

EG3205 Part II. Multiprocessor and Multicore Systems

Laboratory Exercise 5

Contents

Lab 5 Introduction	1
Lab 5 Description	1
Creating your Qsys Design	4
Implementing the Hardware Design	13
Software Implementation	13

Lab 5 Introduction

This lab exercise aims to introduce you to a dual-core design based on the Nios II soft processor core, which you will serve as a stepping stone for your design work in Assignment 2.

The purpose of Lab 5 is to create a dual-core processor as the slave for a two-node distributed system similar to the one made in Lab 4. The two nodes will be connected to a CAN bus, implemented through the use of CAN-SPI modules.

Lab 5 Description

In the dual-core processor node, one core, which is referred to as Event Triggered (ET) core, is able to handle event-triggered tasks. The other core, which is referred to as Time Triggered (TT) core, is able to handle time-triggered tasks.

Inter-processor communication is achieved via a shared message buffer and the access of the two cores to the buffer is ensured by a mutual exclusion (mutex) IP core. The details on the hardware configuration are given below in “Creating your Qsys Design”

The ET core of the dual-processor node will be connected to the CAN bus as a slave. The other node on the CAN bus, which will act as a master, will be assumed to generate “bursts” of data at random time intervals. Your ET core will receive and process these data.

The dual-core processor design is intended for use in safety-critical embedded applications.

Task 1

To begin with, you need to develop your Qsys design that we will be using in this Lab as illustrated in Fig. 1:

- The Nios II processor configuration should be designed as illustrated in section “Creating your Qsys Design” below which provides a step-by-step guide to the design process.
- The boards must be connected via the CAN-SPI modules as you have previously done

Once completed move on to Task 2.

Task 2

Your **second task** is to analyse the provided code and understand:

- the code supporting the mutex IP core;
- the functions performed by the ET and TT schedulers in making the slave application run;
- the code related to the CAN controller MCP2515 on the CAN-SPI board.

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

Files provided

ET_slave_app.zip
TT_slave_app.zip

Acknowledgements:

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).

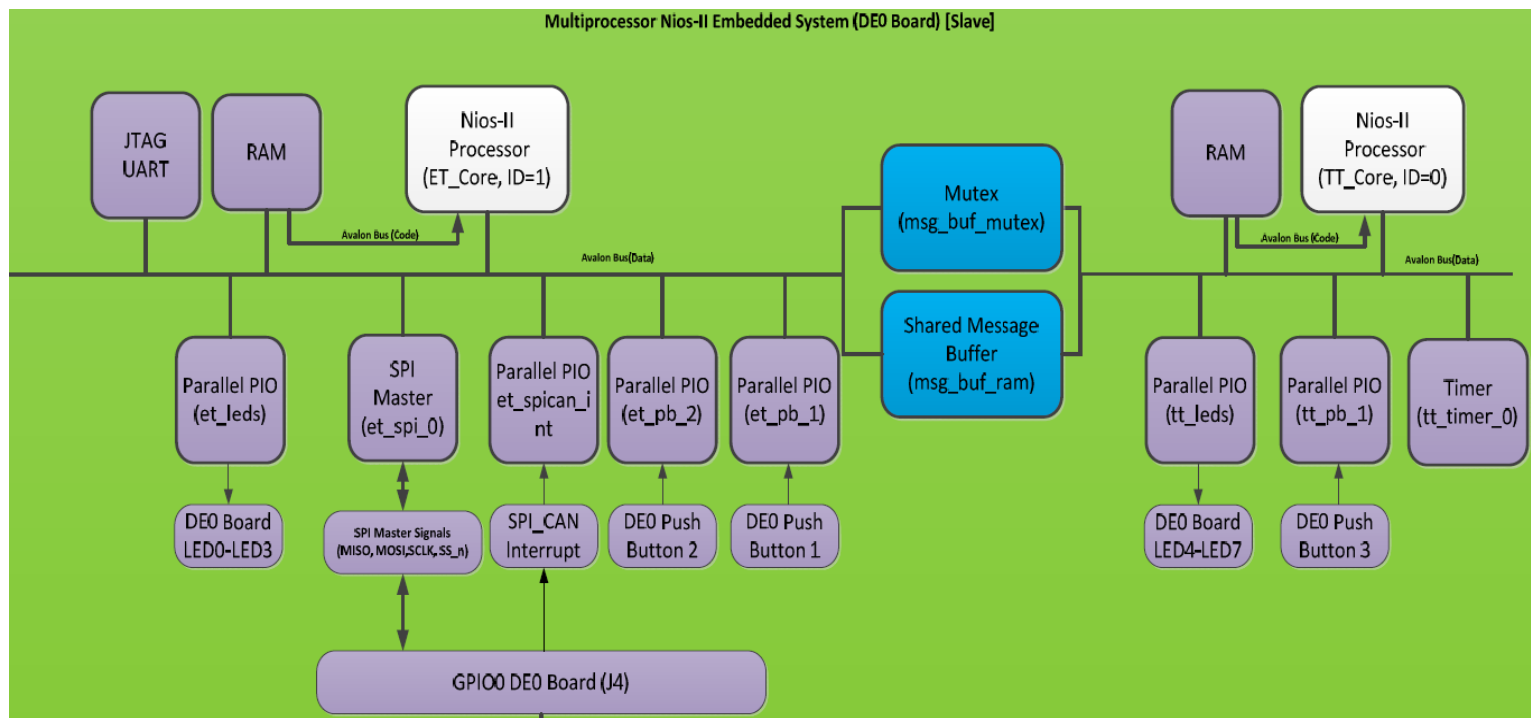
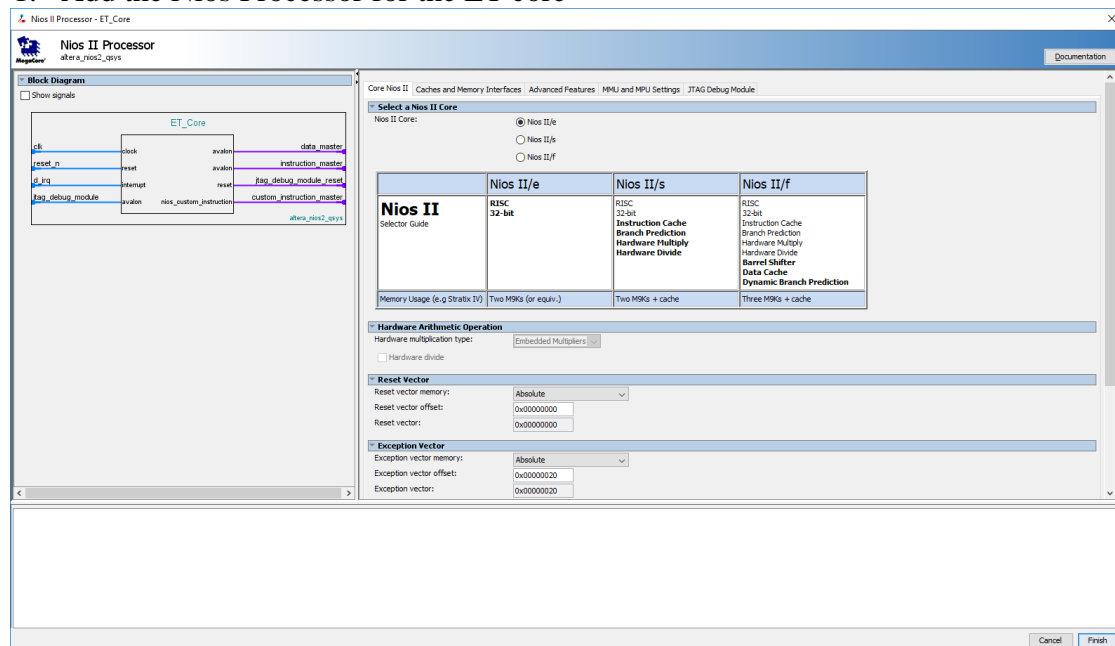


