ECE3073 Computer Systems

Laboratory Session 2

Nios Assembler and Instruction Timing

1. Objectives

This laboratory exercise introduces the Altera Quartus IDE and the system integration tool Qsys. Using these two software tools you will be able to design the 'hardware' of a Nios processor to load onto a Cyclone II FPGA. You will then write a short Nios program in assembler that will be used to discover how long it takes Nios processor instructions to execute. These short laboratory exercises have been written to help develop your understanding of:

- The Altera Quartus software and Qsys
- Assembler programming for the Nios processor
- The time it takes the Nios processor to execute different classes of instructions
- The use of an oscilloscope to measure short time periods

Equipment

§ DE2 FPGA Development Board
§ A USB memory device provided by you to backup your design files
§ A 2 or 4 channel 1GS/s digital oscilloscope
§ Breakout pins to allow connection of CRO probes to DE2 board

2. Preliminary work

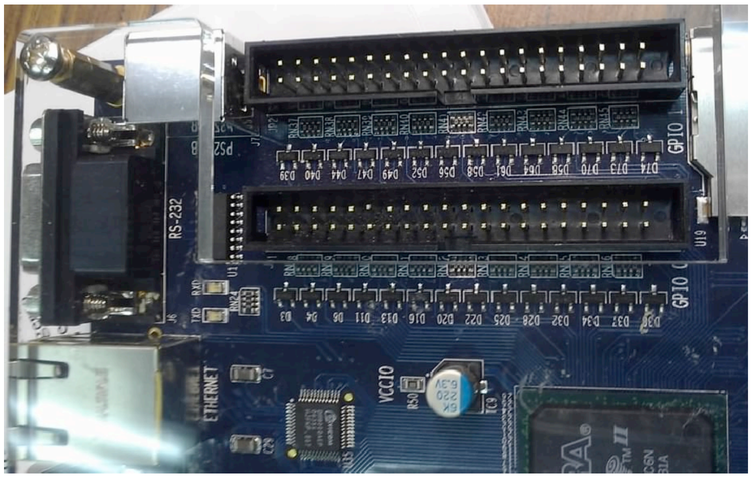
You must complete this preliminary work before attending the lab session.

a) For the main general purpose oscillator on the DE2 board:

what is the oscillator frequency (Hertz) …………………………………………….

and what is the clock period (ns) …………………………………………………

b) On the following photograph of the DE2 board identify which pins on the 40 pin connectors you will attach the oscilloscope probe to make connection to GROUND and to the GPIO_1[2] pin. **Hint:** look at the DE2 Schematics. Also note that in the DE2 pin assignments, the GPIO_1 array begins at GPIO_1[0].



Demonstrator initials to indicate satisfactory completion of this work prior to the lab

3. Entering the FPGA 'Hardware' design

a) Open Quartus using the desktop shortcut or by searching for "Quartus II 13.0sp1 (64-bit)" using the Windows start menu on the taskbar. If a project automatically opens, close it using **File > Close Project**.

b) Create a new project for your FPGA as follows:

i. Select from the Quartus II window **File > New Project Wizard…**. This will take you through the steps of creating a new Quartus Project, to which you will then add a NIOS processor via Qsys.

ii. If the Introduction window is displayed click on **Next**. In the 'Directory, Name, Top-Level Entity' window select a directory (folder) to store your project (do not use a folder on the Desktop as it is network mapped). Use a folder name that is unique to avoid clashes with other students and also to avoid reserved words. Remember there must be NO spaces anywhere in the directory path or the file name. Similarly give your project a unique and meaningful name such as Lab2. Then click **Next**.

iii. The next screen, 'Add Files', asks if you want to add any files to your design. These are pre existing files you may have created for another project. For now, leave it blank and click **Next**.

iv. The next screen, Family & Device Settings, allows you to select your FPGA device. This will depend on the development board you are using. First, select the FPGA family. For the DE2 board, the family of device (that is to say the architecture of the semiconductor) is Cyclone II. You can speed up the process by selecting the package type (FBGA), pin count (672) and speed grade (6). Look on your board, to check the part number (EP2C35F672C6 for the DE2), select this from the Available devices. Click **Next**.

v. You will use the default tools, so the next screen, EDA Tool Settings, can be left as it is. Click **Next**.

vi. The last screen, Summary, shows a summary of your selections. When you click **Finish**, your project will be created.

At the moment, all you have is an empty shell of a project. You now need a design entry file to start building the design.

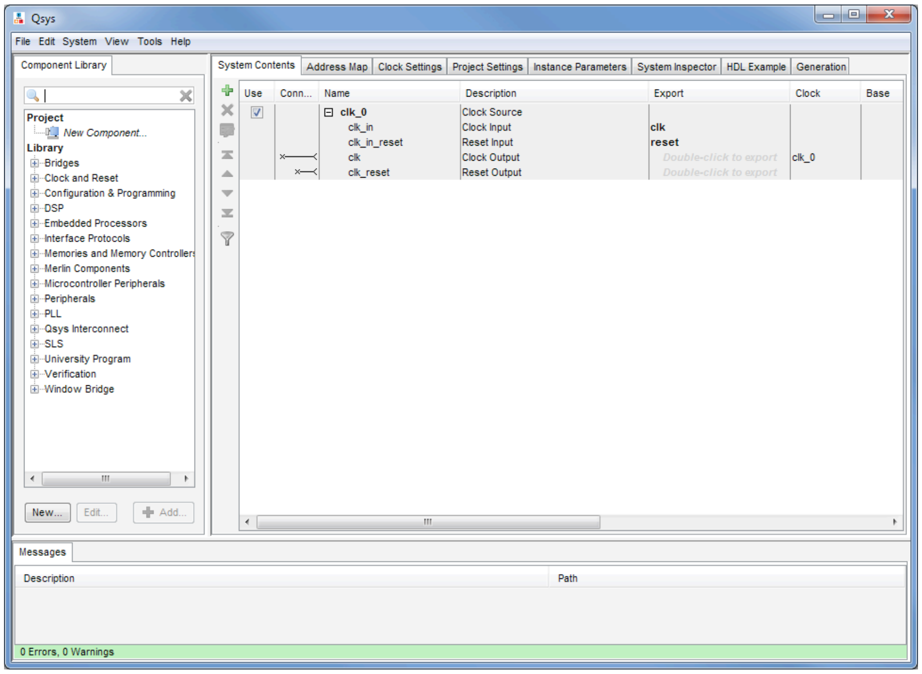
c) Select **File > New** and in the 'New' window that appears select **Design Files > Verilog HDL file**. Then click **OK**.

