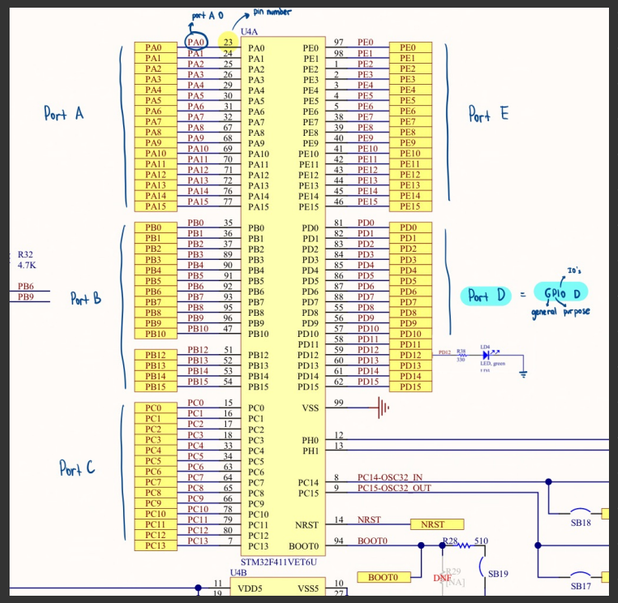
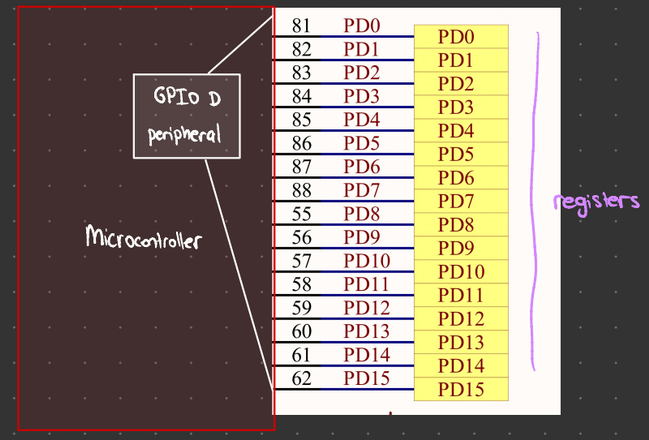
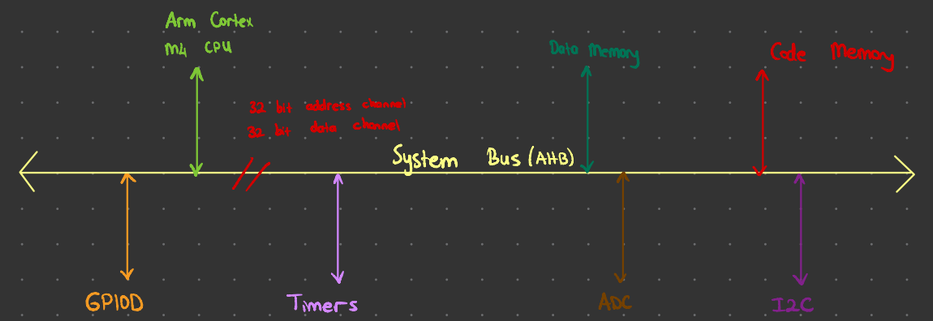




GPIOD is a peripheral which is used to control pins of the Port D. With this GPIO D, you can control what data you can transmit over the pins, or you can write data to the port. You can access the registers of this peripheral using memory adresses, hence you can also say that this peripheral registers are memory mapped. Every register has own memory adress. By using this address, you can access these register.



System bus which connects the processor with the peripheral and with memory. This bus is a central bus which connects peripherals, processors, and the memories. and also this bus is also called as system bus, which is based on AHB specification. AHB => Advanced High Performance

This bus comprises of two channel.

When the processor puts these memory locations on the adress bus, the address bus will be targetting to the code memory of the microcontroller.

