facilitator to get these two files: `nios_setup_v2.sopcinfo` and `hello_world_lab.sof` as if you generated them from the Hardware Design lab. You will be able to complete all subsequent steps with these two files.

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 the

SBT Tools → NIOS II Software Build Tools for Eclipse (Tools -> Nios II Software build tools for Eclipse). You can use the default location that Eclipse picks for you.

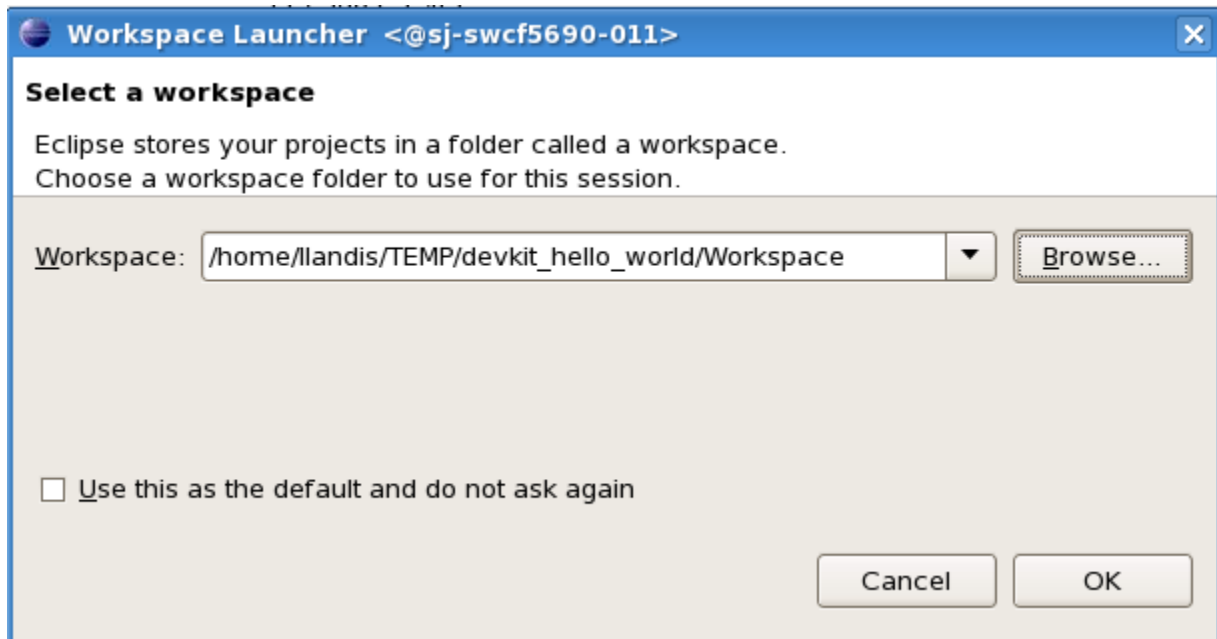
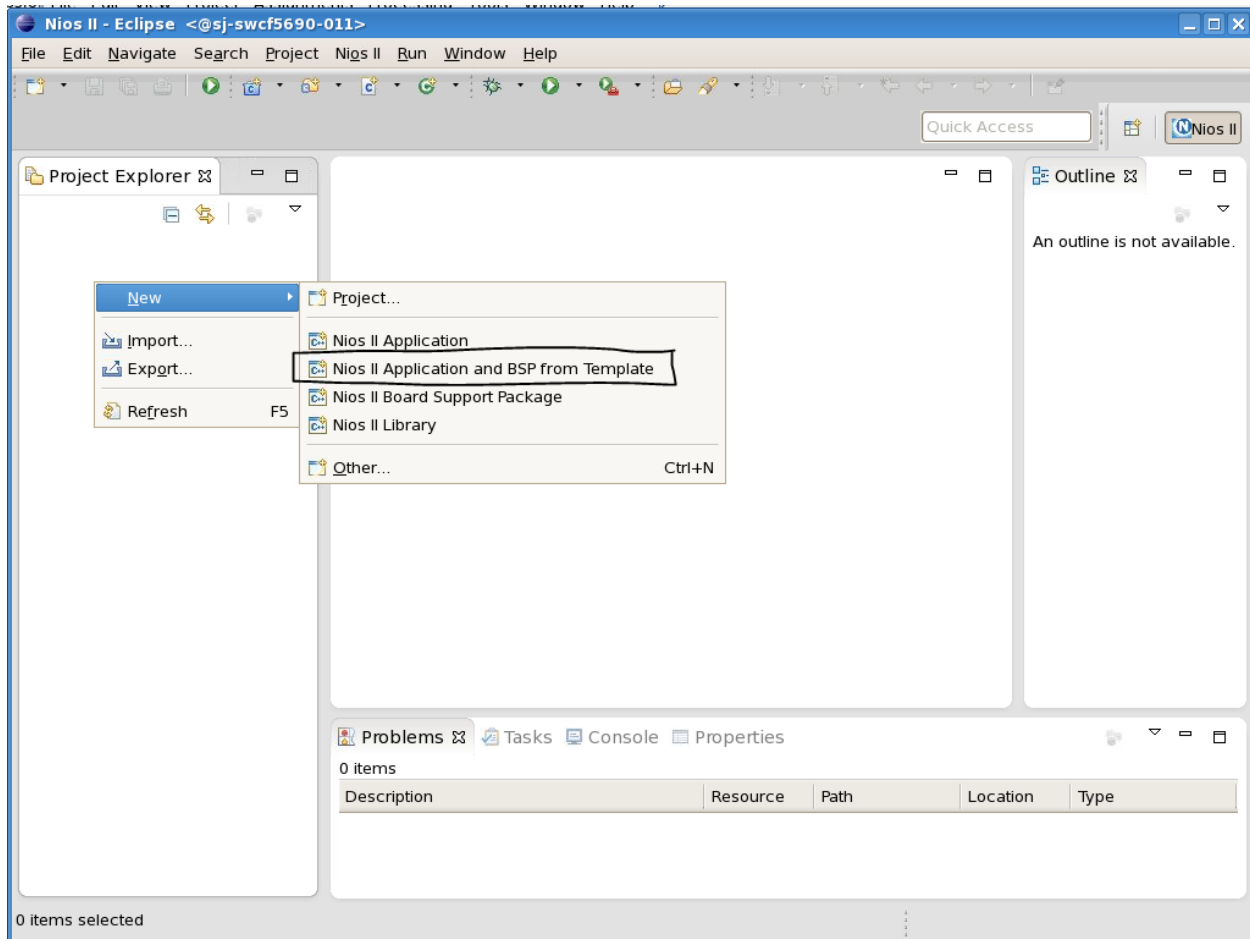


Figure 41: Initial Workspace setup

Click OK in the Workspace launcher.

