Microcontroller Lab Assignment 01: Road Traffic Management

Prepared by: Dr. Mosaddek Tushar, Professor
Other Faculty: Redwan Ahmed Rizvee
Computer Science and Engineering, University of Dhaka,

Version 1.0

Demo Due Date: The following weeks as separate parts.

Please report if you find errors or anomalies in the description.

Jan 21, 2023

Contents

1 Objectives and Policies

1.1 General Objectives

The objectives of the lab assignment are to understand and have hands-on training in microcontroller (STM32F446RE) GPIO configuration and the operation of GPIO pins and ports as input and output.

1.2 Assessment Policy

The student must complete and demonstrate the assignment by the due date. After completing the assignment student must submit a report and upload the code to the microcontroller lab website or google classroom. The students are obligated to complete the given problem; however, we encourage further assignment extension to replicate a realistic application environment. The student may get bonus marks for demonstrating an innovative extension of the lab assignment.

2 Alert

It would be best if you take care of the microcontroller STM32F446RE; the input power to any GPIO pin must not exceed 3.6V. Do not damage or deform any GPIO pin, and do not use a conducting surface as the operating podium; otherwise, it may damage the microcontroller. Always connect the microcontroller to the computer USB type-A to type-B to power on and program.

3 What to do?

The assinment has two parts: (Later see the detail description)

- Write a simple blinky program to blink the on board User-LED.
- Design and develop a road traffic controller system's MCU (Micro-Controller Unit) firmware (program).

4 Assignment Problem Description

To understand the operation and procedure for communication with the outside world through GPIOx ports and PINs, you must develop and solve the following two lab assignments.

4.1 Blinky Program

To create a simple onboard LED blinking program, you must configure the GPIO pin 'PA5'. PA5 pin resides under the GPIOA port. The configuration steps are:

- Configure $RCC->AHB1ENR$ register to enable GPIOA clock
- Configure Pin PA5 for output.
- Write a function given below to set and reset a GPIOx port
- Use ms_delay(delay) for time between on and off.

4.2 Traffic Control System

The following figure 1 presents traffic at the crossroad. In our country, traffic police usually control road traffic; however, the government has policies to alleviate our country into a digital and gradually smart Bangladesh. Thus it (police hand controlled) is against the policy. Now, the CSE third-year student wants to act positively. They plan to develop an intelligent traffic management system to reach to government's goal and make a usable traffic system for Bangladesh. The students want to emphasize the practical traffic control system (firmware) to control road traffic now rather than a high-level decision-making system. They plan to develop it first and then take a left turn for the rest of the work for Fourth-year, MS, or Ph.D. students to use the intelligence in the traffic control system. Before going to the field implementation, they try it first in the microcontroller lab as their lab assignment.

Figure 1: Traffic in the Cross Road

To simplify, they first define some constraints: the traffic can only go forward and left directions. They put three color LED (RYG – Red, Yellow, and Green) for the trafic signal. Traffic cannot flow forward or turn left when the red LED (in front of them) is ON. Traffic can flow straight and turn left only if the green LED is ON. The delay between RED to green is 60 seconds, from green to yellow 5 seconds, and from yellow to RED another 2 seconds. However, the students want to make it more adaptive. Their developed system recognizes the traffic on either side of the road. At the time of detection, if their application finds that the traffic load is less (one or two), they keep the green light glowing for 15 seconds more. However, their application maintains the same timing for an equal number of vehicles. They place three LEDs in each road direction to calculate the traffic load. Three LED ON indicates higher traffic load. Two or one means the traffic load is low. When the green light (LED) is ON, the developed solution moves traffic forward and takes a left turn. Also, the application adds random traffic to each road. It means switching ON a LED (vehicle);

4.2.1 Technical Specification

You need 12-LED to place on either side of the crossroad to display traffic and another 12-LED for traffic lights. Let us assume that the lights are visible from either direction. Therefore, we can reduce the number of LEDs from 12 to 6 for the traffic signal. It would be best if you had 12 (traffic load) output and input (both common) to identify and show the current traffic. The configuration steps are given as follows:

- Enable clocks for the selected ports
- Configure 12 as output and 12-input (same 12-LEDs) for traffic
- Configure another 12 PINs as output for the traffic lights/signal.
- Develop the process to control (i) Traffic lights, (ii) Incoming vehicles (random), and (iii) Input the traffic load and process to control the traffic light.
- Select the ports and pins for traffic lights and traffic (input and output) beforehand.
- Use the developed functions created in the blinky program to ON and OFF the LED.

