Open Source Fitness Band

Nathan Immerman
Joshua Kaufman
Tyler Kohan
Amit Shah
Steven Sloboda

Project Proposal
EECS 473 F15

Thursday, October 8th, 2015

Table of Contents

EXECUTIVE SUMMARY

What we are proposing is to build an *open source, open hardware* fitness band that would pave the way for community driven development and hacking of wearable fitness technology. We want make it simple for developers to create robust applications that incorporate data provided by this wearable. Additionally, we want to create a community driven by the desire to iterate and improve upon the hardware itself to open up new doors in wearable technology. Our proposed solution incorporates a GPS module, an Inertial Measurement Unit, a Heart Rate Monitor, an LCD, a buzzer, and a Bluetooth Low Energy module to pair with a smartphone. Relevant and current data will be displayed on the LCD, as well as the time, and more settings and long term data will be available on a smartphone Dashboard application. Power consumption, size, and ease of use (from a programmer's point of view) are driving factors in our design.

HIGH-LEVEL DESCRIPTION

An increasing number of fitness wearables are hitting the market, all of them on closed source platforms. Proprietary products that don't follow the open source model often limit or remove users and hackers ability to customize or improve their devices by modifying or writing their own applications for them. Proprietary wearables also make it difficult for developers to create and distribute their modifications and applications for other users to utilize. Thus, the idea that we are proposing is to build an open source, open hardware fitness band that would pave the way for community driven development and hacking of wearable fitness technology. In a sense, we would like to create "the Arduino" of fitness bands — a platform that is easy to use, modify, and incorporate into one's own projects.

A successful open source fitness band will have to meet certain visual and functional criteria in order to be a viable platform. Firstly, it must have a reasonable form factor that allows it to be worn comfortably on a wrist without feeling too heavy or cumbersome. Thus, size is an important factor that drives our design of the fitness band. Next, the band must be able to operate for at least one day on a full charge of battery, otherwise users will be discouraged from using it — a wearable that requires intermittent tethering to an external power source all day would defeat the purpose of a wearable. Hence, we need to extend the battery life of the fitness band as long as possible, so low power consumption is another driving factor of our design. With regards to functionality, our fitness band must be able to compete with the existing fitness bands on the market. As a result, it must incorporate location and speed tracking with GPS, heart rate monitoring, step counting, and off-loading of stored workout data from the fitness band to the connected phone. In the spirit of open source, accompanying this functionality will be an easy to use, cleanly-coded Application Programming Interface (API) that will allow application programmers to develop robust phone applications that utilize the data provided by the fitness band. In addition to the software API, we will make our hardware interfaces open source too, which will allow for easier interchanging of parts in future developments of the hardware design. Therefore, a driving factor in our software design will be code readability and ease of use for application programmers.

Proposed Designs

Both of our proposed designs have the same functionality but largely differ on user experience. The device will include a GPS module for location tracking, a heart rate monitor, an IMU for step counting, a buzzer, and Bluetooth Low Energy hardware for wireless off-loading of stored workout data. The MPU will process the data from the

sensors, turning the constant stream of data into meaningful statistics, and then push those numbers to a smartphone via BLE for longer term storage or further processing. The user interface of the device is what differentiates the two proposed designs and will greatly effect how the user interacts with these common, core features.

The first is a design that does not include a screen. A device without a screen is inherently a passive device, i.e., it is worn by the user but the user has no reason to physically interact with the device — all of the interaction would be done on a smartphone. This design is simpler to make low power, small, and unobtrusive to the daily activities of the user. On the other hand, this device design is difficult to interact with because it relies exclusively on the smartphone for interaction.

The second design includes a screen. The addition of a screen allows for interaction directly with the fitness band and creates a more welcoming user experience. The user can see the time, heart rate, and recently processed data from the IMU and GPS directly on the fitness band without having to use a smartphone. To see more aggregated data, the user must offload the data stored on the fitness band to a smartphone. For example, the user could see “Today’s Step Count” on the fitness band and “This Week’s Step Count” on the smartphone Dashboard. Additionally, the screen allows for more flexibility for improvements and additions after the hardware is fabricated than a device without a screen. The largest concern with a design with a screen is battery life. Typically, screens can consume large amounts of power which limits the longevity of the device and results more frequent recharging.

Ultimately, the design that we have chosen is the fitness band with a screen. We chose this design because we were able to find a screen that operates on the order of 10uW which eliminated our concern that the inclusion of a screen would draw too much power. We also researched competing fitness bands on the market and found that there is a product available, the Fitbit Surge[1], with a screen that is comparable in size with our chosen screen. We take the Fitbit Surge as evidence that even with the large size of our screen, the fitness band will be useable and comfortable to wear. Having a screen will also enable us to create a more engaging, flexible, and overall more useful device.

[1] https://www.fitbit.com/surge

IMPLEMENTATION DETAILS

User Interface

The user interface for our fitness band will consist of information displayed on the devices screen as well as a Dashboard application on a smartphone. The device screen will provide the user with quick, time relevant information such as current time and “today’s step count”, while the Dashboard will provide a location for more robust data and interaction. The Dashboard will also display all of the settings, have the ability to change the watch face, and display long term fitness data. The user will also have the ability to measure their heart rate just by using buttons on the fitness band.

To give application programmers easy access to the data gathered by the GPS module, heart rate monitor, and IMU our fitness band software will include a well documented application programming interface for remote procedure calls over the Bluetooth Low Energy. The API is included in Appendix C and includes calls to access information such as stored location data, heart rate data, and IMU step counts.

