MUSIC IN MOTION

University of Central Florida
EEL 4915C
Senior Design II
Summer 2014
Group 7

Zoilo Boehme Eric Hoofnagle Lane Starratt

Table of Contents

Table of Tables

Table of Figures

1 – Music In Motion Executive Summary

Imagine that you are running through your neighborhood in the afternoon. The sun is just behind the clouds and the air is crisp and refreshing. With each step you take, you become liberated. You begin to immerse yourself in the experience as the music you listen to takes you over. Each powerful stomp of your foot falls perfectly on beat with the song, you are synchronized. The atmosphere of your run is matched by the sound waves in your headphones, your body waves back to the music. You notice that it is no longer guiding you, somehow you are in control, and you are composing this music with the movement of your body. You are an instrument and the freedom that comes with this experience is your melody.

Music in Motion is a concept for a running device which creates and plays music while the user runs and provides the feedback that tracks progress and help one to perform. Essentially, it is a concatenation of a fitness monitor, biofeedback device, and a music player. This device is practical. Various metrics are be sampled, some of which are biological, in order to provide a more complete description of the user's fitness session than anything that exists on the market today. This device motivates the user. The collected data is interpreted by a number of algorithms which quantify each fitness session, tracks progress over time, and congratulates the user for breaking previous records in real time. This device is entertaining. Simply by running, it lets the user truly listen to his or her body for the first time and it does so in the form of music. This supplementary feature adds an element of novelty to the device which is both interactive and entertaining. Thus, the biofeedback is presented via two entirely distinct channels: biological data and music.

The intention is to provide detailed and useful feedback to the user in an entertaining manner. This device uses various metrics, such as heart rate, running distance, and running speed. We use these metrics to control the creation of the music and also in order to give the user real-time feedback. This real-time feedback is given in the form of a verbal congratulations for breaking certain milestones that our MIM device monitors and keeps track of. We incorporate the following milestones: top speed, farthest distance, step-counter, and best race (mile, 5k, 10k, half-marathon, etc.). In addition, the device analyzes the runner's cardiovascular response to the fitness session so as to quantify their current physiological condition and progress. This is executed using the user's heart rate, distance, and speed data which is processed by an algorithm of our design.

The musical biofeedback is purely for entertainment and novelty. Using a combination of signal processing and musicianship, we turn the user's biometric data into music. A digital audio sequencing synthesizer is integrated into the device which receives control signals that are generated by the microprocessor based upon input signals from an accelerometer and a GPS. A series of matrices

store the musical information that is used by the synthesizer to create the audio output.

The tempo of the song is directly controlled by the speed of the runner. This is achieved by detecting the impulses received by the accelerometer, filtering the data, processing it, and continuously feeding back the frequency of the signal to the microprocessor which creates a variable tempo clock that enables the audio sequencer to stay in sync with the pace of the runner. A digital control system has been designed to perform this task. The appearance of this system is that the music coming from the user's headphones is completely synchronized with and controlled by their running. This manifests in the melody, rhythm, and tempo of the song.

Music in Motion is a device similar to an mp3 player with a twist. This over-arching ideal allowed us to scale the device with multiple functions and applications with sole focus on the running/workout aspect. With design features that can be easily added to the device, the use and novelty doesn't have to end on release of the initial design. In this way, the practicality comes from runner's accessibility to an exciting, fulfilling experience every time out rather than listening to the same downloaded music over and over again. Music in Motion allows the user to finally participate in an active experience with a music player. No longer do our users passively run to music, they co-create the entire journey to cardiovascular greatness.

2 – Description

2.1 – Project Motivation

When brainstorming ideas for the project, many factors were involved in the decision to design the Music In Motion System. Some of these include practicality, difficulty of implementation, pricing, and business application. Other secondary factors including how much enjoyment the designers would get from the process of designing and constructing the project as well as how serious the application would be for the user (casual vs. professional use). Ultimately, the Music in Motion concept balanced many conflicting aspirations for the project among the group members.

Initially the Music in Motion system was envisioned as a means to provide the user with a unique experience every time he or she goes on a run. Simply put, the runner should get lost in the music. One of the best ways to go about this is to synchronize oneself to the music. But what does this look like? What does it feel like to be the music while you run? These are the questions that were addressed while designing the concept of Music in Motion. Initially, a vision of the user end experience was developed and a variety of implementation possibilities were discussed. One of the ways for the user to get lost in the music it is by synchronizing the runner's physical body. After all, is this not what dancing is? Additionally, because the main rhythmic bodily movement of a runner is the powerful stomp of their foot as it lands on the ground, we chose this as the motion to synchronize. The beautiful part about this is that one does not need perfect rhythm, nor do they need to be consciously attempting to synchronize their run with the music. MIM system matches its tempo to their own running pace automatically. Going one step further, this device creates music based on the runner's current experience. The person's body has now become an instrument that the MIM device uses as input in order to create music.

With the anticipation of designing a unique idea finalized, the process of researching and conceptualizing the specifications required to bring this dream to life. These specs included the sensor control system, tempo control system, audio engine, PCB, power requirements, user end aesthetics, and more. A balance was struck between bought ICs, such as GPS, and custom subsystems that were originally designed for the Music In Motion system such as digital audio converters and signal processing microcontrollers. This project was designed to be modular in fashion, such that there is flexibility and scalability. The main motivation, ultimately, is to create interconnections between cumulative engineering knowledge and practical engineering procedure. Additionally, this project ties up the loose ends in practical understanding in order to put the cherry on top of this undergraduate engineering experience.

2.2 – Goals and Objectives

From the most general standpoint, the main focus of this project is to create a device which provides the user with a completely unique running experience. This experience is one in which is currently unavailable on the market. It is one where the user can automatically synchronize with the music simply by running. In addition, the music will be created while he or she is running. This being the case, the music will take a specific direction based on the inputs from the running metrics. The end effect will be that the music will be somewhat of a soundtrack to the user's run and even the atmosphere will match the cardiovascular intensity. This effect will add a greater sense of immersion into the run giving it the intensity it deserves. It will help the user stay present in the moment and focus their efforts on the task at hand – running.

That being said the main function that is implemented is the feedback control system which synchronizes the music. This is accomplished by collecting the impulse data from the runner's pitter-patter via an accelerometer. This impulse data is low-pass filtered and then used to control the tempo of the song. The impulse train has a certain frequency, which we 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. This feature is implemented as the main function of the MIM device and other secondary features are added to round out the experience such as the GPS for distance calculation and potentially an application alongside Music In Motion. One concept that was considered for future implementation was a module for cardiovascular detection throughout the run. This module would take in heart rate data. The metric would be used to quantify the user's cardiovascular health. It is a similar concept to the stress test used at hospitals to evaluate a patient's cardiovascular system. There are bodies of equations to be researched that contain within them estimates of one's response to fitness stimuli whereby the input variables are the measured metrics. We did not have enough time to implement this feature but given a month or so, this would have been used along with the Android application.

There were several goals and objectives for the presentation of the device itself. The hardware needed to be a compact package with minimal wires, weight, and dimensions. This is important to the user because they do not want to be bogged down while exercising simply because of baggage they are lugging around, especially when competitors (such as the Nike Fit or IPod) are of a small size and weight. Nevertheless, this is merely a small-term project. That being said, if this was a final product, it would be much more compact and designed more for efficiency and small size. Although it was our intention to minimize the size as much as possible, there was no expectation that the design would be a ready-to-mass-produce device in two semesters in senior design. The size was not an ultimate determining factor of our success.

This device incorporates an elastic wide-width belt attached to the back of the MIM device behind the wooden frame. This design has a pocket for the external 9-volt battery along with a Velcro attachment feature so the belt is adjustable and secure. We found the device worked best when used as a belt attachment based on the way a runner moves. With this design, it was apparent that the most aesthetically and kinesthetically pleasing design is one that minimizes the appearance of the sensors. We have a solution for this. The accelerometers are integrated within the device, on the PCB, because the pulse can be detected from any part of the body. The typical runner has an extraneous motion to their running, which causes some junk data in the accelerometer; however this is filtered out by using low-pass filters. This is because the signal noise can be attributed to vibratory motions and are therefore in the high frequency bandwidth.

The aim of the Music in Motion system was 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 aerobic 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 sophistication and elegance of the user interface was to be in the Android application. When the user arrives home from their run, they could directly download their session from the device and review it on this application. They can do this by simply interfacing with the MIM device via USB. The application represents the data graphically and numerically through charts and tables. The data needs to be interpreted for the user and be as simple as possible while still presenting all of the crucial information. The trends representing all sessions will quantify the user's progress in an aesthetic manner. This will encourage the runner to use the product more often and thus provide incentive to run frequently in order to measure their progress. Progress was to be tracked in terms of milestones such as running distance and speed. Also, certain key distances can have record times, such as 5k, 10k, half-marathon, etc. Ultimately, the Android application was scrapped at the end due to prototyping communication and USB inactivity on the PCB, as well as simply running out of time and focusing on the main features of Music In Motion.

Below is a breakdown of each initially intended integrated-subsystem in our design and what goals we had at the end of design for each module.

Accelerometer – The MIM system uses accelerometer data to determine when the impact of the footfalls occurs. The tempo control section uses runner's previous velocity data to predict when the next footfall happens and places the musical beat at the time of the prediction then makes adjustments based on the difference between the actual footfall and predicted one. There is a feedback control that allows for adjustment if the user slows down or speeds up. With

enough axes and correct sensitivity of motion, the rhythm of the runner's footsteps are detected without an interrupt of the experience in the eyes of the runner.

Android Application – The android application is obviously not in the design of Music In Motion but rather downloaded on to an android smartphone. The user must open the application, input the type of run they wish to have, and click the send button to the device if connected. The information is sent and should notify the user when finished with transmission. The goal of the application is to provide extra features to the runner not included in the device itself, making the running experience more unique every time out.

Filters, A/D converter, and D/A converter – The signals created by the accelerometer and the music outputted to the audio jack both need to be filtered in some way. Much of the device can handle the conversion without the need of added filter (the individual modules/processors handle the filtering for you) but there are spots of data transfer that must use A/D or D/A conversion depending on the function. These converters should work exactly as intended every time so most if not all data is encoded or decoded correctly.

GPS Module – The GPS module is included in the device to keep track of latitude/longitude strings as a way of letting know when the user reaches a certain location. The GPS sends data constantly to get a good idea of precise distances that the accelerometer would have trouble handling. The GPS store strings within the sensor microcontroller. These are used for distance computations for the milestones as well as after the run is over when the raw data is sent to the application (if connected). GPS coordinates are consistently updated and stored based on most recent and checkpoint coordinates in conjunction with the time and distance during the run. Coordinates are clocked in and allocated space for the time being until the data is no longer needed. The GPS module in the device is accurate enough where the correct direction and distance are compiled, and the user doesn't feel that the distance they ran accompanied with the distance time isn't correct. This is very important for experienced runners who need precise distance and time so the progression of the run feels authentic.

Sensor Microcontroller – The microcontroller for the design controls three separate ICs (tempo control microcontroller, accelerometer, and GPS) and handle the main software coding for Music In Motion in the memory on-chip. The main factors with choosing the microcontroller is making sure it has enough pins for I/O to connect to the accelerometer, GPS module, and other microcontroller. The clocking is managed by the clocks of the sensors. There is no clocking required between microcontroller communications as the GPIO is based on interrupt UART protocol. Finally, the sensor MCU handles most signals from the sensors and the transfer of data as a channel from the sensors to the tempo control subsystem. The ultimate goal of the microcontroller was to have a high enough frequency to handle all ICs without bottlenecking or deadlocking any sections of data passing through the microcontroller.

USB Port – For essential needs, the use of the USB port is to allow transfer of data to and from the Android application used on any Android phone or tablet. The variables chosen by the runner on their phone translate precisely to the expectations of the music when out on the run. There would have been a separate mode for the allowance of data transfer to and from the sensor microcontroller on the PCB to the Android device without interrupting the run.

9-volt Battery Power System – A standard 9-volt battery is linked up to the device to power all active ICs. There are regulators throughout the device to manage the power levels supplied in order to not fry or damage any of the parts. The regulations include 9V to 5V, 9V to 3.3V, and 5V to 3.3V and 1.8V. The battery also lasts a long enough period of time so the runner can have multiple sessions without changing out the battery.

1/8" audio output – Finally, in order for the user to actually use the device, a 1/8" headphone jack is on the side of the device for the user to plug in their headphones and listen to the music created. The only objective of the audio output is that it works all the time during execution and produces the sound intended to be output from the FPGA.

2.3 – Specifications and Requirements

The specifications/requirements of each IC or integral piece of Music In Motion system were mostly met when the final design was completed. Certain requirements are weighted in importance more so than others but all were attempted and meant for integrating by the end of the design process. The following tables give the quantified specifications of each subsystem respectively.

Table 1 describes the specifications for the accelerometer. For this project all axes are used in order to determine the changes in the signal with respect to typical and atypical running motion, therefore the accelerometer requires three axes.

Accelerometer	
Category	**Requirement**
Number of Axes	3
Resolution	8 bits or more
Maximum Swing	±2g → ±8g
Dimensions	15 mm x 15 mm x 5 mm (max)

Table 1 – Accelerometer Requirements

Table 2 describes the content of the Android application. It summarizes the size and user interface in relation to its algorithms. For example, the number of run settings refers to the number of permutations of user options such as 5k, 10k, and

marathon. The runner would have been able to utilize this application to customize the Music In Motion system to their own personal needs.

Android Application	
Category	**Requirement**
API	Android 2.2 or higher
Application Size	Less than 20 MB
Splash Screen Wait time	5 seconds
Number of Run Settings	At least 10
Number of Saved Settings	3
Transfer Time	No more than 8 seconds

Table 2 – Android Application Requirements

Table 3 lists of microcontroller requirements, which is the most critical point of this design in terms of power, performance, cost, and size. With these requirements, it will be possible to satisfy the goals and objectives.

Sensor and Tempo Microcontroller	
Category	**Requirement**
Speed	At least 50 MHz
On-Chip Memory	128K Words x 16-bit
Communication Channels	SPI, UART, I^2C, PWM capability, General I/O Pins
Supply Voltage	3.3 V

Table 3 – Microcontroller Requirements

Table 4 quantifies the bandwidth, channels, precision and communication protocol for the GPS on the Music in Motion PCB. It is used by the microcontroller to determine the runners speed, direction, and distance as well as to determine the control signals that will control the audio output from the tempo control. The interface is the most important aspect in context to system level integration due to the available pins on the microcontroller. It is important to keep track of which subsystems are using what protocols in order to have the communication system working.

GPS Module	
Category	**Requirement**
Number of Channels	20 - 88
Minimum Horizontal Position Accuracy	8 meters
Interface Type	UART

Table 4 – GPS Module

Table 5 describes the requirements for the actual Music in Motion device. The main factors that contribute to an enjoyable user experience are listed and specified. These are important because the user wants to be able to run freely without being weighed down asymmetrically by a device that is supposed to be of assistance. The user wants to seamlessly integrate with the running device and these specifications describe, in quantitative terms, how to accomplish this.

Size of Device	
Category	**Requirement**
Height	5 in.
Width	5 in.
Depth	1.25 in.
Weight	5 oz.
Strap	Elastic wide-width belt
Material	Sturdy, compact plastic enclosure w/ wooden frame

Table 5 – Size Requirements

Table 6 describes the Music In Motion device final I/O requirements. A USB port is utilized to program and update the device during the final testing stage. It also serves as the user's interface for customizing the device when synced to the Android application.

USB Port	
Category	**Requirement**
Port B (Device Port)	5 in.
Size of USB Port	Micro-USB
Version Compatibility	USB 2.0

Table 6 – USB Port Requirements

2.4 – Final Budget

In the original design plan a budget of $700 was determined. A funding request was submitted to Boeing for this amount. However, the full amount was not available and the funding amount received is $485. By utilizing design elements that are currently in stock, we were able to budget the project to a much more reasonable amount that is closely equal to the amount received by Boeing. The new total projected was around $483.00. The circuit components and miscellaneous either range too much in price to get a good estimate or the amount is so minute that it is not worth accounting for in the graph. The extra parts due to readjustment of the design or damaged parts increased the total budget to around

$580. Table 7 below shows all parts and respective pricing for everything purchased for Music In Motion throughout the two semester time period.

Music In Motion Budget						
ITEM	**FUNCTION**	**PURPOSE**	**OBTAINED**	**AM**	**PRICE/ UNIT**	**ITEM COST**
Digilent Basys2 - 250	FPGA Dev	Prototyping	In stock	1	$0.00	$0.00
TI MSP430 LaunchPad	Microprocessor Dev	Prototyping	In stock	3	$0.00	$0.00
TI C2000 Piccolo Development Board	MCU Dev	Prototyping	TI	1	$22.00	$22.00
ADXL345 Accelerometer Development Board	Accelerometer Dev	Prototyping & Final	Adafruit	1	$22.00	$22.00
ADXL345 IC	Accelerometer IC	Final	SparkFun	2	$9.95	$19.90
Antenova m10382 Development Board	GPS Dev	Prototyping	Mouser	1	$24.35	$24.35
Ultimate GPS Development Board	GPS Dev	Prototyping	Adafruit	1	$39.95	$39.95
FGPMMOPA6H IC	GPS IC	Final	Adafruit	1	$29.95	$29.95
VS1053B Development Board	Audio Codec Dev	Prototyping	Adafruit	1	$24.95	$24.95
VS1053B IC	Audio Codec IC	Final	Adafruit	1	$12.50	$12.50
Arduino Uno Rev3	MCU Dev	Prototyping	RadioShack	3	$29.99	$89.97
USB Wire A to B	Connector	Prototyping	RadioShack	2	5.89	$11.78
PCB Manufacturing	Board Printing	Final	Advanced Circuits	1	$107.49	$107.49
PCB Stencil	Assembly Tool	Final	Advanced Circuits	1	$134.00	$134.00
PCB Parts	Circuit Components	Final	Newark/Digikey	80	N/A	$43.32
Total						**$582.16**

Table 7 – Budget Spreadsheet

2.5 – Roles and Responsibilities

In general, responsibilities for this project are assigned in regard to the strength of each group member. The write-up for each section of the paper is assigned based on the parts that were assigned. Early on, strengths and weaknesses were discussed for this project. It was necessary work on bolstering certain areas and covering up others. Throughout the process, it became apparent that certain tasks, that originally had been assigned one person, were more appropriate for another group member, either due to the knowledge base of the newly assigned individual or too much to cover for one person in the group to handle that section. With Music In Motion having such clearly defined components and clearly divided sections of work, either through software or hardware, it became apparent that assigning a concept or physical component would be fully encompassed by one person. So if someone in the group wanted to work on the accelerometer, that would include researching accelerometer technologies, finding the right one for the project, figuring out the schematic for the part, and how the accelerometer would be used in conjunction with the other pieces of the device.

Here are the flexibly assigned parts/concepts assigned to the following group members displayed in Table 8.

Zoilo Boehme	Eric Hoofnagle	Lane Starratt
D/A Conversion	GPS	Tempo Control
Power	Sensor Microcontroller	Signal Manipulation
Accelerometer	USB Technology	Audio Engine
Audio Amplifiers	Software Architecture Overview	Hardware Architecture Overview
Sensor Data Transfer	Java Application Programming	Modulation Techniques

Table 8 – Project Member Responsibilities

By staying focused on these individually assigned sections and having open and clear lines of communication, the Music In Motion system was completed with utmost professionalism and minimal amount of issues. Deadlines were met efficiently and mostly ahead of schedule. This gave us time to fine-tune and reorganize anything about the project we put off during initial procedures.

2.6 – Milestones

The milestones for this project were designed such that the Music in Motion device is ahead of schedule in relation to the rest of the senior design class. It has

become apparent that it can be dangerous to aim for the end. This is mostly due to unforeseeable circumstances, such as problems with system level integration. In order to overcome this, the following milestone chart (Table 11) has been devised which helped facilitate the completion of this project before its scheduled due date.

Although the schedule in Table 9 are simply guidelines to the commitment in each step, they were followed with integrity in order to keep the professional standard of the group intact. Each group member relied on the other so with all members committing their equal share, all stages were completed on time. Some of the sections such as prototyping and designing varied in time completion based on project complications and when parts were bought and received. A positive side effect of finishing the project over the two semesters is gaining the knowledge of anything researched or designed and can be applied to future employment.

Music in Motion Milestone Chart			
Senior Design 1	**Jan 27 – February 17**	**February 18 – March 31**	**April 1 – April 28**
	Stage I	***Stage II***	***Stage III***
	Brainstorming	Prototype Designs	Gather Parts from List
	High Level Design	Budget Revisions	Initial Prototyping
	Sponsorship and Bill of Materials	Parts List	Finalizing Project Document
	Initial Documentation	Software Initialization	Begin System Level Integration
Senior Design 2	**May 12 – June 1**	**June 2 – June 21**	**June 22 – July 26**
	Stage IV	***Stage V***	***Stage VI***
	Complete Software Requirements	Construct Device from Final Design	Software Level Testing
	Finish Prototyping	Synchronize Software and Hardware	Full Functional Testing
	Final Design Revisions	System Level Testing	Compile Documentation for Presentation
	Compile Notes and Information for Presentation	Hardware Level Testing	Minor User End Revisions

Table 9 – Milestone Chart

Table 9 splits up the phases into six subsections which are approximately three weeks each. These subsections proved to be even more imperative to adhere to in Senior Design II because of the limited resources such as time and cost.

3 – Research and Background

3.1 – Relevant Technologies

In order to aid in the discussion of project research and expedite the design process, it is important to research technologies and projects that are relevant to Music in Motion. This foundation will help establish a point of reference for which design concepts can be loosely based. The most similar devices for which the MIM project can be compared to for its common qualities of impulse detection are pedometers and running devices.

3.1.1 – Pedometers

Pedometers are a relevant product to the Music in Motion device. The interrelation is such that both technologies rely on being able to sense when the user's foot hits the ground. A pedometer is a bit more flexible in design, for the most part, than the MIM device. This is because pedometers do not require an incredibly sensitive method of sensing foot impacts. On the contrary, this project needs to sense the impact and immediately process the signal and use it to control the tempo of the audio. In either case, the signal needs to be filtered, converted, and processed. The signal for running and walking are very similar, therefore the typical protocol for handling the pedometer's signal processing will aid in the understanding and design of the MIM device sensors. Below in Figure 1 is a picture of a typical pedometer signal.

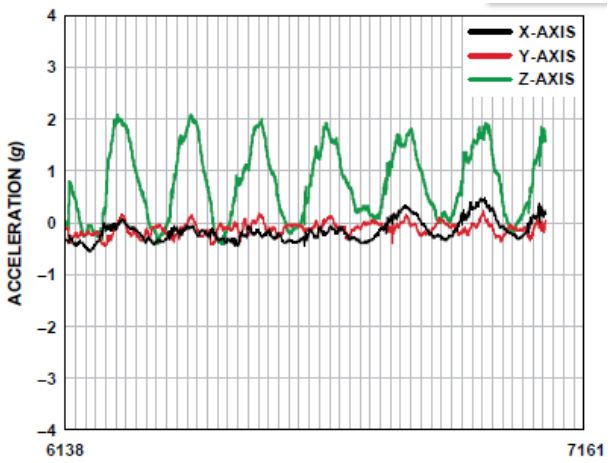


Figure 1 – Discrete Time Signal of Two Footsteps

For pedometer applications, the walking signal is sampled for analog to digital converters at around 100 Hz. This is much faster than what is completely essential, due to it being at least 50 times faster than the signal itself. Also, 100 Hz minimizes the amount of data that needs to be handled, which helps in limiting allocation of microprocessor resources (Zhao).

3.1.2 – Running Devices

Running devices also include the main principle of the pedometer, detecting foot impacts, except at a higher frequency. Running devices can easily manage using the same sampling rate as a typical pedometer as well. The main difference between the two is that the running device usually also tracks a person's distance, velocity, and time. Many running devices take care of these additional features using GPS or an accelerometer. These two options seem to be the most practical and efficient methods to solve this problem. The use GPS and accelerometer for this function will be investigated in the design section of this paper with explicit detail.

3.1.3 – Tempo Effects

Most time based musical instrument effect modules include a feature that enables the user to adjust timing variables on the fly. There are three general approaches to this and the Music In Motion system utilizes elements from each of them in the design of its Tempo Control System. Delay based audio effects use a tap tempo detection method. Loop based tempo effects use a tempo length detection method. Tempo stretching effects use complex algorithms to re-synthesize prerecorded audio and adjust playback speeds in real time without affecting pitch information.

Temporal audio effects such as reverb and delay sustain or repeat sounds. In the analog domain this was originally done by looping tape machines and delay times were dependent upon the speed of the motors driving the tape machine. The drawback to this system comes when delay times are changed the pitch content of the effect signal is changed as well. Digital delays made it easier to adjust the delay variables while retaining the frequency content, which is a crucial feature in a real world musical context. Most of these effects allow for a musician to tap a footswitch to set the variable value and often includes a buffer that stores the average time of the most recent series of taps.

Loop based audio effect modules use a punch-in/punch-out method for determining the tempo length variable. Typical use is to punch-in at the top of a measure and punch-out at the end. All the audio recorded by the unit will be repeated until the next punch-in point where the recorded will either be erased or summed with the current input signal. These types of units are convent for syncing samples of live instruments with an acoustic drummer. Often they also include reverb and delay effects that take their timing variable setting from the global measure length so the delay time are automatically synced to the master tempo.

Even though digital delay effects can handle timing variable changes better, by essentially holding a short sound for a time period and the playing it back at adjustable intervals, they are still susceptible to the same destruction of frequency

information problems that analog delay systems have when changing the length of a sound file. Traditionally these systems remove a percentage of samples at regular intervals to manipulate the rhythmic content of a sound, which increases the frequency of the sound, or copy and add a percentage of samples, which decreases the frequency of the sound. In 2001 the software development company Ableton released the first version of Live. This audio engine does not replay sound files. Instead, it looks at the original file that it needs to replicate, analyzes it, and feeds those parameters into a complex synthesizer and generates a new signal. In this way it can accurately reproduce the originals sound at any speed.

The Music In Motion Tempo Control System utilizes elements of each of these tempo modulation systems. A tap tempo averaging system is implemented that determines the runner's general pace. Tempo length is estimated by calculating the time between the most recent down beat and upbeat and predicting the position of the next down beat by assuming that the upbeat is the midpoint between them. This allows for greater sensitivity to sudden changes in the runners velocity. Finally, these two tempo measurements are routed to a synthesizer module that generates a Soundtrack at the required tempo in real time.

3.1.4 – Audio Synthesis

In terms of the musical landscape, audio signal synthesis is an incredibly new invention. The collective conscious of the Human species has endeavored to express itself by inducing ordered vibrations in the ambient air pressure for time immemorial and the Music In Motion system uses this inherent desire in a novel way to assist runners in their quest for a peak experience.

Archeologists have found simple flutes that date nearly seventy thousand years and the Aboriginal people of Australia have been utilizing the didgeridoo for communication and artistic expression for nearly forty thousand years. Modern concepts of structured harmonics and overtones can be traced back to approximately 3000 BC where images of musical instruments begin to appear in Mesopotamian artifacts (Clint Goss, n.d.). Even going so far as to create a class of professional musician. This appears to be the time in history when music moves away from ritual ceremony and towards the formal study of music.

In the early first century the scientific study of frequency ratios begins and is often attributed to Claudius Ptolemy and his study of the mathematics of music theory with his influential writing, *Harmonics.* This brought music into the realm of science. He used a monochord, a single stringed instrument used in the Greek musical laboratories, to empirically subdivide octave ranges in an effort to standardize musical instruments. Ptolemy's work was expanded by Pythagoras of Samos and through rigorous mathematical analysis developed the Pythagorean Tuning based upon a tonal relationship ratio of 3:2. It can create rich musical overtones when playing chords in the same key as the fundamental tuning.