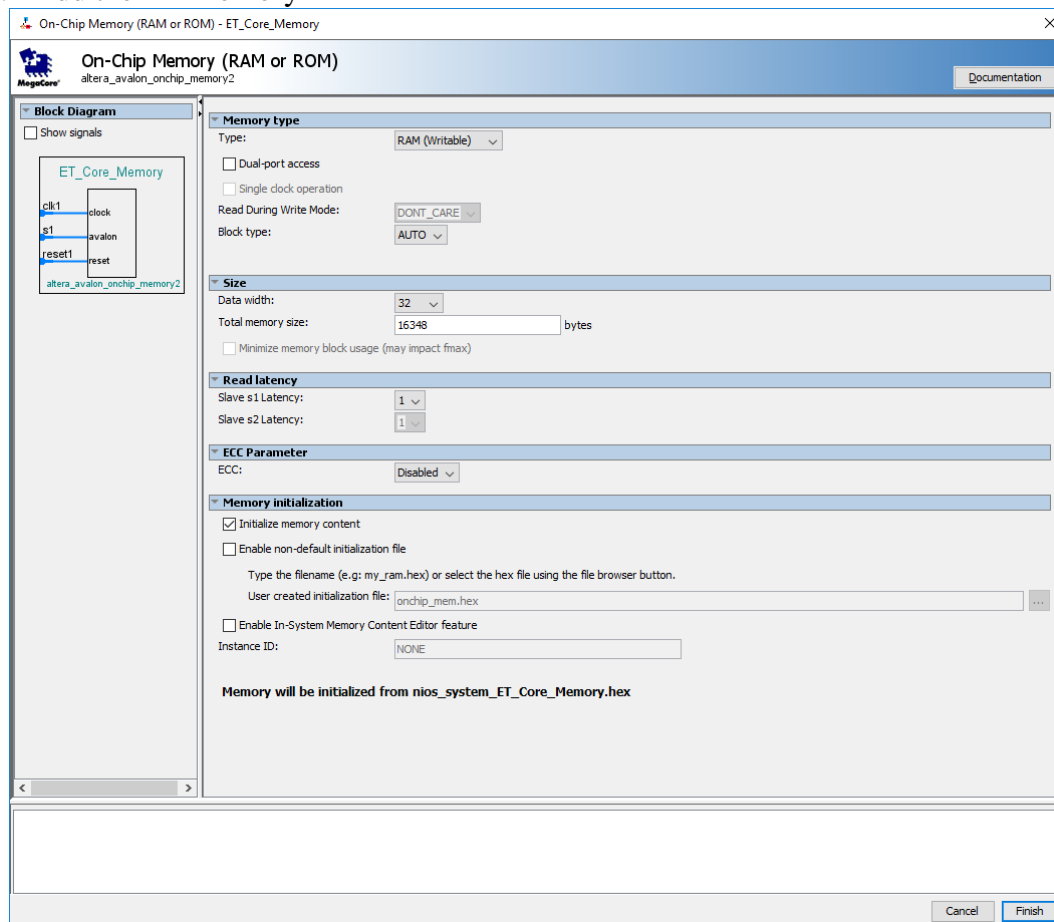
Figure 1. A dual-core design based on the Nios II soft processor core serving as a slave node in a multi-processor system.

