**California State University, Fresno**

**Lyles College of Engineering**

**Electrical and Computer Engineering Department**

**TECHNICAL REPORT**

**Experiment Title:** Software Design with the NIOS II Processor, Assignment 5

**Course Title:** ECE 178 Embedded Systems

**Date Submitted:** April 6, 2015

**Honor Code Statement:**

**“I have done my own work and have neither given nor received**

**unauthorized assistance on this work.”**

|  |  |
| --- | --- |
| **Prepared By:** |  |
| **Christopher Hays** |  |
|  |  |
| **Signature:** |  |

**INSTRUCTOR SECTION**

**Comments:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TABLE OF CONTENTS**

**Section:** **Page:**

Title Page ……………………………………………………………………...…. 1.

Table of Contents ……………………………………………………………….... 2.

1. Statement of Objectives ……………………………………………………….. 3.

2. Theoretical Background …………………….…………………………………. 3.

3. Experimental Procedure ………………………………………………………. 3.

3.1 Equipment Used ……………………………………………………... 3.

3.2 Laboratory Procedure ……………………………………………...… 4.

4. Analysis ……………………………………………………………………...... 6.

5. Conclusions ……………………………………………………………………. 20.

6. Appendix ………………………………………………………………………. 21.

**1. STATEMENT OF OBJECTIVES**

The objective of this assignment was to design interactive eye-tracking and hand-synchronization software for an embedded system that uses the NIOS II processor. The embedded system was designed and tested in the previous assignment and the same one was used here; the programs were created in assembly language and made use of the Altera DE2 development board FPGA as well as its inputs and outputs. The program implemented a timer interrupt to test the user’s reaction time and works as such: After an initial greeting, a random amount of time passes and then the green LEDs begin flashing while the elapsing time is displayed on the seven-segment displays. The user then presses the stop button as fast as they can and the final elapsed time is displayed, showing their reaction time.

**2. THEORETICAL BACKGROUND**

The NIOS II is a 32-bit embedded processor designed for the Altera family of FPGAs. The *Altera Monitor Program* is used to specify the NIOS II build, compile, and load any user-defined programs. This program visually represents the register and memory contents, and provides debugging tools. *Qsys* is a system integration tool that represents the various components of a system graphically, creating an easy way to make/visualize connections between components. Every piece of the system is modeled and libraries of pre-made IP cores are available to be added to a custom system. Once the system components are connected to each other, *Qsys* is used to generate Verilog code and block diagrams to be used within *Quartus II.*

Hardware interrupts are used to pause program execution and execute an exception handler routine. This routine determines which interrupt has taken place and calls the appropriate interrupt service routine (ISR). These ISRs are typically short so the program can resume normal operation. With the NIOS II processor, all interrupts are non-vectored and the interrupt priority is handled by software. In this assignment a system timer and input buttons are used to generate hardware interrupts. The JTAG UART serial communication module is used to generate interrupts in part 2.

**3. EXPERIMENTAL PROCEDURE**

**3.1 Equipment Used**

Altera Monitor Program Software

Altera DE2 FPGA Development Board

Quartus II

Qsys

NIOS II CPU

USB Blaster

**3.2 Laboratory Procedure**

The first step was to determine how to organize the program and how to handle the timer interrupt aspect. As a way of controlling the logic, a software design was created using system states. Any interrupt will give control to the exception handler, which determines the interrupt and calls the appropriate ISR. These ISRs will change the system state register depending on the current state and current input. The six system states are as follows:

* READY (state 1)
* LED OFF (state 2)
* LED ON (state 3)
* STOPPED (state 4)
* OUT (state 5)
* OOPS (state 6)

These states each have their own initialization routine and an infinite loop that constantly checks the current system state. The READY state clears the seven-segment displays and prints the word “hello” as an output. It then waits for user input to start the program (the “start” button), which puts the system into the LED OFF state. The LED OFF state clears the seven-segment display then reads the value from the random number generator. This number is then adjusted to be between 2 and 8 seconds so the user cannot get used to the timing of the synchronization test. The system timer is set with a period of 1ms and the timer is started. The system then loops here as it waits for the random amount of time to pass; if the user presses the “stop” button in this state the program will enter the OOPS state, printing “oops” on the display and stopping.

