Homework 2 - Using Interrupts

Zackary McClamma*†

September 30, 2019

1 Introduction

In assignment consisted of configuring the NIOS II Processor in QuartusPrime and adding the 2 of the 7-segment displays and a button as peripherals. It also included creating software that would display the values 01-12 on the two 7-segment displays Figure **??** by using an interrupt to iterate the value on the display whenever the button is pressed. This assignment also requiered the use of SignalTap to measure the latency of the interrupt which the results of can be seen in Figure 3.

2 System Design

This system is very similar to the system designed in the first homework assignment, with two major differences. First in this assignemnt a second 7-Segment display was added, and second but most importantly, the button in this assignment uses an interrupt as opposed to polling in the previous assignment. The process is defined in 1 below. In the figure the blue arrows indicate VHDL code, the red arrows indicate C code, and the green arrows indicate that both VHDL and C are being transmitted.

3 Theory of Operation

The basic idea of this system is that the 7-segment display iterates through the values (01-12), with the value being incremented every time the button (KEY3) is pressed. When the button is pressed it sends an interrupt signal to the processor. The processor then saves its current state and immediately loads the interrupt service routine. The count value is then incremented and the button is reset back to its default state and the processor then reloads its state from before the interrupt signal was received.

*University of Dayton

†Dept. of Electrical and Computer Engineering, University of Dayton, 300 College Park, Dayton, OH 45469-0226, U.S.A. E-mail: mcclammaz1@udayton.edu

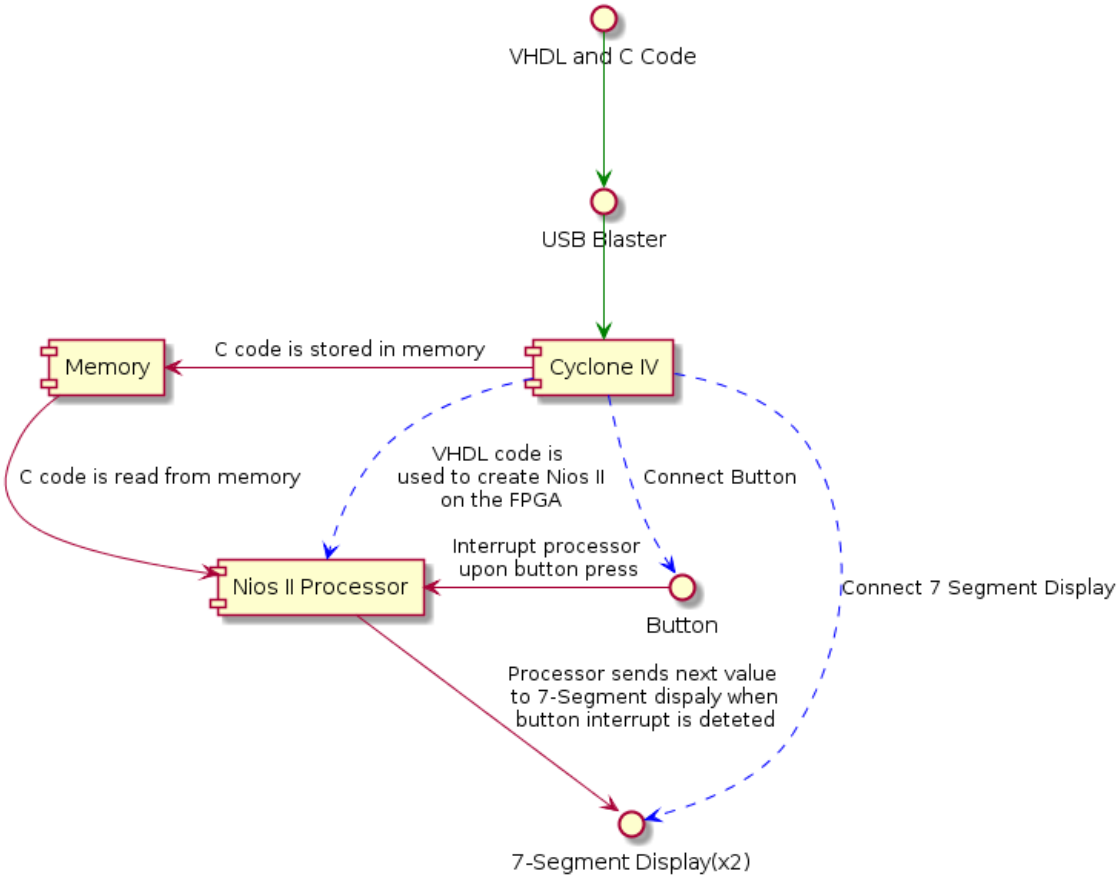
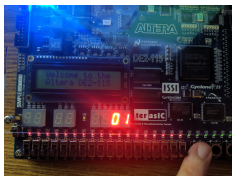


