Cyclone 10 LP Nios II ‘Hello World’ Lab Manual

Date: 8/25/2017

Revision: 1.0

©2017 Intel Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, INTEL, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Intel Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Table of Contents

Theory of Operation

This lab showcases how to create an embedded system in programmable logic using Intel's "soft" Nios II processor.

In this design, the Nios II processor 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, 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. The lab is split into Hardware Section and Software Section

Hardware and Software Requirements

The reference design requires the following hardware and software to run the test:

- Quartus® Prime Software Version: 17.0
- Cyclone 10 LP FPGA Evaluation Kit

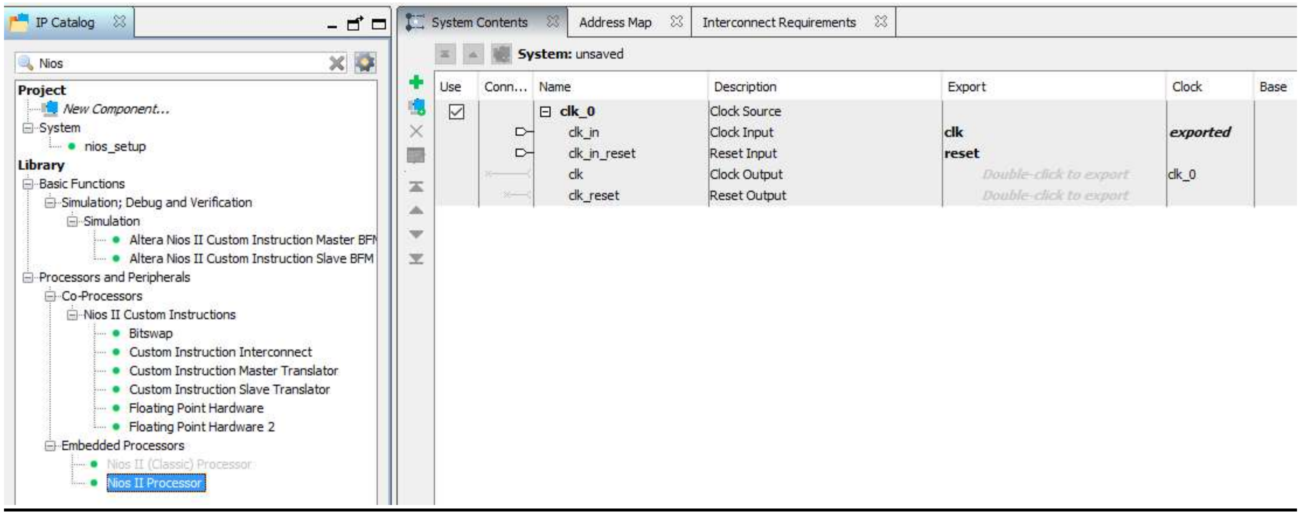
To directly use this reference design:

1) Download this design 'C10LP_NiosII_hello_world.par' from Design Store.
2) Open the .par in Quartus Prime 17.0.0 Standard Edition.
3) Close the project, then rename the design from 'C10LP_NiosII_hello_world_project' to 'C10LP_NiosII_hello_world'.
4) Unzip the 'software.zip' folder, and copy the 'software' folder in this project main directory.
5) Re-open the project in Quartus Prime, perform compilation, then download the .sof into dev kit.
6) Next, follow the steps in section 'SOFTWARE DESIGN' on this document to build the Nios II and run the test.

HARDWARE DESIGN

Build the Qsys based processor system

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



IP Catalog

Nios

- **Project**
  - *New Component...*
  - System
    - nios_setup
- **Library**
  - Basic Functions
    - Simulation; Debug and Verification
      - Simulation
        - Altera Nios II Custom Instruction Master BFM
        - Altera Nios II Custom Instruction Slave BFM
  - Processors and Peripherals
    - Co-Processors
      - Nios II Custom Instructions
        - Bitswap
        - Custom Instruction Interconnect
        - Custom Instruction Master Translator
        - Custom Instruction Slave Translator
        - Floating Point Hardware
        - Floating Point Hardware 2
    - Embedded Processors
      - Nios II (Classic) Processor
      - Nios II Processor

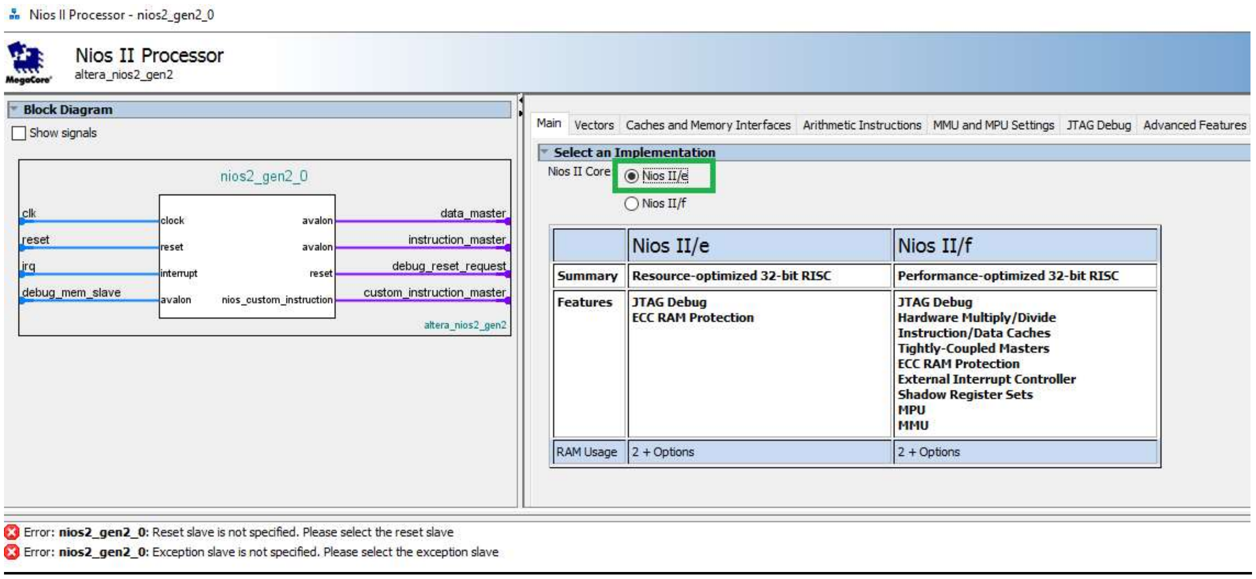
System Contents | Address Map | Interconnect Requirements

System: unsaved

Use	Conn...	Name	Description	Export	Clock	Base
✓		**clk_0**	Clock Source			
		clk_in	Clock Input	**clk**	***exported***	
		clk_in_reset	Reset Input	**reset**		
		clk	Clock Output	*Double-click to export*	clk_0	
		clk_reset	Reset Output	*Double-click to export*		

Figure 1: Qsys main panel

Next, add the required 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.



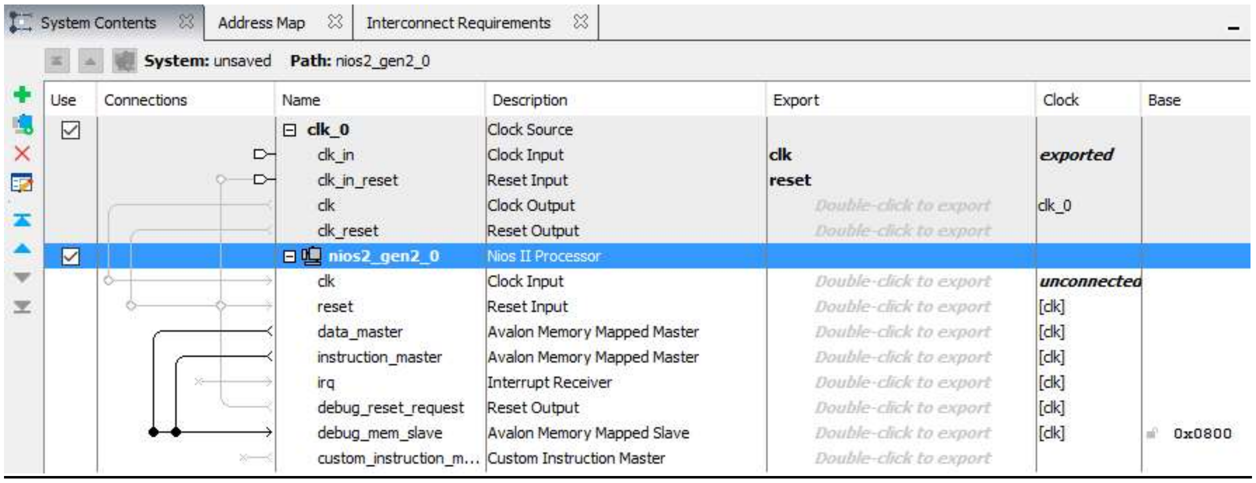
Nios II Core: [✓ Nios II/e] Nios II/f

	Nios II/e	Nios II/f
Summary	Resource-optimized 32-bit RISC	Performance-optimized 32-bit RISC
Features	JTAG Debug ECC RAM Protection	JTAG Debug Hardware Multiply/Divide Instruction/Data Caches Tightly-Coupled Masters ECC RAM Protection External Interrupt Controller Shadow Register Sets MPU MMU
RAM Usage	2 + Options	2 + Options

```
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
```

Figure 2: Nios II Configuration panel

Click finish and you will see the Nios II/e processor in your connection diagram (The Qsys system errors reported will be addressed them soon.)

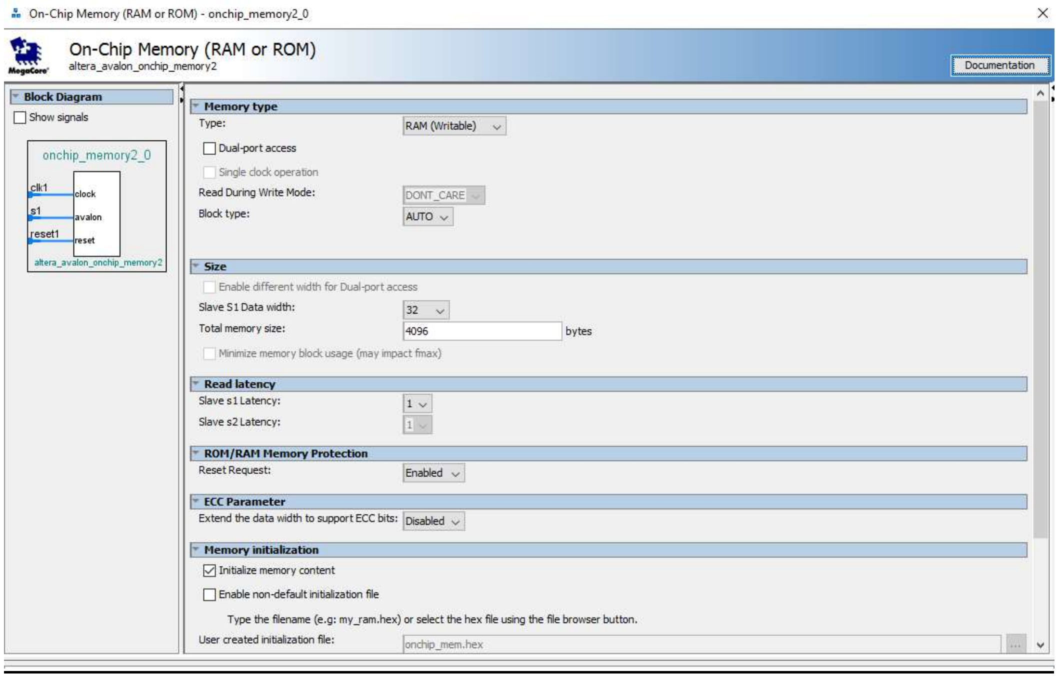


Use	Connections	Name	Description	Export	Clock	Base
✓		⊟ clk_0	Clock Source			
		clk_in	Clock Input	clk	exported	
		clk_in_reset	Reset Input	reset		
		clk	Clock Output	Double-click to export	clk_0	
		clk_reset	Reset Output	Double-click to export		
✓		⊟ nios2_gen2_0	Nios II Processor			
		clk	Clock Input	Double-click to export	unconnected	
		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
		custom_instruction_m...	Custom Instruction Master	Double-click to export		

Figure 3: Qsys System Contents panel

Qsys has an efficient way of making connections by clicking on the nodes on 'wires' in the connections panel on the 2nd 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 into the system.

