Ean Barnawell ECE 474
Kevin Lu

Lab 2 Report

Ean Barnawell, 2177193
Kevin Lu, 2167804 Jul 14, 2023

Assignment: **Lab 2 - Digital I/O and timing of outputs**

Page length guidance for written documents refers to a single-spaced, 11 or 12-point font.

Introduction: (¼ page)
There are two main learning objectives for this lab:

- Understanding and the ability to change registers through bit manipulation.
- Understand schedulers and be able to implement concurrent tasks

The large majority of bit manipulation in this lab is setting bits and clearing bits which can be done through a 1 bit mask and either or, or complement and. In addition, since this lab focuses on low-level hardware, we need to understand which registers correspond to what function within the hardware. For example, DDRx and PORTx will set a pin to output and send signals to the pin. As for the second learning objective, this lab tests the usage of round-robin scheduling to have a combination of LEDs, speaker, and thumbstick/grid functions.

Methods and Techniques:(½ page)

Part 1:Hardware bit manipulation
The first activity is a simple LED switch on and off through pinMode and digitalReads. This functionality was explored in the previous lab so the skills, instruments and tests were all the same. The next activity is to re-implement switching LEDs on and off through bit manipulation on registers. This required understanding of the first learning objective, and did not require additional tests and instruments beyond the first activity. The testing was simply checking whether the LEDs would turn on and off in a sequential manner.

Part 2:16-Bit Timer/Counter
The second part of the lab was to research and implement functionality of a 16-bit timer/counter and play a tone sequence through a speaker. The skill set required here is further understanding of how each register corresponds to functionality on the hardware. The bit manipulation is similar to the previous activity, however, realizing how everything comes together is the main test in this part. The actual test is simply adhering to the tone sequence and the instrument required is a speaker.

Part 3: Concurrent Tasks
The third part of this lab focuses on the second learning objective. The first section of this part is increasing the modularity of the code. Then understanding how each task can be performed concurrently in a round-robin scheduler fashion. Specifically, flashing LEDs and a speaker playing “Mary had a little lamb”. The skillset required here is understanding schedulers and

millis() function to create concurrent tasks. There are no further instruments required here as we are combining the first and second part of the lab. The tests required are adhering to the desired functionality. However, further tests through print() statements will be useful to debug the implementation.

Part 4: Interactive Display

Experimental Results:(1-3 pages as necessary)

Part 1:Hardware bit manipulation

For the first section of our lab, the test results were simple as we needed to see whether the green, blue, and red lights would run for 333ms sequentially in that order. A swift implementation of using bit manipulation on registers showed that this functionality succeeded. The result was a great looking set of blinking lights, each turning on and off at the time described in the lab document.

Part 2:16-Bit Timer/Counter

For this part of the lab, we had the task of producing a tone using one of the on board Timer/Counters that are standard in the chipset of an Arduino Mega 2560. Using the provided data sheet, and as well as being provided some code examples in class, we were able to fashion two functions together, that would enable us to use a CTC mode, which has the T/C count up to whatever preset value of OCRnA, then can either Set, Clear or Toggle Output on a match with those values. Our result was able to produce three tones(400Hz, 250Hz, and 800Hz) in order with a second of silence at the end, set to repeat in a loop.

Part 3: Concurrent Tasks

In this part of the lab, we seek to combine parts 1 and 2 together, where we are able to have our tasks occur in a timed event, where we have blinking LED's go for 2 seconds, then have our speaker play a song for one cycle, then on to having both tasks run at the same time for 10 seconds. Through diligent coding and communication between both lab partners, we were able to accomplish this by slightly modifying our code used in part 2. Our results enabled us to pass the demo portion of this lab.

Part 4: Interactive Display

In the final part of this lab, we were to add an interactive XY LED display matrix that was given input via thumbstick. We used the provided starter code to give us a clear framework for the function for our LED 8x8 display. We had this for our final demo portion of the lab, where were to have the song play through our speaker as we had our LED display matrix going at the same time.

