Lab 2: Digital I/O and Timing of Outputs

Arvin Nusalim - 2276767
Samantha Reksosamudra - 2276717 January 31, 2024
Assignment: Lab 2
No table of contents entries found.

A. Introduction

The learning objectives in this lab included: (1) manipulating hardware registers/ bits, without the use of existing libraries, to perform low-level hardware functions, and (2) coordinating multiple concurrent tasks with round-robin scheduling. The lab explored the digital I/O on the Arduino Mega board to do multiple tasks at the correct timing. Tasks include flashing a sequence of LEDs, playing a tone on the buzzer, and flashing an 8x8 matrix of LEDs. Since the Arduino Mega board is unable to execute different tasks at the same time, the execution of multiple tasks will highly rely on the verification of light motion and audio tone. Oscilloscopes were used to verify the timing and frequency output throughout this lab.

B. Methods and Techniques

The following materials were used to complete this laboratory activity:

1. Arduino Mega Microcontroller board
2. USB- Type-B cable (with USB-A or USB-C for the computer end)
3. External Arduino power supply (120VAC power adapter)
4. 3 LEDs
5. 3 250-500 Ohm resistors
6. Passive Buzzer
7. 8x8 LED Matrix
8. 2-way thumbstick control
9. Solderless Breadboard
10. Wires

Procedure 1: The first procedure in this lab involved wiring 3 LED connected with 330Ω resistors and use pinMode, digitalWrite, and delay() function to test whether they flash on and off in a sequential pattern with a period of 1 second. Next, replace the pinMode and digitalWrite functions into registers that control the pins as well as the delay function by utilizing millis() function.

Procedure 2: The second procedure in this lab involved setting up a 16-bit timer/counter on the Arduino board. The board already has timer control registers, and by setting and clearing the registers, the timer mode can be set (i.e. CTC mode). To output a tone on the buzzer on the correct timing, the clock and prescaler value must also be set accordingly. Figure 1 below describes the mode of operation of each timer/counter.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Fig. 1 Waveform Generation Mode Bit Description

Procedure 3: The third procedure in this lab involved running 2 tasks simultaneously. First, create Task A from procedure 1.4.2 and Task B from procedure 2.4. On another task (Task C), that calls Task A for 2 seconds, Task B for 1 cycle, and both Task A and Task B simultaneously.

Procedure 4: The fourth procedure in this lab involved using a thumbstick to control a dot on the 8x8 matrix LED. The thumbstick consists of two voltage dividers, each senses the X or Y-direction. The final objective is to control the dot while continuing the LED sequence and playing the “Mary Had A Little Lamb” song.

C. Experimental Results

1. Procedure 1

Macro functions such as pinMode (to initialize the digital pins) and digitalWrite (to assign HIGH or LOW values to the hardware wired to the pin) were used to show simple flashing LEDs. Each LED flashes once every 333ms, so a period is 1000ms (i.e. 1 second). However, this can achieve the same results by using hardware bit manipulation. Data Direction Registers (DDR) is used to set up LED pins as output, while PORT is used to toggle bit values. By shifting the bits according to the board’s hardware guide, the exact pins can be controlled. Bit 1 and 0 means HIGH and LOW, respectively. A delay_task() function was created that used the millis() function in an empty while loop to replace the delay() function. So without changing the hardware setup, the new code (without macro functions) will display the same result as using pinMode, digitalWrite, and delay() function.

