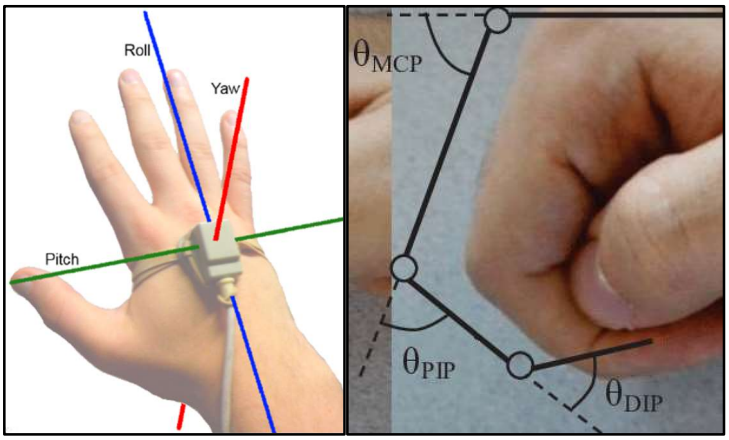
EEL4924C Electrical Engineering Design II
Preliminary Design Report (PDR)

Hand Tracking Glove

Submitted by:
(Daniel Hamilton, d.hamilton@ufl.edu, 352-514-4490)

Project Summary

The goal of this project is to build a wearable glove that will track bends in each finger and 3-dimensional hand orientation of the right hand. This device will capture only MCP and PIP joint bends. DIP joint bends can be closely estimated in software based on PIP joints. The glove will produce messages containing sensor data that can be transferred to a user-end application via Bluetooth. Any Bluetooth 2.0 capable device will be able to connect to the glove and decode the messages following the glove's message protocol. My user-end application to demonstrate the glove's capabilities will be an interactive audio effects rack utilizing the hardware platform from the Real-Time Digital Signal Processing course (EEL4750).



Figures 1 and 2: **Hand orientation (left) and MCP/PIP finger bends (right)**

Project Features

In this section, I will discuss the core features and objectives of this project. The technical implementation will be discussed later in the *Technology Selection* section. This project will feature two PCBs - one for the glove and one for the battery charger. The features for each PCB are separated into two lists below.

Glove PCB Features

- Capture bends in 2 joints for all 5 fingers on the right hand. Most other hand tracking gloves I have found only estimate the bend of 3-5 fingers using one sensor per finger. It is not uncommon for the MCP joint to bend without bending the PIP joint. By capturing the bends of multiple joints per finger, I am broadening the possible applications for my glove.
- Finger sensor bend resolution must be high enough to detect noticeable differences in MCP or PIP joint movements. The Nintendo Power Glove, which inspired this project, could only capture 4 positions per finger. This means that the user's finger could move a significant amount without the glove capturing any change. My aim is to improve this flaw with my design.
- Provide sensor data that can be converted into hand orientation values on all 3 axes (either on the glove or in the desktop application). Another downfall of the Nintendo Power Glove is that it could only capture yaw and roll (not pitch). My glove will provide sensor readings that can be used to capture all 3 axes - pitch, yaw, and roll.
- The glove must operate wirelessly. I will power the glove with a rechargeable battery and transmit data to the application wirelessly using Bluetooth.
- The device should be lightweight and comfortable to wear.
- Provide a text-based user interface on the glove using an LCD. This display will show the battery's charge percentage and Bluetooth connection status.

Charger PCB Features

- A battery charger circuit that safely charges the Li-ion battery to max capacity. This circuit will implement Constant Current (CC) and Constant Voltage (CV) circuits. The circuit will automatically switch between CC, CV, and open circuit outputs based on the battery's current and voltage thresholds.
- The charger PCB will play a 4 note tune when the glove is connected or disconnected. This will be completed using analog and digital hardware only and will not require any software/microcontroller assistance.
- The charger PCB will automatically detect when the glove PCB is connected or disconnected even if the Li-ion battery on the glove PCB is fully depleted.

Figure 3 is a high-level diagram showing the hardware required to implement the features described above. More details are provided on each of the features in the section below.

