EG3205 Part II. Multiprocessor and Multicore Systems

Laboratory Exercise 4

Contents

Lab 4 Introduction

The purpose of Lab 4 is to create a two-node distributed time-triggered (TT) system. Each node should contain a Nios II based processor. The system should be based on a **Shared-Clock TT scheduler**. The two processors should communicate over a CAN bus network. CAN-SPI modules should be used to implement the system.

Lab 4 Description

In Lab 3 you completed the experimental setup for a two-node distributed system. In this exercise, your goal is to work on the same hardware setup and implement the software of a time-triggered **Shared-Clock scheduler** to connect a single-core master node and a single-core slave node over a CAN bus. The two-node system will then be used to perform some simple tasks.

Task 1

To begin with, you should develop the hardware set-up that you created at the end of Lab 3:

- The Nios II processor configuration on each board should be designed as illustrated in Fig. 1. Section "Creating your Qsys Design" below provides a step-by-step guide to the design process.
- The boards should be connected as shown in Fig 2.

Once completed move on to Task 2.

Task 2

Your **second task** is to use the provided Shared-Clock TTC scheduler code and apply the software to the hardware as required to create a shared-clock network. To achieve this, you are provided with the code for the master and slave software applications (`app_master.zip` & `app_slave.zip`), as well as for a simple SPICAN driver communicating with the MCP2515 CAN Controller (`SPICAN_Driver.zip`).

Chapter 28 of the PTTES book provides information about shared-clock schedulers using CAN. You can use the SC-CAN implementation for an Infineon C515c microcontroller (page 686) as a starting point.

A step-by-step guide to the software implementation is provided in section “Implementing Shared-Clock Scheduling” below.

You are expected to demonstrate a successful system operation on the hardware setup showing the following outcomes:

1) A heartbeat LED blinking continuously on each board as follows:
   a) LED2 - on Master Board,
   b) LED3 - on Slave Board.
2) When Button 0 on the master board is pressed, LED 0 on the slave board blinks.

Task 3

Your **third task** is to analyse the code and understand the functions performed by the scheduler in making the application run as detailed in section “Demonstrate Understanding” below.

Upload a brief report with a summary of your findings to Blackboard.

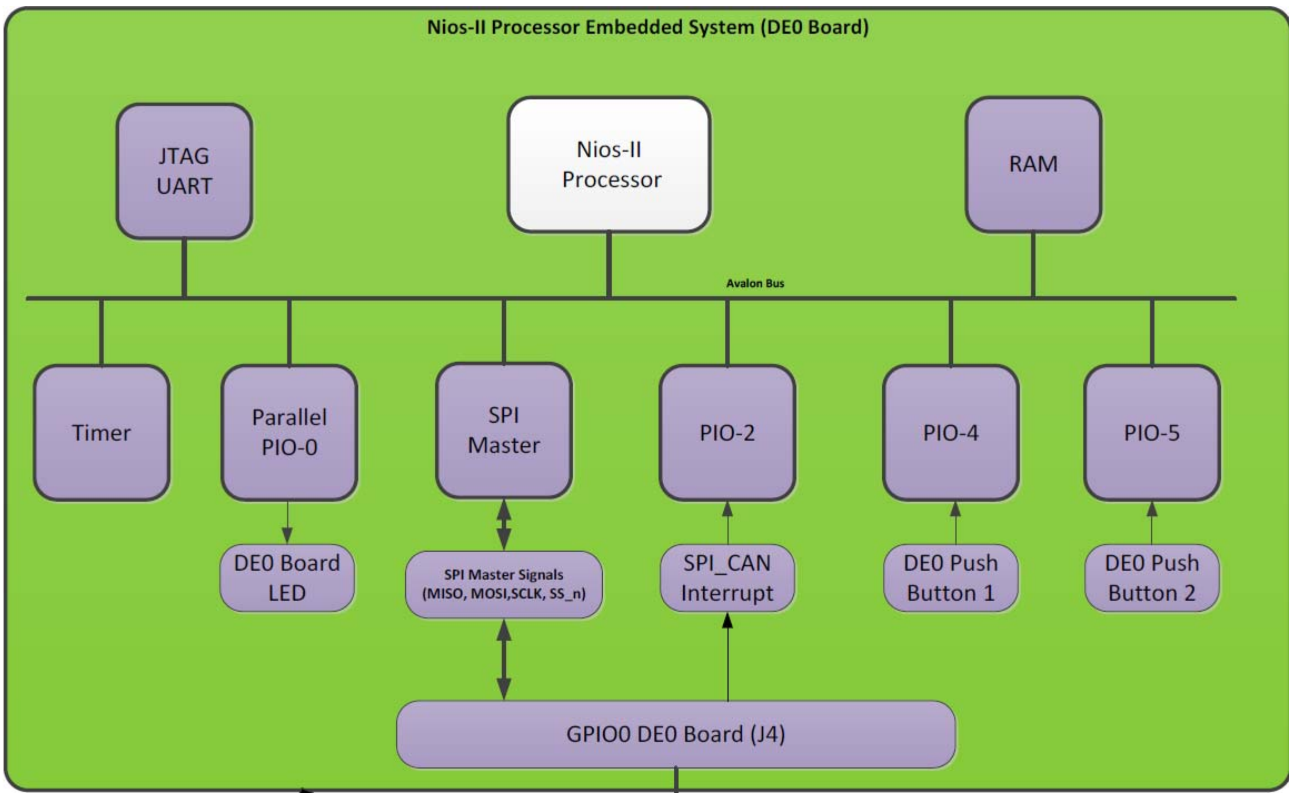


Figure 1

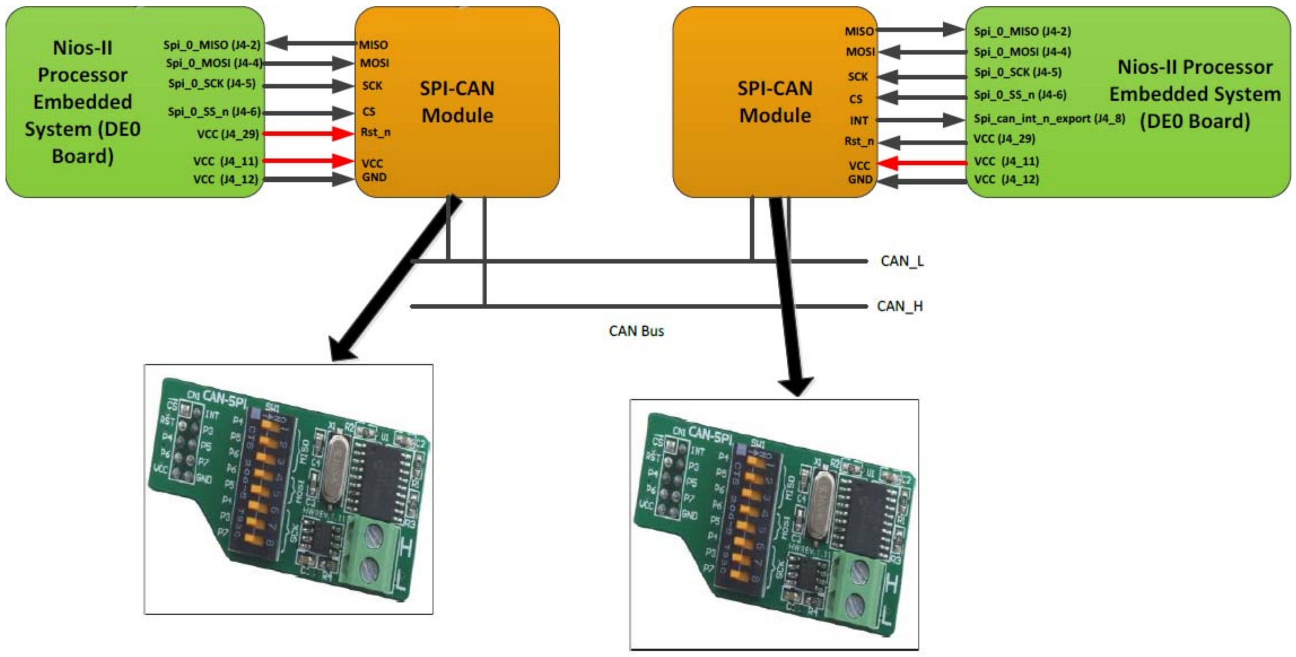


Figure 2

A communications diagram illustrating the Shared-Clock Scheduler is provided in an Appendix.

