GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

LAB2

Name - Surname	Yüksel YURTBAHAR
Student ID	220102002062
Name - Surname	Çağrı BÜLBÜL
Student ID	220102002032
Name - Surname	Mehmet Emre CAN
Student ID	220102002042

1. Introduction

The purpose of this laboratory is to transition from Assembly programming (Lab 1) to C programming while working directly at the register level without using the HAL library. In this experiment, various LED patterns such as the 'Knight Rider' animation were implemented using buttons and LEDs with the STM32F103C8T6 microcontroller.

2. Components and Setup

STM32F103C8T6 Microcontroller Board

ST-Link V2 Programmer

8 × 5mm Red LEDs

1 × 1kΩ Resistors

Breadboard

Jumper Wires

1 × Tactile Push Button

Problem 1: On-Board LED Toggle

The main objective of this problem is to successfully transition from Assembly programming (used in the previous lab) to C programming for controlling the microcontroller.

The specific technical goals are:

- To write C code that interacts directly with the hardware registers, without using the HAL (Hardware Abstraction Layer) library.
- To configure the necessary RCC (Reset and Clock Control) register to enable the clock for the GPIO port connected to the on-board LED.
- To configure the specific GPIO pin for the on-board LED (PC13) as a general-purpose output.
- To create a software-based delay loop.
- To use the Output Data Register (ODR) to toggle the state of the on-board LED at a rate of 1 second.

This problem was the foundational step in moving from assembly to C programming for register-level control. We learned the complete, low-level process for initializing and controlling a single GPIO pin. This involved manually enabling the correct peripheral clock in the RCC (Reset and Clock Control) registers for the target port (GPIOC). Following that, we learned to use the port's Configuration Register to set the pin's mode to “output”. Finally, I implemented the toggle logic by directly manipulating the pin's bit in the Output Data Register (ODR) within a loop, using a simple software delay to achieve the required 1-second blinking rate. This exercise provided a clear understanding of the minimum register setup required to make an LED blink.

C code:

```
#include "main.h"
void timer(uint32_t delay); //defining the delay function
# define led_delay 800000  //Represents the amount of delay for LED

int main(void)
{
 RCC ->APB2ENR |= RCC_APB2ENR_IOPCEN;/*Clock has been opened for port C,
                                it is now active. APB2ENR = Advanced peripheral 2 bus was used to access GIO,
                                Enable register worked to switch to Clock mode. */
 GPIOC -> CRH &=~(0xF << 20);//CF and MOD bits of pin 13 are cleared.(CF[00],MOD[00])
 GPIOC -> CRH |= (0x1 << 20);//The 13 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
  while (1) // infinite loop
  {
              GPIOC -> ODR ^=(1 << 13); //If pin 13 is on, turn it off, if it is off, turn it on.(Toggle MOD)
     timer(led_delay); //make a 1 second delay
  }

}
void timer(uint32_t delay)//delay function for 1 second delay
{
            for ( uint32_t i = 0 ; i < delay ; i++){
                        __NOP(); //Wait 1 clock cycle without doing anything(No operation)
            }
}
```