A new .v file will open up with a generic name. Click **File > Save As…** and the default name will change to your project name. Make sure the correct directory for your project is selected and the 'Add File to Current Project' box is ticked. (WARNING, Quartus 13.0 may not automatically default to your project directory! You may have to search for it.)

d) CRITICAL STEP. There is one more critical step you must perform. This is necessary to protect your FPGA hardware from unwanted signals driving prewired connections, for example enabling two memory chips to talk to the FPGA at once. This could have unwanted consequences like overheating due to short circuits. Select **Assignment >Device**, and the 'Device' window appears. Click on the **Device and Pin Options** button. In the pin options screen select **Unused Pins** and change the Reserve all unused pins option to "**As Input tri-stated**". That way no connections along pre existing copper tracks will be driven until you assign the pins yourself. Click **OK** (once for each window) to exit and you are ready to begin!

e) Adding a NIOS microprocessor to your design (this section of the lab exercise contains some text and diagrams from the ALTERA document Introduction to the Altera Qsys System Integration Tool)

First step is to open up Qsys either via its icon or **Tools > Qsys**. In a new project, where you are creating a NIOS microprocessor on your FPGA from scratch, you will see the following screen:



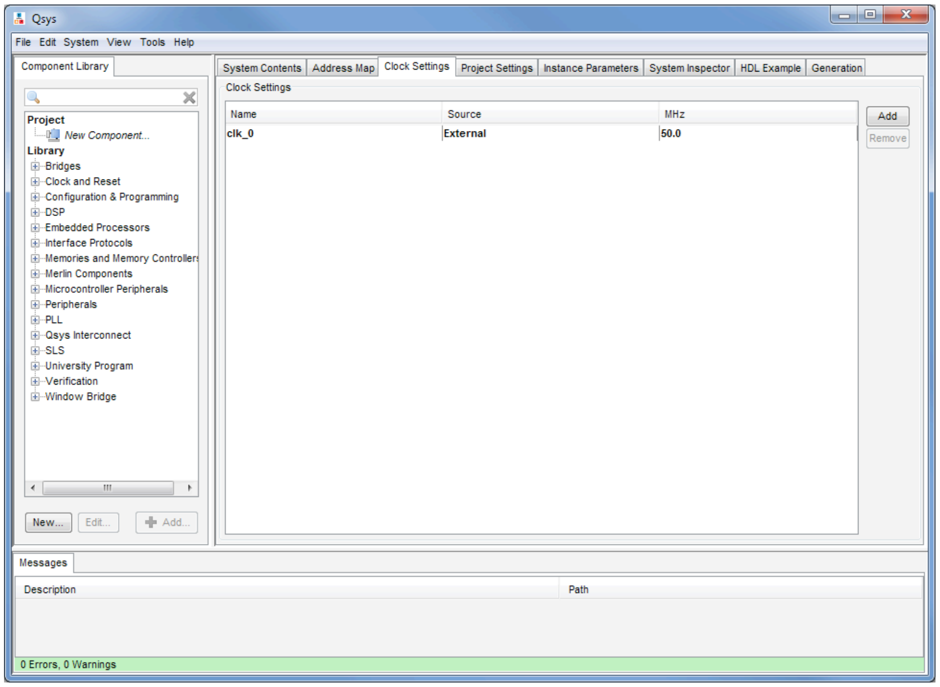
Use	Conn...	Name	Description	Export	Clock	Base
✓		clk_0	Clock Source			
		clk_in	Clock Input	clk		
		clk_in_reset	Reset Input	reset		
		clk	Clock Output	Double-click to export	clk_0	
		clk_reset	Reset Output	Double-click to export		

Description	Path

0 Errors, 0 Warnings

Figure 1. Create a new Nios II system.

This is the System Contents tab of the Qsys tool, which is used to add components to the system and configure the selected components to meet the design requirements. The available components are listed on the left side of the window.



