Before this project, I want to add an additional project to understand how button works. Purpose of this project is when you press the button it will turn on the LED. In the below, I finded the schematic of button and this is how it is made.

Firstly, I need to understand how button's works. After deciding that, I asked chatGpt. Here is the answer.

Activating a button on an STM32 microcontroller involves configuring the relevant GPIO pin and setting up an external interrupt to detect button presses.

→ external interrupt: They are used to cause interrups in the program when their logical value changes or they are at a certain state. Since this function is usually used to monitor external logical signals, these pins are called external interrupt pins.

1. GPIO Configuration:

```
// Register definitions for STM32F411E-DISCO
uint32_t volatile *pRCC_AHB1ENR = (uint32_t*) (0x40023800 + 0x30);
uint32_t volatile *pGPIOA_MODER = (uint32_t*) (0x40020000 + 0x00);

void button_gpio_init(void) {
    // Enable GPIOA clock
    *pRCC_AHB1ENR |= (1 << 0);

    // Configure PA0 as input
    *pGPIOA_MODER &= ~(0x3 << 0); // Clear mode bits for PA0
    *pGPIOA_MODER |= (0x0 << 0);  // Set PA0 as input
}
```

clock enable offset

↳ mode register offset

→ you can understand all this with reference manual.

2. External Interrupt Configuration:

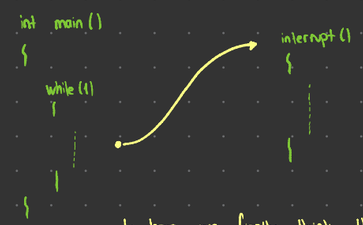
External interrupt/event controller (EXTI)

The external interrupt/event controller consists of 21 edge-detector lines used to generate interrupt/event requests. Each line can be independently configured to select the trigger event (rising edge, falling edge, both) and can be masked independently. A pending register maintains the status of the interrupt requests. The EXTI can detect an external line with a pulse width shorter than the Internal APB2 clock period. Up to 81 GPIOs can be connected to the 16 external interrupt lines.

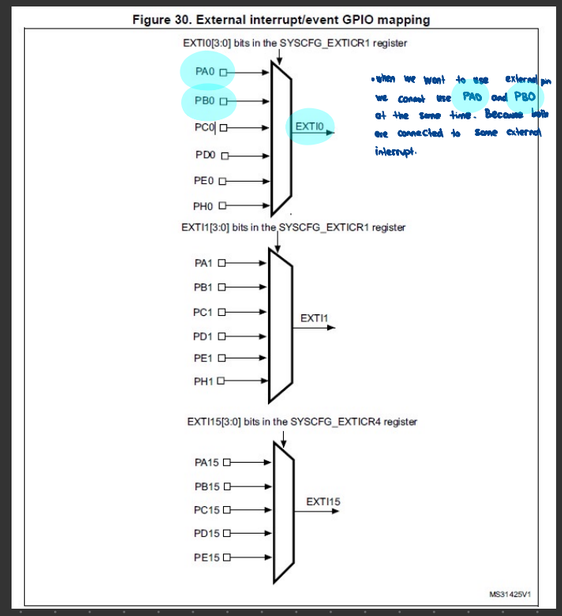
also this is 23 external.

for my STM32F411E-Disco board, we have 52 maskable interrupt channel (I took that info from reference manual)

the logic of interrupt function:



In here, we firstly think that, our program working according to code blocks and at the same point our function needs to stop and run the interrupt function first, ↳ (externaly cutted wit any other component in any time) Then it will continue where it interrupted.



-referance manual (external interrupt/event) GPIO mapping

The five other EXTI lines are connected as follows:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB OTG FS Wakeup event
- EXTI line 21 is connected to the RTC Tamper and TimeStamp events
- EXTI line 22 is connected to the RTC Wakeup event

