GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

Lab #1 Experiment Report

Hazırlayanlar
200102002051 – Arda DERİCİ
200102002061 – Serdar BAŞYEMENİCİ
200102002025 – Umut Mehmet ERDEM

1. Introduction

In this experiment, writing assembly code for stm32, working logic of the pins used, memory access and connection diagram were learned. Jumper cables, STM32 microcontroller, breadboard, resistors and leds were used. The code is debugged into STM32CubeIDE. Writing assembly code in stm32 and circuit diagram setup were done using the stm32 referance manual, schematic diagram and datasheet.

2. Problems

2.1. Problem I

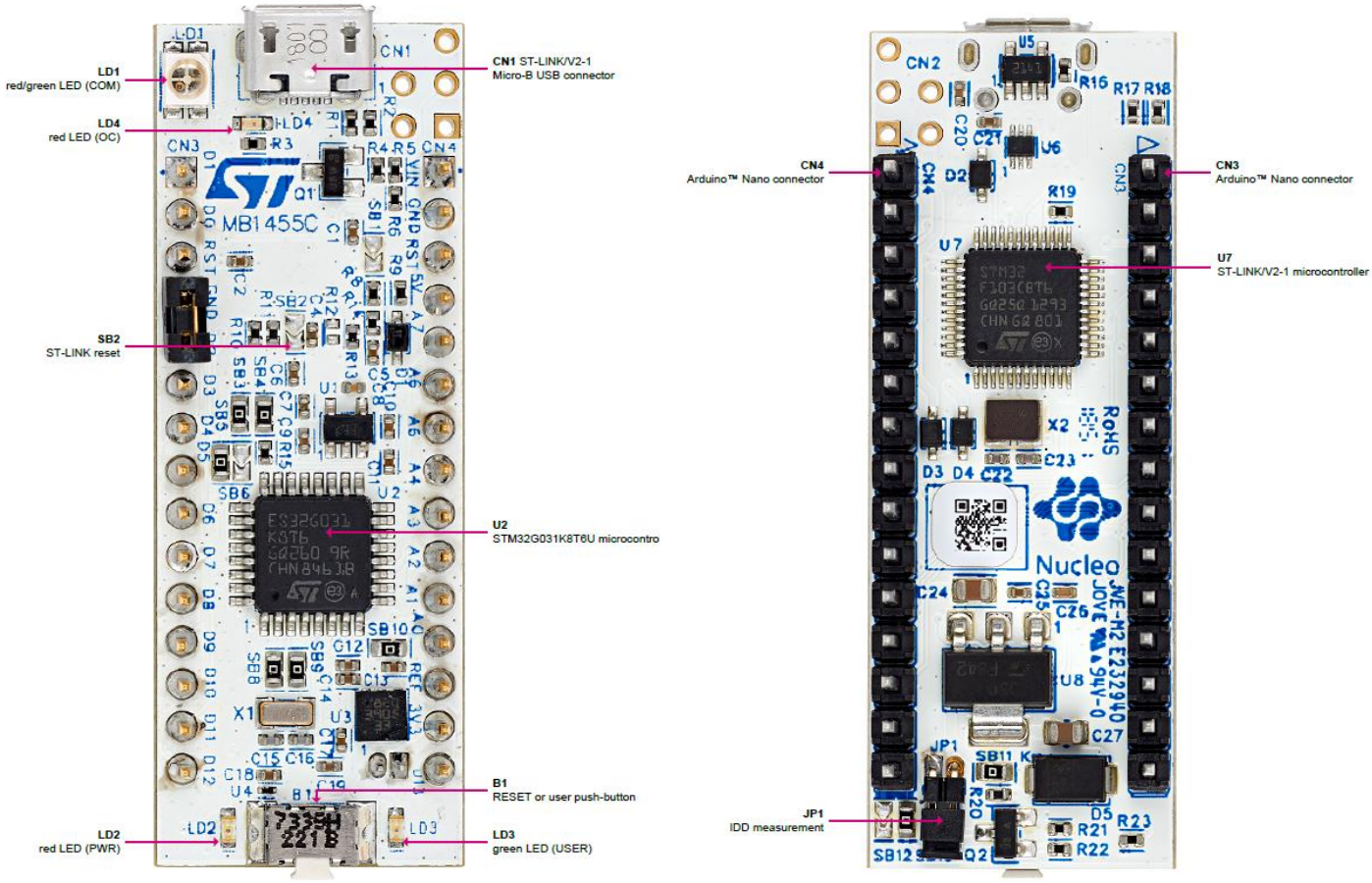

Figure 1. NUCLEO-STM32G031K8

The NUCLEO-STM32G031K8 board includes:

STM32G031K8 microcontroller with ARM Cortex-M0+ core for low power consumption, fast response times and a wide range of application areas,

ST-Link/V2-1 Debugger with STM32103C8T6 microcontroller, which allows you to perform programming and debugging operations by connecting the card to your computer,

ST-Link/V2-1 micro-B USB connector, which allows the card to be connected to your computer for programming and debugging.

LD1 LED indicating that the cable is connected, LD2 LED indicating that the power is on, and LD3 LED that are user-defined,

B1 button, which allows us to use and reset the card.

The main system consists of:
• Two masters:
– Cortex® -M0+ core
– General-purpose DMA
• Three slaves:
– Internal SRAM
– Internal Flash memory
– AHB with AHB-to-APB bridge that connects all the APB peripherals

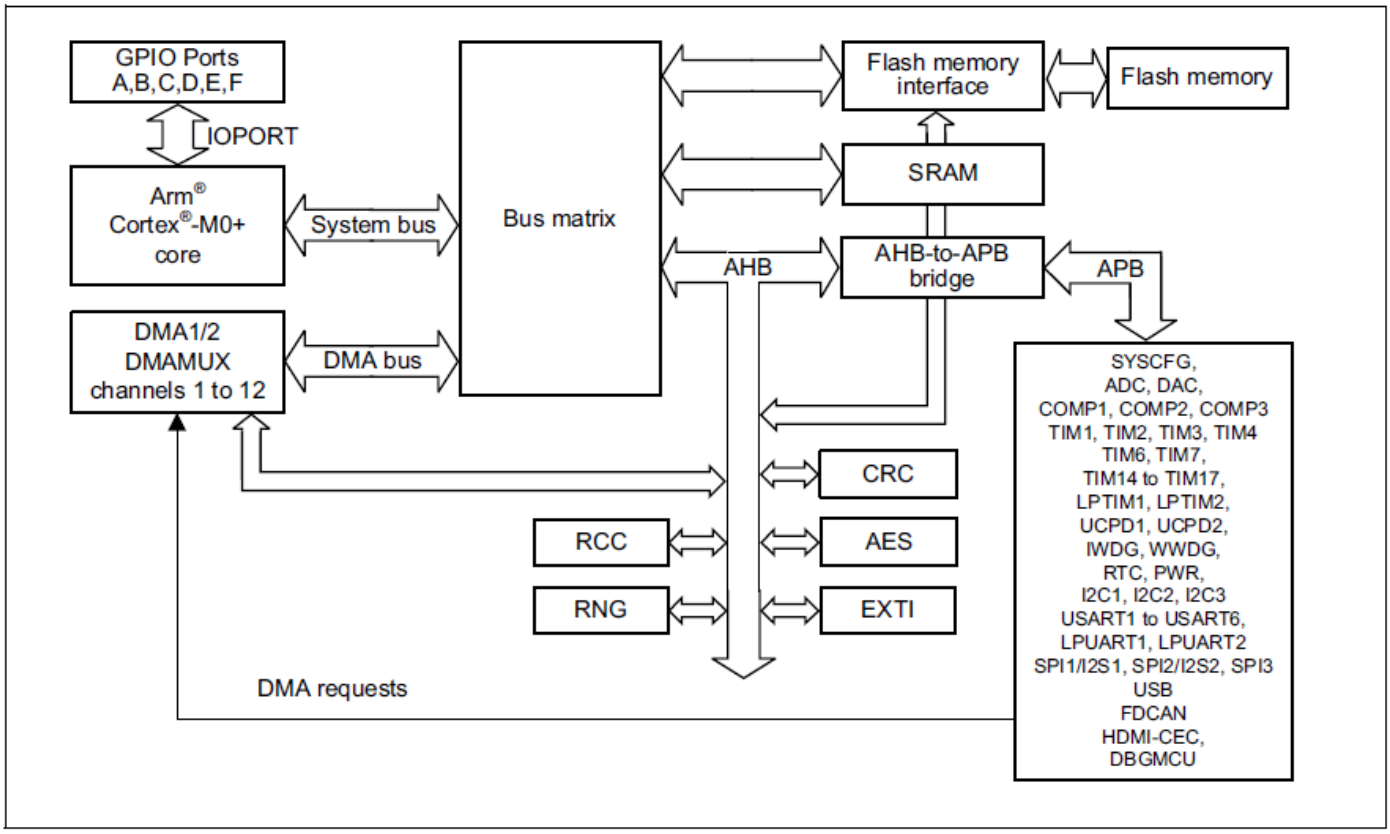

Figure 2. System architecture

System bus (S-bus): This bus connects the system bus of the Cortex®-M0+ core (peripheral bus) to a bus matrix that manages the arbitration between the core and the DMA.

AHB (Advanced High-Performance Bus) and APB (Advanced Peripheral Bus) BUS: They represent data buses with separate speed and priority levels that enable data transmission to high-performance and low-speed peripherals in STM32 microcontrollers.

AHB-to-APB bridge (APB): The AHB-to-APB bridge provides full synchronous connections between the AHB and the APB bus.

Embedded SRAM: The SRAM can be accessed by bytes, half-words (16 bits) or full words (32 bits), at maximum system clock frequency without wait state and thus by both CPU and DMA.

Flash memory overview

The Flash memory is composed of two distinct physical areas:
• The main Flash memory block. It contains the application program and user data if necessary.
• The information block. It is composed of three parts:
– Option bytes for hardware and memory protection user configuration.
– System memory which contains the proprietary boot loader code.

– OTP (one-time programmable) area.
The Flash interface implements instruction access and data access based on the AHB protocol. It implements the prefetch buffer that speeds up CPU code execution. It also implements the logic necessary to carry out the Flash memory operations (Program/Erase) controlled through the Flash registers.

RCC (Reset and Clock Control): manages system clocks and controls reset operations in STM32 microcontrollers.

GPIO (General-Purpose Input/Output): Specifies the type of pin to be used in STM32, allowing it to receive input and output via these pins.

IOPORT (Input/Output Port) bus: enables data transmission by managing the data paths of these ports within the microcontroller.

