



UNIVERSITY OF  
**FLORIDA**

---

**EEL 4914C Senior Design**

**Final Report**

**Fall 2007**

**EEgrip**

**Team: NiteFlex**

**Submitted by:**

**Sean Nelson**

**[galagher@ufl.edu](mailto:galagher@ufl.edu)**

**(407) 234-1173**

**Gina Wansor**

**[gwansor@ufl.edu](mailto:gwansor@ufl.edu)**

**(561) 702-0411**

# Table of Contents

Table of Contents.....	2
Project Abstract.....	3
Introduction .....	3
Project Features/Objectives .....	3
Project Concept/Technology Selection.....	7
Design Procedure and Setbacks.....	8
Prototype Cost Analysis .....	10
Project Task Breakdown .....	11
Timeline .....	12
Appendix A – Project Code C Programming .....	13
Appendix B – Project Flowcharts and Diagrams.....	46
Appendix C – Datasheets .....	48
Appendix D – Project Resources .....	49
Appendix E – Pictures of Prototype Development.....	49

## Project Abstract

This project's primary goal was to design an artificial hand that performs controlled movements, and had a secondary goal of being able to grasp an object. This biologically inspired artificial hand focused on the use of the Nitinol<sup>1</sup> wire to act as "muscles" with a variety of electrical components and sensors acting as the nervous system.

## Introduction

This project has a multitude of applications in the engineering industry. With its focus on using Nitinol wire as "muscles", it creates an interesting path for further research for artificial hands in the biomedical engineering field. The main purpose would be in the medical field, where the artificial hand could be used for patients who have trouble with movement in their hand or for patients who do not have a hand. Another possibility would be in a hazardous environment where objects cannot be handled directly. The general idea of the EEgrip is to provide a proof-of-concept of using the Nitinol wire to create an artificial hand that models the human hand.

## Project Features/Objectives

1. EEgrip has three fingers – a pointer finger, a middle finger, and a thumb.
2. EEgrip is designed with Nitinol wire technology which is used to act as "muscles" to move the hand. These "muscles" are located in the forearm and there are strings acting as tendons, which are used to translate the motion to the digits, very similar to how a human hand functions as seen in figure 1.
3. The artificial hand contains two kinds of sensors. The first, flex sensors, are used to determine the positioning of the fingers. The processor reads from the input flex sensors (on a glove) and tells the fingers on the hands where they should be. The second type of sensors is the force sensor. These sensors are used to determine if the hand is holding an object and if it is holding the object too tightly, not to close anymore.
4. The microprocessor, Atmega2560, is used to model the nervous system. The microprocessor acts as the brain, while the sensors serve as the nervous system. The wires transmit the electrical signals from the "brain" and "nervous system" to the "muscles". This concept is shown in the block diagram given in figure 2.

---

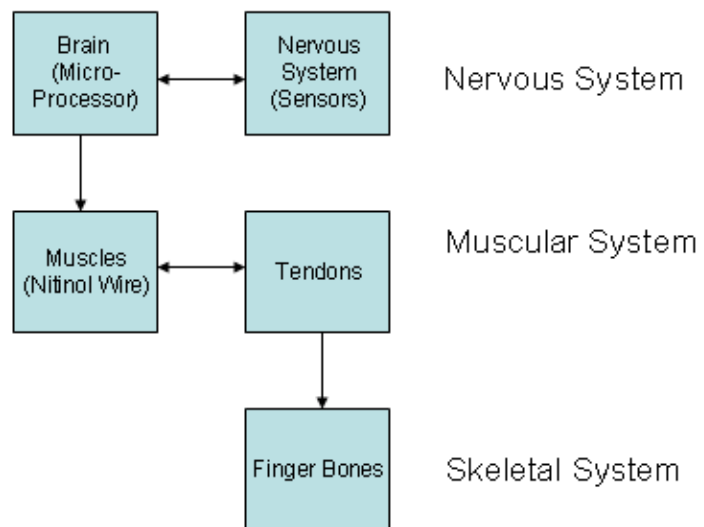
<sup>1</sup> Nitinol indicates any nickel-titanium shape memory alloy.



**Figure 1** Picture showing tendon and muscle attachments from the finger bones on the hand to the muscles in the arm for a human hand. [3DScience.com]

### Performance Objectives

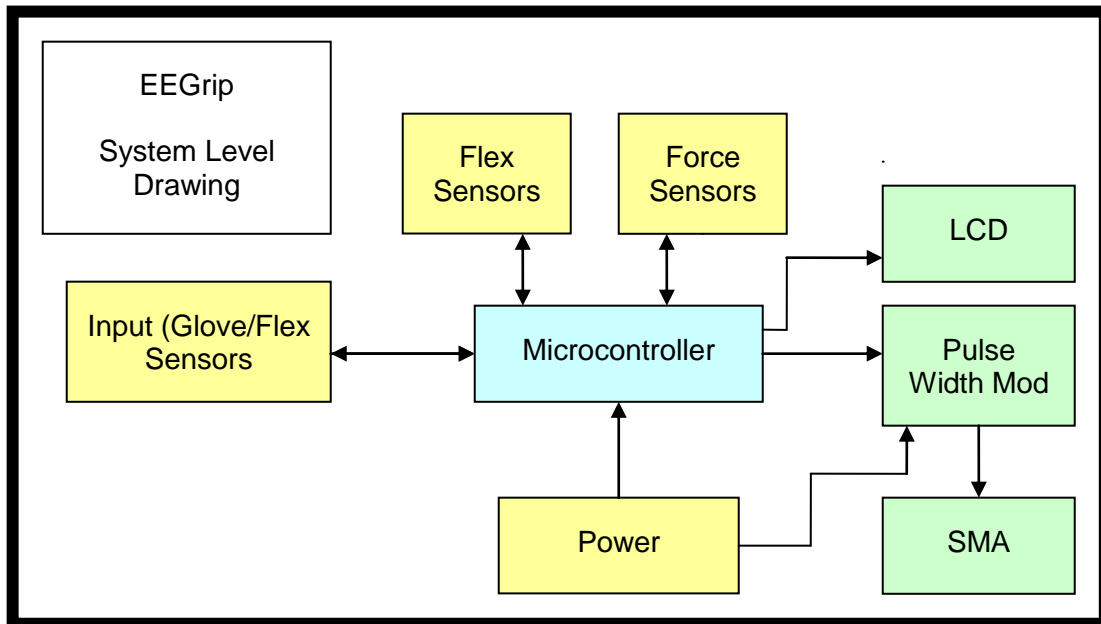
1. The artificial hand performs controlled motions via commands given by the microprocessor. This is demonstrated by having a user wear the glove and control the movement of the three fingers.
2. The artificial hand also performs gripping motions that will allow it to hold certain objects. This shall be demonstrated by activating the hand to grip around some large object. Sensors will provide the feedback necessary for the microprocessor to control this gripping motion.



