**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 6

**Course Title:** ECE 178 Embedded Systems

**Date Submitted:** April 29, 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 ……………………………………………………………………. 9.

6. Appendix ………………………………………………………………………. 10.

**1. STATEMENT OF OBJECTIVES**

The objective of this assignment was to design software that would read a 12-key keypad and display the output on an embedded system that uses the NIOS II processor. The interface used the GPIO 40-pin input/output of the DE2 development board. 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.

**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.*

Keypads are used in a variety of embedded systems and can be read by either the interrupt method or the polling method. Using the interrupt method will read the keypad only when a hardware interrupt is generated from a button press. The key value information will be used as needed then the program can resume normal operation. In the polling method, the columns and rows of the keypad are constantly being written to and read while the program waits for input. When input is detected, a subroutine can be called to handle it quickly before the program resumes scanning the keypad.

**3. EXPERIMENTAL PROCEDURE**

**3.1 Equipment Used**

Altera Monitor Program Software

Altera DE2 FPGA Development Board

Quartus II

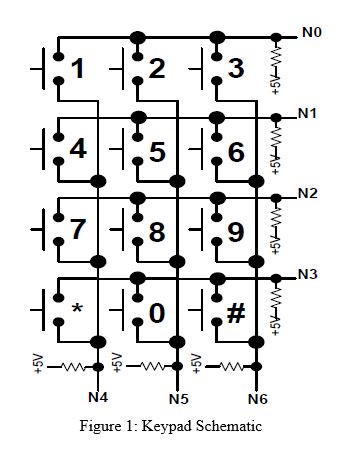
Qsys

NIOS II CPU

USB Blaster

**3.2 Laboratory Procedure**

The first step was to decide on the method of reading the keypad. The polling method was chosen because when using interrupts with the GPIO of the DE2, an interrupt is generated on each logic change. This means an interrupt will occur when the key is pressed and when it is lifted as well. This, along with the de-bouncing, made it feel like the key was constantly being polled anyway so the polling method was chosen.



The keypad was arranged with 3 columns and 4 rows, with 10k pull-up resistors on each pin for a default logic value of 1. The main program ran in an infinite loop and within each loop the following happened:

* The first column was set to logic zero
* The rows were scanned one at a time
* A position counter was incremented after each row scan
* If a zero was detected on a row, a de-bounce function was called
* The de-bounce function waits until the key is lifted and calls a display function
* The display function outputs the key value to the display
* If no zero was detected, continue the scan on column 2 then column 3

The keypad was interfaced to the DE2 board via a 40-pin ribbon cable connected to the GPIO. A cable for a Raspberry PI was used, so the pinout did not exactly match up to the DE2, but each pin was tested and enough of them were found to be useful. Figure 2 shows a diagram of the usable pins; the corresponding number was written next to pins that would work and pins that did not were left blank. A 7-pin bi-directional PIO core was created in Qsys, with 3 pins for output and 4 for input.

