Lab 2: Digital I/O and Timing of Outputs

Nakseung Choi, 1572578
Kejin Li, 1978130
01-May-2022
Assignment: ECE474 Lab 2

Introduction:

In this lab, we learned to drive analog inputs and digital outputs by directly setting them up with low-level hardware functions (firmware programming.) Using these digital outputs, we were able to flash an 8x8 matrix of LEDs and make a tone on the speaker at different frequencies. We also learned to use 16-bit timer/counter (CTC mode) to compare a set value N (OCRnA) with TCNTn running at different periods of time with a different pre-scale value. With this compare mode, we were able to play a sequence of tones at different frequencies. Finally, we learned how to build a scheduler to manage different tasks and timings.

Methods and Techniques:

For part 2.4, we used several different techniques to verify the functionality of our program. To begin, the concept of CTC mode and pre-scaling was confusing at first. In order to understand what these meant, we drew the entire block diagram and tried to understand the concept of 16-bit counter/timer. Then, we calculated a N value for 400 Hz with a pre-scale of 256 (200kHz) and tried to run the speaker with the N value, which was about 78. Using an oscilloscope, we were able to verify that the speaker operates at a frequency of 392Hz, which was reasonably close. However, the N value we calculated was a decimal number, not an integer. Therefore, we calculated N values with a different pre-scale and then we were able to determine that with a pre-scale of 8, N values became an integer. Again, we verified the frequencies with an oscilloscope, but the frequencies were still off by 1-5 Hz. We thought that this might be a hardware issue, such as a long wire giving more resistances to the frequencies. We verified this with our TA and confirmed that the frequencies were reasonably close.

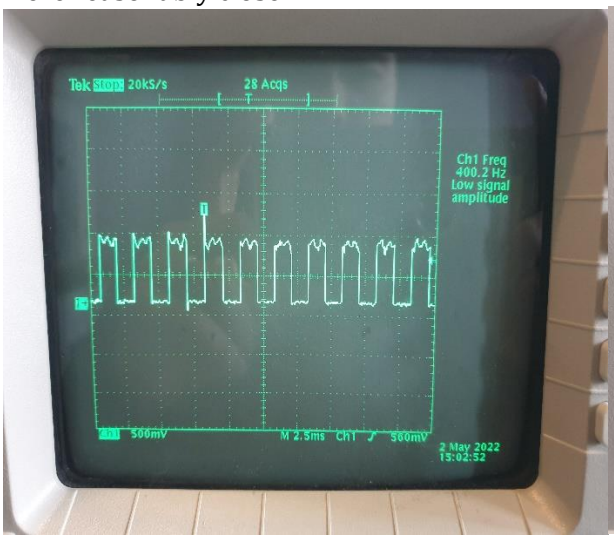

Figure 1. 400Hz with a pre-scale of 8.

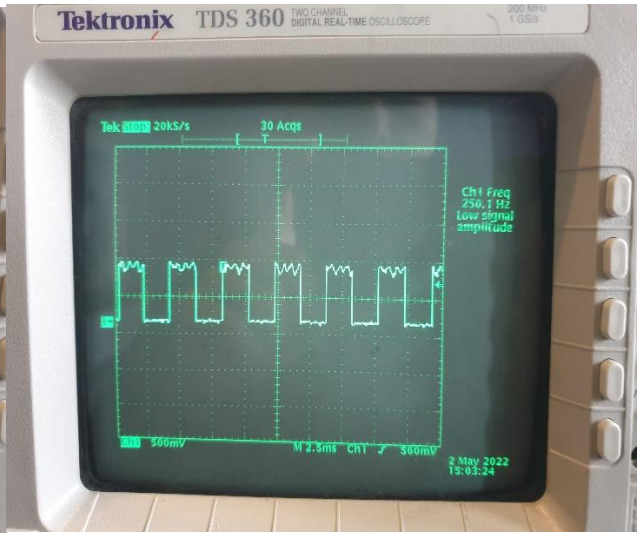

Figure 2. 250Hz with a pre-scale of 8.

