Senior Project

Sign Language Glove

By
Nicholas Born

Senior Project
ELECTRICAL ENGINEERING DEPARTMENT
California Polytechnic State University
San Luis Obispo
2010

Table of Contents

List of Tables and Figures

Acknowledgements

Thank you to all my friends and family. My parents, Anne and Mike, my brothers and sisters, my family at St. Joseph House and Casa Maria, and all of Newman, especially Fathers John and Kevin.

I would also like to thank Professors DePiero and Derickson for helping and supporting me through my college career.

Introduction

Between 500,000 and 2 million Americans use American Sign Language (ASL) as their primary language. As such, a device capable of translating those movements into text would be useful in many areas, such as teaching and the entertainment industry.

The goal of this project was to show that the creation of a glove device that takes in sign language letters, interprets them into the desired characters, and can input those characters into a computer is feasible. I chose to work on this project for many different reasons. One reason is many members of my family know sign language and use it extensively so knowledge of this area would be fun and enjoyable. The other main reason is my interest in computer input and interaction. A glove type device allows new and different interactions with the computer. Ideally, computer interaction would involve one hand on the mouse and one on a keyboard like device. This glove would allow one handed keyboard capability.

Background

American Sign Language (ASL)

American Sign Language is the predominant form of visual hand language used in North America. It consists of different hand gestures for different letters, numbers, and words. Each sign consists of hand and finger orientation and movement. The American Manual Alphabet consists of 26 signs standing for the 26 letters in the alphabet [6]. These signs include finger orientation and, in a couple of cases, movement of the hand. Figure 1 below shows all the hand gestures in the Manual Alphabet.

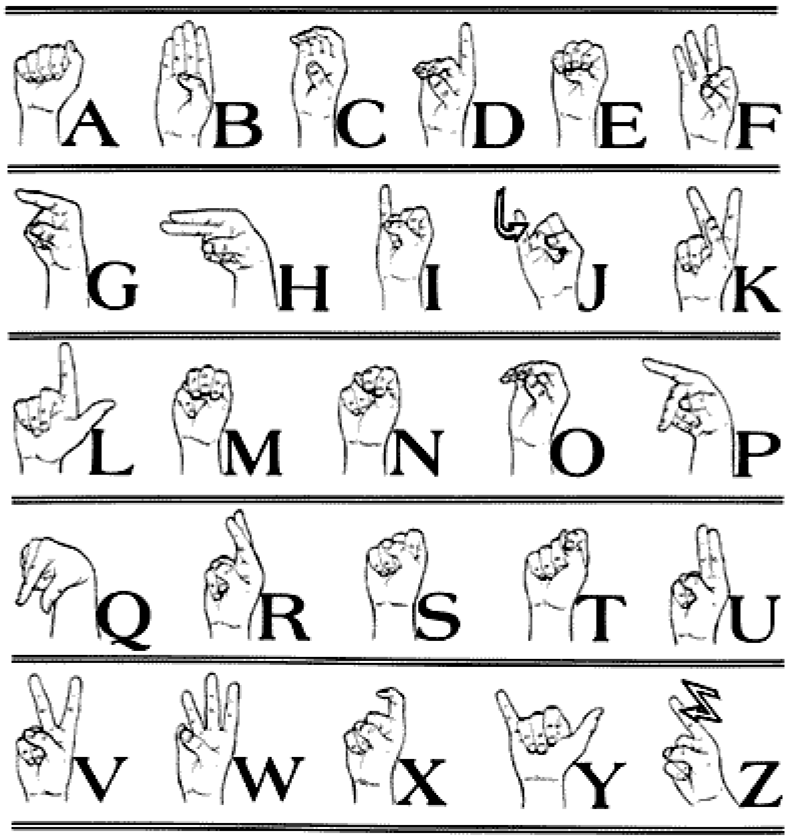
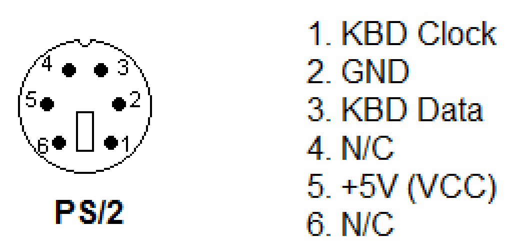


Figure 1: American Manual Alphabet
(http://jellyfishenglish.files.wordpress.com/2007/05/signalpha-web.gif)

Keyboard Interface

The standard interface for keyboard to PC involves codes with header information. It is a bi-directional communications interface between the PC’s keyboard Bios and the keyboard [2]. The two are connected through a 6-pin wire [3]. Figure 2 below shows the pins and their values.



1. KBD Clock
2. GND
3. KBD Data
4. N/C
5. +5V (VCC)
6. N/C

Figure 2: PS/2 wiring
(http://www.beyondlogic.org/keyboard/keybrd.htm)

The keyboard interface is a synchronous interface. A clock signal is sent from the keyboard to the computer to synchronize all communications. Normally the clock line idles high until a signal wants to be sent. When the keyboard is ready to send something, it starts the clock cycle and outputs a start bit of low on the data line. Then, it sends the code or keyboard command it wants sent as an 8 bit binary code. At the end, a parity bit is attached and a high stop bit is appended. Figure 3 below shows the communications for the keyboard to computer.

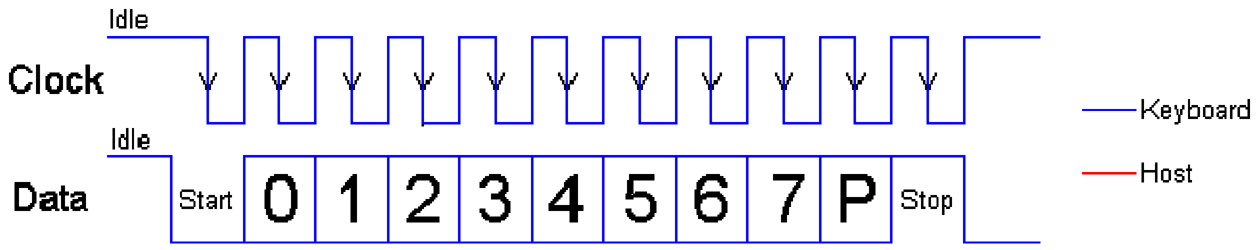


Figure 3: Keyboard to computer waveform
(http://www.beyondlogic.org/keyboard/keybrd.htm)

Every button on the keyboard has its own code. Figure 4 below shows all the keys on a normal keyboard and all the codes associated with each key.

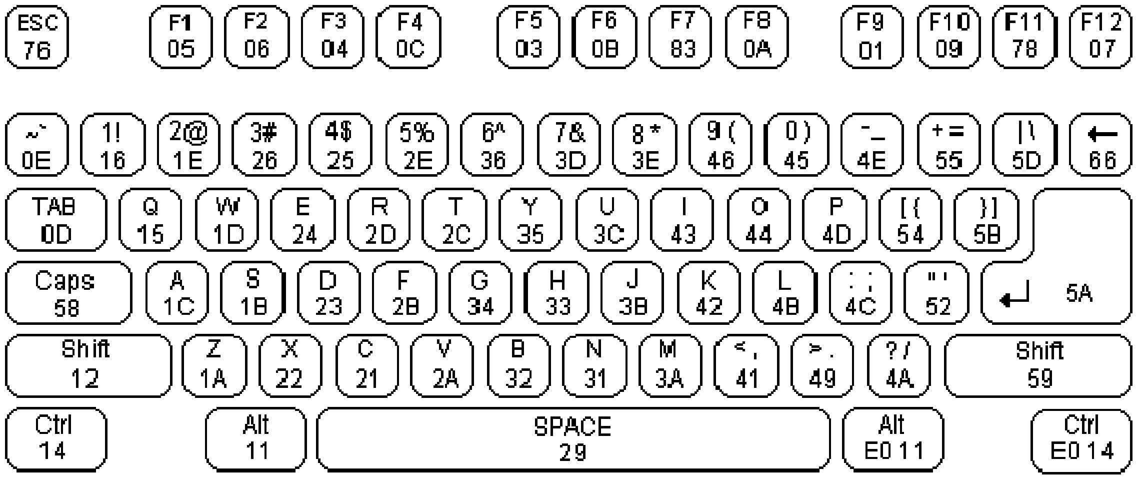


Figure 4: Scan codes for keyboard keys
(http://www.beyondlogic.org/keyboard/keybrd.htm)

Requirements

High Level Requirements:

The glove shall take in standard American Manual Alphabet as an input.

The processor shall translate these hand gestures into digital signals.

The processor shall output these digital signals to the computer into any word processing program.

Detailed Requirements:

The device shall use a FPGA for processing the input signals.

The FPGA shall have Input and Output built into itself, including LEDs and buttons, to allow debugging.

The code for this project shall be VHDL Mixed.

The device shall use the PS/2 port connection to communicate with the computer.

The device shall use the PS/2 protocol to communicate with the computer.

Design

Finger Sensors

There are many different types and brands of sensors on the market. For this project, many different types of bend sensors were researched. There were 3 main types of bend sensors looked at for this project, 2 force sensors and 1 optical sensor.

The first type of sensors looked at were force sensors. Force sensors are force sensitive resistors, resistors that change their resistance relative to applied pressure. The bending of fingers creates pressure at the bend points. This pressure pushes two conductive plates together which lowers the overall resistance. With a voltage divider, this change in resistance can be translated to a change in voltage and can be measured.

The first kind of force sensor was a metal plate sensor. It is constructed by layering two conductive layers of metal with a layer of resistive material. Usually another layer is also placed in the sensor to make the sensor 'bounce back' to its original position. This whole group of materials is then encapsulated in a protective material to protect the sensor and keep it together. Figure 5 below shows the materials and how the sensor is layered. For more information on this type of sensor, visit http://www.imagesco.com/articles/flex/sensor-pg1.html.

