HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER ENGINEERING
BBM434: EMBEDDED SYSTEMS LABORATORY

LABORATORY ASSIGNMENT – 7 REPORT

ANALOG TO DIGITAL CONVERSION and PWM

ŞERAFETTİN BERK SEVGİ 21328414

CEMAL ÜNAL 21328538

OBJECTIVE

In this experiment, it is requested to turn on or turn off some LEDs using a potentiometer and control the brightness of these LEDs using **P**ulse-**W**idth **M**odulation (PWM).

THEORETICAL BACKGROUND FOR THE LAB

Before explaining the methods of this experiment, it will be good to give information about some concepts to be used in this experiment.

A **LED** (light-emitting diode) is a two-lead semiconductor light source. It emits light when a suitable current is applied to the LED.

A **potentiometer** is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. Potentiometer is essentially a voltage divider used for measuring voltage. Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. In this experiment potantiometer is used to when to turn on or when to turn off the LEDs.

Sample visual of potentiometer that is used in this experiment

Sample visual of LED that is used in this experiment

In electronics, an **analog-to-digital converter** (ADC) is a system that converts an analog signal, such as a sound picked up by a microphone, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current.

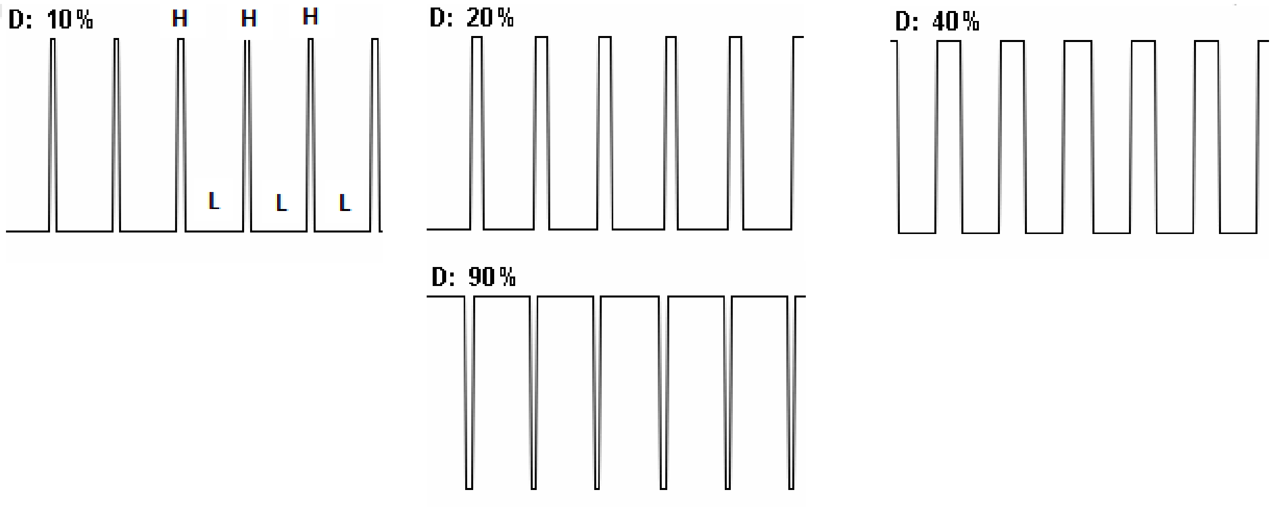
Pulse-**W**idth **M**odulation (PWM) is a modulation technique that is used to encode a message into a pulsing signal. Its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors. The basic idea of PWM is to create a digital output wave of fixed frequency, but allow the microcontroller to vary its duty cycle.

A **duty cycle** is the fraction of one period in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off cycle. A duty cycle can be formulated as:

$$D = PW/T$$

D is Duty Cycle
PW is pulse width
T is total period of the signal

For instance 40% duty cycle means the signal is 40% of the time is on but 60% of the time is off. To explain duty cycle graphically, below figures can be considered:



Graph representations according to different duty cycle ratios.