2.2. Problem II

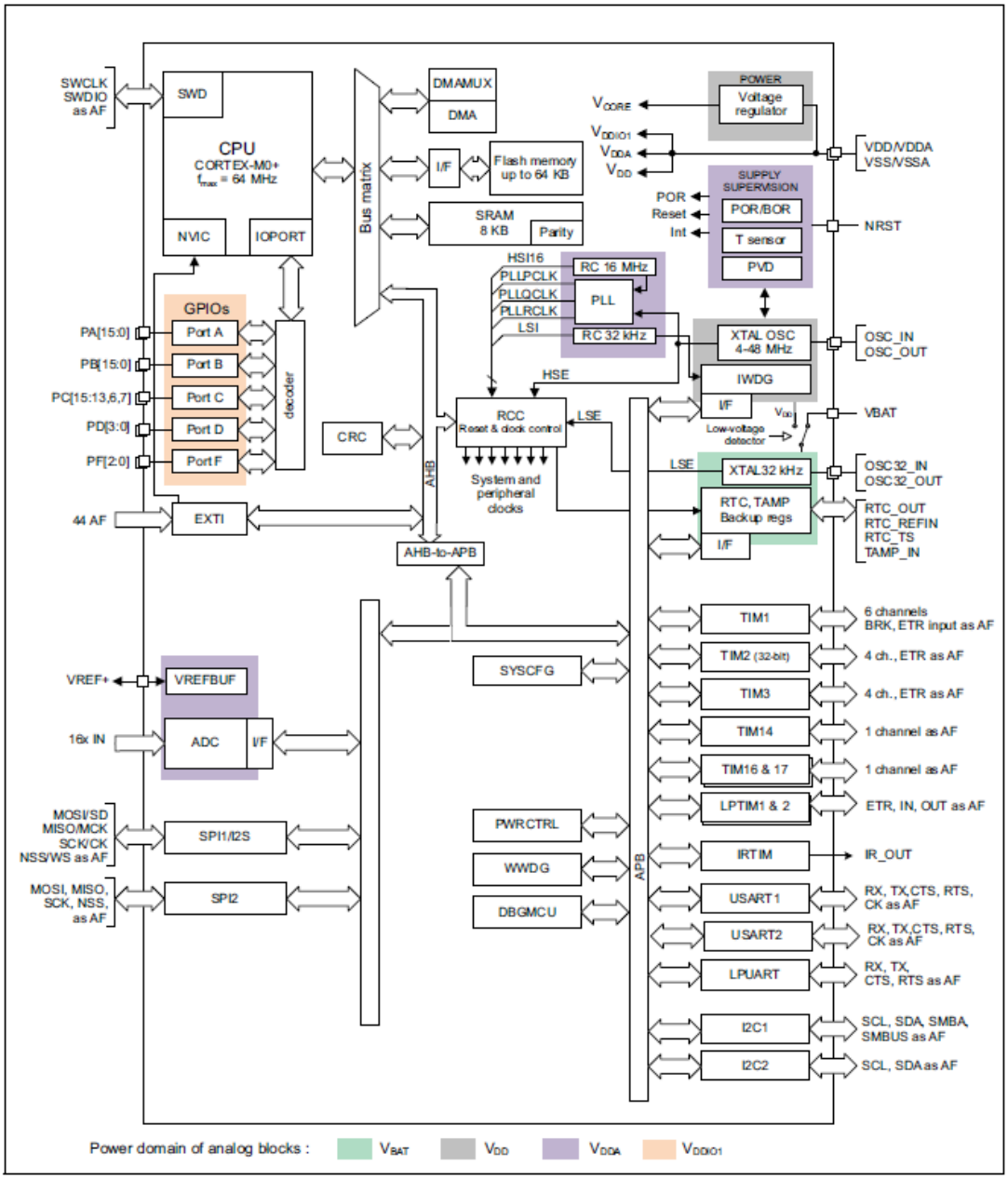

Figure 3. STM32G031 Block Diagram

In Problem 2, the aim is to use A8 pin as output, to pass current through the LED and the resistor and to ensure that it reaches the ground pin, thus lighting the LED. For this; First of all, RCC was activated over the AHB line of the port we will use to control the reset and clock line, and thanks to this, the reset and clock control for GPIOs was activated via IOPORT. This process we have done can be understood by looking at Figure 3.

Bus	Boundary address	Size	Peripheral	Peripheral register map
-	0xE000 0000 - 0xE00F FFFF	1MB	Cortex®-M0+ internal peripherals	-
IOPORT	0x5000 1800 - 0x5FFF FFFF	~256 MB	Reserved	-
	0x5000 1400 - 0x5000 17FF	1 KB	GPIOF	*Section 7.4.12 on page 248*
	0x5000 1000 - 0x5000 13FF	1 KB	GPIOE	*Section 7.4.12 on page 248*
	0x5000 0C00 - 0x5000 0FFF	1 KB	GPIOD	*Section 7.4.12 on page 248*
	0x5000 0800 - 0x5000 0BFF	1 KB	GPIOC	*Section 7.4.12 on page 248*
	0x5000 0400 - 0x5000 07FF	1 KB	GPIOB	*Section 7.4.12 on page 248*
	0x5000 0000 - 0x5000 03FF	1 KB	GPIOA	*Section 7.4.12 on page 248*
AHB	0x4002 6400 - 0x4FFF FFFF	~256 MB	Reserved	-
