GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

Lab #4 Experiment Report

Prepared by
200102002051 – Arda DERİCİ
200102002025 – Umut Mehmet ERDEM
200102002061 – Serdar BAŞYEMENİCİ

1. Introduction

The aim of this laboratory is to enable communication between the PC and MCU, utilizing bidirectional data transmission and parsing techniques. In the first problem, the focus is on exploring the functionality of UART (Universal Asynchronous Receiver-Transmitter) and understanding its designated purpose. The task involves utilizing the UART protocol to write to the console, essentially involving an investigation into the practical application and implementation of UART for communication purposes. The research focuses on understanding the PWM (pulse width modulation) signal and learning how to manipulate it to control LED brightness. So when the PWM signal is driven, the brightness of the LED can be adjusted softly using varying duty cycles. in the next problem, the implementation involves utilizing a PWM signal to drive an LED at variable speeds, with the duty cycle set using a keypad. For this, the connections of the rows and columns in the keypad hardware are investigated.

2. Problems

2.1. Problem I

In this problem, it is connected board to the PC using UART protocol.

2.1.1. Flow Chart

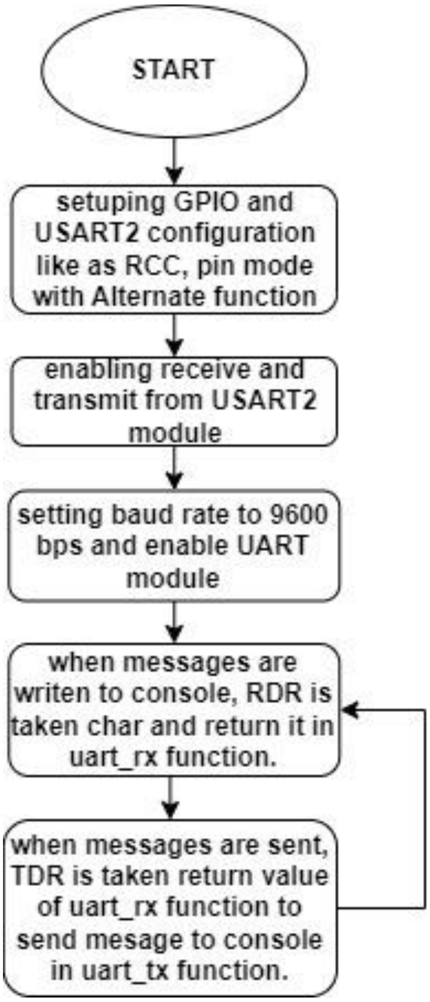


Figure 1 - Flow chart for Problem I.

2.1.2. Theoretical and Mathematical Work

$$\text{Tx/Rx baud} = \frac{f_{CK}^{\;\text{Peripheral Clock}}}{8 \times \underbrace{\text{USARTDIV}}_{\text{Divide factor to generate different baud rates}}} \qquad \text{if } OVER8 = 1$$

$$\text{Tx/Rx baud} = \frac{f_{CK}}{16 \times \text{USARTDIV}} \qquad \text{if } OVER8 = 0$$

Figure 2 - Calculation of 9600 baud rate.

2.1.3. C Code of the Problem I

C code for Problem I is as follows:

```
/* author: Umut Mehmet ERDEM | Arda DERİCİ | Serdar BAŞYEMENİCİ
* problem1.c
*/

#include "stm32g0xx.h"

void GPIO_Config(void);
void USART2_Config(void);
void uart_tx(uint8_t c);
uint8_t uart_rx(void);
int _print(int f, char *ptr, int len);
void print(char *s);

int main(void){

	GPIO_Config();
	USART2_Config();

	while(1){
		uart_tx(uart_rx()); // received characters sent via console are
printed via transmitter
	}

	return 0;
}

void printChar(uint8_t c){
	while(!(USART2->ISR & USART_ISR_TXE_TXFNF)); // when messages are sent.
	USART2->TDR = c; // Transmit data register is taken character to send a
message.
}
```

