Nios 3.1 Exercise Manual

NB: Throughout these labs it is important to enter the names of any peripherals or memories within SOPC Builder EXACTLY as shown. These are referenced by pin settings and C–Code examples and as such are CASE SENSITIVE.

Lab 1
My First Nios System

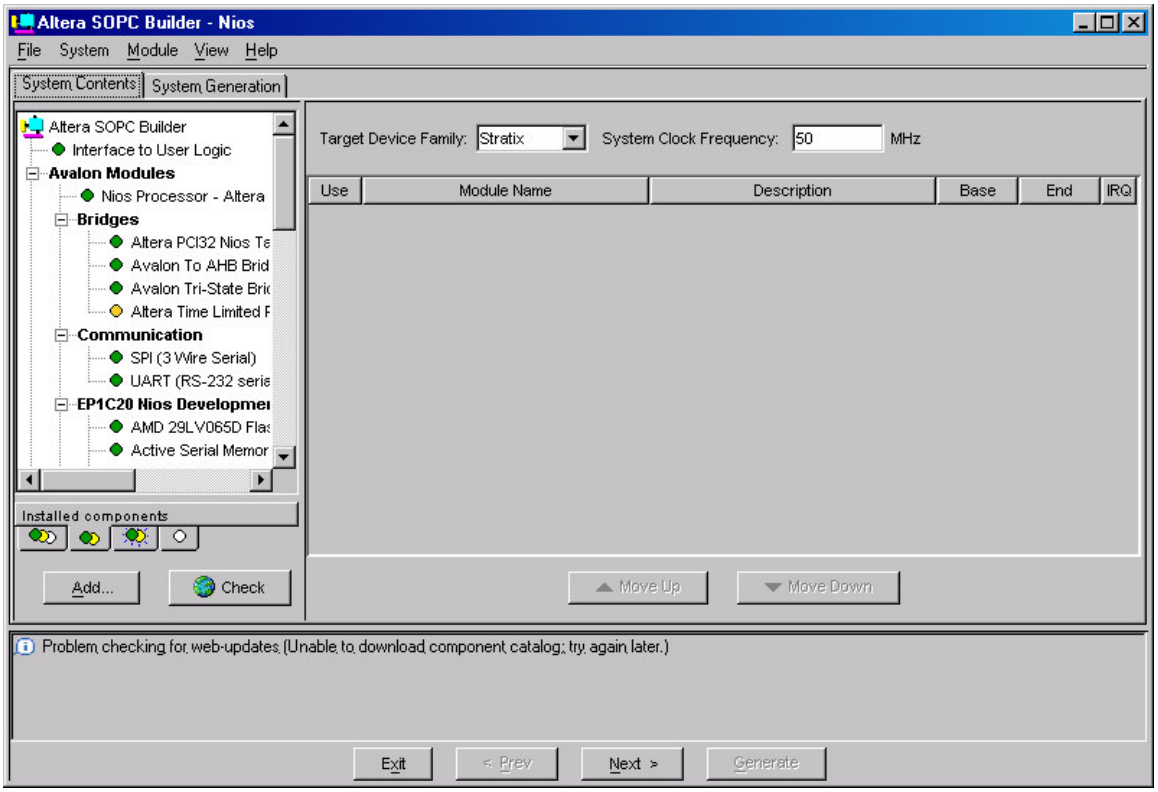
Hardware set up requirements:

ByteBlaster cable connected to LPT and ByteBlaster connection on the board
Serial cable connected to COM and Console connection on the board
Power supply connected to the board

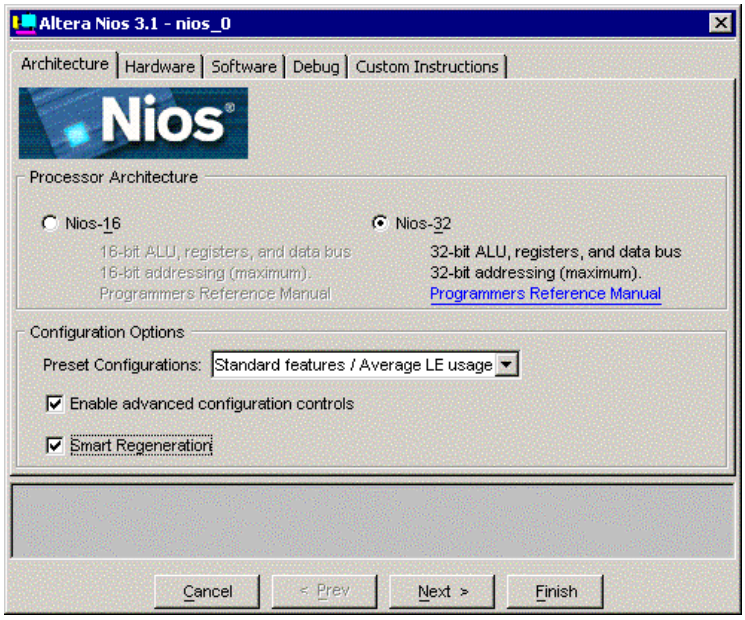
1. Start the Quartus II Software and open the Quartus II project via the menu option **File => Open Project…** Browse to directory **c:\nios_lab** and select the project **nios.quartus**. Click **Open**.

