

EG3205 Part II. Multiprocessor and Multicore Systems

Laboratory Exercise 2

Contents

Introduction to Lab 2.....	1
Description of Lab 2	1
Evaluating hardware and software memory consumption.....	3
Creating a TTC scheduler and a matching processor platform.....	5
Demonstrate Understanding.....	8

Introduction to Lab 2

In Lab 1, you familiarised yourself with FPGAs, the Intel ECAD software Quartus II ver. 13.1 and the Nios II design environment. You implemented a Nios II processor using Quartus II, which ran a Superloop scheduler.

In this exercise, your task is to implement an upgraded design using the Nios II “soft” processor IP core and a Time-Triggered Co-operative (TTC) scheduler.

Note: Complete the Lab 1 work if you did not manage to finish before moving on to this exercise.

Description of Lab 2

You are provided with the C-code of a TTC scheduler. You are expected to achieve the following tasks:

Task 1:

Compile and run your Lab 1 design and evaluate the hardware and software memory consumption.

Task 2:

Capture the hardware configuration of the Nios II based system that will run the TTC scheduler using the Qsys integration tool and combine it with the provided code `basic_nios_II_flash_led_ttc` via the Eclipse based Software Build Tool (SBT).

Task 3:

Demonstrate understanding of the previous tasks by briefly writing in a word document:

- An outline of the Qsys configuration of a Nios II based system running the provided TTC scheduler.
- An explanation of what the TTC scheduler C-code does and how it works.
- A comparison of the TTC and Superloop schedulers.

Name the document Lab 2 and upload it to Blackboard.

Demonstrate correct operation of the Nios II based system on the DE0 FPGA board.

Files provided:

Please download the file `basic_nios_II_flash_led_ttc.zip` from blackboard

Acknowledgements:

The first version of this exercise was created by Dr Keith Athaide (March 2012).

The second version of this exercise was developed by Dr. M. Fayyaz(June 2015).

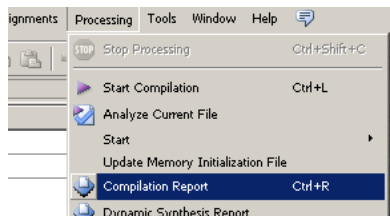
This version of the exercise was developed by Sam Kennedy (October 2018).

Evaluating hardware and software memory consumption

1. Evaluating hardware consumption

To verify hardware consumption, in Quartus, open the “Compilation Report” from the “Processing” menu.

Quartus IDE



Total logic elements	1,329 / 15,408 (9 %)
Total combinational functions	1,248 / 15,408 (8 %)
Dedicated logic registers	740 / 15,408 (5 %)
Total registers	740
Total pins	10 / 347 (3 %)
Total virtual pins	0
Total memory bits	75,776 / 516,096 (15 %)
Embedded Multiplier 9-bit elements	0 / 112 (0 %)
Total PLLs	0 / 4 (0 %)

2. Evaluating software memory consumption

In the Eclipse IDE, open the object dump file generated during the compilation process. The size of code and data can be seen under the heading “Sections”, it is given in hexadecimal.