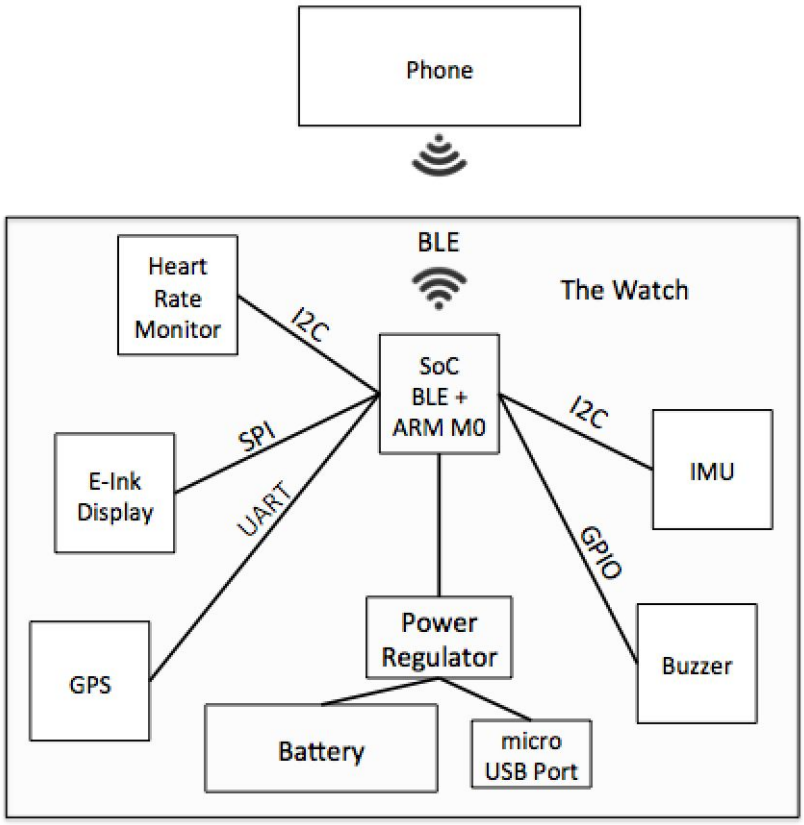
Two example use cases that developers could use our smartphone API for are a run/bike route tracker app and a dynamic workout app. Currently, the run/bike route tracker apps on the market all require the runner/rider to carry his/her phone (which includes the GPS). An app developed for our fitness band would allow the runner/biker to go on a run or bike ride with his/her fitness band on, and leave their smartphone at home. After the run or bike ride, the smartphone would then sync with the fitness band, collect the GPS data, and then display statistics about route taken. The second app, a dynamic workout trainer, would be able to guide a user through a workout that is tailored to his/her performance. The app could dynamically instruct the user to do certain exercises for different amounts of time, modifying those instructions based on the step count and heart rate data collected from the fitness band. For instance, the app could tell the user to do exercise A until it recognizes that he/she has a certain heart rate level and then switch to exercise B. The app could also make use of the buzzer to notify the user when time goals or intervals have been hit. These are just two examples of the types of apps that are made possible by our API and are not feasible with proprietary fitness bands on the market today.

Hardware Peripherals

The various components of the fitness band will be connected to the MPU by standard hardware interfaces including SPI, I2C, and UART. The fitness band will also

include a serial port, we are planning on using micro USB, which will provide power to the device during battery recharge. A standard JTAG interface will be exposed on the PCB for reprogramming of the MCU. The connections between the components are shown in Figure 1, below.

Figure 1: High level block diagram of parts



For our microprocessor, we chose the Nordic Semiconductor nRF51822 Multiprotocol Bluetooth low energy/2.4 GHz RF System on Chip (SoC). Important factors that played into our decision were power consumption, supported hardware interfaces, and the inclusion of a Bluetooth low energy radio. Since our fitness band will most often not be doing any particularly expensive computations, we were able to select a less powerful but also less power drawing processor — an ARM Cortex M0. This chip has 256 kB of embedded flash program memory and 32 kB of RAM which should be sufficient for the slimmed down RTOS and the minimal amount of software that we plan to run on the MCU. Since the microprocessor will be interfacing with several other components that have SPI, I2C, and UART interfaces, the nRF51822 was a good choice as it includes hardware support for all of them. With regards to power consumption, the nRF51822 is specifically marketed as an ultra low power MCU with

multiple levels of sleep states so it should be able to meet our low power requirements. We estimate that the SoC will draw about 20mA worst case (everything on) and when in sleep mode about 300uA. Specific current draw numbers vary highly for different use cases so we will have to experiment to get a better estimate of the current draw. In addition, this SoC requires an external antenna which comes in a package form factor (not made with PCB traces). Since none of our team members are extremely familiar with antenna design, we will seek help from the Mesh Lab Staff who are familiar with this particular SoC and have done antenna layout with this SoC in the past.

The display that we have chosen is a 1.3" SHARP Memory LCD. This display has the visual and power characteristics of an e-ink display, but with fast refresh rates. We choose this display because it has a very low power usage (average 3.6uA) and the fact that the display was a dot-matrix display rather than a segment display. A dot matrix display is more visually appealing than a seven segment e-ink display and considering that our fitness band is made to be worn, we decided that aesthetics were an important factor in our part choice.

The GPS module that we have selected is the MTK3339. The main reasons why we chose this part were the excellent documentation, the on-chip antenna, data logging capabilities, and the lower current draw than other comparable GPS modules. This GPS unit is able to collect 16 hours of location data, which includes time, data, latitude, longitude, and height, which is stored on internal flash storage. The data logging capabilities combined with the low power draw (20mA on average during use) made this unit a good choice.

The inertial measurement unit that we chose is the MPU-6050. From our research, we found that this IMU had one of the lowest power consumptions compared to other IMUs and it was contained in one chip — some of the other components we investigated were separate accelerometers and gyroscopes. The single chip feature is important due to our size constraints. Additionally, the relatively low power consumption aligns well with the power requirements of our fitness band. Also, several of our team members have prior experience with devices similar to the MPU-6050 which should help speed up development time.

We expect to also include a buzzer on the device to provide user feedback as they are using the device. We found a buzzer that has a relatively small form factor and that only draws 3mA when buzzing.

