General Purpose Input/Output pins

Input/Output pins are the gates to help microprocessor communicate with the outside world. There are some GPIO ports on a MCU, each pin can be configured to work in different modes, or to work together in a specific protocol. By using GPIO with different settings, a MCU can be configured to do many different application, and connect to a vast of external peripherals.

#arm #stm32 #gpio

Last update: 2021-06-09 22:48:22

Table of Content

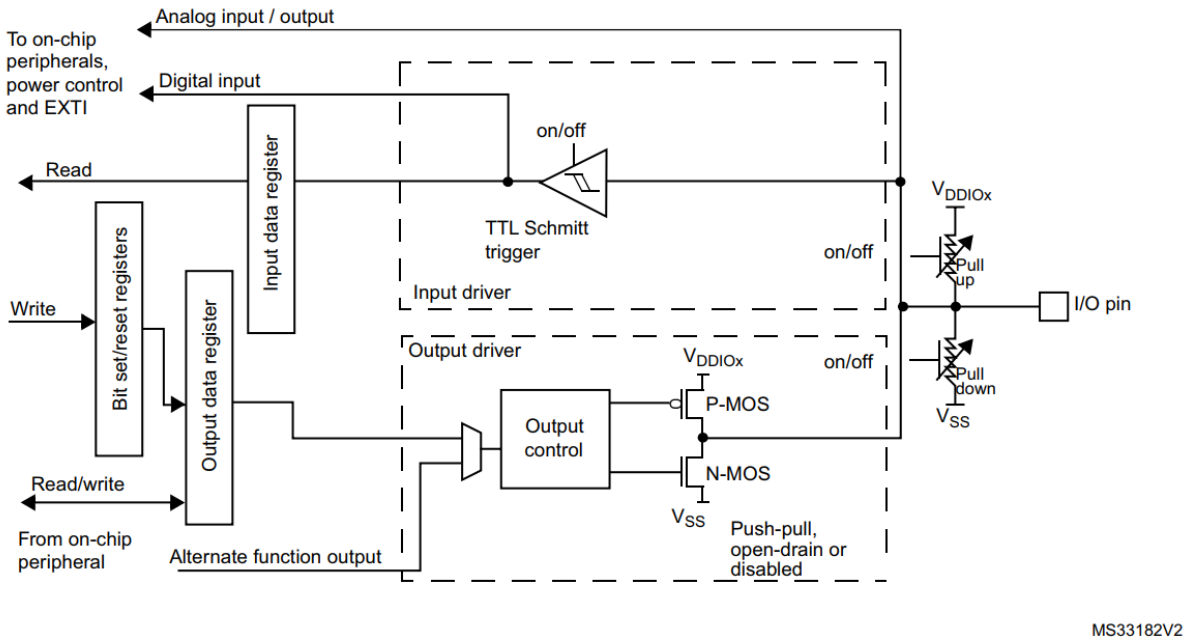
GPIO notes

- 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



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

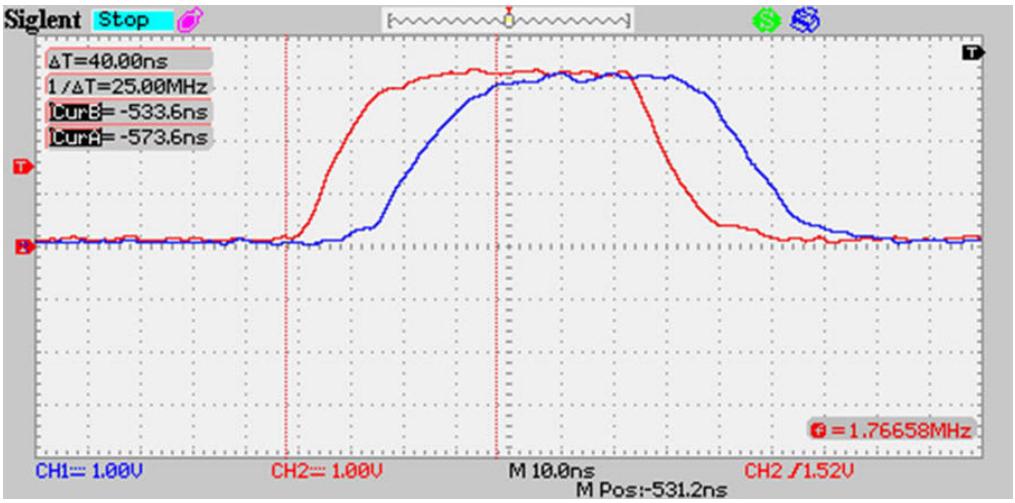
When a GPIO pin is set to the output mode, there is an option to configure the pin speed mode. Refer to datasheet (e.g. DS8668) to check the I/O AC characteristics table to note the maximum frequency in different conditions.