Eclipse IDE

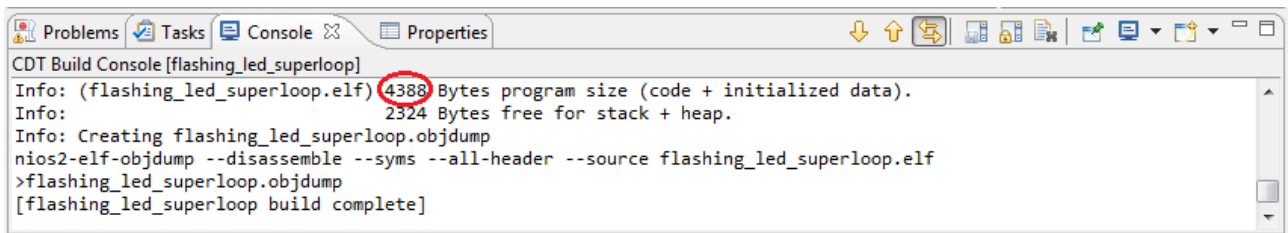
```

Flashing_LED_SuperLoop.elf:      file format elf32-littlenios2
Flashing_LED_SuperLoop.elf
architecture: nios2, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000020

Program Header:
  LOAD off 0x00001000 vaddr 0x00000000 paddr 0x00000000 align 2**12
    filesz 0x00000020 memsz 0x00000020 flags r-x
  LOAD off 0x00001020 vaddr 0x00000020 paddr 0x00000020 align 2**12
    filesz 0x0000090c memsz 0x0000090c flags r-x
  LOAD off 0x0000192c vaddr 0x0000092c paddr 0x00000ef4 align 2**12
    filesz 0x000005c8 memsz 0x000005c8 flags rw-
  LOAD off 0x000024bc vaddr 0x000014bc paddr 0x000014bc align 2**12
    filesz 0x00000000 memsz 0x000001a0 flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
 0 .entry          00000020  00000000  00000000  00001000  2**5
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 1 .text           000008fc  00000020  00000020  00001020  2**2
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 2 .rodata         00000010  0000091c  0000091c  0000191c  2**2
   CONTENTS, ALLOC, LOAD, READONLY, DATA
 3 .rwdata         000005c8  0000092c  00000ef4  0000192c  2**2
   CONTENTS, ALLOC, LOAD, DATA, SMALL_DATA
 4 .bss            000001a0  000014bc  000014bc  000024bc  2**2
   ALLOC, SMALL_DATA
 5 .comment        00000023  00000000  00000000  00001ef4  2**0
   CONTENTS, READONLY
 6 .debug_aranges  000002f0  00000000  00000000  00001f18  2**3
   CONTENTS, READONLY, DEBUGGING
 7 .debug_info     00003f4c  00000000  00000000  00002208  2**0
  
```

.entry contains the reset code, .text the remaining code, .rodata the read-only data, .rwdata the read-write data and .bss the zero-initialised data. The sum of the sizes of these five regions is normally output during compilation.



```
CDT Build Console [flashing_led_superloop]
Info: (flashing_led_superloop.elf) 4388 Bytes program size (code + initialized data).
Info: 2324 Bytes free for stack + heap.
Info: Creating flashing_led_superloop.objdump
nios2-elf-objdump --disassemble --syms --all-header --source flashing_led_superloop.elf
>flashing_led_superloop.objdump
[flashing_led_superloop build complete]
```

The number of bytes free for the stack and the heap are calculated by subtracting both the sum of the five regions and the `.rwdata` region size from the total memory size (the `.rwdata` region is normally copied from a read-only storage memory to a read-write memory – in this case, it just so happens that the storage memory is read-write).

The *stack* is the memory set aside for execution of tasks, it stores variables in an ordered way, executing last in, first out. The space is managed efficiently, hence the name stack.

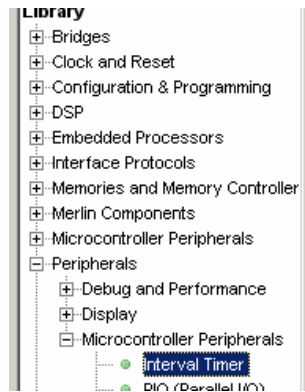
The *heap* is the memory that is used for dynamic memory allocation. It is much freer and is not closely managed, hence the name heap.