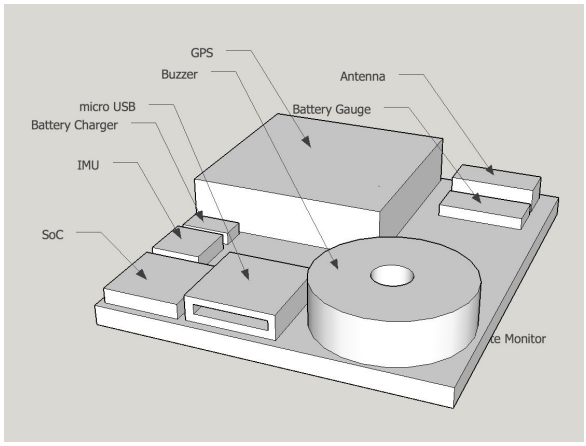
When deciding on a heart rate monitor, size, power, and reasonable accuracy were the most important factors. We narrowed down our options to the TI AFE4403 and the Maxim Integrated MAX30100 due to their small sizes and low power consumptions. The tradeoff between the two parts was their data resolution; the TI has 22 bits whereas the Maxim has 16. Ultimately, we selected the MAX30100 because both its idle and running power consumption were lower and because higher accuracy heart rate measurements are not critical for the fitness band, so a lower resolution is acceptable.

Minimum Size of the Device

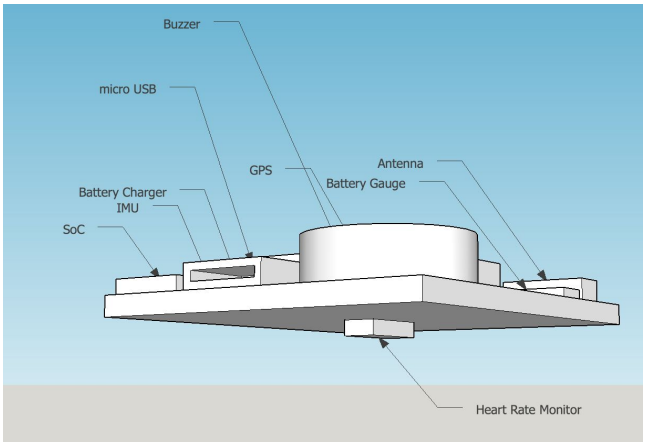
At the moment, we are only able to estimate the size of our device by assuming that we could put parts right up against each other and ignore how parts are going to actually be connected. In Figure 2 on page 8, picture A shows that all of the peripheral devices (except the screen) fit onto a PCB that is 27mm x 31mm, with some extra space. Note, that this is not a pcb layout, so the placements are not necessarily logical (the SoC with BLE is not next to the antenna). Rather this is a proof that the peripherals can fit on a PCD under the LCD as we had hoped. In Figure 2 on page 8, picture B shows that the heart rate monitor is located on the underside of the board because the heart rate monitor will need to be up against the user's wrist to be able to measure their heart rate. Also, picture C in Figure 2 on page 8, shows how our PCB, battery, and LCD will be layered.

picture D in Figure 2 on page 8, shows the minimum dimensions of the entire device. The LCD is also 27mm x 31mm, the same size as the minimum PCB and will be stacked above the PCB. In between the PCB and the LCD will be the battery. The battery is slightly longer that the LCD, which extends our minimum length to 35mm. picture D also shows our minimum depth of 9.8mm which is a summation of the heights of (listed from top to bottom) the LCD, the battery, the GPS (tallest component on the PCB, the thickness of the PCB, and excluding the heart rate monitor since it will be placed into an opening in the case in order to have contact with the wearer. In addition, we will be designing a case to surround our device, which will add a couple of millimeters to all of the dimensions and brings our estimated dimensions to 30mm x 38mm x 11mm.

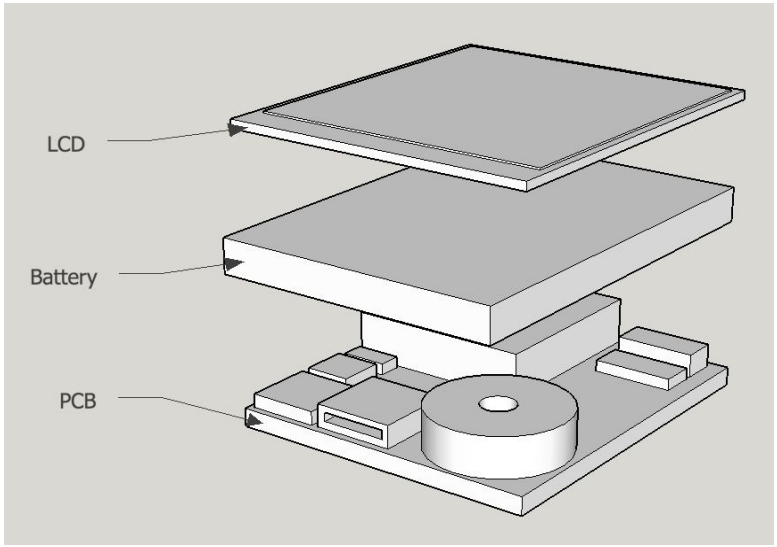
Figure 2: Minimum Layout of Peripheral Devices



