Sensor Driven Audio Control System Running Device – Music In Motion

Zoilo Boehme, Eric Hoofnagle, Lane Starratt

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The objective of this project is to provide the user with a motivational, musical experience while running via biofeedback control and GPS data collection. The biofeedback control will be allocated impulse responses from the user's footsteps and create a rhythm from each footfall. The GPS data allows for distance notification in real-time. Data collected during the running session is stored afterwards in an Android developed application for user reference.**

***Index Terms* — Computer generated music, data integration, FIR digital filters, linear feedback control, linear predictive coding.**

I. Introduction

From the most general standpoint, the main focus of this project is to create a device which defines a completely unique running experience for the user. This experience is one in which is currently unavailable on the market where the user can synchronize the music's tempo with his/her run. The effect is a complete sense of immersion with the music whereby the user becomes synchronized with the music. This device transcends the typical passive listening process of any running device and creates an active symbiosis between the runner and their environment.

The feedback control system which synchronizes the music is the main attribute of the design for Music in Motion (MIM). This will be accomplished by collecting the impulse data from the runner's footsteps via a digital accelerometer. This impulse data will be digitally filtered and then used to control the tempo of the song. The impulse train of foot-falls will have a certain frequency, which we can map out to be quarter-notes or eighth-notes for the music that is created in real-time. This is the most essential feature of the entire project.

Additionally, the MIM device will automatically keep track of certain milestones that are of usual interest for runners. Using a GPS, the runner's position and velocity will be monitored such that athletic accomplishments can be tracked. The user will be notified of their progress through auditory alerts that are triggered and played through the headphones. This module will track every mile which will always keep the user informed of their progress in real-time.

This device incorporates a clip that allows the user to attach the device to either their running pants or an elastic belt, depending upon the needs of the runner. This clip will free the hands of the user and keep the MIM device firmly attached to their person for the sake of comfort and simplicity. Additionally, the device is roughly 4"x4"x1" which is fairly small considering its stage in the product life cycle. It is small enough to be convenient, but not small enough to be competitive. As a prototype, it is sufficient and within our specifications.

The aim of the Music in Motion system is to be as simple as possible to use, meaning very few buttons and no display on the device itself. Simplicity is the key for a product in which the user needs not to be distracted by options—it must seamlessly integrate with the runner. This is because a person who is engaging in a physically and mentally taxing anaerobic exercise does not want to be pulled out of 'the zone' in order to fiddle with their device that is supposed to be aiding their fitness session instead of impeding it.

The system layout includes two sensors communicating with the first microcontroller responsible for interpreting sensor data and passing along control lines. The second microcontroller takes the control lines and produces music based on the tempo input. The other control signals are for alerts that notify a milestone is reached. The actual music is synthesized in the MIDI chip. This chip outputs the analog audio to a headphone jack.

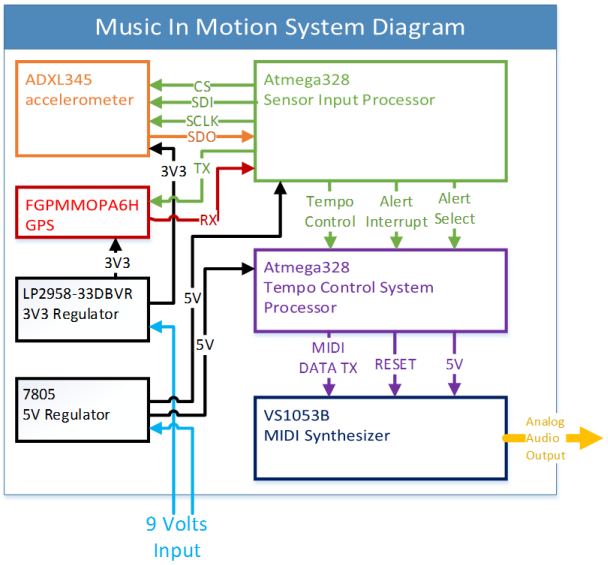

Figure 1 – System Block Diagram for Music In Motion

The figure above is a visual diagram for all the subsystems involved in the project. The 9-volt battery power supply is voltage regulated for each component in the design.

II. Foot-Fall Detection

The user's running session will be essentially represented by an impulse train for the purposes of the music's tempo control. This distinguishes the MIM device from pedometers or running devices because it implores a higher quality signal, lower error rate, and lesser latency (between the signal and audio response). An accelerometer seems to be a great tool for detecting user impulses. This is because they fundamentally are changes in acceleration. Figure 1 [1] below shows the range of motion in terms of acceleration for a person taking one step.