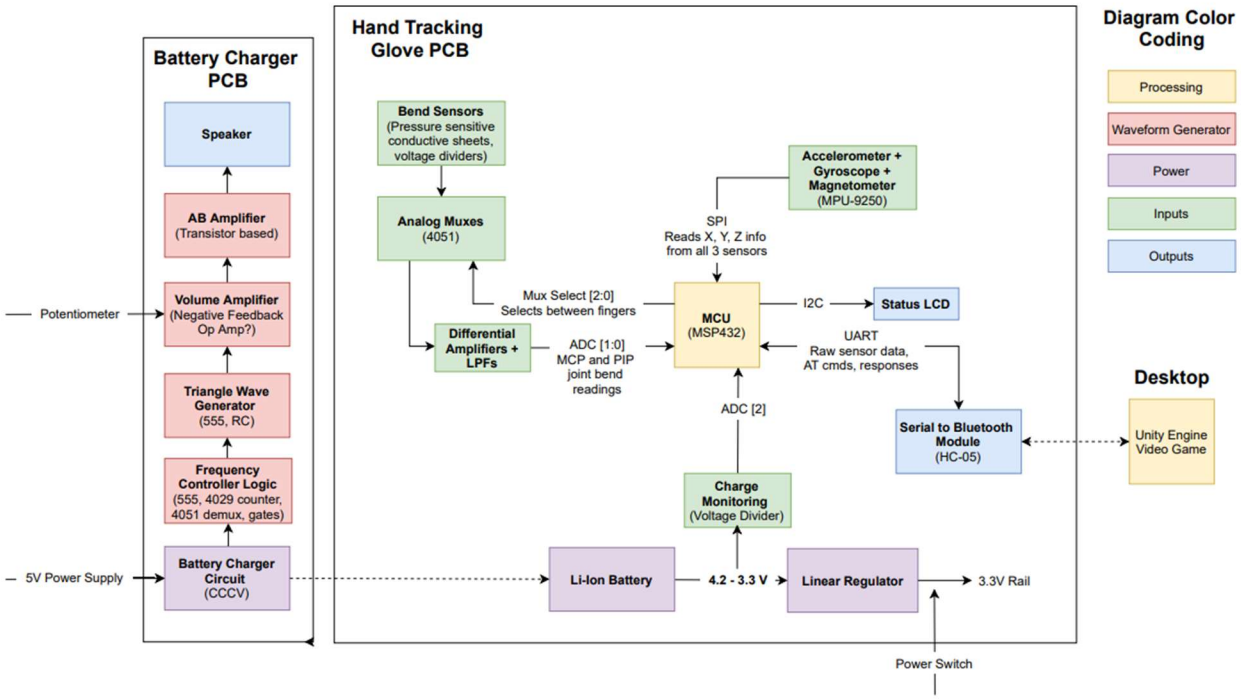


Figure 3: **High-level diagram of the hardware on the battery charger and hand tracking glove PCBs.**

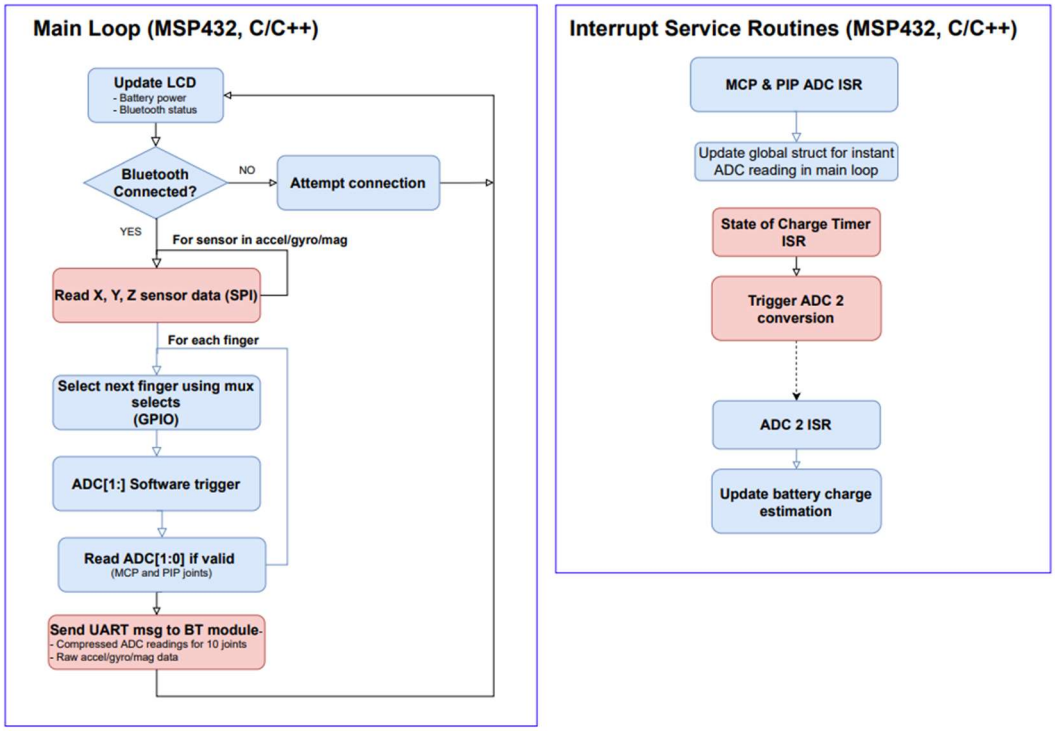


Figure 4: **Software flowchart for the glove's MSP432 software.**

Technology Selection

Capturing Joint Bends

I will use “flex” sensitive resistors attached above and below each joint, which will change resistance as the joint bends. When used in a voltage divider, the output voltage can be measured through the microcontroller’s 12-bit ADC and joint bends can be estimated based on the change in voltage. Each finger’s MCP and PIP joints will be captured, and the DIP joint will be ignored.

The primary issue with using 10 analog sensors is that most microcontrollers do not have 10+ ADC lines. To solve this problem, I will use two 4051 8x1 analog multiplexer ICs to switch between finger voltages to output. There will be an MCP specific mux and a PIP specific mux. These will allow the microcontroller to select which finger to take ADC measurements on using three GPIO as mux selects. The downside to this approach is that the microcontroller will only be able to take 2 ADC measurements in parallel instead of 10.

Another issue with using flex sensors is pricing. The popular Adafruit flex sensors are $10 per sensor, which will cost $100 for the number of sensors I need (or more if I need replacements). To solve this problem, I will make homemade flex sensors using a pressure-sensitive conductive material and copper foil. The resistance of this material decreases as pressure increases, which works well as a flex sensor because finger bends will add pressure to the material.

Using homemade flex sensors creates a new problem - I will not know what the resistance range will be until the sensors are built. If the resistance range is low, the output voltage range the ADC can capture will be low. A small number of ADC steps between a straight joint and a bent joint will create inaccurate joint angle estimations. This problem can be solved using an op amp to subtract out the minimum voltage and amplify the area of interest to 0 - 3.3 V. For example, if my voltage range is 2.51 - 2.56 V, I can subtract 2.51V and amplify the remaining 0.00 - 0.05V range to 0 - 3.3 V. This would result in a larger number of ADC steps for the voltage range of interest.

