HOW TO TURN ON A LED

STM32 Course Portfolio

Adrian Felipe Gongora Suarez

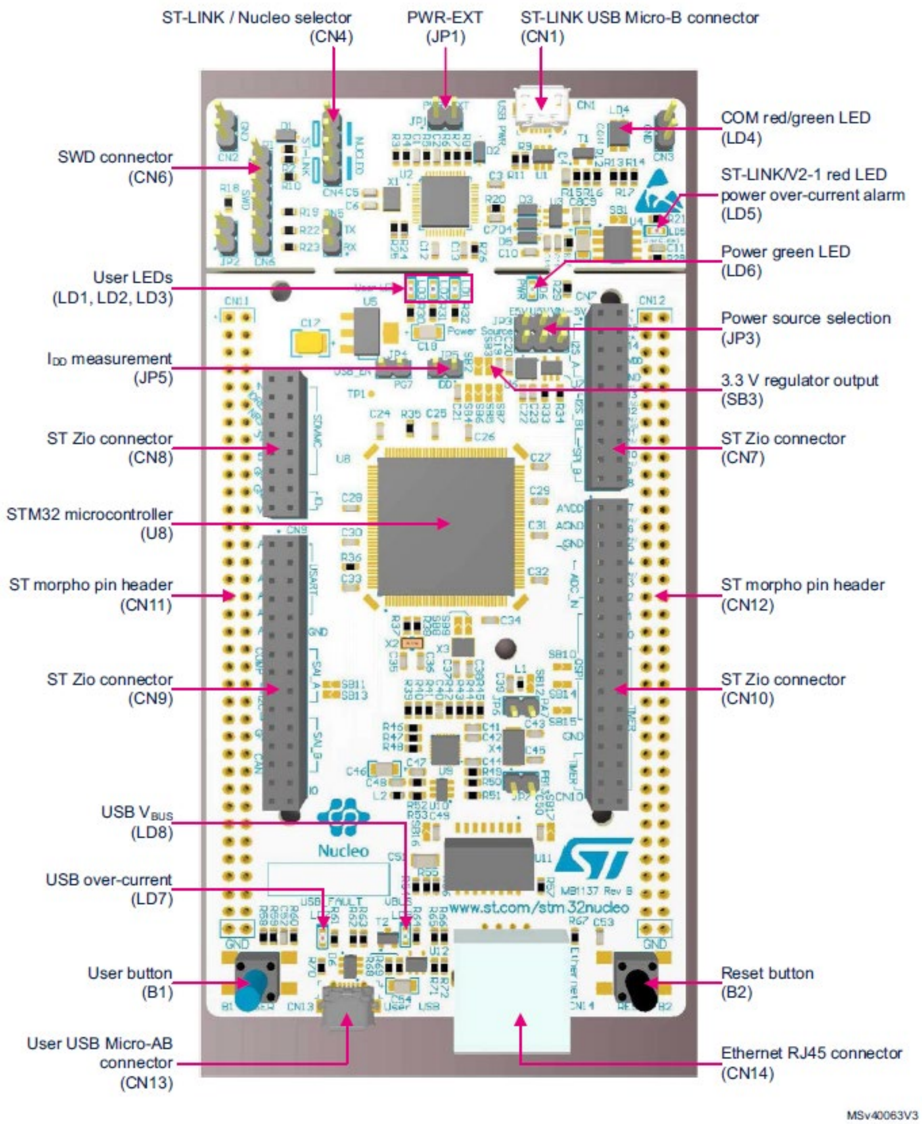
03/01/2023 – 03/02/2023

Index

Coding Guide

1. Identify the LEDs in the Board NUCLEO-STM32F466ZE

The first step to make a blinking LED is to identify the On-Board LED or given LED that you want to use in this case the NUCLEO board used for this guide has 3 ON board LEDs.



We are going to use the LD1, first red LED in the board which in the User manual states being connected to the PB0 pin of the STM32F446.