Creating your Qsys Design

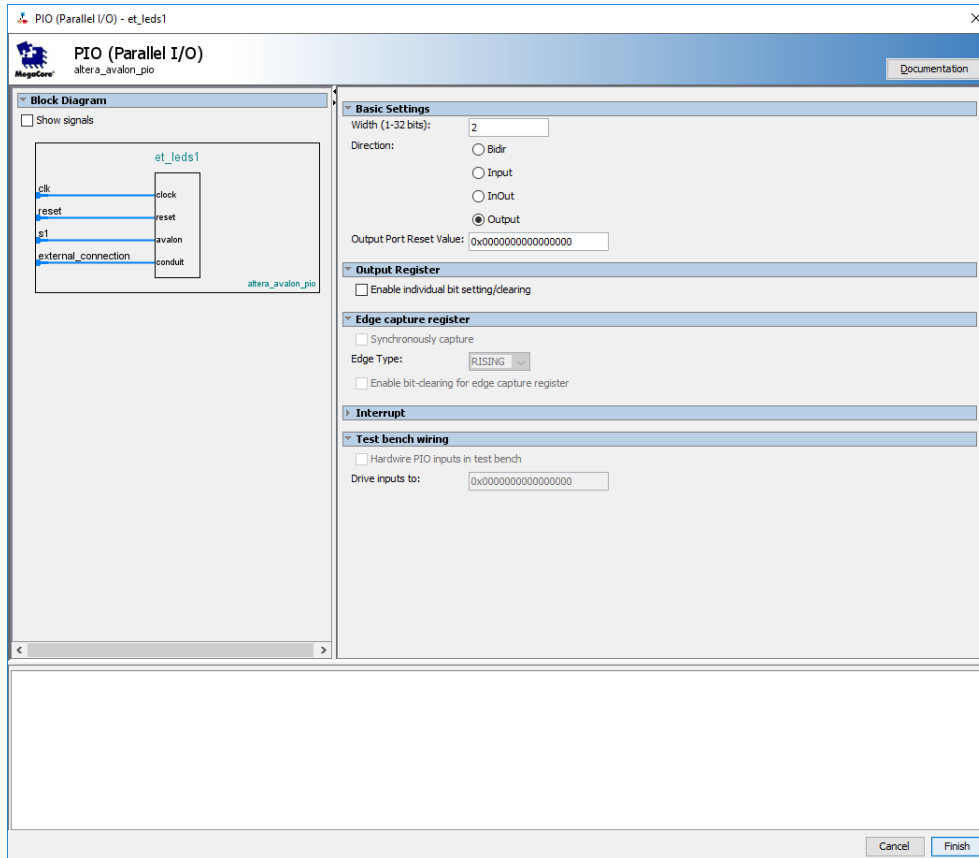
1. Add the Nios Processor for the ET core



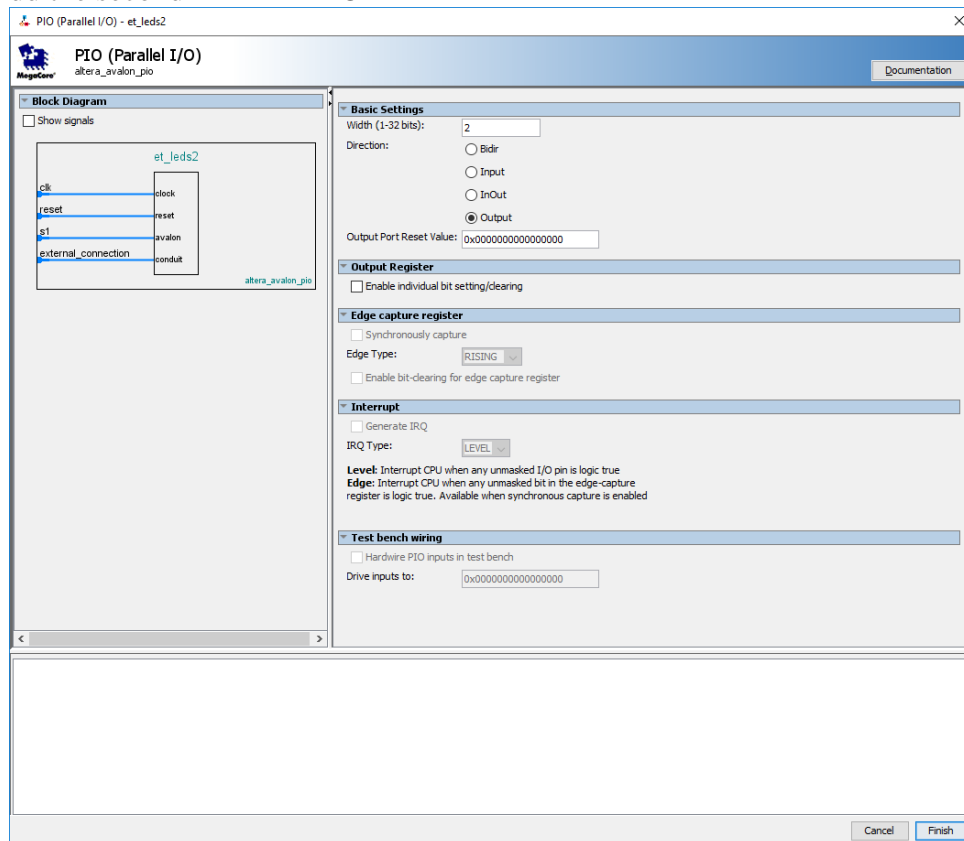
2. Add the ET Memory



3. Add the first ET LED PIO



4. Add the second ET LED PIO



5. Add the ET SPI

SPI (3 Wire Serial) - et_spi_0

SPI (3 Wire Serial)
altera_avalon_spi

Block Diagram

☐ Show signals

Master/Slave

Type: Master

Number of select (SS_n) signals (one for each slave): 1

SPI clock (SCLK) rate: 128000 Hz

Actual clock rate: 127551.0 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

Cancel Finish

6. Add the ET SPICAN PIO

PIO (Parallel I/O) - et_spican_int

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: 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