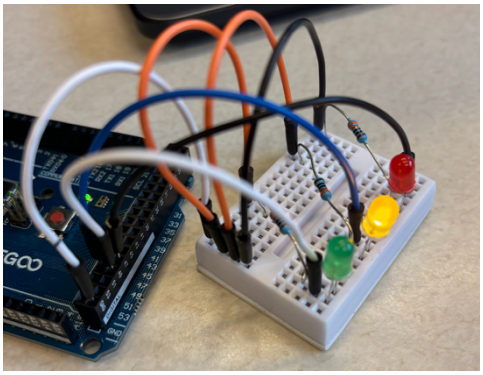

Fig.2 LED setup connected with pin 47, 48, 49

2. Procedure 2

The hardware setup in this procedure only included a passive buzzer wired to the Arduino board. Since CTC mode was used for a 16-bit timer, the timer control registers were set accordingly in the setup function. To play a tone on the buzzer, the OCR4A value can be set by using the equation below.

$$OCR4A = \frac{F_{CPU}}{2 \cdot prescaler \cdot frequency} \quad (1)$$

Since the global variable 'duration' is set to 1000ms, the buzzer played each tone (i.e. frequency) for every second.

3. Procedure 3

New tasks (or functions) were created to perform the LED sequence flash and the timer tone output, called Task A and B, respectively. Task C was also created to control the operation of both Task A and B. Using a while loop, the task can be executed in a loop for a set duration. So Task C included executing task A for 2 seconds, and clearing all the LEDs before starting task B (if not, then the last LED will stay on). Similarly, the buzzer was set to off after it finished.

The final objective of this procedure was to execute both Task A and B at the same time. Though it is impossible to do so, the Arduino can make it seem like they are executed at the same time by taking advantage of how each line of code runs very fast (i.e. a few hundred microseconds). A global variable 'counter' was used to help this process. It translates which LED is currently on and uses the flashing motion as an indicator of which tone to play next instead of setting a set duration like in Procedure 2.

Through the human eye, it will seem as if the buzzer and LED are executed at the same time, but instead, the Arduino board still executes them one by one.

4. Procedure 4

The 2-way thumbstick control gives inputs of row and column that range from 0 to 1023. As the LED matrix takes input of integer for the row and 8 bits for the column, a convert function needs to be created. It takes in the row and column from 2-way thumbstick control and turns it into integers that range from 0 to 7. Although the row are ready to be used, the column type is not suitable for the input of the LED matrix. The input can be obtained by shifting 1 to the left by the column value.

The final objective of this procedure was to execute both Task A , and Task B, as well as the LED matrix with 2-way thumbstick control at the same time. This can be done by using a while loop to loop until the songs end. Inside the while loop, Task B and Task A are called before using another while loop to loop through the LED matrix for a period of time. In this case, 334 ms is used resulting in the LED and note of the song changing every 334 ms and the LED matrix keeps on looping.

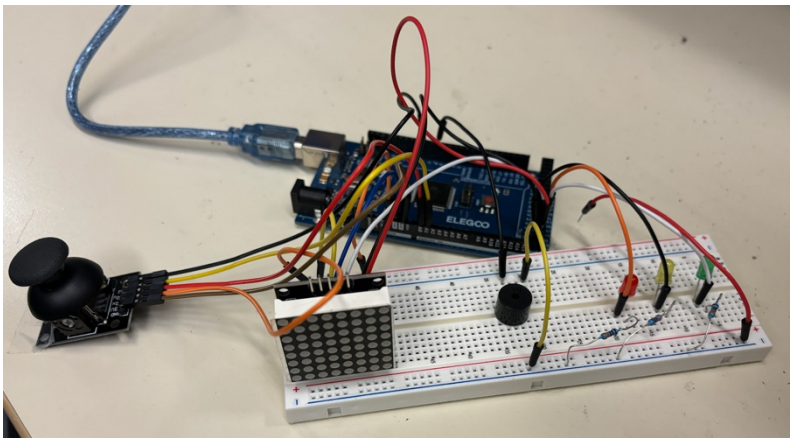

Fig.4 8x8 LED matrix connected with 2-way thumbstick control to run simultaneously with passive buzzer and 3 LEDs.