When the register that counts the timer interrupts matches the random value, the system is put into the LED ON state. This state restarts the timer at zero, allowing the timer counter register to count the elapsed time from this point. Another infinite loop here flashes the LEDs and displays the elapsing time on the seven-segment displays; when one second has passed with no input the system will go the OUT state, printing “out” on the display and stopping. If the user presses the “stop” button while the LEDs are flashing the system will enter the STOPPED state, displaying the elapsed time since the LEDs started flashing. It should be noted that pressing a button that is not appropriate for the current state will do nothing, with the exception of the “clear” button, which works in any state.

In part 2 of the assignment, JTAG read interrupts are enabled, calling the exception handler every time data is available to read from the host keyboard. The JTAG ISR determines what key was pressed and takes the appropriate action; the “c” key acts as the “clear” button, the “s” key acts as the start button, and the “p” key acts as the stop button. The JTAG keys simply act as the appropriate button, calling the same subroutines and using the same logic.

In order to have the elapsed time display in decimal instead of hex, a binary to BCD converting circuit was used to convert the value to BCD then the value was further decoded to seven-segment using software. This circuit was designed in Verilog and uses the “shift and add 3” algorithm for converting binary to BCD. Figure 1 shows an example converting 4 bits; this was adapted to 12 bits for this assignment. The algorithm is as follows:

* Shift the binary number left one bit.
* If 12 shifts have taken place the algorithm is done.
* If the binary value in any of the BCD columns is greater than 4, add 1.
* Go to the beginning.

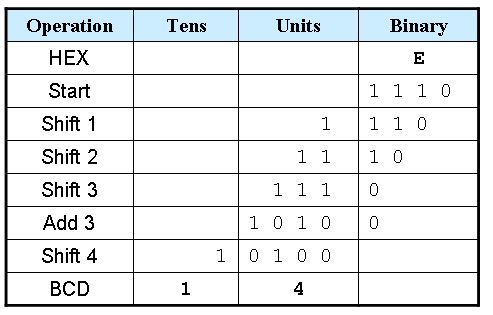


Figure 1: The Binary to BCD Algorithm

The random number used in the LED OFF state was determined by reading a linear feedback shift register (LFSR) that was also designed using Verilog. This register starts with an initial value and every clock cycle the 4th bit is run through an XOR gate with the 1st bit; this result is then fed back as the 0th bit while the register shifts forward. This creates a pattern of 5-bit numbers that, when accessed at a random time by the user, become pseudo-random. Figure 2 shows a block diagram of the LFSR.

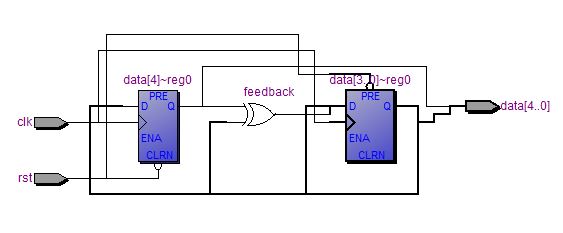


Figure 2: Linear Feedback Shift Register

**4. ANALYSIS**

The program ran as intended and everything worked according to specifications. Figure 3 shows the Quartus II block diagram of the custom hardware that ran the software written, including the binary to BCD circuit and the LFSR. Figure 4 shows successful compilation and Figure 5 shows the detail of the system in Qsys. System and program download are shown in figures 6 and 7. Following this, a line-by-line breakdown analysis of the code is presented.

