Lab 2: Digital I/O and timing of outputs

Juan Trejo, 1939336

Introduction:

In the lab, I acquired knowledge on controlling analog inputs and digital outputs through direct setup using low-level hardware functions (firmware programming). By utilizing digital outputs, I successfully illuminated an 8x8 matrix of LEDs and produced sound on a speaker at varying frequencies. This comparison allowed me to generate a sequence of tones at different frequencies. Lastly, I acquired skills in constructing a scheduler to effectively manage various tasks and their respective timings.

Methods and Techniques:

For part 1.2, I used the same technique from Lab 1 to set up the LEDs and get them to flash in a sequence. All I needed was to have the 3 LEDs connected to the board through equal resistors and get them to light up using the digitalWrite() function.

Part 1.4 was a little trickier because I had to get the LEDs to light up in the same sequence as 1.2 without using the helpful digitalWrite() function or pinMode(). Instead I used direction registers and port output.

For part 2.4, identifying which technique to verify the functionality of our program was very daunting and confusing at first. So I drew a couple of block diagrams to try and understand the concept of a 16-bit counter/timer. Through multiple failed calculations and rewritten code, I was eventually able to figure out how to generate the necessary frequencies for the tones.

Part 3 was pretty straightforward after some days of troubleshooting and debugging. Since it was simply taking previous parts in this lab and making them tasks to run in a sequence and concurrently.

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.

Experimental Results:

In Part 1.2, we followed a familiar process since we had already done it in lab 1. To enable Pin 47, 48, and 49 using firmware programming, we referred to the Pinout-Mega256 sheet. We discovered that activating PL2, PL1, and PL0 was necessary for these specific pins. Consequently, we enabled these pins by setting the corresponding bit fields to "1" (for example, DDRL |= 1 << DDL2, DDL1, and DDL0). Additionally, we configured these pins as output mode by setting their respective bitfields to "1" as well (for example, PORTL |= 1 << DDL2, DDL1, and DDL0). By connecting the same hardware setup with 220Ω resistors as in lab 1, we successfully flashed each LED for a duration of 333ms.

In Part 2.4, we encountered some challenges that took us a considerable amount of time to comprehend. To gain a thorough understanding of how the 16-bit timer/counter works, we resorted to drawing block diagrams and timing diagrams for TCNTn and OCnx on a whiteboard. This process involved an hour-long discussion. The ATmega2560 operates at 16MHz with a pre-scale of 1 (no pre-scale), resulting in a data read rate of 62.5 nanoseconds per second. By applying a pre-scale, we can decrease this period or frequency. For example, with a pre-scale of 8, the system operates at a frequency of 200kHz, equivalent to a period of 5 microseconds (μs). To produce a speaker frequency of 400Hz, we determined that the value of N (half cycle toggle) should be 5000. Through similar calculations, we derived N values for frequencies of 250Hz (4000) and 800Hz (1250). To start, we cleared the previous data stored in the control registers by setting TCCR4A and TCCR4B to 0. Since TCCR4B lacks a CTC mode function, we set CTC mode using TCCR4A, and we set the waveform generation mode and pre-scale of 8 by assigning 1 to the respective bitfields (e.g., TCCR4B |= 1 << CS41 | 1 << WGM42). Additionally, we configured Pin 6 as an output mode (DDRH |= 1 << DDH3 and PORTH |= 1 << DDR3). Finally, we set OCR4A to the calculated N values with a time delay of 1 second, resulting in the speaker playing different frequencies for one second.

For Part 3.1, we implemented a function called "part_3_1" that called the functions from Part 2.4 at various time intervals. We also created a "sleep" function to turn off all LEDs and speakers by setting the corresponding PORT bitfields to "0" for one second. The "part_3_1" function is simply called these functions, allowing the LEDs to run for two seconds, the speaker to play at different frequencies for four seconds, and a one-second pause (sleep).

In Part 3.2, we developed a function called "scheduler_part_3_2." Within this function, an if statement compares a variable called "time" with the total time, "delay_time_part3." With a delay time of 1ms, the "time" variable incremented by one unit every 1ms, and the entire program lasted for 16 seconds. The if condition was set as (time % delay_time_part3 == 0 || time % delay_time_part3 < 1000). This condition allowed the execution of the "LED_part_3_2" function inside the if statement for one second. We configured this function to run for two seconds, followed by (time % delay_time_part3 == 2000 || time % delay_time_part3 < 6000) to run the "speaker_part_3_2" function from 2 seconds to 6 seconds. Next, (time % delay_time_part3 ==

6000 || time % delay_time_part3 < 16000) caused "LED_part_3_2" and "speaker_part_3_2" to run concurrently for 10 seconds, followed by one second of sleep. Finally, an if statement checked if the "time" variable was equal to 16000 and reset it to 0 to repeat the aforementioned steps.