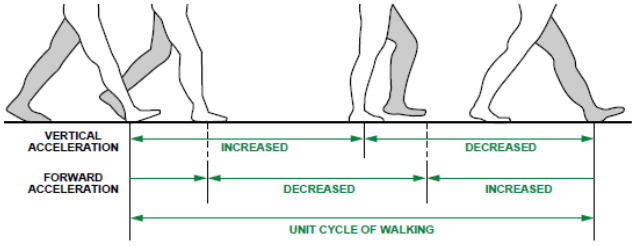

Figure 2 - Visual representation of acceleration change while walking

A. Accelerometer

The accelerometer is a crucial component to have for this project. This is the backbone of the sensor data which will be used to control the tempo of the Music in Motion running device. The basic function for the accelerometer will be to collect impulse data. These impulses are the result of each footstep from the user's running stride, which generates an immediate spike in acceleration as one's foot lands on the floor. This spike is the spectacle of interest and this signal, known as an impulse train, has a certain frequency which will also be of interest. The frequency of this impulse train can be mapped out to the quarter notes of the song played by the MIM device via means of continuous feedback.

The ADXL345 accelerometer by Analog Devices, Inc. is the focus of the project as it provides the essential rhythm to the music created by the audio engine. Through research we found that a 3-axis accelerometer can provide enough dynamics to determine when a user steps down during a running session. The three axes covers all range of motion for the runner and with digital filtering and other algorithms; all three axes are used to determine the footfall without any emphasis on a particular axis. What this means is the device itself can be held at any orientation and the footfall detection will still process and convert the same way making which axis is preferable arbitrary.

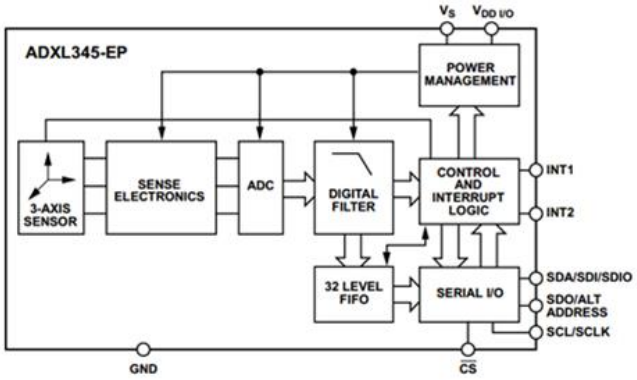

Figure 3 – Functional Block Diagram of ADXL345 Accelerometer

The figure above demonstrates the system level architecture of the ADXL345. This diagram shows much of the functionality that proved to be important factors such as: signal smoothing, power management, interrupt logic, output format, maximum swing, sensitivity level, bandwidth, interface, and mounting type. With digital the ADXL345 there are two options for the interface output protocol: SPI (3 or 4 wire) and I2C; 4-wire SPI was used. The output format is digital, which is useful for digital filtering. The ADXL345 has a selectable measurement range of ±2 g, ±4 g, ±8 g, or ±16 g. It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, that allows the device to be used as a tilt sensor. The raw data received is already converted digitally by the accelerometer and provided as integers for all three axes making the process of observing the behavior of the runner during prototyping much simpler. The data is digitally filtered using adjusted averages to create impulse signals to send to the second ATmega328p microcontroller to produce the music around the signals. Initialization of the accelerometer via software only requires setting a range for the sensitivity of the device once it is attached to the microcontroller.

B. Digital Filtering

The bandwidth of the signal collected from the runner is between 1 Hz and 7 Hz. It is a very narrow bandwidth and is also a very low frequency. It is highly improbable that a runner or even a person walking will venture outside of this bandwidth. To check the corner cases, an extremely fast runner, with short legs thus a short running stride would not step seven times in a second.

It is apparent then that in order to reduce the noise from this signal and maximize its quality, a low-pass filter must be implemented. A digital filter is ideal for this project due

to the monetary and time cost that comes with changing or tweaking an analog filter for a system design.