However, modulation outside of that key will be dissonant. This system was utilized for approximately one thousand years.

J.S. Bach's 1722 composition *The Well-Tempered Clavier* popularized the Circular Temperament Tuning. This is the system that is still utilized by western music today. It looks at a larger spectrum and divides each octave evenly into twelve notes. These dozen notes are further subdivided into keys, which each contain seven notes, and a major and minor key structure so that key changes sound more harmonious and are spaced at even intervals across instruments that are played on a keyboard. Perhaps Bach had an intuitive understanding of over tone structure and the logarithmic nature of frequency relationships and their relationship to the human hearing range. Bach did this approximately one hundred years before Fourier gave us the strict mathematical understanding of complex wave constructions (Goeth, n.d.).

Music In Motion utilizes this deeply ingrained human desire for music by synchronizing a Soundtrack to the runner and his environment thereby focusing the runner and allowing them to transcend their previous conceptions of their own limitations.

3.1.4.1 – Analog Audio Synthesis

The first major musical audio synthesis innovation came in 1896 when Thaddeus Cahill, an American inventor from Iowa, developed the Telharmonium. It weighed two hundred tons and was driven by a dozen steam powered electromagnetic generators, had a velocity sensitive keyboard, was polyphonic and polytimbral. As there was no such thing as a public address system at this time the audio signal output of this machine was routed through the phone system for enjoyment in hotels, restaurants, and homes. An early foreshadowing of our modern media streaming protocols.

It was not until 1929 that the term synthesizer was first used when French engineers Edouard Couplex and Joseph Givelet invented the Automatically Operating Musical Instrument of the Electric Oscillating Type. Obviously, people called it the Couplex-Givelet Synthesizer. It utilized punched tape as a controller in lieu of the traditional keyboard and is the precursor to the modern step sequencer type synthesizer.

Synthesizers remained a musical oddity that were utilized primarily in the domain of science-fiction movie soundtracks until R.A. Moog, often considered the grandfather of audio synthesis, combined voltage-controlled oscillators with voltage-controlled amplifiers to create a highly customizable, modular, and easily transportable synthesizer. The popularity of this device led to the success of Wendy Carlos' record *Switched on Bach* which finally gave legitimacy to the synthesizer as an instrument and it became pervasive in popular music. It is because these audio synthesis methods have become so commonplace that the

Music In Motion Soundtrack can be used to focus the runner as opposed to being a distraction (Apple, n.d.).

3.1.4.2 – Digital Audio Synthesis

Digital audio synthesis had its genesis in 1957 when Max Mathew authored the program *Music I* for the IBM 704. It was able to calculate a single triangle wave but without the ability to do so in real time it was not particularly musically useful. The first great breakthrough came in 1962 when John Larry Kelly, Jr at Bell Labs implemented a vocoding method, multiband envelope filters that mimics the amplitude envelops generated by the human larynx and oral cavity, also on an IBM 704. It was able to play the American standard *Daisy Bell* complete with musical accompaniment.

In 1976 the New England Digital Corporation (NED) developed the Synclavier. All the on board processors had to be developed directly by NED for the Synclavier and the original system relied on frequency modulation to generate an audio signal. A few years later the first sixteen bit audio sampling system was implemented on a magnetic storage disk allowing for the blending of sampled acoustic sounds with FM synthesis. The Synclavier was extensively used by Frank Zappa, whose modern atonal compositional style was uniquely suited to the Synclavier's extensive and ever expandable timbral pallet.

Digital processing power continued to increase throughout the 1980's and allowed for the development of the digital signal processor (DSP). The first commercially available fully DSP based synthesizer was Peavey's DPM-3 and was based around the Motorola 56001 processor. It became the archetype for DSP based synthesis. Today, personal computers have more than enough resources to implement a huge amount of digital audio synthesis and it is not uncommon to observe a musical performance where the artist generates the entire musical content from a laptop computer.

Without these advancements in digital technologies the Music In Motion system would remain an unrealizable dream as its audio engine and control system design are a direct descendant of the work of these digital pioneers (Apple, n.d.).

3.1.4.3 – Audio Synthesis Methods

An incredible amount of audio synthesis methods have been developed over the years. Analog synthesis methods typically rely on voltage controlled oscillators and amplifiers to electronically simulate the transverse atmospheric pressure waves created by traditional acoustic instruments, albeit in a much more complex and customizable manner. Originally, digital synthesis methods were designed to model analog synthesis but as processors became faster and storage space

became increasingly more affordable a new generation of synthesis methods were developed that relied on mathematical abstractions and data format manipulation.

Additive synthesis is the procedure most commonly used for generating audio synthesis because of its straight forward and easy to implement procedure. Sinusoidal waveforms are easily generated in both the analog and digital domains and the work of Joseph Fourier demonstrated that simple summing of these waveforms theoretically allow for the creation of any complex waveform. Modern signal analysis allows for easy decomposition of these complex waveforms into their fundamental and harmonic sinusoids. Additive synthesis methods can use these Fourier transforms maps to model acoustic instruments as well as create completely novel unnatural sounds. The primary drawback to this method is that it may take a large amount of resources to recreate a complex waveform as each harmonic requires a dedicated oscillator. Synthesizers utilizing this for acoustic modeling may require hundreds or even thousands of oscillators.

Subtractive synthesis is based upon these same principles only in a reverse manner. A complex waveform is generated and routed through a series of band filters that remove harmonics to create musically useful waveforms. Typically this is a triangle or square wave although any periodic functions with overtones can be used. This procedure became more popular with the advent of digital oscillators that can easily recreate complex periodic functions. It requires a single oscillator for each note of polyphony and can utilize either analog or digital filtering systems. Subtractive synthesis requires fewer resources than additive synthesis and has an expressive tonal pallet but is not well suited to acoustic modeling.

Frequency modulation synthesis requires two oscillators for each note of polyphony. The output of the modulator oscillator drives the frequency select of the carrier oscillator. This method is often used for analog audio synthesis in conjunction with subtractive synthesis as the frequency modulation scheme can easily create complex analog waveforms with rich overtone structures that can be further refined by subtractive band filters. Most modern synthesizers are multitimbral. To achieve this the modulation index itself must be variable. While this method can create a wide range of harmonic structures with very few resources these overtones can easily become unwieldy and careful analysis and planning are required to implement a modulation index scheme that results in musically significant output.

Wavetable synthesis is the first purely digital form of synthesis. A single cycle of a periodic waveform is stored in memory and functions in the same way that an oscillator does in a traditional analog synthesizer. These wavetables are algorithmically computed, can be simple or complex, and are generally only restricted by their sampling rate and bit-depth. Early wavetable synthesis could easily create any harmonic structure but their susceptibility to aliasing and quantization error meant that they had an audio fidelity well below that of an analog synthesizer, even if they were more versatile. At modern bit-depths and sample

rates the differentiation between analog synthesis and digital wavetable synthesis is generally regarded to be below the tolerance of human perception and require very little processing power and memory allocation.

Sample-based synthesis relies upon the same fundamental principles as wavetable. Instead of using a single period of a waveform a sound file is used to generate the audio output. This system is primarily used to replicate acoustic instruments with a digital synthesizer. The most traditional implementation is the digital piano and organs. In this case each note from the original instrument is recorded and passed through an analog to digital converter and loaded as a sound file into the synthesizer. Through the use of amplitude envelops to recreate dynamic playback and looping specific segments of the sound file to create variable sustain an accurate and expressive recreation of the original instrument can be replicated. Many creative sound designers and composers quickly realized that nontraditional sounds, such as bird calls or industrial noise, could be loaded into the synthesizer and manipulated to create a timbral pallet that would have been impossible to create through traditional oscillator based synthesis. The primary drawback to sample-based based synthesis is its digital storage requirements. Some of the consumer sample-based instruments can be as large as a few gigabytes while professional composers sample libraries run well into the terabyte range.

Granular synthesis further expands upon the principles of sample-based synthesis. The sound file can be accessed at the sample level are divided into sections, or grains. These grains can be shuffled, reversed, phase shifted, layered, time stretched or compressed, convolved, and manipulated in almost any other way the designer can imagine. The resulting output is nearly impossible to predict and often bears no resemblance to the original sound file. While the harmonic pallet that can be generated through this procedure is expansive it requires a great deal of processing power to achieve a musically significant audio signal output in real time.

The future of audio reproduction is Wave Field Synthesis (WFS). This procedure utilizes large microphone and speaker arrays recreate three dimensional sonic environments in which a sound source can be placed at any spatial location within the field. In the late 1990's microphone arrays began to be used to create a multichannel recording of an audio impulse in a particular acoustic environment. The acoustic signature transfer function is derived from this information and used to digitally control and modify the acoustic signature of a space. Modern control systems allow for arrays of hundreds of microphone and speaker systems to be precisely timed. Tuning these systems is still somewhat of an art form at the mathematical models that predict spatial audio source location can only predict accurately in a dead room. Reflections from an acoustically active environment will distort the spatial image. Even though it is possible to create anechoic spaces they are not currently possible for a larger scale environment that is comfortable for people to experience, and as they say if a tree falls in the forest.

As the Music In Motion Audio Engine is essentially a digital synthesizer all of these options could potentially be used in its design. Sample-based synthesis generates a wide variety of sounds and is easily controlled in real time but its digital storage requirements are not effective for this design. The audio engine design for the Music In Motion is built around the VLSI VS1053B MIDI controlled digital synthesizer. It utilizes a bank of digital oscillators to model acoustic instruments and includes the General MIDI Soundbank. This soundbank includes a wide variety of instruments, such as pianos, bass and treble guitars, horns, strings, and rhythmic instruments. The MIDI control allows for the Music In Motion system a total of sixteen control channels and the MIM design uses two channels per instrument. This channel layering allows for the creation of unique timbers not inherent in the VS1053B standard soundbank. Utilizing the VS1053B allows this design to be inexpensive, not require excessive processing power or memory, and is responsive enough to make the adjustments required by the control system in real time.

3.2 – Impulse 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). This is essential because the music is almost instantaneously changed by this signal, therefore it is much more sensitive than simply incrementing a step counter, such as in a pedometer. A high quality signal can be attained by a delicate balance between the processing resources of the microprocessor and/or FPGA and the resolution of the sensor used. Additionally, assuming a fixed allocation of processing resources for the impulse detection: the bit depth of a signal will be traded off for the sampling rate and vice versa. The bit depth of the signal will be directly proportional to the signal quality and thus inversely proportional to the error rate. The sampling rate will be directly proportional to the delay between this impulse control signal and the response (updated tempo of the music). Therefore, a balance needs to be struck between these elements. There cannot be an abundance of both because that will limit the resources of the microprocessor. This translates to three elements in direct design competition: bit depth, sampling rate, and microprocessor resources. This section blow explores the various options in sensors for detecting impulse signals.

3.2.1 – Microphone

It is possible to use a microphone in order to capture the user's foot landings. Before the advent of the accelerometer, the use of microphones for such applications was common practice. The advantage of this method is that it is easy to understand and implement. Additionally, it gives some flexibility in the design of

the product. Perhaps, in later development the sound recording from the microphone can be used to add a new dimension to the audio algorithm. An example of such would be the heaviness of breathing. The disadvantage of this method is that is it very expensive in terms of system resources because audio files require a lot of data. Also, this method assumes an easily audible sound. What if the user is running barefoot on grass? Then this method is obsolete. For this reason, this section will move on.

3.2.2 – Accelerometer

An accelerometer seems to be a great tool for detecting user impulses. This is because they fundamentally are changes in acceleration. The importance of the accelerometer lies in certain metrics that need to be analyzed: output format, axis, resolution, maximum swing, sensitivity level, bandwidth, output data rate, and dimensions.

The biggest concern so far has been the output format of the accelerometer because this will determine the communication protocol to interface with the microcontroller. This is crucial in the sensor network design because there are a limited number of I/O pins on the microcontroller and the communication protocol could vastly complicate or simplify the design of the network.

Analog seems more complicated as a protocol. This is because, according to research, many accelerometers need to interface with a device with an output impedance of less than 10 Ω. (Dimension Engineering LLC., n.d.) Analog also appears to be more complex because one needs to consider the minimum and maximum voltages for the I/O pins of the microcontroller. The advantage to an analog output is the fact that the signal filtering can be implemented with an nth order Butterworth low-pass filter. However, as stated above, the I/O pins on the microcontroller are limited. This translates to a propensity toward analog due to the fact that digital requires extra lines. This is because digital requires a clocking signal in order to interface the accelerometer with the microcontroller. On the contrary, digital is much easier to work with in regards to the software because one does not need to access the on board A/D converter in the C2000. This seems like a viable option for the MIM device's impulse detection subsystem.

3.2.3 – Mechanical System: Ball and Switch

The most basic pedometers use a very simple way in which to detect the footsteps of a user. This is the mechanical system of a ball and switch. This sensor has a chamber containing a ball that slides up and down naturally through movement. When the ball slides down it hits a switch which in turn increments a counter. This same system could be applied to the Music in Motion device with ease. It is incredibly simple to implement and decently accurate as well. This method requires almost no system resources because the signal is itself an electrical

impulse; therefore, it also requires no filtering. The only issue is the sensitivity is not nearly as responsive as the accelerometer.

3.3 – Signal Processing

The signal processing methods that are researched in regards to this project are for the applications of the impulse signal quality and audio signal quality. For the impulse signals, the focus is in noise reduction and high frequency minimization. This will drastically increase the convenience and ease of foot fall detection and therefore decrease the error rate. For the audio signal, the main focus is in researching the common practices for audio signal processing. In order to design an effective system, audio amplifiers, compressors, and filters will be investigated.

3.3.1 – Impulse Filtering

The primary concern when filtering a periodic signal, such as the impulse train data that will be collected from the accelerometer, is the bandwidth. The bandwidth of this signal is between 1 Hz and 10 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 ten 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. Additionally, through prototyping the main axis will be determined. It is possible that only the z-axis will be used for the impulse train data.

3.3.2 – Digital Filters

Due to the nature of the signal, SPI format, a digital filter will be used to realize the high frequency and noise attenuation. Digital filters, borrow all of the main ideas from analog filters but they perform much better. A digital filter is ideal for this project, especially in the prototyping phase. This is due to the monetary and time cost that comes with changing or tweaking an analog filter for a system design. With modern computing, any changes can be made by simple coding and the accuracy yielded from digital filtering is incredible.

3.3.2.1 – Finite Impulse Response (FIR) Filters

FIR filters are generally the simplest digital filters to design and do not have feedback, which is why they are finite. These filters are typically linear phase, which is incredibly important for the impulse signal that will be filtered. Introducing

non-linear phase in this signal is the same as distorting the signal for this 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. The disadvantage to FIR is that they require much more memory than alternative digital filters, which may present a problem. Figure 2 is a picture of a generalized signal flow of a FIR filter.

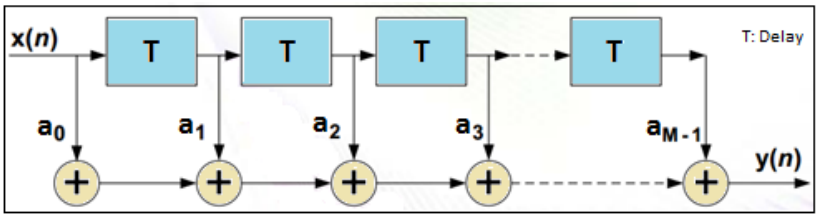


Figure 2 – General FIR Design

Essentially this diagram shows the weighted sum of filtered amplitudes and their associated delayed elements. Therefore, the filter multiplies each sample by a given coefficient that is related to the type of filter implemented. For example, a low pass filter would have small coefficients for high frequency components that are sampled. (Venkat)

3.3.2.2 – Conventional Infinite Impulse Response Filters

IIR filters have feedback; therefore they give a better result than the FIR filters given the same order. Conventional IIR filters were designed directly from the analog filters that came before them. There are several problems associated with conventional IIRs. The main disadvantage, specifically for the C2000, is the register width requirements. Figure 3 is a diagram of the general IIR design which shows the recursive element.

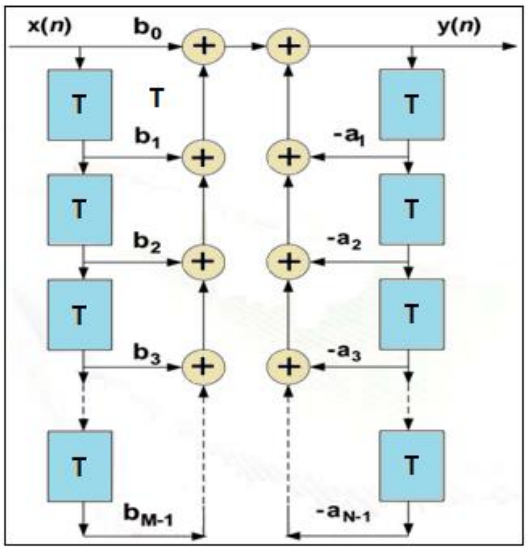


Figure 3 – General IIR Design

This recursion is essentially infinite feedback of the signal and its delayed counterparts with weighted coefficients. Conventional IIRs are not practical due to scalability for low end microcontrollers. (Venkat)

3.3.2.3 – Lattice Wave Digital Filters (LWDF)

This type of IIR filter that was made specifically made for implementing on low power microcontrollers, such as the C2000. This is of crucial importance because wave digital filters allows for real time signal processing on the C2000 and this project requires real time signal processing. This could be bypassed by changing the accelerometer to an analog output one, using an analog filter, and then an ADC. However, that route is much more time consuming and risky to implement. It is more efficient to have a software solution for the filter that can be easily changed and tweaked.

Lattice Wave Digital Filtering is done by processing a signal through a network of adapters, such as the one in Figure 4.

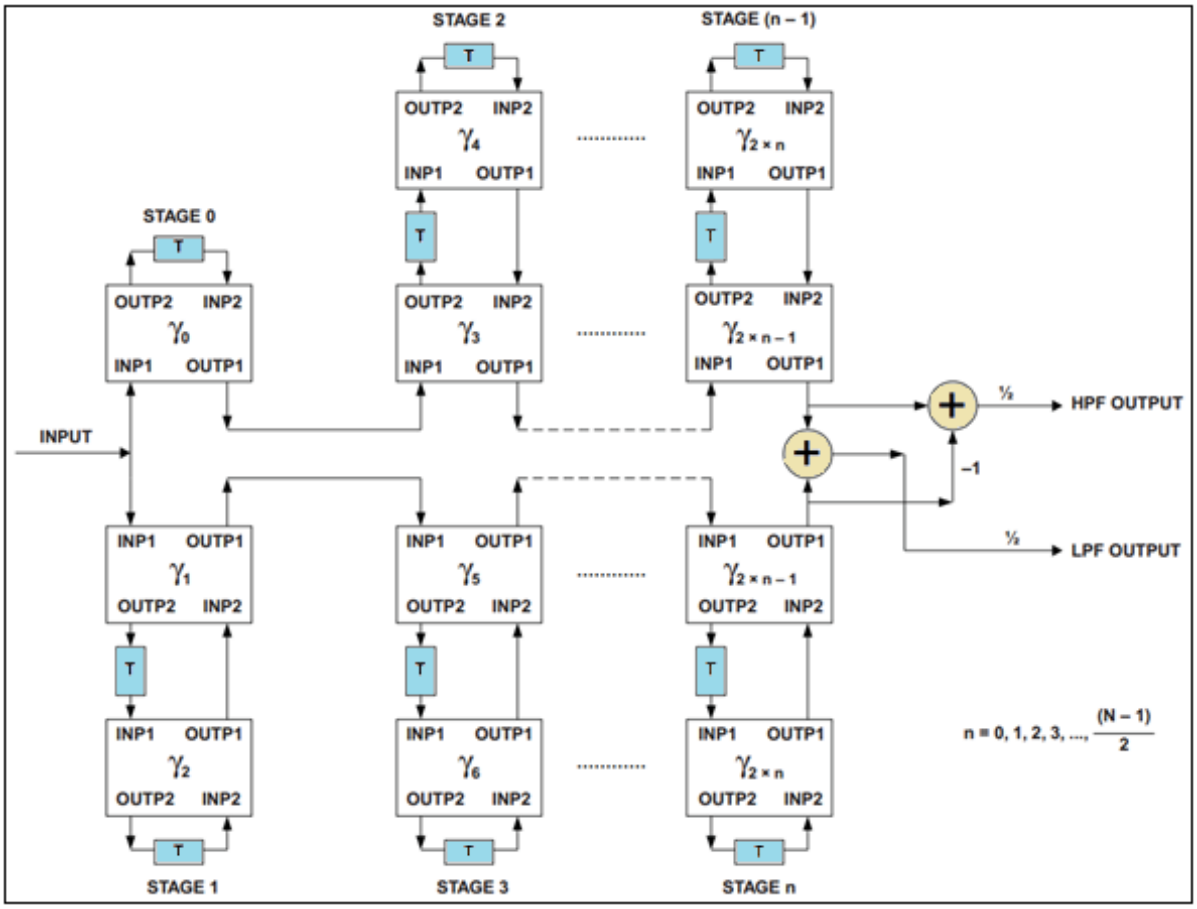


Figure 4 – TI Digital Filters Datasheet

This is a generalized LWDF configuration, which gives you both an output for a low-pass and high-pass filtered version of the input signal. The larger the order of the filter, the more adapters required. Additionally, the specifications of the filter are accounted for by calculating the gamma value that is inside each adaptor in Figure 4. There are various types of adaptors one can use for the design. All the parameters are determined once the design specification is completed and in the context of the processor that will handle the data. (Venkat)

3.4 – GPS

GPS technology is highly regarded as convenient, stable, as well as a rather inexpensive hardware to implement with many types of systems and designs. A GPS module will be implemented and structured onto the PCB as a chip but will serve a relaxed, secondary purpose and retrieve live coordinates to keep track of the runner's progress as well as serve the runner in real-time to get a good feel of how well they are doing on the run. Most GPS systems on the market range in price depending on the type of GPS and needs of the buyer. GPS hardware include from standalone chips, positioning modules, or positioning antennas depending on the implementation of the respective hardware.

For Music In Motion, finding a GPS module that works in conjunction with the microcontroller is key. There is nothing we can get from the GPS unit if the microcontroller cannot receive the data from the chip, through the processor, and into memory. This is, first and foremost, the important approach for finding the correct GPS module for the project. GPS reliably is also in priority as the user needs to have accurate feedback on all accounts when the GPS takes in coordinates for later use. GPS technology became mainstream around 2005 and has been used in mobile devices, military technology, and professional applications since then. That means many of the devices on the market, for a reasonable price, are fairly accurate and dependable for low deviation coordinates of someone's current position on Earth. Typically with most GPS devices, you'll find you get your location +/- 10 meters. Now variables can cause a major difference on how accurate your GPS device truly is. Time of day and, more importantly, clarity of reception (cloud coverage, sunny/rainy) help paint the true picture of accuracy. One of the benefits of the MIM device is in its practicality. Since the device is used outside on a run and since most runners run during a sunny, mostly clear day, the device reaps the benefits of clear reception for the GPS in order to give the most accurate feedback. This is especially important for more seasoned runners who keep track of every minute (Mando, GPS Accuracy, n.d.).

Below is a compilation of other factors taken into consideration when exploring options into the right fit for the project.

Size – GPS modules on the market year to year decrease in size and allow room for other parts on a circuit board while also requiring less data and control lines.

GPS requirements in regard to size is more about how well the size of the antenna matches the module rather than the size of the entire package itself. Granted, many companies sell products with both included which removes the need to worry about if they match. The type of antenna that is attached to the GPS module also greatly affected the size. The two common types of antennas used are the helix and patch antennas which will be discussed in greater detail later on. The helix extends from the side of the module which typically causes an awkward protrusion and makes for difficult placement on the PCB. The patch antenna lays on top of the module saves more space horizontally but extends higher vertically from the PCB.

Refresh Rate – The typical GPS module on the market comes with a refresh rate around one hertz. Some more advanced models have a refresh rate ranging from 5 Hz to even a max of 10. The amount of data processed at that frequency can overwhelm the processor if it is managing other tasks at the same time. Our project does not need to accumulate GPS coordinates every second, so it would be more of a benefit if the device had a slower read-out frequency so the microcontroller can handle other actions during that time. Last resort would be to modulate the frequency of the GPS or more realistically, ignore coordinate readouts for how long is appropriate for the project.

Power Requirements – Considering most GPS modules compute coordinates every second, the amount of power needed is quite high averaging about 3.3V and 30 mA on the market today. An aside from the module needing a base level of power, the antenna requires an amplifier in most designs in order to draw extra power as needed. Some modules will have more power efficiency than the average, but typically those modules do not include the antenna and must be added separately.

Channel Tracking – It's common knowledge that GPS technology requires the communication and channeling between satellites orbiting and the device. In order for max efficiency of retrieving coordinates, multiple channels are allocated to the receiver. The module does not know which satellites are in view at a given time, different frequencies will openly search until there is a lock with one of the channels. At that time, the other channels will shut down to save energy. 12 -14 channels is usually a comfortable minimum on GPS modules but will require a longer wait time to find a lock. For a faster reading and lower power consumption, channels range from 20 all the way up to 100 if necessary but with higher number of channels comes with a price tag and over-saturating the device. The reason power can be saved with more channels is that the GPS doesn't need to be on all the time if, when it is turned on, for a brief second or two has more channels tracking satellites and finding locks faster.

Antennas – made of a ceramic material trimmed and fitted to properly pick up signals sent from orbital satellites. This frequency is set at roughly 1.575 GHz. There isn't much in variation when it comes to antennas on the market. Some less

common designs include chip-type or helical but they are more expensive and require more amplification and filtering. The more important issue is ensuring there is little to no blockage of signal between the antenna to the satellite and back (SparkFun, n.d.).

3.4.1 – Antenna Type and Position Detection Protocol

This section goes over the select variations among antenna types attached either internally or externally with the GPS module and which antenna would be appropriate for Music In Motion. There is a possibility that the module decided on will include an internal antenna which will not weigh as much in decision-making for the type of antenna than if the antenna was decided to be externally.

Antennas on consumer GPS receivers only work with the L1 code frequency of 1.57542 GHz GPS carrier frequency. This frequency contains the C/A code, encrypted P-code, and the Navigation message. Stated before is the fact a minimum of 12 channels is required for the GPS antenna to track all visible satellites. GPS data generally occupies a rather narrow bandwidth while the signal itself is encoding using spread spectrum because the data will be less susceptible to noise as well as a secure communication. The resulting transmit signal has a bandwidth of approximately 20MHz (fractional bandwidth of 1.26% in reference to the L1 code frequency). This information is only relevant to the fact this type of bandwidth reception is difficult with a limited antenna size. That is where different types of antennas come into action when deciding the most applicable to the project Music In Motion.

One very important distinction among antennas used for GPS vs. other common application antennas is the concept of right-hand circular polarized signals (RHCP). To keep the concept simple, normal signals can be transmitted using linear polarization and in regard to patch and helix antennas, the names themselves are derived from the application of receiving the circular signal from the source. Below in Figure 5 is a representation of how the signal is transmitted using RHCP.

