GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRONICS
ENGINEERING

# ELEC 335

## Microprocessors Laboratory

## LAB1

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

## Problem 1 – STM32F103C8T6 Board Components and Peripherals

In this section, the STM32F103C8T6 (Blue Pill) development board is examined in detail. All the integrated circuits (ICs), onboard peripherals, and their electrical connections with the microcontroller are identified and explained.

### 1.1. Main Microcontroller (STM32F103C8T6)

This section focuses on introducing the STM32F103C8T6 microcontroller and outlining its essential properties. Built around the ARM Cortex-M3 32-bit core with a RISC architecture, it operates at clock speeds of up to 72 MHz, providing a balance between performance and energy efficiency. The chip is equipped with 64 KB of Flash memory for program storage and 20 KB of SRAM for data handling, all powered by a 3.3 V supply.

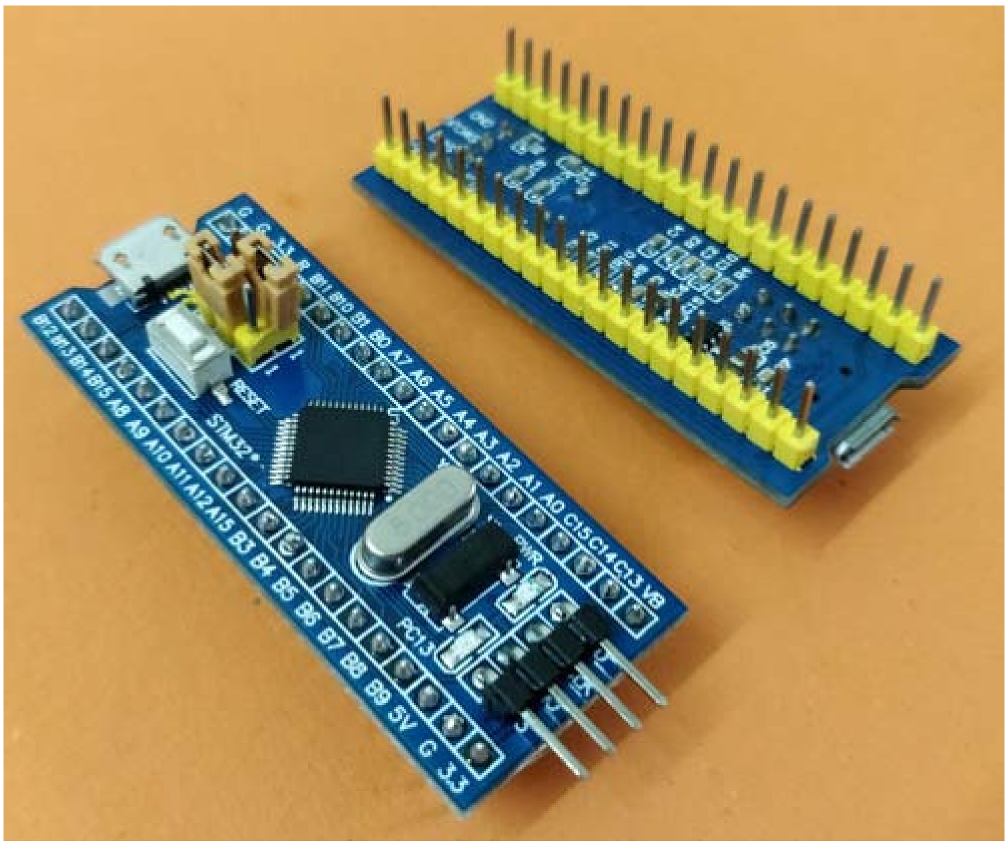

**Figure 1.** STM32F103C8T6 Board

In terms of functionality, the STM32F103C8T6 offers a versatile set of peripherals. It supports multiple communication interfaces—including three USART ports for serial transmission, two I²C buses for device interconnection, three SPI channels for rapid data transfer, and a CAN interface designed for robust industrial communication. Furthermore, it incorporates a 12-bit Analog-to-Digital Converter with 10 input channels, timers for PWM generation and signal capture, and GPIO ports A, B, and C that enable direct interaction with sensors, switches, or LEDs. The device also provides USB 2.0 full-speed connectivity, making it suitable for embedded applications requiring efficient communication and control.