A FIR filter was chosen due to the simple nature of its design and linear phase property. Introducing non-linear phase in this signal is the same as distorting the signal for this time sensitive application. This is due to the high sensitivity in periods between impulses because of its use in the control system for updating the tempo of the music. Additionally, FIR filters are very stable and are not too resource intensive on the microcontroller dedicated to analyzing the accelerometer data of the MIM device. The figure below shows a FIR filter for n terms. In case of MIM, the software has n = 4. These 4 values are stored in a queue where the newest value is added to the end and the 4th going to 5th value is removed from the front of the queue. This helps store the data easily and add/drop the right coordinates each reading. We felt the displayed values of the accelerometer were best represented with 4 terms used in the rolling average after extensive prototyping of different values.

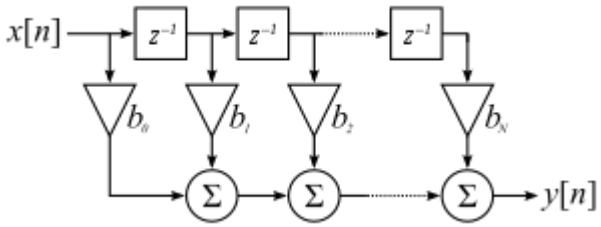

Figure 4 – FIR digital filter basic diagram

Digitally filtering also opened the door to monitoring sudden changes in the runner's motion either due to the runner dramatically speeding up or slowing down and/or stopping completely. When an average of the previous 4 values starts to exceed a set standard deviation, the average changes along with it. If there is a hiccup in the detection of a footfall or a random outlier, the average will not account for that change as the trend only occurred once or, in some cases, twice. The change must occur over 4 or 5 terms to set the new average. What this means is the music translation of this phenomenon shows a delay in accustoming the beat with the new runner's footstep rhythm. There isn't any way the device could immediately sync with the runner unless the music awkwardly jumps up and down relative to beats per second.

One other minor adjustment that must be made with the data read from the accelerometer is the normalizing of the values to zero. It represents an offset for each axis when two readings are used as reference rather than an isolated reading. If a reference point is created for all values then the algorithms for finding footfalls does not have to account for where the data of axis falls in relation to another.

$$\begin{aligned} x_f &= x_i - x_{f\text{-}1} \\ y_f &= y_i - y_{f\text{-}1} \\ z_f &= z_i - z_{f\text{-}1} \end{aligned} \quad (1)$$

For example, in the first equation above, x-final is equal to the most recently read value from accelerometer minus the x-final from before. This is done for all three axes.

C. Accelerometer Communication

Mentioned before is that the microcontroller and accelerometer communicate through a standard 4-wire protocol. The chip select for the protocol is set high so when the microcontroller needs to access the accelerometer, it will send a signal to chip select to set it low to start transmission of data whether it is to read or write. Typically there are two 8-bit clock signals that are used for both read and write. The exceptions is if there is more than one byte read from the accelerometer which can be set as high as desired. The initialization of the accelerometer sends one byte to let the accelerometer know to access the data format register (0x31) and then a byte is sent to that address afterwards in the next clock cycle which configures the range and bit-mode for the accelerometer. Other registers can be configured specifically for other features such as tap, double tap, or FIFO communication. The device for this application only requires the setup of the range and bit-mode. The interrupt pin was never established for MIM so there is no need to use the DATA_READY register for data interrupts.

For our specific application, the reading from the data registers of the accelerometer requires 7 clock cycles. The first 8-bits is to establish read-mode, decide on single-byte or multi-byte transmission, and then send the 6-bit address. The last 6 bytes are read from the registers starting from x to y then z. Since the accelerometer is 8-bit but the data is 16-bit, there are two devoted registers for each axis. The microcontroller stores the data in a static array and parses the data appropriately to each axis. The graph below displays the clock, data in and data out for the accelerometer when accessing those registers. The data specific to the figure above will always have the same address on the purple signal although the yellow signal will constantly change based on incoming acceleration values.