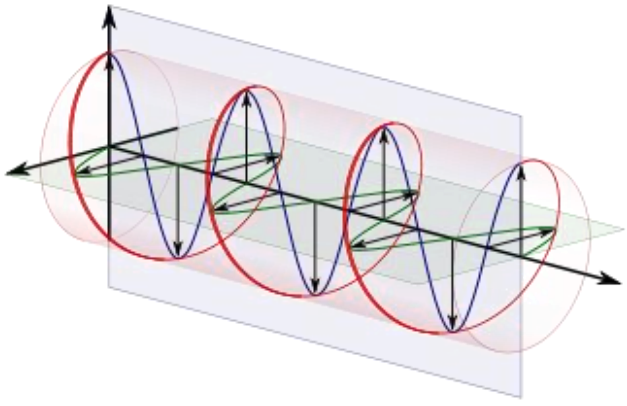


Figure 5 – Right-Hand Circular Polarized Signal

3.4.1.1 – Patch Antennas

The more common choice of the two main types of antenna is a patch antenna. Described by the name, patch antennas are flat, and they generally save space other type of antennas have difficulty with. Patch consists of a ceramic, metal body and are mounted on a metal base plate and often cast in a housing for protection. More detail is provided in the Helix antenna section below because patch antennas are not ideal for the project Music In Motion. Patch antennas are preferably placed and mounted on a flat surface such a roof or the dashboard of a car. The gain is much higher than a helix antenna while providing a lower cost and wide array of sizes on the market. Patch antennas of 25mm by 25mm show optimal performance and are cost-efficient. Patches smaller than 17mm by 17mm tend to demonstrate moderate navigation performance.

The patch antenna is still possible for the design for our group because the benefits of this type might outweigh the lack of performance from the helix antenna but the decision will not be made until the design stage of the project.

3.4.1.2 – Helix Antennas

More formally labelled as a quadrifilar helix antenna, the size is dependent on the dielectric material the fills the space between the active parts of the antenna. If the size is decreased due to a higher dielectric ceramic yield, the performance is compromised inversely. Simply put, antenna gain will decrease with decreasing size of the antenna. Helix antennas more aptly suit the Music In Motion designed as they are tailored to applications with changes in antenna orientation (resulting from the movement of the body during exercise). Helix designs are better for navigational situations and are typically more robust than patch antennas on the market.

3.4.1.3 – Comparison between Helix and Patch Antennas

As mentioned before, the application for each is one of the top if not the most important factor in deciding between the two. Patch antennas are better suit for stationary situations and locations while helix accounts for orientation-changes as well as satellite-detection-deficient environments. Since the application is a hand-held device, for lack of a better classification, the antenna must be designed in a way that accounts for optimum antenna orientation which the helix is better suited for. The biggest drawback for helix is the size-to-performance ratio that the patch design achieves at a higher result for most product comparisons. Below is Table 10, which details the positives and negatives of the helix and patch designs.

Helix antennas	Patch antennas
+ omnidirectional + robust – cost – space requirements	+ high gain + low cost + large variety of sizes available on market – Less isolation between feed and antenna when compared to helix antenna

Table 10 – Positives and Negatives of Patch and Helix Antennas

3.4.1.4 – Active versus Passive Designs

Not nearly as important as antenna type but equally notable is the decision between an active or passive antenna. Active antennas include a built-in Low Noise Amplifier (LNA) which counter acts the losses attributed to the cable when the signal is transmitted to the receiver. Since passive antennas do not include an LNA, they are required to only be at most 3 feet away from the receiver as well as forcing the receiver to have an LNA on-board. Since the device does not require such a distance between the GPS receiver and the antenna, a passive antenna is more applicable to Music In Motion not only in size but minimum requirements it still maintains (taoglas - antenna solutions, n.d.).

3.4.1.5 – Other Antenna Types

Types such as chip, fractal, PCB, PIFA, or dipole antennas are not nearly as common as the mainstream Helix or Patch antennas due to the dramatic performance decline despite the much smaller size capability. The drop-off in performance is too much to overcome with most GPS-inclined systems as navigation using these types of antennas becomes obsolete.

3.4.2 – Data Transmission Protocol

The GPS module I/O, in comparison to other devices integrated in our design, is simple and universal in execution that there isn't much in finding the correct I/O association with the microcontroller used in the system. Based on the program stored in the microcontroller, which controls the lines between the microcontroller and the GPS, it does not necessarily play master role to the GPS because the GPS can determine when the coordinates are ready to be sent. This also means the microcontroller can determine if it is ready to receive said coordinates by allocating a control bit only when it is ready to receive the data. Below is an example of a string read to a terminal from a GPS in order to give an idea of what the strings will look like once produced by the module and what they mean.

```
$GPGGA,235317.000,4003.9039,N,10512.5793,W,1,08,1.6,1577.9,M,-20.7,M,,0000*5F
```

- Time: 235317.000 is 23:53 and 17.000 seconds in Greenwich mean time
  Longitude: 4003.9040,N is in the format degrees.decimal minutes, north
- Latitude: 10512.5792,W is in the format degrees.decimal minutes, west
- Number of satellites seen: 08
- Altitude: 1577 meters

Some of the other values are either extraneous or unnecessary (Mando, GPS Message Formats, n.d.).

There are a range of communication protocols supported by GPS receivers today. The most common being UART asynchronous communication while more new-age protocols include SPI and more recently I2C. The following sections will address the different communication protocols in reference to GPS systems and how they are most efficiently applied.

3.4.2.1 – UART (Universal Asynch. Receiver/Transmitter)

One of the most flexible computer hardware used for translating information through serial communications. UARTS are common in microcontrollers and other peripherals and most accessibly transfer data back and forth from the integrated circuit and a computer/external device. Some other variations of UART include DUART (dual UART), OCTART (eight UARTs) and USART which allows synchronous communication. Since the standard of UART is asynchronous, the two devices (GPS and microcontroller) do not need to communicate on based on an external clock but rather on start-stop bits that are delivered within the data sent to and from each device.

Just as a reference for port and pin communication, the standard Rs-232 is used for the model in Figure 7. The Tx and Rx from the native device pass through the RS-232 UART Level Converter and is dictated by both the native device as well as the device connected from the serial port. For Figure 6, the simple diagram shows how the parallel information passed to the UART (Control lines, addresses, and data) are handled and passed as linear data to and from the UART.

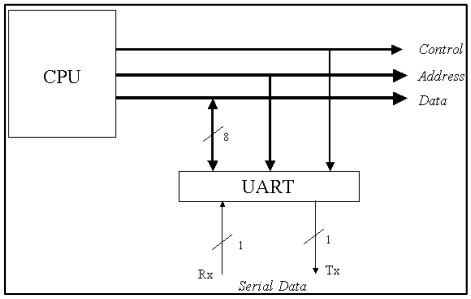


Figure 6 – Simple Diagram between Parallel and Serial

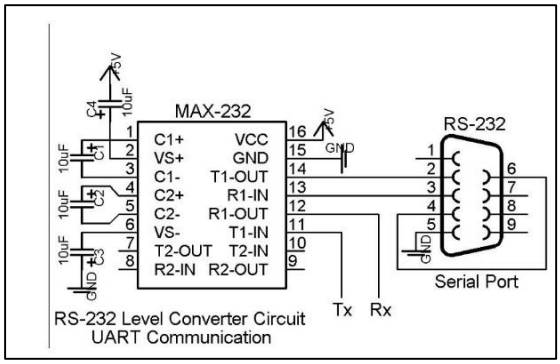


Figure 7 – Pin Connections w/ Rs-232

3.4.2.2 – SPI (Serial Peripheral Interface)

A synchronous serial protocol, unlike UART, that uses a single master and one or more slave combinations to transfer data. Synchronous means a clock is used in the process of data transfer and that clock is determined by the master device and the slave must align its clock cycles based on the master. Shift registers are implemented in each device so the data sent out sends the most significant bit and receives the least significant bit first. The model below in Figure 8 shows the four lines between a single master/slave relationship.

SCLK (Serial Clock)	MOSI (Master Output)	MISO (Master Input)	SS (Slave Select)
Clock signals determined by Master	Data output by master device to slave	Data input to master device from slave	Selects the slave currently in use (Active low)

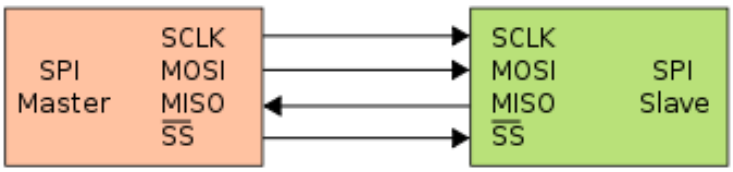


Figure 8 – Basic Diagram between SPI Master and SPI Slave

3.4.2.3 – I^{2}C (Inter-Integrated Circuit)

The most recent and well-equipped serial protocol uses two bidirectional lines to dictate clock and data signals from the master device to all other slave devices. This proves I2C is a synchronous, master/slave serial protocol similar to SPI, but generally has better performance and is easy to program drivers for. I2C does not use separate slave selects like SPI but rather addresses the device it wants to communicate with and creates a connection for data transfer. The addresses specifically sends a 7-bit address to find the slave as well as a 1-bit read/write for determining the direction of the data (Total Phase, Inc., n.d.). Pull resistors, like the ones at the top of Figure 9, alleviate any conflicts if one device sends high while another on the same line sends low. I2C is the most efficient, well-rounded protocol of the three types but is probably more appropriate for communication

between two complex data I/O devices rather than between a microcontroller and a GPS.

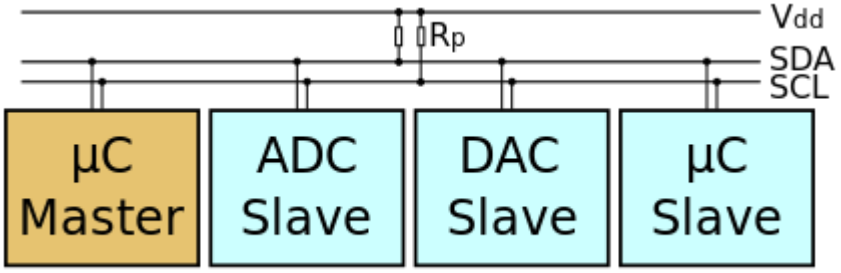


Figure 9 – Basic Diagram between I²C Master and all Slaves

The choice between using the serial protocols UART, SPI, or I2C is strongly influenced by more important communication the microcontroller needs for other devices in the system such as the FPGA, accelerometer, and memory storage. Since the information isn't as complex as the pulse signals from the accelerometer and constant communication between the microcontroller and FPGA, the seemingly-appropriate protocol decision would be on UART. The microcontroller needs to make sure there is enough I/O on the chip and also prioritize communication, but that is not the job of the GPS to worry about.

3.4.3 – Clock and Power Requirements

Clock requirements only come into play if the I2C or SPI serial protocols are used. For both protocols, the master device determines the clock and when data is transferred to and from the devices. In I2C protocol, the pull-up resistors not only make sure no conflicts occur along the data lines but also communicate with the master clock to work in congruence on either the positive or negative edge depending on the technology of the master device. In the case of our project and in most situations, the microcontroller will be the clock and also determine when the coordinates from the GPS will be sent. The frequency of this transfer is limited by the GPS device but it can always be decreased if the amount of coordinates needed is less than the max achieved based on the frequency of the GPS. For data transmission on the SPI connection, the master device sets the clock less than or equal to the maximum frequency of the slave device similar to the I2C protocol. The difference is during one clock cycle using SPI, the master sends a bit through the MOSI and the slave reads it while the slave sends a bit through the MISO and the master reads it.

Most modules on the market have a standard voltage average of 3.3V with a small deviation because of how similar modules must be to handle the information passed in through the antenna. Current, on the other hand, varies tremendously depending on the antenna type, how many channels are on the antenna and the amount of power the amplifier requires for that specific antenna (which is usually influenced by type/channels). The amps for modules range from 20mA to 75mA

but that mostly is influenced by the antenna. The serious consideration for power is just adding it to the power minimum requirements for the battery implemented on the PCB.

3.5 – Microcontroller

Of all the components in the design for Music In Motion, the microcontroller bridges the gap for data transfer and clocking decisions for the FPGA, accelerometer, GPS, and memory module. The microcontroller is required to take in the pulses from the accelerometer and create control signals for the FPGA to handle. It needs to read in the GPS coordinates and store them in memory to use for later coordinate calculations for milestone feedback. The connection between the microcontroller and the FPGA is going to be direct so act as one device when the data is sent from the microcontroller to the FPGA. The memory module has a minor role of storing averages not only for the coordinates in regards to milestone calculations but also runner value tracking as well as tempo statistics that can be manipulated for more complex algorithms that brings more intricacy to the run.

The microcontroller for Music In Motion can be thought of a temporary data storage and manipulator, waiting to go to the final destination a la other components on the device. Considering the many functions of the microcontroller, it needs to have a high enough frequency and I/O ports so the components can all connect and also handle the different data being passed in constantly.

3.5.1 – Data Processing

With the exception of clocking and data priority, most of the mechanics are handled by the accelerometer, GPS, and FPGA. The only temporary memory storage has to do with the coordinates taken in from the GPS and stored on the processor for computation and memory reorganization once the calculations have been made. Other than that, data comes in, the microcontroller's software algorithms handle the data, and then the data is passed on through.

One of the biggest factors in data processing for the microcontroller of choice will to be avoiding bottlenecks throughout the CPU. Many factors of bottlenecking to take into account are data/port priority of the components, making sure the register file and cache are large enough to handle immediate data and send the data before new computations need to be made, and, obviously, the clock speed of the processor doesn't limited important and decisive computations required such as the tempo control and FPGA control signals. Interrupt vectors from the microcontroller will be assigned via software to confirm priority among the ICs and is handled through software. The convenience of microcontrollers in this day and age is that most processes required by many components can be managed through software without the worry of hardware limiting options on how to go about deciding on what the microcontroller should in real-time.

Most of the data that is used in congruence with the microcontroller's clock speed is the pulse readings from the accelerometer that become the tempo in the FPGA for the music produced. The secondary data, which only occurs once a second (1 Hz) is the coordinates issued by the GPS once they are gathered from the satellites, are channeled through the antenna and passed out of the module into the microcontroller. In Figure 10 below there is a demonstration of the flow of data for all components through the microcontroller.

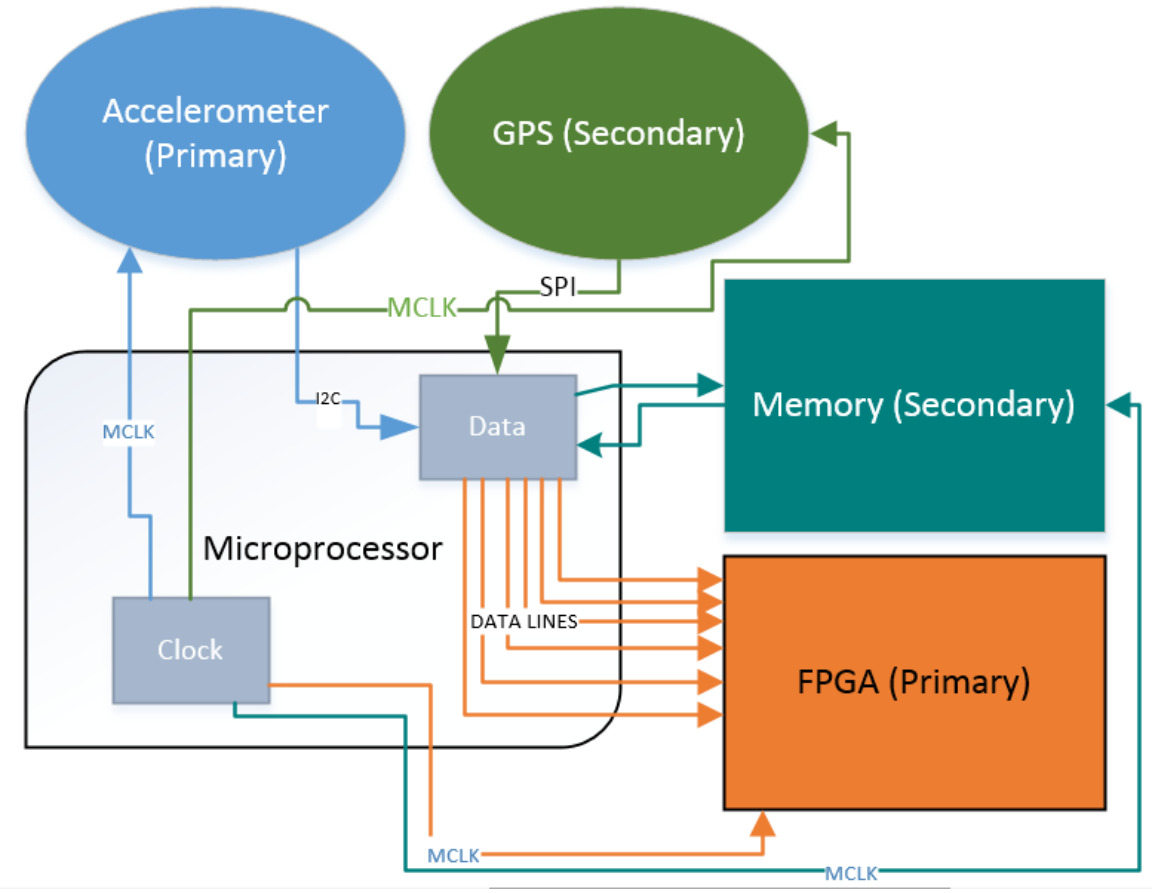


Figure 10 – Microcontroller Data Lines to/from IC

The clock in the microcontroller represents the master clock of all peripherals connected. The accelerometer data is received, processed and sent to the FPGA as the primary function of the microcontroller. There are two reasons for this; one, the tempo needed for the runner is immediate and in real-time, and two, The GPS coordinates do not need to be gathered on every clock cycle and are used to calculate rough distances for the milestones that occur only a few times during the run. Below are the four relationships of the microcontroller and other ICs in detail.

3.5.1.1 – Accelerometer

The accelerometer plays slave to the master clock from microcontroller in a traditional sense that the data from accelerometer will only be sent over, pass through the ADC (analog-digital converter) and store temporary in the microcontroller. The ADC can produce up to how ever high the frequency of the microcontroller but most likely isn't necessary to produce that many samples considering all the other actions and processes the microcontroller must handle in other areas of the device. In essence, the accelerometer is not limited by the hardware and freely passes the signals to the microcontroller without any interference or interruption. Overall, the microcontroller software determines the sample rate of the accelerometer and the data is transferred at that frequency. Whenever a digital signal from the accelerometer is ready to pass to the microcontroller, it takes top priority, triggers the interrupt, and retrieves the data.

Figure 11 below demonstrates the sampled data on an example reading from accelerometer. The threshold value in the middle of the sampled data represents a determination of whether the runner/walker has taken a step or not. The number of samples taken is an important variable the microcontroller has to account for when retrieving samples at particular frequencies. All of this will be handled in the software portion.

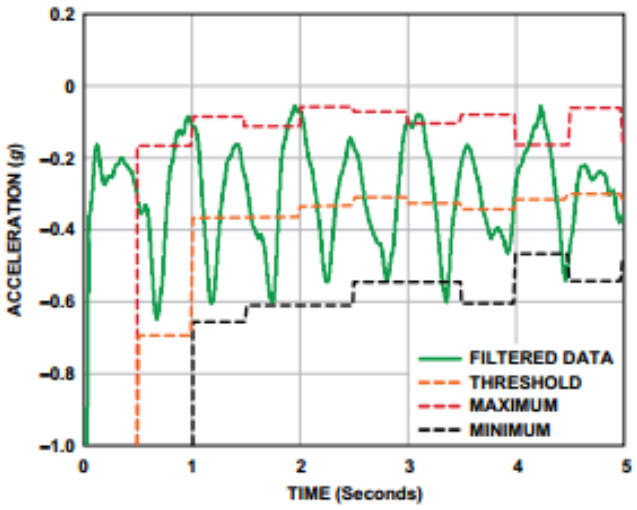


Figure 11 – Example of Filtered Data from Accelerometer

3.5.1.2 – GPS

The GPS takes the same slave role to the master clock as the accelerometer. The data represented by the GPS is a string passed through without any requirement of converting or altering. Once the microprocessor receives the GPS coordinates, although sparingly, it processes through the many algorithms of calculating distances, averages, and time, and finally stored those values in the memory module for later use to notify the runner of achievements and milestones. The

strings created/received by the module through the antenna has distinct float values such as time, longitude, latitude, cardinal direction, among other factors that may or may not be used. The microprocessor compares those floating point values with ones still on chip or stored in memory to calculate new distances and averages. Those final values are stored in memory and the cycle repeats.

3.5.1.3 – Memory

Once, again memory plays the slave role to the microcontroller which clocks the data. The difference between the memory module and the GPS/accelerometer the communication between memory and microcontroller is full duplex. Not only does the microcontroller send data to the memory module for storage after it has filled its use, but also retrieves data from the memory whenever it needs to use it again. The memory module is only connected to the microcontroller considering the only data stored in ROM needs to be coordinates from the GPS, control signals based on user input from the android application, and tempo-controlled data based on the accelerometers signal output. The priority of memory access is also low compared to the accelerometer as most of the time, data is handled on the microcontroller's memory cache and only transferred to memory when the data's purpose is null in real-time. Memory access and hierarchy works the same as any architecture with an address line, R/W, data lines, and master clock.

3.5.1.4 – FPGA

The communication between the microcontroller and the FPGA is straight-forward and simple compared to the relationship between the microcontroller and all other devices. The FPGA is always requesting the control signals provided by the microcontroller and will receive them as soon as the microcontroller is ready to send. The data lines are one-to-one, half-duplex and implement pulse-width modulation to encode information for transmission. With this in mind, a low duty cycle corresponds to low power for only when the control signals are needed from the microcontroller. This means the data is sent one-way and only when the data is available from the microcontroller. The data sent by each individual line will have distinct control over the music synthesizer managed by the FPGA. This allows the data to get to the correct location in the audio engine and process the data without lag in the music conducted on the fly. If the data was sent all on one data line, the control signals would have to be decoded by the FPGA and sent to the proper locations which could cause an unwanted stoppage in the experience of the music.

In Figure 12, the data control signals are sent from the microcontroller to the FPGA which control all the different variables needed to create the music. The microcontroller chosen for the project would need to have enough I/O lines which the C2000 prototyped in the diagram might not work for the final design.

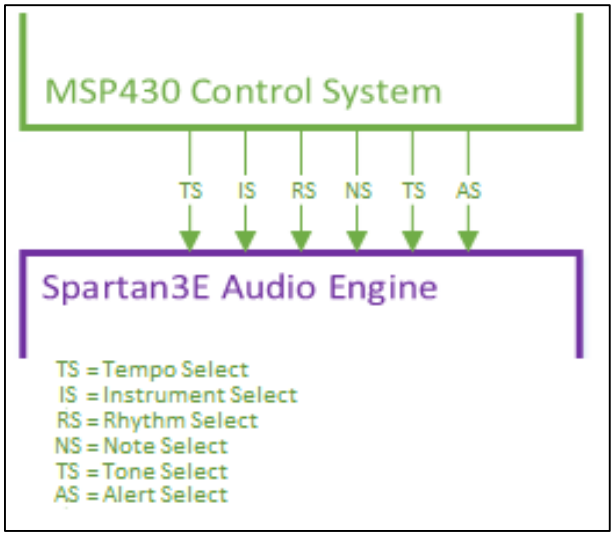


Figure 12 – Data Lines from MCU to FPGA

On top of enough I/O lines for the FPGA, overall, there needs to be high enough max to accommodate for all four ICs attached to the microcontroller. The amount is arbitrary now because of the unknown FPGA, accelerometer and GPS used for the final design, but a rough estimate would anywhere from 20-40 I/O lines.

3.5.2 – Clocking Requirements

Most of the micro-analysis of the clocking schemes associated with each I/O port are going to greatly vary from processor to processor so the details in specific are discussed in the sections for the C2000 and C2000 series microcontrollers. As for the I/O connections with the integrated circuits, the clock speed of the microcontroller is a more of a priority than other factors when deciding on a microcontroller. In Figure 10, it demonstrates how the primary ICs (accelerometer and FPGA) take precedence to the secondary ICs (memory module and GPS). This is true for data transfer as well as clocking priority (seeing as they go hand-in-hand). All connections will have their own clocking scheme but a table of interrupts exists for setting and attributing priority and external factors that will cause one IC to transfer data over another. If the GPS wants to send coordinates to the microprocessor on the same clock cycle as the expected accelerometer pulse data, the pulse data will come first while the GPS holds the coordinates until the microcontroller is ready to receive by sending the interrupt command to the GPS (master to slave). There shouldn't be any bottleneck or deadlock issues for the microcontroller if all the connections and software interrupt priorities are set correctly because the microcontroller controls all signal, data, and clock information that is transferred throughout the entire design.

When it comes to the relation between the clock speed and what frequencies would best suit the design at hand, another important consideration is the power consumption that comes with higher vs. lower clock speeds. Low clock speeds on a microcontroller best fit circumstances when the microcontroller is idle, waiting for

a signal or specific event to occur externally of the device (i.e. button, switch). This allows for longer-lasting batter applications and doesn't work up a lot of heat in the process. In the case of our design, the microcontroller would be using a more vital, battery consumption role as a signal processor.

3.5.3 – PWM Control Output Requirements

Because of the control over pulse-width signals sent digitally from the microcontroller to the FPGA, it's sent up nicely to use direct pin I/O to create array of sorts of PWM signals controlled by the microcontroller clock. As the pulses are sent through individual lines, it will vary the selection process of the FPGA in terms of the different variables that are present in the audio synthesizer.

Overall, there needs to be enough control/data lines from the microcontroller to the FPGA for each one of the possible variables associated with the music. In Figure 12, it shows the different select lines that will use PWM to determine factors such as tempo (from the accelerometer), instrument, rhythm, note selection, etc. so the synthesizer knows exactly how the next measure(s) of music will turn out. In Figure 13 below, the example PWM control signal is influenced by the two analog signals. They are compared to form the PWM signal with varying pulses which is sent as an encoding from the microcontroller and decoded on the other side by the FPGA. Essentially, the pulse-width modulated signal transitions to the audio engineering that occurs within the FPGA as the last phase before outputted to sound through the headphones.

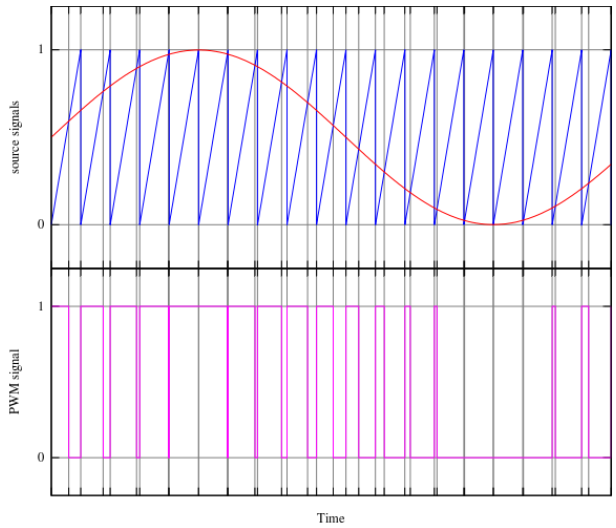


Figure 13 – Sampled PWM Signal from a Carrier and Message Waveform

3.5.4 – Memory Requirements

Non-volatile memory on the microcontroller itself should not be a problem with the Music In Motion implementation. The more important feature is being able to store

as much on the cache during execution for faster calculations but even that shouldn't be an issue. Most microcontrollers nowadays have a FLASH/ROM and RAM embedded memory modules that have enough space to account for immediate data values.

For Music In Motion, there needs to enough memory for a number of exclusive processes occurring either remaining idle, accessed and sent, or constantly running which will take up the space on the microcontroller:

- Bootstrapped program (Main software file)
- Generated control signals
- Saved comparison variables (tempo feedback, milestone check)
- Program variables
    - Software generated
    - GPS coordinates
- Accelerometer data

Software memory management can cut down on much of the used memory on the device such as limiting the amount of code in the bootstrapped file as well as not wasting memory on unnecessary memory allocation for variables and objects. Another element of memory consideration that isn't too important to consider when choosing a microcontroller include the interrupt table which will map to each section of the microcontroller as well the peripherals attached to it. The peripherals themselves have designated to memory in reserved address space. These factors along with individual microcontroller memory organization isn't all that important to look into when most devices now account for all of this in designated and reserved spots in memory. Whenever a company advertises how much memory is available for a microcontroller, it's a standard that they are only referring to user available memory and not reserved interrupts or forbidden address space used for the microcontroller hardware.