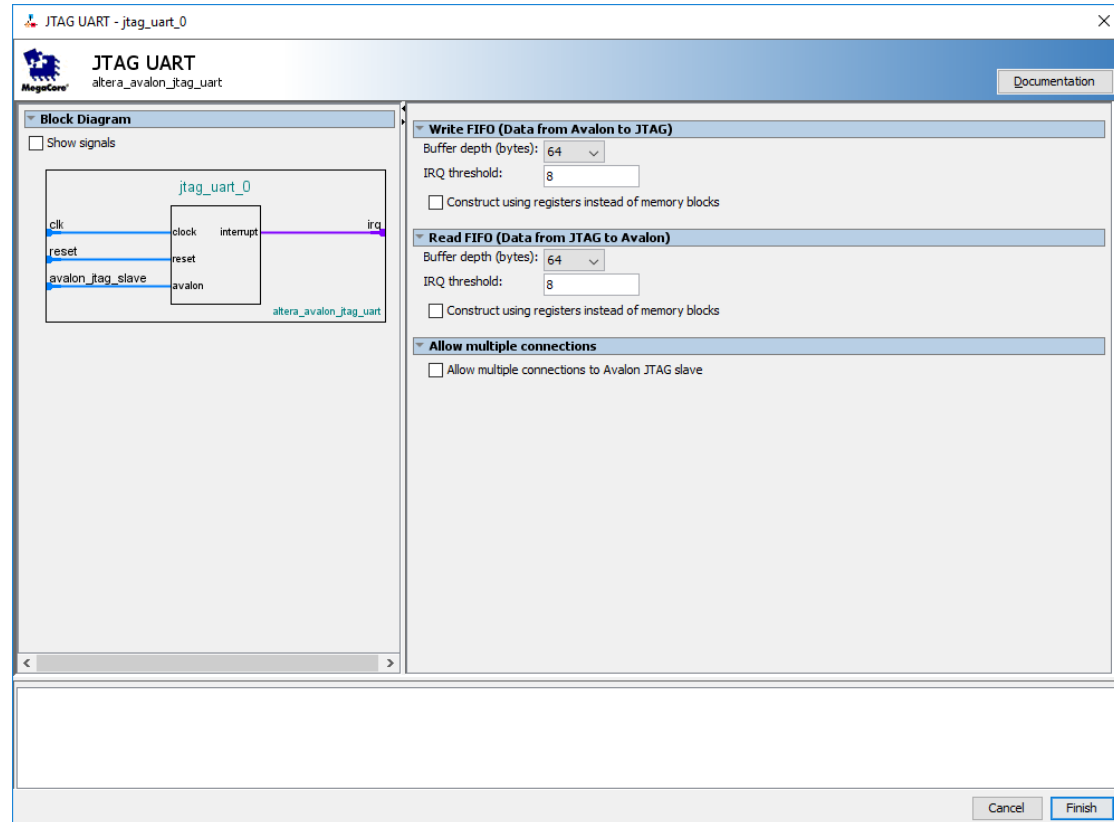
Test bench wiring

☒ Hardwire PIO inputs in test bench

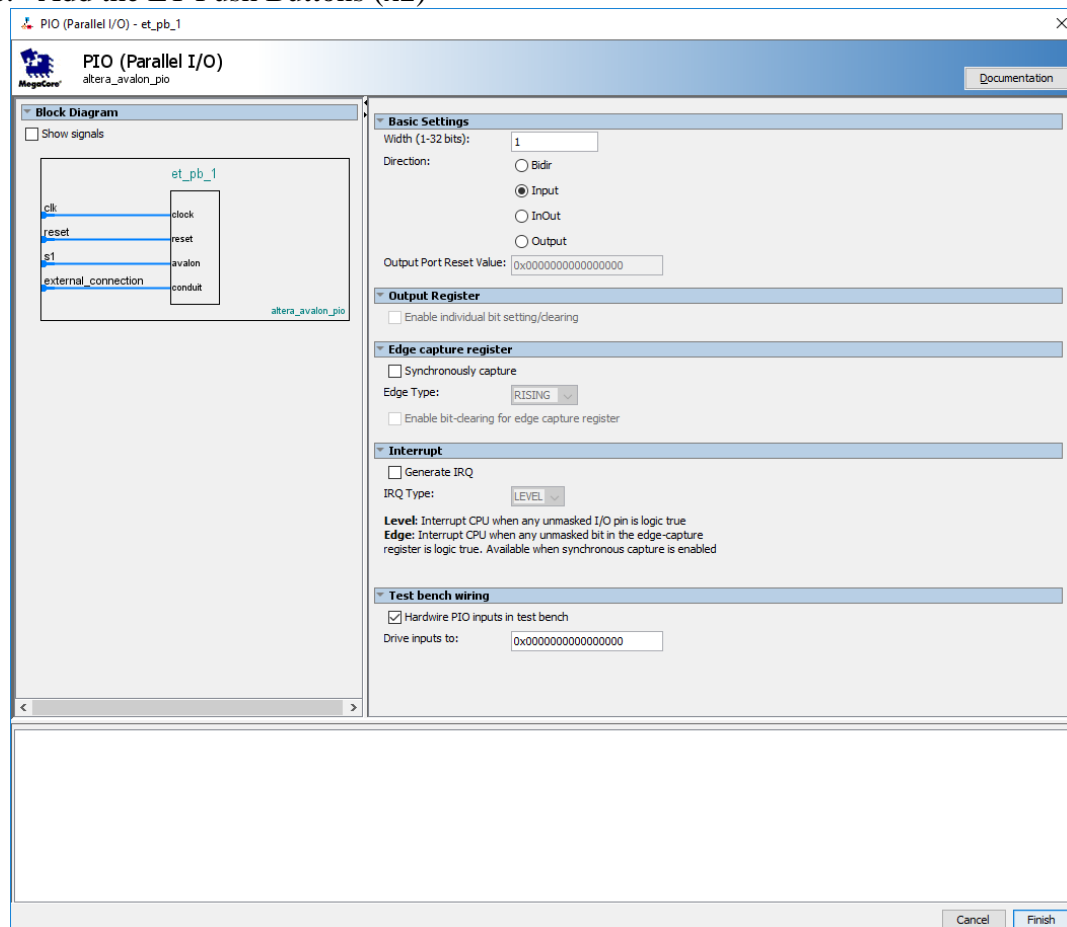
Drive inputs to: 0x0000000000000000

Cancel Finish

7. Add the JTAG UART



8. Add the ET Push Buttons (x2)



9. Add the TT Core

Nios II Processor
altera_nios2_qsys

Block Diagram
Show signals

TT_Core

clk → clock
reset_n → reset
s1 → avalon
tag_debug_module → tag_debug_module_reset
tag_debug_module → nios_custom_instruction
nios_custom_instruction → custom_instruction_master
custom_instruction_master → altera_nios2_qsys

Core Nios II Caches and Memory Interfaces Advanced Features MMU and MPU Settings JTAG Debug Module

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

Cancel Finish

10. Add the TT Memory

On-Chip Memory (RAM or ROM) - TT_core_memory

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Block Diagram
Show signals

TT_core_memory

clk1 → clock
s1 → avalon
reset1 → reset
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: 16384 bytes
☐ Minimize memory block usage (may impact fmax)