### 1.2. Main ICs on the Blue Pill Board

In this part, the main integrated circuits (ICs) on the Blue Pill board are introduced. Each IC's function and its role in the overall operation of the STM32F103C8T6 system are explained in detail.

| IC Name / Label | Function | Notes |
|---|---|---|
| **STM32F103C8T6** | Main microcontroller | Handles all processing and I/O control |
| **AMS1117-3.3** | Voltage regulator | Converts 5 V (from USB or pin) to 3.3 V for MCU |
| **CH340G / CP2102** | USB-to-Serial converter | Enables USB communication between MCU (USART1) and PC |
| **8 MHz Crystal (Y1)** | Main external oscillator | Provides stable clock input to the MCU |
| **32.768 kHz Crystal (Y2)** | Real-Time Clock oscillator | Used by RTC peripheral |
| **Reset Button (RST)** | Manual reset | Pulls NRST pin low |
| **BOOT0 Jumper** | Boot mode selection | 0 = Flash memory (user program)<br>1 = System memory (UART bootloader) |
| **BOOT1 Jumper** | Secondary boot selector | 0 = Normal operation<br>1 = SRAM boot (debug mode) |
| **Power LED (PWR)** | Indicates board powered | Connected to 3.3 V rail |
| **User LED (PC13)** | Test LED | Active low; PC13 drives it |
| **Micro-USB connector** | Power + Serial interface | 5 V input and USB-UART bridge |
| **Pin Headers (2×20)** | GPIO breakout | Exposes all I/O pins for prototyping |

**Table1.** Main ICs on the Blue Pill

### 1.3. Peripheral Connections to STM32F103C8T6

In this section, the peripheral components connected to the STM32F103C8T6 microcontroller are presented. The table and the figure below show how each peripheral is interfaced with the microcontroller through its specific pins and functions.