5 Preliminaries

This section contains a detailed description of the operating characteristics of the MCU digital input/output system. The primary target is to make you understand the internal architecture of the STM microcontroller's digital input/output system. Including the ARM Cortex-M4 microprocessor, the MCU has several digital and analog input and output communication ports known as GPIO ports. The **GPIO ports** are connected to the microprocessor through an MCU internal BUS AHB1 via the AHB bus matrix.

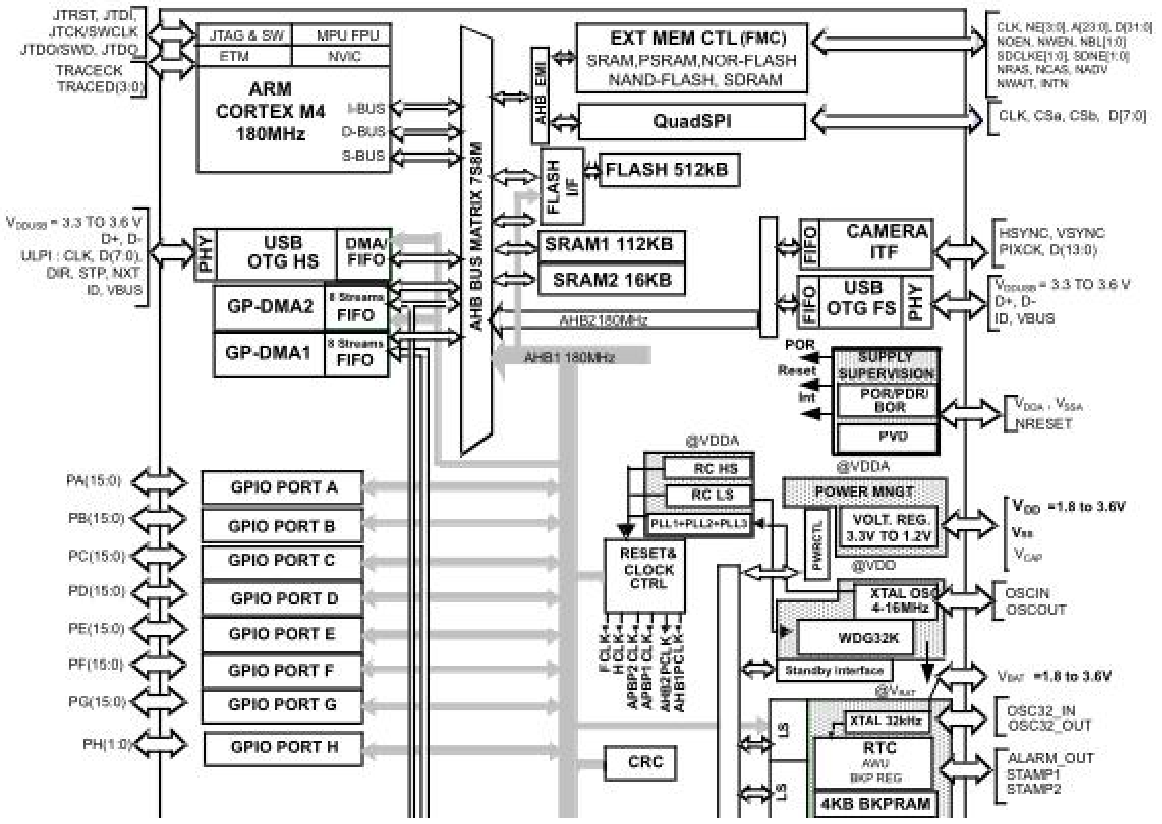


Figure 2: GPIOx connection bus AHB1 to Cortex-M4

The maximum clock speed of the Bus **AHB1** is 180 MHz; however, you can configure it at a desired clock speed of up to 180 MHz. For our current assignment, we configure the clock speed to **180 MHz**. The STM32F446re has **eight** GPIO ports as GPIOA, GPIOB, GPIOC, GPIOD, GPIOE, GPIOF, GPIOG, and GPIOH. Figure 2 shows the bus architecture that connects GPIOx to the microprocessor (Cortex-M4). The commercial name of our MCU Development board STM32F446RE is NUCLEO-64 or NUCLEO-F446RE.