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

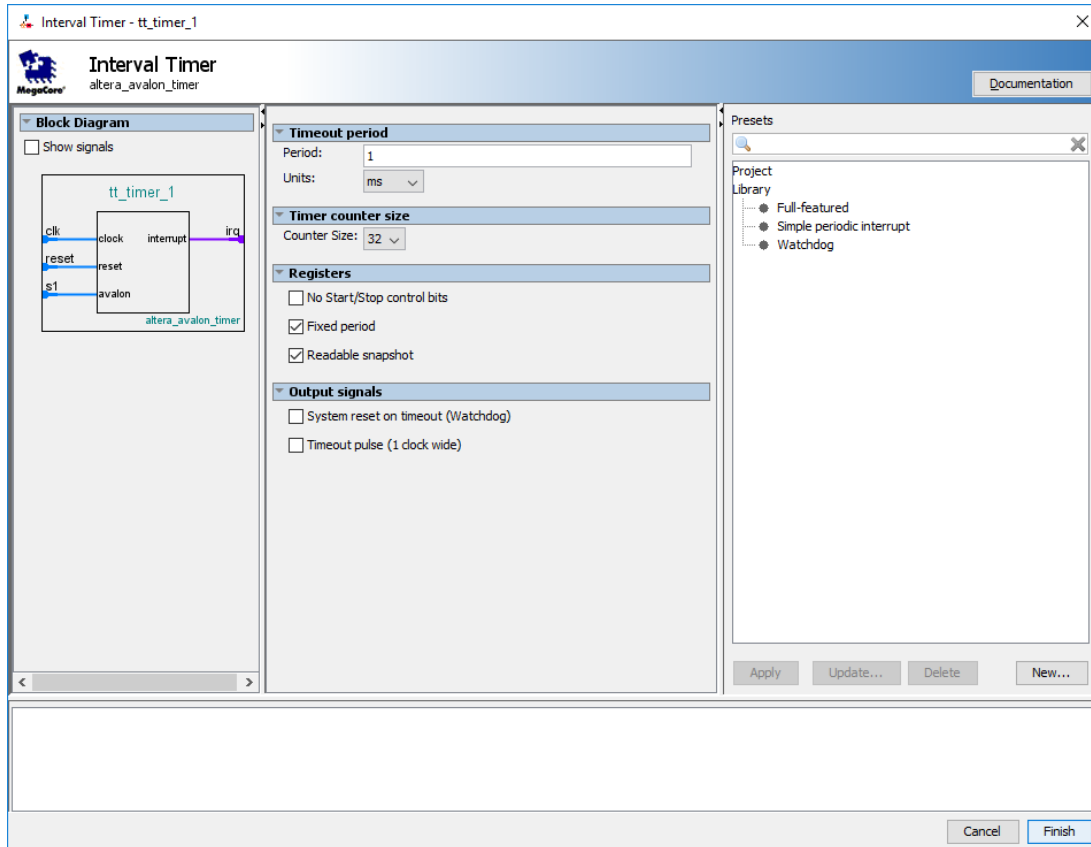
ECC Parameter
ECC: Disabled

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

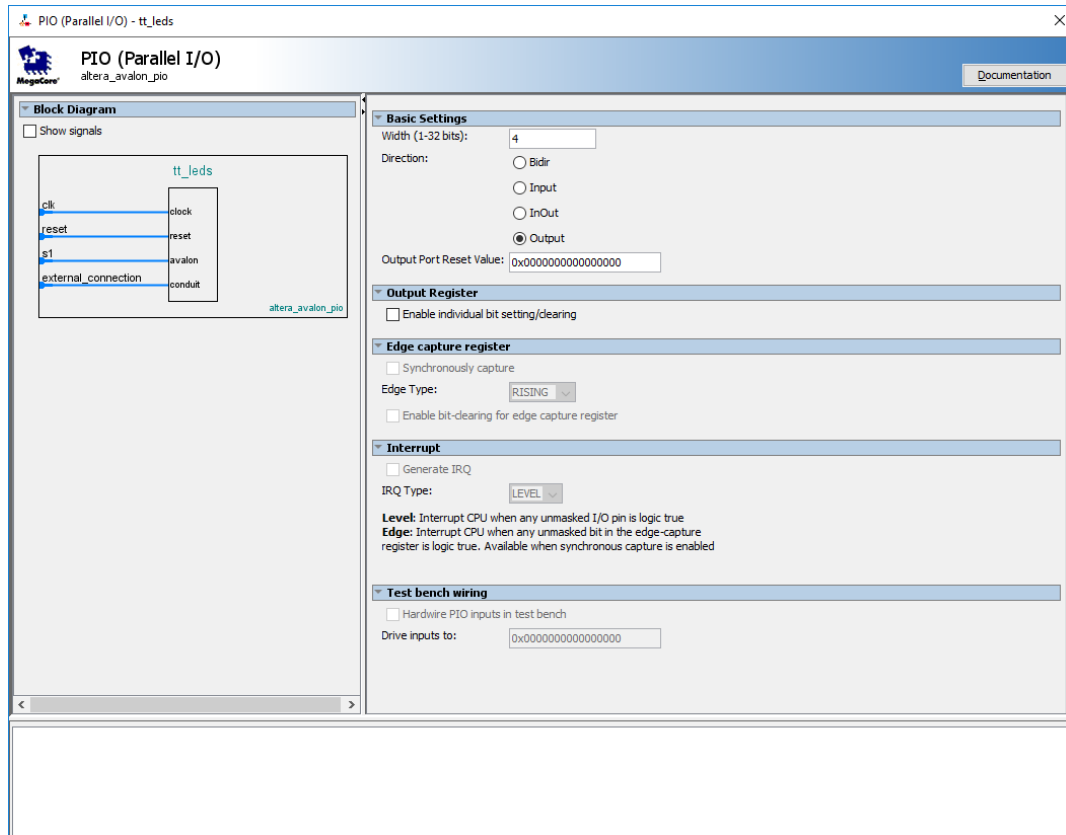
Memory will be initialized from nios_system_TT_core_memory.hex