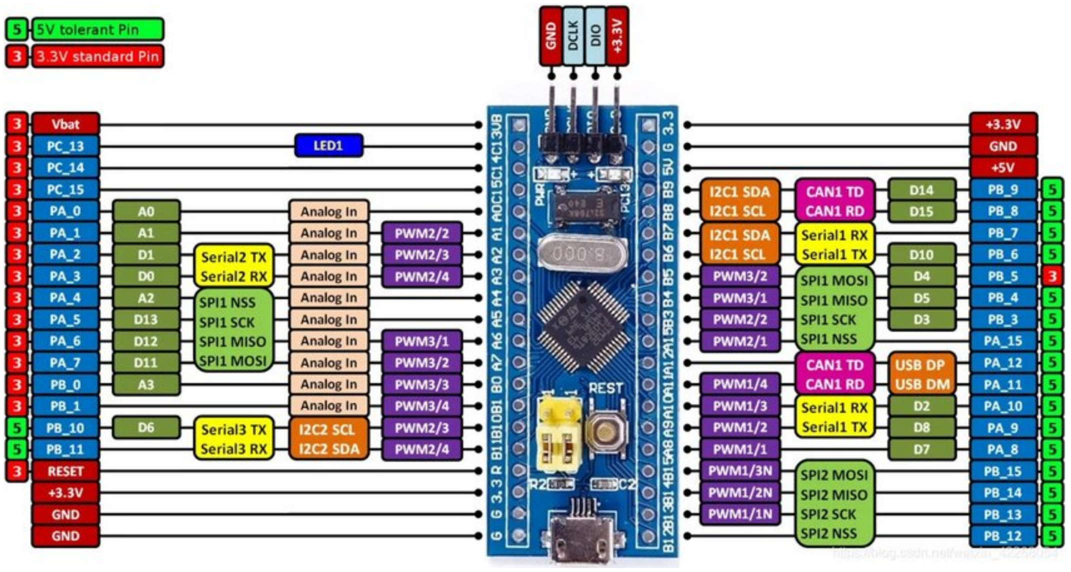

**Figure 2.** STM32F103C8T6 Input / Output Pins (microcontrollerslab, 2025)

| Peripheral | STM32 Pin(s) | Function |
|---|---|---|
| **User LED** | PC13 | Output (active low) |
| **Reset Button** | NRST | Manual reset |
| **BOOT0 Jumper** | BOOT0 | Boot configuration |
| **USB-UART (CH340G)** | PA9 (TX1), PA10 (RX1) | Serial communication with PC |
| **Voltage Regulator (AMS1117)** | Input: 5 V, Output: 3.3 V | Power supply for MCU |
| **8 MHz Crystal** | OSC_IN (PA0) / OSC_OUT (PA1) | System clock |
| **32.768 kHz Crystal** | PC14, PC15 | RTC clock |
| **Power Input** | 5 V (USB or pin), GND | Board power source |
| **3.3 V Output** | VCC pin | For powering sensors/modules |

**Table 2.** Peripheral Connections to Blue Pill

**Problem 2 - Implementing LED Control via Assembly on STM32F103C8T6**

In this experiment, the goal is to control an LED connected to pin PA1 of the STM32F103C8T6 (Blue Pill) microcontroller using ARM Assembly (Thumb-2) instructions.

The purpose is to understand the use of memory-mapped I/O (memory-mapped I/O) directly at the register level—without relying on high-level libraries or HAL functions.

**Memory-Mapped I/O and Registers**

To light up the LED, three fundamental register groups must be configured:

1. **RCC (Reset and Clock Control):** On the microcontroller, the clock signals for peripherals (including GPIO ports) are disabled by default to save power. Before using a port, its clock signal must be enabled via the RCC_APB2ENR (APB2 Peripheral Clock Enable Register).

2. **CRL (Configuration Register Low):** After the port's clock is enabled, each pin must be configured (e.g., input, output, analog). The GPIOA_CRL register sets the mode (e.g., "Push-Pull Output") and speed (e.g., "50Mhz") for pins PA0 through PA7.
3. **ODR (Output Data Register):** Once the pin is configured as an output, the ODR register is used to send a logic 1 (HIGH) or logic 0 (LOW) value to that pin.

According to the Reference Manual (RM0008) and the definitions in the code, the relevant addresses are:

- **RCC Base:** 0x40021000 (clk_zero)
- **RCC_APB2ENR:** 0x40021018 (clk_evo)
- **GPIOA Base:** 0x40010800 (base_gpio)
- **GPIOA_CRL:** 0x40010800 (crl_gpio)
- **GPIOA_ODR:** 0x4001080C (out_gpio)

Assembly Code:

```
.syntax unified @specify that you will use the modern arm language
.thumb @specify that you will be using the thumb 2 set
.equ clk_zero , 0x40021000 @I assigned the clock reset control unit to clk_zero.
.equ clk_evo ,clk_zero + 0x18 @It assigns the peripheral clocks to the exact address of the
record that activates them.
.equ base_gpio , 0x40010800 @The general purpose PA is the basic entry and exit address
.equ crl_gpio , base_gpio + 0x00 @Port allocation assigns lower register addresses. From
PA0 to PA7
.equ out_gpio ,base_gpio + 0x0C @Keeps the output data record
.equ open_clk , (1<<2) @Holds the bit that opens the GPIO clock
.equ pa_1_pin , (1<<1) @The bit value representing PA1 thousand was assigned
.global main @Provides access to the connector from outside the main function
main :@Start execution
LDR R0,=clk_evo @Load to clok_evo R0 register
LDR R1,[R0]     @R0 read the current value R1 load (clock setting)
ORR R1,R1, #open_clk @Add the open_clk value to the value in R1 with the OR
operation.
STR R1,[R0] @Write back the new value R1 to R0 (GPIO is now operational)
LDR R0 , =crl_gpio @Load address R0 in circle_gpio
LDR R1 , [R0] @Read R0 current value and load R1
LDR R2 ,= 0xFFFFFF0F @It resets the bit settings of PA1 by masking.
AND R1,R1,R2 @Clear the setting bits of PA1 using the AND command. (The settings of
other pins are preserved.)
LDR R2, = 0x30 @It represents the setting MODE1=0b11 (50Mhz) and CNF1=0b00
(Push-Pull). 50 mhz push-out output.
STR R1 , [R0] @PA1 is out now
LDR R0, =out_gpio @load out_gpio to R0
LDR R1, [R0] @Read the current value R0 and load it into R1.
ORR R1 ,R1 ,#pa_1_pin @Or logical operations set the PA1 bit value to 1.
STR R1 ,[R0] @The PA1 pin becomes High and the LED lights up
loop: @Prevent the program from running randomly and terminating
  B loop
```