Next, the Eclipse SBT will launch. Right click in the area called Project Explorer and select New→Nios II Application and BSP from Template. 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.



**Figure 42: Creating the initial project in the Eclipse SBT**

Next you will see a panel that requests information to setup your design. First, you need to navigate to your working directory and click on the .sopcinfo file. The .sopcinfo file informs Eclipse on what your Qsys system contains. Click OK.

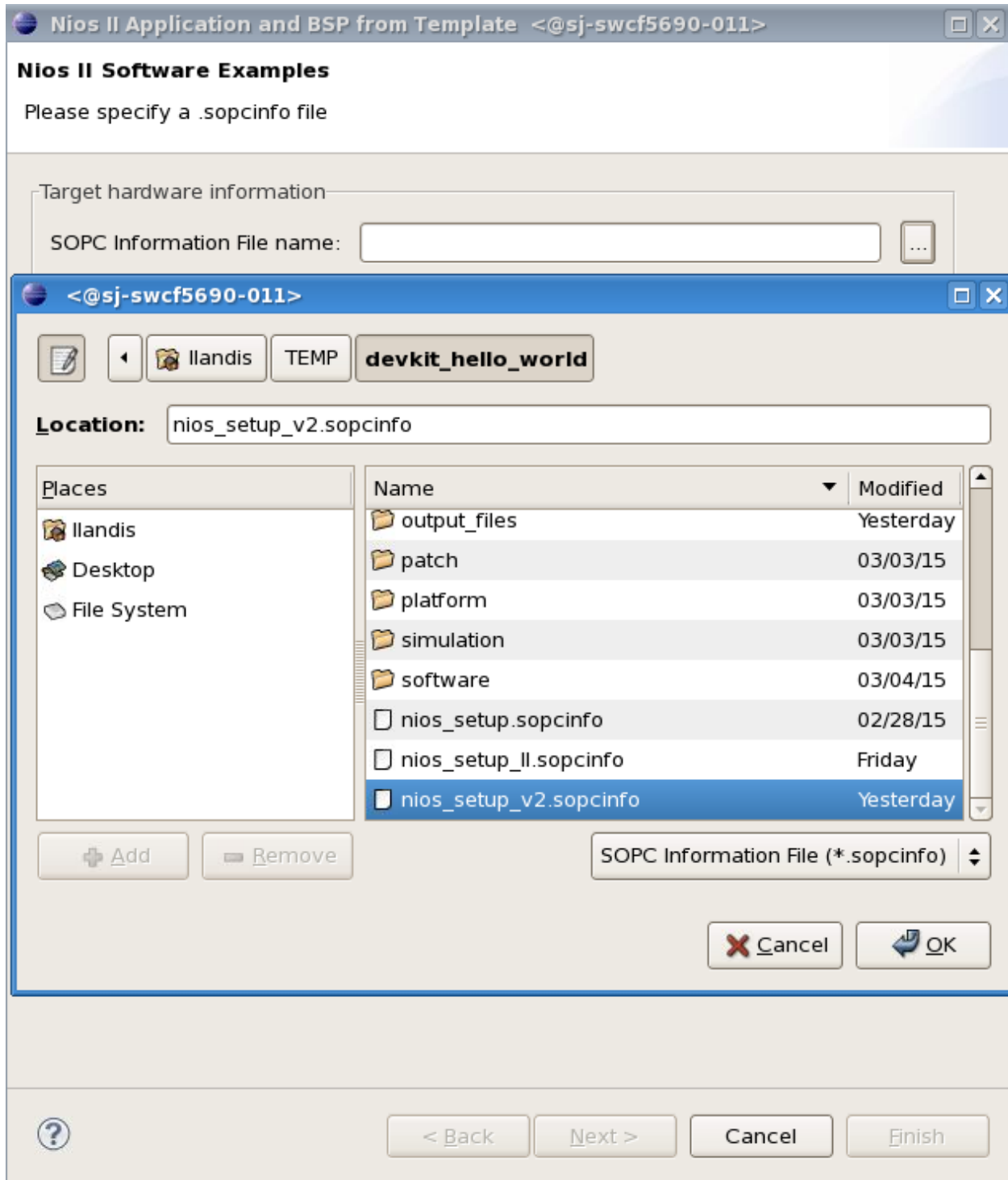


Figure 43: Navigating to correct .sopcinfo file

Fill in the Project name. Call it hello\_world\_sw. Next you will be asked to pick a template design. The Hello World Small is a software application to write "Hello from Nios II" to the screen. Click Finish. Note: make sure to pick Hello World Small and NOT "Hello World" or you will not have enough memory in your FPGA design to store the program executable.