	0x4002 6000 - 0x4002 63FF	1 KB	AES	*Section 20.7.18 on page 522*
	0x4002 5400 - 0x4002 5FFF	3 KB	Reserved	-
	0x4002 5000 - 0x4002 53FF	1 KB	RNG	*Section 19.7.4 on page 473*
	0x4002 3400 - 0x4002 4FFF	3 KB	Reserved	-
	0x4002 3000 - 0x4002 33FF	1 KB	CRC	*Section 14.4.6 on page 343*
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved	-
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH	*Section 3.7.22 on page 118*
	0x4002 1C00 - 0x4002 1FFF	3 KB	Reserved	-
	0x4002 1800 - 0x4002 1BFF	1 KB	EXTI	*Section 13.5.16 on page 335*
	0x4002 1400 - 0x4002 17FF	1 KB	Reserved	-
	0x4002 1000 - 0x4002 13FF	1 KB	RCC	*Section 5.4.25 on page 218*
	0x4002 0C00 - 0x4002 0FFF	1 KB	Reserved	-
	0x4002 0800 - 0x4002 0BFF	2 KB	DMAMUX	*Section 11.6.7 on page 313*
	0x4002 0400 - 0x4002 07FF	1 KB	DMA2	*Section 10.6.7 on page 296*
	0x4002 0000 - 0x4002 03FF	1 KB	DMA1	*Section 10.6.7 on page 296*
APB	0x4001 5C00 - 0x4001 FFFF	32 KB	Reserved	-
	0x4001 5800 - 0x4001 5BFF	1 KB	DBG	*Section 40.10.5 on page 1378*
	0x4001 4C00 - 0x4001 57FF	3 KB	Reserved	-
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17	*Section 25.6.21 on page 830*
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16	*Section 25.6.21 on page 830*