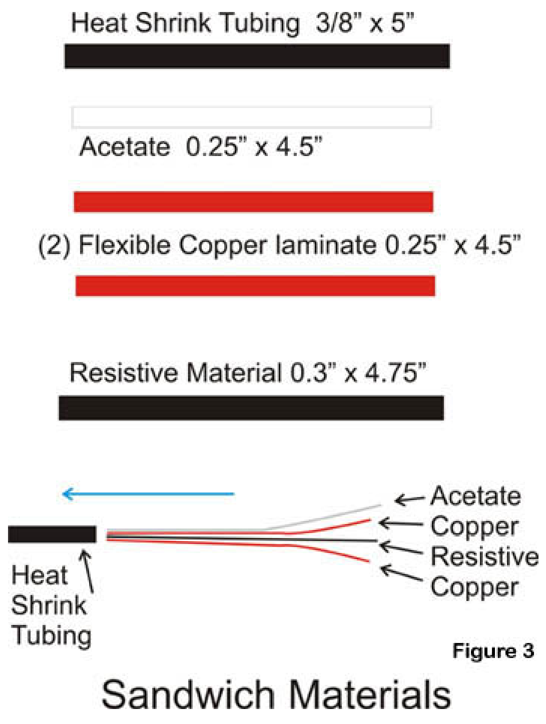

Heat Shrink Tubing 3/8" x 5"
Acetate 0.25" x 4.5"
(2) Flexible Copper laminate 0.25" x 4.5"
Resistive Material 0.3" x 4.75"
Acetate
Copper
Resistive
Copper
Heat Shrink Tubing
Figure 3
Sandwich Materials

Figure 5:Metal plate force sensor [4]

The second type of sensor looked at was the fabric sensor. Fabric sensors are another type of force sensor, made of a strong cloth with conductive thread and a resistive material. Conductive thread is sewn onto two pieces of cloth with approximately 0.6-inch spaces between each stitch. These two sheets of stitched material are placed facing each other with the resistive material in between. As with the metal plate sensor, applying pressure pushes the two conductive sections together, which lowers the overall resistance. Figure 6 below shows the stitching and the order that the sensor is put together.

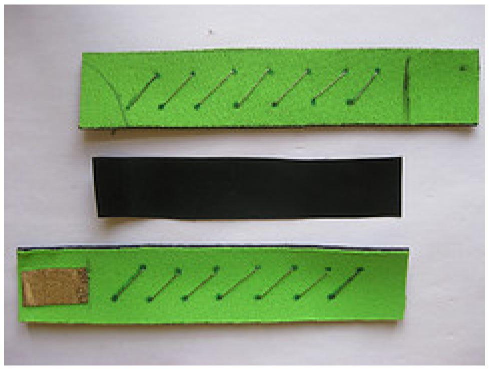


Figure 6: Fabric sensor materials and form [1]

The last kind of flex sensor looked at was an optical flex sensor. An led is shown down a tube towards a detector. As the finger is bent, the light is blocked proportional to the amount of bend. The idea for this design came from http://mellottsvrpage.com/VirtualInstrument/VirtualInstrumentGloveDevice.doc.

The metal plate force sensor was chosen because it had the highest reliability of all the sensors while still being simplistic. It was originally thought to build the sensors myself, but due to time constraints, the sensors were bought.

Processor

It was realized early on that a form of development board would be needed for this project. It was decided that we would use a Nexys 2 prototyping board. It was picked because of prior experience with the board we already had one available. The two options for development were either running an embedded processor with C programming or VHDL. VHDL was chosen because of previous experience and because it seemed that having a lower level, but more powerful language would aid in the development of the project.

Input format

There were really only two choices for input format, either the PS/2 port or USB. PS/2 is the standard keyboard interface. The pros for this format are the ease of interfacing. Once this format was set up, any program that uses the keyboard would be able to interact with the glove. However, with the PS/2 port, all the processing would have to be done before entering the computer. The USB would have allowed a more dynamic interface, using all pre-processing, no pre-processing, or a mixture of the two. It also had the advantage of easier diagnostics of design and coding (since the computer would be doing some or all of the translation from voltage to character). The PS/2 format was eventually chosen primarily due to its ease of interfacing. Once set up, any computer with a PS/2 port could use the glove.

Overall Design

Figure 7 below shows the over all design decided upon for the system.

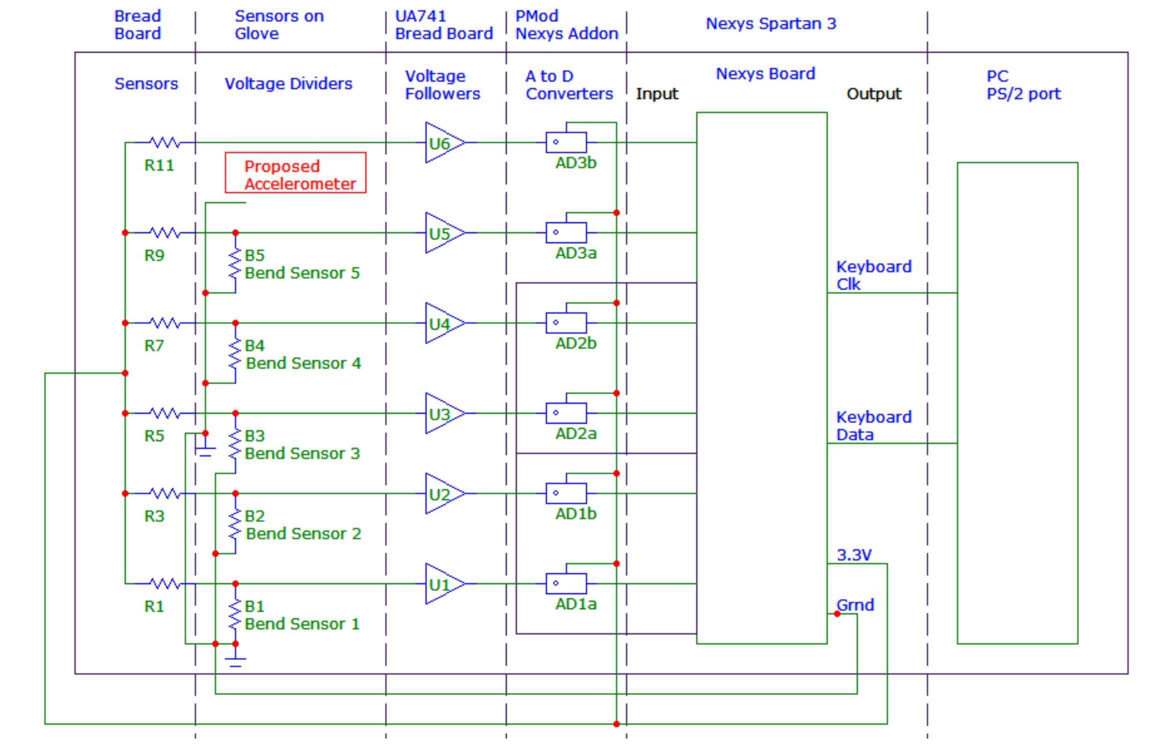


Figure 7: System Design

As you can see from the figure above, the system is composed of 6 basic parts. The sensors change their resistance with bend of fingers. To translate this change of resistance into a change of voltage, voltage dividers are used with a 3.3V high down to ground. The ground and Voltage high are provided by the Nexys Board. Next, voltage followers are used to keep the Nexys board resistance separate from the voltage dividers. Next, A to D converters convert the 3.3 V down to 0 V analog signal into a digital signal ranging from 4096 down to 0. Next, the Nexys board takes in this digital signal and processes this into a usable character output. The code then uses a state machine to output the transmission code, which includes the character code, to the PC.

Construction

Construction of the glove device involved both hardware and software. A two-tiered design process was decided upon. First, the Nexys board to PC interface would be researched, designed, and built. Then, the glove device itself would be built and the interface between it and the Nexys board would be made. Finally, the whole device would be integrated together to work as a whole.

Nexys to computer

First, the interface had to be understood. An oscilloscope was borrowed from the EE department to aid in this task and an old keyboard was taken apart and its innards used as a test dummy. Probes were attached to the keyboard clock and data lines and the output was observed. A single letter (in this case, f) was chosen to replicate its behavior and understand the interface. A state machine was created in VHDL to replicate the behavior of the keyboard. Figure 8 below shows an example of the analysis being done. (Note: for this particular output, I was off by a bit. The final design had all the bits proper)

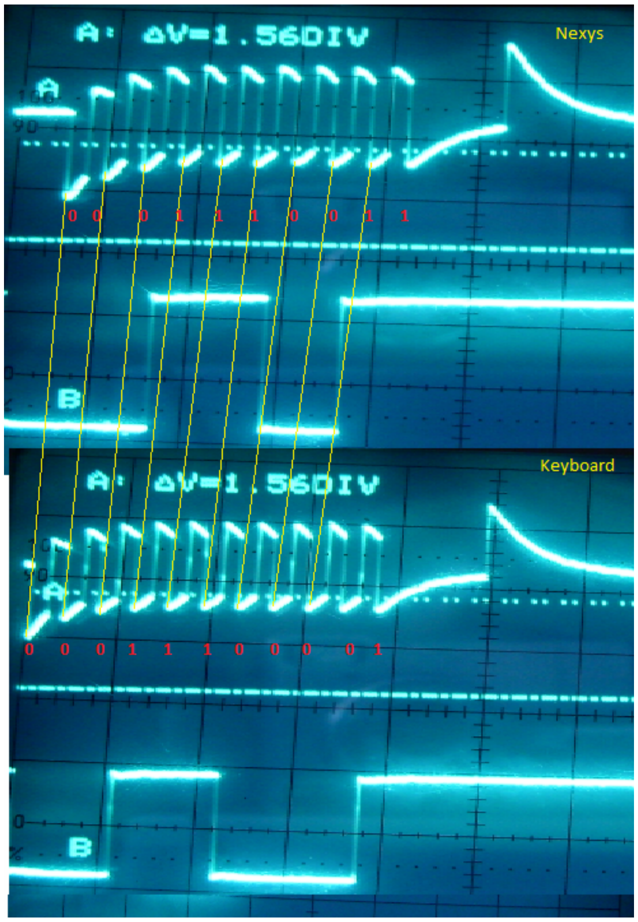


Figure 8: Comparison of keyboard output and Nexys output for letter f

Appendix D contains the complete code for the state machine used.

Glove

Figure 9 below shows the glove constructed for this project.