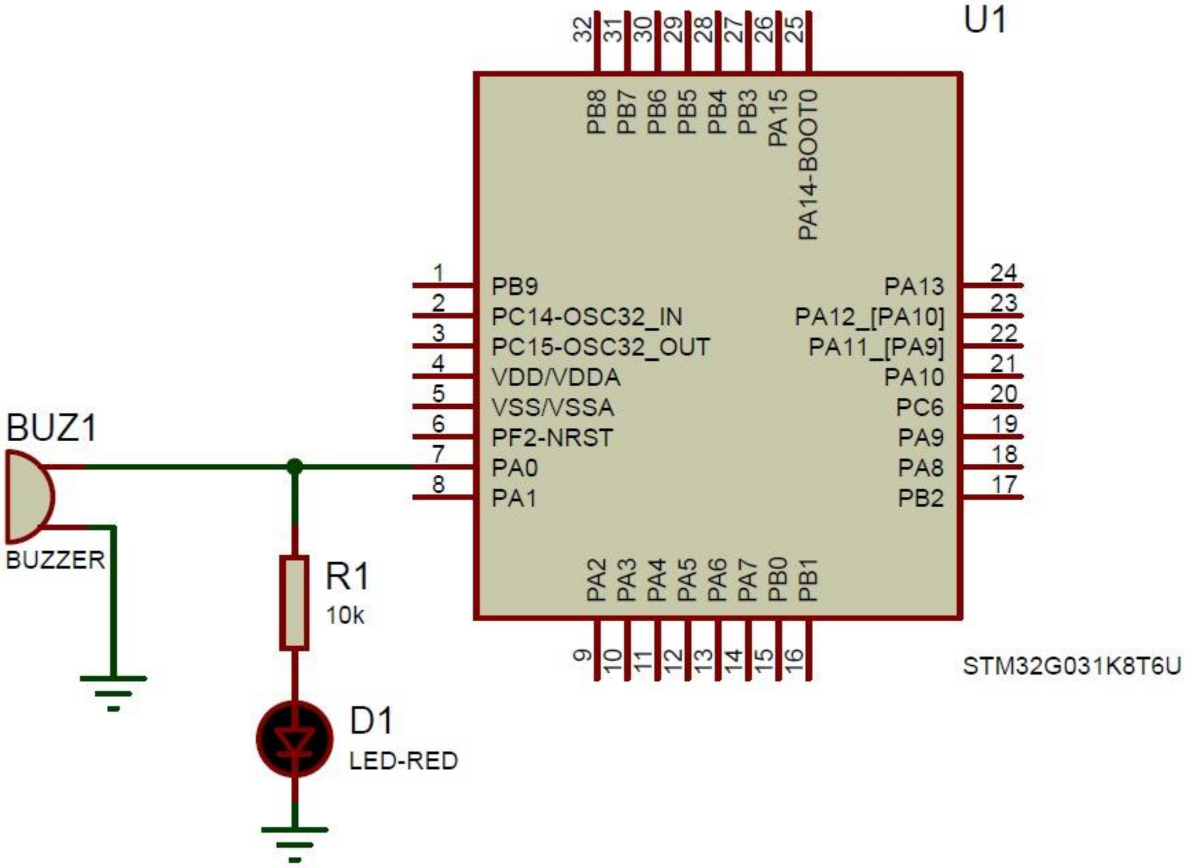
```
int _print(int f, char *ptr, int len)
{
	/*in for loop, i of is increasing until equal to len
	 * and meanwhile, chars of 2nd parameter of _print function is writen
	 * into the printChar character by character increasing ptr of 2nd
parameter
	 * of _print function  */
	for(volatile int i = f; i<len; i++){
		printChar(*ptr);
		ptr++;
	}
	return len; // return length
}

void print(char *s){
	int length = 0; // to count length of character
	/* i is pointer of string and length is increasing until i equals NULL
character*/
	for(char *i = s; *i != NULL; i++) length++;
	_print(0, s, length);
}

uint8_t uart_rx(void){
	while(!( USART2->ISR & USART_ISR_RXNE_RXFNE)); // when messages are
detected.
	return (uint8_t)USART2->RDR; // RDR[8:0]: Receive data value
}

void uart_tx(uint8_t c){
	printChar(c); // printing character by character
}

void GPIO_Config(void){
	// input-output A port clock enable
	RCC->IOPENR |= RCC_IOPENR_GPIOAEN;
	/* modes of GPIOA PA2 and PA3 pins are selected as alternate function.
	 * like that 0b1111_1010_1111;*/
	GPIOA->MODER &= ~((3U << 2*2) | (3U << 2*3));
	GPIOA->MODER |= (2U << 2*2) | (2U << 2*3);

	/* PA2 and PA3 pins used for USART2_TX and USART2_RX are selected
	 * with GPIOx_AFRL = AFRL_AFSELy(Alternate Function register -
	 * Alternate function selection for port x pin y)
	 * AF1 -->> USART2_RX, USART2_TX*/
	GPIOA->AFR[0] |= GPIO_AFRL_AFSEL2_0;
	GPIOA->AFR[0] |= GPIO_AFRL_AFSEL3_0;
}
```

```
void USART2_Config(void){
      RCC->APBENR1 |= RCC_APBENR1_USART2EN; // RCC APB peripherals clock
enable for USART2
      USART2->CR1 = 0x00; // clear all
      USART2->CR1 |= USART_CR1_UE; // UE: USART enable

      /* Baud rate of 9600, PCLK1 at 16 MHz
       * TX/RX baud rate = f_clk/(16*USARTDIV)
       * 9600 = 16MHz/(16*USARTDIV) --->>> USARTDIV = 104.1666667
       * IEEE754 floating-point --->>> mantissa = 104, fraction = 0.167*16 =
2.672 ≈ 3*/
      USART2->BRR |= (3 << 0) | (104 << 4);

      USART2->CR1 |= USART_CR1_RE; // RE: Receiver enable
      USART2->CR1 |= USART_CR1_TE; // TE: Transmitter enable
}
```

2.2. Problem II

In this problem, it is implemented a PWM (pulse width modulation) signal and drive an external LED using varying duty cycles.