Locate the On-Chip Memory (RAM or ROM) component and click add. Use all the default settings except 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.



On-Chip Memory (RAM or ROM) - onchip_memory2_0

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Block Diagram
☐ Show signals

Memory type
- Type: RAM (Writable)
- ☐ Dual-port access
- ☐ Single clock operation
- Read During Write Mode: DONT_CARE
- Block type: AUTO

Size
- ☐ Enable different width for Dual-port access
- Slave S1 Data width: 32
- Total memory size: 4096 bytes
- ☐ Minimize memory block usage (may impact fmax)

Read latency
- Slave s1 Latency: 1
- Slave s2 Latency: 1

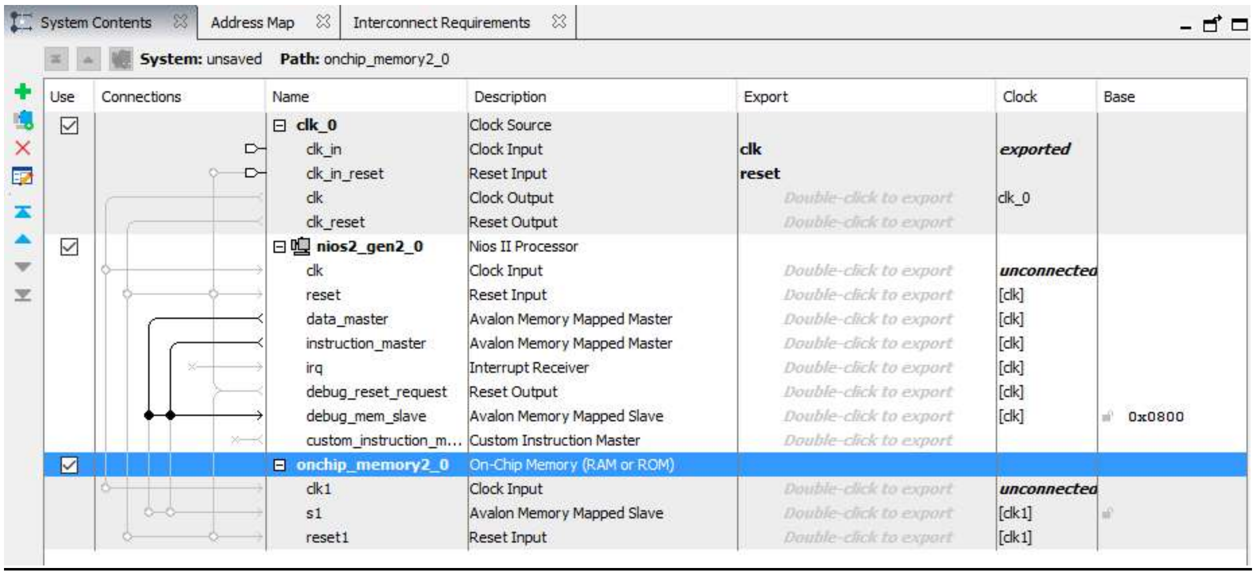
ROM/RAM Memory Protection
- Reset Request: Enabled

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

Figure 4: 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.



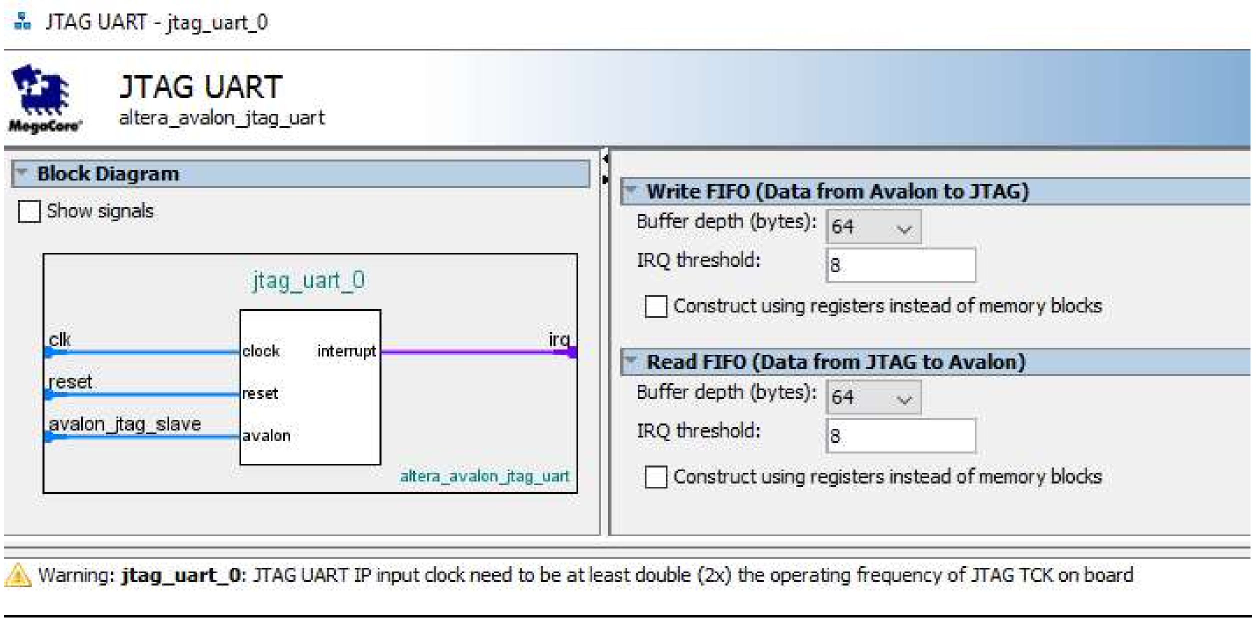
System Contents | Address Map | Interconnect Requirements

System: unsaved Path: onchip_memory2_0

Use	Name	Description	Export	Clock	Base
☑	clk_0	Clock Source			
	clk_in	Clock Input	clk	exported	
	clk_in_reset	Reset Input	reset		
	clk	Clock Output	Double-click to export	clk_0	
	clk_reset	Reset Output	Double-click to export		
☑	nios2_gen2_0	Nios II Processor			
	clk	Clock Input	Double-click to export	unconnected	
	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
	custom_instruction_m…	Custom Instruction Master	Double-click to export		
☑	onchip_memory2_0	On-Chip Memory (RAM or ROM)			
	clk1	Clock Input	Double-click to export	unconnected	
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	
	reset1	Reset Input	Double-click to export	[clk1]	

Figure 5: System contents with NIOSII and On-chip Memory

The next component need to 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.



- [ ] Show signals

Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64

IRQ threshold: 8

- [ ] Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64

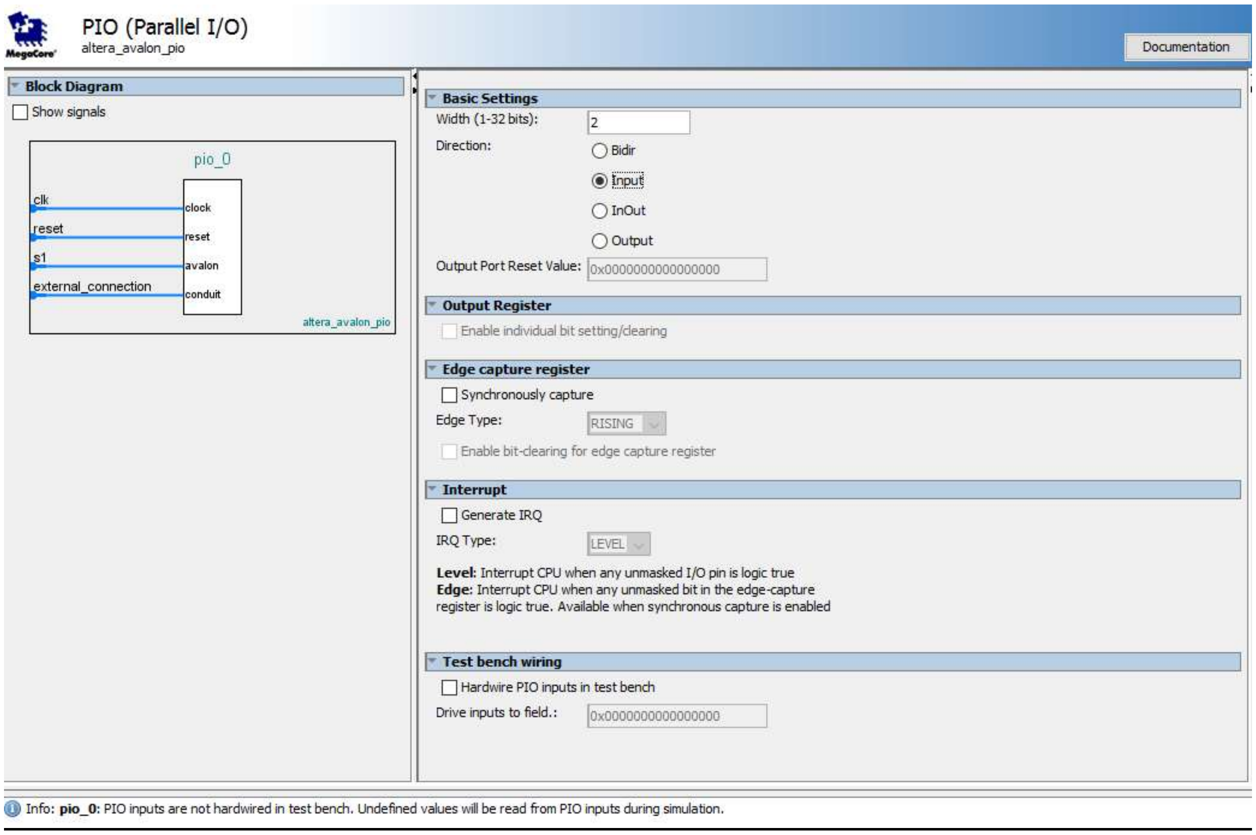
IRQ threshold: 8

- [ ] Construct using registers instead of memory blocks

Warning: **jtag_uart_0**: JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board

Figure 6: JTAG UART configuration panel

The next two components SWITCH and LED are instances of general purpose parallel IO components in the IP catalog. Search for parallel IO (PIO) and select this block. For the switch block, you will set this up as a 2-bit input interface using the settings shown below.