Clock Settings

Name	Source	MHz
clk_0	External	50.0

Description	Path

0 Errors, 0 Warnings

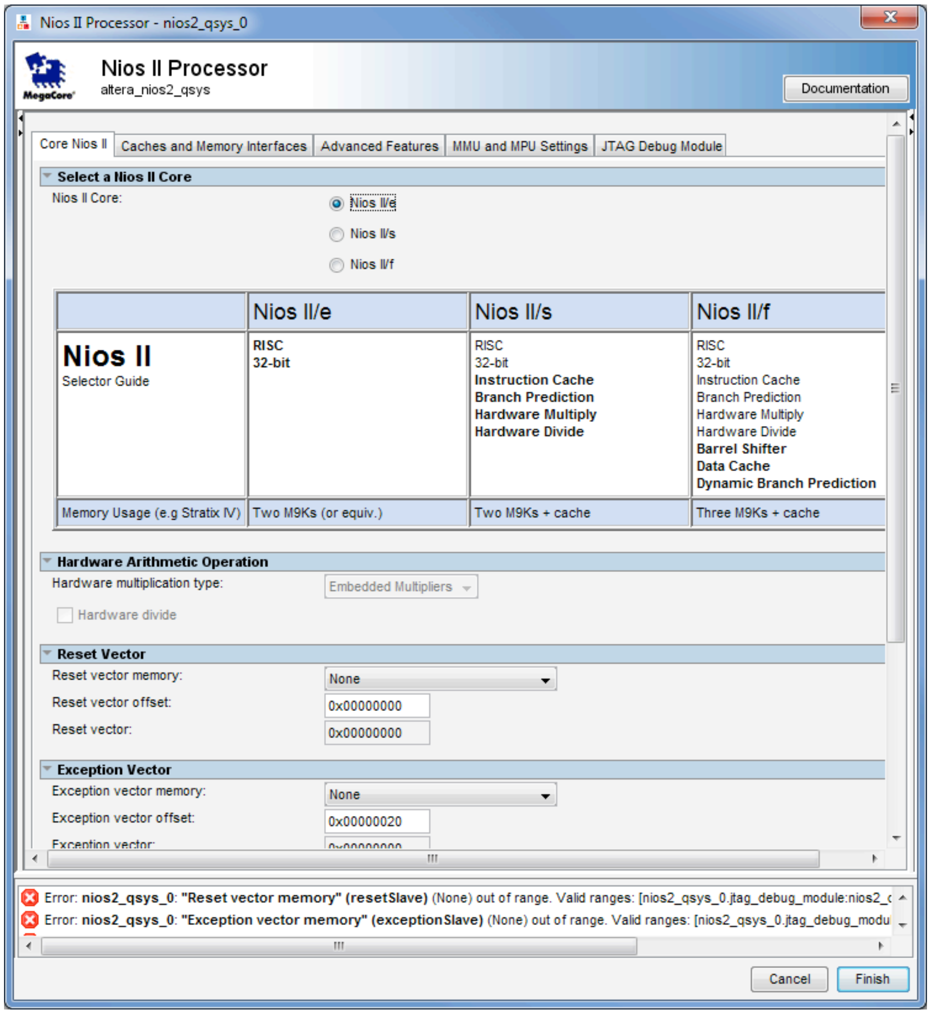
Figure 2. The Clock Settings tab.

The hardware system that will be generated using the Qsys tool runs under the control of a clock. For this exercise we will make use of the 50-MHz clock that is provided on the DE2 board. In Figure 1 click on the Clock Settings tab (near the top of the screen) to bring this tab to the foreground, as illustrated in Figure 2. Here, it is possible to specify the names and frequency of clock signals used in the project. If not already included in this tab, specify a clock named clk_0 with the source designated as External and the frequency set to 50.0 MHz. The settings are made by clicking in each of the three columns: Name, Source and MHz.

Return to the System Contents tab.

Next, specify the processor as follows:

On the left side of the Qsys window expand Embedded Processors, select Nios II Processor and click Add, which leads to the window in Figure 3.



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: None

Reset vector offset: 0x00000000

Reset vector: 0x00000000

Exception Vector

Exception vector memory: None

Exception vector offset: 0x00000020

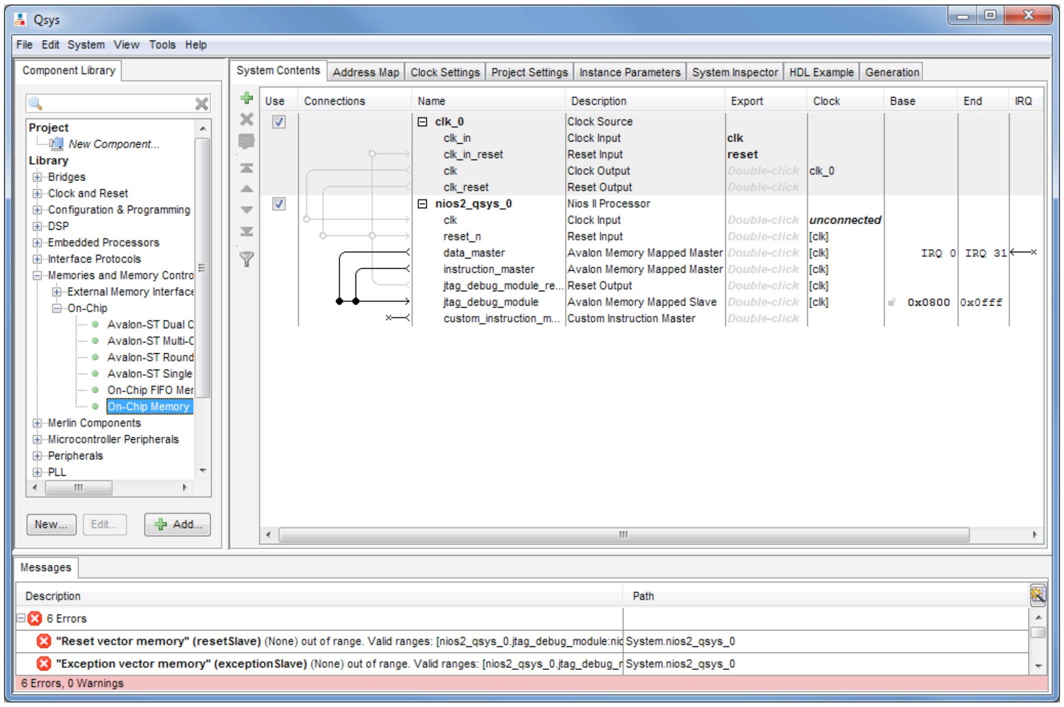
Exception vector: 0x00000000

Error: **nios2_qsys_0**: **"Reset vector memory" (resetSlave)** (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_module:nios2_c

Error: **nios2_qsys_0**: **"Exception vector memory" (exceptionSlave)** (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_modu

Figure 3. Create a Nios II processor.