Cancel Finish

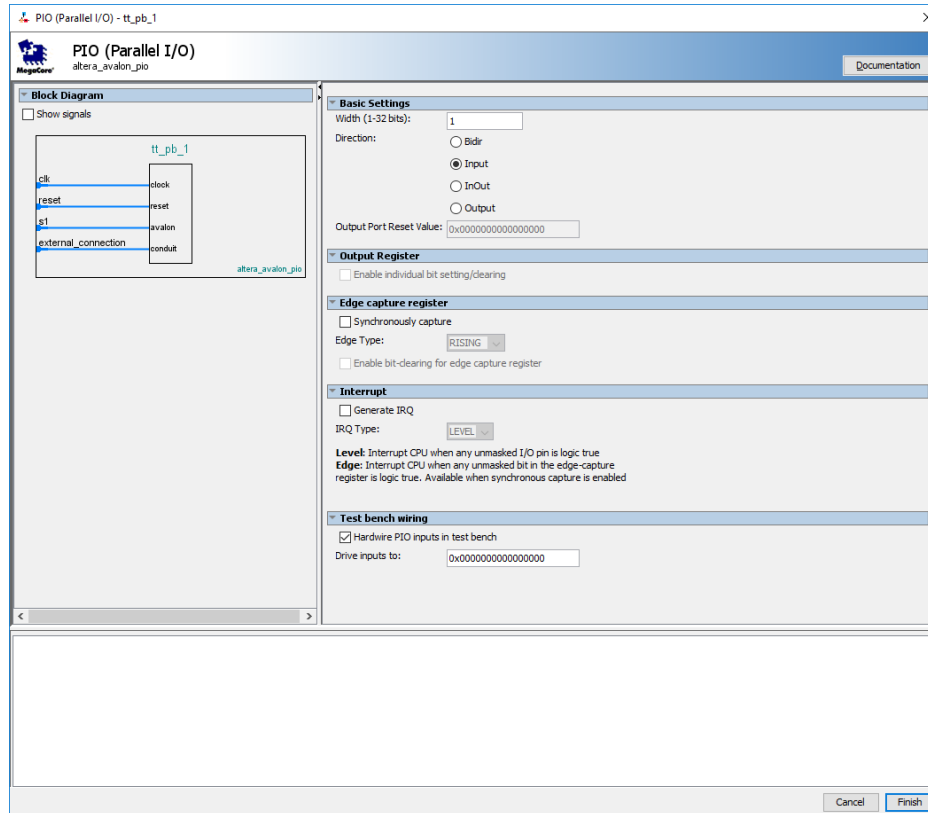
11. Add the TT Timer



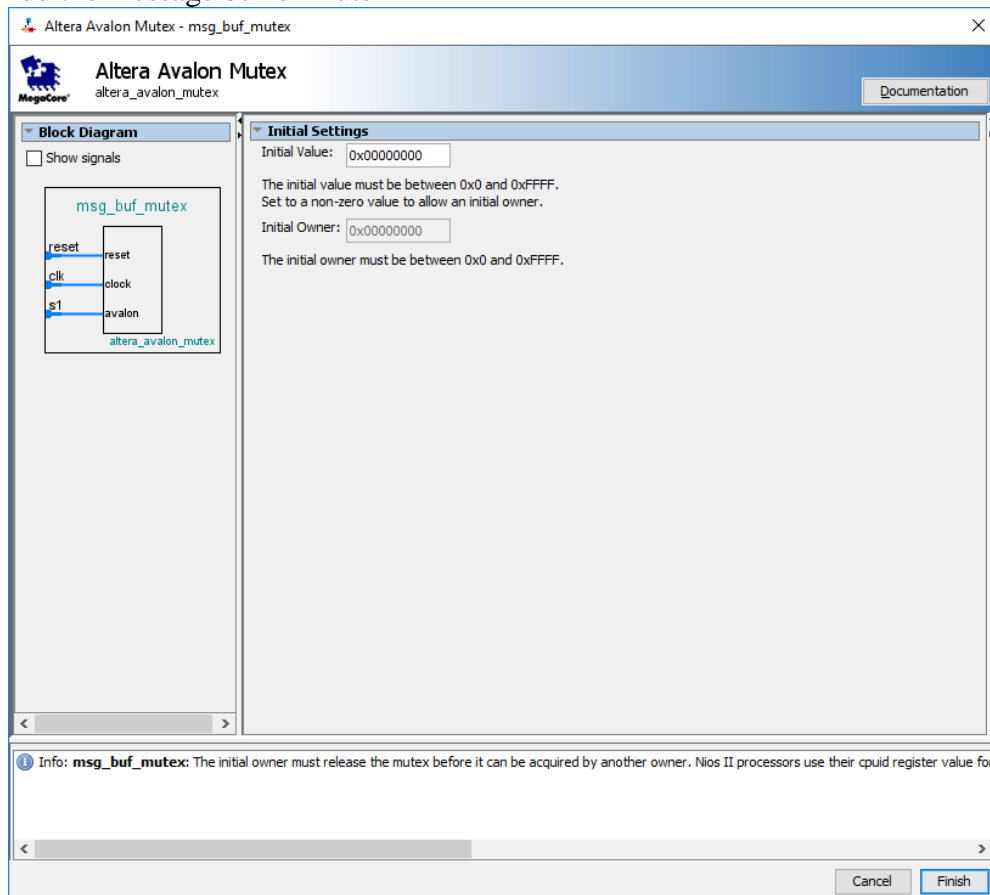
12. Add the TT LED PIO



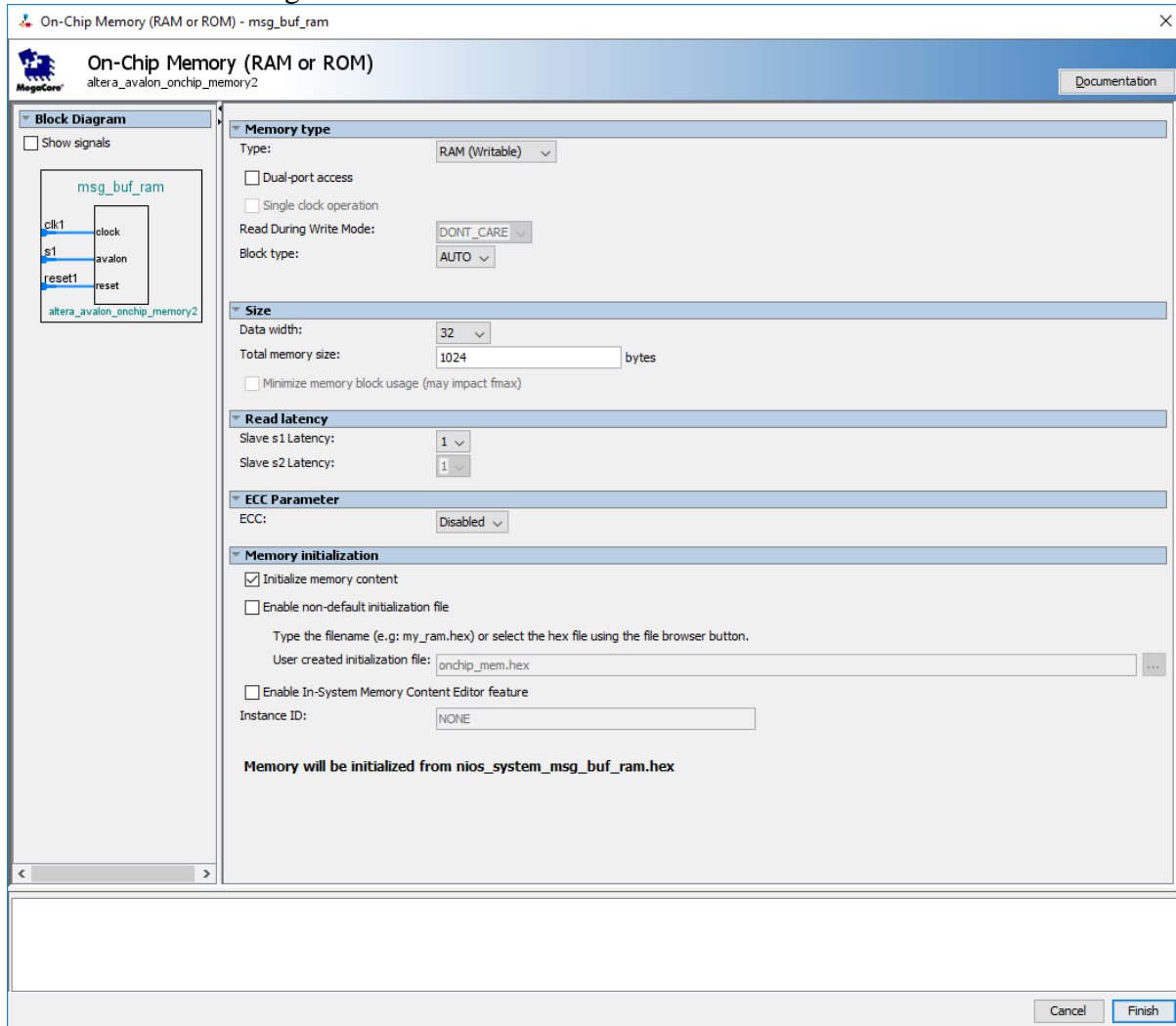
13. Add the TT Push Button



14. Add the message buffer mutex



15. Add the message buffer RAM



Once all components are added, and connect them, export the conduits, set memory addresses, and IRQs so that your design looks like the image on the next page.

Make sure to name the components identically and match up your base memory addresses

Connections	Name	Description	Export	Clock	Base	End	IRQ	T
	clk_0 clk_in clk_in_reset clk clk_reset	Clock Source Clock Input Reset Input Clock Output Reset Output	clk <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	exported clk_0				
	ET_Core clk reset_n data_master instruction_master jtag_debug_module_r... jtag_debug_module custom_instruction_m...	Nios II Processor Clock Input Reset Input Avalon Memory Mapped Master Avalon Memory Mapped Master 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>	clk_0 [clk] [clk] [clk] [clk] [clk]		IRQ 0	IRQ 31	
	ET_Core_Memory clk1 s1 reset1	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]	# 0x0000	0x3fdb		
	et_leds1 clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9470	0x947f		
	et_leds2 clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0xa000	0xa00f		
	et_spi_0 clk reset spi_control_port external	SPI (3 Wire Serial) 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>	clk_0 [clk] [clk]	# 0x9420	0x943f		
	et_spican_int clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9460	0x946f		