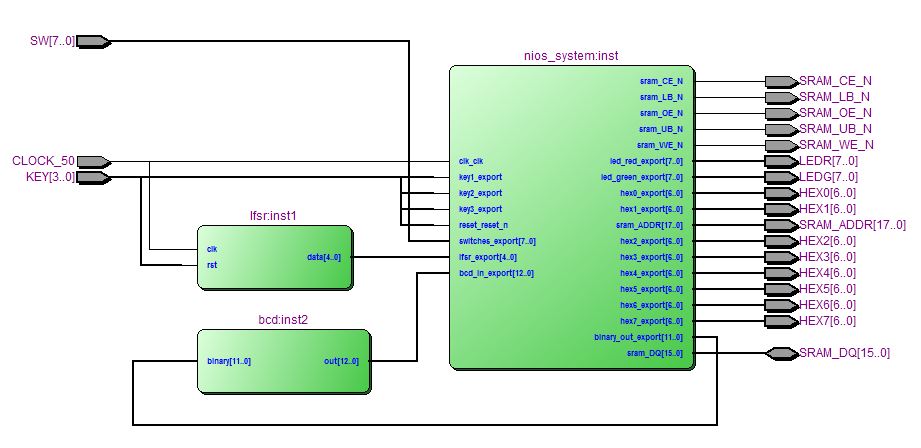


Figure 3: Quartus II Block Diagram

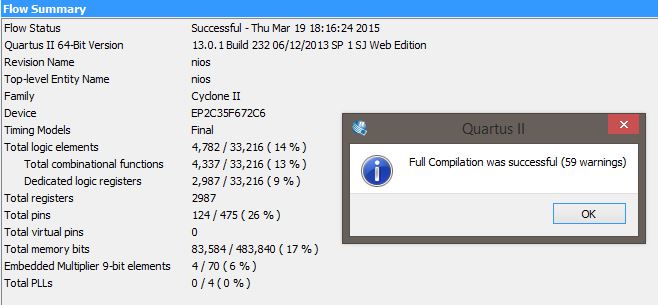


Figure 4: Successful Compilation in Quartus II

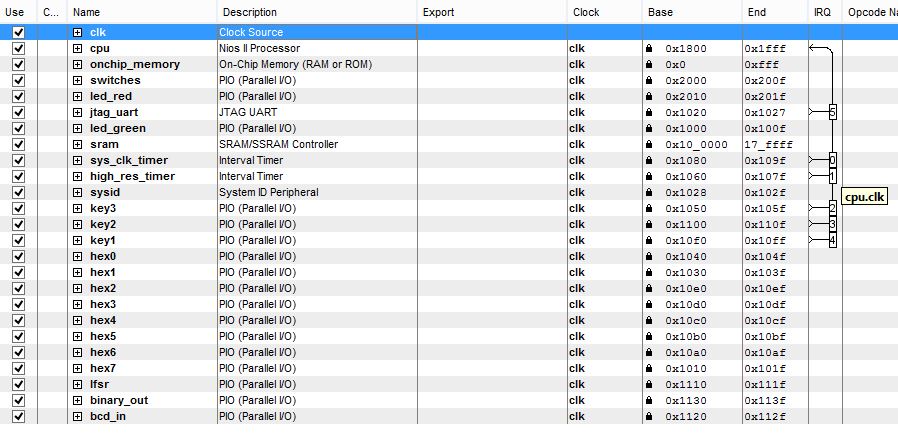


Figure 5: The Qsys System Details

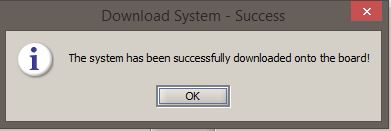


Figure 6: System Download

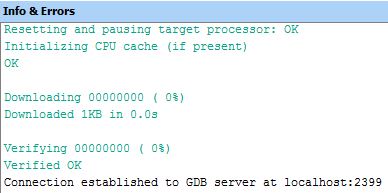


Figure 7: Program Download

**The Assembly Code:**

/\* Christopher Hays \*/

/\* ECE 178 Assignment 5 \*/

/\* Spring 2015 \*/

/\* Interactive eye tracking hand coordination embedded system \*/

/\*\*\*\*\*\*\*\* RESET VECTOR \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.section .reset, "ax" /\* label the reset vector \*/

RESET:

movia sp, 0xff0 /\* the end of the stack \*/

br \_start /\* branch to start \*/

/\*\*\*\*\*\*\*\* EXCEPTION VECTOR \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.section .exceptions, "ax" /\* label the exception vector \*/

EXCEPTION\_HANDLER:

addi sp, sp, -0x4 /\* allocate stack \*/

stw ra, 0(sp) /\* store ra \*/

rdctl et, ipending /\* read the ipending reg \*/

andi et, et, 0b111101 /\* verify the interrupt \*/

beq et, r0, EXCEPTIONS\_DONE /\* if no ints are pending we are done \*/

CHECK\_HARDWARE\_INTERRUPTS: /\* check in the desired order of priority \*/

subi ea, ea, 4 /\* decrement the ea because this is a \*/

/\* hardware interrupt \*/

CHECK0:

andi r13, et, 0b00001 /\* check for irq0, the timer \*/

beq r13, r0, CHECK2

stw r0, 0(r14) /\* clear the TO bit of the timer status register \*/

call isr0 /\* call the ISR \*/

CHECK2:

andi r13, et, 0b00100 /\* check for irq2, key3 "CLEAR" \*/

beq r13, r0, CHECK3

stw r0, 8(r23) /\* disable the pio interrupt \*/

stw r0, 0xc(r23) /\* clear the edge capture register of the pio \*/

call isr2 /\* call the ISR \*/

stw r2, 8(r23) /\* enable the interrupt now that we are done \*/

CHECK3:

andi r13, et, 0b01000 /\* check for irq3, key2 "START" \*/

beq r13, r0, CHECK4

stw r0, 8(r22) /\* disable the pio interrupt \*/

stw r0, 0xc(r22) /\* clear the edge capture register of the pio \*/

call isr3 /\* call the ISR \*/

stw r2, 8(r22) /\* enable the interrupt now that we are done \*/

CHECK4:

andi r13, et, 0b10000 /\* check for irq4, key1 "STOP" \*/

beq r13, r0, CHECK5

stw r0, 8(r21) /\* disable the pio interrupt \*/

stw r0, 0xc(r21) /\* clear the edge capture register of the pio \*/

call isr4 /\* call the ISR \*/

stw r2, 8(r21) /\* enable the interrupt now that we are done \*/

CHECK5:

andi r13, et, 0b100000 /\* check for irq5 \*/

beq r13, r0, EXCEPTIONS\_DONE

call jtag\_isr /\* call the jtag isr \*/

EXCEPTIONS\_DONE:

ldw ra, 0(sp) /\* restore the return address \*/

addi sp, sp, 0x4 /\* de-allocate stack \*/

eret /\* return from the exception handler \*/

/\*\*\*\*\*\*\*\* CONSTANTS \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.equ redleds, 0x00002010

.equ greenleds, 0x00001000

.equ hex0, 0x00001040

.equ hex1, 0x00001030

.equ hex2, 0x000010e0

.equ hex3, 0x000010d0

.equ hex4, 0x000010c0

.equ hex5, 0x000010b0

.equ hex6, 0x000010a0

.equ hex7, 0x00001010

.equ key1, 0x000010f0

.equ key2, 0x00001100

.equ key3, 0x00001050

.equ timerbase, 0x00001080

.equ jtag, 0x00001020

.equ lfsr, 0x00001110

.equ binary\_out, 0x00001130

.equ bcd\_in, 0x00001120

.equ ms, 50000

/\*\*\*\*\*\*\*\* START \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.global \_start

\_start:

movia r2, 0x1 /\* a register to hold a 1 constant \*/

movia sp, 0xff0 /\* create a stack pointer address \*/

movia r23, key3 /\* set base address of the key interrupts \*/

movia r22, key2

movia r21, key1

movia r20, hex4 /\* set base address of the hex displays \*/

movia r19, hex3

movia r18, hex2

movia r17, hex1

movia r16, hex0

movia r15, greenleds /\* set base address of green leds \*/

movia r14, timerbase /\* set base address of the system timer \*/

movia r10, lfsr /\* base of the lfsr PIO \*/

movia r6, jtag /\* base of jtag \*/

movia r7, binary\_out /\* base of binary output PIO \*/

movia r8, bcd\_in /\* base of bcd input PIO \*/

stwio r2, 8(r23) /\* enable the key3 interrupt \*/

stwio r2, 8(r22) /\* enable the key2 interrupt \*/

stwio r2, 8(r21) /\* enable the key1 interrupt \*/

wrctl status, r2 /\* this enables global interrupts \*/

movia r13, 0b111101 /\* enables jtag, irq4, irq3, irq2, timer \*/

wrctl ienable, r13

stwio r2, 4(r6) /\* enable the read interrupt for the jtag \*/

movia r4, 0x0 /\* the main millisecond counter \*/

movia r5, 0x0 /\* counter for random timer \*/

movia r11, 0x0 /\* counter for LED flash \*/

movia r3, 0x1 /\* the state register, begin in the READY state \*/

movia r13, 0b1000 /\* set timer into idle mode \*/

stwio r13, 4(r14) /\* this sets the stop bit \*/

/\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LOOP:

/\* r13 is the generic reg for operations \*/

/\* use r3 for the program state \*/

READY:

subi r13, r3, 0x1 /\* branch if we are NOT in state 1 \*/

bne r13, r0, LED\_OFF

call ready\_ini /\* initialize the ready state \*/

READY\_LOOP: /\* loop in the ready state \*/

subi r13, r3, 0x1

beq r13, r0, READY\_LOOP

LED\_OFF:

subi r13, r3, 0x2 /\* branch if we are NOT in state 2 \*/

bne r13, r0, LED\_ON

call led\_off\_ini /\* initialize the led\_off state \*/

LED\_OFF\_LOOP: /\* loop in led\_off state \*/

subi r13, r3, 0x2

beq r13, r0, LED\_OFF\_LOOP

LED\_ON:

subi r13, r3, 0x3 /\* branch if we are NOT in state 3 \*/

bne r13, r0, STOPPED

call led\_on\_ini /\* initialize the led\_on state \*/

LED\_ON\_LOOP: /\* loop in led\_on state \*/

call flash /\* flash the leds \*/

call display /\* display the millisecond counter on 7-segments \*/

subi r13, r3, 0x3

beq r13, r0, LED\_ON\_LOOP

STOPPED:

subi r13, r3, 0x4 /\* branch if we are NOT in state 4 \*/

bne r13, r0, OUT\_STATE

call stopped\_ini /\* initialize the stopped state \*/

STOPPED\_LOOP: /\* loop in the stopped state \*/

subi r13, r3, 0x4

beq r13, r0, STOPPED\_LOOP

OUT\_STATE:

subi r13, r3, 0x5 /\* branch if NOT in state 5 \*/

bne r13, r0, OOPS

call out\_state\_ini /\* initialize the out state \*/

OUT\_STATE\_LOOP: /\* loop in out state \*/

subi r13, r3, 0x5

beq r13, r0, OUT\_STATE\_LOOP

OOPS:

subi r13, r3, 0x6 /\* branch if NOT in state 6 \*/

bne r13, r0, END\_MAIN

call oops\_ini /\* initialize the oops state \*/

OOPS\_LOOP: /\* loop in oops state \*/

subi r13, r3, 0x6

beq r13, r0, OOPS\_LOOP

END\_MAIN: /\* end main \*/

br LOOP

/\*\*\*\*\*\*\*\* ISR SUBROUTINES \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.global isr0 /\* define isr0, the timer interrupt \*/

isr0:

subi r13, r3, 0x2 /\* branch if we are NOT in state 2 \*/

bne r13, r0, ISR0\_NOTCASE2

addi r5, r5, 0x1 /\* increment the random timer counter \*/

sub r13, r5, r9 /\* check against random counter timeout value \*/

beq r13, r0, SET\_LED\_ON /\* branch if random counter matches timeout value \*/

ret

SET\_LED\_ON:

movia r3, 0x3 /\* set state as LED\_ON \*/

ret

ISR0\_NOTCASE2:

subi r13, r3, 0x3 /\* branch if we are NOT in state 3 \*/

bne r13, r0, SKIP\_ISR0

addi r4, r4, 0x1 /\* increment the main millisecond counter \*/

addi r11, r11, 0x1 /\* increment the LED timeout counter \*/

subi r13, r4, 1000 /\* branch if a second has passed by \*/

beq r13, r0, SET\_OUT

ret

SET\_OUT:

movia r3, 0x5 /\* set state as OUT\_STATE \*/

SKIP\_ISR0: /\* this ensures that isr0 does nothing when in the other states \*/

ret /\* return \*/

.global isr2 /\* define isr2, the key3 interrupt "CLEAR" \*/

isr2:

movia r3, 0x1 /\* put the system into the READY state \*/

movia r4, 0x0 /\* set the main millisecond counter back to 0 \*/

movia r5, 0x0 /\* set the random timer counter to 0 \*/

movia r11, 0x0 /\* set the led timeout counter to 0 \*/

ret

.global isr3 /\* define isr3, the key2 interrupt "START" \*/

isr3:

subi r13, r3, 0x1 /\* branch if we are NOT in state 1 \*/

bne r13, r0, SKIP\_ISR3

movia r3, 0x2 /\* put the system into LED\_OFF state \*/

SKIP\_ISR3: /\* this ensures that isr3 does nothing when in the other states \*/

ret /\* return \*/

.global isr4 /\* define isr4, the key1 interrupt "STOP" \*/

isr4:

subi r13, r3, 0x2 /\* branch if NOT in state 2 \*/

bne r13, r0, ISR4\_NOTCASE2

movia r3, 0x6 /\* set system to OOPS state \*/

ret

ISR4\_NOTCASE2:

subi r13, r3, 0x3 /\* branch if NOT in state 3 \*/

bne r13, r0, SKIP\_ISR4

movia r3, 0x4 /\* set system to STOPPED state \*/

SKIP\_ISR4: /\* this ensures that isr4 does nothing when in the other states \*/

ret

.global jtag\_isr

jtag\_isr:

ldwio r13, 0(r6) /\* read the jtag data reg \*/

andi r13, r13, 0xff /\* mask the input \*/

subi sp, sp, 4 /\* allocate stack \*/

stwio r12, 0(sp) /\* push r12 \*/

CHECK\_C:

subi r12, r13, 0x63 /\* check for ascii 'c' \*/

bne r12, r0, CHECK\_S

stwio r13, 0(r6) /\* output to the jtag data reg \*/

movia r3, 0x1 /\* put the system into the READY state \*/

movia r4, 0x0 /\* set the main millisecond counter back to 0 \*/

movia r5, 0x0 /\* set the random timer counter to 0 \*/

movia r11, 0x0 /\* set the led timeout counter to 0 \*/

CHECK\_S:

subi r12, r13, 0x73 /\* check for ascii 's' \*/

bne r12, r0, CHECK\_P

stwio r13, 0(r6) /\* output to the jtag data reg \*/

subi r13, r3, 0x1 /\* branch if we are NOT in state 1 \*/

bne r13, r0, JTAG\_END

movia r3, 0x2 /\* put the system into LED\_OFF state \*/

CHECK\_P:

subi r12, r13, 0x70 /\* check for ascii 'p' \*/

bne r12, r0, JTAG\_END

stwio r13, 0(r6) /\* output to the jtag data reg \*/

subi r13, r3, 0x2 /\* branch if NOT in state 2 \*/

bne r13, r0, JTAG\_NOTCASE2

movia r3, 0x6 /\* set system to OOPS state \*/

ret

JTAG\_NOTCASE2:

subi r13, r3, 0x3 /\* branch if NOT in state 3 \*/

bne r13, r0, JTAG\_END

movia r3, 0x4 /\* set system to STOPPED state \*/

JTAG\_END:

ldwio r12, 0(sp) /\* pop r12 \*/

addi sp, sp, 4 /\* de-allocate stack \*/

ret

/\*\*\*\*\*\*\*\* SYSTEM STATE SUBROUTINES \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.global ready\_ini /\* state 1 \*/

ready\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

stwio r0, 0(r15) /\* turn off the green leds \*/

call seven\_segments\_off /\* turn off seven segment displays \*/

call print\_hello /\* print hello on seven-segments \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global led\_off\_ini /\* state 2 \*/

led\_off\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

call seven\_segments\_off /\* clear the seven-segments \*/

/\* start random timer, r9 holds the timeout value \*/

ldwio r13, 0(r10) /\* read lfsr \*/

andi r13, r13, 0b01110 /\* take the middle 3 bits \*/

srli r13, r13, 1 /\* shift them right by 1 \*/

addi r13, r13, 1 /\* add 1 for a range of 1 to 8 \*/

bne r13, r2, IN\_RANGE /\* branch if value is between 2 and 8 \*/

addi r13, r13, 1 /\* if random value is 1, add 1 to become 2 \*/

IN\_RANGE:

slli r9, r13, 10 /\* multiply by 1024, shift left logical \*/

/\* makes range of approx 2000 - 8000 ms \*/

addi r13, r0, %lo(ms) /\* write the period value to timer ip core \*/

stwio r13, 0x8(r14) /\* period is 1 ms \*/

addi r13, r0, %hi(ms) /\* every timeout interrupt \*/

/\* increases the counter \*/

stwio r13, 0xc(r14)

movia r13, 0b111 /\* start the timer and enable the int \*/

stwio r13, 0x4(r14) /\* write to timer control reg \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global led\_on\_ini /\* state 3 \*/

led\_on\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

movia r13, 0b111 /\* start timer, continuous, with interrupt \*/

stwio r13, 0x4(r14) /\* write to timer control reg \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global stopped\_ini /\* state 4 \*/

stopped\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

movia r13, 0b1000 /\* set timer into idle mode \*/

call display /\* display the reaction time \*/

stwio r13, 4(r14) /\* write to timer control reg \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global out\_state\_ini /\* state 5 \*/

out\_state\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

call seven\_segments\_off /\* clear the display \*/

call print\_out /\* print out to seven-segments \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global oops\_ini /\* state 6 \*/

oops\_ini:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

call seven\_segments\_off /\* clear the display \*/

call print\_oops /\* print oops to seven-segments \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

/\*\*\*\*\*\*\*\* OTHER SUBROUTINES \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

.global flash /\* flashes leds every 100 ms \*/

flash:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

subi r12, r11, 200 /\* check the led time counter \*/

bne r12, r0, NOT200 /\* branch if the counter is not 200 \*/

movia r13, 0b1000 /\* if 200, flash led3 \*/

stwio r13, 0(r15)

subi r11, r11, 200 /\* reset the led time counter \*/

NOT200:

subi r12, r11, 100 /\* check the led time counter \*/

bne r12, r0, NOT100 /\* branch if the counter is not 100 \*/

movia r13, 0b0100 /\* if 100, flash led2 \*/

stwio r13, 0(r15)