2. This series of exercises is designed for use on multiple editions of Nios development boards. This step involves running a TCL script to set the correct device settings and pinout for the board being used. From the **Tools** menu select **Tcl Scripts** then select the relevant script from the **project** folder and click **Run**. If unsure of which script to run please consult the workshop co-ordinator.

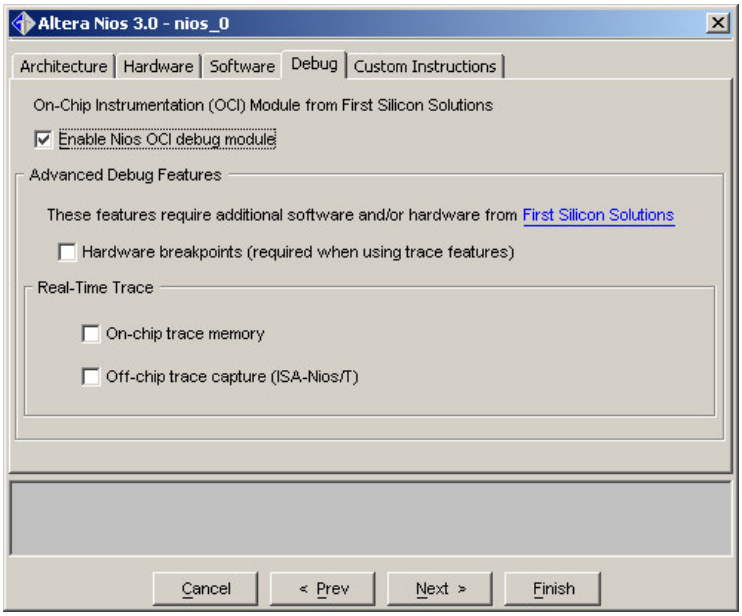
3. Start SOPC Builder via the menu option **Tools => SOPC Builder…** In the next window provide the system name **nios** and choose **VHDL** as the implementation language. Choose. The blank SOPC builder window opens. Set the Target Device Family to **Stratix** or **Cyclone** depending upon the board being used and ensure that the System Clock Frequency is set to **50 MHz**.



4. From the left hand window pane select **Nios Processor** and click **Add**. For the processor architecture select **Nios-32**. Select the preset configuration **Standard features / Average LE usage**. Tick the box marked **Enable advanced configuration controls** and also tick the box marked **Smart Regeneration**. Click the **Debug** tab.