Choose Nios II/e which is the economy version of the processor. The Nios II processor has reset and interrupt inputs. When one of these inputs is activated, the processor starts executing the instructions stored at memory addresses known as reset vector and exception vector, respectively. Since we have not yet included any memory components in our design, the Qsys tool will display corresponding error messages. Ignore these messages as we will provide the necessary information later. Click Finish to return to the main Qsys window, which now shows the Nios II processor specified as indicated in Figure 4.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		clk_0	Clock Source					
		clk_in	Clock Input	clk				
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Double-click	clk_0			
		clk_reset	Reset Output	Double-click				
✓		nios2_qsys_0	Nios II Processor					
		clk	Clock Input	Double-click	unconnected			
		reset_n	Reset Input	Double-click	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ 0	IRQ 31	
		instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
		jtag_debug_module_re...	Reset Output	Double-click	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]	0x0800	0x0fff	
		custom_instruction_m...	Custom Instruction Master	Double-click				

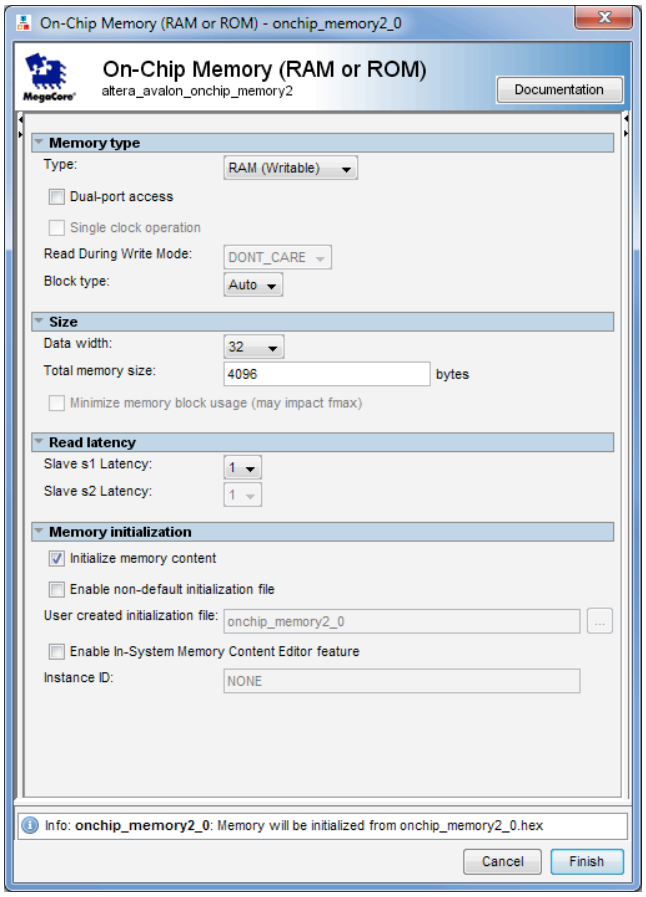
Description	Path
6 Errors	
"Reset vector memory" (resetSlave) (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_module:nic	System.nios2_qsys_0
"Exception vector memory" (exceptionSlave) (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_r	System.nios2_qsys_0

6 Errors, 0 Warnings

Figure 4. Inclusion of the Nios II processor in the design.

To specify the on-chip memory perform the following:

- Expand the category Memories and Memory Controllers, and then expand to select On-Chip > On-Chip Memory (RAM or ROM), and click Add
- In the On-Chip Memory Configuration Wizard window, shown in Figure 5, ensure that the Data width is set to 32 bits and the Total memory size to 4K bytes (4096 bytes) of RAM
- Do not change the other default settings
- Click Finish, which returns to the System Contents tab as indicated in Figure 6


On-Chip Memory (RAM or ROM) - onchip_memory2_0
On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2
Documentation
Cancel
Finish

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: 4096 bytes

☐ Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1

Slave s2 Latency: 1

Memory initialization

☑ Initialize memory content

☐ Enable non-default initialization file

User created initialization file: onchip_memory2_0

☐ Enable In-System Memory Content Editor feature

Instance ID: NONE

Info: **onchip_memory2_0**: Memory will be initialized from onchip_memory2_0.hex

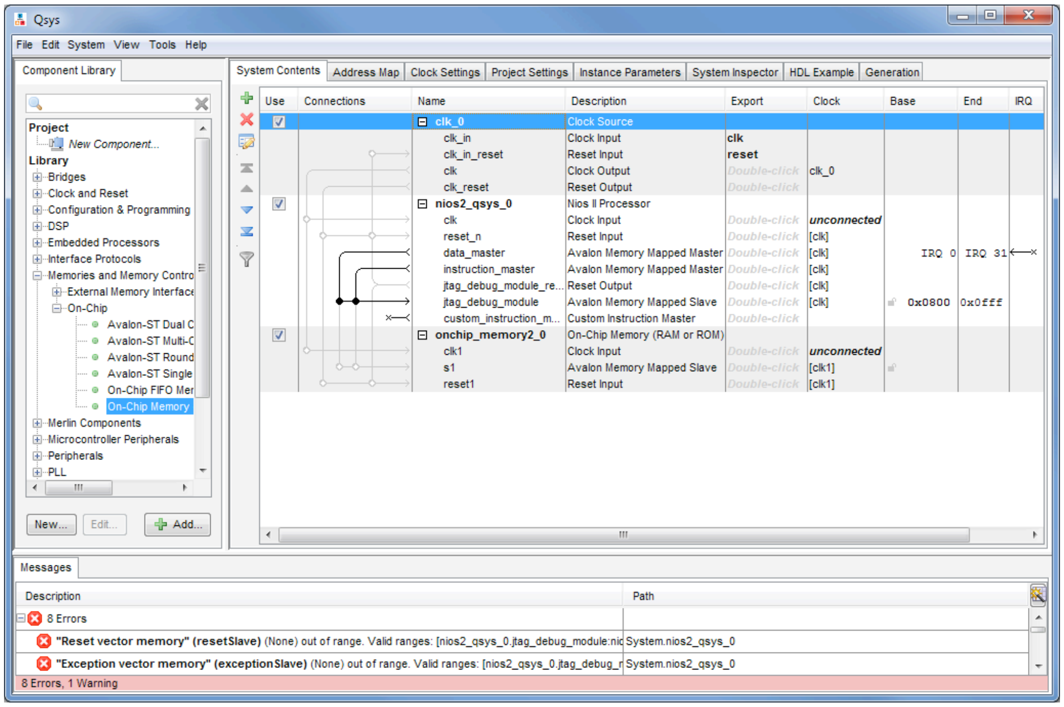
Figure 5. Define the on-chip memory.