5.1 GPIOx Port (Replace 'x' with A, .., or H)

The MCU has Ardiouno UNO V3 compatible 64-GPIO pins organized 16-pins as a GPIO port. The datasheet and reference manual designed these pins as PA1, PA2, $\cdots$ PH16, where 'P' refers to 'PIN,' 'A' refers to GPIO port A or GPIOA, and '1' is the PIN for PA1. Note that the micro-controller may have some ports (or pins) but not all; see the datasheet table-10 for the available ports and pins. The ARM architect (Advanced RISC Machines Ltd.) keeps this flexibility for the microcontroller manufacturer. The MCU can communicate to the outside world through these GPIOx ports. The programmer or developers can use the ports' PINs for various functionality, including ADC, DAC, USART, SPI, I2C, Audio, Video, PWM, and control. Figure 3 shows the PIN diagram of the NUCLEO-F446RE MCU development board.

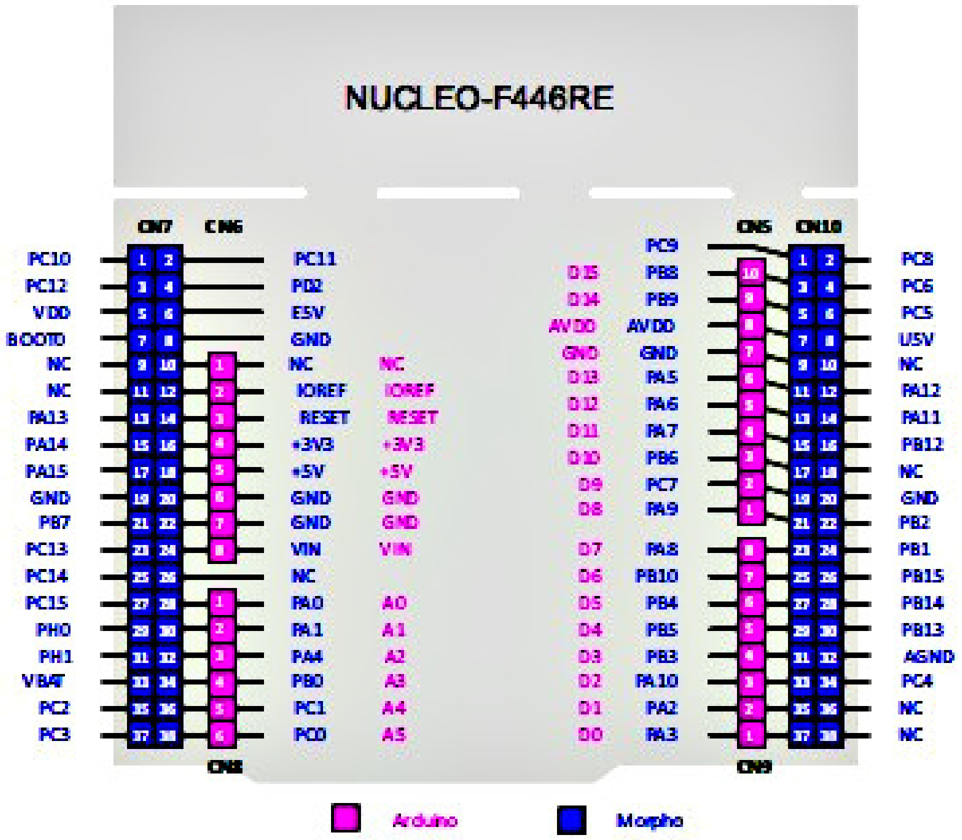
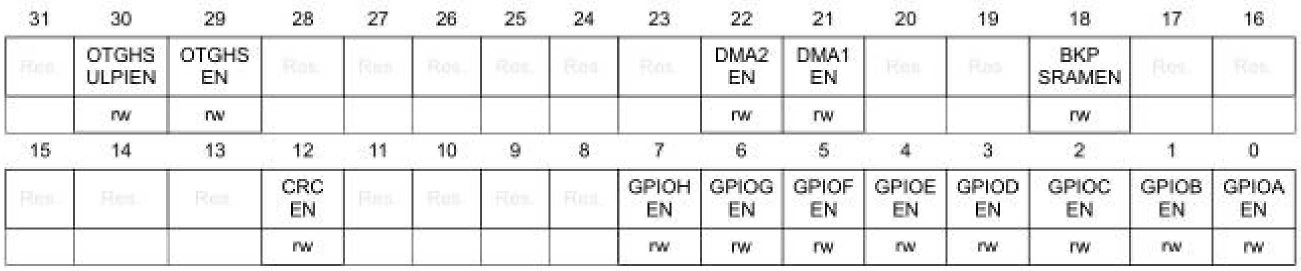


