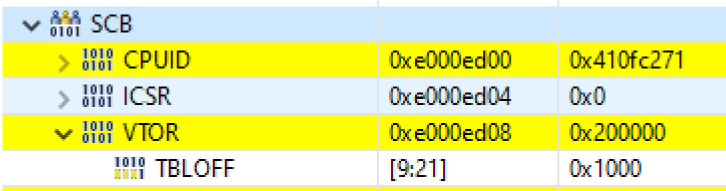
JOURDREN Vincent
2G1TD1TP1

TP – Oscilloscope Generator

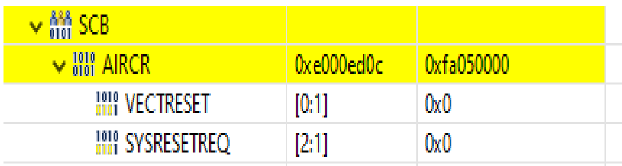
6 RESET Vector

Task 4: Downloading the code and finding the first instruction

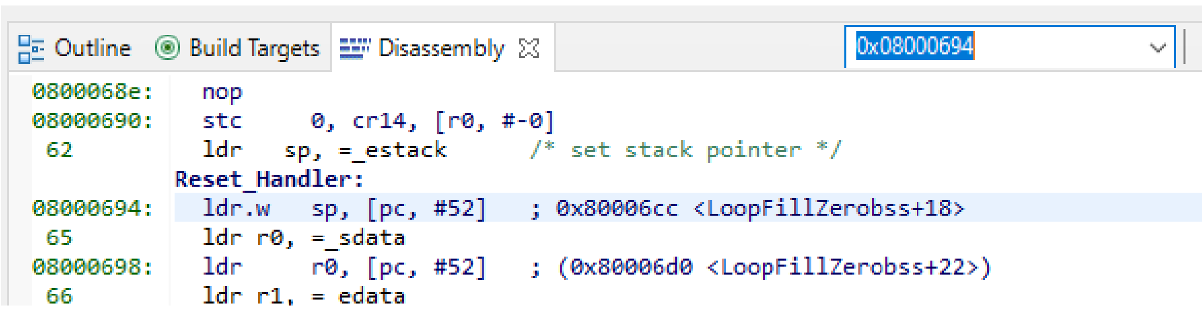
The starting value of SCB_VTOR is 0x200000.
The SCB_VTOR contains the beginning of the vector table that allows to locate the interrupt processing codes.



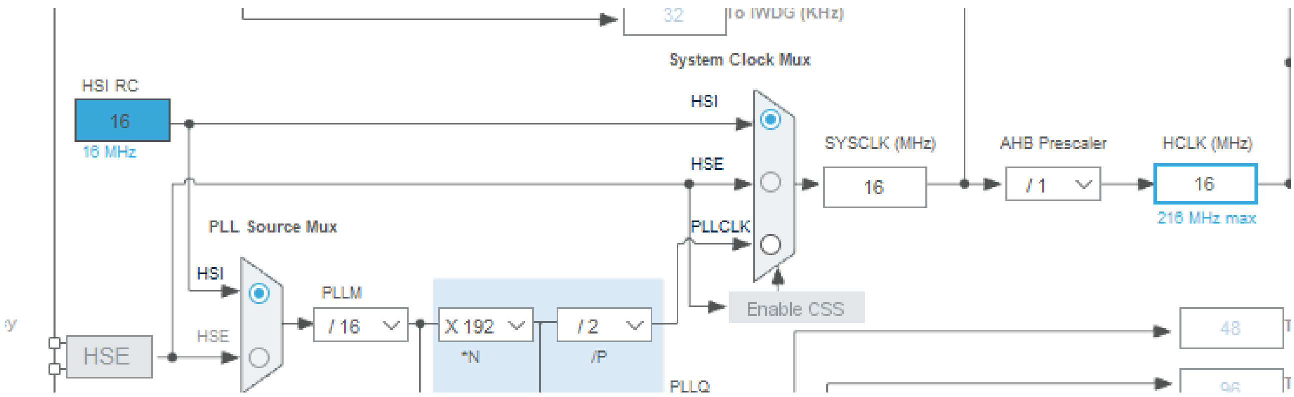
The RESET vector is placed in the SCB_AIRCR register



The first instruction is written to address 0x08000694
The assembly code for this instruction is: LDR SP, [PC, #52].
PC Program counter is the register indicating the next instruction.