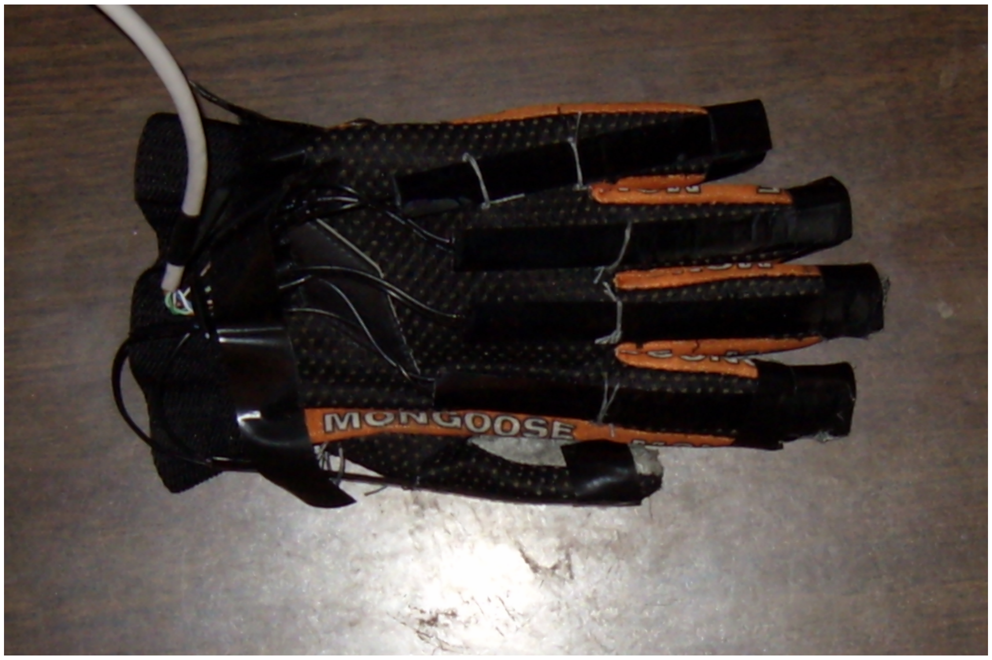


Figure 9: Finished Glove

The glove consists of a modified biking glove (a glove found lying around), 5 sensors, some Cat-5 cable, electrical tape, and thread. Since the glove was a biking glove, it had some padding on the fingers to help protect against a fall. First, these pads were removed because they impeded the movement of the fingers. Second, the sensors were taped to the end of the fingers to keep them in place. Then, the sensors were attached to the wires in the Cat-5 cable. Cat-5 cable has 8 wires. Five of the wires attached directly to sensors. Two of the wires used as grounds for the sensors, each of the grounds sharing 3 sensors. The last wire was set aside of a proposed accelerometer to aid in the detection of movement for certain letters such as j and z as well as detect the orientation of the glove for detecting similar signs such as determining the difference between h and u. Originally, the wires were merely bent such that they would stay in contact, but they occasionally fell off. Eventually, they were securely soldered together.

Next, the Cat-5 wire was taped to the glove by applying electrical tape around the wrist section of the glove. This provided a malleable but strong attachment that kept the wires to the sensors from having too much strain. Next, the lower part of the sensors was attached to the glove so that movements were repeatable. At first, this was done with electrical tape that had tape taped to the sticky side of the tape as well. This allowed the sensors to slide against the tape. This did not prove to work too well due to the high coefficient of friction, however, so a new set up was devised. Loops of thread were sewn into the glove with the sensors within the loops. This allowed the sensors to slide against the thread with minimal resistance to movement and kept the sensors relatively flush with the glove.

Voltage Dividers

Since the Nexys board ADCs use voltage inputs, the resistance change in the sensors had to transformed into a voltage change. This was done using five voltage dividers, one per sensors. A simple voltage divider is shown below in Figure 10.

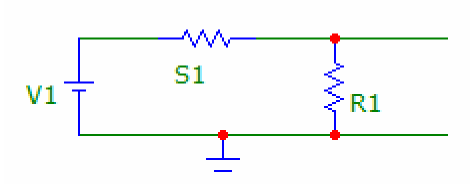


Figure 10: Simple Voltage Divider

When the sensors were first received, the resistance change was measured. Each sensor varied between approximately 10k ohms down to 100 ohms (S1). With these values, resistors of 10k were chosen for the voltage divider (R1). However, over time, the sensors changed to ranging from 200kohms down to 100kohms. With this value, some of the resistors were changed to 100kohms.

Voltage Followers

Initially, the design did not call for voltage followers. However, after initial use, it was realized that the board configuration was changing the input resistance, which changed what the ADC was detecting and would vary with each running of the program. Because of this, Voltage followers made of op amps were used to separate the resistance from each side of the project. Five UA741CN were used for the op amps. Since the op amp has internal losses, secondary power source of higher voltage was needed to power the op amps so that a full voltage swing was achievable. Four AA rechargeable batteries were placed in series to achieve this at 4.8V. Figure 11 below shows a voltage follower.

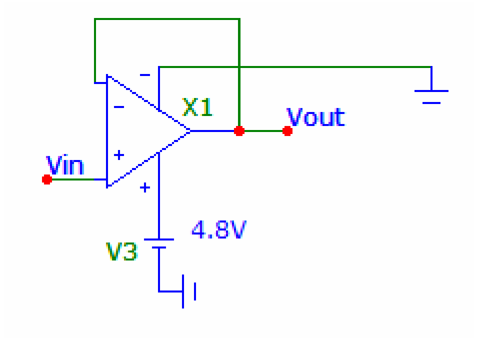


Figure 11: Voltage Follower

Code for ADC integration

After the glove was built, the Analog to Digital Converters (ADCs) had to be integrated into the design. The ADCs chosen were the PmodAD1, two 8-bit A/D converters built onto one add-on to the Nexys board. They were chosen because they were by the same company that makes the Nexys board and would thus be easily integrated into the design. VHDL code was found online at the Digilent website for interfacing with their ADC with their board, and the code is found in Appendix D.

Code for Glove input

There were two trains of thought for how to do glove input and interpretation. The first and simpler thought was to set voltage levels for each input and when it was between these voltages, have it say it is a certain value. This is known as parallelpipe classification. Figure 12 below shows how this would work for two inputs.

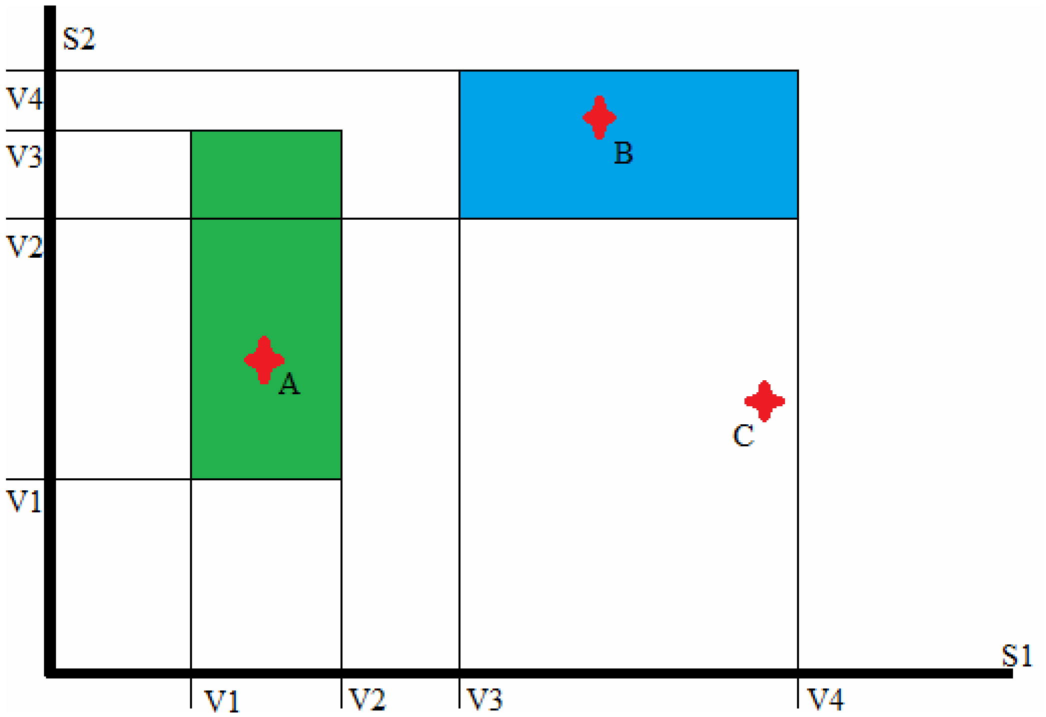


Figure 12: Parallelpipe Classification

As seen in Figure 12 above, imagine 3 inputs are recorded. The first input, A, fell between V1 and V2 for S1 and between V1 and V3 for S2, so it must be of class green. Input B fell between V3 and V4 for S1 and between V2 and V4 for S2, so it must be class blue. Input C, however, did not meet both requirements, so it is neither class green or blue. This type of analysis can be expanded for as many inputs as required.

The other kind of classifier would be a Euclidean Distance Classifier. Figure 13 below shows how this sort of classifier would work.

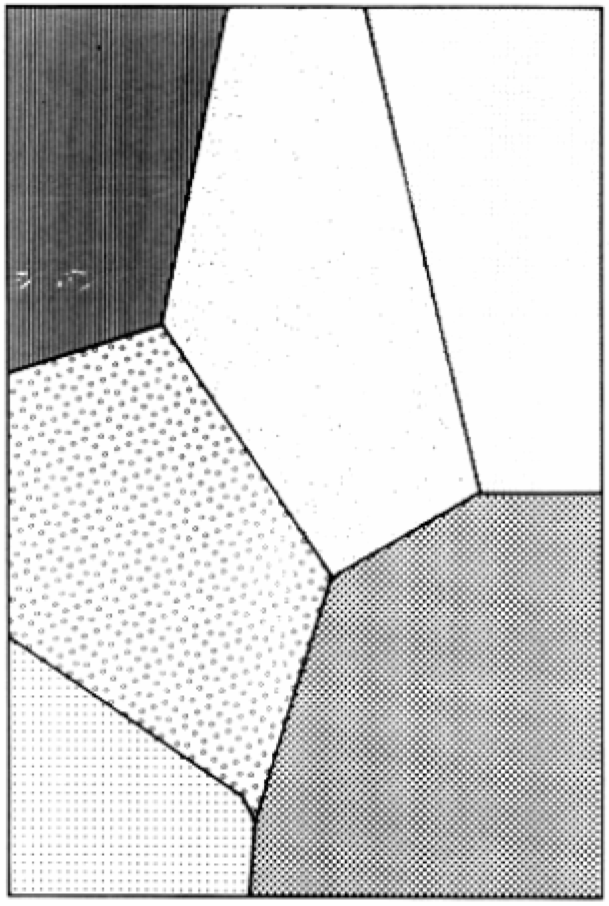


Figure 13: Euclidean Distance Classifier

A Euclidean Distance Classifier consists of centroids that stand for the desired class. When a value is inputted, the distance from each of the centroids is calculated and the point is classified into whatever it is closest to. This scheme can be made more stable by limiting the distance a value can be from the centroid, essentially creating circles of allowed values around the centroids. This would make it so extreme values were not counted in a scheme. Also, this would make it so the glove would not always be outputting a value. This could be made more efficient by only checking certain centroids if one of the input values is above or below a set voltage. For example, if the index finger is above 3 volts (aka it is straight), only check centroids where the finger is straight.

