The Metronome

EE3376

By: Marc Villarreal, Luis Santos, Esteban Acosta

Introduction

In this project we are learning to work with and create our own version of a metronome device that will work for any musician. The purpose of a metronome is to be able to set a beat or tempo that will allow the musician using the device to keep beat with the music he or she will be performing. The metronome is mainly used by the conductors of the band when teaching songs as well as it is used for practice by the individual band members. In this project we will create a workable metronome that will allow the users to select the different settings that a metronome has just by a click of a button.

Requirements

1. Provide a way to set the beats per minute (BPM) (Done)
2. Provide a way to select either straight beats or time signature output. (Done)
3. Provide a mechanical visual indication of BPM (Done)
4. Provide a way for the user to enable or disable audio output of straight beats or time signature beats (Done)
5. Show available battery power (Done)
6. Must be Portable (Done)
7. Include an Android-based remote output user interface. (Done)
8. Allow the user to store at least three setting profiles and be able to store the current session in case of power failure (Done)
9. Incorporated all techniques and peripherals learned in the 8 Labs from EE3376 (Done)

Design Choices and Reasoning

A normal metronome is able to provide the composer/musician a constant beat. This means that for our metronome we need to be able to provide a constant beat to the user with a desired speed. However, unlike a normal metronome that is not digital, our metronome needs to be able to notify the user an end of a measure beat depending on the time signature the user requires, this requires finding a way to keep the sound on sync with the servo motor in the desired bpm and time signature. Our system also has to have the ability to save the current session's settings as well as give the user the ability to save his/her own desired settings and be able to load them up by selecting said profile.

Beats Per Minute and Servo

In order to provide accurate representations of beats per minute to the user is to figure out how much time passes in between each beat. This is achieved using this formula, $\frac{1}{bpm} * 60s$,this will give us the amount of time in between beats for each beats per minute setting. So with that in mind, how can we move the servo at the right amount of time for each beat? Well we took in mind the settings in which the servo motor has to be controlled. The servo is controlled with a 50hz signal (appx. 20ms period) and the maximum duty cycle is 3ms and minimum is 1ms. So in order to change the speed of the servo we need to change the duty cycle with respect to the bpm. Using the formula, $\frac{\min duty\ cycle - \max duty\ cycle}{\left(\frac{\left(\left(\frac{1}{bpm}\right)*60s\right)}{PWM\_Period}\right)}$, gives us the value to increment/decrement the duty cycle every time the timer ISR is entered. With this formula using the respective bpm, we were able to find the value needed to increment/decrement the duty cycle which results in the servo arm arriving in the opposite direction in $\left(\frac{1}{bpm} * 60s\right)$ seconds. After finding the values for each bpm setting we wrote all of the values in an array named BPM_array. This way when the user wants to change the BPM the program will simply take the value needed by representing all 39 BPM setting with its own index from 0-38, 38 being 208 bpm and 0 being 40bpm.

Buzzer and Timer Signatures

For the time signatures all we needed is to be able to count the beats and change the pitch of the buzzer to notify the end of the measure. The user needs to also have the ability to have the buzzer give straight beat feedback or to turn off the buzzer. We chose to use a global variable named "buzzer" in which it will store 1 of 3 integers, 0 being off, 1 being time signature mode, and 2 being straight beats mode. By using a switch case statement, we can have the buzzer do what we want by simply changing the value of the integer when the user enters the command. For the timing of the buzzer we chose to do this in the same timer ISR where the duty cycle is being incremented/decremented. This way the buzzer will be on synch with servo and the respective bpm. We also decided to keep the code that controls the pitch of the buzzer in its own source file name "buzzer.c" and "buzzer.h" to keep the project's main file from being to cluttered in code that does not affect the overall project.

Flash Memory

The user will also have the ability to save up to 3 profiles and will have a saved session in case the device powers down for any reason. This is achieved by using the flash memory controller. For the emergency saving session had to write the data that is important for the metronome, in this case will be the index numbers for the bpm, time signature, and buzzer state. Saving these values is really easy since we can save them as bytes, for this feature we save them in segment 4 of the flash memory so it does not interfere with any addresses in memory in which may cause a failure (segment 8 would cause 0xfffe error). With segment 4 we can safely erase the segment for new settings. For the profiles we were thinking of using 1 or 2 segments, but due to time constraints we decided to use 3 segments for each individual profile to keep it simple.