7 The clocks before and after initialization



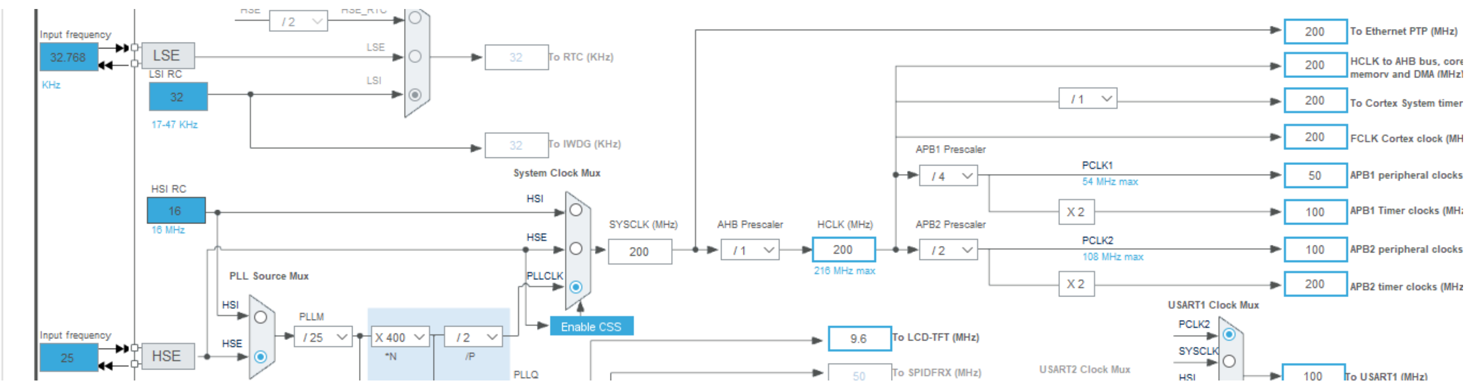
Register	Address	Value
STM32F746		
RCC		
PLLCFGR	0x40023804	0x24003010
CFGR	0x40023808	0x0

We then have :

- RCC_PLLCFGR = 0010 0100 0000 0000 0011 0000 0001 0000
- RCC_CFGR = 0x0

According to RCC_CFGR: SW=0 => HSI = 16 Mhz ; HPRE=0 => /1 ; PPRE1=0 => /1 ; PPRE2=0 => /1 ;

We conclude that all clocks SYSCLK, HCLK, APB1, APB2 and Timers APB1 and APB2 are set to the internal HSI clock at 16 MHz at start-up by default.



Register	Address	Value
STM32F746		
RCC		
PLLCFGR	0x40023804	0x29406419
CFGR	0x40023808	0x940a

We then have :

- RCC_PLLCFGR = 0010 1001 0100 0000 0110 0100 0001 1001
- RCC_CFGR = 0000 0000 0000 0000 1001 0100 0000 1010

According to RCC_CFGR : SW=10 => PLL ; HPRE=0 => /1 ; PPRE1=101 => /4 ; PPRE2=100 => /2 ;

According to RCC_PLLCFGR : PLLM = 011001 => /25 ; PPLN=110010000 => x400 ; PLLP=00 => /2 ; PLLSRC=1 => HSE ;

We have : HSE = 25 Mhz, alors SYSCLK = HSE ∗ PLLN PLLM ∗ PLLP = 15 ∗ 400 25 ∗ 2 = 200 MHz

HCLK = SYSCLK / HPRE = SYSCLK = 200 MHz

APB1 peripheral clocks = HCLK PPRE1 = 200 4 = 50 MHz
APB1 timer clocks = HCLK PPRE1 ∗ 2 = 200 2 = 100 MHz
APB2 peripheral clocks = HCLK PPRE2 = 200 2 = 100 MHz
APB2 timer clocks = HCLK PPRE2 ∗ 2 = 200 MHz

After LAB_Init, the values of the clocks and timers are modified with the above values. Our theoretical results are in agreement with the clock configuration of our project (picture above).

8 The green LED

We will implement 2 types of internal devices: a GPIO to control the logic levels to the LEDs and a Timer to count the time.

Preparation 8 : Registers for the LED control

- From the course and the card documentation, we know that to activate a GPIO, the register RCC_AHB1ENR must be modified.