D. Code Documentation

1. Procedure 1

The setup function is used to initialize the pins. Instead of using pinMode, a data direction register (DDR) was used. The ports of pins 47, 48, and 49 are PL2, PL1, and PL0 consecutively. The pin can be initialized by adding 1 shifted to left by 2 for pin 47, depending on the pin number.

```
void setup() {
  // PART 1.2 - using pinMode and digitalWrite
  // initialize digital pins as an output.
  // pinMode(LED1, OUTPUT);
  // pinMode(LED2, OUTPUT);
  // pinMode(LED3, OUTPUT);

  //PART 1.4 - using DDR and PORT
  DDRL |= 1 << 2;   //pin 47
  DDRL |= 1 << 1;   //pin 48
  DDRL |= 1 << 0;   //pin 49
}
```

Fig.5 Digital pin initialization sketch

Similar on the setup function, to set the LED on, the PORTL was modified by using an or-equal operator (I=) and to set the LED off without changing other pins, the PORTL was modified by using an and-equal operator (&=). The code in Figure 6 will let the LEDs flash on and off in a sequential pattern with a period of 1 second.

```
void loop() {
  // PART 1.2 - using pinMode and digitalWrite
  // digitalWrite(LED1, HIGH);
  // // digitalWrite(LED2, LOW);
  // digitalWrite(LED3, LOW);
  // delay(333);

  // digitalWrite(LED1, LOW);
  // digitalWrite(LED2, HIGH);
  // // digitalWrite(LED3, LOW);
  // delay(333);

  // // digitalWrite(LED1, LOW);
  // digitalWrite(LED2, LOW);
  // digitalWrite(LED3, HIGH);
  // delay(333);

  // PART 1.4 - using DDR and PORT
  PORTL |= 1 << PORTL2;         // set pin 47 (or PORTL2) HIGH
  PORTL &= ~(1 << PORTL0);      // set pin 49 (or PORTL0) LOW
  delay_task(333);

  PORTL &= ~(1 << PORTL2);      // set pin 47 (or PORTL2) LOW
  PORTL |= 1 << PORTL1;         // set pin 48 (or PORTL1) HIGH
  delay_task(333);

  PORTL &= ~(1 << PORTL1);      // set pin 48 (or PORTL1) LOW
  PORTL |= 1 << PORTL0;         // set pin 49 (or PORTL0) HIGH
  delay_task(333);
```

Fig.6 Hardware bit manipulation sketch

2. Procedure 2

Initialize TCCR4B and TCCR4A to be able to set the mode and prescaler as well as toggle the buzzer pin when TIMER4_ON_BIT reaches MAX. WGM42 is used to select the CTC mode and CS41 is used to set the prescaler to 8.

```
void setup() {
  bit_set(DDRH, PH3);       // activate digital pin 6
  TCCR4B = 0;
  TCCR4A = 0;
  TCCR4B |= (1 << WGM42) | (1 << CS41);    // set to CTC mode and prescaler 8
  TCCR4A |= (1 << COM4A0);                // toggle (buzzer) pin when TIMER4_ON_BIT reaches MAX
}
```

Fig.7 Setting timer control registers

Bit_set and bit_clr are used to activate or deactivate a certain pin. To play the tone, play_tone() function is created that takes in the input of frequency and duration of the tone. If the frequency is 0, then set the OCR4A value to 0 and if it's not 0, use Equation (1) to find the correct OCR4A value.

```
/*****************************************************************************
  simplified I/O and other functions for this assignment
*****************************************************************************/
void bit_set(volatile uint8_t &reg, uint8_t position) {
  reg |= 1 << position;
}

void bit_clr(volatile uint8_t &reg, uint8_t position) {
  reg &= ~(1 << position);
}

void play_tone(int freq, int duration) {
  if (freq == 0) {
    OCR4A = 0;
  } else {
    OCR4A = F_CPU / (2 * prescaler * freq);
  }
  unsigned long start = millis();
  while (millis() - start < duration) {}
}
```