Observe that while the Nios II processor and the on-chip memory have been included in the design, no connections between these components have been established. To specify the desired connections, examine the Connections area in the window in Figure 7. The connections already made are indicated by filled circles and the other possible connections by empty circles, as indicated in Figure 6. Clicking on an empty circle makes a connection. Clicking on a filled circle removes the connection.

Make the following connections:

- Clock inputs of the processor and the memory to the clock output of the clock component
- Reset inputs of the processor and the memory to both the reset output of the clock component and the jtag_debug_module_reset output
- The s1 input of the memory to both the data_master and instruction_master outputs of the processor

The resulting connections are shown in Figure 8.

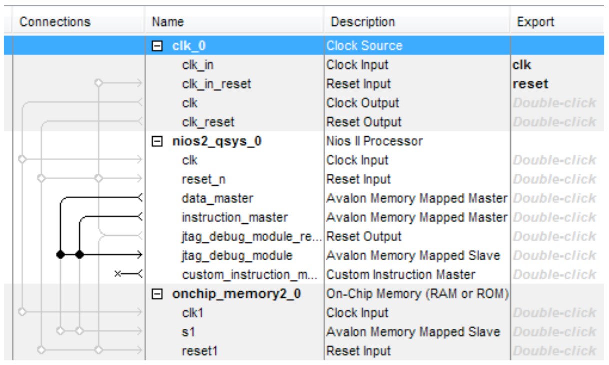


Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		**clk_0**	Clock Source					
		clk_in	Clock Input	**clk**				
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	Double-click	clk_0			
		clk_reset	Reset Output	Double-click				
✓		**nios2_qsys_0**	Nios II Processor					
		clk	Clock Input	Double-click	*unconnected*			
		reset_n	Reset Input	Double-click	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ 0	IRQ 31	
		instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
		jtag_debug_module_re...	Reset Output	Double-click	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]	0x0800	0x0fff	
		custom_instruction_m...	Custom Instruction Master	Double-click				
✓		**onchip_memory2_0**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Double-click	*unconnected*			
		s1	Avalon Memory Mapped Slave	Double-click	[clk1]			
		reset1	Reset Input	Double-click	[clk1]			

Description	Path
8 Errors	
"Reset vector memory" (resetSlave) (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_module:nic	System.nios2_qsys_0
"Exception vector memory" (exceptionSlave) (None) out of range. Valid ranges: [nios2_qsys_0.jtag_debug_r	System.nios2_qsys_0

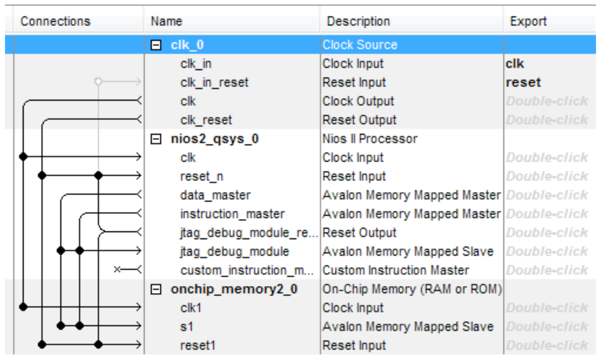
8 Errors, 1 Warning

Figure 6. The on-chip memory included on a DE-series board.



Connections	Name	Description	Export
	clk_0	Clock Source	
	clk_in	Clock Input	**clk**
	clk_in_reset	Reset Input	**reset**
	clk	Clock Output	Double-click
	clk_reset	Reset Output	Double-click
	nios2_qsys_0	Nios II Processor	
	clk	Clock Input	Double-click
	reset_n	Reset Input	Double-click
	data_master	Avalon Memory Mapped Master	Double-click
	instruction_master	Avalon Memory Mapped Master	Double-click
	jtag_debug_module_re...	Reset Output	Double-click
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click
	custom_instruction_m...	Custom Instruction Master	Double-click
	onchip_memory2_0	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	Double-click
	s1	Avalon Memory Mapped Slave	Double-click
	reset1	Reset Input	Double-click

Figure 7. Connections that can be made.



Connections	Name	Description	Export
	clk_0	Clock Source	
	clk_in	Clock Input	**clk**
	clk_in_reset	Reset Input	**reset**
	clk	Clock Output	Double-click
	clk_reset	Reset Output	Double-click
	nios2_qsys_0	Nios II Processor	
	clk	Clock Input	Double-click
	reset_n	Reset Input	Double-click
	data_master	Avalon Memory Mapped Master	Double-click
	instruction_master	Avalon Memory Mapped Master	Double-click
	jtag_debug_module_re...	Reset Output	Double-click
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click
	custom_instruction_m...	Custom Instruction Master	Double-click
	onchip_memory2_0	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	Double-click
	s1	Avalon Memory Mapped Slave	Double-click
	reset1	Reset Input	Double-click

Figure 8. The connections that are now established.