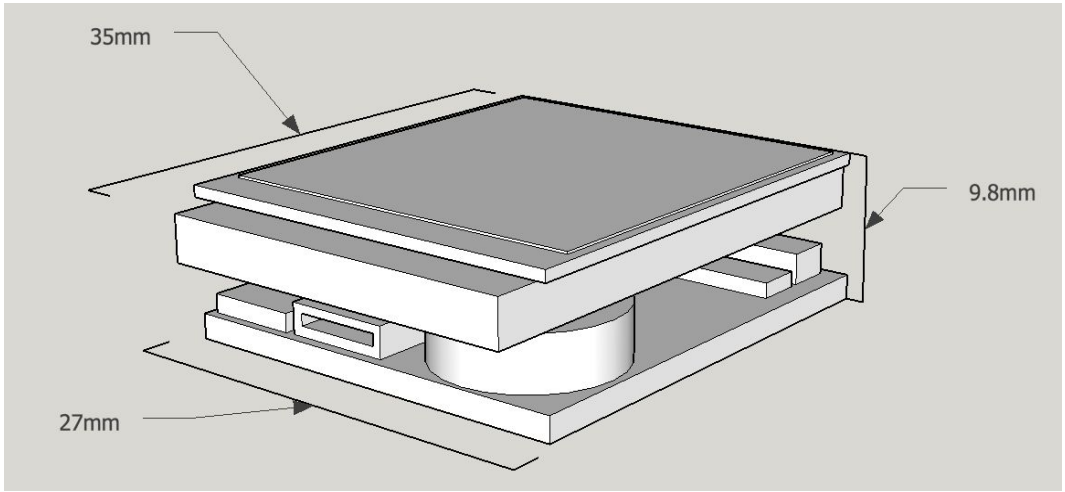
A



B



C



D

Battery and Power

We have decided to go with a lithium polymer battery which has a capacity of 400mAh. This battery has a maximum discharge current of 400mA. If we ran all of our components continuously, listed in Table 1 below, and all of the components were drawing their worst case current draw we would need about 110mA. This is around 25% of the amount of current that the battery is able to supply which should be sufficient and should not result in significant power spillage. In addition, on average we won't be running all of our devices continuously. The IMU and LCD will be on the most, then the SoC will be on much of the time - all of the other devices will only be on sporadically. So on average we will need 20mA + 3.9mA + 4uA = ~24mA, since this battery has a capacity of 400mAh then our device can run for approximately 400/24 = 16 hours on single charge.

We are also including a Lithium Polymer USB Battery Charger IC, the MAX1555, to assist in recharging the battery. We will also be incorporating a Battery Gage, the DS2782, to estimate the remaining battery life of the device.

Table 1: Current Consumption Estimates

Item	Minimum	Average (when in use)	Maximum	Voltage
SoC w/ BLE	4mA (awake) 3uA (asleep)	5mA	20mA	3.3V
IMU	500uA	3.9mA	3.9mA	3.3V
GPS	N/A	20mA	30mA	3.3V
LCD	2uA	4uA	4uA	3.0V
HRM (IC) HRM (LED)	1uA N/A	600uA 20mA	1.2mA 20mA	3.3V 1.8V
Buzzer	N/A	3mA	35mA	3.3V
Total Current	506uA	~53mA	~110mA	

Product Comparison

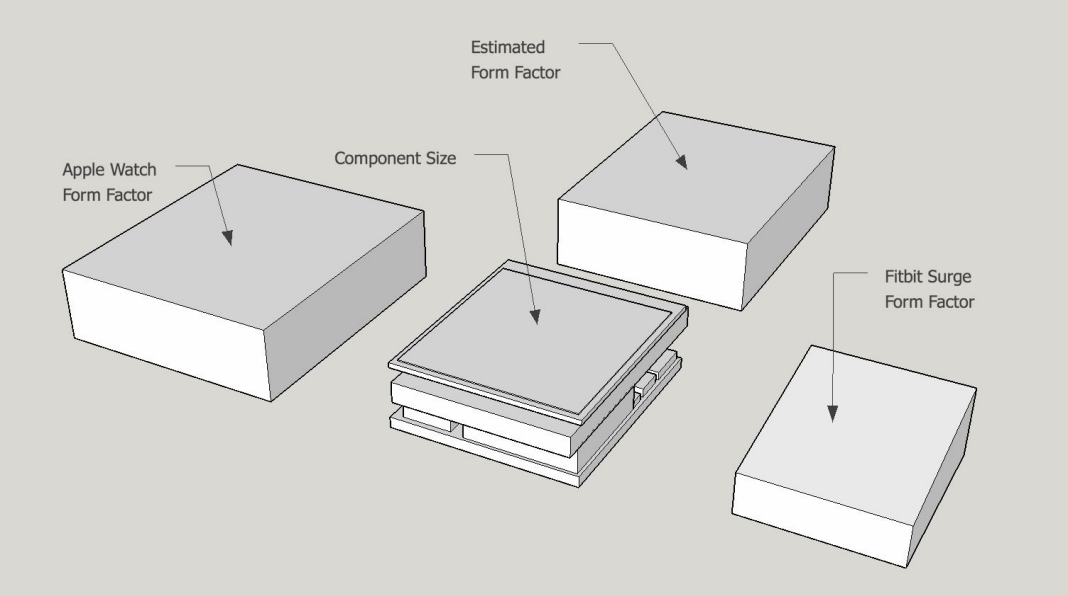
To estimate our devices feasibility in the growing wearable market we compared our device to two relevant competitors: the Apple Watch and the Fitbit Surge. In Table 2 on page 10, we summarize the dimensions as well as the weight of all 3 devices. Our

device is slightly smaller than the Apple Watch and slightly larger than the Fitbit Surge, which can also be seen visually in Figure 3 below. As seen by people's willingness to wear the Apple Watch and Fitbit Surge, we anticipate the size of our device to be accepted by consumers. In terms of weight, we sit right in the middle, i.e., ~25 grams heavier than the Fitbit Surge and ~20 grams lighter than the Apple Watch. Since we are a mix between the Apple Watch and Fitbit Surge it is very fitting that our weight fall between the two as well.

Table 2: Current Consumption Estimates

Device	Width (mm)	Height (mm)	Depth (mm)	Weight (g)
Apple Watch[2]	33.3	38.6	10.5	72
Fitbit Surge[3]	20.9	24.4	10	32
Open Source Band	30	38	11	58

