GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

ELECTRONICS ENGINEERING

ELEC 334

MICROPROCESSORS

LAB 01

Name - Surname	Yağmur DERYA
Student ID	171024011

1. Introduction

In this lab, it is aimed to get familiarized with the Nucleo G031K8 board, understand the ICs and their connections with the microprocessor and practice assembly language.

2. Problems

2.1. Problem 1 – All ICs and Their Usage

2.1.1. 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

2.1.2. DMA bus

This bus connects the AHB master interface of the DMA to the bus matrix that manages the access of CPU and DMA to SRAM, Flash memory and AHB/APB peripherals.

2.1.3. Bus matrix

The bus matrix manages the access arbitration between the core system bus and the DMA master bus. The bus matrix is composed of masters (CPU, DMA) and slaves (Flash memory interface, SRAM and AHB-to-ABP bridge).

2.1.4. AHB-to-APB Bridge

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

2.1.5. Embedded SRAM

This device has 8 Kbytes of SRAM regardless of the parity check configuration. 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

2.1.6. Flash memory

The Flash interface implements instruction access and data access based on the AHB protocol. It implements the prefetch buffer that speeds up CPU code execute. It also implements the logic necessary to carry out the flash memory operations controlled through the flash registers.

2.1.7. Embedded boot loader

The embedded boot loader is located in the System memory, programmed by ST during production. It is used to reprogram the Flash memory using one of the following serial interfaces:

- USART on pins PA2/PA3, PA9/PA19
- 12C on pins PB6/PB7 or PB10/PB11

2.1.8. Power control (PWR)

The STM32G0x1 devices require a 1.7 V to 3.6 V operating supply voltage.

Two embedded linear voltage regulators supply all the digital circuitries except for the Standby circuitry and the RTC domain.
The voltage regulators are always enabled after a reset.
The dynamic voltage scaling is a power management technique which consists in increasing or decreasing the voltage used for the digital peripherals according to the application performance and power consumption needs.

The peripheral registers can be accessed by half-words (16-bit) or words (32-bit).

2.1.9. Peripheral clock

Each peripheral clock can be enabled by the corresponding enable bit of the RCC_AHBENR or RCC_APBENRx regiters.

When the peripheral clock is not active, the peripheral registers read or write accesses are not supported.

2.1.10. Direct memory access controller (DMA)

It is a bus master and system peripheral. It is used to perform programmable data transfers between memory-mapped peripherals and memories, upon the control of an off-loaded CPU. The DMA controller features a single AHB master architecture. There is one instance of DMA with 5 channels on STM32G031xx. Each channel is dedicated to managing memory access requests from one or more peripherals. The DMA includes an arbiter for handling the priority between DMA requests.
Each channel may handle a DMA transfer between a peripheral register located at a fixed address and a memory address. The DMA channel is programmed at block transfer level.

2.1.11. Serial peripheral interface (SPI)

The SPI allow synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt.

2.1.12. Universal synchronous receiver transmitter (USART)

It offers a flexible means to perform Full-duplex data exchange with external equipments requiring an industry standard NRZ asynchronous serial data format. A very wide range of baud rates can be achieved through a fractional baud rate generator.

High-speed data communications are possible by using the DMA for multibuffer configuration.

2.1.13. Universal asynchronous receiver transmitter (UART)

It is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It is usually an individual integrated circuit used for serial communications over a computer or peripheral device serial port.

2.1.14. Low-power universal asynchronous receiver transmitter (LPUART)

The LPUART is an UART which enables bidirectional UART communications with a limited power consumption.

