



西安电子科技大学  
XIDIAN UNIVERSITY

XIDIAN UNIVERSTIY

2022 / EE205062

SOLUTIONS OF Travaux Pratiques

---

## Course: Microprocessor II

---

*Author*

KANG Gaalok

*Rev. Date*

June 15, 2022

*Professor*

Yoann CHARLON

UNIVERSITÉ CÔTE D'AZUR

YANG Xi

XIDIAN UNIVERSITY

March 24 – June 15

# Contents

<b>1</b>	<b>GPIO / LEDs</b>	<b>2</b>
1.1	Operate the Registers . . . . .	3
1.2	Use the Functions in CMSIS . . . . .	3
<b>2</b>	<b>Timer</b>	<b>5</b>
2.1	Operate the Registers . . . . .	5
2.2	Use the Functions in CMSIS . . . . .	5
<b>3</b>	<b>PWM</b>	<b>8</b>
<b>4</b>	<b>UART</b>	<b>11</b>
4.1	Interface and Parameters of UART . . . . .	11
4.2	Use the Functions in CMSIS . . . . .	11
4.3	Verify on Board . . . . .	14
<b>5</b>	<b>NVIC</b>	<b>15</b>
5.1	NVIC . . . . .	15
5.2	Use the Functions in CMSIS . . . . .	16

# 1 GPIO / LEDs

In this section, we will use mainly 2 methods to light the LEDs through the GPIO port:

- by operating the registers directly;
- by using the *standard drivers library* CMSIS.

By referring to the technical manual<sup>1</sup>, we know that there are 4 LEDs we can program,

- LD3: orange LED, connected to the I/O PD13;
- LD4: green LED, connected to the I/O PD12;
- LD5: red LED, connected to the I/O PD14;
- LD6: blue LED, connected to the I/O PD15.

We will take LD4 as an example.

For the GPIO, it have 3 output type: pull-up, pull-down and float.

The pull-up/down type to ensure that if there is nothing connected to the pin and your program reads the state of the pin, will it be high (pulled to VCC, pull-up) or low (pulled to ground, pull-down).

In the internal circuit, these types are controlled by a resistor. A low resistor value is called a strong pull-up (more current flows), a high resistor value is called a weak pull-up (less current flows).

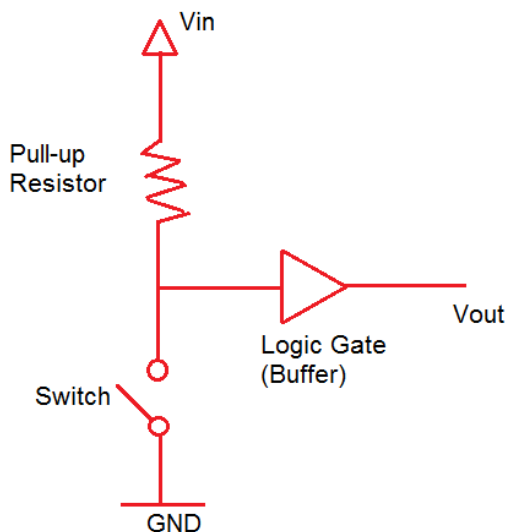


Figure 1: Pull-up Resistor Schematic

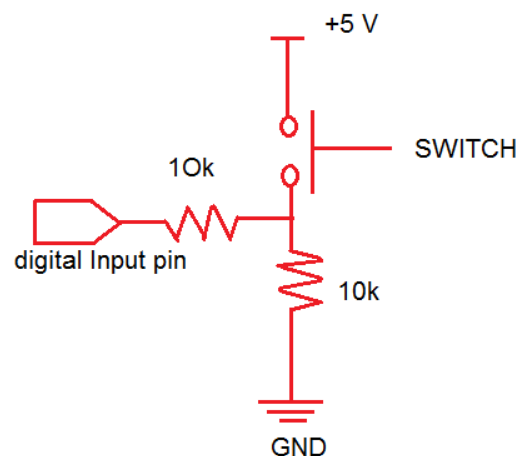


Figure 2: Pull-down Resistor Schematic

<sup>1</sup>UM1472, page 18

## 1.1 Operate the Registers

We mainly refer the Reference Manual<sup>2</sup>.