Figure 4. STM32G0x1 peripheral register boundary addresses

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	RCC_ IOPENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	Reset value																												0	0	0	0	0	0

Figure 5. RCC register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**GPIOx_MODER**	MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]		MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
	Reset value port A	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	Reset value port B to F	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x04	**GPIOx_OTYPER (x = A to F)**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	**GPIOx_OSPEEDR (x = A to F)**	OSPEED15[1:0]		OSPEED14[1:0]		OSPEED13[1:0]		OSPEED12[1:0]		OSPEED11[1:0]		OSPEED10[1:0]		OSPEED9[1:0]		OSPEED8[1:0]		OSPEED7[1:0]		OSPEED6[1:0]		OSPEED5[1:0]		OSPEED4[1:0]		OSPEED3[1:0]		OSPEED2[1:0]		OSPEED1[1:0]		OSPEED0[1:0]	
	Reset value port A	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value port B to F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**GPIOx_PUPDR (x = A to F)**	PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[1:0]		PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
	Reset value port A	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Reset value port B to F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**GPIOx_IDR (x = A to F)**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
	Reset value																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
0x14	**GPIOx_ODR (x = A to F)**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6. GPIO register map and reset values

7.4.1 GPIO port mode register (GPIOx_MODER) (x =A to F)

Address offset:0x00

Reset value: 0xEBFF FFFF for port A

Reset value: 0xFFFF FFFF for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Figure 7. GPIOx_MODER

7.4.6 GPIO port output data register (GPIOx_ODR) (x = A to F)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OD[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the OD bits can be individually set and/or reset by writing to the GPIOx_BSRR register (x = A..D, F).

