**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, ,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, , 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 econds. 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”