Patrick Austin

CPE 301 – 1104

Assignment # 7

10/24/2016

Assignment description:

In this lab we built a circuit and wrote code to drive a speaker by generating certain note frequencies corresponding to serial keyboard input on the PC the board was connected to. For example, pressing ‘a’ on the keyboard sent serial data to the board, whose code then generated a square wave on an output pin at the frequency corresponding to musical note ‘A’, which was wired up to the speaker which then produced the note. We also used the function generator to produce some of the musical note frequencies, which we then manipulated and compared to the speaker output.

Problems encountered:

My board/code worked without much difficulty, generating all the expected notes from the speaker. It took a few minutes of review and recollection to get fluent with the use of the wave generator again, since we hadn’t really used it since the first lab, but once we had hooked things up correctly we got the results we expected.

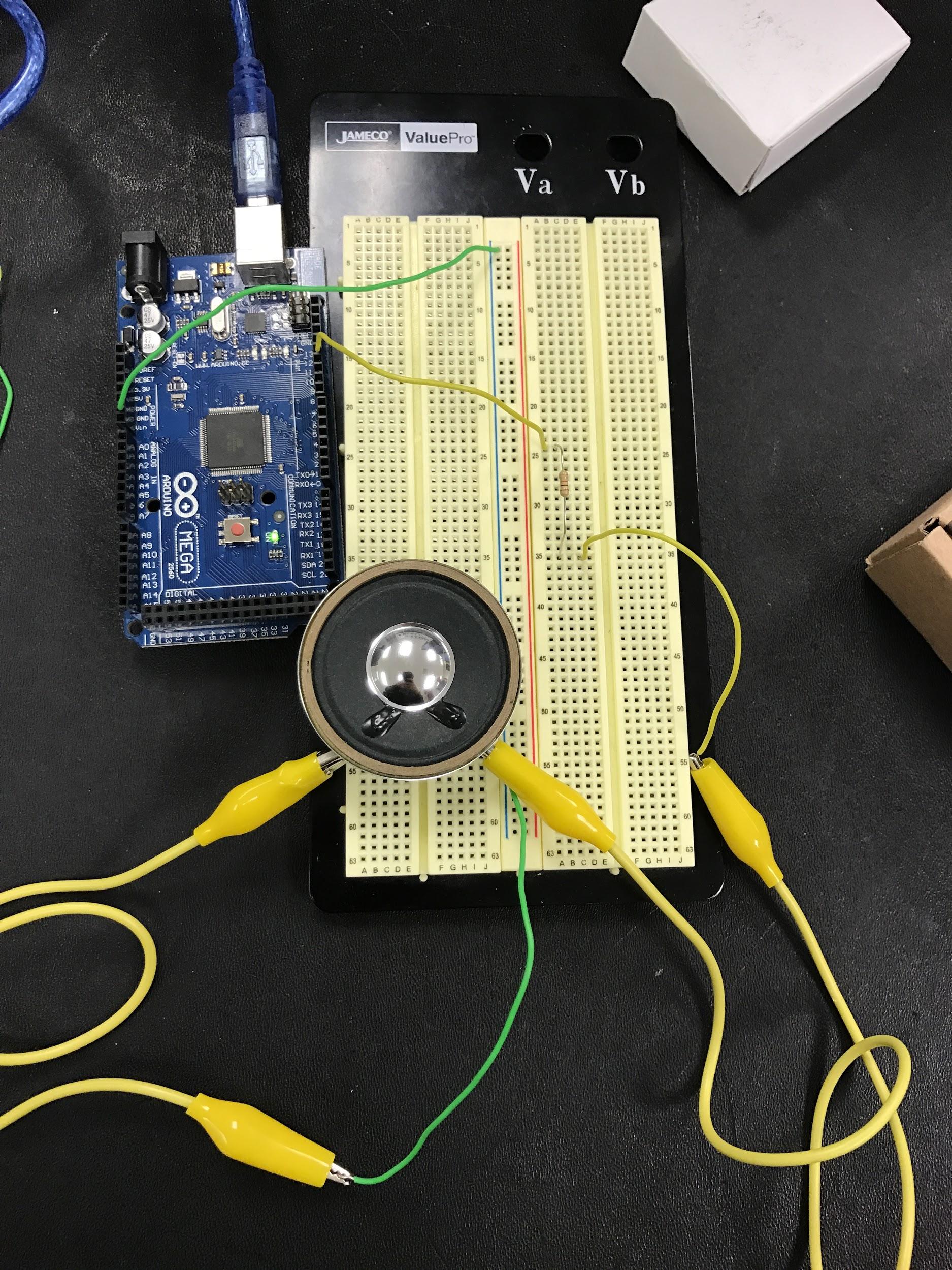
Lessons learned:

Reviewed basics of Arduino GPIO, timer usage, and manipulating the function generator. Did more breadboarding, although in the case of this lab the board wasn’t too complicated. Also got more experience in using Putty and serial communication on the Arduino. Goes without saying that we hadn’t used speakers in these labs before, and it was pretty cool (and conceivably useful) to be able to manipulate one with a microcontroller.

Description of completed lab:

The circuit and code worked together as expected, and we were able to verify the generation of the specified notes. I implemented regular notes with lower case inputs from the keyboard and sharps with upper case (ie, ‘a’ on the keyboard produced note A, ‘A’ on the keyboard produced note A sharp.)

Completed circuit:



The circuit ran this program (extra credit implemented):

//Patrick Austin

//CPE 301 Lab 7

//Revision Number 1

//Revision date: 10/24/2016

//hardware pointer declarations

volatile unsigned char\* myPortDDRB = (unsigned char\*) 0x24;

volatile unsigned char\* myPortB = (unsigned char\*) 0x25;

volatile unsigned char\* myTIFR1 = (unsigned char\*) 0x36;

volatile unsigned char\* myTCCR1A = (unsigned char\*) 0x80;

volatile unsigned char\* myTCCR1B = (unsigned char\*) 0x81;

volatile unsigned char\* myTCCR1C = (unsigned char\*) 0x82;

volatile unsigned int\* myTCNT1 = (unsigned int\*) 0x84;

volatile unsigned char\* myTIMSK1 = (unsigned char\*) 0x6F;

//global variable for storing serial input data

byte byteRead;

//function prototypes

void myDelayMicroSeconds( unsigned long microSeconds );

unsigned long getDelay( unsigned char input );

void setup()

{

\*myPortDDRB = \*myPortDDRB | 0x40; //enable output on specified pin, pin 6

\*myTCCR1A = 0; //zero out timer controls to enable normal mode, timer off

\*myTCCR1B = 0;

\*myTCCR1C = 0;

\*myTIMSK1 = 0; //zero out interrupt register to disable interrupts

Serial.begin(9600); //enable serial at 9600 baud

}

void loop()

{

if (Serial.available()) //if serial data has been sent...

{

byteRead = Serial.read(); //store the data

Serial.write(byteRead); //echo the data to console

}

unsigned long delayTime = getDelay(byteRead); //get frequency to match the input

\*myPortB = \*myPortB | 0x40; //enable output pin

myDelayMicroSeconds(delayTime); //wait 1/2 period

\*myPortB = \*myPortB & 0xBF; //disable output pin

myDelayMicroSeconds(delayTime); //wait 1/2 period

}

//function implementations

void myDelayMicroSeconds( unsigned long microSeconds ) //reused from homework 5

{

\*myTCCR1B = \*myTCCR1B & 0xF8; //set timer to off

//calculate preload value for the timer. need preload value such that the time needed //for the timer to go from the preload value to raising the overflow flag will take //'microSeconds' microseconds. this implementation uses the 1 prescaler.

\*myTCNT1 = (unsigned int) (65536 - (long) (microSeconds / .0625));//get preload value

\*myTCCR1B = \*myTCCR1B | 0x01; //enable timer with 1 prescaler

while ( (\*myTIFR1 & 0x01) == 0 ) //until the overflow flag is raised, do nothing

{}

\*myTCCR1B = 0; //delay complete, turn off the timer

\*myTIFR1 = \*myTIFR1 | 0x01; //reset the oveflow flag by writing a 1, finished

}

//for a char corresponding to a note (convention: regular notes lower case, sharps //upper case), return an appropriate timer delay value (ie 1/2 the period) in //microseconds

unsigned long getDelay ( unsigned char input )

{

if ( input == 'a' ) //to generate a 440hz square wave,

//go high 1/2 period, low 1/2 period.

return 1136; //For example 1/440 = 2.27272, half that is 1.1363,

//so want approx 1136 microsec delay.

