

General Purpose Input/Output pins

Understand about GPIO hardware, working modes and speeds, alternative function and input interrupts. Steps to control a GPIO using HAL. Example to blink LEDs and get input from a button.

#arm #stm32 #gpio

Last update: April 28, 2021

Table of Content

1. Hardware
 - 1.1. Voltage and Current
 - 1.2. Input mode
 - 1.3. Output mode
 - 1.4. Output Speed
 - 1.5. Bit atomic operation
 - 1.6. Input interrupt
 - 1.7. Alternate function
 - 1.8. Analog input/output
 - 1.9. Locking pin
2. Memory Map
3. Register Map
4. HAL Software
5. Lab: Toggle LEDs
 - 5.1. Create project
 - 5.2. Setup button's interrupt
 - 5.3. Generated code
 - 5.4. User code
 - 5.5. Build and Run
6. Appendix
 - 6.1. Bare-metal

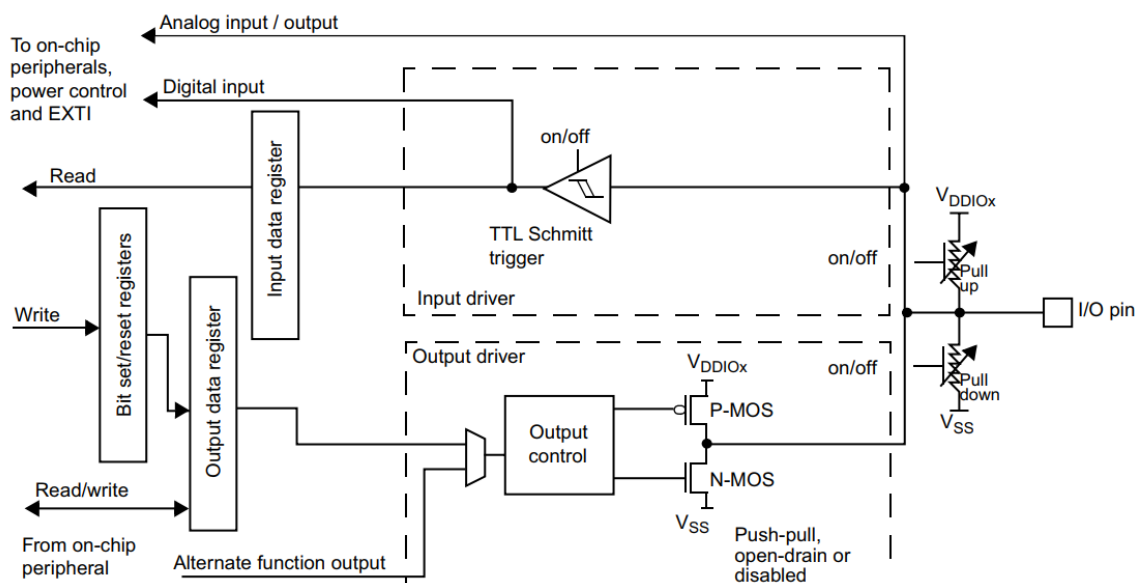
GPIO notes

- Enable clock source on GPIO port when use it
- APB2 bus speed determines the sampling rate of all GPIO inputs
- Can select mode, speed, alternative function on a GPIO pin
- Can have external interruption
- Can lock a GPIO after initializing
- Disconnect a GPIO pin by setting it into input floating mode
- Save power by setting GPIO pins to Analog mode ([Schmitt trigger](#) is disabled)

1. Hardware

Each GPIO Pin has a complex structure to function as both input and output:

- Protection Diodes
- [Pull-up and Pull-down](#) resistors on input
- [Schmitt trigger](#) to convert input to digital value
- [Open-Drain or Push-Pull](#) gate on output
- [Multiplexer](#) for Alternate Function
- Input and Output data registers
- Control registers



MS33182V2

A GPIO pin structure

1.1. Voltage and Current

Always assume that all GPIO pins are **NOT 5V tolerant by default** until find out in the datasheet (such as DS8668 for STM32F0x) that a specific pin is 5V tolerant, only then it can be used as a 5V pin.