Specify the input parallel I/O interface as follows:

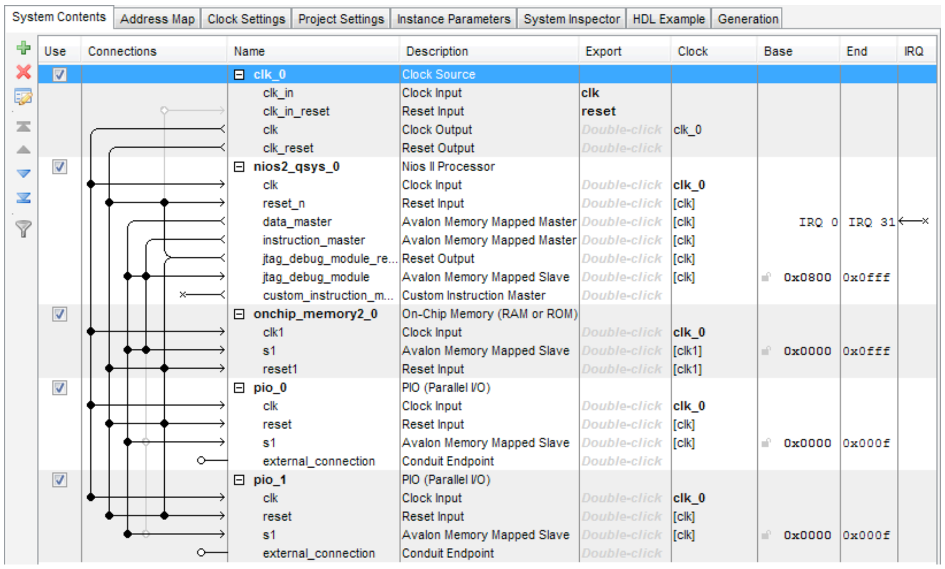
- Select Peripherals > Microcontroller Peripherals > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard
- Specify the width of the port to be **8 bits** and choose the direction of the port to be **Output**.
- Click Finish.

Repeat the process of creating a parallel I/O interface to produce a **single bit** output port.

Specify the necessary connections for the two PIOs:

- Clock input of the PIO to the clock output of the clock component
- Reset input of the PIO to the reset output of the clock component and the jtag_debug_module_reset output
- The s1 input of the PIO to the data_master output of the processor

The resulting design is depicted in Figure 9.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ clk_0	Clock Source					
		clk_in	Clock Input	clk				
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Double-click	clk_0			
		clk_reset	Reset Output	Double-click				
✓		⊟ nios2_qsys_0	Nios II Processor					
		clk	Clock Input	Double-click	clk_0			
		reset_n	Reset Input	Double-click	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ 0	IRQ 31	
		instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
		jtag_debug_module_re...	Reset Output	Double-click	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]	0x0800	0x0fff	
		custom_instruction_m...	Custom Instruction Master	Double-click				
✓		⊟ onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Double-click	clk_0			
		s1	Avalon Memory Mapped Slave	Double-click	[clk1]	0x0000	0x0fff	
		reset1	Reset Input	Double-click	[clk1]			
✓		⊟ pio_0	PIO (Parallel I/O)					
		clk	Clock Input	Double-click	clk_0			
		reset	Reset Input	Double-click	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click	[clk]	0x0000	0x000f	
		external_connection	Conduit Endpoint	Double-click				
✓		⊟ pio_1	PIO (Parallel I/O)					
		clk	Clock Input	Double-click	clk_0			
		reset	Reset Input	Double-click	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click	[clk]	0x0000	0x000f	
		external_connection	Conduit Endpoint	Double-click				

Figure 9. The system with all components and connections.

Note that the Qsys tool automatically chooses names for the various components. The names are not necessarily descriptive enough to be easily associated with the target design, but they can be changed. Right-click on the pio_0 name and then select Rename. Change the name to pio_8_LEDR. Similarly, change pio_1 to pio_1_GPIO_1_2.

Observe that the base and end addresses of the various components in the designed system have not been properly assigned. These addresses can be assigned by the user, but they can also be assigned automatically by the Qsys tool. We will choose the latter possibility. However, we want to make sure that the on-chip memory has the base address of zero. Double-click on the Base address for the on-chip memory in the Qsys window and enter the address 0x00000000. Then, lock this address by clicking on the adjacent lock symbol. Now, let Qsys assign the rest of the addresses by selecting System > Assign Base Addresses (at the top of the window).

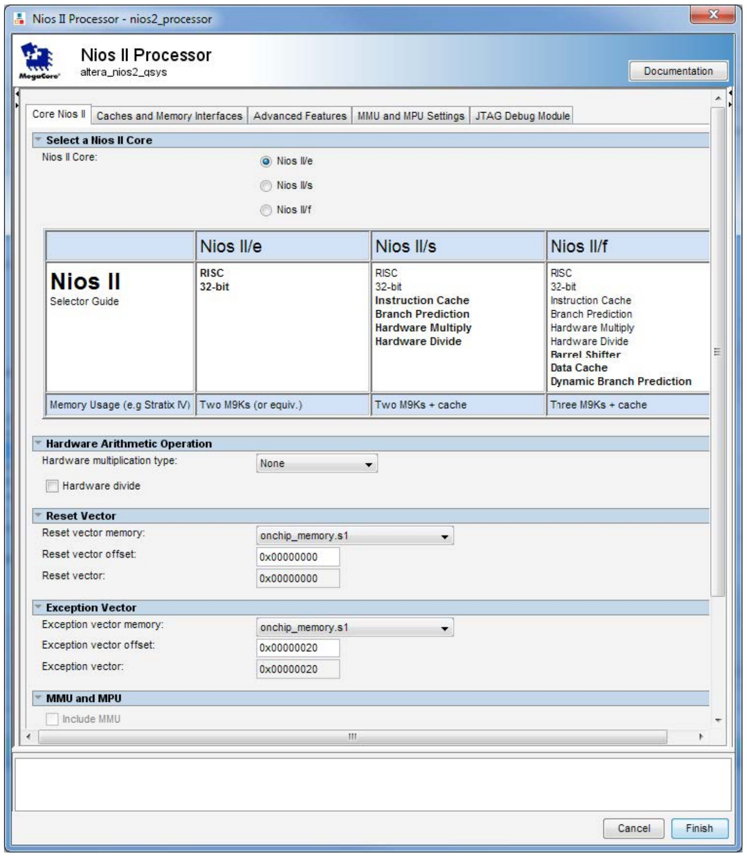
At this point you should make a note of the base addresses here:

8-bit PIO base address ………………………

1-bit PIO base address ……………………..