This arrangement is called as Memory map of the processor. This is fixed by the ARM Cortex Mx architecture. When you try to code something you should follow this.

from ARM Cortex Mx technical reference manual

About peripherals registers

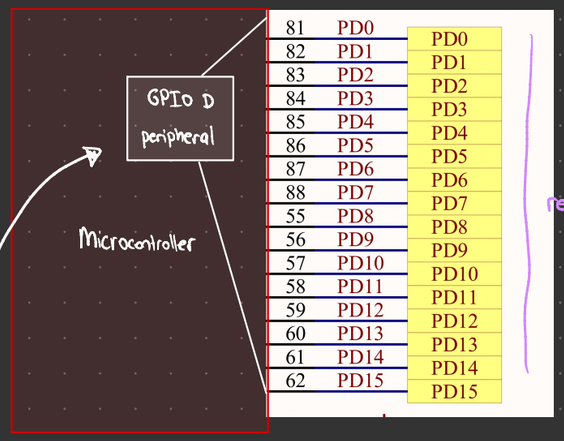
→ All peripherals registers in STM 32 microcontroller are 32 bits wide

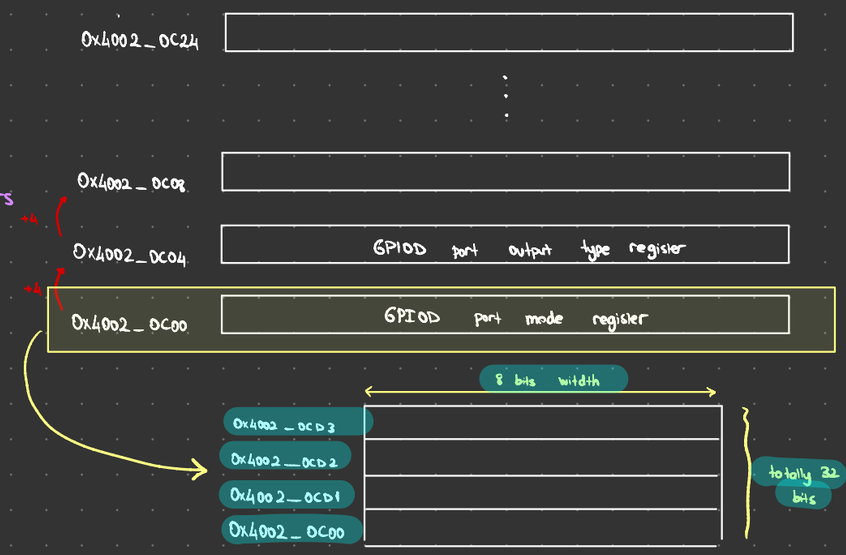
→ Different peripherals have different number of peripheral registers. It is depend on how complex is." ↳ I think this means every Peripheral for our port. It uses some point of different peripheral.

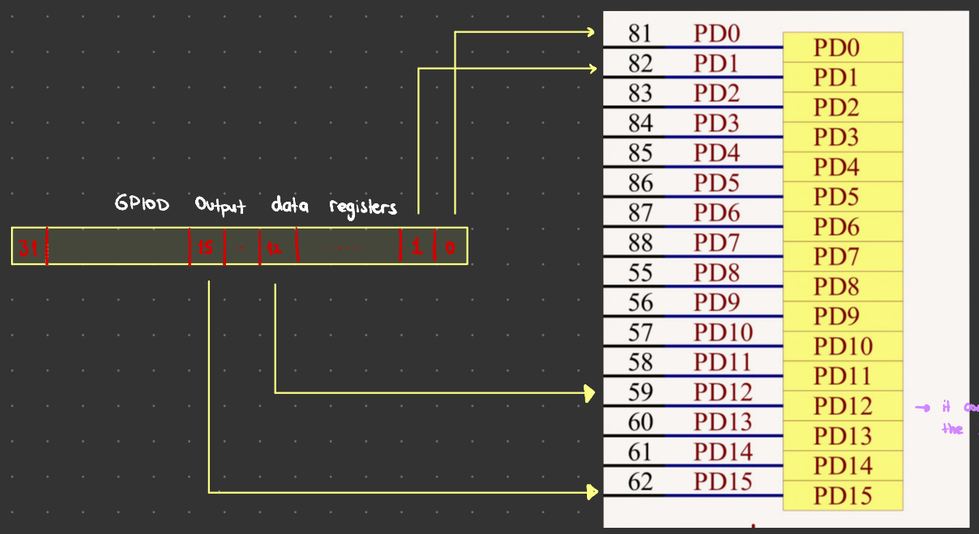
Video 116:

1) GPIOD port mode register : uses for control the mode of IO pin, whether you want it output or input.
2) GPIOD port output type register
3) GPIOD port output speed register
4) GPIOD port pull-up/pull-down register
5) GPIOD port input data register : if you want to read data from IO pins, you use that.
6) GPIOD port output data register : write data on IO pins
7) GPIOD port bit set/reset register
8) GPIOD port configuration lock register
9) GPIOD alternate function low register
10) GPIOD alternate function high register

For example, if you want to turn on the LED, then you have to write 1 into the output data registers.







very important!