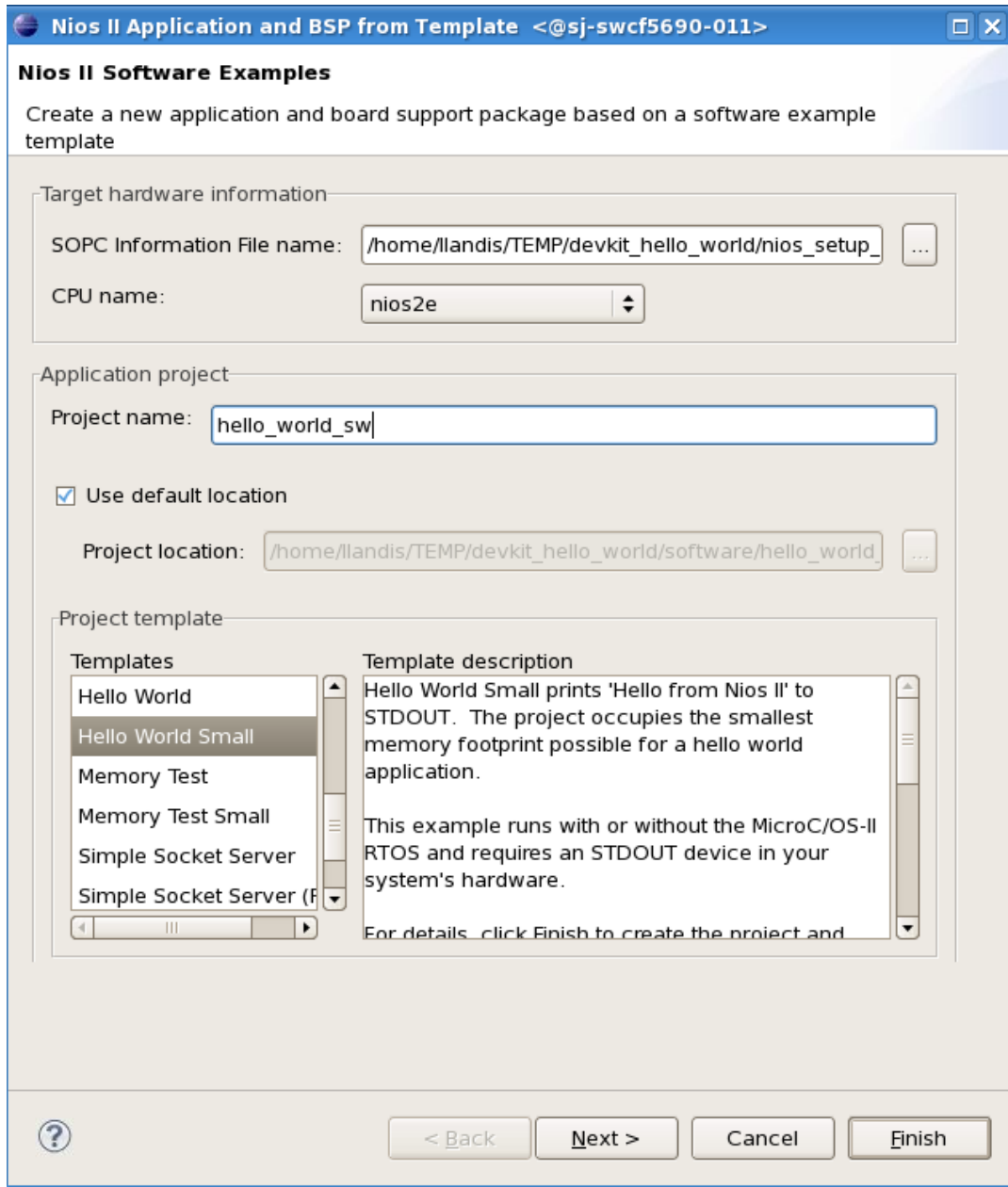
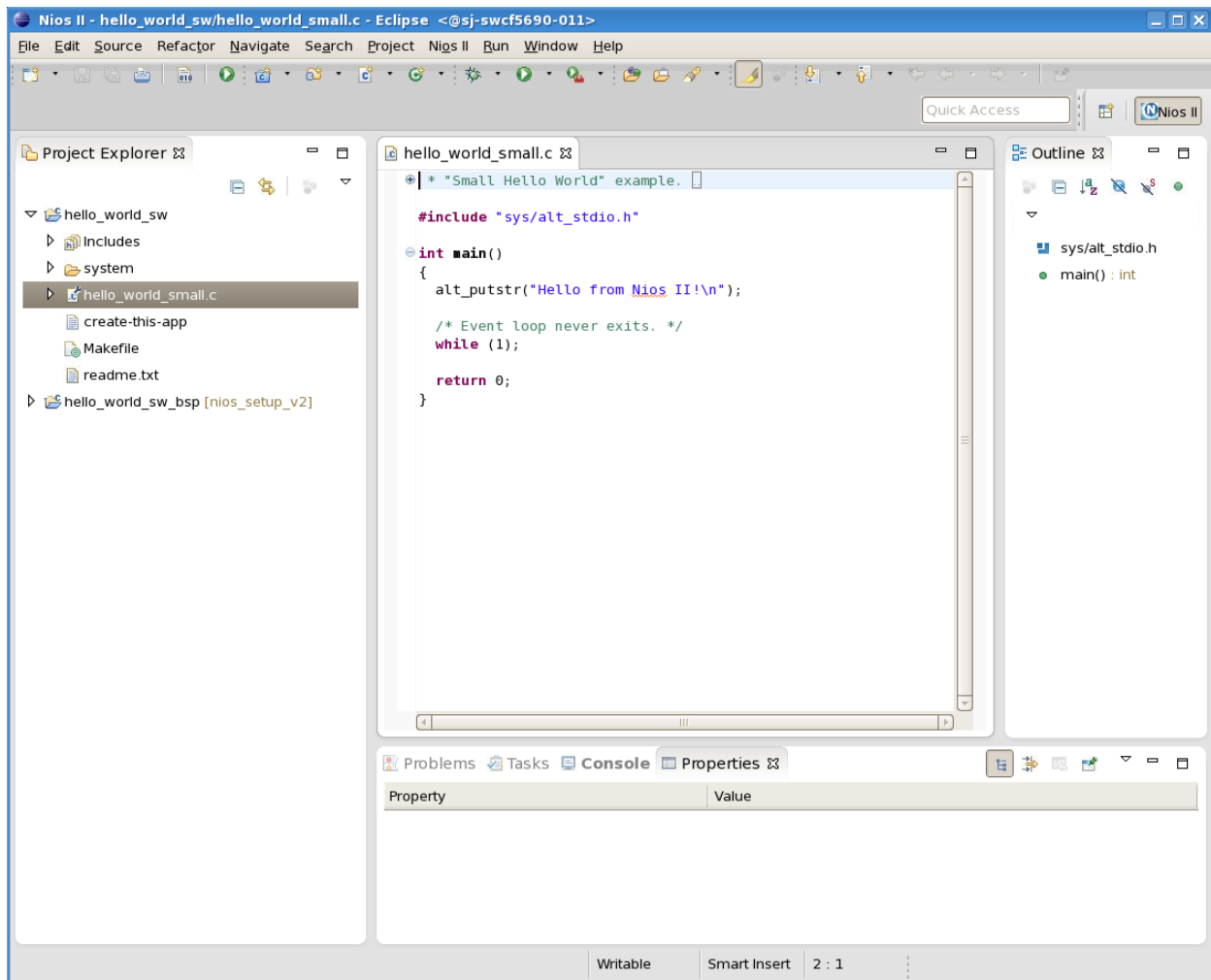


Figure 44: completing the Nios II Software Examples setup screen with project name and project template.