Battery

For the battery we decided to notify the user if the battery is low by sending a simple message to the android app. This way we can send the user the message if and only when a certain threshold is reached. For example, if the adc10 register has a read value lower than 4 volts when using the 1.5v reference it will send a message to the app through uart and Bluetooth. This way we could keep the setup of the adc10 simple and straight forward. However, we do not want the adc10 to take a sample too many times as it will waste power. So we used the second timer (Timer1A) and using the wait variable we will be able to take only one sample every 10 seconds. This way we can would only that that sample if the wait variable has the value of 25, when that is reached the adc10 ISR checks if the value of ADC10MEM changed, if it didn't the battery hasn't changed and will exit the ISR.

App and communications

We can communicate with the metronome via Bluetooth with an app created with the MIT app inventor. When the app sends a command to the metronome the microcontroller will translate the character associated with the commands to index values for the BPM, time signature arrays and the buzzer value, which in the UART ISR it will handle the translation of these characters via a switch structure, the ISR will also write to the flash the new values in order for the session to be saved. This way we make this ISR the slowest as far as performance, but is making the necessary changes that the user made. Once it is done we will send to the user a "Changes done" prompt on the phone app. The TX ISR will also send any updates (battery) to the user by a notification "Battery Low" which will need a change of battery. When the Battery low is indicated, the micro will be receiving less than 4 volts of battery. The app will also display the current settings of the metronome as well as provide save profile feature using a list to choose the space in which to save the new profile.

System

The metronome will be made up of a SG90 tower pro servo used as the beats per minute mechanical visualization for the user. For the audio indicator we will be using the a piezzo buzzer which with a spate source file were able to make two different pitches, one for normal beats (Low note) and for the end of measure beat (High note). We display the information such as buzzer mode, time signature, and beats per minute we use a 16x2 LCD display. Using BPM for beats per minute, BZ for buzzer mode, TS for time signature we can display all of the current session settings in the LCD. The same information will be shown in the app as well however using the LCD to display all the info makes it more of a complete system. The battery will be a portable battery charger that can give 5v output. However, to show how the battery indicator works, we will be using either a circuit to change the amount we give to the microcontroller, or we will use individual AA batteries to demonstrate when a low voltage is received.

Project Logic Flowchart – Filename: “Logic_Metronome.pdf ”

Project Software Flowchart – Filename: “Software_Flow.pdf”

Implementation

Time Constraints

When it came down to working on the project as a group we needed to balance our group work with the amount of time each of us had to work on it. Two of the three members have work on a daily basis with this we needed to work around each other schedule in order to complete this project. The times that we usually met to work was always in the afternoon almost every day starting from when we were assigned the project. We also were able to use the time in our labs and on the weekends to get a good amount of work done on our project. Family issues and other important life matters also played a factor in the time constraints but we were able to work around each other's schedule to make it work.

Set Backs

With this project there were many setbacks that we as a group had to overcome in order to make the project work properly. As a group we had two major setbacks that required us to completely stop what we were working on and go back and debug the whole program. One was the fact that we needed the data info on the servo motor for the actual speed of each beat per minute and it caused many bugs in our code that we had to find and change to make the speed be up to the par that we needed. The second setback was in one of our last labs, our whole system would not turn on and would not give us any sort of feedback on to why none of the peripherals that we had were not turning on. To fix

that issue we needed to enable the interrupts by one simple command. Up until now no other setbacks have happened.

Organization

In the beginning of the project our organization was hectic and needed work. We ended up coming up with a priority plan that allowed us to be more organized with our time and work in order to create the project in order from greatest to least priority. We started with the servo motor and the buzzer to make sure the basic use of a metronome which is sound and visual aspect were working correctly. Next was the aspect of being able to transfer the data from the app to the device itself. With that we implemented the LCD to show the user what they were selecting as well as working on the ADC to measure the battery life for the user. Implementing each one of these in the order that we chose was what led us to succeed in the design that we chose and led us to complete the project with full confidence.

Testing

Servo Motor

When testing the servo motor, we wanted to make sure that the BPM was working well with the different speeds that we had calculated for the servo motor. We looked and made sure that the servo was moving at the full 180 degrees. Also with the servo motor we had to make sure that when it was on the specific BPM setting that it was also in sync with the Buzzer.

Buzzer

When testing the buzzer, we needed to make sure that the sound of the buzzer would match the sound of the Time signature if that option was selected. Also while testing for the time signature we made sure to test the buzzer to be in sync with the servo motor as close as possible so our design of the physical metronome was accurate and helpful.