Finally, I will use an active low-pass filter on each mux output to reduce noise on the ADC readings. This may become especially important if I need to amplify a very small voltage range as previously described because the noise will also be amplified.

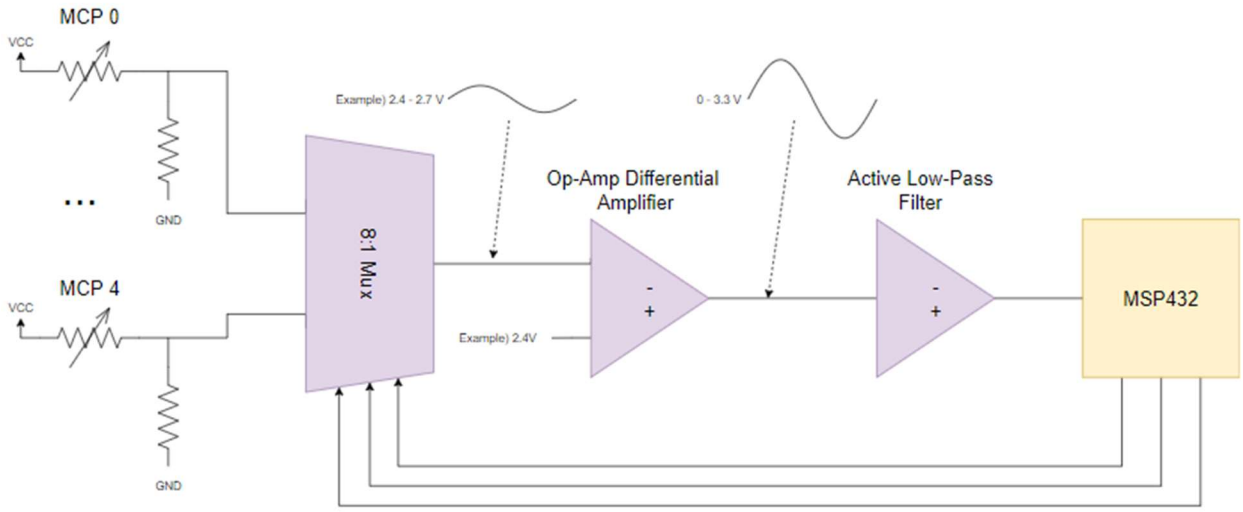


Figure 5: **Mid-level hardware diagram showing MCP flex sensor reading circuit. The glove PCB will contain an identical PIP flex sensor circuit.**

The 12-bit ADC will provide 4096 (2^12) ADC steps assuming the full voltage range from GND - VCC is available. The 12-bit values will be compressed down to 8-bit values to decrease message length, which could decrease latency when transferring data to the demo application.

Provide Data for Calculating 3-Axes Hand Orientation

My first idea to capture hand rotation around pitch, yaw, and roll axes was to simply use a gyroscope. In theory, the angular velocity data could be integrated to determine rotational position (orientation). However, high-precision gyroscopes are expensive ($200 - $1000+ range) and cheaper gyroscopes become highly inaccurate due to drift. The alternative I will be using is to combine a cheap gyroscope with an accelerometer and magnetometer using a sensor fusion algorithm. As shown in Figure 6 below, the gyroscope data is used to find an estimate for the rotational position or “attitude.” Accelerometer data measures the direction of gravity and can be used to offset the gyroscope’s pitch and roll drifts. The magnetometer is treated as a compass to point north. This data can be used to correct the yaw drift.

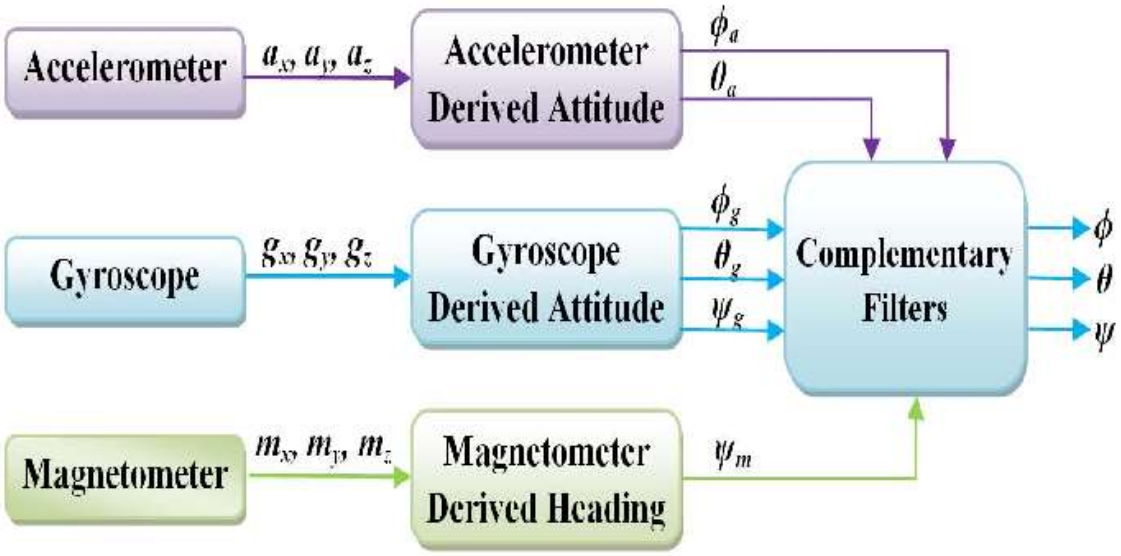


Figure 6: **High-level diagram showing combination of sensor data to produce orientation results.**

An important aspect to consider is whether implementation of the sensor fusion algorithm should be done on the device or on the user-end application. Initially, I planned to perform the calculations on the glove side to make the user-end application easier to develop -- the user would simply receive the orientation data without having to do any processing. However, choosing to transfer raw accelerometer, gyroscope, and magnetometer sensor data instead of processing the data into quaternions on the glove directly provides two benefits listed below.