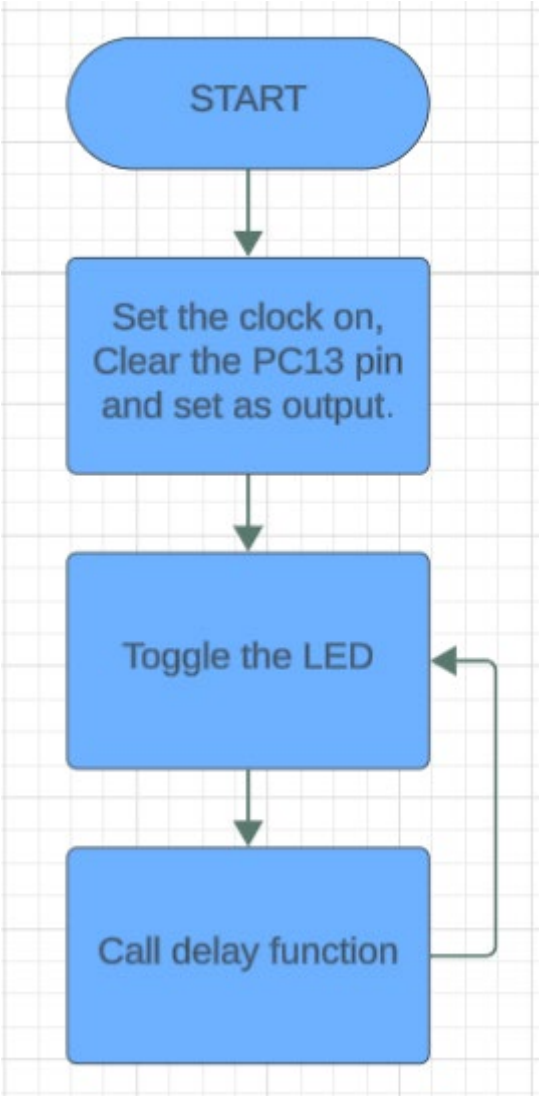

Figure 1. The Flowchart of the Problem 1

Problem 2: Button Controlled LED

The objective of this problem was to control the on-board LED using an external tactile button. The required behavior was for the LED to turn ON only when the button is pressed and turn OFF as soon as it is released.

To implement this functionality, the GPIO pin connected to the button was first configured as an input. Inside the main program loop, the Input Data Register (IDR) was continuously read to poll the button's current state. The program logic then changed the LED output (via the ODR) according to the button's press or release state, directly linking the LED's illumination to the button's action.

In this problem, we learned how to interface the microcontroller with an external input. We focused on configuring a GPIO pin as a digital input to detect a button press. We learned to continuously read the pin's state by polling the Input Data Register (IDR) within a loop. This exercise taught us the basic logic of "reading" from the outside world and using that data to make a decision, in this case, conditionally writing to the Output Data Register (ODR) to control the LED.

C Code:

```
#include "main.h"
int main(void)
{
        RCC ->APB2ENR |= RCC_APB2ENR_IOPCEN;/*Clock has been opened for port C,
                                it is now active. APB2ENR = Advanced peripheral 2 bus was used to access GIO,
                                Enable register worked to switch to Clock mode. */
        RCC ->APB2ENR |= RCC_APB2ENR_IOPAEN;/*Clock has been opened for port A,
                                it is now active. APB2ENR = Advanced peripheral 2 bus was used to access GIO,
                                Enable register worked to switch to Clock mode. */
        GPIOA -> CRL &=~(0xF << 0);//CF and MOD bits of pin A0 are cleared.(CF[00],MOD[00])
        GPIOA -> CRL |= (0x8 << 0);//The A0 pin input  is set to pull-push for button .(CF[10],MOD[00])
        GPIOA ->ODR  |= (1 << 0);
        GPIOC -> CRH &=~(0xF << 20);//CF and MOD bits of pin 13 are cleared.(CF[00],MOD[00])
        GPIOC -> CRH |= (0x1 << 20);//The 13 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
 while (1)
 {
  if(!(GPIOA ->IDR & (1 << 0))){
        GPIOC ->ODR &=~(1 << 13);
  }
  else{
        GPIOC ->ODR |=(1 << 13);
 }
}
}
```

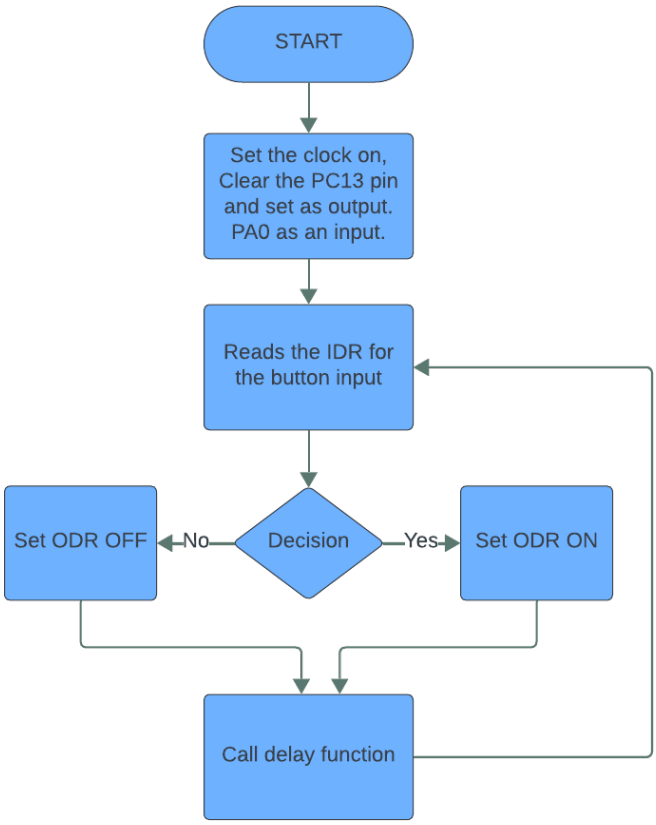

Figure 2. The Flowchart of the Problem 2

Problem 3: 8 External LEDs Toggle

All 8 external LEDs are required to toggle ON and OFF at the same time, at a rate of 1 second.