PIO (Parallel I/O)
altera_avalon_pio

Block Diagram
☐ Show signals

Basic Settings
Width (1-32 bits): 2
Direction: Bidir [✓ Input] InOut Output
Output Port Reset Value: 0x0000000000000000

Output Register
☐ Enable individual bit setting/clearing

Edge capture register
☐ Synchronously capture
Edge Type: RISING
☐ Enable bit-clearing for edge capture register

Interrupt
☐ Generate IRQ
IRQ Type: LEVEL

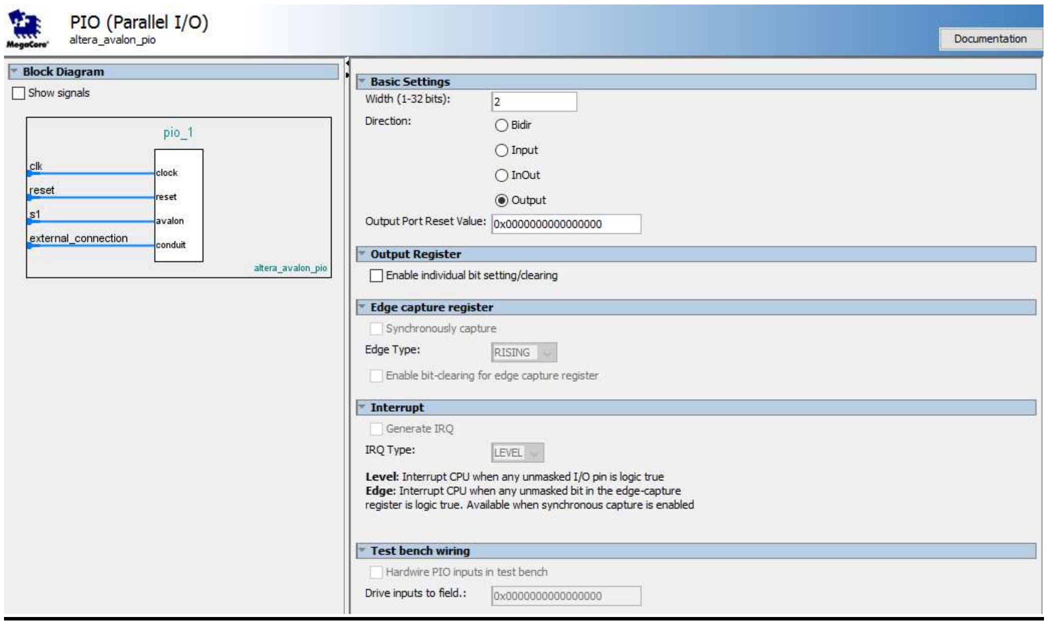
Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Test bench wiring
☐ Hardwire PIO inputs in test bench
Drive inputs to field.: 0x0000000000000000

Info: **pio_0:** PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

Figure 7: Parallel IO configuration panel for Push Button input

Next, 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 2 bit output.



PIO (Parallel I/O)
altera_avalon_pio

Block Diagram
☐ Show signals

Basic Settings
Width (1-32 bits): 2
Direction: Bidir Input InOut [✓ Output]
Output Port Reset Value: 0x0000000000000000

Output Register
☐ Enable individual bit setting/clearing

Edge capture register
☐ Synchronously capture
Edge Type: RISING
☐ Enable bit-clearing for edge capture register

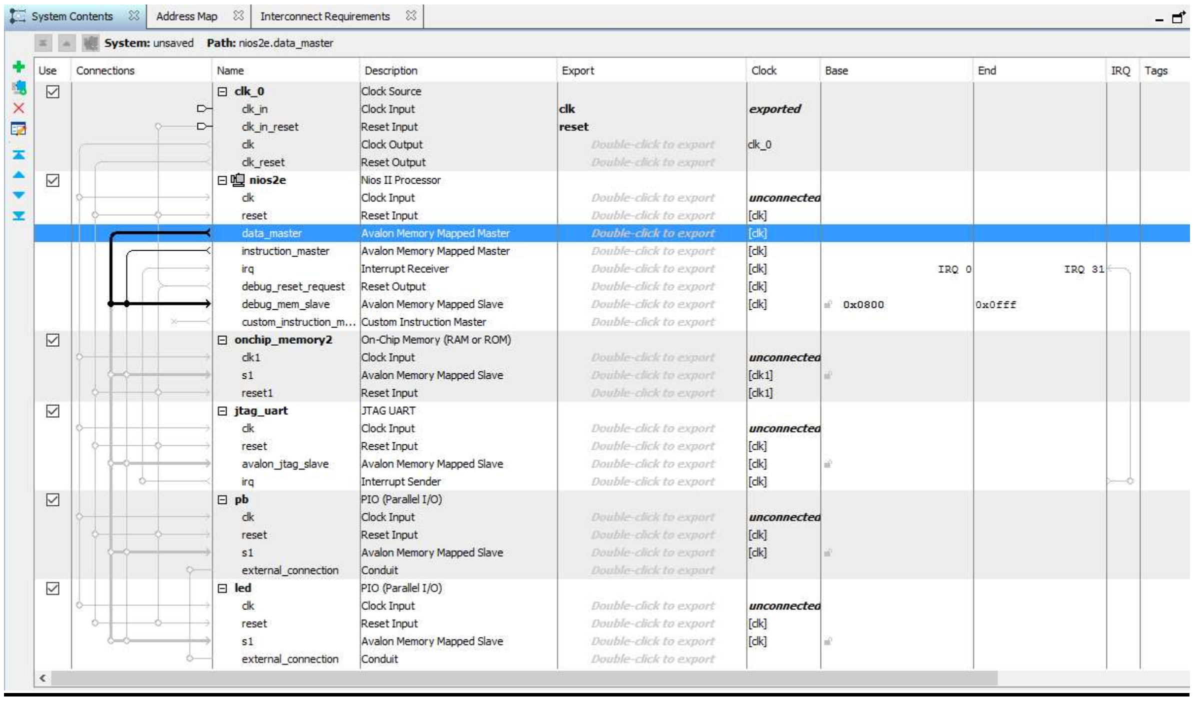
Interrupt
☐ Generate IRQ
IRQ Type: LEVEL

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Test bench wiring
☐ Hardwire PIO inputs in test bench
Drive inputs to field.: 0x0000000000000000

Figure 8: Parallel IO configuration panel for LED outputs

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, 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.



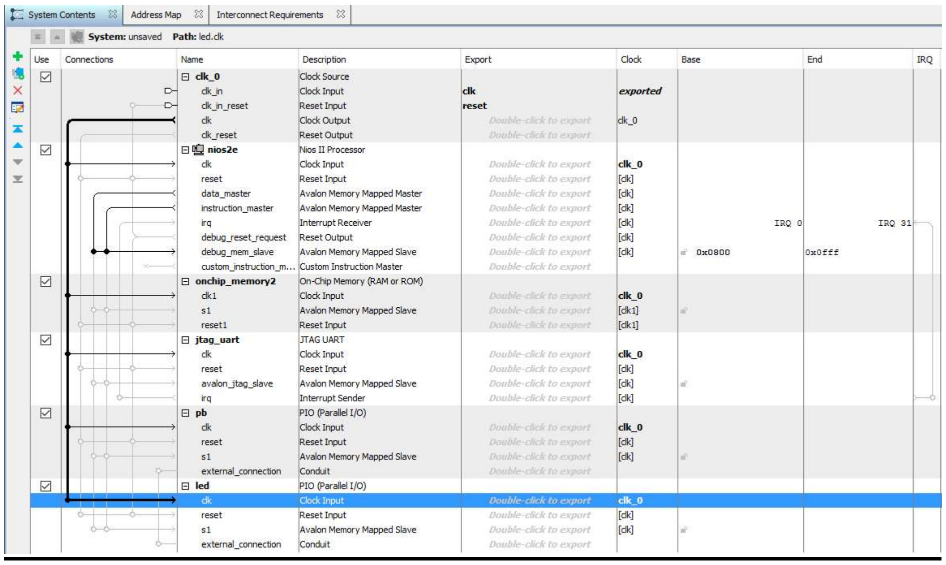
System Contents | Address Map | Interconnect Requirements

System: unsaved **Path:** nios2e.data_master

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		⊟ **clk_0**	Clock Source						
		clk_in	Clock Input	**clk**	***exported***				
		clk_in_reset	Reset Input	**reset**					
		clk	Clock Output	*Double-click to export*	clk_0				
		clk_reset	Reset Output	*Double-click to export*					
✓		⊟ **nios2e**	Nios II Processor						
		clk	Clock Input	*Double-click to export*	***unconnected***				
		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]	IRQ 0	IRQ 31		
		debug_reset_request	Reset Output	*Double-click to export*	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0800	0x0fff		
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*					
✓		⊟ **onchip_memory2**	On-Chip Memory (RAM or ROM)						
		clk1	Clock Input	*Double-click to export*	***unconnected***				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]				
		reset1	Reset Input	*Double-click to export*	[clk1]				
✓		⊟ **jtag_uart**	JTAG UART						
		clk	Clock Input	*Double-click to export*	***unconnected***				
		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]				
✓		⊟ **pb**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	***unconnected***				
		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*					
✓		⊟ **led**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	***unconnected***				
		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 9: System Content connections starting panel

The next step consists of making the appropriate connections between the components within Qsys.
Click the clk net coming out of clk_0. When first selected, it will be gray color. Make connections by clicking on the small open circles on the lines that intersecting with the 5 other components.