Both classifiers were implemented, but it turned out that the Euclidean Distance Classifier took up too much room due to the board not being able to implement multiplications and squaring efficiently [5]. Eventually, the parallelpipe classifier was implemented fully and its code can be found in Appendix D, in the concurrent1 process.

Final Integration

The final interfacing of everything together went pretty smoothly. It took a lot of copying and pasting of code from multiple programs due to the way the problem was approached. The finished hardware setup is shown below in Figure 14. The finished code is in Appendix D.

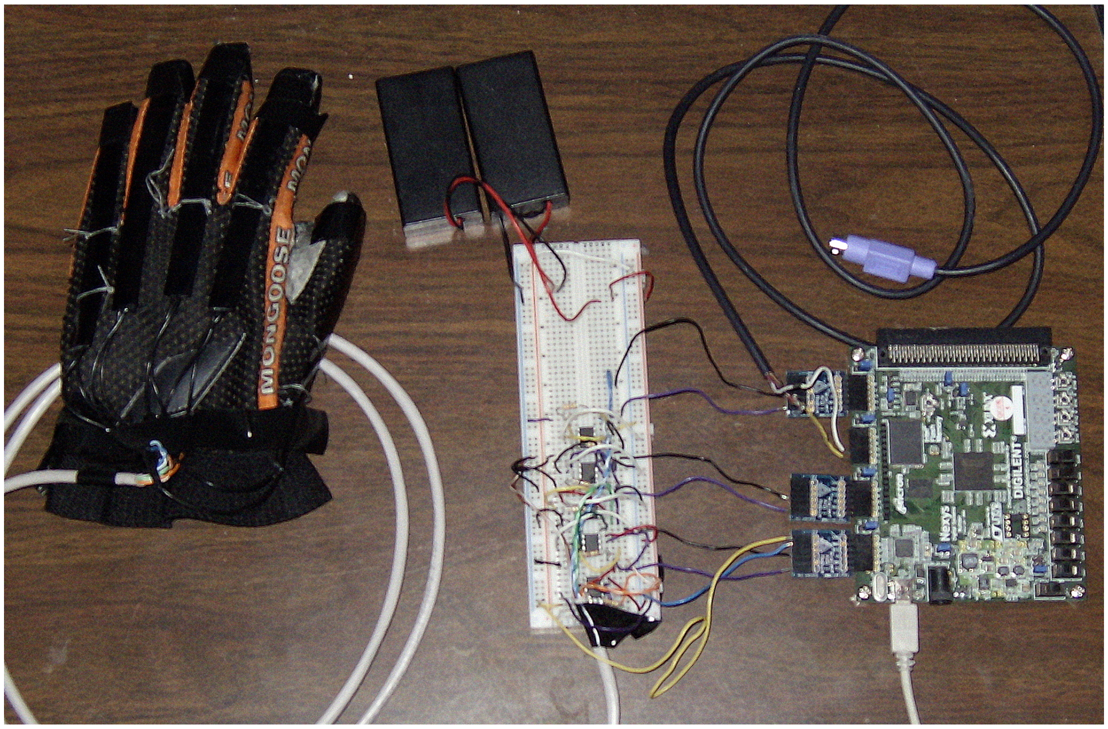


Figure 14: Finished Hardware Setup

Testing

A number of test schemes were created for this project since the output types were so limited. The only available outputs were the Nexys board's LEDs and 7-segment display and the computer's characters.

Since the first part of the circuit built was the Nexys input to the computer, this was the first part tested. An oscilloscope was used to look at the signals between the two. When the project began, two identical keyboards were acquired. This allowed one to be opened up and messed around with while the other would be in working condition for comparing IO signals. The signals between the Nexys board and PC were compared to those of the keyboard and PC (as shown in Figure 7). This allowed the code used for the IO to be directly compared quickly and easily.

To output characters, a scheme was created using the Nexys board's built in buttons and switches. Having only switch one on would allow A, B, C, and D to be output. Throwing switch two would allow E, F, G, H to be output. This was set up for all the switches, including spaces and enters after all the letters of the alphabet were set up and working properly. To test whether the buttons were working properly, the LEDs of the board were setup to shine the character code of the letter it thought it was outputting. Later on, the 7-seg display was setup to display the character it was outputting. Appendix D shows the codes used for the alphabetical 7-segment output.

One of the earliest problems identified in the IO was the order of the character code. It was assumed that the character codes would be First In, First Out (FIFO). However, when the first outputs were working, most of the letters would not work. One of the few letters that did work was E. The character code for E is ‘24’. In binary, this is ‘00100100’, which is the same backwards and forwards. Once this was figured out, it was easy to test this theory by pressing T and getting an output of a G (as they are mirror images of each other). The code was changed to accommodate this and the output worked perfectly.

Once the output was working, work was begun on the input stages. The glove was built and the voltage dividers and followers put in place. To ensure the input was working properly, a scheme for testing the input voltage was used to measure the response. A multimeter was used to measure the actual voltage change on the sensor. Code was written to have the Nexys board LEDs shine differently when different input voltages were measured. The shine sequence was a simple bar graph meter, where, at the highest voltage, all the LEDs would be on, and at the lowest voltage, none of the LEDs would be on. Switches were used to control which sensor was being measured.

A more sophisticated scheme was created to allow quick set up of the voltage levels for a classifier scheme. The 7- segment display was set up to display the output of the ADC of the sensor being measured. Different sensors could be selected by use of the switches. This scheme was difficult to set up do to the lack of experience with seven segment displays. This approach was eventually abandoned due to the lack of the ability to divide by 10s. The output of the ADC was a binary number. This was easily convertible to an integer using VHDL commands. This number would be of the form X,XXX (e.g. 4,021). However, to convert this into 4 integers (or binary numbers), each containing the nth number of the (the first being 4, next being 0, third being 2, and the last being 1), one would need to be able to divide by 1000, 100, and 10. Since the Nexys board is inherently binary, it was only able to divide by factors of 2. One idea to get around this was to multiply by a number and then divide by a number that is a multiple of 2 to get around 10, but the number calculated was not close enough for classification purposes. This scheme was eventually abandoned. The classification voltage levels were done by measuring the input voltages and calculating the binary equivalent.

Finally, the classifier was created and upper and lower voltage levels were measured and recorded. These were converted into binary ranging from 0 up to 4096. Due to the voltage followers being powered from the same ground as the sensors themselves, the voltage inputs were limited on the lower edge. This meant the dynamic range of the sensors was limited, but also created some stability, making lower voltages be the same.

A scheme was created to visualize the inputs and compare the different voltage inputs for the letters. Figure 15 below shows the scheme used.

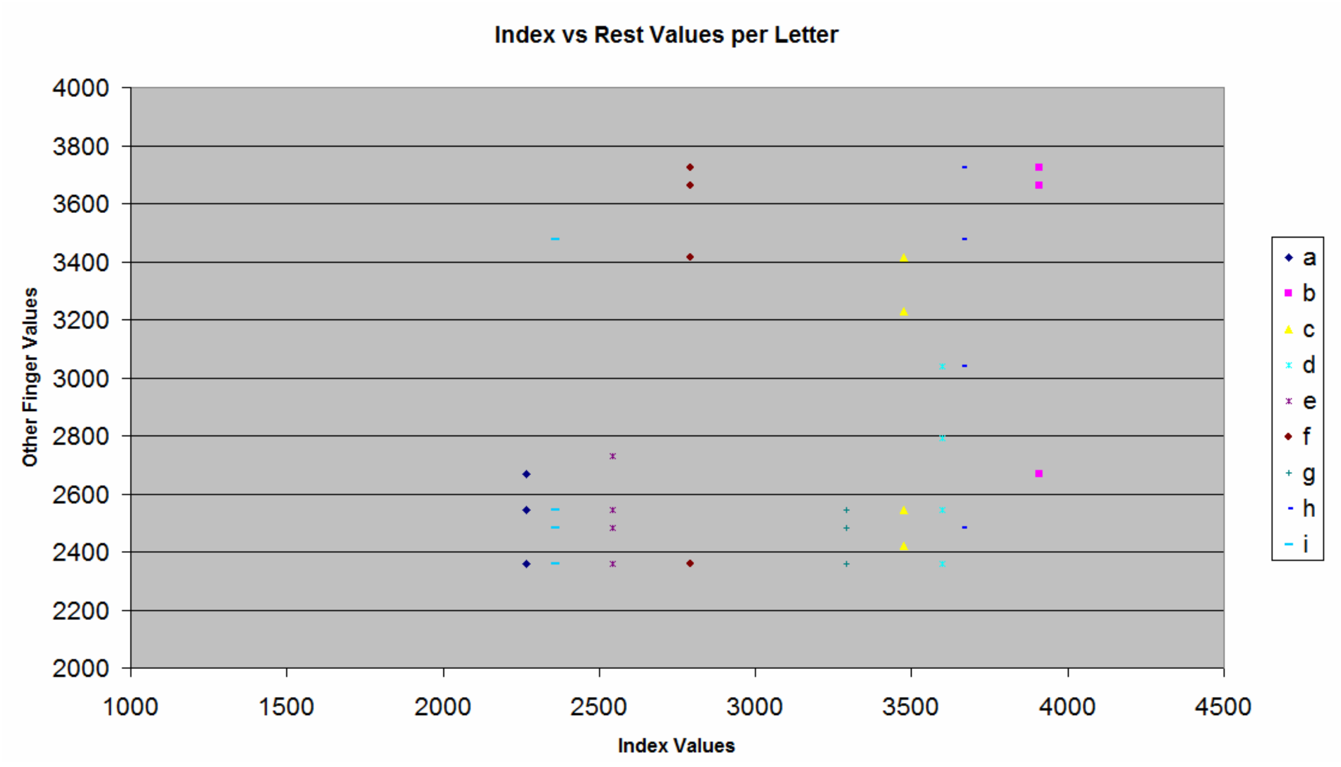


Figure 15: Visualization scheme for inputs

The figure above graphs the index finger's voltage level (in binary) to those of the other fingers, per character output. For example, the index finger value of A is around 2250, so the x-value is around 2250. The other 4 fingers are all very closely spaced, since most all of the fingers of the letter A are closed. The letter I has a very similar characteristic, since most of the fingers are closed. However, one very large difference is the pinky finger, which is way higher than any of the other fingers, as seen in the graph above. This sort of visualization can easily show where problems might occur in distinguishing some characters from others.

In the end, only the letter A through I were functional and letters E, G, and H were so similar that they caused debounce issues and caused the computer to cease to function for a short amount of time. More letters could have easily been made functional by measuring their threshold voltages and adding them to the VHDL program.

Conclusions and Recommendations

This project was a success overall. It was long and difficult, but well worth it. It covered many different topics, ideas, and areas of engineering. This project involved many changes along the way to accommodate problems found in the original design. Both software and hardware were used to show the viability of a product.