2.2.1. Schematic Diagram and Flow Chart



Figurel 3 - Schematic diagram for Problem II.

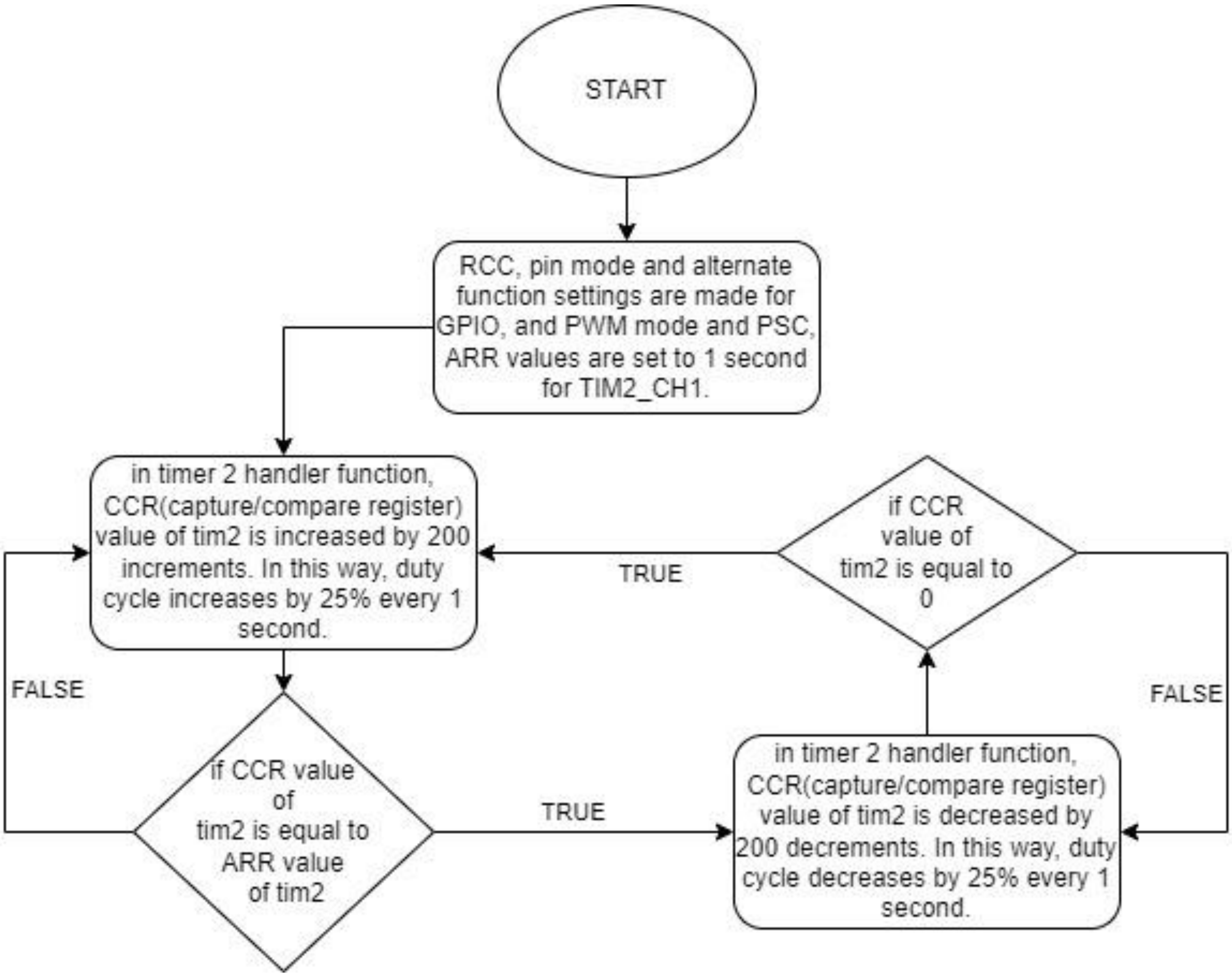


Figure 4 – Flow chart for Problem II.

2.2.2. Theoretical and Mathematical Work

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

Figure 5 - Calculation of timer.

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

Figure 6 - Calculation of PWM.

