EC450 Project Report: Light Conductor

Team 4

Reva Scharf

Caitlin Manes

# Goal and Design

The goal of our project was to create a system that would allow the user to conduct music through the use of light. We designed our system to use two photoresistors, one to control the music’s tempo and one to control the dynamics (volume). We also included a calibration mode for the photoresistors to enable the system to be functional in different environments with varying levels of ambient light.

On the first start up of our system, it will automatically enter the first stage of calibration mode. This is indicated with the red on-board LED on. The red LED indicates a request for a low light calibration, which is recorded with a push of the on-board button. After this the red LED turns off and the green LED turns on, indicating a request for a medium light level calibration. Again, this level is recorded with a button push. After that, the red LED turns back on and both red and green LEDs are illuminated. This indicates a request for a high light level calibration. Once again, the button push records the level. After this, all LEDs turn off and calibration mode is finished.

When the system is started up again after its initial boot, it will not enter calibration mode automatically, but rather use stored values in the flash memory from a previous calibration. However, if the user desires to recalibrate the system, calibration mode can be entered with one push of the on-board button and it will cycle through the process described above.

At this point, if the user starts “beating” a tempo over the tempo photoresistor, the song will begin playing at the tempo the user inputs. If they speed up or slow down their conducting, the song will follow their tempo. If the user ceases conducting, the music will continue playing at the last tempo given by the user. The volume of the system will be at a maximum with no hand over the photoresistor (i.e. at full ambient light level) and will decrease with lower light levels.

# Implementation

The following is a detailed look into how we implemented the various functions of our project:

**Calibration Mode:**

Calibration mode makes use of flash memory, a button, and two LEDs. Everything in the calibration mode is generally based around the calib\_state variable, which tells us which state of calibration mode we are currently in. When calib\_state is 0 we are not in any sort of calibration mode. When calib\_state is 1 we are calibrating the low light level reading, 2 is a medium reading, and 3 is a high reading. All button handling is done in the watchdog timer interrupt handler (WDT handler). The LEDs are turned on or off corresponding to their calib\_state in the WDT handler as well. Once all values (which are being read in through the analog to digital converter (ADC) described below) are recorded in calibration mode, we set a state variable indicating that the calibration has finished. After that we are able to write our new data to flash memory.

Four variables are set up within the information D segment in the flash memory: isCalibration, lowCalib, medCalib, and highCalib. The first variable, isCalibration, operates as a Boolean 0 or 1 indicating if there is calibration data already stored in the flash memory. This is what lets us go straight into calibration or skip it depending on the data available. The other three variables predictably hold the low, medium, and high calibration data values. In order to write these values to flash memory, a sequence of steps must be taken. Before new data can be written to flash, first everything in that memory segment must be erased. This is done with a function called eraseD. Another method was created called writeDword, which writes a value to a specified address in memory segment D. Finally, so that erasing is not overlooked, a routine called updateData was created which erases the memory segment and then writes the values of the four variables at once so nothing is skipped. When our calibration mode has finished, updateData is called with the values for the four variables (a 1 for isCalibration and the corresponding light readings for the low, medium, and high calibrations).

**ADC Multiple Inputs:**

Two ADC inputs are used: input channels A0 and A1. They are pins P1.0 and P1.1, respectively. For the setup, a sequence of channels is used, and two blocks are transferred per conversion. The ADC10IE interrupt is then enabled, and ADC10 is turned on. The MSC bit is also used to setup multiple sample conversion. The ADC conversion is implemented using 3 variables. There is an updates variable, a last\_updates variable, and an array of length 2 called adc\_vals. The ADC values in adc\_vals will be between 0 and 1023, inclusive. There is a start\_conversion() function that starts the ADC conversion to be put into adc\_vals if ADC is not already converting. The updates variable is incremented by one after the ADC conversion has started. Then in the Watchdog Timer (WDT) interrupt handler, if updates is greater than last\_updates, the adc\_vals are used to control tempo and change the volume of the speaker. The variable last\_updates is then set to updates. The function start\_conversion() is also called so updates will be greater than last\_updates in the future. This control structure works because in the main() function, updates and last updates are set to zero, and start\_conversion() is called.