We will now make some modifications to the code to connect the LED[1] to the push button switch through software. Click the right arrow next to hello\_world\_sw. It will show the contents of your project. Double-click hello\_world\_small.c . Note the command alt\_putstr to write text to the terminal. This is part of the Altera HAL (Hardware Abstraction Layer) set of software functions. Note that the alt\_putstr command is used versus a standard C printf function because the code space is more compact using the HAL commands. Code using HAL functions without an operating system is referred to as “bare metal” programming. A complete list of these functions can be found in the Nios II Software Developer’s Handbook

[https://www.altera.com/en\\_US/pdfs/literature/hb/nios2/n2sw\\_nii5v2.pdf](https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2.pdf) .



Next you need to add a library declaration, define integer switch\_datain, and a few HAL functions to connect the LEDs to the Push Buttons:

```
#include <sys/alt_stdio.h>
#include <stdio.h>
#include "altera_avalon_pio_regs.h"
#include "system.h"

int main()
{
    int switch_datain;
    alt_putstr("Hello from Nios II!\n");

    /* Event loop never exits. Read the PB, display on the LED */
    while (1){
        switch_datain = IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE);
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,switch_datain);
    }

    return 0
}
```

Note the use of the variables SWITCH\_BASE and LED\_BASE. These variables are created by importing the information from the .sopcinfo file. You can find defined variables in the system.h file under the hello\_world\_sw\_bsp project. Double click on system.h file and inspect the defined variable names for SWITCH\_BASE and LED\_BASE. These must match your hello\_world\_small.c code. Edit the hello\_world\_small.c source code so that it matches the code

shown above. Click the save  icon.

Right click on the hello\_world\_sw project. Left click Build. This compiles the software application and the BSP (drivers).

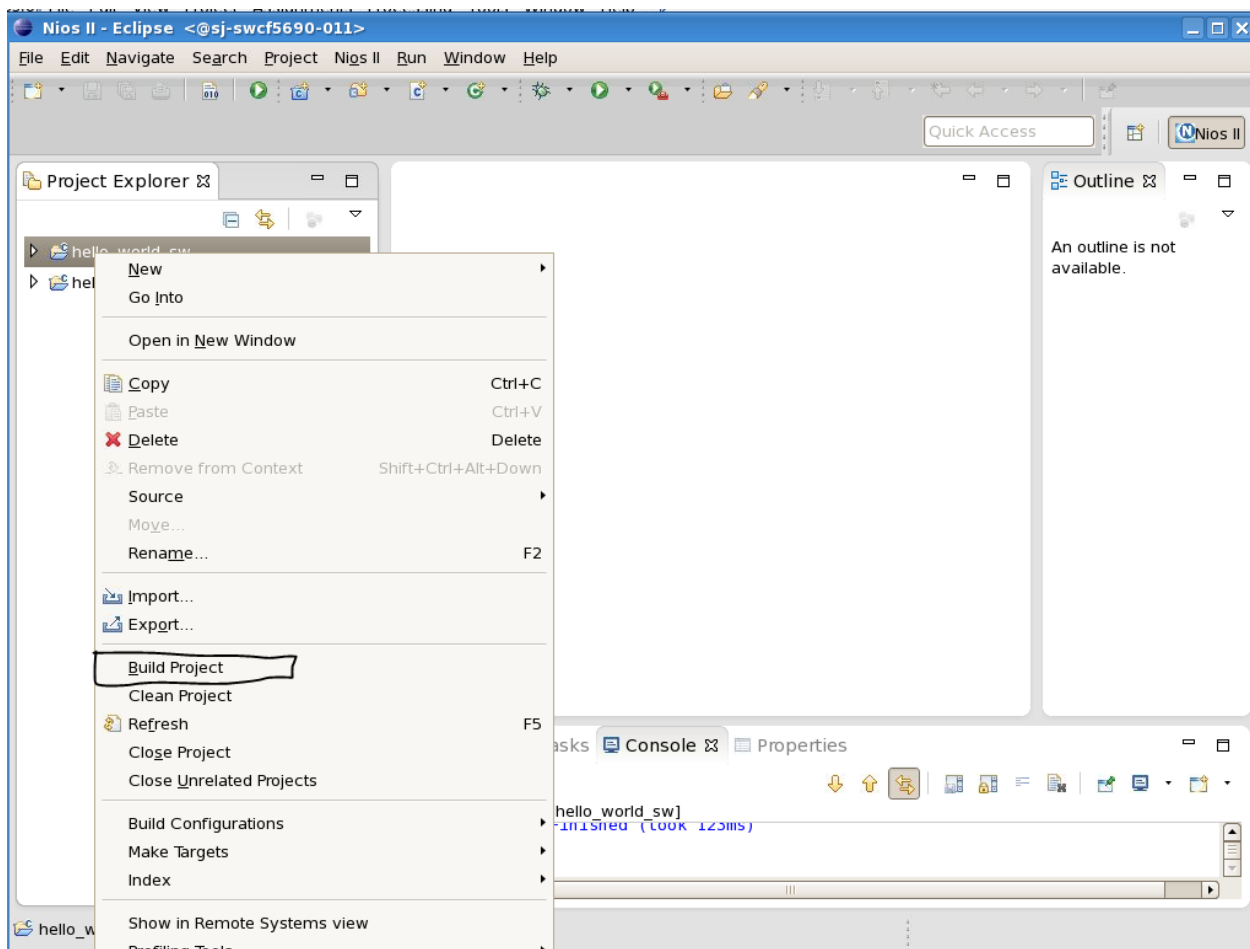


Figure 45: Launching the build

Once the build completes, you should observe a ".elf" file (executable load file) under the hello\_world\_sw project. 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 can be missing semicolons, mismatched brackets and such.