System Contents | Address Map | Interconnect Requirements

System: unsaved **Path:** led.clk

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
☑		**clk_0**	Clock Source					
		clk_in	Clock Input	**clk**	***exported***			
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	*Double-click to export*	clk_0			
		clk_reset	Reset Output	*Double-click to export*				
☑		**nios2e**	Nios II Processor					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		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]	IRQ 0	IRQ 31	
		debug_reset_request	Reset Output	*Double-click to export*	[clk]			
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0800	0x0fff	
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*				
☑		**onchip_memory2**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	*Double-click to export*	**clk_0**			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]			
		reset1	Reset Input	*Double-click to export*	[clk1]			
☑		**jtag_uart**	JTAG UART					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		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]			
☑		**pb**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		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*				
☑		**led**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to export*	clk_0			
		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 10: System contents after clock connection

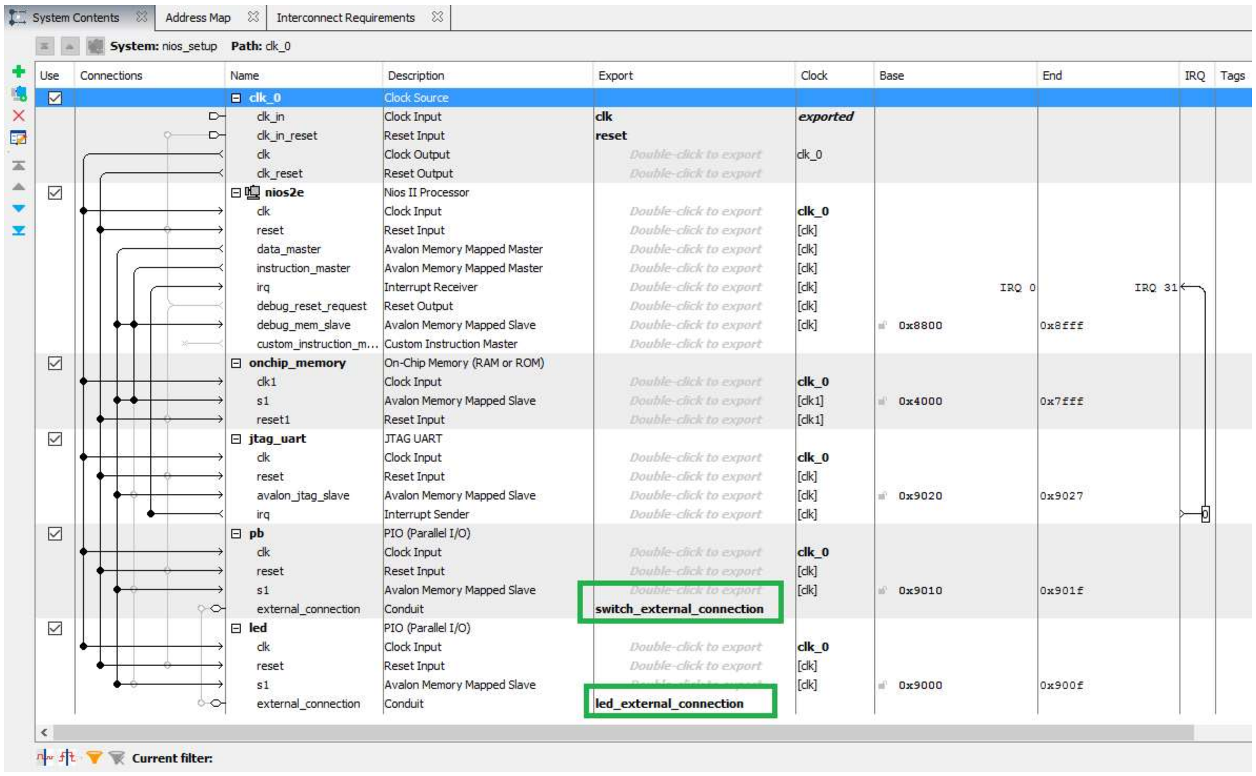
Perform the same operation to connect the clk_reset to the resets on the other components.

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.

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.

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. Use the default setting for the IRQ number.

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 28. This will bring these ports out of the Qsys component to connect to the top level design.



System Contents | Address Map | Interconnect Requirements

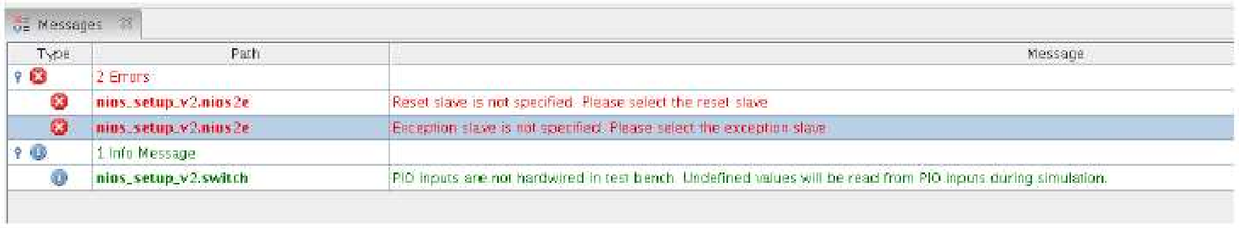
System: nios_setup **Path:** clk_0

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		⊟ **clk_0**	Clock Source						
		clk_in	Clock Input	**clk**	*exported*				
		clk_in_reset	Reset Input	**reset**					
		clk	Clock Output	Double-click to export	clk_0				
		clk_reset	Reset Output	Double-click to export					
✓		⊟ **nios2e**	Nios II Processor						
		clk	Clock Input	Double-click to export	**clk_0**				
		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]	IRQ 0	IRQ 31		
		debug_reset_request	Reset Output	Double-click to export	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x8800	0x8fff		
		custom_instruction_m...	Custom Instruction Master	Double-click to export					
✓		⊟ **onchip_memory**	On-Chip Memory (RAM or ROM)						
		clk1	Clock Input	Double-click to export	**clk_0**				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x4000	0x7fff		
		reset1	Reset Input	Double-click to export	[clk1]				
✓		⊟ **jtag_uart**	JTAG UART						
		clk	Clock Input	Double-click to export	**clk_0**				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x9020	0x9027		
		irq	Interrupt Sender	Double-click to export	[clk]			0	
✓		⊟ **pb**	PIO (Parallel I/O)						
		clk	Clock Input	Double-click to export	**clk_0**				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x9010	0x901f		
		external_connection	Conduit	**switch_external_connection**					
✓		⊟ **led**	PIO (Parallel I/O)						
		clk	Clock Input	Double-click to export	**clk_0**				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x9000	0x900f		
		external_connection	Conduit	**led_external_connection**					

Current filter:

Figure 11: 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.



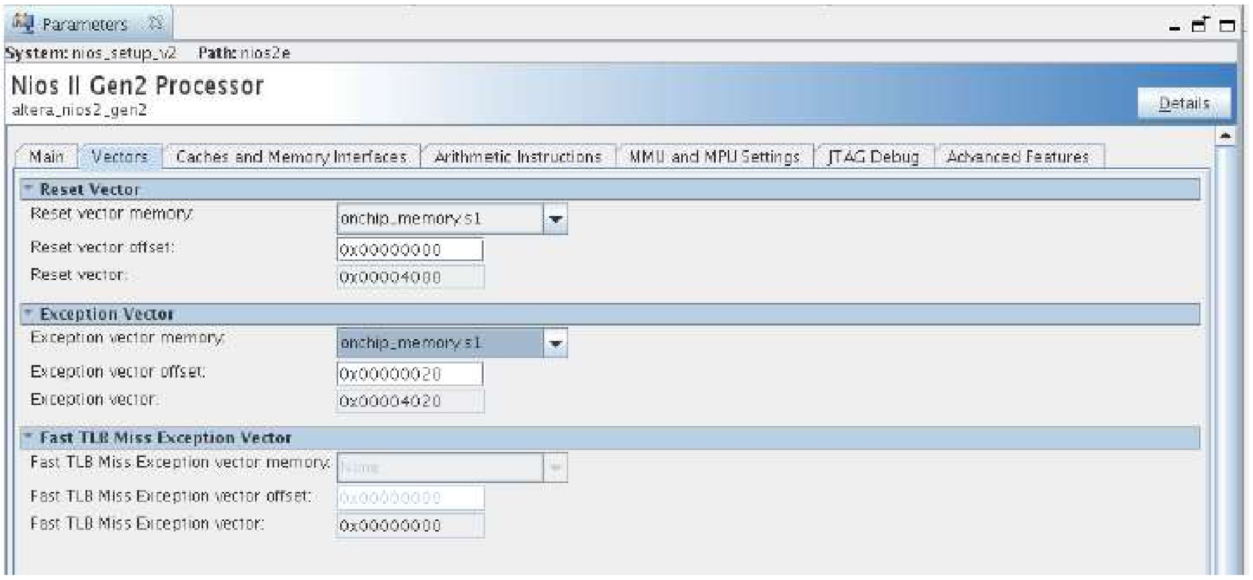
Messages

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 12: 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 straight forward 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 on-chip memory at these respective locations upon reset or interrupt. The 2 errors now resolved.



System: nios_setup_v2   Path: nios2e

Nios II Gen2 Processor
altera_nios2_gen2

Reset Vector

Reset vector memory:	onchip_memory.s1
Reset vector offset:	0x00000000
Reset vector:	0x00004000

Exception Vector

Exception vector memory:	onchip_memory.s1
Exception vector offset:	0x00000020
Exception vector:	0x00004020

Fast TLB Miss Exception Vector

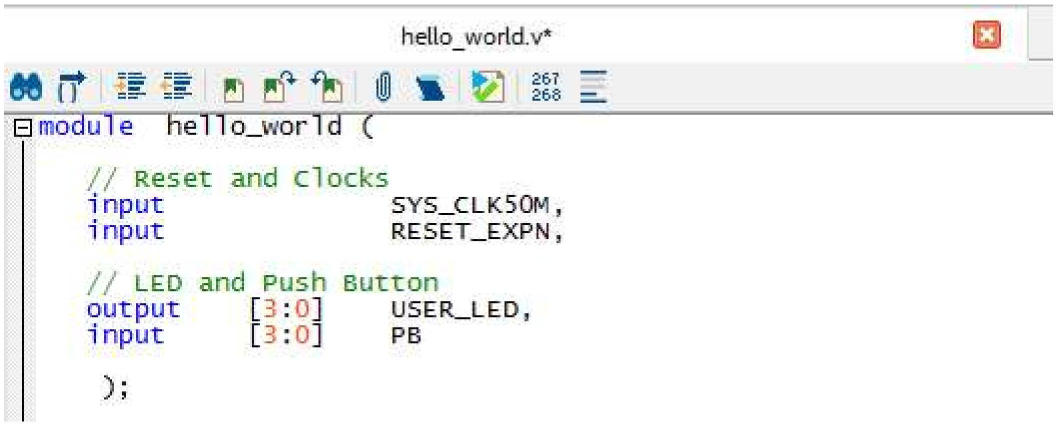
Fast TLB Miss Exception vector memory:	None
Fast TLB Miss Exception vector offset:	0x00000000
Fast TLB Miss Exception vector:	0x00000000

Figure 13: Assign vectors in the NIOS2E panel

Save the design once again. Note that by saving, you still have not generated the files that you need for Quartus 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.
This completes the Qsys section of the lab.