One way this project could have been improved was keyboard initialization. Currently, when the computer boots up, the keyboard is somehow initialized by the computer and are able to talk to each other. I could find no documentation online about what this initialization is, how long it takes, how to do it, so it was not replicated in the Nexys board. This meant that if the Nexys board wanted to be used to connect to the computer via the PS/2 port, the computer had to have been booted up with another keyboard in the PS/2 port already and the two are swapped while the computer is on (something that is not suggested by everyone).

Another way this could have been improved is to program in C. With C, different options for classification might have been available. The program would have most likely been less efficient in space, but more easily changeable and easier to interface overall. On a similar note, if the USB had been used rather than the PS/2 port, integration would have been easier as there would have been more options for viewing inputs. This would have, however, meant inventing a new interface, which has its things.

If I would to continue this project, I would change the sensors. They had a couple of problems. For one thing, they began to fall apart slightly. The design of the sensor calls for the ends of the shrink tubing they are encased in to be closed up using some sort of sealant or caulk. This material began to loosen and eventually the sensor innards were moving around, which changed the bend characteristic of the sensor drastically. Eventually, I bought some super glue to glue the sensors back together, and they have held out very well since then.

Also, after a certain amount of use, their bend characteristic seems to change permanently. When shipped, they are around 10k optimal resistance which shrinks when bent to around 100 ohms. After use, their resistance will change very little until a certain amount of use is reached, at which point they change drastically. Instead of a 10k optimal resistance, they have around 200k optimal which drops to around 50k ohms. This means that the resistors in the voltage dividers have to be changed and the overall sensitivity of the sensor is decreased.

Another problem was the sensor settling time. When the sensor was bent, it took around 15 seconds for the voltage drop to settle in the voltage divider. This is due to some sort of hysteresis inherent in the sensors. This would not be so bad except this voltage drop is around .2 volts, and when the overall voltage change is only around 2 volts, that is around 10% of the whole voltage range. For some letters, this small change made a big difference. For example, C and D were very similar, and so when held at a single location, the output would change over time from D to C even though the hand had not moved at all. This effect could be handled relatively by more complex software (i.e. software that knew how long a letter had been held and adjusted the voltage levels for classification accordingly).

Another improvement would be in the output debounce. In VHDL, it is difficult to have synchronous signals interact with asynchronous signals. If a different language was used or a more experienced VHDL programmer wrote the code, the debounce for the hand characters would be much more fluid. In the end, some letters that were close to others would cause multiple outputs at once, which caused the computer to be unresponsive for a little while. Better debounce would prevent this.

Also, a better attachment for wire that attaches to the glove would be beneficial. The current scheme, taping the wire directly to the back of the glove, did not work the best and caused the sensors wires to be bent, which affected their response to movements. Shortening the wires from the sensors to the cable to the Nexys board would also improve this.

Lastly, the scheme used for visualizing the hand gestures in a two dimensional graph could be improved by adding error. Currently, the graph merely looks at the average of the upper and lower limit of the voltages for the classifier. The version of excel I was using did not allow me to add different amounts of error for different points of the graph. If this could be implemented, it would allow the user to glance at the graph and see where characters that are very similar actually overlap in values and those instances could be planned for and avoided more easily.

Bibliography

1. Fabric Sensor
http://www.kobakant.at/DIY/?p=20

2. Keyboard interface
http://www.beyondlogic.org/keyboard/keybrd.htm

3. Keyboard Interface 2
http://www.computer-engineering.org/ps2protocol/

4. Flex Sensors
http://www.imagesco.com/sensors/flex-sensor.html

5. Square root in VHDL for Euclidean Distance Classifier
http://vhdlguru.blogspot.com/2010/03/vhdl-function-for-finding-square-root.html

6. Sign Language Gestures
http://www.lifeprint.com/asl101/pages-layout/handshapes.htm

Appendices

Appendix A: FPGA information

From the Spartan 3 reference manual:

> Digilent’s Nexys circuit board is an integrated circuit development platform based on a Xilinx Spartan 3 FPGA. The Nexys board provides large external memory arrays, a collection of useful I/O devices, and numerous ports, making it an ideal platform for experiments with FPGA-based digital systems, including embedded cores like Xilinx’s MicroBlaze.

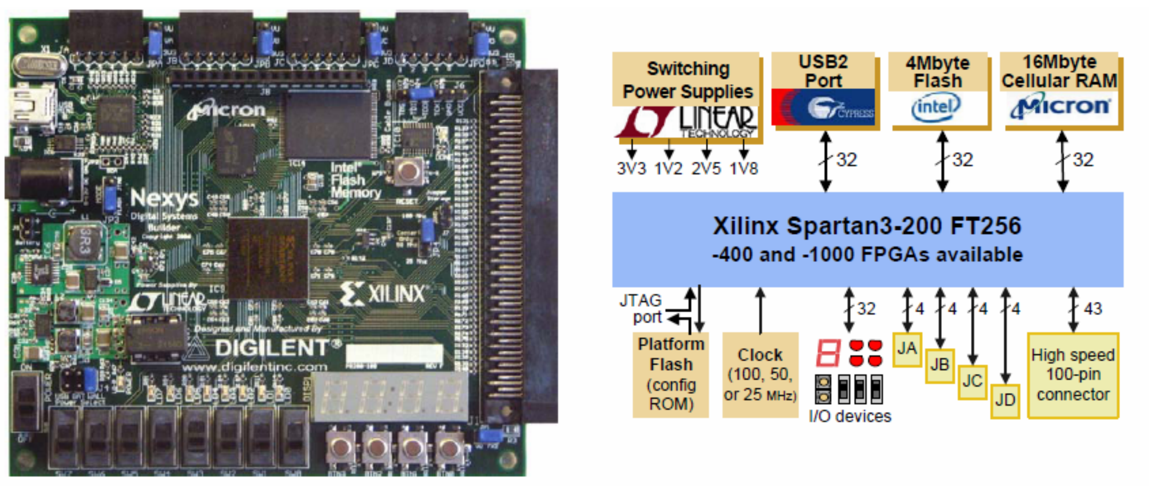

Switching Power Supplies
USB2 Port
4Mbyte Flash
16Mbyte Cellular RAM
3V3 1V2 2V5 1V8
32
32
32
Xilinx Spartan3-200 FT256
-400 and -1000 FPGAs available
JTAG port
32
4
4
4
4
43
Platform Flash (config ROM)
Clock (100, 50, or 25 MHz)
I/O devices
JA
JB
JC
JD
High speed 100-pin connector

Figure 16:Nexys Board and Nexys Board IO

Table 1:Nexys Board IO for this project

IO	Format	Nexys ID	Nexys PIN	From	To	Variable	Color
CS	-	JA-1	T14	Hand	Nexys	CSA	
Hand Thumb	A-D	JA-2	R13	Hand	Nexys	Hnd_thmb	Orange
Hand Index	A-D	JA-3	T13	Hand	Nexys	Hnd_index	white orange
Clk	D	JA-4	R12	Hand	Nexys	IO_clk	
GND	-	JA-5	GND	Hand	Nexys	-	Br, WBr
3.3 V	-	JA-6	V	Hand	Nexys	-	
CS	-	JB-1	T12	Hand	Nexys	CSB	
Hand Middle	A-D	JB-2	R11	Hand	Nexys	Hnd_mid	Green
Hand Ring	A-D	JB-3	P8	Hand	Nexys	Hnd_ring	WGr
Clk	D	JB-4	T10	Hand	Nexys	IO_clk	
GND	-	JB-5	GND	Hand	Nexys	-	Br, WBr
3.3 V	-	JB-6	V	Hand	Nexys	-	
CLK	D	JC-1	D5	Nexys	Comp	kbd_clk	
Char code	D-serial	JC-2	P9	Nexys	Comp	kbd_data	
-	-	JC-3	A5	Nexys	Comp	-	
-	-	JC-4	A7	Nexys	Comp	-	
GND	D	JC-5	GND	Nexys	Comp	-	
-	-	JC-6	V	Nexys	Comp	-	
CS	-	JD-1	A9	Hand	Nexys	CSD	
Hand Little	A-D	JD-2	A12	Hand	Nexys	Hnd_ltle	Blue
Accelorameter	A-D	JD-3	C10	Hand	Nexys	Hnd_acc	WBl
Clk	D	JD-4	D12	Hand	Nexys	IO_clk	
GND	-	JD-5	GND	Hand	Nexys	-	Br, WBr
3.3 V	-	JD-6	V	Hand	Nexys	-	

Useful information and Links:
Board: Xilinx Spartan3-200 FT256
Development Environment used: Xilinx ISE Design Suite 11
Device Programmer: Digilent JTAG Programmer
Reference Manual: http://www.digilentinc.com/Data/Products/NEXYS/Nexys_rm.pdf

Appendix B: Parts List, Cost, and Time Schedule Allocation

Cost Allocation:

Five biflex sensors:	$50.00
Five 10k resistors:	$1
Five UA731 op amps:	$1
Three Pmod-DA2, Two 8-bit D/A outputs:	$60.00
One Nexys Spartan 3 Board:	$125
Cat-5 Cable:	$10
Total Cost:	**$247**

Time Allocation:

Nexys to Computer Interface:	25 hours
Glove Construction:	5 hours
Glove Interfacting with Nexys:	20 hours
Classifier:	25 hours
System Integration:	15 hours
Total Time:	**90 hours**