The behavior of the Nios II processor when it is reset is defined by its reset vector. It is the location in the memory device from which the processor fetches the next instruction when it is reset. Similarly, the exception vector is the memory address of the instruction that the processor executes when an interrupt is raised. To specify these two parameters, perform the following:

- Right-click on the nios2_processor component in the window displayed in Figure 8, and then select Edit to reach the window in Figure 10
- Select 'onchip_memory2_0.s1' to be the memory device for both reset and exception vectors, as shown in Figure 10
- Do not change the default settings for offsets
- Observe that the error messages dealing with memory assignments shown in Figure 6 will now disappear
- Click Finish to return to the System Contents tab



Nios II Processor
altera_nios2_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: None
Hardware divide

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

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

MMU and MPU
Include MMU

Figure 10. Define the reset and exception vectors.

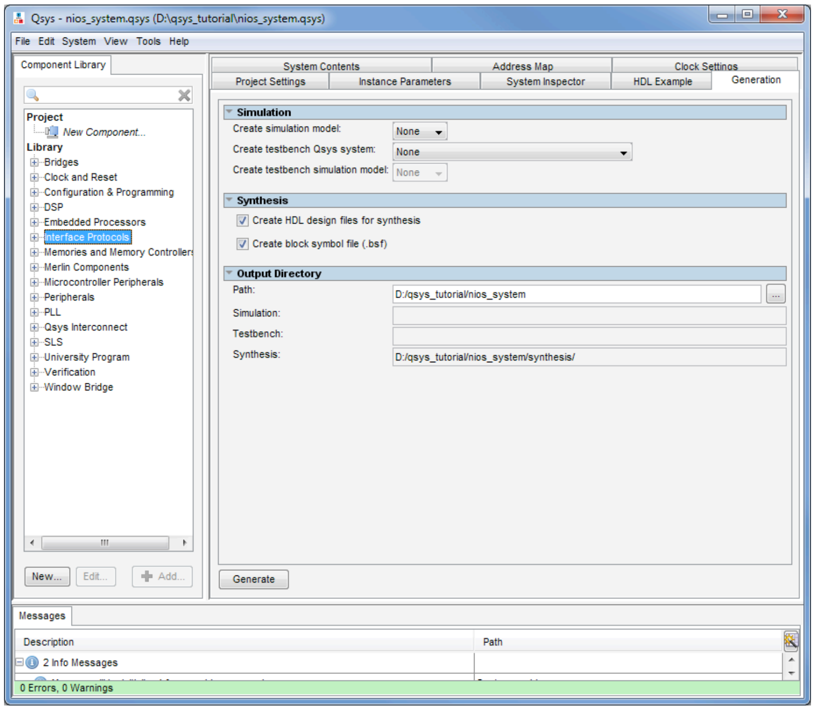
So far, we have specified all connections inside our nios_system circuit. It is also necessary to specify connections to external components, which are LEDs and digital outputs in our case. To accomplish this, double-click on 'Double-click to export' (in the Export column of the System Contents tab) for external_connection of the 8_LEDR PIO, and type the name red_leds. Similarly, establish the external connection for GPIO2, called gpio_1_2. This completes the specification of our nios_system.

Having specified all components needed to implement the desired system, it can now be generated. Save the specified system; you could use the name **nios_system** (**do not** use the same name as your Quartus project or top level verilog file). Then, select the Generation tab, which leads to the window in Figure 10. Select 'None' for the options 'Simulation > Create simulation model' and 'Simulation > Create testbench Qsys system', because in this exercise we will not deal with the simulation of hardware. Click Generate on the bottom of the window. When successfully completed, the generation process produces the message "Generate Completed".

NIOS processor generated correctly ☐

Exit the Qsys tool to return to the main Quartus II window.

Changes to the designed system are easily made at any time by reopening the Qsys tool. Any component in the System Contents tab of the Qsys tool can be selected and edited or deleted, or a new component can be added and the system regenerated. Also note that you can use the 'HDL Example' tab to get a template for instantiation.



Qsys - nios_system.qsys (D:\qsys_tutorial\nios_system.qsys)

Simulation
- Create simulation model: None
- Create testbench Qsys system: None
- Create testbench simulation model: None

Synthesis
- ✓ Create HDL design files for synthesis
- ✓ Create block symbol file (.bsf)

Output Directory
- Path: D:/qsys_tutorial/nios_system
- Simulation:
- Testbench:
- Synthesis: D:/qsys_tutorial/nios_system/synthesis/

Messages: 2 Info Messages

0 Errors, 0 Warnings

Figure 11. Generation of the system.

Please follow the demo video link here for Rama Technology : Schematics Approach

Instantiation of the Module Generated by the Qsys Tool

The Qsys tool generates a Verilog module that defines the desired Nios II system. In our design, this module will have been generated in the nios_system.v file, which can be found in the directory Lab2/nios_system/synthesis (assuming that you named the project folder Lab2). The Qsys tool generates Verilog modules, which can then be used in designs specified using Verilog.

Normally, the Nios II module generated by the Qsys tool is likely to be a part of a larger design. However, in the case of this simple example there is no other circuitry needed. All that is required is to instantiate the Nios II system in the top-level Verilog module, and connect outputs of the parallel I/O ports, as well as the clock and reset inputs, to the appropriate pins on the FPGA device.

The Verilog code in the nios_system.v file is quite large. Figure 12 depicts the portion of the code that defines the input and output ports for the module nios_system. The 8-bit output vector is called red_leds_export. The 1-bit output vector is called gpio_1_2_export. The clock and reset signals are called clk_clk and reset_reset_n, respectively. Note that the reset signal was added automatically by the Qsys tool; it is called reset_reset_n because it is active low.

```
module nios_system (
        input   wire          clk_clk,            //            clk.clk
        input   wire          reset_reset_n       //        reset.reset_n
        output  wire [7:0]    red_leds_export     //     red_leds.export
        output  wire          gpio_1_2_export     //       gpio_1_2.export
);
```

Figure 12. A part of the generated Verilog module.

