LED project by STM32cubeIDE

6.4 LEDs

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

- Slow blinking Red/Off: at power-on before USB initialization
- Fast blinking Red/Off: after the first correct communication between the PC and ST-LINK/V2-1 (enumeration)
- Red LED On: when the initialization between the PC and ST-LINK/V2-1 is complete
- Green LED On: after a successful target communication initialization
- Blinking Red/Green: during communication with target

User LD2: the green LED is a user LED connected to Arduino signal D13 corresponding to STM32 I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to *Table 11* to *Table 23* when:

- the I/O is HIGH value, the LED is on
- the I/O is LOW, the LED is off

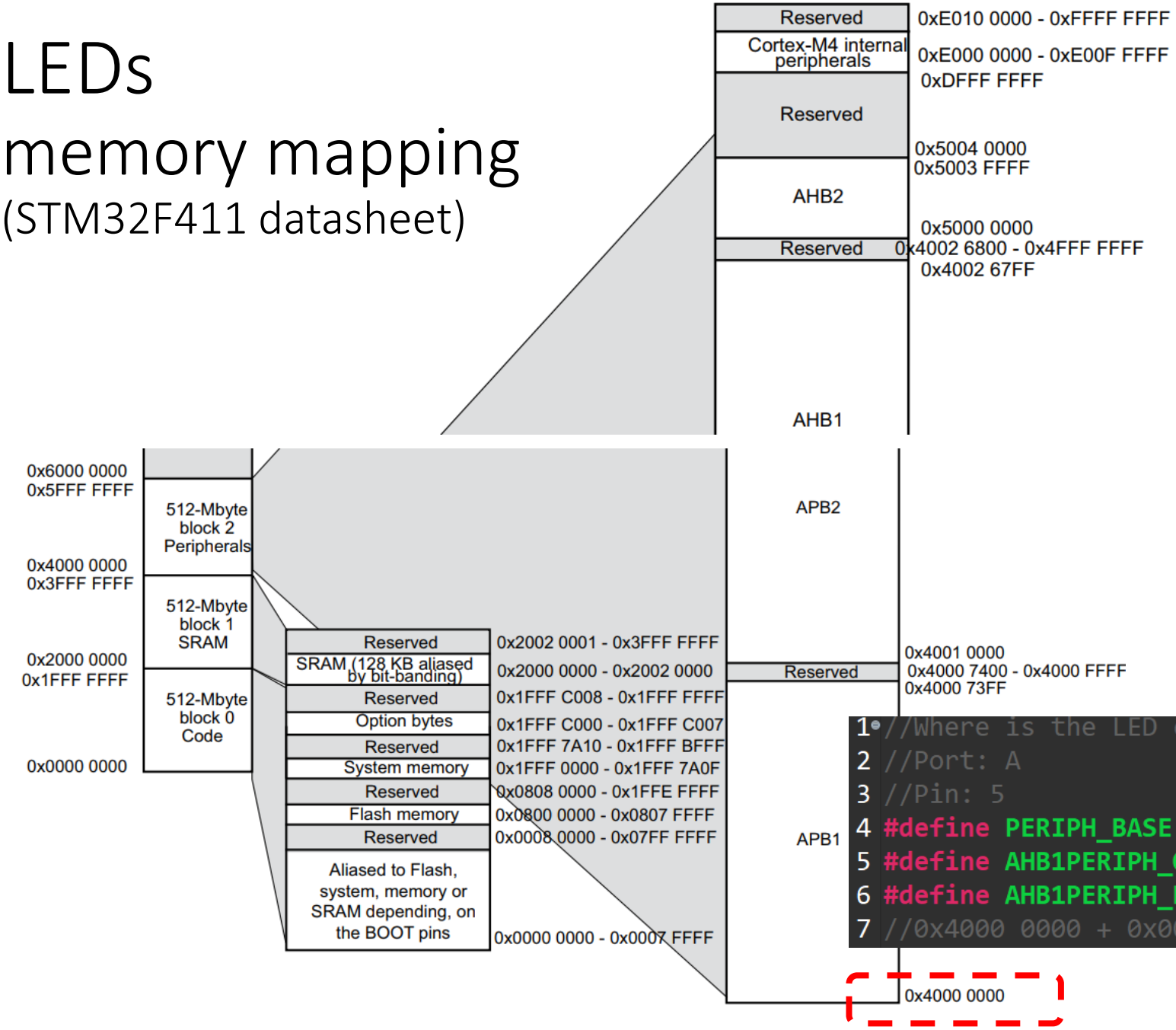
The user can program USER LD2!