2.2.3. C Code of the Problem II

```
/* author: Umut Mehmet ERDEM | Arda DERİCİ | Serdar BAŞYEMENİCİ
* problem2.c
*/
#include "stm32g0xx.h"

void GPIO_Config(void);
void TIM2_Config(void);

uint8_t limit_flag = 0;

void TIM2_IRQHandler(void) {
    // PWM Duty Cycle[%] = (CCRx/ARR)*100;

      /* capture-compare value of TIM2 is increasing until
       *  equal to period value of TIM2, when TIM2_CCR is equal to TIM2_ARR,
       *  limit_flag is 1 and TIM2_CRR is decreasing until equal to 0.*/
      if(!(limit_flag) && TIM2->CCR1 < TIM2->ARR){
            TIM2->CCR1+=200;
            if(TIM2->CCR1 >= TIM2->ARR) limit_flag = 1;
      }
      else if(limit_flag && TIM2->CCR1>0){
            TIM2->CCR1-=200;
            if(TIM2->CCR1<=0) limit_flag = 0;
      }

    TIM2->SR &= ~(1U << 0); // Clear update status register
}
int main(void){

      GPIO_Config();
      TIM2_Config();

      while(1){
      }

      return 0;
}
void GPIO_Config(void){
      // input-output A port clock enable
      RCC->IOPENR |= RCC_IOPENR_GPIOAEN;
      // select PA0 mode as Alternate Function
      GPIOA->MODER &= ~(3U << 2*0);
      GPIOA->MODER |= (2U << 2*0);

      /* PA0 pin used for TIM2_CH1 are selected
       * with GPIOx_AFRL = AFRL_AFSELy(Alternate Function register -
       * Alternate function selection for port x pin y)
       * AF2 -->> TIM2_CH1*/
      GPIOA->AFR[0] |= GPIO_AFRL_AFSEL0_1;
}
```

```
void TIM2_Config(void){
      RCC->APBENR1 |= RCC_APBENR1_TIM2EN; // Timer 2 clock enable

      TIM2->CR1 = 0; // zero out the control register just in case
      TIM2->CR1 = TIM_CR1_ARPE; // Auto-reload preload enable

      TIM2->CCMR1 |= (6U << 4); // PWM mode 1 is selected.
      TIM2->CCMR1 |= TIM_CCMR1_OC1PE; // Output Compare 1 Preload Enable

      TIM2->CCER |= TIM_CCER_CC1E; // Capture compare ch1 enable

      TIM2->CNT = 0; // zero out counter

      // tim update freq = TIM_CLK/((TIM_PSC+1)*TIM_ARR) for 1 s interrupt
      TIM2->PSC = 9; // prescaler
      TIM2->ARR = 16000; // period

      TIM2->CCR1 = 0; // zero out duty for ch1 in TIM capture-compare register 1

      // Update Generation: Re-initialize the counter and generates an update
of the registers.
      TIM2->EGR |= TIM_EGR_UG;

      TIM2->DIER |= TIM_DIER_UIE; // Update interrupt enable

      TIM2->CR1 |= TIM_CR1_CEN; // TIM2 Counter enable

      NVIC_SetPriority(TIM2_IRQn, 1); // Setting Priority for timer handler
      NVIC_EnableIRQ(TIM2_IRQn); // timer handler enable
}
```

C code of Problem II is as above.

2.3. Problem III

In this problem, it is implemented a PWM signal and drive an LED at different speeds. It is used keypad to set the duty cycle.

2.3.1. Flow Chart and Schematic Diagram

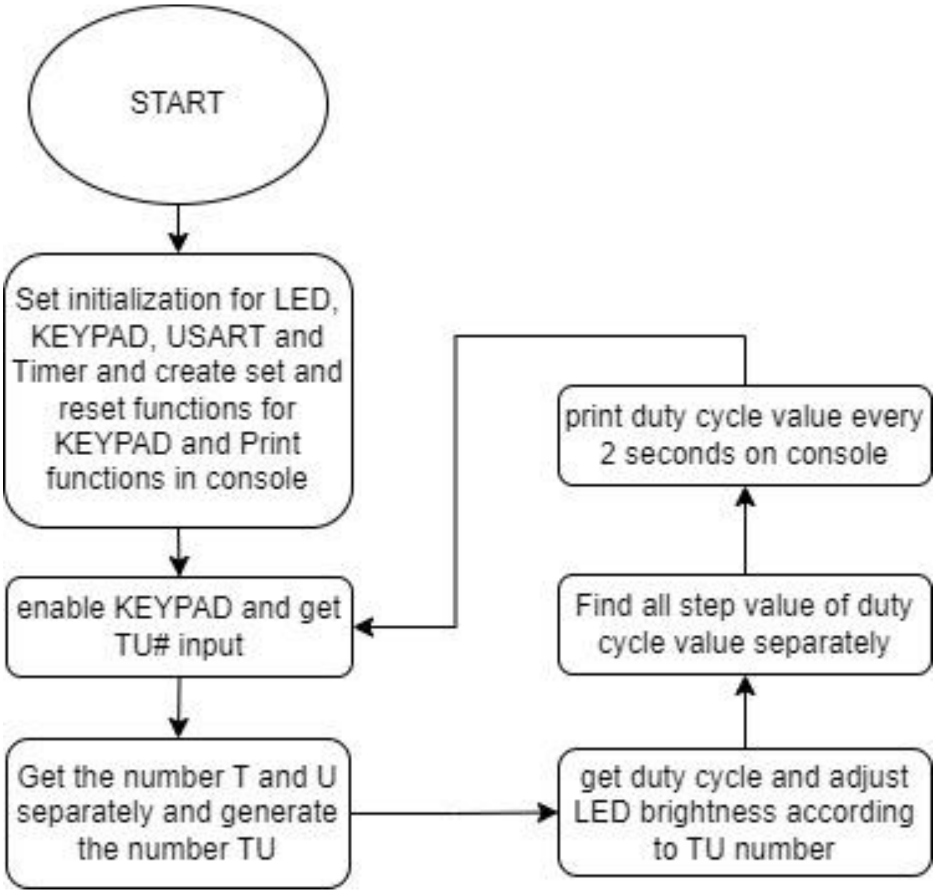


Figure 7 - Flow chart for Problem III.