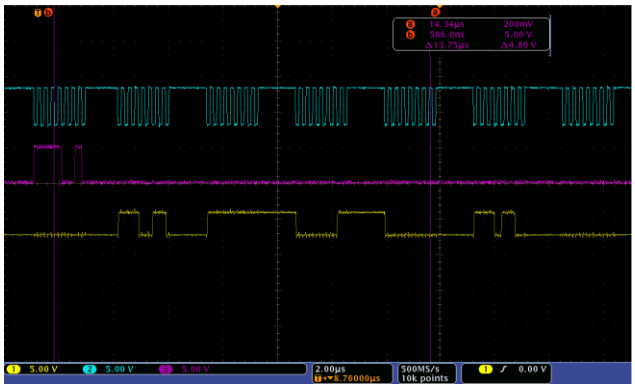

Figure 5 – Oscilloscope readout of all data registers for accelerometer

D. Data Interpretation and Algorithm Tuning

Digital Filtering helps with establishing the ground rules for how footsteps can be detected but doesn't do the actual detection. We briefly mentioned before that using the rolling averages of each axis, there can be footfall averages that fluctuate based on a continuing pattern of change in the data for that respective axis. For instance, if the rhythm is at a constant 100 beats per minute and all of sudden one value makes the rhythm much faster, it won't affect the average until there are enough values that are faster than the original 100 beats per minute. This makes sure the rhythm isn't jittery just in case there are false negatives which happen quite often. Once the average crosses a certain threshold either positive or negative, the average adjusts to the new runner's motion. On the other end of the spectrum, to avoid any false positives we created a latency or delay after a step has been established. This make sure that the music doesn't create simultaneous beats because the accelerometer algorithms detect two steps very close to each other. This occurs because the axes exceed the threshold multiple times in a step. This would normally trigger a few beats close to each other but the latency prevents any detection until after an allotted time (0.2 sec).

The figure below is an accumulation of 3 to 4 steps in any given session graphed with the x, y, and z acceleration axes. The impulse signals shown on the oscilloscope show each footstep and their time difference. In the case of the graph, the time difference between steps is 26 milliseconds which is very close to the latency delay after a step is confirmed. The threshold for detection for a footstep is, if any axis crosses over, the absolute value of 400. We designated that value after constant software testing to see what would be the most optimal threshold for detection. In the higher graph displayed, although there are multiple signals above the threshold for detection, only one footstep is acknowledged because of the latency added on to the end of the detection. The purple spiked signal can be translated to the footsteps displayed in the lower graph.

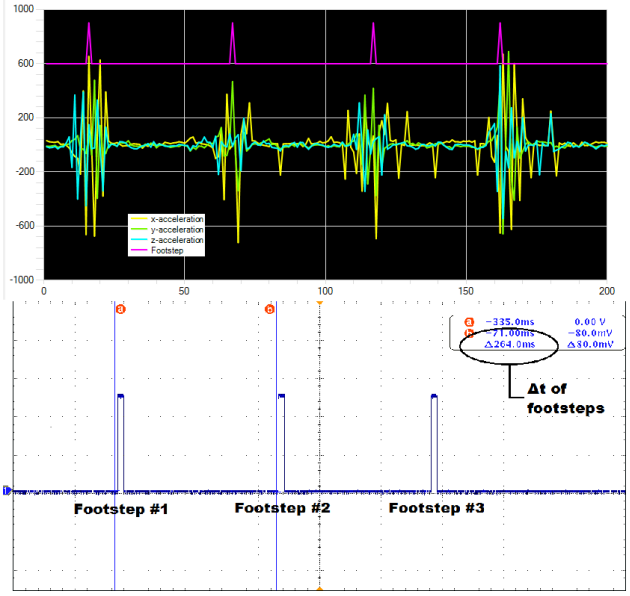

Figure 6 – Running session graphed out for x, y, z, and detected values

Figure 7 – Impulse detected footsteps with time differential

III. TEMPO CONTROL SYSTEM

Music In Motion has an audio control system that takes the tempo signal from the Atmel processor which handles the sensor input and utilizes this signal to create a modulating tempo map. It has three states, Start, Run, and Pause. The tempo input signal is a series of pulses that are not necessarily periodic. In order to create a useable and musical tempo map the control system integrates three tempo measurement systems.

The Start state is the initial state where the system has been initialized and is waiting for the first pulse. This pulse is sent when the runner first begins running. Once the first pulse is received by the control system the variable ThisBeat references a millisecond timer and stores the current time, sets the variable BeatLength to zero, and sets the current state to Play. Now that the control system is in the Play state the second tempo pulse will trigger the first down beat event and will continue to play music as long as tempo pulses continue to be received from the sensor processor. The third state, Pause is triggered in the event that no tempo pulse is received within one second. In this case the system will pause the music at the current measure. When the runner begins to move again the sensor processor will begin to send pulses again the system moves back into the Play state. At this point the music will begin to to play again from the same position.