Below in Figure 14 is a diagram taken from the C2000 family guide that illustrates the memory breakdown on the C2000. The Flash/ROM is used by the bootstrap program when all the real-time execution occurs. The RAM will store the immediate values while using the function registers at the top of memory. This is just an example of the memory organization of the C2000 and not an implication of using the C2000 for the final design.

The white area with no label is non-accessible memory that is used by the microcontroller alone for dealing with memory behind the scenes during execution. Most of the interrupt vector table is allocated to I/O protocols. An I/O sends a signal to the microcontroller memory, is looked up in the interrupt vector table, and forces the microcontroller to interact with that particular I/O. Once the exchange between ICs is over, the microcontroller goes back to whatever action has most priority. The special function registers are the most immediate in memory and are used mainly for programmable variables and objects for the user to calculate functions during run-time.

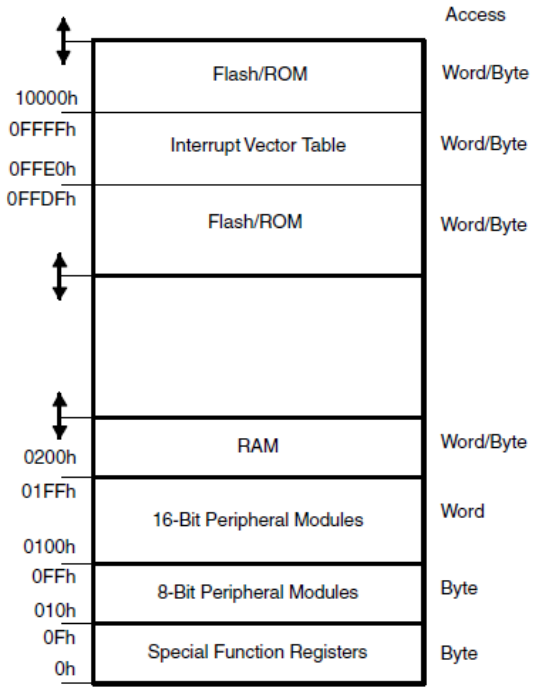


Figure 14 – Memory Breakdown in C2000

3.5.5 – Data External Transfer USB Port (Protocol)

There are a number of ways to go about implementing and interfacing USB in connection with the microcontroller. The design for Music In Motion is using USB for the user to charge the device as well as send data from the phone application to the device to dictate and customize the run to the user's liking. Without going down one path of options to firmly, there are three implementations that can be considered and all have their advantages and disadvantages.

3.5.5.1 – Hardware Implemented USB Interface

Certain microcontrollers on the market have standard USB interface already planted in the hardware that makes the implementation of attached USB simpler and without the hassle of creating protocols and extra coding that would inhibit other microcontrollers needing to find a way to communicate with the USB hardware. The extra work necessary for this implementation would be making sure the operating system on the other end (computer, tablet, or phone) has the correct drivers to start access to the device and transfer data.

The disadvantages that come with this method is not only do commercial (or amateur) microcontrollers not have this type of interface included in the hardware, but also if it does, the microcontroller can be rather expensive in comparison to other microcontrollers that use RS232 communication. This method would be the simplest way of bridging the gap between USB and microcontroller but is out of the budget of Music In Motion (Češko, n.d.).

3.5.5.2 – Universal USB/Native Interface Conversion

The second option is almost tricking the microcontroller into believing it is communicating with an arbitrary communication line rather than adjusting firmware to support the USB technology. This method uses a converter which can be used with any communication ports whether it be plain busses, RS232, SPI, or I2C just to name a few. The microcontroller doesn't need to "know" how the USB works because it's not aware it is communicating with a USB whatsoever.

However, the drawbacks also include a higher price as a result the sizing of the physical section increases due to the converter. Once again, since the budget is already tight spending more money on something as minor as USB connectivity is not necessary (Češko, n.d.).

3.5.5.3 – Firmware USB Protocol Emulation

The final option is the most plausible due to the cheaper microcontroller used for the project (doesn't require a high end microcontroller but that is for another section). The USB protocol needed for connection between both ICs is emulated through firmware located in the microcontroller. The speed of a USB transfer is much faster than what is actually needed to be sent from the phone (simple strings sent on boot of the device). The microcontroller isn't executing the audio engine or accelerometer at the beginning which makes up most of the microcontroller's activity. The low-speed data transfer of the USB is usually around 1.5 Mbit/s and since all that's sent are strings, the only requirement of the USB is just to make sure the data can be sent. The USB technology is more for powering the device when the power is depleted than a mass transfer of data that most would use a USB for on a computer or hand-held device.

USB technology is simple compared to any other IC added to the design. It boils down to four lines out of the port which control VCC, data-in, data-out, and ground. The circuitry involved with USB is more complex but that is saved for the USB Section 3.8 (Češko, n.d.).

3.5.6 – Microcontroller Comparisons

The point of the comparison section is to discuss the series of microcontrollers from Texas Instruments Inc. in regards to how they would be used and implemented. What follows is a comparison of the strengths and weaknesses of what both offer. Then ultimately make the decision easier as to which line of microcontrollers would be the best fit for Music In Motion. The research of these devices is not to confirm one or the other as the microcontroller for the project, but more of an understanding of how each particular processor would manage the other ICs in the design.

3.5.6.1 – MSP430 Series

The MSP430 in comparison with the C2000 has a distinct of advantage of requiring less power to operate as the architecture is less complex as well as other features being limited. With that said, the concern with the MSP430 is if it would be able to provide enough I/O, clock speed, and data handling so nothing is inhibited for the accelerometer, GPS, or FPGA when working constantly with the microcontroller. The MSP430 is a 16-bit architecture which is probably the main difference between choosing the MSP430 and C2000 series microcontrollers. The MSP430 stops at 16-bit while C2000 go all the way up to 64-bit.

The greatest quality of the MSP430 is its low-power consumption and versatile microprocessors for many simple applications and easy understanding of the hardware. With less pin-outs and simplicity of design, the MSP430 is perfect for single application processes for a low-cost. The top CPU speed in the series is 25 MHz though limiting the capabilities that other high-end microcontrollers can handle without any problems. There are standard peripherals that come with the MSP430 which are vital to the Music In Motion design which include an internal oscillator, timer with pulse-width modulation, watchdog, USART, SPI, I²C, and 16-bit analog-to-digital converters.

3.5.6.2 – TMS320LC240xA Series

The TMS320 microcontroller line from Texas Instruments Inc. has a line of products that range from 16-bit architecture up to 64-bit architecture for intensive applications. The C2000 in particular are 32-bit and for the project, that is as high as we need. That allows for enough bus length for I/O purposes and also allows more flexibility of memory and data transfer. The obvious drawback is the series has a higher price threshold and a more complicated hardware that requires more attention to detail. The C2000 is designed for real-time control application with different models in the series including Cortex M3, Delfino, and Piccolo sub-families among other lesser known lines. The C2000 is known for its high-performance set of built-in control peripherals such as PWM and ADC which is especially important for Music In Motion data transfer. For external support, it has communication capabilities for I2C, SPI, serial (SCI), and external memory interface. Also included in the series are up to 40 individually programmable, multiplexed general-purpose input/output (GPIO) pins. There are plenty more features included among the ones listed above, but for the sake of MIM, they are either unnecessary and/or extraneous.

In Figure 15 below, different sections of the TMS320 can be used to connect to the peripherals that are used for Music In Motion. The GPS connection could use SCI or SPI and the convenience of the analog-digital converter (ADC) is perfect for the analog signals created by the accelerometer to translate digitally to the microcontroller. For the audio control signals needed for the FPGA can be

transferred through Event Manager A and B. The external memory interface allows for easy synchronization of the memory module and microcontroller so the communication is stress-free and shouldn't require driver configuration.

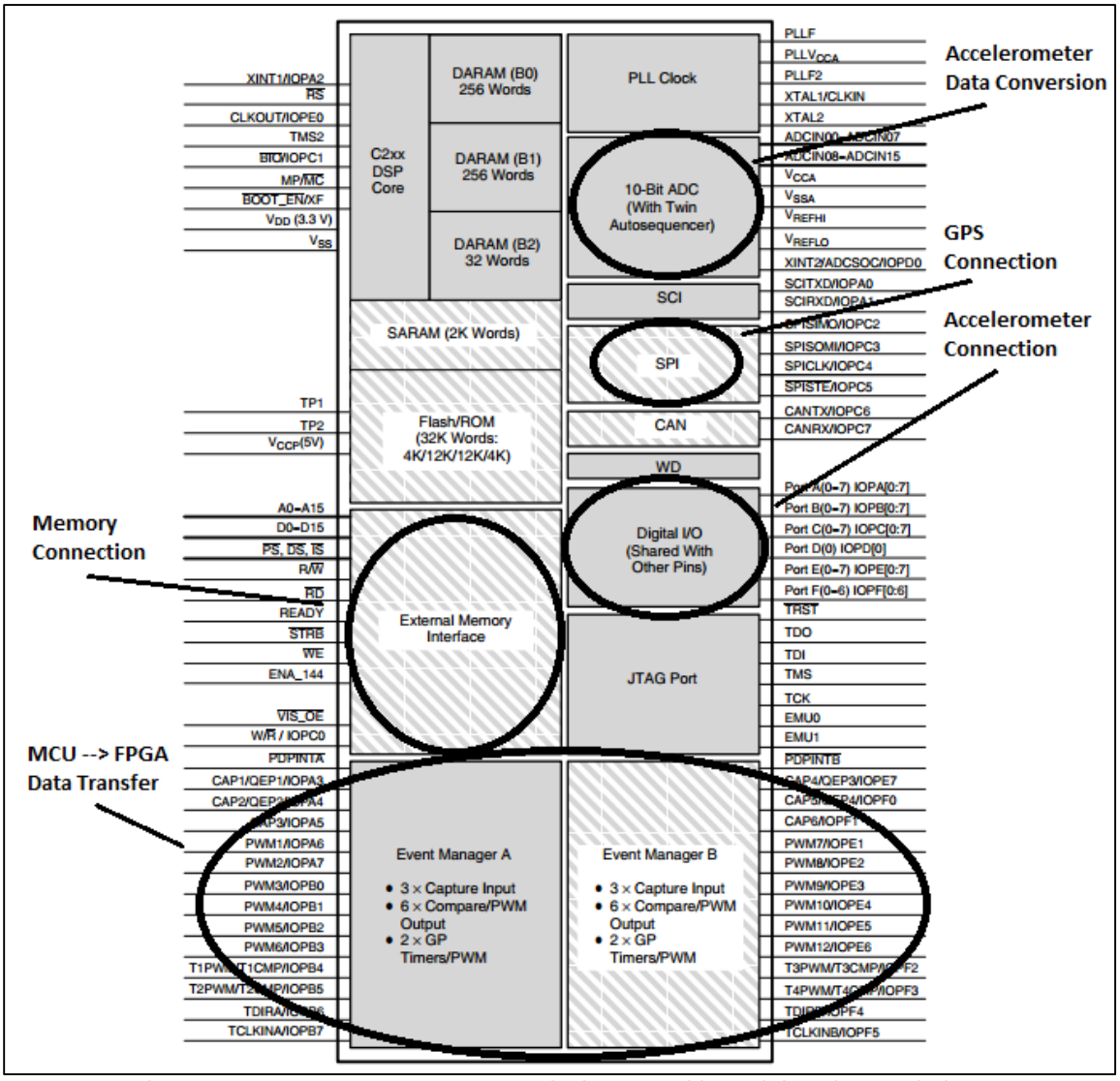


Figure 15 – TMS320LC2000 General Diagram with Peripheral Associations

The TMS320C2000 series MCUs make more sense in context of the project and requirements than the C2000. Although the MSP430 is small, compact, and easy to manage, it doesn't provide all the features and power required by the final Music In Music design.

Depending on the requirements for the other modules in the design in terms of I/O capabilities, the diagram above could vastly change from the final design. The circled areas, for the most part, is just a guess at how each IC will connect to the microcontroller. The GPS, accelerometer, and memory module may not be connected the same way as above as well the necessary use of the on-board ADC as well as the PWM connection for the FPGA.

3.6 – D/A and A/D Conversion

3.6.1 – Digital to Analog

The digital to analog converter is one of the main determining factors for the level of sound quality that will be outputted from the Music in Motion device, assuming a good bit depth in the digital output signal for the audio. After the PCM audio data is compiled and sent out of the FPGA audio engine it needs to be converted into an analog waveform. Only then can it be amplified and sent out through a speaker. In order to convert it to analog a DAC will be used. Instead of simply buying a chip, the design will be implemented from reference designs. As a result, the circuit will be customizable, which will help to balance the trade-off between quality and size. There are two main types of digital to analog converters that this section will investigate in detail. The result of which will be a decision on which one will be implemented. The actual design will be covered in depth in the design section.

3.6.1.1 – The Binary Weighted Type

The binary weighted DAC is a simple circuit that does not require too much hardware or complexity in design. The bits are represented by a weighted summing amplifier that has switches on each branch. The switches represent each bit being either on or off. Once a branch is turned on, its sum is weighted in relation to its branch resistor relative to the reference feedback resistor on the operational amplifier. The simplicity of this design is enticing due to the speed at which this section could be prototyped. Figure 16 illustrates the binary weighted type design.

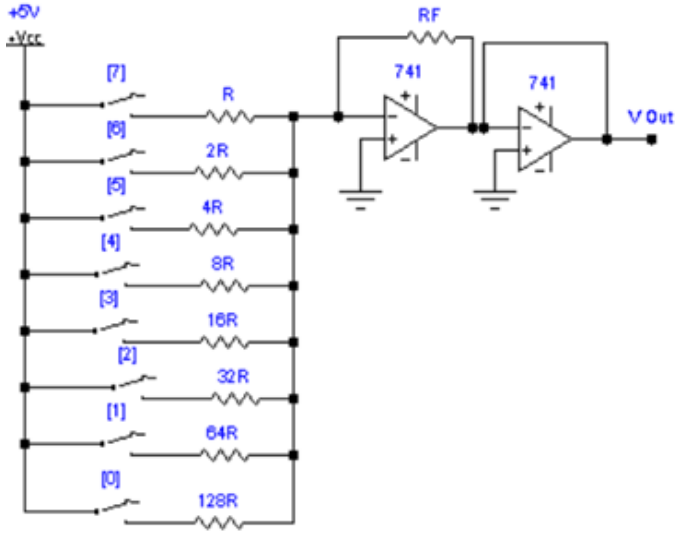


Figure 16 – 8-bit Binary Weighted DAC

The main issue with this type of DAC is the fact that as the number of input bits increases, the current on the least significant bit will decrease. As a result, there eventually will be an attenuation of the signal. (Rabiee)

3.6.1.2 – The R-2R Ladder Type

The most modern digital to analog converters follow this format. This is because they are only slightly more conceptually complicated than the binary weighted type to understand and the R-2R does not have the same problem that the binary weighted type has as described below in Figure 17.

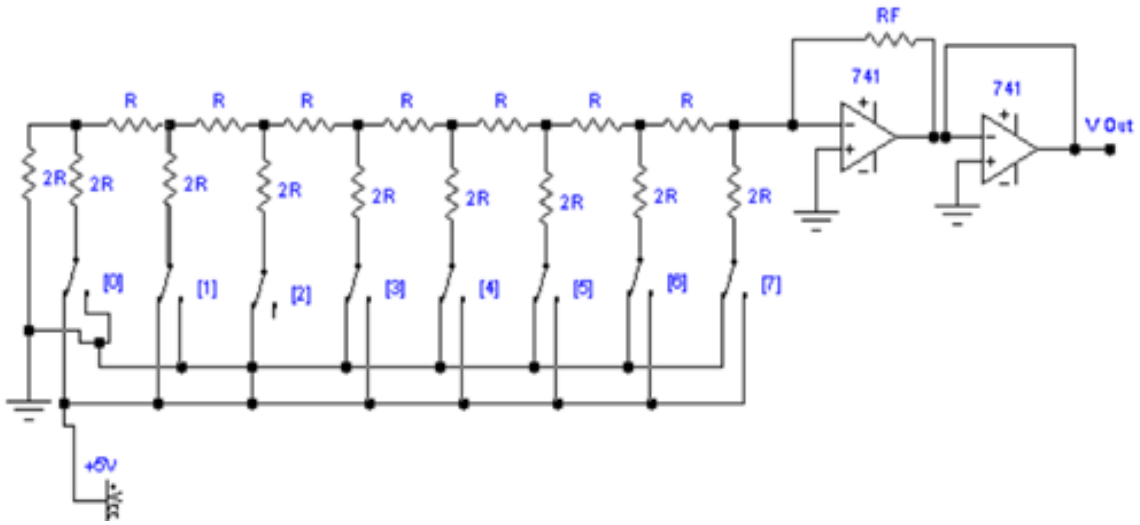


Figure 17 – 8-bit R-2R DAC

First Stage – The first stage of this basic 8 bit R-2R DAC example is the resistor network. This stage of the DAC is what actually converts the digital byte, in this case, into an analog voltage signal. The principle behind this configuration is the voltage divider. The most significant bit is not divided; the 2nd most significant bit has 1 voltage divider. As the bits become less and less significant, the voltage division is compounded across multiple branches. The switches in this resistor network are simply models representing the on and off or V_{HIGH} and V_{LOW} of the digital input signal.

Second Stage – The second stage, the first operational amplifier, is necessary if one desires to customize the DAC in order to fit a specific application. This is done by choosing the feedback resistor labeled R_F to give the value of gain desired for the audio signal. This being the case, it is possible to implement a variable resistance in order to tweak the DAC's gain during the prototype phase. This flexibility will prove to be integral to a successful and quick prototype test. This desired impedance variability can be prototyped using the resistor boxes in the UCF electronics labs.

If the variable resistance is decided to be preferable for the final implementation, a potentiometer can be used. This will give an added feature of instantaneous and easy to access volume control for the user. However, it appears after some quick research that potentiometers are large. This is bulkiness is not preferable for the MIM device, as its dimensions are specified to be limited.

Third Stage – The second operational amplifier is the third stage. This is a simple buffer that is meant to induce a phase-shift of 180º and result in a gain of 1 from its input to output. This is essential for two reasons. Primarily, having this operational amplifier drastically reduces the loading effect. The loading effect with this buffer at the output is insignificant, therefore is negligible in later calculations. This is desirable because the DAC's output will be driving the speakers, which have a load. Speakers typically do not have huge output impedances; however, without the op-amps the loading effect will attenuate the signal much more noticeably. (Rabiee)

3.6.1.3 – DAC specifications for the MIM device

Bit Depth Resolution – This digital to analog converter will be used for the audio signal. Using the on board memory module of the FPGA requires a custom audio synthesizer which will save memory resources and increase the responsiveness of the audio output. The resolution will be sixteen bit for the audio signal which outputs from the FPGA which matches CD quality audio signals. This will allow the output to have excellent dynamic range of 65,536 step values.

3.7 – USB Port

Universal Serial Bus (USB) communication provides two important and complex functions required by Music In Motion. In order to charge the lithium ion batteries built into the design, a USB port can provide enough current recharge the device. We found the most convenient way to recharge was through USB although there are other methods discussed in greater detail in the section 3.7 of the document. The other feature the USB enables is the ability to transfer data from an android phone to Music In Motion that uses our application to add variability to the run. All the ICs on the PCB are connected to either designated I/O pins or MCU enabled serial communication. The user wouldn't be able to access communication with the microcontroller through these methods so the USB allows that external access. The USB port integrated circuit is pivotal for the device to function as well as play a secondary role of user data transfer and off-time battery charging.

3.7.1 – Data Transmission Protocol

The data transmission was discussed in minor detail in the microcontroller Section 3.5.5. The information discussed above is focused on the established connection

between the output of the USB port and how the microcontroller receives it. Very little data is actually sent from the phone because the user is only deciding between a few attributes of the run such as distance, time, difficulty of run, etc.

USB communication is based on a system of pipes that is a connection between a host that controls the information and a peripheral device; when this connection is initially established, an endpoint is created at the peripheral end of the device and is “permanent” until, obviously, the physical connection of the two ends are disconnected. The host must decide what actions will take place as the peripheral device has no say in the decision. This is similar to the master/slave relationship of SPI or I2C except the host can permit the peripheral device to decide when it should send data on its own terms. This can cause problems if the allowance creates bottlenecking on the host controller’s end. Most of these issues can easily be resolved through software decisions, and worst case, the USB can make all the data transfer decisions without the peripheral choosing when to start an action.

The data can be sent through the USB by three different methods: isochronous, interrupt, or bulk. Isochronous guarantees a certain data rate but is susceptible to data loss when transferring. This would be for applications such as audio/video streaming which require a large wave of data transfer at a high frequency. Interrupt transfers require quick responses such as keyboard or mouse connections. The final method of transfer is in bulk. This will send large of data all at once until everything associated with that file/action has been sent. If a file is being transferred from flash drive to a computer would be an example of bulk transfer. For the design, it makes sense the data will be sent as bulk because the user is applying the variables set in the phone application at the start-up of Music In Motion before the run.

3.7.2 – Physical Port

The physical port for Music In Motion will mostly use micro-USB because of the ability to transfer information homogeneously (cannot tell which end is which) while the size of the micro can maximize space for other components on the PCB. The mini-USB could work as well but the problem with mini is that it usually deals with one-way information which we don’t want to limit on the design. The standard is considered the default because of standard-to-standard cables so easily available on the market but the amount of space it would take up along with the unnecessary max transfer speed leaves micro-USB as most optimal for the design.

There are two types of pipes consistent in USB: stream and message pipes. Each type of pipe serves its purpose and transferring and signaling data for device A and B communication. Message pipes are bi-directional that send control signals between the two devices for simple commands and interrupts so each device knows what’s going on at the other end. This allows data to run smooth without bottlenecking or deadlocking. Steam pipes are unidirectional which send data from B to A and vice versa.

The most common standard USB port is 2.0 while 3.0 technology is slowing growing in the electronic community.

3.8 – Power

The power design for the Music in Motion device is integral to the success of this project. The main reason is for user convenience. A successful user end product such as the MIM running device relies heavily on how easy to use it is and how efficient it may be for its purposes. In this case, the efficiency is presented in the form of a compact size and using the least amount of PCB space as possible while maintaining a proper heat dissipation, power requirements, functionality, and safety standard. Additionally, the efficiency must be introduced as a qualitative weight requirement that is simply lightweight.

These qualitative specifications rely heavily on the power system of this design. This is because the heaviest part of this device is by far the battery. Additionally, much of the PCB space will be used for giving power to the devices. In order to do this properly, the power needs to be manipulated such that each sub-system's power requirements are met with exact specificity. This requires building additional circuits, which requires additional PCB space. Therefore, the efficiency of the power system design is crucial to the size of the final product.

3.8.1 – Battery Comparisons

There are a vast amount of options regarding battery choices for a power system design. The following table presents the main battery criteria that were discovered during the research phase of this project.

Table 11 shows the possible rechargeable batteries to be implemented for the power system in the Music in Motion device. In order keep the options open and to research the power for the best possible solution, the second table, Table 12, presented below that shows the possible non-rechargeable batteries. (ICCNexergy, n.d.)

Battery	Voltage	Energy Density	Advantages	Disadvantages
Lithium Ion	3.6V	240Wh/kg 550Wh/L	Small and lightweight, self-discharge rate is low as well	Expensive and needs a protection circuit to be implemented to keep voltage and current at safe levels.
Lithium Ion Polymer	3.6 V	260Wh/kg 540Wh/L	Superior stability for over-voltage and more safe than Lithium Ion	Expensive and needs a protection circuit to be implemented to keep voltage and current at safe levels.
Lithium Ion Iron Phosphate	3.3 V	108Wh/kg 220Wh/L	Superior cycle life and highest Li-ion safety rating	Low energy density compared to other Li-ions
Nickel Cadmium	1.2 V	90Wh/kg 210Wh/L	Cheap, long battery life, high power	Memory effect and containing toxic cadmium, which needs recycling
Nickel Metal Hydride	1.2 V	125Wh/kg 400Wh/L	Superior capacity to NiCD, less memory effect, and less toxic	Charging mechanism is a bit more complex than NiCD
Sealed Lead Acid	1.5 V	60Wh/kg 100Wh/L	Easy to maintain, low self-discharge rate, and cheap	Bulky and heavy

Table 11 – Battery Comparisons for Rechargeable Batteries

Battery	Voltage	Energy Density	Advantages	Disadvantages
Alkaline	1.5 V	125Wh/kg 400Wh/L	Cheap with a decent drain rate	Bad performance at low temperatures and high impedance
Lithium Iron Disulfide	1.5 V	310Wh/kg 560Wh/L	Superior rate capability	Requires multiple batteries for any excess of 1.5 volts
Lithium Manganese Dioxide	2.9 V	240Wh/kg 500Wh/L	Very long life, wide operating temperature and superior durability	Subject to shipping regulations (all Li-ion)
Lithium Sulfur Dioxide	2.8 V	265Wh/kg 400Wh/L	Good high rate capability	Subject to shipping regulations (all Li-ion)
Lithium Thionyl Chloride	3.6 V	520Wh/kg 1050Wh/L	Incredibly high energy density and service life is 15-20 years	Subject to shipping regulations (all Li-ion)

Table 12 – Battery Comparisons for Non-Rechargeable Batteries

3.8.2 – Charging

The main purpose of the research into battery charging is to figure out the following focal points: inputs voltage ranges, input current ranges, capacity with charge time, and safety. After carefully considering the battery options available, only the lithium ion type batteries will be covered further, for the sake of not writing unnecessary elements in this section of the paper. These are the most serious candidates for the Music in Motion device, therefore will be more thoroughly researched than the others. The reason for this is because, according to the tables above, they have the greatest energy density and output voltage. Therefore the next subsections will research lithium ion batteries with greater depth. (Battery University, n.d.)

3.8.2.1 – Input Voltage and Current

The USB will be used to charge the battery for this device and it delivers 5 volts and 500 mA. Any lithium ion type battery can handle this voltage and current in order to charge it to capacity. Li-ion batteries cannot be overcharged; therefore the current will respond to the terminal voltage such that there will be no leakage current.

3.8.2.2 – Capacity and Charge Time

These two variables are intimately connected. In efficiently use the Li-ion battery, it must be charged correctly. Once the battery is at its terminal voltage, it simply will not accept any more current, regardless of its capacity. This means, once it has reached it terminal voltage, it should be near full capacity. If charged incorrectly, it can reach terminal voltage with only 50% capacity. In order to avoid this, the charge time is extended by using a small current. The Li-ion batteries have no trickle current, unlike lead acid batteries, for example. Usually, Li-ion batteries have a terminal voltage of 4.2 V per cell. A higher voltage may increase capacity, but will cause oxidation and reduce its life span.

Figure 18 below demonstrates the characteristics of input current and battery voltage at multiple phases during charging.

