**HACETTEPE UNIVERSITY COMPUTER ENGINEERING DEPARTMENT**


**Group Name:** Grup

**Student Information:** Merve Müge Deliktaş - 21526896
Onur Cankur – 21526791


**Course Information:** BBM 434 - Embedded Systems Laboratory
Spring – 2019

**Report Information:** Lab-07 Experiment Report

## Short Brief of Lab-07 and Function Explanations

- **Variables:**

```
void EnableInterrupts(void);
void WaitForInterrupt(void);
void DAC_Out(unsigned long data);
void LED_Init(void);
void Sound_Init(void);
void SysTick_Init(unsigned long count);
void SysTick_Handler(void);
void DAC_Init(void);

unsigned long ADCvalue1;
unsigned long ADCvalue2;
unsigned int count = 0;
unsigned int period = 20;
unsigned int i = 0;
unsigned int angle1 = 0;
unsigned int angle2 = 0;
unsigned int max_open_leds = 0;
unsigned int brightness = 0;
unsigned int on_percentage = 0;
unsigned long duty_cycle = 0;
unsigned long leds[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20};

const unsigned char SineWave[32] = {8,9,10,12,13,14,14,15,15,15,14,14,13,12,10,9,8,6,5,3,2,1,1,0,0,0,1,1,2,3,5,6};
unsigned char Index=0; // Index varies from 0 to 31

unsigned int DO = 1908;
unsigned int RE = 1700;
unsigned int MI = 1515;
unsigned int FA = 1432;
unsigned int SOL = 1275;
unsigned int LA = 1136;
unsigned int SI = 1012;
unsigned int DO_ = 956;
```

Figure 1 - variables

From Figure 1, you can see some variables that we used to implement the system. There some important variables that should be mentioned before function explanations. 6 LEDs are used in this lab for the bonus part and leds[] array is storing these LEDs. SineWave array is used to create a sine wave. We create this array using this website. Finally, we assigned a value for each note. For example, for the "DO" note, we calculate $16MHz/(32 * 262Hz)$. 16MHz is the speed of our launch pad, 32 is the length of our sine wave array and 262Hz is representing DO note.

- **LED_Init:**

```
/*
 * PortB is used to connect LEDs.
 * PortB5-0
*/
void LED_Init(void){
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB;
  delay = SYSCTL_RCGC2_R;
  GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R&0xFFFFFF0F)+0x00000000;
  GPIO_PORTB_AMSEL_R &= ~0x3F;
  GPIO_PORTB_DIR_R |= 0x3F;
  GPIO_PORTB_AFSEL_R &= ~0x3F;
  GPIO_PORTB_DEN_R |= 0x3F;
}
```

Figure 2 - LED initializations

From Figure 2, you can see our LED initializations. We used 6 LEDs for the bonus part which are connected to Port B.

- **DAC_Init**

```
// **************DAC_Init*********************
// Initialize 3-bit DAC
// Input: none
// Output: none
void DAC_Init(void){unsigned long volatile delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD; // activate port D
  delay = SYSCTL_RCGC2_R; // allow time to finish activating
  GPIO_PORTD_AMSEL_R &= ~0x0F; // no analog
  GPIO_PORTD_PCTL_R &= ~0x0000FFFF; // regular GPIO function
  GPIO_PORTD_DIR_R |= 0x0F; // make PD-0 out
  GPIO_PORTD_AFSEL_R &= ~0x0F;// disable alt funct on PD2-0
  GPIO_PORTD_DEN_R |= 0x0F;// enable digital I/O on PD2-0
}
```

Figure 3 - DAC initialization

From Figure 3, you can see the DAC initialization. Port D is used to initialize DAC.

- **Sound_Init**

```
// **************Sound_Init*********************
// Initialize Systick countic interrupts
// Input: interrupt count
//        Units of count are 12.5ns
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void Sound_Init(void){
  DAC_Init();  // Port D is DAC
  Index = 0;
  NVIC_ST_RELOAD_R = 2-1;// reload value
  NVIC_ST_CTRL_R = 0; // disable SysTick during setup
  NVIC_ST_CURRENT_R = 0; // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x20000000; // priority 1
  NVIC_ST_CTRL_R = 0x0007; // enable SysTick with core clock and interrupts
}
```

Figure 4 - Sound initialization

Sound_Init function is actually the same with the initialization of SysTick. NVIC_ST_RELOAD_R value is equal to 1 because we want to call SysTick handler as soon as possible after the code runs. Then, according to the note values, RELOAD value is updating which can be seen in Figure 7.