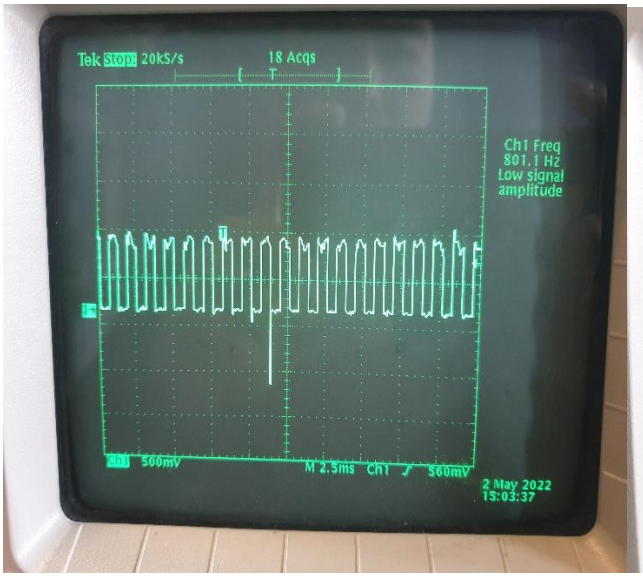

Figure 3. 800Hz with a pre-scale of 8.

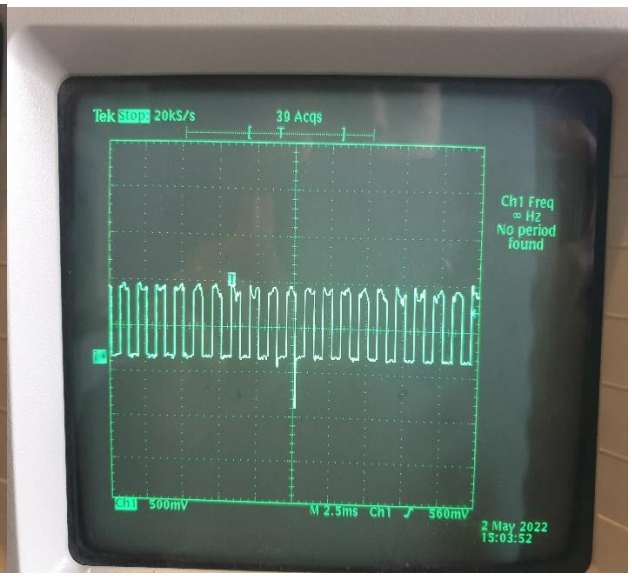

Figure 4. Pause for one second.

For part 4, Serial.print function was employed to verify that we get analog input values from the joystick. The same technique was employed again to verify the values scaled down to the range of 0 to 7 for columns and rows of the 8x8 LEDs. This is shown below:

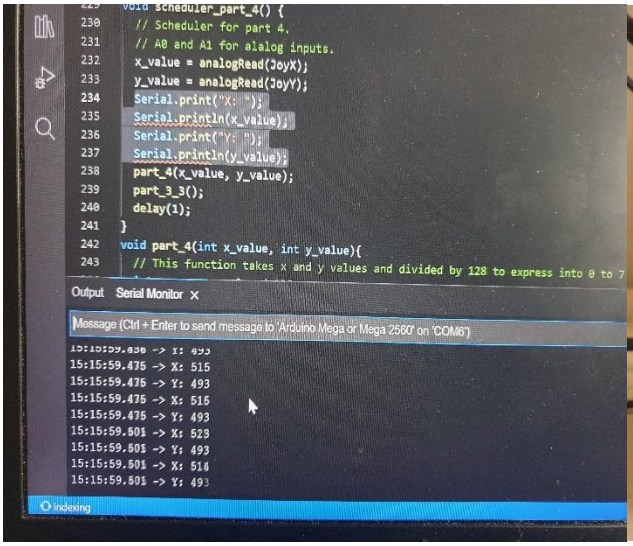

Figure 5. X and Y values from analog inputs of Joystick

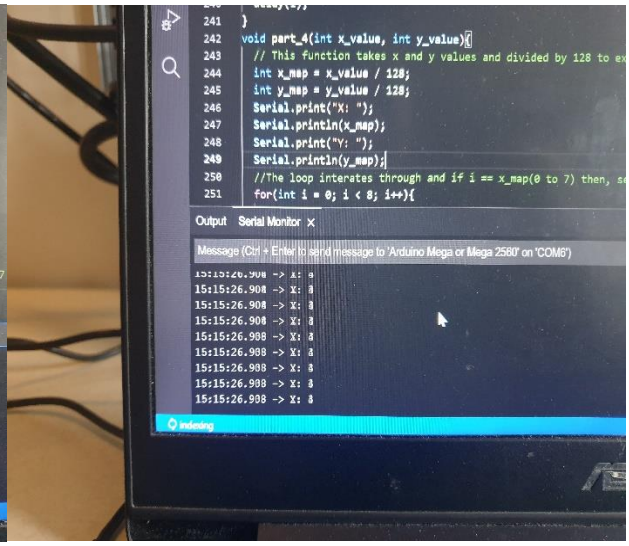

Figure 6. Converted values from 0 to 7 for 8x8 LED

Experimental Results:

Part 1.2 was fairly straightforward because have performed it in lab 1. In order to enable Pin 47, 48, and 49 with firmware programming, we looked at the Pinout-Mega256 sheet and determined that PL2, PL1, and PL0 needed to be activated for these pins. Thus, we enabled the pins by giving “1” to the corresponding bitfields (e.g. DDRL |= 1 << DDL2, and DDL1 and DDL0.) Then, we also set those pins to be output mode by applying 1 to the bitfields (e.g. PORTL |= 1 <<DDL2, DDL1, and DDL0.) Setting up the same hardware with 220Ω from lab 1, we were able to flash each LED for 333ms.

For part 2.4, this took us a long time to understand. We had to draw the block diagram and the timing diagrom for TCNTn and OCnx on the white board and discussed it about one hour to fully

understand how 16-bit timer/counter works. ATmega2560 runs at 16MHz with a pre-scale of 1(no pre-scale.) This means that it reads the data every 62.5 nano period per seconds. Using a pre-scale, we can decrease this period (frequency.) For instance, with a pre-scale of 8, the system operates at a frequency of 200kHz, which is equal to $5\mu s$. If we run a speaker at a frequency of 400Hz, the N value has to be $\frac{5000}{2} = 2500$ for a half cycle to toggle. Using this calculation, we also found N values for 250Hz (4000) and 800Hz (1250,) we first set TCCR4A and TCCR4B to be 0. This cleared the previous data stored in the control registers. Because TCCR4B does not have a function for CTC mode, we had to set CTC mode using TCCR4A and set waveform generation mode and a pre-scale of 8 by giving 1 to the bitfields (e.g. TCCR4B |= 1 << CS41 | 1 << WGM42.) Next, we enabled Pin 6 to be output mode (DDRH |= 1 << DDH3 and PORTH |= 1 << DDR3.) We also set OCR4A to be equal to the N values with a time delay of 1 second. This would run the speaker at different frequencies for one second.

For part 3.1, we created a function called part_3_1, which calls the same functions from part 2.4 at different time periods. We also had to create a function called sleep. This function turns off all of the LEDs and speakers by setting the corresponding bitfield of the PORTs to be "0" for one second. All that part_3_1 does is to call these functions and run them at different time periods. Therefore, LEDs run for two seconds, the speaker runs at different frequencies for four seconds, and pause(sleep) for one second.

For part 3.2, we created a function called scheduler_part_3_2. In this function, we created an if statement that compares a variable called "time" and the total time called "delay_time_part3." With a delay time of 1ms, the time variable increases by one increment every 1ms and the total time of this program lasts for 16 seconds. Therefore, we set this if condition as (time % delay_time_part3 == 0 || time % delay_time_part3 < 1000). This will run the LED_part_3_2 function inside the if statement for one second. We set this function to run for two seconds, and then (time % delay_time_part3 == 2000 || time % delay_time_part3 < 6000) such that speaker_part_3_2 runs from 2 seconds to 6 seconds. Next, (time % delay_time_part3 == 6000 || time % delay_time_part3 < 16000) will run LED_part_3_2 and speaker_part_3_2 concurrently for 10 seconds following one second of sleep. Finally, an if statement checks if the time variable is equal to 16000 and set to be 0 to repeat these steps again.

For part 3.3, using the given array of different frequencies, we created an unsigned int variable called "time" in a function. This increases by one increment every 1ms and then set OCR4A to be equal to melody[time/150] such that it plays the tone for the "time/150" amount of time. Then, it plays the next tone until time / 50 is equal to the size of the array (sizeof(melody)/sizeof(melody[0]). We also defined a variable for the total time that is used in if statement conditions to compare with "time" increment. For example, if the time is equal to 0, LEDs start to flash for two seconds. Next, starting from 2000ms until time is equal to 2000ms + (number_of_tones x 150ms), it plays "Mary Has a Little Lamb." The next step is to turn both LEDs and "Mary Has a Little Lamb" on for 10 seconds. Thus, if we set the time to be from 2000ms + (number_of_tones x 150ms) to (12000ms+ number_of_tones x 150ms), both LEDs and the song stops at the total time.