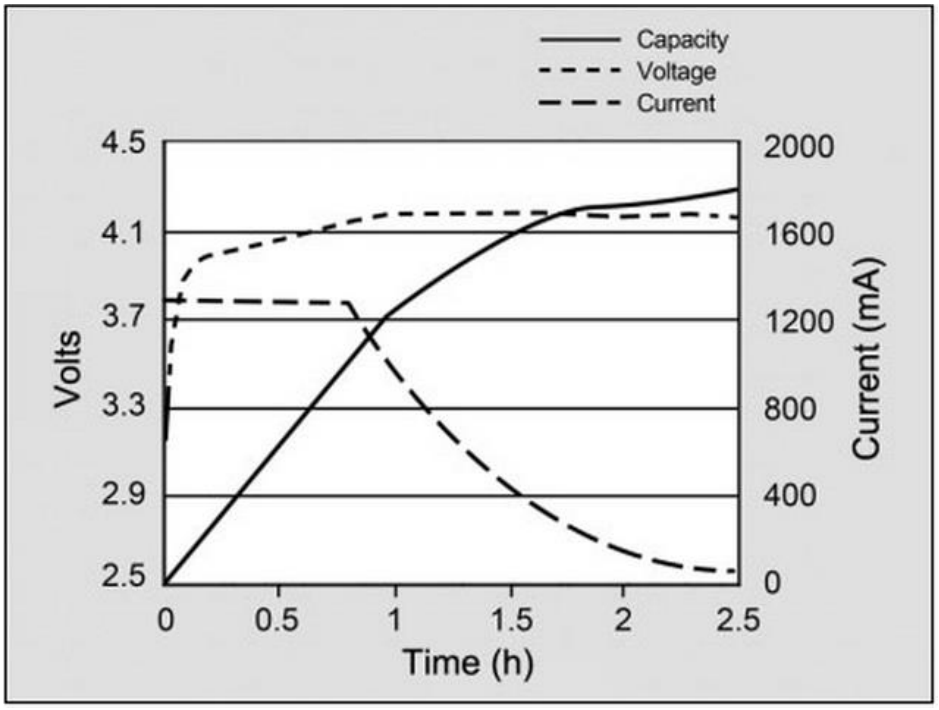


Figure 18 – Lithium Ion Battery Characteristics during Charging

During the first stage, the input current will remain constant until the terminal voltage is reached. Once it is reached, the saturation charge phase is entered. This stage has the largest duration and the most important in order to optimize charging efficiency. This graph illustrates a correctly charged battery's characteristics. Ideally, the terminal voltage is constant while the battery's capacity is increasing and the input current is simultaneously decaying exponentially. After that, the final stage represents the ideal capacity reached while eliminating the input current.

3.8.2.3 – Safety

Lithium Ion batteries come equip with built in safety provisions, such as charge and pressure regulating circuits. This is advantageous because it does not need to be designed and the battery can be deemed safe for the consumer and the MIM device. Regardless, it is still crucial to take certain measures in order to guarantee proper use of the battery and to stay in adherence to its safety standards. The following precautions should be observed and taken:

- Discontinue charging the battery if it is heating up
- Charge at normal temperatures, i.e. not freezing
- The battery does not need to be fully charged
- Do not feed the battery more than 1 ampere
- Do not forget to take it off of the charger

3.8.3 – Estimated Life Cycle

Dabbing our feet a little more into the science behind the Lithium Ion battery, it is known for Li-ion batteries that overall lasting power of the battery depletes over time whether the battery is used frequently or not so often. Going with the Li-ion battery, we know that there are safety issues in regards to the circuit as well as being expensive, but going off strictly battery life, it is top of the line and what we expect to power the device. One regard is the stress conditions the device must go through during operation which is either extreme temperature conditions. Figure 19 displays loss rates for Li-ion batteries:

Lithium Ion Life Cycle Losses by Temperature		
0 °C (32 °F)	25 °C (77 °F)	40 °C (104 °F)
6%	20%	35%

Figure 19 – Loss Rates by Temperature per year

We take two things into account when talking about battery life: how long will it last for a single charge and how long it plans to last over the lifetime of the device. Well the latter we do not have any control over as they start degrading as soon as they leave the factory. They will only last two to three years from the date of manufacture whether you use them or not. So for the runner's sake, we want to optimize the life of the battery for a decent amount of runs without having to plug it in to charge.

4 – Project Design

4.1 – Hardware Design

The advantages and disadvantages will be analyzed for integrating the GPS and accelerometer with the microcontroller. Furthermore, analyzing both the signal processing and control system design. The tempo control and audio control module resource utilization for the audio engine is also discussed in detail.

4.1.1 – 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.

There were certain factors considered for the scope of this project that were imperative in considering for selecting the proper accelerometer: output format, axis, maximum swing, sensitivity level, bandwidth, interface, and mounting type. There are two options for the interface output protocol: SPI (3 or 4 wire) and I^2C, both of which are easily implementable inputs formats for the ATmega328p. The choice between these two options was determined to be SPI 4-wire as there were enough pins available for 4-wire and made communication seamless when one pin is assigned for MISO and the other for MOSI (Master-out Slave-In and vice versa).

Accelerometer Chosen – 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.

This section covers each of the specifications discussed in the research section in regards to why the ADXL345 was chosen. There is a huge pool of research and information regarding this accelerometer. In addition, there is even documentation

on this accelerometer in relation to its uses for a pedometer, a device which detects footsteps and counts them. This information can be used later on as a reference for the design of the filtering and communication protocols.

In addition, there is an on board digital filter and a 32 level FIFO. This first-in first-out buffer minimizes the activity and work load on the sensor microcontroller and therefore lowers the overall system's power consumption. Figure 20 illustrates the internal system diagram of the ADXL345 three axis digital accelerometer.

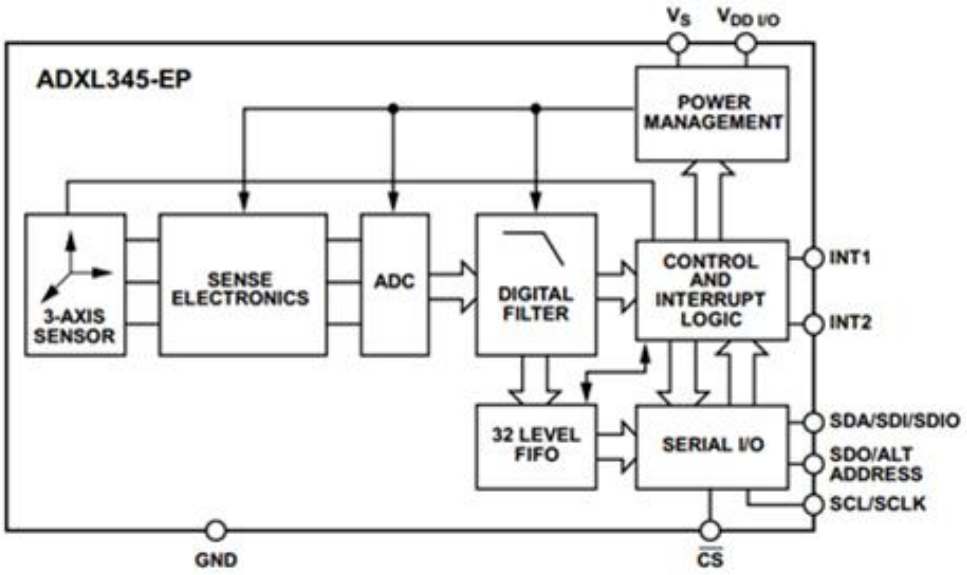


Figure 20 – Functional Block Diagram of ADXL345 Accelerometer

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.

Axis – This is a three-axis capable accelerometer: X, Y, and Z; these correspond to the vertical (yaw-axis), forward (roll-axis), and side (pitch-axis). It is advantageous to have the option of three-axis even if the objective only requires one axis. For our project, it adds the ability for the device to move in any direction, causing a signal to be generated when passing above the absolute value of the established threshold. The axis is arbitrary because the motion of the runner and the orientation of gravity, proved through testing, that the x, y, and z axis can all create proper footstep signal generation without false positives. This is because

the orientation of the accelerometer may change and this will result in the useful data appearing on a different axis than previously recorded. That being said, it is expected that only one axis at a time will provide useful data (the impulse train of interest). The other 2 axes will simply be considered noise to be filtered out by a low-pass filter. In order to minimize the complexity of this three-dimensional configuration, we placed the accelerometer in an orientation that remains somewhat consistent with the impulse train. For example, when prototyping, it is found that the arm swinging motion causes problems with useless data, the orientation can be adjusted. This can be achieved by placing the accelerometer on a part of the body that does not move in more than one dimension, which we determined to be at the waist with the help of an adjustable belt. The runner's waist simply bounces up and down, which corresponds to the yaw-axis (vertical).

Resolution – This accelerometer has a user-selectable resolution ranging from 10 to 13 bits. 10 bits was plenty for the purposes of this project because peaks are easily detectable. Also, peaks are very distinguishable in relation to the noise margin; therefore the SNR (sound noise ratio) is very high. As a result, 10 to 13 bits is excellent turned out to be a nice range for Music In Motion.

Maximum Swing – Maximum swing is essentially the limiting rails of the observable input in relation to acceleration, measured in g (9.81 m/s2). For example, an accelerometer with ±10g can measure -98.1 m/s2 to +98.1 m/s2. Any input outside of this domain will simply hit the rails and clip. The scope of this project requires a maximum swing of ≈±5g. This accelerometer has the maximum swing options of ±2g, ±4g, ±8g, ±16g output resolution. The setting chosen for Music In Motion was the ±16g resolution due to the coverage/range it provides for the project.

Sensitivity Level – The level of sensitivity achieved by the ADXL345 is a function of the output resolution chosen. For ±2g it gives 256 LSB/g, ±4g it gives 128 LSB/g, ±8g it gives 64 LSB/g, ±16g it gives 32 LSB/g. These reported values are the typical, not minimum or maximum. This is the quantitative description of the sensitivity level. Qualitatively, it can simply be said that the ADXL345 has a sensitivity level that was sufficient for this project.

Bandwidth and Output Data Rate – The bandwidth is defined at half of the output data rate. For this project and its applications the output data rate was set to 800 Hz, therefore the bandwidth is 400 Hz. This number was chosen because the ADXL345 has been tested extensively using this output data rate and the results are documented on the data sheet.

Dimensions – Small and thin: 3 mm × 5 mm × 1 mm LGA package. This is a very small chip for an accelerometer and because the breakout was custom made during the prototyping stage, it remained small on the final PCB because the design was refined several times. This was important to the small dimension of the final Music in Motion product.

4.1.2 – GPS

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. The patch design is standard for GPS modules but requires an initialization process before the GPS can receive coordinates. The process of the microcontroller polling data from the GPS is easy due to the simplification of the UART communication.

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.

4.1.3 – Audio Engine Hardware Architecture

Two primary Audio Engine designs were prototyped and tested for the Music In Motion System. One design utilizes and FPGA while the other is built around the VLSI VS1053B MIDI controlled synthesizer. The primary advantage of the FPGA design is that the Tempo Control system can be designed to function in parallel to the Audio Engine on a single integrated circuit. However, this design in labor intensive and requires the entire audio and tempo control system to be designed from scratch. Conversely, using the VS1053B allows for a much quicker implementation and shorter design time. The final design has been implemented

with an Atmega328 microcontroller as the tempo controller and the VS1053B as the audio engine. Therefore, this design includes two primary subsystems.

4.1.3.1 – Audio Control System

The Music In Motion final design utilizes the VS1053B design. This design was implemented after extensive FPGA prototyping and comparison to the MIDI synthesizer. From a strictly engineering perspective the FPGA design is superior, given unlimited time and resources. However, being that this is a senior design project and time and resources are limited the VS1053B has been utilized as the core of the MIM audio engine.

This IC is a class D audio amplifier and digital synthesizer and MIDI control signals are transmitted across a standard 31.25kBaud UART protocol. In order for the unit to function as a real time MIDI synth, as opposed to operating as a MIDI file player, a specific power up sequence is required. To accomplish this the VS1053B will be powered by the Atmega328 that function as the tempo controller. The microcontroller must set the GPIO pin 3 to high before sending power to the synth.

4.1.3.2 – Tempo Control System

This system is implemented on an Atmel328 microcontroller. The input pulses are received from the sensor microcontroller over an interrupt pin. A rising edge input triggers a series of functions that then trigger the MIDI control signals utilized by the VS1053B to generate output. This microcontroller processes the input pulses from the sensor microcontroller into the musical tempo map as well as storing the instrument and song data that represent the musical output of the system.

4.2 – Software Architecture

The software-oriented construction of Music In Motion can be divided into three overview sections that operate different components of the project:

1. Sensor Control System - for control signals that function GPS and accelerometer, ports and data transfer.
2. Audio Engine – for taking signals from microcontroller and composing music for audio output.
3. Android Application (Unimplemented) – Software app native to Android devices that customizes individual runs separate from the Music In Motion device.

4.2.1 – Sensor Control System

The programming language used for the microcontroller of sensor control system is C. The C language allows for maximum memory management and efficiency,

and it also creates a natural control of different sets of data accessed from separate communication channels (SPI, UART, etc.). These channels have function calls that establishes the connection between the microcontroller and IC on startup.

A very basic interpretation of what the code will be doing can be explained in the next few sections but most of the minor details do not affect the software aspects overall for everything the microcontroller handles. The microcontroller will always run as long as the device is turned on. This means the watchdog timer is turned off because there is no need for the microcontroller to go into sleep/idle while there are constant operations needed to be executed. The stored program should loop through each IC protocol and check the buffers and see if the data is ready to be collected. This will always be accounted for since it's expected the GPS and accelerometer will constantly be sending data. Aside from data retrieval, the microcontroller will calculate values for the audio microcontroller using different control algorithms and organize milestone execution from the received GPS data. The final execution of a loop in the code is checking to see if the audio microcontroller is ready to receive the data and subsequently send it.

The code size stored in the program data is a decent size to accommodate for all protocols that communicate among ICs, temporary dynamic memory used in algorithm execution and data storage, and intermediate functions that handle tasks with data being transferred or constructing complicated control signals for the audio microcontroller.

4.2.1.1 – Sensor Input Processing

Once that data is stored on the microcontroller from the sensors, it will be treated homogenously despite knowing which IC it originally came from. In the code, most of the data access will be divided into functions that allocate a certain set of memory needed for each sensor input retrieval. There is also protocols supplied by the company distributing the IC which is used in the code without having to create protocols from scratch.

Accelerometer – 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 Figure 21 below.

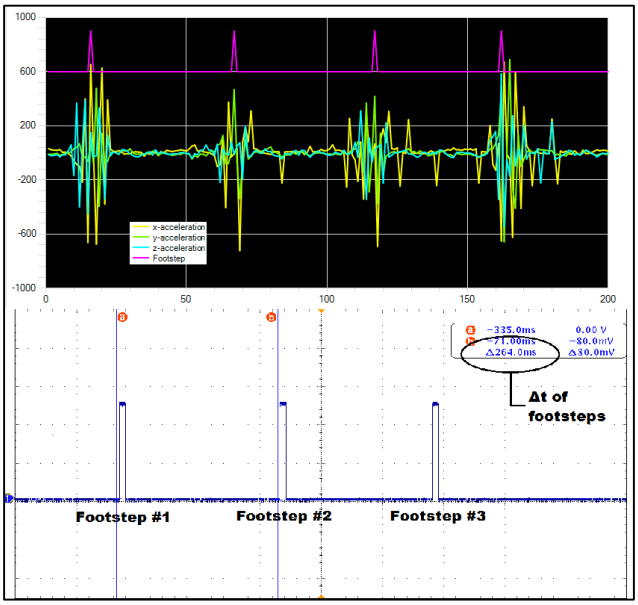


Figure 21 - Running Session Graphed and Impulse Detection

GPS – 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. Figure 22 below shows the formula for converting these values. The cardinal directions make the coordinate values either positive or negative.

North / East→ Positive;
West / South → Negative;
$\text{Minutes} = \text{Latitude}_{\text{initial}} \% 100$
Degree Minutes = (int) (Degree Minutes / 100)
Degree Decimal = Degree Minutes + (Minutes / 60)

Figure 22 - Equations for Cardinal Directions

Figure 23 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 23 - GPS Strings and Parsed Data

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. Figure 24 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.

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

Figure 24 - Haversine Equations

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. Figure 25 below shows the distance calculation using the speeds given by the GPS data over the time interval between the readings.

1 knot = 0.514444 meters per second
Distance = $[[(V_i \cdot 0.514) + (V_{i-1} \cdot 0.514)] / 2] \cdot \Delta t$
Distance = Distance · 0. 000621371 "Miles to Meters"

Figure 25 - Kinematic Equation with Speed and Time

4.2.1.2 – Control Signal Generation and Transmission

Generated the proper control signal is key for the microcontroller. This is the most important aspect it must handle as wrong transmission could result in funky results for the audio output from the audio microcontroller. The microcontroller takes the data from both the GPS and accelerometer, translates it in real-time, and constructs the signals based on algorithms that take a multitude of factors into account.

How long the runner has been running is a huge consideration because it is the most telling of how tired or in the moment the runner is. If the run is just starting, the algorithms in code will construct control signals that will generate calmer, less extreme music because the run is just beginning. As the time calculated increases, the music should grow more intense and thus the control signals will adjust for that assuming the algorithms in place are correct. A more immediate factor in the experience is the tempo which is established by the runner's feet. Because of the constant feedback, the code is readjusting the values from the accelerometer to make sure the beat is in tune when that particular control signal is sent to the audio microcontroller. As for the instruments and melody, there are plenty of different considerations as well as random generation that can help choose instrument combinations.

4.2.2 – Audio Engine

The MIM Audio Engine controls the Soundtrack playback. Four primary control modules encompass the Audio Engine. These are the Tempo Control Module (TCM), Soundtrack Control Module (SCM), Alert Control Module (ACM), and the Output Control Module (OCM). Six PWM input signals from the microprocessor are utilized by these modules to generate a PCM audio signal (16bit 48.8 kHz) with a bandwidth marginally larger than that of CD quality audio. Four PWM signals are utilized by the SCM to generate rhythm and note values for each of the individual instruments. The fifth PWM signal is used by the TCM to create real-time tempo mapping while the sixth PWM signal. The OCM utilizes the Xilinx IPcore function generators as virtual oscillators and uses and additive synthesis method to generate the PCM audio signal output.

4.2.2.1 – Tempo Control Module

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 Figure 26. This value is the sum of one third the InstTempo value and two thirds the RollAveTempo value.

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

Figure 26 - Rolling Average Tempo

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 in Figure 27. 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.

$$\begin{aligned} AveTempo4 &= AveTempo3 \\ AveTempo3 &= AveTempo2 \\ AveTempo2 &= AveTempo1 \\ AveTempo1 &= InstTempo \end{aligned}$$

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

Figure 27 - Average Tempo Calculations

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 Figure 28. 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.

$$NextBeat = \frac{InstTempo}{4} + \frac{RollAveTempo}{4} + \frac{AveTempo}{2} - 12$$

Figure 28 - Next Beat Calculation

4.2.2.2 – 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.

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.

4.2.2.3 – Alert Control System

The Alert Control System generates two primary alerts. The first alert that is generated is the GPS fix alert. A quick sequence of five cymbal sounds are generated to indicate to the runner that the GPS has communication to the satellites. Secondarily, an alert is generated at every one tenth of a mile that is traversed by the runner. This alert is a single crash cymbal sound.
An interrupt pin and a second GPIO pin, which is used as an alert select line, are used to determine which interrupt to generate. When a rising edge pulse is received on the alert interrupt pin an alert function is called. This function sets all notes to off, which stops any music currently playing, and then polls the alert select pin. If this pin is set low then the GPS fix alert is played and if it is set high then the one tenth mile distance milestone alert is generated.

4.2.3 – Android Phone Application (Unimplemented)

The phone application only works for Android devices. The software development was only handled for Android and did not account for Apple users. The application is not mandatory in the execution of the device as its purpose is to send data to Music In Motion as extra variables to consider when going for a run. These variables include weight, height, gender, intensity of run, settings for milestones, etc. which will change the experience of the run in a more dynamic way then just turning on the device and going. Not all the variables need to be selected in order for the device to receive the data. Any variables that do not get sent are set as null in the memory of the microcontroller. If the device is plugged in, the data can be sent over, and at the same time, the data from the previous run is sent back to compile in the application for the user to look at. If the application is not used for the run, the variability in the music will be established by the GPS and accelerometer alone and all values associated with application data will be set to null.

All the coding for the application is programmed in Java using the Android Development Tools based Eclipse which provides packages and API for user-friendly coding. Most of the application development consists of graphically

designing the application and determining what each event does inside the application.

4.2.3.1 – Software Interface

The interface for the application is simple. Once the application is opened, a splash screen appears for roughly 3-5 seconds introducing the Music In Motion app. After that the user has a choice of viewing previous results or establishing a connection with Music In Motion. If there is not a USB connection between the device and the phone/tablet, the option to access variables will not be available as it only makes sense to send data through if there is a connection.

On the next screen (event), the user will decide on a number of drop-downs that are associated with each potential variable which can be selected. Once that selection is made, the data is changed in the code and is what will be sent through USB to the device. If the user decides to change what drop-downs were chosen, the variables will update accordingly. After all the variables are decided on, the bottom of the event displays a button that, when pressed, sends the data over and confirms the transfer. From there, the user can either go back to the main menu of the app or change variables and resend the data.

4.2.3.2 – Navigation and Accessibility

Navigation is easy as there are only three events that exist in the application: main menu, past results and update run settings. The application can be downloaded from the Google Play store and once opened, the main menu will make it easy to decide which option you'd like to choose. If you want to set your run results, you click the "connect to USB" option and the event will change to that java file. Otherwise, the user can view past results which will go to the results event.

To view the results, the event allows the user to drag up and down on the screen to see different runs and calculated averages. The last 5 runs are stored on the phone as well as averages and top times/speeds/distances from those runs. If the user wishes to go back to the main menu, there is a back option in the top right. There will always be a way to back out to the main menu.

In the run settings event, the user will either be using drop-downs or text boxes to send information. The text boxes will tell the user whether or not the information that is typed in is invalid or not. The user can leave text and drop-down fields blank. The user has the ability to scroll to the bottom to choose to send the information over to the device. Once again, they can back out by choosing the back option in the top right.

The application is simple and robust so the user attributes it as an add-on to Music In Motion. The user shouldn't feel the application is too much to handle for adjusting settings for each run.

4.3 – Block Diagrams

The Music In Motion system has three main sections. Many of the areas are represented as “black boxes” as you can’t tell what is going inside each based on the diagram but shows more of a flow of data/signals and what the main process of the device is about. The sensor array input, microprocessor and memory storage, and the FPGA audio engine.

4.3.1 – General Block Diagram

Figure 29 illustrates the data flow between these elements. Information about the runner and their environment is collected by the accelerometer and the GPS input sensors. This data is routed to the microprocessor where it is filtered, analyzed, and distributed both within the device and to the user via the on board USB connection. Running metrics, such as distances and time of each run, are sent to the memory module for tracking the user’s development. The signal from the accelerometer is used to generate a PWM tempo clock signal that is utilized by the audio engine to determine the variable quarter-note lengths that will synchronize the music to the runner's stride. Direction, location, and elevation information from the GPS is used the microprocessor to generate the PWM signals that control the rhythmic and note output from the audio engine. Finally, the audio engine outputs an unsigned sixteen bit PCM audio signal to the DAC which converts the digital PCM signal to the two channel analog audio signal that is enjoyed by the runner.

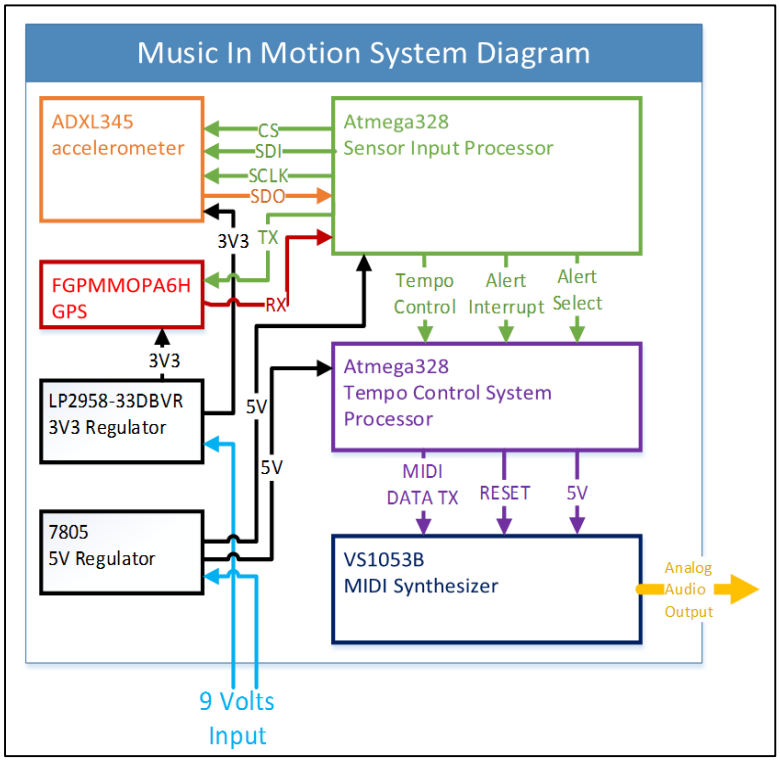
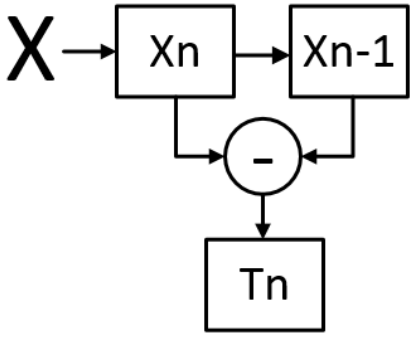


Figure 29 – MIM Overall Diagram

4.3.2 – Tempo Control System Diagrams

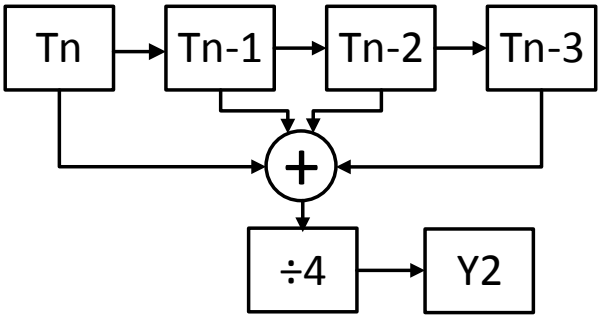
The Tempo Control System processes the input pulses from the sensor processor and generated the tempo map output and utilizes three primary tempo measurements. Initially the instantaneous tempo is measured as the milliseconds which have passed from this current pulse input and the most recent pulse input. This measurement is shown in the following Figure 30.



X=Tempo Input Pulses From Sensor Processor
Tn=Instantaneous Tempo Measurement

Figure 30 - Instantaneous Tempo Measurement

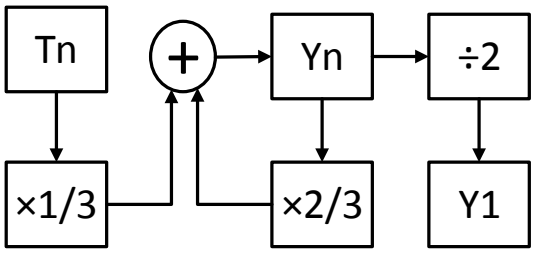
Utilizing just the instantaneous tempo measurement allowed for the pulse to be placed on the runner's footfall in nearly every instance. However, this in not consistent enough to be used as a musical tempo and the jitter that occurs in the musical output is distracting. To counter act this two more measurements are used. Figure 31 shows the standard average tempo measurement.