- Configured 8 separate GPIOA pins as outputs.
- Wrote to the Output Data Register (ODR) of the corresponding port(s) to set all 8 pins HIGH or LOW simultaneously.
- Called a software delay between each state change (ON to OFF, and OFF to ON) to achieve the 1-second toggle rate.

In this problem, we expanded our output control from a single LED to an array of eight. We learned how to configure multiple sequential GPIOA pins (e.g., PA0 through PA7) as outputs. The key lesson was how to manipulate the Output Data Register (ODR) to control all eight pins simultaneously. Instead of toggling each pin individually, we learned to write a single value (like 0xFF for ON and 0x00 for OFF) to the register, ensuring all LEDs changed state at the exact same time. This demonstrated efficient management of an entire port, or a part of it, as a single unit.

C Code:

```
#include "main.h"
void timer(uint32_t delay); //defining the delay function
# define led_delay 800000
int main(void)
{
	RCC ->APB2ENR |= RCC_APB2ENR_IOPAEN;/*Clock has been opened for port A,
	                                      it is now active. APB2ENR = Advanced peripheral 2 bus was used to access GIO,
	                                      Enable register worked to switch to Clock mode. */

	GPIOA -> CRL &=~(0xF << 0);//CF and MOD bits of pin A0 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 0);//The A0 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 4);//CF and MOD bits of pin A1 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 4);//The A1 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 8);//CF and MOD bits of pin A2 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 8);//The A2 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

    GPIOA -> CRL &=~(0xF << 12);//CF and MOD bits of pin A3 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 12);//The A3 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 16);//CF and MOD bits of pin A4 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 16);//The A4 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 20);//CF and MOD bits of pin A5 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 20);//The A5 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 24);//CF and MOD bits of pin A6 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 24);//The A6 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])

	GPIOA -> CRL &=~(0xF << 28);//CF and MOD bits of pin A7 are cleared.(CF[00],MOD[00])
	GPIOA -> CRL |= (0x1 << 28);//The A7 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])


 while (1)
 {
	 GPIOA -> ODR ^=(1 << 0); //If A0 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 1); //If A1 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 2); //If A2 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 3); //If A3 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 4); //If A4 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 5); //If A5 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 6); //If A6 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 GPIOA -> ODR ^=(1 << 7); //If A7 is on, turn it off, if it is off, turn it on.(Toggle MOD)
	 timer(led_delay);
 }

}

void timer(uint32_t delay)//delay function for 1 second delay
{
	for ( uint32_t i = 0 ; i < delay ; i++){
		__NOP(); //Wait 1 clock cycle without doing anything(No operation)
	}

}
```

Below is the circuit schematic used for the 8-LED setup. As shown in the diagram, eight red LEDs are connected to the STM32F103C8T6 microcontroller. The anode of each LED is connected to a separate GPIO pin, from PA0 through PA7.

For simplicity in this schematic, all eight LEDs share a common cathode. This common line is connected to Ground (GND) through a single 1kOhm current-limiting resistor (R1).