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. STM32Cube HAL Usage

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 now 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.

3. Lab 1: Blink Led

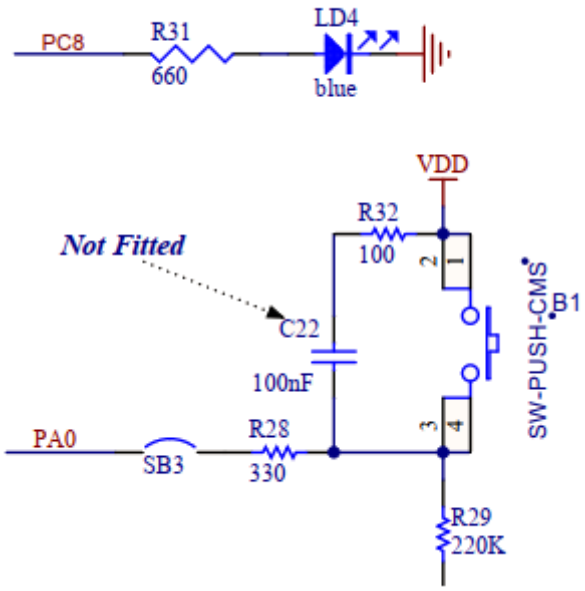
Requirement

- An LED and a Push button
- Blink the LED every 100ms when press and hold a button
- Blink the LED every 500ms when the button is released

Target board

Any board that has GPIOs connected to an LED and a button. In this tutorial, a STM32F0 Discovery board will be used, the schematic shows below connection:

STM32F051R8	Mode	External peripheral
PC8	Output Push Pull, No Pull-up and No Pull-down	Blue LED
PA0	Input, No Pull-up and No Pull-down	Push button, active High



The Blue LED and the Push button on STM32F0 Discovery board

3.1. Setup new project

Before starting to write code to blink the LED, there are some steps need to be done to set up the MCU. It is easy to do with support from CubeMX.

Select the MCU

The STM32F0 Discovery board has STM32F051R8 MCU.

Set up clocks

Under the Clock Configuration tab:

- PLL Source is **HSI**, with PLL Multiplier is *12*
- Set **HCLK** to *48 MHz*

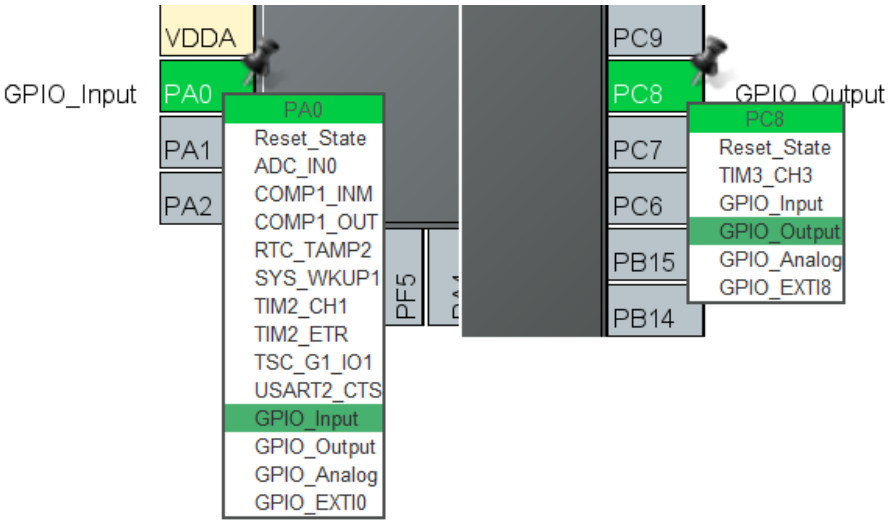
Set up debugger

To program code to the target MCU, go to the System Core under the Pinout & Configuration tab to select **SYS** module:

- Enable **Debug Serial Wire** This will automatically assign **PA14** to SWCLK and **PA13** to SWDIO, which are pins to communicate with debugger on SWD interface

3.2. Setup LED and Button

Under the Pinout View, Left-click on **PC8** and set as a *GPIO_Output*, on **PA0** and set as a *GPIO_Input*. Right-click on these pins to set new name for them, such as `LED` and `BUTTON`.



Set GPIO mode for pins

3.3. Generated source code

Save the CubeMX settings by `Ctrl + S`,and then press `Alt + K` to start generating source code.

Custom defines

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

main.h

```
#define BUTTON_Pin GPIO_PIN_0
#define BUTTON_GPIO_Port GPIOA
#define LED_Pin GPIO_PIN_8
#define LED_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(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : BUTTON_Pin */
  GPIO_InitStruct.Pin = BUTTON_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BUTTON_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LED_Pin */
  GPIO_InitStruct.Pin = LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);

}
```

3.4. User code

Add some lines of code to implement the application requirements in the main *while* loop. Note that, the button is active high, it means if the button is pressed, it pulls the input pin to a High logic level.

```
int main(void) {
  while (1) {
    if (HAL_GPIO_ReadPin(BUTTON_GPIO_Port, BUTTON_Pin) == GPIO_PIN_SET) {
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
      HAL_Delay(100);
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
      HAL_Delay(100);
    } else {
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
      HAL_Delay(500);
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
      HAL_Delay(500);
    }
  }
}
```

3.5. Download to the board

After compiling, download the firmware to the board and observe the LED in action with the button.

⚠ There is some delay between the blink pattern (500ms to 100ms), because in current source code, the delay function prevents MCU to react immediately to user's action.

4. Lab 2: Button interrupt

Requirement

- An LED and A Push button
- Press on the button to toggle and print out the number of raising edges

Target board

Any board that has GPIOs connected to an LED and a button. In this tutorial, a STM32F0 Discovery board with the connection being the same as the previous lab.