Tn=Instantaneous Tempo Measurement
Y2=Average Tempo Measurement

Figure 31 - Average Tempo Measurement

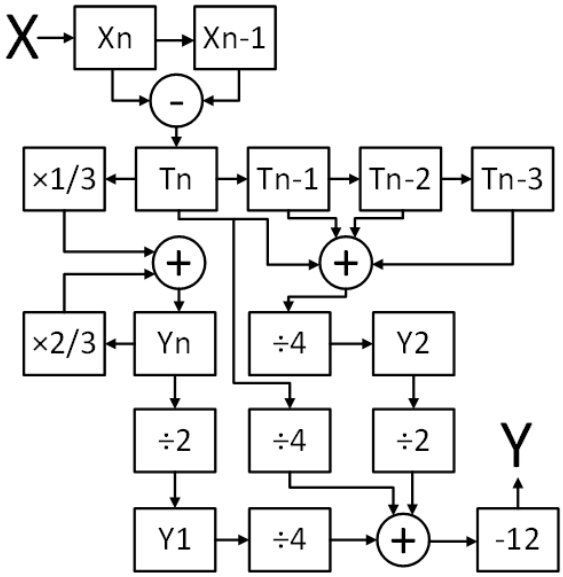
This average measurement did not turn out to be sufficient to smooth out the tempo and another tempo filter was instantiated, the rolling average tempo measurement. This is an infinite impulse response process that is two thirds the current measurement and one third the instantaneous tempo measurement and can be seen in the Figure 32.



Tn=Instantaneous Tempo Measurement
Y1=Rolling Average Tempo Measurement

Figure 32 - Rolling Average Tempo

Finally these three measurements are summed together into the final tempo map output. Figure 33 shows X as the sensor processor pulse input then the full discrete time filtering process with Y as the final processed tempo map output.



X=Tempo Input Pulses From Sensor Processor
Tn=Instantaneous Tempo Measurement
Y1=Rolling Average Tempo Measurement
Y2=Average Tempo Measurement
Y=Processed Tempo Map Output

Figure 33 - Discrete Time Tempo Diagram

4.3.3 – FPGA Audio Engine Prototype Diagrams

Figure 34 illustrates the signal flow through FPGA MIM Audio Engine prototype. The PWM Tempo Signal is routed to the Tempo Control Module which generates the Tempo Control Signal. The Alert Signal runs to the alert module which determines if a positive or negative alert is to be generated and sends the appropriate alert note and tone signal to the Oscillator Block. The final control signals are routed to the Soundtrack Control Module which utilizes the Instrument Decoder to route the signals to the appropriate instrument module. Finally, the Oscillator Block generates the sixteen bit PCM Audio Output Signal.

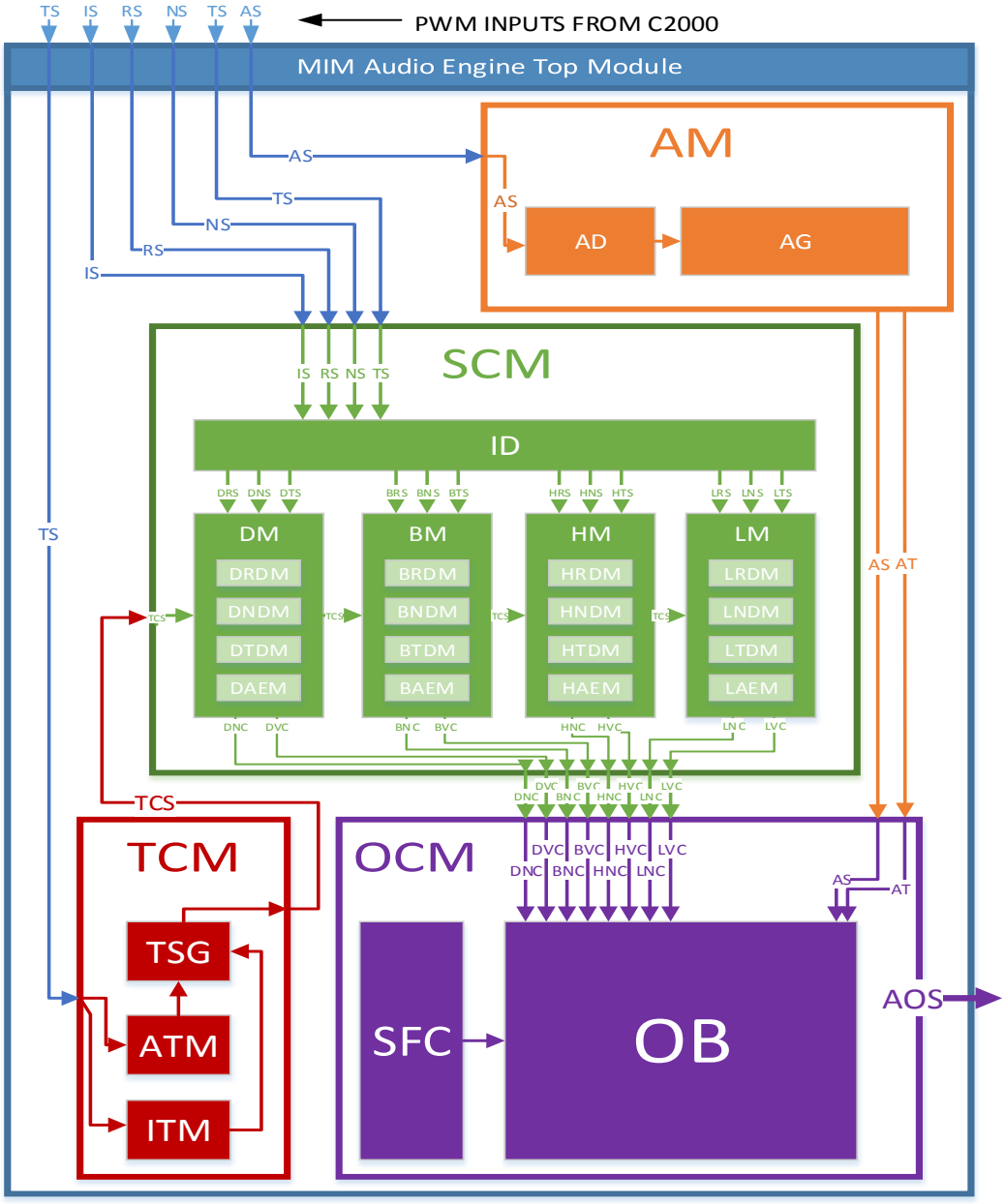


Figure 34 – FPGA Prototype Audio Engine Diagram

The functioning of the Rhythm Block, which exists inside the Oscillator Block and generates the rhythmic instrument signal, is diagrammed in Figure 35. Four oscillators function as the fundamental frequencies for the kick drum, floor tom, snare drum, and cymbal signals and they are enabled by the Alert Signal or the Drum Note Control signal. Outputs of these oscillators are routed to both the velocity control modules and the effect modules. The Rhythm Processor Block sums and compresses the signals and generates the Rhythm Block Out audio signal.

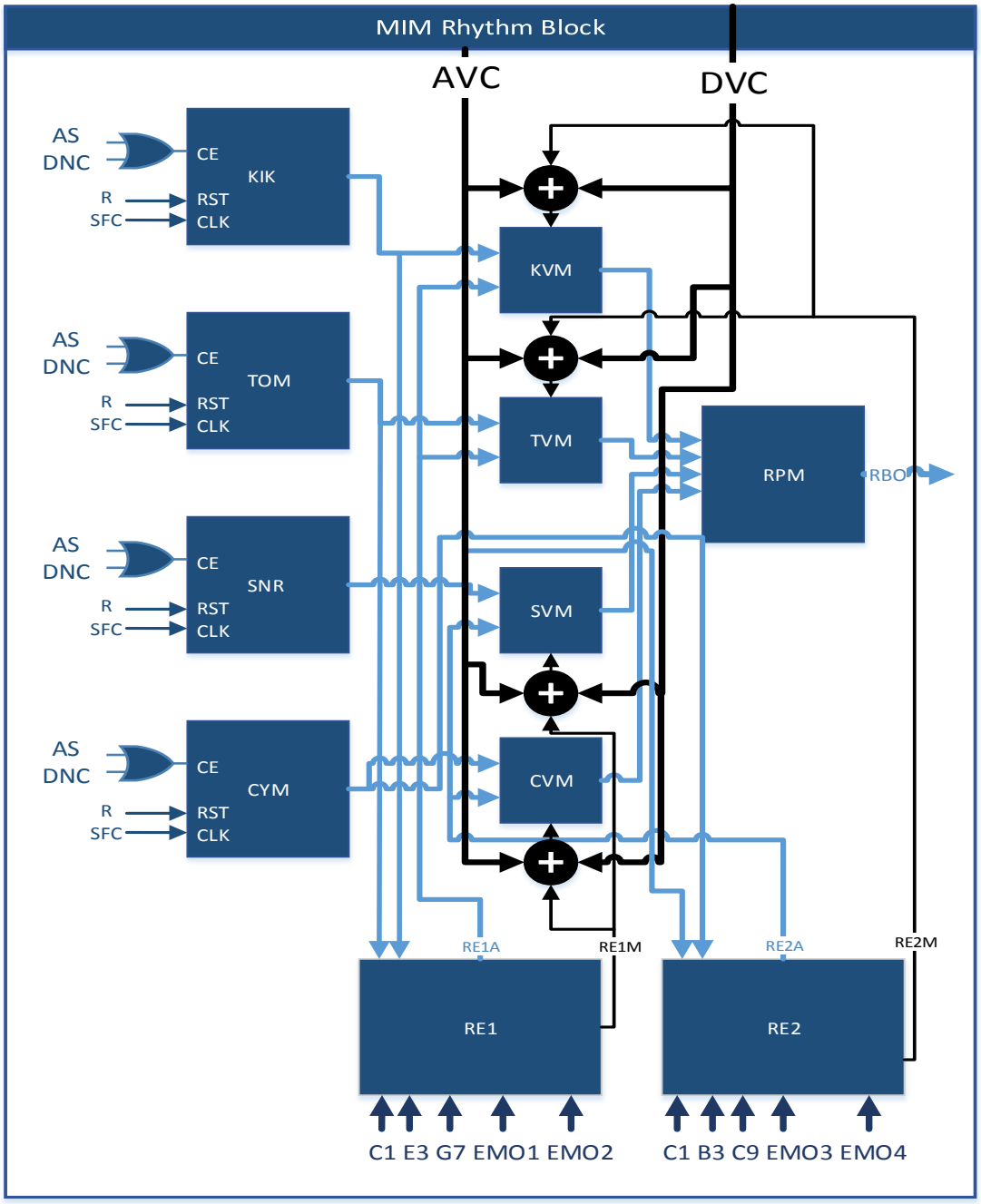


Figure 35 – Prototype Rhythm Block

The functioning of the Harmony Block, which exists inside the Oscillator Block, is illustrated in Figure 36. This block includes the oscillators for the Bass, Harmony, and Lead instruments which generate the melodic elements of the Soundtrack. The Bass and Harmony have some overlap in fundamental frequencies and can utilize the same oscillator block. The signals from these blocks are routed to both the velocity and effect modules to create the appropriate ADSR envelope. Each instrument has its own processor module which creates Bass Block Output, Harmony Block Output, and Lead Block Output audio signals.

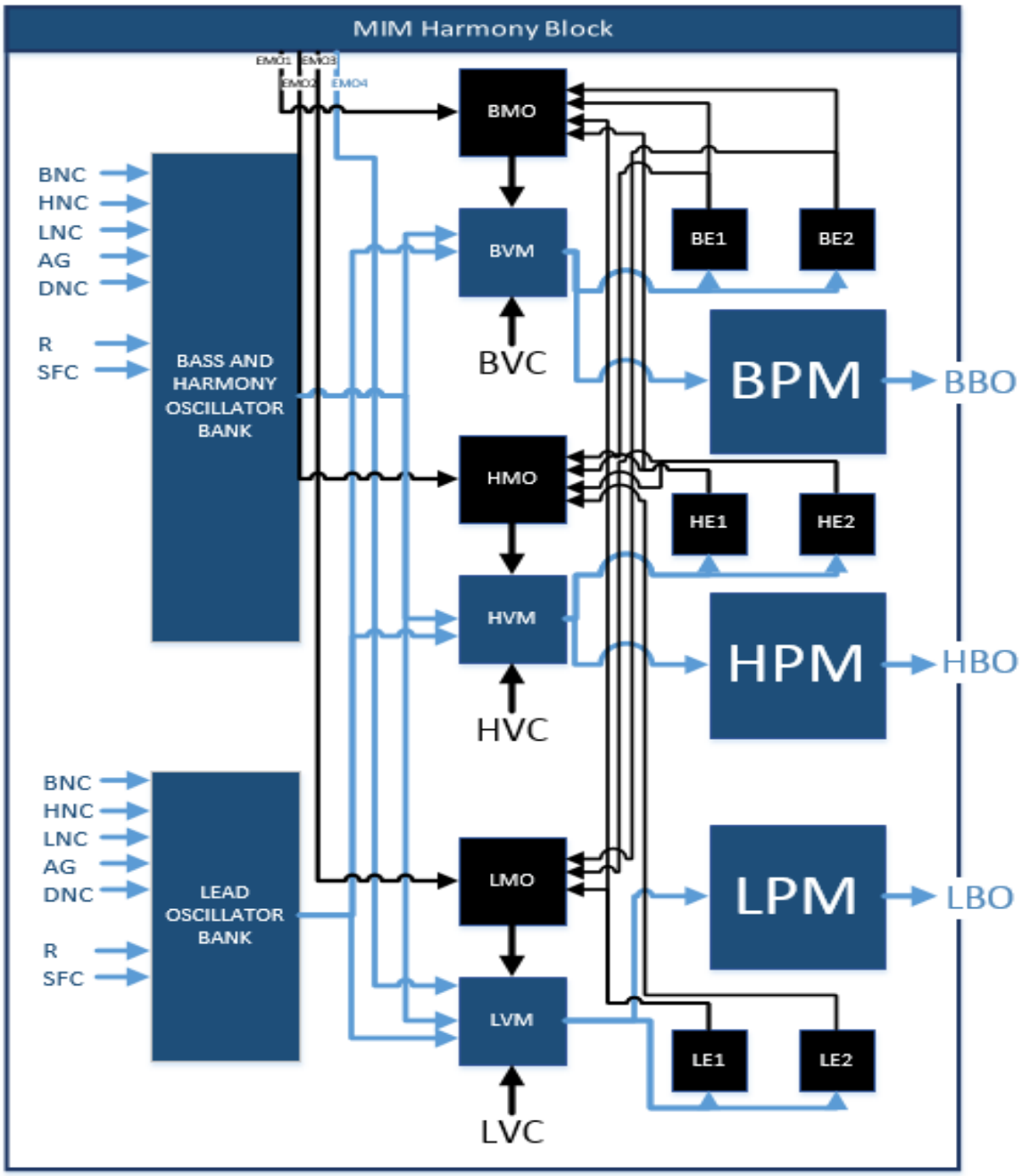


Figure 36 – Prototype Harmony Block

Figure 37 shows the functioning of the Effect Block which generates both small signal modulation outputs as well as audio signal outputs. Four High Frequency Oscillators are routed into four Processor Modules that sums and compresses the signals into a small signal modulation output and an audio signal output.

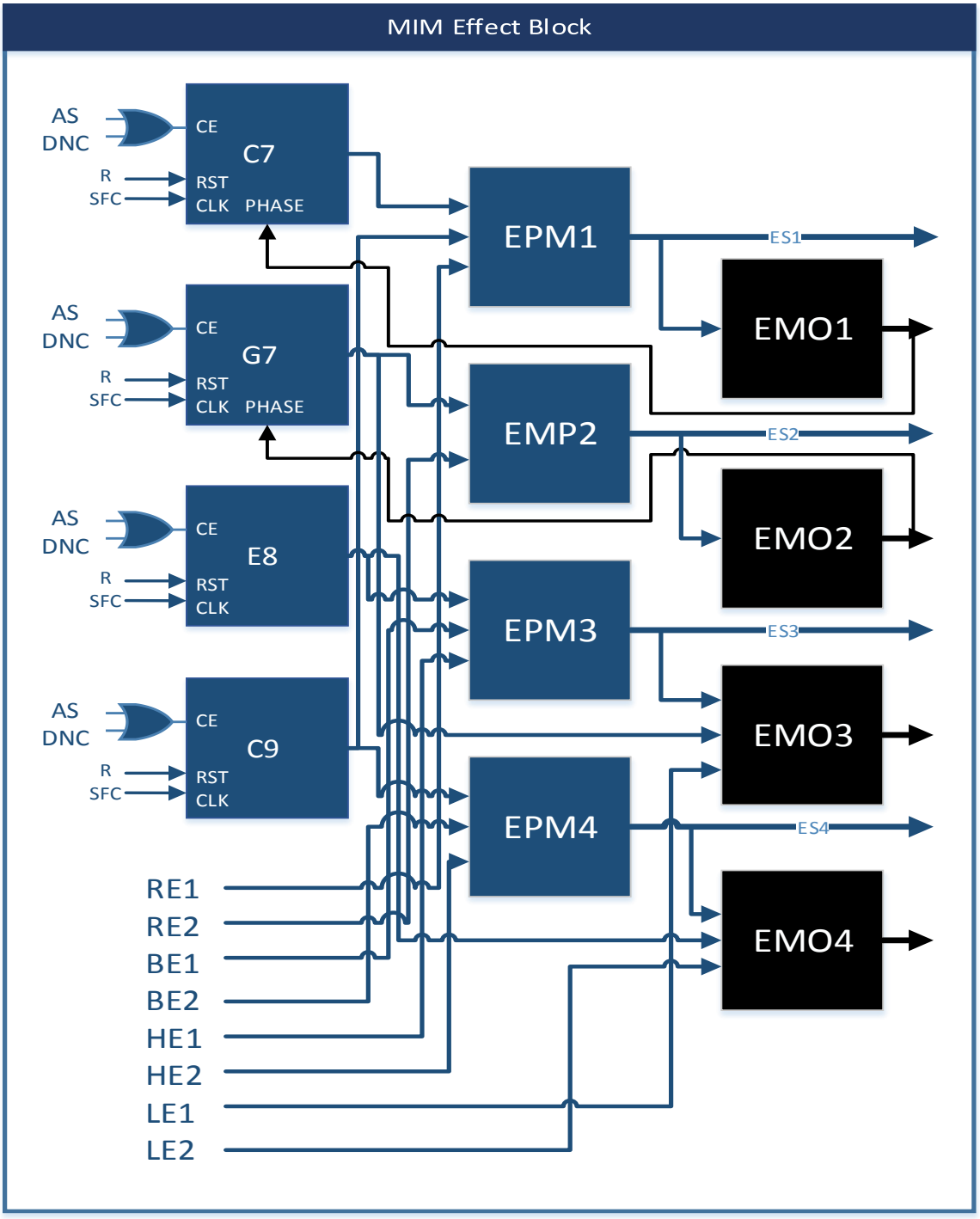


Figure 37 – Prototype Effect Block

The Multiplexed Granular Synthesis method is displayed in Figure 38. The Sample Frequency Clock and the Tempo Control Signal are used to generate a Grain Size Clock signal which ensures that each grain size is an even subdivision of the quarter-note length. Different audio signals are routed into the multiplexor depending upon which module the granular effect is located and the Grain Select Counter cycles through these inputs to create a rotary type granular signal for audio output, ADSR modulation, or frequency modulation.

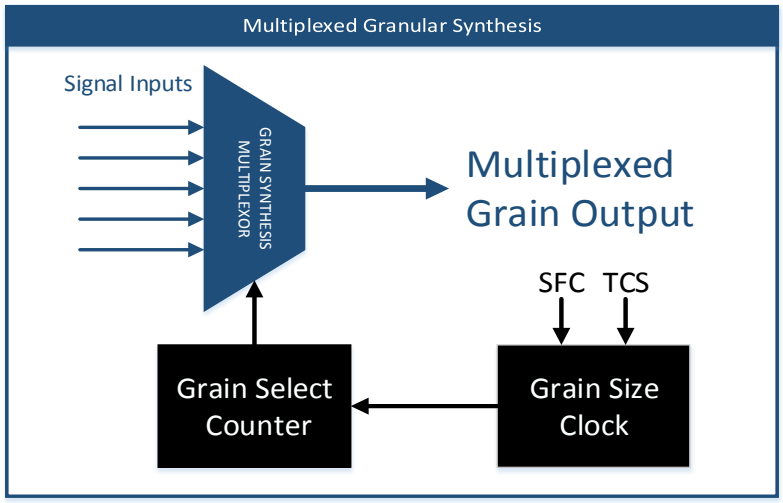


Figure 38 – Prototype Multiplexed Granular Synthesis

Figure 39 demonstrates the Reverse Granular Synthesis method. The input signal is routed to a first in last out buffer and a multiplexor. When the buffer is not full it is set to wright and the multiplexor selects the original input signal. Once the buffer is full it switches to read mode and the multiplexor selects the output of the buffer which reads the signal back in reverse.

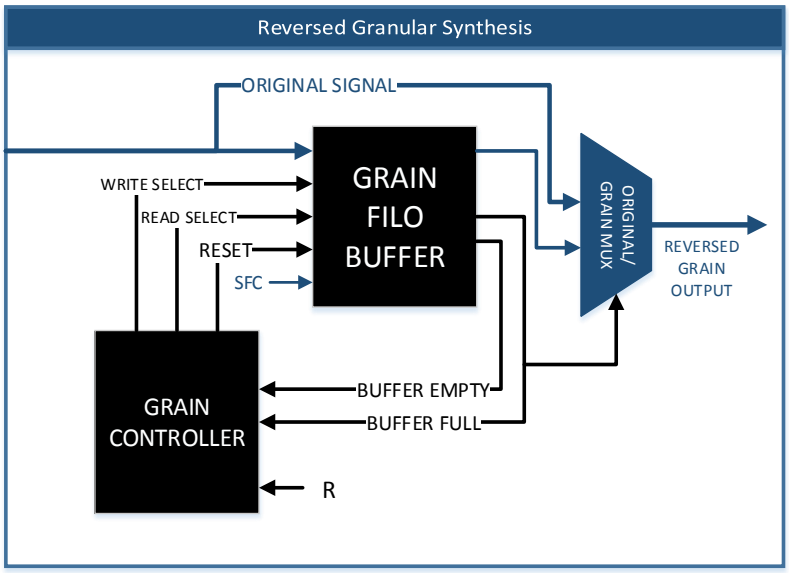


Figure 39 - Prototype Reversed Granular Synthesis

Table 13 defines the acronyms in the Audio Engine block diagram Rhythm Section of MIM.

MIM – Rhythm Section		
DM	Drum Module	Generates non-tonal rhythmic data
DRDM	Drum Rhythm Data Machine	Stores Drum rhythmic information
DNDM	Drum Note Data Machine	Stores Drum type information
DTDM	Drum Tone Data Machine	Stores Drum harmonic information
DAEM	Drum Amplitude Envelope Machine	Stores Drum ADSR information
DNC	Drum Note Control	Enables Drum oscillators
KVM	Kick Velocity Module	Controls Kick amplitude envelope
TVM	Tom Velocity Module	Controls Tom amplitude envelope
SVM	Snare Velocity Module	Controls Snare amplitude envelope
CVM	Cymbal Velocity Module	Controls Cymbal amplitude envelope
RE1A	Rhythm Block Effect Audio 1	Drum audio effect 1
RE1B	Rhythm Block Effect Modulator 1	Drum amplitude envelope modulator 1
RE2A	Rhythm Block Effect Audio 2	Drum audio effect 2
RE2B	Rhythm Block Effect Modulator 2	Drum amplitude envelope modulator 2
RPM	Rhythm Processing Module	Generates Rhythm group PCM audio signal
RBO	Rhythm Block Output	Output of the Rhythm submodule

Table 13 – Prototype Rhythm Section Key

Table 14 defines the acronyms in the Audio Engine block diagram Bass Instrument Section.

MIM – Bass Section		
BM	Bass Module	Generates Bass instrument data
BRDM	Bass Rhythm Data Machine	Stores Bass rhythmic information
BNDM	Bass Note Data Machine	Stores Bass pitch information
BTDM	Bass Tone Data Machine	Stores Bass harmonic information
BAEM	Bass Amplitude Envelope Machine	Stores Bass ADSR information
BNC	Bass Note Control	Enables Bass oscillators
BVC	Bass Velocity Control	Controls Bass amplitude envelope
BE1	Bass Effect One	Bass Multiplexed Grain effect
BE2	Bass Effect Two	Bass Reverse Grain effect
BVC	Bass Velocity Control	Bass ADSR control signal
BMO	Bass Modulation Oscillator	Generates amp. envelope modulation signal
BPM	Bass Processor Module	Sums and compresses the Bass oscillators
BBO	Bass Block Output	16 bit unsigned Bass signal

Table 14 – Prototype Bass Section Key

Table 15 defines the acronyms in the Audio Engine block diagram Harmony Instrument Section.

MIM Diagram Key 3 – Harmony Section		
HM	Harmony Module	Generates Harmony instrument data
HRDM	Harm. Rhythm Data Machine	Stores Harmony rhythmic information
HNDM	Harmony Note Data Machine	Stores Harmony pitch information
HTDM	Harmony Tone Data Machine	Stores Harmony harmonic information
HAEM	Harm. Amp. Envelope Machine	Stores Harmony ADSR information
HNC	Harmony Note Control	Enables Harmony oscillators
HVM	Harmony Velocity Module	Controls Harmony amplitude envelope
HMO	Harmony Modulation Oscillator	Generates amplitude envelope modulation signal
HE1	Harmony Effect 1	Harmony Multiplexed Grain effect
HE2	Harmony Effect 2	Harmony Reverse Grain effect
HVC	Harmony Velocity Control	Harmony ADSR control signal
HPM	Harmony Processor Module	Sums and compresses the Harmony oscillators
HBO	Harmony Block Output	16 bit unsigned Harmony signal

Table 15 – Prototype Harmony Section Key

Table 16 defines the acronyms in the Audio Engine block diagram Lead Instrument Section.

MIM Diagram Key 4 – Lead Section		
LM	Lead Module	Generates Lead instrument data
LRDM	Lead Rhythm Data Machine	Stores Lead rhythmic information
LNDM	Lead Note Data Machine	Stores Lead pitch information
LTDM	Lead Tone Data Machine	Stores Lead harmonic information
LAEM	Lead Amplitude Envelope Machine	Stores Lead ADSR information
LNC	Lead Note Control	Enables Lead oscillators
LVC	Lead Velocity Module	Controls Lead amplitude envelope
LMO	Lead Modulation Oscillator	Generates amp. envelope modulation signal
LE1	Lead Effect 1	Lead Multiplexed Grain effect
LE2	Lead Effect 2	Lead Reverse Grain effect
LVC	Lead Velocity Control	Lead ADSR control signal
LPM	Lead Processor Module	Sums and compresses the Lead oscillators
LBO	Lead Block Output	16 bit unsigned Lead signal

Table 16 – Prototype Lead Section Key

Table 17 defines the acronyms in the Audio Engine block diagram Effects Section in MIM.

MIM Diagram Key 5 – Effects Section		
EPM1	Effect Processor Module 1	Additive synthesis effect processor 1
EPM2	Effect Processor Module 2	Additive synthesis effect processor 2
EPM3	Effect Processor Module 2	Multiplexed granular effect processor
EPM4	Effect Processor Module 4	Reverse granular effect processor
EMO1	Effect Modulation Output 1	Additive synthesis modulation signal 1
EMO2	Effect Modulation Output 2	Additive synthesis modulation signal 2
EMO3	Effect Modulation Output 3	Multiplexed granular modulation signal
EMO4	Effect Modulation Output 4	Reverse granular modulation signal
ES1	Effect Signal 1	Additive synthesis audio effect 1
ES2	Effect Signal 2	Additive synthesis audio effect 1
ES3	Effect Signal 3	Multiplexed granular audio effect
ES4	Effect Signal 4	Reverse granular audio effect

Table 17 – Prototype Effects Section Key

Table 18 defines the acronyms in the Audio Engine block diagram Control Section in MIM.