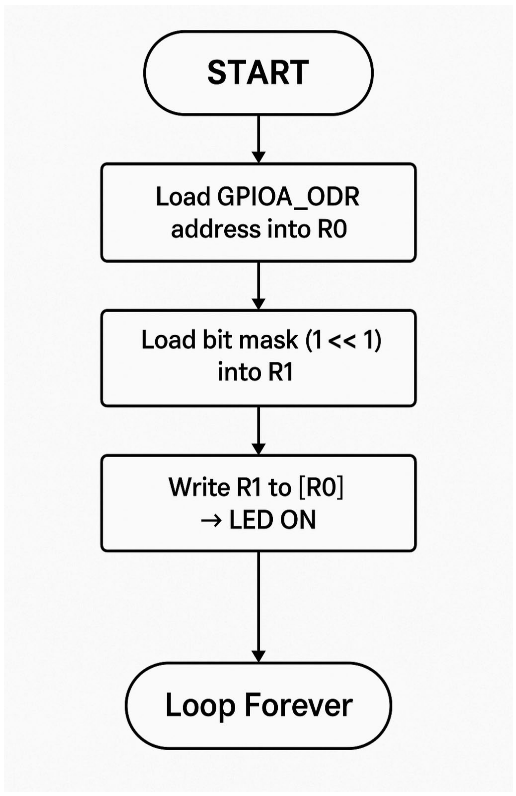
START
Load GPIOA_ODR address into R0
Load bit mask (1 << 1) into R1
Write R1 to [R0] → LED ON
Loop Forever

**Figure 3.** Flowchart Of the Problem 2

The flowchart above illustrates the logical sequence of the assembly program used to turn on an LED connected to pin PA1. It begins by loading the GPIOA_ODR register address into register R0, then loads the LED bit mask (1 << 1) into R1, and writes this value to the ODR register to set the corresponding bit high, thereby turning the LED on. Finally, the program enters an infinite loop to keep the LED continuously lit.