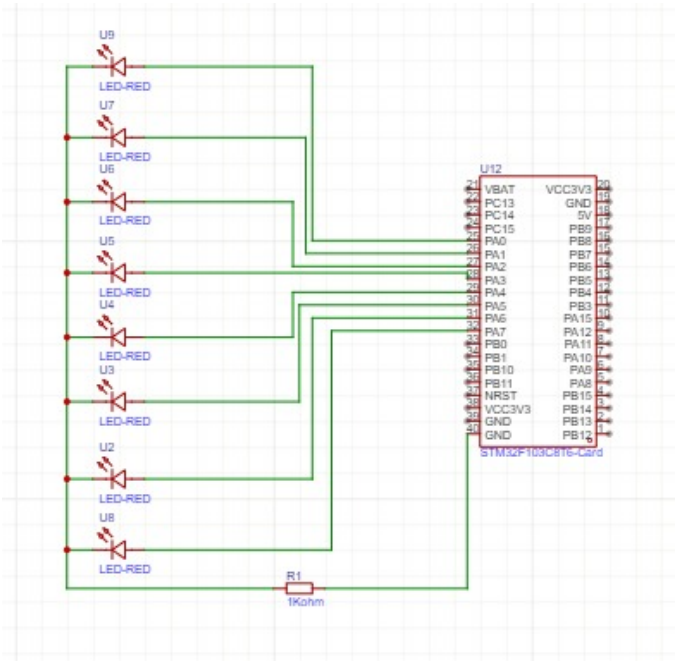

Figure 4. The Schematic of the Problem 3

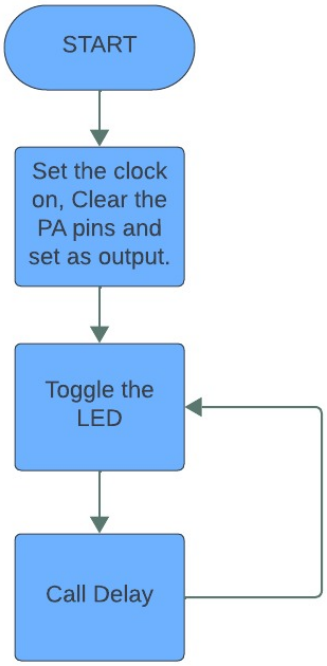

Figure 5. The Flowchart of the Problem 3

Problem 4: Direction-Controlled Shift Pattern

This problem required the implementation of a 3-LED pattern shifting across an 8-LED array, with the direction of the shift controlled by a button.

To achieve this, bit masking techniques were used to create the 3-LED pattern. Logic was implemented to handle both right and left shifting of this pattern across the 8 pins. A variable was used to track the current mode (e.g., "shifting right" or "shifting left").

The button was configured as an input. A button press was detected (either by polling or an interrupt) to toggle the shift direction. To create a visible animation, a delay of approximately 100 ms was implemented between each transition, as specified in the lab requirements. The pattern would shift indefinitely in one direction until the button was pressed, at which point it would reverse.

	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8
t0	■							
t1	■	■						
t2	■	■	■					
t3		■	■	■				
t4			■	■	■			
t5				■	■	■		
t6					■	■	■	
t7						■	■	■
t8	■						■	■
t9	■	■						■
t10	■	■	■					
t11		■	■	■				
t12			■	■	■			

Table 1. Shift Pattern of the Problem 4

This problem combined input and output logic. We learned how to create a dynamic multi-LED pattern using bit masking and bit shifting operations. The main takeaway was managing program "state" by using a variable to track the current shift direction (left or right). We then learned to use an input pin (the button) to modify this state variable, effectively changing the behavior of our output animation (toggling the direction) based on user interaction. This demonstrated a complete input-state-output loop.

C code:

```
#include "main.h"
void timer(uint32_t delay); //defining the delay function
# define led_delay 80000
# define high_delay 160000
int main(void)
{		volatile uint8_t taraf = 0;
		volatile uint8_t together = 0x07;
		 RCC ->APB2ENR |= RCC_APB2ENR_IOPAEN;/*Clock has been opened for port A, it is now active. APB2ENR =
Advanced peripheral 2 bus was used to access GPIO, Enable register worked to switch to Clock mode. */
				GPIOA -> CRL &=~(0xF << 0);//CF and MOD bits of pin A0 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 0);//The A0 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 4);//CF and MOD bits of pin A1 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 4);//The A1 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 8);//CF and MOD bits of pin A2 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 8);//The A2 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 12);//CF and MOD bits of pin A3 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 12);//The A3 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 16);//CF and MOD bits of pin A4 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 16);//The A4 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 20);//CF and MOD bits of pin A5 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 20);//The A5 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 24);//CF and MOD bits of pin A6 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 24);//The A6 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 28);//CF and MOD bits of pin A7 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 28);//The A7 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA ->CRH &=~(0xF << 0);//CF and MOD bits of pin A8 are cleared.(CF[00],MOD[00])
				GPIOA -> CRH |= (0x8 << 0);//The A8 pin İNPUT is set to pull-push for button.(CF[10],MOD[00])
				GPIOA ->ODR |=(1 << 8);
 while (1)
 {		GPIOA -> ODR &=~(0xFF);
		GPIOA -> ODR |= together;
		timer(led_delay);
   if(!(GPIOA ->IDR & (1 << 8))) {
		taraf ^= 1;
		timer(high_delay);
   }
   if (taraf == 0) // left go
		if(together == 0xE0){
			together = 0x07;
		}
		else{
			together <<= 1;
		}
   else if (taraf == 1) // right go
   {
		if (together == 0x07){
			together = 0xE0;
		}
		else{
			together >>=1;
		}
   }
     }
	}
void timer(uint32_t delay)//delay function for 1 second delay
{
	for ( uint32_t i = 0 ; i < delay ; i++){
		__NOP(); //Wait 1 clock cycle without doing anything(No operation)
	}
}
```