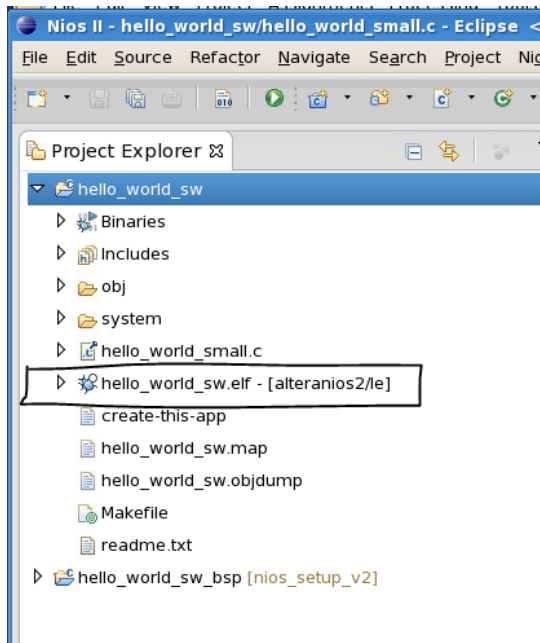


Figure 46: The presence of `hello_world_sw.elf` indicates the software build ran successfully

## MAX10 FPGA Development Kit Cable Connections and Switch Settings

To work with the MAX 10 development kit in the context of this lab, you will need to connect the power supply to the DC Input and a USB cable connecting the kit to a host PC. It is very important to note that there are 2 USB connectors: USB Blaster and USB UART. You must connect the USB cable to the USB Blaster (U12) connector. The USB blaster utilizes circuitry that formats the image into a data stream that downloads from the PC to FPGA. If you connect to the USB UART connector, your image will fail to download.

Make sure your development kit is powered up (blue button on kit) and LEDs are on.



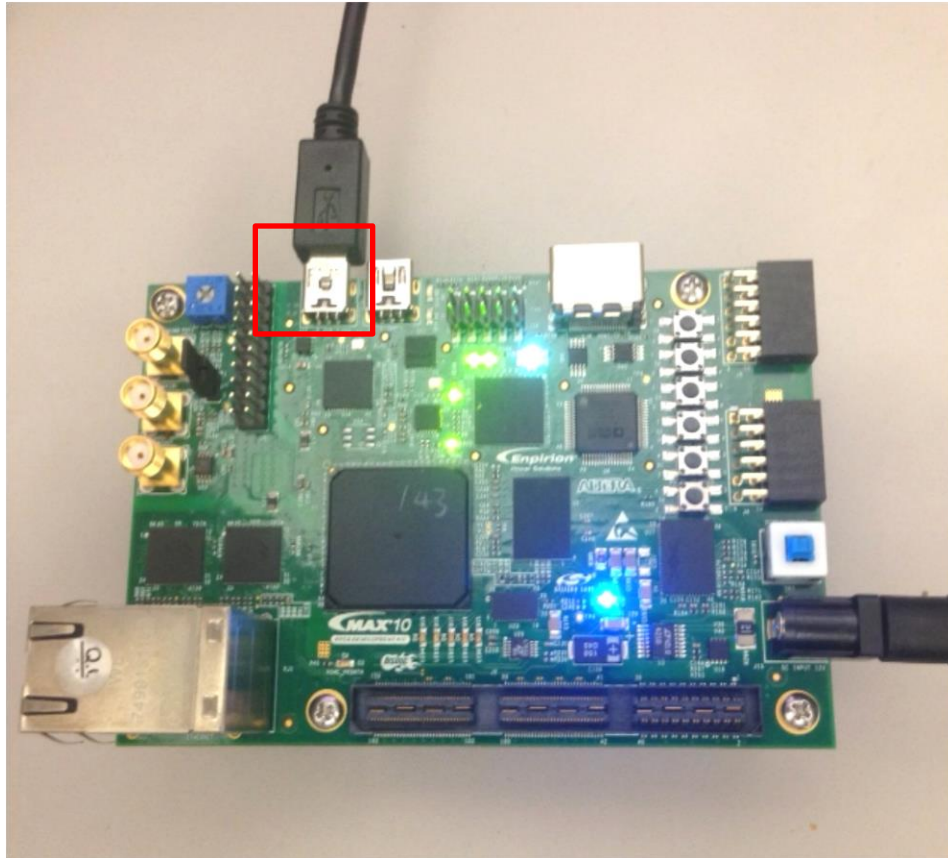


Figure 47: Proper location to connect USB Blaster cable

If you are only performing the Software design lab, you must first launch Quartus. If you have just performed the Hardware Design lab, then Quartus should already be open. Launch the Programmer: Tools → Programmer.

Click Auto Detect and you should see something similar to Figure 48. Make sure the device number selected matches with the device you are working with. Select the device in the window (Assignments -> Device) as shown below.