<u>Files provided</u>

```
SPICAN_Driver.zip
app_master.zip
app_slave.zip
```

<u>Acknowledgements:</u>

The first version of this exercise was created by Dr Keith Athaide (March 2012).
The second version of this exercise was developed by Dr. M. Fayyaz (June 2015).
This version of the exercise was developed by Sam Kennedy (November 2018).

Creating your Qsys Design

Quartus IDE

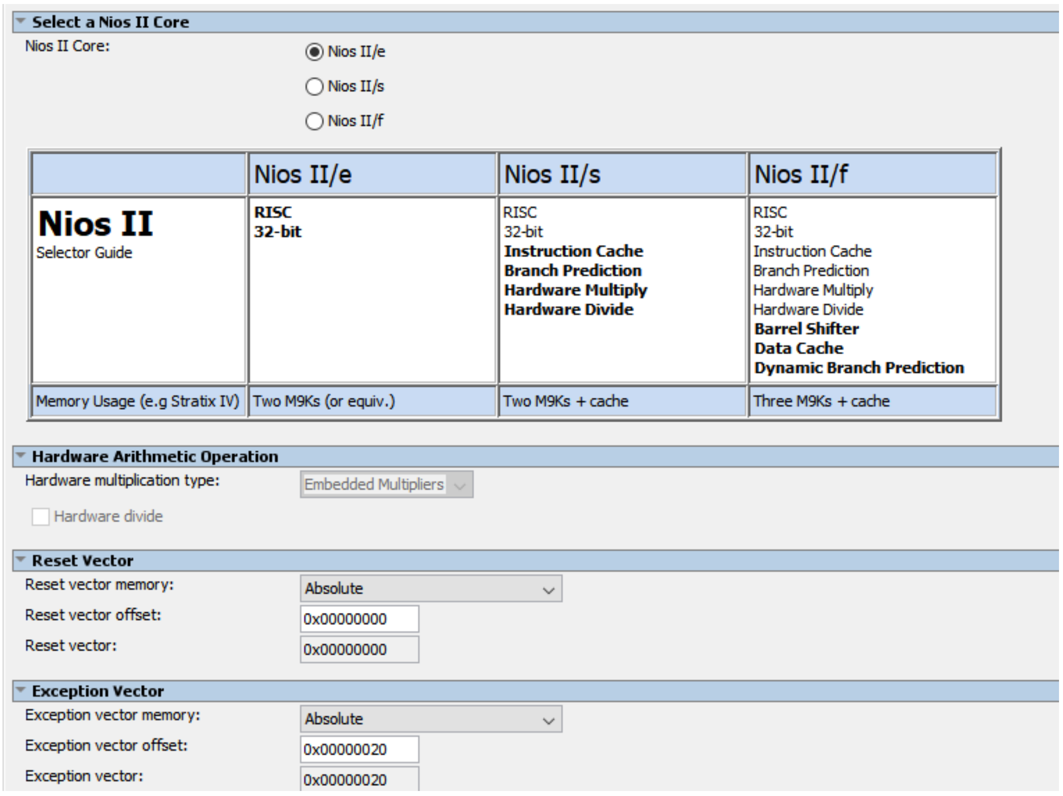
Create a new folder in your C drive area named Lab_4.

Please ensure that all directories you use are pointing to the correct folder as this is a common source of errors.

Open Quartus II (IDE) v.13.1. Make a new project and save it to Lab_4. Open Qsys - under the Tools menu.

1. Add a Nios II Processor in the same way as before

Qsys



Select a Nios II Core

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

	Nios II/e	Nios II/s	Nios II/f
Nios II Selector Guide	**RISC** **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Memory Usage (e.g Stratix IV)	Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache

Hardware Arithmetic Operation

Hardware multiplication type: Embedded Multipliers

Hardware divide: ☐

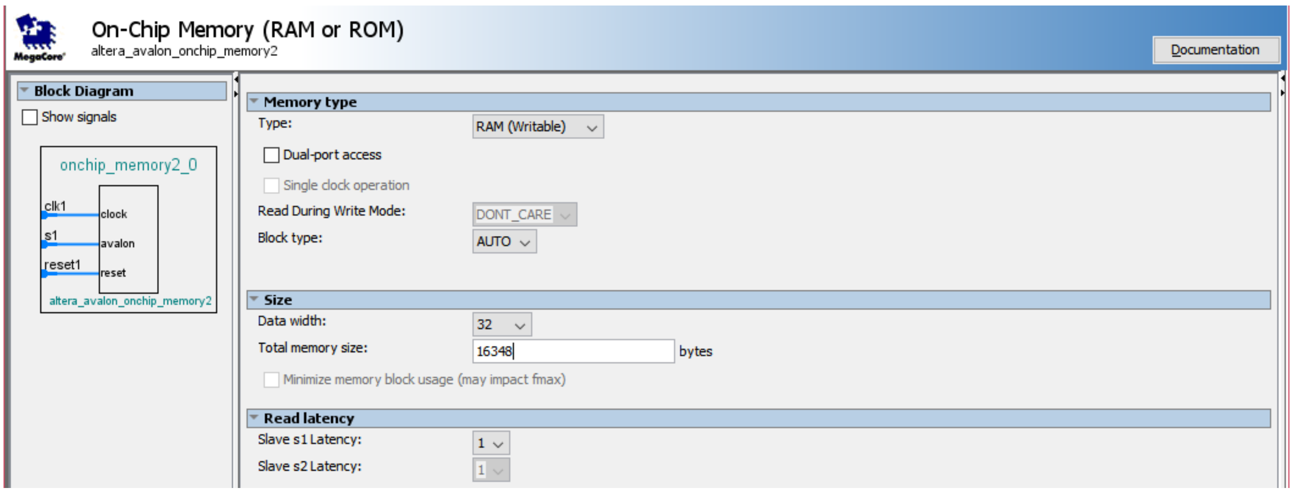
Reset Vector

Reset vector memory: Absolute
Reset vector offset: 0x00000000
Reset vector: 0x00000000

Exception Vector

Exception vector memory: Absolute
Exception vector offset: 0x00000020
Exception vector: 0x00000020