MIM Diagram Key 4 – Control Section		
R	Refresh	Resets the oscillator block accumulators
TCM	Tempo Control Module	Generates quarter note lengths
ATM	Average Tempo Module	Calculates average tempo of last 12 footfalls
ITM	Instantaneous Tempo Module	Predicts time until next footfall
TSG	Tempo Signal Generator	Creates Tempo signal map
TCS	Tempo Control Signal	Determines the length of a quarter-note
OCM	Output Control Module	Generates 16 bit PCM audio output signal

Table 18 – Prototype Control Section Key

4.3.4 – Sensor Control System Diagram

Figure 40 shows the loop cycle of the code implemented in the sensor control system. The initialization starts when the device is powered on. The only thing the user must do before starting to run is lay the device flat in an outside area in order to detect a fix for the GPS system. If the user does not want GPS notifications, the user can automatically begin running. The accelerometer is the main sensor that constantly detects acceleration in the x, y, and z axes. If any of the axes passes over a certain absolute value threshold AND passes both the latency and time threshold requirements, then the data is interpreted as a footstep. No footstep 0.2 seconds after can be detected. During that latency period, the GPS sends data to the microcontroller for parsing and distance calculations. If the

current distance hits the next 0.1 mile mark, then the user hears a notification. After the GPS is done, the loop repeats using the new accelerometer data for footfall detection.

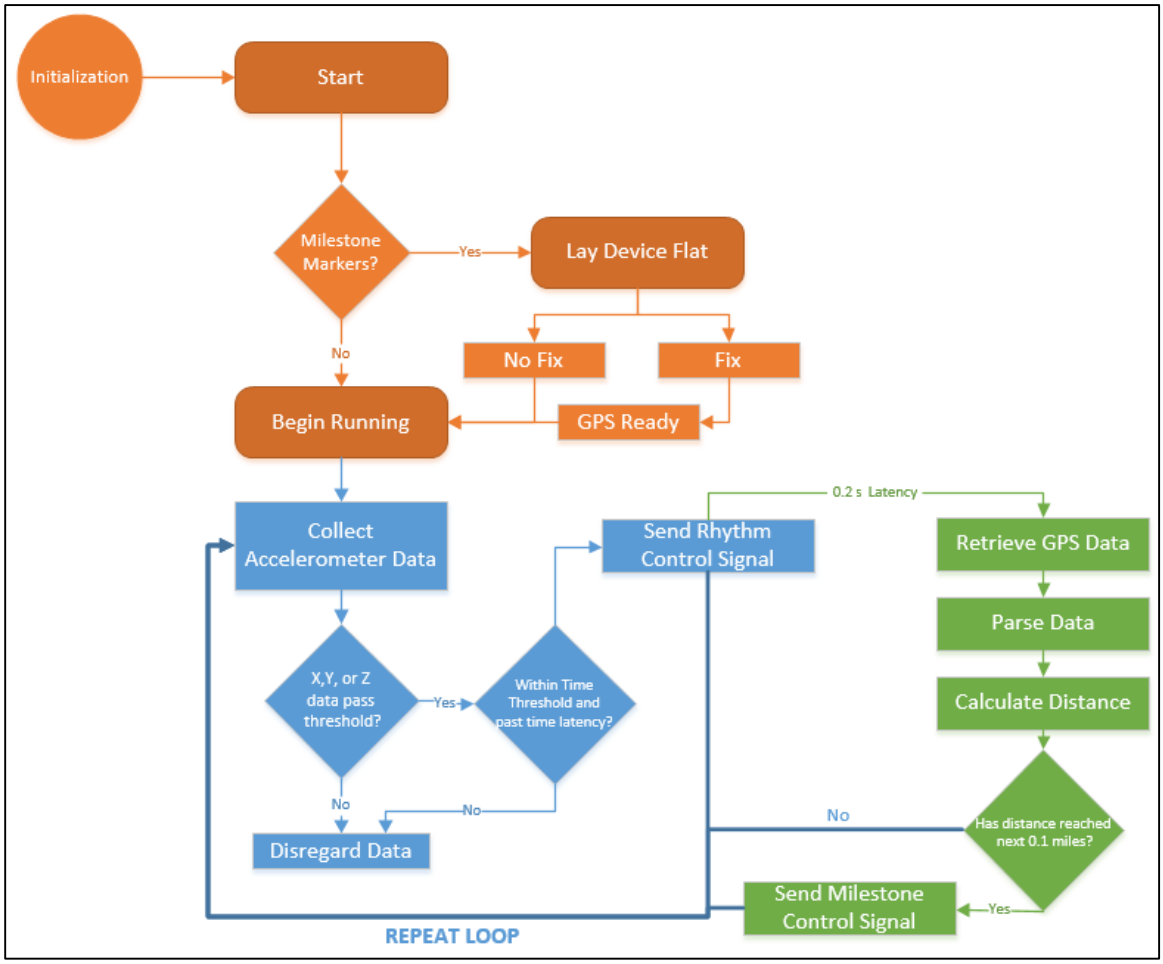


Figure 40 – Sensor Control Activity Diagram

4.4 – Power

This section explores the required voltages and currents for each sub-system and then investigate exactly what power system is required in order to handle these given loads. The device is generally powered by a standard nine volt battery. This input voltage is sent to three voltage regulators and is stepped down to 5 and 3.3 volts to power each of the digital integrated circuits that are utilized by the MIM device.

A 7805 voltage regulator steps the nine volt input down to five for use by the two Atmega328 microprocessors that are used to process the sensor input data and control that audio engine output. This regulator design is shown in the following Figure 41.

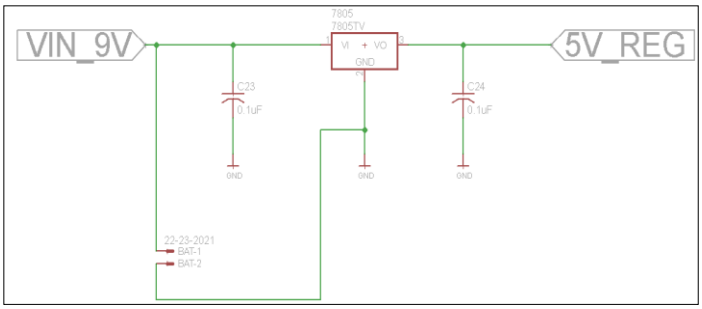


Figure 41 - 9V to 5V Regulator

An LP2985 regulator steps the input voltage down to 3.3 volts. This is used as the input voltage for the GPS and accelerometer circuits. This regulator is shown in the following Figure 42.

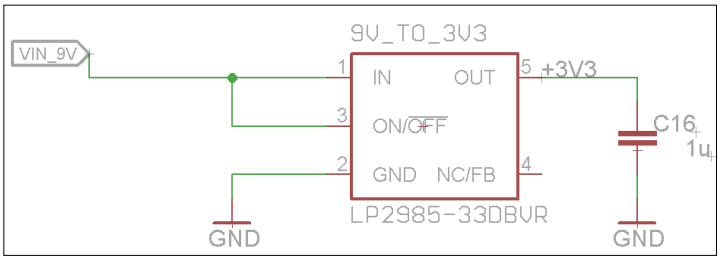


Figure 42 - 9V to 3V3 Regulator

The third voltage regulator is used to step a five volt signal from the audio microcontroller down to 3.3 volts. This is done because the audio synthesizer chip requires a specific power up sequence and needs 3.3 volts at input pin three before the start up voltage is received. The voltage at pin three is received from the LP2985 so that as soon as the MIM device is powered on the required voltage is at pin three. After the audio microcontroller is powered on it will run an initialization function that then powers the audio synth chip in order to set the synth to real time MIDI operation. The design for this regulator, an AP7312, is shown in the following Figure 43.

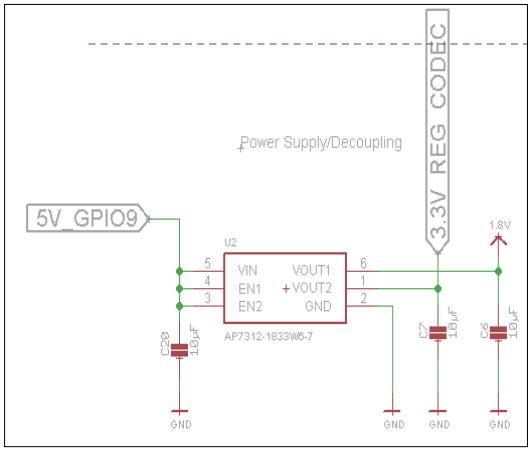


Figure 43 - Audio Initialization Regulator

5 – Prototype Construction and Coding

To prototype the MIM Audio Running System a series of, scaled down, modular systems have been designed so that integral sub systems could be quickly built, tested, and evaluated. This allows for several design systems to be evaluated and compared simultaneously and independently, allowing for a better system optimization. These basic prototype systems are the Sensor Array, microprocessor, and Audio Engine.

The prototyping was be hierarchical for this project; this ensured that every base is covered so that the problems that arose in the beginning of the prototyping were easily identified and fixed by the end of the final design. In relation to this project, hierarchical refers to component, subsystem, system level testing, and with each of this divided between software and hardware. Ultimately the entire system was integrated one subsystem at a time and re-tested each time, until the device worked properly.

5.1 – Data Input Subsystem

This section concerns the input data from the user of the Music in Motion device. This translates to the accelerometer, signal processing, and physical connections that make this whole subsystem.

5.1.1 – Accelerometer

The ADXL345 accelerometer was be prototyped using a breakout board that was bought from Adafruit. For the sake of saving much needed time, buying the breakout board was the easiest and most efficient way to begin prototyping the accelerometer. Additionally, the breakout board was be used as a reference design for the PCB that integrates the entire hardware system. This board needed header pins soldered onto it and then it was placed onto a breadboard for prototyping.

Initially, the main objective is to get the accelerometer to interface with the microcontroller, which was done using 4-pin SPI. Essentially, the main difficulty with prototyping was getting all of the pieces to the puzzle communicating with one another; therefore this was the primary focus in the beginning.

Once this communication protocol is fully operational, the accelerometer functionality was be prototyped. This phase is mostly about figuring out what typical pedometer type data looks like. Once the patterns are recognized, the data was manipulated and the sensitivity of the system can be adjusted. The challenge here was to figure out where the main signal is coming from, because the ADXL345 utilizes an X, Y, and Z axis. This corresponds with many degrees of freedom and

also a good portion of unnecessary data that needs to be thrown away. Many prototype tests were be conducted in order to determine which axis holds the desired signal and how the runner's bodily movements can introduce noise. For example, if the device was strapped to the runners arm, this could introduce unwanted acceleration data. This is due to the swinging motion of the typical runner's arm, which could affect the axis that is used for the control system. Because of these types of challenges, many of the high level design solutions of the control system were tweaked during this phase. For example, perhaps it is best to strap the device to the user's waist because that minimizes multi-axis acceleration. If you look at a runner, the waist typically only accelerates up and down, while the velocity is forward (and in some cases with minimal acceleration).

5.1.2 – Digital Filtering

Once the accelerometer was communicating with the microcontroller and the patterns were recognized such that the desired axis is identified, the filter prototyping began. The digital filter is the solution to processing this signal in order to achieve the desired quality for control system use. The signal itself did not need too much processing because foot-step impulses have very good sound noise ratios from what the research shows. However, in order to achieve the greatest accuracy, the signal was processed through the microcontroller. The testing phase showed that a simple finite impulse response filter, which measures the average time between the last four pulses, working in conjunction with a digital low pass filter is sufficient for isolating footfalls from the accelerometer data.

The digital filter in this system was used in prototyping to minimize noise in the accelerometer data. Due to the capabilities of the microcontroller as a real-time and low-power DSP, a digital filter was used instead of an analog filter.

5.2 – Digital to Analog Converter

This section describes the prototyping for the digital to analog converter that works in conjunction with the FGPA audio engine prototype. The prototyping for the DAC consisted of determining whether additive synthesis on an FPGA and a sixteen bit 2R ladder type converter would be sufficient or if a predesigned synthesizer and converter is preferable. The DAC works fairly well, although there are some errors present. It was simulated through Multisim while the prototyping phase was still in its early stages. Early prototyping indicated that the FPGA with custom converter design was insufficient for this project. A very solid monophonic synthesizer was designed but the scope of resources required to design a full music player was excessive for the resources and time available for this project.

The audio signal processing begins with the digital to analog converter. This DAC is custom designed therefore its performance is not on par with a manufactured chip. The type of DAC chosen as a reference design is the R-2R ladder type. This

parallel input circuit serves to begin the audio processing. In order to design and prototype this circuit simultaneously without any extra cost, Multisim was used. Figure 44 below is the design for the digital to analog converter for the audio output of the system.

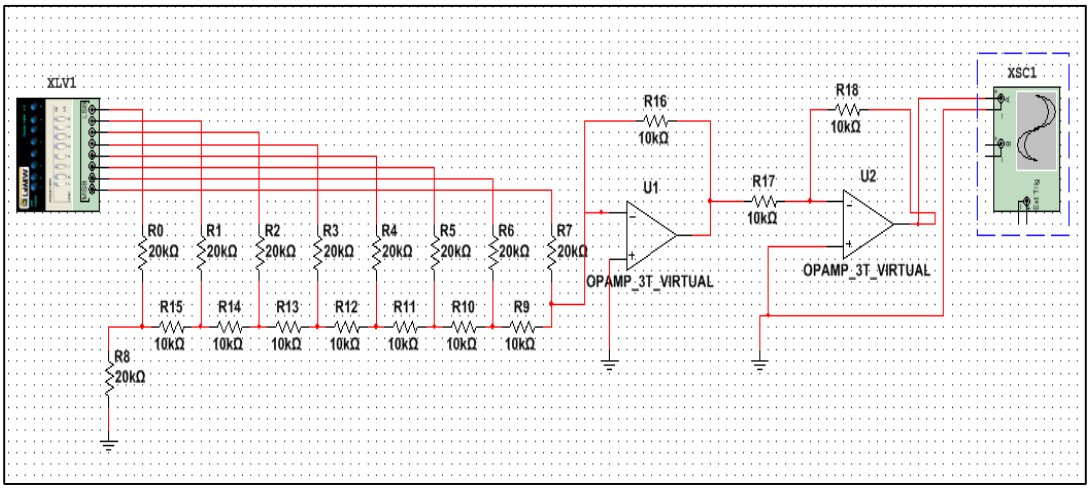


Figure 44 – Digital to Analog Converter

The figure above is the initial design for the DAC and it contains three phases. The first phase is the R-2R ladder which converts an 8 bit number into a voltage signal using voltage division. The second phase, which is the first operational amplifier, is used for gain. In this figure, it was set to an arbitrary value of 10kΩ. The third part of this design is for a secondary gain stage and primarily to invert the output once more in order to output a positive voltage signal.

5.2.1 – Software Prototyping

This subsystem was prototyped in Multisim for its ease, convenience, and speed. Once the design was completely error free, it was designed using vector board. The design performed decently, however there were some unexpected results. It is as though the steady state result is perfect, however in the beginning of the simulation, the output is changing in frequency. It may be that there is performance degradation in the beginning of the simulation. Moreover, this degradation is observed is not input dependent as it present throughout multiple types of digital input signals.

Table 19 below represents the inputs, which are 8-bit digital signals. These binary numbers are inputted, in parallel, to the digital to analog converter. The outputs of each signal were recorded and are displayed below.

RAMP				WALKING ONES				ALTERNATING ONES			
A3	A2	A1	A0	B3	B2	B1	B0	C3	C2	C1	C0
0	0	0	0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	0	0	1	0	1
0	0	1	1	1	0	0	0	1	0	1	0
0	1	0	0	0	0	0	1	0	1	0	1
0	1	0	1	0	0	1	0	1	0	1	0
0	1	1	0	0	1	0	0	0	1	0	1
0	1	1	1	1	0	0	0	1	0	1	0
1	0	0	0	0	0	0	1	0	1	0	1
1	0	0	1	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	0	0	1	0	1
1	0	1	1	1	0	0	0	1	0	1	0
1	1	0	0	0	0	0	1	0	1	0	1
1	1	0	1	0	0	1	0	1	0	1	0
1	1	1	0	0	1	0	0	0	1	0	1
1	1	1	1	1	0	0	0	1	0	1	0

Table 19 – 4-Bit Example of Digital Input to DAC

For the sake of conserving space, the figure above shows a 4 bit example of the actual 8 bit input signal. Each bit represents an input branch into the digital to analog converter, which has a parallel input. The 8 bit ramp signal simply counts from 0 to 255. The walking ones signal works similar to one hot encoding, whereby only one digit at a time is high and all others are low. The 'hot digit' cycles, or shifts left until it reaches the most significant bit and then restarts back to the leas significant bit. The alternating ones signal simply alternates between high and low in a checkered pattern on the truth table. The following three figures are the responses, or outputs, of the digital to analog converter with respect to these inputs. Figures 45 shows an analog output with respect to its digital inputs.

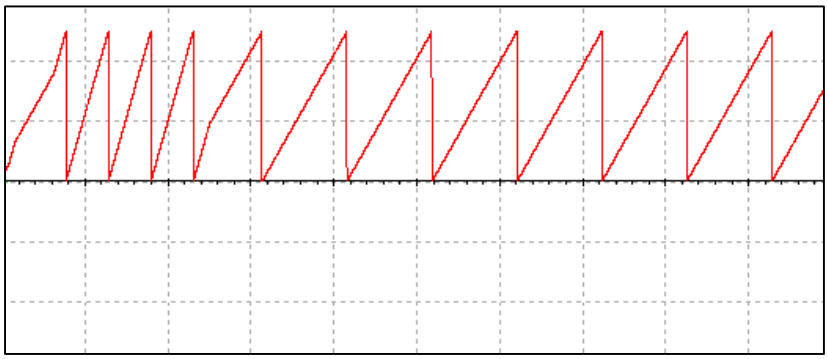


Figure 45 – Output of DAC with Ramp Input

It appears as though the DAC works, functionally, but only completely correctly after an initial period of time. This beginning period of error prone response is observed in all three of the figures. After which, the output is exactly correct. It is possible that the actual response desired is the steady state response.

5.2.2 – Hardware Prototyping

The digital audio converter was then implemented with a series of resistors and opamps and soldered onto vector board. This prototype took a sixteen bit PCM input and output a mono analog audio signal. Testing indicated that the conversion is accurate, however very low frequency signals contain some digital noise in the output signal. The following Figure 46 shows the DAC hardware design with the analog waveform output.

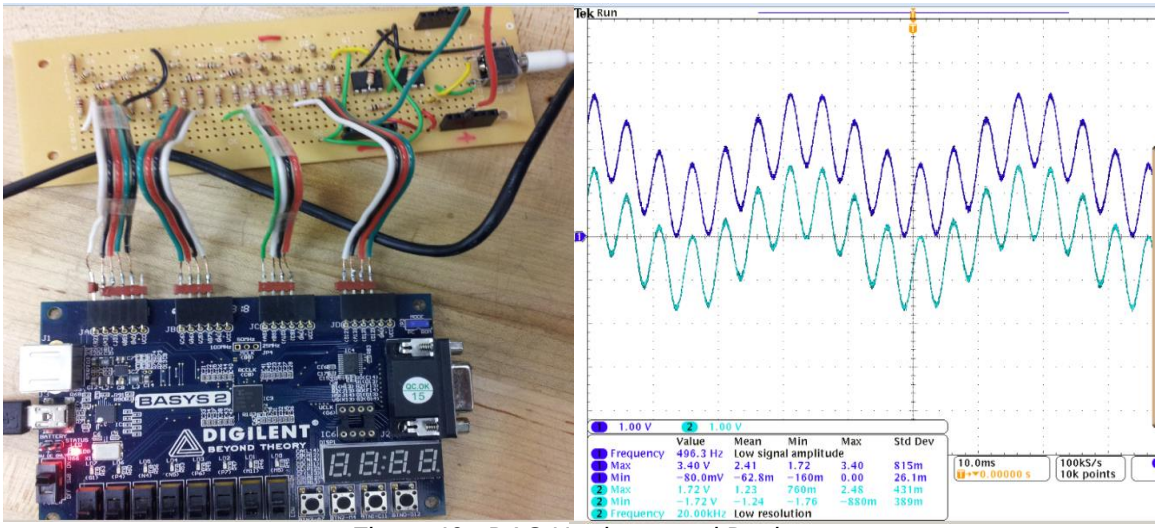


Figure 46 - DAC Hardware and Design

5.3 – Audio Engine Prototype Construction

The MIM FPGA prototype design consists of six primary sub-modules. These are the Audio Engine, Tempo Control Module, Soundtrack Control Module, Sequencing Module, Alert Module, and the Output Control Module. Initially a prototype design was implemented to test each module independently. After the first stage of testing a secondary prototype was utilized to specifically test for optimum connectivity both within the Audio Engine and to the MIM system as a whole.

5.3.1 – Tempo Module

To test the Tempo Control Module a simple PWM audio synthesizer was developed on a Basys2-250 FPGA development board. This includes a four step sequencer control system, which gives the control system a sixteenth-note quantization. An MSP430 Launchpad is utilized to generate a PWM signal at 2 Hz and used to trigger each four step sequence.

The MIM system begins to generate audio when signal is detected in the PWM tempo control input. Tempo data from the accelerometer is converted to a PWM control signal and passes from the Microprocessor to the FPGA over a single wire. This tempo control scheme utilizes two symbols. At each footfall a pulse, the Tempo Long Pulse (TLP), is generated. The second symbol, Tempo Short Pulse (TSP), is generated when the runner is at the top of his stride. Assuming that the Runner has an even gate and the accelerometer is receiving typical flat surface running data, the TSP is generated midway between footfalls.

A predictive tempo control system is utilized by the MIM system and using two symbols allows for the TCM to anticipate the next footfall is two separate ways. The TCM stores the average time between TLPs over the last five strides in a register named AveTempo. Then stores the time between the most recent TLP and TSP in a register named InstTempo. When determining the placement of the next quarter-note the TCM averages the values in these two registers and begins counting down to the next quarter-note.

Upon receiving the first TSP signal, indicating that the Runner is midway through his first stride, the TCM counter measures the time until the first TLP. In this situation the AveTempo is empty so the TCM generates the next foot fall at twice the time between the first TSP and TLP. Once the second TLP has been received a value is stored in the AveTempo register. At this point the TCM begins use the average of the two registers to determine the placement to the next quarter-note.

The output Tempo Control Signal (TCS) is a twenty eight bit value that indicates the length of each sixteenth-note and is generated by using the average of these two tempo mapping schemes. This allows for speeds as low as 0.37Hz, or in musical terms 22bpm. In this way the MIM Tempo Control Module generates truly useful predictive tempo data in a variety of running environments. The instantaneous tempo detection system utilizing both the TSP and TLP symbols allow for quick reaction to acceleration and deceleration in flat street running type environments while the average tempo detection system allows for smooth tempo mapping even in extreme free running type environments and can accommodate speeds ranging from the most casual walks to a high speed race.

This control signal has two symbols. A long pulse represents a quarter-note while a short pulse represents an eighth note. The MPS430 alternates these short and long pulses. A long pulse represents a quarter-note downbeat that occurs at each

of the runner's footfalls and simulates the symbol that is generated by the C2000 microprocessor when the accelerometer input Y-axis velocity data derivative moves from a negative value to zero. A short pulse represents an eighth-note upbeat value. These pulses simulate the symbol that is generated by the sensor microprocessor when the accelerometer input Y-axis velocity data derivative moves from a positive value to zero. At this instant the runner should be at the top of their stride and mid-way to the next footfall. In musical terms this gives the sequencer a sixteenth-note quantization and initial tempo of 60 bpm. Stepping down the master 100 MHz clock to a 2 Hz tempo clock means that there are 50 million master clock cycles for each quarter-note and 12.5 million master clock cycles for each sixteenth-note. The MSP430 then randomly varies the PWM control signal from 0.5 Hz to 4 Hz.

The module monitors this input from the MSP430 and predicts the position of the next footfall in two ways. An average tempo module keeps track of the tempo of the input over the past several pulses. When a short pulse is received from the MSP430 the instantaneous tempo module counts the master clock cycles between the long pulse and short pulse and uses this data to approximate the position of the next foot fall. The Tempo Control Module takes the average of these two predictions and adjusts the Tempo Clock counters in real time, either increasing or decreasing its sixteenth-note subdivision lengths, and thereby keeping the clocks synchronized.

The primary objective of this prototype design is to determine if this tempo control design results in a smooth and musically sensible tempo map. Expecting that the majority of the operation time is spent with a fairly steady input tempo does not require for the controller to have a particularly fast reaction time. However, if the system is not responsive enough the time it takes to match the runners tempo after the initial acceleration may be too slow and cause the uses to become frustrated with the system.

5.3.2 – Soundtrack Module

The musical content for the MIM system is generated by the Soundtrack Control Module. Five primary sub-modules control the musical composition functionality. Initially, the Instrument Decoder routes the Rhythm Select (RS), Note Select (NS), and Tone Select (TS) control signals to the appropriate instrument module. These instruments are Drums, Bass, Harmony, and Lead. Each signal utilizes two symbols to make adjustments to each instrument's settings, an increment and decrement symbol. Inside the instrument modules, a set of four finite state machines control the output signals from each Instrument Module. These are the Rhythm Data Machine (RDM), Amplitude Envelope Machine (AEM), Note Data Machine (NDM), and Tone Data Machine (TDM).

The RDM uses two separate finite state machine to create the rhythmic patterns for each instrument. First, the Tempo Control Signal is used to increment a counter

that steps through a sixteen state machine that represents one measure of rhythmic information, with each state representing a sixteenth-note note value and pointing to a sixteen bit velocity value that determines the amplitude value of the note played at that rhythmic position. Next, the RS input uses a short pulse (RDEC) to decrement the pattern state and a long pulse (RINC) to increment the pattern state. Each of these patterns is a different rhythmic phrase and as the pattern state changes the rhythmic output of each instrument changes. Drum and bass module rhythmic patterns can only be changed on measure boundaries, while all other patterns can be changes on quarter-note boundaries.

Fundamental note frequencies for each instrument are stored in the NDM. This module functions in a similar manner to the RDM, in that the TCS is used to step through a state machine that stores note values for one measure. Each sixteenth-note subdivision holds a twenty four bit value that is decoded by the OCM and each bit is routed to the CE input for a bank of oscillators. This allows for the tonal instrument modules to have a three octave range and for the non-tonal rhythmic instrument to create a bank of five complex audio waveforms. Storing rhythmic and note data separately allows for the creation of more complex patterns. If the note patterns value changed while the rhythmic pattern remains the same it creates a sense of change and musical development. The note pattern value for the Drum module can be changed on measure boundaries, while all others can be changed on quarter-note boundaries.

Timbral information is generated by the TDM. By adding overtone content the tonal characteristics of each instrument module each signal begins to take on unique and distinctive characteristics. In this way individual instruments distinguish themselves sonically. To generate these tone maps, a series of recordings of acoustic instruments were analyzed for their spectral content. Using this data overtone maps were generated that allow the Music In Motion Audio Engine to model a variety of acoustic sounds. Again, this module sequences this data into quarter-note steps and steps through this information based on the TCS input. Tone patterns can be changed at any time and are not limited by measure or quarter-note boundaries.

Note velocities are generated by the AEM. Velocity values from the RDM are fed into the AEM and a standard Attack Decay Sustain Release (ADSR) amplitude envelope is generated. Each of the Instrument Modules requires a different ADSR envelope. The Drum module uses envelopes with short attack and release times with a small decay to create impulsive rhythmic sounds. Since the Bass module is integral to both rhythmic and melodic content an ADSR envelope is used that is similar the one utilized by the Drum module. A short attack allows the patterns to accentuate rhythm patters. However, a longer decay, sustain, and release allow for the Bass to add tonal content as well. By allowing the ADSR envelops to be customizable for the Harmony and Lead modules, a variety of instrument types can me modeled. Long attack, decay, and sustain settings can be used to simulate woodwinds while short attack and sustain paired with long decay can model brass

instruments. The NDM and TDM output signals are routed through the AEM as well and the ADSR envelope is applied to the Velocity Multipliers of the active oscillator outputs to create the amplitude envelopes for each note value.

