EE 478 Final Project: Sit-Smart

**KENDRICK TANG  
XIN CHEN  
YANBO ZOU**

Table of Contents

[1. ABSTRACT 4](#_Toc414372693)

[2. INTRODUCTION 4](#_Toc414372694)

[3. SYSTEM DESIGN REQUIREMENT 4](#_Toc414372695)

[System Description 4](#_Toc414372696)

[Specification of External Environment 4](#_Toc414372697)

[System Input and Output Specification 5](#_Toc414372698)

[System Inputs 5](#_Toc414372699)

[System Outputs 5](#_Toc414372700)

[User Interface 5](#_Toc414372701)

[UML Use Cases and Textual Information 5](#_Toc414372702)

[System Functional Specification 7](#_Toc414372703)

[Operating Specification 7](#_Toc414372704)

[Reliability and Safety Specification 7](#_Toc414372705)

[4. DESIGN PROCEDURE 8](#_Toc414372706)

[5. SYSTEM DESIGN DESCRIPTION 8](#_Toc414372707)

[System Description 8](#_Toc414372708)

[Specification of External Environment 8](#_Toc414372709)

[System Input and Output Specification 8](#_Toc414372710)

[System Inputs 8](#_Toc414372711)

[System Outputs 9](#_Toc414372712)

[User Interface 10](#_Toc414372713)

[UML Use Cases and Textual Information 12](#_Toc414372714)

[System Functional Specification 13](#_Toc414372715)

[Operating Specification 15](#_Toc414372716)

[Reliability and Safety Specification 15](#_Toc414372717)

[7. HARDWARE IMPLEMENTAION 15](#_Toc414372718)

[8. SOFTWARE IMPLEMENTATION 20](#_Toc414372719)

[(1)Master Node 20](#_Toc414372720)

[(2)Touchscreen UI subsystem 23](#_Toc414372721)

[(3) Sensor-subsystem 24](#_Toc414372722)

[9. HARDWARE FACTORY COST 24](#_Toc414372723)

[10. TEST PLAN 25](#_Toc414372724)

[11. TEST SPECIFICATIONS 25](#_Toc414372725)

[Posture Feedback 25](#_Toc414372726)

[Posture Control 26](#_Toc414372727)

[Posture Logging 26](#_Toc414372728)

[Bad Posture Recognition 27](#_Toc414372729)

[Local Interface 27](#_Toc414372730)

[Online Interface 28](#_Toc414372731)

[Interface Syncing 28](#_Toc414372732)

[12. Test Cases 29](#_Toc414372733)

[Posture Feedback 29](#_Toc414372734)

[Posture Control 29](#_Toc414372735)

[Posture Logging 30](#_Toc414372736)

[Bad Posture Recognition 30](#_Toc414372737)

[Local Interface 31](#_Toc414372738)

[Online Interface 31](#_Toc414372739)

[Interface Syncing 31](#_Toc414372740)

[13. FAILURE MODES ANALYSIS 32](#_Toc414372741)

[Failure analysis consequences: 32](#_Toc414372742)

[Failure analysis likelihood: 33](#_Toc414372743)

[Failure analysis matrix: 33](#_Toc414372744)

[Failure Analysis: 34](#_Toc414372745)

[14. SYSTEM RESULT AND TIMING DIAGRAM 35](#_Toc414372746)

[15. SYSTEM ERROR ANALYSIS 40](#_Toc414372747)

[16. SUMMARY 41](#_Toc414372748)

[17. CONCLUSION 41](#_Toc414372749)

[APPENDIX A. CODE 42](#_Toc414372750)

[Master 42](#_Toc414372751)

[Touchscreen UI sub-system 58](#_Toc414372752)

[Sensor subsystem 70](#_Toc414372753)

# 1. ABSTRACT

In this report, we introduce our final capstone project Sit-Smart, which aims to help the users to maintain a good sitting posture so as to reduce risks of getting bodily injury, its full development cycle and related documentations. It will first go over the design phase by illustrating how we designed and planned to implement various parts of our system in the design documents. Then it will shift gear to talk about how we tested our system after integration to make sure it works as expected in the test plan section. It also covers the analysis of the results and errors of the system, and the problems we encounter during design and implementation.

# 2. INTRODUCTION

The project presented in this report is the Sit-Smart System. The Sit-Smart System uses I2C to communicate between the different modules; collects, processes, and presents the user’s posture; and uses synchronized local and online interfaces. The purpose of this capstone is to use the exercise the skills, tools, and knowledge that has been accumulated thus far in our lives. This includes designing a system with multiple components, serial communication, data processing, C programming, debugging, use of oscilloscope and logic analyzer, purchasing compatible components and reviewing data sheets, working in a team, and communication. The technical portion of this project includes building, testing, debugging, and integrating multiple modules together with careful consideration to timing and compatibility. The non-technical portion of this project exercises our ability to clearly communicate our ideas, and includes the design reviews, presenting the project to the class, and this document.

# 3. SYSTEM DESIGN REQUIREMENT

## System *Description*

This specification defines the design for the Sit-Smart system. The Sit-Smart system will be able to save one posture as the Desired Posture, and determine the deviation between the Current Posture and the Desired Posture. Posture logging can be enabled and disabled and posture logs will contain time-stamped deviations between the Desired Posture and Current Posture every second in a log file saved to an SD card. Current Posture will be graphically presented to the user through the User Interface (UI).

## Specification of External Environment

The Sit-Smart system is to be used at a desk. It will be designed to operate in a commercial temperature and lighting environment, and the unit will be powered by an external power supply.

## System Input and Output Specification

### System Inputs

The system will receive user controls through the User Interface (UI) – see the User Interface section below. The system also passively receive posture information through sensors in the seat bottom. The sensors consist of force sensitive resistors configured in a voltage divider. There are sixteen sensors arranged in an evenly spaced four-by-four configuration in the seat bottom (Fig 3.1).

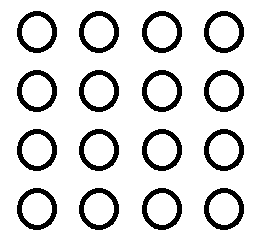


Figure 3.1 Sensor configuration

### System Outputs

There are two system outputs. The first is an audible signal whenever the user’s posture deviates by bad for long enough. Bad posture is determined by the setting, and long enough is determined by the setting. This audible signal will persist until the user returns to good posture, defined to be not bad posture. The second output will be a posture log written to an SD card. The posture log will contain time-stamped deviations between the Desired Posture and Current Posture every second in a log file saved to an SD card.

### User Interface

The UI will consist of a touch screen where the user can specify the two dimensions of sensitivity ( and ), control posture logging, set the Desired Posture, view the graphical representation of their posture, and play the simple game.

## UML Use Cases and Textual Information



Figure 3.2 Requirement specification use case diagram

Set Sensitivity

Sets system sensitivity between low, medium, and high.

Exceptions: if power off, does nothing.

Start Posture Log

Starts a new file, and begin recording posture deviation.

Exceptions: if power off, does nothing. If memory is full, does nothing.

Stop Posture Log

Stops recording posture deviation, and closes current file.

Exceptions: if power off, does nothing. If no file started, does nothing.

Bad Posture Warning System

Makes noise when the user has posture has bad posture for too long.

Exceptions: if power off, does nothing. If posture has not been set, does nothing. If in game mode, does nothing.

## System Functional Specification

The system consists of a User Interface (UI), a network of Sensor Processing Units (SPUs), and a Main Processing Unit (MPU) (Fig 3.3).

The User Interface (UI) will use a Gameduino touch screen interface, and an Arduino Uno. The user will set two dimensions of sensitivity for the Sit-Smart system: one dimension () for posture deviation, and one dimension () for time. The dimension determines how much the user’s current posture can deviate from the user’s desired posture before it is identified as bad, and the dimension determines how long the user can have bad posture before the warning goes off. The UI will communicate with the MPU via I2C serial communication.

The SPUs will consist of sensors, and a PIC which will be used to record sensor values and communicate to the MPU via I2C serial communication.

The MPU determines posture based on sensor readings provided by the Sensor Processing Units (SPUs) and compares it to the desired posture specified by the user (through the UI).



Figure 3.3 Requirement specification high level block diagram

## Operating Specification

The system shall operate in an office environment, which uses commercial lighting and temperature settings.

* Temperature Range: 0-85C
* Power 5.0 V DC

## Reliability and Safety Specification

The system should be able to calibrate based on each user’s input data about their physical builds.

The system and the chair have to be able to undertake a maximum of 300 pounds of weight.

# 4. DESIGN PROCEDURE

To design this system, we first decided what general functionality the system should provide. This list was eventually refined to: local interface, online interface, display heat-map of posture, control with posture (via a simple game), SD logging of posture, and monitor posture with a warning system. The system was then divided into three major sections: sensor processing units, main processing unit, and the user interface unit. Each system was then designed according to what functionality it should provide. The functions were designed as independent modules, assuming that the other modules are functioning properly. Each independent module was then tested with simulated auxiliary modules. The independent modules were then carefully integrated together and tested at each stage.

# 5. SYSTEM DESIGN DESCRIPTION

## System Description

This specification defines the design for the Sit-Smart system. The Sit-Smart system will be able to save one posture as the Desired Posture, and determine the deviation between the Current Posture and the Desired Posture. Posture logging can be enabled and disabled, and posture logs will contain time-stamped deviations between the Desired Posture and Current Posture every second in a log file saved to an SD card. Current Posture will be graphically presented to the user through the User Interface (UI).

## Specification of External Environment

The Sit-Smart system is to be used at a desk. It will be designed to operate in a commercial temperature and lighting environment, and the unit will be powered by an external power supply.

## System Input and Output Specification

### System Inputs

The system will receive user controls through the User Interface (UI) – see the User Interface section below. Information is passed between the UI and the Main Processing Unit (MPU) through I2C serial communication.

The system also passively receives posture information through sensors in the seat bottom. The sensors consist of force sensitive resistors configured in a voltage divider (Fig 5.1). There are sixteen sensors arranged in an evenly spaced four-by-four configuration in the seat bottom clustered into four groups of four (Fig 5.2). Each cluster of four sensors are processed by a different PIC to minimize the amount of analog wiring. Each PIC then communicates the sensor values to the MPU with I2C serial communication.



Fig 5.1. Voltage divider and force sensitive resistor

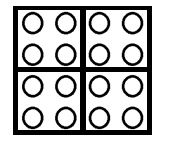


Fig 5.2 Sensor configuration

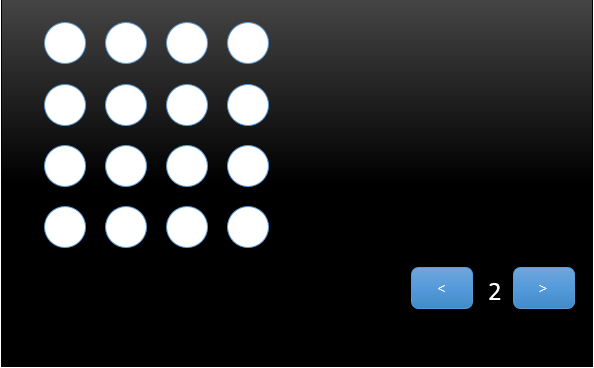
### System Outputs

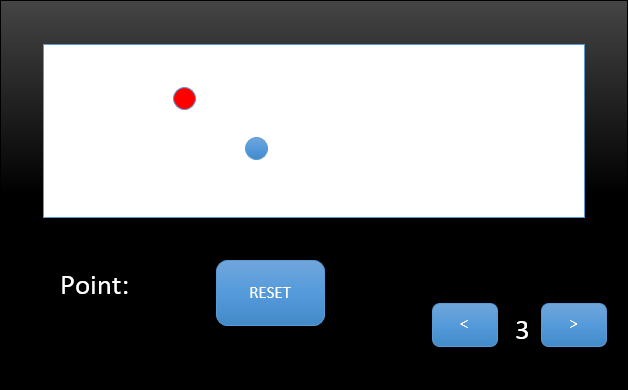
There are two system outputs. The first is an audible signal whenever the user’s posture deviates by bad for long enough. Bad posture is determined by the setting, and long enough is determined by the setting. This audible signal will persist until the user returns to good posture, defined to be not bad posture. The second output will be a posture log written to an SD card. The posture log will contain time-stamped deviations between the Desired Posture and Current Posture every second in a log file saved to an SD card.

### User Interface

The UI will consist of a Gameduino touch screen and an Arduino Uno. There will be three screens (Fig 5.3) for user control where the user can specify the two dimensions of sensitivity ( and ), control posture logging, set the Desired Posture, view the graphical representation of their posture, and play the simple game.





  
Fig 5.3 User interface screens

There is a separated Web UI (Fig 5.4) which can do almost the exactly same work as the touchscreen UI. The system would automatically connect itself to the user designated local Wi-Fi network and start a webserver using the WIFI module integrated in Intel Edison. User can access to this webpage to control the whole system with any device which is able to connect to Wi-Fi and use a web browser.

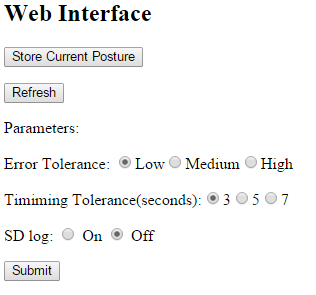


Fig 5.4 A simple local Wi-Fi web interface

## UML Use Cases and Textual Information



Fig 5.5 Design specification use case diagram

Set Sensitivity

Sets system sensitivity between low, medium, and high.

Exceptions: if power off, does nothing.

Start Posture Log

Starts a new file, and begin recording posture deviation.

Exceptions: if power off, does nothing. If memory is full, does nothing.

Stop Posture Log

Stops recording posture deviation, and closes current file.

Exceptions: if power off, does nothing. If no file started, does nothing.

Bad Posture Warning System

Makes noise when the user has posture has bad posture for too long.

Exceptions: if power off, does nothing. If posture has not been set, does nothing. If in game mode, does nothing.

## System Functional Specification

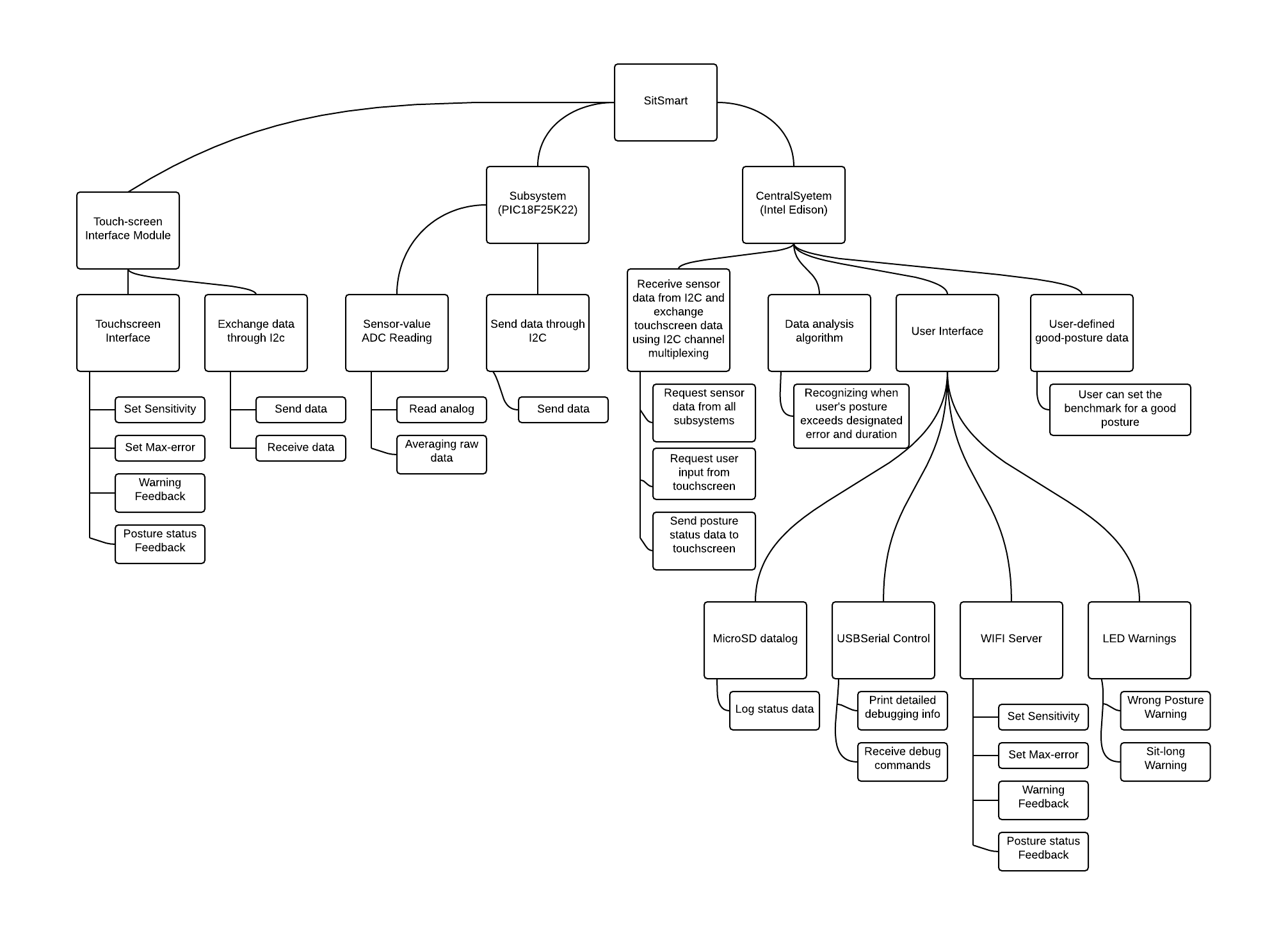
The system consists of a User Interface (UI), a network of Sensor Processing Units (SPUs), and a Main Processing Unit (MPU) (Fig 5.6).