There are three tempo measurement schemes implemented in the Music In Motion audio control system. When the control system is in the Start state and receives the first tempo pulse from the sensor processor the variable ThisBeat stores the current time. Once the next pulse is received the time stored in ThisBeat is moved to the variable LastBeat. This allows the control system to calculate the first tempo measurement. The variable InstTempo is the difference between ThisBeat and LastBeat. Utilizing this measurement allows for an accurate prediction of the exact position of the runner's next footfall. However, since there is often a variation in the timing between these footfalls this signal is does not generate a consistent enough tempo map to be used musically and the audio output will sound like a poorly practiced middle school band.

In order to ensure that the tempo map is a usable musical signal two input signal averaging schemes have been implemented. First a rolling average tempo is implemented and stored into the variable RollAveTempo. Initially, the RollAveTempo variable is set when the system is in the Start state. When the second pulse is received then the RollAveTempo variable is set to the InstTempo value, which is the time elapsed between the first and second pulses. Once the system is in the Run state the system begins taking the average values. This average is shown in Equation 2. This value is the sum of one third the InstTempo value and two thirds the RollAveTempo value.

$$RollAveTempo = \frac{InstTempo}{3} + (2)\frac{RollAveTempo}{3} \qquad (2)$$

The second average value takes the value of the four most recent pulses. This value is initialized is the same manner as RollAveTempo. When the second pulse is received the variables AveTempo1, AveTempo2, AveTempo3, and AveTempo4 are set to the InstTempo value. Once the system is in Run mode then the system will begin to take the average of these four values. When the next pulse is received it is stored into AveTempo1 and each value is shifted down and the least recent value is lost. This shifting pattern is shown is Equations 3. Once the shifting has been completed then the average is stored in the variable AveTempo. Equation 4 shows the calculation used to get the value for this variable.

$$\begin{aligned} AveTempo4 &= AveTempo3 \\ AveTempo3 &= AveTempo2 \\ AveTempo2 &= AveTempo1 \\ AveTempo1 &= InstTempo \end{aligned} \qquad (3)$$

$$AveTempo = \frac{(AveTempo4 + AVeTempo3 + AveTempo2 + AveTempo1)}{4} \qquad (4)$$

In order to calculate the variable NextBeat, which is used by the audio output controller to quantize the output data transmission, all three of these tempo measurement values are utilized. This calculation is shown in Formula 7. Empirical testing has shown that the most musically useful tempo map is generated by using one fourth the InstTempo measurement, one fourth the RollAveTempo measurement, and one half the AveTempo measurement with a twelve millisecond offset. This offset value is used to counteract any latency in the Music In Motion system.

$$NextBeat = \frac{InstTempo}{4} + \frac{RollAveTempo}{4} + \frac{AveTempo}{2} - 12 \qquad (5)$$

IV. AUDIO CONTROL SYSTEM

The Music In Motion audio control system serves two primary functions. First, the system quantizes each pulse into smaller musical subdivisions. Each beat, which is represented by the NextBeat variable, is a quarter note musical value. The audio control system further subdivides each beat into four sixteenth note values as well as two eighth note triplet values. This allows for a total of six rhythmic subdivisions per quarter note pulse. Second, the musical song structure is stored and played back according to the tempo map. Finally, the system controls the audio synthesizer to generate the musical output. The audio output is generated by the VS1053 audio synthesizer and is controlled through MIDI protocol.

In order to quantize audio output the master clock timer variable, BeatLength, is utilized to measure the timing of each beat. Six functions have been developed that subdivide each pulse. These are Down, Six1, Trip1, Eight, Trip2, and Six2. The divisions between Down, Six1, Eight, and Six2 represent sixteenth notes while the division between Down and Eight represent and eighth note value. The divisions between Down, Trip1, and Trip2 subdivides the beat into triplet values. As the counter BeatLength increments each of these functions are called after the appropriate time has passed.

These functions also store the note on and off control data that make up the musical structure. There are two primary data structures that store this musical structure. The struct Rint is used for rhythmic instruments and Minst is used for melodic instruments. The Rinst struct is comprised of two arrays, two pointers to those arrays, a counter variable, and an ON variable. The Minst struct is

comprised of three arrays, three pointers to those arrays, a counter variable, and an ON variable. The two arrays stored in the rhythmic structure store note length and velocity. These instruments do not require a note value as each rhythmic instrument only plays one sound. There are four rhythmic instrument structs which are KIK, SNARE, HAT, and TOM and are used to simulate the standard rock style drum kit. MIDI control protocol allow for sixteen separate control channels that can be addresses with a four bit word [2]. Each of these instruments utilize two of these channels so that two instrument sounds can be layered together to create a more versatile and dynamic tonal pallet. The three arrays stored in the melodic instrument structures are note value, note length, and velocity. There are three melodic instrument structures which are BASS, KEY, and LEAD. Each of these instrument structures have two corresponding functions that are used to turn the current note on or off.

