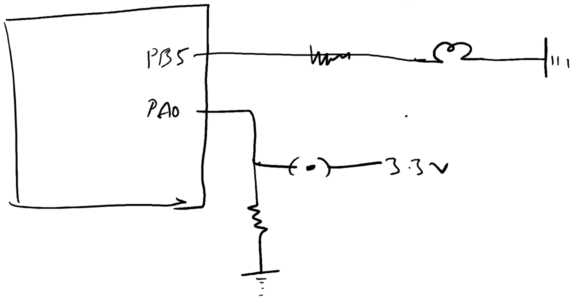
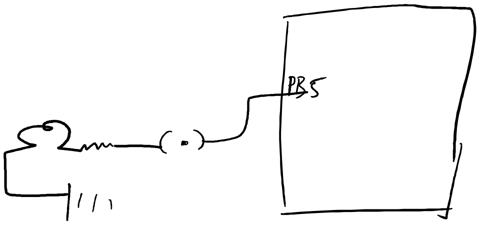
STM32 Stock

14 April 2025 16:03

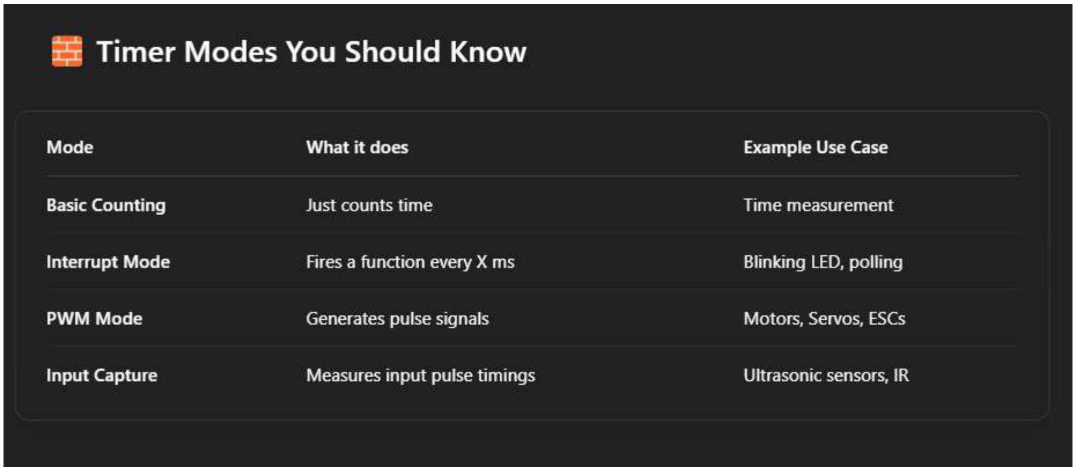
DIGITAL SIGNALS



NORMAL SWITCH



STM32 TIMERS



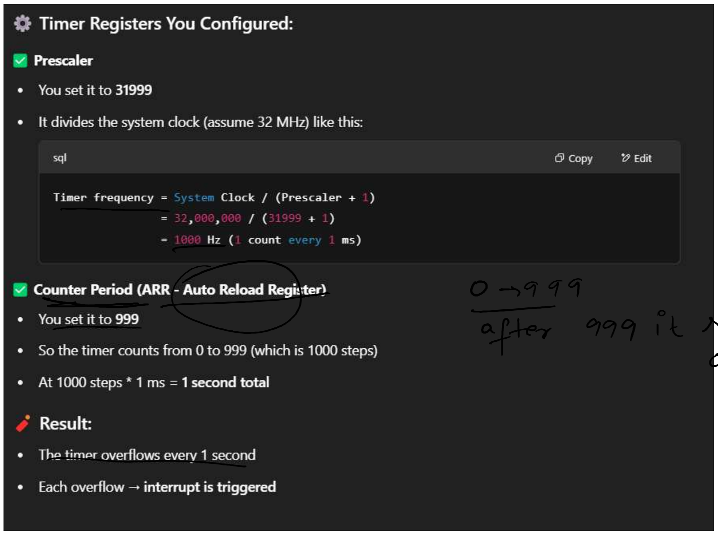
🧱 Timer Modes You Should Know

Mode	What it does	Example Use Case
Basic Counting	Just counts time	Time measurement
Interrupt Mode	Fires a function every X ms	Blinking LED, polling
PWM Mode	Generates pulse signals	Motors, Servos, ESCs
Input Capture	Measures input pulse timings	Ultrasonic sensors, IR

timer frequency = $\frac{1000Hz}{1s}$

Prescalar -> frequency -> 1000Hz

ARR -> reload after how much count.



⚙️ Timer Registers You Configured:

☑ **Prescaler**

- You set it to **31999**
- It divides the system clock (assume 32 MHz) like this:

```sql
Timer frequency = System Clock / (Prescaler + 1)
                = 32,000,000 / (31999 + 1)
                = 1000 Hz (1 count every 1 ms)
```

☑ **Counter Period (ARR - Auto Reload Register)**

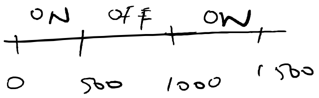
- You set it to **999**
- So the timer counts from 0 to 999 (which is 1000 steps)
- At 1000 steps * 1 ms = **1 second total**

Result:

- The timer overflows every 1 second
- Each overflow → **interrupt is triggered**