  
Fig 5.6 Requirement specification high level block diagram

The User Interface (UI) will use a Gameduino touch screen interface, and an Arduino Uno. The user will set two dimensions of sensitivity for the Sit-Smart system: one dimension () for posture deviation, and one dimension () for time. The dimension determines how much the user’s current posture can deviate from the user’s desired posture before it is identified as bad, and the dimension determines how long the user can have bad posture before the warning goes off. The UI will communicate with the MPU via I2C serial communication. For the graphical representation of posture, the UI will constantly receive posture information from the MPU. During game mode, the UI will further process posture to control the game.

The SPUs will consist of force sensitive resistors in a voltage divider, ADCs, and will communicate with the MPU via I2C serial communication.

The MPU determines posture based on sensor readings provided by the Sensor Processing Units (SPUs) and compares it to the desired posture specified by the user (through the UI). The MPU relays posture values to, and user controls from the UI. Lastly, the MPU will be able to log posture to the SD card.

   
*Fig 5.7 Functional decomposition*

## Operating Specification

The system shall operate in an office environment, which uses commercial lighting and temperature settings.

* Temperature Range: 0-85C
* Power 5.0 V DC

## Reliability and Safety Specification

The system should be able to calibrate based on each user’s input data about their physical builds.   
The system and the chair have to be able to undertake a maximum of 300 pounds of weight.

# 7. HARDWARE IMPLEMENTAION

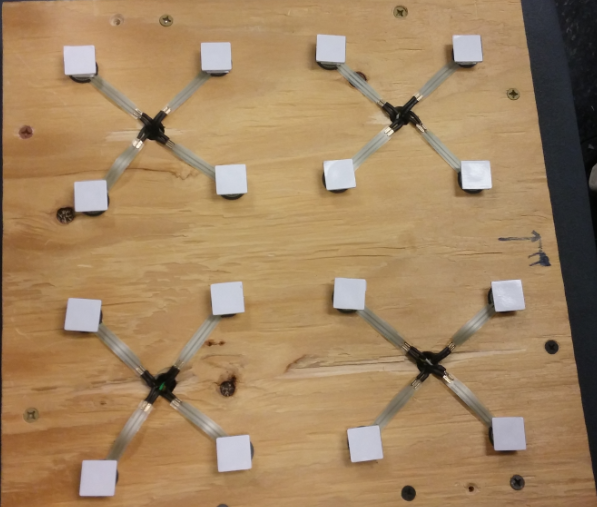
There are three main modules to the Sit-Smart System: the sensor processing units, the main processing unit, and the user interface unit. The sensor processing units each consist of a PIC18F25K22 and four force sensitive resistors. In this prototype, there are four sensor processing units. The main processing unit is an Intel Edison development board. The user interface unit is an Arduino Uno with a Gameduino.

The sensors are mounted with the adhesive that they come with onto a 14” X 14” piece of plywood. The PICs and related circuitry is implemented on breadboards and mounted on the other side of the plywood. A set of 2 X 2’s are used to frame the bottom side of the plywood to protect the breadboards and circuitry. Everything is wired with 22 gauge wiring since there aren’t any high power signals in the system. The Bill of Materials is provided below.

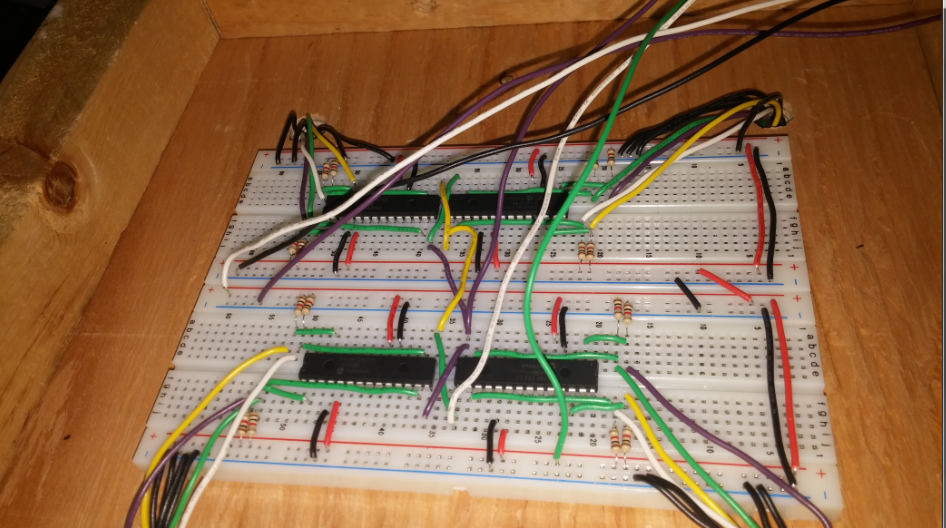


Figure 7.1 Photo of the FSR sensor

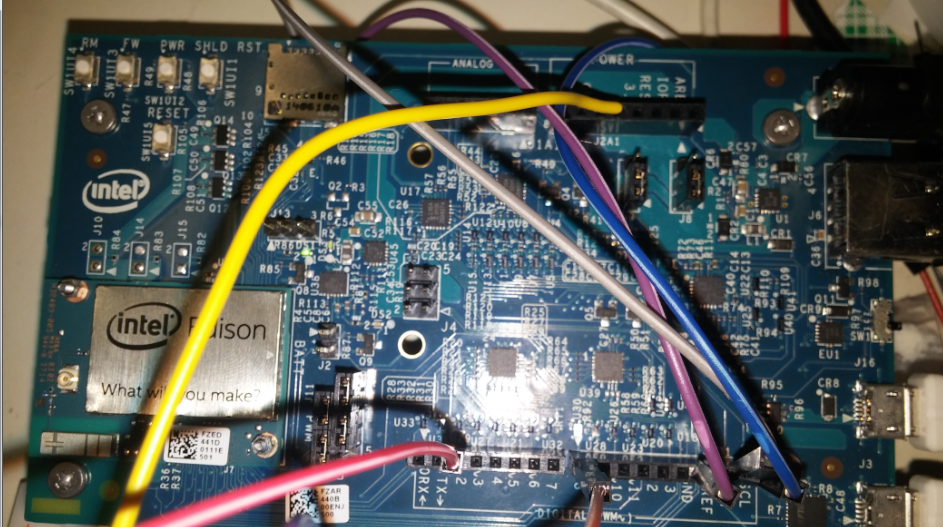
The figure above is showing one of the FSR (Force Sensitive Resistor) we used in this project. The resistance between two pins would vary from ~1M Ohm to 100 Ohm according to the force applied to it (0N to 100N). Although there is a big manufacturing error on this sensor (up to 25% between units), we can still get enough data for user’s sitting posture by comparing the data acquired by a sensor only to data from the same sensor.

   
Figure 7.2 4 x 4 sensor array

The figure above is showing the 4 x 4 sensor array we build for this project. FSR sensor units including the voltage dividers are ranged in 4 groups. Each handled by a PIC based subsystem. Every 4 sensors in a group are wired to the PIC MCU on the back of this board through the holes. For each FSR sensor, an rubber stopper is glue to the top of it in order to concentrate the force to the active area of the sensor, making sure that the board is not going to supporting user’s weight and prevent the sensors from getting enough force.

  
Figure 7.3 Back of the sensor-array board

All PIC-based subsystems are built and mounted at the back of the sensor-array board. Those subsystems are powered by the same power rail as the master node. And the I2C signals (SDA, SCL) of them are wired to the same bus connecting to master node.

  
Figure 7.4 Master system implemented on Intel Edison

The diagram above is showing the master node of the whole system on an Intel Edison board. It is doing all the computation and controls the LAN communication through I2C as the I2C master. It’s wired to the I2C bus and two LED warning lights.

  
Figure 7.5 Touchscreen subsystem

The figure above is showing the touchscreen subsystem implemented on an Arduino Uno board with a touchscreen SHIELD. This touchscreen subsystem takes in charge of user interfacing. User can control parameters, enable/disable MicroSD card logging, store good posture and monitor sensor-array reading. In addition, user can play a game with it. The IP address of the webserver interface is also displayed on this screen.

# 8. SOFTWARE IMPLEMENTATION

All of the software was written using Arduino flavor of C. Libraries provided by the PIC, the Intel Edison, the Gameduino, and the Arduino Uno, were used for enabling things like serial communication, building the touch screen interface, and building the website. The rest of the source code was written by all three team members, and can be found in the appendix.

*Pseudo code*

## (1)Master Node

Set-up;

set LED warning's pins to output mode;

begin serial with baud rate of 115200;

set MicroSD card adapter up;

set up I2C as master;

if(MicroSD card does not exists)

system stops;

Set the initial file name;

set Wifi webserver up;

until(wifi webserver is up and running)

{

try connected to designated wifi network;

print ssid and password on serial debug channel;

}

print webserver-ok info on serial debug channel;

wait 5 seconds and start the system;

Main-loop:

system timer control();

Every SYSTEM\_DELAY{

printSystemTime();

queryData();

updateStatus();

postureCheck();

printStatusToSerial();

}

Every LOGGING DELAY{

SDcardLogging();

}

warningControl();

wifiserver();

System Timer Control():

store the power-up timer in seconds in global variable;

queryData():

Until(polled from all sensor-subsystem)

{

set the I2C address of the sensor-subsystem to poll from;

Acquire all 8 bits x 8 data from I2C;

convert all data from 8bits style into 16byte long integers;

print debug info on serial channel;

}

UpdateStatus():

Set the I2C communication address to touchscreen subsystem;

if(web server parameter changed)

send new parameter to touchscreen unit for update;

else

get touchscreen UI's parameter values;

if parameter(setting posture) is set;

set the system flag to set posture;

postureCheck():

if(user is sitting the chair and not playing the game){

if(good posture data has been set already){

compare all sensor's value to stored value;

squared the differences;

if(sensor's difference is bigger than regular noises)

sum the differences together;

store the total error in global;

if(error is bigger than max tolerance){

if(wrong-posture timer not started yet)

start wrong-posture timer;

else if(wrong-posture timer hits max time){

print debug info on serial terminal;

set wrong-posture warning;

if(sitting timer has not started yet)

start sitting timer;

else{

print debug info about   
 sitting timer and current error;

if( sitting time is bigger

than designated time)

set sitting-too-long alarm;

}

}

}else{//not exceeding the error tolerance

clear wrong-posture warning;

reset and stop wrong-posture timer;

}

}

}

else if (user is sitting on chair and playing the game){

reset sitting timer and wrong-posture timer;

clear the stored posture data;

stop logging the posture data onto SD;

}else{//user is not sitting on the seat

reset sitting timer and wrong-posture timer;

stop SD logging;

clear wrong-posture warning and sitting-too-long warning;

}

printStatusToSerial():

print current error, wrong-posture timer, sitting timer

and all system control signals to serial debug terminal;

SDcardLogging():

if(SD logging is disabled){

if( "logN" file already exists)

set name to logN+1;

}else{ //logging enabled

open the file;

write power-up timer in seconds, error, all 16 sensor readings

to MicroSD card;

close the file to store the changes;;

}

warningControl():

if(sitting-too-long warning is set)

set sitting-too-long warning LED to turn on;

else

turn off the sitting-too-long warning LED;

if(wrong-posture warning is set)

set wrong-posture warning LED to turn on;

else

turn off the wrong-posture warning LED;

wifiserver():

if(there's a client request on HTTP server){

scan each line of the client's request:

if(current line is at the end of HTTP request header)

send standard HTTP respond header to client; //\*\*HTTP/1.1 200 OK

//\*\*Content-type:text/html

//(and two blank lines)

print the body of the HTML interface page

according to the current parameter of the system;

if(client hits a button on the web interface)

change parameters or |  
 set posture according to the button;

}

disconnect client's connection;

send debug info to serial terminal;

## (2)Touchscreen UI subsystem

Setup:

setup touchscreen;

setup I2C as slave;

setup serial debug terminal;

setup I2C interrupts;

main-loop:

switch(currentpage)

case 1: PageOne();

case 2: PageTwo();

case 3: PageThree();

PageOne(): //system control

draw the interface on the touchscreen accroding to previous received

paramaters from Master;

print the IP address of the Webserver;

PageTwo(): // the heat map

draw the heat map interface according to previous received

sensor data;

PageThree(): //the game

if(red is not generated yet)

generate a red dot on screen, randomly;

if(sensor data is showing a lean)

calculate where to lean;

move the blue dot

if(blue dot reaches dot)

erase red dot;

score+1;

generate a new red dot randomly;

if(reset button is pressed)

reset score;

erase red dot;

draw reset button;

I2C request Interrupt:

if(request parameters)

send parameters;

if(request whichpage)

send whichpage;

I2C receive Interrupt:

if(it's parameters)

receive and update parameters;

if(it's IP)

receive and update IP;

if(it's sensor data)

receive and update sensor data;

## (3) Sensor-subsystem

Setup:

set the systemclock up;

set 4 ADCs up;

set up I2C as slave;

Main-loop:

delay(30 instruction cycles);

get all 4 ADC values;

Store all 4 ADC values and keep the latest 30 sets of values;

I2C Request Interrupt:

Average latest 30 sets of 4 ADC values;

send average values of 4 ADCs;

# 9. HARDWARE FACTORY COST

|  |  |  |
| --- | --- | --- |
| **Item** | **Qty** | **Price/Qty** |
| Intel Edison Development Board | 1 | $80.00 |
| Gameduino | 1 | $70.00 |
| Arduino Uno | 1 | $25.00 |
| PIC18F25K22 | 4 | $2.50 |
| Force Sensitive Resistor (Round 0.5”) | 16 | $7.00 |
| 22 gauge wiring & jumper cables | 1 | $10.00 |
| Solder (100 gram) | 1 | $6.00 |
| Wood for sensors | 1 | FREE |
| Pack of rubber stoppers | 2 | 5.00 |
| Total | | $323.00 |

Table 9.1 Factory cost estimate

# 10. TEST PLAN

For the Sit-Smart system, we will need to test the following items:

* Posture feedback
* Posture control
* Posture logging
* Bad posture recognition

In addition to testing what the Sit-Smart should do, we will also need to test the user interface:

* Local interface
* Online interface
* Interface syncing

# 11. TEST SPECIFICATIONS

## Posture Feedback

Verify that the graphical representation of posture correctly represents the current posture of the user.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs.