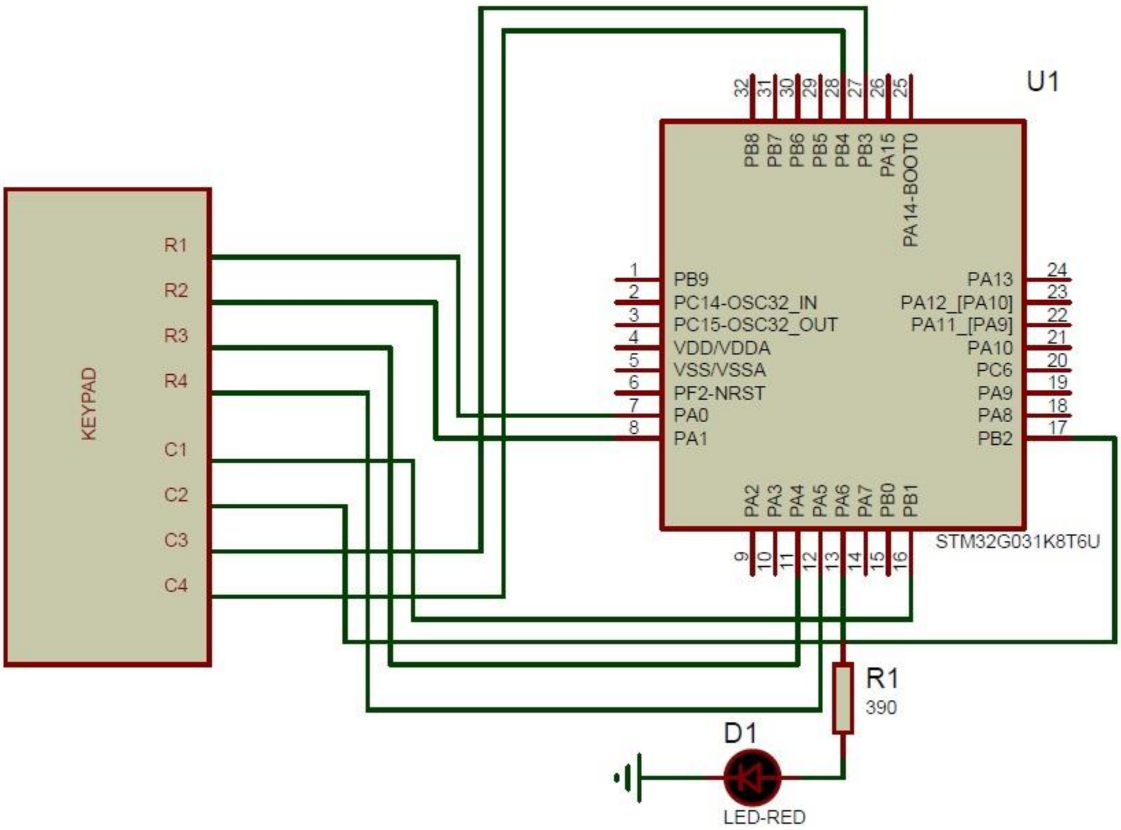


Figure 8 - Schematic diagram for Problem III.

2.3.2. Theoretical Research

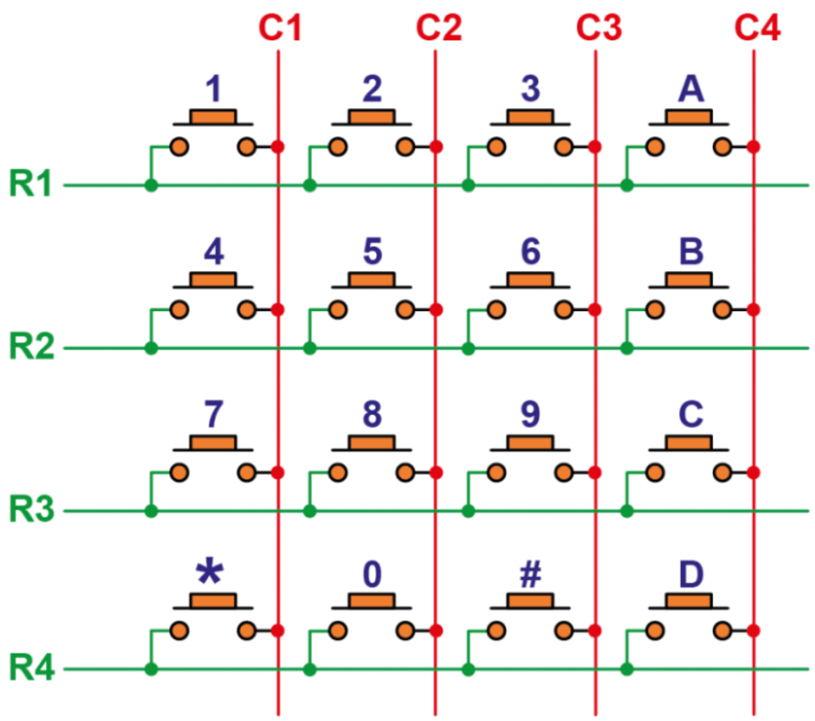


Figure 9 - 4x4 keypad hardware.