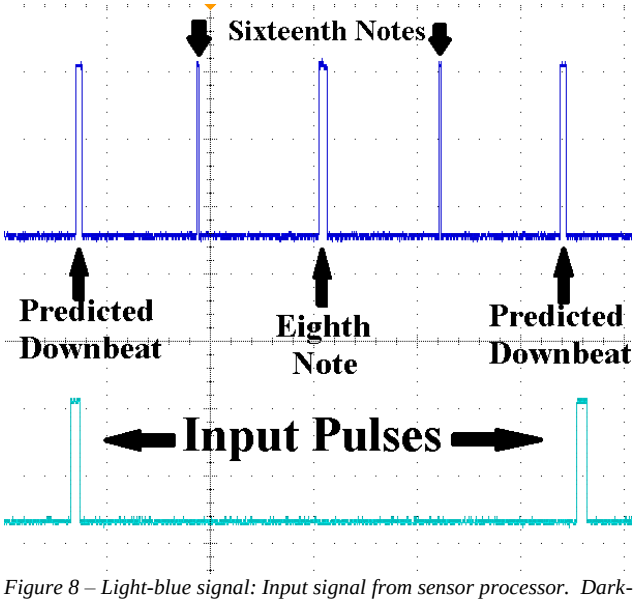

Figure 8 – Light-blue signal: Input signal from sensor processor. Dark-blue signal: Quantized output tempo map.

Two counter variables stores the current song position and there are one hundred twenty eight measures, the length of most standard pop songs. A beat counter is incremented with every down beat and every four beats the measure variable is incremented. Inside the quantization functions, Down, Six1, Trip1, Eight, Trip2, and Six2, a series of conditional statements will call either the note on or note off functions for each the instruments. These on and off controller functions are passed the MIDI note values and velocity values that are sent to the VS1053 and these notes are either turned on or off. MIDI control requires that each note on message be followed by a note off message. If the conditional statements turn one instrument then each time a sixteenth note passes the struct counter is incremented. When this counter is equal to the value that the note length pointer is pointing to in the note length array then the note off functions is called for that instrument. The note velocity and value array pointers are incremented within each note off function so that the next time the on function is called the next note on function is called the new values are passed to it.

VI. GPS AND ALERTS

The GPS module is a subsystem of Music In Motion and plays a secondary role to the accelerometer sensor. The GPS tracking allows for distance and speed calculations use to inform the runner how far they've ran and what speed they are going. Once all the values are calculated for notification, a unique sound bit is played either during a key distance marker or speed. Since the user is cannot adjust the milestone distances/speeds, there are defaults that are used in every run.

For Music In Motion, the GPS module of choice is the Antenova M10382. It has up to 50 channels which is good enough for a simple project like MIM the precision doesn't need to be top notch as routing is not needed [3]. The patch design is standard for GPS modules but requires an initialization process before the GPS can receive coordinates [4]. The process of the microcontroller polling data from the GPS is easy due to the simplification of the UART communication.

A. Initialization

The GPS coordinates do not automatically send to the module as soon as MIM is powered on. In terms of hardware, the device must be facing upwards in an outside area in order to pick a fix from satellites. Once the fix is made, the coordinates are sent at a set update rate. If the fix is not established and the device begins to move, the fix will not be established and there will not be any notification of milestones for the user.

There are three default update rates: 1 Hz, 5 Hz, and 10 Hz. Music In Motion uses the 1 Hz rate as it is not too intensive on the microcontroller and works well with other clocked software in the project. The software originally designed for Music In Motion had us checking the GPS every second just like the update rate. We found this interfered with the constant checking of accelerometer data so we had to alter when the GPS needed to be accessed. The best moments for communication is when the latency occurs after a footfall and the accelerometer is not needed for 2 milliseconds. During this time, the microcontroller can poll the data and use the values for the various

milestone calculations. By the time the accelerometer is needed again, the entire GPS software cycle is completed.