Input/Output:

Input is touching the Gameduino touch screen interface and force onto the seat bottom. The output is the graphical representation of posture on the second screen provided by the UI.

Test Specification:  
Verify that each heat-dots light up when they should, and that multiple heat-dots can light up simultaneously.

## Posture Control

Verify that the user can control the game properly. Need to verify that up, down, left, and right movement works.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs.

Input/Output:

Input is touching the Gameduino touch screen interface and force onto the seat bottom. The output is the movement of the cursor in the game (third screen provided by the UI).

Test Specification:  
Verify that control via posture is possible. The user should be able to decide where to move the cursor and should be able to move the cursor with their posture.

## Posture Logging

Verify that posture logging works properly. Need to verify proper file opening and closing, and file name enumeration. Also need to verify proper formatting of output posture logs.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs. Use an initially empty SD card.

Input/Output:

Input is touching the Gameduino touch screen interface and force onto the seat bottom. The output is the posture logs written to the SD card.

Test Specification:

Verify file enumeration, and file content by opening multiple posture logs.

## Bad Posture Recognition

Verify that bad posture can be recognized. Need to verify that user inputs tau and epsilon properly function.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs.

Input/Output:

Input is touching the Gameduino touch screen interface and force onto the seat bottom. The output is bad posture warning.

Test Specification:

Verify that the three different tau values change the time required to set off the alarm. Verify that the three different epsilon values change the degree of badness required to set off the alarm.

## Local Interface

Verify that the user can interface and change system variables through the touch screen.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs. Enable the serial communication to a local PC.

Input/Output:

Input is touching the Gameduino touch screen interface and force onto the seat bottom. The output is the serial communication to the local PC.

Test Specification:

Verify user control through the local interface by making changes to the system variables through the touch screen, and monitoring the changes through the local PC.

## Online Interface

Verify that the user can interface and change system variables through the online interface.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs. Enable the serial communication to a local PC.

Input/Output:

Input is online interface and force onto the seat bottom. The output is serial communication to the local PC.

Test Specification:

Verify user control through the online interface by making changes to the system variables through the website, and monitoring the changes through the local PC.

## Interface Syncing

Verify that changes in the local interface are reflected in the online interface, and vice versa.

Test setup:

Initialize MPU and UI. Then supply power to the SPUs.

Input/Output:

Input is touching the Gameduino touch screen interface, the website, and force onto the seat bottom. The output is the status of the Gameduino touch screen interface, and the status of the website.

Test Specification:

Verify that the two user interfaces are synced together by making changes in one system and monitoring the changes in the other. Make sure that both changes in the local interface induce changes in the online interface, and that changes in the online interface induce changes in the local interface.

# 12. Test Cases

## Posture Feedback

First, using a finger, touch a single sensor without touching any of the other sensors and check to see that the corresponding heat-dot lights up. Repeat with the other 15 sensors. Next, using multiple fingers, touch random clusters of sensors and ensure that the corresponding cluster of heat-dots light up. Lastly, sit on the Sit-Smart and move up, down, left, and right.

## Posture Control

First, some definitions: the “front” of the seat is away from the “back of the seat”. The “left”, “right”, and “back” of the seat are defined in the obvious way. The “up”-ward direction of the touch screen is towards the top of the interface. The other directions are defined in the obvious way.

Using fingers, touch multiple sensors near the “front” and check to see that the cursor moves “up”. Then check that the “left” sensors cause the cursor to move “left”. Then check that the “right” sensors cause the cursor to move “right”. Then check that the “back” sensors cause the cursor to move “down”. Lastly, repeat by sitting on the Sit-Smart and leaning “forward”, “left”, “right”, and “backward”.

## Posture Logging

To verify file naming, content, etc., use Putty or similar software to access the Linux OS on the Intel Edison breakout board.

First, initiate a posture log and log no deviation from posture by not moving. Stop the posture log to verify that the file closes.

## Bad Posture Recognition

In this prototype, there are three tau values (3, 5, and 7 seconds), and three epsilon values (LOW, MEDIUM, HIGH). Instead of testing all nine combinations of these two control variables, we will simply test varying epsilon for the three second tau value, and then varying tau for the epsilon set to LOW.

|  |  |  |
| --- | --- | --- |
| Epsilon | Tau (sec) | Expected Result |
| Low | 3 | Slightly slouching for more than tau seconds should sound the alarm. |
| Medium | 3 | Slightly slouching will not set off the alarm. Slouching against the seat back for more than tau seconds should sound the alarm. |
| High | 3 | Only extreme slouching for more than tau seconds will sound the alarm. |
| Low | 5 | Slightly slouching for more than 5 seconds should sound the alarm. |
| Low | 7 | Slightly slouching for more than 7 seconds should sound the alarm. |

Figure 12.1 Test table for bad posture recognition

## Local Interface

For each of the following changes made on the local interface, check the serial communication on the local PC to verify that the system has actually changed the system variable. First, check all three values of epsilon. Then check all three values of tau. Then turn posture logging on, and then off. Then set posture.

## Online Interface

Repeat the test cases for the local interface, but on the website.

## Interface Syncing

For each of the following changes made on the local interface, refresh the website to verify that the change is reflected on the online interface. First, check all three values of epsilon. Then check all three values of tau. Then turn posture logging on, and then off. Then set posture. Repeat the steps, but make the changes on the online interface and verify that the changes are reflected on the local interface.

# **13. FAILURE MODES ANALYSIS**

## Failure analysis consequences:

|  |  |  |
| --- | --- | --- |
| Class | Title | Description |
| I | Catastrophic | Conditions that could result in death or permanent disabilities; complete destruction of equipment and facilities. |
| II | Critical | Conditions that could result in severe injury; major damage to equipment and facilities. |
| III | Moderate | Conditions that could result in minor injury needing more than minor first-aid; minor damage to equipment and facilities. |
| IV | Negligible | Conditions that could result in the need for minor first-aid treatment but no long-term effects to personal safety or health; damage to equipment and facilities slightly more than normal wear and tear. |

Table 13.1 Table of failure analysis consequences

## Failure analysis likelihood:

|  |  |
| --- | --- |
| Likelihood | Description |
| A | Likely to occur |
| B | Probably will occur |
| C | Could occur |
| D | Unlikely to occur |
| E | Improbable |

Table 13.2 Failure analysis likelihood table

## Failure analysis matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Likelihood estimate | | | | |
| Consequence | A | B | C | D | E |
| I | 1 | 1 | 2 | 3 | 4 |
| II | 1 | 2 | 3 | 4 | 5 |
| III | 2 | 3 | 4 | 5 | 6 |
| IV | 3 | 4 | 5 | 6 | 7 |

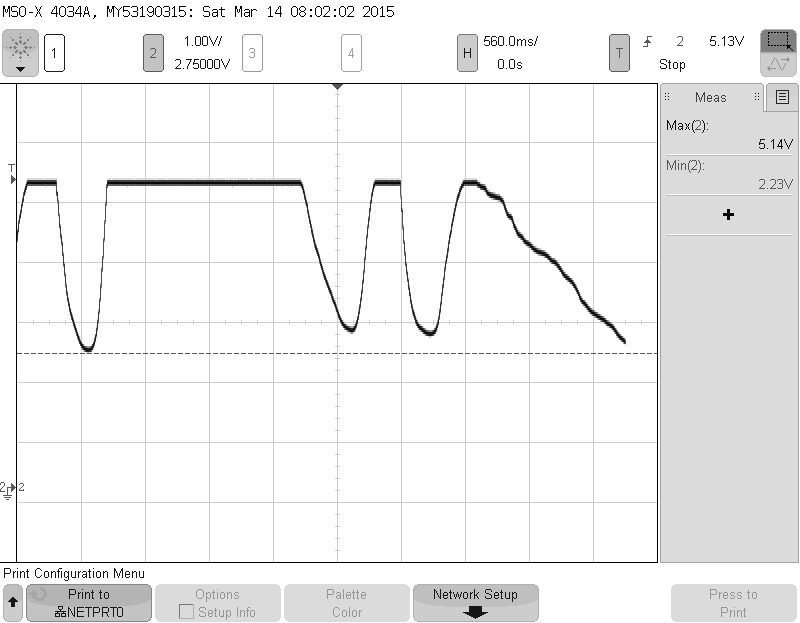
Table 13.3 Failure analysis matrix table

## Failure Analysis:

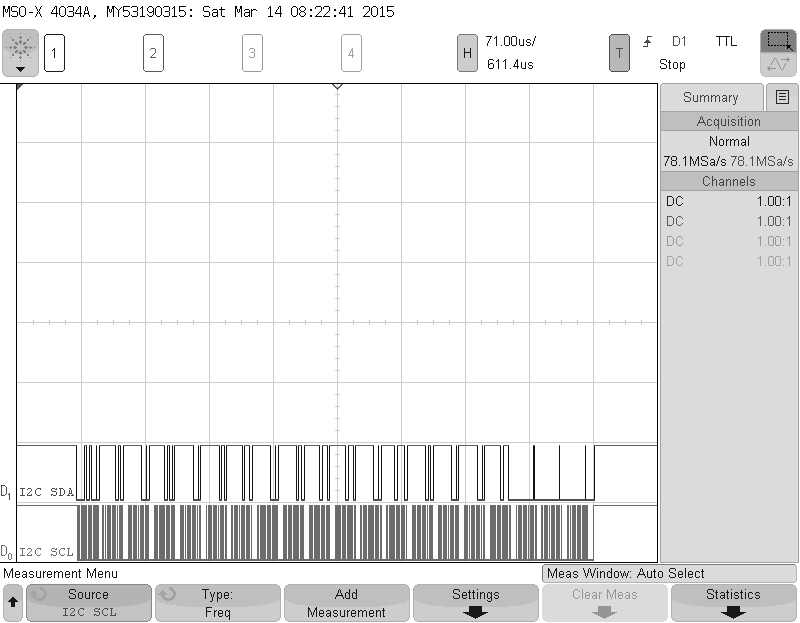
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Potential Failure | Consequence | Likelihood | Hazard ID | Comments |
| Sensor breaks | IV | C | 5 | We will be able to determine if a sensor breaks if it doesn’t change values when force is applied. This can be seen through the user interface. |
| PIC breaks | IV | C | 5 | By using the MPU to check the I2C values sent by the PICs, the MPU will be able to determine if the PICs are properly sending values. |
| MPU breaks | IV | E | 7 | The Edison and breakout board are quality pieces. We do not foresee any problems with them. |
| UI Breaks | IV | E | 7 | N/A |

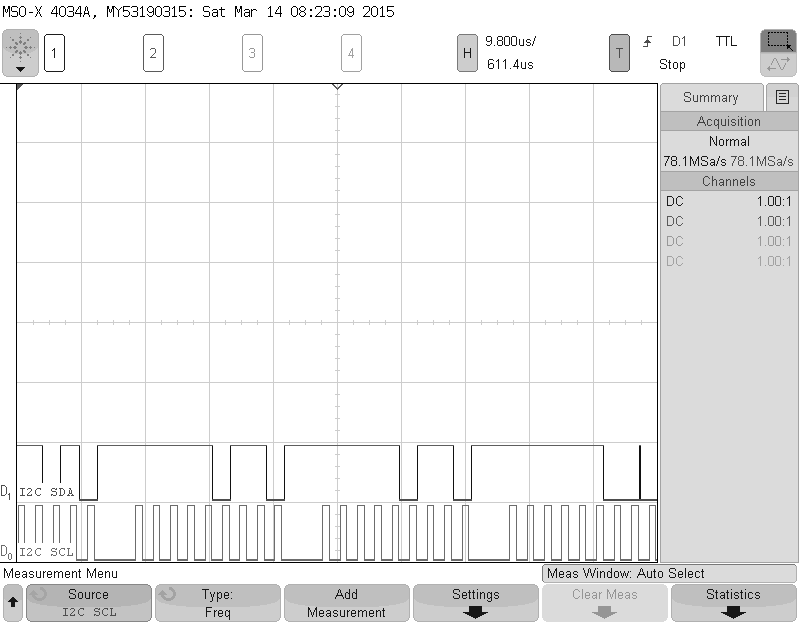
Table 13.4 Failure Analysis table

# **14. SYSTEM RESULT AND TIMING DIAGRAM**

 Figure 14.1 Analog voltage output for voltage-divider sensor unit

In our design, we used a voltage divider to read from the FSR (Force Sensitive Resistor). As explained previously, we are able to convert the resistance readings of FSR into voltage readings. The figure above is showing a test run to see the output range of our FSR-to-voltage unit. When using ~5V power VCC, we can get to a range from approximately 2.0V to 5.0V, which is good enough for ADC to get enough accurate info from this sensor unit.

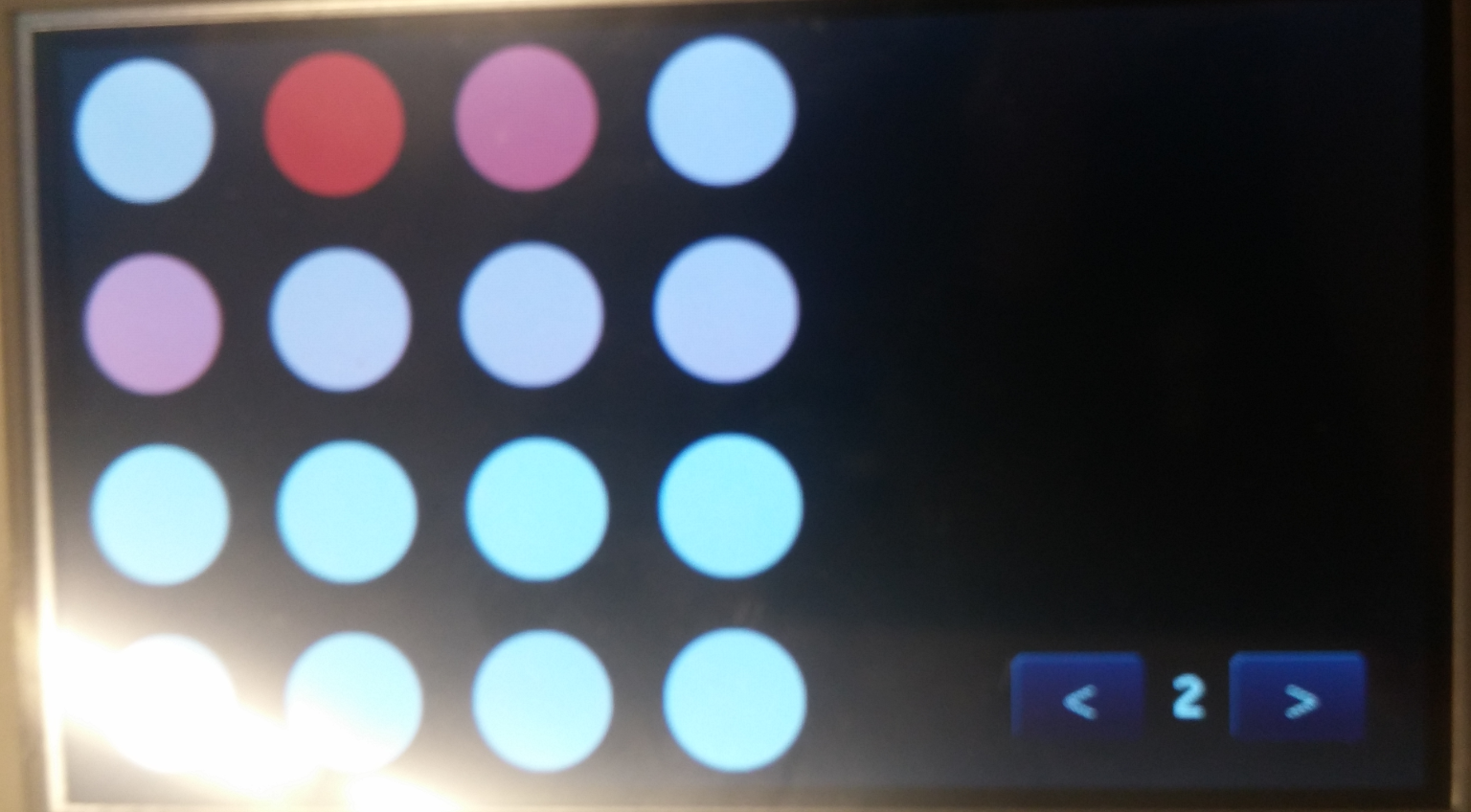
Figure 14.2 I2C waveform, showing communication readings from sensor subsystem

Figure 14.3 Zoomed view of working I2C wave form.

