# ECE3073 Computer Systems

Laboratory Session 3

Polling and polling latency in C code

## 1. Objectives

The first two laboratory exercises were based on machine code and assembler. Now you are moving on to using C code. In this exercise you will investigate simple C code programming of the Nios processor to write polling loops. This will require you to specify a more complex Nios processor system with multiple input and output ports. You will then investigate the latency in responding to a service request both theoretically using the instruction timings you found last week and practically using an oscilloscope. For many real-time applications the speed with which a microprocessor system responds to outside events is very important. This lab will give you an idea of the factors limiting response time and how to measure it. In this exercise you will:

- Be introduced to C code programming of the Nios processor
- Develop a more complex Nios processor system
- Write a polling loop in C code
- Use an oscilloscope to measure the latency of the polling loop
- Working from the disassembly of your C code, work out the theoretical latency of your polling loop.
- Compare the theoretical and measured values of polling latency.

### 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.

Read through the whole of this document so that you understand all of the things you will be required to do.

In this lab exercise you will add an always block to your top level Verilog file that generates a periodic waveform and connect this to the PIO_in1 input. This can be achieved by using a simple 20-bit counter that increments by 1 on each positive edge of CLOCK_50. The rising edge of the most significant bit (MSB) in the counter can be used as a request for service.

What is the frequency of this MSB rising edge? Your answer ……..

And what period does this correspond to (in milliseconds)……

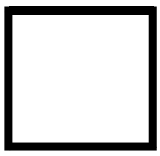
This is what your C code should do:

1) initialize registers with pointers to I/O ports, and any variables required
2) perform an infinite loop that does the following:
    a) if there is a rising edge on the output of the most significant bit of the counter:
        - write a '1' to GPIO_1[10], and then
        - increment an 8-bit binary number displayed on the 8 red LEDs (LEDR[7:0]) and

- write a ‘0’ just before returning to the polling loop.

b) when KEY[1] is pressed then toggle the state of green LED (LEDG[1]). Note that it is not guaranteed that reading an output port will return the last value you wrote there (depends on the port architecture), so you will need to use an integer variable to store the current state to display on the LED.

As part of your preliminary work write C code to implement the polling loop (further details are provided later in this document). Of course you do not know the port addresses yet, so you can leave them blank in you preliminary code.

Demonstrator to initial for satisfactory completion of preliminary work 

## 3. Create a new Nios system

For this laboratory exercise you will need a Nios system with additional input/output capabilities.

a) Use Qsys to create and connect up a microprocessor system with:

- a Nios II/e processor,
- 8192 bytes of RAM
- one 8-bit PIO output port (to drive LEDR[7:0])
- two 1-bit PIO output ports (one to flag the start of the counter service routine and one to drive green LED LEDG[1])
- two 1-bit PIO input ports (one to read KEY[1] and the other to read the most significant bit (MSB) of the 20-bit counter)

At this point you should make a note of the base addresses here (make sure to "Assign Base Addresses" first):

8-bit red LED PIO base address ……

1-bit PIO flag base address ……..……..

1-bit PIO green LED base address ……..

1-bit PIO key base address ……..

1-bit PIO counter MSB base address ……..

b) In your top-level Verilog file include:

- an instance of your Nios microcontroller system
- the Nios reset input connected to KEY[0]
- the Nios clock connected to the system 50-MHz clock
- a 20-bit counter clocked from the 50-MHz system clock with the MSB signal fed into the Nios 1-bit 'MSB' PIO input port and also connected to pin GPIO_1[2] of the DE2 board 40-pin connectors.
- the other 1-bit PIO input port connected to KEY[1]
- the 1-bit 'flag' PIO output connected to GPIO_1[10]
- the other 1-bit PIO output connected to the green LED LEDG[1]
- the 8-bit PIO output connected to red LEDs LEDR[7:0]
- directly connect the KEY[0] to LEDG[0] so you have visual confirmation of the reset signal