2.1.15. STM32G031K8 pin connections

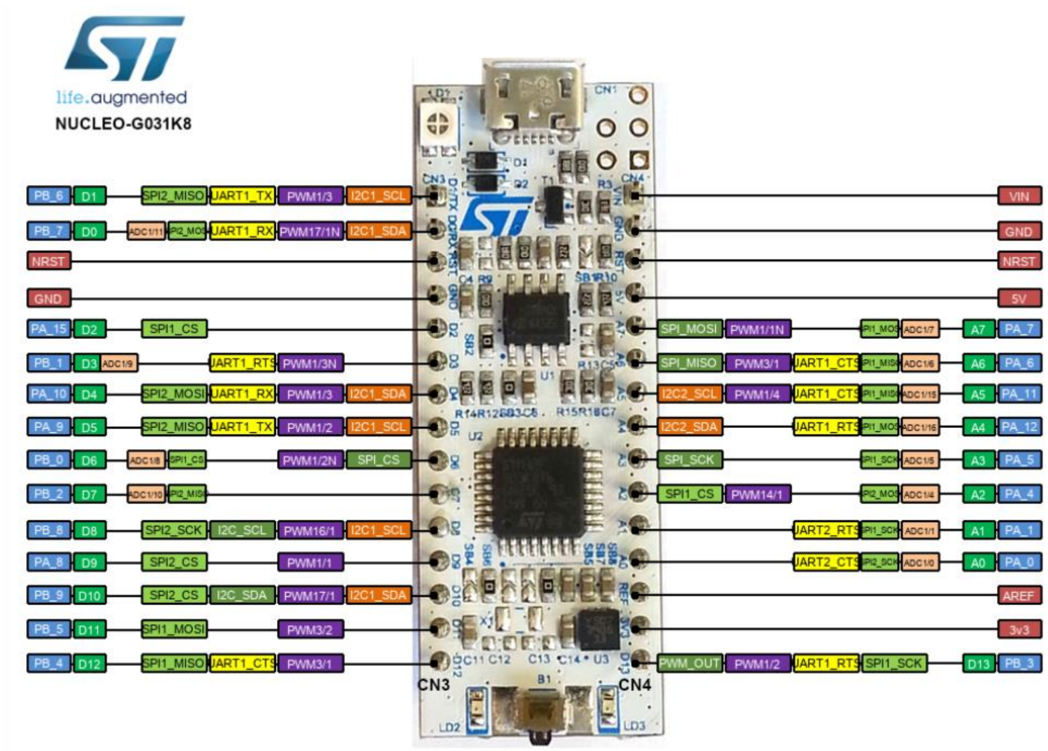

Figure 1. STM32G031K8

2.2. Problem 2 – Assembly code to light up 1 LED at PA8

In this problem, it is assumed that the LED is connected to pin PA8. By using RM0444 Reference manual;

from page 47, GPIOA base variable,

from page 205, port mode register variable,

from page 207, output data register variable is assigned.

```
Stack_Size      EQU     0x00000400


RCC_BASE             EQU    0x40021000    ;
RCC_IOPENR           EQU    RCC_BASE   + (0x34)
```

```
;GPIO-A control registers
GPIOA_BASE    EQU    (0x50000000)
GPIOA_MODER          EQU     GPIOA_BASE + (0x00)
GPIOA_ODR            EQU           GPIOA_BASE + (0x14)


AREA    STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE   Stack_Size
__initial_sp

    THUMB

    AREA    RESET, DATA, READONLY
    EXPORT  __Vectors

__Vectors
    DCD     __initial_sp                   ; Top of Stack
    DCD     Reset_Handler                  ; Reset Handler
    DCD     NMI_Handler                    ; NMI Handler
    DCD     HardFault_Handler              ; Hard Fault Handler

    AREA    |.text|, CODE, READONLY


; nmi handler
NMI_Handler    PROC
    EXPORT  NMI_Handler
    B .
    ENDP


; hardfault handler
HardFault_Handler    PROC
    EXPORT  HardFault_Handler
    B .
    ENDP


; entry function
Reset_Handler    PROC
    EXPORT  Reset_Handler
    ; Edit below this line

       ; enable GPIOA clock, bit1 on IOPENR
       LDR R6, =RCC_IOPENR
       LDR R5, [R6]
       ; movs expects imm8, so this should be fine
       MOVS R4, 0x1
       ORRS R5, R5, R4
       STR R5, [R6]

       ; setup PA8 for led 01 for bits 17-16 in MODER
       LDR R6, =GPIOA_MODER
       LDR R5, [R6]
       ; cannot do with movs, so use pc relative
       LDR R4, =0x3000
       MVNS R4, R4
       ANDS R5, R5, R4
       LDR R4, =0x1000
       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]

        B .
        ENDP

        END
```

