


Instructions for Blink Led in Bare metal

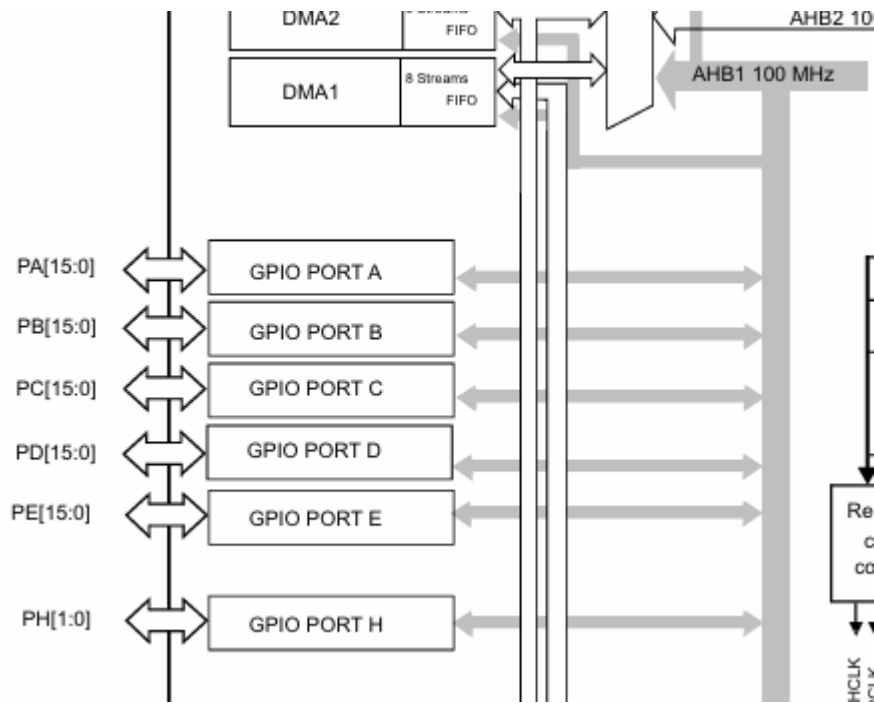
Owner	 Angel David Sanchez
Tags	Embedded software

Step 1. Find the register of RCC (Reset Clock and Control) and its offset.

- The first step is to find the RCC register and its base address, to do this we need to dive into the datasheet and reference manual, where we need to find the **memory map** and the **block diagram**, in my case I found something like this:

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	Section 22.16.6: OTG_FS register map on page 754
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	Section 9.5.11: DMA register map on page 197 Section 3.8: Flash interface registers on page 59 Section 6.3.22: RCC register map on page 136 Section 4.4.4: CRC register map on page 69 Section 8.4.11: GPIO register map on page 163
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		
0x4002 3800 - 0x4002 3BFF	RCC		
0x4002 3000 - 0x4002 33FF	CRC		
0x4002 1C00 - 0x4002 1FFF	GPIOH		
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		

- Here we can see that the base address for RCC is 0x40023800, after finding this, we need to find the necessary offset for the register to work like we want, and to find such offset, we need to find the system bus to which the Port of my Led is connected. In my case is like this:



- Now, my led is connected on my Port C pin 13, that means that the bus that I need is the AHB1, the next thing to do is find the RCC of that system bus and activated using the offset. In my case I got something like this:

RM0383

Reset and clock control (RCC) for STM32F411xC/E

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
			rw					rw				rw	rw	rw	rw

Now, knowing the base address of the RCC and the necessary offset for my system bus to work I need to add them together to get one single address.

—> **0x40020800 + 0x30 = 0x40020830** ← and with that I have my first address

Step 2. Find Base address of PortC and Mode Register.

To find our second address memory, we need to have located the Port and pin of our led, in my case is Port C, pin 13, knowing that I have to find the base address of Port C:

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	Section 22.16.6: OTG_FS register map on page 754
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	Section 9.5.11: DMA register map on page 197
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		Section 3.8: Flash interface registers on page 59
0x4002 3800 - 0x4002 3BFF	RCC		Section 6.3.22: RCC register map on page 136
0x4002 3000 - 0x4002 33FF	CRC		Section 4.4.4: CRC register map on page 69
0x4002 1C00 - 0x4002 1FFF	GPIOH		Section 8.4.11: GPIO register map on page 163
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

- In my case the base address of my PortC is 0x40020800
- Now the next step is to find the GPIO Port Mode, this will allow us to set the port in either input or ourput.

8.4 GPIO registers

This section gives a detailed description of the GPIO registers.