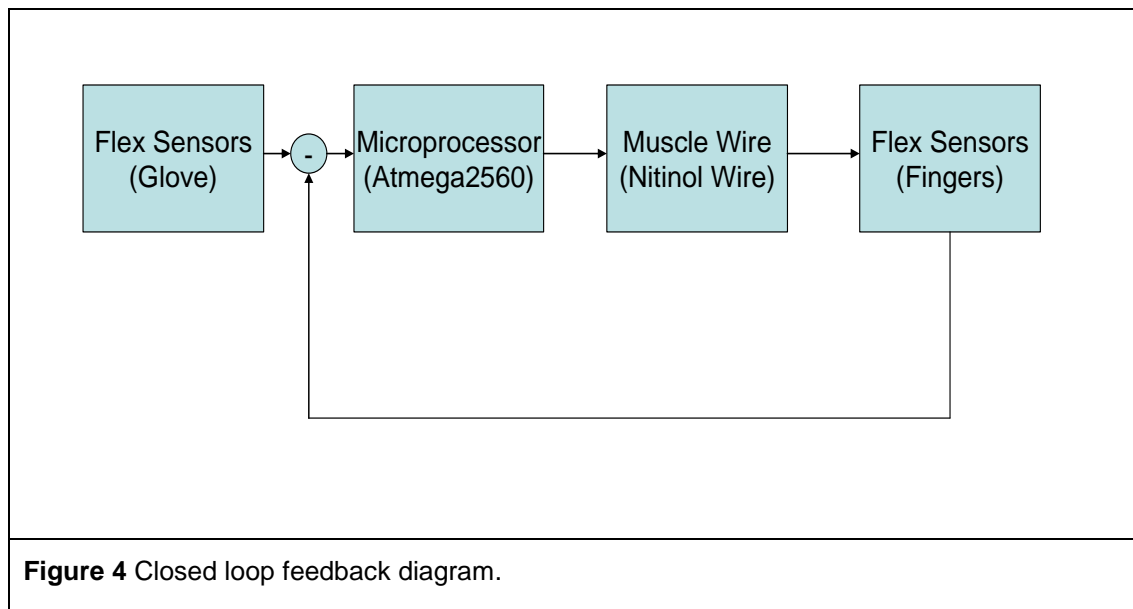
**Figure 2** Picture showing artificial hand system block diagram.

## Electrical Design and System

A block diagram of the electrical design for the artificial hand is shown below in figure 3. The artificial hand is run by a microcontroller, the Atmega2560, in combination with force sensors, flex sensors, and Nitinol wire. This configuration enables closed loop control of the motion of the hand demonstrated by the closed loop diagram provided in figure 4.



**Figure 3** Artificial hand system block diagram.

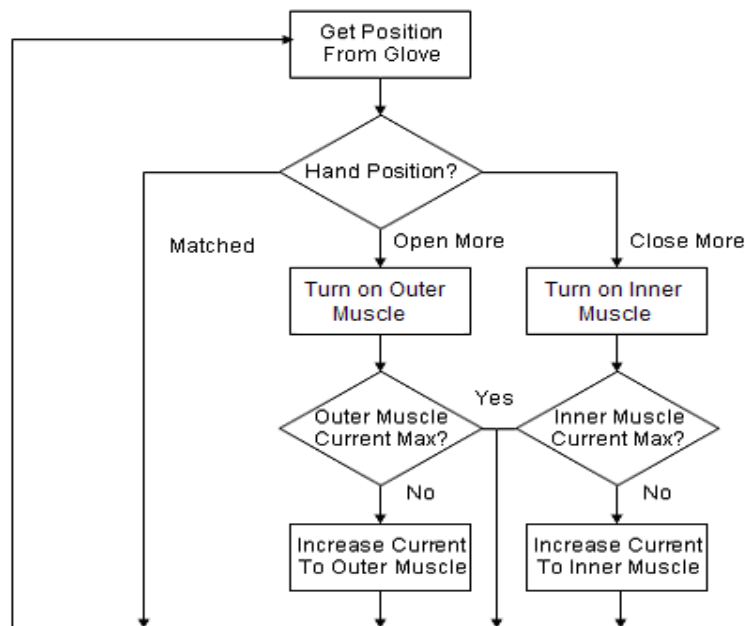


**Figure 4** Closed loop feedback diagram.

A glove, containing the flex sensors, is connected to the base of the hand and provides input to the microcontroller of the artificial hand. This input gives feedback from the user to control the actions of the hand.

The main component of the artificial hand consists of the mechanical layout, which contains the controller, sensors, and Nitinol wire. Flex and force sensors are placed in the fingers of the artificial hand. These sensors provide input and feedback to the microcontroller to help determine the position of the fingers and hand. The flex sensor's position on each finger is compared with the equivalent flex sensor that is on the glove and adjusts the finger accordingly. The fingers are adjusted by the Nitinol wire by running a current through the wire. This is done using Pulse Width Modulation (PWM) to control the current. If the fingers are grasping an object, the force sensors allow for the hand to check if it is grasping the object too tightly. If it is grabbing the object too tightly – the current to the wire is stopped by the controller. The microcontroller will provide power to these sensors, as well as provide the control signal for the current regulation circuit.

For a complete schematic of the electrical components of the artificial hand, please see Appendix B.



**Figure 4** Artificial hand control algorithm for one finger.

## Project Concept/Technology Selection

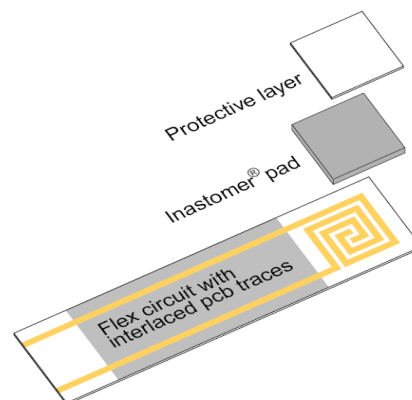
Presently, there are a few products on the market that are comparable; however, most concentrate on the use of motors to actuate the hand. Other technologies include pneumatics or “Air Muscles” and hydraulics. Each requires pumps to move air or oil to move the hand. The EEgrip implements a revolutionary option using Nitinol wire. This idea was stumbled upon over the previous summer and sparked our interest.

### Hardware

Nitinol provides “motor-less motion” to move the artificial hand very similar to the way muscles work in the human body. When a current is run through this wire, it heats up and causes the wire to contract by a percentage of the overall length; when the wire cools it expands back to its original size. Using the Nitinol wire, this artificial hand demonstrates a technology that has not been a standard in the industry for prosthetics. This allows us to experiment with a technology that is innovative and cutting edge in the field of electrical engineering, as well as biomedical engineering. This demonstrates the simulation of muscles, which may allow improved actuation speed, realistic feedback control, and usability.

For the processor, we chose to use the Atmega2560. We chose it for a number of reasons, but mostly because this processor has the number of A/D converter ports that were needed, as well as the capability to run PWM through the processor.

Flex sensors were chosen for the input and for positioning the fingers. These flex sensors are actually a bend sensor technology, which work as potentiometer that works on bend sensitive ink. This gives a way to provide a user interface with a glove, that can provide feedback to the processor and tell the fingers how to position themselves. This is done by controlling the amount of current or voltage that is run to the Nitinol wire which pulls the finger open or closed.



The force sensors were added to the project to allow for the EEgrip to grasp objects without squeezing too tightly, in case the object might be fragile or a different shape

than what might be grasped by the input. These sensors use a material called Inastomer material, which changes resistance in response to forces like compression.

## Software

C was used to run the software for the project. The C programming language has a lot of flexibility and is one of the main languages used with the Atmel processors. This was also the language that our team is the most familiar with.

## Design Procedure and Setbacks

At first, we came up with a general design that we wanted to use for the artificial hand. This included picking a processor (the Atmega2560) and sensors (the flex sensors and force sensors) that we wanted to use for our project. After talking with multiple people, we decided we would like to run our wires with Pulse Width Modulation(PWM), using the processor.

For general testing and building, we used a main PCB board for the microprocessor and a few smaller boards for the flex and force sensors. We also used an LCD for debugging purposes. One of the first problems we encountered was hooking the LCD to the microprocessor. We were attempting to use the LCD on port C; however, port C on the Atmega2560 is a special function port. When moving the LCD to port A or D, it worked on both. Using the boards for the electrical components was a good way to measure if the sensors and processor were working.