2. Add on-chip memory



On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Memory type

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

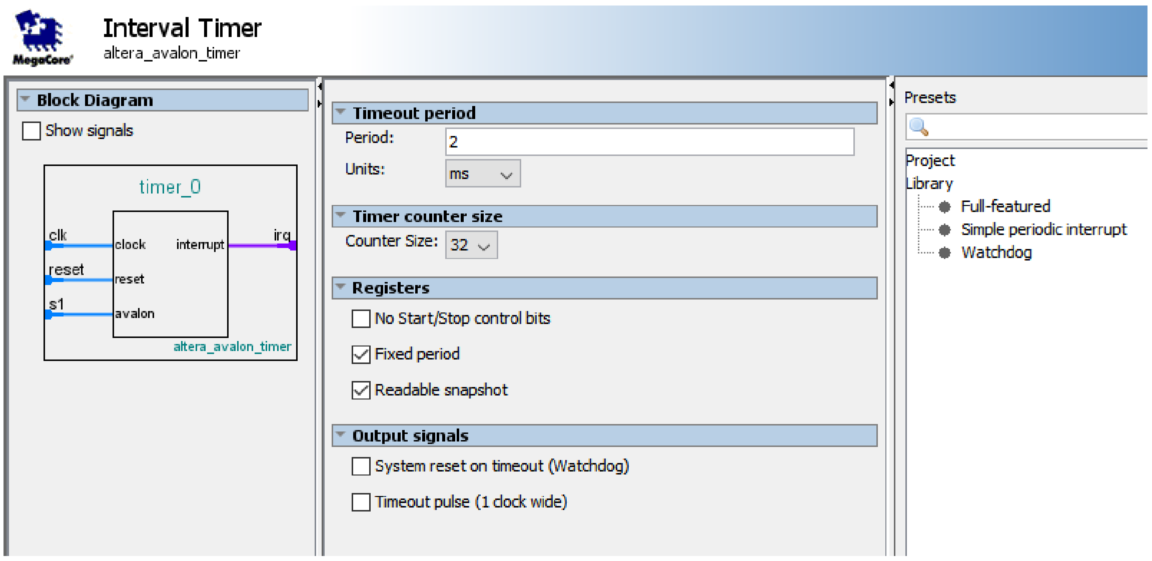
Size

Data width: 32
Total memory size: 16348 bytes
Minimize memory block usage (may impact fmax): ☐

Read latency

Slave s1 Latency: 1
Slave s2 Latency: 1

3. Add an interval timer as below:


Interval Timer
altera_avalon_timer

Block Diagram
☐ Show signals

Timeout period
Period: 2
Units: ms

Timer counter size
Counter Size: 32

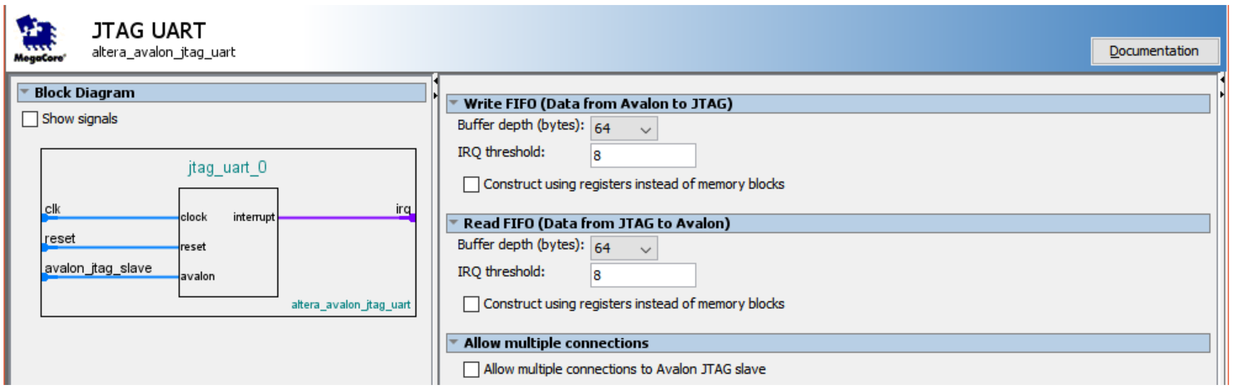
Registers
☐ No Start/Stop control bits
☑ Fixed period
☑ Readable snapshot

Output signals
☐ System reset on timeout (Watchdog)
☐ Timeout pulse (1 clock wide)

Presets
Project
Library
- Full-featured
- Simple periodic interrupt
- Watchdog

4. Add a JTAG-UART with default settings.

   Joint Test Action Group (JTAG) Universal Asynchronous Receiver-transmitter (UART) is an industry standard hardware device for asynchronous serial communication.
   The JTAG UART is used as part of integrated circuits such as FPGAs for communication with a computer.


JTAG UART
altera_avalon_jtag_uart

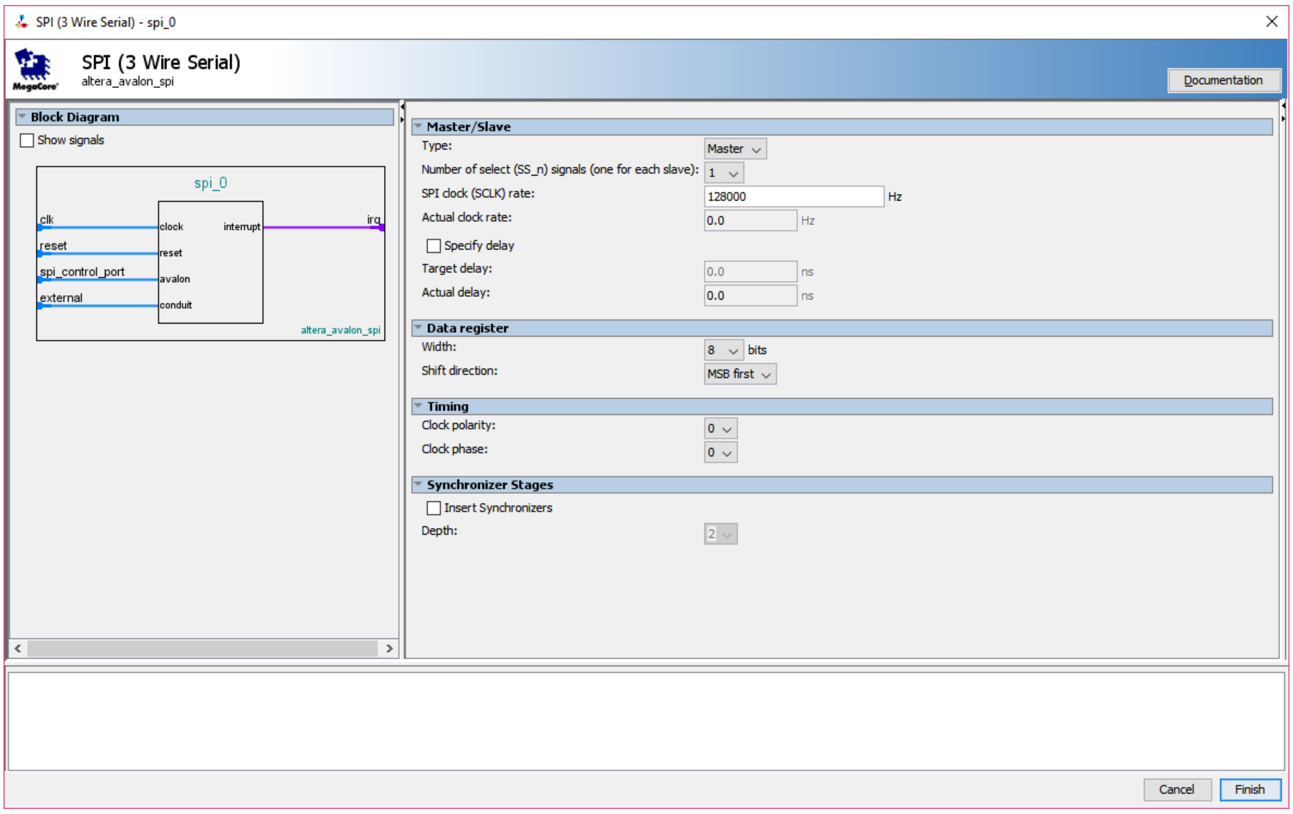
Block Diagram
☐ Show signals

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

Allow multiple connections
☐ Allow multiple connections to Avalon JTAG slave

5. Add an SPI 3 (Serial Peripheral Interface) Master peripheral with default settings.


SPI (3 Wire Serial) - spi_0

SPI (3 Wire Serial)
altera_avalon_spi

Master/Slave

Type: Master
Number of select (SS_n) signals (one for each slave): 1
SPI clock (SCLK) rate: 128000 Hz
Actual clock rate: 0.0 Hz
☐ Specify delay
Target delay: 0.0 ns
Actual delay: 0.0 ns