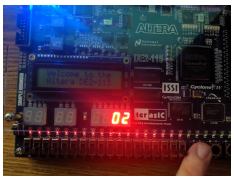
Figure (1) Embedded System Block Diagram

4 Results

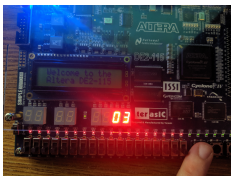
4.1 Photos of Working System



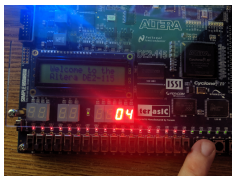
(a) Display 1



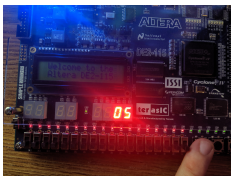
(b) Display 2



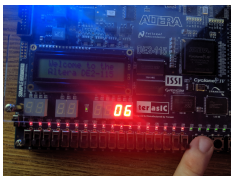
(c) Display 3



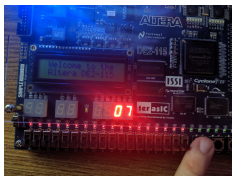
(d) Display 4



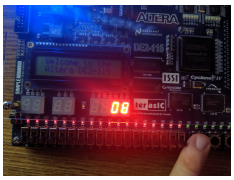
(e) Display 5



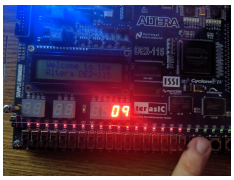
(f) Display 6



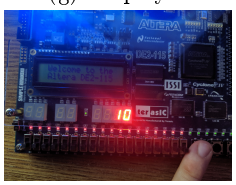
(g) Display 7



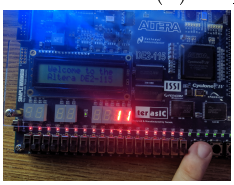
(h) Display 8



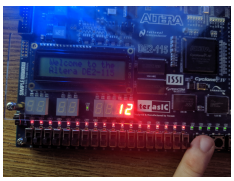
(i) Display 9



(j) Display 10



(k) Display 11



(l) Display 12

Figure (2) Running 7 Segment Display

4.2 Signal Tap Snapshot

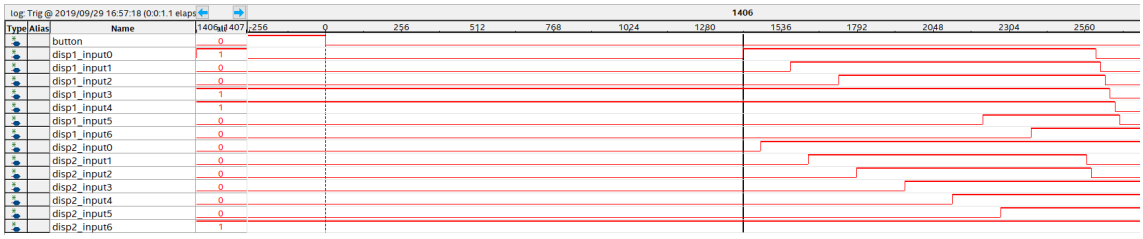


Figure (3) Interrupt Latency Measured with Signal Tap

4.3 Latency Calculation

To calculate the latency based off of Figure 3, we first must calculate the amount of time per cycle using the following equation where T is the time per cycle.

$$f = frequency = 50MHz$$

$$T = 1/f = 1/50MHz = 0.02\mu s$$

Once we have the time per cycle we then use that value to calculate the total latency using the number of cycles between the button press and the change in the state of the 7-segment display, which according to the snapshot in Figure 3 is approximately 1406 cycles. This calculation is done in the following equation where n is the number of cycles.

$$Latency = T * n = 1406cycles * 0.02\mu s = 28.12\mu s$$

As you can see from the equation above the latency between the button press and the response from the 7-segment display is approximately $28.12\mu s$.