For a summary of register bits, register address offsets and reset values, refer to [Table 26](#).

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

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
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

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

- Since we want to turn on a led, we want to send a signal, we need to put the port in Output and for this we need to put the configuration of “01” which means that the port is in “**general purpose output mode**” and with that set will be able to send a signal.
- Also we needed to add the offset of the modern port which in my case is 0x00
- —> **0x40020800 + 0x00 = 0x40020800** and with that we get our **second address memory**

Step 3. Base Address of GPIOC output data register

- Finally to get the final address memory, where when we either set a mode register in input or output we get a different offset, since we are using it as output we need to find that offset, specially the one that send data out the register, we use the same base address of the PortC and we just add this final offset.

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

Address offset: 0x14

Reset value: 0x0000 0000

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

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).

—> $0x40020800 + 0x14 = 0x40020814$ ← - Final address memory.

Step 4. Coding using shifting operation

Now, we need to code and activate the led of our board.

```
int main(void)
{
    uint32_t *pClkCReg = (uint32_t*)0x40023830;
    uint32_t *pPortCModReg = (uint32_t*)0x40020800;
    uint32_t *pPortCOutReg = (uint32_t*)0x40020814;

    *pClkCReg = 0;
    *pClkCReg |= (1 << 2);
    *pPortCModReg = 0;
    *pPortCModReg |= (1 << 26);
    *pPortCOutReg = 0;
    *pPortCOutReg |= (1 << 13);
    /* Loop forever */
    for(;;);
}
```

- The first thing we do is declare and initialize our pointers variables, we declare them as uint32_t* (Unsigned integer) because this way we have more control over the variable size and for more portability and clarity.
- Next, we use pointers (*variable) and set it for each of our address (RCC, Mode Port, Output Port) and use typecasting to convert our hexadecimal memory address to uint32_*.

- The next thing to do is first clear the registers and then set them to the bits that we need high.
- For setting we use “|=” and for clearing we use “&=~”.

Step 4.1 Coding using bit mask

- This is bare metal C programming

```
int main(void)
{
    uint32_t *pClkCReg = (uint32_t*)0x40023830; //Pointer to RCC Register
    uint32_t *pPortCModReg = (uint32_t*)0x40020800; //Pointer to Port C Mode Register
    uint32_t *pPortCOutReg = (uint32_t*)0x40020814; //Pointer to Port C output data Register

    *pClkCReg |= 0x04; //Enabling the clock for GPIO peripherals in the AHB1 Bus

    //Configuration Mode
    *pPortCModReg &= 0xF3FFFFFF; //Clearing the port C
    *pPortCModReg |= 0x04000000; //Setting the port C as output

    *pPortCOutReg |= 0x2000; //Turning on Led

    /* Loop forever */
    for(;;);
}
```

- The code starts the same way, defining the pointer to the correct registers (I've explain how to get the register in the previous steps)
- After doing that we first need to set (enable) the clock for the peripherals (ports GPIO, specifically Port C), and to do that we need to take a look at this:

RM0383

Reset and clock control (RCC) for STM32F411xC/E

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

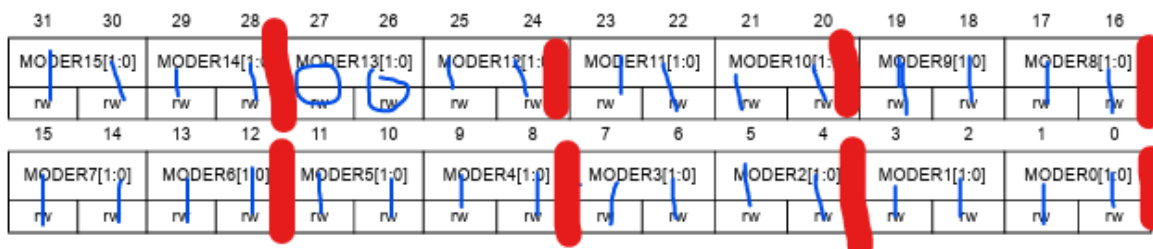
Address offset: 0x30

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				GPIOH EN	Reserved		GPIOEEN	GIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw

- Here we can see that the GPIOC is in the third position (starting from 0) and we need to send it a high signal (1) so the hexadecimal address for that would be **0x04**;
- The next step is to clear the GPIO port mode register, now, to clear it we use (&=) and to get the correct address we need to put a 0 in the moder register of the pin where our led is connected and a 1 in the rest of them:



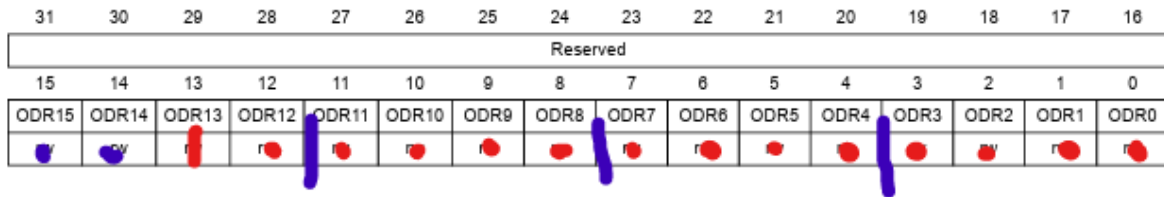
- Bits $2y:2y+1$ **MODERy[1:0]**: Port x configuration bits ($y = 0..15$)
 These bits are written by software to configure the I/O direction mode.
 00: Input (reset state)
 01: General purpose output mode
 10: Alternate function mode
 11: Analog mode

- After doing this we get a ñarge binary number and for a better aesthetic we convert it to hexadecimal and the result would be: **0xF3FFFFFF**
- After clearing the port we need to set it to the correct address (sending a high signal in only the pin that we want to turn on) and to do that we first need to look at the port configuration, since we need an output, the configuration of our port mode register should be “01” (General Purpose Output mode) and do the contrary, the result would be: **0x04000000**
- Finally the last hexadecimal address would be to turn on the led and the specified pin 13, to do this we need to look at the GPIO port output data register.

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

Address offset: 0x14

Reset value: 0x0000 0000



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

Bits 15:0 ODRy: Port output data (y = 0..15)

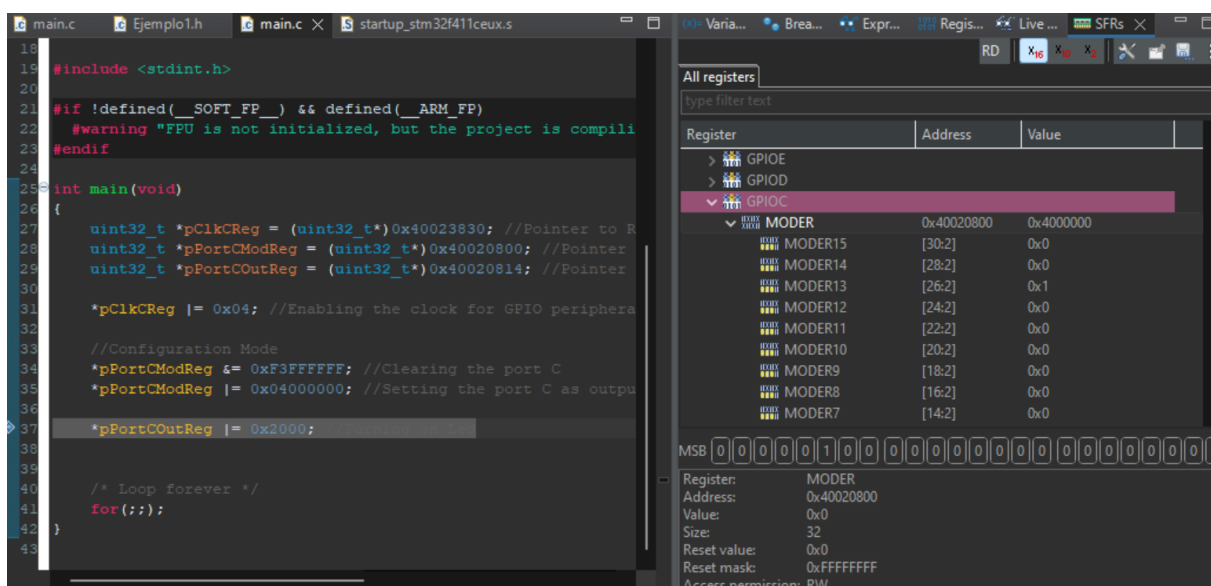
These bits can be read and written by software.

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).

- We need to turn on the led on the pin 13, so we send a high signal to that single bit and the rest we send a low signal, again we would get a large binary number, so we convert it to hexadecimal and the result would be: **0x2000**

Step 5. Results

- One of the ways to prove that our code and circuit is working is to look at the registers and see how they change, in stm32cubeide, we get a feature that help us do this, while running the code we enter to the window of STR and we can see how these change:



- https://youtu.be/wrf_X738vdc

- <https://www.notion.so/Instructions-for-Blink-Led-in-Bare-metal-69803041ca8c4b65bb8d33badcd75353?pvs=4#9c2d6edb05b449f8b85dc8560a92953f>