Building the top level design

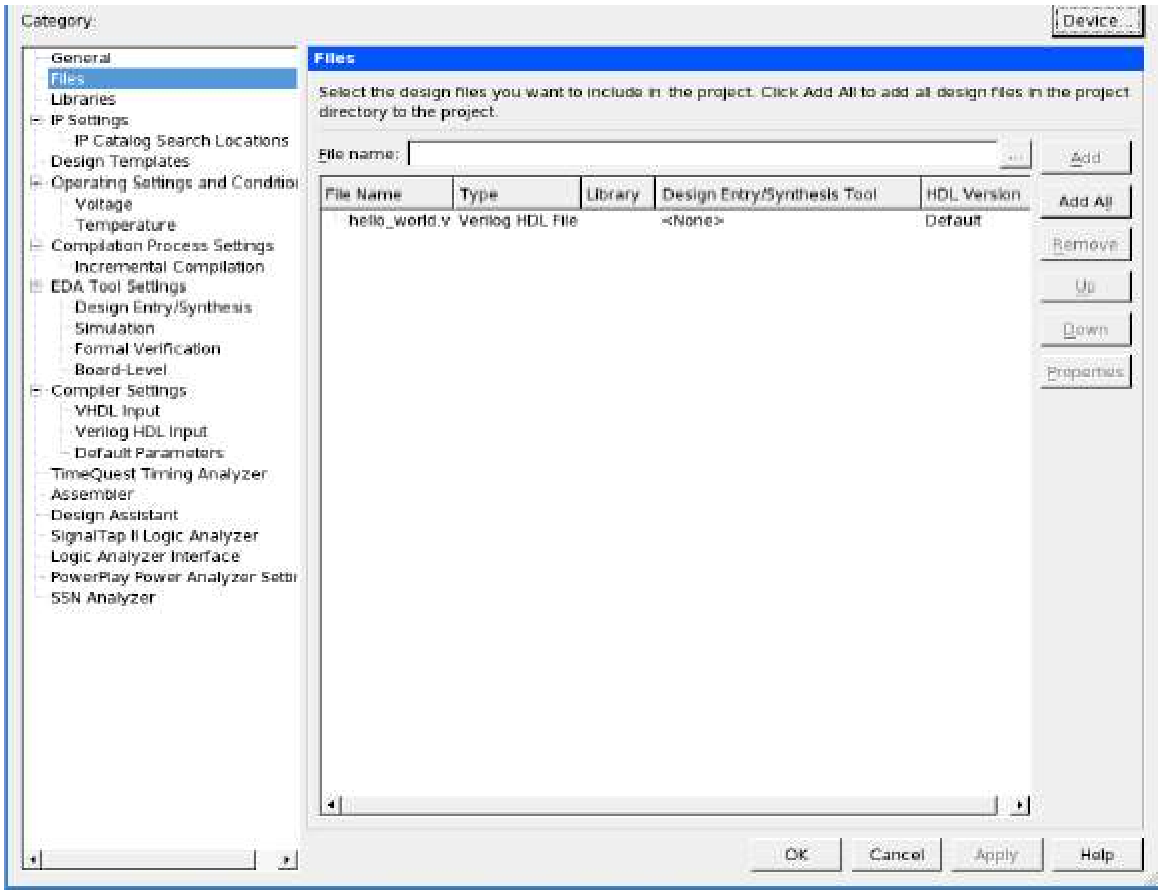
Create a file and save the file as hello_world.v



```
module  hello_world (

    // Reset and Clocks
    input               SYS_CLK50M,
    input               RESET_EXPN,

    // LED and Push Button
    output     [3:0]    USER_LED,
    input      [3:0]    PB

    );
```

Figure 14: Top level file hello_world.v

Include the hello_world.v file in the project. Go to Project Add/Remove Files in Project. Confirm that hello_world.v is listed.

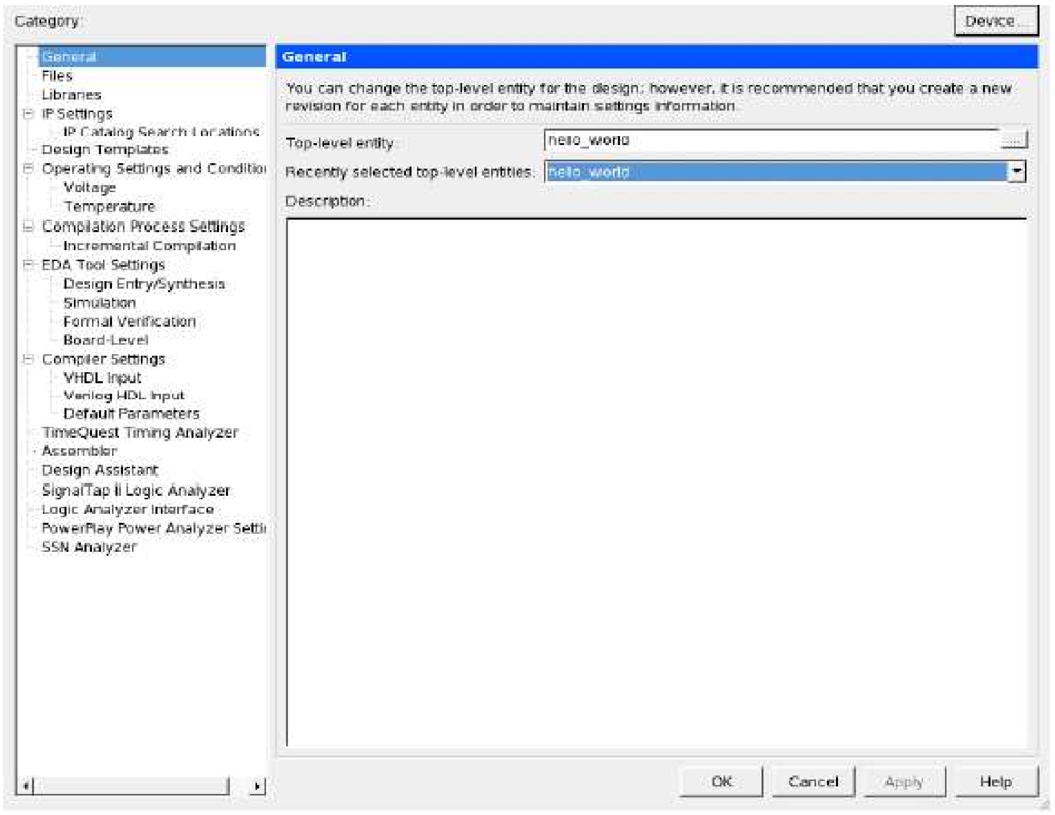


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version
hello_world.v	Verilog HDL File		<None>	Default

Figure 15: Add/Remove Files pane

Next, click on General, select the hello_world.v and set as top level entity.



General

You can change the top-level entity for the design; however, it is recommended that you create a new revision for each entity in order to maintain settings information.

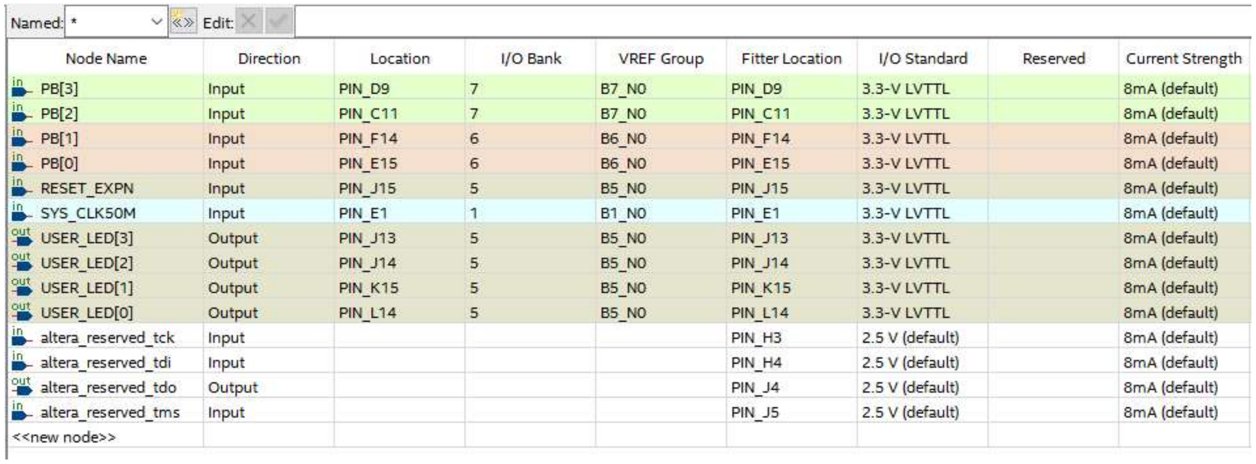
Top-level entity: hello_world

Recently selected top-level entities: hello_world

Description:

Figure 16: Settings pane

Launch Assignments □Pin Planner. Assign the pin locations, IO standard and current strength settings, as below:

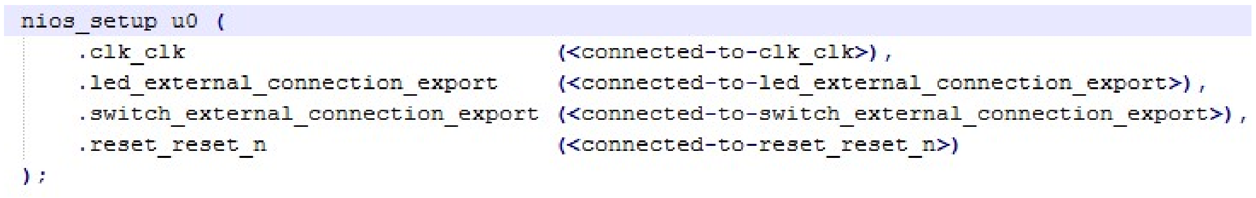


Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
PB[3]	Input	PIN_D9	7	B7_N0	PIN_D9	3.3-V LVTTL		8mA (default)
PB[2]	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)
PB[1]	Input	PIN_F14	6	B6_N0	PIN_F14	3.3-V LVTTL		8mA (default)
PB[0]	Input	PIN_E15	6	B6_N0	PIN_E15	3.3-V LVTTL		8mA (default)
RESET_EXPN	Input	PIN_J15	5	B5_N0	PIN_J15	3.3-V LVTTL		8mA (default)
SYS_CLK50M	Input	PIN_E1	1	B1_N0	PIN_E1	3.3-V LVTTL		8mA (default)
USER_LED[3]	Output	PIN_J13	5	B5_N0	PIN_J13	3.3-V LVTTL		8mA (default)
USER_LED[2]	Output	PIN_J14	5	B5_N0	PIN_J14	3.3-V LVTTL		8mA (default)
USER_LED[1]	Output	PIN_K15	5	B5_N0	PIN_K15	3.3-V LVTTL		8mA (default)
USER_LED[0]	Output	PIN_L14	5	B5_N0	PIN_L14	3.3-V LVTTL		8mA (default)
altera_reserved_tck	Input				PIN_H3	2.5 V (default)		8mA (default)
altera_reserved_tdi	Input				PIN_H4	2.5 V (default)		8mA (default)
altera_reserved_tdo	Output				PIN_J4	2.5 V (default)		8mA (default)
altera_reserved_tms	Input				PIN_J5	2.5 V (default)		8mA (default)
<<new node>>								