	jtag_uart_0 clk reset avalon_jtag_slave	JTAG UART Clock Input Reset Input Avalon Memory Mapped Slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x9488	0x948f		
	et_pb_1 clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9450	0x945f		
	et_pb_2 clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9440	0x944f		
	TT_Core clk reset_n data_master instruction_master jtag_debug_module_r... jtag_debug_module custom_instruction_m...	Nios II Processor Clock Input Reset Input Avalon Memory Mapped Master Avalon Memory Mapped Master 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>	clk_0 [clk] [clk] [clk] [clk] [clk]	# 0x8800	0x8fff	IRQ 0	IRQ 31
	TT_core_memory clk1 s1 reset1	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]	# 0x0000	0x3fff		
	tt_timer_1 clk reset s1	Interval Timer Clock Input Reset Input Avalon Memory Mapped Slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x9400	0x941f		
	tt_leds clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9430	0x943f		
	tt_pb_1 clk reset s1 external_connection	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>	clk_0 [clk] [clk]	# 0x9420	0x942f		
	msg_buf_muxex reset clk s1	Altera Avalon Muxex Reset Input Clock Input Avalon Memory Mapped Slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	[clk] clk_0 [clk]	# 0x9480	0x948f		
	msg_buf_ram clk1 s1 reset1	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]	# 0x9000	0x93ff		
	prio_0 clk reset s1	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	# 0x94a0	0x94af		
	external_connection	Conduit	tt_test					