To represent the duty cycle with different formula, above graphs can be considered. The system is designed in such a way that H+L is constant. The duty cycle is defined as the fraction of time the signal is high. The second formula can be written as with this informations:

$$DutyCycle = H/(H+L)$$

Duty cycle varies from 0 to 1.

PROCEDURE AND RESULTS

In this experiment lightning of the LEDs are controlled by rotating a potentiometer. Since the potentiometer provides analog signals to the circuit, these signals are need to be converted from analog to digital.

<u>Connection</u>

First pin of potentiometer is connected to 3.3V, middle pin is connected to PE2, and the last pin of potentiometer is connected to the ground.

<u>Calculation of Lightning LED Number</u>

Value of potentiometer varies approximately from 0 to 4095 with rotation angle goes from 0 to 270 degrees. Since six LEDs are used in this experiment, 270 degrees must be divided into equal parts. After this division operation (270/6), the result will be 45. This means, at every 45 degrees rotation of potentiometer one LED will be on. (i.e. at 45 degrees only first LED will be on, at 90 degrees first and second LEDs will be on, and finally at 270 degrees all LEDs will be on.) When the degree of the potentiometer is 270 degrees, if its rotated in the reverse way (from 270 degrees angle to 0 degree angle) LEDs start to turning off one by one. At the angle 0 all LEDs will be off.

To perform this operation, first the **angle of rotation** must be measured. To measure the angle, following formula can be used:

$$angle = (ADCvalue * 270)/4095;$$

angle is the angle of rotation
ADCvalue is the current value of the potentiometer
270 is the total angle degree of potentiometer
4095 is the upper value of potentiometer

After obtaining the angle value of the potentiometer **number of LEDs will be lighted** must be found. To measure this number, following formula can be used:

$$NumOfFullBrightnessLeds = angle/45;$$

NumOfFullBrightnessLeds is the number of LEDs to be lighted
angle is the angle of rotation

Measuring the Brightness of the LEDs Using PWM

Rotation of potentiometer and brightness control of the LEDs are done in the Systick_Handler. Reload value of the Systick is set to 160. This means every 0.01 seconds Systick_Handler is called.

At each call of Systick, a counter value is incremented by 1. When this counter value reach to 200 value of potentiometer is read. **Angle** value, **number of LEDs** to be lighted, **percentage of brightness** and **PWM ON duration**(Duration of High) are also measured here. Before leaving this section, counter value value set to 0.

If PWM ON duration value is greater than the counter value, then turn on the related LEDs according to the angle of the potentiometer. If PWM ON duration value is **not** greater than the counter value then turn off the related LEDs.

VIDEO OF LABORATORY ASSIGNMENT DEMO

Video of this experiment can be found at the following link: https://youtu.be/JufBca99bes