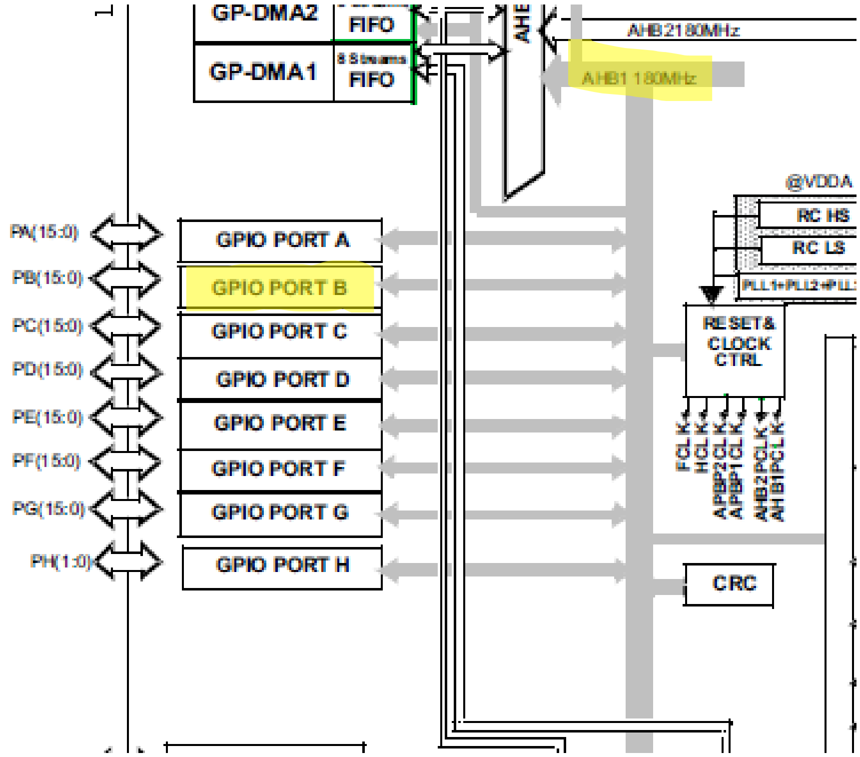
2. Identify the registers needed for the operation of the code.

Now we need 3 registers to make a blinking LED, these are in order; RCC enable register, MODDER for the PORTB and the OUTPUT register for the PORD.

RCC Register (enable the PORTD):

To find all of the registers we need to know navigate a memory map, or table, in the datasheet of the STM32F446 we have the memory table for the different parts of the microcontroller, aswell as the block diagram which tells us where the different parts are connected.

First using the Block diagram, we need to identify the BUS in which the PORTB is located or linked, in this case is the AHB1.



With this information now we can find the RCC register that we need, which is the enable register of the AHB1, using the reference manual, we can pinpoint the register

6.3.10 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
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

RCC_AHB1ENR, is the register, here we can see the PIN 3 or GPIOB is the PIN what we need, here we also see the Address offset, this is important using the memory map and this offset we can find the exact register address.

AHB1	0X4002 5000 - 0X4002 5FFF	Reserved
	0x4002 4000 - 0x4002 4FFF	BKPSRAM
	0x4002 3C00 - 0x4002 3FFF	Flash interface register
	0x4002 3800 - 0x4002 3BFF	RCC
	0X4002 3400 - 0X4002 37FF	Reserved
	0x4002 3000 - 0x4002 33FF	CRC
	0x4002 2C00 - 0x4002 2FFF	Reserved
	0x4002 2800 - 0x4002 2BFF	
	0x4002 2400 - 0x4002 27FF	
	0x4002 2000 - 0x4002 23FF	
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	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

0x4002 3800 is the base memory address in which we add 0x30 and we get:

0x4002 3830 as the memory address of the register. In the program we put this address as a pointer of a pointer with a typecast which makes us pass the pointer an address to mask.

```
int main(void)
{
	uint32_t * pC1kcrtlreg = (uint32_t*)0x40023830;
```