2.3. Problem 3 - Assembly code to light up 4 LED

In this problem, it is assumed that the 4 LEDs are connected to pins PA11, PA12, PB4, PB5. By using RM0444 Reference manual;

from page 47, GPIOA and GPIOB base variables,
from page 205, port mode register variables,
from page 207, output data register variables are assigned.

```
Stack_Size      EQU     0x00000400


RCC_BASE             EQU    0x40021000    ;
RCC_IOPENR           EQU    RCC_BASE   + (0x34)


;GPIO-A control registers
GPIOA_BASE    EQU    (0x50000000)
GPIOA_MODER          EQU     GPIOA_BASE + (0x00)
GPIOA_ODR            EQU           GPIOA_BASE + (0x14)

;GPIO-B control registers
GPIOB_BASE   EQU         (0x50000400)
GPIOB_MODER    EQU        GPIOB_BASE + (0x00)
GPIOB_ODR      EQU        GPIOB_BASE + (0x14)


AREA    STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE   Stack_Size
__initial_sp

    THUMB

    AREA    RESET, DATA, READONLY
    EXPORT  __Vectors

__Vectors
    DCD     __initial_sp                    ; Top of Stack
    DCD     Reset_Handler                   ; Reset Handler
    DCD     NMI_Handler                     ; NMI Handler
    DCD     HardFault_Handler               ; Hard Fault Handler

    AREA    |.text|, CODE, READONLY


; nmi handler
NMI_Handler    PROC
```

```
    EXPORT  NMI_Handler
    B .
    ENDP


; hardfault handler
HardFault_Handler    PROC
    EXPORT  HardFault_Handler
    B .
    ENDP


; entry function
Reset_Handler    PROC
    EXPORT  Reset_Handler
    ; Edit below this line

       ; PA11
       ; enable GPIOA clock, bit1 on IOPENR
       LDR R6, =RCC_IOPENR
       LDR R5, [R6]
       ; movs expects imm8, so this should be fine
       MOVS R4, 0x1
       ORRS R5, R5, R4
       STR R5, [R6]

       ; setup PA11 for led 01 for bits 23-22 in MODER
       LDR R6, =GPIOA_MODER
       LDR R5, [R6]
       ; cannot do with movs, so use pc relative
       LDR R4, =0x400000
       ORRS R5, R5, R4
       STR R5, [R6]

       ; turn on led connected to A11 in ODR
       LDR R6, =GPIOA_ODR
       LDR R5, [R6]
       LDR R4, =0x800
       ORRS R5, R5, R4
       STR R5, [R6]

       ; PA12
       ; movs expects imm8, so this should be fine
       MOVS R4, 0x1
       ORRS R5, R5, R4
       STR R5, [R6]

       ; setup PA12 for led 01 for bits 25-24 in MODER
       LDR R6, =GPIOA_MODER
       LDR R5, [R6]
       ; cannot do with movs, so use pc relative
       LDR R4, =0x1000000
       ORRS R5, R5, R4
       STR R5, [R6]

       ; turn on led connected to A12 in ODR
       LDR R6, =GPIOA_ODR
       LDR R5, [R6]
       LDR R4, =0x1000
       ORRS R5, R5, R4
       STR R5, [R6]
```

```
; PB4
; movs expects imm8, so this should be fine
MOVS R4, 0x2
ORRS R5, R5, R4
STR R5, [R6]

; setup PB4 for led 01 for bits 9-8 in MODER
LDR R6, =GPIOB_MODER
LDR R5, [R6]
; cannot do with movs, so use pc relative
LDR R4, =0x300
MVNS R4, R4
ands R5, R5, R4
LDR R4, =0x100
ORRS R5, R5, R4
STR R5, [R6]

; turn on led connected to B4 in ODR
LDR R6, =GPIOB_ODR
LDR R5, [R6]
LDR R4, =0x1000
ORRS R5, R5, R4
STR R5, [R6]

; PB5
; movs expects imm8, so this should be fine
MOVS R4, 0x2
ORRS R5, R5, R4
STR R5, [R6]

; setup PB5 for led 01 for bits 11-10 in MODER
LDR R6, =GPIOB_MODER
LDR R5, [R6]
; cannot do with movs, so use pc relative
LDR R4, =0x400
ORRS R5, R5, R4
STR R5, [R6]

; turn on led connected to B5 in ODR
LDR R6, =GPIOB_ODR
LDR R5, [R6]
LDR R4, =0x2000
ORRS R5, R5, R4
STR R5, [R6]

B .
ENDP

END
```

2.4. Problem 4 – Toggle

In this problem I used the code which is written for Problem 2 and added a delay and toggle.