Figure 17: Pin Planner

Add the Nios II system into design

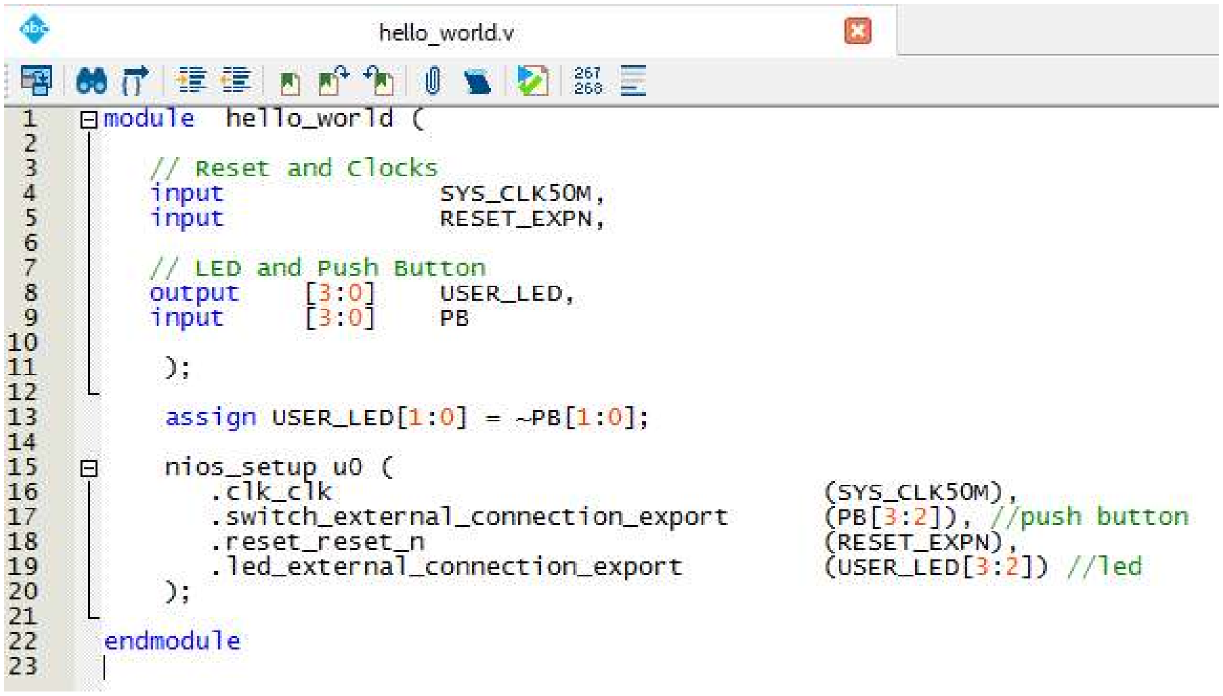
Now that you have the hello_world entity completed and syntactically correct, you will need to add the Nios II Qsys system into your design. Qsys makes this task quite convenient. Go to File □Open and navigate to the name of the Qsys project you created (the one shown in the lab is called nios2_setup). You should see a file called nios_setup_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 is shown below:



```
nios_setup u0 (
    .clk_clk                              (<connected-to-clk_clk>),
    .led_external_connection_export       (<connected-to-led_external_connection_export>),
    .switch_external_connection_export    (<connected-to-switch_external_connection_export>),
    .reset_reset_n                        (<connected-to-reset_reset_n>)
);
```

Figure 18: nios_setup

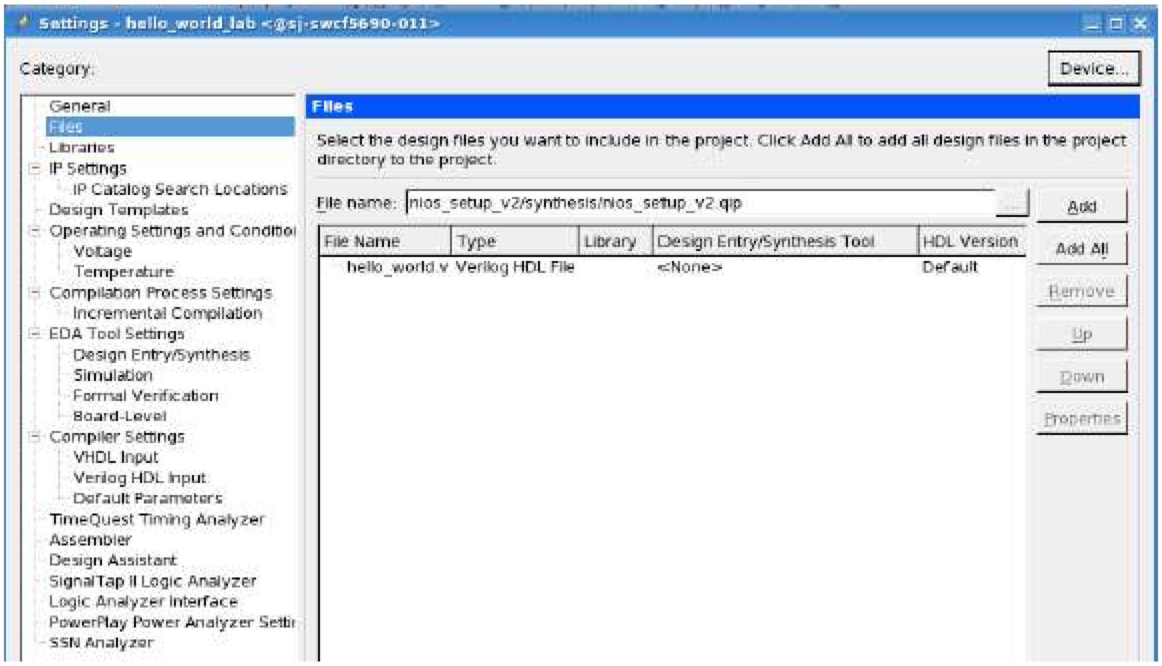
Connect the IO ports to the nios_setup_v2 by inserting into the Verilog file hello_world.v. Next, connect the push button switches to the LEDs in two different ways to demonstrate how the connection can be made through the FPGA fabric, and in the software that we use the Nios II executes. To simplify 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 and 1 turn *off* LEDs through a hardware connection, while activating push buttons 3 and 2 turn *on* the respective LEDs. Take two push buttons ([1:0]) and connect to the LEDs[1:0] with an inverted assignment. The other two LEDs [3:2] will be connected non-inverted in software by connection to the Qsys system.



```
module  hello_world (

    // Reset and Clocks
    input               SYS_CLK50M,
    input               RESET_EXPN,

    // LED and Push Button
    output     [3:0]    USER_LED,
    input      [3:0]    PB

     );

     assign USER_LED[1:0] = ~PB[1:0];

     nios_setup u0 (
        .clk_clk                                  (SYS_CLK50M),
        .switch_external_connection_export        (PB[3:2]), //push button
        .reset_reset_n                            (RESET_EXPN),
        .led_external_connection_export           (USER_LED[3:2]) //led
     );

endmodule
```

Figure 19: hello_world.v after making connections to Nios II 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/synthesis/nios_setup.qip . The qip file contains pointers to the location of all the generated source files generated from Qsys and necessary settings required to compile.


Settings - hello_world_lab <@sj-swcf5690-011>

Category:

Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

File name: nios_setup_v2/synthesis/nios_setup_v2.qip

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version
hello_world.v	Verilog HDL File		<None>	Default

Figure 20: Add/Remove Files from Project .qip file

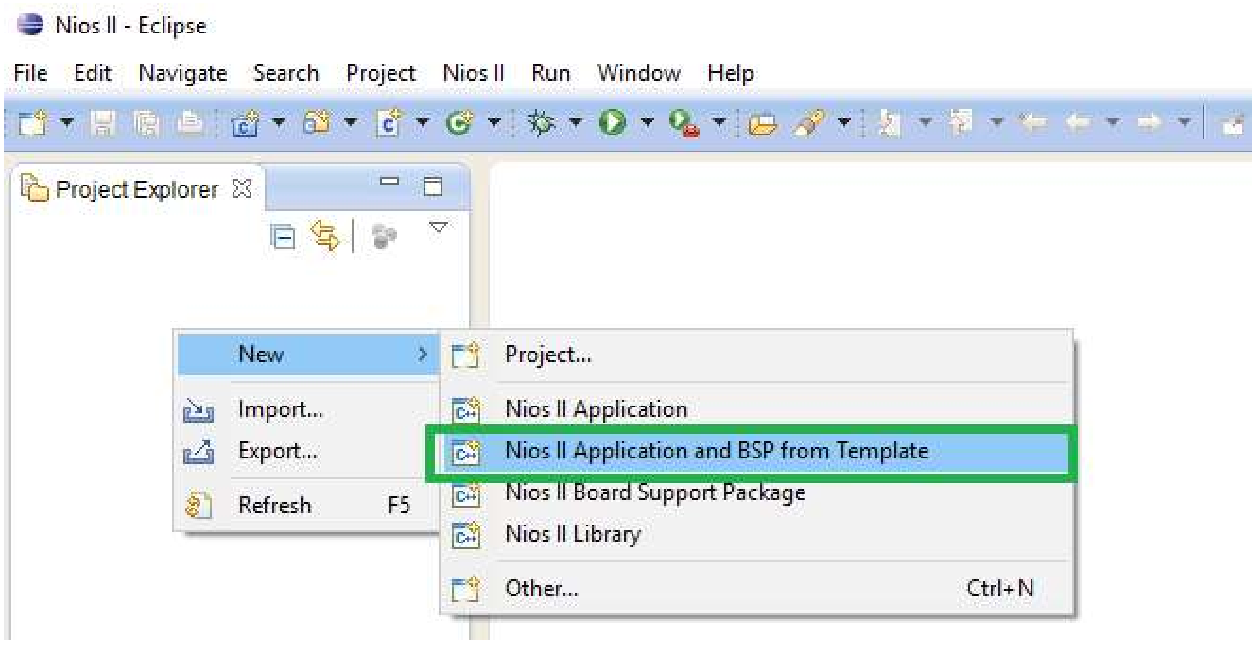
Now you can compile the design which will run analysis & synthesis, fitter (place and route in FPGA terminology), Assembler (generate programming image) and TimeQuest (the static timing analyzer). This can be achieved by clicking on the ‘Start Compilation’ button.

SOFTWARE DESIGN

Creating the Software for the “Hello World” design

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. You can use the default location that Eclipse picks for you.