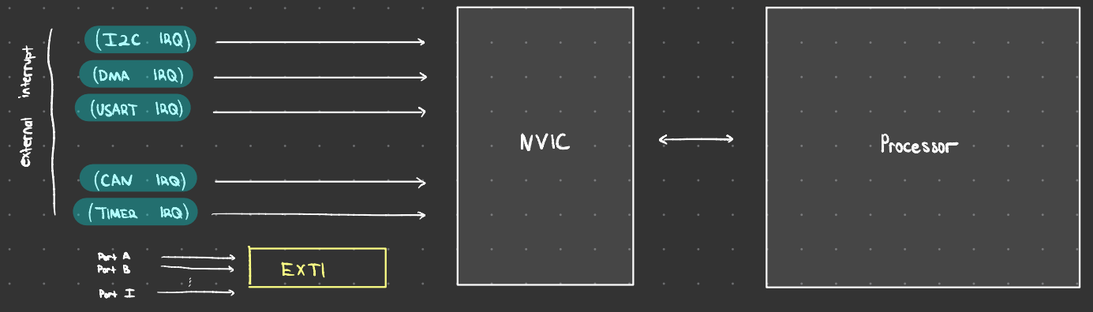
these are the additional 7 EXTI lines.

a structure built by birds or insect to leave their eggs in to develop.

NVIC (Nested Vector Interrupt Control)

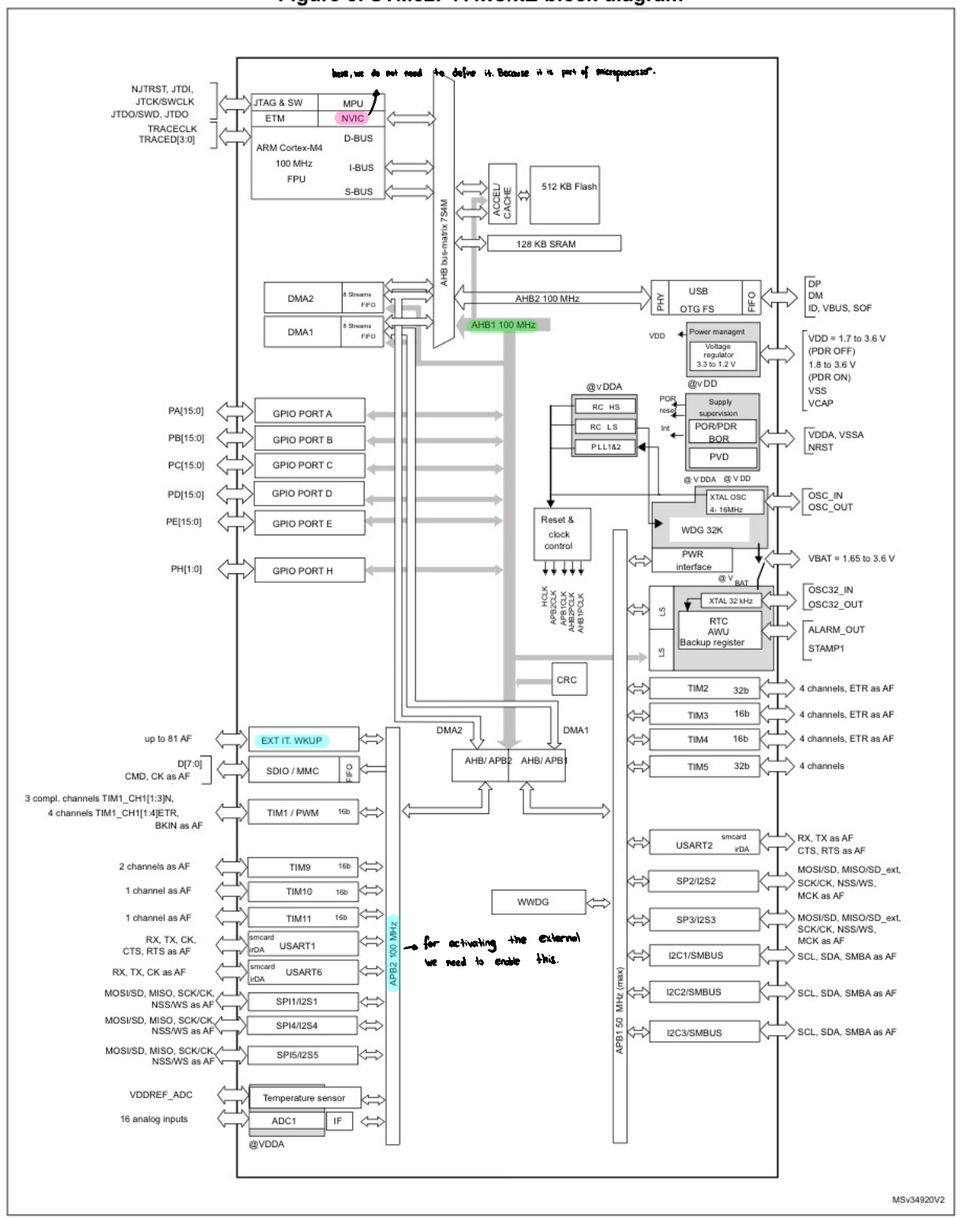
to fit one object inside another, or to fit inside in this way.