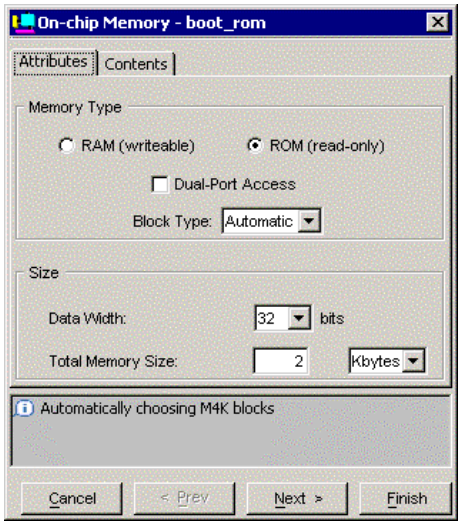


5. Tick the box marked **Enable Nios OCI Debug Module** and click **Finish**.

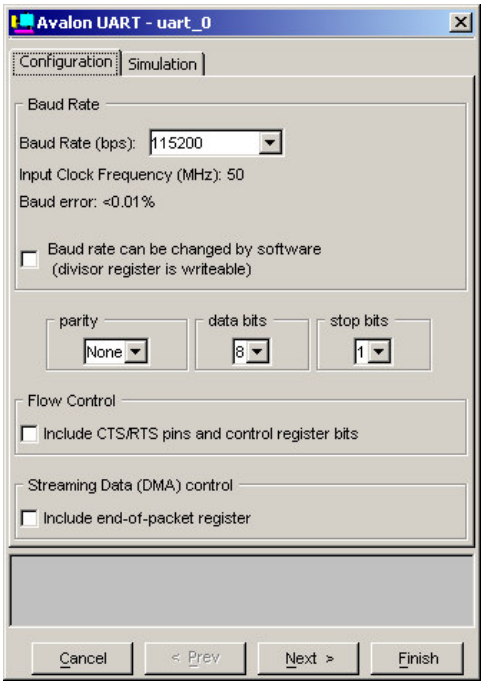


6. Rename the processor by **right clicking** on the current name and **selecting rename**. Type in **cpu** and hit enter.

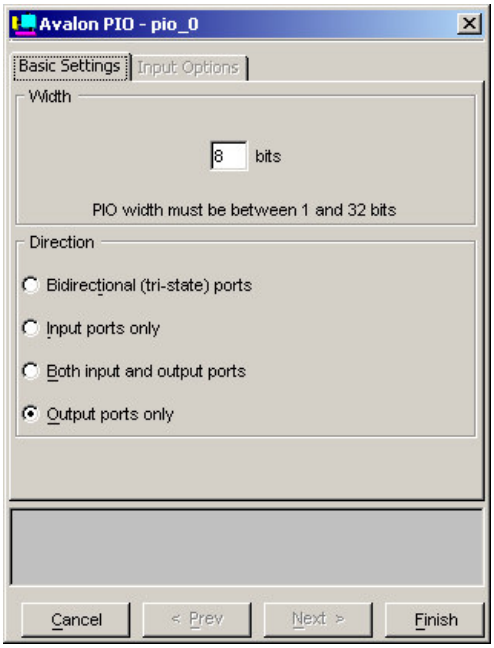
7. From the left hand window pane select **On-Chip Memory (RAM or ROM)** from the **Memory** section and click **Add**. Select **ROM (read only)** with a Data Width of **32** bits and Total Memory Size of **2 Kbytes**. Click **Next.** Select **GERMS Monitor** and click **Finish**. Rename the memory to **boot_rom**



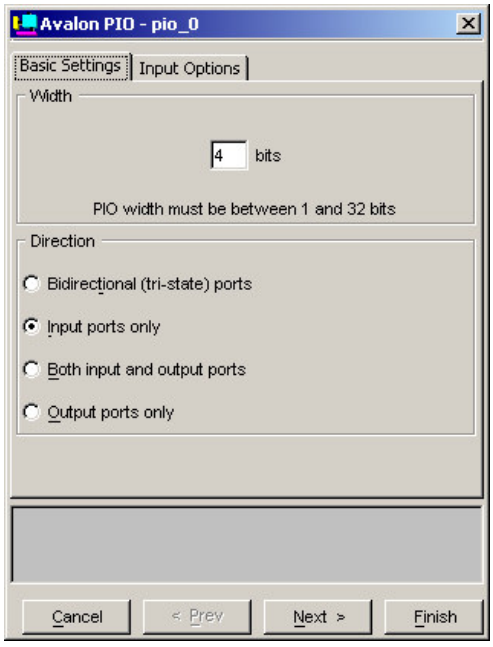
8. From the left hand window pane select **UART (RS-232 serial port)** from the **Communication** section and click **Add**. The default baud rate should be **115200**. Accept the defaults. The screen should appear as shown. Click **Finish**. Rename the peripheral to **uart1**.



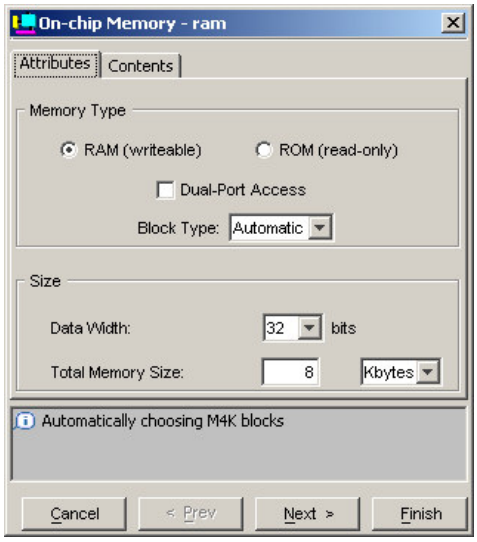
9. From the left hand window pane select **PIO (Parallel I/O)** and click **Add**. Enter a width of **8** bits, with **output ports only**. Click **Finish**. Rename this peripheral **led_pio**.