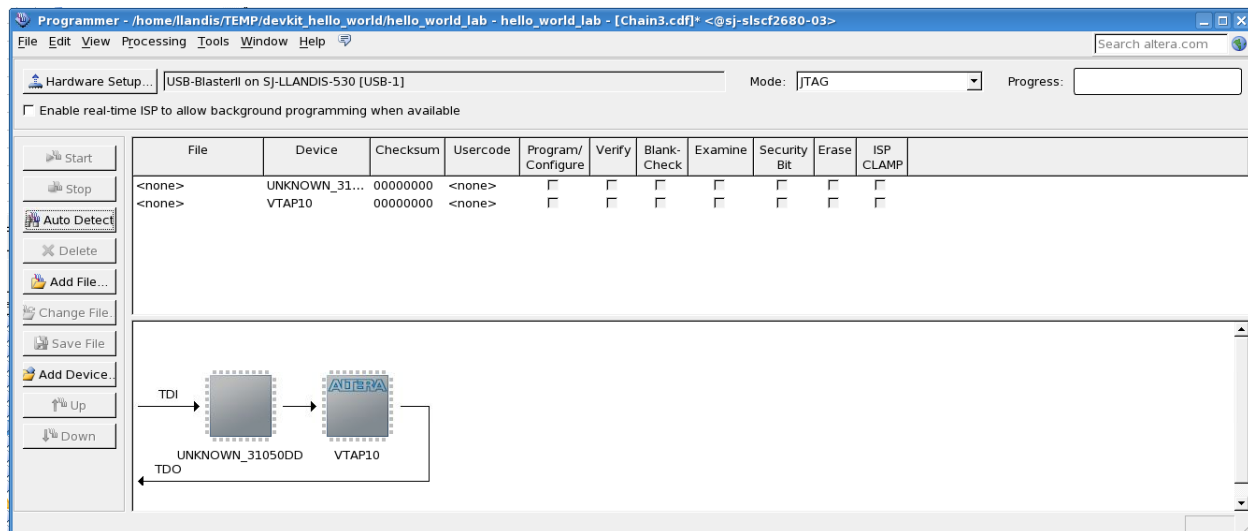
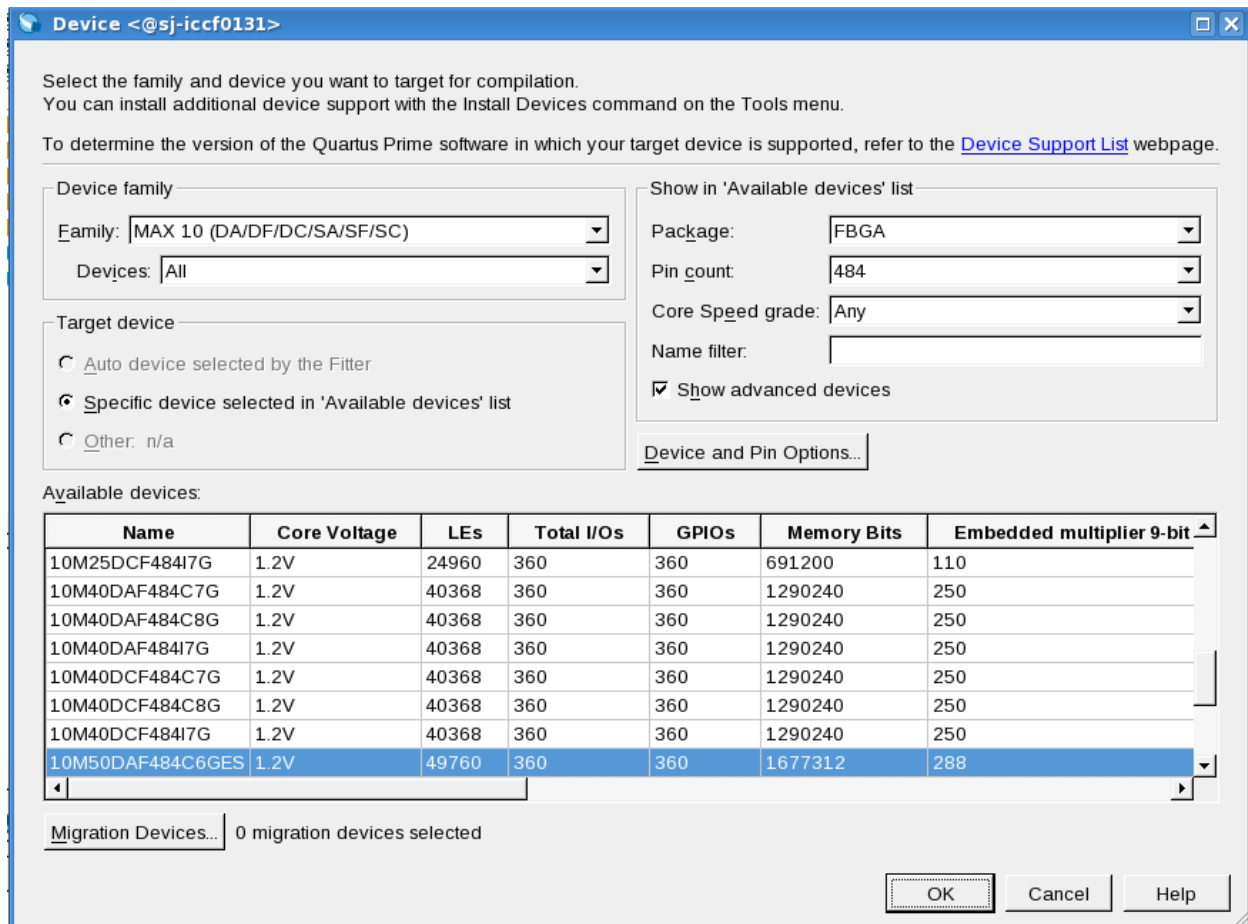


Figure 48: Programmer after Auto Detect

Next, you need to download what is called a “.sof” file or SRAM object file. This is the programming image file that gets downloaded in the FPGA. The default location is <working\_directory>/output\_files. Right click on the first row <none> under File and click on Change File. Navigate to the output\_files directory and select hello\_world\_lab.sof. Click Open. In the first row under Program/Configure click in the check box as shown in Figure 49.

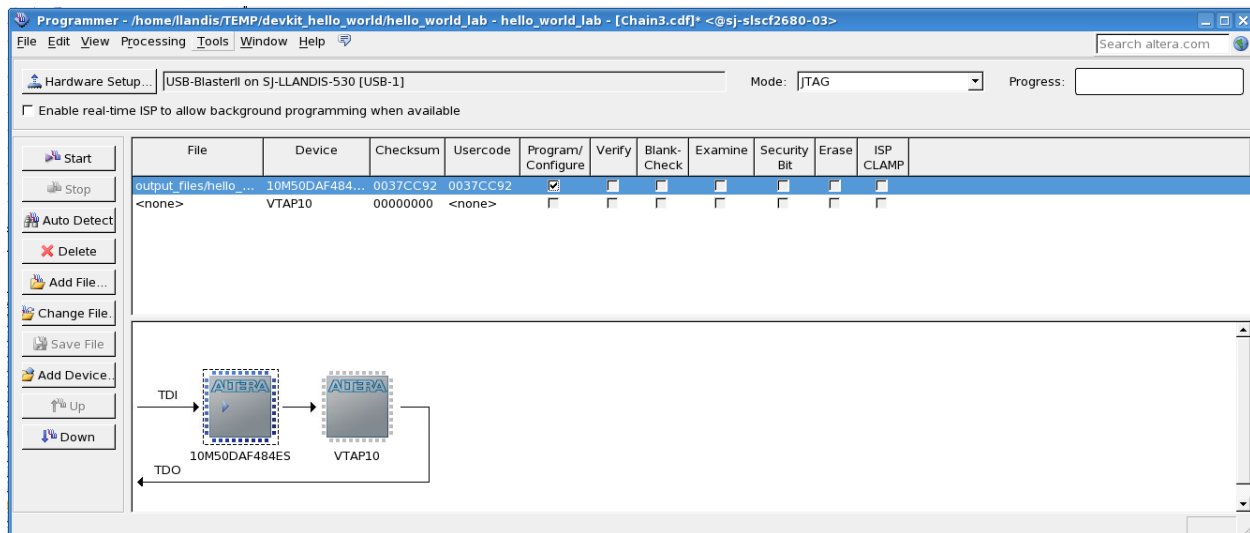


Figure 49: Programmer after adding hello\_world\_lab.sof file

Click Start. When programming is complete, the Progress meter should read 100% (Successful).