Figure 8. GPIOx_ODR

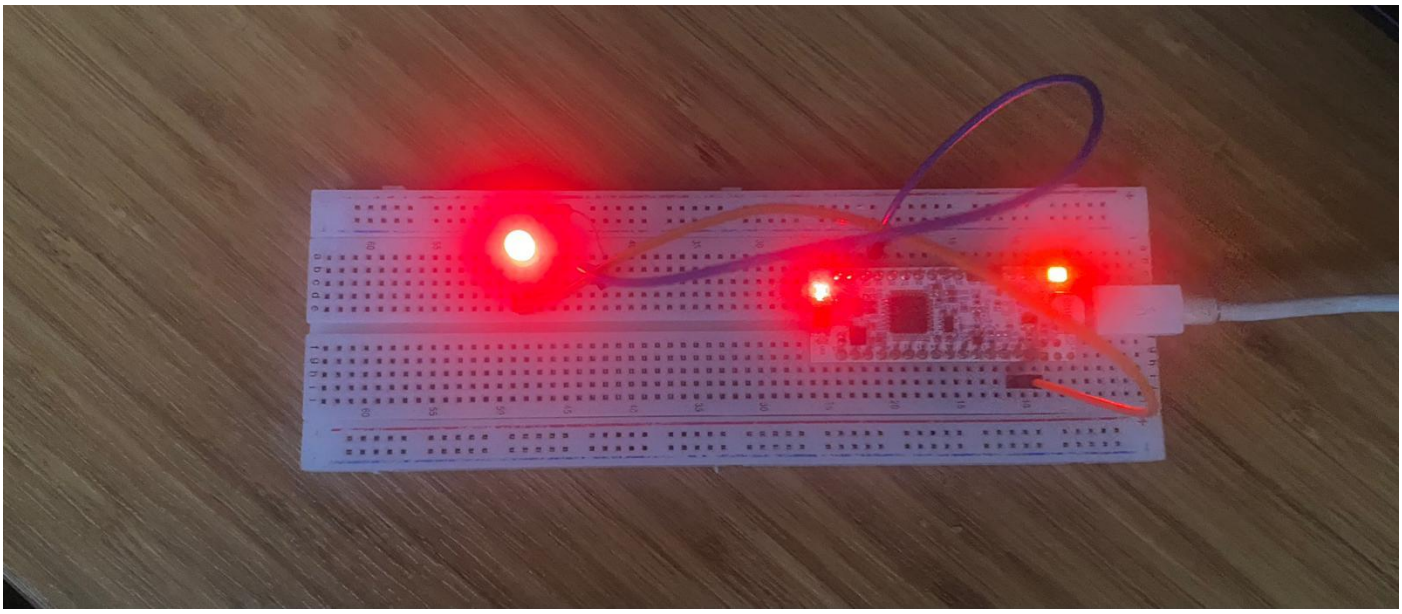

Figure 9. Installation and operation of the circuit

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
 * problem2.s
 */

.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb

/* make linker see this */
.global Reset_Handler
/* get these from linker script */

.word _sdata
.word _edata
.word _sbss
.word _ebss

/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset


.equ GPIOA_BASE,       (0x50000000)          // GPIOA base address

.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset

.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR register offset

/* vector table, +1 thumb mode */
.section .vectors
vector_table:
      .word _estack             /*     Stack pointer */
      .word Reset_Handler +1    /*     Reset handler */
      .word Default_Handler +1  /*       NMI handler */
      .word Default_Handler +1  /* HardFault handler */
      /* add rest of them here if needed */
```

```
/* reset handler */
.section .text
Reset_Handler:
	/* set stack pointer */
	ldr r0, =_estack
	mov sp, r0

	/* initialize data and bss
	 * not necessary for rom only code
	 * */
	bl init_data
	/* call main */
	bl main
	/* trap if returned */
	b .

/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4

	LoopFillZerobss:
		cmp r2, r4
		bcc FillZerobss

	bx lr

/* default handler */
.section .text
Default_Handler:
	b Default_Handler
```

```
/* main function */
.section .text
main:
	/* enable GPIOA clock, bit0 on IOPENR */
	ldr r6, =RCC_IOPENR
	ldr r5, [r6] // R5 = 0x00000000 -> Reset value
	/* movs expects imm8, so this should be fine */
	movs r4, 0x1 // for GPIOA clock enable
	orrs r5, r5, r4
	str r5, [r6] // to send from R5 data to R6 memory

	/* setup PA8 for bits 16-17 in MODER */
	ldr r6, =GPIOA_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	ldr r4, =0x30000 // to change mode of 8th port
	mvns r4, r4 //inverse
	ands r5, r5, r4 // and ->> inverse + and = bics
	ldr r4, =0x10000
	orrs r5, r5, r4
	str r5, [r6]

	/* turn on led connected to A8 in ODR */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	ldr r4, =0x100
	orrs r5, r5, r4
	str r5, [r6]

	/* for(;;); */
	b .