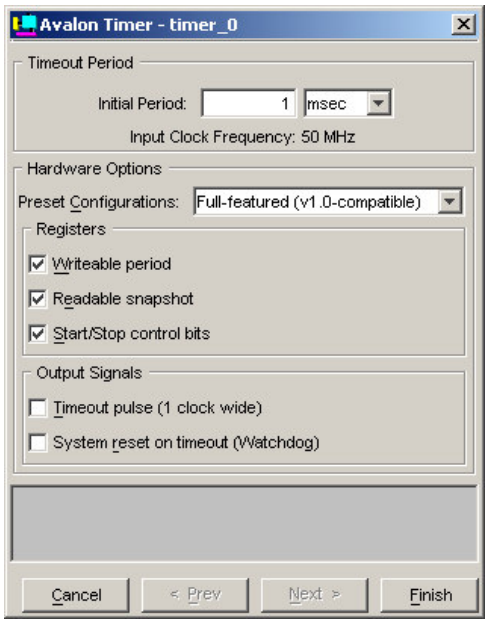
10. From the left hand window pane select **PIO (Parllel I/O)** and click **Add**. Enter a width of **4** bits, with **Input ports only** and click **Finish**. Rename this peripheral **button_pio**.



11. From the left hand window pane select **On-Chip Memory (RAM or ROM)** from the **Memory** section and click **Add**. Select **RAM (Writeable)** with a Data Width of **32** bits and Total Memory Size of **8 Kbytes**. Click **Finish.** Rename the memory to **ram**.



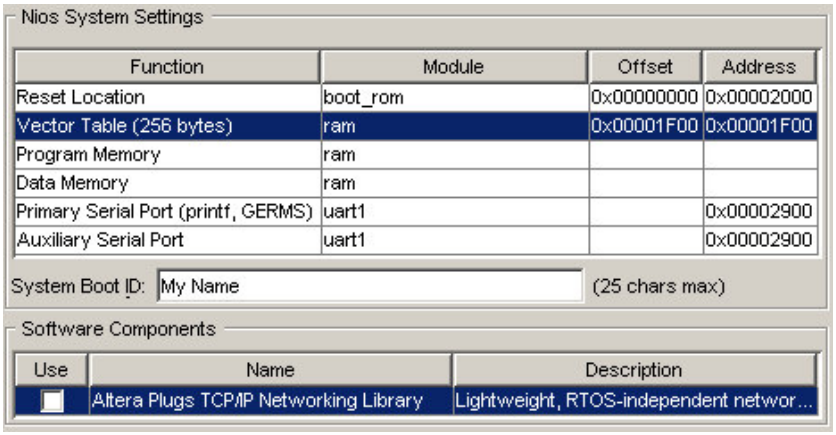
12. From the left hand window pane select **Interval timer** from the **Other** section and click **Add**. Accept the default options by clicking **Finish** and rename the peripheral to **timer1**.



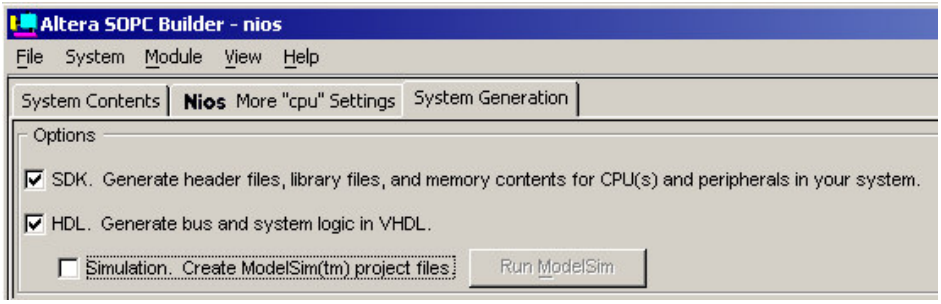
13. To ensure that all base addresses are valid, right click on any of the base addresses in the table and select **Auto-Assign Base Addresses**. This step should result in the following view within SOPC Builder.