And for the configuration we use a AND passing the number of bit in the hexadecimal.

```
*pC1kcrtlreg |= 0x00000002;
(o 0x08)
```

By this point we have a RCC enabled for the whole AHB1 BUS

MODDER Register (enable the PIN B0 as an OUTPUT):

The same process we do with the MODDER register using the memory map we can view that the base address of the GPIOB is 0x4002 0C00 with the reference manual the Offset of the register MODDER is 0x00

7.4.1 GPIO port mode register (GPIOx_MODER) (x = A..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)

And the PIN for the PIN B0 is actually PIN0, for MODER0. In the code it ends up like this.

```
uint32_t * pPortDModeReg = (uint32_t*)0x40020400; // Enable the port Register
```

Now we need to configure this register, for this we need to either do another AND operator to set the bit 0 what we need to enable to configure as a General-Purpose Output Mode.

```
*pPortDModeReg |= 0x00000001;
```

At this point we have the PIN0 (PB0) as an output.

OUTPUT register (Turn on the LED):

Finally the last register is to enable and turn on the LED this is a OUTPUT register for the desired PORT, using the reference manual we can search and find the register.

General-purpose I/Os (GPIO) **RM0390**

7.4.6 GPIO port output data register (GPIOx_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

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

For this register the Offset is 0x14 and the PIN that we need is the PIN0 which corresponds to PB0. The code looks like this:

```
uint32_t * pPortDOutReg = (uint32_t*)0x40020414; // Register for the OUTPUT Pins
```

And for the assignment for the PIN to turn ON:

```
*pPortDOutReg |= 0x00000001;
```

To turn OFF:

```
*pPortDOutReg &= ~0x00000001;
```

*An OR with a NOT is used to clear a given value***

3. The Blinker & Program

The general program is as follows, first we define the Pointers for the Masks.

```
	uint32_t * pC1kcrtlreg = (uint32_t*)0x40023830;
	uint32_t * pPortDModeReg = (uint32_t*)0x40020400;
	uint32_t * pPortDOutReg = (uint32_t*)0x40020414;
```

Then we configure each register with the pointers:

```
	*pC1kcrtlreg |= 0x00000002;
	*pPortDModeReg |= 0x00000001;
```

Now at this point we have the RCC enabled and the PORTB as an OUTPUT.

Finally in the main LOOP we but the Blinker.

```
	while(1){

		*pPortDOutReg |= 0x00000001;
		delay(100000);

		*pPortDOutReg &= ~0x00000001;
		delay(100000);
	}
```

This is simple blinker using a delay function, here we SET the value of the OUTPUT PORT for the pin PB0 then we stop with a DELAY function which uses a simple FOR loop with a given value (limit) to count until it reaches, using volatile variable which the compiler does not optimize.

```
void delay (int x) // Delay function using volatile int (not optimize Assembly code)
{
  volatile int i,j;
  for (i=0 ; i < x ; i++)
  {
```

```
        j++;
    }
    return;
}
```

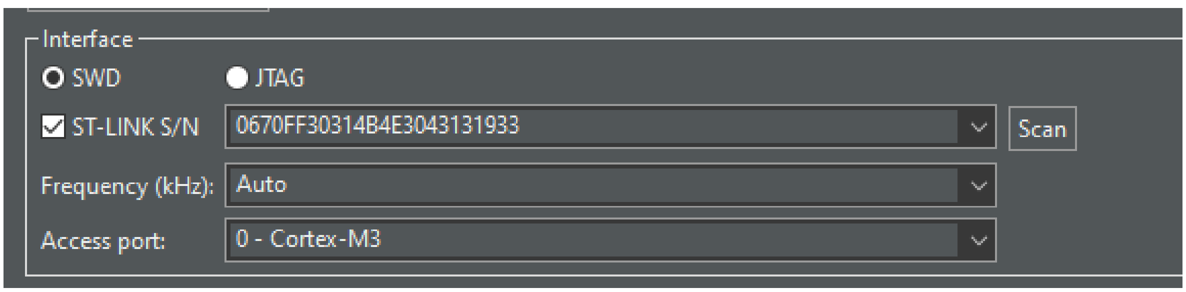
The returning to the lines

```
                *pPortDOutReg &= ~0x00000001;
                delay(100000);
        }
```

We clear the bit to turn the LED off and then delay, and repeat.