- The base address of the RCC register is 0x4002 3800, to which an offset 0x30 must be added to obtain the desired register, which is 0x4002 3830.

- More precisely, the 8th bit must be set to 1. This bit corresponds to the GPIOI module (as i is the 9th letter of the alphabet). The 8th bit of address 0x4002 3830 must therefore be set to 1 to activate the GPIOI.

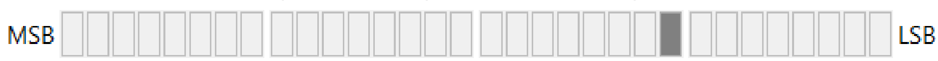
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	Reserved	
	rw	rw	rw	rw	rw	rw			rw	rw			rw		

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

Task 9 : Activation of GPIOI

Indeed, by displaying the AHB1ENR register in debug mode, we can see that the 8th bit of the address "0x4002 3830" is at 1, so the GPIOI port is already activated.
Moreover, our LED is well switch on.

Register	Address	Value
AHB1ENR	0x40023830	
GPIOIEN	[8:1]	
AHB1LPENR	0x40023850	
GPIOILPEN	[8:1]	
GPIOI		



Preparation 10 : Configuration for a LED pin

The registers to be used to configure the GPIOI pins are :
- GPIOI_MODER
- GPIOI_OTYPER
- GPIOI_PUPDR
- GPIOI_OSPEEDR
- GPIOI_ODR
- GPIOI_IDR

From the documentation and the course, we know that the base address to modify the GPIOI module is: 0x40022000, and that there are 16 pins available for the GPIOI port.

0x40022000	GPIO Port I 16 pins

According to the STM32 technical documentation, we need 7 bits to configure a pin: 2 bits for "MODER", 1 for "OTYPER", 2 for "OSPEEDR", 2 for "PUPDR".
bits: 2 bits for "MODER", 1 for "OTYPER", 2 for "OSPEEDR", 2 for "PUPDR".

7 bits to define a configuration:

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]		I/O configuration	
01	0	SPEED [B:A]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1		0	0	GP output	OD
	1		0	1	GP output	OD + PU
	1		1	0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	

The addresses of the various registers to be modified are also known:

Registre	MODER	OTYPER	PUPDR	OSPEEDR	IDR	ODR
Adresse	0x4002 2000	0x4002 2004	0x4002 200C	0x4002 2008	0x4002 2010	0x4002 2014
Bits à modifier	2,3	1	3,2	3,2	1	1

Pin I1 should be configured as a GP output push-pull without PU/PD as it is not needed as it is an LED and not a push button/switch.

Required work 11 : Pin configuration and test

In order to configure the pin, we use CubeMX view :

```
/*Configure GPIO pins : ARDUINO_D7_Pin ARDUINO_D8_Pin LED_GREEN_Pin LCD_DISP_Pin */
GPIO_InitStruct.Pin = ARDUINO_D7_Pin|ARDUINO_D8_Pin|LED_GREEN_Pin|LCD_DISP_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
```

We set the pin to GP OUTPUT PP mode.

Task 12: “LED DispGreen

To make this LED blink, we create the "LED_DispGreen" function in the "LED.c" file, which should turn on the LED if the LSB of val is 1 and turn it off otherwise

```
void LED_DispGreen(int val){
    if (val==1){
        GPIOI->ODR |= 0x2;  // if the LED is off we put the first bit of GPIOI_ODR at 1
    }                                                             //= We switch on the LED
    else{
        GPIOI->ODR &= ~0x2; // if the LED is on we put the first bit of GPIOI_ODR at 0
    }                                                                         //= We switch off the LED
}
```

9 Let’s take a step back : a review of this first part

In this section we have seen what happens in our card during the boot process. After the reset, the Cortex M4 microprocessor fetches the address of the first PC instruction from ROM. The vectors represent the addresses of the interrupt code. The SCB_VECTOR represents the address of the beginning of the vector table.

The clocks of the STM32 microcontroller are then initialized, which are defined and parameterized by theperipheral RCC and its registers RCC_CFGR and CFGR_PLLCFGR (initially with default operating frequenciesoperating frequencies and then after an initialization). These registers are locatedin the "Memory for peripheral".

Finally, we activated a peripheral (GPIOI) by writing to the memory of the RCC module - again in the "Memory for peripheral" - but this time if on the AHB1ENR buses by setting the bit that concerns the GPIOI clock to 1 bit which concerns the GPIOI clock and which is called GPIOIEN. For this we used the Cube interfacethe CubeMX view interface.