Use	Module Name	Description	Base	End	IRQ
☑	⊞ **cpu**	Nios Processor - Altera Corporation	**0x00002800**	0x000028FF	
☑	⊞ **boot_rom**	On-Chip Memory (RAM or ROM)	**0x00002000**	0x000027FF	
☑	⊞ **uart1**	UART (RS-232 serial port)	**0x00002900**	0x0000291F	**16**
☑	⊞ **led_pio**	PIO (Parallel I/O)	**0x00002940**	0x0000294F	
☑	⊞ **button_pio**	PIO (Parallel I/O)	**0x00002950**	0x0000295F	
☑	⊞ **ram**	On-Chip Memory (RAM or ROM)	**0x00000000**	0x00001FFF	
☑	⊞ **timer1**	Interval timer	**0x00002920**	0x0000293F	**17**

14. Click **Next**. This page is where several system settings are made. Untick the **Altera Plugs TCP/IP Networking Library**. Set the **Reset Location** to **boot_rom**. Set the **Program Memory, Data Memory,** and **Vector Table** selections to **ram**. Set the offset for the **Vector Table** at **0x00001F00**. Change the **Primary Serial Port (printf, GERMS)** to **uart1**. Type in your name for the **System Boot ID**. Click **Next**.



15. All checkboxes should be checked by default. Uncheck the **Simulation** box.



16. Now click **Generate**. SOPC Builder will now produce the parameterized Nios processor system. Once completed click **Exit**.

17. Now start compilation in Quartus by selecting Start Compilation from the Processing menu.

Do not wait for compilation to complete. We will continue from this point during the next lab.

END OF LAB 1

Lab 2
Software Flow

1. We will now download the Nios design created in the previous lab to the Nios development board. Within Quartus select the **Programmer** from the **Tools** menu. Tick the **Program/Configure** checkbox and then click the Start Programming icon  .

2. Open a Nios SDK Shell via the Windows Start Menu (Start, Programs, Altera, Nios Development Kit 3.10, Nios SDK Shell). Change directory with the command **cd c:/nios_labs/cpu_sdk/my_src**.

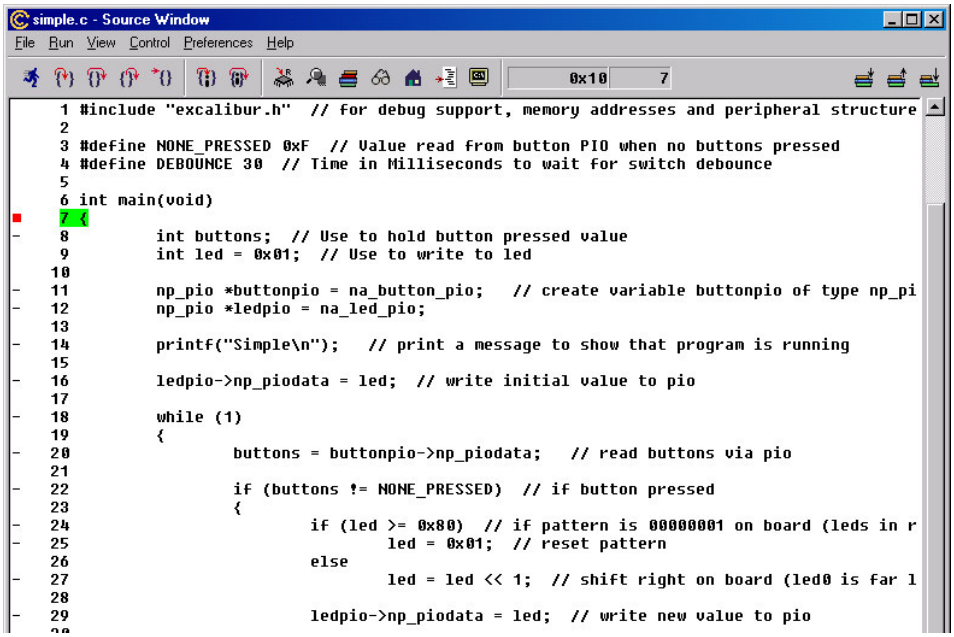
3. Enter terminal mode with the command **nr –t**. Now press the **Escape** key on the PC to reset GERMS within the Nios core on the development board. Nios should respond by sending the Boot ID (your name) to the SDK Shell. Press **Ctrl-C** to exit terminal mode.

4. Compile example code with the command **nb simple.c**.

5. Download the program with the command **nr simple.srec**. Each press of any of the buttons on the board should shift the lit LED one space right. When finished press the **CPU Reset** button on the board and press **Ctrl-C** on the PC.