4. Program and debug.

Now to program and debug, we need to connect the board with a MicroUSB cable and then set up the debugger with the settings for the ST-LINK.

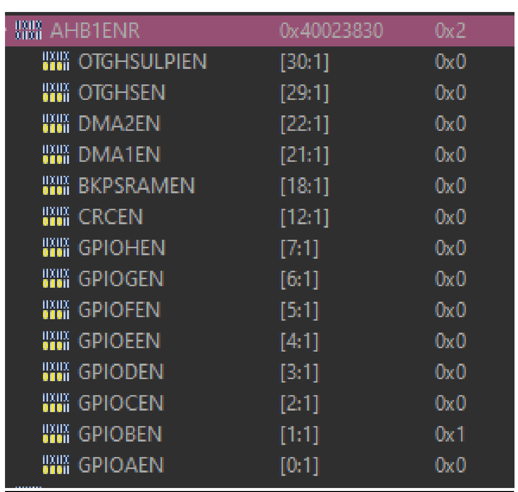


We can debug.

Debugging

Now in this part we check for every line, the effect on the registers.

For the RCC register we can view that the second PIN is in fact ON



AHB1ENR	0x40023830	0x2
OTGHSULPIEN	[30:1]	0x0
OTGHSEN	[29:1]	0x0
DMA2EN	[22:1]	0x0
DMA1EN	[21:1]	0x0
BKPSRAMEN	[18:1]	0x0
CRCEN	[12:1]	0x0
GPIOHEN	[7:1]	0x0
GPIOGEN	[6:1]	0x0
GPIOFEN	[5:1]	0x0
GPIOEEN	[4:1]	0x0
GPIODEN	[3:1]	0x0
GPIOCEN	[2:1]	0x0
GPIOBEN	[1:1]	0x1
GPIOAEN	[0:1]	0x0

\

Now for the register MODDER we can view that in fact the PB0 is set as 1 (OUTPUT)

MODER	0x40020400	0x281
MODER15	[30:2]	0x0
MODER14	[28:2]	0x0
MODER13	[26:2]	0x0
MODER12	[24:2]	0x0
MODER11	[22:2]	0x0
MODER10	[20:2]	0x0
MODER9	[18:2]	0x0
MODER8	[16:2]	0x0
MODER7	[14:2]	0x0
MODER6	[12:2]	0x0
MODER5	[10:2]	0x0
MODER4	[8:2]	0x2
MODER3	[6:2]	0x2
MODER2	[4:2]	0x0
MODER1	[2:2]	0x0
MODER0	[0:2]	0x1