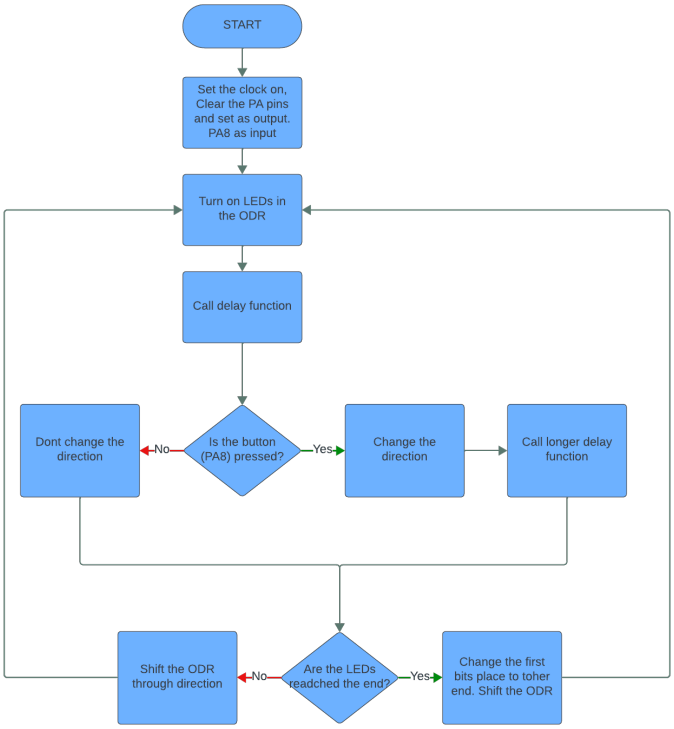

Figure 6. The Flowchart of the Problem 4

Below is the circuit schematic used for the 8-LED and button setup. As shown in the diagram, eight red LEDs have their anodes connected to separate GPIO pins, from PA0 through PA7. For simplicity, all eight LEDs share a common cathode. This common line is connected to Ground (GND) through a single 1kOhm current-limiting resistor (R1). In addition, a tactile button (KEY1) is connected to pin PA8 to serve as the user input, allowing the shift direction to be toggled.