**Digital Potentiometer SPI Communication and Speaker Volume:**

The digital potentiometer (4131-103) uses SPI for communication. SPI communication is set up using pins P1.4 for the clock, P1.2 for Slave In Master Out (SIMO), and P2.0 for the select bit. Whenever a value from the ADC is going to be sent to the digital potentiometer for the volume, it has to be converted to a value between 0 and 128, inclusive, since these are the digital potentiometer’s minimum and maximum values. The function getValueForPot() is used to make this conversion. First, bounds are set. If the ADC value is higher than the high calibration, it is set to the high calibration value. If the ADC value is lower than the low calibration value, it is set to the low calibration value. Then, the following equation is used to convert the ADC value from between 0 and 1023 to between 0 and 128, both inclusive:

The lightToPotFactor is calculated after calibration is performed. First, a lightRange variable is calculated:

Then, the lightToPotFactor is calculated:

Note, some of these variable names were changed for clarification or formatting from very similar names in the actual code. For example, lightRange is actually light\_range and highCalibrationValue is actually highCalib. Once an appropriate value is calculated for the digital potentiometer, it is sent using the following method. First, the select bit is set to low. The digital potentiometer is then sent the value of 0. The code waits for the TX buffer to be empty next. Once the TX buffer is empty, the converted ADC value is sent to the digital potentiometer. The code waits for the TX buffer to be empty and for TX to complete after. The select bit is then set to high. This completes the communication to the digital potentiometer, which uses that value to set the volume for the speaker.

**Speaker Frequency:**

The speaker uses Timer A to generate an appropriate frequency for a specific note. It uses SMCLK as its source clock with a clock divider of one, and it is in up mode. The speaker is set to the first note in the Jeopardy\_Melody array and starts out off. Pin P1.5 is used to output the speaker frequency. Whenever a new note is selected in the WDT interrupt handler, the TACCTL0 is or’d with OUTMOD\_4 if the next note is actually a note to turn the timer on. If the next note is a rest, TACCTL0 is set to 0 to turn the timer off.

**Beat Detection:**

There are two states the beat ADC can be in: low\_state and high\_state. The low\_state variable indicates the hand is between the low and medium values determined in calibration. The high\_state indicates the hand is between medium and high values determined in calibration. Two other variables are used: reached\_low and reached high. If the ADC value is less than or equal to lowCalib + 2, reached\_low is set to true. If the ADC value is greater than or equal to highCalib, reached\_high is set to true. To calculate the beat, the state and the reach\_high and reached\_low values are used. If the state is high\_state, and the ADC value is greater than medCalib and reached\_low is true, state is changed to high\_state, reached\_high is set to false, and a beat\_count variable is incremented to register a down beat detection. If the state is low\_state, and the ADC value is less than medCalib and reached\_high is true, state is set to low\_state and reached\_low is set to false. This allows a beat to be detected when the hand begins to come back up.

**Note Length Calculation:**

We calculate the length of a note based off of the number of WDT interrupts that occur between down beat detections. The variable quarter\_count is incremented in each WDT interrupt and when a beat is detected, this value is stored in WDT\_per\_quarter. Our array of note durations is based off of a quarter note as 1 (i.e. a half note is 2, an eighth note is .5, etc), letting us use WDT\_per\_quarter as a multiplier against this array to set the tempo.

# Project Success Assessment

For the most part we consider our project to have been a success. It met nearly all of the goals we set out to achieve. Our system does successfully enable the user to control the tempo of a song via conductor-like movements aimed at a photoresistor. However, we wish our volume control had been more effective. Due to hardware oversights, our system did not play as loudly as we would have liked, making the volume very hard to change in minimal amounts.

# Next Steps

During our project demonstrations, it was made clear to us that there were a couple of ways to have fixed our volume issues. One method we might consider is using pulse width modulation (PWM), instead of our digital potentiometer, to control the volume of our system. Another method continues use of the digital potentiometer, but with potential modifications. We were made aware that our digital potentiometer chip had a low wiper resistance of 75Ω and we were only using a 4Ω speaker. This accounted for our volume issue. Using a larger speaker in conjunction with reducing the low wiper resistance would enable us to have a louder volume output and thus be able to hear the changes in volume more easily.

# Division of Work

Our code was written modularly and then brought together at the end. Reva was responsible for the flash memory and calibration components of the project. Caitlin was responsible for the ADC, digital potentiometer, and speaker components of the project. Bringing the code together was done as a group as well as all debugging and finalizing aspects.