Creating a TTC scheduler and a matching processor platform

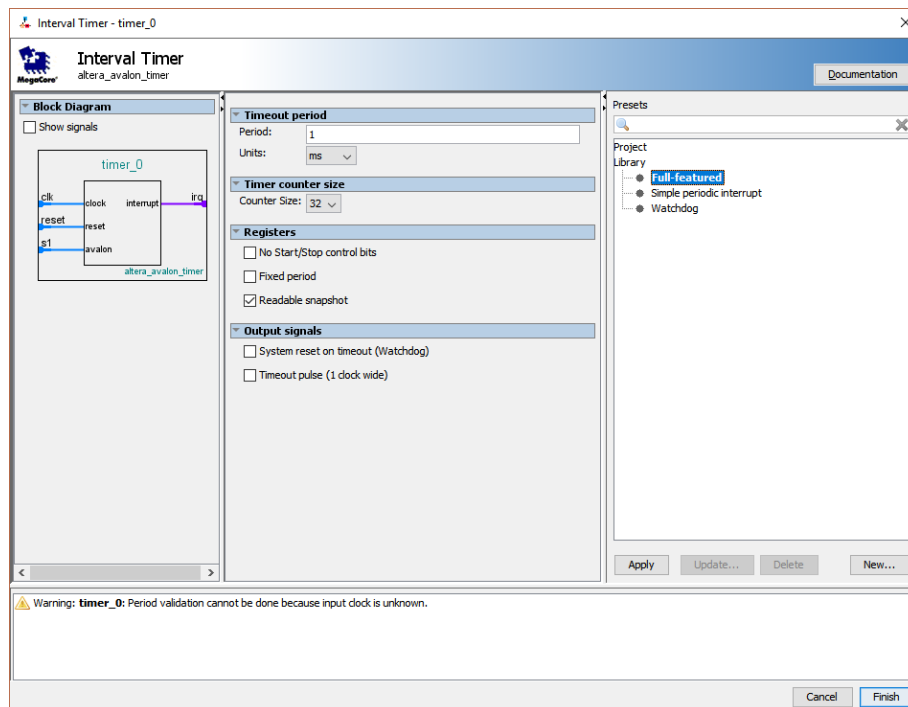
We'll now adapt the processor design created in Lab 1 to support a timer-based interrupt and add a TTC scheduler to the system.

Please proceed as follows:

1. In Qsys, add an Interval Timer peripheral.

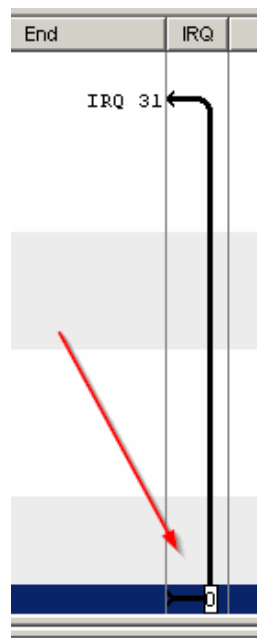


2. Choose the pre-set “Full-featured” and click “Finish”.



3. Connect the clock, reset and slave interface of the new timer peripheral in the same way as those of the PIO peripheral have been connected.
4. Set the base address of the timer peripheral to 0xa020.
5. Click in the IRQ column so as to connect the timer peripheral to IRQ 0 of the Nios core.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	T	
<input checked="" type="checkbox"/>		<div>clk_0</div> <div>clk_in</div> <div>clk_in_reset</div> <div>clk</div> <div>clk_reset</div>	Clock Source Clock Input Reset Input Clock Output Reset Output	clk reset <i>Double-click to export</i> <i>Double-click to export</i>	exported clk_0			<div><div></div><div></div></div>		
<input checked="" type="checkbox"/>		<div>nios2_qsys_0</div> <div>clk</div> <div>reset_n</div> <div>data_master</div> <div>instruction_master</div> <div>jtag_debug_module_r...</div> <div>jtag_debug_module</div> <div>custom_instruction_m...</div>	Nios II Processor Clock Input Reset Input Avalon Memory Mapped Master Avalon Memory Mapped Master Reset Output Avalon Memory Mapped Slave Custom Instruction Master	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk] [clk] [clk] [clk]		IRQ 0	IRQ 31		
<input checked="" type="checkbox"/>		<div>onchip_memory2_0</div> <div>clk1</div> <div>s1</div> <div>reset1</div>	On-Chip Memory (RAM or ROM) Clock Input Avalon Memory Mapped Slave Reset Input	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk1] [clk1]	0xa800	0xffff			
<input checked="" type="checkbox"/>		<div>pio_0</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk]	0x0000	0xffff			
<input checked="" type="checkbox"/>		<div>timer_0</div> <div>clk</div> <div>reset</div> <div>s1</div>	Interval Timer Clock Input Reset Input Avalon Memory Mapped Slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0xa000	0xa00f			
					led					