In part 3.3, we implemented a function that utilized an array of different frequencies. Within this function, we created an unsigned int variable called "time." This variable is incremented by one every millisecond. We set OCR4A, which controls the tone generation, to be equal to melody[time/150]. This allowed us to play the tone for a duration specified by "time/150." The function continued to play subsequent tones until "time/50" was equal to the size of the array (sizeof(melody)/sizeof(melody[0]).

We also introduced a variable for the total time, which was used in if statement conditions to compare with the "time" increment. For instance, when the time was 0, the LEDs started flashing for two seconds. Then, starting from 2000ms until the time equaled 2000ms plus (number_of_tones multiplied by 150ms), it played the melody "Mary Has a Little Lamb." The next step was to turn on both the LEDs and "Mary Has a Little Lamb" for 10 seconds. Therefore, if we set the time between 2000ms plus (number_of_tones multiplied by 150ms) and (12000 ms plus number_of_tones multiplied by 150 ms), both the LEDs and the song stopped at the total time.

In part 4, we utilized DDF0 and DDF1 as analog inputs and stored the values from a joystick in the variables "x_value" and "y_value." These values ranged from 0 to 1023 but were converted to the range of 0-7 and stored in the integer variables "x_map" and "y_map." This mapping allowed us to control the lighting of an 8x8 LED matrix based on the joystick's coordinates. By using a for loop with the condition (int i = 0; i < 8; i++), we compared "i" with "x_map." If they were equal (i.e., i == x_map), we set the same number of "y_map" (representing the column) to 1, causing the LED at the corresponding x and y coordinate to turn on.

Code Documentation:

```
/*
 * University of Washington
 * ECE 474 prof. Vikram Iyer
 * Juan Trejo, 1939336
 * Lab 2: Digital I/O and Timing of Outputs
 */
#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, NOTE_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, NOTE_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, NOTE_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 TCCRnB does not have compare output mode. 0b01000000
  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), pin 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 of 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 = 200,000Hz / 400kHz = 500. 500 / 2 = 250
  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 one second each, a total 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(){
  /*  RR 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) < 2000+(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:

Initially, I found it challenging to set GI/O pins directly without relying on built-in functions, especially in part 2 of the lab. However, once I grasped the concept, conducting the remaining procedures became easier. I now feel confident in manipulating hardware registers, bits, CTC mode, and even other modes without relying on user-friendly Arduino codes. This lab provided a valuable opportunity for me to engage in low-level hardware functions and firmware programming. With significant time and effort, I successfully completed all the tasks assigned in this lab.

Teamwork Breakdown:

Recapping what I told the class staff on the ED discussion board, my partner Chenyu Hu did not help at all. Despite multiple attempts to communicate throughout the process, the only responses I got from them are "ok, sounds good, and still lost". In my mind I had feared this would happen due to the fact that I got no help from them for lab 1 either.

After having told the class staff this, I completed the lab and typed out this report on my own.

Discussion and Conclusions:

The most challenging aspect of the lab for me was part 2.4, which involved utilizing CTC mode to accurately operate the speaker at various frequencies. Initially, I needed to comprehend the workings of CTC mode through the block diagram. Additionally, understanding the functions of the low-level hardware components such as TCNT, TCCR, OCRA, and OCA was crucial. It took me approximately two hours to fully grasp the concept and begin implementing the hardware functions. This lab allowed me to enhance my skills in reading the ATMEGA2560 datasheet, enabling me to manipulate any available features on the microcontroller. Looking ahead, I anticipate the need to utilize different advanced schedulers in the upcoming lab, prompting me to engage in self-study on schedulers prior to its commencement.

Links to Demo Videos:

1.4 Flashing LEDs
2.4 Speaker Tone
3.2 Concurrent Tasks
3.3 Concurrent Tasks
4.0 Interactive Displays

Q&A Video: https://youtu.be/5m_J9D1-wLs