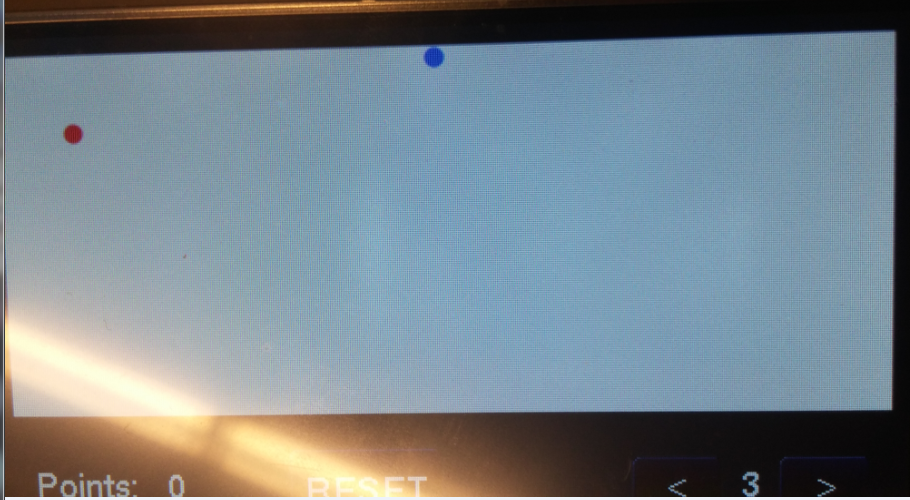
Our system is using I2C as the main protocol for the communication between all subsystems (slave) and master node. 4 sensor subsystems (PIC18F25K22 based) and touchscreen user interface subsystem (Arduino Uno based) are the slaves in this I2C inter-chip network while the master node (Intel Edison based) is playing the master in I2C. The figures above are showing the I2C communication signals’ waveform when master is polling sensor readings from all sensor subsystems. From the I2C SCL signal, you can tell it is sending 8 bits x 8 times and it’s from subsystem to master. The subsystem which is currently sending in the diagram above is sending the sensor-ADC data (16-bits) from 4 sensors to the master node.

  
Figure 14.4 Touchscreen UI control interface

The figure above is showing the display on touchscreen interface for system control. User can control the system sensitivity and time tolerance according to their needs. Also user can use this interface to enable sensor data logging on the MicroSD card with a sampling rate of 1 s. More importantly, user can set the ‘right posture’ they want to maintain on this touchscreen interface. The IP address of the webserver interface is also displayed on this screen.

  
Figure 14.5 Touchscreen UI showing sensor readings graphically

The figure above is the ‘heat map’ of sensor-array’s reading graphically. User can monitor their sitting posture monitored by the sensor-array graphically. The more pressure on a sensor, redder the point corresponding to that sensor would become. The color is ranged from white to dark red.

  
Figure 14.6 Sample game on our system

The diagram above shows the simple game we built on our system using user’s sitting posture as input. By leaning body left, right, from and back, the user can control the blue dot on the screen. The goal of this game is to get the blue dot to the red dot as many times as possible. Every single point acquired would be counted as 1 point in score and the UI would display the score and allow user to reset the score.

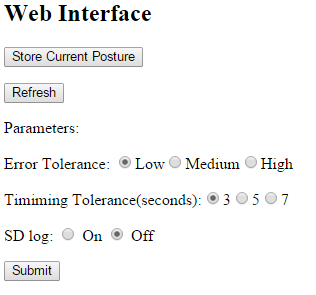
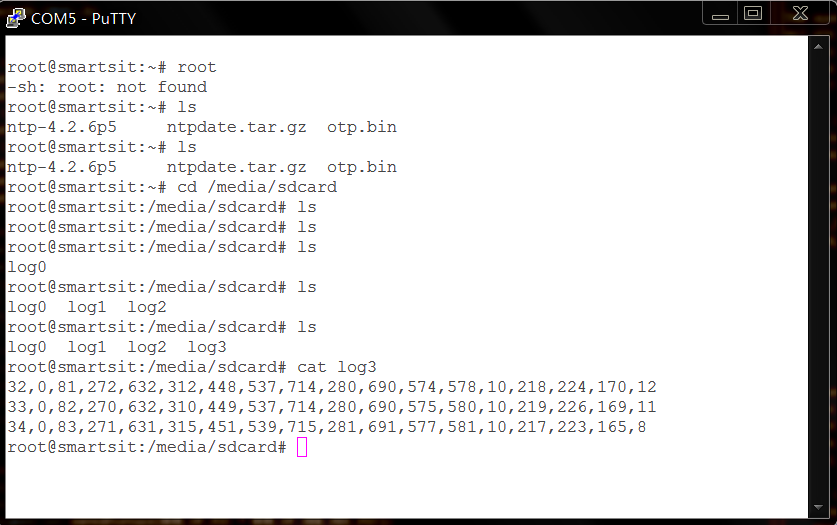


Figure 14.7 Web interface though WIFI

There’s a web interface implemented on the master node using the integrated WIFI module of Intel Edison. User can control the system’s parameters as same as the touchscreen UI. Also, touchscreen UI and web UI are synchronized together, which means they will changed the other one when one of them is changed.

  
Figure 14.7 Text data file log on MicroSD card

The data log system is implemented on the Intel Edison using the integrated MicroSD card reader. It can record the sensor data every single second. As shown in the screenshot above, the data is stored in this way: power-up time(s), current calculated error and all 16 sensors’ data. And every time you start a new logging process, it would write the a new file called ‘logXXX’; XXX is a number used to distinguish files from other log files.

# 15. SYSTEM ERROR ANALYSIS

There is one drawback of our system that is due to the low resolution of the 4x4 sensor grid that is intended to capture the force applied by the user when sitting on the chair. Due to the limitation of budget, we chose to design a 4x4 sensor grid instead of any larger dimension because we figured that 16 sensors placed in a grid are enough for us to demonstrate a sensor grid is able to record a good posture and reflect when a deviation from the recorded good posture occurs. However, a 4x4 dimension has a limit on how much deviation it can detect on the user. The more sensors we have in the grid, the smaller deviation the system can capture. The 4x4 system can sometimes miss a small slouch if the slouch is very subtle that the center of mass of the user does not move by a lot. This caused the system to miss a small slouch like this and thinks that the user is still sitting in a good posture. A 4x4 sensor grid is enough to cover a user slouch a way to improve the system performance to detect is to increase the number of sensor.

# 16. SUMMARY

This report has gone over the major steps that we took in designing this final project Sit-Smart whose goal is to help users with maintaining good sitting posture. All of the design documents including design spec, design procedure, system description, and hardware/software implementation are explained above. The testing section also explained what and how we tested our system to make sure that the system is fully working to meet all our design specifications. The result section shows the outputs of various subsystems to illustrate how they work. We also analyzed why an error exists that can occasionally cause the system to not be able to function ideally due to the low resolution of our sensor grid that captures the force applied on the chair.

# 17. CONCLUSION

Throughout the design, build, debugging, and verification stages of this project, we exercised our skills developed during the electrical engineering curriculum. In this project, we demonstrate our capacity for embedded system design by carefully designing a non-trivial system with an approach that enabled us to quickly debug and revise individual modules without affecting the entire system. In conclusion, we successfully completed what we sought out to accomplish. Our approach, using an array of force sensors, is only one way to measure posture. Other methods include using goniometers, or a variable capacitor array. Our approach, and the two alternative approaches have their pros and cons, and the appropriate approach to use will be application specific. For our project, the low-resolution array of force sensors seemed appropriate as we were only trying to demonstrate a proof-of-concept. While the low resolution ended up being one of the major crutches, it was much cheaper than the alternatives. The main recommendation is to improve the resolution by either using more sensors or using the variable capacitor array. This will greatly improve the ability to identify and differentiate different postures.

# APPENDIX A. CODE

## Master

Using Intel Edison IDE

#include <Wire.h>

#include <SD.h>

#include <SPI.h>

#include <WiFi.h>

// defined by hardware configuration

#define NUM\_OF\_PIC 4

#define NUM\_OF\_SEN\_PER\_PIC 4

// SD log interval 1s

#define SD\_INTERVAL 1000

// I2C between MPU and UI

#define UI\_address 10

#define MAX\_SIT\_TIME 5

#define SYSTEM\_DELAY 100

//==========================WIFI variables ===================================//

char ssid[] = "testrouter"; // your network SSID (name)

char pass[] = "smartsit"; // your network password

int port = 81;

int status = WL\_IDLE\_STATUS;

WiFiServer server(port);

int wifi\_tau = 1;

int wifi\_err = 1;

int wifi\_sd = 0;

int wifi\_set = 0; //reset to 0 in main //set posture

int wifi\_flag = 0; //reset to 0 in main //variable changed

//==========================Sensor values ===================================//

unsigned int rawSensorValues[NUM\_OF\_PIC][NUM\_OF\_SEN\_PER\_PIC];

unsigned int currentPosture[NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC];

unsigned int savedPosture[NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC];

//================Status variable, updated from UI============================//

// these variables will be used to communicated on I2C

byte tau;

byte epsilon;

byte logPosture;

byte savePosture;

//================User status booleans========================================//

// these are used only by master

boolean user\_set\_posture = false; // true when user presses SET on UI, false otherwise

boolean playing\_game = false; // true when user switches to game page on UI, false otherwise

boolean user\_sitting = false; // true when user is sitting, false otherwise

//=========================Error==============================================//

int error;

int toleranceExceeded;

unsigned long pretime = 0;

unsigned long pre\_sd\_time = 0;

//=========================Timing=============================================//

unsigned long curr\_millis;

unsigned long last\_millis;

unsigned long delta\_millis;

int system\_time; // increments once system starts

int timer1; // Bad posture alarm timer

int timer2; // Maximum sitting timer

//=======================LED variables========================================//

int led\_badPosture = 2; // timer1 led\_badPosture for sitting badly for too long

int warning\_badPosture = 0;

int led\_sittingLong = 8;

int warning\_sittingLong = 0;

//========================SD variables========================================//

char filename[100];

int filenum = 0;

//sd card flag

int card = 1;

//for SD reader

const int chipSelect = 4;

File dataFile;

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

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

//============================ Main program =============================//

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

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

void setup() {

// LED pins set up

pinMode(led\_badPosture, OUTPUT);

pinMode(led\_sittingLong, OUTPUT);

// I2C set up

Wire.begin();

// Serial comm set up

Serial.begin(115200);

// SD set up

pinMode(10, OUTPUT);

if (!SD.begin(chipSelect))

{

Serial.println("Card failed\_badPosture, or not present");

card = 0;

}

sprintf(filename,"log%d",filenum);

Serial.println("SD card initialized.");

// wifi set up

while ( status != WL\_CONNECTED)

{

Serial.print("Attempting to connect to Network named: ");

Serial.println(ssid); // print the network name (SSID);

status = WiFi.begin(ssid, pass);

// wait 5 seconds for connection:

}

Serial.println("Wifi Server Connected");

server.begin();

printWifiStatus();

Serial.println("System start in 5 secs");

delay(5000);

}

void loop() {

// System time

updateSystemTimer();

if (millis() - pretime >= SYSTEM\_DELAY)

{

pretime = millis();

// Print system time

printSystemTime();

// Get sensor values

queryData();

// Update status from either Wifi/screen interface, sync up master, wifi, and screen

updateStatus();

// CheckPosture, timer1/timer2

process();

// Print current status

printStatus();

}

//Record on SD card

if (millis() - pre\_sd\_time >= SD\_INTERVAL) {

pre\_sd\_time = millis();

SDcard();

}

// Give warnings if user sits for too long or in a bad posture for too long

warning();

// wifiserver() always run

wifiserver();

}

// Increment system timer every one second

void updateSystemTimer(void) {

if (curr\_millis == 0)

{

last\_millis = millis();

}

curr\_millis = millis();

delta\_millis = curr\_millis - last\_millis;

if (delta\_millis > 1000)

{

system\_time++;

curr\_millis = 0;

}

}

// Request values from the 16 sensors

// Results saved in rawSensorValues[4][4] and currentPosture[16]

void queryData() {

for(int i = 1; i<=NUM\_OF\_PIC;i++)

{

Wire.requestFrom(i,8);

for(int j = 0; j < 4; j++)

{

rawSensorValues[i-1][j] = Wire.read() \* 256;

rawSensorValues[i-1][j] += Wire.read();

}

}

// reorder the raw data into a new array for sending to UI

currentPosture[0] = rawSensorValues[2][2];

currentPosture[1] = rawSensorValues[2][0];

currentPosture[2] = rawSensorValues[3][2];

currentPosture[3] = rawSensorValues[3][0];

currentPosture[4] = rawSensorValues[2][3];

currentPosture[5] = rawSensorValues[2][1];

currentPosture[6] = rawSensorValues[3][3];

currentPosture[7] = rawSensorValues[3][1];

currentPosture[8] = rawSensorValues[0][2];

currentPosture[9] = rawSensorValues[0][0];

currentPosture[10] = rawSensorValues[1][2];

currentPosture[11] = rawSensorValues[1][0];

currentPosture[12] = rawSensorValues[0][3];

currentPosture[13] = rawSensorValues[0][1];

currentPosture[14] = rawSensorValues[1][3];

currentPosture[15] = rawSensorValues[1][1];

user\_sitting = isSeated();

//print out for debug

Serial.print(rawSensorValues[2][2], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[2][0], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[3][2], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[3][0], DEC);

Serial.println(" ");

Serial.print(rawSensorValues[2][3], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[2][1], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[3][3], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[3][1], DEC);

Serial.println(" ");

Serial.print(rawSensorValues[0][2], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[0][0], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[1][2], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[1][0], DEC);

Serial.println(" ");

Serial.print(rawSensorValues[0][3], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[0][1], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[1][3], DEC);

Serial.print(" ");

Serial.print(rawSensorValues[1][1], DEC);

Serial.println(" ");

Serial.println(" ");

}

// If Wifi has changed status, take them and sync up UI

void updateStatus() {

Wire.beginTransmission(UI\_address);

// If user has made changes on wifi interface, update them

if (wifi\_flag == 1)

{

Serial.println("got in wifi code");

// Load all wifi variables

tau = wifi\_tau;

epsilon = wifi\_err;

logPosture = wifi\_sd;

// Reset wifi variables changed flag

wifi\_flag = 0;

}

else

{

// Get status from UI

getFromUI();

// Update master's wifi variables so that they are reflected on the web page

if (tau == 0) {

wifi\_tau = 1;

} else if (tau == 50) {

wifi\_tau = 2;

} else if (tau == 100) {

wifi\_tau = 3;

}

if (epsilon == 0) {

wifi\_err = 1;

} else if (epsilon == 50) {

wifi\_err = 2;

} else if (epsilon ==100) {

wifi\_err = 3;

}

wifi\_sd = logPosture;

// At the time, keep track of if user has set new posture

// from the WiFi interface, if so, raise the SET flag

if (wifi\_set == 1) {

savePosture = 1;

wifi\_set = 0;

}

if (savePosture == 1) {

user\_set\_posture = true;

warning\_badPosture = 0;

}

}

sendToUI();

Wire.endTransmission();

}

void process(void) {

if ((user\_sitting == true && playing\_game == false))

{

if (user\_set\_posture == true) {

checkPosture();

}

if (timer2 == 0)

{

Serial.println("timer2 started");

timer2 = system\_time;

}

else

{

Serial.print("timer2 is now :");