Figure 3: NUCLEO-F446RE pin diagram

5.1.1 GPIOx Port Configuration

Keil has a rich library for detailed configuration of the different components of the MCU. Now, we only discuss the configuration of the GPIOx port. Each GPIOx port has a set of registers (32 bits) to configure the ports and pins of the port for a target application. Before going into the detail of the register sets, first, we need to enable a GPIO port clock. In the reference manual, the RCC chapter (Chapter 6: Reset and clock control (RCC)) has detailed clock configuration information to enable a GPIO port. Section 6.3.10 of chapter 6 discusses the RCC AHB1 peripheral clock enable register (RCC_AHB1ENR). All the peripherals connected to AHB1 need this register to configure. The following Figure 4 shows 32-bits RCC_AHB1ENR register:



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPIEN	OTGHS EN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 EN	DMA1 EN	Res.	Res.	BKP SRAMEN	Res.	Res.
	rw	rw							rw	rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CRC EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw	rw	rw	rw	rw	rw	rw	rw

Figure 4: RCC_AHB1ENR register

To enable the GPIOA port, you must write ‘1’ in the bit position ‘0’ of the RCC_AHB1ENR.

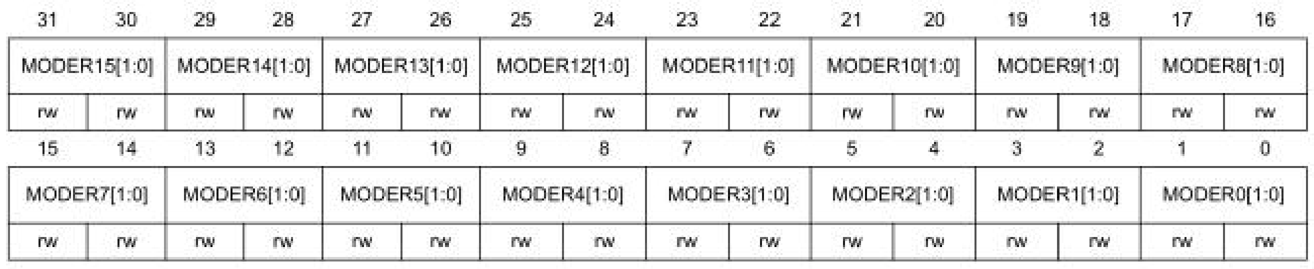
Using the ARM IDE Keil you can access/update the RCC_AHB1ENR register using the following command:

```
/**
* Comments:
* ‘<<’: shift operation, ‘|=’: bitwise OR,
* ‘RCC->AHB1ENR’ -- the register address defined
* in KEIL Library (Base: 0x4002 3800) + 0x30 (offset)
**/

RCC->AHB1ENR |= (1<<0); //Enable GPIOA
```

Each GPIOx port associated essential registers set (for each GPIOx port)listed below (see reference manual, Chapter 7):