LD3 PWR: the red LED indicates that the STM32 part is powered and +5V power is available.

When the Nucleo board is plugged in USB, the LD3(RED LED) comes on

LEDs memory mapping

(STM32F411 datasheet)



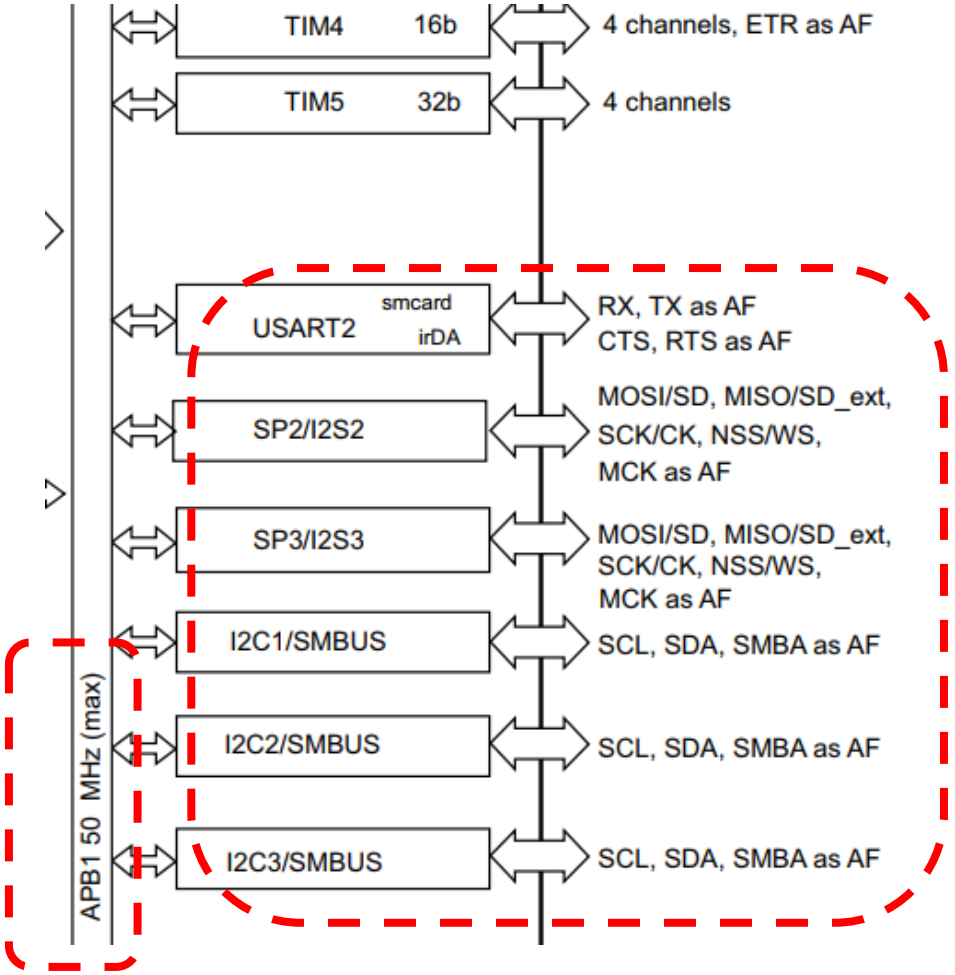

```
1 //Where is the LED connected?  You can find in Nucleo board reference
2 //Port: A
3 //Pin: 5
4 #define PERIPH_BASE           (0x40000000UL)   //UL: unsigned long (start
5 #define AHB1PERIPH_OFFSET     (0x00020000UL)
6 #define AHB1PERIPH_BASE       (PERIPH_BASE + AHB1PERIPH_OFFSET)
7 //0x4000 0000 + 0x0002 0000 = 0x40020000 (GPIOA address: 0x40020000)
```

LEDs bus

The peripherals need buses

- Every part of the microcontroller, every peripheral, every module needs access to the clock, needs to be clocked!
- The **buses carry the clock to the various part of the microcontroller**!