Appendix C: Diagrams of System

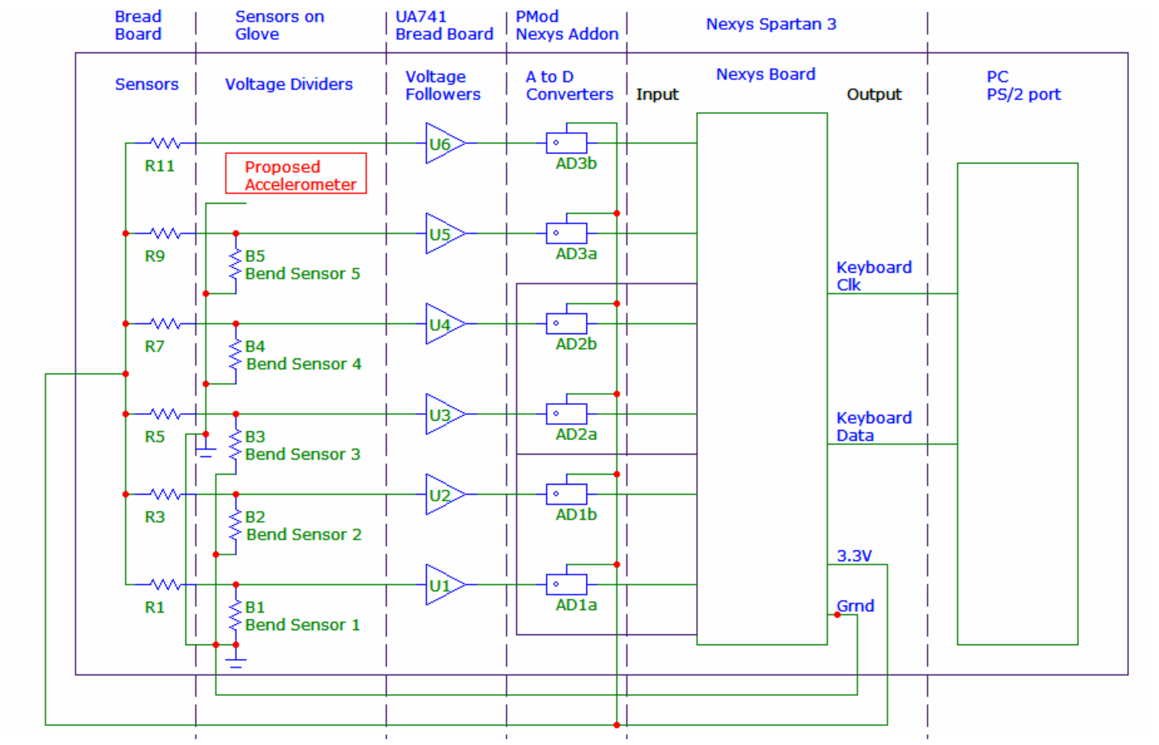
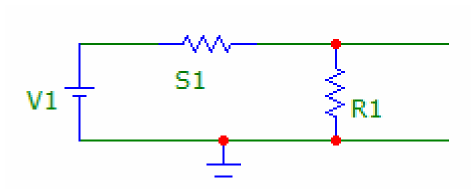


Figure 7: System Design



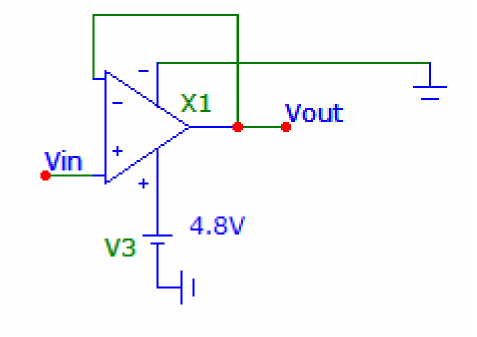


Figure 10: Simple Voltage Divider

Figure 11: Voltage Follower

Pmod Link:
http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,401,499&Prod=PMOD-AD1
Sensor Link: http://www.imagesco.com/sensors/flex-sensor.html
Nexys Link: See Appendix A

