# ECE3073 Computer Systems

Laboratory Session 2

Nios assembler and instruction timing

Week 3 – Semester 1 2012

## 1. Objectives

This laboratory exercise introduces the Altera Quartus IDE and the System On a Programmable Chip (SOPC) Builder. 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 can 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 the System On a Programmable Chip Builder
- 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 store 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

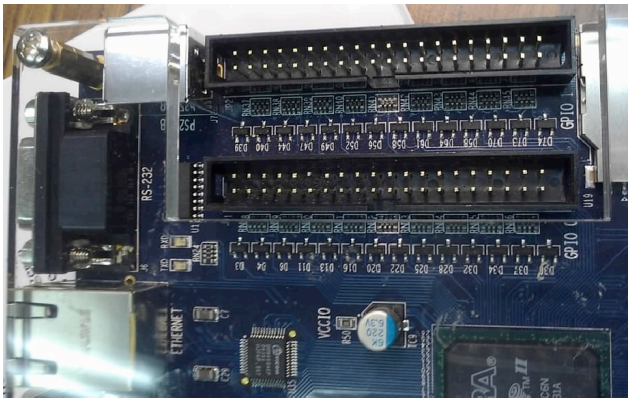
You must complete this preliminary work before attending the lab session.

a) For the 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.




Demonstrator initial satisfactory completion before attending the lab ☐

**3. Entering the FPGA 'Hardware' design**

a) Open Quartus using the desktop short-cut or **Windows Start  Programs > Altera_10.1 > Quartus II 10.1sp1 > Quartus II 10.1**. If a project 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 the SOPC builder.

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 (if you did not create one then do that now). 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. 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 our 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 10.1 does not automatically default to your project directory! You 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

i. First step is open up the System On Programmable Chip builder or SOPC builder either via its icon or **Tools > SOPC Builder**. In a new project, where you are creating a NIOS microprocessor on your FPGA from scratch, you will see the 'Create new System' dialog box. Type in a name for your NIOS Micro processor and don't use the same name as your project! This causes circular arguments and conflicting file names to be generated. Choose something different and unique. You should choose **Verilog** as the HDL that will be used to write the definition of your microprocessor. Remember that a NIOS processor is simply a configuration file for the FPGA, written entirely in Verilog or VHDL. The processor only exists as a configuration of the FPGA. It can be erased and modified like any other configuration. It just so happens that the NIOS configuration makes the FPGA work like a stored program architecture microprocessor. Click **OK** for the Create New System window.
   The SOPC Builder allows you to specify exactly what components to put into your NIOS microcontroller. You can customize just about everything from the size of the cache to what peripherals and memory you want to include. All of this information gets translated by the SOPC builder into a Verilog or VHDL file that describes the system you want to create.

ii. To begin with, you need to have a NIOS microprocessor. You can find the NIOS II processor in the **Component Library > Processors >Nios II Processor**. Click **Add** to add this processor to your design. A new dialog box opens for the NIOS II Processor which includes all of the ways in which the NIOS processor may be customized.

iii. For this first example you are going to include all of the memory and the processor on the FPGA. That means we won't be using any connections to the external memory chips. The wires to these chips are already there, but they will remain in the tristate condition (high impedence, as if they don't exist). Because you are going to use 'On Chip' memory, you want the processor to be as small as possible. In other words you want it to use as few resources of the FPGA as possible. For this you want the processor core to use up as few logic elements, or LE's as possible. A NIOS II/e core uses between 600 to 700 LE's. It's a simple RISC architecture with nothing fancy built in. By contrast, the NIOS II/f uses between 1400 and 1800 LE's and has Instruction Cache, Branch Prediction, Hardware Multiply/Divide, Data Cache and a whole lot more fancy stuff! You will be using the basic procesor. Click on the **Nios II/e** radio button. For the moment that is all you need. But you can see a warning message in the message box at the bottom of the dialog. The message is warning you that the Reset Vector and Exception Vector have not been associated with a memory location yet. You will need to do that before you try and compile this design or it just wont work. Every processor has to know where to start processing instructions. This is called the Reset Vector. Some processors have their Interrupt service routines located at a fixed memory location. The Nios procesor allows the user to specifiy where to put the interrupt service routines. The start of this block is called the Exception Vector. You will fill these in once you add memory to your microcontroller system. For now click the **Finish** button. The dialog box should close and return you to the SOPC builder system. A new device, cpu_0 has been added to the system. What you are starting to create is the system memory map. In the message box you will be reminded that you still have to add the reset and exception vectors. This is nothing to panic about. Just a friendly reminder.