0 -> 999
after 999 it reloads/overflows

for 2Hz ->
prescalar : 500

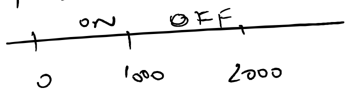


for 1Hz

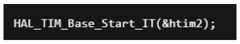
Timer Code :-

for 1Hz

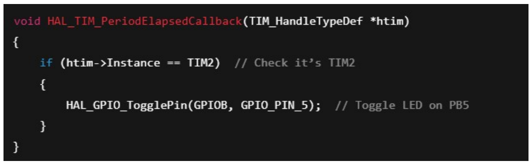
presc → 999



Timer Code :-



```
HAL_TIM_Base_Start_IT(&htim2);
```



```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM2)  // Check it’s TIM2
    {
        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_5);  // Toggle LED on PB5
    }
}
```

Timer + Button Combo Project

Note :- GPIO input requires full 3.3V input so no resistor

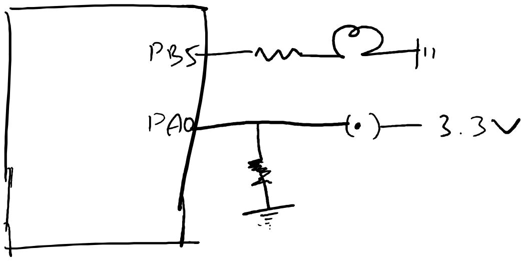
Requirement :-

1> Digital input

2> Digital output

} the rest is software part.

Diagram :-



Workflow :-

1> check if dig inp & output is working.

2> Use Timer & check if timer works

1> initiate timer

2> write timer overload function.

Toggle timer Blink button project

first set GPIO mode to GPIO_EXTI0 in IOC file.

using a static int variable → records the state.

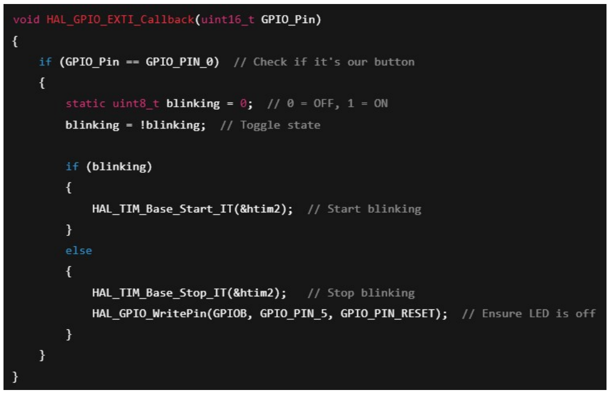
=> How to detect the click of a button?

EXTI → external interrupts → any external interrupt apart from the regular flow of compiler. eg:- click of a button.

executed after an EXTI

flow of compiler
eg:- click of a button.

executed after an EXTI

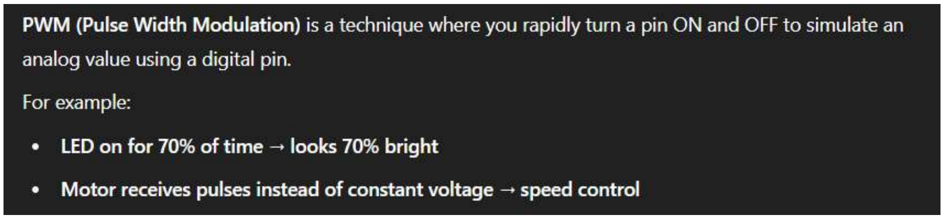


```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_0)  // Check if it's our button
    {
        static uint8_t blinking = 0;  // 0 = OFF, 1 = ON
        blinking = !blinking;  // Toggle state

        if (blinking)
        {
            HAL_TIM_Base_Start_IT(&htim2);  // Start blinking
        }
        else
        {
            HAL_TIM_Base_Stop_IT(&htim2);   // Stop blinking
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);  // Ensure LED is off
        }
    }
}
```

) runs only during 1st execution of function
static used so that when the function is called again variable is same as old.

PWM (Pulse width modulation) :-



PWM (Pulse Width Modulation) is a technique where you rapidly turn a pin ON and OFF to simulate an analog value using a digital pin.

For example:

- **LED on for 70% of time → looks 70% bright**
- **Motor receives pulses instead of constant voltage → speed control**

generate PWM signals **without blocking your CPU**.

Uses :- dimming LEDs, varying rpm of motors etc...

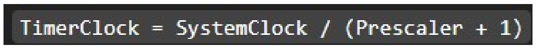
Technical Terms :-

* Frequency :- how many cycles per second?
* Duty cycle :- what percentage of a cycle is current on.

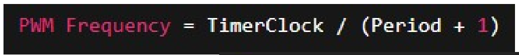
Using PWM :-

* take any timer & set any one channel to PWM gen

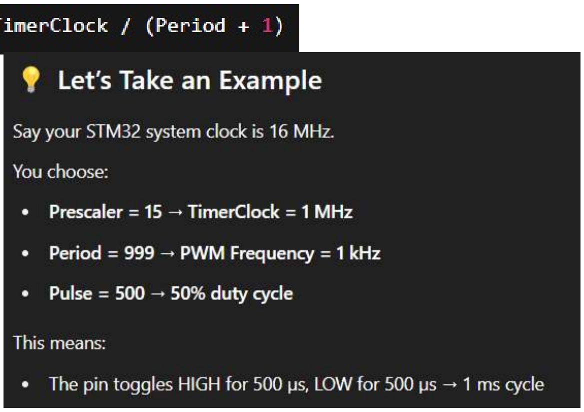
What's going on?



```
TimerClock = SystemClock / (Prescaler + 1)
```



```
PWM Frequency = TimerClock / (Period + 1)
```



💡 **Let's Take an Example**

Say your STM32 system clock is 16 MHz.

You choose:

- **Prescaler = 15 → TimerClock = 1 MHz**
- **Period = 999 → PWM Frequency = 1 kHz**
- **Pulse = 500 → 50% duty cycle**

This means:

- The pin toggles HIGH for 500 µs, LOW for 500 µs → 1 ms cycle