LCD

When we tested the LCD, we needed to find out where exactly in the code would be the best fit for the LCD settings so when the user inputs command it is shown through the LCD directly. We tested different areas of the code but ultimately came up with the idea to place the LCD instructions in the UART receive interrupt vector in order to properly show correct info to the user upon selection of the inputs.

UART

When testing the UART we had to make sure that the Bluetooth device was working properly with the signal from our app. We tested to make sure that the RX and the TX were communicating and completing the job at hand when received the users command.

Maintenance

The maintenance on our system would be:

1. Battery maintenance for when the battery is low we will change it out or charge it up.
2. Servo motor maintenance for when the motor gives out or stops working we will need to be able to change out the motor or try to fix the problem.
3. LCD maintenance for when any of the components of the LCD go out we will need to be able to find the problem either with the connections or with the actual LCD.
4. Buzzer maintenance for when the buzzer goes out or no longer works we will need to be able to figure out the problem and make sure the buzzer comes back on for the project.

Lessons Learned and Project Conclusion

In this project we built a musical metronome that's aids a music instructor to direct a band or work in music in general. The metronome we designed is adjustable to beat and time signature to the user's needs. This will aid the user to different genres of music and music composition types. The design is portable so it has its own independent DC power source and can be positioned anywhere the user needs because it can be controlled wirelessly using Bluetooth from any android device. Working as a team of three members in this project, we learned to work in uniform to accomplish a common overall goal and many small goals in between. The main idea was to assign something to the team member who was strongest at a particular area of the project that needed to be completed. This was at times difficult because of some team members work schedules but we met during times that worked with everyone schedules. Moving on to the technical work of the project, we seem to have used many if not all of the concepts and code techniques that we used in all the previous labs. We used or course registers with respect to pins on the micro, interrupts, character arrays, an lcd with its respective code, a buzzer, timers, serial Tx and Rx, analog to digital (ADC) code commands, and switch cases. We basically are sending individual characters from our application to the microcontroller to give it an instruction and switch what it is doing at that particular moment. We also are sending individual characters from the micro to the phone to communicate battery level with our ADC code. This will be analog rage of approximately 4V to 5.5V. If we go below this threshold the micro will not operate as it is not being supplied with enough power. One thing about taking on a project like this is having patience for code that does not do what you want it to do. We had to also figure out that the HC-06 Bluetooth module that we are using was having some issues connecting to our Android devices properly so we had to play with different applications that will pair with the HC-06. This was a time consuming experience in the case that we had to spend time figuring out what the HC-06 liked being paired with even through our BAUD rate matched the transmitter and receiver. Overall this was a great team project because it was able to give us the best out of every team member knowledge and insight. Our team organized goals for the project well and delivered throughout the whole two weeks we worked on this project together.

Project Conclusion

In this musical metronome project, we learned how to work as a team and put each team mate strongest skills at work for each of our short term goals and deadlines. This metronome by far challenged us the most than any of our part labs. We were able to setup out state machine to receive instructions from a terminal or android application and execute them in terms of moving a servo from -180 to +180 degrees repeatedly to act like a musical metronome. Our LCD screen is also functional with displaying both beats per minute (BPM) and time signature (TS). We used code and header file from lab four to do this. We also used code like that similar to lab five which utilized pulse width modulation (PMW) to move the servo back and forth at a set pulse using clocks. One thing with our code and the servo motion is that it had to move back and forth at a certain BPM with meant that we had to calculate this with what we knew about our internal clock and the user of other clocks we used in our previous labs. We initially were testing our design via USB as a serial connection (TX/RX) form of transmission with a command terminal like PUTTY or one used with iOS called Cool Term. All the commands implemented on our code worked successfully after we could figure out how to make the switch cases on the code work with each character being received on the microcontroller. After we received our HC-06 bluetooth receiver, we were able to successfully test our design wirelessly. We coded an Android application to work on Bluetooth but we were having issues using Android Studio to work with Bluetooth so we switched to the MIT app inventor. For the ADC, we setup the code so that it sent characters to the Android App or terminal within certain thresholds of voltage of about 4 to 5.5V. All an all this was a great project and we did get most of the points for all the parameters given in the design instructions with only minor details either not working or not implemented in the design but given the timeframe we had to complete the project I think we could have easily scored a full 100% with a little bit more time.