To **initialize** the LD4, we need to configure the registers :

- RCC\_AHB1ENR bit 3 SET 1, to active clock of port D;
- GPIOD\_MODER bit [25,24] SET [01], to configure pin 12 *Mode* as **General purpose output mode**;
- GPIOD\_OTYPER bit 12 SET 0, to configure pin 12 *OutputType* as **Output push-pull**;
- GPIOD\_OSPEEDR bit [25,24] SET [10], to configure pin 12 *OutputSpeed* as **High speed**;
- GPIOD\_PUPDR bit [25,24] SET [10], to configure pin 12 *Pull-up/down* as **Pull-down**.

To SET or CLEAR 1 bit on bit x, we use

```
1 Register |= (1 << x); //SET
2 Register &= ~(1 << x); //CLEAR
```

**Attention** If we SET multi-bits on bit [y...x], we must **CLEAR** bit [y...x] first!

To **light** the LED LD4 on or off, we configure the register

- GPIOD\_BSRR bit 12 (BS12) SET 1, to SET pin 12 and light LD4 on;
- GPIOD\_BSRR bit 28 (BR12) SET 1, to CLEAR pin 12 and light LD4 off.

**Attention** If both BSx and BRx are set, **BSx has priority**!

## 1.2 Use the Functions in CMSIS

**Attention** To ensure that we can use the drivers library, we need to add some files to our project.

We use these functions:

- LED\_Initialize();
- LED\_On();
- LED\_Off().

```
1 /*-----
2  * LED_Initialize:  Initialize LEDs
3  * Parameters:  (none)
4  * Return:      (none)
5  *-----*/
6 void LED_Initialize (void);
7
8 /*-----
```

---

<sup>2</sup>RM0090

```
9  * LED_On: Turns on requested LED
10 * Parameters:  num - LED number
11 * Return:      (none)
12 *-----*/
13 void LED_On (uint32_t num);
14
15 /*-----
16 * LED_Off: Turns off requested LED
17 * Parameters:  num - LED number
18 * Return:      (none)
19 *-----*/
20 void LED_Off (uint32_t num);
```

## 2 Timer

In this section, we blink the LEDs with the frequency = 1Hz (0.5s for light on and 0.5s for light off). We need to find a clock frequency = 2Hz and change the LED state in a periode,

We use TIM3 which the clock frequency = 84MHz. To set the new frequency = 2Hz, we need to divide it by 42M. It means that the interruption of TIM3 will be active automatically every 0.5 second.

Set prescaler PSC = (10k-1) and counter periode ARR = (4200-1). Then we can get this new frequency.

$$f_{new} = \frac{f_{CLK}}{(PSC + 1)(ARR + 1)}$$

### 2.1 Operate the Registers

To **initialize** the TIM3, we need to configure the registers :

- RCC\_APB1ENR bit 1 SET 1, to active the clock of TIM3;
- TIM3\_CR1 SET 0;
- TIM3\_PSC SET (10 000 - 1) and TIM3\_ARR SET (4200 - 1), to configure the scalar of the frequency;
- TIM3\_DIER bit 0 SET 1, to configure the update interruption mode;
- TIM3\_CR1 bit 0 SET 1, to enable all the settings.

To **initialize** the Interruption, we need to configure the registers :

- NVIC\_ISER[0] bit 29 SET 1.

Then, in the IRQ of the NVIC, first, we need to check whether it is in the interruption. Use TIM3\_SR bit 0 as a flag, if it is **1**, then in the interruption. After we change the state of the LED, we CLEAR the flag by SETTING TIM3\_SR bit 0 to 0.

**Attention** The function name of TIM3's IRQ is TIM3\_IRQHandler!

**Attention** In the IRQ, we **can't CLEAR** the flag of interruption first!

### 2.2 Use the Functions in CMSIS

To **initialize** the TIM3, we use these functions:

```

1 /**
2  * @brief Enables or disables the Low Speed APB (APB1) peripheral clock.
3  * @param RCC_APB1Periph: specifies the APB1 peripheral to gates its
4  *                        clock.
5  *                        This parameter can be any combination of the following values:
6  *                        @arg RCC_APB1Periph_TIM3: TIM3 clock
7  *                        ... ..