iv. Your microprocessor needs memory to store both the program, and any variables that it uses. You are going to create the simplest of systems with as few external connections as possible for this exercise. From the Component Library select **Memories and Memory Controllers > On-Chip > On-Chip Memory (RAM or ROM).** A typical microprocessor needs some RAM and some ROM! If this were an actual ASIC microprocessor you would need to add a RAM chip for data and a ROM chip for program, then connect wires to interface them. You are going to do the same thing here but instead of adding chips, you will use part of the leftover FPGA space for RAM and some for ROM. All of the connections to the memory will be internal to the FPGA.

One small note, as you may have already surmized. You are actually pretending to create ROM. It is afterall created on a fully programmable FPGA and therefore isn't true ROM! But for the purposes of an introductory discussion to microprocessors, data goes in RAM, program goes in ROM.

**Double Click On-Chip memory (RAM or ROM)**. The dialog box for this component should appear. You can select a number of features. Start by selecting RAM (the default), with a data width of 32 bits and a total memory size of 1024 bytes. You won't be creating too many variables so you don't need to use huge amounts of space unnecessarily. Click **Finish** to return to the SOPC builder. You know that you chose RAM, but the name of the component you just added is very generic. You should change it to be more useful. Right click on the component name for the memory and select **Rename** from the menu. Edit the name to Onchip_RAM. Remember spaces

will cause a problem! Spaces should not be used in NIOS or Quartus or anything to do with them. In either Pin names, component names or in directory paths! Spaces cause problems for the compiler in trying to locate files or identifiers. Repeat the process of adding on chip memory but this time add 8192 bytes of ROM. Call it Onchip_ROM.

v. Your microcontroller system is starting to take shape. You have the processor core, ROM to store the program and RAM for variables. You need to go back to the processor now and add the reset and exception vectors. Double click on the **cpu module** under 'Module' in the system design window. You should see the NIOS II processor dialog again. Now you can select Onchip_ROM for both the Reset Vector and the Exception Vector. This simply says the program counter will start at an address in ROM, and all the interrupt service routines are also located in ROM. Not surprising since they are all part of the program and should be in ROM. Click **Finish** to return to the SOPC builder.

vi. Your simple microcontroller system is nearing completion. What you don't have at the moment is a connection to any form of I/O device. Your system can sit and crunch instructions all day long, but right now, it cannot affect the outside world. You need to add at least some form of I/O. You are going to add a means of communicating with our NIOS microprocessor via PIO parallel ports. From the Component Library select **Library > Peripherals > Microcontroller Peripherals > PIO (parallel I/O).** In the PIO (parallel I/O) widow select width of 8 bits and a direction of output. Then click **Finish.** Repeat the previous process and add a single bit output port. You can rename them to improve your documentation (the 8-bit port will be driving red LEDs and the 1-bit port will be acting as a flag bit to indicate when the processor is executing a certain group of instructions). You now have a complete Microcontroller system. That is a Microprocessor (NIOS) somewhere to store a program (ROM), somewhere to store variables (RAM), and some way of communicating with the outside world (8-bit PIO and 1-bit PIO).

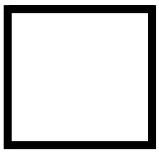
vii. You may notice two messages in the console window. All the error messages and warnings appear in the console window. These two messages are talking about the memory map. Specifically, there is a problem with the allocation of addresses in the memory map. To fix this, click on **System > Auto-Assign Base Address** (NOT THE SYSTEM GENERATION TAB). Go back to the SOPC Builder screen and note which of the Base and End numbers have changed. When you write your assembler program to interact with the PIO I/O ports you will need to know the base address assigned to each. Make a note of the base addresses here:

8-bit PIO base address ……

1-bit PIO base address ……..