At the same time, the wire was tested. First, the 100 micrometer diameter wire was tested; this one worked at a good speed, but the wire did not have enough strength for our purposes. The 300 micrometer diameter wire was also tested; this wire had the necessary strength, however the contraction rate and cool down rate were too slow. We then began using Flexinol springs, which are springs made from the Nitinol wire. They provided a good actuation speed, as well as the strength needed for our artificial hand.

At this point, we began working on integrating software into our hardware. We programmed the Atmega2560 to read analog data from each of the sensors using the A/D ports and read the values to the LCD. We then programmed to output the PWM for one of the fingers. Here, we varied the current, depending on the position of the flex sensors (comparing the glove flex sensor to the finger flex sensor). After adding these features, we started testing the hardware with the PWM, with the MOSFETs, and with the springs.



This is the point that we ran into a huge challenge and setback. We originally designed to have one spring contract to close the finger and a second spring contract to open the finger. However, when testing this all of the hardware together, the actual rate was very slow. The springs needed about 45 seconds to cool down, and essentially, the two springs were



competing against each other. To work with that setup, we tried several methods to cool down the wire. The following methods were used in attempts of improving the spring reaction:

- adjusting the code by cutting out delays and turning off current at certain points
- using a small fan to cool down the springs
- putting the springs in parallel
- temperature sensors to record temp of the spring

Adjusting the code and using a small fan to cool down the springs showed little improvement with the hand movement. Putting the springs in parallel made the actuation difficult, since the springs seemed to be different lengths (depending if one had an extra coil). The temperature sensors showed some improvement, as we cut off the current when the wire reached a certain temperature. However, this was not enough improvement for our purposes.

At this point, we switched to the 200  $\mu\text{m}$  diameter wire. This wire proved to actuate quick enough for our purposes, be strong enough to allow the fingers to grasp an object, and did not heat up as much as the previous wires.

We then made a PCB board containing all of our electrical components, designed with Protel, the design software provided in the senior design lab. It took about four total boards to get the final working board. There were challenges with putting 100-pin microprocessor on the board, and some of the design specs changed.

We seemed to have some software issues with the PWM at this point. The PWM was letting out a signal at the end of each period, which affected our application, because the MOSFET would read about 10V (out of the 18V supply) which would make the wire actuate. We tried adjusting the period or settings of the PWM, but the best solution seem to be only turning on the ports for the PWM when actually needing the PWM.

After this was fixed, the MOSFET gates were left floating, and resistors were added to tie them low. This completed the hardware and software components of the artificial hand.

## **Mechanical Design**

For the mechanical layout, we started out testing on a toy hand. With this particular model, there was a lot of resistance, but worked for testing purposes. We then found a finger that was made from a mold, and chose to use these fingers in our project. We used three fingers to create a “hand” with a pointer finger, middle finger, and thumb. For the palm and the forearm, we used a metal bracket to hold the fingers and connected it with a metal frame. With this project, there were a lot of mechanical challenges. The first became the resistance of the fingers, which was generally fixed by the new finger mold.

The second challenge came when tying the finger with the wire. The challenge comes when using 1 meter of the wire, and compacting it into a smaller space, our forearm. For this, we implemented a “pulley” system using thread spindles and a limited number of bearings. Although this method worked, it still caused a lot of friction. With more bearings, this method could be improved. Some additional friction is added in various spots of the mechanical setup. With some mechanical study, the finger movement could be improved.

For more specific diagrams of our mechanical parts, see Appendix B.

## Prototype Cost Analysis

Figure 5 shows a compilation of components and estimated cost for the assembling our prototype for the artificial hand. This is just of the basic components that were actually used in the prototype. Many other supplies were bought and tested, but did not fit what we were looking for. With total parts bought and tested, building the prototype cost between 400 and 500 dollars.

Item	Parts	Quantity	Units	Market Cost Each	Our Cost Each	Estimated Cost
1	Microprocessor (Atmega2560)	1	Per	\$14.00	\$0.00	\$0.00
2	Flex sensors	6	Per	\$6.00	\$0.00	\$0.00
3	Force sensors	3	Per	\$10.00	\$0.00	\$0.00
4	MOSFETs	6	Per	Varies	\$0.00	\$0.00
5	Terminal Blocks	6	Per 4	\$6.00	\$12.00	\$12.00
6	Resistors, BJTs, LEDs, Caps, etc.	n/a	Per	Varies	\$0.00	\$0.00
7	LCD screen	1	Per	\$12.00	\$0.00	\$0.00
8	Switches	2	Per	\$1.00	\$0.00	\$0.00
9	IC sockets	4	Per	\$1.00	\$0.00	\$0.00
10	PCB board	1	Per	\$20.00	\$0.00	\$0.00
11	Nitinol [Flexinol Wire]	6	1 meter	\$4.50	\$4.50	\$27.00
13	Cloth glove	1	Per	\$5.00	\$5.00	\$5.00
14	Molded Fingers	3	Per	\$55.00	\$5.00	\$165.00
15	Metal Frame	1	Per	\$10.00	\$5.00	\$5.00
16	Misc Mechanical Parts	n/a	Per	\$50.00	\$10.00	\$10.00

**Total Estimated Cost for Prototype**

	<b>\$224.00</b>
--	-----------------

Figure 5 Parts list.

# Project Task Breakdown

<b>Task</b>	<b>Team Member</b>
<b>Introduction</b>	GW, SN
<b>Research/Preliminary Design</b>	GW, SN
<b>Design Overview and Layout</b>	GW, SN
<b>Acquire and Test Parts</b>	
Order Mechanical Parts	GW, SN
Test Mechanical Parts	SN
Order Electrical Parts	SN
Test Electrical Parts (Breadboard)	SN
Test Electrical Parts with Software	GW
<b>Build Breadboard Design/Functional Component</b>	
Electrical Hardware Prototype	SN
Software Prototype	GW
Mechanical Prototype	SN
<b>Develop Software for Design</b>	GW
<b>Develop Hardware for Design</b>	SN
<b>Merge Hardware and Software Designs</b>	GW, SN
<b>Debug</b>	GW, SN
<b>Fine tune</b>	GW, SN
<b>Demo (team)</b>	GW, SN

Figure 6 Task breakdown.

# Timeline

Figure 7 uses a Gantt chart to show an estimated time schedule of the project development and assembly. Figure 8 provides a more detailed breakdown of the schedule.

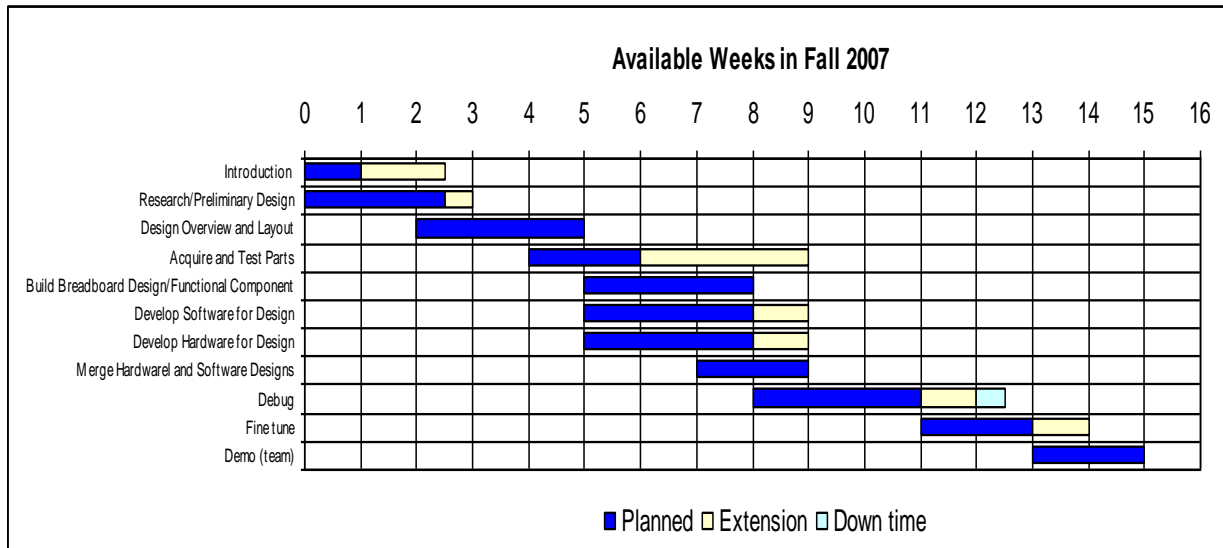


Figure 7 Gantt Chart.