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	I/O Standard	Reserved	Current Strength	S
clk_clk	Unknown	PIN_G21	6	B6_N1	2.5 V (default)		8mA (default)	
spl_0_MISO	Unknown	PIN_AB16	4	B4_N1	3.3-V LVCMOS		2mA (default)	
spl_0_MOSI	Unknown	PIN_AA16	4	B4_N1	3.3-V LVCMOS		2mA (default)	
spl_0_SCLK	Unknown	PIN_AA15	4	B4_N1	3.3-V LVCMOS		2mA (default)	
spl_0_SS_n	Unknown	PIN_AB15	4	B4_N1	3.3-V LVCMOS		2mA (default)	
spl_can_int_n_export	Unknown	PIN_AB14	4	B4_N1	3.3-V LVCMOS		2mA (default)	
tt_led_export[0]	Unknown	PIN_F2	1	B1_N0	2.5 V (default)		8mA (default)	
tt_led_export[1]	Unknown	PIN_E1	1	B1_N0	2.5 V (default)		8mA (default)	
tt_led_export[2]	Unknown	PIN_C1	1	B1_N0	2.5 V (default)		8mA (default)	
tt_led_export[3]	Unknown	PIN_C2	1	B1_N0	2.5 V (default)		8mA (default)	
et_pb_1_export	Unknown	PIN_H2	1	B1_N1	2.5 V (default)		8mA (default)	
et_pb_2_export	Unknown	PIN_G3	1	B1_N0	2.5 V (default)		8mA (default)	
tt_pb_1_export	Unknown	PIN_F1	1	B1_N0	2.5 V (default)		8mA (default)	
tt_test_export	Unknown	PIN_AA20	4	B4_N0	3.3-V LVCMOS		2mA (default)	
et_led1_export[1]	Unknown	PIN_J2	1	B1_N1	2.5 V (default)		8mA (default)	
et_led1_export[0]	Unknown	PIN_J1	1	B1_N1	2.5 V (default)		8mA (default)	
et_led2_export[1]	Unknown	PIN_H1	1	B1_N1	2.5 V (default)		8mA (default)	
et_led2_export[0]	Unknown	PIN_J3	1	B1_N1	2.5 V (default)		8mA (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 the slave DE0 board by using Programmer under the Tools menu.

Software Implementation

Now we have two applications for the slave. There is now code for the Event-Triggered Core and the Time-Triggered Core.

Eclipse IDE

Start by opening the Eclipse based Software Build Tool from the Tools tab in Quartus II. Make sure that if you are opening Nios from the Quartus process that has the Lab 5 project open.

As we have a dual-core slave, where the cores perform different functions, we will need to create two **separate** applications, one called **ET_slave_app** and one called **TT_slave_app**, each with their **own** board support package. Follow the below 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 (ET_slave_app or TT_slave_app) 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 (ET or TT) 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 (ET or TT) files. Click on ‘Select All’ to include all files into the project. Click Finish to complete the process the projects should appear as in the screenshot below. Once the folder is added, clean and build the project.
6. If you have errors, try cleaning and building again to see if this clears any issues.

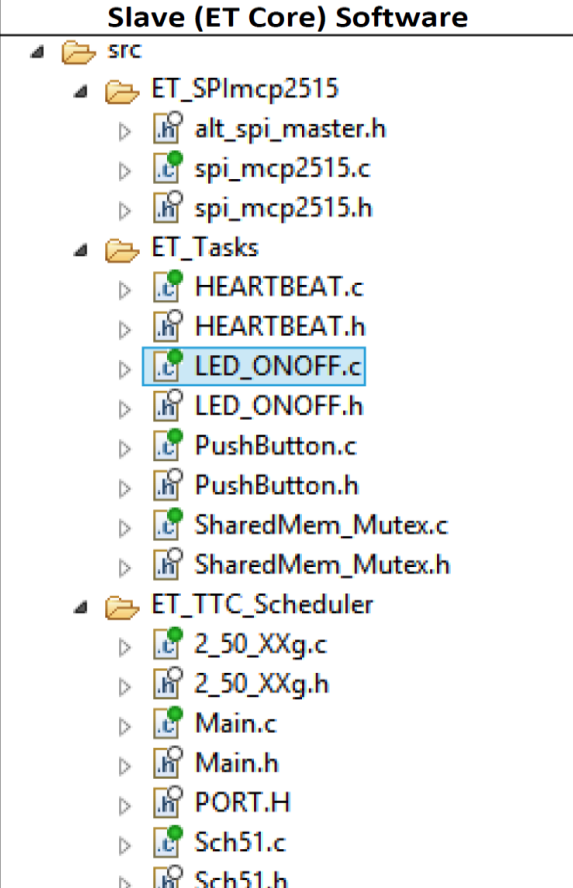
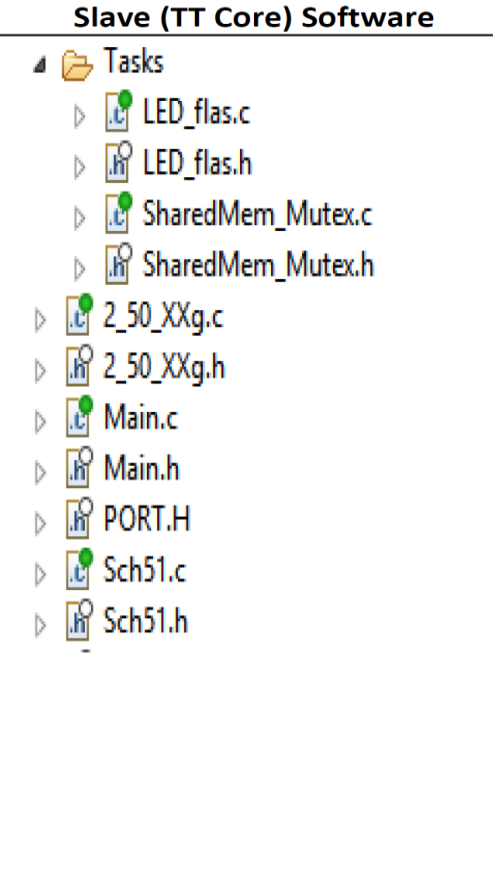
Slave (ET Core) Software	Slave (TT Core) Software
 <ul style="list-style-type: none"> src <ul style="list-style-type: none"> ET_SPLmcp2515 <ul style="list-style-type: none"> alt_spi_master.h spl_mcp2515.c spl_mcp2515.h ET_Tasks <ul style="list-style-type: none"> HEARTBEAT.c HEARTBEAT.h LED_ONOFF.c LED_ONOFF.h PushButton.c PushButton.h SharedMem_Mutex.c SharedMem_Mutex.h ET_TTC_Scheduler <ul style="list-style-type: none"> 2_50_XXg.c 2_50_XXg.h Main.c Main.h PORT.H Sch51.c Sch51.h 	 <ul style="list-style-type: none"> Tasks <ul style="list-style-type: none"> LED_flas.c LED_flas.h SharedMem_Mutex.c SharedMem_Mutex.h 2_50_XXg.c 2_50_XXg.h Main.c Main.h PORT.H Sch51.c Sch51.h

Figure 2. Software Configurations of Slave Node (ET core and TT core).

Prof. Tanya Vladimirova

20 November 2018