The **maximum current** that could be sourced or sunk into any GPIO pin is **25mA** as mentioned in the datasheet.

1.2. Input mode

- Input Floating (Hi-Z)
- Input Pull-Up
- Input Pull-Down

Read about [Pull-Up/ Pull-Down](#)

When a GPIO pin is set to the input mode, the data present on the I/O pin is sampled into the Input Data Register (IDR) every APB2 clock cycle. This means the APB2 bus speed determines the input sampling speed for the GPIO pins.

1.3. Output mode

- Output Open-Drain
- Output Push-Pull

Read about [Open-Drain and Push-pull](#)

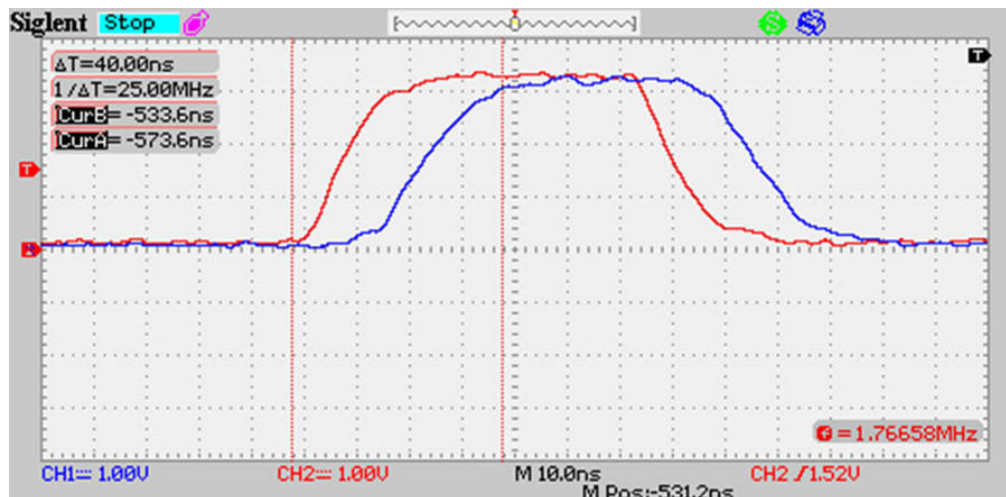
When a GPIO pin is set to the output mode, there is an option to configure the pin speed mode. Refer to datasheet (e.g. DS8668) to check the I/O AC characteristics table to note the maximum frequency in different conditions.

1.4. Output Speed

GPIO speed is not related to switching frequency, it defines the slew rate of a GPIO, that is how fast it goes from the 0V level to VDD one, and vice versa.

Below image shows the slew rate of 2 speed modes:

- Red line: high speed
- Blue line: low speed



Slew rate of 2 speed modes

1.5. Bit atomic operation

There is no need for the software to disable interrupts when programming the Output Data Register (ODR) at bit level. Use Bit Set/Reset Register (BSRR) to select individual bit operation.

1.6. Input interrupt

When in input mode, all ports have external interrupt capability. Read more about [Interrupt](#).

1.7. Alternate function

- Alternate Function Push-Pull
- Alternate Function Open-Drain

Pin can be used for an alternate function from a peripheral by setting the Alternate Function register (AF).

1.8. Analog input/output

In analog mode, pin is directly wired to a analog module (ADC, DAC)

1.9. Locking pin

The locking mechanism allows the IO configuration to be frozen. When the LOCK sequence has been applied on a port bit, it is no longer possible to modify the value of the port bit until the next reset.

2. Memory Map

Program memory, data memory, registers and I/O ports are organized within the same linear 4GB address space. **The bytes are coded in memory in Little Endian format.**

Refer to the document RM0091 for STM32F0xto see the memory map for GPIO ports:

Boundary Address	Size	Peripheral

0x4800 1400 - 0x4800 17FF	1KB	GPIOF
0x4800 1000 - 0x4800 13FF	1KB	GPIOE
0x4800 0C00 - 0x4800 0FFF	1KB	GPIOD
0x4800 0800 - 0x4800 0BFF	1KB	GPIOC
0x4800 0400 - 0x4800 07FF	1KB	GPIOB
0x4800 0000 - 0x4800 03FF	1KB	GPIOA