Team: NiteFlex		Project: EEGrip				
Task Name		Planned	Extension	Down time		
Introduction	0	1	1.5	0	2.5	
Research/Preliminary Design	0	2.5	0.5	0	4	
Design Overview and Layout	2	3	0	0	5	
Acquire and Test Parts	4	2	3	0	9	9/30/2007
Build Breadboard Design/Functional Component	5	3	0	0	8	9/30/2007
Develop Software for Design	5	3	1	0	9	10/31/2007
Develop Hardware for Design	5	3	1	0	9	10/31/2007
Merge Hardware and Software Designs	7	2	0	0	9	11/19/2007
Debug	8	3	1	0.5	12.5	11/19/2007
Fine tune	11	2	1	0	14	11/28/2007
Demo (team)	13	2	0	0	14	11/28/2007

**Figure 8** Gantt Chart details.

This schedule was designed to be flexible to allow for unexpected delays and for the most part was followed, give or take a week.

## Appendix A – Project Code C Programming

List of included .c files:

**test1.c** : Main file for calling functions

**fwire.c** : File to call functions to operate PWM & movement of fingers

**flex.c** : File for flex sensors (reading in, AtoD, and converting for LCD)

**force.c** : File for force sensors (reading in, AtoD, and converting for LCD)

**LCD.c** : File for LCD functions

**avrutil.c** : File for functions of processor --- including AtoD ports

```
//Filename : test1.c
```

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <avr/signal.h>
```

```
#include <inttypes.h>
```

```
#include "string.h"
```

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```
#include "math.h"
```

```
#include "avrutil.h"
```

```
#include "lcd.h"
```

```
#include "fwire.h"
```

```
#include "force.h"
```

```
#include "flex.h"
```

```
int main()
```

```
{
```

```
    //Initialize AtoDs, Timer, LCD, and wire functions
```

```
    init_AtoD();
```

```
    init_timer();
```

```
    init_LCD();
```

```
    init_fwire();
```

```

timer_bdelay(100);

lcd_clrscr();

char buffer3[25];

char buffer4[25];

while(1)
{
    timer_bdelay(100);

    //initialize force & flex sensors prior to running the finger movement
    run_force();

    run_flex();

    //run_temp();

    //initialize movement of fingers
    closePointerFinger();

    closeMiddleFinger();

    closeThumb();

    //print to LCD screen for purposes of debugging
    lcd_home(0);

    //lcd_clrscr(); //clear screen or go back to home
    //flx = read_adc(0); //read AtoD values of ADC0

    //sprintf(buffer0,"ADC0=%4d", flx); //converts values to decimal

    //flx1 = read_adc(1); //read AtoD values of ADC0 from the 2nd flex sensor
    //sprintf(buffer1, " ADC1=%4d\n" , flx1);

    //Conversion of PWM numbers for fingers for output to LCD screen
    //sprintf(buffer3, "PWMT0=%4d", OCR1A); //thumb
    //sprintf(buffer4, "PWMT0=%4d", OCR1B);
    //sprintf(buffer3, "PWMM0=%4d", OCR3B); //middle

```

```

    //sprintf(buffer4, "PWMMC=%4d", OCR3C);
    //sprintf(buffer3, "PWMPO=%4d", OCR5A); //pointer
    //sprintf(buffer4, "PWMPC=%4d", OCR5B);
    //Print test to screen
    //lcd_print(buffer0);
    //lcd_print(buffer1);
    //lcd_print(buffer3);
    //lcd_print(buffer4);
}
return 0;
}

```

**//Filename: fwire.c**

```
#ifdef FWIRE_H
```

```
#ifndef FWIRE_C
```

```
#define FWIRE_C
```

```
int pointerG; //flx=pointerG
```

```
int pointerF; //flx=pointerF
```

```
int middleF;
```

```
int middleG;
```

```
int thumbF;
```

```
int thumbG;
```

```
int forcep;
```

```
int forcem;
```

```
int forcet;
```

```
/*=====*/  
=====*/
```

```
//Description:      Setup PWM generation to control current/voltage for movement of flexinol  
wire [fingers]
```

```
//Precondition:     Voltage = 5V, Period of 20ms, current (I) in wire begins at 0A.
```

```
//Postcondition:    The closing and opening flexinol wire is setup to output PWM on PORTB,  
PORTE, and PORTL
```

```
//Returns:
```

```
void init_fwire()
```

```
{
```

```
    //Setup PWM using PORTB,PORTE,PORTL as outputs
```

```
    //DDRB |= 0x60;          //Set B5:6 to output for PWM Thumb
```

```
    //DDRE |= 0x30;          //Set E4:E5 to output for PWM Middle
```

```
    //Setup Fast PWM, Mode 14 [Page 148]
```

```
    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM11);
```

```
    //Clear OC1A/B when TCNT1 == OCR1A/B, Set when TCNT1 == 0x0000
```

```
    TCCR1B = _BV(WGM13) | _BV(WGM12) | _BV(CS11) | _BV(CS10);          //Clear  
                                                                    TCNT1 when TCNT1 == ICR1
```



```

//TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM11);
//Clear OC1A/B when TCNT1 == OCR1A/B, Set when TCNT1 == 0x0000

//TCCR1B = _BV(WGM13) | _BV(WGM12) | _BV(CS11) | _BV(CS10);           //Clear
                                                                    TCNT1 when TCNT1 == ICR1

TCCR3A = _BV(COM3B1) | _BV(COM3C1) | _BV(WGM31);
//Clear OC3B/C when TCNT3 == OCR3B/C, Set when TCNT1 == 0x0000

TCCR3B = _BV(WGM33) | _BV(WGM32) | _BV(CS31) | _BV(CS30);           //Clear
TCNT3 when TCNT3 == ICR3

TCCR5A = _BV(COM5A1) | _BV(COM5B1) | _BV(WGM51);
//Clear OC5A/B when TCNT5 == OCR5A/B, Set when TCNT4 == 0x0000

TCCR5B = _BV(WGM53) | _BV(WGM52) | _BV(CS51) | _BV(CS50);           //Clear
TCNT4 when TCNT4 == ICR4

ICR1= 2500; //Set TOP

ICR3= 2500;

ICR5= 2500;

OCR1A = 0;           //Closing spring at rest
OCR1B = 0;           //Opening spring at rest

OCR3B = 0;           //Closing spring at rest
OCR3C = 0;           //Opening spring at rest

OCR5A = 0;           //Closing spring at rest
OCR5B = 0;           //Opening spring at rest

}

```

```
//=====
```

```
//Description: Control movement of pointer finger by output of the PWM generation
```

```
//Precondition: init_fwire, run_force, and run_flex were called
```

```
//Postcondition: Current will be on or off depending on the position of flex sensors
```

```
//Returns:
```

```
void closePointerFinger()
```

```
{  
  
    if ((pointerG > (pointerF + 4)) && (OCR5A <= ICR5) && (forcep>=230))  
    {  
        DDRL |= 0x08; //Set L3 to output for PWM Pointer  
        OCR5A = abs(pointerG - pointerF)*100 + 450;  
        OCR5B = 0;  
    }  
    else if ((pointerG < (pointerF - 4)) && (OCR5B <= ICR5))  
    {  
        DDRL |= 0x10; //Set L3 to output for PWM Pointer  
        OCR5B = abs(pointerG - pointerF)*100 + 450;  
        OCR5A = 0;  
    }  
    else  
    {
```

```

        DDRL &= 0x00;
        OCR5A = 0;
        OCR5B = 0;
    }
}