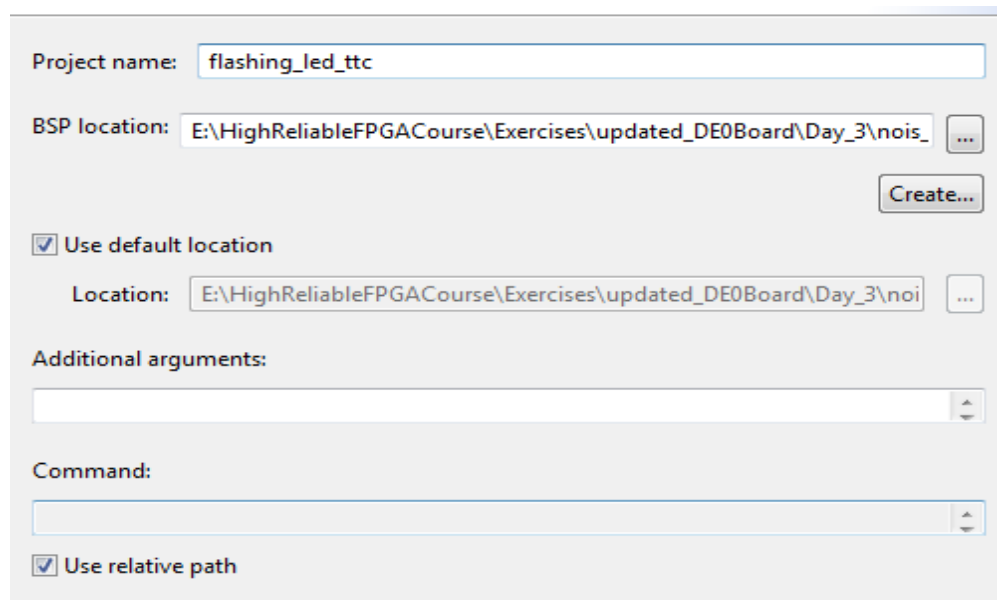


Programmer

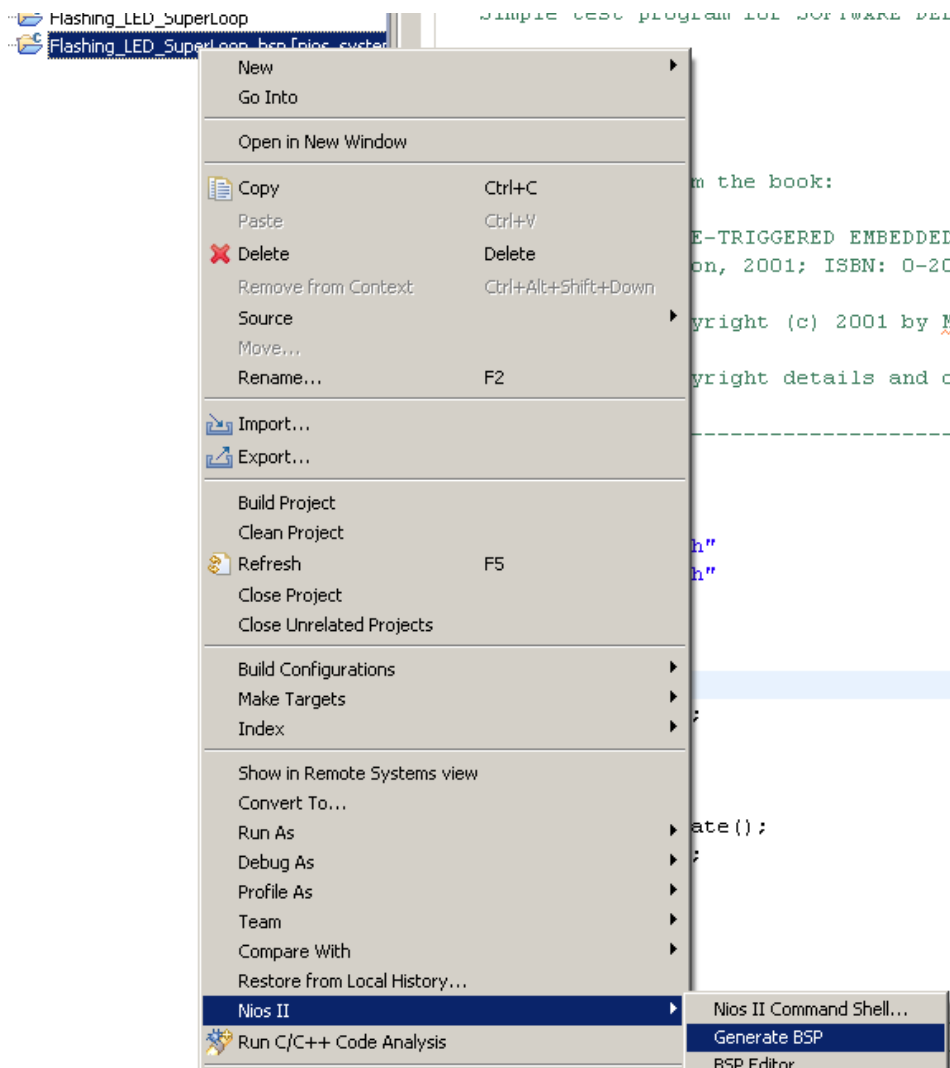
6. Generate the design from Qsys, compile under Quartus, download to the development board using the Programmer.

Eclipse IDE

7. Create a new Nios II application project without BSP. Go to file menu, under new, click Nios II application. A dialog will appear, give a suitable project name e.g. flashing_led_ttc. Browse the BSP location created for super loop. Click finish to complete project.



8. Right-click on the 'flashing_led_ttc' project. Select "Import" from the "File" menu. Then expand "General" and select "File System".
9. A "File System" dialog will open. Click browse to locate the source folder that includes the 'TTC Scheduler' program files. Click on 'Select All' to include all files into the project. Make sure that 'Create top-level folder' is checked. Click finish to complete this process
10. Regenerate the BSP in the Nios II Eclipse IDE: right-click the BSP project and choose "Generate BSP" under "Nios II".



11. Under project, click Clean and Build the workspace.

Note 1: `main.c` uses `alt_main` instead of `main`. If `alt_main` is not defined, the BSP's HAL performs certain initialisations – like turning on interrupts before calling “main” – which were not needed for this example.

Note 2: The Nios II core doesn't have a low power mode – the scheduler dispatch is called continuously.

12. Select the project and choose “Nios II Hardware” under the “Run As” submenu on the “Run” menu.

13. Your application should now run successfully.

Demonstrate Understanding

Using Eclipse, navigate through the code for the TTC Scheduler. Examine the various source and header files. Analyse the functions implemented and the code they contain. How does the Main.c

script call the various functions? What jobs do these functions do? What is causing the LED to flash? Why a timer is needed in the NIOS II system?

The code is primarily taken from the Patterns for Time-Triggered Embedded Systems book by Michael Pont which is available on blackboard.

Prof. Tanya Vladimirova
06/11/2018