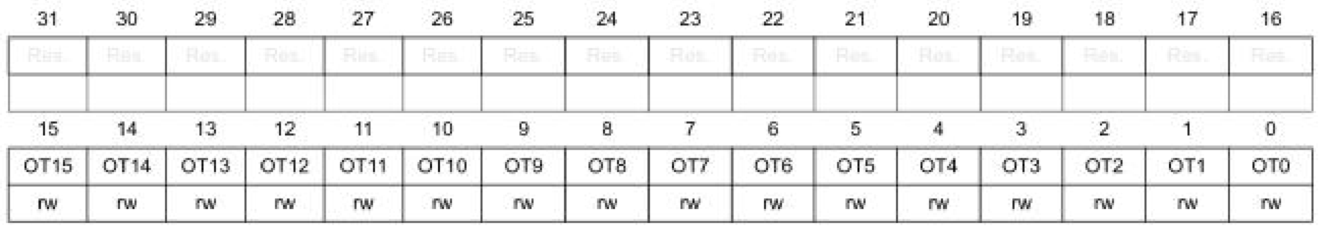
- **GPIOx_MODER** : Mode register of GPIOx. Use to configure the associated pin for (i) Input (00b), (ii) General purpose output (01b), (iii) Alternate Function Mode (10b), and (iv) Analog mode (11b). Two-bits for each PIN. Default ‘00’



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 5: GPIOx MODER Register

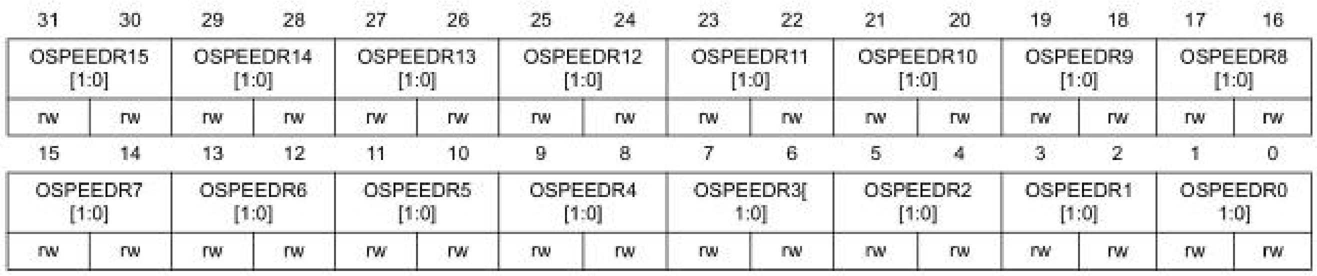
- **GPIOx_OTYPER**: GPIO port output type register. Upper 16-bits reserved. Lower 16 bits defines the output type: (i) Output push-pul (‘0’), (ii) Output open drain (‘1’). Default ‘0’.



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
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 6: GPIOx Output Type Register

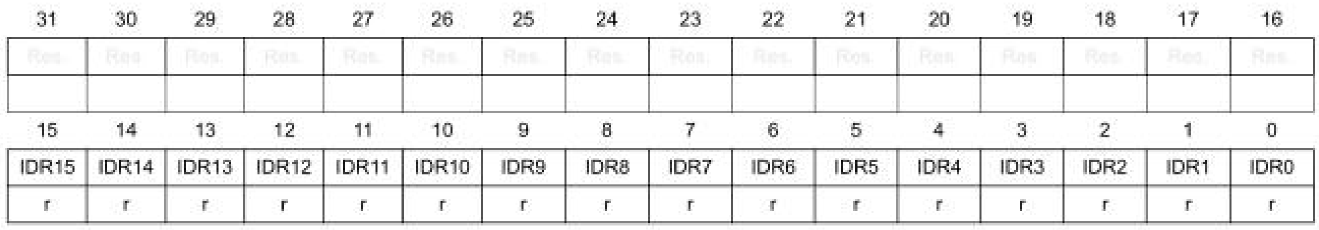
- **GPIOx_OSPEEDR**: GPIO port output speed register. Two bits for each pin speed configuration. Options are (i) ‘00’: Low speed, (ii) ‘01’: Medium speed, (iii) ‘10’: Fast speed, and (iv) High speed. Default ‘00’



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [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**
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3[1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 7: GPIOx PIN Speed configuration Register