```

```

/*=====
=====*/

```

```

//Description:      Control movement of middle finger
//Precondition:    init_fwire, run_force, and run_flex were called
//Postcondition:   Middle finger opens or closes
//Returns:

```

```

void closeMiddleFinger()

```

```

{

    if ((middleG > (middleF + 4)) && (OCR3B <= ICR3) && (forcem>=230))
    {
        DDRE |= 0x10;                //Set E4 to output for PWM Pointer
        OCR3B = abs(middleG - middleF)*100 + 450;
        OCR3C = 0;
    }

    else if ((middleG < (middleF - 4)) && (OCR3C <= ICR3))
    {

```

```

        DDRE |= 0x20;                //Set E5 to output for PWM Pointer
        OCR3C = abs(middleG - middleF)*100 + 450;
        OCR3B = 0;
    }
else
    {
        DDRE &= 0x00;
        OCR3B = 0;
        OCR3C = 0;
    }

}

/*=====
=====*/

```

```

//Description:      Control movement of Thumb
//Precondition:    init_fwire, run_force, and run_flex were called
//Postcondition:   Thumb opens or closes
//Returns:

```

```

void closeThumb()

```

```

{

    if ((thumbG > (thumbF + 4)) && (OCR1A <= ICR1) && (forcet>=230))
    {

```

```

        DDRB |= 0x20;                //Set B5 to output for PWM Pointer
        OCR1A = abs(thumbG - thumbF)*100 + 450;
        OCR1B = 0;
    }

    else if ((thumbG < (thumbF - 4)) && (OCR1B <= ICR1))
    {
        DDRB |= 0x40;                //Set B6 to output for PWM Pointer
        OCR1B = abs(thumbG - thumbF)*100 + 450;
        OCR1A = 0;
    }

    else
    {
        DDRB &= 0x00;
        OCR1A = 0;
        OCR1B = 0;
    }

}

#endif

#endif

//Filename: force.c

#ifdef FORCE_H
#ifndef FORCE_C
#define FORCE_C

```

```

int forcep;
int forcem;
int forcet;
char buffer6[25];

void run_force()

{
    forcep = read_adc(6); //read AtoD values of ADC4 of force sensor
    sprintf(buffer6, "ADC6=%4d", forcep);
    //lcd_print(buffer6);

    forcem = read_adc(7); //read AtoD values of ADC7 of force sensor
    //sprintf(buffer6, "ADC7=%4d", forcem);
    //lcd_print(buffer6);

    forcet = read_adc2(0); //read AtoD values of ADC8 of force sensor
    //sprintf(buffer6, " ADC8=%4d", forcet);
    //lcd_print(buffer6);*/

}

#endif
#endif

```

**//filename is flex.c**

```
#ifndef FLEX_H
```

```
#ifndef FLEX_C
```

```
#define FLEX_C
```

```
int pointerG;
```

```
int pointerF;
```

```
int middleG;
```

```
int middleF;
```

```
int thumbG;
```

```
int thumbF;
```

```
char buffer0[25];
```

```
char buffer1[25];
```

```
void run_flex()
```

```
{
```

```
    pointerG = read_adc(0); //read AtoD values of ADC0
```

```
    sprintf(buffer0,"poiG =%4d", pointerG); //converts values to decimal
```

```
    //lcd_print(buffer0); //print values to LCD screen
```

```
    middleG = read_adc(1);
```

```
    //sprintf(buffer0, "midG=%4d" , middleG);
```

```
    //lcd_print(buffer0);
```

```

        thumbG = read_adc(2);
//sprintf(buffer0, "thuG=%4d" , thumbG);
        //lcd_print(buffer0);

pointerF = read_adc(3);
sprintf(buffer1, " poiF=%4d\n" , pointerF);
        //lcd_print(buffer1);

middleF = read_adc(4);
//sprintf(buffer1, " midF=%4d\n" , middleF);
        //lcd_print(buffer1);

thumbF = read_adc(5);
//sprintf(buffer1, "thuF=%4d\n" , thumbF);
        //lcd_print(buffer1);
}

#endif

#endif

//Filename      : lcd.c
//Author         : Thomas A. McDonley (darkeyes@ufl.edu, darkeyes_z@hotmail.com)
//Description:
//Copyright      : (C) 2007

```



//EULA : GNU

/\* LCD Hitachi HD44780u in 4bit mode  
Copyright (C) 2007 Thomas A. McDonley

This program is free software; you can redistribute it and/or  
modify it under the terms of the GNU General Public License  
as published by the Free Software Foundation; either version 2  
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program; if not, write to the Free Software  
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

\*/

/\*Modified by Gina M. Wansor Fall 2007\*/

#ifdef LCD\_H  
#ifndef LCD\_C  
#define LCD\_C

```

//=====
//Description:
//Precondition:
//Postcondition:
//Returns:
void init_LCD()
{
    DDRD |= 0x7F; //PORTD(6) is Enable
                //PORTD 5-0 = R/W | RS |
DB7 | DB6 | DB5 | DB4

    timer_bdelay(15); //Delay for 15ms to allow VCC to settle

    lcd_writeCmd(0x33, 5, 1); //Manual Initialization of LCD Driver
    lcd_writeCmd(0x32, 5, 2); //Four bit operation

    lcd_writeCmd(0x28, 1, 2); //Enable 2 lines
    lcd_writeCmd(0x0F, 1, 2); //Display on; Cursor On; Blink On
    lcd_writeCmd(0x01, 1, 2); //Clear Screen; Cursor Home
}
//=====
//=====
//=====
//Description: Abstracts how commands are written to the LCD driver
//Precondition: The lcd has been init and is in 4 bit mode

```

```

//Postcondition:      The command has been executed

//Returns:

void lcd_writeCmd(char cmd, unsigned int wdelay, unsigned int edelay)
{
    PORTD = ( (cmd & 0xF0) >> 4) | 0x40;          //Write upper nibble to cmd register

    PORTD ^= 0x40;                                // Enable = 0... Falling edge
signals to read bus

    timer_bdelay(wdelay);                          //Delay 1ms for write to settle
[Hold time > 100us]

    PORTD = ( (cmd & 0x0F) ) | 0x40;              //Write lower nibble to cmd register

    PORTD ^= 0x40;

    timer_bdelay(edelay);                          //Delay 2ms (Wait for
instruction to execute)

}

//=====
=====

//=====
=====

//Description:      Abstracts how data is written to the LCD
//Precondition:      the lcd has been init and is in 4 bit mode
//Postcondition:     The character appears on the screen at the current cursor position
//Returns:

void lcd_writeData(char data, unsigned int wdelay, unsigned int edelay)
{