To calculate how many delay loop is needed for 1 second, I tested the code with a random delay number and calculate how much time the delay part takes.

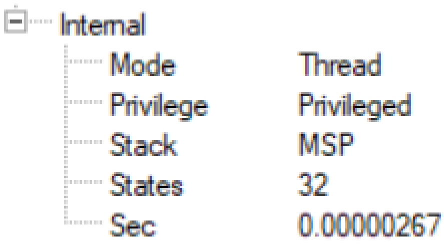

Figure 2. Before delay.

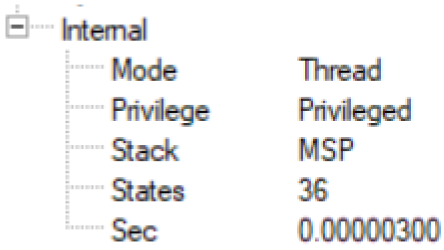

Figure 3. After delay.

Delay loop takes $(300 - 267)x10^{-8} = 33x10^{-8}$ seconds. To make delay 1 second we need $\frac{1}{33x10^{-8}} = 3030303$ loops.

To turn of the led, we need to make ODR 0. To make ODR 0, i used ANDS command.

```
        Stack_Size      EQU     0x00000400


        RCC_BASE             EQU     0x40021000    ;
        RCC_IOPENR           EQU     RCC_BASE    + (0x34)


        ;GPIO-A control registers
        GPIOA_BASE     EQU     (0x50000000)
        GPIOA_MODER          EQU      GPIOA_BASE + (0x00)
        GPIOA_ODR            EQU            GPIOA_BASE + (0x14)


        AREA    STACK, NOINIT, READWRITE, ALIGN=3

        Stack_Mem       SPACE   Stack_Size
        __initial_sp

            THUMB

            AREA    RESET, DATA, READONLY
            EXPORT  __Vectors

        __Vectors
            DCD     __initial_sp                   ; Top of Stack
            DCD     Reset_Handler                  ; Reset Handler
            DCD     NMI_Handler                    ; NMI Handler
            DCD     HardFault_Handler              ; Hard Fault Handler

            AREA    |.text|, CODE, READONLY


        ; nmi handler
        NMI_Handler     PROC
            EXPORT  NMI_Handler
            B .
            ENDP


        ; hardfault handler
        HardFault_Handler    PROC
```

```
            EXPORT  HardFault_Handler
            B .
            ENDP


        ; entry function
        Reset_Handler    PROC
            EXPORT  Reset_Handler
            ; Edit below this line

               ; enable GPIOA clock, bit1 on IOPENR
               LDR R6, =RCC_IOPENR
               LDR R5, [R6]
               ; movs expects imm8, so this should be fine
               MOVS R4, 0x1
               ORRS R5, R5, R4
               STR R5, [R6]

               ; setup PA8 for led 01 for bits 17-16 in MODER
               LDR R6, =GPIOA_MODER
               LDR R5, [R6]
               ; cannot do with movs, so use pc relative
               LDR R4, =0x3000
               MVNS R4, R4
               ANDS R5, R5, R4
               LDR R4, =0x1000
               ORRS R5, R5, R4
               STR R5, [R6]

        toggle
               ; 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]

               LDR R0, =3030303
        delay
               SUBS R0, #1
               CMP R0, #0
               BNE delay


               ; turn of led connected to A8 in ODR
               LDR R6, =GPIOA_ODR
               LDR R5, [R6]
               MOVS R4, #0x0
               ANDS R5, R5, R4
               STR R5, [R6]

               B toggle

               B .
               ENDP

               END
```

3. References

[1] RM0444 Reference manuel

[2] https://github.com/fcayci/stm32g0