

## ECE3073 Computer Systems

### Laboratory Session 4

### Interrupts and Interrupt Latency

Semester 1. 2020

#### 1. Objectives

In the previous laboratory exercise you looked at polling as a way for a computer to respond to outside events. An alternative to polling is the use of interrupts and they are the subject of this exercise. Functionally the program developed in this lab will be similar to lab 3 (8 red LEDs will count in binary fashion on the rising edge of the MSB of a 20-bit counter counting the 50MHz clock and push button KEY1 will toggle green LED LEDG1 (Note: this function for KEY1 is slightly different to the previous lab as it includes de-bouncing code). However, the counter will be serviced by an interrupt routine not by polling. This lab will give you an idea of the programming structures and hardware required to support interrupts together with interrupt latency. In this exercise you will:

- Write a C/assembly program to support interrupt servicing of external events
- Use an oscilloscope to measure interrupt latency
- Work from the assembler code to calculate the theoretical interrupt latency
- Compare the theoretical and measured values of interrupt 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 (6 Marks)

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.

For each of the following initializations indicate the processor control register or PIO register map register that is involved and the bit pattern that must be written to the register to perform the required initialization:

a) setup rising edge triggered interrupts for the counter PIO input

register: .....

initialization: .....

b) set interrupt enable mask bit for counter IRQ level n (where n will be the IRQ level/number the counter PIO interrupt is assigned to)

register: .....

initialization: .....

c) enable Nios II interrupts

register: .....

initialization: .....

Demonstrator initials for satisfactory completion of preliminary work



### 3. Design a new Nios system

For this laboratory exercise you will need a Nios system which is very similar but not identical to the one you designed for lab3. You can modify your previous design or start from the beginning:

a) Using Qsys create 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 the corner RED LED (PIN\_H15) , this LED gets ON and OFF for every alternative rising edge of the MSB counter pulse. One to Green LED - LEDG (PIN\_E21)
- two 1-bit PIO input ports (one to read KEY[1] and the other to read the most significant bit (MSB) of the 1 Hz counter . [Read the parallel input/output \(PIO\) core section in the Nios embedded peripherals datasheet on Moodle. Use this information to setup rising edge triggered interrupts on the PIO counter MSB input.](#)

To attach the interrupt request line from the counter PIO to the Nios processor click on the bubble in the Qsys IRQ column and make a note of the allocated interrupt number.

After you compile your Nios system take note of the base address of all of the Nios peripherals (these will probably be the same as for the previous exercise) and the interrupt request (IRQ) level/number allocated to the counter PIO.

Counter PIO interrupt level .....

8-bit red LED PIO base address .....

1-bit PIO RED LED (PIN\_H15) ..... ( This refers to the corner most RED LED)

1-bit PIO green LED base address .....

1-bit PIO key base address .....

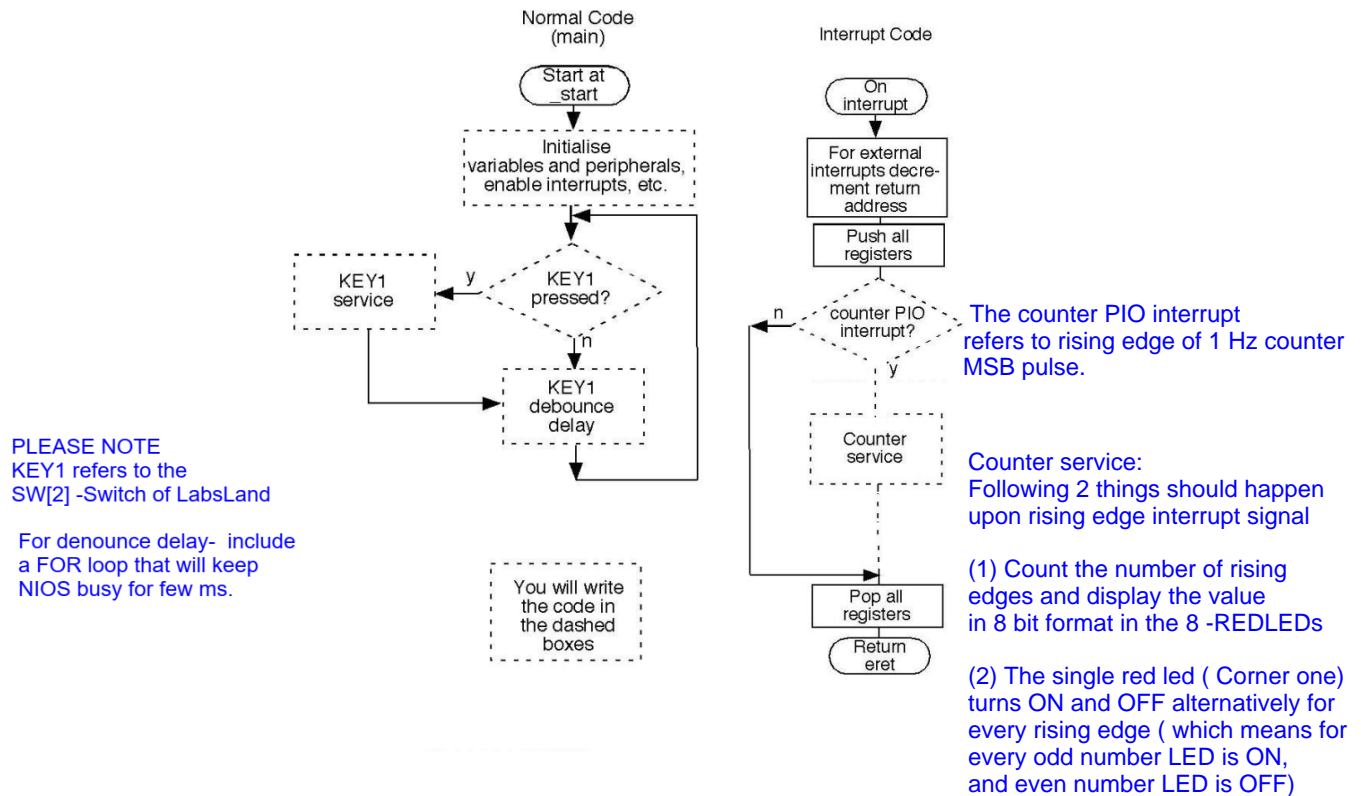
1-bit PIO counter MSB base address .....