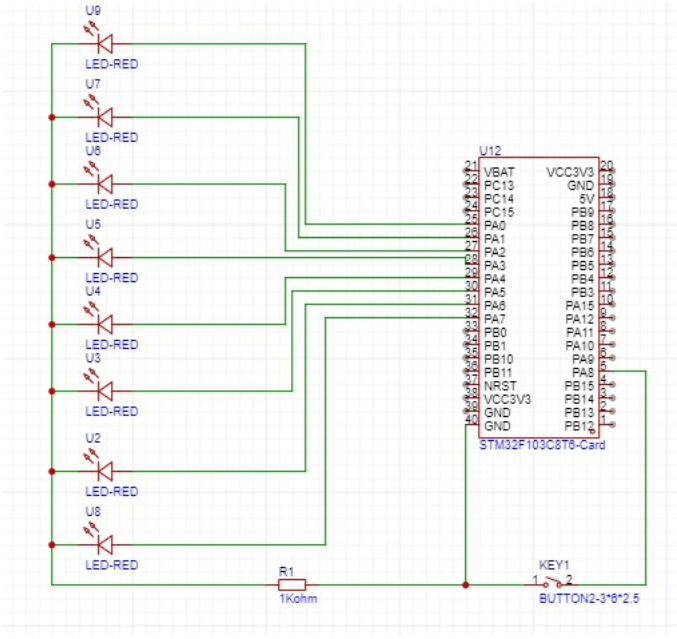

Figure 7. The Schematic of the Problem 4

Problem 5: Knight Rider

This problem involved creating the classic "Knight Rider" (Kara Şimşek) animation. This required a 3-LED pattern to continuously move back and forth across the 8-LED array. Bidirectional shifting logic was implemented, using a variable to keep track of the current direction (left or right). The core of the logic involved detecting boundary conditions. When the pattern reached the end (e.g., LED8 was lit), the direction variable was flipped to make it shift back towards LED1. When it reached the other end (LED1 was lit), it was flipped again to shift towards LED8. This entire animation was placed inside a while(1) loop to ensure it ran continuously. As required, a delay of approximately 100 ms was included after each single shift to create a smooth and visible scanning effect.

Table 2. Pattern on LEDs for Problem 5

	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8
t0	■							
t1	■	■						
t2	■	■	■					
t3		■	■	■				
t4			■	■	■			
t5				■	■	■		
t6					■	■	■	
t7						■	■	■
t8					■	■	■	
t9				■	■	■		
t10			■	■	■			
t11		■	■	■				
t12	■	■	■					
t13		■	■	■				
t14			■	■	■			

Below is the circuit schematic used for the 8-LED setup. As shown in the diagram, eight red LEDs are connected to the STM32F103C8T6 microcontroller. The anode of each LED is connected to a separate GPIO pin, from PA0 through PA7. For simplicity in this schematic, all eight LEDs share a common cathode. This common line is connected to Ground (GND) through a single 1kOhm current-limiting resistor (R1). The same circuit as Problem 3.

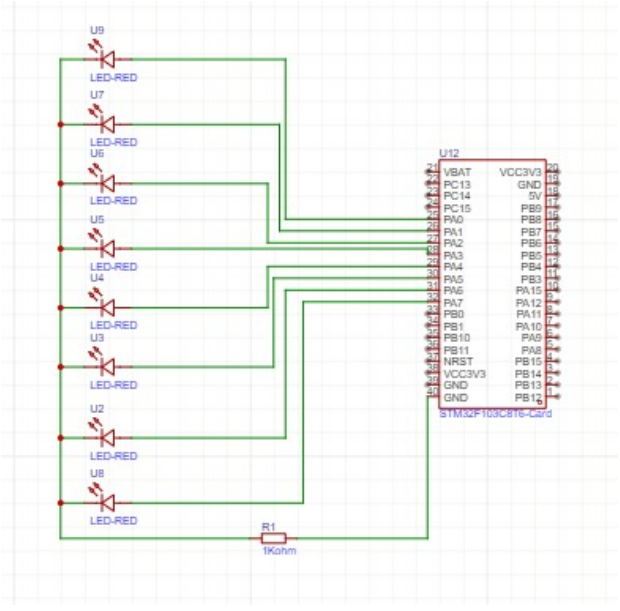

Figure 8. The Schematic of the Problem 5

C code:

```
#include "main.h"
void timer(uint32_t delay); //defining the delay function
# define led_delay 80000
int main(void)
{
		volatile uint8_t taraf = 0;
			volatile uint8_t together = 0x07;
			 RCC ->APB2ENR |= RCC_APB2ENR_IOPAEN;/*Clock has been opened for port A,
								it is now active. APB2ENR = Advanced peripheral 2 bus was used to access GIO,
								Enable register worked to switch to Clock mode. */
				GPIOA -> CRL &=~(0xF << 0);//CF and MOD bits of pin A0 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 0);//The A0 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 4);//CF and MOD bits of pin A1 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 4);//The A1 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 8);//CF and MOD bits of pin A2 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 8);//The A2 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 12);//CF and MOD bits of pin A3 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 12);//The A3 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 16);//CF and MOD bits of pin A4 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 16);//The A4 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 20);//CF and MOD bits of pin A5 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 20);//The A5 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 24);//CF and MOD bits of pin A6 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 24);//The A6 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
				GPIOA -> CRL &=~(0xF << 28);//CF and MOD bits of pin A7 are cleared.(CF[00],MOD[00])
				GPIOA -> CRL |= (0x1 << 28);//The A7 pin 10 mhz output is set to pull-push.(CF[00],MOD[01])
 while (1)
 {		GPIOA -> ODR &=~(0xFF);
			 GPIOA -> ODR |= together;
			 timer(led_delay);
			 if (taraf == 0){
					if(together == 0xE0){
						taraf = 1 ;
					}
					else{
						together <<= 1 ;
					}
			 }
			 else if (taraf == 1){
					if(together == 0x07){
						taraf = 0;
					}
					else {
						together >>= 1 ;
					}
			 }
 }

}

void timer(uint32_t delay)//delay function for 1 second delay
{
		for ( uint32_t i = 0 ; i < delay ; i++){
			__NOP(); //Wait 1 clock cycle without doing anything(No operation)
```