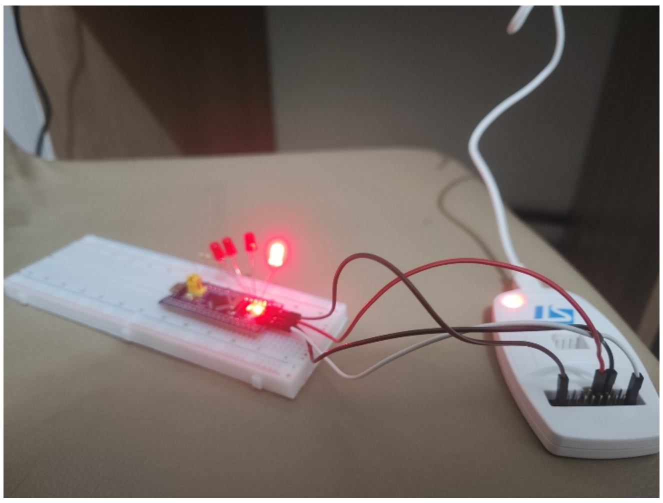

**Figure 4.** Protype Of the Problem 2

Through this experiment, we learned how to control a microcontroller pin directly by manipulating its hardware registers using assembly instructions. We configured the RCC, CRL, and ODR registers manually, gaining insight into how each step—enabling the peripheral clock, setting pin mode and speed, and writing to the output data register—affects the physical behavior of the pin. Overall, this experiment gave us foundational experience in bare-metal programming, helped us understand the relationship between the ARM Assembly code and the STM32 hardware architecture, and prepared us for more complex multi-pin and timing-based applications in later experiments.

**Problem 3 – Implementing Multiple LED Control via Assembly on STM32F103C8T6**

The aim of this experiment is to extend the assembly code to control four LEDs connected to pins PA1, PA2, PA3, and PA4 on the STM32F103C8T6 (Blue Pill) board. In this task, each LED is connected in series with a current-limiting resistor.

This task requires a full "bare-metal" configuration sequence: first enabling the peripheral's clock, then configuring the specific pins as outputs, and finally writing to the output register to turn the LEDs on.

**Pin and Register Configuration**

To control four pins, we must first enable the clock for the GPIOA port, then configure all four pins (PA1-PA4) as outputs, and finally write to the GPIOA_ODR (Output Data Register).

Each LED corresponds to one bit in the ODR register:

| LED | PIN | ODR Bit |
|---|---|---|
| LED1 | PA1 | Bit 1 |
| LED2 | PA2 | Bit 2 |
| LED3 | PA3 | Bit 3 |
| LED4 | PA4 | Bit 4 |

**Table 3.** LED to ODR Connection

To turn on all four LEDs simultaneously, bits 1–4 of the ODR register must be written as logic high (1). Thus, the bitmask representing these four LEDs is:

**0x1E** (which is 0b00011110 in binary, setting bits 1, 2, 3, and 4)

The relevant addresses used in the code are:

- **GPIOA Clock Enable (RCC_APB2ENR):** 0x40021018 (clk_evo)
- **GPIOA Config (GPIOA_CRL):** 0x40010800 (crl_gpio)
- **GPIOA Output (GPIOA_ODR):** 0x4001080C (out_gpio)

Assembly Code:

```
.syntax unified @specify that you will use the modern arm language
.thumb @specify that you will be using the thumb 2 set
.equ clk_zero , 0x40021000 @I assigned the clock reset control unit to clk_zero.
.equ clk_evo ,clk_zero + 0x18 @It assigns the peripheral clocks to the exact address of the
record that activates them.
.equ base_gpio , 0x40010800 @The general purpose PA is the basic entry and exit address
.equ crl_gpio ,  base_gpio + 0x00 @Port allocation assigns lower register addresses. From
PA0 to PA7
.equ out_gpio , base_gpio + 0x0C @Keeps the output data record
.equ open_clk , (1<<2) @Holds the bit that opens the GPIO clock
.equ pa_all , 0x1E @It represents the PA1, PA2, PA3 and PA4 bins collectively.
.equ pa_clean ,0xFFF0000F @Resets the pin's setting bits from PA1 to PA4.
.equ pa_light ,0x00033330 @The value that sets all pins from PA1 to PA4 to push-pull 50
mHz.
.global main @Provides access to the connector from outside the main function
main :@Start execution
LDR R0,=clk_evo @Load to clok_evo R0 register
LDR R1,[R0]     @R0 read the current value R1 load (clock setting)
ORR R1,R1, #open_clk @Add the open_clk value to the value in R1 with the OR operation.
STR R1,[R0] @Write back the new value R1 to R0 (GPIO is now operational)
LDR R0 , =crl_gpio @Load address R0 in circle_gpio
LDR R1 , [R0] @Read R0 current value and load R1
LDR R2 ,= pa_clean @It resets the bit settings of PA1,PA2,PA3 and PA4 by masking.
AND R1,R1,R2 @Clear the setting bits of PA1 using the AND command. (The settings of
other pins are preserved.)
LDR R2, = pa_light @It represents the setting MODE1=0b11 (50Mhz) and CNF1=0b00
(Push-Pull).
ORR R1 ,R1 ,R2 @The new R1 in R2 is added to the R2 with the OR operation, this sets the
PA1,PA2,PA3 and PA4 50 mhz push-out output.
STR R1 , [R0] @PA1,PA2,PA3 and PA4 are out now
LDR R0, =out_gpio @load out_gpio to R0
LDR R1, [R0] @Read the current value R0 and load it into R1.
ORR R1 ,R1 ,#pa_all @Or logical operations set the PA1,PA2,PA3 and PA4  bit value to 1.
STR R1 ,[R0] @The PA1,PA2,PA3 and PA4  pin become High and the LED lights up
loop: @Prevent the program from running randomly and terminating
  B loop
```