Fig.8 Prototype functions in Procedure 2

On the loop function, calling play_tone() with different frequencies and duration will create the desired tone.

```
void loop() {
  play_tone(f1, duration);
  play_tone(f2, duration);
  play_tone(f3, duration);
  play_tone(0, duration);
}
```

Fig.9 Sketch to play a frequency tone on buzzer

3. Procedure 3

Part 1 created a task that C is used to call taskA() for 2 seconds, taskB() for 4 seconds, and no output for 1 second.

```
// Control operation for Task A and Task B
void taskC(int taskA_dur, int taskB_dur, int delay_dur) {
  taskA(taskA_dur);
  taskB(taskB_dur);
  delay_task(delay_dur);
}
```

Fig.10 Part 3.1 task C sketch

Part 2 created a taskC() that is used to call taskA() for 2 seconds, taskB() for 1 music cycle, Task A and Task B run at the same time for 10 seconds, and no output for 1 second.

```
// Control operation for Task A and Task B
void taskC(int taskA_dur, int both_dur, int delay_dur) {

  unsigned long start = millis();

  // Do Task A for 2 sec
  while(millis() - start < taskA_dur) {
    taskA();
  }

  // Clear all LEDs
  bit_clr(PORTL, PORTL2); bit_clr(PORTL, PORTL1); bit_clr(PORTL, PORTL0);

  // Set to TRUE once we played task B one time
  taskB_only = true;
  // Do Task B once
  taskB();

  counter = 0;
  taskB_only = false;
  // Do both tasks simultaneously
  while(millis() - start < both_dur) {
    taskB();
    taskA();
  }

  // Clear all LEDs
  bit_clr(PORTL, PORTL2); bit_clr(PORTL, PORTL1); bit_clr(PORTL, PORTL0);

  // Do nothing
  delay_task(delay_dur);
}
```

Fig.11 Part 3.2 task C sketch

Part 3 includes modifying taskB() to play the tune “Mary Has a Little Lamb”. The frequency of all tones were defined and the notes to play the song were stored in an array called ‘melody’.

```
#define c 261 // OCR4A = 3830
#define d 294 // OCR4A = 3400
#define e 329 // OCR4A = 3038
#define f 349 // OCR4A = 2864
#define g 392 // OCR4A = 2550
#define a 440 // OCR4A = 2272
#define b 493 // OCR4A = 2028
#define C 523 // OCR4A = 1912
#define R 0

int melody[] = { e, R, d, R, c, R, d, R, e, R,e, R,e, R,d, R,d, R,d, R,e, R,g,
R,g, R,e, R,d, R,c, R,d, R,e, R,e, R,e, R,e, R,d, R,d, R,e, R,d, R,c, R,c };
```

Fig.12 “Mary Had A Little Lamb” tone array for Part 3.3

On taskB(), a for-loop was used to iterate through the array of tones to play the song by using the play_tone() function to play the specific frequency or tone.

```
// LED Sequence
void taskA() {
  if (counter % 3 == 0) {
    bit_set(PORTL, PORTL2);         // set pin 47 (or PORTL2) HIGH
    bit_clr(PORTL, PORTL0);        // set pin 49 (or PORTL0) LOW
    // delay_task(333);
  } else if (counter % 3 == 1) {
    bit_clr(PORTL, PORTL2);       // set pin 47 (or PORTL2) LOW
    bit_set(PORTL, PORTL1);       // set pin 48 (or PORTL1) HIGH
    // delay_task(333);
  } else {
    bit_clr(PORTL, PORTL1);       // set pin 48 (or PORTL1) LOW
    bit_set(PORTL, PORTL0);       // set pin 49 (or PORTL0) HIGH
    // delay_task(334);
  }
  // counter += 1;     // increment after a sequence is finished

}

// Timer Tone Output
void taskB() {
  if (taskB_only) {
    for (int i = 0; i < sizeof(melody)/sizeof(melody[0]); i++) {
      play_tone(melody[i]);
      delay_task(333);
    }
  } else {
    play_tone(melody[counter]);
  }
}
```