From implementing the "Knight Rider" animation, we learned how to create a more complex, state-based program. The key challenge was managing the bidirectional movement. This required using a state variable (e.g., direction) to control whether the 3-LED pattern was shifting left or right. We learned how to implement boundary detection by checking if the pattern had reached the limits (LED1 or LED8). When a boundary was detected, the logic would flip the state variable,

reversing the shift direction. This problem solidified us understanding of combining bitwise operations (to create the pattern) with conditional logic (to manage the back-and-forth movement) inside a continuous while(1) loop.

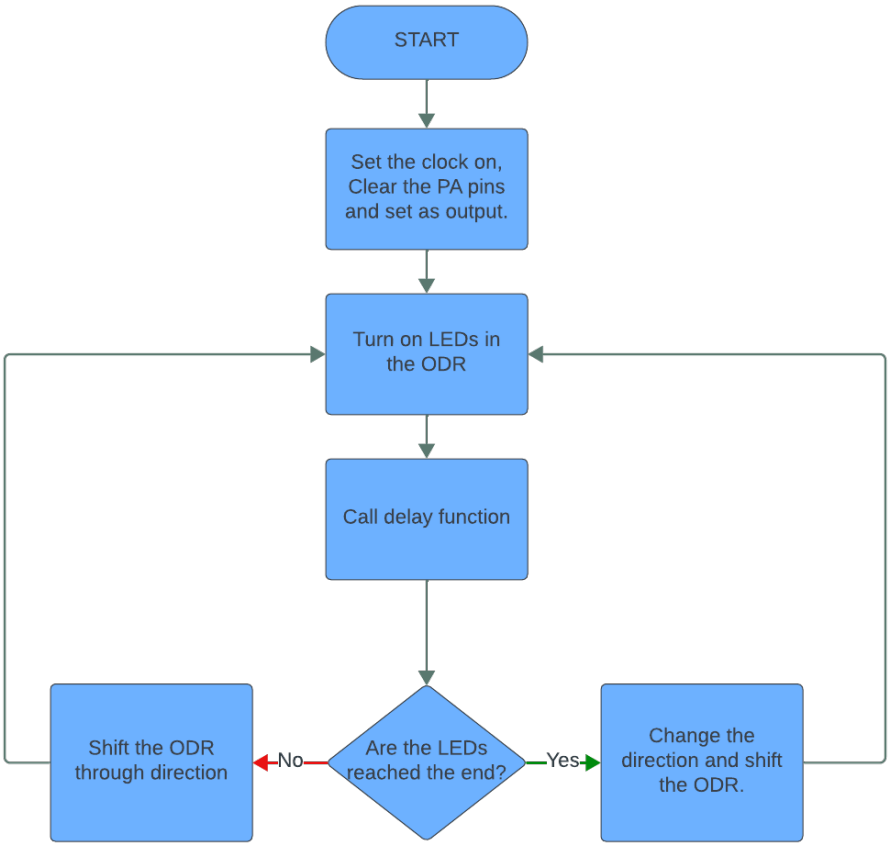

Figure 9. The Flowchart of the Problem 5

References

ERC Handbook. (2025). Retrieved from https://erc-bpgc.github.io/handbook/electronics/Development_Boards/STM32/

st. (2025). Retrieved from https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html

st. (2025). Retrieved from https://www.st.com/en/development-tools/st-link-v2.html

STMicroelectronics. (2025). *Reference manual for STM32F10x microcontrollers (RM0008)*. Retrieved from https://www.st.com/resource/en/reference_manual/rm0008-stm32f10xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

STMicroelectronics. (2025). *STM32F103C8T6 datasheet.* Retrieved from https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

ARM. (2025). *Cortex-M3 technical reference manual.* Retrieved from https://developer.arm.com/documentation/ddi0337