5 Conclusion

I received your feedback on Homework 1 a few hours after I had finished this project, and would like to note that I did recieve your feedback about using a 7 bit PIO with a std_logic_vector instead of using individual bits for the display and so I was not able to incorporate that into this homework but I plan on doing so in the future. I think that the most useful thing that I got out of this assignment was learning how to use the SinalTap tool, and how the system.h file works and what it is used for.

A
VHDL Code

```
-- ECE532 HW2
-- Zackary McClamma
--04-Sep-2019

library ieee;
  use ieee.std_logic_1164.all;

entity homework2 is
  port
  (
    i_clk      : in     std_logic;
    i_rst_n    : in     std_logic;
          disp1_input0 : out std_logic;
          disp1_input1 : out std_logic;
          disp1_input2 : out std_logic;
          disp1_input3 : out std_logic;
          disp1_input4 : out std_logic;
          disp1_input5 : out std_logic;
          disp1_input6 : out std_logic;
          disp2_input0 : out std_logic;
          disp2_input1 : out std_logic;
          disp2_input2 : out std_logic;
          disp2_input3 : out std_logic;
          disp2_input4 : out std_logic;
          disp2_input5 : out std_logic;
          disp2_input6 : out std_logic;
          button : in std_logic

  );
end homework2;

architecture sch of homework2 is

  component homework2_cpu is
  port (
    clk_clk        : in std_logic;
    reset_reset_n : in std_logic;
          disp1_input0_export : out std_logic;
          disp1_input1_export : out std_logic;
          disp1_input2_export : out std_logic;
          disp1_input3_export : out std_logic;
          disp1_input4_export : out std_logic;
          disp1_input5_export : out std_logic;
          disp1_input6_export : out std_logic;
          disp2_input0_export : out std_logic;
          disp2_input1_export : out std_logic;
```

```
            disp2_input2_export : out std_logic;
            disp2_input3_export : out std_logic;
            disp2_input4_export : out std_logic;
            disp2_input5_export : out std_logic;
            disp2_input6_export : out std_logic;
            button_export : in std_logic
    );
    end component homework2_cpu;

    signal w_disp1_input0 : std_logic;
    signal w_disp1_input1 : std_logic;
    signal w_disp1_input2 : std_logic;
    signal w_disp1_input3 : std_logic;
    signal w_disp1_input4 : std_logic;
    signal w_disp1_input5 : std_logic;
    signal w_disp1_input6 : std_logic;

begin

          disp1_input0 <= w_disp1_input0;
          disp1_input1 <= w_disp1_input1;
          disp1_input2 <= w_disp1_input2;
          disp1_input3 <= w_disp1_input3;
          disp1_input4 <= w_disp1_input4;
          disp1_input5 <= w_disp1_input5;
          disp1_input6 <= w_disp1_input6;

    u0 : component homework2_cpu
          port map
    (
      clk_clk        => i_clk,
      reset_reset_n => i_rst_n,
            disp1_input0_export => w_disp1_input0,
            disp1_input1_export => w_disp1_input1,
            disp1_input2_export => w_disp1_input2,
            disp1_input3_export => w_disp1_input3,
            disp1_input4_export => w_disp1_input4,
            disp1_input5_export => w_disp1_input5,
            disp1_input6_export => w_disp1_input6,
            disp2_input0_export => disp2_input0,
            disp2_input1_export => disp2_input1,
            disp2_input2_export => disp2_input2,
            disp2_input3_export => disp2_input3,
            disp2_input4_export => disp2_input4,
            disp2_input5_export => disp2_input5,
            disp2_input6_export => disp2_input6,
            button_export => button
    );

end sch;
```

B C Code

B.1 Headers

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Homework 2
 * Date: 30 SEP 2019
 * File: homework2.h
 *
 * */
#ifndef HW2_HEADER_H_
#define HW2_HEADER_H_

#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"