- **DAC_Out**

```
// **************DAC_Out*********************
// output to DAC
// Input: 3-bit data, 0 to 7
// Output: none
void DAC_Out(unsigned long data){
  GPIO_PORTD_DATA_R = data;
}
```

Figure 5 - DAC_Out()

DAC_Out function simply takes data and assigns it to port that is connected with the LEDs.

- **Main**

```
int main(void) {

  ADC0_Init();
  LED_Init();
  Sound_Init();

  while(1){
    WaitForInterrupt();
  }
}
```

Figure 6 - Main function

Init functions are called in main and SysTick interrupt is waited in an infinite while loop.

- **SysTick_Handler**

SysTick_Handler can be seen from Figure 7. To be able to change the note and change the brightness of a LED, angle degree must be stored in a variable which is named angle1 and angle2 respectively. Value of potentiometer is defined 0 to 4095. This value is assigned at a rate of 270 degrees. Therefore angle can be calculated as $angle = (ADCvalue * 270) / 4095$. After calculating the angle, according to it, RELOAD value of the SysTick changes. Details of the bonus part of the lab will be explained later in this report.

```c
void SysTick_Handler(void) {
    count++;
    ADCvalue1 = ADC0_In(); // ADC value for sound
    Index = (Index+1)&0x1F; // index goes through 0 to 31
  DAC_Out(SineWave[Index]); // output one value each interrupt
    angle1 = (ADCvalue1 * 270) / 4095;

    if(angle1 <= 34){  // DO
        NVIC_ST_RELOAD_R = DO-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 68){  //RE
        NVIC_ST_RELOAD_R = RE-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 102){  //MI
        NVIC_ST_RELOAD_R = MI-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 136){  //FA
        NVIC_ST_RELOAD_R = FA-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 170){  //SOL
        NVIC_ST_RELOAD_R = SOL-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 204){  //LA
        NVIC_ST_RELOAD_R = LA-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 238){  //SI
        NVIC_ST_RELOAD_R = SI-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    else if(angle1 <= 270){  //DO
        NVIC_ST_RELOAD_R = DO_-1;
        NVIC_ST_CTRL_R = 0x0007;
    }
    if(count == period) {
        ADCvalue2 = ADC0_In();

        angle2 = (ADCvalue2 * 270) / 4095;
        max_open_leds = (angle2 > 265) ? 6 : angle2 / 45;
        brightness = (angle2 % 45);

        // Duration of HIGH
        duty_cycle = (brightness * period) / 45;
        count = 0;
    }

    if(duty_cycle > count) { // Duration of HIGH
        for(i = 0; i < max_open_leds; i++) {
            GPIO_PORTB_DATA_R |= leds[i];   // Turn on LEDs
        }
    } else {  // Duration of LOW
        GPIO_PORTB_DATA_R &= ~leds[max_open_leds - 1];   // Turn off LEDs
    }
}
```