Data register

Width: 8 bits
Shift direction: MSB first

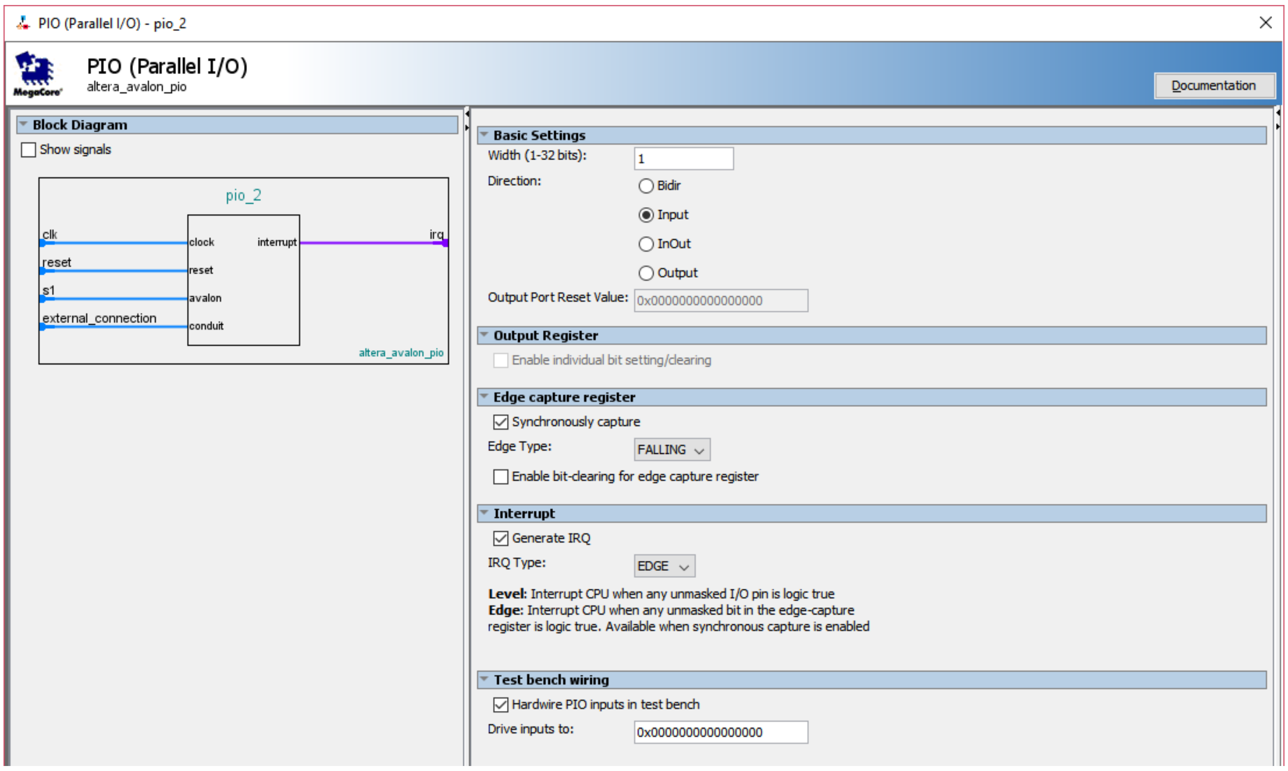
Timing

Clock polarity: 0
Clock phase: 0

Synchronizer Stages

☐ Insert Synchronizers
Depth: 2

6. Add a PIO (Parallel Input/Output) for the SPI Interrupt.


PIO (Parallel I/O) - pio_2

PIO (Parallel I/O)
altera_avalon_pio

Basic Settings

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

Output Register

☐ Enable individual bit setting/clearing

Edge capture register

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

Interrupt

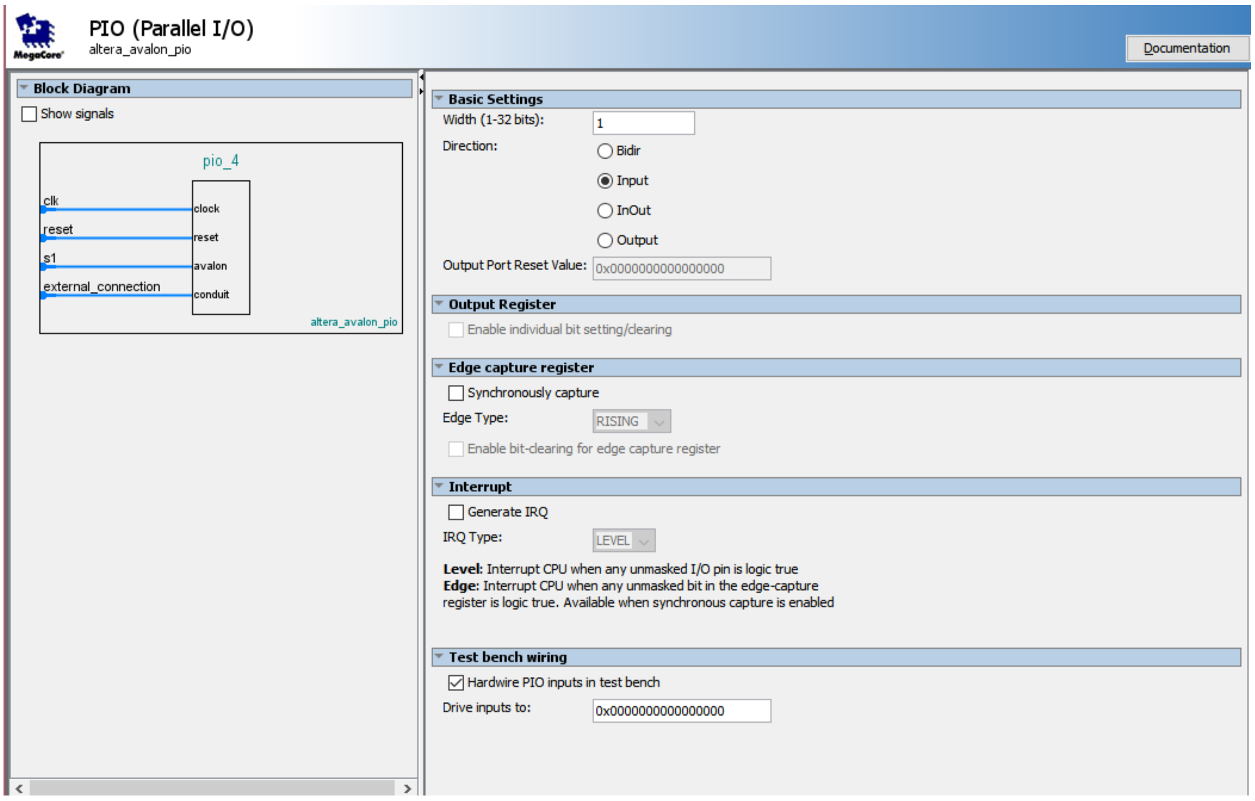
☑ Generate IRQ
IRQ Type: EDGE

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: 0x0000000000000000