10 Light animation with interrupt handler

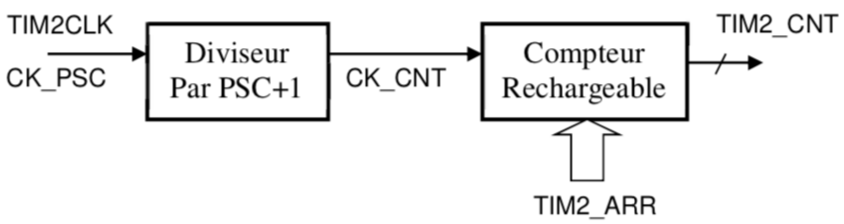
Required word 14 : Timer clock frequencies after « LAB Init »

The frequency of the timers are :
f(APB1TIM) = 108 MHZ
f(APB2TIM) = 216 MHz

Timer2 is on the APB1TIM bus so its clock frequency is 100 MHz.

Work requested 15 : principle of a TIMER

Diagram showing the operating principle of a timer :



TIM2_CLK : frequency feeding the clock (108 MHz here)
TIM2_PSC : The first frequency divider is the prescaler.
CK_CNT : frequency divided by PSC+1 with respect to TIM2CLK → clock signal that paces the counter.
TIM2_ARR : sets the amplitude of evolution of the counter, in the case of a count, it counts from 0 to TIM2_ARR (thus a cycle of TIM2_ARR+1). The ARR value corresponds to the value at which the counter restarts when it reaches the end of its cycle.
TIM2_CNT : stores the count value.

Job requirement 16 ; Activation and configuration of TIMER2

We configure the internal clock as the clock source.
Then we modify the Prescaler and Counter Clock values in the code given in the course in order to obtain 4 complete counts per second.
So we have PSC = 30000 and ARR = 900

Registers for configuring TIM2:

TIM2_PSC at address 0x40000028
TIM2_ARR at address 0x4000002C

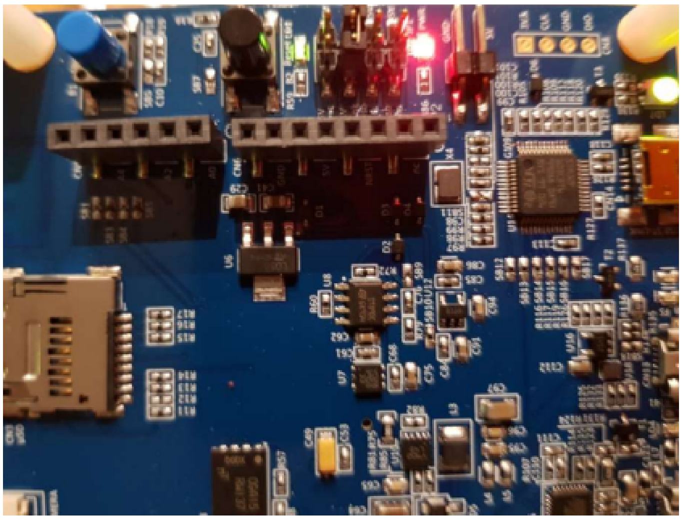
```
/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 29999;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 899;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
```

Required work 17 : TIMER2 authorization and LED flashing

TIMER2 interrupts must be authorised at the NVIC.

NVIC registers concerned by Timer 2: NVIC_ISERx, NVIC_ICERx for interrupt authorisation and NVIC_IPR for priority definition.

The TIMER2 interrupt is number 28, so it will be authorised in the ISER0 register, which must be set to 0x10000000.



Application 18 : Displaying an absolute time using Timer3

We will use the graphic libraries (LCD) to display somewhere on the screen the absolute time elapsed since the reset of the card in seconds by implementing the TIMER3.

<u>Timer :</u>

TIMER3 is also on APB1 so TIM3CLK = 108 MHz
We choose 10 activations per second, we repeat the calculations of TIMER2 and we obtain :
PRESC = 2700 and ARR = 4000.