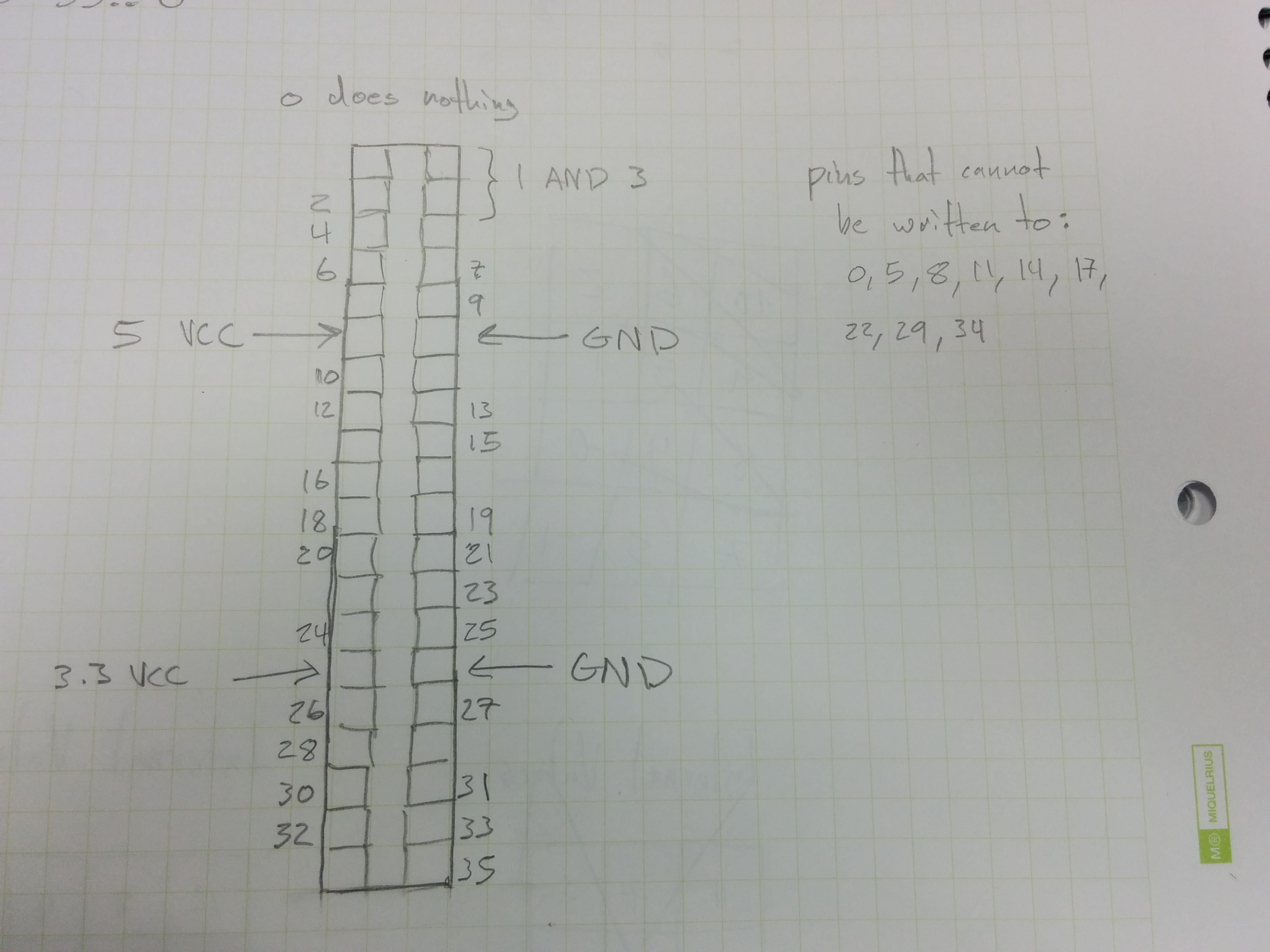


Figure 2: The GPIO Pinout.

The key position counter, which incremented after each row scan, kept track of exactly which button was being scanned. This value was reset every time the main program loop reset. Figure 3 shows that the column/row scan method began scanning in the bottom right corner of the keypad and worked its way up each column before starting the next one. The position counter values for each key are shown. This position value would be used as an offset to a lookup table, used to determine the actual value of the key being pressed, then this key value was converted from binary to BCD and output to the seven-segment display and red LEDs.



Figure 3: Key Position Value Map

**4. ANALYSIS**

The program ran as intended and everything worked according to specifications. Figures 4 through 9 show the Qsys system, the successful compilation, the RTL block diagram, the system download and code compilation, as well as a picture of the keypad on the breadboard. When pressed, the correct key value would display on the seven-segment display of the DE2 board, indicating success. A line-by-line breakdown of the code is in the Appendix.