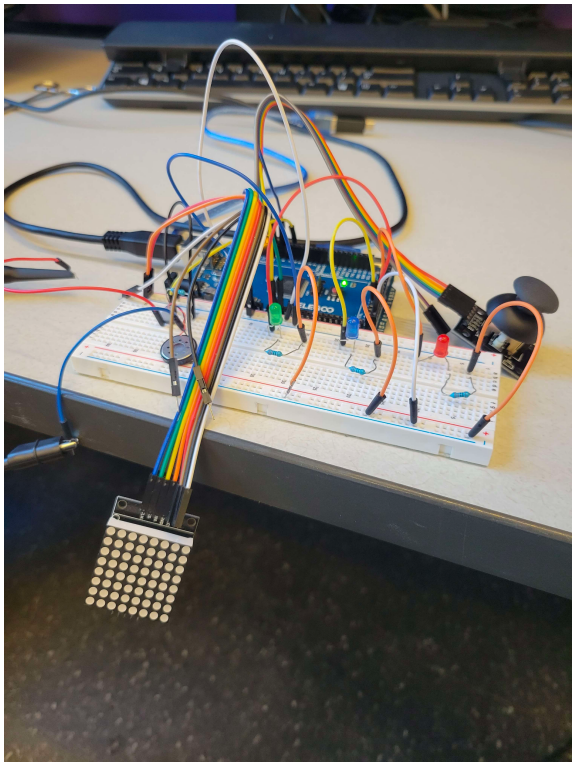

Figure 1. Our demo circuit with the speaker, thumbstick, series of LED's as well as the Display 8x8 LED Matrix.

Code Documentation:

Part 1: Hardware bit manipulation

```
//properly assigning pins based off of
//provided layout guide
const uint8_t redLED = PORTL2;
const uint8_t blueLED = PORTL1;
const uint8_t greenLED = PORTL0;
void setup(){
 // sets the bit corresponding to  the pin in DDRL to 1, indicating that it is an output pin.
  DDRL |= (1 << redLED);
  DDRL |= (1 << blueLED);
  DDRL |= (1 << greenLED);
}
void loop(){
  // Turn on the red LED
  PORTL |= (1 << redLED);
  PORTL &= ~(1 << blueLED);
  PORTL &= ~(1 << greenLED);
  delay(333); // Wait for .333 second
  // Turn on the blue LED
  PORTL &= ~(1 << redLED);
  PORTL |= (1 << blueLED);
  PORTL &= ~(1 << greenLED);
  delay(333); // Wait for .333 second
```

```
  // Turn on the green LED
  PORTL &= ~(1 << redLED);
  PORTL &= ~(1 << blueLED);
  PORTL |= (1 << greenLED);
  delay(333); // Wait for .333 second
```

Our code for part 1 is very standard in the way that we assigned pins on the Arduino board to output signals that we designated with bit masking techniques to turn on said assigned pins. We coupled this output with delays in the void loop() section to have a never ending cycle of LED's lighting up in their designed order.

Part 2: 16-Bit Timer/Counter

```
void tone_gen(double hertz,int loopCount) {
  bit_set(TCCR4B, WGM42);
  bit_set(TCCR4B, CS42);
  TCNT4 = 0;
  OCR4A = hertz;
   // Number of times to run the loop
   while (loopCount > 0) {
    //Checking if the Output Compare flag bit for OCF4A in TIFR4 is set
    // This flag is set when counter 4 reaches the value in OCR4A.
    if (TIFR4 & (1 << OCF4A)) {
      bit_set(TIFR4, OCF4A);
      PORTB ^= (1 << LEDP12);
      loopCount--; // Decrement the loop counter
    }
  }
}
```

This is the way we did part 2 initially, where we had two parameters being passed into the function, one that was the value where OCR4A was being defined as the top value to flag the T/C where to restart counting, and then another value being passed to the while loop that defined how long the loop would go for. This works in the following way: to play a 250Hz tone, we would pass in 500 as the value of Hertz for the OCR4A value, then have that while loop go for 125 cycles. If we multiply these numbers together, we get 62500, meaning that to get this tone to play for 1 second, we would need to have the timer count to 500, 125 times, due to our prescale of 256 and the internal clock being 16MHz. An interesting function, but a bit more complex than it needed to be. We then coupled this with a silent version of this function, to play three tones (400Hz, 250Hz, and 800Hz) with a second of silence at the end.