7. Add 2 Parallel I/Os for the Push Buttons on the board (one for each of the two we will be using).



PIO (Parallel I/O)
altera_avalon_pio

Block Diagram
☐ Show signals

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

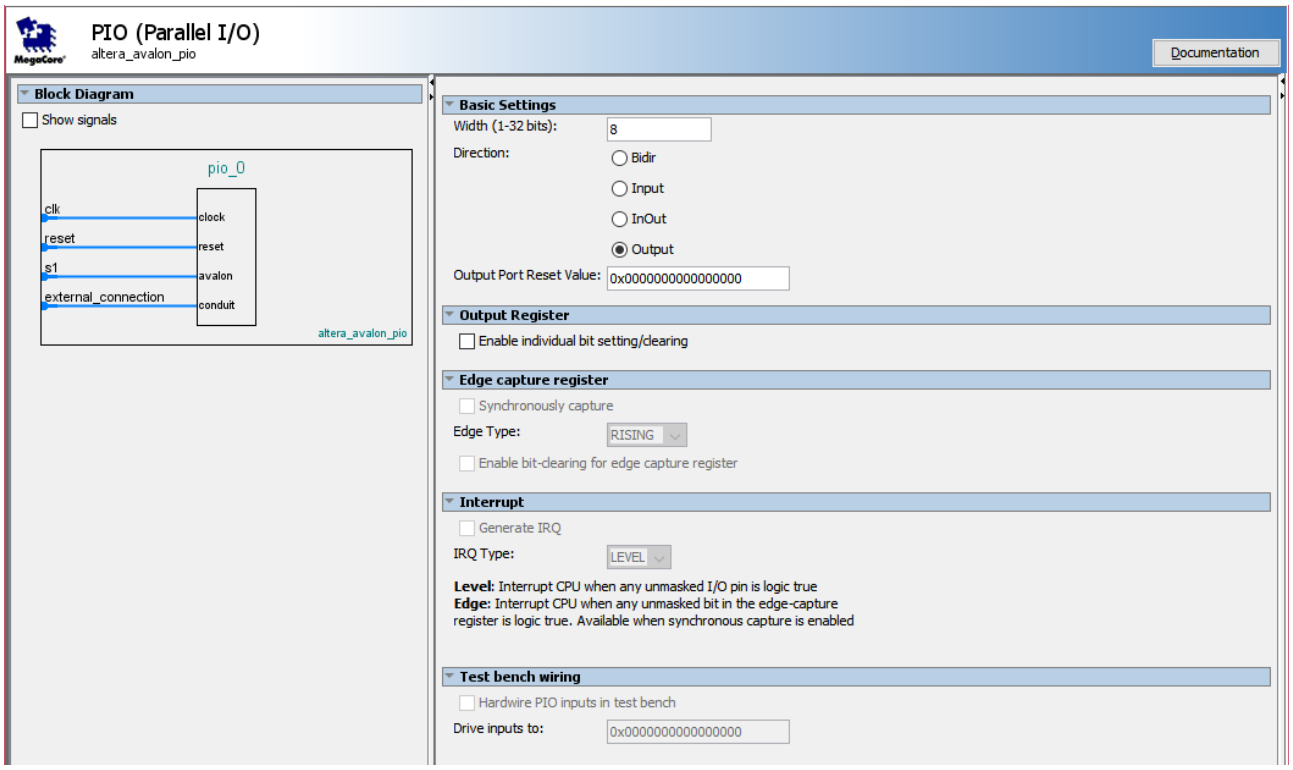
Output Register
☐ Enable individual bit setting/clearing

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: 0x0000000000000000

8. Add a Parallel I/O for the LEDs.



PIO (Parallel I/O)
altera_avalon_pio

Block Diagram
☐ Show signals

Basic Settings
Width (1-32 bits): 8
Direction: ○ Bidir ○ Input ○ InOut [✓ Output]
Output Port Reset Value: 0x0000000000000000

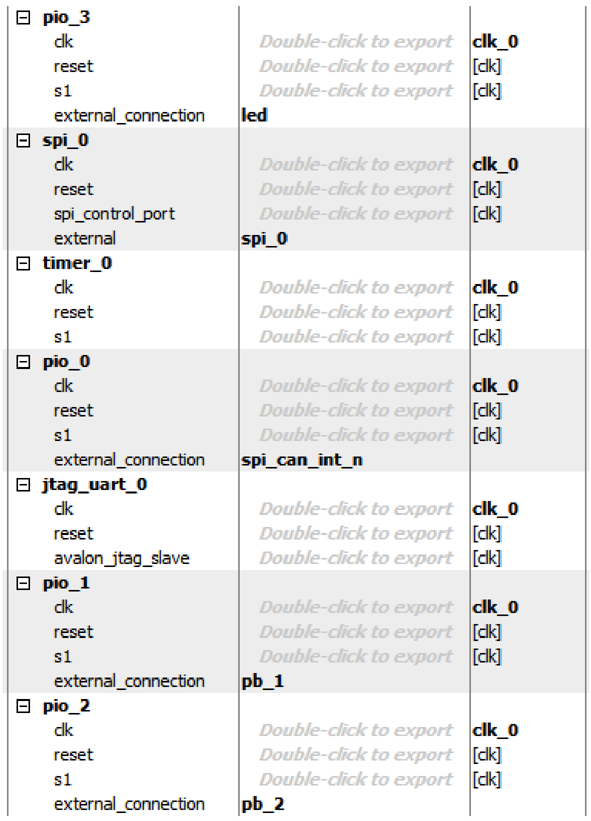
Output Register
☐ Enable individual bit setting/clearing