NOTE:
APB: Advanced Peripheral Bus (50MHz)
AHPB: Advanced High Performance Bus (100MHz)



Eg. APB1 bus provides access to USART, SP2, SP3 …

GPIOA connects to AHB1

It means a clock is supplied through AHB1

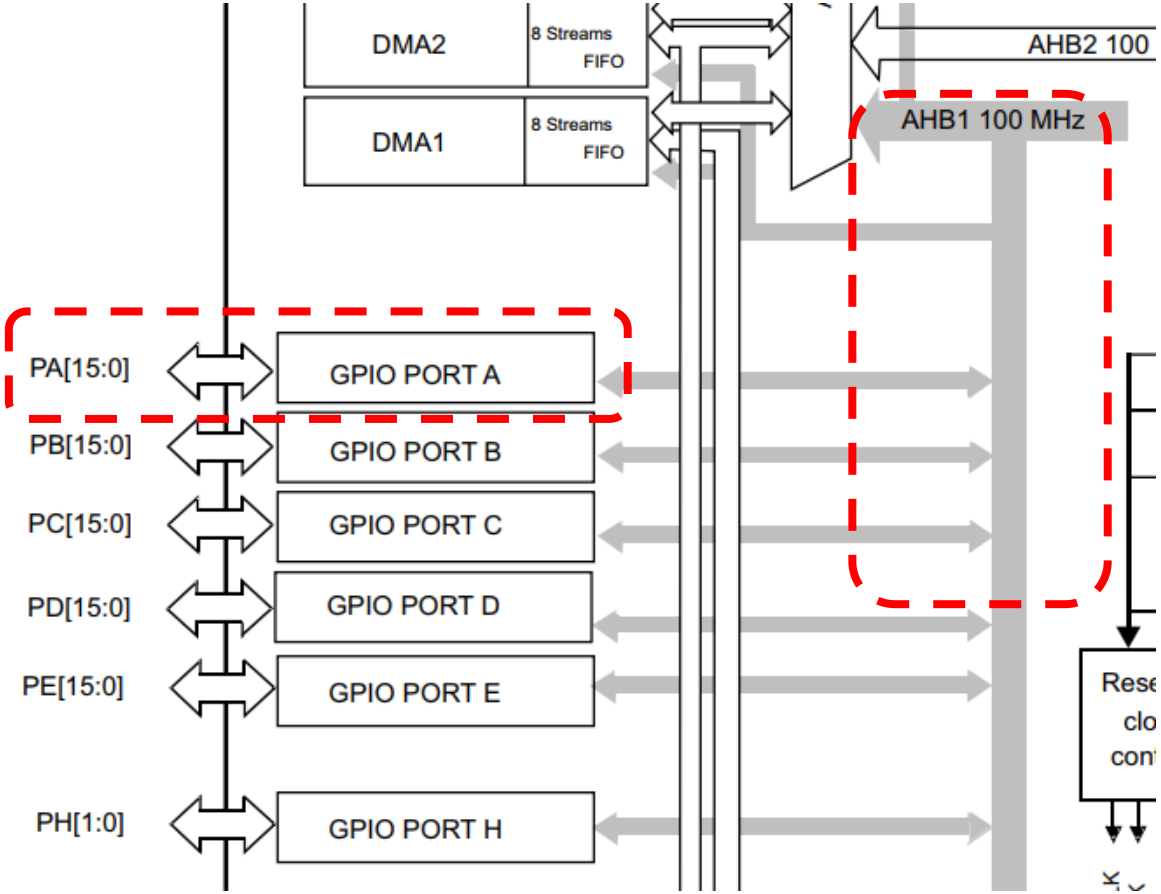
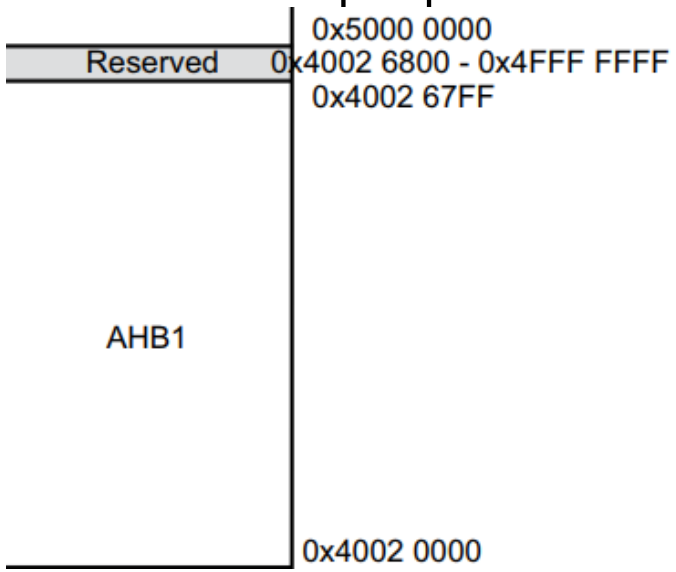