Work required 19 : Timer Vector 2,3,4 and others

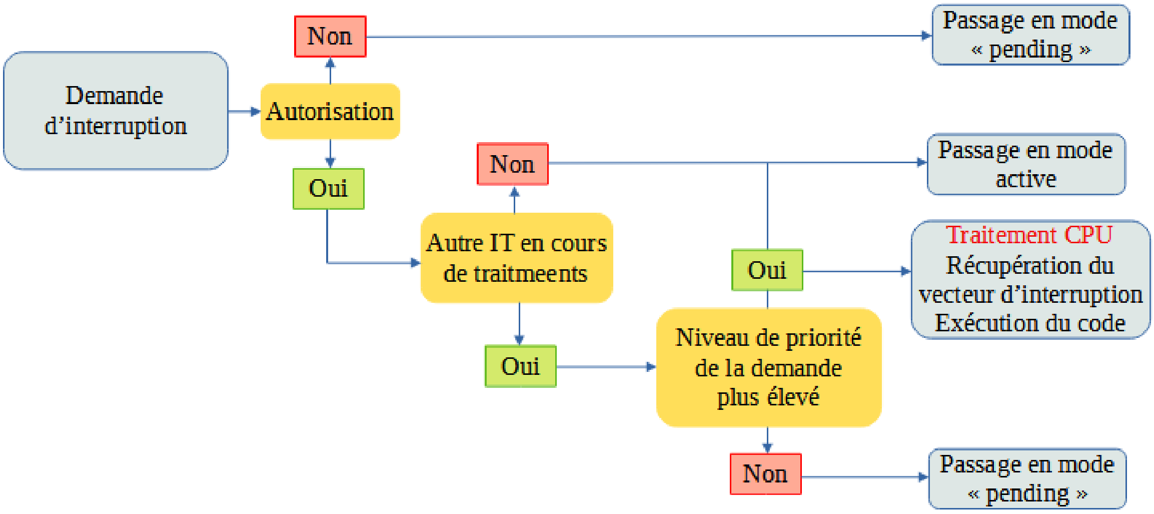
To find the values of each timer vector, we must look at the value of the SCB_VTOR register.
To do this, we use the Memory window offered in the Debug mode, where we can read the value of the timer vectors at each of their addresses.
In the vector table you can find the position of the timers and the address where the vector is located.

TIM2:
vector position: 28
Address where the vector is located: 0x0000 00B0
Calculation of the address: 28+16=44 -> 44*4=176-> 0xB0 in hexadecimal
Value of the vector: 0x72AE 699D

TIM3:
vector position: 29
Address where the vector is located: 0x0000 00B4
Calculation of the address: 29+16=45 -> 45*4=180-> 0xB4 in hexadecimal
Value of the vector: 0x614B A9CF

TIM4:
position of the vector: 30
Address where the vector is located: 0x0000 00B8
Calculation of the address: 30+16=44 -> 44*4=184-> 0xB8 in hexadecimal
Value of the vector: 0xCF68 367144

Assignment 21 : Make your own picture sheet of this assessment



Rq : Pending interrupts occur when all priority interrupts have been completed.

12 Adjustable blinking of the green LED

Task 22 : Light animation for the green LED

We add a function LED_SetFreqGreen to change the flashing frequency of the green LED for frequencies of 1Hz, 2 Hz, 3 Hz and 4 Hz. for frequencies of 1Hz, 2 Hz, 3 Hz and 4 Hz.

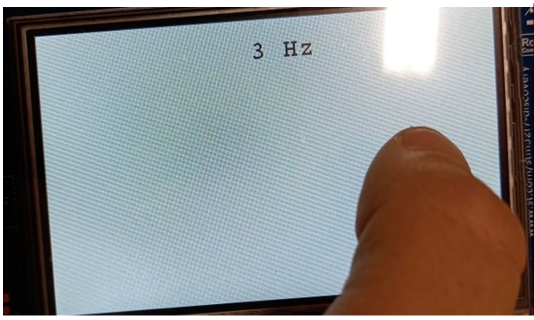
```
void LED_SetFreqGreen(int f){
    if (f==1){
        htim2.Init.Period = 3599; // f = 1 Hz
    }
    else if (f==2){
        htim2.Init.Period = 1799; // f = 2 Hz
    }
    else if (f==3){
        htim2.Init.Period = 1199; // f = 3 Hz
    }
    else if (f==4){
        htim2.Init.Period = 899; // f = 4 Hz
    }
}
```