Serial.println(system\_time - timer2);

if ((system\_time - timer2) > MAX\_SIT\_TIME) {

Serial.println("Hey, you have sit for too long, get up man!");

warning\_sittingLong = 1;

}

}

} else if (user\_sitting == true && playing\_game == true) {

timer1 = 0;

timer2 = 0;

user\_set\_posture = false;

warning\_sittingLong = 0;

logPosture = 0;

} else {

timer1 = 0;

timer2 = 0;

user\_set\_posture = false;

warning\_sittingLong = 0;

warning\_badPosture = 0;

}

}

// Send current posture data to the User Interface

void sendToUI()

{

byte data[NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC + 3];

for(int i = 0; i < NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC; i++)

data[i] = currentPosture[i] >> 2;

data[16] = tau;

data[17] = epsilon;

data[18] = logPosture;

//Wire.beginTransmission(UI\_address);

Wire.write(data, NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC + 3);

}

void getFromUI()

{

Wire.requestFrom(UI\_address,5);

epsilon = Wire.read();

tau = Wire.read();

logPosture = Wire.read();

savePosture = Wire.read();

playing\_game = Wire.read();

}

void checkPosture()

{

// Compute error

error = 0;

for (int i = 0; i < NUM\_OF\_PIC \* NUM\_OF\_SEN\_PER\_PIC; i++)

{

if (savePosture)

{

savedPosture[i] = currentPosture[i];

}

else

{

int difference = savedPosture[i] - currentPosture[i];

int abs\_diff = abs(difference);

if (abs\_diff >20)

error = error + abs\_diff;

}

}

// watch timer1 if a posture is set by the user

if (error > errorMap(epsilon)) {

if (timer1 == 0)

{

Serial.println("timer1 started");

timer1 = system\_time;

} else {

// debug info

Serial.print("timer 1 is : ");

Serial.println(system\_time-timer1);

if ((system\_time - timer1) > timeMap(tau)) {

warning\_badPosture = 1;

Serial.println("tolerance exceeded!");

}

}

} else if (timer1 != 0) {

timer1 = 0;

// debug info

Serial.println("timer1 reset");

warning\_badPosture = 0;

}

if (savePosture == 1)

savePosture = 0;

// debug printout

Serial.print("error is ");

Serial.println(error, DEC);

}

// returns how much error tolerance is based on epsilon

int errorMap(int eps)

{

switch(eps)

{

case 0:

return 225;

case 50:

return 700;

case 100:

return 1500;

}

}

// returns how many seconds before alarm is issued based on tau

int timeMap(int tau) {

switch(tau)

{

case 0:

return 3;

case 50:

return 5;

case 100:

return 7;

}

}

// returns true if user is sitting on the chair

// returns false otherwise

boolean isSeated(void) {

for (int i = 0; i < 16; i++) {

if (currentPosture[i] < 800)

return true;

}

return false;

}

void fileName()

{

if(!logPosture)

{

sprintf(filename,"log%d",filenum);

while(SD.exists(filename))

{

filenum++;

sprintf(filename,"log%d",filenum);

}

}

}

void SDcard()

{

fileName();

if(logPosture)

{

dataFile = SD.open(filename, FILE\_WRITE);

dataFile.print(millis()/1000,DEC);

dataFile.print(",");

dataFile.print(error, DEC);

int i = 0;

for(int i = 0 ; i< 16; i++)

{

dataFile.print(",");

dataFile.print(1023-currentPosture[i], DEC);

}

dataFile.println();

dataFile.close();

}

}

void wifiserver() {

WiFiClient client = server.available();

if (client) {

Serial.println("Web UI: new client");

String currentLine = "";

while (client.connected()) {

if (client.available()) {

char c = client.read();

Serial.write(c);

if (c == '\n') {

// if the current line is blank, you got two newline characters in a row.

// that's the end of the client HTTP request, so send a response:

if (currentLine.length() == 0) {

// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)

// and a content-type so the client knows what's coming, then a blank line:

client.println("HTTP/1.1 200 OK");

client.println("Content-type:text/html");

client.println();

// the content of the HTTP response follows the header:

client.println("<html>");

client.println("<head>");

client.println("<title>Sit-Smart</title>");

client.println("<meta http-equiv=\"refresh\" content=\"20; url=?\" />");

client.println("</head>");

client.println("<body>");

client.println("<h2>Web Interface</h2>");

client.println();

//store button

client.println("<p><a href=\"?set\"><input name=\"Set\" type=\"button\" value=\"Store Current Posture\" /></a></p>");

//refresh button

client.println("<p><a href=\"?\"><input name=\"Refresh\" type=\"button\" value=\"Refresh\" /></a></p>");

//tau/error tolerance input box

client.println("<form action=\"\"");

client.println("<p>Parameters:</p>");

client.print("<p>Error Tolerance: ");

int i;

for (i = 1; i <= 3; i++)

{

client.print("<input type=\"radio\" name=\"err\" value=");

client.print(i);

if(i == wifi\_err)

client.print(" checked =\"checked\"");

client.print(" />");

if(i == 1)

client.print("Low");

else if(i == 2)

client.print("Medium");

else if(i == 3)

client.print("High");

}

client.print("</p>");

client.print("<p><p>Timiming Tolerance(seconds):");

for (i = 1; i <= 3; i++)

{

client.print("<input type=\"radio\" name=\"tau\" value=");

client.print(i);

if(i == wifi\_tau)

client.print(" checked =\"checked\"");

client.print(" />");

if(i == 1)

client.print("3");

else if(i == 2)

client.print("5");

else if(i == 3)

client.print("7");

}

client.print("</p>");

// checked=\"checked\"

client.println("<p>SD log: <input type=\"radio\" name=\"sd\" value=1");

if(wifi\_sd)

client.println(" checked=\"checked\"");

client.println("> On <input type=\"radio\" name=\"sd\" value=0");

if(!wifi\_sd)

client.println(" checked=\"checked\"");

client.println(" > Off</p>");

//hidden tag

client.println("<input type=\"hidden\" name=\"para\" value=1 />");

//submit button

client.println("<p><input type=\"submit\" value=\"Submit\" /></p>");

//sscanf testing

int test1=0,test2=0,test3=0;

sscanf("a=1&b=2&c=3","a=%d&b=%d&c=%d",&test1,&test2,&test3);

client.println(test1);

client.println(test2);

client.println(test3);

//end tags

client.println("</form>");

client.println("</body>");

client.println("</html>");

// The HTTP response ends with another blank line:

client.println();

// break out of the while loop:

break;

}

else { // if you got a newline, then clear currentLine:

currentLine = "";

}

}

else if (c != '\r') { // if you got anything else but a carriage return character,

currentLine += c; // add it to the end of the currentLine

}

if (currentLine.endsWith("&para=1 HTTP/1.1")) {

//GET /?err=1&tau=1&sd=0&para=1 HTTP/1.1

//10 16 21

wifi\_flag = 1;

char s[100];

currentLine.toCharArray(s, 100);

sscanf(s,"GET /?err=%d&tau=%d&sd=%d&para=1 HTTP/1.1",&wifi\_err, &wifi\_tau, &wifi\_sd);

Serial.println("\n Updated:");

Serial.println(currentLine.charAt(10));

Serial.println(currentLine.charAt(16));

Serial.println(currentLine.charAt(21));

}

if (currentLine.endsWith("/?set HTTP/1.1")) {

wifi\_set = 1;

}

}

}

client.stop();

Serial.println("Web UI: client disonnected\n/////////////////////////////////////////////////");

}

}

void printWifiStatus() {

// print the SSID of the network you're attached to:

Serial.print("SSID: ");

Serial.println(WiFi.SSID());

// print your WiFi shield's IP address:

IPAddress ip = WiFi.localIP();

Serial.print("IP Address: ");

Serial.println(ip);

byte data[4] = {ip[0], ip[1], ip[2], ip[3]};

Wire.beginTransmission(UI\_address);

Wire.write(data, 4);

Wire.endTransmission();

// print the received signal strength:

long rssi = WiFi.RSSI();

Serial.print("signal strength (RSSI):");

Serial.print(rssi);

Serial.println(" dBm");

// print where to go in a browser:

Serial.print("To see this page in action, open a browser to http://");

Serial.print(ip);

Serial.print(":");

Serial.println(port);

}

void warning(void) {

if (warning\_badPosture == 1)

digitalWrite(led\_badPosture, HIGH);

else

digitalWrite(led\_badPosture, LOW);

if (warning\_sittingLong == 1)

digitalWrite(led\_sittingLong, HIGH);

else

digitalWrite(led\_sittingLong, LOW);

}

void printSystemTime(void) {

Serial.println();

Serial.println();

Serial.println();

Serial.println();

Serial.println();

Serial.println();

Serial.println();

Serial.print("The system has run : ");

Serial.print(system\_time);

Serial.println(" seconds");

}

void printStatus(void) {

Serial.print(" epsilon is ");

Serial.print(epsilon, DEC);

Serial.print(" ");

Serial.print(" wifi\_err is ");

Serial.println(wifi\_err);

Serial.print(" tau is ");

Serial.print(tau, DEC);

Serial.print(" ");

Serial.print("wifi\_tau is ");

Serial.println(wifi\_tau);

Serial.print(" logPosture is ");

Serial.print(logPosture, DEC);

Serial.print(" ");

Serial.print(" wifi\_sd is ");

Serial.println(wifi\_sd);

Serial.print("savePosture is ");

Serial.print(savePosture, DEC);

Serial.print(" ");

Serial.print("wifi\_set is ");

Serial.println(wifi\_set);

Serial.print(" wifi\_flag is ");

Serial.println(wifi\_flag);

Serial.print("playing\_game is ");

Serial.println(playing\_game, DEC);

Serial.println();

}

## Touchscreen UI sub-system

Using Arduino IDE

#include <EEPROM.h>

#include <SPI.h>

#include <GD2.h>

#include <Wire.h>

#define TAG\_SLIDER\_SENSITIVITY 101

#define TAG\_SLIDER\_TIME 102

#define TAG\_LOG\_BUTTON\_START 103

#define TAG\_LOG\_BUTTON\_STOP 107

#define TAG\_BUTTON\_SET 104

#define TAG\_BUTTON\_LEFT 105

#define TAG\_BUTTON\_RIGHT 106

#define RADIUS 25

#define GAME\_SENSITIVITY 200

#define TAG\_BUTTON\_RESET 110

uint16\_t slider\_value\_sensitivity;

uint16\_t slider\_value\_time;

uint16\_t currentPos\_sensitivity;

uint16\_t currentPos\_time;

uint16\_t toggle\_log;

uint16\_t toggle\_log\_position;

int busy;

const char \* pageNum = "1";

//=============status variables=====================//

byte sensitivity\_level = 0;

byte time\_tolerance = 0;

byte pos\_logging\_en = 0;

byte set\_posture = 0;

//===========pressure values from master============//

byte pressure[16];

int points;

char game\_buff[12];

int xval;

int yval;

int minval;

int colorValue;

int target\_x = 50;

int target\_y = 50;

int player\_x = 240;

int player\_y = 105;

int playingGame;

void setup()

{

GD.begin();

Serial.begin(9600);

Wire.begin(10);

Wire.onRequest(requestEvent);

Wire.onReceive(receiveEvent);

}

void loop() {

// put your main code here, to run repeatedly:

if (pageNum == "1")

pageOne();

if (pageNum == "2")

pageTwo();

if (pageNum == "3")

pageThree();

}

void pageOne()

{

GD.get\_inputs();

GD.Clear();

GD.ClearColorRGB(0x000000);

// draw text labels

GD.cmd\_text(10, 16, 21, 0, "Set Sensitivity: ");

GD.cmd\_text(10, 65, 21, 0, "Set Time Tolerance: ");

GD.cmd\_text(10, 130, 21, 0, "Posture Logging: ");

GD.cmd\_text(10, 200, 21, 0, "Set Posture: ");

// draw and track SENSITIVITY slider

GD.Tag(TAG\_SLIDER\_SENSITIVITY);

GD.cmd\_slider(170,22,270,10,0, currentPos\_sensitivity,65535);

GD.cmd\_track(170,22,270,10,TAG\_SLIDER\_SENSITIVITY);

// draw and track TIME slider

GD.Tag(TAG\_SLIDER\_TIME);

GD.cmd\_slider(170,71,270,10,0, currentPos\_time,65535);

GD.cmd\_track(170,71,270,10,TAG\_SLIDER\_TIME);

// draw SET button

GD.Tag(TAG\_BUTTON\_SET);

GD.cmd\_button(160, 200, 60, 45, 30, 0, "SET");

GD.cmd\_track(160, 200, 60, 45, TAG\_BUTTON\_SET);

// this is for starting posture log

GD.Tag(TAG\_LOG\_BUTTON\_START);

if (pos\_logging\_en)

GD.cmd\_button(160, 130, 60, 45, 30, OPT\_FLAT, "ON");

else

GD.cmd\_button(160, 130, 60, 45, 30, 0, "ON");

GD.cmd\_track(160, 130, 60, 45, TAG\_LOG\_BUTTON\_START);

GD.Tag(TAG\_LOG\_BUTTON\_STOP);

if (pos\_logging\_en)

GD.cmd\_button(230, 130, 60, 45, 30, 0, "OFF");

else

GD.cmd\_button(230, 130, 60, 45, 30, OPT\_FLAT, "OFF");

GD.cmd\_track(230, 130, 60, 45, TAG\_LOG\_BUTTON\_STOP);

// This is for changing pages

GD.cmd\_text(395, 228, 23, 0, pageNum);

GD.Tag(TAG\_BUTTON\_LEFT);

GD.cmd\_button(340, 225, 45, 30, 22, 0, "<");

GD.cmd\_track(340, 225, 45, 30, TAG\_BUTTON\_SET);

GD.Tag(TAG\_BUTTON\_RIGHT);

GD.cmd\_button(415, 225, 45, 30, 22, 0, ">");

GD.cmd\_track(415, 225, 45, 30, TAG\_BUTTON\_SET);

switch (GD.inputs.track\_tag & 0xff) {

case TAG\_SLIDER\_SENSITIVITY:

slider\_value\_sensitivity = GD.inputs.track\_val;

if ( slider\_value\_sensitivity > 65535/3 & slider\_value\_sensitivity < 65535\*2/3 & slider\_value\_sensitivity !=0) {

currentPos\_sensitivity = 65535/2;

sensitivity\_level = 50;

}

else if (slider\_value\_sensitivity <65535/3 & slider\_value\_sensitivity !=0) {

currentPos\_sensitivity = 0;

sensitivity\_level = 0;

}

else if (slider\_value\_sensitivity >65535\*2/3 & slider\_value\_sensitivity !=0) {

currentPos\_sensitivity = 65535;

sensitivity\_level = 100;

}

break;

case TAG\_SLIDER\_TIME:

slider\_value\_time = GD.inputs.track\_val;

if ( slider\_value\_time > 65535/3 & slider\_value\_time < 65535\*2/3 & slider\_value\_time !=0) {

currentPos\_time = 65535/2;

time\_tolerance = 50;

}

else if (slider\_value\_time <65535/3 & slider\_value\_time !=0) {

currentPos\_time = 0;

time\_tolerance = 0;

}

else if (slider\_value\_time >65535\*2/3 & slider\_value\_time !=0) {

currentPos\_time = 65535;

time\_tolerance = 100;

}

break;

case TAG\_BUTTON\_SET:

for (int i=0; i<5; i++) {

GD.cmd\_button(160, 200, 60, 45, 30, OPT\_FLAT, "SET");

}

set\_posture = 1;

break;

case TAG\_LOG\_BUTTON\_START:

for (int i=0; i<5; i++) {

GD.cmd\_button(160, 130, 60, 45, 30, OPT\_FLAT, "ON");

}

pos\_logging\_en = 1;

break;

case TAG\_LOG\_BUTTON\_STOP:

for (int i=0; i<5; i++) {

GD.cmd\_button(230, 130, 60, 45, 30, OPT\_FLAT, "OFF");