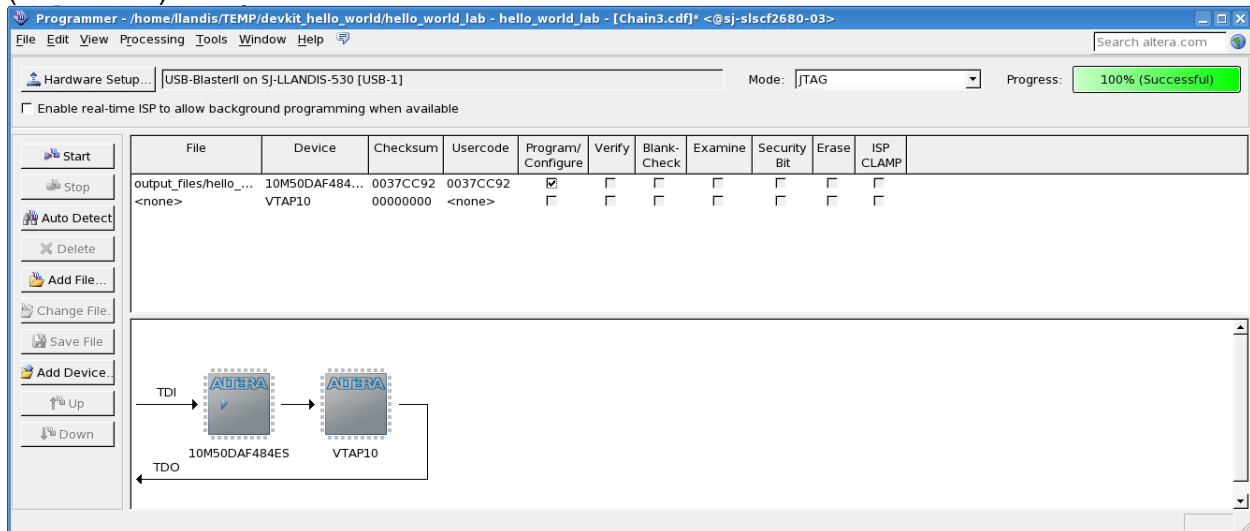


Figure 50: Programmer after completing .sof download

Now it is time to download the .elf (software executable) into the Nios II processor. Return to the Eclipse SBT tools. Right click on hello\_world\_sw and select Run as → Run Nios II Hardware. Click on the Target Connection tab. The connection should indicate that Eclipse has connected to USB-blaster. If the connection is not identified, you can Click Refresh Connections. Note that you might need to stretch the window wider to see the Refresh Connections button. Once the connection is made to the USB-Blaster, you should observe something similar to Figure 51. Click Run.

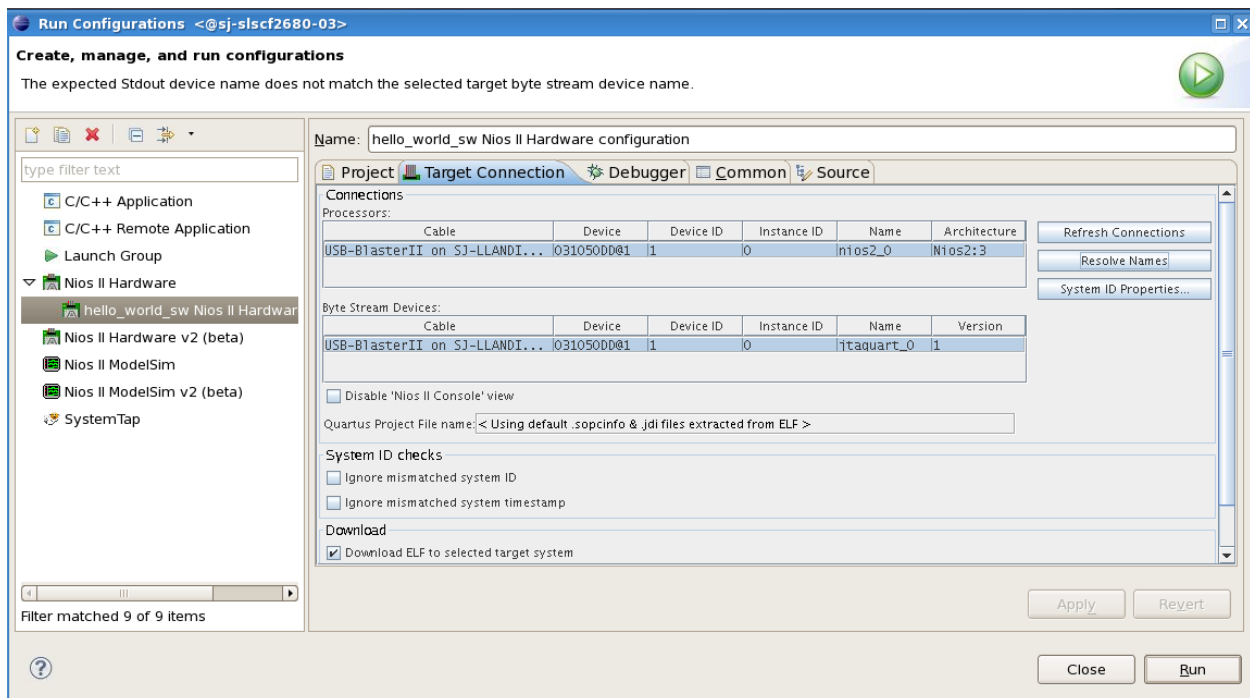


Figure 51: Run (Nios II) Configurations Window in Eclipse.

Now you have hardware and software downloaded into your MAX10 FPGA Development Kit. You should observe “Hello from Nios II” in the Nios II Console tab.