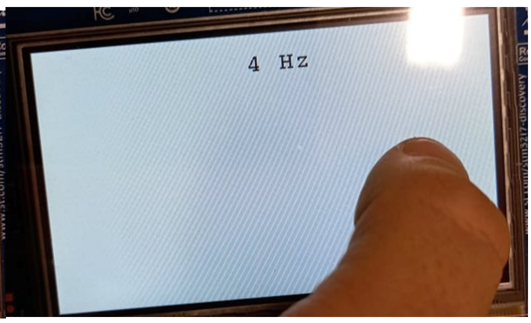
Job requirement 23: Tactile adjustment of the green LED frequency

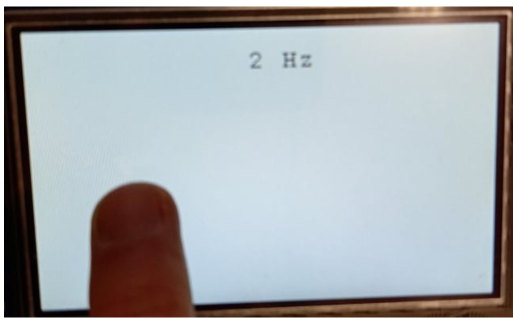
A new timer is implemented and processed to adjust the flashing frequency of the green LED using the tactile properties of the STM32F7's display.
We write this code in HMI.c.

The processing of this new timer will scan the state of the surface. If the surface of the screen has been touched, then we propose to recover the position of the "touch" and to increment the frequency if it corresponds to a right portion of the screen or to decrement the frequency for a left portion of the screen.

Timer 4 is set up with a frequency of 25Hz to try to scan the surface state frequently.





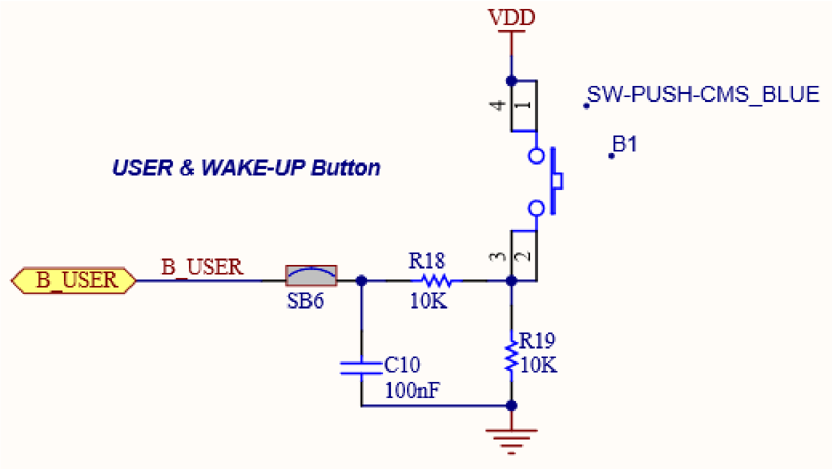


Preparation 24: Beautiful Blue Button/ schematics

The capacitance is used to prevent voltage surges when the push button is pressed.

The time constant is associated with the resistance and capacitance of a switch assembly and is equal to RC.

τ = 10KΩ x 100nF = 1ms



Work requested 27: but by the way, why the interrupts

Another way to do this with the button would have been to poll the logic level of bit 11 of GPIOI_IDR bit 11 of GPIOI_IDR: we wait until the logic level changes using a while.

The problem with this method is that the CPU (microprocessor) is then dedicated to this task.
We will not be able to implement any other functionality on our system.

The solution is therefore to use interrupts as seen previously: the CPU only acts during event and is not monopolised.

13 Digital signal generation

Job requested 29: Activation of a pin in “GENE Init” and test

Configuration of the selected pin on the Arduino connector :

-	PI3	D7	8	CN4 digital
TIM12_CH1	PH6	D6	7	
TIM5_CH4,SPI 2_NSS	PI0	D5	6	
-	PG7	D4	5	
TIM3_CH1	PB4	D3	4	
-	PG6	D2	3	
USART6_TX	PC6	D1	2	
USART6_RX	PC7	D0	1	

We choose to use pin 0 of port A: PA0 → We will therefore configure the GPIOA. We'll use timer 5 on the APB1 bus and we'll go into push pull output mode.

During our configuration the GPIOAEN will indeed change from 0 to 1.

After checking the GPIOA_MODER register: We are on the PA0 pin, so we look at GPIOA_MODER0.