3. Register Map

Register	Offset	Description

GPIOx_MODER	0x00	I/O mode: 00: Input mode (reset state) 01: General purpose output mode 10: Alternate function mode 11: Analog mode
GPIOx_OTYPER	0x04	Output type: 0: Output push-pull (reset state) 1: Output open-drain
GPIOx_OSPEEDR	0x08	Output speed: x0: Low speed 01: Medium speed 11: High speed
GPIOx_PUPDR	0x0C	Pull-up/ pull-down 00: No pull-up, pull-down 01: Pull-up 10: Pull-down 11: Reserved
GPIOx_IDR	0x10	Input data
GPIOx_ODR	0x14	Output data
GPIOx_BSRR	0x18	Bit Set/Reset 0: No action on the corresponding ODRx bit 1: Set/Reset the corresponding ODRx bit
GPIOx_LCKR	0x1C	Configuration lock
GPIOx_AFRH	0x20	Alternate Function low register
GPIOx_AFRH	0x24	Alternate Function high register 0000: AF0 0001: AF1 0010: AF2 0011: AF3 0100: AF4

GPIOx_BRR	0x28	Bit	0101: AF5
			0110: AF6
			0111: AF7
			Reset
			0: No action on the corresponding ODx bit
			1: Reset the corresponding ODx bit

4. HAL Software

The Hardware Abstract Layer (HAL) is designed so that it abstracts from the specific peripheral memory mapping. But, it also provides a general and more user-friendly way to configure the peripheral, without forcing the programmers to know how to configure its registers in detail.

excerpt from [Description of STM32F0 HAL and low-layer drivers](#)

How to use GPIO HAL

1. Enable the GPIO AHB clock using the following function : `__HAL_RCC_GPIOx_CLK_ENABLE()` .
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()` .
 - Configure the IO mode using `Mode` member from `GPIO_InitTypeDef` structure
 - Analog mode is required when a pin is to be used as ADC channel or DAC output.
 - In case of external interrupt/event, select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
 - Activate Pull-up, Pull-down resistor using `Pull` member from `GPIO_InitTypeDef` structure.
 - In case of Output or alternate function mode selection: the speed is configured through `Speed` member from `GPIO_InitTypeDef` structure.
 - In alternate mode is selection, the alternate function connected to the IO is configured through `Alternate` member from `GPIO_InitTypeDef` structure.
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()` .
4. `HAL_GPIO_DeInit` allows to set register values to their reset value. It's also recommended to use it to unconfigure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in `EXTI` & `SYSCFG` registers.
5. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()` .
6. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()` or `HAL_GPIO_TogglePin()` .
7. To lock pin configuration until next reset use `HAL_GPIO_LockPin()` .

8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. The LSE oscillator pins **OSC32_IN** and **OSC32_OUT** can be used as general purpose (**PC14** and **PC15**, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
10. The HSE oscillator pins **OSC_IN** and **OSC_OUT** can be used as general purpose **PF0** and **PF1**, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

5. Lab: Toggle LEDs

This project aims to learn how to configure GPIO via STM32CubeIDE and STM32CubeMX.

Target board: STM32F0 Discovery

Application requirements:

- Turn on Green LED and Blue LED at startup
- In main loop, toggle Green LED every 500ms
- If user press User button, toggle the Blue LED

5.1. Create project

Create new project via CubeMX and setup GPIO for LEDs and button.

- Green Led is on PC9, select *GPIO_Output*
- Blue Led is on PC8, select *GPIO_Output*
- Button is on PA0, select *GPIO_EXTI0* /* do not select *GPIO_Input* */ to detect interruption

5.2. Setup button's interrupt

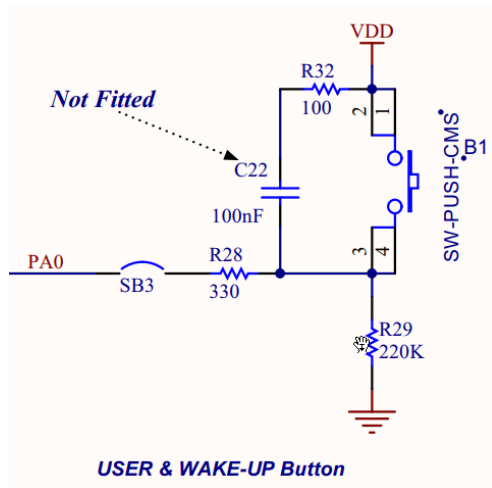
Check the board schematic to know how the button is wired. As seen below, the button is pulled to GND by default, and then connected to VDD when pressed. Therefore, to capture the action of pressing down the button, rising edge will be used to detect the transition.