- **Increased user application control.** Some applications may require 9-DOF (degrees of freedom) sensor fusion of all 3 sensors since it provides the most accuracy, while other applications that produce disruptive electromagnetic fields will need to exclude the magnetometer and perform 6-DOF sensor fusion. Additionally, some applications may only data from the sensors for a different application such as calculating position from the accelerometer data. These uses are all possible if the raw data is transferred.

- **Decreased uC space on glove's PCB.** Sensor fusion algorithms require high processing power and preferably a FPU for the floating point math. The microcontroller I am most comfortable with that could handle this processing is the TMS320 DSP, which takes up a lot of PCB space in the QFP package. Offloading the processing to the user-side application allows me to choose a smaller and less powerful processor (MSP432), which takes up a small amount of space on the PCB. This helps reduce the PCB size, which keeps the glove light and comfortable to wear.

For these reasons, I am choosing to simply read and compress the sensor data before transferring it to the user-end application for processing.

Bluetooth Message Transmission

I chose to use Bluetooth for wireless data transmission because Bluetooth allows the device to be connected to most computers or phones without additional hardware. For the breakout board, I am using the HC-05 UART to Bluetooth 2.0 module. I chose this board because (1) it can send *and* read

messages through a UART interface, and (2) it has Arduino drivers that I believe I can modify to work with the MSP432.

Battery Charge Monitoring

I will estimate the battery's remaining charge or "State of Charge (SOC)" using a method called "Coulomb Counting." This method involves estimating the state of charge by subtracting the current consumption from the initial capacity. The battery percentage can be *estimated* with the following formula:

$$(Q_{previous} - i_{current} * deltaTime) / Q_{total} * 100\%$$

Unfortunately, the error in this estimation will increase each time the battery is recharged. However, my goal is just to provide a rough estimation so the user knows to plug it in when it's in the 10-25% range, so this will work well for my application.

An alternative that I considered is Open Circuit Voltage (OCV) estimation. With this approach, the battery charge can be estimated by comparing the battery's open circuit voltage to the linear portion of the battery's discharge curve. This would be simpler and more accurate since it does not require knowledge of the starting capacity; however, the battery would have to be an open circuit for each measurement. This would power off the microcontroller, disabling the necessary ADC pin along with the rest of the glove's functionality.

Battery Charger Circuit (CCCV)

Recharging Lithium-ion batteries involves a 2-step process - Constant Current and Constant Voltage (CCCV). The first stage provides the battery with a constant flow of the max charging current until the battery reaches a 4.2V threshold. At this threshold, a transistor will switch the charger to the second stage where it provides constant voltage until the battery's current draw reaches a minimum threshold of around 10% of the max charge current. At this threshold, a second transistor will switch the charging cut off the charging circuit from the battery.

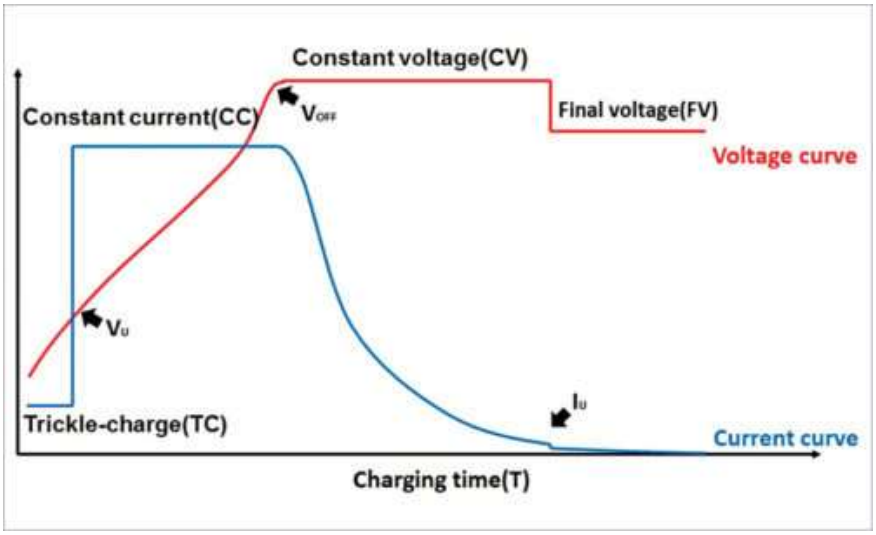


Figure 7: **Charge characteristics curve showing expected circuit outputs.**

My current plan is to implement this circuit using a current mirror circuit as the constant current (CC) supply, and a 5V linear regulator with a voltage divider as a 4.2V constant voltage (CV) power supply. I am still working on the circuit designs to (1) switch between power supplies at the 4.2V threshold and (2) enable or disable the charger at the 10% current threshold.

Waveform Generator & Amplification

This circuit will play a 4-note tune either forwards or backwards when triggered. I've attached a high level diagram of how this circuit will work in Figure 8 below. I've also listed an explanation of the steps starting from the bottom after the charger PCB is connected to the glove PCB. Throughout this explanation, “connection” or “disconnection” refers to the charger PCB connecting/disconnecting from the glove PCB.