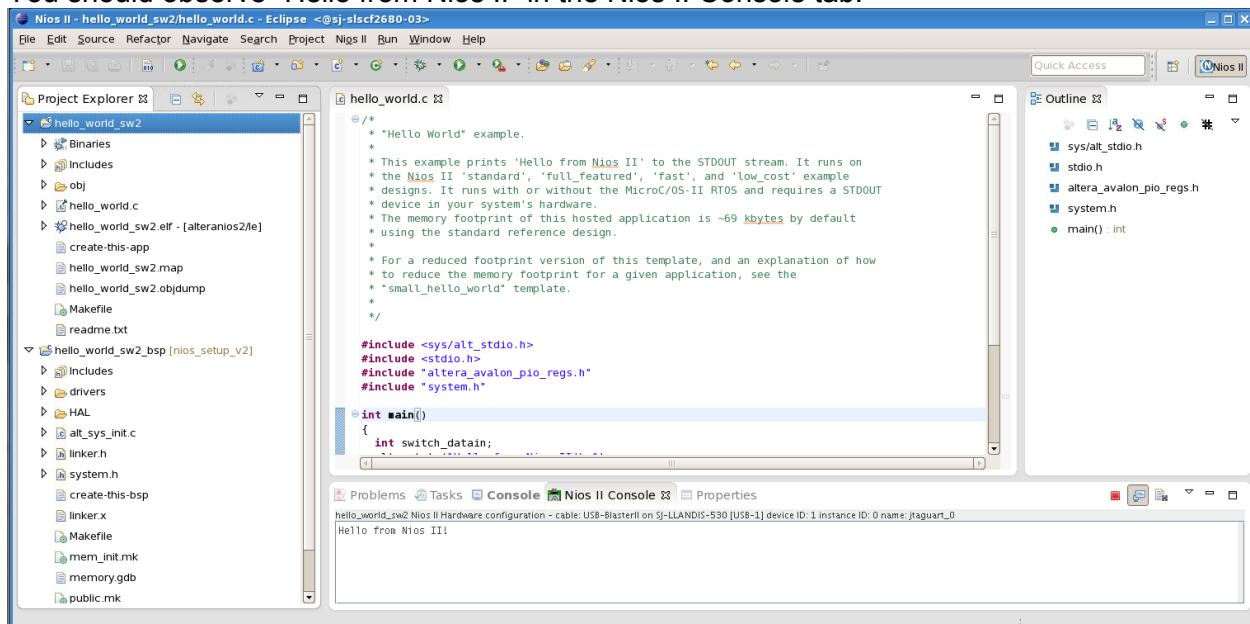


Figure 52: “Hello from Nios II!” displayed on the Nios II Console

You can also test the connections between push button and LEDs. Recall that 2 buttons [1:0] are connected in hardware are inverted so by default the LEDs are on. Buttons [3:2] are connected in the C code and illuminate LEDs [3:2] when pushed. Refer to the diagram below and confirm that the push buttons operate the LEDs based on the hello\_world.v and

hello\_world.c source files. Be careful not to hit the top and bottom push buttons PULSE\_NCONFIG or FPGA\_RESETN or you can disrupt the FPGA programmed status.

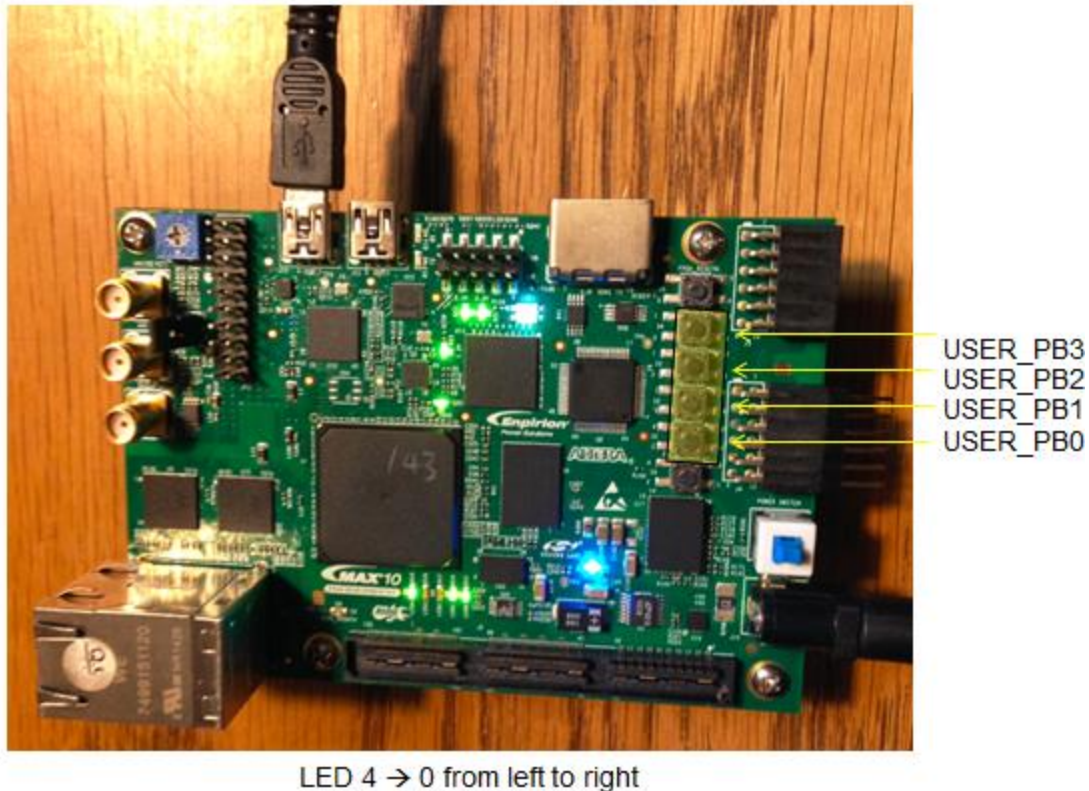


Figure 53: Operating the hello\_world lab push buttons

## Lab Summary

You now have completed the hardware and software sections of this lab. This includes:

1. Loading the Device Kit pin settings into Quartus
2. Using Qsys to build a Nios II based system
3. Instantiating the Qsys component into your top level design
4. Add some connections between push buttons and LEDs
5. Compiling your hardware
6. Importing the Nios II based system into the Eclipse Software Build Tools
7. Building a software project
8. Modifying a software template to perform some simple IO functions
9. Compiling your software
10. Downloading the hardware image into the development kit.
11. Downloading the software executable into the development kit.
12. Testing the hardware

There is a wealth of resources from Altera and partners to take classes on Embedded Hardware, Embedded Software and reference design starting points to advance your skills using Altera's powerful Nios II based hardware and software tools.