Appendix D: Final VHDL Code

```
library IEEE;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity test1 is
  Port (
                         CLK  : in  STD_LOGIC;
                         SW   : in  std_logic_vector(0 to 7);
                         BTN0 : in  STD_LOGIC;
                         BTN1 : in  STD_LOGIC;
                         BTN2 : in  STD_LOGIC;
                         BTN3 : in  STD_LOGIC;
                         LED_out  : out std_logic_vector(7 downto 0);
                         D7S  : out std_logic_vector(0 to 7);
                         A    : out std_logic_vector(0 to 3);
                         kbd_clk   : inout std_logic;
                         kbd_data  : inout std_logic;

                         SDATA1   : in std_logic;
                         SDATA2   : in std_logic;
                         SDATB1   : in std_logic;
                         SDATB2   : in std_logic;
                         SDATD1   : in std_logic;
                         SDATD2   : in std_logic;
                         SCLKA    : out std_logic;
                         SCLKB    : out std_logic;
                         SCLKD    : out std_logic;
                         nCSA     : out std_logic;
                         nCSB     : out std_logic;
                         nCSD     : out std_logic
);

end test1;

architecture Mixed of test1 is

type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13);
signal state_n, state_c: state_type;
--type debounce_state is (d0 d1 d2 d3);
--signal d_n, d_c: debounce_state;
signal slow_clk: std_logic_vector (24 downto 0);
signal count2: std_logic_vector (15 downto 0);
signal half_sec, k25hz, transmit, slow_clk2: std_logic;
signal khz_clk: std_logic_vector (11 downto 0);
signal letter_d, letter_a, letter_ones, letter_tens, letter_hund, letter_thou: std_logic_vector(0 to 7):= not "00000000";
signal LED : std_logic_vector(0 to 7):= "00000000";
signal output1, output2, output3, output4, output5, output6, output7: std_logic_vector(0 to 10);
signal output_p : std_logic;
signal input: std_logic_vector(0 to 8);
signal i, j, k, l: integer;
signal parity_int: integer;
signal parity_bit: std_logic;
signal dBTN0, dBTN1, dBTN2, dBTN3: std_logic_vector(5 downto 0);
signal pBTN0, pBTN1, pBTN2, pBTN3: std_logic;

type states is (Idle,
        ShiftIn,
        SyncData);
signal current_state : states;
signal next_state    : states;

signal temp1        : std_logic_vector(15 downto 0);
signal temp2        : std_logic_vector(15 downto 0);
signal temp3        : std_logic_vector(15 downto 0);
signal temp4        : std_logic_vector(15 downto 0);
signal temp5        : std_logic_vector(15 downto 0);
signal temp6        : std_logic_vector(15 downto 0);
signal clk_div      : std_logic;
signal clk_counter   : std_logic_vector(1 downto 0);
signal shiftCounter  : std_logic_vector(3 downto 0) := x"0";
signal clk_25k_count : std_logic_vector(11 downto 0);
signal clk_25khz                  : std_logic;
signal enShiftCounter: std_logic;
signal enParalelLoad : std_logic;
signal DATA1                             : std_logic_vector(11 downto 0);
signal DATA2                       : std_logic_vector(11 downto 0);
signal DATB1                             : std_logic_vector(11 downto 0);
signal DATB2                       : std_logic_vector(11 downto 0);
signal DATD1                             : std_logic_vector(11 downto 0);
signal DATD2                       : std_logic_vector(11 downto 0);
signal int_data1                  : integer;
signal int_data2                  : integer;
signal int_datb1                  : integer;
signal int_datb2                  : integer;
signal int_datd1                  : integer;
signal int_datd2                  : integer;
signal START                       : std_logic;
signal DONE                        : std_logic;
signal led_data1                  : std_logic_vector(7 downto 0):= "00000000";
signal int_to_d7s                 : integer;
signal bin_dig0                   : std_logic_vector(7 downto 0);
signal digit0                            : integer;
signal digit1                            : integer;
signal digit2                            : integer;
signal digit3                            : integer;
signal REM0                                      : integer;
signal REM1                                      : integer;
signal D7s_thou                   : std_logic_vector(7 downto 0);
signal D7s_hund                   : std_logic_vector(7 downto 0);
signal D7S_tens                   : std_logic_vector(7 downto 0);
signal D7s_ones                   : std_logic_vector(7 downto 0);
type D7s_states is   (first,
                                                        second,
                                                        third,
                                                        fourth);
signal current_st : D7s_states;
signal next_st    : D7s_states;

signal cur_ind : integer;
signal cur_mid : integer;
signal cur_rng : integer;
signal cur_pnk : integer;
signal cur_tmb : integer;
signal flag : std_logic;
```

```
begin

frequency_divider : process (CLK)
        begin
        if rising_edge (CLK) then khz_clk <= khz_clk +1;
                slow_clk <= slow_clk +1;
                count2 <= count2 +1;
        end if;
k25hz <= khz_clk(11);
SLOW_CLK2 <= COUNT2(15);
half_sec <= slow_clk(24);
--kbd_clk <= half_sec;
--kbd_clk <= k25hz;
--kbd_data <= k25hz;
end process;

BTN_Debounce : process(k25hz, BTN0, BTN1, BTN2, BTN3, half_sec)
begin
        if rising_edge(k25hz) then
                dBTN0(0) <= BTN0;
                dBTN0(1) <= dBTN0(0);
                dBTN0(2) <= dBTN0(1);
                dBTN0(3) <= dBTN0(2);
                dBTN0(4) <= dBTN0(3);
                dBTN0(5) <= dBTN0(4);
        end if;
        pBTN0 <= dBTN0(0) and dBTN0(1) and dBTN0(2) and dBTN0(3) and dBTN0(4) and not dBTN0(5);

        if rising_edge(k25hz) then
                dBTN1(0) <= BTN1;
                dBTN1(1) <= dBTN1(0);
                dBTN1(2) <= dBTN1(1);
                dBTN1(3) <= dBTN1(2);
                dBTN1(4) <= dBTN1(3);
                dBTN1(5) <= dBTN1(4);
        end if;
        pBTN1 <= dBTN1(0) and dBTN1(1) and dBTN1(2) and dBTN1(3) and dBTN1(4) and not dBTN1(5);

        if rising_edge(k25hz) then
                dBTN2(0) <= BTN2;
                dBTN2(1) <= dBTN2(0);
                dBTN2(2) <= dBTN2(1);
                dBTN2(3) <= dBTN2(2);
                dBTN2(4) <= dBTN2(3);
                dBTN2(5) <= dBTN2(4);
        end if;
        pBTN2 <= dBTN2(0) and dBTN2(1) and dBTN2(2) and dBTN2(3) and dBTN2(4) and not dBTN2(5);

        if rising_edge(k25hz) then
                dBTN3(0) <= BTN3;
                dBTN3(1) <= dBTN3(0);
                dBTN3(2) <= dBTN3(1);
                dBTN3(3) <= dBTN3(2);
                dBTN3(4) <= dBTN3(3);
                dBTN3(5) <= dBTN3(4);
        end if;
        pBTN3 <= dBTN3(0) and dBTN3(1) and dBTN3(2) and dBTN3(3) and dBTN3(4) and not dBTN3(5);

end process;
```

```
Hand_debounce : process(k25hz)
begin
LED_out <= LED;
end process;


concurrent1 : process (CLK, k25hz, half_sec, cur_ind, cur_mid, cur_rng, cur_pnk, cur_tmb)
begin
if rising_edge(k25hz) then
        if    cur_ind < 2669 and cur_ind > 2110 --and cur_mid < 2607 and cur_mid > 2482
        and cur_rng < 2731 and cur_rng > 2607 and cur_pnk < 2482 and
        cur_pnk > 2234 and cur_tmb < 2855 and cur_tmb > 2234 then
                letter_d <= not "11101110"; --a
                letter_a <= "00011100"; --            LED <= "00111100";--      else LED <= "00000000";
        elsif cur_ind < 4096 and cur_ind > 3724 and --cur_mid < 4096 and cur_mid > 3227 and
        cur_rng < 3724 and cur_rng > 3600 and cur_pnk < 3972 and
        cur_pnk > 3475 and cur_tmb < 2979 and cur_tmb > 2358 then
                letter_d <= not "00111110"; --b
                letter_a <= "00110010";
        elsif cur_ind < 3724 and cur_ind > 3227 and --cur_mid < 2607 and cur_mid > 2482 and
        cur_rng < 3600 and cur_rng > 3227 and cur_pnk < 3475 and
        cur_pnk > 2979 and cur_tmb < 2607 and cur_tmb > 2234 then
                letter_d <= not "10011100"; --c
                letter_a <= "00100001";
        elsif cur_ind < 4096 and cur_ind > 3103 --and cur_mid < 2607 and cur_mid > 2482
        and cur_rng < 3600 and cur_rng > 2482 and cur_pnk < 3103 and
        cur_pnk > 2482 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "01111010"; --d
                letter_a <= "00100011";
        elsif cur_ind < 3165 and cur_ind > 2731 --and cur_mid < 2607 and cur_mid > 2482
        and cur_rng < 2731 and cur_rng > 2234 and cur_pnk < 3165 and
        cur_pnk > 2482 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "10011110"; --e
                letter_a <= "00100100";
        elsif cur_ind < 3103 and cur_ind > 2482 --and cur_mid < 3848 and cur_mid > 2979
        and cur_rng < 3724 and cur_rng > 3600 and cur_pnk < 3972 and
        cur_pnk > 3475 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "10001110"; --f
                letter_a <= "00101011";
        elsif cur_ind < 3475 and cur_ind > 3103 --and cur_mid < 2607 and cur_mid > 2482
        and cur_rng < 2731 and cur_rng > 2234 and cur_pnk < 2482 and
        cur_pnk > 2234 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "10111100"; --g
                letter_a <= "00110100";
        elsif cur_ind < 3972 and cur_ind > 3475 --and cur_mid < 3848 and cur_mid > 3103
        and cur_rng < 2731 and cur_rng > 2234 and cur_pnk < 3227 and
        cur_pnk > 2855 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "01101110"; --h
                letter_a <= "00110011";
        elsif cur_ind < 2482 and cur_ind > 2234 --and cur_mid < 2607 and cur_mid > 2482
        and cur_rng < 2731 and cur_rng > 2234 and cur_pnk < 3600 and
        cur_pnk > 3351 and cur_tmb < 2482 and cur_tmb > 2234 then
                letter_d <= not "01100000"; --i
                letter_a <= "01000011";

--              if pBTN1 = '1' then letter_d <= not "01111000"; --j
--                      letter_a <= "00111011";
--              end if;
--              if pBTN2 = '1' then letter_d <= not "00001110"; --k
--                      letter_a <= "01000010";
--              end if;
--              if pBTN3 = '1' then letter_d <= not "00011100"; --l
--                          letter_a <= "01001011";
--                  end if;
--          end if;
--          if SW(3) = '1' then
--                  if pBTN0 = '1' then letter_d <= not "10101010"; --m
--                          letter_a <= "00111010";
--                  end if;
--                  if pBTN1 = '1' then letter_d <= not "11101100"; --n
--                          letter_a <= "00110001";
--                  end if;
--                  if pBTN2 = '1' then letter_d <= not "11111100"; --o
--                          letter_a <= "01000100";
--                  end if;
--                  if pBTN3 = '1' then letter_d <= not "11001110"; --p
--                          letter_a <= "01001101";
--                  end if;
--          end if;
--          if SW(4) = '1' then
--                  if pBTN0 = '1' then letter_d <= not "11100110"; --q
--                          letter_a <= "00010101";
--                  end if;
--                  if pBTN1 = '1' then letter_d <= not "00001010"; --r
--                          letter_a <= "00101101";
--                  end if;
--                  if pBTN2 = '1' then letter_d <= not "10110110"; --s
--                          letter_a <= "00011011";
--                  end if;
--                  if pBTN3 = '1' then letter_d <= not "00011110"; --t
--                          letter_a <= "00101100";
--                  end if;
--          end if;
--          if SW(5) = '1' then
--                  if pBTN0 = '1' then letter_d <= not "01111100"; --u
--                          letter_a <= "00111100";
--                  end if;
--                  if pBTN1 = '1' then letter_d <= not "01010100"; --v
--                          letter_a <= "00101010";
--                  end if;
--                  if pBTN2 = '1' then letter_d <= not "01010110"; --w
--                          letter_a <= "00011101";
--                  end if;
--                  if pBTN3 = '1' then letter_d <= not "10010010"; --x
--                          letter_a <= "00100010";
--                  end if;
--          end if;
--          if SW(6) = '1' then
--                  if pBTN0 = '1' then letter_d <= not "01110110"; --y
--                          letter_a <= "00110101";
--                  end if;
--                  if pBTN1 = '1' then letter_d <= not "01001010"; --z
--                          letter_a <= "00011010";
--                  end if;
--                  if pBTN2 = '1' then letter_d <= not "00000010"; --space
--                          letter_a <= "00101001";
--                  end if;
--                  if pBTN3 = '1' then letter_d <= not "11100100"; --enter
--                          letter_a <= "01011010";
--                  end if;
--          end if;
--end if;
            else letter_d <= not "00000001"; -- .
            end if;
```

```
D7S <= letter_d;
A <= "0111";
end if;
i <= 0;
parity_bit <= not (letter_a(0) xor letter_a(1) xor letter_a(2) xor letter_a(3) xor letter_a(4) xor letter_a(5) xor letter_a(6)
xor letter_a(7));
--          parity: while (i < 8) loop
--                      if letter_a(i) = '1' then parity_int <= parity_int + 1;
--                      end if;
--                      i <= i +1;
--          end loop parity;
--          if (parity_int = 0 OR parity_int = 2 OR parity_int = 4 OR parity_int = 6 or parity_int = 8) then parity_bit <=
'0';
--                      else parity_bit <= '1';
--          end if;
if(rising_edge(k25hz)) then
            state_c <= state_n;
            output1 <= "0" & letter_a & parity_bit & "1";
            output2 <= output1;
            output3 <= output2;
            output4 <= output3;
            output5 <= output4;
            output6 <= output5;
            output7 <= output6;
end if;

--if (output1 = output2) then output_p <='1';
--else output_p <='0';
--if (output1 = output2 and output2 = output3 and output3 = output4 and output4 = output5 and output5 = output6 and
output6 /= output7) then
--          output_p <= '1';
if (output1 = output2 and output2 = output3 and output3 = output4 and output4 /= output5) then
            output_p <= '1';
else output_p <='0';
--end if;
end if;
flag <= output_p;
end process;

concurrent2 : process(k25hz, clk, output2, output3, output4, output5, output6, output7, output1, state_n, flag)
begin

case state_c is
--and output2 /= output1 and output2 /= "00000000000"
            when s0 => kbd_data <= '1';
                        kbd_clk <= '1';
                                    if (flag = '1') then