Figure 3: Form Factor Comparison with Similar Products

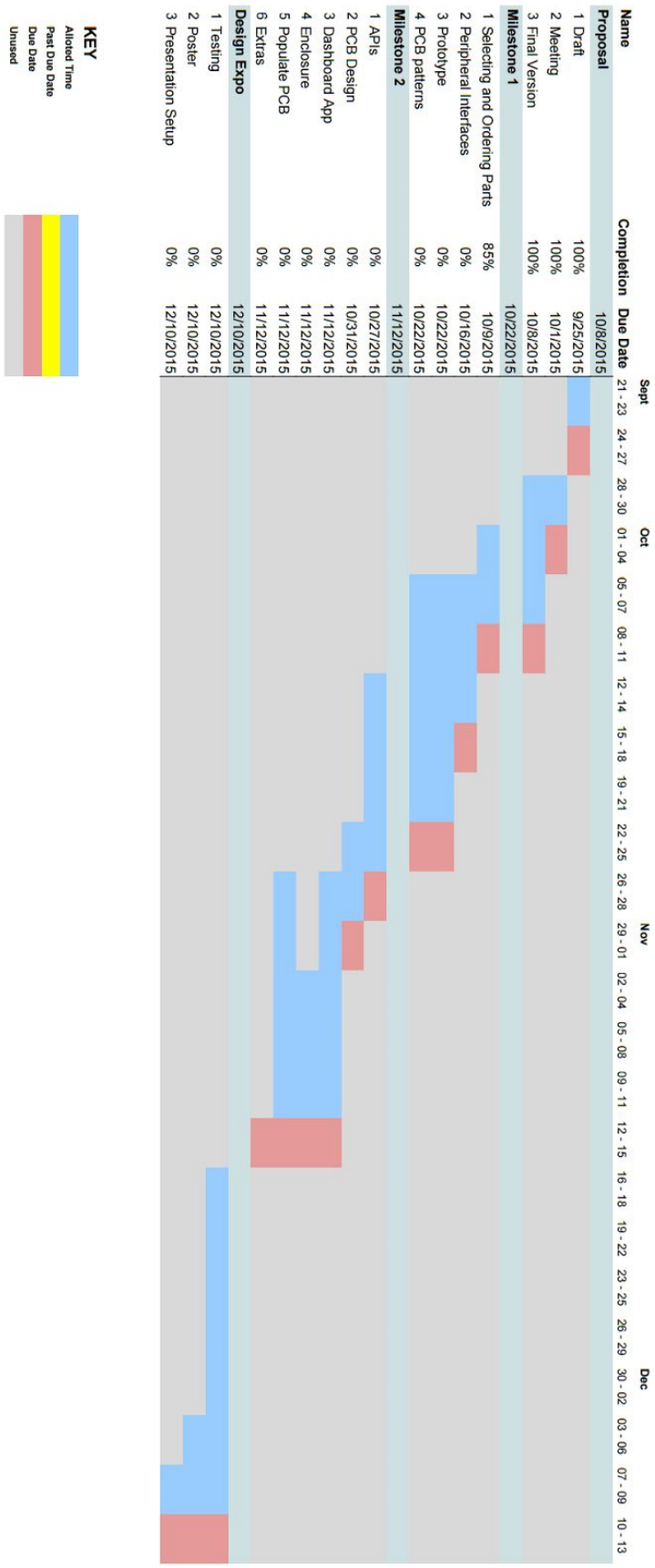


[2] www.apple.com/watch/
[3] www.fitbit.com/surge/

WAY FORWARD AND SCHEDULE

Although our schedule — displayed below as a Gantt chart in Figure 4 on page 12 — is reasonable, it has several dependencies. First, the design specification must be finalized before parts can be ordered. Next, the development of the peripheral interfaces is dependent upon the arrival of parts and the time required to get the evaluation boards up and running. This deadline is very tight so this task may bleed into Milestone 2. Additionally, the enclosure is dependent upon the dimensions of the PCB so it has to wait until the PCB has been designed. Lastly, the PCB must be ordered and arrive before we can begin populating the board with our components.

Figure 4: Project Schedule Gantt Chart



DESIGN EXPO AND FINAL PRESENTATION

At the design expo, we plan on demonstrating a fully functional fitness band, a companion dashboard application, and ideally a game that uses the fitness data from the band. We will be able to show that the band's major components are gathering data and can display it to the screen. Also, we will show that data can be wirelessly off-loaded from the band to the companion dashboard application. The dashboard app will help us convey the potential for app development using our fitness band. Lastly, the game that is playable using the fitness band is a stretch goal due to time constraints, but it would help show off the potential for app development using our open source, open hardware platform as well. We will demonstrate our device by having a team member wear and use the band as well as use the dashboard app on his smartphone.

Other deliverables will include a poster describing our project, a laptop displaying a video of our project, any prototype boards that we used for testing, and the code and board schematics for the fitness band. Our group will need power outlets for charging devices but other than that, we will not have any special requirements.

BUDGET AND MATERIALS

The budget for the fitness band was greatly influenced by the driving factors of our design: low power consumption and small form factor. This is reflected in the costs of many of the parts, as we selected the part that fit our requirements the best rather than the least expensive part. This will not be the iteration intended for mass production so price of parts are not as important. The total cost of materials, however, came out to $784 which is well under the $1,000 maximum, as seen in Table 3, below.

Table 3: Proposed Budget of $784 Under the $1,000 Limit

Item	Unit Price	Quantity	Total Price
LCD: LS013B7DH03	$20	1	$20
LCD: Breakout	$40	1	$40
Heart Rate Monitor: MAX30100	$5	2	$10
BLE + ARM M0 SoC: nRF51822	$5	2	$10
BLE Antenna: FR05-S1-N-0-102	$1	2	$2
Balun (for antenna): BAL-NRF01D3	$1	2	$2
BLE + ARM M0 SoC: NRF52-PREVIEW-DK	77	1	$77
GPS: MTK3339	$30	1	$30
GPS: Adafruit breakout	40	1	$40
IMU: MPU-6050	$10	1	$10
IMU: MPU-6050 breakout	$40	1	$40
PCB	$30	5	$150
LiPo battery 400mAh: GSP652535	6.95	2	$14
LiPo battery charger IC: MAX1555-1	1.95	2	$4
Battery gage: DS2782 (Maxim)	5	2	$10
Case Build	$75	1	$75
Misc	$150	1	$150
Shipping	$100	1	$100
Buzzer: PS1240P02BT	$1	1	$1
Total	---	---	$784

CONCLUSION

We hope to build an open source, open hardware wearable fitness band platform. It will incorporate a GPS module, an Inertial Measurement Unit, a Heart Rate Monitor, an LCD, and a Bluetooth Low Energy module to pair with a smartphone. By using the LCD and a Dashboard smartphone application we will able to provide the user with relevant information both quickly and robustly. Most of our parts were chosen with the main considerations being power and size. We were successfully able to find a screen that we were able to incorporate without a significant cost to power consumption. In addition, all of our parts, an all evaluation boards, came in under the $1,000 budget at $784, which is ideal in case we encounter any unforeseen costs.