Figure 7 - SysTick_Handler

## Board Pictures

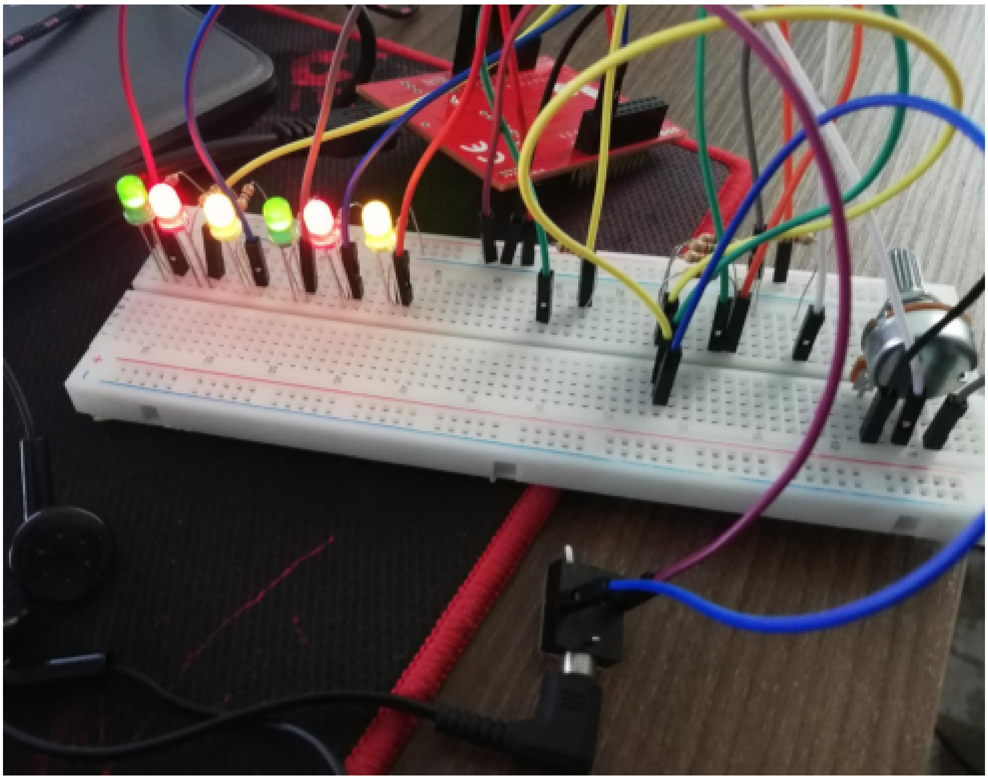

Figure 8 - Max brightness of 6 LEDs

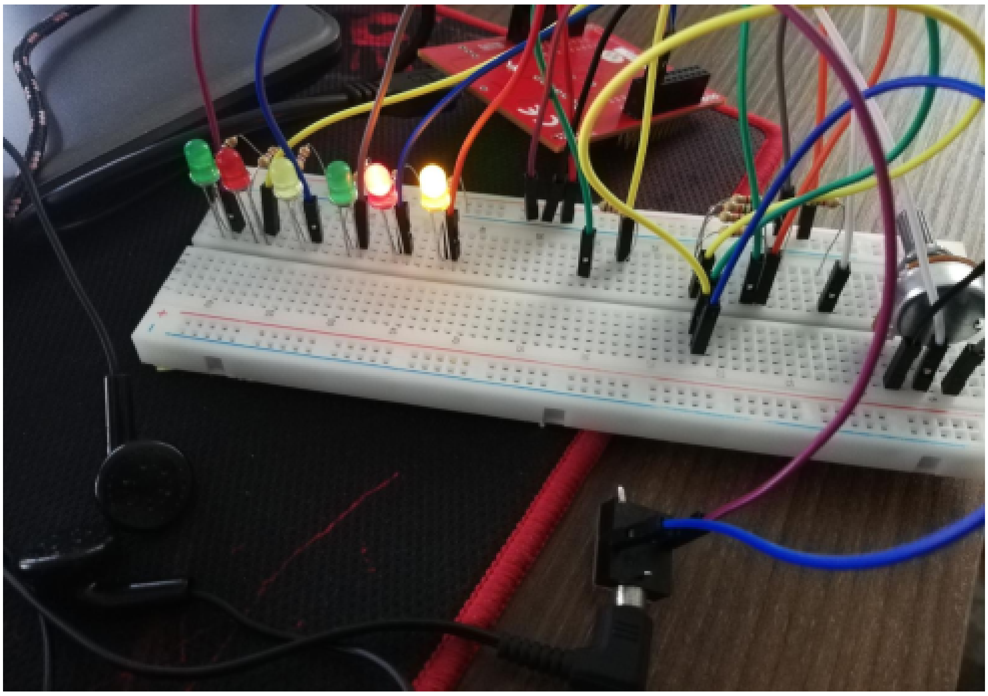

Figure 9 - Max brightness first two LEDs

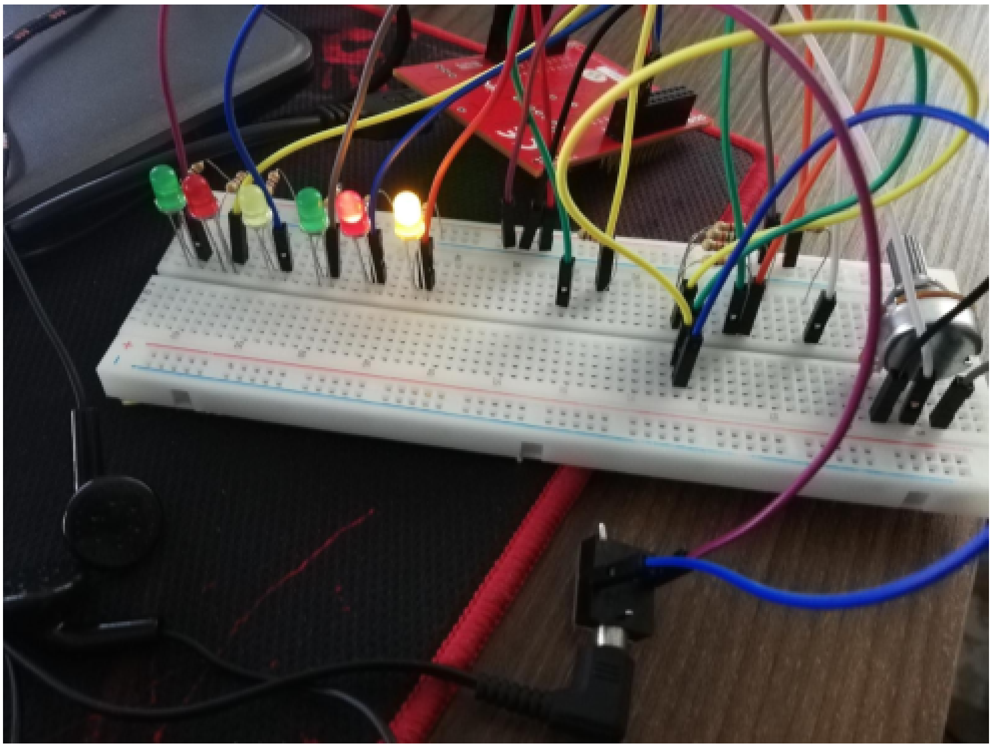

Figure 10 - Half brightness of second LED

**Theoretical Information**

- **Pulse-Width Modulation(PWM):**

It is basically, a square wave with a varying high and low time.PWM has many applications such as controlling servos and speed controllers, limiting the effective power of motors and LEDs.

- **Duty Cycle:**

The percentage of time in which the PWM signal remains high(on-time) is called as a duty cycle. If the signal is always on, it is in 100% duty cycle and if it is always off, it is 0% duty cycle.

*duty_cycle = (brightness * period) / 45*

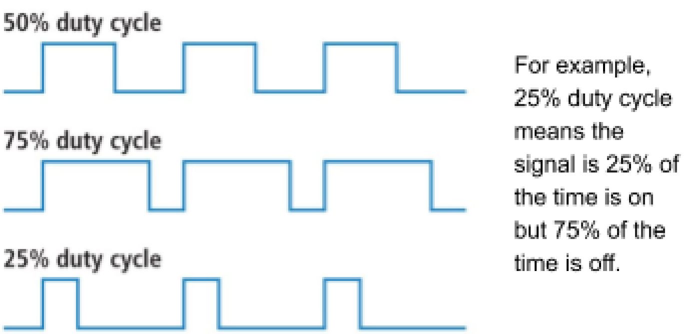


Figure 11 - Duty Cycle example

**Bonus Part Explanation**

For the bonus part, we used 6 LEDs. Their brightness should be changed according to the angle of a potentiometer. Since there are 6 LEDs, their brightness should be maximum for every 45(270 / 6) degree.

To be able to calculate the brightness of the LEDs,

***brightness = (angle2 % 45);***

is used and the number of LEDs that have maximum brightness is calculated using:

***max_open_leds = (angle2 > 265) ? 6 : angle2 / 45;***

Since the result of the division of numbers between 265 and 270 by 45 is not 6, we hardcoded it.

Also, a period is set in this part to be able to make PWM. During this period, duration of high and low state of the LEDs should be calculated.

After calculating the duration of high state, during this duration, LEDs must be open according to the angle and must be closed during the duration of low state.

The most important part of this part is setting a short period. If period is too long, blinking on the LEDs is shown and it is not a situation that we want to face.