                                                state_n <= s1;
                                    else state_n <= s0;
                                    end if;

            when s1 => kbd_data <= output1(0);
                        kbd_clk <= k25hz;
                        state_n <= s2;

            when s2 => kbd_data <= output1(8);
                        kbd_clk <= k25hz;
                        state_n <= s3;

            when s3 => kbd_data <= output1(7);
                        kbd_clk <= k25hz;
                        state_n <= s4;
```

```
        when s4 => kbd_data <= output1(6);
                kbd_clk <= k25hz;
                state_n <= s5;

        when s5 => kbd_data <= output1(5);
                kbd_clk <= k25hz;
                state_n <= s6;

        when s6 => kbd_data <= output1(4);
                kbd_clk <= k25hz;
                state_n <= s7;

        when s7 => kbd_data <= output1(3);
                kbd_clk <= k25hz;
                state_n <= s8;

        when s8 => kbd_data <= output1(2);
                kbd_clk <= k25hz;
                state_n <= s9;

        when s9 => kbd_data <= output1(1);
                kbd_clk <= k25hz;
                state_n <= s10;

        when s10 => kbd_data <= output1(9);
                kbd_clk <= k25hz;
                state_n <= s11;

        when s11 => kbd_data <= output1(10);
                kbd_clk <= k25hz;
                state_n <= s12;

        when s12 => kbd_data <= '1';
                kbd_clk <= '0';
                state_n <= s13;

        when s13 => kbd_data <= '1';
                kbd_clk <= '0';
                state_n <= s0;

        end case;
end process;


--Test2 code

    clock_divide : process(clk)
    begin
      if (clk = '1' and clk'event) then
        clk_counter <= clk_counter + '1';
                                    clk_25k_count <= clk_25k_count + '1';
      end if;
    end process;

    clk_div <= clk_counter(1);
    SCLKA <=  not clk_counter(1);
                 SCLKB <=  not clk_counter(1);
                 SCLKD <=  not clk_counter(1);
                 start <= clk_25k_count(11);
```

```
counter : process(clk_div, enParalelLoad, enShiftCounter)
        begin
            if (clk_div = '1' and clk_div'event) then

                if (enShiftCounter = '1') then
                   temp1 <= temp1(14 downto 0) & SDATA1;
                   temp2 <= temp2(14 downto 0) & SDATA2;
                   temp3 <= temp3(14 downto 0) & SDATB1;
                   temp4 <= temp4(14 downto 0) & SDATB2;
                   temp5 <= temp5(14 downto 0) & SDATD1;
                   temp6 <= temp6(14 downto 0) & SDATD2;
                   shiftCounter <= shiftCounter + '1';
                elsif (enParalelLoad = '1') then
                   shiftCounter <= "0000";
                   DATA1 <= temp1(11 downto 0);
                   DATA2 <= temp2(11 downto 0);
                   DATB1 <= temp3(11 downto 0);
                   DATB2 <= temp4(11 downto 0);
                   DATD1 <= temp5(11 downto 0);
                   DATD2 <= temp6(11 downto 0);
                end if;
            end if;
        end process;


SYNC_PROC: process (clk_div)
   begin
      if (clk_div'event and clk_div = '1') then
                                        current_state <= next_state;
      end if;
   end process;


OUTPUT_DECODE: process (current_state)
   begin
      if current_state = Idle then
            enShiftCounter <='0';
            DONE <='1';
            nCSA <='1';
                                                    nCSB <='1';
                                                    nCSD <='1';
            enParalelLoad <= '0';
--                                                  LED <= "00000010";
        elsif current_state = ShiftIn then
            enShiftCounter <='1';
            DONE <='0';
            nCSA <='0';
                                                    nCSB <='0';
                                                    nCSD <='0';
            enParalelLoad <= '0';
--                                                  LED <= "00000100";
        else --if current_state = SyncData then
            enShiftCounter <='0';
            DONE <='0';
            nCSA <='1';
                                                    nCSB <='1';
                                                    nCSD <='1';
            enParalelLoad <= '1';
--                                                  LED <= "00001000";
        end if;
   end process;
```

```
 NEXT_STATE_DECODE: process (current_state, START, shiftCounter)
begin

   next_state <= current_state;  -- default is to stay in current state

   case (current_state) is
      when Idle =>
         if START = '1' then
            next_state <= ShiftIn;
         end if;
      when ShiftIn =>
         if shiftCounter = x"F" then
            next_state <= SyncData;
         end if;
      when SyncData =>
         if START = '0' then
         next_state <= Idle;
         end if;
      when others =>
         next_state <= Idle;
   end case;
end process;


LEDbling: process (Data1, Data2, Datb1, Datb2, Datd1, Datd2, k25hz, slow_clk2)
begin
int_data1 <= to_integer(unsigned(Data1));
int_data2 <= to_integer(unsigned(Data2));
int_datb1 <= to_integer(unsigned(Datb1));
int_datb2 <= to_integer(unsigned(Datb2));
int_datd1 <= to_integer(unsigned(Datd1));
int_datd2 <= to_integer(unsigned(Datd2));
cur_ind   <= int_data1;
cur_mid   <= int_data2;
cur_rng   <= int_datb1;
cur_pnk   <= int_datb2;
cur_tmb   <= int_datd1;

if rising_edge(k25hz) then
--          if SW(0) = '1' then
--                      if    index_thres(7) > 4096 then LED <= "11111111";
--                      elsif index_thres(7) > 3584 then LED <= "01111111";
--                      elsif index_thres(7) > 3072 then LED <= "00111111";
--                      elsif index_thres(7) > 2560 then LED <= "00011111";
--                      elsif index_thres(7) > 2048 then LED <= "00001111";
--                      elsif index_thres(7) > 1536 then LED <= "00000111";
--                      elsif index_thres(7) > 1024 then LED <= "00000011";
--                      elsif index_thres(7) > 512  then LED <= "00000001";
--                      else LED <= "00011000";
--                      end if;
--          elsif SW(1) = '1' then
--                      if    int_data2 > 4096 then LED <= "11111111";
--                      elsif int_data2 > 3584 then LED <= "01111111";
--                      elsif int_data2 > 3072 then LED <= "00111111";
--                      elsif int_data2 > 2560 then LED <= "00011111";
--                      elsif int_data2 > 2048 then LED <= "00001111";
--                      elsif int_data2 > 1536 then LED <= "00000111";
--                      elsif int_data2 > 1024 then LED <= "00000011";
--                      elsif int_data2 > 512  then LED <= "00000001";
--                      else LED <= "00011000";
--                      end if;
```

```
--		elsif SW(2) = '1' then
--				if    int_datb1 > 4096 then LED <= "11111111";
--				elsif int_datb1 > 3584 then LED <= "01111111";
--				elsif int_datb1 > 3072 then LED <= "00111111";
--				elsif int_datb1 > 2560 then LED <= "00011111";
--				elsif int_datb1 > 2048 then LED <= "00001111";
--				elsif int_datb1 > 1536 then LED <= "00000111";
--				elsif int_datb1 > 1024 then LED <= "00000011";
--				elsif int_datb1 > 512  then LED <= "00000001";
--				else LED <= "00011000";
--				end if;
--		elsif SW(3) = '1' then
--				if    int_datb2 > 4096 then LED <= "11111111";
--				elsif int_datb2 > 3584 then LED <= "01111111";
--				elsif int_datb2 > 3072 then LED <= "00111111";
--				elsif int_datb2 > 2560 then LED <= "00011111";
--				elsif int_datb2 > 2048 then LED <= "00001111";
--				elsif int_datb2 > 1536 then LED <= "00000111";
--				elsif int_datb2 > 1024 then LED <= "00000011";
--				elsif int_datb2 > 512  then LED <= "00000001";
--				else LED <= "00011000";
--				end if;
--		elsif SW(4) = '1' then
--				if    int_datd1 > 4096 then LED <= "11111111";
--				elsif int_datd1 > 3584 then LED <= "01111111";
--				elsif int_datd1 > 3072 then LED <= "00111111";
--				elsif int_datd1 > 2560 then LED <= "00011111";
--				elsif int_datd1 > 2048 then LED <= "00001111";
--				elsif int_datd1 > 1536 then LED <= "00000111";
--				elsif int_datd1 > 1024 then LED <= "00000011";
--				elsif int_datd1 > 512  then LED <= "00000001";
--				else LED <= "00011000";
--				end if;
--		elsif SW(5) = '1' then
				if    int_datd2 = 2055 then LED <= "11111111";
				elsif int_datd2 = 2056 then LED <= "01111111";
				elsif int_datd2 = 2057 then LED <= "00111111";
				elsif int_datd2 = 2058 then LED <= "00011111";
				elsif int_datd2 = 2054 then LED <= "00001111";
				elsif int_datd2 = 2053 then LED <= "00000111";
				elsif int_datd2 = 2052 then LED <= "00000011";
				elsif int_datd2 = 2051 then LED <= "00000001";
				else LED <= "00011000";
				end if;
--		else LED <= "00000000";
--		end if;

		if SW(7) = '1' then
				if    SW(0) = '1' then int_to_d7s <= int_data1;
					elsif SW(1) = '1' then int_to_d7s <= int_data2;
					elsif SW(2) = '1' then int_to_d7s <= int_datb1;
					elsif SW(3) = '1' then int_to_d7s <= int_datb2;
					elsif SW(4) = '1' then int_to_d7s <= int_datd1;
					elsif SW(5) = '1' then int_to_d7s <= int_datd2;
				end if;

				digit0	<= (int_to_d7s * 2097) / 2097152;
				REM0		<= (int_to_d7s) mod (1024);
				digit1	<= (REM0 * 41) / 4096;
				REM1		<= ((REM0 * 41 / 4096) mod (100 * 41 / 4096))* 41 / 4096;
				digit2	<= (REM1 * 410) / 4096;
				digit3	<= ((REM1 * 410 / 4096) mod (10* 410 / 4096)) * 410 / 4096;
```

```
                    bin_dig0 <= conv_std_logic_vector(digit1, 8);
--                  LED <= bin_dig0;
                    if          digit0 = 0 then letter_thou <= "00000010";
                                elsif digit0 = 1 then letter_thou <= "10011110";
                                elsif digit0 = 2 then letter_thou <= "00100100";
                                elsif digit0 = 3 then letter_thou <= "00001100";
                                elsif digit0 = 4 then letter_thou <= "10011000";
                                else letter_thou <=  "00000000";
                    end if;
                    D7S_thou <= letter_thou;

                    if          digit1 = 0 then letter_hund <=  "00000011";
                                elsif digit1 = 1 then letter_hund <=  "10011111";
                                elsif digit1 = 2 then letter_hund <=  "00100101";
                                elsif digit1 = 3 then letter_hund <=  "00001101";
                                elsif digit1 = 4 then letter_hund <=  "10011001";
                                elsif digit1 = 5 then letter_hund <=  "01001001";
                                elsif digit1 = 6 then letter_hund <=  "01000001";
                                elsif digit1 = 7 then letter_hund <=  "00011111";
                                elsif digit1 = 8 then letter_hund <=  "00000001";
                                elsif digit1 = 9 then letter_hund <=  "00001001";
                                else letter_hund <=  "00000000";
                    end if;
                    D7S_hund <= letter_hund;

                    if          digit2 = 0 then letter_tens <=  "00000011";
                                elsif digit2 = 1 then letter_tens <=  "10011111";
                                elsif digit2 = 2 then letter_tens <=  "00100101";
                                elsif digit2 = 3 then letter_tens <=  "00001101";
                                elsif digit2 = 4 then letter_tens <=  "10011001";
                                elsif digit2 = 5 then letter_tens <=  "01001001";
                                elsif digit2 = 6 then letter_tens <=  "01000001";
                                elsif digit2 = 7 then letter_tens <=  "00011111";
                                elsif digit2 = 8 then letter_tens <=  "00000001";
                                elsif digit2 = 9 then letter_tens <=  "00001001";
                                else letter_tens <=  "00000000";
                    end if;
                    D7S_tens <= letter_tens;

                    if          digit3 = 0 then letter_ones <=  "00000011";
                                elsif digit3 = 1 then letter_ones <=  "10011111";
                                elsif digit3 = 2 then letter_ones <=  "00100101";
                                elsif digit3 = 3 then letter_ones <=  "00001101";
                                elsif digit3 = 4 then letter_ones <=  "10011001";
                                elsif digit3 = 5 then letter_ones <=  "01001001";
                                elsif digit3 = 6 then letter_ones <=  "01000001";
                                elsif digit3 = 7 then letter_ones <=  "00011111";
                                elsif digit3 = 8 then letter_ones <=  "00000001";
                                elsif digit3 = 9 then letter_ones <=  "00001001";
                                else letter_ones <=  "00000000";
                    end if;
                    D7S_ones <= letter_ones;
                    end if;
          end if;

--if rising_edge(k25hz) then
--     current_st <= next_st;
--
--     case (current_st) is
--       when first =>
--                                        D7s <= d7s_thou;
--                                              A <= "1110";
--                                              next_st <= second;
--                                  when second =>
--                                              D7s <= D7s_hund;
--                                              A <= "1101";
--                                              next_st <= third;
--        when third =>
--                                              D7s <= D7s_tens;
--                                              A <= "1011";
--                                              next_st <= fourth;
--        when fourth =>
--                                              D7s <= D7s_ones;
--                                              A <= "0111";
--                                              next_st <= first;
--      end case;
--end if;

end process;

end Mixed;
```

Appendix E: Seven segment codes for letters

Letter	7 seg + DP	Letter	7 seg + DP
A	11101110	S	10110110
B	00111110	T	00011110
C	10011100	U	01111100
D	01111010	V	01010100
E	10011110	W	01010110
F	10001110	X	10010010
G	10111100	Y	01110110
H	01101110	Z	01001010
I	01100000	0	00000011
J	01111000	1	10011111
K	00001110	2	00100101
L	00011100	3	00001101
M	10101010	4	10011001
N	11101100	5	01001001
O	11111100	6	01000001
P	11001110	7	00011111
Q	11100110	8	00000001
R	00001010	9	00001001

Table 2: 7-Segment Display codes (with Decimal Point)

(Note: a decimal point was added to the numbers to allow them to differentiated from letters in the cases where they were similar or the same.)