It designed for making processors job uncomplicate. The logic is we are using this NVIC when we have interrupt which comes from external. It firstly goes NVIC then processor.



I also added some informations for NVIC, do not forget to check it.

Figure 3. STM32F411xC/xE block diagram



1. The timers connected to APB2 are clocked from TIMxCLK up to 100 MHz, while the timers connected to APB1 are clocked from TIMxCLK up to 100 MHz.

```c
#include <stdint.h>
#include <stdio.h>

//global shared variable between main code and ISR
uint8_t volatile g_button_pressed = 0;

uint32_t g_button_press_count =0;

void button_init(void);

uint32_t volatile *pRCCAHB1ENR      = (uint32_t*) (0x40013C00 + 0x14);
uint32_t volatile *pRCCAPB2ENR      = (uint32_t*) (0x40023800 + 0x30);
uint32_t volatile *pRCCAHB1LPENR    = (uint32_t*) (0x40023800 + 0x44);
uint32_t volatile *pPORTHMODER      = (uint32_t*) (0x40020000 + 0x00);
uint32_t volatile *pSYSCFGEXTICR1   = (uint32_t*) (0x40013C00 + 0x08);
uint32_t volatile *pEXTIRTSR        = (uint32_t*) (0x40013C00 + 0x08);
uint32_t volatile *pNVICIRQEnReg    = (uint32_t*) 0xE000E100;

int main(void)
{
    button_init();

    while(1)
    {
        //Disable interrupt
        *pEXTIMaskReg &= ~( 1 << 0);

        if(g_button_pressed){
            //Some delay until button debouncing gets over
            for(uint32_t volatile i=0;i<500000/2;i++);
            g_button_press_count++;
            printf("Button is pressed : %lu\n",g_button_press_count);
            g_button_pressed = 0;
        }
    }
}
```

Handling the interrupt in main loop:
- In the main loop, the program checks the `g_button_pressed` flag
- The main loop then executes the code to handle the button press, such as incrementing the press count and printing a message.

pending register's offset

offset of interrupt mask register

tam oturmadı

rising trigger selection register's offset

→ in this function, we defined button.

→ After pressing the button, you need to make it clear in every step. Because in every ms it will refresh itself. Become of that, we are masking it and disable it.

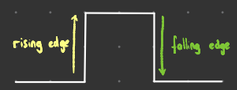
```c
void button_init(void)
{
    //Enable interrupt
    *pEXTIMaskReg |= ( 1 << 0);
}

void button_init(void)
{
    *pClkCtrlReg |= ( 1 << 0);

    *pClkCtrlRegApb2 |= ( 1 << 14);

    *pEXTTIEdgeCtrlReg |= ( 1 << 0);

    *pEXTIMaskReg |= ( 1 << 0);

    *pNVICIRQEnReg |= ( 1 << 6);
}

/* This is button interrupt handler*/
void EXTI0_IRQHandler(void)
{
    //Make this flag SET . if button pressed
    g_button_pressed = 1;

    *pEXTTIPendReg |= ( 1 << 0);
}
```

external interrupt named as System Configuration controller clock and when you looked to reference manuel, for activating SYSCFG EN you need activate 14th bit on APB2.

tam oturmadı adresini bulmadım.

flag is in here!

As we know from property of signals, The shape of signal is



and we want to use rising edge because when we push button it will do something (print or turn on LED)

10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	Reserved		TR18	TR17	TR16
									rw	rw			rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Bits 22:0 **TRx:** Rising trigger event configuration bit of line x
0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

Because of this part, we are making 0th bit (1)

EXTI masking: this term "mask" typically refers to the ability to enable or disable specific interrupt lines.