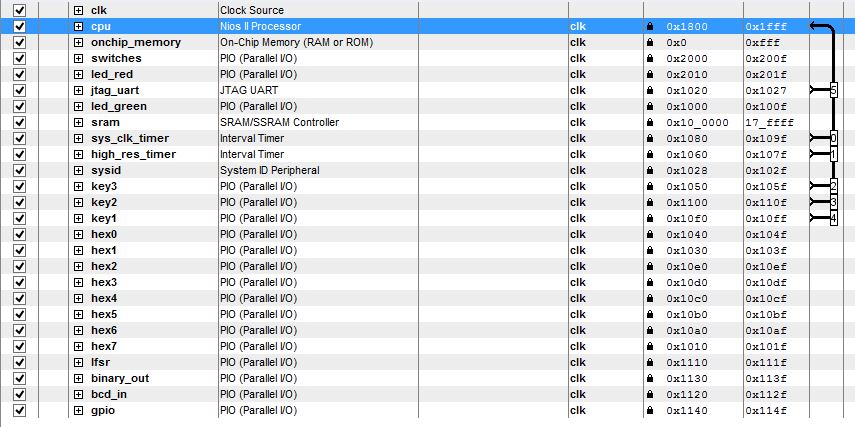


Figure 4: Custom Qsys System

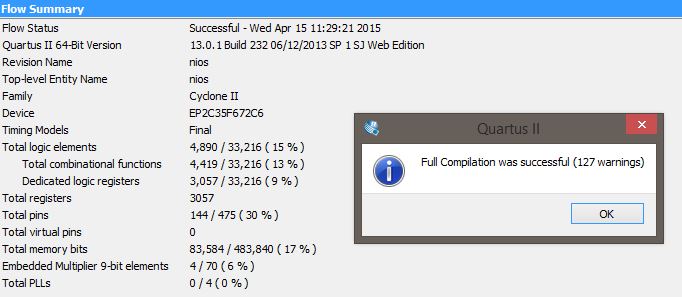


Figure 5: Successful Compilation



Figure 6: RTL Block Diagram

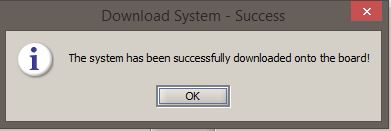


Figure 7: Successful Download

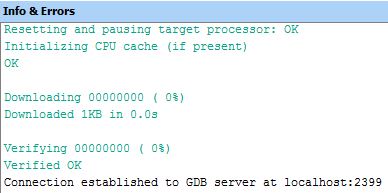


Figure 8: Successful Code Compilation and Download

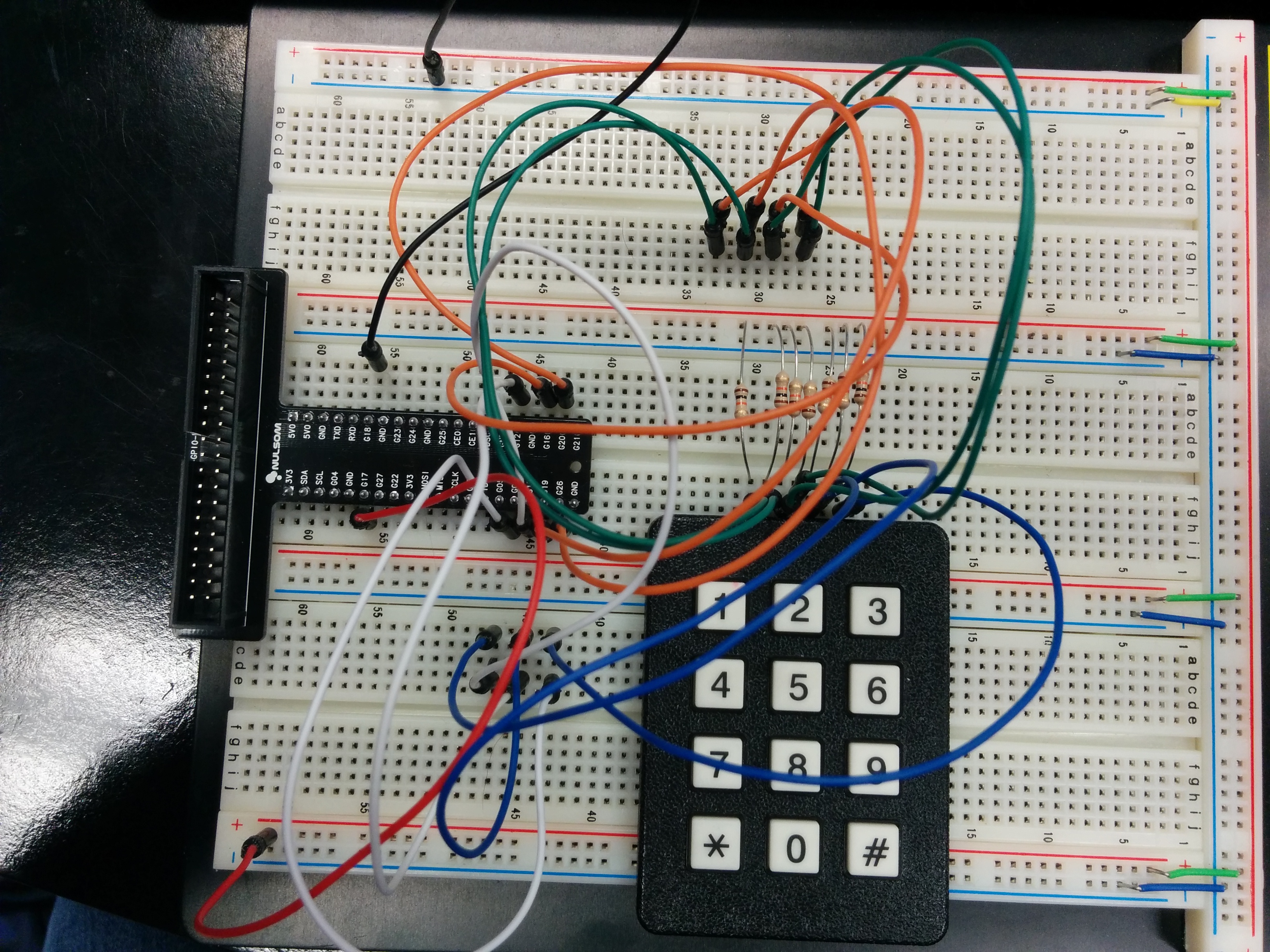


Figure 9: The Keypad and Ribbon Cable Breakout Attachment

**5. CONCLUSION**

This assignment demonstrated how to interface a custom processor on an FPGA with outside hardware. This particular embedded system polls a keypad and the resulting presses could be used in many applications. The GPIO connection is ideal for interfacing with outside hardware, although the hardware interrupts for it act differently than the rest of the system. This knowledge will help with the final ECE-178 project and the Senior Design project, which also interfaces with outside hardware.

**The Assembly Code:**

/\* Christopher Hays \*/

/\* ECE 178 Assignment 6 \*/

/\* Spring 2015 \*/

/\* Keypad interface \*/

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

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 gpio, 0x00001140

.equ binary\_out, 0x00001130

.equ bcd\_in, 0x00001120

.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 key\_table, 0x320 /\* the address of the lookup tables \*/

.equ seven\_segment\_lookup, 0x420

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

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

.global \_start

\_start:

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

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

call clear\_display

movia r4, gpio

movia r5, redleds

movia r6, greenleds

movia r15, binary\_out

movia r16, bcd\_in

movia r3, 0x7

stwio r3, 4(r4) /\* set inputs and outputs of gpio \*/

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

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

main\_loop:

movia r23, 0x0 /\* no key press flag\*/

movia r22, 0x0 /\* position of the current key \*/

movia r3, 0b110 /\* ground the first column \*/

stwio r3, 0(r4) /\* output to gpio \*/

call check\_key /\* scan the rows \*/

beq r23, r0, first\_done /\* call display if key press \*/

call display

first\_done:

movia r3, 0b101 /\* ground the second column \*/

stwio r3, 0(r4) /\* output to gpio \*/

call check\_key /\* scan the rows \*/

beq r23, r0, second\_done /\* call display if key press \*/

call display

second\_done:

movia r3, 0b011 /\* ground the third column \*/

stwio r3, 0(r4) /\* output to gpio \*/

call check\_key /\* scan the rows \*/

beq r23, r0, third\_done /\* call display if key press \*/

call display

third\_done:

br main\_loop

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

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

.global check\_key /\* scans the rows for a key press \*/

check\_key:

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

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

movia r7, 0x4 /\* counter to loop 4 times \*/

ldwio r3, 0(r4) /\* read the gpio input \*/

andi r10, r3, 0x78 /\* mask input bits, store in r10 \*/

stwio r3, 0(r6) /\* output to green leds \*/

check\_loop:

srli r10, r10, 0x1 /\* shift logical right \*/

andi r3, r10, 0x4 /\* mask bit 2 \*/

bne r3, r0, continue\_scan /\* if zero call debounce \*/

call debounce

continue\_scan:

addi r22, r22, 0x1 /\* increment the key counter \*/

subi r7, r7, 0x1 /\* decrement row counter \*/

bne r7, r0, check\_loop /\* loop if not zero \*/

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

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

ret

.global debounce /\* debounces the current key press \*/

debounce:

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

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

debounce\_loop: /\* loops until the input is all ones \*/

/\* the key has been let go \*/

ldwio r3, 0(r4) /\* read the gpio input \*/

andi r11, r3, 0x78 /\* mask input bits, store in r11 \*/

srli r11, r11, 0x1 /\* shift logical right \*/

andi r3, r11, 0x4 /\* mask bit 2 \*/

beq r3, r0, debounce\_loop /\* if zero, start the loop over \*/

srli r11, r11, 0x1

andi r3, r11, 0x4

beq r3, r0, debounce\_loop

srli r11, r11, 0x1

andi r3, r11, 0x4

beq r3, r0, debounce\_loop

srli r11, r11, 0x1

andi r3, r11, 0x4

beq r3, r0, debounce\_loop

addi r23, r23, 0x1 /\* set the key pressed flag \*/

movia r7, 0x1 /\* set the scan loop counter \*/

subi r22, r22, 0x1 /\* correct the key position \*/

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

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

ret

.global display /\* displays the current key press \*/

display:

addi sp, sp, -0x8 /\* allocate stack \*/

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

movia r8, key\_table

add r20, r8, r22 /\* add key position to the lookup table address \*/

ldbio r22, 0(r20) /\* read the byte at this address \*/

/\* now we have the final key value \*/

stwio r22, 0(r5) /\* write key value to the red leds \*/

movia r8, seven\_segment\_lookup

stwio r22, 0(r15) /\* write to binary out \*/

nop

ldwio r22, 0(r16) /\* read the bcd in \*/

stw r22, 4(sp) /\* write the bcd to the stack \*/

andi r22, r22, 0xf /\* mask lower 4 bits \*/

add r20, r8, r22 /\* add bcd value to table address \*/

ldbio r22, 0(r20) /\* read decoded value \*/

movia r8, hex0 /\* 7 segment address \*/

stwio r22, 0(r8) /\* write to seven segment \*/

movia r8, seven\_segment\_lookup

ldw r22, 4(sp) /\* load from the stack \*/

andi r22, r22, 0xf0 /\* mask middle 4 bits \*/

srli r22, r22, 4 /\* shift right logical \*/

add r20, r8, r22 /\* add bcd value to table address \*/

ldbio r22, 0(r20) /\* read decoded value \*/

movia r8, hex1 /\* 7 segment address \*/

stwio r22, 0(r8) /\* write to seven segment \*/

movia r23, 0x0 /\* clear the key pressed flag \*/

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

addi sp, sp, 0x8 /\* de-allocate stack \*/

ret

.global clear\_display

clear\_display:

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

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

movia r3, 0xff /\* clear the 7 segments \*/

movia r4, hex0

stwio r3, 0(r4)

movia r4, hex1

stwio r3, 0(r4)

movia r4, hex2

stwio r3, 0(r4)

movia r4, hex3

stwio r3, 0(r4)

movia r4, hex4

stwio r3, 0(r4)

movia r4, hex5

stwio r3, 0(r4)

movia r4, hex6

stwio r3, 0(r4)

movia r4, hex7

stwio r3, 0(r4)

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

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

ret

.org 0x300

key\_lookup\_table:

.byte 11, 9, 6, 3, 0, 8, 5, 2, 10, 7, 4, 1

.org 0x400

seven\_segment\_lookup:

.byte 0xc0, 0xf9, 0xa4, 0x30, 0x19, 0x12, 0x02, 0x78, 0x00, 0x18

.end