//myDelayMicroseconds function will take this value

//divided by .0625 to find the number of

//ticks that the timer should wait.

else if (input == 'A' ) //repeat this process for the other notes

return 1072;

else if (input == 'b' )

return 1012;

else if (input == 'c' )

return 956;

else if (input == 'C' )

return 903;

else if (input == 'd' )

return 851;

else if (input == 'D' )

return 801;

else if (input == 'e' )

return 759;

else if (input == 'f' )

return 716;

else if (input == 'F' )

return 676;

else if (input == 'g' )

return 668;

else if (input == 'G' )

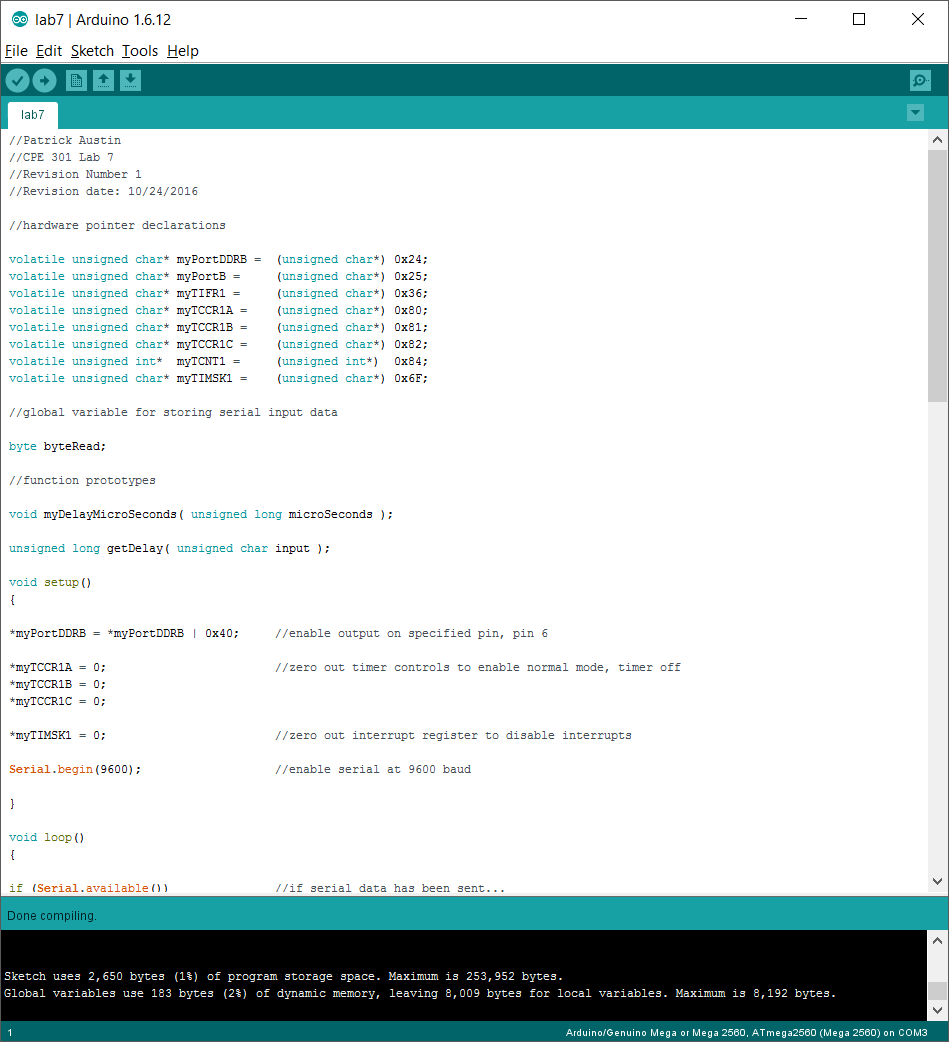
return 602;

else

return 0; //if no input, arbitrarily generate no delay.

}

Picture of code compiling successfully:



Problem 3 question: Using the signal generator to replicate a note, we observed that corresponding sounds were different from what the speaker connected to the board produced. And moreover the sound changed when the generated wave was switched between square, sine, and jagged waveforms. This result made sense. A sine wave is a very different shape than a square wave, producing many more voltage levels. Moving smoothly between a voltage max and min will sensibly produce a different sound than a harsh square wave fluctuation between only the minimum and maximum voltage.