	/* this should never get executed */
	nop
```

To carry out the operation theoretically explained above, the RCC address in the AHB Bus in Figure 4 is assigned to the RCC_BASE terminology with .equ, and the IOPENR (Input-output clock activation register) address is assigned to the RCC_IOPENR terminology by summing it with 0x34 address offset in the RCC_BASE. Similarly, in Figure 4, in IOPORT, the GPIOA address is added to the GPIO_BASE terminology, this terminology is added with the 0x00 address offset given for GPIOx_MODER and this result is assigned to terminology GPIOA_MODER (Mode Register), and the GPIO_BASE terminology is added with the 0x14 address offset given for GPIOx_ODR (Output Data Register) and this result is assigned to terminology GPIOA_ODR. Then, these terminology were assigned to the R6 register, and the bits to be kept in the R5 register were changed using the encrypted "ldr" instruction in the memory of this register, using logical operations (e.g. mvns, ands, orrs, bics) and looking at Figure 5, Figure 7 and Figure 8. It is assigned to the memory register specified by R6 again with the "str" instruction.

2.3 Problem III

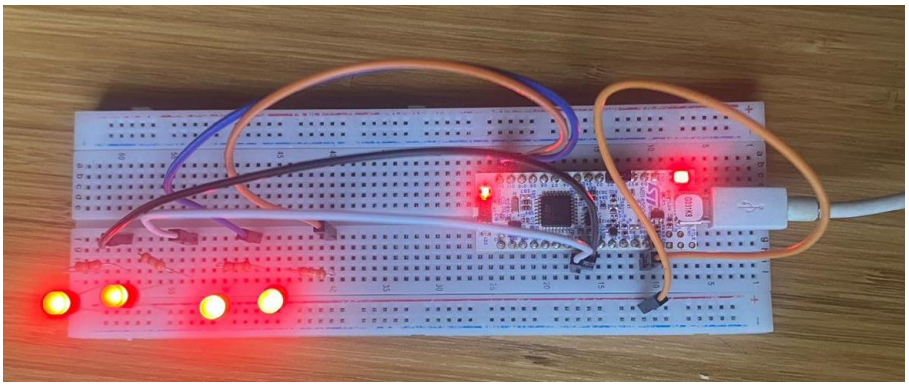

Figure 10. Installation and operation of the circuit

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
 * problem3.s
 */


.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOA_BASE,       (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR register offset

.equ GPIOB_BASE,       (0x50000400)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR register offset


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
       .word _estack             /*     Stack pointer */
       .word Reset_Handler +1    /*     Reset handler */
       .word Default_Handler +1  /*       NMI handler */
       .word Default_Handler +1  /* HardFault handler */
       /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
       /* set stack pointer */
       ldr r0, =_estack
       mov sp, r0

       /* initialize data and bss
        * not necessary for rom only code
        * */
       bl init_data
       /* call main */
       bl main
       /* trap if returned */
       b .
```

```
/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4

	LoopFillZerobss:
		cmp r2, r4
		bcc FillZerobss

	bx lr


/* default handler */
.section .text
Default_Handler:
	b Default_Handler


/* main function */
.section .text
main:
	/* enable GPIOA clock, bit2 on IOPENR */
	ldr r6, =RCC_IOPENR
	ldr r5, [r6] // R5 = 0x00000000 -> Reset value
	/* movs expects imm8, so this should be fine */
	movs r4, 0x3 // for GPIOA and GPIOB clock enable
	orrs r5, r5, r4
	str r5, [r6] // to send from R5 data to R6 memory

	/* setup PA11 and PA12 in MODER */
	ldr r6, =GPIOA_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	ldr r4, =0x3C00000 // to change mode of 11th and 12th ports
	mvns r4, r4 //inverse
	ands r5, r5, r4 // and ->> inverse + and = bics
	ldr r4, =0x1400000
	orrs r5, r5, r4
	str r5, [r6]
```

```
/* setup PB4 and PB5 in MODER */
	ldr r6, =GPIOB_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	ldr r4, =0xF00 // to change mode of 4th and 5th ports
	mvns r4, r4 //inverse
	ands r5, r5, r4 // and ->> inverse + and = bics
	ldr r4, =0x500
	orrs r5, r5, r4
	str r5, [r6]

	/* turn on led connected to A11 and A12 in ODR */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	ldr r4, =0x1800
	orrs r5, r5, r4
	str r5, [r6]

	/* turn on led connected to B4 and B5 in ODR */
	ldr r6, =GPIOB_ODR
	ldr r5, [r6]
	ldr r4, =0x30
	orrs r5, r5, r4
	str r5, [r6]

	/* for(;;); */
	b .

	/* this should never get executed */
	nop
```

In Problem 3, different GPIOB and GPIOA ports than in Problem 2 were used simultaneously. Figure 4 and Figure 5 show how these are activated. Then, the desired pins of these GPIOs for their own blocks (A and B) are set so that PA11, PA12, PB4, PB5 pins give output by looking at Figure 7 and Figure 8.