2.3.3. C Code of the Problem III

```
/* author: Umut Mehmet ERDEM | Arda DERİCİ | Serdar BAŞYEMENİCİ
* problem3.c
*/

#include "stm32g0xx.h"

uint32_t ten_thousands, thousands, hundreds, tens;
uint32_t number;
uint32_t is_ten_digit = 1;
uint8_t ten = 0;
uint8_t unit = 0;
uint32_t counter = 0;
uint32_t dutyCycle = 0;

void USART2_Config(uint32_t);
void keypad_Config(void);
void TIM3_Config(void);
uint8_t conIntToAlp(uint8_t);
void printChar(uint8_t);
void SetKeypad(void);
void ResetKeypad(void);
void delay(volatile uint32_t);

int main(void) {

	ResetKeypad();
	TIM3_Config();
	keypad_Config();
	USART2_Config(9600);
	SetKeypad();

    while(1) {
    }

    return 0;
}

uint8_t conIntToAlp(uint8_t a){
	if(a==0)
		return 48;
	else if(a==1)
		return 49;
	else if(a==2)
		return 50;
	else if(a==3)
		return 51;
	else if (a==4)
		return 52;
	else if(a==5)
		return 53;
```

```
	else if (a==6)
		return 54;
	else if(a==7)
		return 55;
	else if(a==8)
		return 56;
	else return 57;
}

void TIM3_IRQHandler(){
	counter+=10; // printing to the console every 2 seconds

	if(counter==2000){
		delay(2000);
		number=TIM3->CCR1;

		ten_thousands=(number)/10000;
		printChar(conIntToAlp(ten_thousands));

		number -= (ten_thousands * 10000);
		thousands=number/1000;
		printChar(conIntToAlp(thousands));

		number -= (thousands * 1000);
		printChar(conIntToAlp(hundreds));

		number -= (hundreds * 100);
		tens = (number/10);
		printChar(conIntToAlp(tens));

		number -= (tens * 10);
		printChar(conIntToAlp(number));
		counter=0;
	}

	TIM3->SR &=  ~(1U<<0);
	TIM3->SR &=  ~(1U<<1);
}

void EXTI0_1_IRQHandler(){
	ResetKeypad();

	GPIOA->ODR |=(1<<0);
	if((GPIOB->IDR>>1)&1){
		if(is_ten_digit==1){ // KEYPAD button control for 1 - first column
			ten=1;
			is_ten_digit=0;
		}
		else{
			unit=1;
			is_ten_digit=1;
		}
	}
```

```
      else{ // KEYPAD button control for 4
            GPIOA->ODR &=  ~(1U <<0);
            GPIOA->ODR |=(1<<1);
            if((GPIOB->IDR>>1)&1){

                  if(is_ten_digit==1){
                        ten=4;
                        is_ten_digit=0;
                  }
                  else{
                        unit=4;
                        is_ten_digit=1;
                  }

            }

            else { // KEYPAD button control for 7
                  GPIOA->ODR &=  ~(1U <<1);
                  GPIOA->ODR |=(1<<4);
                  if((GPIOB->IDR>>1)&1){
                        if(is_ten_digit==1){
                              ten=7;
                              is_ten_digit=0;
                        }
                        else{
                              unit=7;
                              is_ten_digit=1;
                        }
                  }

                  else{
                        GPIOA->ODR &=  ~(1U <<4);
                        GPIOA->ODR |=(1<<5);
                        if((GPIOB->IDR>>1)&1){
                        }
                  }
            }
      }

      SetKeypad();
      EXTI->RPR1 |=(1<<1);
}

void EXTI4_15_IRQHandler(){
      ResetKeypad();

      GPIOA->ODR |=(1<<0);
      if((GPIOB->IDR>>4)&1){ // KEYPAD button control for A
      }
```

```
	else{
		GPIOA->ODR &=  ~(1U <<0);
		GPIOA->ODR |=(1<<1);
		if((GPIOB->IDR>>4)&1){ // KEYPAD button control for B
		}
		else {
			GPIOA->ODR &=  ~(1U <<1);
			GPIOA->ODR |=(1<<4);
			if((GPIOB->IDR>>4)&1){ // KEYPAD button control for C
			}
			else{
				GPIOA->ODR &=  ~(1U <<4);
				GPIOA->ODR |=(1<<5);
				if((GPIOB->IDR>>4)&1){ // KEYPAD button control for D
				}
			}
		}
	}

	SetKeypad();
	EXTI->RPR1 |=(1<<4);
}

void EXTI2_3_IRQHandler(){
	ResetKeypad();
	if((EXTI->RPR1>>2)&1){ // KEYPAD button control for 2 - 2nd column
		GPIOA->ODR |=(1<<0);
		if((GPIOB->IDR>>2)&1){
			if(is_ten_digit==1){
				ten=2;
				is_ten_digit=0;