Table 10. STM32F411xC/xE register boundary addresses

Bus	Boundary address	Peripheral
	0xE010 0000 - 0xFFFF FFFF	Reserved
Cortex®-M4	0xE000 0000 - 0xE00F FFFF	Cortex-M4 internal peripherals
	0x5004 0000 - 0xDFFF FFFF	Reserved
AHB2	0x5000 0000 - 0x5003 FFFF	USB OTG FS
AHB1	0x4002 6800 - 0x4FFF FFFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 4FFF	Reserved
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2000 - 0x4002 2FFF	Reserved
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1400 - 0x4002 1BFF	Reserved
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA

You can think each peripheral as a house
Each room inside the house has an address
Our peripherals have addresses, **but there are a number of registers inside our peripherals**.
These registers have their own addresses!
These registers are just an offset from the address of the peripheral itself.



RCC module is in charge of enabling each bus to transport a clock

RCC systems: reset clock control!

Also connected to AHB1 bus!

```
#define RCC_OFFSET  (0x3800UL)
#define RCC_BASE    (AHB1PERIPH_BASE + RCC_OFFSET)
```

	0x4002 6800 - 0x4FFF FFFF	Reserved
	0x4002 6400 - 0x4002 67FF	DMA2
	0x4002 6000 - 0x4002 63FF	DMA1
	0x4002 5000 - 0x4002 4FFF	Reserved
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0x4002 3400 - 0x4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
AHB1	0x4002 2000 - 0x4002 2FFF	Reserved
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1400 - 0x4002 1BFF	Reserved
	0x4002 1000 - 0x4002 13FF	GPIOE
	0x4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA

```
#define GPIOA_OFFSET
(0x0000UL)
#define GPIOA_BASE
(AHB1PERIPH_BASE + GPIOA_OFFSET)
```

```
1  //Where is the LED connected?  You can find in Nucleo board reference manual
2  //Port: A
3  //Pin: 5
4  #define PERIPH_BASE          (0x40000000UL)   //UL: unsigned long (starting point)
5  #define AHB1PERIPH_OFFSET    (0x00020000UL) // =0x0200000
6  #define AHB1PERIPH_BASE      (PERIPH_BASE + AHB1PERIPH_OFFSET)
7  //0x4000 0000 + 0x0002 0000 = 0x40020000 (GPIOA address: 0x40020000)
8  #define GPIOA_OFFSET         (0x0000UL)  //= 0x 0000 0000
9  #define GPIOA_BASE           (AHB1PERIPH_BASE + GPIOA_OFFSET)
10
11 #define RCC_OFFSET           (0x3800UL)
12 #define RCC_BASE             (AHB1PERIPH_BASE + RCC_OFFSET)
```

```
1  //Where is the LED connected?  You can find in Nucleo board reference manual
2  //Port: A
3  //Pin: 5
4  #define PERIPH_BASE       (0x40000000UL)   //UL: unsigned long (starting point)
5  #define AHB1PERIPH_OFFSET (0x00020000UL) // =0x0200000
6  #define AHB1PERIPH_BASE   (PERIPH_BASE + AHB1PERIPH_OFFSET)
7  //0x4000 0000 + 0x0002 0000 = 0x40020000 (GPIOA address: 0x40020000)
8  #define GPIOA_OFFSET      (0x0000UL)  //= 0x 0000 0000
9  #define GPIOA_BASE        (AHB1PERIPH_BASE + GPIOA_OFFSET)
10
11 #define RCC_OFFSET        (0x3800UL)
12 #define RCC_BASE          (AHB1PERIPH_BASE + RCC_OFFSET)
```

Now, we would have to find the rooms inside these houses that we need to interact with the rooms!
Here are the registers inside this peripheral that we need to interact with.
RCC has number of registers. RCC has a register for enabling clock for APB1 another register for enabling for APB2.
So we can think of all these separate registers as rooms!

We know we are interested in AHB1. => we would get to the register for enabling clock for AHB1

GPIOA has a number of registers => There is a register for configuring a pull-up/pull-down resistor to a pin
There's another register for configuring the speed of a pin
A register for setting a pin as an input pin / as an output pin / as an analog pin
A register for reading input data
A register for storing data

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOHEN	Reserved		GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw					rw			rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bit 1 **GPIOBEN:** IO port B clock enable

Set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled

Bit 0 **GPIOAEN:** IO port A clock enable

Set and cleared by software.