(b) Lab 4 follows Lab3b - 1 Hz counter

IMPORTANT: FOLLOW THE LAB4- Supporting.doc word file to view the schematics and qsys settings

#### 4. C normal and interrupt code

You are provided with skeleton code (use files from Moodle) for a normal code main function and interrupt code. The following figure shows flowcharts for this code. Your C code should implement the flowchart elements in the dashed boxes. You need to refer to your lecture notes and the Nios embedded peripherals datasheet on Moodle in order to learn how to set up and service an interrupt.



Lab4\_main.c

```

#include "nios2_ctrl_reg_macros.h"

// function prototypes

int main(void);
void interrupt_handler(void);
void the_exception (void);

/* Declare volatile pointers to I/O registers. This will ensure that the
resulting code will bypass the cache*/
volatile int * example_pointer = (int *) 0x1000;

/*****
 * This program demonstrates use of interrupts in the DE2 Basic Computer.
 *****/
int main(void)
{
}

```

## Lab4\_exception.c

```

#include "nios2_ctrl_reg_macros.h"

/* function prototypes */
void main(void);
void interrupt_handler(void);
void the_exception (void);

/* global variables */
extern int* example_pointer;

// extern indicates to the linker that the variable has been declared
// elsewhere. This declaration does not allocate any memory.

/* The assembly language code below handles CPU reset processing */
void the_reset (void) __attribute__ ((section (".reset")));
void the_reset (void) /
*****
* Reset code. By giving the code a section attribute with the name ".reset"*
* we allow the linker program to locate this code at the proper reset      *

* vector address. This code just calls the main program.                  *
******/
{
asm (".set          noat");           // Magic, for the C compiler
asm (".set          nobreak");        // Magic, for the C compiler
asm ("movia         r2, main");        // Call the C language main program
asm ("jmp           r2");
}

/* The assembly language code below handles CPU exception processing. This
* code should not be modified; instead, the C language code in the function
* interrupt_handler() can be modified as needed for a given application.
*/
void the_exception (void) __attribute__ ((section (".exceptions")));
void the_exception (void)
/*****
* Exceptions code. By giving the code a section attribute with the name
* ".exceptions" we allow the linker program to locate this code at the
* proper exceptions vector address.
* This code calls the interrupt handler and later returns from the
* exception.
*****/
{
asm ( ".set          noat" );           // Magic, for the C compiler
asm ( ".set          nobreak" );        // Magic, for the C compiler
asm ( "subi          sp, sp, 128" );      // make space on the stack
asm ( "stw           et, 96(sp)" );      // save exception temporary
asm ( "rdctl         et, ctl4" ); // read control register ctl4 ipending
asm ( "beq           et, r0, SKIP_EA_DEC" ); // Interrupt is not external
asm ( "subi          ea, ea, 4" );        // if external must decrement
// ea by one instruction for
// external interrupts, so that
// the interrupted instruction
// will be run

asm ( "SKIP_EA_DEC:" );
asm ( "stw           r1, 4(sp)" );        // Save all registers
asm ( "stw           r2, 8(sp)" );
asm ( "stw           r3, 12(sp)" );
asm ( "stw           r4, 16(sp)" );
asm ( "stw           r5, 20(sp)" );