```

```
PORTD = ( (data & 0xF0) >> 4) | 0x40 | 0x10; //Write upper nibble to cmd register... Enable = 1... RS = Data register
```

```
PORTD ^= 0x40; // Enable = 0... Falling edge signals to read bus
```

```
timer_bdelay(wdelay); //Delay 1ms for write to settle [Hold time > 100us]
```

```
PORTD = ( (data & 0x0F) ) | 0x40 | 0x10; //Write lower nibble to cmd register
```

```
PORTD ^= 0x40;
```

```
timer_bdelay(edelay); //Delay 2ms (Wait for instruction to execute)
```

```
}
```

```
//=====
```

```
//=====
```

```
//Description: Shifts the display either right or left
```

```
//Precondition: dir is equal to 0 to shift left and 1 to shift right
```

```
//Postcondition: display has been shifted
```

```
//Returns:
```

```
void lcd_shiftDisplay(unsigned short int dir)
```

```
{
```

```
    if(dir == 0)
```

```
        lcd_writeCmd(0x1C, 1, 2);
```

```
    else
```

```
lcd_writeCmd(0x18, 1, 2);
```

```
}
```

```
//=====
```

```
//=====
```

```
//Description:      Turns the lcd display off
```

```
//Precondition:
```

```
//Postcondition:    The display is off
```

```
//Returns:
```

```
void lcd_off()
```

```
{
```

```
    lcd_writeCmd(0x08, 1, 2);
```

```
}
```

```
//=====
```

```
//=====
```

```
//Description:      Turns the lcd display on
```

```
//Precondition:      Cursor and blink are set according to what is desired to be on
```

```
//Postcondition:     The display is on
```

```
//Returns:
```

```
void lcd_on(unsigned short int cursor, unsigned short int blink)
```

```

{
    if(!cursor && !blink)        lcd_writeCmd(0x0C, 1, 2);
    else if(cursor && blink)      lcd_writeCmd(0x0F, 1, 2);
    else                          lcd_writeCmd(0x0E, 1, 2);
}

//=====
=====

//=====
=====

//Description:      Take the cursor to the beginning of a line
//Precondition:     home = 0 for first line, 1 for second
//Postcondition:    The cursor is at the beginning of a line
//Returns:
void lcd_home(unsigned short int home)
{
    if(home == 0) lcd_writeCmd(0x02, 1, 2);
    else          lcd_writeCmd(0xC0, 1, 2);
}

//=====
=====

//=====
=====

//Description:      Prints a null terminated character array to the LCD. '\n' will cause it to
goto line 2

```

```
//Precondition:      string != NULL and is a pointer to a character array. ex: lcd_print("Hello
/nWorld!")
```

```
//Postcondition:
```

```
//Returns:
```

```
void lcd_print(char* string)
```

```
{
```

```
    int i=0;
```

```
    if(string == 0) return;                //Error: Dereferencing a Null pointer
```

```
    while(string[i] != '\0')
```

```
    {
```

```
        if(string[i] == '\n')
```

```
            lcd_home(1);
```

```
        else
```

```
            lcd_writeData(string[i], 0, 1);
```

```
        i++;
```

```
    }
```

```
}
```

```
//=====
```

```
//=====
```

```
//Description:      Clears the LCD screen and moves the cursor to line0 home
```

```
//Precondition:
```

```
//Postcondition:
```

```
//Returns:
```

```

void lcd_clrscr()
{
    lcd_writeCmd(0x01, 0, 2);
}

//=====
=====

#endif

#endif

//Filename    : avrutil.c

//Author      : Thomas A. McDonley (darkeyes@ufl.edu, darkeyes_z@hotmail.com)
//Description:
//Copyright   : (C) 2007
//EULA        : GNU

/*    AVR Utilities
    Copyright (C) 2007 Thomas A. McDonley

    This program is free software; you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation; either version 2
    of the License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of

```



MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

\*/

```
#ifndef AVRUTIL_H
#ifdef AVRUTIL_C
#define AVRUTIL_C
```

```
//=====
```

```
//Description:
```

```
//Precondition:
```

```
//Postcondition:
```

```
//Returns:
```

```
void init_AtoD()
```

```
{
```

```
    ADMUX = ADC_VREF_TYPE;    //Internal 5V Reference, Left Aligned
```

```
    ADCSRA = 0x85;           //Enable AtoD @ 125.000 kHz
```

```
}
```

```
//=====
```

```
//=====
=====
```

```
//Description:
```

```
//Precondition: adc_input is the pin0-7 that we would like to sample
```

```
//Postcondition:
```

```
//Returns:
```

```
unsigned char read_adc(unsigned char adc_input)
```

```
{
```

```
    ADCSRB &= ~_BV(MUX5);          //Select Portf0-7
```

```
    ADMUX = adc_input | ADC_VREF_TYPE;
```

```
    ADCSRA |= 0x40;                // Start the AD conversion
```

```
    while ((ADCSRA & 0x10)==0) ;    // Wait for the AD conversion to complete
```

```
    ADCSRA |=0x10;                 // Clear Conversion Complete Flag
```

```
    return ADCH;
```

```
}
```

```
//=====
=====
```

```
//=====
=====
```

```
//Description:
```

```
//Precondition: adc_input is the pin8-15 that we would like to sample
```

```
//Postcondition:
```

```
//Returns:
```

```
unsigned char read_adc2(unsigned char adc_input)
```

```

{
    ADCSRB |= _BV(MUX5);                //Select Portk0-7
    ADMUX = adc_input | ADC_VREF_TYPE;
    ADCSRA |= 0x40;                    // Start the AD conversion
    while ((ADCSRA & 0x10)==0) ;      // Wait for the AD conversion to complete
    ADCSRA |=0x10;                    // Clear Conversion Complete Flag
    return ADCH;
}

//=====
=====

//=====
=====

//Description:
//Precondition:
//Postcondition:
//Returns:
void write_16reg(int regAddress, int i) // avr/iom128.h
{
    unsigned char sreg;
    /* Save global interrupt flag */
    sreg = SREG;

    cli();
    _SFR_IO16(regAddress) = i;
}

```

```

        /* Restore global interrupt flag */
        SREG = sreg;
    }

//=====
=====

//=====
=====

//Description:
//Precondition:
//Postcondition:
//Returns:
int read_16reg(int regAddress)
{
    unsigned char sreg;
    int i;

    /* Save global interrupt flag */
    sreg = SREG;

    cli();
    i = _SFR_IO16(regAddress);

    /* Restore global interrupt flag */
    SREG = sreg;
}

```

```

    return i;
}

//=====

//=====

//Description:      Setup a general purpose timer on Timer0
//Precondition:     none
//Postcondition:    Timer0 will count in 1ms intervals
//Returns:

void init_timer()
{
    //TIFR0 |= _BV(OCF0A);

    TCCR0B = _BV(WGM01) | _BV(CS00) | _BV(CS01);
            //CTC

    TCNT0 = 0;                                     //Start our count at
zero

    TIMSK0 |= _BV(OCIE0B);                         //enable output compare
interrupt [pg 140]

    OCR0B = 125;                                    //Match in 1ms

    sei();
}

```

```
//=====
=====
```

```
//=====
=====
```

//Description:

//Precondition:

//Postcondition: Modifies the current according to the time passed

//Returns:

SIGNAL(SIG\_OUTPUT\_COMPARE0B)

```
{
    ms_count++;
}
```

```
//=====
=====
```

```
//=====
=====
```

//Description: Returns the current Timer count (ms\_count)

//Precondition: init\_timer() was called

//Postcondition:

//Returns: ms\_count

unsigned int timer\_gettime()

```
{
    return ms_count;
}
```

```
//=====
=====
```

```
//=====
=====
```

```
//Description:      Returns the difference between ms_count and stime
//Precondition:     stime is the start time given as a return value from timer_gettime()
//Postcondition:
```

```
//Returns:          Time Elapsed in ms
```

```
unsigned int timer_getelapsedtime(unsigned int stime)
```

```
{
    return ms_count - stime;
}
```

```
//=====
=====
```

```
//Description:      Does a blocking delay for the specified amount of time (in ms)
//Precondition:     time is in ms
//Postcondition:
```

```
//Returns:          Time Elapsed in ms
```

```
void timer_bdelay(unsigned int time)
```

```
{
    unsigned int start;

    start = timer_gettime();
    while( timer_getelapsedtime(start) < time)
        ;
}
```

```
//Block
```

```
}  
  
//=====
```

---

```
=====
```

```
#endif  
  
#endif  
  
#ifndef FWIRE_H  
#define FWIRE_H  
  
#include <avr/io.h>  
#include <avr/interrupt.h>  
#include <avr/signal.h>  
#include <inttypes.h>  
  
void init_fwire();  
void closePointerFinger();  
void closeMiddleFinger();  
void closeThumb();  
  
#include "fwire.c"  
  
#endif  
  
#ifndef FLEX_H  
#define FLEX_H
```



```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>

void run_flex();

#include "flex.c"

#endif

//Filename      : lcd.h

//Author        : Thomas A. McDonley (darkeyes@ufl.edu, darkeyes_z@hotmail.com)
//Description:
//Copyright     : (C) 2007
//EULA         : GNU

/*      LCD Hitachi HD44780u or LC7985 compatible in 4bit mode
      Copyright (C) 2007 Thomas A. McDonley

      This program is free software; you can redistribute it and/or
      modify it under the terms of the GNU General Public License
      as published by the Free Software Foundation; either version 2
      of the License, or (at your option) any later version.
```

This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.