			}
			else{
				unit=2;
				is_ten_digit=1;
			}
		}

		else{
			GPIOA->ODR &=  ~(1U <<0);
			GPIOA->ODR |=(1<<1);
			if((GPIOB->IDR>>2)&1){ // KEYPAD button control for 5
				if(is_ten_digit==1){
					ten=5;
					is_ten_digit=0;
				}
				else{
					unit=5;
					is_ten_digit=1;
				}
			}
```

```
			else {
				GPIOA->ODR &=  ~(1U <<1);
				GPIOA->ODR |=(1<<4);
				if((GPIOB->IDR>>2)&1){ // KEYPAD button control for 8
					if(is_ten_digit==1){
						ten=8;
						is_ten_digit=0;
					}
					else{
						unit=8;
						is_ten_digit=1;
					}
				}

				else{
					GPIOA->ODR &=  ~(1U <<4);
					GPIOA->ODR |=(1<<5);
					if((GPIOB->IDR>>2)&1){ // KEYPAD button control
for 0
						if(is_ten_digit==1){
							ten=0;
							is_ten_digit=0;
						}
						else{
							unit=0;
							is_ten_digit=1;
						}
					}
				}
			}
		}

		EXTI->RPR1 |=(1<<2);
	}
	else{
		GPIOA->ODR |=(1<<0);
		if((GPIOB->IDR>>3)&1){
			if(is_ten_digit==1){ // KEYPAD button control for 3 - 3rd
column
				ten=3;
				is_ten_digit=0;
			}
			else{
				unit=3;
				is_ten_digit=1;
			}
		}
		else{
			GPIOA->ODR &=  ~(1U <<0);
			GPIOA->ODR |=(1<<1);
```

```
if((GPIOB->IDR>>3)&1){
                        if(is_ten_digit==1){ // KEYPAD button control for 6
                              ten=6;
                              is_ten_digit=0;
                        }
                        else{
                              unit=6;
                              is_ten_digit=1;
                        }
                  }

                  else {
                        GPIOA->ODR &=  ~(1U <<1);
                        GPIOA->ODR |=(1<<4);
                        if((GPIOB->IDR>>3)&1){ // KEYPAD button control for 9
                              if(is_ten_digit==1){
                                    ten=9;
                                    is_ten_digit=0;
                              }
                              else{
                                    unit=9;
                                    is_ten_digit=1;
                              }
                        }

                        else{
                              GPIOA->ODR &=  ~(1U <<4);
                              GPIOA->ODR |=(1<<5);
                              if((GPIOB->IDR>>3)&1){ // KEYPAD control for #
                                    TIM3->CCR1=(16000*((ten*10)+unit)/100);
                              }
                        }
                  }
            }

            EXTI->RPR1 |=(1<<3);
      }

      SetKeypad(); // all output is set
}

void printChar(uint8_t b){
      USART2->TDR =(uint16_t)b;
      while(!(USART2->ISR&(1<<6)));
}
void SetKeypad(void){
      GPIOA->ODR |= (1<<0);  // R1 is set
      GPIOA->ODR |= (1<<1);  // R2 is set
      GPIOA->ODR |= (1<<4);  // R3 is set
      GPIOA->ODR |= (1<<5);  // R4 is set
}
```

```
void ResetKeypad(void){
      GPIOA->ODR &=  ~(1U <<0); // R1 is reset
      GPIOA->ODR &=  ~(1U <<1); // R2 is reset
      GPIOA->ODR &=  ~(1U <<4); // R3 is reset
      GPIOA->ODR &=  ~(1U <<5); // R4 is reset
}

void delay(volatile uint32_t time){
    for(; time>0; time--);
}

void keypad_Config(void){

       GPIOA->MODER &=  ~(3U<<2*0); // PA0 - output R1
       GPIOA->MODER |=  (1U<<0);

       GPIOA->MODER &=  ~(3U <<2*1); // PA1 - output R2
       GPIOA->MODER |=  (1 << 2);

       GPIOA->MODER &=  ~(3U <<2*4); // PA4 - output R3
       GPIOA->MODER |=  (1 << 8);

       GPIOA->MODER &=  ~(3U <<2*5); //PA5 - output R4
       GPIOA->MODER |=  (1 << 10);

       GPIOB->MODER &= ~(3U << 2*1); // PB1 - input C1
       GPIOB->PUPDR |= (2U << 2*1);

       GPIOB->MODER &= ~(3U << 2*2); // PB2 - input C2
       GPIOB->PUPDR |= (2U << 2*2);

       GPIOB->MODER &= ~(3U << 2*3); // PB3 - input C3
       GPIOB->PUPDR |= (2U << 2*3);