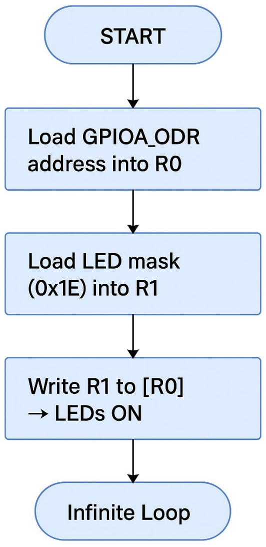
START
Load GPIOA_ODR address into R0
Load LED mask (0x1E) into R1
Write R1 to [R0] → LEDs ON
Infinite Loop

**Figure 5.** Flowchart Of the Problem 3

The flowchart above illustrates the logical sequence of the assembly program used to turn on four LEDs connected to pins **PA1**, **PA2**, **PA3**, and **PA4**. It begins by loading the **GPIOA_ODR** register address into register **R0**, then loads the combined LED mask value **(0x1E)** into **R1** and writes this value to the ODR register to set all corresponding bits high, thereby turning all LEDs on simultaneously. Finally, the program enters an infinite loop to keep the LEDs continuously lit.

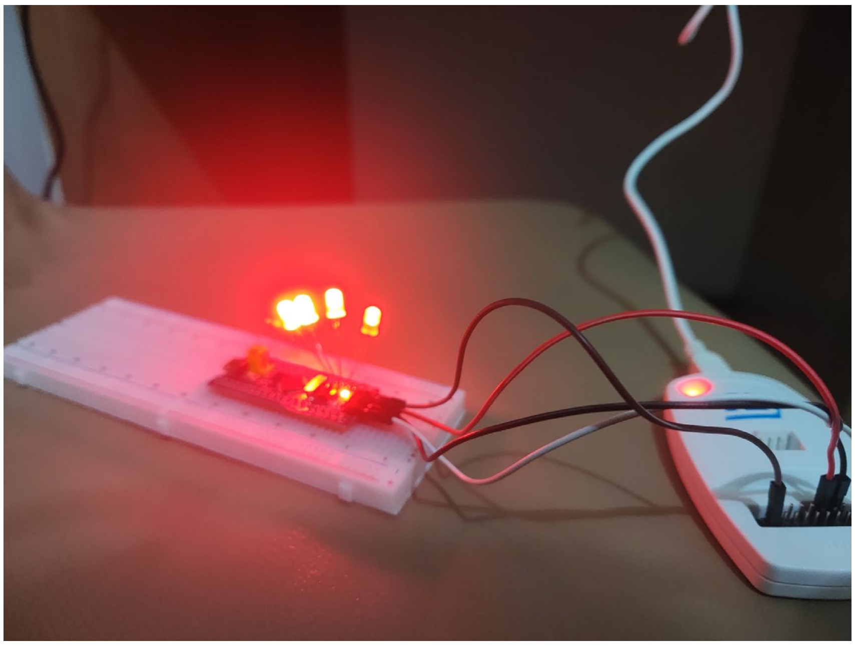

**Figure 6.** Protype Of the Problem 3

Through this experiment, we learned how to **extend single-pin GPIO control** to multiple pins by efficiently using **bitmask operations** in assembly. We realized that multiple output pins can be activated at once by setting the corresponding bits in the **Output Data**