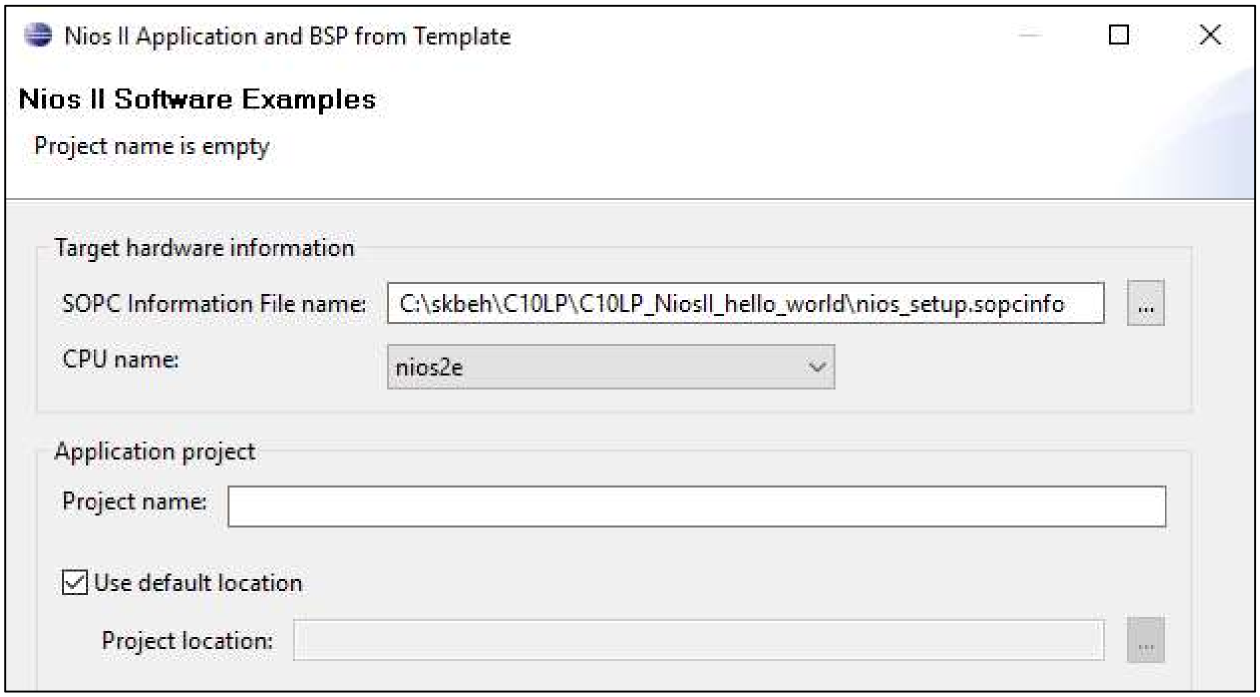
Click OK in the Workspace launcher.
Next, the Eclipse SBT will launch. Right click in the area called Project Explorer and select New→Nios II Application and BSP from Template. The BSP is the “Board Support Package” that contains the drivers for things like translating printf C commands to the appropriate instructions to write to the terminal.



Nios II - Eclipse
File Edit Navigate Search Project Nios II Run Window Help
Project Explorer
New
Import...
Export...
Refresh F5
Project...
Nios II Application
Nios II Application and BSP from Template
Nios II Board Support Package
Nios II Library
Other... Ctrl+N

Figure 21: 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 trains Eclipse on what your Qsys system contains. Click OK.


Nios II Application and BSP from Template

Nios II Software Examples

Project name is empty

Target hardware information

SOPC Information File name: C:\skbeh\C10LP\C10LP_NiosII_hello_world\nios_setup.sopcinfo

CPU name: nios2e

Application project

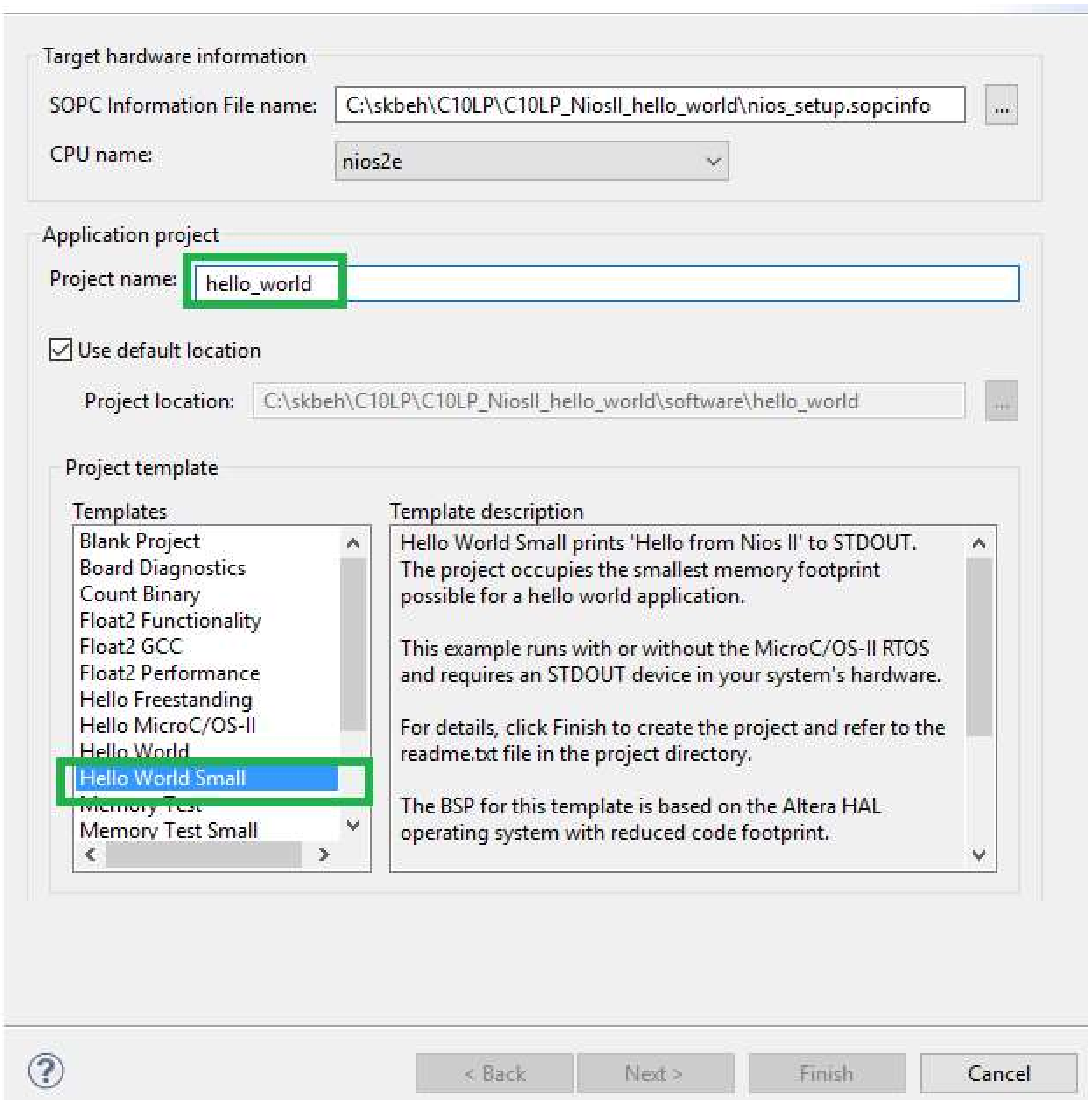
Project name:

☑ Use default location

Project location:

Figure 22: Navigating to correct .sopcinfo file

Fill in the Project name. Call it hello_world. 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.



Target hardware information

SOPC Information File name: C:\skbeh\C10LP\C10LP_NiosII_hello_world\nios_setup.sopcinfo

CPU name: nios2e

Application project

Project name: hello_world

☑ Use default location

Project location: C:\skbeh\C10LP\C10LP_NiosII_hello_world\software\hello_world

Project template

Templates: Blank Project, Board Diagnostics, Count Binary, Float2 Functionality, Float2 GCC, Float2 Performance, Hello Freestanding, Hello MicroC/OS-II, Hello World, **Hello World Small**, Memory Test, Memory Test Small

Template description:

Hello World Small prints 'Hello from Nios II' to STDOUT. The project occupies the smallest memory footprint possible for a hello world application.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

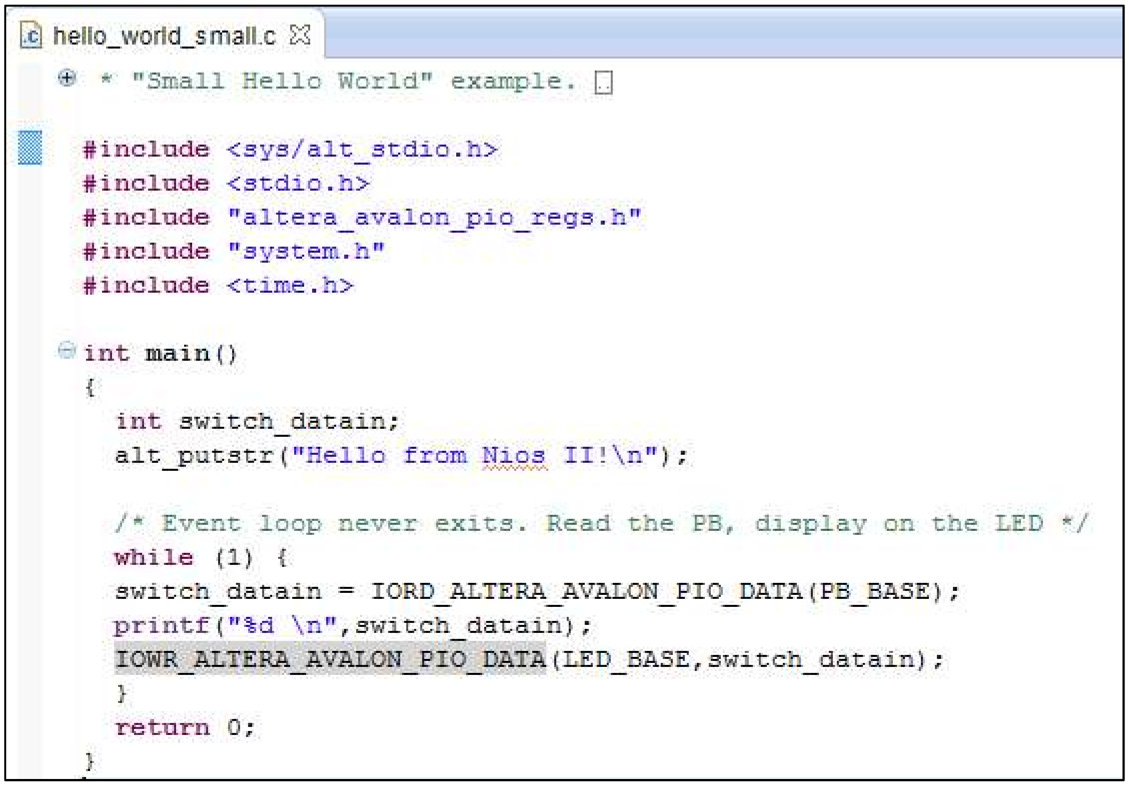
The BSP for this template is based on the Altera HAL operating system with reduced code footprint.

Figure 23: completing the Nios II Software Examples setup screen with project name and project template.

Now, make some modifications to the code to connect the LEDs to the push button switches through software. Click the right arrow next to hello_world. 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. 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

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"
#include <time.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(PB_BASE);
  printf("%d \n",switch_datain);
  IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,switch_datain);
  }
  return 0;
}
```



```
⊕ * "Small Hello World" example. 