#define DISP1_INPUT0 0x00040000
#define DISP1_INPUT1 0x00041000
#define DISP1_INPUT2 0x00042000
#define DISP1_INPUT3 0x00043000
#define DISP1_INPUT4 0x00044000
#define DISP1_INPUT5 0x00045000
#define DISP1_INPUT6 0x00046000

#define DISP2_INPUT0 0x00047000
#define DISP2_INPUT1 0x00048000
#define DISP2_INPUT2 0x00049000
#define DISP2_INPUT3 0x0004A000
#define DISP2_INPUT4 0x0004B000
#define DISP2_INPUT5 0x0004C000
#define DISP2_INPUT6 0x0004D000

volatile int edge_capture = 0;
volatile int counter;

void displayNumber(int number);
void init_button();
void button_isr(void* context, alt_u32 id);
```

```
#endif /* HW2_HEADER_H_ */
```

B.2 Source

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Homework 2
 * Date: 30 SEP 2019
 * File: main.c
 *
 * */

#include <stdio.h>
#include <unistd.h>
#include "homework2.h"

volatile int counter;

int main(){
        int tst;
        counter = 0;    //Initialize counter to 0
        displayNumber(counter);         //Initialize display to 00
        init_button();  //Resister button ISR

        // Enable Interrupt
        tst = alt_ic_irq_enable(PIO_14_IRQ_INTERRUPT_CONTROLLER_ID,
                                                        PIO_14_IRQ);

        //Check if Interrupt was enabled successfully
        if (tst<0)
                printf("\nFailed to enable interrupt, enable returned %d", tst)
        else
                printf("\n Interrupt Enabled.");

        // Enter infinite loop
        while(1)
        {
                //Check if ISR was triggered
                if (edge_capture !=0)
                {
                        counter++;      //Iterate counter

                        // If counter is greater than 12 reset to 0
                        if (counter > 12)
                        {
                                counter = 0;
                        }

                        // Display new counter value
```

```
                        displayNumber(counter);

                        //Reset edge capture value
                        edge_capture = 0;
                }
        }
        return 0;
}


void displayNumber(int num)
{
        int* display1[] = {DISP1_INPUT0,DISP1_INPUT1,
                                DISP1_INPUT2,DISP1_INPUT3,
                        DISP1_INPUT4,DISP1_INPUT5,
                                DISP1_INPUT6};
        int* display2[] = {DISP2_INPUT0,DISP2_INPUT1,
                                DISP2_INPUT2,DISP2_INPUT3,
                        DISP2_INPUT4,DISP2_INPUT5,
                                DISP2_INPUT6};

        for(int i = 0; i < sizeof(display1)/sizeof(display1[0]); i++)
        {
                *display1[i] = 0x01;
                *display2[i] = 0x01;
        }

        switch(num)
        {
        case 0:
                *display2[0] = 0x00;
                *display2[1] = 0x00;
                *display2[2] = 0x00;
                *display2[3] = 0x00;
                *display2[4] = 0x00;
                *display2[5] = 0x00;

                *display1[0] = 0x00;
                *display1[1] = 0x00;
                *display1[2] = 0x00;
                *display1[3] = 0x00;
                *display1[4] = 0x00;
                *display1[5] = 0x00;
                return;
        case 1:
                *display2[0] = 0x00;
                *display2[1] = 0x00;
                *display2[2] = 0x00;
                *display2[3] = 0x00;
                *display2[4] = 0x00;
```

```
        *display2[5] = 0x00;

        *display1[1] = 0x00;
        *display1[2] = 0x00;
        return;
case 2:
        *display2[0] = 0x00;
        *display2[1] = 0x00;
        *display2[2] = 0x00;
        *display2[3] = 0x00;
        *display2[4] = 0x00;
        *display2[5] = 0x00;

        *display1[0] = 0x00;
        *display1[1] = 0x00;
        *display1[3] = 0x00;
        *display1[4] = 0x00;
        *display1[6] = 0x00;
        return;
case 3:
        *display2[0] = 0x00;
        *display2[1] = 0x00;
        *display2[2] = 0x00;
        *display2[3] = 0x00;
        *display2[4] = 0x00;
        *display2[5] = 0x00;

        *display1[0] = 0x00;
        *display1[1] = 0x00;
        *display1[2] = 0x00;
        *display1[3] = 0x00;
        *display1[6] = 0x00;
        return;
case 4:
        *display2[0] = 0x00;
        *display2[1] = 0x00;
        *display2[2] = 0x00;