The MIM Audio Running System FPGA prototype is controlled by the Soundtrack Control Module. A PWM control signal has been devised that is decoded to select between two different instruments, one rhythmic and one melodic, that each has four distinct patterns. The PWM audio synthesizer is simple and can generate tones that are not necessarily musical but can be quickly and easily developed. This is ideal for testing the control system where audio fidelity is of no consequence and allows focus upon the control system and a great amount of flexibility during the testing phase.

Two pulses are used for the selection. A short pulse is used for the instrument select while a long pulse is used as a pattern select. These pulses are further subdivided such that there are two short pulses, SP1 and SP2, and four long pulses, LP1 through LP4. Short pulses are used to update the Instrument Select Register (ISR) while long pulses are used to update the Pattern Select Register (PSR).

A value of zero in the ISR indicates that the Rhythmic Instrument pattern is being selected and a value of one indicates that the Melodic Instrument pattern is being selected. Initially, a default value of zero is set in this register and is set to one when a SP2 signal is received. The ISR is reset to zero when a SP1 signal is received.

Long pulses LP1 through LP4 are used to update the values of the Rhythmic Instrument PSR, when the ISR is zero, and the Melodic Instrument PSR, when the ISR is one. Each pattern contains the note on and pitch data of four sixteenth-notes. The PSR can be updated on each sixteenth-note boundary. This allows for more complex patterns to be created from the four basic patterns stored in memory.

The PSR is a sixteen bit register where each four bit chunk represents the note data for each sixteenth-note. For example, PSR [15:12] holds the data for the first sixteenth-note of a quarter-note section. PSR [15] determines the note on value, a one in this position indicates that the instrument plays a note and a zero results in no audio being generated. Frequency information is contained in the next chunk, PSR [14:12]. This value allows for the selection of eight separate tones to be generated during that quarter-note section. Each instrument handles quarter-note transitions in a different manner. The Rhythmic Instrument forces a note off value for the last thirty-second note of each quarter-note to enforce the rhythmic element of the instrument regardless of the control signal input. Conversely, the Melodic Instrument continuously generates audio data given that the note on value is continually one.

5.3.3 – Sequencing Module

Audio output generation is controlled by the Sequencing Module. It receives the tempo signal from the Tempo Control Module and the ISR PSR data from the Soundtrack Control Module and generates two channels of PWM audio. A finite state machine is utilized to decode the data from the PSR and select which pattern to generate by accessing the Pattern Storage Registers (PTR). These values are used to enable the counter that generates the PWM audio output as well as determine its frequency.

The Sequencing Module also utilizes a Pattern Update Register (PUR) which is used to generate new patterns for the PTR so as to generate non-repetitive music. The PUR is randomly updated by storing data coming in from the ISR and PSR. An exclusive or operation is executed on certain patterns stored in the PTR with the PUR. This allows patterns to establish, become musically sensible, and then be manipulated so that the audio experience does not become boring and repetitive.

5.3.4 – Audio Engine

The Music in Motion Audio Engine prototype is designed around the Xilinx Spartan3E-XC3S250E FPGA. Six PWM control signal inputs are routed through a series of decoders and state machines that manipulate and sum the output of twenty nine oscillators to generate a sixteen bit unsigned PCM audio signal output at 48.8 kHz. This sampling rate is a natural result of stepping down the 100MHz internal clock inherent in the Spartan3E by 2048 and is sufficient for reproducing audio frequencies up to 24.4 kHz. This is above the hearing range of individuals who may happen to have exceptional hearing and is greater than that of CD quality formats.

Nearly sixty percent of the chip resources are devoted to implementing the Xilinx IP core modules that function as the oscillators which generate the sinusoidal signals that are used in an additive synthesis method to create the output signal. By using the outputs of twenty one generators to create the frequencies of the seven notes in the key of C in octaves six, four, and two, and simultaneously stepping each output by half, the system achieves a six octave range in terms of generating fundamental frequencies. This is slightly less than a grand piano and is approximately equal to the combined range of a modern bass and guitar. Four oscillators are used to generate signals that are used specifically to generate non-tonal rhythmic sounds. Four high frequency oscillators are used to implement overtone content and special effects for the alert tones. Each of the twenty five oscillators, used for fundamental frequencies and rhythmic sounds, are stepped down to simultaneously generate a signal one octave lower than the original. The four high frequency oscillators are each stepped down by thirds to generate a total of twelve signals. Two custom signals are generated by summing other oscillator

outputs to create complex waveforms. This allows for sixty four discrete waveforms that can be amplitude adjusted and summed together to output complex audio signals from 32.5 Hz to 20 kHz.

The twelve finite state machines that store the oscillator control information utilize approximately thirty percent of the chip resources. Each instrument module has four of these state machines. They contain the data necessary to generate the rhythmic, amplitude envelope, note, and tone data required to craft the raw oscillator signals into a musical format. Each state machine works as a sequencer that subdivides each quarter-note and controls the oscillator output signals to create the Soundtrack audio output.

5.3.5 – Alert Module

There are two types of alerts generated by the prototype Alert Module, a positive alert and a negative alert. When certain predefined achievements are met by the runner the microprocessor generates a long pulse, the Alert Positive Signal (APS) and transmits this symbol to the AS input. If these achievements have not been met an Alert Negative Signal (ANS) is transmitted.

The ACM functions in a similar manner to the Instrument Modules that exist within the SCM, excepting that only two signals need be generated. These two output signals, the Alert Note Control (ANC) and the Alert Velocity Control (AVC), determine the type of signal that is generated. These Alerts do not interrupt Soundtrack playback and are designed to be noticeable by the user while not being a distraction. When a signal is received by the ACM a simple decoder determines if the symbol is an APS or ANS. A simple two state machine, the Alert State Machine (ASM), generates the ANC and AVC required to inform the runner of his progress.

When an APS signal is required to be generated, the ASM generates an ANC signal that creates an output tone composed of major third intervals and even order harmonics. This creates a tone that is psycho-acoustically pleasant to the runner's ear. The AVC generates an amplitude envelope that smoothly oscillates from full amplitude to half amplitude, often referred to as tremolo, for two quarter-notes. This tremolo signal is generated for two quarter-notes, silence for two quarter-notes, then tremolo for two more quarter-notes. To the runner, the alert appears to flutter over top of the soundtrack and indicate that the runner as just accomplished a preset milestone.

ANS signals are composed of diminished third intervals and odd order harmonics, creating a tone distinct from the APS and easily identifiable and a negative alert. Here, the AVC generates an amplitude envelope that jumps sharply from full amplitude to half amplitude, often referred to a gating, for two quarter-notes. The same pattern of, two on, two off, two on, is repeated for the ANS.

5.3.6 – Output Control Module

The Output Control Module prototype generates the Audio Output Signal (AOS). This is the PCM audio signal that comprises the Soundtrack. Two modules are used by the OCM to generate this PCM signal, the Sampling Frequency Clock (SFC) and the Oscillator Block (OB). Inside the OB a series of IPcore modules, with sixteen bit two's compliment signed outputs, are used as tone generators. Outputs from the oscillators are routed through a summing block that adds 32768 to the signal for unsigned signal processing. The SFC drives the clock input for these modules at 48.8 kHz, are enabled by the DNC, BNC, HNC, LNC, and ANC, and have their accumulators refreshed and synchronized at the top of every quarter-note.

Outputs from the tone generators are routed to Velocity Modules (VM) that control the amplitude envelopes of the signal. Each of these signal are then routed to a discrete VM that utilizes the DVC, BVC, HVC, and AVC signals to create the required ADSR envelope. These velocity control signals are sixteen bit unsigned values that represent amplitude attenuators and are quantized to 128 parts per quarter-note. When full signal is desired the velocity value is zero and the signal passes through the block unaffected. If a one quarter attenuation is desired the velocity value is 16384. Each of the velocity control signals from the instrument modules that utilize a particular oscillator are summed together is this module and a simple signal processing procedure is implemented to ensure that the output signal from the VM does not overflow below a zero value if the sum of the velocity values are greater than the value output by the tone generator.

These signals are routed to a seventeen bit Processing Module (PM) that sums these signals and operates a simple audio compression scheme to ensure that the audio signal does not exceed the sixteen bit output requirements. A simple three stage compression algorithm uses the difference between the Maximum Value (MV) of 65536 and Current Value (CV) to smooth out the signal and reduce it to the required sixteen bit depth. At stage one, any value over 32768 is reduced by (MV-CV)/8, stage two reduces any value over 49152 by (MV-CV)/4, stage three reduces any value over 57344 by (MV-CV)/2, and finally a brick wall limiter reduces any value over MV to 65536.

In order to replicate both non-tonal rhythmic instruments and tonal harmonic instruments, the OB tone generators are divided into three sections. A block of four generators make up the Rhythm Block (RB), twenty two generators make up the Tonal Block (TB), and four generators function as High Frequency Oscillators (HFO) for frequency and amplitude modulation effects in the Effect Block (EB). This allows for the Tonal Section to generate signals that cover a five octave range and the Rhythm Section to generate a wide variety of non-tonal sounds.

The four fundamental oscillators that make up the rhythm block are set at frequencies outside of the Circular Temperament Tuning and are used to model a

kick drum (KIK), floor tom (TOM), snare drum (SNR), and a cymbal (CYM). Signals C1, E3, B3, G7, C9, and the HFO signals are routed from the TB and EB and to the RB and used to generate non-ordered harmonics. A granular method is implemented whereby all these signals are multiplexed and the select input is driven by a high frequency counter that determines the size of the grains and determines which signal the next grain gets pulled from. Two of these generates the complex waveform signals called the Rhythm Effects (RE1, RE2). These signals distinguish the rhythmic information from the melody and harmony structure of the Soundtrack. They are routed through the Rhythm Velocity Modules (RVM) and then to the Rhythm Processing Module (RPM) which sums and compresses these signals into the Rhythm Block Output (RBO).

A modern Circular Temperament Tuning is applied to the twenty two oscillators that comprise the TB. They are tuned in the key of C, all natural notes, beginning in the second octave (C2) and ending at C4 for the oscillators utilized by the Bass and Harmony Modules. The Lead Module uses seven fundamental oscillators that range from D5 to C6. Six granular synthesis modules generate the Bass Effect (BE1, BE2), Harmony Effect (HE1, HE2), and Lead Effect (LE1, LE2) signals. They are all of the reverse type granular synthesis. A grain buffer is filled sample by sample and while the buffer is filling the effect signal is in phase with the original signal. Once the buffer is full the samples are played back in reverse order until the buffer is empty and the cycle repeats. Each of the two effects utilize counters and the TCS to generate grain sizes that are functions of the quarter-note length. Signal BE1 has 8192 grains per quarter-note, signal BE2 has 4092, HE1 has 2048, HE2 has 1024 grains, LE1 has 512 grains, and LE2 has 256 and each buffer gets reinitialized each quarter note at the same time that the oscillator accumulators are refreshed. They are available as audio signals to the Bass Processing Module (BPM), which generates the Bass Block Output (BBO), Harmony Processing Module (HPM), which generates the Harmony Block Output (HBO), and Lead Processing Module (LPM) which generates the Lead Block Output (LBO). These granular signals are also summed with the HFO signals and reduced to a complex small signal oscillator for use as high frequency Tonal Modulation Oscillators (TMO) which are routed to the Bass Velocity Modules (BVM), Harmony Velocity Modules (HVM), and Lead Velocity Modules (LVM) for modulation of the ADSR gates.

The Effect Block also includes four oscillators. These oscillate at 2093 Hz (C7), 3136 Hz (G7), 5274 Hz (E8), and 8372 (C9). These intervals form a C major chord and the two highest frequencies, E8 and C9, exist in the octave range above a typical piano. Additive synthesis and both types of granular synthesis, reverse and multiplexed, are implemented in the EB to generate four audio Effect Signals (ES1, ES2, ES3, ES4). ES1 and ES2 are additive synthesis, ES3 is reverse granular, and ES4 s multiplexed granular. The EB has also has access to effect signals RE1, RE2, BE1, BE2, HE1, HE2, LE1, and LE2. Each signal has its own VM and send audio signals to the Effect Processor Module (EPM), which generates the Effect Block Output (EBO) signal. It also generates four complex waveform small

signal Effect Modulation Oscillators (EMO1, EMO2, EMO3, EMO4) which are used to modulate the BVM and TVM parameters and to phase modulate the C7 and G7 oscillators.

The Output Processor Module (OPM) sums and compresses the RBO, BBO, HBO, EBO signals and generates the AOS. This signal is sent through the digital audio converter and audio amplifier circuit that transforms the audio signal into an analog format that can be enjoyed by the runner.

6 – Project Prototype Testing

6.1 – Test Environment

The environment in which the product is used is very specific and does offer some challenges. These challenges were overcome by being anticipated and tested. This section covers the expected environment in which the MIM device can be used. Additionally, corner case environments are analyzed and tested as well in order to cover as many user-end scenarios as possible. The following parameters are integral in the test environment: temperature, moisture, duration, and acceleration.

6.1.1 – Temperature

Temperature is a key testing parameter in any electronics endeavor. This is especially true for the MIM device due to an atmosphere of a lot of sunlight and high body temperature and proximity. The following parameters were tested for in regards to the temperature: high operating temperature, low operating temperature, quick transitioning from high to low and vice versa.

High Operating Temperature – the maximum temperature that Music in Motion could see is 130ºF (54.4ºC). This temperature, in fact, could not even be achieved by running in the desert. It might only be seen in regular use by accidently leaving the device in a vehicle, which on a hot day could achieve 130ºF by trapping the heat inside. This proposed scenario is one that is not only possible, but very likely. Therefore, the MIM device was tested for its temperature operating conditions using this scenario. The device was left in a car for several hours in a closed car and suffered no ill effects.

Low Operating Temperature – the proposed scenario of leaving the device in one's vehicle can also yield the lowest temperature that can be encountered by this device under the care of a user. Given a very cold climate that the user might live in, such as Minnesota, leaving the device in one's car would subject the device to its raw temperature. The coldest temperature in which this device could be tested is -20º F (-28.9ºC). This extreme temperature was unachievable in Florida but the Music In Motion device was left in a home freezer, which was estimated to be between 0-10º F, for two hours. The device suffered no ill effects.

6.1.2 – Duration

In order to have a reasonable user-end product, one must have a device duration that can exceed its purposes. Once the device was functional, it underwent the duration test many times in order to see what the average battery lifespan is under

the load of the hardware. The duration must be longer than its application, which means that Music in Motion must last longer than the user's run. The final device design is powered by a standard nine volt battery. Two types of batteries were tested for duration, generic batteries and Duracell. Generic batteries did not last as long as the Duracell and averaged forty seven minutes of use before the voltage dropped below the seven volts required by the voltage regulator. The Duracell batteries lasted an average of fifty nine minutes. While this running time might not be sufficient for very experienced runners a time of nearly one hour with non-generic batteries is enough for most causal runners.

6.1.3 – Acceleration

This is perhaps the most fundamental element to the hardware testing. The device must be sensitive enough in order to pick up a brisk walk and durable enough to be shaken to generate a reasonable accelerometer signal. The acceleration is unpredictable as far as what the device might experience, although its typical use is around 5g at most. In either case, typical or atypical, the device was tested in order to verify its performance among a range of accelerations.

The acceleration testing included a walking test, running test, and shaking test. Initially the shaking test was utilized on the prototype to set the required boundary conditions utilized by the sensor microcontroller to identify footfalls. The results of this test were used to fine tune the device control protocols and the results of this test can be seen if the following Figure 47. This clearly shows that the predictive control system is accurately matching the input from the accelerometer.

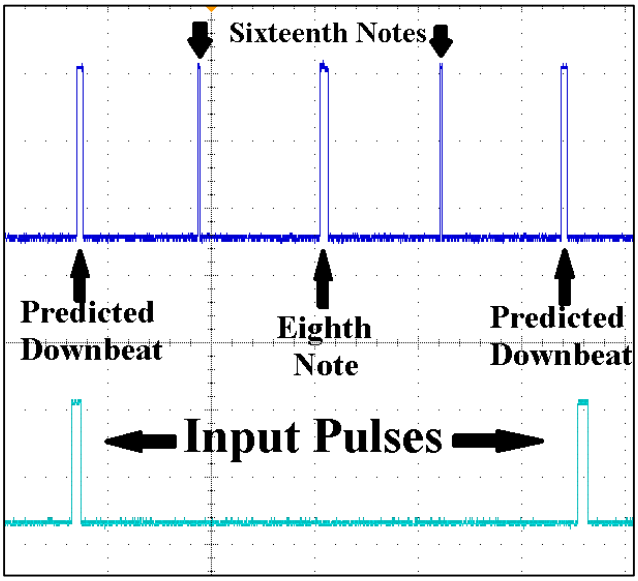


Figure 47 - Running Test Output

Once the final prototype was assembled the running test was implemented and this test is as simple as it sounds, just put the device on and run. The device

passed this test with flying colors. The Music In Motion system does an excellent job of matching the playback tempo to the pace of the runner's footfalls. The walking test was more problematic and the system does not do an efficient job of differentiating the slight acceleration of walking footfalls and accelerometer signal noise.

6.2 – Full Functional Simulation

Once the individual test environments were established and simulated, the full functional simulation must follow. This is the point in which the device is fully assembled and tested on a system level in reference to each test environment. The standard use simulation consisted of a typical user experience. This experience can be described as a light jog, at a reasonable pace, at a typical temperature, for an average distance, on pavement. To be more specific, the tester ran with the MIM device and essentially used it several times in order to give qualitative feedback. A light jog consists of a typical running stride whereby the heel must not slam down on the ground; this corresponds for the demand in accelerometer sensitivity. A reasonable pace tests the typical frequency of the running signal that was be sampled and filtered within a certain bandwidth. Typical temperatures can be simulated from 50ºF to 90ºF whereby most runners who exercise do so within this temperature range. This was accomplished by executing the simulation outside. Testing runs averaged one half mile. This standard use simulation was conducted on pavement as well because this is where the typical runner exercises. Additionally, it yields a great output signal from the accelerometer to the control system. In adherence to these specifications, the typical use the Music in Motion device was simulated such that we can observe its response and the device excels under these conditions.

7 - PCB Design and Assembly

7.1 Prototype PCB

The Music In Motion system was prototyped primarily on development boards and did not require a large sale PCB. However, after the initial DAC prototype was tested on a bread board the second stage DAC prototype was implemented upon vector board. The 2R Ladder DAC along with the op amps and headphone audio output were hand soldered to the board to create a custom DAC development board. This will allow for an easy integration with the microprocessors, accelerometer, and GPS development broads. After this initial prototyping testing was completed a secondary prototype was then designed utilizing an accelerometer development board, GPS development board, and a MIDI synthesizer development board all designed by Adafruit and two Arduino Unos. This prototype represented the final design and used the same hardware that would be implemented of the final Music In Motion device. Once this prototype was tested completely and the device was functioning correctly the final PCB could be designed.

7.2 Final 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.

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. The following Figure 48 shows the Eagle board file of the final design implementation.

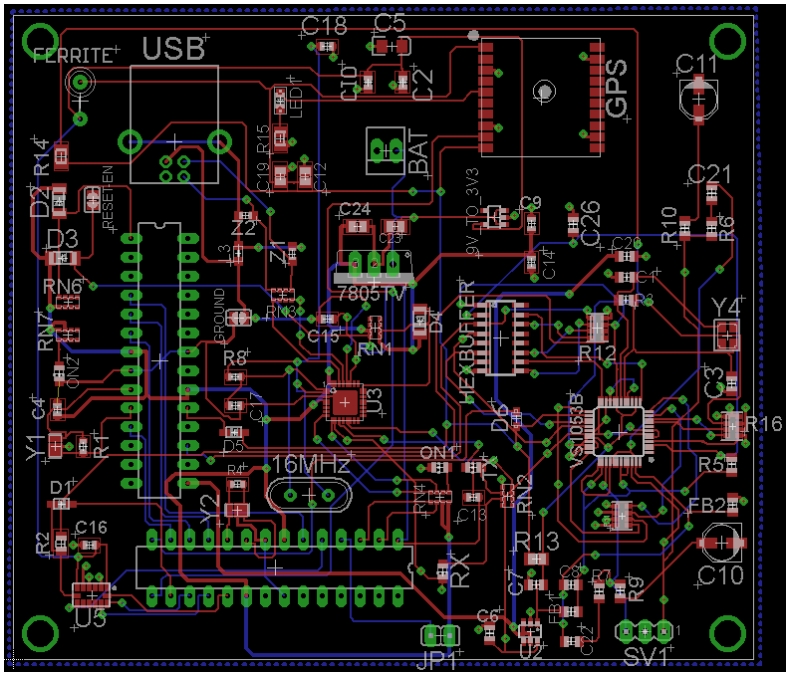


Figure 48 - PCB Layout

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.

8 – Administrative Content

8.1 – Permissions

TI:

TI grants permission to download, print copies, store downloaded files on a computer and reference this information in your documents only for your personal and non-commercial use. But remember, TI retains its copyright in all of this information. This means that you may not further display, reproduce, or distribute this information without permission from Texas Instruments. This also means you may not, without our permission, "mirror" this information on your own server, or modify or re-use this information on another system.

TI further grants permission to non-profit, educational institutions (specifically K-12, universities and community colleges) to download, reproduce, display and distribute the information on these pages solely for use in the classroom. This permission is conditioned on not modifying the information, retaining all copyright notices and including on all reproduced information the following credit line: "Courtesy of Texas Instruments". Please send us a note describing your use of this information under the permission granted in this paragraph. Send the note and describe the use according to the request for permission explained below.

Battery University:

Permission to Use Material

This letter confirms that the following name has been given permission to use the below named figure.

Name: Eric Hoofnagle, University of Central Florida
Article Name: Charging Lithium-ion
Figure Title: Figure 1: Charge stages of lithium-ion
Reference name for material used: courtesy of www.batteryuniversity.com
Image (below)

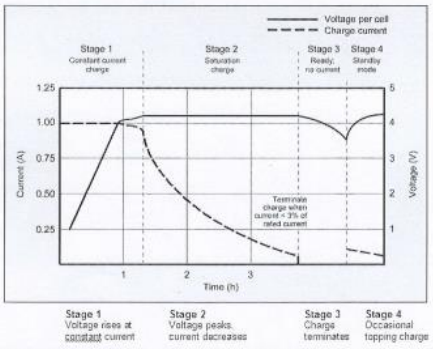


Figure 1: Charge stages of lithium-ion. Li-ion is fully charged when the current drops to a predetermined level or levels out at the end of Stage 2. In lieu of trickle charge, some chargers apply a topping charge when the voltage drops to 4.05V/cell (Stage 4).

Date: April 28/2014 Signature:

u-blox:

Terms & Conditions: u-blox website

Print E-mail page

Note: the following only applies to content provided on this website. For general terms and conditions for doing business with u-blox,click here.

Your access to and use of the website www.u-blox.com and documents on this website is subject to the following terms and conditions. By accessing and browsing this site, you agree to accept without limitation or qualification the following terms and conditions

Use of information
This website and documents on this website (hereinafter this website) are put at your disposition for information purposes only and information contained therein is not binding. A download, copy or print of this website is only permitted for personal non-commercial use. You may not distribute, modify, transmit, repost or use this website for commercial purposes or establish links to it without written permission of u-blox.
Disclaimer

SparkFun Electronics:

Product Photos: SparkFun product photos may be used without permission for educational purposes (research papers, school projects, etc.). Permission must be granted for commercial use and proper credit to SparkFun must be given. For inquiries about the use of our product photos or permission to use them, please contact marketing@sparkfun.com.

Analog to Digital (ADC) and Digital to Analog (DAC) Converters – Dr. M. Rabiee:

Hello Eric,

That is fine, you can use those figures from my paper.
Can you describe your senior design project?

Good Luck,

Max

M. Rabiee, Ph.D., P.E.
Professor, Department of Electrical Engineering and Computing Systems
College of Engineering and Applied Science

Rhodes 822, ML-30
University of Cincinnati
Cincinnati, OH 45221
Phone: (513)556-6559

rabieem@ucmail.uc.edu

mark as un

Eric Hoofnagle
Sun 4/27/2014 3:30 PM
Sent Items

Dr. Rabiee,

We are a senior design group at the University of Central Florida and we are requesting to use "Figure 1: 8-bit, Binary Weighted Type DAC" and "Figure 2. 8-bit, R-2R Type Digital to Analog Converter" from the following paper: Analog to Digital (ADC) and Digital to Analog (DAC) Converters.

We would like to use the two figures for our educational Senior Design I paper at the University of Central Florida.

Thank you,
Eric Hoofnagle
erichoofnagle@knights.ucf.edu

Analog Devices:

8.2 – Datasheets

TMS320 Microcontroller:
http://www.ti.com/lit/ds/symlink/tms320f28020.pdf

C2000 Microcontroller (for reference):
http://www.ti.com/lit/ug/slau056j/slau056j.pdf

Spartan-3E FPGA:
http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

GPS Module:
http://www.mouser.com/ds/2/23/M10382_11MD-0043-1-PS-8622.pdf

Memory Module:
http://www.atmel.com/Images/Atmel-8812-SEEPROM-AT24CM01-Datasheet.pdf

Accelerometer:
http://www.analog.com/static/imported-files/data_sheets/ADXL345-EP.pdf

Low Power Digital-to-Analog Converter:
http://www.ti.com/lit/ds/symlink/lm321.pdf

USB Port:
http://portal.fciconnect.com/Comergent//fci/drawing/10118193.pdf

8.3 – References

Apple. (n.d.). *A Brief History Of The Synthesizer*. Retrieved from https://documentation.apple.com/en/logicexpress/instruments/index.html#chapter=A%26section=5%26tasks=true.

Battery University. (n.d.). *Charging from a USB Port*. Retrieved from http://batteryuniversity.com/learn/article/charging_from_a_usb_port

Češko, I. I. (n.d.). *Implementation USB into microcontroller: IgorPlug-USB (AVR)*. Retrieved from http://www.cesko.host.sk/IgorPlugUSB/IgorPlug-USB%20(AVR)_eng.htm

Clint Goss, P. D. (n.d.). *Flutes of Gilgamesh and Ancient Mesopotamia*. Retrieved from http://www.flutopedia.com/mesopotamian_flutes.htm.

Dimension Engineering LLC. (n.d.). *Robotics Radio Control Power Electronics*. Retrieved from https://www.dimensionengineering.com/info/accelerometers.

Goeth, P. (n.d.). *J.S. Bach's Well-tempered Clavier*. Retrieved from http://www.bachwelltemperedclavier.org/

ICCNexergy. (n.d.). *Custom Battery Systems*. Retrieved from http://www.iccnexergy.com/battery-systems/battery-chemistry-comparison-chart/.

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

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

Oellers, H. (n.d.). *Wave Field Synthesis*. Retrieved from http://www.holophony.net/Wavefieldsynthesis.htm.

Rabiee, M. (n.d.). Converters, Analog to Digital (ADC) and Digital to Analog (DAC). Eastern Kentucky University.

SparkFun. (n.d.). *GPS Buying Guide*. Retrieved from https://www.sparkfun.com/pages/GPS_Guide

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

Total Phase, Inc. (n.d.). *7-bit, 8-bit, and 10-bit I2C Slave Addressing*. Retrieved from Total Phase: http://www.totalphase.com/support/articles/200349176

Venkat, K. (n.d.). Optimized Digital Filtering for the MSP430. *MSP430 Advanced Technical Conference.*

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