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

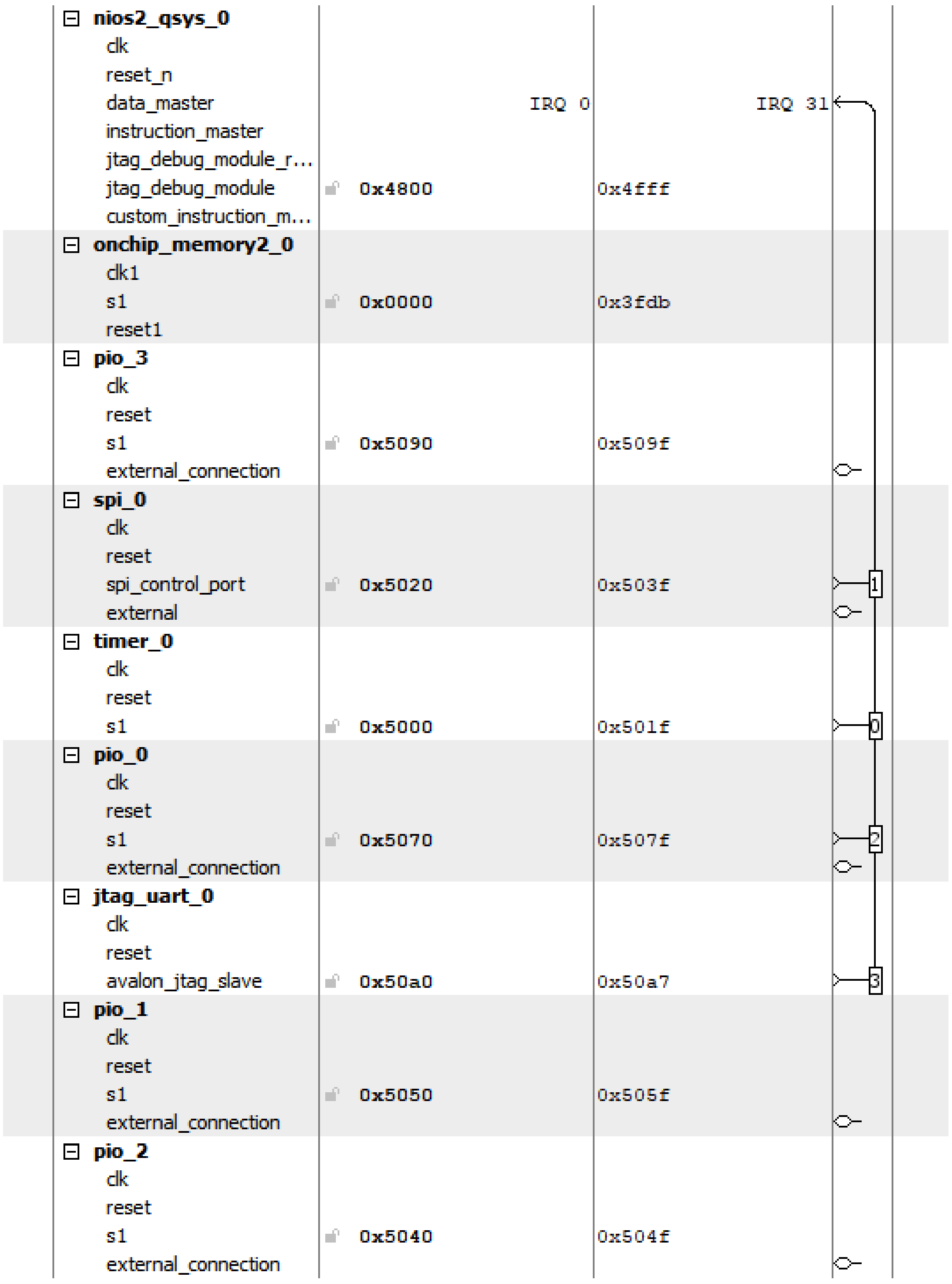
Test bench wiring
☐ Hardwire PIO inputs in test bench
Drive inputs to: 0x0000000000000000

9. Export the Conduits of the PIOs and SPI as below by double clicking in the export column and typing a name as below.



⊟ **pio_3**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
s1	Double-click to export	[clk]
external_connection	**led**	
⊟ **spi_0**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
spi_control_port	Double-click to export	[clk]
external	**spi_0**	
⊟ **timer_0**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
s1	Double-click to export	[clk]
⊟ **pio_0**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
s1	Double-click to export	[clk]
external_connection	**spi_can_int_n**	
⊟ **jtag_uart_0**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
avalon_jtag_slave	Double-click to export	[clk]
⊟ **pio_1**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
s1	Double-click to export	[clk]
external_connection	**pb_1**	
⊟ **pio_2**		
clk	Double-click to export	**clk_0**
reset	Double-click to export	[clk]
s1	Double-click to export	[clk]
external_connection	**pb_2**	

10. Assign base addresses and export IRQs as seen here. Ensure that the Base and End addresses are assigned to the correct components.



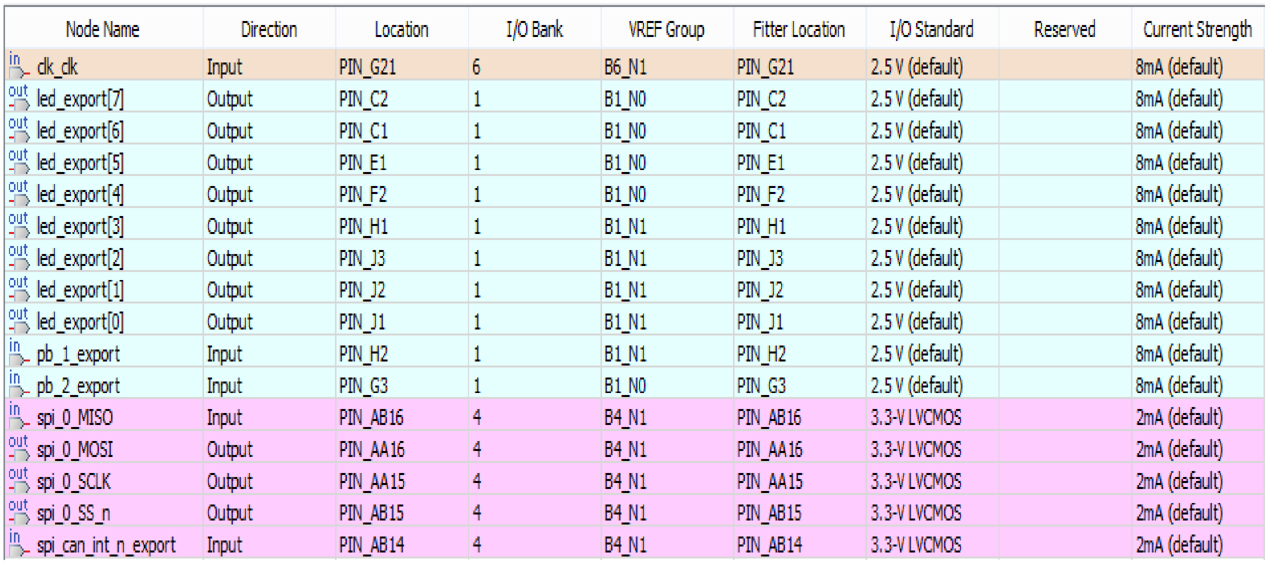
⊟ **nios2_qsys_0**			
clk			
reset_n			
data_master		IRQ 0	IRQ 31
instruction_master			
jtag_debug_module_r...			
jtag_debug_module	0x4800	0x4fff	
custom_instruction_m...			
⊟ **onchip_memory2_0**			
clk1			
s1	0x0000	0x3fdb	
reset1			
⊟ **pio_3**			
clk			
reset			
s1	0x5090	0x509f	
external_connection			
⊟ **spi_0**			
clk			
reset			
spi_control_port	0x5020	0x503f	1
external			
⊟ **timer_0**			
clk			
reset			
s1	0x5000	0x501f	0
⊟ **pio_0**			
clk			
reset			
s1	0x5070	0x507f	2
external_connection			
⊟ **jtag_uart_0**			
clk			
reset			
avalon_jtag_slave	0x50a0	0x50a7	3
⊟ **pio_1**			
clk			
reset			
s1	0x5050	0x505f	
external_connection			
⊟ **pio_2**			
clk			
reset			
s1	0x5040	0x504f	
external_connection			

11. Once you are sure the design in complete (with no errors) generate from the Generate menu with standard settings and exit from Qsys back to the Quartus IDE.