APPENDICES

Appendix A: Project Goals

Design Criteria	Importance	Will/Expect/Stretch
Bluetooth Connectivity	Fundamental	Will
LCD Screen	Fundamental	Will
IMU	Fundamental	Will
User friendly APIs	Fundamental	Will
Rechargeable battery w/ 16 hours of battery life	Fundamental	Expect
GPS	Important	Expect
Smartphone dashboard app	Important	Expect
Heart Rate Monitor	Important	Expect
Vibrator or buzzer	Optional	Expect
Smartphone game app	Optional	Stretch

Appendix B: Milestones 1 and 2

Milestone 1
Screen can display both time and information from IMU
Prototype can display its sensor data on a phone via bluetooth
Battery Circuitry is designed and tested
PCB layout completed

Milestone 2
Dashboard phone app displaying real-time fitness information like number of steps or distance traveled
PCB Fully Assembled and tested
Finished Version of APIs are written and tested
Fitness Band Enclosure and strap designed and manufactured

Appendix C: Software Interfaces

Maxim DS2782 Standalone Fuel Gauge IC (Coulomb Counter)

```
/* DS2782.h
 * Interface for the Maxim DS2782 Standalone Fuel Gauge IC.
 */

#ifndef _DS2782_H
#define _DS2782_H

#include <stdint.h>


/* The modes of operation that the DS2782 can be in.
 */
enum DS2782_modeEnum = {
    ACTIVE, SLEEP
};


/* Initialize the DS2782 unit to ready it for communication.
 */
void DS2782_init(/* TODO pass in the MCU pins and I2C handle */);


/* Set the parameters for the DS2782 that allow it to accurately track the
 * battery functionality. This only needs to be called ONCE for the lifetime of
 * the battery and should be called after DS2782_init.
 */
void DS2782_configParams(/* TODO determine exactly which parameters will be set
                            for our use case */);


/* Put the DS2782 into a specified mode of operation.
 */
void DS2782_setMode(enum DS2782_modeEnum mode);
```

```
/* Get an instantaneous voltage reading from the battery.
 */
uint16_t DS2782_getInstVoltage();


/* Get an instantaneous current reading from the battery.
 */
uint16_t DS2782_getInstCurrent();


/* Get an average current reading from the battery.
 */
uint16_t DS2782_getAvgCurrent();


/* Get an instantaneous temperature reading from the battery.
 */
uint16_t DS2782_getInstTemp();


/* Get the battery's full capacity.
 */
uint16_t DS2782_getFullCapacity();


/* Get the battery's remaining active capacity.
 */
uint16_t DS2782_getActiveCapacity();


/* Get the battery's remaining standby capacity.
 */
uint16_t DS2782_getStandbyCapacity();


/* The registers and corresponding addresses in the DS2782.
 */
enum DS2782_regsisterEnum = {
```

```
    STATUS = 0x01,      // Status (R/W)
    RAAC_MSB = 0x02,    // Remaining Active Absolute Capacity MSB (R)
    RAAC_LSB = 0x03,    // Remaining Active Absolute Capacity LSB (R)
    RSAC_MSB = 0x04,    // Remaining Standby Absolute Capacity MSB (R)
    RSAC_LSB = 0x05,    // Remaining Standby Absolute Capacity LSB (R)
    RARC = 0x06,        // Remaining Active Relative Capacity (R)
    RSRC = 0x07,        // Remaining Standby Relative Capacity (R)
    IAVG_MSB = 0x08,    // Average Current Register MSB (R)
    IAVG_LSB = 0x09,    // Average Current Register LSB (R)
    TEMP_MSB = 0x0A,    // Temperature Register MSB (R)
    TEMP_LSB = 0x0B,    // Temperature Register LSB (R)
    VOLT_MSB = 0x0C,    // Voltage Register MSB (R)
    VOLT_LSB = 0x0D,    // Voltage Register LSB (R)
    CURRENT_MSB = 0x0E, // Current Register MSB (R)
    CURRENT_LSB = 0x0F, // Current Register LSB (R)
    ACR_MSB = 0x10,     // Accumulated Current Register MSB (R/W)
    ACR_LSB = 0x11,     // Accumulated Current Register LSB (R/W)
    ACRL_MSB = 0x12,    // Low Accumulated Current Register MSB (R)
    ACRL_LSB = 0x13,    // Low Accumulated Current Register LSB (R)
    AS = 0x14,          // Age Scalar (R/W)
    SFR = 0x15,         // Special Feature Register (R/W)
    FULL_MSB = 0x16,    // Full Capacity MSB (R)
    FULL_LSB = 0x17,    // Full Capacity LSB (R)
    AE_MSB = 0x18,      // Active Empty MSB (R)
    AE_LSB = 0x19,      // Active Empty LSB (R)
    SE_MSB = 0x1A,      // Standby Empty MSB (R)
    SE_LSB = 0x1B,      // Standby Empty LSB (R)
    EEPROM = 0x1F,      // EEPROM Register (R/W)
    /* TODO: add these if necessary later on */
    // User EEPROM block 0 from 0x20 to 0x2F. Lockable. (R/W)
    // Additional user EEPROM block 0 from 0x30 to 0x37. Lockable. (R/W)
    // Parameter EEPROM from 0x60 to 0x7F. Lockable. Block 1. (R/W)
    // Unique ID from 0xF0 to 0xF7. (R)
    FXN_CMD = 0xFE      // Function Command Register (R)
};


/* Low-level write interface to the DS2782. This is exposed to support any
 * functionality without a specific function in this interface.
```

```
 */
void _DS2782_writeReg(enum DS2782_regEnum, uint8_t data);


/* Low-level read interface to the DS2782. This is exposed to support any
 * functionality without a specific function in this interface.
 */
uint8_t _DS2782_readReg(enum DS2782_regEnum);

#endif
```