**Register (ODR)**, rather than controlling each pin individually. By completing the bare-metal initialization we deepened our knowledge of **ARM Assembly (Thumb-2)** and direct hardware control. Overall, this experiment strengthened our teamwork and practical embedded-systems skills by showing how to manipulate multiple GPIO lines simultaneously through precise register-level programming.

### Problem 4 - 1-Second LED Toggling

This experiment extends Problem 2 by adding a software delay routine that makes the LED connected to **PA1** toggle at approximately **1-second intervals**. Through this task, we learned how to design and implement a precise **software delay** using **nested loops** that consume a deterministic number of **CPU cycles** based on the system clock frequency. As a team, we gained a better understanding of how to control microcontroller peripherals directly through **memory mapped I/O** on the **STM32F103C8T6 (Blue Pill)**.

We examined the relevant registers from the STM32 reference manual (RM0008), including:

- **RCC_APB2ENR** at 0x40021018, which enables the GPIOA peripheral clock (bit 2).
- **GPIOA_CRL** at 0x40010800, which configures PA0–PA7, including **MODE1** and **CNF1** for PA1.
- **GPIOA_ODR** at 0x4001080C, which controls output data, with **bit 1** driving PA1.

We also configured **PA1** as a **50 MHz general-purpose push-pull output** (MODE1 = 0b11, CNF1 = 0b00) and analysed how these configurations bits define the electrical behavior of the pin. By calculating loop iterations (~16 million cycles at 16 MHz) and relating them to timing, we developed a deeper understanding of how **instruction-level execution time** translates into **real-time behavior**.

Overall, this experiment helped us improve our teamwork in low-level programming, strengthen our grasp of **ARM Assembly (Thumb-2)**, and gain practical insight into how hardware registers and timing interact within embedded systems.

Assembly Code:

```
.syntax unified @specify that you will use the modern arm language
.thumb @specify that you will be using the thumb 2 set
.equ clk_zero , 0x40021000 @I assigned the clock reset control unit to clk_zero.
.equ clk_evo ,clk_zero + 0x18 @It assigns the peripheral clocks to the exact address of the record
that activates them.
.equ base_gpio , 0x40010800 @The general purpose PA is the basic entry and exit address
.equ crl_gpio , base_gpio + 0x00 @Port allocation assigns lower register addresses. From PA0 to
7
.equ out_gpio ,base_gpio + 0x0C @Keeps the output data record
.equ open_clk , (1<<2) @Holds the bit that opens the GPIO clock
.equ pa_1_pin , (1<<1) @The bit value representing PA1 thousand was assigned
.equ out_loop ,70 @The number of external loops was assigned a value of 70
.equ in_loop ,26600 @The number of inner loops was assigned a value of 26600.
.global main @Provides access to the connector from outside the main function
main :@Start execution
 LDR R0,=clk_evo @Load to clok_evo R0 register
 LDR R1,[R0]     @R0 read the current value R1 load (clock setting)
 ORR R1,R1, #open_clk @Add the open_clk value to the value in R1 with the OR operation.
 STR R1,[R0] @Write back the new value R1 to R0 (GPIO is now operational)
 LDR R0 , =crl_gpio @Load address R0 in circle_gpio
 LDR R1 , [R0] @Read R0 current value and load R1
 LDR R2 ,= 0xFFFFFF0F @It resets the bit settings of PA1 by masking.
 AND R1,R1,R2 @Clear the setting bits of PA1 using the AND command. (The settings of other
pins are preserved.)
 LDR R2, = 0x30 @It represents the setting MODE1=0b11 (50Mhz) and CNF1=0b00 (Push-Pull).
 ORR R1 ,R1 ,R2 @The new R1 in R2 is added to the R2 with the OR operation, this sets the PA1
50 mhz push-out output.
 STR R1 , [R0] @PA1 is out now
 LDR R3, =out_gpio @load out_gpio to R3
loop:
@ Light ON
 LDR R1, [R3] @Read the current value R3 and load it into R1.
 ORR R1 ,R1 ,#pa_1_pin @Or logical operations set the PA1 bit value to 1.
 STR R1 ,[R3] @The PA1 pin becomes High and the LED lights up
@ Light OFF
 LDR R1, [R3] @Read the current value R3 and load it into R1.
 BIC R1 ,R1 ,#pa_1_pin @Sets the bit of R1 to zero
 STR R1 ,[R3] @The PA1 pin becomes Low and the LED lights down
 BL delay_1s @1 second delay
 B loop @Prevent the program from running randomly and terminating
.type delay_1s ,%function @we have defined that delays_1 is a function
delay_1s:
 LDR R1 ,=out_loop @Load the value out_loop into register R1
 .outer_loop:
 LDR R2 ,=in_loop @Load the value in_loop into register R2
.inner_loop:
  SUBS R2,R2, #1 @Reduce the value in R2 by 1
  BNE .inner_loop @If R2 is not zero after the subtraction, return to in_loop (repeat the inner
loop).
  SUBS R1,R1, #1 @Reduce the value in R2 by 1
  BNE .outer_loop @If R1 is'nt zero after the subtraction return to out_loop(repeat the outer loop).
  BX lr @Returns to the address held in the Lr register. Delay completed
.size delay_1s, .-delay_1s
```