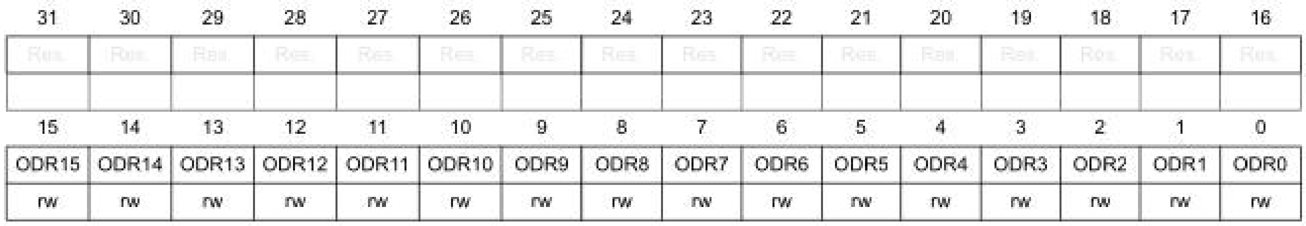
- **GPIOx_IDR**: Read-only. GPIOx input register. Upper 16 bits reserved. Each bit of the lower 16-bit register shows the input pin's current state. '0': if no input or grounded (active low), '1': active input – high. Default '0'



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**
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Figure 8: GPIOx Input Status Register

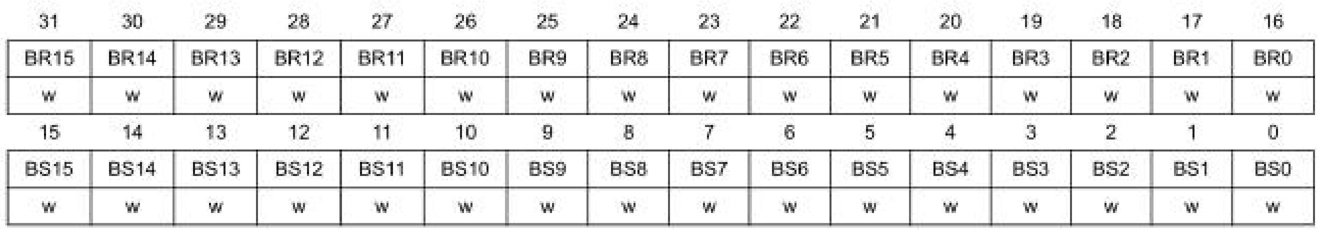
- **GPIOx_ODR**: GPIOx port output data register. Upper 16 bits reserved. Each LSB bit of 16 display the output status of the corresponding pin of the port. This register is read and writable. We are using **GPIOx_BSRR** register; we can set or reset individual pins (or bits). Default '0'



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**
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 9: GPIOx Output Register

- **GPIOx_BSRR**: Each lower bit of 0-15 is dedicated to setting a pin. We are writing this bit to the '1' set bit corresponding **GPIOx_ODR** bit; thus, the pin output is high. To reset the pin, write '1' to the upper 16-bit. Each bit of 16-31 is dedicated to resetting the pin; thus, set '0' to **GPIOx_ODR** bit. Default '0'



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Figure 10: GPIOx port PIN Set/Reset Register

5.1.2 Data Structure associate to the GPIOx port

GPIOx Init Data Structure. You must use this data structure to initialize a GPIOx port. Put these in GPIO.h (Given as source – Download from classroom)

```
typedef struct
{
        uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
        This parameter can be any value of @ref GPIO_pins_define */

        uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
        This parameter can be a value of @ref GPIO_mode_define */

        uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
        This parameter can be a value of @ref GPIO_pull_define */

        uint32_t Speed;     /*!< Specifies the speed for the selected pins.
        This parameter can be a value of @ref GPIO_speed_define */

        uint32_t Alternate;  /*!< Peripheral to be connected to the selected pins.
        This parameter can be a value of @ref GPIO_Alternate_function_selection */
}GPIO_InitTypeDef;
```

Also, use the following definition for the pins of GPIOx port.

```
#define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
#define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
#define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
#define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
#define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
#define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
#define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
#define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
#define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
#define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
#define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
#define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
#define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
#define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
#define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
#define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
#define GPIO_PIN_All               ((uint16_t)0xFFFF)  /* All pins selected */
#define GPIO_PIN_MASK              0x0000FFFFU /* PIN mask for assert test */
```