        *display2[3] = 0x00;
        *display2[4] = 0x00;
        *display2[5] = 0x00;

        *display1[1] = 0x00;
        *display1[2] = 0x00;
        *display1[5] = 0x00;
        *display1[6] = 0x00;
        return;
case 5:
        *display2[0] = 0x00;
        *display2[1] = 0x00;
        *display2[2] = 0x00;
```

```
            *display2[3] = 0x00;
            *display2[4] = 0x00;
            *display2[5] = 0x00;

            *display1[0] = 0x00;
            *display1[2] = 0x00;
            *display1[3] = 0x00;
            *display1[5] = 0x00;
            *display1[6] = 0x00;
            return;
    case 6:
            *display2[0] = 0x00;
            *display2[1] = 0x00;
            *display2[2] = 0x00;
            *display2[3] = 0x00;
            *display2[4] = 0x00;
            *display2[5] = 0x00;

            *display1[0] = 0x00;
            *display1[2] = 0x00;
            *display1[3] = 0x00;
            *display1[4] = 0x00;
            *display1[5] = 0x00;
            *display1[6] = 0x00;
            return;
    case 7:
            *display2[0] = 0x00;
            *display2[1] = 0x00;
            *display2[2] = 0x00;
            *display2[3] = 0x00;
            *display2[4] = 0x00;
            *display2[5] = 0x00;

            *display1[0] = 0x00;
            *display1[1] = 0x00;
            *display1[2] = 0x00;
            return;
    case 8:
            *display2[0] = 0x00;
            *display2[1] = 0x00;
            *display2[2] = 0x00;
            *display2[3] = 0x00;
            *display2[4] = 0x00;
            *display2[5] = 0x00;

            *display1[0] = 0x00;
            *display1[1] = 0x00;
            *display1[2] = 0x00;
            *display1[3] = 0x00;
            *display1[4] = 0x00;
```

```
                *display1[5] = 0x00;
                *display1[6] = 0x00;
                return;
        case 9:
                *display2[0] = 0x00;
                *display2[1] = 0x00;
                *display2[2] = 0x00;
                *display2[3] = 0x00;
                *display2[4] = 0x00;
                *display2[5] = 0x00;

                *display1[0] = 0x00;
                *display1[1] = 0x00;
                *display1[2] = 0x00;
                *display1[5] = 0x00;
                *display1[6] = 0x00;
                return;
        case 10:
                *display2[1] = 0x00;
                *display2[2] = 0x00;

                *display1[0] = 0x00;
                *display1[1] = 0x00;
                *display1[2] = 0x00;
                *display1[3] = 0x00;
                *display1[4] = 0x00;
                *display1[5] = 0x00;

                return;
        case 11:
                *display2[1] = 0x00;
                *display2[2] = 0x00;

                *display1[1] = 0x00;
                *display1[2] = 0x00;
                return;
        case 12:
                *display2[1] = 0x00;
                *display2[2] = 0x00;

                *display1[0] = 0x00;
                *display1[1] = 0x00;
                *display1[3] = 0x00;
                *display1[4] = 0x00;
                *display1[6] = 0x00;
                return;
        }
}

void button_isr(void* context, alt_u32 id){
```

```
        //Set context to edge_capture_ptr value
        volatile int* edge_capture_ptr = (volatile int*) context;
        //Read edge capture register of PIO_14 (Button)
        *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(PIO_14_BASE);

        //Write 1 to edge capture register
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_14_BASE, 0x1);
        return;
}

void init_button()
{
        //Type cast edge_capture variable so it can
        //be passed to register function
        void* edge_capture_ptr = (void*) &edge_capture;

        //Write 1 to IRQ Mask
        IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PIO_14_BASE, 0x1);
        //Write 0 to edge capture of PIO_14(Button)
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_14_BASE, 0x0);

        //Register Interrupt
        alt_ic_isr_register(PIO_14_IRQ_INTERRUPT_CONTROLLER_ID,
                                                PIO_14_IRQ,
                                                (void*)button_isr,
                                                edge_capture_ptr,
                                                0x0);
}
```