Select **GPIO** in *Pinout and Configuration* tab, then click on PA0 pin config:

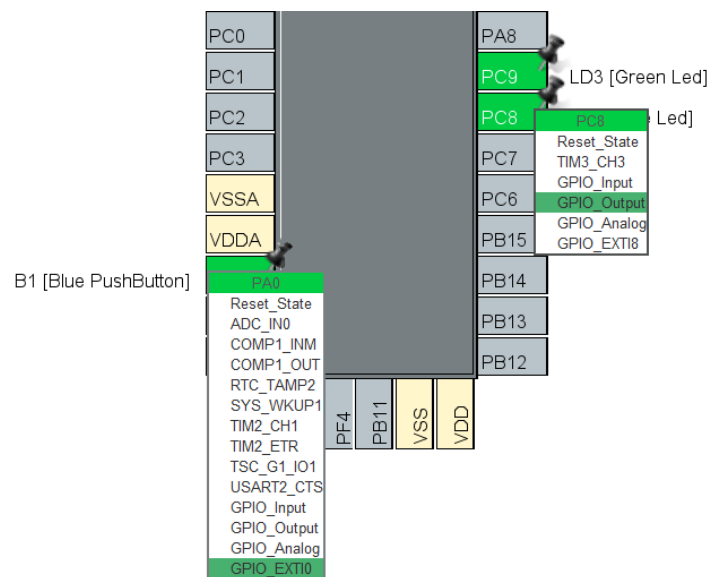
- GPIO Mode: External Interrupt Mode with Rising Edge trigger
- GPIO PU/PD: No

Move to **NVIC** tab:

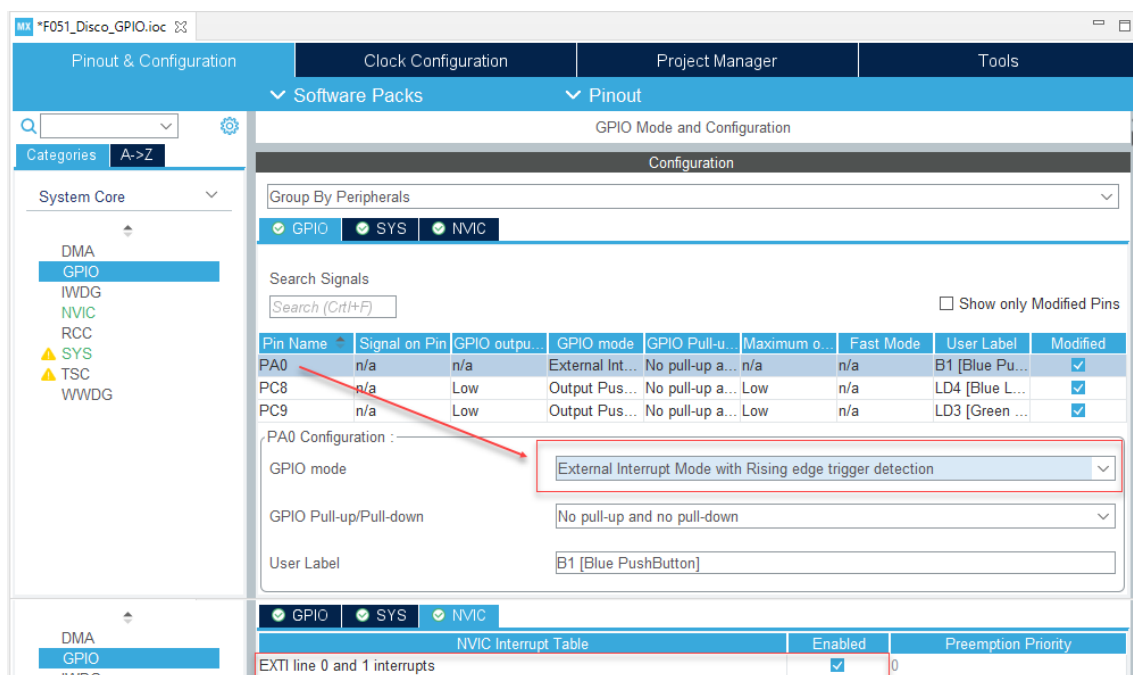
- Check on EXTI line 0 and 1 interrupts



Schematic of User button on pin PA0



Assign GPIO to Leds and button



Enable interrupt on button

5.3. Generated code

Generate code with `alt` + `k`.

Custom defines

Any custom name for a pin will be defined in *main.h*:

main.h

```
#define B1_Pin GPIO_PIN_0
#define B1_GPIO_Port GPIOA
#define LD4_Pin GPIO_PIN_8
#define LD4_GPIO_Port GPIOC
#define LD3_Pin GPIO_PIN_9
#define LD3_GPIO_Port GPIOC
```

Init functions

In the *main.c*, IDE generates `SystemClock_Config()` to setup system clocks, and `MX_GPIO_Init()` to initialize GPIOs.

main.c

```
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LD4_Pin|LD3_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin : B1_Pin */
    GPIO_InitStruct.Pin = B1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

    /*Configure GPIO pins : LD4_Pin LD3_Pin */
    GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    /* EXTI interrupt init*/
    HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
}
```

Interrupt handlers

The the override function of the EXTI0 interrupt handler is implemented in `_it.c` file:

stm32f0xx_it.c

```
void EXTI0_1_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}
```

When tracing the code of the function `HAL_GPIO_EXTI_IRQHandler()`, it will call to the callback function `HAL_GPIO_EXTI_Callback()` which should be overridden in *main.c*.

5.4. User code

Add some lines of code to implement the application requirements:

Turn on all leds at startup

Before main while loop:

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8 | GPIO_PIN_9, GPIO_PIN_SET);
```

Toggle the green led every 500ms

Inside the main while loop:

```
while (1)
{
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_9);
    HAL_Delay(500);
}
```

Toggle the blue led when press on button

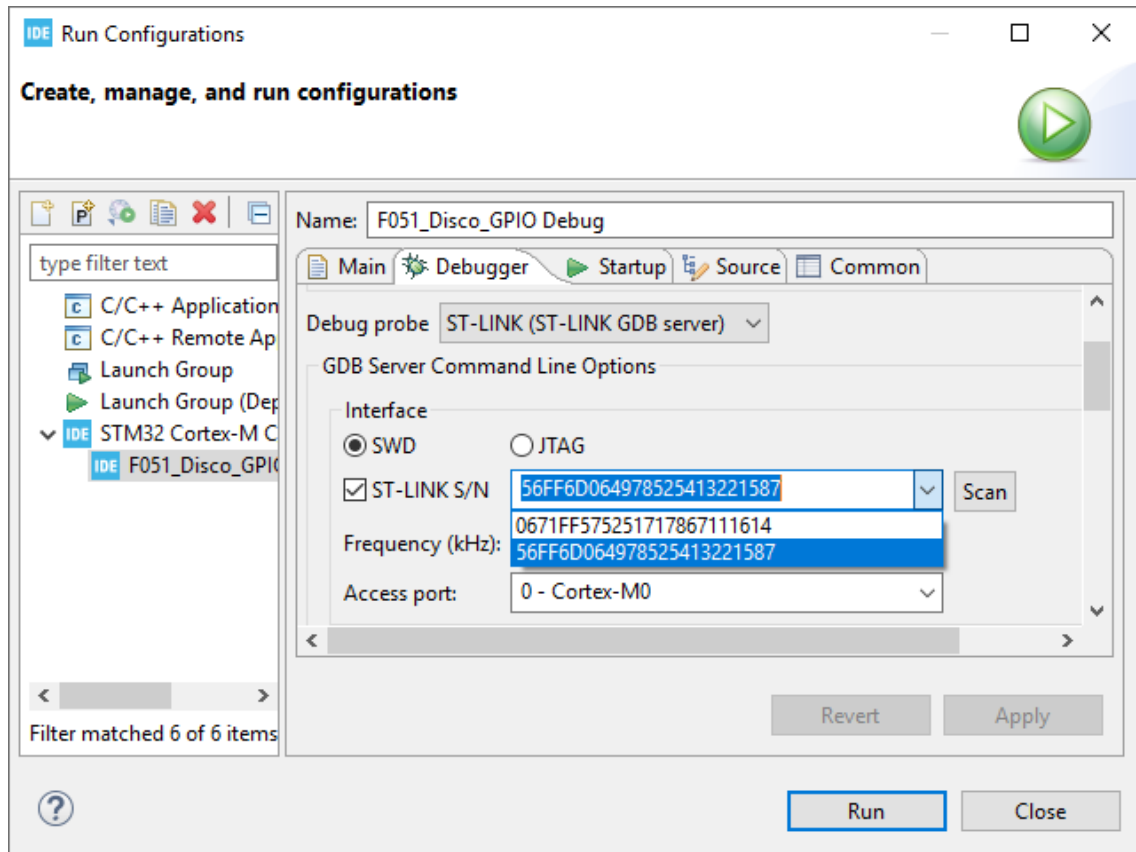
Override `HAL_GPIO_EXTI_Callback` and call to HAL function to toggle the pin state:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_PIN_0 == GPIO_Pin) {
        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);
    }
}
```