Implementing the Hardware Design

Quartus IDE

1. Open the Pin Planner tool in Quartus and complete the connections as seen below, **make sure to alter the I/O Standard voltage for the SPI I/Os**.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
clk_clk	Input	PIN_G21	6	B6_N1	PIN_G21	2.5 V (default)		8mA (default)
led_export[7]	Output	PIN_C2	1	B1_N0	PIN_C2	2.5 V (default)		8mA (default)
led_export[6]	Output	PIN_C1	1	B1_N0	PIN_C1	2.5 V (default)		8mA (default)
led_export[5]	Output	PIN_E1	1	B1_N0	PIN_E1	2.5 V (default)		8mA (default)
led_export[4]	Output	PIN_F2	1	B1_N0	PIN_F2	2.5 V (default)		8mA (default)
led_export[3]	Output	PIN_H1	1	B1_N1	PIN_H1	2.5 V (default)		8mA (default)
led_export[2]	Output	PIN_J3	1	B1_N1	PIN_J3	2.5 V (default)		8mA (default)
led_export[1]	Output	PIN_J2	1	B1_N1	PIN_J2	2.5 V (default)		8mA (default)
led_export[0]	Output	PIN_J1	1	B1_N1	PIN_J1	2.5 V (default)		8mA (default)
pb_1_export	Input	PIN_H2	1	B1_N1	PIN_H2	2.5 V (default)		8mA (default)
pb_2_export	Input	PIN_G3	1	B1_N0	PIN_G3	2.5 V (default)		8mA (default)
spi_0_MISO	Input	PIN_AB16	4	B4_N1	PIN_AB16	3.3-V LVCMOS		2mA (default)
spi_0_MOSI	Output	PIN_AA16	4	B4_N1	PIN_AA16	3.3-V LVCMOS		2mA (default)
spi_0_SCLK	Output	PIN_AA15	4	B4_N1	PIN_AA15	3.3-V LVCMOS		2mA (default)
spi_0_SS_n	Output	PIN_AB15	4	B4_N1	PIN_AB15	3.3-V LVCMOS		2mA (default)
spi_can_int_n_export	Input	PIN_AB14	4	B4_N1	PIN_AB14	3.3-V LVCMOS		2mA (default)

2. Generate a bit stream file by compiling in Quartus II under the Compilation tab or using the compilation button.

Programmer

3. Download the design to the Cyclone III FPGA on both the master and slave DE0 boards by using Programmer under the Tools menu.

Implementing the Shared-Clock Scheduling

The TTC scheduler code used before has been adapted into the Shared-Clock TTC scheduler that we are using in this exercise. The code for this more complex scheduler involves the sending and acknowledgement of messages between the master and slave nodes.

Eclipse IDE

Start by opening the Eclipse based Software Build Tool from the Tools tab in Quartus II.

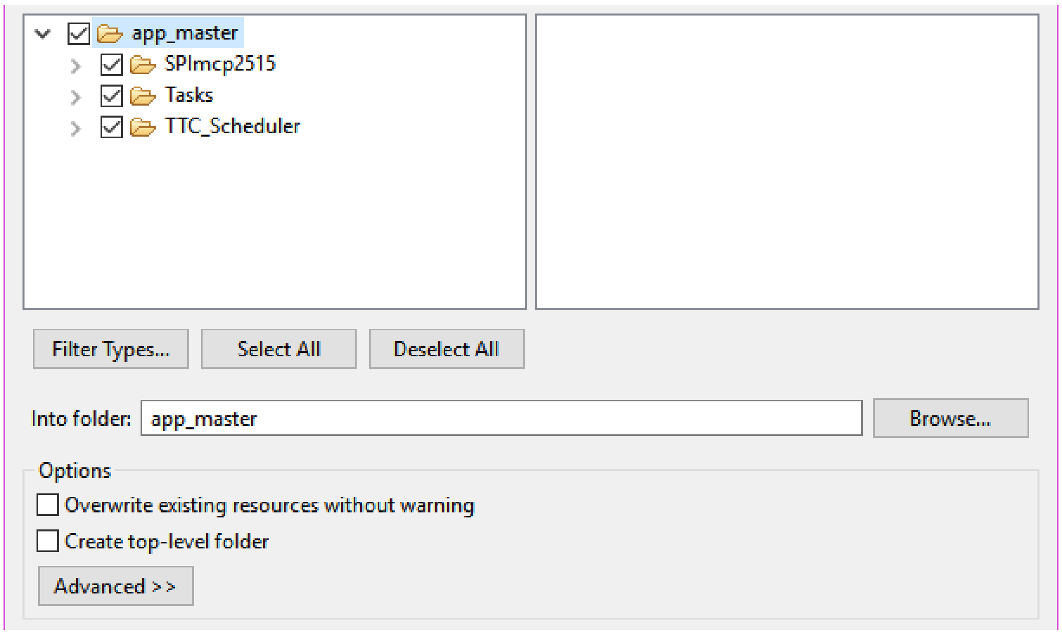
As we have a Master board and a Slave board that fulfil different functions, we will need to create two **separate** applications, one called **master_app** and one called **slave_app**, each with their **own** board support package. Follow the instructions below **twice**, first for master, and then a second time for the slave.

To create the applications, follow the following steps for each application:

1. Open the Nios II Software Build Tool for Eclipse and select File -> New -> Nios II

Application and BSP from template.

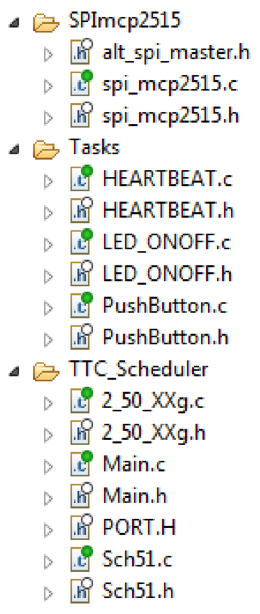
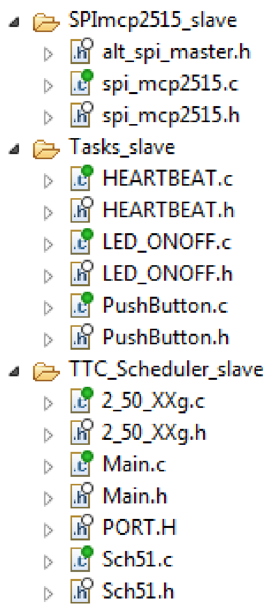
2. Select the "SOPC Information File and Blank project" option in the displayed window. Give the name for the project (app_master or app_slave) and select Finish to complete this. (Do not click Next on the window).
3. Once done, a Board Support Package (BSP) is automatically generated in the Nios II Software Build Tool for Eclipse.
4. Right click on your project (master or slave) and select "Import" from the menu, to import the code (download from blackboard).
5. Expand "General" and select "File System". A "File System" dialog will open. Click browse to locate the source folder that includes the correct (master or slave) files. Click on 'Select All' to include all files into the project. As we are importing the sub folders, "Create top-level folder" doesn't need to be selected (see screenshot below). Click Finish to complete the process. Once the folder is added, clean and build the project.
6. You will need to check and edit your PORT.h file to ensure that your inputs and outputs are defined correctly. Use the system header file under the project BSP or your Qsys design to do this.



Into folder: app_master

Options
☐ Overwrite existing resources without warning
☐ Create top-level folder

We now have the software that will to run in our applications, which should appear as below.