       GPIOB->MODER &= ~(3U << 2*4); // PB4 – input C4
       GPIOB->PUPDR |= (2U << 2*4);

       EXTI->RTSR1 |=(1U<<1); // PB1 as interrupt
       EXTI->EXTICR[0] |= (1U<<8*1);
       EXTI->IMR1 |=(1<<1);
       NVIC_SetPriority(EXTI0_1_IRQn,1);
       NVIC_EnableIRQ(EXTI0_1_IRQn);

       EXTI->RTSR1 |=(1U<<2); // PB2 as interrupt
       EXTI->EXTICR[0] |= (1U<<8*2);
       EXTI->IMR1 |=(1<<2);

       EXTI->RTSR1 |=(1U<<3);// PB3 as interrupt
       EXTI->EXTICR[0] |= (1U<<8*3);
       EXTI->IMR1 |=(1<<3);
       NVIC_SetPriority(EXTI2_3_IRQn,0);
       NVIC_EnableIRQ(EXTI2_3_IRQn);
```

```
       EXTI->RTSR1 |=(1U<<4);  // PB4 as interrupt
       EXTI->EXTICR[1] |= (1U<<8*0);
       EXTI->IMR1 |=(1<<4);
       NVIC_SetPriority(EXTI4_15_IRQn,2);
       NVIC_EnableIRQ(EXTI4_15_IRQn);
}

void USART2_Config(uint32_t bdr){
      // enable GPIOA
      RCC->IOPENR |= (1U << 0);
      RCC->APBENR1 |= (1U << 17);

      GPIOA-> MODER &= ~(3U << 2*2);
      GPIOA-> MODER |=  (2U << 2*2);

      GPIOA-> AFR[0] &= ~(0xFU << 4*2);
      GPIOA-> AFR[0] |=  (1U << 4*2);

      GPIOA-> MODER &= ~(3U << 2*3);
      GPIOA-> MODER |=  (2U << 2*3);

      GPIOA-> AFR[0] &= ~(0xFU << 4*3);
      GPIOA-> AFR[0] |=  (1U << 4*3);

      USART2 -> CR1 = 0;
      USART2 -> CR1 |= (1U << 3); // Transmitter
      USART2 -> CR1 |= (1U << 2); // Receiver

      USART2 -> CR1 |= (1U << 5);

      USART2 -> BRR = (uint16_t)(SystemCoreClock / bdr);

      USART2 -> CR1 |= (1U << 0); // usart enable
}

void TIM3_Config(void){
      /* Enable GPIOB and GPIOA clock */
      RCC->IOPENR |= (3U << 0);

      /* Setup PA6 as alternate function */
      GPIOA->MODER &= ~(3U << 2*6);
      GPIOA->MODER |=  (2<< 2*6) ;

      GPIOA->AFR[0]  &= ~(0xFU<<4*6);
      GPIOA->AFR[0]  |=  (1<<4*6);

      RCC->APBENR1 |= RCC_APBENR1_TIM3EN; // Timer 3 clock enable
      TIM3->CR1 = 0;
      TIM3->CR1 |= TIM_CR1_ARPE; // Auto-reload preload enable
      TIM3->CNT = 0; // zero out counter
```

```
	// tim update freq = TIM_CLK/((TIM_PSC+1)*TIM_ARR) for 1 sec interrupt
	TIM3->PSC = 10;
	TIM3->ARR = (16000);
	TIM3->DIER |= TIM_DIER_UIE; // update interrupt enable
	TIM3->CCMR1 |= TIM_CCMR1_OC1PE; // Output Compare 1 Preload Enable

	// PWM mode 1 is selected.
	TIM3->CCMR1 &= ~ (1u<<16); //0
	TIM3->CCMR1 |=(1u<<6); //1
	TIM3->CCMR1 |= (1u<<5); //1
	TIM3->CCMR1 &= ~(1u<<4);  // 0
	TIM3->CCER |= TIM_CCER_CC1E; // Capture compare ch1 enable
	TIM3->CCR1 = 0;

	// Update Generation: Re-initialize the counter and generates an update
of the registers.
	TIM3->EGR |= TIM_EGR_UG;

	TIM3->CR1 |= TIM_CR1_CEN; // TIM3 counter enable

	NVIC_SetPriority(TIM3_IRQn,3);  // Setting Priority for timer handler
	NVIC_EnableIRQ(TIM3_IRQn); // timer handler enable
}
```

3. Conclusions and Comments

At the end of Lab #4, communication between the PC and MCU is achieved using bidirectional data transmission and parsing techniques. PWM and UART is used in STM32CubeIDE. In this context, PWM and UART operating principles are learned by researching. Also, it is set by determining the duty cycle setting. It is learned by examining the keypad hardware (in Figure 9). Duty cycle is set with the value entered from the keypad connected to the card, and the brightness of the LED is set according to this value. The theoretical knowledge acquired about the Keypad hardware is applied in practice to set the duty cycle, leading to a better understanding of its functionality.

4. References

- https://github.com/fcayci/stm32g0
- https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- https://www.st.com/resource/en/datasheet/stm32g031k8.pdf
- https://www.st.com/resource/en/schematic_pack/mb1455-g031k8-c01_schematic.pdf
- https://www.st.com/resource/en/user_manual/um2591-stm32g0-nucleo32-board-mb1455-stmicroelectronics.pdf
- drawio.com