#include <sys/alt_stdio.h>
#include <stdio.h>
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include <time.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(PB_BASE);
  printf("%d \n",switch_datain);
  IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,switch_datain);
  }
  return 0;
}
```

Figure 24: Nios II Software hello_world_small.c coding after modification.

Right click on the hello_world project. Left click Build. This compiles the software application and the BSP (drivers).

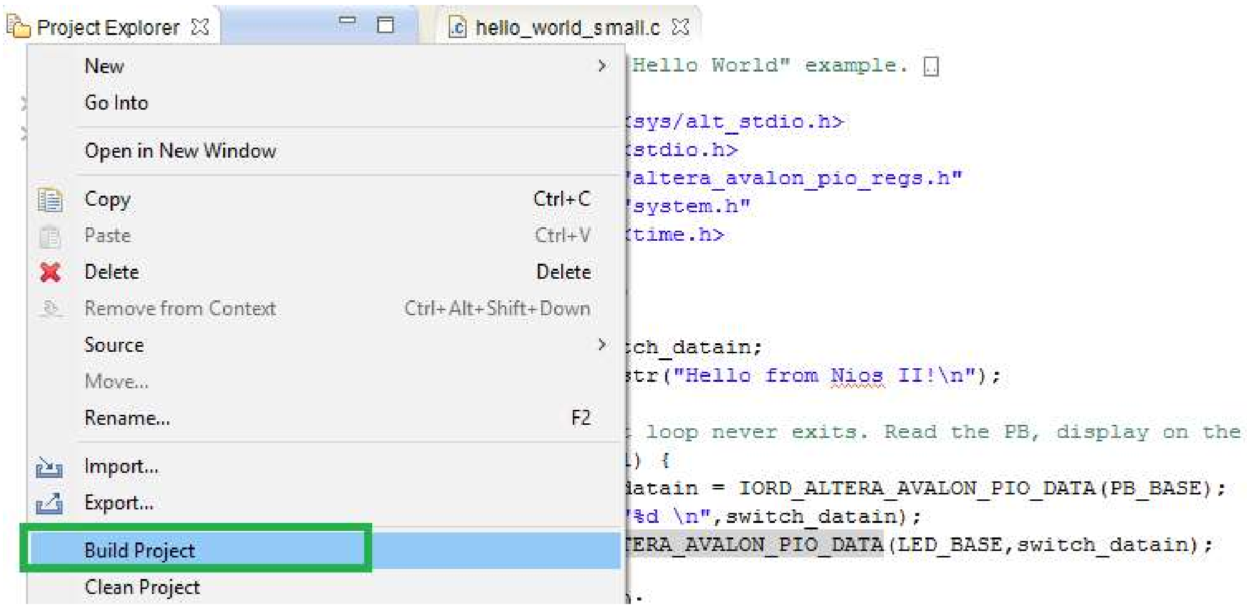


Figure 25: Launching the Build Project

Downloading the hardware into Cyclone 10 LP Development Kit

To work with the Cyclone 10 LP development kit in the context of this lab, you just need to connect the USB blaster from PC to the dev kit. The development kit will be power-up by the USB Blaster cable itself from PC (no extra power supply needed).

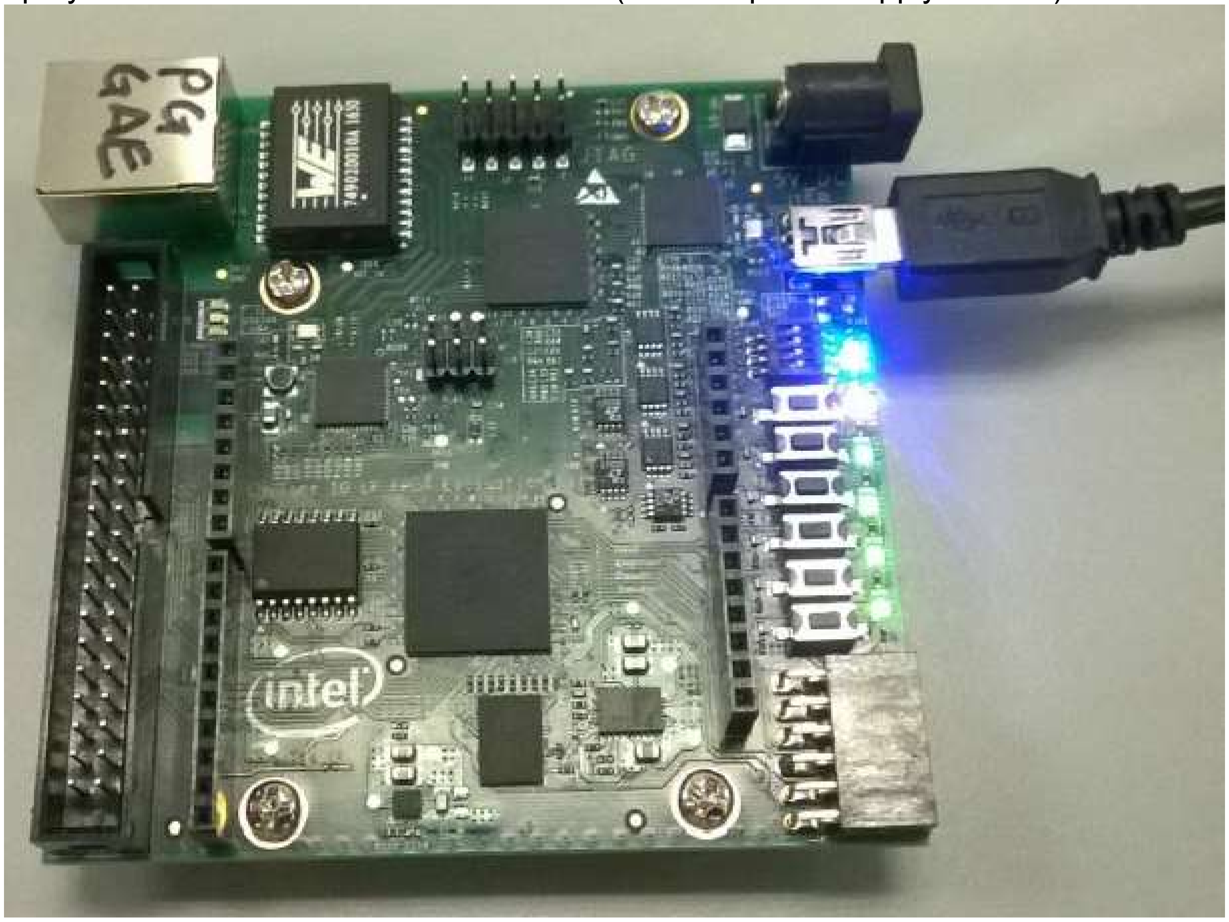
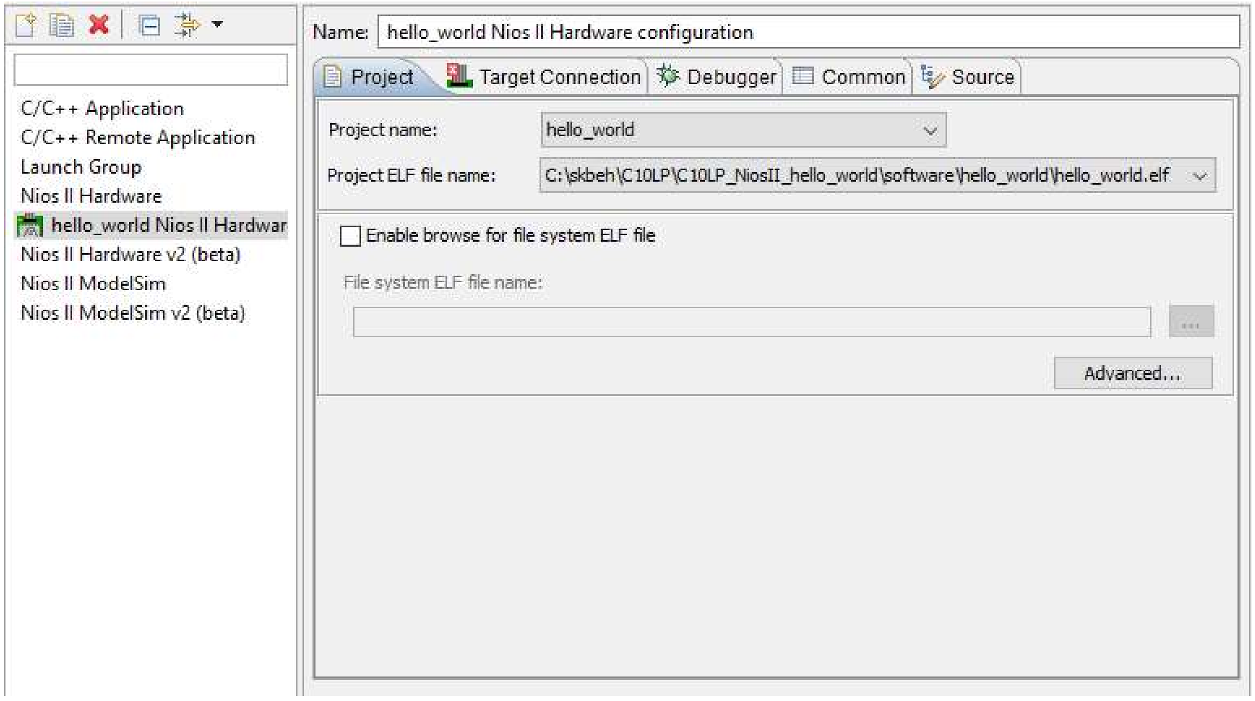


Figure 26: Cyclone 10 LP Development Kit connect to USB Blaster cable from PC.

In Quartus, launch the Tools →Programmer.
Click Auto Detect and make sure the USB Blaster is detected before download the ".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.
Click Start. When programming is complete, the Progress meter should read 100% (Successful).

Run the Nios II Software

Download the .elf (software executable) into the Nios IIe 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.



Name: hello_world Nios II Hardware configuration

Project name: hello_world

Project ELF file name: C:\skbeh\C10LP\C10LP_NiosII_hello_world\software\hello_world\hello_world.elf

Enable browse for file system ELF file

File system ELF file name:

Figure 27: Run (Nios II) Configurations Window in Eclipse.

Now you have hardware and software downloaded into the Cyclone 10 LP Development Kit. You should observe “Hello from Nios II” in the Nios II Console tab.
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. Press the push buttons to operate the LEDs based on the hello_world.v and hello_world_small.c source files. Be careful not to hit the push buttons CONFIG or RESET or you can disrupt the FPGA programmed status.
(Note: By default, Nios II Console tab will continuously print out the value of USER_LED[3:2]=0x3).
Press PB0: USER_LED[0] will Off
Press PB1: USER_LED[1] will Off
Press PB2: USER_LED[3:2] value is 0x2
Press PB3: USER_LED[3:2] value is 0x1
Press both PB2 and PB3 simultaneously: USER_LED[3:2] value is 0x0

Summary

The hardware and software sections of this lab includes:
1. Using Qsys to build a Nios II based system
2. Instantiating the Qsys component into your top level design
3. Add some connections between push buttons and LEDs
4. Compiling your hardware
5. Importing the Nios II based system into the Eclipse Software Build Tools
6. Building a software project
7. Modifying a software template to perform some simple IO functions
8. Downloading the hardware image into the Cyclone 10 LP Development Kit
9. Downloading the software executable into the Cyclone 10 LP Development Kit
10. Testing the hardware

Revision and History

Date	Version	Changes
8/25/2017	1.0	Initial Release