- When connected, the *Connected* pin is set low. When connected, the pin will be high.
- There will be 2 npn transistor based circuits for detecting positive edges and negative edges to notify the rest of the circuit that a connection/disconnection has just happened. The positive edge detector will pulse for a disconnection and the negative edge detector will pulse for a connection.
- The DFF is used to control whether the 4029 counter will count up or down. This is because I want the tune to be played forward or backwards depending on if the glove is connected or disconnected. The user can audibly tell the difference between a connection or disconnection.
- The edge detectors will trigger a monostable 555 timer, which will enable an astable mode 555 timer long enough to output 4 pulses.
- The 4029 counter will count from 0-3 or 3-0 depending on if it is in up or down mode.
- The 4051 demux decodes the 0-3 binary values and selects a resistor value to be used in the next stage's 555 timer for frequency selection. Some experimentation on the breadboard will determine if any BJTs or MOSFETs are necessary for this stage.
- The 3rd 555 timer and RC circuit will output a triangle wave at the frequency selected in the previous stage. This output is what will be played through the speaker, so the resistors need to be carefully tuned to notes in the song.
- The volume controller will amplify the signal in the range 0V - 3.3V. I am currently planning to implement this using a negative feedback op amp.
- The AB amplifier will keep a 1:1 voltage gain but allows for more current to be drawn from the power supply rather than the volume control amplifier. Output will be fed into the speaker.

The easy way out of this would be to have a microcontroller on the charger PCB poll for connection and then play 4 notes through the DAC using a triangle wave LUT and a timer. The main reason I am choosing to go this route is to meet the analog / hardware complexity requirements.

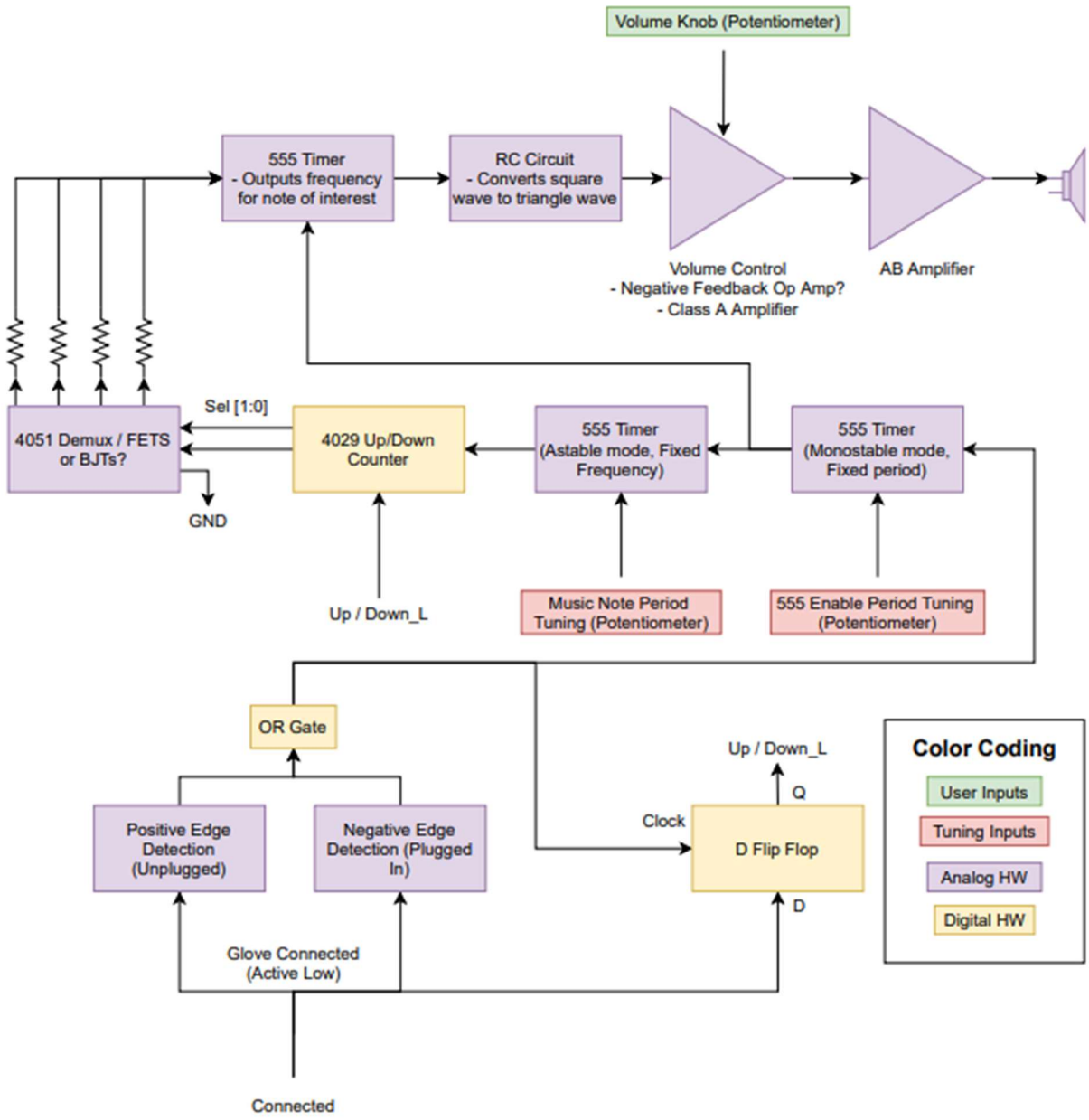


Figure 8: **High-level block diagram of hardware used in the waveform generator and speaker amplifier.**

Glove & Charger Connection Detection

The circuit for detecting connection between the charger PCB and the glove PCB will use a pull-up resistor and a disconnectable ground wire looping between the two PCBs. When the PCBs are disconnected, the pull-up resistor will pull the *Connected* pin high. When the PCBs are connected, the ground signal will pull the *Connected* pin low. Other approaches I considered required the battery on the glove PCB to have some charge remaining or interaction from the microcontroller. Since one of my design goals is to have the waveform generator work independently of the microcontroller, I went with the approach illustrated in Figure 9.