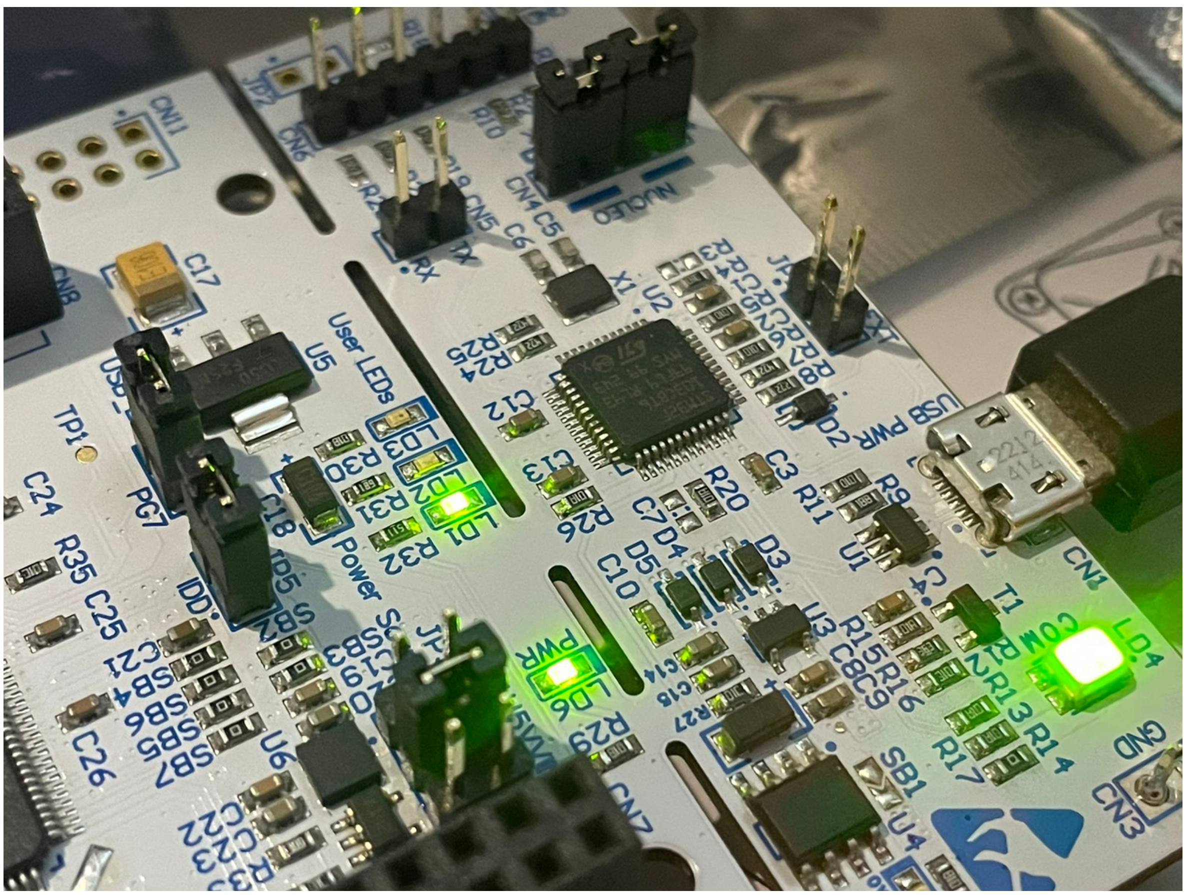
Now for the Delay we can view that the ODR (OUTPUT register) sets the PIN to ON.

ODR	0x40020414	0x1
ODR15	[15:1]	0x0
ODR14	[14:1]	0x0
ODR13	[13:1]	0x0
ODR12	[12:1]	0x0
ODR11	[11:1]	0x0
ODR10	[10:1]	0x0
ODR9	[9:1]	0x0
ODR8	[8:1]	0x0
ODR7	[7:1]	0x0
ODR6	[6:1]	0x0
ODR5	[5:1]	0x0
ODR4	[4:1]	0x0
ODR3	[3:1]	0x0
ODR2	[2:1]	0x0
ODR1	[1:1]	0x0
ODR0	[0:1]	0x1

And after the Delay and the clear the OUPUT register clears the PB0

ODR	0x40020414	0x0
ODR15	[15:1]	0x0
ODR14	[14:1]	0x0
ODR13	[13:1]	0x0
ODR12	[12:1]	0x0
ODR11	[11:1]	0x0
ODR10	[10:1]	0x0
ODR9	[9:1]	0x0
ODR8	[8:1]	0x0
ODR7	[7:1]	0x0
ODR6	[6:1]	0x0
ODR5	[5:1]	0x0
ODR4	[4:1]	0x0
ODR3	[3:1]	0x0
ODR2	[2:1]	0x0
ODR1	[1:1]	0x0
ODR0	[0:1]	0x0

Finally, here is a physical photo of the LED turning ON.



LD1 is the LED programmed