B. Calculations and Application

The GPS data received from any satellite detected must be parsed in order for the microcontroller to use it for milestone calculation. There are two modes of data reception the GPS module can receive: GPRMC and GPGGA. The minimum required is GPRMC as it provides latitude, longitude, fix result, and speed which is what is used for the project. The GGA was originally used to the ability to find the altitude as well. The change was made to just RMC when the haversine distance calculation based on Earth's radius and altitude changes became too complex for the simple application of finding small distance values. Once the data is acquired and parsed, the values are used in distance calculation. As for speed, since the GPS outputs user speed directly, there isn't much to configure except the value is received in knots which must be configured to miles per hour for the other equations. The coordinates received are converted from standard float values to more standard degree format values. Equations 6 [5] below shows the formula for converting these values. The cardinal directions make the coordinate values either positive or negative.

$$\begin{gathered}\text{North / East} \rightarrow \text{Positive;} \\ \text{West / South} \rightarrow \text{Negative;} \\ \text{Minutes} = \text{Latitude}_{\text{initial}} \,\%\, 100 \\ \text{Degree Minutes} = (\text{int})\,(\text{Degree Minutes} / 100) \\ \text{Degree Decimal} = \text{Degree Minutes} + (\text{Minutes} / 60)\end{gathered} \tag{6}$$

Figure 8 below shows the translation between the strings of data received from the satellite and the conversions made to make them usable for calculation. Since all the information is parsed and used for calculation it must be done immediately after a footstep as mentioned before because of the demand of resources from the microcontroller.

```
Time: 19:37:18.0
Date: 16/7/2014
Fix: 1 quality: 0
Location: 2836.0710N, 8111.9179W
Speed (knots): 0.45
Angle: 0.00
Altitude: 0.00
Satellites: 0
$PGTOP,11,2*6E
$GPGGA,193719.000,2836.0712,N,08111.9187,W,1,3,2.64,181.1,M,-31.1,M,,*6B
$GPRMC,193719.000,A,2836.0712,N,08111.9187,W,0.43,0.00,160714,,,A*7F
```

Figure 8 – Data output from GPS and data conversions

In the software, there is only two coordinates needed at a time, so when a coordinate x_n is retrieved via satellite, the coordinate x_{n-2} is cleared from memory. The initial option of calculating was comparing two consecutive GPS coordinates. The basis for comparing distance between two coordinates is using the haversine algorithm. Equations 7 below shows all the calculations required to find the flat-line distance. In order to account for altitude change, simple trigonometry is applied using the last equation below.

$$\begin{gathered}a = \sin^2(\Delta\text{lat}/2) + \cos \text{lat1} \cdot \cos \text{lat2} \cdot \sin^2(\Delta\text{long}/2) \\ c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{(1-a)}) \\ \text{distance} = \text{Earth\_Radius} \cdot c\end{gathered} \tag{7}$$

The problem with using the haversine equations for this project is the application is on too much of a small-scale. The equations can convolute the small distance of when a runner moves a couple of feet. We decided to switch to simply averaging the two most recent velocities and finding the distance after a period of time. Equation 8 below shows the distance calculation using the speeds given by the GPS data over the time interval between the readings.

$$\begin{gathered}1 \text{ knot} = 0.514444 \text{ meters per second} \\ \text{Distance} = [[(V_i \cdot 0.514) + (V_{i-1} \cdot 0.514)] / 2] \cdot \Delta t \\ \text{Distance} = \text{Distance} \cdot 0.000621371 \text{ “Miles to Meters”}\end{gathered} \tag{8}$$

V. PCB

The PCB for the Music in Motion system was created using Eagle CAD. This was one of the greatest challenges of this project due to the learning curve of both the software and the design of a completely electronic device with the lack of real world experience.

A. Design

Reference designs were used for the peripheral subsystems which were then reverse engineered in order to meet the design specifications. The development boards often included more circuitry than what was needed, such as redundant voltage regulators. Therefore, everything that was irrelevant was omitted from the design. Once each individual subsystem was designed to meet the project needs, they were integrated with one another according to the prototype. While the PCB was being designed, the prototype was simultaneously being created in order to design the final project in accordance to physical connections that have actually demonstrated to function as expected. This method drastically reduced the number of anticipated errors.