SHARP LS013B7DH03 1.3” Memory LCD

```
/* LS013B7DH03.h
 * Interface for the LS013B7DH03 LCD.
 */

#ifndef _LS013B7DH03_H
#define _LS013B7DH03_H

#include <stdint>
#include <stdbool.h>


/* Initialize the LCD.
 */
void LS013B7DH03_init(/* TODO pass in the MCU pins and SPI handle */);


/* Stores the bit map of the current contents of the screen as a 1-D array.
 * When a row is edited in this array, the correpsonding bit in
 * LS013B7DH03_editedRows should be set. Afterward, LS013B7DH03_pushImage
should be
 * called.
 */
bool LS013B7DH03_bitMap[16384];


/* Keeps track of which lines of the image have been edited so that only the
 * sections of the screen requiring edits will be written to. This is to save
 * CPU time, and refresh rate
 */
bool LS013B7DH03_editedRows[128];


/* Write a row of the bit map using the corresponding row in the
 * LS013B7DH03_bitMap
 * array, to the LCD.
 */
void LS013B7DH03_writeRow(uint8_t row);
```

```
/* Calls LS013B7DH03_writeRow() for all the lines of the LCD that require
 * updating and clears the LS013B7DH03_editedRows bits for those rows.
 */
void LS013B7DH03_pushImage();


/* Low-level write interface to the LS013B7DH03. This is exposed to support any
 * functionality without a specific function in this interface.
 */
void _LS013B7DH03_writeReg(uint8_t reg, uint8_t data, bool EXTCOMIN);

#endif
```

Maxim MAX30100 Heart Rate Monitor

```
/* MAX30100.h
 * Interface for the MAX30100 Heart Rate Monitor.
 */

#ifndef HR_H
#define HR_H

#include <stdint.h>
#include <stdbool.h>


//Register Constants (all support R/W)
#define FIFO_DATA_REG 0x05
#define MODE_CONFIG_REG 0x06
#define SPO2_CONFIG_REG 0x07
#define LED_CONFIG_REG 0x09
#define TEMP_INT_REG 0x16
#define TEMP_FRAC_REG 0x17


/* This will setup the Heart Rate Monitor properly.
 * Need to configure the IC's mode of operation for our application.
 */
void MAX30100_init();


/* Takes a 10 second reading of the user's heart rate
 * Returns the number of beats in one minute
 */
uint8_t MAX30100_takeReading();


/* If en is true, then put device into power saving mode. If en is false,
 * put device into normal operation mode
 */
void MAX30100_powerSaveEnable(bool en);
```

```
uint16_t _MAX30100_buffer[16];
uint16_t _MAX30100_temperature_val;


/* Helper function for writing registers on the IC.
 * Will be used in constructor for configuration
 */
void _MAX30100_writeReg(uint8_t reg, uint8_t data);


/* Helper function for reading registers on the IC.
 * Will be used in getOneSample() and getTemp()
 */
uint8_t _MAX30100_readReg(uint8_t reg);


/* Each I2C transaction will return 4 bits of useful info. Need to read 4 times
 * for one sample. Store each sample in buffer for takeReading to process
 */
uint16_t _MAX30100_getOneSample();


/* Gets current temperature. Useful for adding precision to our measurements
 */
uint16_t _MAX30100_getTemp();

#endif
```

InvenSense MPU-6050 IMU

```
/* MPU6050.h
 * Interface for the MPU-6050 intertial measurement / digital motion processor
 * unit.
 */


#ifndef _MPU6050_H
#define _MPU6050_H

#include <stdint.h>


/* Initialize the IMU.
 */
void MPU6050_init(/* TODO pass in the MCU pins and I2C handle */);


/* Enable the accelerometer.
 */
void MPU6050_accelEnable();


/* Disables the accelerometer.
 */
void MPU6050_accelDisable();


/* Enable the gyroscope.
 */
void MPU6050_gyroEnable();


/* Disable the gyroscope.
 */
void MPU6050_gyroDisable();
```

```
/* Enable the Magnetometer.
 */
void MPU6050_magEnable();


/* Disables the Magnetometer
 */
void MPU6050_magDisable();


/* Set the sample rates for the accelerometer and the gyroscope.
 */
void MPU6050_setSampleRates(uint16_t acclRate, uint16_t gyroRate);


/* Configures the FIFO Buffer.
 */
uint16_t MPU6050_configFifo();


/* Reads from the FIFO Buffer.
 */
uint16_t MPU6050_readFifo(char * buf);


/* Bit masks for DMP configuration.
 */
enum MPU6050_dmpFeatureEnum = {
   TAP = 0x001,
   ANDROID_ORIENT = 0x002,
   LP_QUAT = 0x004,
   PEDOMETER = 0x008,
   X6_LP_QUAT = 0x010,
   GYRO_CAL = 0x020,
   SEND_RAW_ACCEL = 0x040,
   SEND_RAW_GYRO = 0x080,
   SEND_CAL_GYRO = 0x100,
};
```

```
/* Enables and Configures the Digital Motion Processor.
 */
void ImuConfigureDMP(enum MPU6050_dmpFeatureEnum mask);


/* Low level read interface to the MPU6050. This is exposed to support any
 * function without a sp6ecific function in this interface.
 */
void _MPU6050_writeReg(uint8_t reg, uint8_t length, uint8_t data[]);


/* Low level read interface to the MPU6050. This is exposed to support any
 * function without a specific function in this interface.
 */
uint8_t _MPU6050_readReg(uint8_t reg);

#endif
```