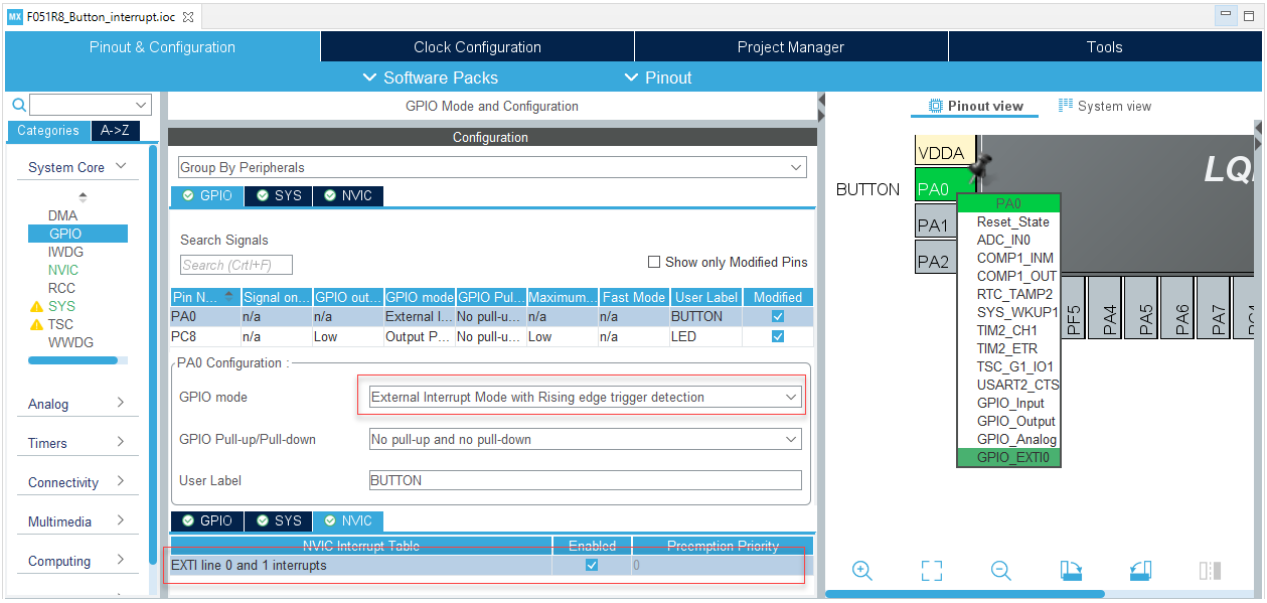
4.1. Setup new project

Start a new project in the same steps described in the previous section.

4.2. Setup external interrupt

To detect the raising edge, button has to be configured as an *External Interrupt Mode with Raising Edge trigger detection*.

To enable interrupt, under the NVIC tab, check on the *EXTI line 0 and line 1 interrupt* option.



Enable External Interrupt mode on the button pin

4.3. Generated code

Note that the generated function to setup GPIO has changed to configure Button pin to interrupt mode, and enable the external interrupt line. Read more in Interrupt.

```
static void MX_GPIO_Init(void) {
    ...
    /*Configure GPIO pin : BUTTON_Pin */
    GPIO_InitStruct.Pin = BUTTON_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(BUTTON_GPIO_Port, &GPIO_InitStruct);

    ...
    HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
}
```

In the file `stm32f0xx_it.c`, there is an implementation of the interrupt handler `EXTI0_1_IRQHandler()` which calls to HAL function `HAL_GPIO_EXTI_IRQHandler()` to clear the pending interrupt flag and finally transfer the work to user's application if there is an overridden function `HAL_GPIO_EXTI_Callback()`.

4.4. Handler interrupt

As mentioned above, the `HAL_GPIO_EXTI_Callback()` will be called to transfer the right to user's application to handle the interrupt.

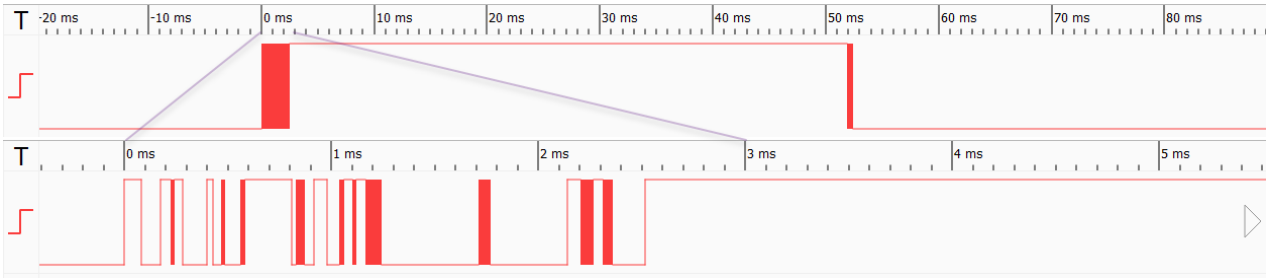
Firstly, whenever the rasing edge is detected, the callback will be called to increase a counter:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}
```

Inside the main while loop, just repeatedly delay in 1000ms.

```
int main() {
    while(1) {
        HAL_Delay(1000);
    }
}
```

Run the program, and slowly press the button, the LED sometimes does not toggle the state. Using a logic analyser to see that The logic level is unstable during the transition, it causes multiple rasing and falling edge before coming to stable. This is called *Bouncing input*. To eliminate it, debounce the input by additional hardware or an internal timer.



Bouncing input on button