}

pos\_logging\_en = 0;

break;

case TAG\_BUTTON\_LEFT:

for (int i=0; i<5; i++) {

GD.cmd\_button(340, 225, 45, 30, 22, OPT\_FLAT, "<");

}

break;

case TAG\_BUTTON\_RIGHT:

if (!busy)

{

for (int i=0; i<5; i++)

{

GD.cmd\_button(415, 225, 45, 30, 22, OPT\_FLAT, ">");

}

pageNum = "2";

busy = 1;

}

break;

default:

busy = 0;

break;

}

//Serial.println(value);

GD.swap();

}

void pageTwo()

{

GD.get\_inputs();

GD.Clear();

GD.ClearColorRGB(0x000000);

// draw page number, left, right buttons

GD.cmd\_text(395, 228, 23, 0, pageNum);

GD.Tag(TAG\_BUTTON\_LEFT);

GD.cmd\_button(340, 225, 45, 30, 22, 0, "<");

GD.cmd\_track(340, 225, 45, 30, TAG\_BUTTON\_SET);

GD.Tag(TAG\_BUTTON\_RIGHT);

GD.cmd\_button(415, 225, 45, 30, 22, 0, ">");

GD.cmd\_track(415, 225, 45, 30, TAG\_BUTTON\_SET);

// draw the heat map

GD.Begin(POINTS);

GD.PointSize(16 \* RADIUS);

minval = 255;

for (int i = 0; i < 16; i++)

if (pressure[i] < minval) minval = pressure[i];

if (minval > 240)

minval = 0;

for (int i = 0; i < 16; i++)

{

xval = i % 4;

yval = i / 4;

colorValue = correctRange(minval, i);

GD.ColorRGB(0xff, colorValue, colorValue);

GD.Vertex2ii(30 + xval \* 70, 30 + yval \* 70);

}

/\*

GD.ColorRGB(0xff, GB\_values[0], GB\_values[0]);

GD.Vertex2ii(30,30);

GD.ColorRGB(0xff, GB\_values[4], GB\_values[4]);

GD.Vertex2ii(30,100);

GD.ColorRGB(0xff, GB\_values[8], GB\_values[8]);

GD.Vertex2ii(30,170);

GD.ColorRGB(0xff, GB\_values[12], GB\_values[12]);

GD.Vertex2ii(30,240);

GD.ColorRGB(0xff, GB\_values[1], GB\_values[1]);

GD.Vertex2ii(100,30);

GD.ColorRGB(0xff, GB\_values[5], GB\_values[5]);

GD.Vertex2ii(100,100);

GD.ColorRGB(0xff, GB\_values[9], GB\_values[9]);

GD.Vertex2ii(100,170);

GD.ColorRGB(0xff, GB\_values[13], GB\_values[13]);

GD.Vertex2ii(100,240);

GD.ColorRGB(0xff, GB\_values[2], GB\_values[2]);

GD.Vertex2ii(170,30);

GD.ColorRGB(0xff, GB\_values[6], GB\_values[6]);

GD.Vertex2ii(170,100);

GD.ColorRGB(0xff, GB\_values[10], GB\_values[10]);

GD.Vertex2ii(170,170);

GD.ColorRGB(0xff, GB\_values[14], GB\_values[14]);

GD.Vertex2ii(170,240);

GD.ColorRGB(0xff, GB\_values[3], GB\_values[3]);

GD.Vertex2ii(240,30);

GD.ColorRGB(0xff, GB\_values[7], GB\_values[7]);

GD.Vertex2ii(240,100);

GD.ColorRGB(0xff, GB\_values[11], GB\_values[11]);

GD.Vertex2ii(240,170);

GD.ColorRGB(0xff, GB\_values[15], GB\_values[15]);

GD.Vertex2ii(240,240);

\*/

switch (GD.inputs.track\_tag & 0xff) {

case TAG\_BUTTON\_LEFT:

if (!busy)

{

for (int i=0; i<5; i++) {

GD.cmd\_button(340, 225, 45, 30, 22, OPT\_FLAT, "<");

}

pageNum = "1";

busy = 1;

}

break;

case TAG\_BUTTON\_RIGHT:

if (!busy)

{

for (int i=0; i<5; i++) {

GD.cmd\_button(415, 225, 45, 30, 22, OPT\_FLAT, ">");

}

busy = 1;

pageNum = "3";

}

break;

default:

busy = 0;

break;

}

//Serial.println(value);

GD.swap();

}

void pageThree()

{

playingGame = 1;

GD.get\_inputs();

GD.Clear();

GD.ClearColorRGB(0x000000);

GD.Begin(RECTS);

GD.Vertex2ii(10, 10);

GD.Vertex2ii(470, 200);

// draw reset button

GD.Tag(TAG\_BUTTON\_RESET);

GD.cmd\_button(160, 220, 80, 45, 23, 0, "RESET");

GD.cmd\_track(160, 220, 80, 45, TAG\_BUTTON\_SET);

sprintf(game\_buff, "Points: %3d", points);

GD.cmd\_text(20, 228, 23, 0, game\_buff );

GD.cmd\_text(395, 228, 23, 0, pageNum);

GD.Tag(TAG\_BUTTON\_LEFT);

GD.cmd\_button(340, 225, 45, 30, 22, 0, "<");

GD.cmd\_track(340, 225, 45, 30, TAG\_BUTTON\_SET);

GD.Tag(TAG\_BUTTON\_RIGHT);

GD.cmd\_button(415, 225, 45, 30, 22, 0, ">");

GD.cmd\_track(415, 225, 45, 30, TAG\_BUTTON\_SET);

GD.Begin(POINTS);

GD.PointSize(5 \* 16);

// draw player

GD.ColorRGB(0,0,255);

GD.Vertex2ii(player\_x, player\_y);

// draw target

GD.ColorRGB(255,0,0);

GD.Vertex2ii(target\_x, target\_y);

GD.ColorRGB(255,255,255);

// update player

updatePlayerPosition();

// update target

if (abs(target\_x - player\_x) < 25 && abs(target\_y - player\_y) < 5)

{

target\_x = (int) random (15, 466);

target\_y = (int) random(15, 196);

points++;

}

switch (GD.inputs.track\_tag & 0xff) {

case TAG\_BUTTON\_RESET:

for (int i=0; i<5; i++) {

GD.cmd\_button(160, 220, 80, 45, 23, OPT\_FLAT, "RESET");

}

points = 0;

break;

case TAG\_BUTTON\_LEFT:

if (!busy)

{

for (int i=0; i<5; i++) {

GD.cmd\_button(340, 225, 45, 30, 22, OPT\_FLAT, "<");

}

pageNum = "2";

playingGame = 0;

busy = 1;

}

break;

case TAG\_BUTTON\_RIGHT:

for (int i=0; i<5; i++) {

GD.cmd\_button(415, 225, 45, 30, 22, OPT\_FLAT, ">");

}

break;

default:

busy = 0;

break;

}

//Serial.println(value);

GD.swap();

}

void updatePlayerPosition()

{

int forward = 0;

int backward = 0;

int left = 0;

int right = 0;

for (int i = 0; i < 8; i++)

{

forward += pressure[i];

if (i % 4 < 2)

left += pressure[i];

else

right += pressure[i];

}

for (int i = 8; i < 16; i++)

{

backward += pressure[i];

if (i % 4 < 2)

left += pressure[i];

else

right += pressure[i];

}

if (left - right > GAME\_SENSITIVITY)

player\_x ++;

else if (right - left > GAME\_SENSITIVITY)

player\_x --;

if (forward - backward > GAME\_SENSITIVITY)

player\_y ++;

else if (backward - forward > GAME\_SENSITIVITY)

player\_y --;

if (player\_x < 15)

player\_x = 15;

else if (player\_x > 465)

player\_x = 465;

if (player\_y < 15)

player\_y = 15;

else if (player\_y > 195)

player\_y = 195;

}

void requestEvent()

{

byte data[5] = {sensitivity\_level,time\_tolerance,pos\_logging\_en,set\_posture, playingGame};

Serial.println(sensitivity\_level);

Serial.println(time\_tolerance);

Serial.println(pos\_logging\_en);

Serial.println(set\_posture);

Serial.println(playingGame);

Serial.println();

Wire.write(data,5);

set\_posture = 0;

}

void receiveEvent(int howMany) {

int count;

/\*

tau = Wire.read();

etc...

\*/

for (int i = 0; i<16; i++) {

pressure[i] = Wire.read();

Serial.print(pressure[i]);

Serial.print(' ');

if ( (i+1)% 4 == 0)

Serial.println();

}

byte receivedTau = Wire.read();

if (receivedTau == 2) {

currentPos\_time = 65535/2;

time\_tolerance = 50;

} else if (receivedTau ==1) {

currentPos\_time = 0;

time\_tolerance = 0;

} else if (receivedTau == 3) {

currentPos\_time = 65535;

time\_tolerance = 100;

}

byte receivedEpsilon = Wire.read();

if (receivedEpsilon == 2) {

currentPos\_sensitivity = 65535/2;

sensitivity\_level = 50;

} else if (receivedEpsilon ==1) {

currentPos\_sensitivity = 0;

sensitivity\_level = 0;

} else if (receivedEpsilon == 3) {

currentPos\_sensitivity = 65535;

sensitivity\_level = 100;

}

byte received\_SDLog = Wire.read();

if (received\_SDLog == 1)

pos\_logging\_en = 1;

Serial.println();

}

int correctRange(int minimumValue, int index)

{

return (int) ((pressure[index] - minval) \* 255 / (255 - minval)),

}

## Sensor subsystem

Using MPLAB with XC8 complier

/\*\*

Generated Main Source File

Company:

Microchip Technology Inc.

File Name:

main.c

Summary:

This is the main file generated using MPLAB?Code Configurator

Description:

This header file provides implementations for driver APIs for all modules selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 2.00

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#include "mcc\_generated\_files/mcc.h"

#include "plib.h"

/\*

Main application

\*/

volatile uint16\_t sensorVal;

volatile int counter;

void main(void)

{

// Initialize the device

SYSTEM\_Initialize();

// If using interrupts in PIC18 High/Low Priority Mode you need to enable the Global High and Low Interrupts

// If using interrupts in PIC Mid-Range Compatibility Mode you need to enable the Global and Peripheral Interrupts

// Use the following macros to:

// Enable high priority global interrupts

//INTERRUPT\_GlobalInterruptHighEnable();

// Enable low priority global interrupts.

//INTERRUPT\_GlobalInterruptLowEnable();

// Disable high priority global interrupts

//INTERRUPT\_GlobalInterruptHighDisable();

// Disable low priority global interrupts.

//INTERRUPT\_GlobalInterruptLowDisable();

// Enable the Global Interrupts

INTERRUPT\_GlobalInterruptEnable();

// Enable the Peripheral Interrupts

INTERRUPT\_PeripheralInterruptEnable();

// Disable the Global Interrupts

//INTERRUPT\_GlobalInterruptDisable();

// Disable the Peripheral Interrupts

//INTERRUPT\_PeripheralInterruptDisable();

//sensorVal = 0x35;

counter = 0;

int count;

while (1)

{

record1[counter]=ADC\_GetConversion(channel\_AN0);

record2[counter]=ADC\_GetConversion(channel\_AN14);

record3[counter]=ADC\_GetConversion(channel\_AN13);

record4[counter]=ADC\_GetConversion(channel\_AN17);

counter ++;

if (counter == MAX)

counter = 0;

}

}

/\*\*

End of File

\*/

/\*\*

@Generated MPLAB?Code Configurator Header File

@Company:

Microchip Technology Inc.

@File Name:

mcc.h

@Summary:

This is the mcc.h file generated using MPLAB?Code Configurator

@Description:

This header file provides implementations for driver APIs for all modules selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Version : 1.02

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#ifndef MCC\_H

#define MCC\_H

#include <xc.h>

#include "pin\_manager.h"

#include <stdint.h>

#include <stdbool.h>

#include "interrupt\_manager.h"

#include "i2c1.h"

#include "adc.h"

#define \_XTAL\_FREQ 16000000

extern volatile uint16\_t sensorVal;

extern volatile int counter;

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes the device to the default states configured in the

\* MCC GUI

\* @Example

SYSTEM\_Initialize(void);

\*/

void SYSTEM\_Initialize(void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes the oscillator to the default states configured in the

\* MCC GUI

\* @Example

OSCILLATOR\_Initialize(void);

\*/

void OSCILLATOR\_Initialize(void);

#endif /\* MCC\_H \*/

/\*\*

End of File

\*/

/\*\*

@Generated MPLAB?Code Configurator Source File

@Company:

Microchip Technology Inc.

@File Name:

mcc.c

@Summary:

This is the mcc.c file generated using MPLAB?Code Configurator

@Description:

This header file provides implementations for driver APIs for all modules selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 1.02

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

// Configuration bits: selected in the GUI

// CONFIG1H

#pragma config IESO = OFF // Internal/External Oscillator Switchover bit->Oscillator Switchover mode disabled

#pragma config PLLCFG = OFF // 4X PLL Enable->Oscillator used directly

#pragma config PRICLKEN = OFF // Primary clock enable bit->Primary clock can be disabled by software

#pragma config FOSC = INTIO67 // Oscillator Selection bits->Internal oscillator block

#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit->Fail-Safe Clock Monitor disabled

// CONFIG2L

#pragma config BOREN = SBORDIS // Brown-out Reset Enable bits->Brown-out Reset enabled in hardware only (SBOREN is disabled)

#pragma config BORV = 190 // Brown Out Reset Voltage bits->VBOR set to 1.90 V nominal

#pragma config PWRTEN = OFF // Power-up Timer Enable bit->Power up timer disabled

// CONFIG2H

#pragma config WDTPS = 32768 // Watchdog Timer Postscale Select bits->1:32768

#pragma config WDTEN = OFF // Watchdog Timer Enable bits->Watch dog timer is always disabled. SWDTEN has no effect.

// CONFIG3H

#pragma config CCP2MX = PORTC1 // CCP2 MUX bit->CCP2 input/output is multiplexed with RC1

#pragma config P2BMX = PORTB5 // ECCP2 B output mux bit->P2B is on RB5

#pragma config HFOFST = ON // HFINTOSC Fast Start-up->HFINTOSC output and ready status are not delayed by the oscillator stable status

#pragma config PBADEN = ON // PORTB A/D Enable bit->PORTB<5:0> pins are configured as analog input channels on Reset

#pragma config CCP3MX = PORTB5 // P3A/CCP3 Mux bit->P3A/CCP3 input/output is multiplexed with RB5

#pragma config MCLRE = EXTMCLR // MCLR Pin Enable bit->MCLR pin enabled, RE3 input pin disabled

#pragma config T3CMX = PORTC0 // Timer3 Clock input mux bit->T3CKI is on RC0

// CONFIG4L

#pragma config LVP = OFF // Single-Supply ICSP Enable bit->Single-Supply ICSP disabled

#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit->Stack full/underflow will cause Reset

#pragma config XINST = OFF // Extended Instruction Set Enable bit->Instruction set extension and Indexed Addressing mode disabled (Legacy mode)

#pragma config DEBUG = OFF // Background Debug->Disabled

// CONFIG5L

#pragma config CP2 = OFF // Code Protection Block 2->Block 2 (004000-005FFFh) not code-protected

#pragma config CP1 = OFF // Code Protection Block 1->Block 1 (002000-003FFFh) not code-protected

#pragma config CP3 = OFF // Code Protection Block 3->Block 3 (006000-007FFFh) not code-protected

#pragma config CP0 = OFF // Code Protection Block 0->Block 0 (000800-001FFFh) not code-protected

// CONFIG5H

#pragma config CPB = OFF // Boot Block Code Protection bit->Boot block (000000-0007FFh) not code-protected

#pragma config CPD = OFF // Data EEPROM Code Protection bit->Data EEPROM not code-protected

// CONFIG6L

#pragma config WRT0 = OFF // Write Protection Block 0->Block 0 (000800-001FFFh) not write-protected

#pragma config WRT1 = OFF // Write Protection Block 1->Block 1 (002000-003FFFh) not write-protected

#pragma config WRT2 = OFF // Write Protection Block 2->Block 2 (004000-005FFFh) not write-protected

#pragma config WRT3 = OFF // Write Protection Block 3->Block 3 (006000-007FFFh) not write-protected

// CONFIG6H

#pragma config WRTC = OFF // Configuration Register Write Protection bit->Configuration registers (300000-3000FFh) not write-protected

#pragma config WRTD = OFF // Data EEPROM Write Protection bit->Data EEPROM not write-protected

#pragma config WRTB = OFF // Boot Block Write Protection bit->Boot Block (000000-0007FFh) not write-protected

// CONFIG7L

#pragma config EBTR3 = OFF // Table Read Protection Block 3->Block 3 (006000-007FFFh) not protected from table reads executed in other blocks

#pragma config EBTR1 = OFF // Table Read Protection Block 1->Block 1 (002000-003FFFh) not protected from table reads executed in other blocks

#pragma config EBTR2 = OFF // Table Read Protection Block 2->Block 2 (004000-005FFFh) not protected from table reads executed in other blocks

#pragma config EBTR0 = OFF // Table Read Protection Block 0->Block 0 (000800-001FFFh) not protected from table reads executed in other blocks

// CONFIG7H

#pragma config EBTRB = OFF // Boot Block Table Read Protection bit->Boot Block (000000-0007FFh) not protected from table reads executed in other blocks

#include "mcc.h"

void SYSTEM\_Initialize(void)

{

OSCILLATOR\_Initialize();

PIN\_MANAGER\_Initialize();

INTERRUPT\_Initialize();

I2C1\_Initialize();

ADC\_Initialize();

}

void OSCILLATOR\_Initialize(void)

{

// SCS INTOSC; IDLEN disabled; IRCF 16MHz\_HFINTOSC/4;

OSCCON = 0x72;

// PRISD enabled; MFIOSEL disabled; SOSCGO disabled;

OSCCON2 = 0x04;

// INTSRC disabled; PLLEN disabled; TUN 0x00;

OSCTUNE = 0x00;

// Set the secondary oscillator

}

/\*\*

End of File

\*/

/\*\*

@Generated Pin Manager Header File

@Company:

Microchip Technology Inc.