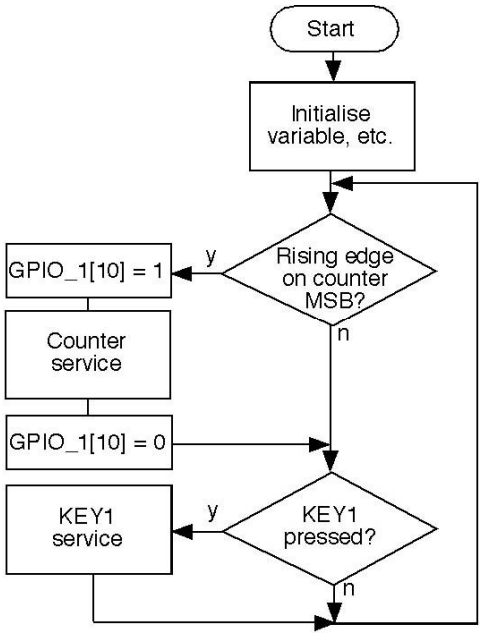
NIOS processor generated correctly

## 4. Write C code for your Nios system

Your Nios processor will be executing a polling loop as follows:

- start by initializing variables and registers which contain the base address of all of the PIOs.
- then enter an infinite loop that does the following:
  - check for the rising edge on the MSB of the counter and if detected then
    - make GPIO_1[10] go high
    - increment the binary number displayed of the 8 red LEDs
    - make GPIO_1[10] go low
    - return to the polling loop
  - check if KEY1 is pressed and if it is then toggle the state of green LED LEDG[1]
  - return to the start of the polling loop

Here is a flowchart to show what is required:




Here is skeleton C code to get you started:

```
/* Declare volatile pointers to I/O registers. This will ensure that the
resulting code will bypass the cache*/

volatile int * InPort_Key1 = (int *) 0x00003010;     // check port address
volatile int * OutPort_GREENLED = (int *) 0x00003020; // check port address

int main(void)
{
int SW_value;       // temporary place to store the Key1 value

while(1)
       {
       SW_value=*(InPort_Key1);          // read Key1
       *(OutPort_GREENLED) = SW_value; // transfer the value to green LED 1
        }
}
```

You can use the Microsoft C development environment to write your C program but it must be compiled in the Altera Monitor environment so don't try to compile it in the Microsoft C environment.

## 5. Run your C code on your Nios system

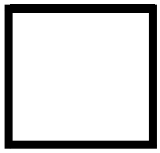
Similar to Lab 2 use the Altera Monitor program to download your Nios processor design into the DE2 board and then compile and load your C-code. Then:

a) Using the instruction timings you measured in the previous exercise and the disassembly of your C code provided by the Monitor calculate the minimum and maximum latency that you expect for the counter service (Note: Nios "**stwio"** instructions take 9 pulses/instruction to execute). Use the flowchart on the previous page to help you determine what the minimum and maximum latency paths would be.

Expected Minimum Latency ……

Expected Maximum Latency……

Expected increase in latency when KEY1 pressed……..

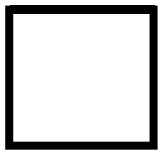
Demonstrator initials for satisfactory latency calculation 

b) Using your oscilloscope measure the latency (Hint: use of single shot may help). Note that at low digital oscilloscope timebase speeds very narrow pulses will disappear completely. Therefore, if you cannot see a short pulse on the oscilloscope you should try increasing the timebase speed (zoom in).

Measured Minimum Latency ……

Measured Maximum Latency……

Measured increase in maximum latency when KEY1 pressed……..

Demonstrator initials for satisfactory latency measurement 

c) How well do the calculated and experimental latency values compare?

……..

What do you think could account for any discrepancy between calculated and experimental values?

……..

Demonstrator initials for satisfactory result from part c

When you have completed 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.

## Appendix A

Layout of DE2 board JP2 socket.