→ If a bit set (1), the corresponding interrupt line is (unmasked), meaning that it is allowed generate interrups.

→ If a bit set (0), the corresponding interrupt line is disabled (masked), meaning that it is not allowed to generate interrups.

EXTI Pending: the term "pending" usually refers to an interrupt or an event that occured but has not been serviced yet. Specificially, it often relates to the Pending Register associated with interrupt handling.

1. Pending Register (EXTI_PR)

→ The EXTI Pending Register is used to (indicate (to show, point, or make clear in another way)) which external interrupt lines have pending interrup requests.

→ Each bit in this register corresponds to a specific external interrupt.

→ If a bit is set (1), it means that an interrupt event has occured on the corresponding interrupt line, but the interrupt has not been serviced (handled) yet.

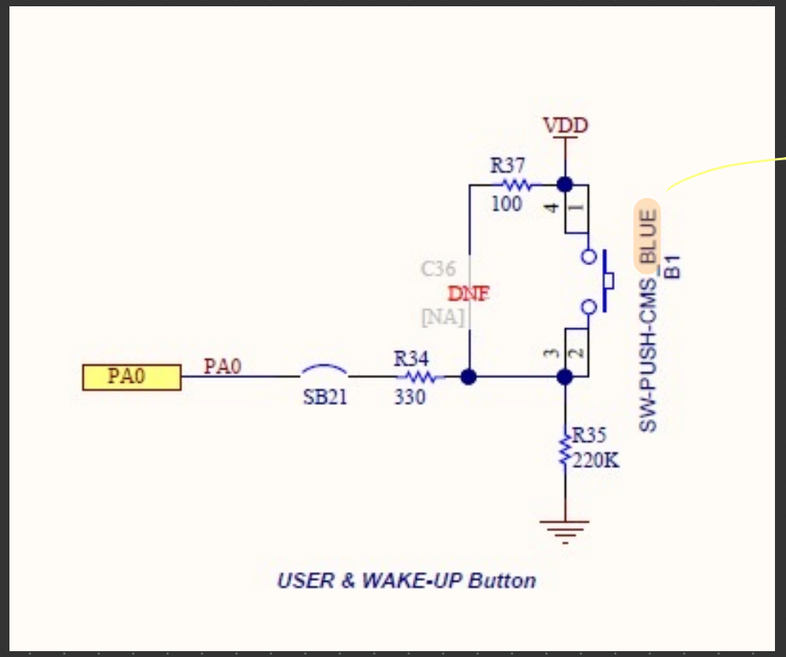
In summary, "pending" in STM32 terms usually means that an interrupt event has occured but has not been serviced yet. The pending register is a mechanism to keep track of which interrups are waiting to be processed.

NVIC:

In the context of STM32 microcontrollers, NVIC stands for Nested Vectored Interrupt Controller. The NVIC is a peripheral that manages the interrupts in the STM32 devices. It is responsible for controlling the priority of interrupts, enabling and disabling interrupts, and managing the nested interrupt requests.
If you have only one external interrupt, you would typically need to configure and activate the NVIC. The NVIC (Nested Vectored Interrupt Controller) is a core component of ARM Cortex-M microcontrollers, including those in the STM32 family. It plays a crucial role in managing interrupts, regardless of the number.

So important part of this topic!

!!! Ask about flags in microcontrollers !!!



Blue pin is connected to PA0 pin.

I took this image from Schematics

- We need to set this button as input. Because, if we don't we cannot count how many times did we push?

indicate left

Usage of 'const' and 'volatile' together

We can also use both 'const' and volatile qualifiers in a variable declaration as our goal.

example:

```
uint8_t volatile * const pReg = (uint8_t*) 0x ..... ;
```

→ This variable shows us, we have constant pointer adress but the data which we keep in this adress, can change. (It comes from type of volatile)

```
uint8_t const volatile * const pReg = (uint8_t*) 0x ..... ;
```

→ We can use const volatile, when we are using const volatile

→ The data which is on that pointer, can change because we added volatile in here. Const for the programmer for keeping it same when programming.

So important!