For part 4, DDF0 and DDF1 were employed for analog inputs, and int x_value and y_value took these inputs from a joystick. These two values were from 0 to 1023 but converted into 0-7 and stored in int variables called x_map and y_map to light up 8x8 LEDs based on the coordinates. Using a for loop with a condition, (int i = 0; i < 8; i++), an if statement compares "i" with "x_map," and if i ==

x_map, the same number of y_map(column) gets 1 so that the LED in the same x and y coordinate turns on. The hardware for this program is shown below:

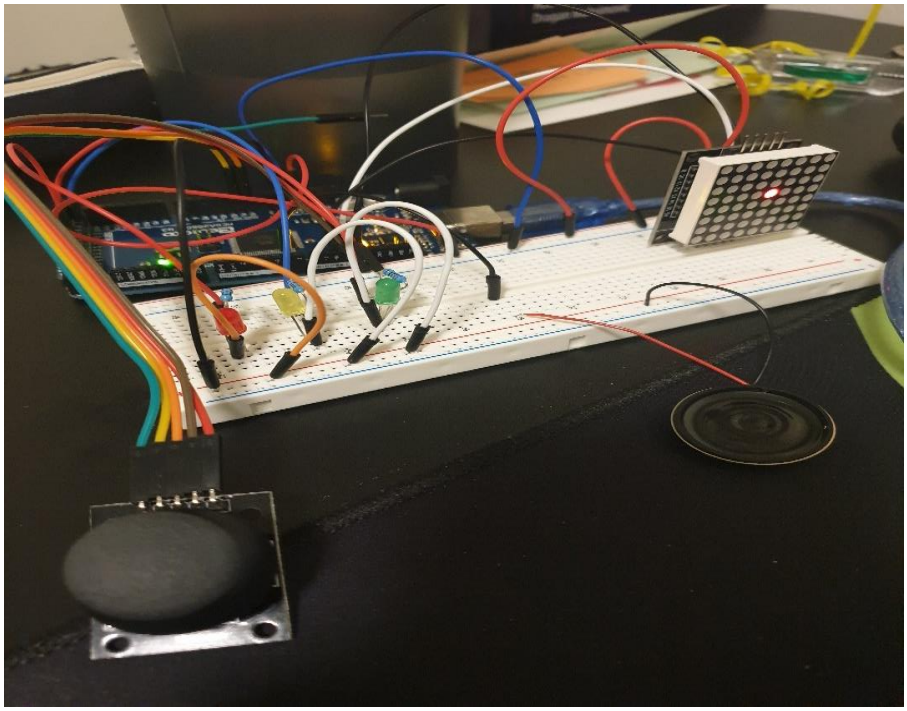

Figure 7. Overall hardware for Lab 2

Code Documentation:

Commented codes are shown below:

```
#include <avr/io.h>

// Part 1 //////////////////////////////////////////////
#define PIN47 47
#define PIN48 48
#define PIN49 49
////////////////////////////////////////////////////////

// part 3.3 ////////////////////////////////////////////
#define NOTE_c 3830 // 261 Hz
#define NOTE_d 3400 // 294 Hz
#define NOTE_e 3038 // 329 Hz
#define NOTE_f 2864 // 349 Hz
#define NOTE_g 2550 // 392 Hz
#define NOTE_a 2272 // 440 Hz
#define NOTE_b 2028 // 493 Hz
#define NOTE_C 1912 // 523 Hz
#define NOTE_R 0
```

```
int melody[] = {NOTE_e, NOTE_R, NOTE_d, NOTE_R, NOTE_c, NOTE_R, NOTE_d, NOTE_R, N
OTE_e, NOTE_R, NOTE_e, NOTE_R, NOTE_e, NOTE_R, NOTE_d, NOTE_R,
NOTE_d, NOTE_R, NOTE_d, NOTE_R, NOTE_e, NOTE_R, NOTE_g, NOTE_R, NOTE_g, NOTE_R, N
OTE_e, NOTE_R, NOTE_d, NOTE_R, NOTE_c, NOTE_R, NOTE_d, NOTE_R, NOTE_e,
NOTE_R, NOTE_e, NOTE_R, NOTE_e, NOTE_R, NOTE_e, NOTE_R, NOTE_d, NOTE_R, NOTE_d, N
OTE_R, NOTE_e, NOTE_R, NOTE_d, NOTE_R, NOTE_c, NOTE_R, NOTE_c};
int note_count = sizeof(melody)/sizeof(melody[0]);
// part 3.3 end ////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

// Part 4 //////////////////////////////////////////////////////////////////////
/////////
#define JoyX DDF0
#define JoyY DDF1
#define OP_DECODEMODE  8
#define OP_SCANLIMIT   10
#define OP_SHUTDOWN    11
#define OP_DISPLAYTEST 14
#define OP_INTENSITY   10

void spiTransfer(volatile byte row, volatile byte data);

byte spidata[2];
int x_map, y_map, x_value, y_value;

int DIN = 12;
int CS =  11;
int CLK = 10;
// part 4 end //////////////////////////////////////////////////////////////////
/////////

/// General time Setting ////
int delay_time_part1 = 333; // LEDs delay
int delay_time_part2 = 1000; //Speaker delay
int delay_time_part3 = 16000; // part 3.2 total time
int delay_time_part3_3 = 12000+(note_count*150); // part 3.3 total time
//////////////////////////////

void setup() {
  Serial.begin(9600);
  // Part 1.2 ////////////////////////////////
  pinMode(PIN47, OUTPUT);
  pinMode(PIN48, OUTPUT);
  pinMode(PIN49, OUTPUT);
```

```
  ////////////////////////////////////////

  // Part 1.4.2 ///////////////////////////
  DDRL |= 1 << DDL2; // Bit 2 of DDRL = Pin 47
  DDRL |= 1 << DDL1; // Bit 1 of DDRL = Pin 48
  DDRL |= 1 << DDL0; // Bit 0 of DDRL = Pin 49
  ////////////////////////////////////////

  // Part 2.4 ////////////////////////////
  // Initialize them to zero. (IMPORTANT)
  TCCR4A = 0;
  TCCR4B = 0;
  TCCR4C = 0;
  ////////////////////////////////////////
  TCCR4A |= (1 << COM4A0); // Enable compare output mode with TCCR4A because TCCR
nB does not have compare output mode.
  TCCR4B |= (1 << CS41); //| (1 << CS40); // set the precale to 8.
  TCCR4B |= (1 << WGM42); // Enable Waveform Generation Mode (set WGM42 to 1)
  TCNT4 = 0;            // counts from 0
  DDRH |= 1 << DDH3; // Enable PIN 6
  PORTH |= 1 << DDH3; // Set portH as an output.
  // Part 2.4 end ////////////////////////////////////////////////////////////////
////////////////////////////////////

  // Part 4.1 start //////////////////////////////////////////////////////////////
////////////////////////////////////
  DDRB |= 1 << DDB6 | 1 << DDB5 | 1 << DDB4; // enable pin 10(CLK), pin 11(CS), p
in 12(DIN).
  PORTB |= 1 << DDB5; //set CS to HIGH.
  DDRF |= 0 << DDF0 | 0 << DDF1; //Analog Input Setting. (A0 and A1)

  spiTransfer(OP_DISPLAYTEST,0);
  spiTransfer(OP_SCANLIMIT,7);
  spiTransfer(OP_DECODEMODE,0);
  spiTransfer(OP_SHUTDOWN,1);
  ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
}
void loop() {
  ////**** Uncomment to run. ****////
  //part_1_2();
  //part_1_4();
  //part_2_4();
  //part_3_1();
  //scheduler_part_3_2();
```

```
  //scheduler_part_3_3();
  //scheduler_part_4();
  ////////////////////////////////
}
void part_1_2(){
  //Each LED flashes for 333ms, total of 1 second.
  digitalWrite(PIN49, LOW);
  digitalWrite(PIN47, HIGH);
  delay(delay_time_part1);
  digitalWrite(PIN47, LOW);
  digitalWrite(PIN48, HIGH);
  delay(delay_time_part1);
  digitalWrite(PIN48, LOW);
  digitalWrite(PIN49, HIGH);
  delay(delay_time_part1);
}
void part_1_4(){
  //LEDs run in the same pattern as part 1.2,
  //but using direction registers and port output.
  PORTL |= 1 << DDL2;
  delay(delay_time_part1);
  PORTL &= !(1 << DDL2);
  PORTL |= 1 << DDL1;
  delay(delay_time_part1);
  PORTL &= !(1 << DDL1);
  PORTL |= 1 << DDL0;
  delay(delay_time_part1);
  PORTL &= !(1 << DDL0);
}
void part_2_4(){
  // System runs at 16MHz, which means 62.5ns per one tick for digitalized signal
  // scaling down to 200kHz with a pre-scale of 8. (5 micro seconds per tick)
  // digitalized signal reads every 5 micro seconds.
  // N = 2,000,000Hz / 400Hz = 5000. 5000 / 2 = 2500
  DDRH |= 1 << DDH3; // Enable PIN 6
  OCR4A = 2500;// 400Hz
  delay(delay_time_part2); // run it at 400Hz for one second.
  OCR4A = 4000;// 250Hz
  delay(delay_time_part2); // run it at 250Hz for one second.