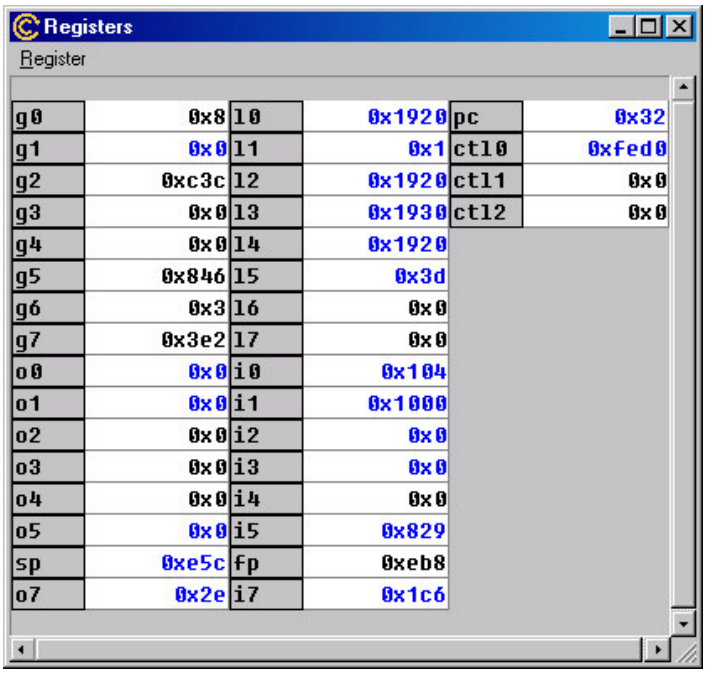
6. Now compile the same code without optimisation with the command **nb –O0 simple.c**. *NB: O0 is the capital letter O followed by the digit zero.*

7. Now start the debugger with the command **nd simple.srec**. The debugger will launch, connect to the target and download the program ready for debug.

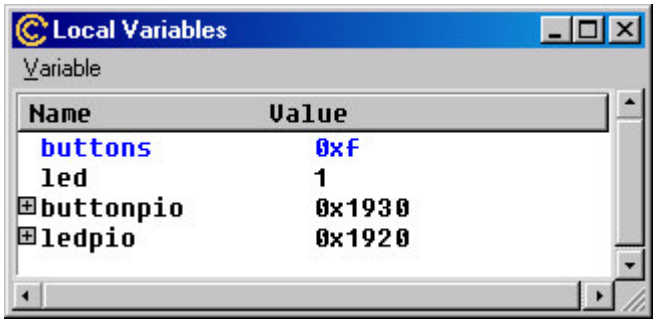


8. The following windows should be open. **Registers**, **Memory** and **Local Variables**. Move and resize these windows to make viewing easier. If not, they can be opened using the view menu.

9. Set a breakpoint on **line 20**. Simply place the cursor on top of **20** and click. The cursor changes to a circle when positioned correctly. You should see a red dot appear next to line 20. Click on the **continue** button **.** Notice that execution stops at this line and the PC and other register values have changed to blue to denote a change in value.



10. Click **next** . Notice that the buttons have been read as shown in the Local Variables window. Change this view to hexadecimal by right-clicking on the buttons value and selecting **Format =>Hex**.



11. Click **next** again. As the if expression is false (no buttons pressed) the statements within the curly braces are not executed. Click **continue**.

12. Hold down switch **SW3** on the board and click **next**. A new value is stored in the variable **buttons**.

13. Click **next**. We now dive into the if curly braces since the condition is true.

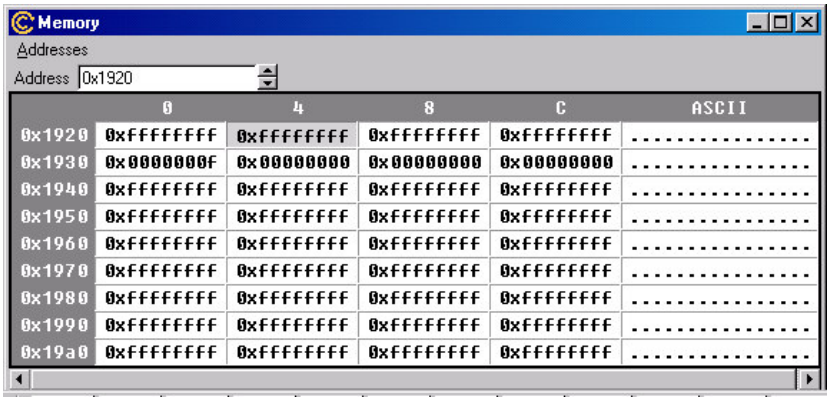
14. Click **continue** and notice the LEDs on the board change. Click **next**.