When you click Generate, your NIOS microcontroller system will be generated as either a Verilog or VHDL file (whichever you selected at the start). The whole thing, including the microprocessor, memory and I/O connections are nothing more than a (very big) HDL file! The HDL file becomes part of the configuration of the FPGA. Click on the **Generate** button and your system should compile. It will take a few minutes. You will be asked if you want to save the design – accept and give your design a meaningful

name. If the process is successful, you will get a message saying 'System generation was successful' at the end of a whole lot of other messages.



NIOS processor generated correctly

If system generation was not successful, something is amiss from the previous steps. If you have an error message in the console window you must clear the message before compiling or you won't be able to start the process.

If you are successful, click **Exit**. Don't worry about saving; all the files are saved automatically.

viii. When you exit, go back to the Quartus software and the .v file you began editing earlier. Open the file that ends in inst.v to find a prototype of how you will instantiate your NIOS processor in your design, something like:

```
//Example instantiation for system 'rarLab2processor'

rarLab2processor rarLab2processor_inst
             (
             .clk_0       (clk_0) ,
             .out_port_from_the_FlagPIO       (out_port_from_the_FlagPIO),
             .out_port_from_the_LEDPIO  (out_port_from_the_LEDPIO),
             .reset_n     (reset_n)
      );
```

ix. Now you must create a Verilog module which:

- 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[2:2].

x. 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 in altera_projects). You can check that pins have been assigned correctly by looking in **Assignments > Pin Planner**.

**xi.** **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.

xii. 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.

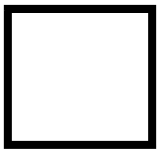
In the console screen you may see quite a few warnings. That is normal and nothing to be too concerned about. Warnings are simply to let you know about a particular condition that is being applied during the compilation process. Many of them can be ignored. Error messages are very important and will 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 over a network or to a 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 right 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.



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 under the **Windows Start** menu **Programs > Accessories > WordPad.** 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 selecting Windows Start **Programs>Altera>University Program>Altera Monitor Program**.

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 .ptf 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 'cpu_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 **YES**.

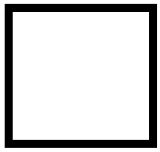
ix) To complete the system we are now going to compile your .s file using the **Actions > Compile** menu item or click on the associated icon (you can check out the function of the icons by 'hovering' the cursor over them).

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.

h) Connect one input of your oscilloscope to the DE2 board. The GROUND clip to the pin you identified as ground in the preliminary work and the probe to GPIO_1[2]. Single step through your program to ensure that it works correctly (the oscilloscope should display the voltage level change on the GPIO_1[2] pin when a '1' and a '0' are written to the associated PIO bit). When you run your program you should now see a regular pulse on the oscilloscope. Measure the high period of the pulse on GPIO_1[2] and record it here:

High period ……

Check that an assembler program was run 

Next insert 5 assembler instructions between the write to make GPIO_1[2] go high and the write to make it go low. Measure the increased high period of the output pulse and work out how many EXTRA clock pulses were required for the 5 instructions and then clock pulses per single instruction.

R-type (add r2, r3, r4) High Period …… Extra clock pulses …… Pulses/instruction ……..

J-type (jmpi label) High Period …… Extra clock pulses …… Pulses/instruction ……..

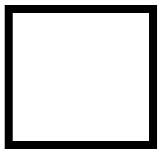
I-type (ldwio r2, 0(r3)) High Period …… Extra clock pulses …… Pulses/instruction ……..

Conditional Branch (branch taken) (bne r2, r0, label)

High Period …… Extra clock pulses …… Pulses/instruction ……..

Conditional Branch (branch not taken) (bne r2, r0, label)

High Period …… Extra clock pulses …… Pulses/instruction ……..



Satisfactory results

When you have complete the lab or when time is running out then start the assessment quiz for this laboratory exercise. The demonstrator will enter the password and record your mark (out of 5, one for each check box).

Ensure that your mark for this exercise is entered before you leave the lab.

Andy Russell 16/02/2012