Master Software	Slave Software
 SPImcp2515 alt_spi_master.h spi_mcp2515.c spi_mcp2515.h Tasks HEARTBEAT.c HEARTBEAT.h LED_ONOFF.c LED_ONOFF.h PushButton.c PushButton.h TTC_Scheduler 2_50_XXg.c 2_50_XXg.h Main.c Main.h PORT.H Sch51.c Sch51.h	 SPImcp2515_slave alt_spi_master.h spi_mcp2515.c spi_mcp2515.h Tasks_slave HEARTBEAT.c HEARTBEAT.h LED_ONOFF.c LED_ONOFF.h PushButton.c PushButton.h TTC_Scheduler_slave 2_50_XXg.c 2_50_XXg.h Main.c Main.h PORT.H Sch51.c Sch51.h

7. Under project, click Clean and Build the workspace.
8. Select the project and choose “Nios II Hardware” under the “Run As” submenu on the “Run” menu.
9. Your application should now run successfully.
10. In this exercise, you have to download two elf files in the following order: first to the slave board and then to the master board.
    - Be careful about the Blaster USB port, while downloading your program on the target DE0 board.
    - Each USB blaster will provide you access for downloading a C program on the associated RAM of the Master or Slave hardware design.

Demonstrate Understanding

Three main application tasks enable the functionality that we desire: **Heartbeat**, **LED On/Off** and **Push Button**:

- Heartbeat
    - The Heartbeat uses the code for a Flashing LED, found on page 240 and elsewhere in Pont PTTES.
- LED On/Off
    - This is adapted from the LED Flashing example.
- Push button
    - The code for a simple switch can be found in Pont PTTES on page 408/409.

Please analyse the code to understand how the above tasks perform their functions.

Please fill in the table below with a description of the Shared-Clock (SCH) TTC scheduler functions listed. The sample code can be found by searching the Pont PTTES book.

Master Node		
	Function Name	**Description**
Shared Clock Scheduler	Void SCH_Init_T0(void)	
	Void SCH_Start(void)	
	Void SCH_Update(void * context)	
	Void SCC_A_Master_Send_Tick_Message(const tByte SLAVE_INDEX)	
	Void SCC_A_Master_Start_Slave(const tByte SLAVE_ID)	
	Void SCC_A_Master_Process_Ack(const tByte SLAVE_INDEX)	
Tasks	Void HEARTBEAT_Update(void)	
	Void LED_ONOFF(void)	
	Void PushButton_Update(void)	
Slave Node		
	Function Name	**Description**
Shared Clock Scheduler	Void SCH_Init_T0(void)	
	Void SCH_Start(void)	
	Void SCH_Update(void * context)	
	tByte SCC_A_Slave_Process_Tick_Message(void)	
	Void SCC_A_Send_Ack_Message(void)	
Tasks	Void HEARTBEAT_Update(void)	
	Void LED_ONOFF(void)	
	Void PushButton_Update(void)	

Appendix: Shared-Clock Scheduler Communication Diagram

The sequence diagram in Fig. 3 shows the communication sequence between the master and slave nodes, the main steps of which are outlined below the figure.

We can see from the sequence diagram that the main tasks occur following connection, which involves sending a message from the master to the slave and receiving an acknowledgement message.

Carefully go through this so that you understand how the nodes will be talking to each other and what tasks are being carried out.

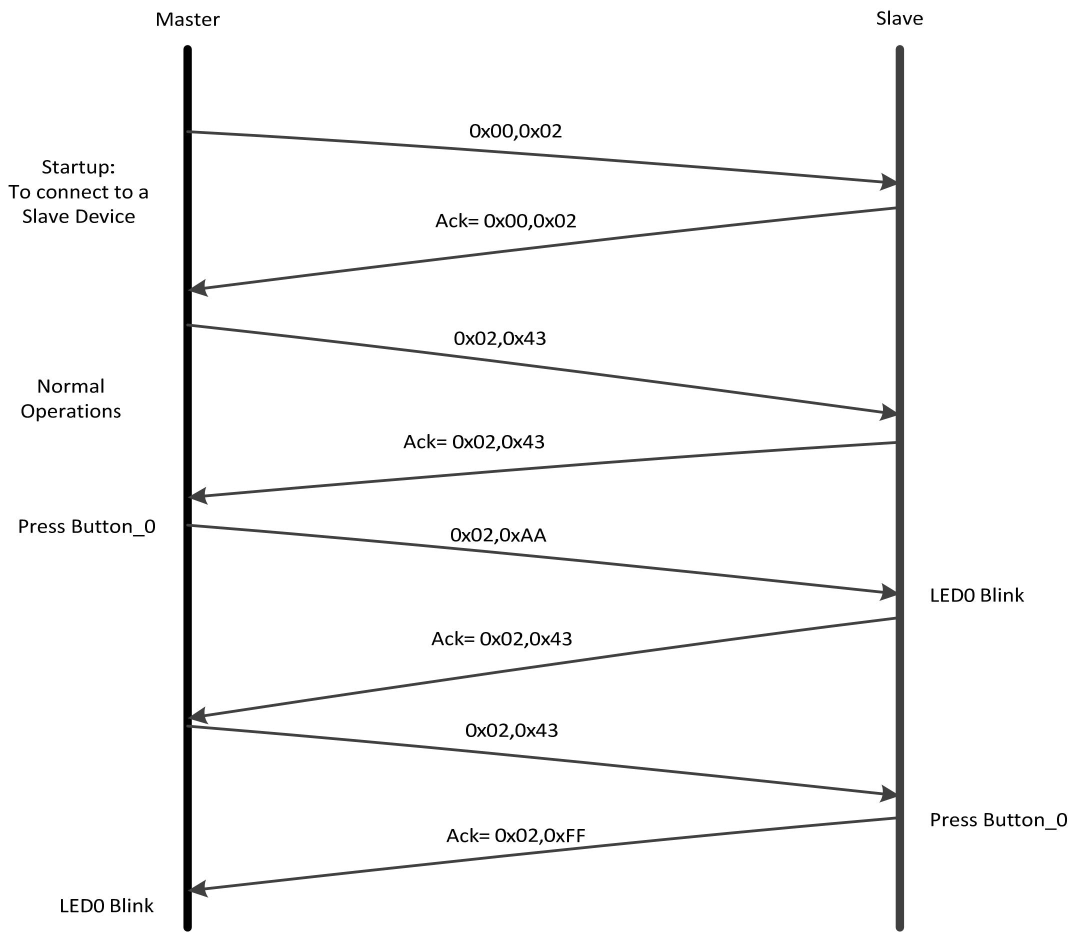


Figure 3

A. **Startup:** At startup, the master will send a message (0x00, 0x02) to the slave to check their presence. On receipt of this message, the slave node will send a reply message (0x00, 0x02) to the master node. The second byte of this message is the slave ID. After a successful exchange of startup messages, the slave is now connected with the master, and can start onward communications.

B. **Normal Operations:** During normal operations, the master will send a message (0x02, 0x43) to the slave. The first byte of this message will be the slave ID while the second byte will be 0x43. On receipt of this message, the slave node will acknowledge the master with an ACK message (0x02,0x43). The first byte of the ACK message will be the slave ID while the following byte will be 0x43.

C. **Master Button0 Press:** On pressing the master board Button0, the master will send a message containing 0x02, 0xFF. The first byte of this message will be the slave ID. In receipt of this message, the slave node will send the same ACK message (0x02,0x43). This will turn ON/OFF LED0 on the slave node board.

D. **Slave Button0 Press:** On pressing the slave board Button0, the slave will send a modified ACK message (0x02, 0xff), which will turn ON/OFF LED0 of the master node board.

E. **LED0 Blinks**

Prof. Tanya Vladimirova

15 November 2018