The figure below shows the final design of the MIM PCB. It is a four-layer board which has the following dimensions 3.920” x 3.423”. This board contains

2xAtmel328 processors, one for sensor control and one for audio control. The sensors include the ADXL345 (accelerometer) and the M10832 (GPS), which will keep track of user milestones and create the tempo of the song through signal processing algorithms. The second Atmel328 controls the audio engine and VS1053B (audio codec IC) which actually generates the music. Moreover, there are 80 additional parts on this board. Most of these parts are passive components such as resistors, capacitors, inductors, and inductive ferrite beads. Nevertheless, included in those 80 additional parts are voltage regulators, the USB DFU bootloader, and the USB connector.

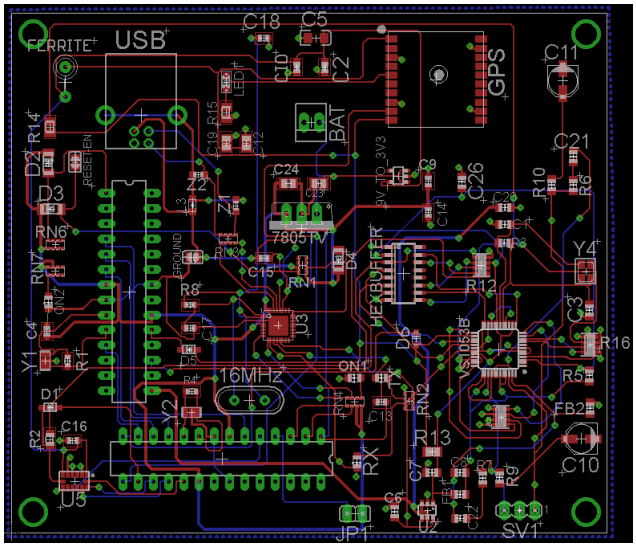

Figure 9 – Music in Motion PCB

B. Assembly

The PCB was manufactured by Advanced Circuits using the $66 student special. The board had minimal errors, none of which were electrically related. The only errors were in the silkscreen layer due to the size of the font. Advanced Circuits warns that if the silkscreen text is less than 0.005" in width, then it will not print; this is exactly what happened. This presented a challenge for the assembly portion of the PCB due to the lack of reference designators. This was fixed by redoing the silkscreen layer with larger text and boldness ratio and taking pictures of this layer for the assembler. Quality Manufacturing Services (QMS) populated the board using a box of parts and board provided to them. The board had both surface mount parts and through-hole parts, although the surface mount was only one sided. A stencil was provided to QMS that was created by Advanced Circuits in order to solder paste the board for assembly, however it proved to be unnecessary as the board was done by hand.

THE ENGINEERS

Zoilo Boehme is graduating with a degree in Electrical Engineering. His professional interests include inventing, electronics, and hardware. He intends to pursue product engineering and one day have his own company. His other goals are to become well adept at investing and following technological trends.

Eric Hoofnagle is graduating with a degree in Computer Engineering (CpE). Eric plans on staying in Orlando and is currently exploring employment options and considering applying to UCF to pursue a master's degree. Eric has interests in application software, embedded application, and DSP.

Lane Starratt is an Electrical Engineering major who has focused his studies on FPGA and Embedded system design. He is also a musician and sound designer who has designed several systems and shows for Theatre UCF. His career goals are to design embedded audio systems and synthesizers.

ACKNOWLEDGEMENT

The authors wish to acknowledge the assistance of Dr. Richie for guidance for two semesters. We would like to thank The Boeing Company for sponsorship. Finally, thanks to Sam Hanna and Quality Manufacturing Services for assembly of parts for the PCB.

REFERENCES

[1] Zhao, N. (n.d.). Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer. Analog Dialogue 44-06, June (2010) 1.

[2] Midi Manufacturers Association. (n.d.). Midi Protocols. Retrieved from Midi.org: http://www.midi.org/aboutmidi/tut_protocol.php

[3] Taoglas - antenna solutions. (n.d.). Internal GPS Active Patch Antenna . Retrieved from http://www.taoglas.com/images/product_images/original_images/Internal%20GPS%20Active%20Patch%20Antenna(APN-13-8-002.A).pdf

[4] Mando. (n.d.). GPS Accuracy. Retrieved from https://learn.sparkfun.com/tutorials/gps-basics/gps-accuracy-

[5] Mando. (n.d.). GPS Message Formats. Retrieved from https://learn.sparkfun.com/tutorials/gps-basics/message-formats