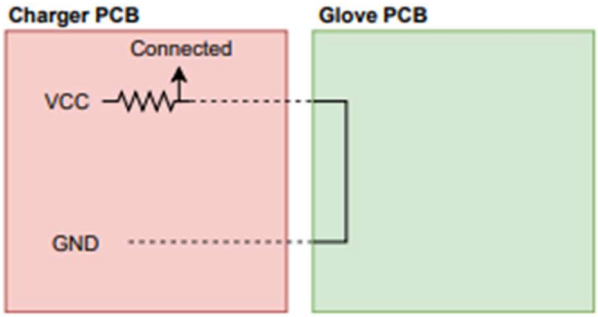


Figure 9: **Connection detection circuit that will trigger 4 note tune circuit.**

Interactive Audio Effect Rack - Demo Application

The goal of this project is to create an expressive and easy-to-use controller. However, a controller is not an interesting project without an application to control. I will use the glove's outputs to control several audio effects on a DSP/codec platform. This section will explain the hardware and software components that will go into my demo.

Because this demo is not core to my project and is simply meant to show the glove's capabilities, I will be using evaluation boards instead of designing a PCB to contain these parts. Figure 10 shows the components used in this application from a high level.

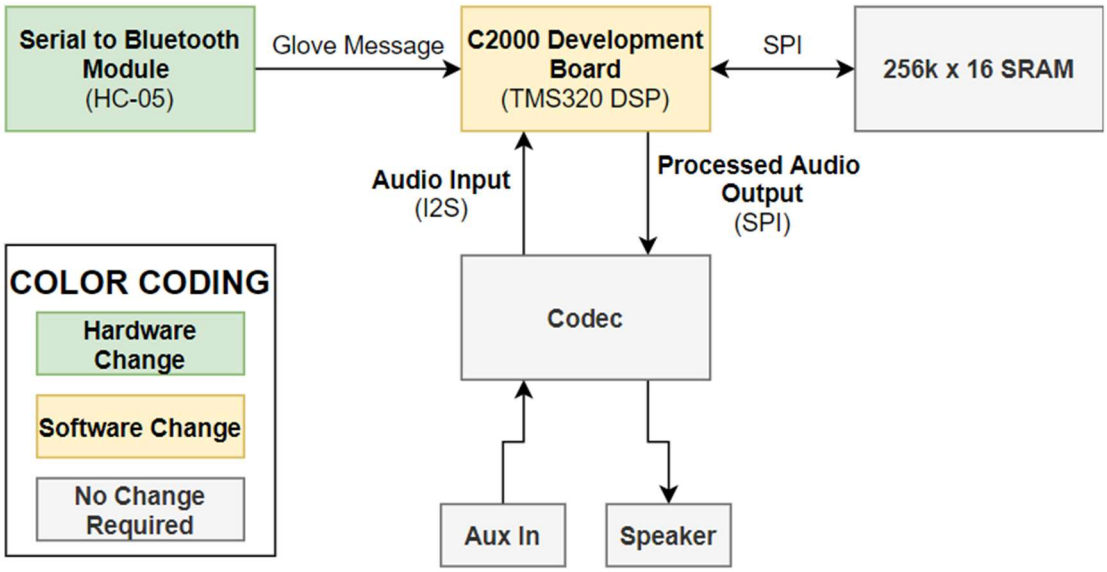


Figure 10: **Hardware platform for the demo application.**

- The C2000 will receive raw sensor data from the glove and read it from the HC-05 module via UART. The microcontroller will decode the message data according to the established message protocol.
- The sensor fusion algorithm will be implemented on the DSP to produce hand orientation in the form of quaternions. Quaternions were chosen over euler angles to handle an edge case called "gimbal effect" where 2 axes can align, locking the 3 axes into a 2 dimensional space. The two sensor fusion filters I found the most documentation on are the Madgwick and Mahony filters.

Although the Madgwick filter is more accurate, it is known to take up a lot of memory and increased CPU time. The DSP for this demo will already be consuming a large portion of the memory to perform FFT/IFFT algorithms, so I am choosing to use the Mahony filter.

- The C2000 will implement several time and frequency domain effects listed in the table below. The microcontroller software will map various sensor readings from the glove to control "knobs" of the effects. For example, the user rotating their hand along the X axis could increase or decrease the cutoff frequency for the low-pass filter.

Effect	Tunable Parameters	Domain	Controlled by
Low-Pass Filter	Cutoff Frequency	Frequency	Hand pitch axis
High-Pass Filter	Cutoff Frequency	Frequency	Pointer finger PIP joint
Pitch Shifting	Pitch (by frequency bin index)	Frequency	Hand roll axis
Echo **(**Optional)**	Delay time	Time	Middle finger PIP joint
Toggle effect rack enable	Enable / Disable	None	Thumb MCP joint

- Audio input will come from the codec via I2S, sampled at 48 kHz. The audio effects discussed previously will be applied to this data and output back to the codec via SPI. Data transportation during the input, processing, and output stages will be handled using 2 DMA channels and ping-pong buffers, which will allow data to be sampled and output in real time.
- The external SRAM will be used for processing time domain effects if time permits for implementing this. To produce this effect, the processor needs to "remember" the previous outputs to sum with the original inputs. Remembering enough outputs to produce this effect consumes a large amount of memory as the delay time increases. This external SRAM will be used to store previous outputs for the algorithm's next iteration.

The software flowchart below shows how I plan to handle the demo application tasks. The TMS320's Core 1 will handle Bluetooth connection, message decoding, and sensor fusion processing. The processed inputs are moved to shared memory to be accessed by Core 2 when needed.