During our configuration, MODER0 effectively changes from 00 to 01 which, according to the GPIOx_MODER's documentation to "General purpose output mode" → we are in output mode.

Job requested 29: Activation of a pin in “GENE Init” and test

We add the TogglePin() function which toggles the signal of the chosen pin at each call of TogglePin, the signal is modified (it goes to 1 if it is 0 or it goes to 0 if it is 1).

To do this, we modify the GPIOI_ODR register in the same way as we did previously to make the LEDs blink (part 8 turn on the LEDs function LED_DisplayGreen).

After checking the GPIOI_ODR0 register, we have GPIOI_ODR0 corresponding to pin PI0 which is set to 1.
Moreover, we can see that thanks to the masking, the other values of the pins are effectively kept.

Job requested 30 : Toggle Arduino pin signal in “GENE_TogglePin

```c
void GENE_TogglePin(void){
	if(toggle%2 == 0){
		GPIOA->ODR |= (1<<0);
	}
	else if(toggle%2 == 1){
		GPIOA->ODR &= 0xFFFFFFFE;
	}
	toggle += 1;
}
```

Task 31: Generation of a periodic signal on the selected pin in “GENE.c”

Using Timer 5, we will generate a digital signal on the PA0 pin of the Arduino with a frequency of 1 kHz. We are still on APB1 (so the clock frequency is 100 MHz).

This corresponds to the following setting:
1kHZ → 1 interrupt per second = [100M / (PSC+1)] *[1/(ARR+1)].

In order to have a usable value, PSC is set to 3000 → ARR = 36.

Required work 32: Generation of a periodic signal on the selected pin in “GENE.c”

We write the function GENE_SetFreqPin(int f) allowing to set the frequency of the signal between 1 and 1000 Hz.

To do this, we modify the values of TIM5_PSC (prescaler of timer 5) and TIM_ARR (arr of timer 5).

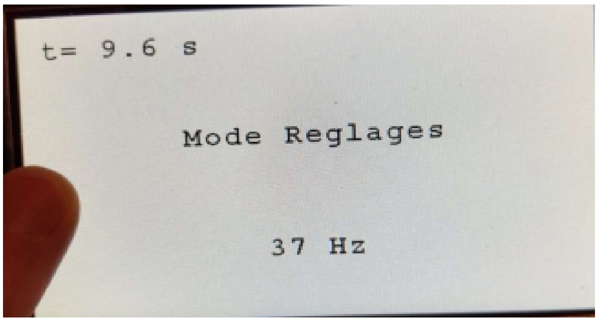
```c
void GENE_SetFreqPin(int f){
	htim5.Init.Period = 36000/f-1;
}
```

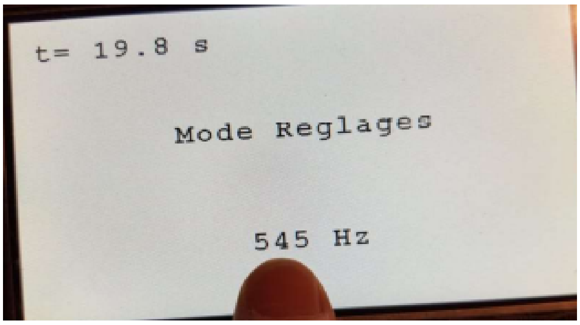
Work required 33: HMI improvement

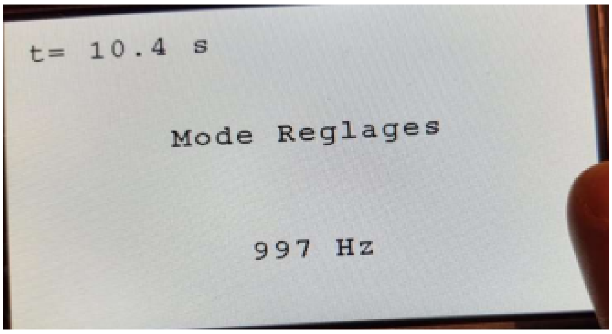
We add a setting mode in which the position of our finger on the screen allows us to set the frequency of the PA0 pin.

Touch screen appearance :

Settings mode







We do get a slider on the touch screen of our STM32 allowing us to modify the frequency from 1Hz (far left) to 1kHz (far right). frequency from 1Hz (far left) to 1kHz (far right).

Neutral mode (allows the frequency to be blocked)