CODE

```
#include "..//tm4c123gh6pm.h"
#include "TExaS.h"

void EnableInterrupts(void);
void WaitForInterrupt(void);

void LED_Init(void);

void SysTick_Init(unsigned long period);
void SysTick_Handler(void);

void ADC0_Init(void);
unsigned long ADC0_In(void);

unsigned long ADCvalue;

unsigned int counter = 0;
unsigned int i = 0;

unsigned int angle = 0;
unsigned int num_of_full_brightness_leds = 0;
unsigned int brightness_percent = 0;
unsigned int on_percentage = 0;

unsigned long pwm_on_duration = 0;

unsigned long leds[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20};

int main(void) {

        ADC0_Init();
        LED_Init();

        SysTick_Init(160);

        EnableInterrupts();

        while(1){
                WaitForInterrupt();
        }
}

/*
        Initializes PORTB0-5 pins as output
        Parameter: None
        Returns: None
*/
void LED_Init(void){
        volatile unsigned long delay;
        SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB; // 1) activate clock for Port B
        delay = SYSCTL_RCGC2_R; // allow time for clock to start
        GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R&0xFFFFFF0F)+0x00000000;// 3)
regular GPIO
        GPIO_PORTB_AMSEL_R &= ~0x3F; // 4) disable analog function on PB0-5
        GPIO_PORTB_DIR_R |= 0x3F;       // 5) set direction to output
        GPIO_PORTB_AFSEL_R &= ~0x3F; // 6) regular port function
        GPIO_PORTB_DEN_R |= 0x3F; // 7) enable digital port
}
```

```c
/*
	Initializes SysTick Timer
	Parameter: Clock cycle to wait
	Returns: None
*/
void SysTick_Init(unsigned long period){
	NVIC_ST_CTRL_R = 0; // disable SysTick during setup
	NVIC_ST_RELOAD_R = period - 1; // reload value
	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 = 0x07; // enable SysTick with core clock and interrupts
}

/*
	Handles SysTick Timer interrupts
	Decides how many LEDs will be turned on
	Decides the brightness values of not fully lighted LEDs
	Parameter: None
	Returns: None
*/
void SysTick_Handler(void) {
	counter++;
	// At each 2 ms
	if(counter == 200) {
		// Current value of potentiometer
		ADCvalue = ADC0_In();

		// Angle value of the current rotation
		angle = (ADCvalue * 270) / 4095;

		// How many LEDs will be turned on
		// Since the result value is rounded, we do not want it.
		// Thats why this if statement is here.
		if(angle > 265)
			num_of_full_brightness_leds = 6;
		else
			num_of_full_brightness_leds = angle / 45;

		// Brightness value of LEDs that are not fully lighted.
		brightness_percent = (angle % 45);

		// Duration of ON
		pwm_on_duration = ((angle % 45) * 200) / 45;

		counter = 0;
	}

	if(pwm_on_duration > counter) {
		// Turn on LEDs
		for(i = 0; i < num_of_full_brightness_leds; i++) {
			GPIO_PORTB_DATA_R |= leds[i];
		}
	} else { // Turn off LEDs
		GPIO_PORTB_DATA_R &= ~leds[num_of_full_brightness_leds - 1];
	}
}
```

```c
/*
        Initialization function to set up the ADC
        SS3 triggering event: software trigger
        SS3 1st sample source:  channel 1
        SS3 interrupts: enabled but not promoted to controller
        Parameter: None
        Returns: None
*/
void ADC0_Init(void){
        volatile unsigned long delay;
        SYSCTL_RCGC2_R |= 0x00000010; // 1) activate clock for Port E
        delay = SYSCTL_RCGC2_R; //      allow time for clock to stabilize
        GPIO_PORTE_DIR_R &= ~0x04; // 2) make PE2 input
        GPIO_PORTE_AFSEL_R |= 0x04; // 3) enable alternate function on PE2
        GPIO_PORTE_DEN_R &= ~0x04; // 4) disable digital I/O on PE2
        GPIO_PORTE_AMSEL_R |= 0x04; // 5) enable analog function on PE2
        SYSCTL_RCGC0_R |= 0x00010000; // 6) activate ADC0
        delay = SYSCTL_RCGC2_R;
        SYSCTL_RCGC0_R &= ~0x00000300; // 7) configure for 125K
        ADC0_SSPRI_R = 0x0123; // 8) Sequencer 3 is highest priority
        ADC0_ACTSS_R &= ~0x0008; // 9) disable sample sequencer 3
        ADC0_EMUX_R &= ~0xF000; // 10) seq3 is software trigger
        ADC0_SSMUX3_R = (ADC0_SSMUX3_R&0xFFFFFFF0)+1;   //? set channel Ain9
(PE2)
        ADC0_SSCTL3_R = 0x0006; // 12) no TS0 D0, yes IE0 END0
        ADC0_ACTSS_R |= 0x0008; // 13) enable sample sequencer 3
}

/*
        Busy-wait Analog to digital conversion
        Parameter: None
        Returns: 12-bit result of ADC conversion
*/
unsigned long ADC0_In(void){
        unsigned long result;
  ADC0_PSSI_R = 0x0008; // 1) initiate SS3
        while((ADC0_RIS_R&0x08)==0){}; // 2) wait for conversion done
        result = ADC0_SSFIFO3_R&0xFFF; // 3) read result
        ADC0_ISC_R = 0x0008; // 4) acknowledge completion
        return result;
}
```