Core 2 performs all audio effect processing. The processed glove inputs from Core 1 are read from the shared memory. These values are used to tune the frequency and time domain effects before being output to the codec. In order to reduce IO delay, core 2 will utilize 3 buffers in ping-pong buffer fashion. In Figure 11, the dotted red line shows how each of the three buffers will be constantly rotating between being used as input, processing, or output buffers. The input buffer will contain unprocessed data from the codec, the processing buffer will contain data that is currently being processed, and the output buffer will contain processed data ready to be sent back out to the codec.

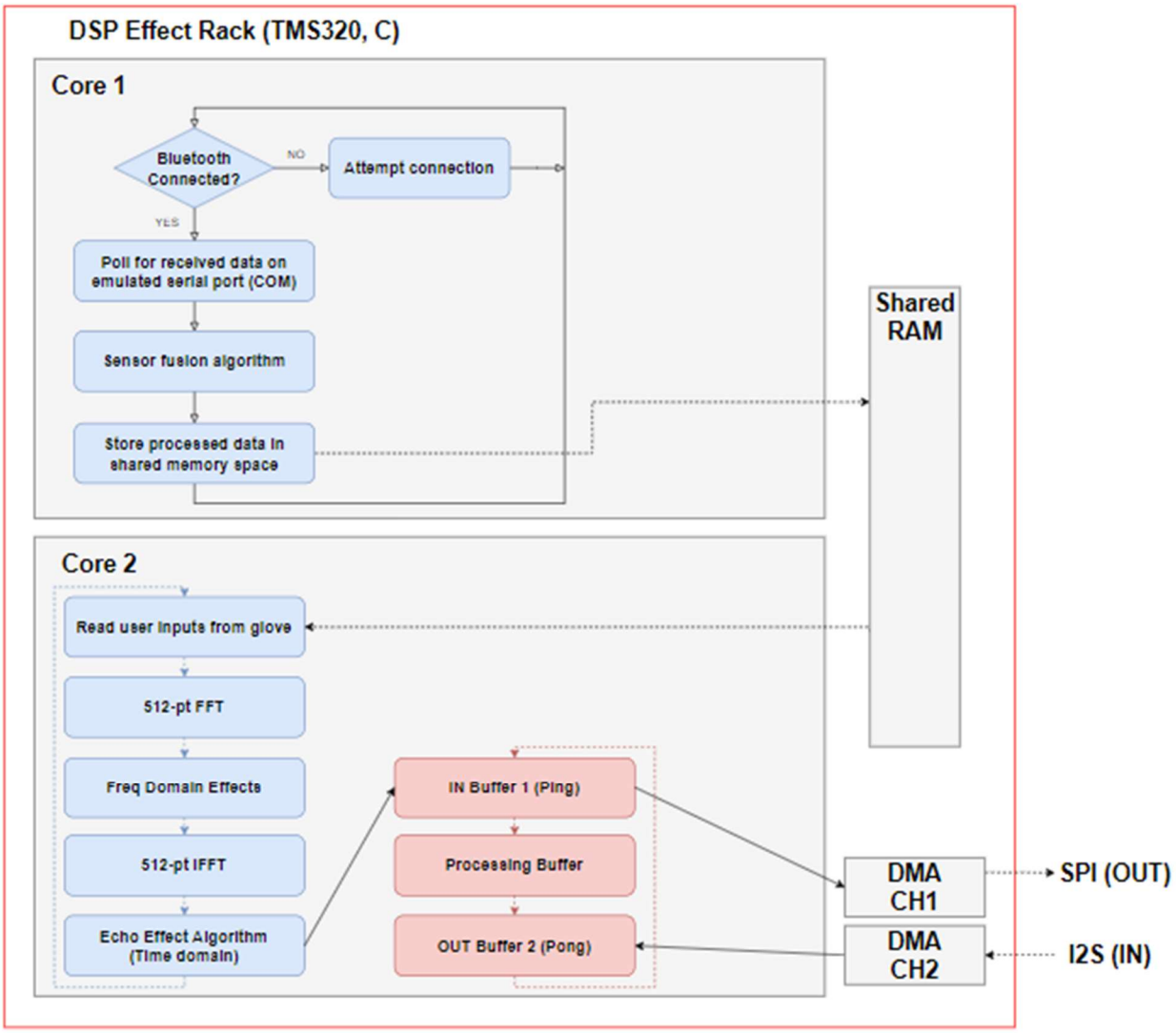


Figure 11: **Software flowchart for the demo application's TMS320 software.**

Gantt Chart

I have included my Gantt chart and the main tasks in the screenshots below. I have provided 3 weeks for design, 4 weeks for hardware implementation and initial PCB design, 4 weeks for software and demo implementations, and 3 weeks for polishing and documentation.