2.4. Problem IV

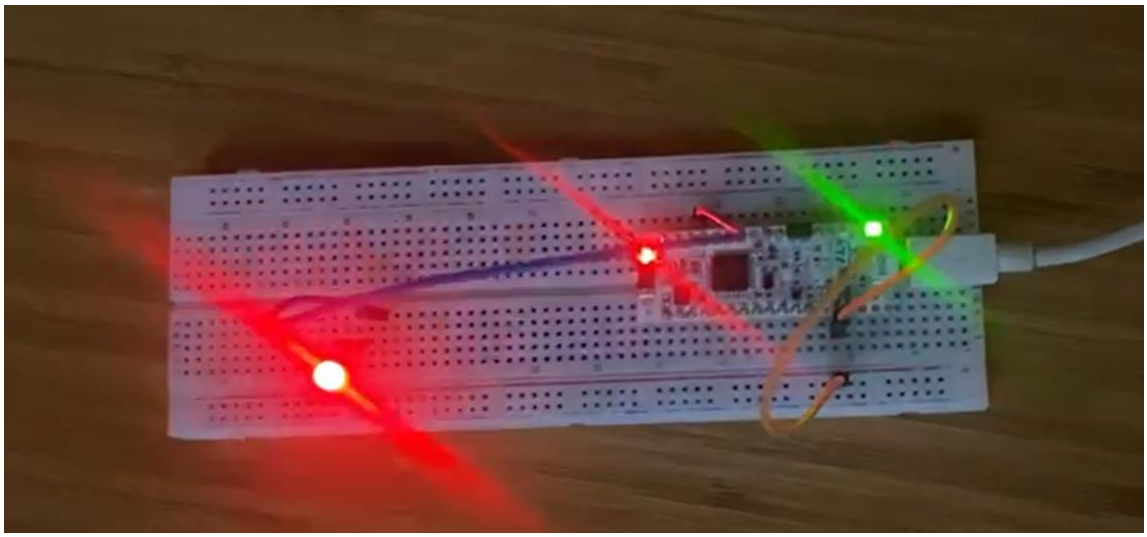

Figure 11. Installation and operation of the circuit

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
 * problem4.s
 */

.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOA_BASE,       (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR register offset


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
       .word _estack             /*     Stack pointer */
       .word Reset_Handler +1    /*     Reset handler */
       .word Default_Handler +1  /*       NMI handler */
       .word Default_Handler +1  /* HardFault handler */
       /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
       /* set stack pointer */
       ldr r0, =_estack
       mov sp, r0

       /* initialize data and bss
        * not necessary for rom only code
        * */
       bl init_data
       /* call main */
       bl main
       /* trap if returned */
       b .
```

```
/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4

	LoopFillZerobss:
		cmp r2, r4
		bcc FillZerobss

	bx lr


/* default handler */
.section .text
Default_Handler:
	b Default_Handler


/* main function */
.section .text
main:
	/* enable GPIOA clock, bit0 on IOPENR */
	ldr r6, =RCC_IOPENR
	ldr r5, [r6] // R5 = 0x00000000 -> Reset value
	/* movs expects imm8, so this should be fine */
	movs r4, 0x1 // for GPIOA clock enable
	orrs r5, r5, r4
	str r5, [r6] // to send from R5 data to R6 memory

	/* setup for PA8 bit for 16-17 bits in MODER */
	ldr r6, =GPIOA_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	ldr r4, =0x30000 // to change mode of 8th port
	mvns r4, r4 //inverse
	ands r5, r5, r4 // and ->> inverse + and = bics
	ldr r4, =0x10000
	orrs r5, r5, r4
	str r5, [r6]
```

```
loop:
	/*
		set led
		delay
		clear led
	*/
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	ldr r4, =0x100
	orrs r5, r5, r4
	str r5, [r6]

	ldr r1, =6000000
	bl delay

	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	ldr r4, =0x100
	bics r5, r5, r4
	str r5, [r6]

	ldr r1, =6000000
	bl delay

	b loop


delay:
	subs r1, #1 // r1 = r1 - 1
	bne delay // r1 is not equal to 0.
	bx lr // r1 is equal to 0.

	/* for(;;); */
	b .
```

This Assembly code contains a simple block of code that enables the microcontroller to blink an LED as in problem 2. In addition, the 'delay' function allows the written program to wait in the loop for a specified period of time (the desired time is 1 second). The 'delay' function decreases the value in register r1 by 1 (subs r1, #1). Then, it checks whether r1 is equal to zero with the bne (branch if not equal) command. If r1 has a non-zero value, it returns to the 'delay' function and the processor continues to reduce and divide again in the next cycle. If r1 is equal to zero, the function is exited with the bx lr command and returned to the place where the function was called with the help of lr (link register).

This method is used to provide a wait for a specified period of time (1 second for this problem 4). The larger the r1 register value, the longer the waiting time will be. In this code block, the waiting time is 6000000 because this value is assigned to register r1 with ldr r1, =6000000 lines of code. By changing this value, the waiting time can be adjusted.

3. 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