5.5. Build and Run

Press **ctrl + b** to build the project and run on the target board.

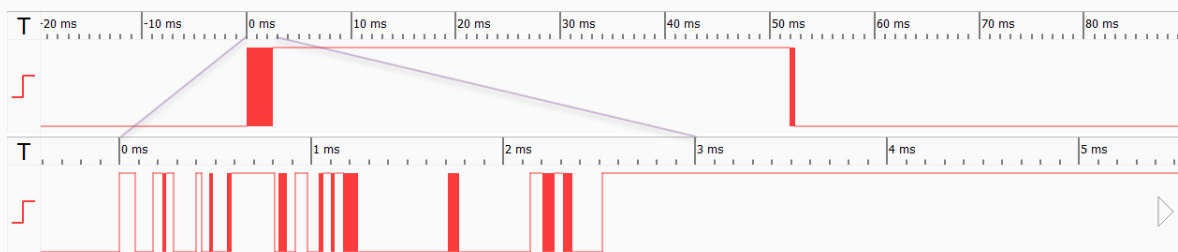
If having multiple boards connected, select the target board by selecting its ST-LINK Serial Number



Select target board

Bouncing input

Sometimes, the blue led does not toggle correctly. It toggles more than once. This happens because of input bouncing: the logic level is unstable during the transition.



Bouncing input on button

To eliminate it, debounce the input by additional hardware or internal timer.

6. Appendix

6.1. Bare-metal

Bare-metal means accessing to the registers directly to read or write value.

Here is an example to turn on PC9 pin /* GPIO C, pin 9 */:

```
#define GPIOC_BASE (0x48000800UL)
#define GPIOx_ODR 0x14
volatile uint32_t *GPIOC_ODR = (uint32_t *) (GPIOC_BASE + GPIOx_ODR);
*GPIOC_ODR |= (1<<9);
```

However, it should be better to use *struct* to manage a GPIO port. ST HAL library has defined `GPIO_TypeDef` struct as below:

```
typedef struct {
    volatile uint32_t MODER;
    volatile uint32_t OTYPER;
    volatile uint32_t OSPEEDR;
    volatile uint32_t PUPDR;
    volatile uint32_t IDR;
    volatile uint32_t ODR;
    volatile uint32_t BSRR;
    volatile uint32_t LCKR;
    volatile uint32_t AFR[2];
    volatile uint32_t BRR;
} GPIO_TypeDef;
```

Then, define a struct pointer pointing to the base address of the target GPIO port, and use it as a GPIO object:

```
#define GPIOC_BASE (0x48000800UL)
#define GPIOC ((GPIO_TypeDef *) GPIOC_BASE)
GPIOC->ODR |= (1 << 9);
```