```

```

asm ( "stw    r6,    24(sp)" );
asm ( "stw    r7,    28(sp)" );
asm ( "stw    r8,    32(sp)" );
asm ( "stw    r9,    36(sp)" );
asm ( "stw    r10,   40(sp)" );
asm ( "stw    r11,   44(sp)" );
asm ( "stw    r12,   48(sp)" );
asm ( "stw    r13,   52(sp)" );
asm ( "stw    r14,   56(sp)" );
asm ( "stw    r15,   60(sp)" );
asm ( "stw    r16,   64(sp)" );
asm ( "stw    r17,   68(sp)" );
asm ( "stw    r18,   72(sp)" );
asm ( "stw    r19,   76(sp)" );
asm ( "stw    r20,   80(sp)" );
asm ( "stw    r21,   84(sp)" );
asm ( "stw    r22,   88(sp)" );
asm ( "stw    r23,   92(sp)" );
asm ( "stw    r25, 100(sp)" );           // r25 = bt (skip r24 = et, because
                                         // it is
                                         saved above)
asm ( "stw    r26, 104(sp)" );           // r26 = gp
// skip r27 because it is sp, and there is no point in saving this
asm ( "stw    r28, 112(sp)" );           // r28 = fp
asm ( "stw    r29, 116(sp)" );           // r29 = ea
asm ( "stw    r30, 120(sp)" );           // r30 = ba
asm ( "stw    r31, 124(sp)" );           // r31 = ra
asm ( "addi   fp,    sp, 128" );

asm ( "call   interrupt_handler" );      // Call the C language interrupt
                                         // handler

asm ( "ldw    r1,    4(sp)" );           // Restore all registers
asm ( "ldw    r2,    8(sp)" );
asm ( "ldw    r3,   12(sp)" );
asm ( "ldw    r4,   16(sp)" );
asm ( "ldw    r5,   20(sp)" );
asm ( "ldw    r6,   24(sp)" );
asm ( "ldw    r7,   28(sp)" );
asm ( "ldw    r8,   32(sp)" );
asm ( "ldw    r9,   36(sp)" );
asm ( "ldw    r10,  40(sp)" );
asm ( "ldw    r11,  44(sp)" );
asm ( "ldw    r12,  48(sp)" );
asm ( "ldw    r13,  52(sp)" );
asm ( "ldw    r14,  56(sp)" );
asm ( "ldw    r15,  60(sp)" );
asm ( "ldw    r16,  64(sp)" );
asm ( "ldw    r17,  68(sp)" );
asm ( "ldw    r18,  72(sp)" );
asm ( "ldw    r19,  76(sp)" );
asm ( "ldw    r20,  80(sp)" );
asm ( "ldw    r21,  84(sp)" );
asm ( "ldw    r22,  88(sp)" );
asm ( "ldw    r23,  92(sp)" );
asm ( "ldw    r24,  96(sp)" );
asm ( "ldw    r25, 100(sp)" );           // r25 = bt
asm ( "ldw    r26, 104(sp)" );           // r26 = gp
// skip r27 because it is sp, and we did not save this on the stack
asm ( "ldw    r28, 112(sp)" );           // r28 = fp
asm ( "ldw    r29, 116(sp)" );           // r29 = ea
asm ( "ldw    r30, 120(sp)" );           // r30 = ba
asm ( "ldw    r31, 124(sp)" );           // r31 = ra

asm ( "addi   sp,    sp, 128" );

asm ( "eret" );

```

```

}

/*****
 * Interrupt Service Routine
 * Services the counter interrupt.
 *
 * ipending - Control register 4 which has the pending external interrupts
 *****/
void interrupt_handler(void)
{

}

```

**Note:** When you compile your code you will need the header file “nios2\_ctrl\_reg\_macros.h” in the same folder as your c code files. You will find the header file on Moodle. This header file contains macro functions that you can use to write to and read from the Nios control registers.

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

Use the Altera Monitor program to load your Nios processor design into the DE2 board followed by compiling and down-loading your C-code.

Demonstrator initials to indicate that your C-code compiles without errors.



Then:

a) Using the instruction timings you measured in lab 1 and the disassembly of your C code provided by the Monitor calculate the interrupt latency that you expect for the counter service. For this exercise assume that latency is the time between the external event happening and the start of the associated service code. [This question refers to theoretical value calculation.](#)

Expected Latency .....

What is major source of latency in this system?

.....

Demonstrator initials for satisfactory latency calculation



b) Using your oscilloscope measure the latency. [To calculate this use oscilloscope in lab or attempt in LabsLand](#)

Measured Latency .....

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.