NOT100:

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global seven\_segments\_off

seven\_segments\_off:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

movia r13, 0xff /\* turn off the seven-segment \*/

stwio r13, 0(r16) /\* they are active low \*/

stwio r13, 0(r17)

stwio r13, 0(r18)

stwio r13, 0(r19)

stwio r13, 0(r20)

movia r12, hex5 /\* unused seven-segments \*/

stwio r13, 0(r12)

movia r12, hex6

stwio r13, 0(r12)

movia r12, hex7

stwio r13, 0(r12)

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global print\_hello

print\_hello:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

/\* write to hex4, hex3, hex2, hex1, hex0 \*/

movia r13, 0x09 /\* H \*/

stwio r13, 0(r20)

movia r13, 0x06 /\* E \*/

stwio r13, 0(r19)

movia r13, 0x47 /\* L \*/

stwio r13, 0(r18)

movia r13, 0x47 /\* L \*/

stwio r13, 0(r17)

movia r13, 0x40 /\* O \*/

stwio r13, 0(r16)

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global print\_out

print\_out:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

/\* write to hex2, hex1, hex0 \*/

movia r13, 0x40 /\* O \*/

stwio r13, 0(r18)

movia r13, 0x41 /\* U \*/

stwio r13, 0(r17)

movia r13, 0x07 /\* T \*/

stwio r13, 0(r16)

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global print\_oops

print\_oops:

subi sp, sp, 4

stwio ra, 0(sp) /\* push ra to the stack \*/

/\* write to hex3, hex2, hex1, hex0 \*/

movia r13, 0x40 /\* O \*/

stwio r13, 0(r19)

movia r13, 0x40 /\* O \*/

stwio r13, 0(r18)

movia r13, 0x0c /\* P \*/

stwio r13, 0(r17)

movia r13, 0x12 /\* S \*/

stwio r13, 0(r16)

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 4 /\* de-allocate \*/

ret

.global display /\* displays the ms counter on the 7-segments \*/

display:

subi sp, sp, 16

stwio ra, 0(sp) /\* push ra to the stack \*/

stwio r13, 4(sp) /\* push r13 to the stack \*/

stwio r4, 0(r7) /\* write ms count to the hardware decoder \*/

nop

ldwio r13, 0(r8) /\* read the bcd\_in \*/

stwio r13, 8(sp) /\* store full bcd reg on the stack \*/

andi r13, r13, 0xf /\* mask lower 4 bits \*/

call decode /\* return the decoded value in r13 \*/

stwio r13, 0(r16) /\* write to hex0 \*/

ldwio r13, 8(sp) /\* load full bcd reg from stack \*/

andi r13, r13, 0xf0 /\* mask middle 4 bits \*/

srli r13, r13, 4 /\* shift right logical for the subroutine \*/

call decode /\* return the decoded value in r13 \*/

stwio r13, 0(r17) /\* write to hex1 \*/

ldwio r13, 8(sp) /\* load the full bcd reg from the stack \*/

andi r13, r13, 0xf00 /\* mask the upper 4 bits \*/

srli r13, r13, 8 /\* shift right logical for the subroutine \*/

call decode /\* return the decoded value in r13 \*/

stwio r13, 0(r18) /\* write to hex2 \*/

ldwio r13, 4(sp) /\* pop r13 from the stack \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 16 /\* de-allocate \*/

ret

.global decode

/\* converts bcd to 7-segment \*/

decode: /\* stores return value in r13 \*/

subi sp, sp, 20

stwio ra, 0(sp) /\* push ra to the stack \*/

stwio r7, 4(sp) /\* push r7 to the stack \*/

stwio r8, 8(sp) /\* push r8 to the stack \*/

CASE0:

subi r7, r13, 0x0 /\* if r13 is zero \*/

bne r7, r0, CASE1

movia r8, 0x000000c0 /\* output 7-segment zero \*/

CASE1:

subi r7, r13, 0x1 /\* etc \*/

bne r7, r0, CASE2

movia r8, 0x000000f9

CASE2:

subi r7, r13, 0x2

bne r7, r0, CASE3

movia r8, 0x000000a4

CASE3:

subi r7, r13, 0x3

bne r7, r0, CASE4

movia r8, 0x00000030

CASE4:

subi r7, r13, 0x4

bne r7, r0, CASE5

movia r8, 0x00000019

CASE5:

subi r7, r13, 0x5

bne r7, r0, CASE6

movia r8, 0x00000012

CASE6:

subi r7, r13, 0x6

bne r7, r0, CASE7

movia r8, 0x00000002

CASE7:

subi r7, r13, 0x7

bne r7, r0, CASE8

movia r8, 0x00000078

CASE8:

subi r7, r13, 0x8

bne r7, r0, CASE9

movia r8, 0x00000000

CASE9:

subi r7, r13, 0x9

bne r7, r0, DONE

movia r8, 0x00000018

DONE:

mov r13, r8 /\* move the answer into r13 \*/

ldwio r8, 8(sp) /\* pop r8 from the stack \*/

ldwio r7, 4(sp) /\* pop r7 from the stack \*/

ldwio ra, 0(sp) /\* pop ra from the stack \*/

addi sp, sp, 20 /\* de-allocate \*/

ret

.end

**The Verilog Code:**

// Christopher Hays

// Linear Feedback Shift Register

// Creates a pseudo-random 5 bit output

// Uses a feedback loop to accomplish this

module lfsr (input clk,

input rst,

output reg [4:0] data

);

wire feedback = data[4] ^ data[1] ; // xor bit 4 and bit 1

always @(posedge clk or negedge rst) begin

if (~rst)

data <= 4'hf; // 4 bit hex F

else

data <= {data[3:0], feedback}; // LSB is replaced with the xor output

end

endmodule

// Christopher Hays

// 12-bit binary to BCD converter

// Uses the "shift and add 3" algorithm

module bcd ( input [11:0] binary,

output reg [11:0] out

);

reg [3:0] hundreds; // holds the bcd values

reg [3:0] tens;

reg [3:0] ones;

integer i;

always @(binary) begin // always on input change

hundreds = 4'd0;

tens = 4'd0;

ones = 4'd0;

for (i = 11; i>=0; i=i-1) begin // for all 12 bits

if (hundreds >= 5) // check if the column has a value >= 5

hundreds = hundreds + 3; // if so, add 3

if (tens >= 5) // check tens

tens = tens + 3;

if (ones >= 5) // check ones

ones = ones + 3;

hundreds = hundreds << 1; // shift all values left by one bit

hundreds[0] = tens[3];

tens = tens << 1;

tens[0] = ones[3];

ones = ones << 1;

ones[0] = binary[i]; // lsb comes from the input, location i

end

out = {hundreds[3:0],tens[3:0],ones[3:0]}; // concatenate output

end

endmodule

**5. CONCLUSIONS**

This assignment demonstrated the ability to create custom software to test the reaction time of the user and display the results. An interval timer and its corresponding interrupt were used to keep track of time while waiting for user input, and JTAG UART serial communication was used to take place of the buttons on the development board. System states were used to keep the flow of the program organized, ensuring that everything happened when it was supposed to and there were no bugs. The method of having the timers constantly interrupt every 1ms and increment a counter was used because the interrupts here are non-vectored, thus you cannot interrupt an interrupt, and this method helped keep the ISRs at a reasonable length. This project incorporated everything learned in the class up to this point and helped solidify these concepts.

**6. Appendix**

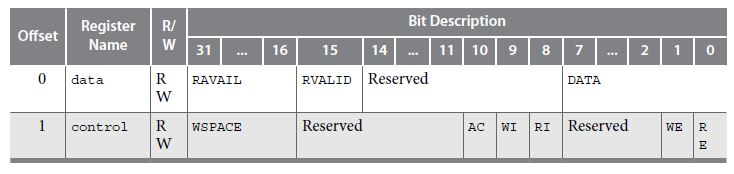


Figure 8: JTAG Register Map

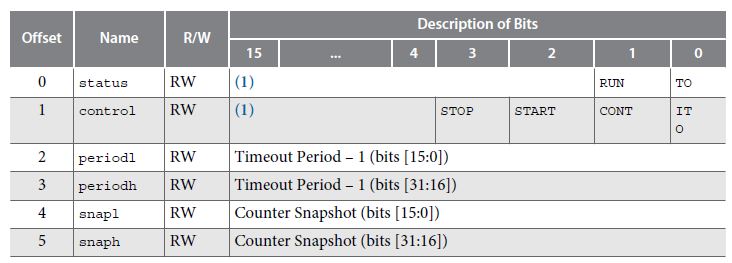


Figure 9: Interval Timer Register Map