GlobalTop Technologies MTK3339 GPS Module

```
/* MTK3339.h
 * Interface for the MTK3339 GPS module.
 */

#ifndef _MTK3339_H
#define _MTK3339_H

#include <stdint.h>


/* Struct to hold commonly used GPS data.
 */
struct gpsDataStruct {
    float latitude;
    float longitude;
    float altitude;
    float utcTime;
};


/* Initialize the GPS module.
 */
void MTK3339_init(/* TODO add the MCU pins and UART handle */);


/* Turn on the GPS functionality.
 */
void MTK3339_enable();


/* Turn off the GPS functionality and put the module into standby mode.
 */
void MTK3339_disable();


/* Begin logging the GPS data using the logger on the GPS module.
 */
```

```
void MTK3339_logStart();


/* End logging the GPS data using the logger on the GPS module.
 */
void MTK3339_logStop();


/* Parse the NMEA sentence to extract GPS data.
 */
void MTK3339_parseSentence(char * sentence, uint8_t len,
                           struct gpsDataStruct * gpsDataPtr);

/* Send an ASCII command to the GPS module.
 */
void MTK3339_sendCommand(char * cmd, uint8_t len);

/* Wait for a sentence of data to be received from the GPS module, then return
 * it in buf;
 */
void MTK3339_waitForSentence(char * buf, uint8_t len);

#endif
```

Fitness Band Bluetooth Low Energy Profile

Attribute Type	Attribute Name	UUID	Description
Primary Service	Generic Access Service	0x1800	Generic device information
Characteristic	Device Name	0x2A00	Name of device. Read only.
Characteristic	Appearance	0x2A01	External appearance of device. Read only.
Primary Service	Heart Rate Service	0x180D	Exposes data from a heart rate monitor.
Characteristic	Heart Rate Measurement	0x2A37	Data format bits and data bits for heart rate.
Descriptor	Heart Rate Measurement Client Characteristic Configuration	0x2902	Allows the client to configure the Heart Rate Measurement characteristic and subscribe to it.
Primary Service	Custom GPS Service	TBD*	Exposes GPS location and movement data.
Characteristic	GPS Module Status	TBD*	The state that the GPS module is in (SLEEP/ACTIVE). Read and write.
Characteristic	GPS Location and Speed	TBD*	The current location and speed of the GPS module. Read only
Descriptor	GPS Location and Speed Client Characteristic Configuration	0x2902	Allows the client to configure the GPS Location and Speed characteristic and subscribe to it.
Primary Service	Custom Pedometer Service	TBD*	Exposes pedometer data.
Characteristic	Pedometer Module Status	TBD*	The state that the pedometer (IMU) is in (SLEEP/ACTIVE). Read and write.
Characteristic	Pedometer Step Count	TBD*	The current number of steps

			that the pedometer has recorded. Read only.
Descriptor	Pedometer Step Count Client Characteristic Configuration	0x2902	Allow the client to configure the Pedometer Step Count characteristic and subscribe to it.
Primary Service	Battery Service	0x180F	Exposes the state of the battery within the device.
Characteristic	Battery Level	0x2A19	Battery level expressed as a integer percentage from 0 to 100. Read only.

* The UUID for the given attribute is not critical for the proposal and is to be determined.

Fitness Band Interface from Perspective of Android SmartPhone

```
public class BtManager{
        private BluetoothGattService currentService;

        // Constructor for init connection. Will scan and connect to appropriate
        // paired device
        public BtManager();

        // Connects to a BLE Service and sets currentService to attService
        private setService(BluetoothGattService attService);

        // returns currentService
        public BluetoothGattService getService();

        // This returns an ArrayList of all the data for a specific service
        // It will be called by other Service methods, and it is exposed to
        // the user so that they can make there own Service methods.
        public ArrayList getCharacteristicData(BluetoothCharacteristic, int uuid);
}

public class HeartRateService
{
        // Empty Constructor because Java
        public HeartRateService();

        // Returns the pulse rate in beats per minutes
        public int getPulseRate();
}

public class GPSSevice
{
        // Empty Constructor because Java
        public GPSSevice();

        // Returns true if GPS is active, false if sleeping
        public boolean getStatus();

        // Returns current Latitude
```

```
        public int getLatitude();

        // Returns current Longitude
        public int getLongitude();

        // Returns the wearer's speed in kilometers per hour
        public int getSpeed();
}


public class PedometerService
{
        // Empty Constructor because Java
        public PedometerService();

        // Returns true if IMU is active, false if sleeping
        public boolean getStatus();

        //Returns the current step count
        public int getStepCount();

        //Reset the step count back to zero
        public void resetStepCount();
}

public class BatteryService
{
        // Empty Constructor because Java
        public BatteryService();

        // Returns a number between 0 and 100 representing the percent remaining
        public int getPercentage();
}
```

Appendix D: Team Agreement

Each group member will devote approximately 15 hours per week to the project, working more or less hours as necessary. In general, group members are free to work on the project weekdays after 6:00 pm and during the day on weekends. We expect that the bulk of the project work will be done on weekends as larger chunks of time can be allocated to working on those days. We agree to check all means of communication once every 24 hours, at minimum, to stay in the loop. Our channels of communication are Asana (a to-do list manager), Github, and GroupMe (an instant messenger). We will meet as a team every Tuesday from 4:00pm - 5:00pm. This meeting will start promptly on Michigan Time. We will use this meeting to discuss the previous week's tasks, current state of the project, and tasks for the upcoming week.

Initially, we had been working as a group to write the proposal and select parts, but now we are transitioning toward independent work that will be reviewed weekly by group members. Individuals will not let other group members time conflicts affect their independent work toward their individual weekly project goals. Although each group member is expected to have a thorough understanding of how all project components work, we are requiring that each group member specializes on a few components of the project, leading their design and development. Tentatively, the specialization is as follows: Nathan will lead development for the IMU and buzzer, Joshua will lead development for the LCD and the heart rate monitor, Tyler will lead development for the GPS and the API, Amit will lead development for the dashboard app and the PCB, and Steven will lead the development of BLE communication and power management techniques.

Since we are all seniors interviewing for jobs, conflicts are bound to happen with onsite interviews. We agree to notify the team immediately if we are going to be away for more than 1 day for any reason, including traveling for interviews. Currently, there are no known conflicts that will pull any of us away from the project for days at a time. The workloads from our other classes are manageable and will not interfere with the project in a significant way.

Nathan Immerman	Joshua Kaufman	Tyler Kohan
Amit Shah	Steven Sloboda	