  OCR4A = 1250;// 800Hz
  delay(delay_time_part2); // run it at 800Hz for one second.
  DDRH &= 0 << DDH3; // Disable PIN 6
  delay(delay_time_part2);
}
void part_3_1(){
```

```
  // This is for part 3.1.
  // This is a kind of scheduler that calls each function.
  part_1_4();
  part_1_4();
  part_2_4();
  sleep();
}
void LED_part_3_2(){
  // task A and part of task C for part 3_2
  // Each LED flashes for 333ms.
  static int time = 0;
  if((time % 1000) == 0){
    PORTL |= 1 << DDL2;
  }else if((time % 1000) == 333){
    PORTL &= !(1 << DDL2);
    PORTL |= 1 << DDL1;
  }else if((time % 1000) == 666){
    PORTL &= !(1 << DDL1);
    PORTL |= 1 << DDL0;
  }else if((time % 1000) == 999){
    PORTL &= !(1 << DDL0);
  }
  time++;
  // when time = one second, time is set to zero.
  if(time == 1000){
    time = 0;
  }
}
void speaker_part_3_2(int total_time){
  // Speaker with a prescale of 8 (same as the previous one.)
  // Using %, the speaker runs at a frequency of 400, 250, 800, and 0 Hz for a to
tal of 4 seconds.
  static int time = 0;
  if((time % 4000) == 0){
    DDRH |= 1 << DDH3; // Enable PIN 6
    OCR4A = 2500; // 400Hz
  }else if((time % 4000) == 1000){
    OCR4A = 4000;// 250Hz
  }else if((time % 4000) == 2000){
    OCR4A = 1250; // 800Hz
  }else if((time % 4000) == 3000){
    DDRH &= 0 << DDH3; // turn off.
  }
  time++;
  // when either time = 4000 or total time = 15999, time set to zero.
```

```
  if(time == 4000 || total_time == 15999){
    time = 0;
  }
}
void sleep(){
  // sleep for one second.
  DDRH &= 0 << DDH3; // speaker off
  PORTL &= !(1 << DDL2); // LED off
  PORTL &= !(1 << DDL1);
  PORTL &= !(1 << DDL0);
  delay(delay_time_part2); // 1 second delay.
}
void scheduler_part_3_2(){
  //scheduler for part 3_2
  // delay_time_part3 == 16 seconds = total time
  static unsigned int time;
  if((time % delay_time_part3) == 0 || (time % delay_time_part3) < 1000){
    LED_part_3_2();
  }else if((time % delay_time_part3) == 1000 || (time % delay_time_part3) < 2000)
{
    LED_part_3_2();
  }else if((time % delay_time_part3) == 2000 || (time % delay_time_part3) < 6000)
{
    speaker_part_3_2(time);
  }else if((time % delay_time_part3) == 6000 || (time % delay_time_part3) < delay
_time_part3){
    LED_part_3_2();
    speaker_part_3_2(time);
  }
  time++;
  if(time == delay_time_part3){
    sleep();
    time = 0;
  }
  delay(1);
}
void part_3_3(int total_time){
  // This plays "Mary Has a Little Lamb."
  static int time = 0;
  DDRH |= 1 << DDH3; // Enable PIN 6
  OCR4A = melody[time/150];
  if(time / 150 == note_count){
    time = 0;
  }
  time++;
```

```
  if(total_time == delay_time_part3_3-1){
    time = 0;
  }
}
void scheduler_part_3_3(){
  /*  scheduler for part 3_2
      LEDs for 2s -> speaker for 7.5s -> both LEDs and speaker for 10 seconds -
> sleep for one second
      delay_time_part3 == 19500 micro seconds = total time
  */
  static unsigned int time;
  if((time % delay_time_part3_3) == 0 || (time % delay_time_part3_3) < 2000){
    LED_part_3_2();
  }else if((time % delay_time_part3_3) == 2000 || (time % delay_time_part3_3) < 2
000+(note_count*150)){
    part_3_3(time);
  }else if((time % delay_time_part3_3) == 2000+(note_count*150) || (time % delay_
time_part3_3) < 12000+(note_count*150)){
    LED_part_3_2();
    part_3_3(time);
  }
  time++;
  if(time == delay_time_part3_3){
    sleep();
    time = 0;
  }
  delay(1);
}
void scheduler_part_4() {
  // Scheduler for part 4.
  // A0 and A1 for alalog inputs.
  static unsigned time;
  x_value = analogRead(JoyX);
  y_value = analogRead(JoyY);
  // Serial.print("X: ");
  // Serial.println(x_value);
  // Serial.print("Y: ");
  // Serial.println(y_value);
  part_4(x_value, y_value);
  part_3_3(time);
  delay(1);
}
void part_4(int x_value, int y_value){
  // This function takes x and y values and divided by 128 to express into 0 to 7
 to generate output through 8x8 LEDs.
```

```
  int x_map = x_value / 128;
  int y_map = y_value / 128;
  // Serial.print("X: ");
  // Serial.println(x_map);
  // Serial.print("Y: ");
  // Serial.println(y_map);
  //The loop interates through and if i == x_map(0 to 7) then, set the column as
1 to turn the light on.
  for(int i = 0; i < 8; i++){
    if(i == x_map){
      spiTransfer(i, 1 << y_map);
    }else{
      spiTransfer(i, 0b00000000); // if no value is read, put to rest. (at center
)
    }
  }
}
void spiTransfer(volatile byte opcode, volatile byte data){
  int offset = 0; //only 1 device
  int maxbytes = 2; //16 bits per SPI command

  for(int i = 0; i < maxbytes; i++) { //zero out spi data
    spidata[i] = (byte)0;
  }
  //load in spi data
  spidata[offset+1] = opcode+1;
  spidata[offset] = data;
  PORTB &= 0 << DDB5; //
  for(int i=maxbytes;i>0;i--)
    shiftOut(DIN,CLK,MSBFIRST,spidata[i-
1]); //shift out 1 byte of data starting with leftmost bit
  PORTB |= 1 << DDB5;
}
```