The definition of GPIOx Pin modes and output types are:

```
#define  GPIO_MODE_INPUT                        0x00000000U   /*!< Input Floating Mode                    */
#define  GPIO_MODE_OUTPUT_PP                    0x00000001U   /*!< Output Push Pull Mode                  */
#define  GPIO_MODE_OUTPUT_OD                    0x00000011U   /*!< Output Open Drain Mode                 */
#define  GPIO_MODE_AF_PP                        0x00000002U   /*!< Alternate Function Push Pull Mode      */
#define  GPIO_MODE_AF_OD                        0x00000012U   /*!< Alternate Function Open Drain Mode     */
#define  GPIO_MODE_ANALOG                       0x00000003U   /*!< Analog Mode  */
```

The speed to access the GPIOx PINs are defined as below:

```
#define  GPIO_SPEED_FREQ_LOW         0x00000000U  /*!< IO works at 2 MHz, See datasheet */
#define  GPIO_SPEED_FREQ_MEDIUM      0x00000001U  /*!< range 12,5 MHz to 50 MHz, See datasheet*/
#define  GPIO_SPEED_FREQ_HIGH        0x00000002U  /*!< range 25 MHz to 100 MHz, See datasheet*/
#define  GPIO_SPEED_FREQ_VERY_HIGH   0x00000003U  /*!< range 50 MHz to 200 MHz, See datasheet*/
```

Enumerator for pin state defined as below:

```
typedef enum
{
        GPIO_PIN_RESET = 0U,
        GPIO_PIN_SET
} GPIO_PinState;
```

Note: Please visit the course website to get detail of the files.

6 Given Code

We provide you GPIO.h, GPIO.c, CLOCK.h, CLOCK.c, SYSINIT.h, SYSINIT.c, USART.h, USART.c files for convenience. These files have necessary functions (some incomplete) you must use in your program—some of the functions you need to implement. Open a project in KEIL and import these files. Before any other function call, your main program must execute the initialized functions for the clock, sysinit, and GPIO configuration. The CLOCK.h and CLOCK.c contain the configuration for the system clock, and sysinit has the ms_delay() function. Moreover, the GPIO has an unimplemented function for input and output configuration. Please visit the course website (google classroom) to download the files. USART.c and USART.h contains UART_SendChar(), UART_SendString(), UART_GetChar(), and UART_GetString () functions to send the output and get input to and from a computer terminal program. These two functions use USART.h and USART.c; therefore you need to configure USART.

6.1 Lab Setup

Use the breadboard to set up the crossroad and connect those LEDs to the MCU using jumper wires. Place the LEDs to look like a crossroad with traffic lights and vehicles.

6.2 Must implement

You must implement the following functions along with other function. It is important that you understand them properly.

- void GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_pin,GPIO_PinState PinState); – This function set or reset a GPIO PIN. See GPIO_Pinstate enumerator. Use GPIOx-¿BSSR register to set or reset the PIN.
- void GPIO_Init(GPIO_TypeDef*, GPIO_InitTypeDef *); – configuration function to configure a GPIOx port and PIN for input, output. In this function, you will set the bits of the required registers. However, you can only alter the existing configuration of the register if you want different. The best way to configure it is as follows (an example – Pullup or Pulldown register):

```
/* Activate the Pull-up or Pull down resistor for the current IO */
temp = GPIOx->PUPDR;        // keep previous configuration to temp
temp &= ~(GPIO_PUPDR_PUPDR0 << (position * 2U)); // update temp
temp |= ((GPIO_Init->Pull) << (position * 2U)); //update temp
GPIOx->PUPDR = temp; //assign register values
```

We will provide you update for more clarification. You can post your questions and queries to google classroom. However, we encourage your teamwork and discussion with your classmates. Do not copy others' code.

6.2.1 Debugging

You can use Putty, Hercules, or Teraterm for debugging and output to the screen. Use the KEIL debugger for your convenience. It is excellent and helps you to see the inside. KEIL using GDB.

7 Submission

You must demonstrate your solution, submit the code, and report (Latex source files and code) to the lab course website. The submission date is two weeks from January 23, 2023. You must demonstrate the solution in front of us in the next LAB class.

Alert: You can discuss with your classmates, however, do not copy code or report from others.