

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

- 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.
- 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.

Altera SOPC Builder - Nios					
System Contents System Generation					
Altera SOPC Builder	Target Device Family: Stratix 💌 System	n Clock Frequency: 50 MHz			
Nios Processor - Altera	Use Module Name	Description	Base	End	IRQ
Bridges					
Altera PCI32 Nios Ta					
Avalon To AHB Brid					
Avalon Tri-State Brid					
Altera Time Limited F					
Communication SDI (2) Mire Seriel)					
● LIART (RS-232 seris					
E-EP1C20 Nios Developmer					
AMD 29LV065D Fla:					
Active Serial Memor					
Installed components					
Add 🧐 Check	Mov	e Up 🔷 Move Down			
Problem, checking for, web-updates, (Ur	iable, to, download, component, catalog;, try, again, lat	er.)			
	Exit < Prev Next	> <u>G</u> enerate			



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.

rocessor Architecture		
C Nios- <u>1</u> 6	Nios- <u>3</u> 2	
16-bit ALU, registers, and o 16-bit addressing (maximur Programmers Reference M	lata bus 32-bit ALU, registers, n). 32-bit addressing (ma anual Programmers Referer	and data bus ximum). nce Manual
Preset Configurations: Standard for Enable advanced configuration	eatures / Average LE usage <mark>▼</mark>	

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

🚸 Altera Nios 3.0 - nios_0	×
Architecture Hardware Software Debug Custom Instructions	
On-Chip Instrumentation (OCI) Module from First Silicon Solutions	
Enable Nios OCI debug module	
Advanced Debug Features	
These features require additional software and/or hardware from First Silicon Solutions	
Hardware breakpoints (required when using trace features)	
Real-Time Trace	
On-chip trace memory	
☐ Off-chip trace capture (ISA-Nios/T)	
	_
Cancel < Prev Next > Finish	

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



 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

📙 On-chip Memory - boot	_rom	×
Attributes Contents		
Memory Type		
C RAM (writeable)	ROM (read-	only)
🗖 Dual-Po	ort Access	
Block Type:	Automatic 💌	
Size		
Data Width:	32 💌 bits	
Total Memory Size:	2	Kbytes 💌
Automatically choosing M48	< blocks	
<u>Cancel</u> < Prev	Next >	<u>F</u> inish

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**.

🛄 Avalon UART - uart_0	×
Configuration Simulation	
Baud Rate	-
Baud Rate (bps): 115200	
Input Clock Frequency (MHz): 50	
Baud error: <0.01%	
Baud rate can be changed by software (divisor register is writeable) parity data bits stop bits None 8	
Flow Control	
Streaming Data (DMA) control	
Include end-of-packet register	
<u>Cancel < Prev</u> <u>N</u> ext > <u>F</u> inish	



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**.

- vviatn	8	bits	
PIO v	vidth must be b	etween 1 and 321	bits
Direction			
C Bidirections	al (tri-state) port	s	
C Input ports	only		
C Both input :	and output ports		
• Output port	sonly		

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**.

📮 Avalon PIO - pio_0	×
Basic Settings Input Options	
Width	
4 bits	
PIO width must be between 1 and 32 bits	
Direction	
C Bidirectional (tri-state) ports	
Innut norts only	
-	
C Both input and output ports	
C Output ports only	
Cancel < Prev Next >	Finish



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.

Cn-chip Memory - ram		
Attributes Contents		
Memory Type		
RAM (writeable)	C ROM (read	l-only)
🔲 Dual-Po	ort Access	
Block Type:	Automatic 💌	
Size		
Data Width:	32 <u> </u> bits	
Total Memory Size:	8	Kbytes 💌
Automatically choosing M4I	(blocks	
<u>Cancel</u> < Prev	<u>N</u> ext ≻	

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**.

🖳 Avalon Timer - timer_0 🛛 💆
Timeout Period
Initial Period: 1 msec 💌
Input Clock Frequency: 50 MHz
Hardware Options
Preset Configurations: Full-featured (v1.0-compatible)
Registers
l <u>W</u> riteable period
I Readable snapshot
☑ Start/Stop control bits
Output Signals
☐ <u>T</u> imeout pulse (1 clock wide)
System reset on timeout (Watchdog)
<u>Cancel</u> < Prev <u>N</u> ext > <u>Finish</u>



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
V	⊞ ср и	Nios Processor - Altera Corporation	0x00002800	0x000028FF	\square
V	boot_rom	On-Chip Memory (RAM or ROM)	0x00002000	0x000027FF	\Box
V	🕀 uart1	UART (RS-232 serial port)	0x00002900	0x0000291F	16
V	⊡ led_pio	PIO (Parallel I/O)	0x00002940	0x0000294F	\square
V	🗄 button_pio	PIO (Parallel I/O)	0x00002950	0×0000295F	\Box
	🕀 ram	On-Chip Memory (RAM or ROM)	0x00000000	0x00001FFF	11
V	⊞ timer1	Interval timer	0x00002920	0×0000293F	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.

Function	Mo	odule	Offset	Address
Reset Location	boot_rom		0×00000000	0x00002000
Vector Table (256 bytes)	ram		0x00001F00	0x00001F00
Program Memory	ram			
Data Memory	ram			
Primary Serial Port (printf, GERMS)	uart1			0x00002900
Auxiliary Serial Port	uart1			0×00002900
System Boot ID: My Name Software Components			(25 chars m	ax)
Use Name	Jse Name Description			
Altera Plugs TCP/P Networking Library			lent networ	

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

Altera SOPC Builder - nios				
<u>File System Module View Help</u>				
System Contents Nios More "cpu" Settings	System Generation			
 Options ✓ SDK. Generate header files, library files, and ✓ HDL. Generate bus and system logic in VHD 	d memory contents for CPU(s) and peripherals in your system.			
Simulation. Create ModelSim(tm) project	files Run ModelSim			

- 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.