TASK NUM	TASK	PROGRESS	START	END
1	**Design**			
1.1	Li-ion Battery Charger - CCCV research, circuit design	75%	1/23/21	1/29/21
1.2	Joint Movement Sensors - Stretch material selection, circuit design	100%	1/29/21	1/30/21
1.3	Wireless Communication - Bluetooth device selection	100%	1/26/21	1/26/21
1.4	Gyroscope, accelerometer, magnetometer selection	100%	1/28/21	1/28/21
1.5	Sensor Fusion - Algorithm comparisons, define expected output	100%	1/30/21	1/30/21
1.6	Order Prototyping Components - Staggering orders so parts can be tested early next week	75%	1/24/21	1/30/21
2	**Hardware Prototyping**			
2.1	Joint Movement Sensors - Build flex sensors, divider and mux circuits, and basic SW to read joint data.	50%	1/31/21	2/4/21
2.2	Wireless Communication - Drivers to transfer data between MSP432 and C2000 dev board.	0%	2/5/21	2/9/21
2.3	Speaker hardware - wave generator and controller logic, voltage amplifier + AB amplifier	0%	2/8/21	2/15/21
2.3	Gyroscope, Accelerometer, Magnetometer - I2C connections + basic SW to read X, Y, Z.	0%	2/10/21	2/15/21
2.4	Status LCD wiring and software to write characters out	0%	2/16/21	2/16/21
2.5	Li-ion Battery Charger - Breadboard the battery charger and indicate charging mode on LEDs (CC, CV, OFF	0%	2/17/21	2/20/21
2.6	Li-ion Battery Charge Guage - HW/SW for Voltage Estimation vs Coloumb Counting experiment	0%	2/21/21	2/23/21
2.7	First PCB Design - Charger PCB + Glove PCB Draft	0%	1/31/21	2/28/21
3	**SW Implementations & HW Improvements**			
3.1	Main C code development combining all inputs and outputs on MSP432	0%	3/1/21	3/11/21
3.2	Mahony sensor fusion implementation on C2000	0%	3/12/21	3/17/21
3.3	DSP Effect Rack software on C2000 and codec	0%	3/18/21	3/28/21
3.4	First PCB assembly & Debugging	0%	3/14/21	3/19/21
3.5	Second PCB Design - Fix hardware issues from Glove PCB 1, reduce PCB size, all SMD for glove PCB	0%	3/20/21	3/24/21
4	**Polish & Demo**			
4.1	Extra time if tasks took longer than expected. Otherwise, stretch goal SW development goes here.	0%	3/29/21	4/6/21
4.2	Second PCB assembly & Debugging	0%	4/7/21	4/14/21
4.3	Writing final report	0%	4/15/21	4/16/21
4.4	Slide creation & presentation	0%	4/17/21	4/18/21

Figure 12: **Simplified task breakdown for Gantt Chart.**

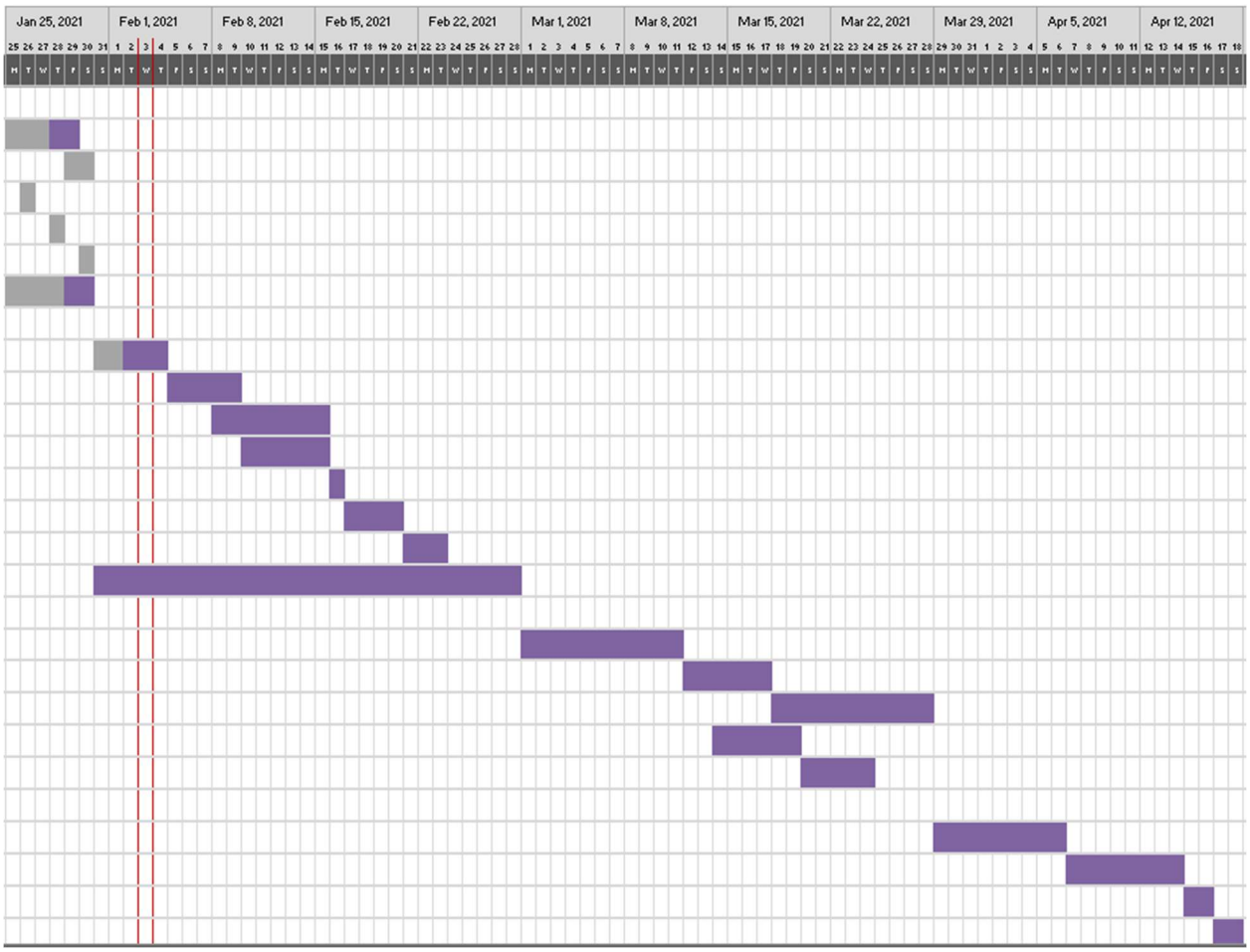


Figure 13: **Gantt chart diagram.**