15. Right-click on **buttons** within the **local variables window** and select **edit**. Now change the value to **0xe** and press enter. Click **next**. The if statement is executed as true because we forced that condition.

    This is useful for emulating external hardware events or other conditions that are difficult to replicate.

16. Change the range shown in the **memory** window by editing the value shown in the **Address** box. Change it to the value shown for **ledpio** in the **local variables** window.

17. Type values in the memory window at location of **ledpio** between **0x00 and 0xff**. Type in new values and hit enter. Notice the LEDs change state.



Because the LED PIO is a memory mapped peripheral, by editing the correct memory location we can write directly to the PIO and change the status of the LEDs.

18. Close the debugger by selecting **File => Exit**. A new dialog box appears to ask if this is really what you wanted, click **yes**.

19. Close the Programmer window within Quartus. When asked if the nios.cdf file should be saved click **No**.

IF YOU HAVE TIME….

Simulate the running of the Simple.c program in ModelSim using the test suite automatically generated by Modelsim.

20. Return to SOPC Builder and edit the ram component so that it remains writeable but is initialised with the build of the **simple.c** file.

21. Edit the uart1 component so that the input character stream **g0↵** is simulated. This input will be interpreted by the GERMS monitor causing execution to continue from the ram component which has been initialised with our program during the step above. **NB** this assumes that the ram component appears at base address **0x0**. If this is not the case then the input character stream should be edited accordingly.

22. On the generation page of SOPC Builder enable simulation and re-generate the System.

23. Once generated launch ModelSim from SOPC Builder. Once started enter the command **s↵** to compile and load the testbench. Then enter the command **w↵** to display a wave window with appropriate signals.

24. Run simulation with the command **run 10us**. Note that UART communication is echoed to the ModelSim console and that the simple.c program was run, indicated by the display of the “Simple” message.

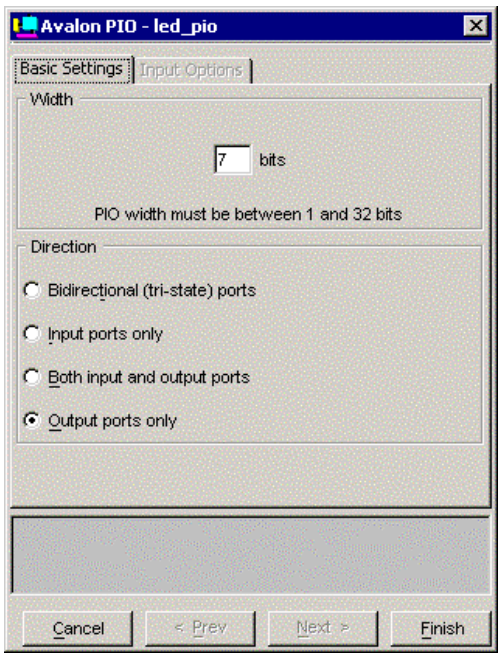
25. Try to find the point in the simulation where the Simple.c program started execution. Hint: look for the point where the instruction address matches the base address of the component where the simple.c program is stored.

END OF LAB 2

Lab 3
User Peripheral

1. Return to Quartus and select **SOPC Builder** from the tools menu.

2. Double-click on the **led_pio**. Cange the number of output bits to **7** and click **Finish**. One of the LEDs on the board will now be driven from a PWM peripheral that we are about to create.



3. From the left hand window pane select **Interface to User Logic** and click **Add**.

4. Select **Avalon Register Slave** as the Bus Interface Type. Check the box marked **Import Verilog, VHDL, EDIF or Quartus II Schematic file** and click **Add**.

5. Browse to the directory **c:\nios_labs** and select the file **avalon_pwm.vhd**. Click **Open**.

6. Click **Read port-list from files**. This reads in the ports of the imported design. Now fill in the Type column as shown below. You do this by clicking in the type area and picking the correct signal type. Click **Next**.