@File Name:

pin\_manager.h

@Summary:

This is the Pin Manager file generated using MPLAB?Code Configurator

@Description:

This header file provides implementations for pin APIs for all pins selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Version : 1.01

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#ifndef PIN\_MANAGER\_H

#define PIN\_MANAGER\_H

#define INPUT 1

#define OUTPUT 0

#define HIGH 1

#define LOW 0

#define ANALOG 1

#define DIGITAL 0

#define PULL\_UP\_ENABLED 1

#define PULL\_UP\_DISABLED 0

// get/set channel\_AN0 aliases

#define channel\_AN0\_TRIS TRISA0

#define channel\_AN0\_LAT LATA0

#define channel\_AN0\_PORT PORTAbits.RA0

#define channel\_AN0\_ANS ANSA0

#define channel\_AN0\_SetHigh() do { LATA0 = 1; } while(0)

#define channel\_AN0\_SetLow() do { LATA0 = 0; } while(0)

#define channel\_AN0\_Toggle() do { LATA0 = ~LATA0; } while(0)

#define channel\_AN0\_GetValue() PORTAbits.RA0

#define channel\_AN0\_SetDigitalInput() do { TRISA0 = 1; } while(0)

#define channel\_AN0\_SetDigitalOutput() do { TRISA0 = 0; } while(0)

#define channel\_AN0\_SetAnalogMode() do { ANSA0 = 1; } while(0)

#define channel\_AN0\_SetDigitalMode() do { ANSA0 = 0; } while(0)

// get/set VrefNeg aliases

#define VrefNeg\_TRIS TRISA2

#define VrefNeg\_LAT LATA2

#define VrefNeg\_PORT PORTAbits.RA2

#define VrefNeg\_ANS ANSA2

#define VrefNeg\_SetHigh() do { LATA2 = 1; } while(0)

#define VrefNeg\_SetLow() do { LATA2 = 0; } while(0)

#define VrefNeg\_Toggle() do { LATA2 = ~LATA2; } while(0)

#define VrefNeg\_GetValue() PORTAbits.RA2

#define VrefNeg\_SetDigitalInput() do { TRISA2 = 1; } while(0)

#define VrefNeg\_SetDigitalOutput() do { TRISA2 = 0; } while(0)

#define VrefNeg\_SetAnalogMode() do { ANSA2 = 1; } while(0)

#define VrefNeg\_SetDigitalMode() do { ANSA2 = 0; } while(0)

// get/set VrefPos aliases

#define VrefPos\_TRIS TRISA3

#define VrefPos\_LAT LATA3

#define VrefPos\_PORT PORTAbits.RA3

#define VrefPos\_ANS ANSA3

#define VrefPos\_SetHigh() do { LATA3 = 1; } while(0)

#define VrefPos\_SetLow() do { LATA3 = 0; } while(0)

#define VrefPos\_Toggle() do { LATA3 = ~LATA3; } while(0)

#define VrefPos\_GetValue() PORTAbits.RA3

#define VrefPos\_SetDigitalInput() do { TRISA3 = 1; } while(0)

#define VrefPos\_SetDigitalOutput() do { TRISA3 = 0; } while(0)

#define VrefPos\_SetAnalogMode() do { ANSA3 = 1; } while(0)

#define VrefPos\_SetDigitalMode() do { ANSA3 = 0; } while(0)

// get/set channel\_AN13 aliases

#define channel\_AN13\_TRIS TRISB5

#define channel\_AN13\_LAT LATB5

#define channel\_AN13\_PORT PORTBbits.RB5

#define channel\_AN13\_WPU WPUB5

#define channel\_AN13\_ANS ANSB5

#define channel\_AN13\_SetHigh() do { LATB5 = 1; } while(0)

#define channel\_AN13\_SetLow() do { LATB5 = 0; } while(0)

#define channel\_AN13\_Toggle() do { LATB5 = ~LATB5; } while(0)

#define channel\_AN13\_GetValue() PORTBbits.RB5

#define channel\_AN13\_SetDigitalInput() do { TRISB5 = 1; } while(0)

#define channel\_AN13\_SetDigitalOutput() do { TRISB5 = 0; } while(0)

#define channel\_AN13\_SetPullup() do { WPUB5 = 1; } while(0)

#define channel\_AN13\_ResetPullup() do { WPUB5 = 0; } while(0)

#define channel\_AN13\_SetAnalogMode() do { ANSB5 = 1; } while(0)

#define channel\_AN13\_SetDigitalMode() do { ANSB5 = 0; } while(0)

// get/set channel\_AN14 aliases

#define channel\_AN14\_TRIS TRISC2

#define channel\_AN14\_LAT LATC2

#define channel\_AN14\_PORT PORTCbits.RC2

#define channel\_AN14\_ANS ANSC2

#define channel\_AN14\_SetHigh() do { LATC2 = 1; } while(0)

#define channel\_AN14\_SetLow() do { LATC2 = 0; } while(0)

#define channel\_AN14\_Toggle() do { LATC2 = ~LATC2; } while(0)

#define channel\_AN14\_GetValue() PORTCbits.RC2

#define channel\_AN14\_SetDigitalInput() do { TRISC2 = 1; } while(0)

#define channel\_AN14\_SetDigitalOutput() do { TRISC2 = 0; } while(0)

#define channel\_AN14\_SetAnalogMode() do { ANSC2 = 1; } while(0)

#define channel\_AN14\_SetDigitalMode() do { ANSC2 = 0; } while(0)

// get/set SCL1 aliases

#define SCL1\_TRIS TRISC3

#define SCL1\_LAT LATC3

#define SCL1\_PORT PORTCbits.RC3

#define SCL1\_ANS ANSC3

#define SCL1\_SetHigh() do { LATC3 = 1; } while(0)

#define SCL1\_SetLow() do { LATC3 = 0; } while(0)

#define SCL1\_Toggle() do { LATC3 = ~LATC3; } while(0)

#define SCL1\_GetValue() PORTCbits.RC3

#define SCL1\_SetDigitalInput() do { TRISC3 = 1; } while(0)

#define SCL1\_SetDigitalOutput() do { TRISC3 = 0; } while(0)

#define SCL1\_SetAnalogMode() do { ANSC3 = 1; } while(0)

#define SCL1\_SetDigitalMode() do { ANSC3 = 0; } while(0)

// get/set SDA1 aliases

#define SDA1\_TRIS TRISC4

#define SDA1\_LAT LATC4

#define SDA1\_PORT PORTCbits.RC4

#define SDA1\_ANS ANSC4

#define SDA1\_SetHigh() do { LATC4 = 1; } while(0)

#define SDA1\_SetLow() do { LATC4 = 0; } while(0)

#define SDA1\_Toggle() do { LATC4 = ~LATC4; } while(0)

#define SDA1\_GetValue() PORTCbits.RC4

#define SDA1\_SetDigitalInput() do { TRISC4 = 1; } while(0)

#define SDA1\_SetDigitalOutput() do { TRISC4 = 0; } while(0)

#define SDA1\_SetAnalogMode() do { ANSC4 = 1; } while(0)

#define SDA1\_SetDigitalMode() do { ANSC4 = 0; } while(0)

// get/set channel\_AN17 aliases

#define channel\_AN17\_TRIS TRISC5

#define channel\_AN17\_LAT LATC5

#define channel\_AN17\_PORT PORTCbits.RC5

#define channel\_AN17\_ANS ANSC5

#define channel\_AN17\_SetHigh() do { LATC5 = 1; } while(0)

#define channel\_AN17\_SetLow() do { LATC5 = 0; } while(0)

#define channel\_AN17\_Toggle() do { LATC5 = ~LATC5; } while(0)

#define channel\_AN17\_GetValue() PORTCbits.RC5

#define channel\_AN17\_SetDigitalInput() do { TRISC5 = 1; } while(0)

#define channel\_AN17\_SetDigitalOutput() do { TRISC5 = 0; } while(0)

#define channel\_AN17\_SetAnalogMode() do { ANSC5 = 1; } while(0)

#define channel\_AN17\_SetDigitalMode() do { ANSC5 = 0; } while(0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

GPIO and peripheral I/O initialization

\* @Example

PIN\_MANAGER\_Initialize();

\*/

void PIN\_MANAGER\_Initialize (void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Interrupt on Change Handling routine

\* @Example

PIN\_MANAGER\_IOC();

\*/

void PIN\_MANAGER\_IOC(void);

#endif // PIN\_MANAGER\_H

/\*\*

End of File

\*/

/\*\*

Generated Pin Manager File

Company:

Microchip Technology Inc.

File Name:

pin\_manager.c

Summary:

This is the Pin Manager file generated using MPLAB?Code Configurator

Description:

This header file provides implementations for pin APIs for all pins selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 1.02

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#include <xc.h>

#include "pin\_manager.h"

void PIN\_MANAGER\_Initialize(void)

{

LATA = 0x00;

TRISA = 0xFF;

ANSELA = 0x2F;

LATB = 0x00;

TRISB = 0xFF;

ANSELB = 0x3F;

WPUB = 0x00;

LATC = 0x18;

TRISC = 0xFF;

ANSELC = 0xE4;

INTCON2bits.nRBPU = 0x01;

}

/\*\*

End of File

\*/

/\*\*

Generated Interrupt Manager Header File

@Company:

Microchip Technology Inc.

@File Name:

interrupt\_manager.h

@Summary:

This is the Interrupt Manager file generated using MPLAB?Code Configurator

@Description:

This header file provides implementations for global interrupt handling.

For individual peripheral handlers please see the peripheral driver for

all modules selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 1.01

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#ifndef INTERRUPT\_MANAGER\_H

#define INTERRUPT\_MANAGER\_H

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will enable global interrupts.

\* @Example

INTERRUPT\_GlobalInterruptEnable();

\*/

#define INTERRUPT\_GlobalInterruptEnable() (INTCONbits.GIE = 1)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will disable global interrupts.

\* @Example

INTERRUPT\_GlobalInterruptDisable();

\*/

#define INTERRUPT\_GlobalInterruptDisable() (INTCONbits.GIE = 0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will enable peripheral interrupts.

\* @Example

INTERRUPT\_PeripheralInterruptEnable();

\*/

#define INTERRUPT\_PeripheralInterruptEnable() (INTCONbits.PEIE = 1)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

This macro will disable peripheral interrupts.

\* @Example

INTERRUPT\_PeripheralInterruptDisable();

\*/

#define INTERRUPT\_PeripheralInterruptDisable() (INTCONbits.PEIE = 0)

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Initializes PIC18 peripheral interrupt priorities; enables/disables priority vectors

\* @Example

INTERRUPT\_Initialize();

\*/

void INTERRUPT\_Initialize (void);

/\*\*

\* @Param

none

\* @Returns

none

\* @Description

Interrupt service routine. Calls module interrupt handlers.

\* @Example

INTERRUPT\_InterruptManager();

\*/

void interrupt INTERRUPT\_InterruptManager (void);

#endif // INTERRUPT\_MANAGER\_H

/\*\*

End of File

\*/

/\*\*

@Generated Interrupt Manager File

@Company:

Microchip Technology Inc.

@File Name:

interrupt\_manager.c

@Summary:

This is the Interrupt Manager file generated using MPLAB?Code Configurator

@Description:

This header file provides implementations for global interrupt handling.

For individual peripheral handlers please see the peripheral driver for

all modules selected in the GUI.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 1.02

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#include "interrupt\_manager.h"

#include "mcc.h"

void INTERRUPT\_Initialize (void)

{

// Disable Interrupt Priority Vectors (16CXXX Compatibility Mode)

RCONbits.IPEN = 0;

// Clear peripheral interrupt priority bits (default reset value)

// SSPI

IPR1bits.SSP1IP = 0;

// ADI

IPR1bits.ADIP = 0;

}

void interrupt INTERRUPT\_InterruptManager (void)

{

// interrupt handler

if(PIE1bits.SSP1IE == 1 && PIR1bits.SSP1IF == 1)

{

I2C1\_ISR();

}

else if(PIE1bits.ADIE == 1 && PIR1bits.ADIF == 1)

{

ADC\_ISR();

}

else

{

//Unhandled Interrupt

}

}

/\*\*

End of File

\*/

/\*\*

ADC Generated Driver API Header File

@Company

Microchip Technology Inc.

@File Name

adc.h

@Summary

This is the generated header file for the ADC driver using MPLAB?Code Configurator

@Description

This header file provides APIs for driver for ADC.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 2.00

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#ifndef \_ADC\_H

#define \_ADC\_H

/\*\*

Section: Included Files

\*/

#include <xc.h>

#include <stdint.h>

#include <stdbool.h>

#ifdef \_\_cplusplus // Provide C++ Compatibility

extern "C" {

#endif

/\*\*

Section: Data Types Definitions

\*/

/\*\*

\* result size of an A/D conversion

\*/

typedef uint16\_t adc\_result\_t;

/\*\* ADC Channel Definition

@Summary

Defines the channels available for conversion.

@Description

This routine defines the channels that are available for the module to use.

Remarks:

None

\*/

typedef enum

{

channel\_CTMU = 0x1D,

channel\_DAC = 0x1E,

channel\_FVRBuf2 = 0x1F,

channel\_AN0 = 0x00,

channel\_AN13 = 0x0D,

channel\_AN14 = 0x0E,

channel\_AN17 = 0x11

} adc\_channel\_t;

/\*\*

Section: ADC Module APIs

\*/

/\*\*

@Summary

Initializes the ADC

@Description

This routine initializes the Initializes the ADC.

This routine must be called before any other ADC routine is called.

This routine should only be called once during system initialization.

@Preconditions

None

@Param

None

@Returns

None

@Comment

@Example

<code>

uint16\_t convertedValue;

ADC\_Initialize();

convertedValue = ADC\_GetConversionResult();

</code>

\*/

void ADC\_Initialize(void);

/\*\*

@Summary

Allows selection of a channel for conversion

@Description

This routine is used to select desired channel for conversion.

available

@Preconditions

ADC\_Initialize() function should have been called before calling this function.