You should have received a copy of the GNU General Public License  
along with this program; if not, write to the Free Software  
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

```
*/
```

```
#ifndef LCD_H
```

```
#define LCD_H
```

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <avr/signal.h>
```

```
#include <inttypes.h>
```

```
#include "avrutil.h"
```

```
void init_LCD();
```

```
void lcd_writeCmd(char cmd, unsigned int wdelay, unsigned int edelay);
```

```
void lcd_writeData(char data, unsigned int wdelay, unsigned int edelay);
```

```
void lcd_shiftDisplay(unsigned short int dir);

void lcd_off();

void lcd_on(unsigned short int cursor, unsigned short int blink);

void lcd_home(unsigned short int home);

void lcd_print(char* string);

#include "lcd.c"

#endif

//Author      : Thomas A. McDonley (darkeyes@ufl.edu, darkeyes_z@hotmail.com)
//Description:
//Copyright   : (C) 2007
//EULA        : GNU

/*   AVR Utilities
    Copyright (C) 2007 Thomas A. McDonley

    This program is free software; you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation; either version 2
    of the License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

```
*/
```

```
#ifndef AVRUTIL_H
```

```
#define AVRUTIL_H
```

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <avr/signal.h>
```

```
#include <inttypes.h>
```

```
#define NULL 0;
```

```
#define ADC_VREF_TYPE 0x60 //Internal 5V  
Reference, Left Aligned
```

```
void init_AtoD();
```

```
unsigned char read_adc(unsigned char adc_input); //Run one sample of the A/D System
```

```
int read_16reg(int regAddress); //Register Addresses can be found  
in avr/iom2560.h
```

```
void write_16reg(int regAddress, int i); // For example TCNT1 is 0x2C
```

```
volatile unsigned int ms_count=0;
```

```
void init_timer();  
//A general purpose timer with a resolution of 1ms
```

```
    unsigned int timer_gettime(); //Returns the
current timer count

    unsigned int timer_getelapsedtime(unsigned int stime); //Returns the difference
between stime and the current time

    void timer_bdelay(unsigned int time); //Does a blocking delay for
the specified amount of time

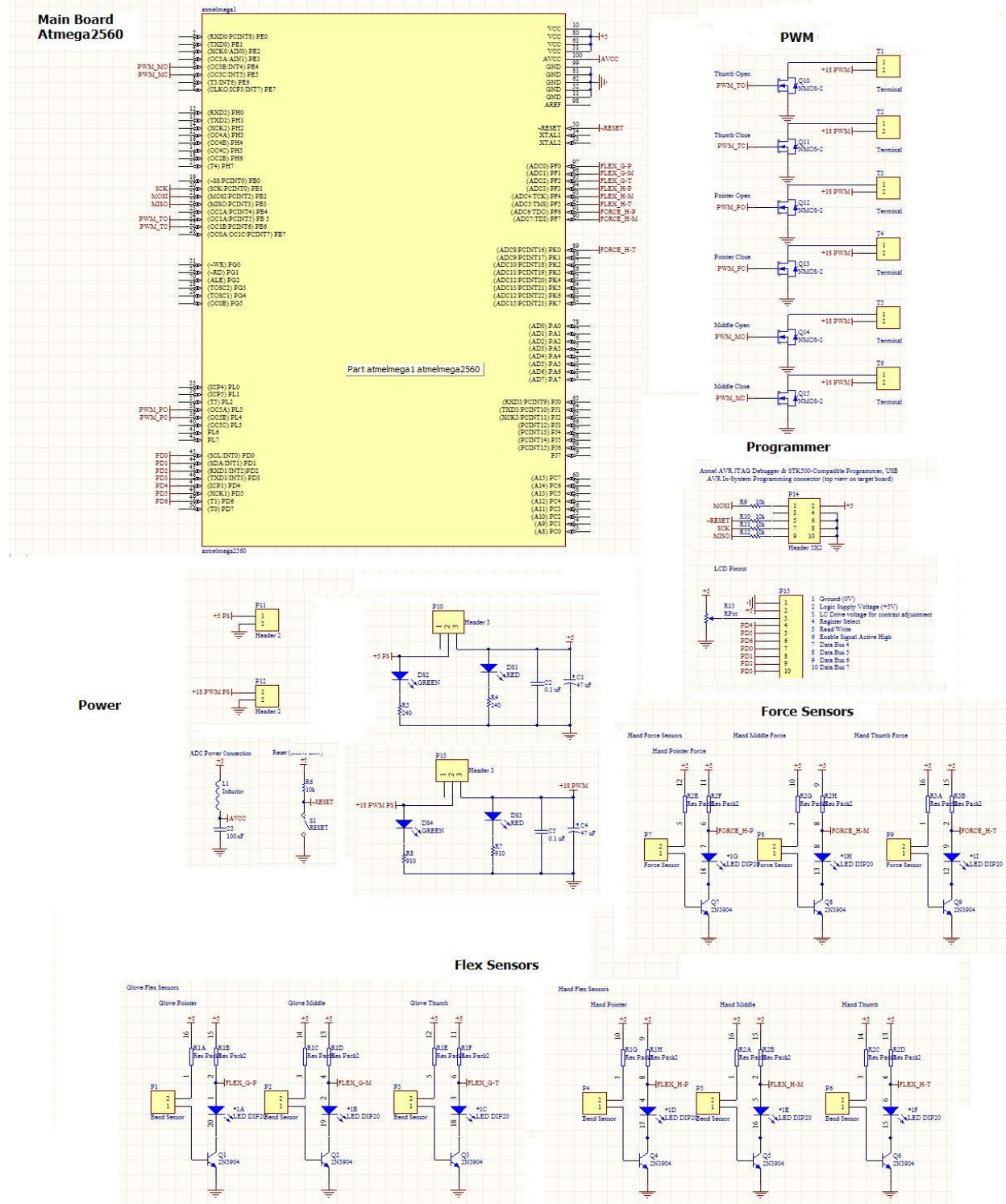
int str_length(char* str);
//Returns the number of characters in a NULL terminated string

#include "avrutil.c"