Port Name	Width	Direction	Shared	Type
clk	1	input		clk
writedata	32	input		writedata
byteenable_n	4	input		byteenable_n
cs	1	input		chipselect
write_n	1	input		write_n
addr	1	input		address
reset_n	1	input		reset_n
readdata	32	output		readdata
pwm_out	1	output		export

7. Select **Simulate User Logic** and click **Next**.

8. The default values for Setup, Wait and Hold Cycles should be set to **0**. Click **Add to System**.

9. Rename the peripheral to **my_pwm**.

Leave SOPC Builder Open. We will return to this stage during the next lab.

END OF LAB 3

Lab 4
Custom Instruction

1. Double click on the **cpu** module within SOPC Builder to open the cpu dialog box. Select the **Custom Instructions** tab.

2. In the right hand window pane select the **USR2** Opcode. In the left hand window pane click **Import**. This will bring up a new dialog box.

3. Click **Add**. Select the file **crc.vhd** and click **Open**. Enter **crc** for the **Top module**.

4. Click Read port-list from files and check that the port list looks as shown below. Click **Add to system**. Change the **Cycle Count** to **2** and click **Finish**. Click **Next**.

Port Name	Width	Direction	Type
clk	**1**	**input**	clk
reset	**1**	**input**	reset
start	**1**	**input**	start
clk_en	**1**	**input**	clk_en
dataa	**32**	**input**	dataa
datab	**32**	**input**	datab
result	**32**	**output**	result

5. Change the Boot ID to your name space **Custom** eg **My Name Custom** and click **Generate**.

The Workshop co-ordinator will now continue with the presentation whilst this new Nios system is generating. At a convenient time you will be asked to compile the design in Quartus. To compile the design select ***Start Compilation*** *from the* ***Processing*** *menu.*

END OF LAB 4

Lab 5
Run Custom Design

1. We will now download the Nios design created in the previous lab to the Nios development board. Within Quartus select the **Programmer** from the **Tools** menu. Tick the **Program/Configure** checkbox and then click the Start Programming icon  .

2. Open a Nios SDK Shell via the Windows Start Menu (Start, Programs, Altera, Nios Development Kit 3.10, Nios SDK Shell). Change directory with the command **cd:/nios_labs/cpu_sdk/my_src**.

3. Enter terminal mode with the command **nr –t**. Now press the **Escape** key on the PC to reset GERMS within the Nios core on the development board. Nios should respond by sending the Boot ID (your name Custom) to the SDK Shell. Press **Ctrl-C** to exit terminal mode.

4. Compile PWM example code with the command **nb pwm.c**.

5. Download the program with the command **nr pwm.srec**. Each press of the keys **1** to **4** on the PC keyboard should set a different brightness setting on LED 7 of the Nios development board. When finished press the **CPU Reset** button on the board and press **Ctrl-C** on the PC.

6. Compile CRC example code with the command **nb crc.c**.

7. Download the program with the command **nr crc.srec**. This program calculates the CRC of the internal boot ROM multiple times and lights LEDs to indicate progress. Make a note of the time taken for this calculation and the CRC result reported. When finished press the **CPU Reset** button on the board and press **Ctrl-C** on the PC.

8. Compile CRC custom instruction example code with the command **nb crcci.c**.

9. Download the program with the command **nr crcci.srec**. This program is similar to the previous one but makes use of the custom instruction that was created during the previous lab. Note that increase in performance. Compare the CRC result with the previous program and the time reported. When finished press the **CPU Reset** button on the board and press **Ctrl-C** on the PC.

IF YOU HAVE TIME…..

Further accelerate the CRC application by adding a DMA Engine and CRC Peripheral. In this scenario Nios will set up a DMA transfer to the dedicated CRC peripheral, wait for the transfer to complete and then read back the result. Because we are moving the loop control function from software to hardware a dramatic speed up in computation is expected.

10. Add a new interface to user logic as an **Avalon Register Slave** and import the file **C:\nios_labs\crc_peripheral.vhd**. For the instantiation select **simulate user logic** and select **0** for all timing options. Call the peripheral **my_crc**.

11. Add a DMA from the other section. Ensure that the width of the DMA register is **13** and that all transactions are allowed within the **Advanced** tab. Call this peripheral avalon_dma.

12. Connect the slaves and masters within the patch panel such that the DMA can read from the **boot_rom** component and write to the **my_crc** component.

13. Change the System Boot ID to your name space **Advanced** eg **My Name Advanced** and re-generate the system.

14. Once generation is completed, compile the design in Quartus, reprogram the board and run the **crcdma.c** program. Check that the CRC result reported is the same as before and note the reduction in time for the calculation.

END OF LAB 5