The nios_system module has to be instantiated in a top-level module (separate Verilog file that you created in part 3(c)) that has to be named Lab2, because this is the name we specified for the top-level design entity in our Quartus II project. For the input and output ports of the Lab2 module we have used the pin names that are specified in the DE2 User Manual (and associated pin assignments CSV file): CLOCK_50 for the 50-MHz clock, KEY for the pushbutton switches, LEDR for the red LEDs and LEDG for the green LEDs. Using these names simplifies the task of creating the needed pin assignments.

Instantiation in a Verilog Module

Figure 13 shows example Verilog code that instantiates the Nios II processor. This code would be placed within the top level Verilog module (Lab2.v) for the project. Note that the instantiation code is a bit different to the code in figure 12, but maintains the same port names.

```
nios_system neos_system_inst
(
.clk_clk(CLOCK_50),
.reset_reset_n(KEY[0]),
.red_leds_export(LEDR[7:0]),
.gpio_1_2_export(GPIO_1[2:2])
);
```

Figure 13 Verilog instantiation of the Nios II processor

Now, you must create a Verilog module which includes an instantiation of the nios_system and:

- connects KEY[0] to LEDG[0],
- connects KEY[0] to the reset input of the NIOS processor,
- attaches the 50MHz clock to the NIOS processor,
- connects the 8-bit PIO output to LEDR[7:0], and
- connects the 1-bit PIO output to GPIO_1[2].

i. Assign the connections from your design to appropriate wires in the DE2 board by **Assignments > Import Assignments** and import the file 'DE2_pin_assignments.csv' (look on Moodle). You can check that pins have been assigned correctly by looking in **Assignments > Pin Planner**.

ii. Finally, before compiling the project, it is necessary to add the **nios_system.qip** file (IP Variation file) to your Quartus II project. This should be located in the "nios_system/ synthesis" folder of your project (you might need to select "All files" as the file type).

iii. **Save your design!** It's a good idea to get in the habit of saving your design on a regular basis, and every so often make a backup, especially when you reach a critical milestone. Repeating effort is always harder.

iv. Your simple NIOS microcontroller is now ready to be compiled to a configuration file for the FPGA. All the HDL files for the design will be processed and a configuration of the Logic Elements of the FPGA will be created, ready to download to the FPGA. There are quite a few HDL files that will be used for both the microprocessor and the peripherals you have included like on-chip memory. The compilation process takes care of creating the file and fitting the design into the chip that you selected at the start of this project. (Note that if your design is too big, or if there are errors the compilation process will terminate. Check the Console window at the bottom of the screen for error messages.
In the console screen you may see quite a few warnings. That is normal and nothing to be too concerned about, but can be important when debugging. Warnings are simply to let you know about a particular condition that is being applied during the compilation process. Many of them can be ignored. Error messages are very important and will cause the compilation process to terminate without completion. You have to resolve any errors before trying again.
Click on the Magenta arrow compile in the tool bar or click on **Processing > Start Compilation**. A dialog box will appear asking you to save changes. Click **Yes** to proceed. The compilation process will take between 5 and 15 minutes depending on the speed of your hardware.
Avoid compiling projects on a network mapped or USB drive. It takes much longer!
Where possible compile to a local hard disk. In the Compilation window of Quartus you will see progress bars indicating the position that each stage in the compilation process has reached. In the

bottom left hand corner of the Quartus screen you will see a percentage completion bar, elapsed time and an indicator depicting that compilation is proceeding. Keep an eye on these. The number of warnings you get may vary, but the message box should tell you Full Compilation was successful! Click **OK**.

You will see at the end of compilation a summary of what was created, what device and family were used and importantly how much of your FPGA resources will be used by this configuration file. It is often a clear indication that something is wrong if the logic utilisation is very low or zero.

Verilog design compiles correctly ☐

f) Programming your NIOS Microprocessor in Assembler

The next step is to write an assembler program to run on the NIOS system that you have just designed. **Your assembler program can be based on the program you used last week (a .s file)**. Any simple text editor can be used to create the assembler program. I recommend that you use WordPad which you can find by searching "WordPad" in the Windows Start menu**.** Your modified assembler program should initialize registers to contain the base address of the 8-bit and 1-bit PIOs and then enter an infinite loop which writes the pattern 0x033 to the 8-bit PIO connected to the red LEDs followed by writing a '1' to the 1-bit PIO and finally writing a '0'to the 1-bit PIO.

Remember that you recorded the base addresses for the PIOs earlier in the lab.

g) You will now use the Altera Monitor Program to assemble your program and to run it.

i) Start the Monitor Program either using the desktop shortcut or by searching for it using the Windows Start menu.

ii) Create a new project by selecting **File > New project**. Insert your current project directory and give the project a unique name including 'mon' to identify it as relating to the monitor. Then click Next.

iii) In the 'Specify a system' window select 'Custom System'. Then locate the .qsys and .sof files in your project folder. Then click **Next**.

iv) For 'Specify a program type' choose 'Assembly Program'. Then click **Next**.

v) In ‘Specify program details’ click on **Add** and then Select your assembler .s program. Then click **Next**.

vi) In the ‘Specify system parameters’ window, check that Host Connection is ‘USB-Blaster [USB-0]’, Processor 'nios2_qsys_0' and Terminal device ‘<none>’. Then click **Next**.

vii) ‘Specify program memory setting’ you should leave the values as set. Then click **Finish**.

viii) Then accept the offer to download the hardware design to the DE2 board. If successful then click on **OK**.

ix) To complete the system we are now going to compile your .s file using the **Actions > Compile** menu item or click on the associated icon (you can check out the function of the icons by ‘hovering’ the cursor over them).

x) When the code is successfully compiled you should download it to the DE2 system using the **Actions > Load** menu item or click on the associated icon. If all goes well you will see a window displaying your assembler program. If you get an error message ‘Resetting and pausing target processor : FAILED’ then the chances are that there is a problem with your NIOS processor. Check that the clock is attached correctly and that it is receiving the correct reset signal.

Complete the Malaysia Version Lab Sheet : Link Here