Fig.13 Part 3.2 task A and B sketch

4. Procedure 4

Taking the output of the 2-way thumbstick control, the convert() function to convert the integer from the range of 0 - 1023 to 0 - 7. The row needed to be change into 1 left shift by the row value to be taken as an input for the spiTransfer() function.

```
void ledMatrix_task() {
  // Get the column and row index
  convert(analogRead(A0), analogRead(A1));  // A0 for X-direction, A1 for Y-direction
  // Shift the row index value
  mov_row = 1 << mov_row;
  // Turn on the corresponding row and column LED
  spiTransfer(mov_col, mov_row);
  // Turn off the whole row
  spiTransfer(mov_col, 0b00000000);
}
```

Fig.14 Sketch to control dot on 8x8 Matrix LED

Below is the spiTransfer() function that is given for this procedure.

```
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;
  digitalWrite(CS, LOW); //
  for(int i=maxbytes;i>0;i--)
    shiftOut(DIN,CLK,MSBFIRST,spidata[i-1]); //shift out 1 byte of data starting with leftmost bit
  digitalWrite(CS,HIGH);
}

void convert(int analogValue1, int analogValue2) {
  mov_col = analogValue1 * 8/1024;     // X-direction
  mov_row = analogValue2 * 8/1024;     // Y-direction
}
```

Fig.15 Helper functions for ledMatrix_task()

To run taskA(), taskB(), and led_Matrix() simultaneously, a while loop needed to be created to run for 1 music cycle. Inside the while loop, taskB() and taskA() are called before using another while loop to loop through the led_Matrix() for a period of time. In this case, 334 ms is used resulting in the LED and note of the song changing every 334 ms and the led_Matrix() function kept looping.

```
// Control operation for Task A and Task B
void taskC(int taskA_dur, int delay_dur) {

  unsigned long start = millis();

  // Reset variables
  counter = 0;
  taskB_only = false;
  start = millis();


  // Do both tasks simultaneously
  while(millis() - start < sizeof(melody)/sizeof(melody[0]) * 333) {
    unsigned long temp = millis();
    taskB();
    taskA();
    while (millis() - temp < 334) {
      // taskB();
      // taskA();
      ledMatrix_task();
    }
    counter++;
  }

  // Clear all LEDs
  bit_clr(PORTL, PORTL2); bit_clr(PORTL, PORTL1); bit_clr(PORTL, PORTL0);
  // Set buzzer to silent
  play_tone(R);

}
```

Fig.16 Task C sketch for simultaneous tasks

E. Overall Performance Summary

During the demo session, the Arduino Mega is connected to all of the components such as 3 LEDs and resistors, passive buzzer, LED matrix and 2-way thumbstick control. Although the buzzer used is not the right buzzer (i.e. it was the active buzzer, instead of the passive) for the procedure as it produces an unsuitable tone for the music. Changing into the right buzzer solved this issue. However, there is a problem when doing the demo. When demoing procedure 3.2, the song did not repeat because it was missing a single line of code, which is to reset the counter back to 0 at every loop.

F. Teamwork breakdown

Coding and debugging sketches were done together to promote discussion and learning as a group. However, some other tasks were divided during the lab:

- Arvin : Document hardware setup, draft part B, D, E, and G of the report
- Samantha : Hardware setup/wiring, fill in other parts and proof-read the report

G. Discussion and conclusions

There are some challenging parts of this lab, which was to find out which mode and prescaler to use as well as the formula to calculate the OCR4A to generate the right tone. Running taskA, taskB, and LED matrix connected to the 2-way thumbstick control simultaneously is another challenging part as well as the procedure that we were proud of. In this lab, we have learned how to work with the Arduino board without using the macro functions instead use hardware bits manipulation.