@Returns

None

@Param

Pass in required channel number

"For available channel refer to enum under adc.h file"

@Example

<code>

uint16\_t convertedValue;

ADC\_Initialize();

ADC\_StartConversion(AN1\_Channel);

convertedValue = ADC\_GetConversionResult();

</code>

\*/

void ADC\_StartConversion(adc\_channel\_t channel);

/\*\*

@Summary

Returns true when the conversion is completed otherwise false.

@Description

This routine is used to determine if conversion is completed.

When conversion is complete routine returns true. It returns false otherwise.

@Preconditions

ADC\_Initialize() and ADC\_StartConversion(adc\_channel\_t channel)

function should have been called before calling this function.

@Returns

true - If conversion is complete

false - If conversion is not completed

@Param

None

@Example

<code>

uint16\_t convertedValue;

ADC\_Initialize();

ADC\_StartConversion(AN1\_Channel);

while(!ADC\_IsConversionDone());

convertedValue = ADC\_GetConversionResult();

</code>

\*/

bool ADC\_IsConversionDone();

/\*\*

@Summary

Returns the ADC conversion value.

@Description

This routine is used to get the analog to digital converted value. This

routine gets converted values from the channel specified.

@Preconditions

This routine returns the conversion value only after the conversion is complete.

Completion status can be checked using

ADC\_IsConversionDone() routine.

@Returns

Returns the converted value.

@Param

None

@Example

<code>

uint16\_t convertedValue;

ADC\_Initialize();

ADC\_StartConversion(AN1\_Channel);

while(ADC\_IsConversionDone());

convertedValue = ADC\_GetConversionResult();

</code>

\*/

adc\_result\_t ADC\_GetConversionResult(void);

/\*\*

@Summary

Returns the ADC conversion value

also allows selection of a channel for conversion.

@Description

This routine is used to select desired channel for conversion

and to get the analog to digital converted value.

@Preconditions

ADC\_Initialize() function should have been called before calling this function.

@Returns

Returns the converted value.

@Param

Pass in required channel number.

"For available channel refer to enum under adc.h file"

@Example

<code>

uint16\_t convertedValue;

ADC\_Initialize();

conversion = ADC\_GetConversion(AN1\_Channel);

</code>

\*/

adc\_result\_t ADC\_GetConversion(adc\_channel\_t channel);

/\*\*

@Summary

Implements ISR

@Description

This routine is used to implement the ISR for the interrupt-driven

implementations.

@Returns

None

@Param

None

\*/

void ADC\_ISR(void);

#ifdef \_\_cplusplus // Provide C++ Compatibility

}

#endif

#endif //\_ADC\_H

/\*\*

End of File

\*/

/\*\*

ADC Generated Driver File

@Company

Microchip Technology Inc.

@File Name

adc.c

@Summary

This is the generated driver implementation file for the ADC driver using MPLAB?Code Configurator

@Description

This source file provides implementations for driver APIs for ADC.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 2.00

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

/\*\*

Section: Included Files

\*/

#include <xc.h>

#include "adc.h"

/\*\*

Section: ADC Module APIs

\*/

void ADC\_Initialize(void)

{

// set the ADC to the options selected in the User Interface

// GO\_nDONE stop; ADON enabled; CHS AN0;

ADCON0 = 0x01;

// PVCFG external; TRIGSEL CTMU; NVCFG external;

ADCON1 = 0x85;

// ADFM right; ADCS FOSC/32; ACQT 0;

ADCON2 = 0x82;

// ADRESL 0x0;

ADRESL = 0x00;

// ADRESH 0x0;

ADRESH = 0x00;

// Enabling ADC interrupt.

PIE1bits.ADIE = 1;

}

void ADC\_StartConversion(adc\_channel\_t channel)

{

// select the A/D channel

ADCON0bits.CHS = channel;

// Turn on the ADC module

ADCON0bits.ADON = 1;

// Start the conversion

ADCON0bits.GO\_nDONE = 1;

}

bool ADC\_IsConversionDone()

{

// Start the conversion

return (!ADCON0bits.GO\_nDONE);

}

adc\_result\_t ADC\_GetConversionResult(void)

{

// Conversion finished, return the result

return ((ADRESH << 8) + ADRESL);

}

adc\_result\_t ADC\_GetConversion(adc\_channel\_t channel)

{

// Select the A/D channel

ADCON0bits.CHS = channel;

// Turn on the ADC module

ADCON0bits.ADON = 1;

// Start the conversion

ADCON0bits.GO\_nDONE = 1;

// Wait for the conversion to finish

while (ADCON0bits.GO\_nDONE)

{

}

// Conversion finished, return the result

return ((ADRESH << 8) + ADRESL);

}

void ADC\_ISR(void)

{

// Clear the ADC interrupt flag

PIR1bits.ADIF = 0;

}

/\*\*

End of File

\*/

/\*\*

MSSP1 Generated Driver API Header File

@Company

Microchip Technology Inc.

@File Name

i2c1.h

@Summary

This is the generated header file for the MSSP1 driver using MPLAB?Code Configurator

@Description

This header file provides APIs for driver for MSSP1.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 2.00

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#ifndef \_I2C1\_H

#define \_I2C1\_H

#include <stdint.h>

#include <stdbool.h>

#include <stddef.h>

#include <xc.h>

#ifdef \_\_cplusplus // Provide C++ Compatibility

extern "C" {

#endif

/\*\*

I2C Slave Driver Status

@Summary

Defines the different status that the slave driver has

detected over the i2c bus.

@Description

This defines the different status that the slave driver has

detected over the i2c bus. The status is passed to the

I2C1\_StatusCallback() callback function that is implemented by

the user of the slave driver as a parameter to inform the user

that there was a change in the status of the driver due to

transactions on the i2c bus. User of the slave driver can use these

to manage the read or write buffers.

\*/

typedef enum

{

I2C1\_SLAVE\_WRITE\_REQUEST,

I2C1\_SLAVE\_READ\_REQUEST,

I2C1\_SLAVE\_WRITE\_COMPLETED,

I2C1\_SLAVE\_READ\_COMPLETED,

I2C1\_SLAVE\_GENERAL\_CALL\_REQUEST,

} I2C1\_SLAVE\_DRIVER\_STATUS;

#define I2C1\_SLAVE\_DEFAULT\_ADDRESS 1

/\*\*

@Summary

Initializes and enables the i2c slave instance : 1

@Description

This routine initializes the i2c slave driver instance for : 1

index, making it ready for clients to open and use it.

@Preconditions

None

@Param

None

@Returns

None

@Example

<code>

// initialize the i2c slave driver

I2C1\_Initialize();

</code>

\*/

void I2C1\_Initialize(void);

/\*\*

@Summary

This function process the I2C interrupts generated by

bus activity

@Description

This function calls a callback function with 1 of 4

possible parameters.

I2C1\_SLAVE\_WRITE\_REQUEST

I2C1\_SLAVE\_READ\_REQUEST

I2C1\_SLAVE\_WRITE\_COMPLETED

I2C1\_SLAVE\_READ\_COMPLETED

The callback function should contain application specific

code to process I2C bus activity from the I2C master.

A basic EEPROM emulator is provided as an example.

\*/

void I2C1\_ISR (void);

/\*\*

@Summary

This varible contains the last data written to the I2C slave

\*/

extern volatile uint8\_t I2C1\_slaveWriteData;

#define MAX 30

uint16\_t record1[MAX];

uint16\_t record2[MAX];

uint16\_t record3[MAX];

uint16\_t record4[MAX];

#ifdef \_\_cplusplus // Provide C++ Compatibility

}

#endif

#endif // \_I2C1\_H

/\*\*

MSSP1 Generated Driver File

@Company

Microchip Technology Inc.

@File Name

i2c1.c

@Summary

This is the generated header file for the MSSP1 driver using

MPLAB?Code Configurator

@Description

This header file provides APIs for driver for MSSP1.

Generation Information :

Product Revision : MPLAB?Code Configurator - v2.10

Device : PIC18F25K22

Driver Version : 2.00

The generated drivers are tested against the following:

Compiler : XC8 v1.33

MPLAB : MPLAB X 2.26

\*/

/\*

Copyright (c) 2013 - 2014 released Microchip Technology Inc. All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute

Software only when embedded on a Microchip microcontroller or digital signal

controller that is integrated into your product or third party product

(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for

additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,

EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER

CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR

OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES

INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR

CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF

SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

\*/

#include "mcc.h"

#define I2C1\_SLAVE\_ADDRESS 1

typedef enum

{

SLAVE\_NORMAL\_DATA,

SLAVE\_DATA\_ADDRESS,

SLAVE\_GENERAL\_CALL,

} SLAVE\_WRITE\_DATA\_TYPE;

/\*\*

Section: Global Variables

\*/

volatile uint8\_t I2C1\_slaveWriteData = 0x55;

int count = 0;

uint8\_t upperVal;

uint8\_t lowerVal;

uint16\_t sum;

/\*\*

Section: Local Functions

\*/

void I2C1\_StatusCallback(I2C1\_SLAVE\_DRIVER\_STATUS i2c\_bus\_state);

uint16\_t getAverage(uint16\_t \*record);

/\*\*

Prototype: void I2C1\_Initialize(void)

Input: none

Output: none

Description: I2C1\_Initialize is an

initialization routine that takes inputs from the GUI.

Comment:

Usage: I2C1\_Initialize();

\*/

void I2C1\_Initialize(void)

{

// initialize the hardware

// BF RCinprocess\_TXcomplete; UA dontupdate; SMP Sample At Middle; P stopbit\_notdetected; S startbit\_notdetected; R\_nW write\_noTX; CKE Idle to Active; D\_nA lastbyte\_address;

SSP1STAT = 0x00;

// SSPEN enabled; WCOL no\_collision; SSPOV no\_overflow; CKP Idle:Low, Active:High; SSPM 7 Bit;

SSP1CON1 = 0x26;

// ACKSTAT received; RCEN disabled; RSEN disabled; ACKEN disabled; SEN disabled; ACKDT acknowledge; GCEN enabled; PEN disabled;

SSP1CON2 = 0x80;

// BOEN disabled; AHEN disabled; SBCDE disabled; SDAHT 100ns; DHEN disabled; ACKTIM ackseq; PCIE disabled; SCIE disabled;

SSP1CON3 = 0x00;

// MSK0 127;

SSP1MSK = (0x7F << 1); // adjust UI mask for R/nW bit

// SSP1ADD 1;

SSP1ADD = (I2C1\_SLAVE\_ADDRESS << 1); // adjust UI address for R/nW bit

// clear the slave interrupt flag

PIR1bits.SSP1IF = 0;

// enable the master interrupt

PIE1bits.SSP1IE = 1;

}

void I2C1\_ISR ( void )

{

uint8\_t i2c\_data = 0x55;

// NOTE: The slave driver will always acknowledge

// any address match.

PIR1bits.SSP1IF = 0; // clear the slave interrupt flag

i2c\_data = SSP1BUF; // read SSPBUF to clear BF

if(1 == SSP1STATbits.R\_nW)

{

if((1 == SSP1STATbits.D\_nA) && (1 == SSP1CON2bits.ACKSTAT))

{

// callback routine can perform any post-read processing

I2C1\_StatusCallback(I2C1\_SLAVE\_READ\_COMPLETED);

}

else

{

// callback routine should write data into SSPBUF

I2C1\_StatusCallback(I2C1\_SLAVE\_READ\_REQUEST);

}

}

else if(0 == SSP1STATbits.D\_nA)

{

// this is an I2C address

if(0x00 == i2c\_data)

{

// this is the General Call address

I2C1\_StatusCallback(I2C1\_SLAVE\_GENERAL\_CALL\_REQUEST);

}

else

{

// callback routine should prepare to receive data from the master

I2C1\_StatusCallback(I2C1\_SLAVE\_WRITE\_REQUEST);

}

}

else

{

I2C1\_slaveWriteData = i2c\_data;

// callback routine should process I2C1\_slaveWriteData from the master

I2C1\_StatusCallback(I2C1\_SLAVE\_WRITE\_COMPLETED);

}

SSP1CON1bits.CKP = 1; // release SCL

} // end I2C1\_ISR()

/\*\*

Example implementation of the callback

This slave driver emulates an EEPROM Device.

Sequential reads from the EEPROM will return data at the next

EEPROM address.

Random access reads can be performed by writing a single byte

EEPROM address, followed by 1 or more reads.

Random access writes can be performed by writing a single byte

EEPROM address, followed by 1 or more writes.

Every read or write will increment the internal EEPROM address.

When the end of the EEPROM is reached, the EEPROM address will

continue from the start of the EEPROM.

\*/

void I2C1\_StatusCallback(I2C1\_SLAVE\_DRIVER\_STATUS i2c\_bus\_state)

{

static uint8\_t EEPROM\_Buffer[] =

{

0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,

0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f,

0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f,

0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,

0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,

0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,

0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,

0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,

};

static uint8\_t eepromAddress = 0;

static uint8\_t slaveWriteType = SLAVE\_NORMAL\_DATA;

switch (i2c\_bus\_state)

{

case I2C1\_SLAVE\_WRITE\_REQUEST:

// the master will be sending the eeprom address next

slaveWriteType = SLAVE\_DATA\_ADDRESS;

break;

case I2C1\_SLAVE\_GENERAL\_CALL\_REQUEST:

// the master will be sending general call data next

slaveWriteType = SLAVE\_GENERAL\_CALL;

break;

case I2C1\_SLAVE\_WRITE\_COMPLETED:

switch(slaveWriteType)

{

case SLAVE\_DATA\_ADDRESS:

eepromAddress = I2C1\_slaveWriteData;

break;

case SLAVE\_GENERAL\_CALL:

// process general call data here

break;

case SLAVE\_NORMAL\_DATA:

default:

// the master has written data to store in the eeprom

EEPROM\_Buffer[eepromAddress++] = I2C1\_slaveWriteData;

if(sizeof(EEPROM\_Buffer) <= eepromAddress)

{

eepromAddress = 0; // wrap to start of eeprom page

}

break;

} // end switch(slaveWriteType)

slaveWriteType = SLAVE\_NORMAL\_DATA;

break;

case I2C1\_SLAVE\_READ\_REQUEST:

if (count == 0) { // 0 send upper byte of adc 1

sum = getAverage(record1);

sensorVal = (uint16\_t) (sum / MAX);

upperVal = (sensorVal >> 8) & 0xff;

lowerVal = (sensorVal) & 0x00ff;

SSP1BUF = upperVal;

count = 1;

} else if (count == 1) { // 1 send lower byte of adc 2

SSP1BUF = lowerVal;

sum = 0;

count = 2;

} else if (count == 2) {

sum = getAverage(record2);

sensorVal = (uint16\_t) (sum / MAX);

upperVal = (sensorVal >> 8) & 0xff;

lowerVal = (sensorVal) & 0x00ff;

SSP1BUF = upperVal;

count = 3;

} else if (count == 3) {

SSP1BUF = lowerVal;

sum = 0;

count = 4;

} else if (count == 4) {

sum = getAverage(record3);

sensorVal = (uint16\_t) (sum / MAX);

upperVal = (sensorVal >> 8) & 0xff;

lowerVal = (sensorVal) & 0x00ff;

SSP1BUF = upperVal;

count = 5;

} else if (count == 5) {

SSP1BUF = lowerVal;

sum = 0;

count = 6;

} else if (count == 6) {

sum = getAverage(record4);

sensorVal = (uint16\_t) (sum / MAX);

upperVal = (sensorVal >> 8) & 0xff;

lowerVal = (sensorVal) & 0x00ff;

SSP1BUF = upperVal;

count = 7;

} else {

SSP1BUF = lowerVal;

sum = 0;

count = 0;

}

break;

case I2C1\_SLAVE\_READ\_COMPLETED:

default:;

} // end switch(i2c\_bus\_state)

}

uint16\_t getAverage(uint16\_t \*record) {

uint16\_t sum = 0;

for(int i= 0; i<MAX; i++)

sum += record[i];

return sum;

}