Part 3: Concurrent Tasks

We started with changing the toneGen function to be able to operate in an void loop(), due to the issues we were having with our original iteration:

```
void tone_gen(int hertz){
  // Initialize registers
  TCCR4B = 0;
  TCCR4A = 0;
  // Set up the timer with prescaler = 256 with CTC mode
  // set your TCCR4B to WGM42
  bit_set(TCCR4B, WGM42);
  //set your TCCR4B to CS42 (Clock Select bit 2) needs to set to 1
  bit_set(TCCR4B, CS41);
  // initialize timer/counter 4 to 0
  TCCR4A = (1 << COM4A0);
  //Initialize output compare register to the value that it will count up to
  OCR4A = F_CPU / (2 * hertz * 8) - 1;
}
```

We ended up going with a simpler version, where we defined the TCCR's, then utilized both 4A and 4B to be able to use the Comparator (COM4A0) register to get it to generate a tone when the OCR4A value matched the Counters. This let us use this function in the loop(), where we were unable to do that with our original version, which was designed to be done in the int main(). We divided all of our original functions into tasks, where we were able to call them in the loop() function how we saw fit, being able to have them go concurrently, or one after the other. TaskA is our flashing LED's, TaskB is our simple tone sequence, TaskC plays our melody, and TaskD is our LED Matrix with thumbstick. TaskE is designed to play the melody "Mary had a little Lamb" at the same time we have LED's flashing. TaskF lets us play with the LED matrix while the LED bulbs in series are flashing, and TaskG lets us hear the simple tone while having the LED's in series flash at the same time. Over all, very modular code, allowing for a mixture of ways for these elements to be combined.

Part 4: Interactive Display

The implementation for part 4 ranges from lines 177 to 207, and 117 to 131, with some initial definitions from lines 24 to 42. The functions required for this were spiTransfer and TaskD. The spiTransfer() function is a helper function that takes in a row index and 8 byte column object to update the 8x8 grid accordingly. Task D serves as the bridge between reading an analog input (value from 0-1023) from the thumbstick and converting its values to a form recognizable to the spiTransfer() function. Once setup together, moving the thumbstick will send a signal and update a singular LED on the 8x8 grid according to the thumbstick's relative location on the board.

Overall performance summary: (½ page)
Our demo succeeded in all the learning objectives for this lab and demonstration objectives. However, our 8x8 grid ran into the issue where the LEDs on the grid would be rapidly blinking rather than a stagnant output. This caused the initial location of the thumbstick inputs to vary between spots (4, 4), (3, 3), (3, 4), and (4, 3). This could possibly be caused due to the grid not having an exact center or implementation glitches when running the grid and speaker function simultaneously.

I believe that our group accomplished the first learning objective very well. We both understand bit manipulation of registers and utilized it in finishing multiple sections of our lab. However, the second learning objective was much more of a struggle. We initially implemented our functions through overriding main() which only made things harder. Because of this decision, we both did not know how to have concurrent tasks as using millis() was out of the question. After changing our implementation to use the default provided main() function, we were able to swiftly implement concurrent functions through use of millis().

Teamwork Breakdown (¼-½ page)
My lab partner and I decided to both work on the lab separately so we can both learn as efficiently as possible. However, we met together to talk about our progress on part 2 which resulted in us researching and Ean implementing a way to specify how long a tone would play for. After having part 2 sorted out, we moved on to part 3 which stumped us for a long time. We decided to have part 4 finished first which Kevin coded, debugged, and integrated the hardware. Once everything else was finished, we met together to finish up part 3 which included mutual effort in coding and debugging. Ean excelled in debugging and understanding how the hardware worked whereas Kevin excelled in coding methods and strategies.

Discussion and conclusions (½ page)
Part one was relatively easy for our team. We ran into a bit of trouble in part two but the provided code was very simple to follow and we understood the functionality of each line. Part four was also relatively simple. The only issue we ran into in part 4 was how to update a specific pixel on the grid rather than a clump of pixels. The most challenging parts of this lab was part 3 with concurrent tasks. Initially, we had our functionality done through main() but implementing a main() function in Arduino will override all the initialization done by Arduino. Because of this, analogRead() would not work, nor will delay(), millis(), and many other functions. We found that continuing down this road will only be more difficult as we would have needed a deep understanding of how C works with arduino software to implement part 3. As a result, we decided to change our implementation using the default main() function but ran into the problem that our tones would not play. After more tweaking of our implementation, we fixed all issues and successfully implemented part 3. In addition to learning official learning objectives, we learned how the Arduino IDE works in regards to main(). As mentioned earlier, Arduino hides a

lot of implementation in their main() function where overriding it would create a lot of gaps in our code.