0: IO port A clock disabled

1: IO port A clock enabled

```
#define GPIOAEN (1U<<0) // 0b0000 0000 0000 0000 0000 0000 0000 0001
```

```
2  //Port: A
3  //Pin: 5
4  #define PERIPH_BASE        (0x40000000UL)   //UL: unsigned lo
5  #define AHB1PERIPH_OFFSET  (0x00020000UL) // =0x0200000
6  #define AHB1PERIPH_BASE    (PERIPH_BASE + AHB1PERIPH_OFFSET)
7  //0x4000 0000 + 0x0002 0000 = 0x40020000 (GPIOA address: 0x4
8  #define GPIOA_OFFSET       (0x0000UL)  //= 0x 0000 0000
9  #define GPIOA_BASE         (AHB1PERIPH_BASE + GPIOA_OFFSET)
10
11 #define RCC_OFFSET         (0x3800UL)
12 #define RCC_BASE           (AHB1PERIPH_BASE + RCC_OFFSET)
13
14 #define AHB1EN_R_OFFSET    (0x30UL)
15 #define RCC_AHB1EN_R       (RCC_BASE + AHB1EN_R_OFFSET)
16 #define GPIOAEN            (1U<<0)
```

Next, let's deal with our GPIOA so we have GPIOA base address

We need to find the registers in the paper that we need to work with

- **Direction register (MODE reg.)**: set the pin to either an input / output
- **Data register**: store the data, essentially
  - If it's an input then the data received is going to be kept in the data
  - If it's an output then the data you want to output, you pass it through the data reg.

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

```
#define MODE_R_OFFSET (0x00UL)
#define GPIOA_MODE_R (GPIOA_BASE + MODE_R_OFFSET)
```

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

Because we are dealing with PA5, we are interested in MODER5 (bit no. 10 and 11)
Eg. Bit 11 = 0, Bit 10 = 1 => OUTPUT MODE

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

```
/* 2. Set PA5 as output PIN */
GPIOA_MODE_R |= (1U<<10);
GPIOA_MODE_R &=~(1U<<11);
```

Alternate function mode example: UART TX/Rx

After we've configured this as **OUTPUT** mode, we would need to write **ON** and **OFF** to a particular register to turn the LED ON/OFF

This is known as the **output data register** because we are dealing with outputs.

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..E and H)

Address offset: 0x14

Reset value: 0x0000 0000

```
#define OD_R_OFFSET (0x14UL)
#define GPIOA_OD_R  (GPIOA_BASE + OD_R_OFFSET)
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
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

To set PIN5 ON, we would have to write one at bit No. 5
To set it OFF, we would have to write it "0"

```
//Port: A                 //Pin: 5
#define PERIPH_BASE         (0x40000000UL)   //UL: unsigned long (starting point)
#define AHB1PERIPH_OFFSET (0x00020000UL) // =0x0200000
#define AHB1PERIPH_BASE   (PERIPH_BASE + AHB1PERIPH_OFFSET)
//0x4000 0000 + 0x0002 0000 = 0x40020000 (GPIOA address: 0x40020000)
#define GPIOA_OFFSET        (0x0000UL)  //= 0x 0000 0000
#define GPIOA_BASE          (AHB1PERIPH_BASE + GPIOA_OFFSET)

#define RCC_OFFSET          (0x3800UL)
#define RCC_BASE            (AHB1PERIPH_BASE + RCC_OFFSET)

#define AHB1EN_R_OFFSET     (0x30UL)
#define RCC_AHB1EN_R        (*(volatile unsigned int *)(RCC_BASE + AHB1EN_R_OFFSET))

#define MODE_R_OFFSET       (0x00UL)
#define GPIOA_MODE_R        (*(volatile unsigned int *)(GPIOA_BASE + MODE_R_OFFSET))

#define OD_R_OFFSET         (0x14UL)
#define GPIOA_OD_R          (*(volatile unsigned int *)(GPIOA_BASE + OD_R_OFFSET))

#define GPIOAEN             (1U<<0) // 0b0000 0000 0000 0000 0000 0000 0000 0001
#define PIN5                (1U<<5)
#define LED_PIN             (PIN5)
```

```
int main()
{
/* 1. Enable clock access to GPIOA */
RCC_AHB1EN_R  |= GPIOAEN;
/* 2. Set PA5 as output PIN */
GPIOA_MODE_R |= (1U<<10);
GPIOA_MODE_R &= ~(1U<<11);

while(1){
/* 3. Set PA5 HIGH*/
//GPIOA_OD_R  |= LED_PIN;
/* 4. experiment 2: toggle PA5 */
GPIOA_OD_R  ^= LED_PIN;
for (int i=0; i<150000;i++);
}
}
```