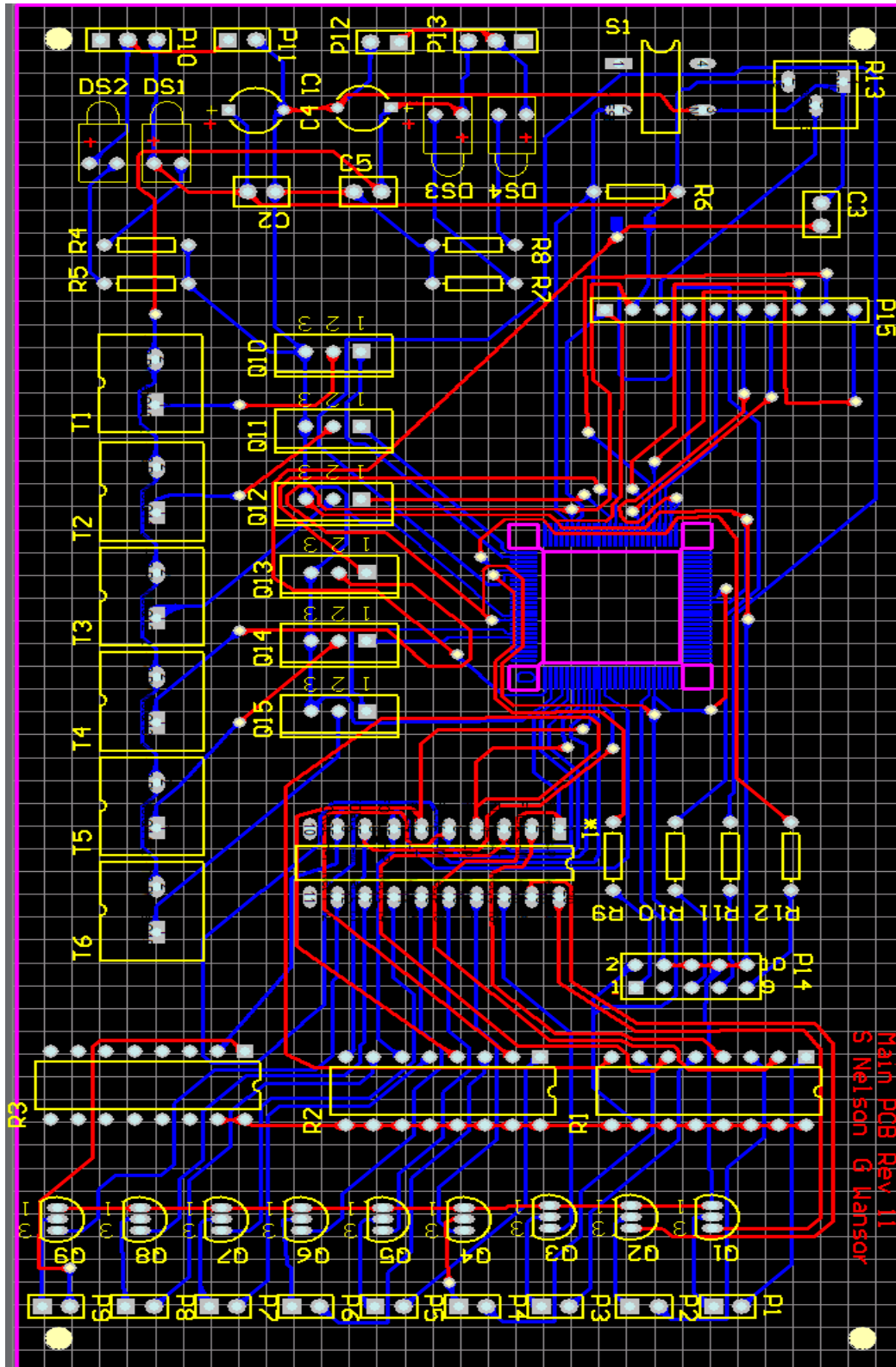
#endif
```

# Appendix B – Project Flowcharts and Diagrams

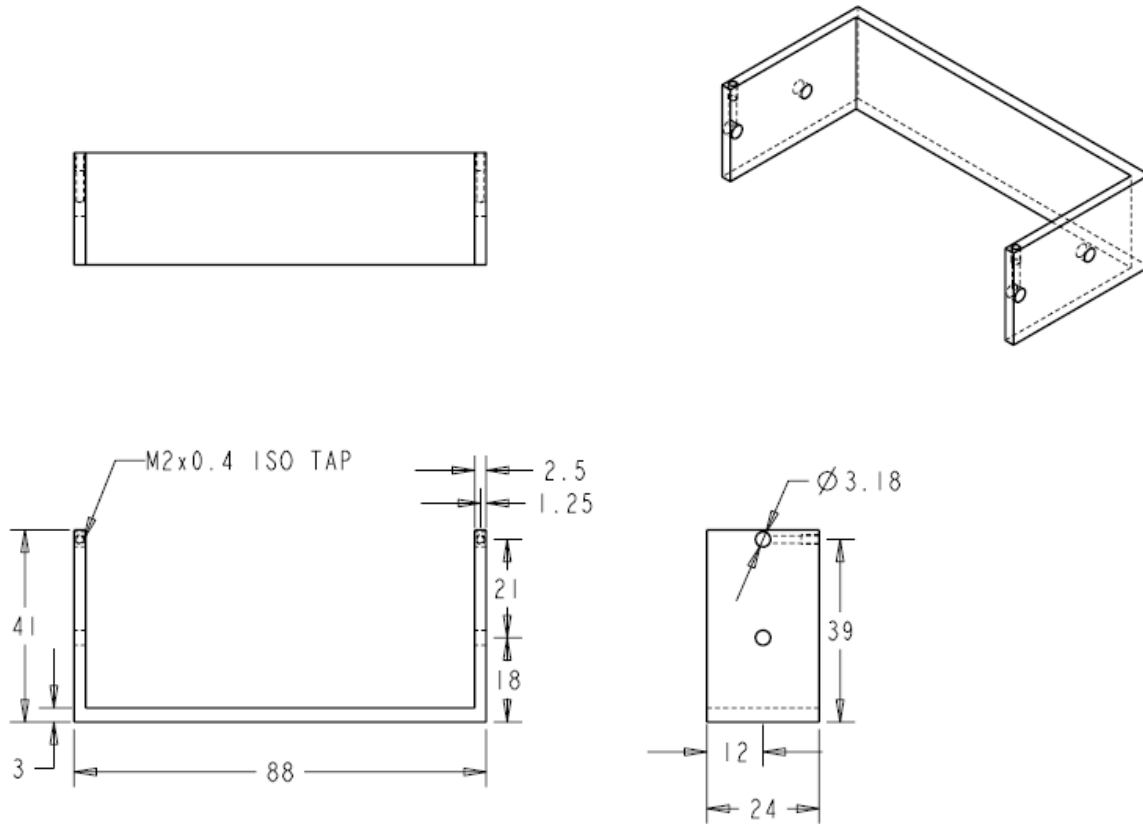
## Overall Schematic – Designed in Protel



# Design for PCB Board



## Mechanical Design Bracket to hold fingers



Designed in Pro-E

## Appendix C – Datasheets

Links:

Atmega2560 - Microprocessor

[http://www.atmel.com/dyn/resources/prod\\_documents/doc2549.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf)

Bend Sensor Technology – Flex Sensors

<http://www.flexpoint.com/technicalDataSheets/electronicDesignGuide.pdf>

Force Sensors

[http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2005/vhl4/juicer\\_webpage/datasheet/InastomerRTechApp.pdf](http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2005/vhl4/juicer_webpage/datasheet/InastomerRTechApp.pdf)

Op Amps

[http://www.datasheetcatalog.com/datasheets\\_pdf/L/M/3/5/LM358.shtml](http://www.datasheetcatalog.com/datasheets_pdf/L/M/3/5/LM358.shtml)



Nitinol Wire

## **Appendix D – Project Resources**

Senior Design Lab  
Senior Design T.A.'s – Andy Sciallo and Thomas McDonley  
Scott Payne – Mechanical Engineering, Graduate Program  
Milind Shastri – Mechanical Engineering, Graduate Program

Joshua Childs, Nick Karlovich  
Mike Stapleton

## **Appendix E – Pictures of Prototype Development**