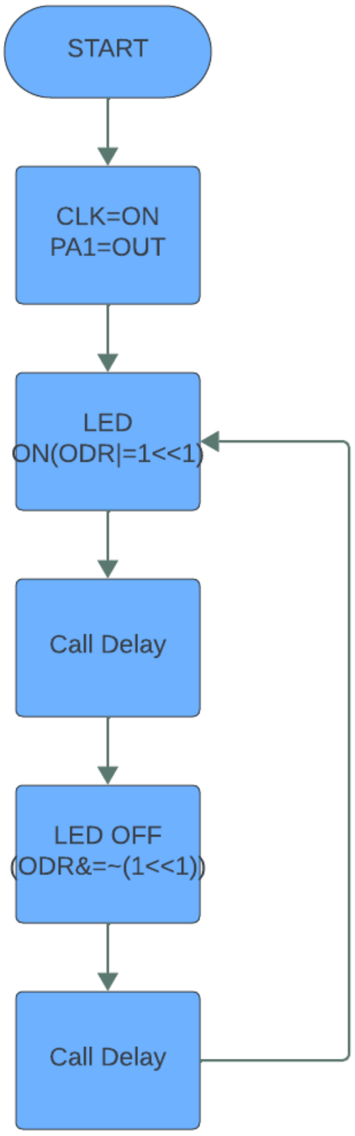

**Figure 7.** Flowchart Of the Problem 4

The flowchart above illustrates the operation of the assembly program that toggles the LED on pin **PA1** at 1-second intervals. The program begins by enabling the **GPIOA clock** and configuring **PA1** as an output pin. Inside the main loop, the LED is turned **ON** by setting bit 1 of the **ODR register**, then the program calls the **delay routine** (delay_1s) to wait approximately one second. After the delay, the LED is turned **OFF** by clearing the same bit in the ODR, followed by another 1-second delay. The loop repeats indefinitely, producing a continuous ON–OFF blinking pattern.

## References

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

*microcontrollerslab*. (2025). Retrieved from https://microcontrollerslab.com/stm32f103c8t6-blue-pill-pinout-peripherals-programming-features/

*robotistan*. (2025). Retrieved from https://maker.robotistan.com/stm32f103c8t6-nedir-ozellikleri-nelerdir-stm32-st-link-utility-nasil-kullanilir/

*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