Lab 2 Software Flow



- 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 ^{Physel}.
- 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.

C s	imple.c -	Source W	indo w										
<u>F</u> ile	<u> B</u> un ⊻	ew <u>C</u> ontrol	Preferences	<u>H</u> elp									
*	(•) (° (? *()	1) 🕅	* /	4 🗸	69 🐔	• •)	0x10	7		₫ =	\$ _
	1 #i	nclude '	'excalibu	r.h" .	// for	debug	suppo	rt, mem	ory addr	esses a	nd peripheral	structu	re 🔺
L	2	aciaa W					and fu					a	
L	5 #U 4 #d	efine N	BUINCE 3	6 //	// V Time i	n Mill	eau fri isecon	ds to w	un riu w ait for	switch	dehounce	'U	
L	5	crime bi	Doonor o	• //	THE I		I SECON.	us co	111 101	JWICCH .	acbounce		
	6 in	t main(void)										
•	7 {		Al accontant and a set		10000000 00		6-31 T-41740740742						
a .	8	i	nt button	s; //	Use t	o hold	butto	n press	ed value				
L	40	1	nt Tea =	uxuı;	// US	ε το w	rite t	0 160					
I	11	ni	n nin *hu	ttonni	n = na	hutto	n nio:	77 ci	reate va	riable (huttonnio of t	une nn i	ni
-	12	n	pio *le	dpio =	na le	d pio;	,					3F- "F_	· -
	13	100			1000 - 1 000								
- I	14	р	'intf("Si	mple\n'	');	// pri	nt a m	essage i	to show	that pro	ogram is runni	ng	
L	15												
- E	10	10	eabio->ub	_proda	ta = 1	ea; /	/ write	e initia	al value	to pio			
2.1	18	iii	nile (1)										
	19	{											
-	20		bu	ttons	= butt	onpio-	>np_pi	odata;	// rea	d butto	ns via pio		
L	21					tonician a					1010A 201302A •003062		
	22		if	(butt	ons !=	NONE_	PRESSE	D) //:	if butto	n press	ed		
	23		{	2-3									
D	24				14 (16		X80) . - 0v04	// 1+ pa	attern 1 acot pat	5 00000	uun on board (leas in	r
F .	25				else	Ten	- 0701	, // 0	eser par	tern			
2	27					led	= led -	<< 1: J	// shift	riaht	on board (led@) is far	1
L	28					100	2007000			3	· · · · · · · · · · · · · · · · · · ·	s exes 0,0708	
-	29				ledpio	->np_p	iodata	= led;	// wri	te new v	value to pio		
	n n				25						28		



- 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 10. Notice that execution stops at this line and the PC and other register values have changed to blue to denote a change in value.

-					
ឲ្យ	0x8	10	0x1920	pc	0x32
g1	0 x 0	11	0x1	ct10	0xfed0
g2	Øxc3c	12	0x1920	ctl1	0 x 0
g3	0x 0	13	0x1930	ct12	0 x 0
g4	0x 0	14	0x1920		
g5	0x846	15	0x3d		
gó	0x3	16	0x 0		
g7	Øx3e2	17	0x 0		
00	0x 0	i0	0x104		
01	0x 0	i1	0x1000		
02	0x 0	i2	0x 0		
03	0x 0	i 3	0x 0		
04	0x 0	14	0x 0		
05	0x 0	i5	Øx829		
sp	0xe5c	fp	Øxeb8		
07	Øx2e	i7	0x1c6		

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**.

C Local Variables		_ 🗆 🗵
Variable		
Name	Value	_ _
buttons	Øxf	
led	1	
⊞buttonpio	0x1930	
⊞ledpio	0x1920	
105 10 10 5 10 5 10		-
•		

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.

C Memory	,				_ 🗆 ×
Addresses					
Address 🗔	(1920				
	6	4	8	C	ASCII
0x1920	Øxffffffff	Øxfffffff	Øxffffffff	Øxffffffff	
0x1930	0x0000000f	0x00000000	0x00000000	0x00000000	
0x1940	Øxffffffff	Øxfffffff	Øxfffffff	Øxfffffff	
0x1950	Øxfffffff	Øxfffffff	Øxfffffff	Øxfffffff	
0x1960	Øxfffffff	Øxfffffff	Øxfffffff	Øxfffffff	
0x1970	Øxffffffff	Øxffffffff	Øxffffffff	Øxffffffff	
0x1980	Øxffffffff	Øxffffffff	Øxffffffff	Øxffffffff	
0x1990	Øxffffffff	Øxffffffff	Øxffffffff	Øxffffffff	
0x19a0	Øxffffffff	Øxffffffff	Øxfffffff	Øxffffffff	
•					•

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. J 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.



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	Туре
cik	1	input	1111	clk
writedata	32	input	1111	writedata
byteenable_n	4	input	1111	byteenable_n
cs	1	input	1111	chipselect
write_n	1	input	1111	write_n
addr	1	input	1111	address
reset_n	1	input	1111	reset_n
readdata	32	output	1111	readdata
pwm_out	1	output	1111	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.



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.
- 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	Vvidth	Direction	Туре	
cik	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.



Lab 5 Run Custom Design



- 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 ^{Phi}.
- 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**.
- 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.

- 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.