- (8) → clearing
- (|) → setting

0x4002 3800 (beginning) - 0x4002 3BFF (end)	RCC

RCC memory map address

0x4002 0C00 - 0x4002 0FFF	GPIOD

GPIOD memory map location

→ memory map and offsets are in reference manual.

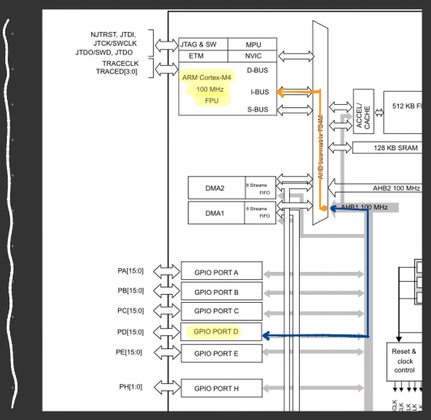
Procedure to turn on the LED

1) Identify the GPIO port (a peripheral) used to connect the LED
2) Identify the GPIO pin where the LED is connected
3) Activate the GPIOD peripheral (Enable the clock)
   → until you enable the clocks for a peripheral, the peripheral is dead and it neither functions nor it takes any configuration values set by you.
4) Configure the GPIO pin mode as output
5) Write to the GPIO pin

How to Enable the peripheral clock?

There is an engine called RCC (reset and clock control). For most of the microcontroller, peripherals are dead and we need to active them. First thing, you would do is activating the peripheral's clock. For activating peripheral's clock you need to firstly look for reference manual and see which bus do you use for your peripherals.

For example, we are using GPIOD port and it connected to the AHB1 bus



"some form of distance measured from some given position"

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30 → You need to add this value to RCC location adress, then you will have the access for controlling RCC.

Reset value: 0x0000 0000

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

31–23	22	21	20–8	7	6	5	4	3	2	1	0
Reserved	DMA2EN	DMA1EN	Reserved	GPIOH EN	Reserved		GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
	rw	rw		rw			rw	rw	rw	rw	rw

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

for enable the clock, you need to make 3rd bit as 1.

Bit 3 **GPIODEN:** IO port D clock enable
Set and cleared by software.
0: IO port D clock disabled
1: IO port D clock enabled

these are memory maps.

Address of the Clock control register (AHB1ENR)
0x4002 3800 + 0x30 → 0x4002 3830

Address of the GPIOD mode register (used to control mode)
0x4002 0C00 + 0x00 → 0x4002 0C00

Address of the GPIOD output data register (used to write)
0x4002 0C00 + 0x14 → 0x4002 0C14

offset

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

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
pin-15	pin-14		pin-12				

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
					pin-2	pin-1	pin-0

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 → "we will use this one for" enable the pin as output
10: Alternate function mode
11: Analog mode

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

Address offset: 0x14

Reset value: 0x0000 0000

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	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
	port 15													...	port 1	port 0

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

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..E and H).

Bitwise Right Left Shift Operator

Set 4th bit of the given data

Data: 0x08
Data: 0x08 | 0x10

mask v. : for this value we need to create one variable

7	6	5	4	3	2	1	0	
0	0	0	0	1	0	0	0	[Data]
0	0	0	1	0	0	0	0	[mask]
0	0	0	1	1	0	0	0	[output]

instead of creating one mask variable we can use bitwise

Set 4th bit of the given data

Data: 0x08
Data: Data | (1 << 4)
mask

0	0	0	0	0	0	0	1	} [1]
0	0	0	1	0	0	0	0	} (mask)

Data = 0x18

Little Note!
"There is no such as 18 in binary"
1 → 0001
8 → 1000
→ 00011000

* Note:
You can use this one as cleaner (&) in every situation.

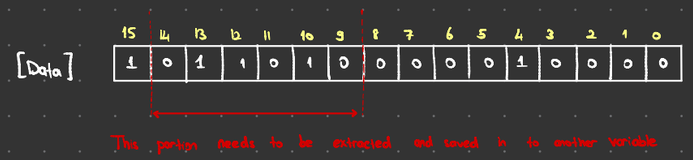
the code we can write for our LED:

```
int main(void)
{
    uint32_t *pClkctrlreg = (uint32_t*)0x40023830;
    uint32_t *pPortDModeReg = (uint32_t*)0x40020C00;
    uint32_t *pPortDOutReg = (uint32_t*)0x40020C14;

    //1. enable the clock for GPOID peripheral in the AHB1ENR
    /*
     * it is same usage for below code.
    uint32_t temp = *pClkCtrlReg;  //read operation
    temp = temp | 0x08; //modify
    *pClkctrlreg = temp; //write back
    */
    *pClkctrlreg |= (1 << 3); // shorcut for this *pClkctrlreg = *pClkctrlreg | 0x08;

    //2. configure the mode of the IO pin as output.
    //a. clear the 26th and 25th bit positions (CLEAR)
    *pPortDModeReg &= ~(3 << 26);
    //b. make the 26th bit position as 1 (SET)
    *pPortDModeReg |= (1 << 26);

    //3. SET 13th bit of the output data register to make i/o pin-12 HIGH
    *pPortDOutReg |= (1 << 13);

    while(1);
}
```