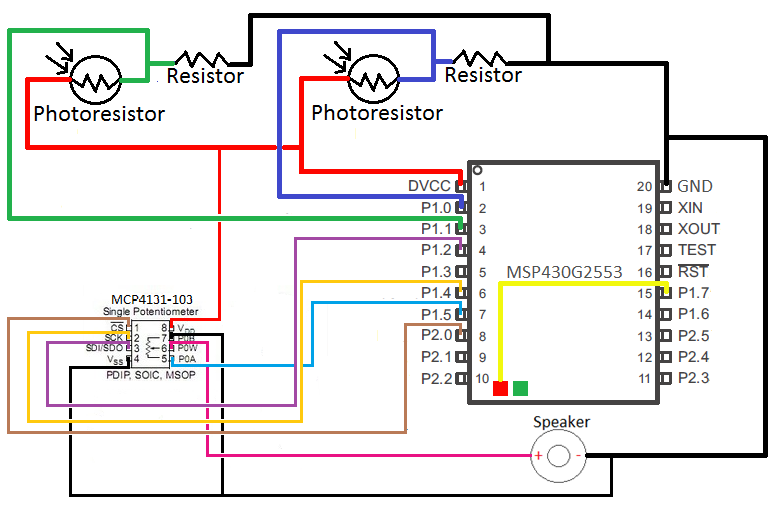


Figure 1: Hardware Schematic

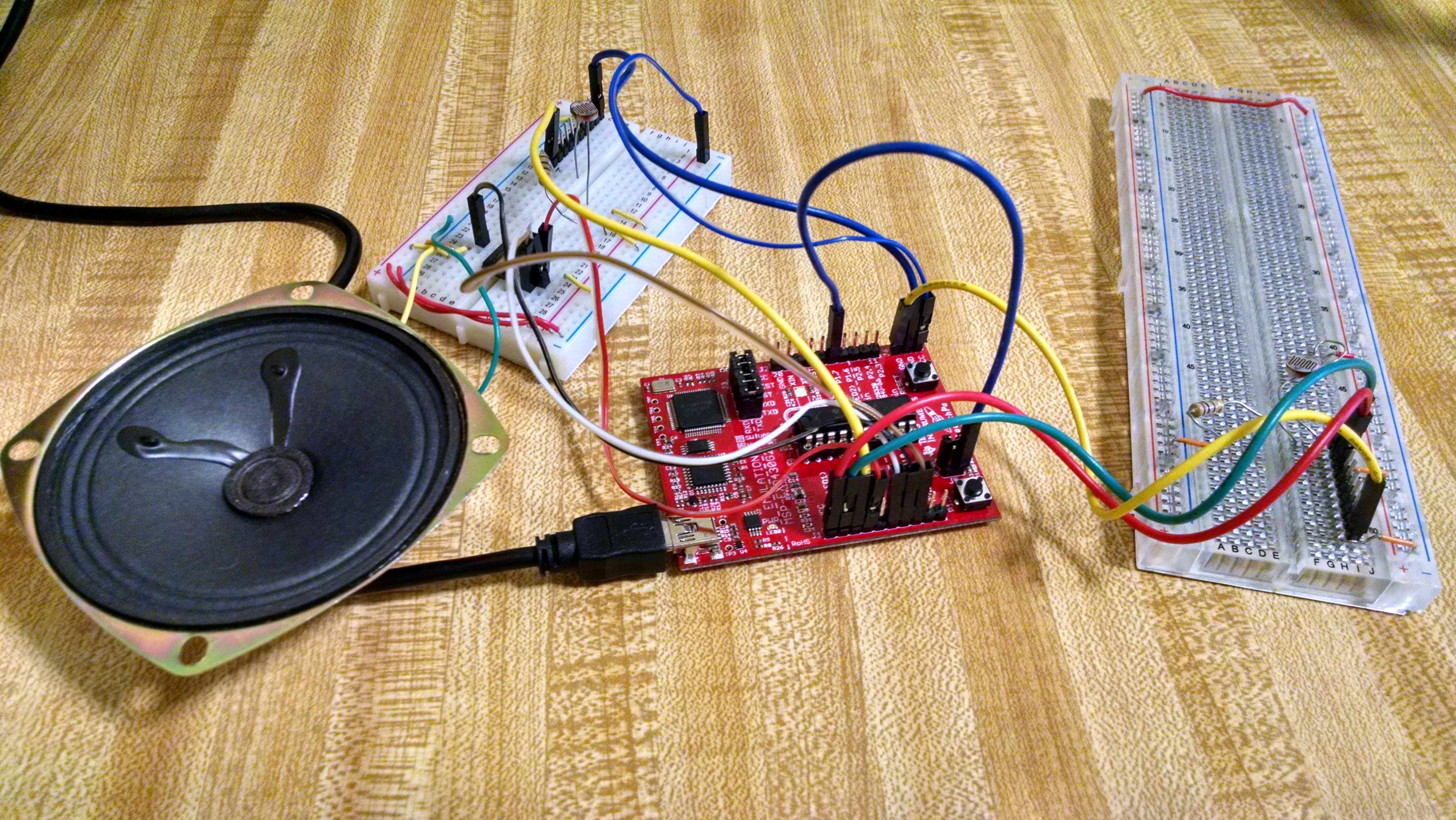


Figure 2: Physical Hardware Setup

\*Note: Due to poor formatting in Word, please see light\_conductor.c and musical\_defines.h for full code.