Overall Performance Summary:

Setting GI/O pins directly not using built-in functions was confusing at first, especially for part 2. Once we understood it, the rest of the lab procedures became easier to conduct. We feel confident to manipulate hardware registers, bits, CTC mode, and even other modes without using any user-friendly codes in Arduino. This lab gave us a good opportunity to perform some of the low-level hardware functions and firmware programming. With much time and effort, we were able to accomplish all of the tasks given in this lab.

Teamwork Breakdown:

For Lab 2, both Kejin and I conducted all of the experiments individually for learning purposes. We both were responsible for building architecture, coding, debugging, integrating hardware and software by ourselves. Whenever we had time, we set up a meeting in the lab, discussed, and shared our codes and ideas with each other. This way, we could both have a chance to perform all of the procedures by ourselves and could help each other whenever we got stuck. We are going to stick with this plan for Lab 3 as well.

Discussion and conclusions:

The most challenging part of this lab was part 2.4, which was using CTC mode to precisely run the speaker at different frequencies. First, we had to understand how CTC mode works (CTC block diagram). Then, we needed to understand the functions of each of the low-level hardware (TCNT, TCCR, OCRA, and OCA). It took us two hours to fully understand the concept and started coding the hardware functions. Being able to read the ATMEGA2560 datasheet was another component we learned from this course. We are now more confident to manipulate any features provided in ATMEGA2560. We are assuming that we would be required to use different types of advanced schedulers for the next lab, so we are planning to self-study more about schedulers before the next lab.