Alt + 126 "negation symbol"

Bit Extraction

- Extract bit position from 9^{th} to 4^{th} bit [14:9] in a given data and save it in to another variable.

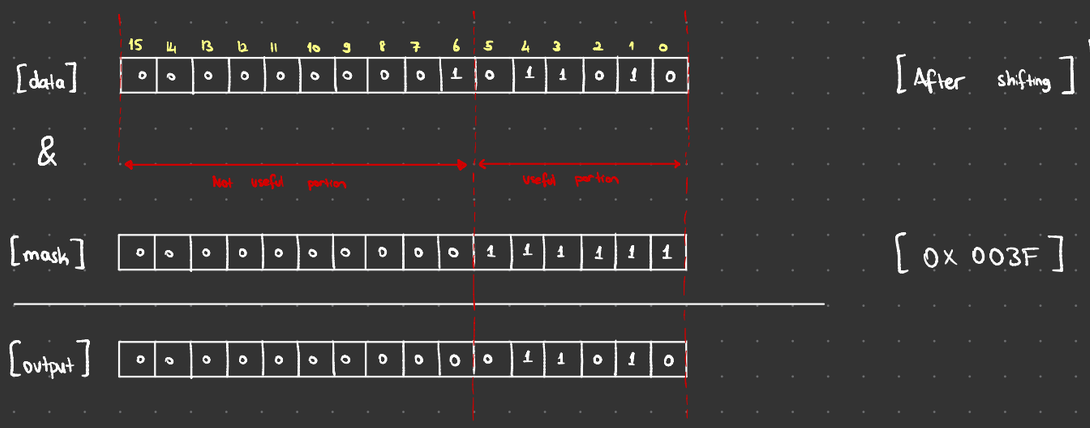


0xB410

[original data]

→ Shift the identified portion to right hand side until it touches the least significant bit (0^{th} bit)

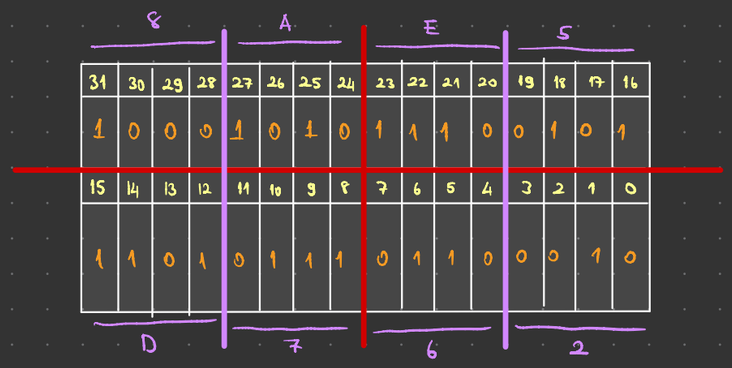
→ Mask the value to extract only 6 bits [5:0] and then save it in to another variable



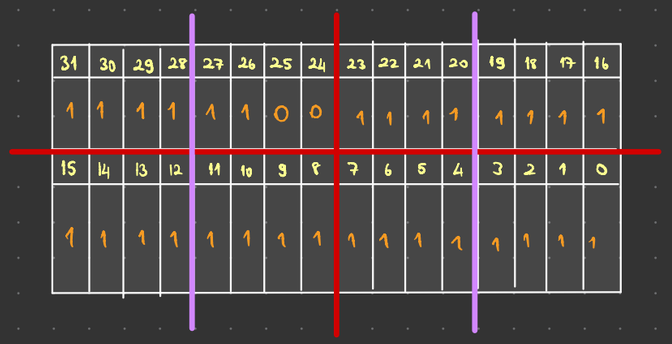
[After shifting]

[0x003F]

```
uint16_t   Data   = 0xB410;
uint8_t    output ;

output = (uint8_t) ( (Data >> 9) & (0x003F))
```



⟹ 0x8AE5D762



⟹ 0xFCFFFFFF

0x8AE5D762 & = 0xFCFFFFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0x 09 (11)

1111 00