```

```

7  * @param  NewState: new state of the specified peripheral clock.
8  *          This parameter can be: ENABLE or DISABLE.
9  * @retval None
10 */
11 void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState
    NewState);
12
13 typedef struct
14 {
15     uint16_t TIM_Prescaler;
16     uint16_t TIM_CounterMode;
17     uint32_t TIM_Period;
18     uint16_t TIM_ClockDivision;
19     uint8_t TIM_RepetitionCounter;
20 } TIM_TimeBaseInitTypeDef;
21
22 /**
23 * @brief  Initializes the TIMx Time Base Unit peripheral according to
24 *          the specified parameters in the TIM_TimeBaseInitStruct.
25 * @param  TIMx: where x can be 1 to 14 to select the TIM peripheral.
26 * @param  TIM_TimeBaseInitStruct:
27 *          pointer to a TIM_TimeBaseInitTypeDef structure that contains
28 *          the configuration information for the specified TIM peripheral.
29 * @retval None
30 */
31 void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef*
    TIM_TimeBaseInitStruct);
32
33 /**
34 * @brief  Enables or disables the specified TIM interrupts.
35 * @param  TIMx: where x can be 1 to 14 to select the TIMx peripheral.
36 * @param  TIM_IT: specifies the TIM interrupts sources to be enabled or
    disabled.
37 *          This parameter can be any combination of the following values:
38 *          @arg TIM_IT_Update: TIM update Interrupt source
39 *          ... ..
40 * @param  NewState: new state of the TIM interrupts.
41 *          This parameter can be: ENABLE or DISABLE.
42 * @retval None
43 */
44 void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState
    NewState);
45
46 /**
47 * @brief  Enables or disables the specified TIM peripheral.
48 * @param  TIMx: where x can be 1 to 14 to select the TIMx peripheral.
49 * @param  NewState: new state of the TIMx peripheral.
50 *          This parameter can be: ENABLE or DISABLE.

```

```

51 * @retval None
52 */
53 void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState);

```

To **initialize** the NVIC, we use this function:

```

1 void NVIC_EnableIRQ (IRQn_Type IRQn);

```

To check the interruption states, we use these functions:

```

1 /**
2 * @brief Checks whether the specified TIM flag is set or not.
3 * @param TIMx: where x can be 1 to 14 to select the TIM peripheral.
4 * @param TIM_FLAG: specifies the flag to check.
5 *           This parameter can be one of the following values:
6 *           @arg TIM_FLAG_Update: TIM update Flag
7 *           @arg TIM_FLAG_CC1: TIM Capture Compare 1 Flag
8 *           @arg TIM_FLAG_CC2: TIM Capture Compare 2 Flag
9 *           @arg TIM_FLAG_CC3: TIM Capture Compare 3 Flag
10 *          @arg TIM_FLAG_CC4: TIM Capture Compare 4 Flag
11 *          @arg TIM_FLAG_COM: TIM Commutation Flag
12 *          @arg TIM_FLAG_Trigger: TIM Trigger Flag
13 *          @arg TIM_FLAG_Break: TIM Break Flag
14 *          @arg TIM_FLAG_CC1OF: TIM Capture Compare 1 over capture Flag
15 *          @arg TIM_FLAG_CC2OF: TIM Capture Compare 2 over capture Flag
16 *          @arg TIM_FLAG_CC3OF: TIM Capture Compare 3 over capture Flag
17 *          @arg TIM_FLAG_CC4OF: TIM Capture Compare 4 over capture Flag
18 * @retval The new state of TIM_FLAG (SET or RESET).
19 */
20 FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, uint16_t TIM_FLAG);
21
22 /**
23 * @brief Clears the TIMx's pending flags.
24 * @param TIMx: where x can be 1 to 14 to select the TIM peripheral.
25 * @param TIM_FLAG: specifies the flag bit to clear.
26 * @retval None
27 */
28 void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG);

```

**Attention** Do not forget to **INITIALIZE** these peripherals in funtion main!



### 3 PWM

In this section, we will know how to calculate the parameter of the PWM.

PWM is one of the application of TIM. Thus, to initialize the PWM, we must initialize TIM first. The functions we have already know. We take TIM4 as an example.

```

1 void TIM4_Initialize(void){
2     TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructtrue;
3     // ENABLE RCC_APB1 for TIM4
4     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);
5     // PCLK/(PSC+1) = 84MHz/1 = 84MHz
6     TIM_TimeBaseInitStructtrue.TIM_Prescaler=0;
7     // 84MHz/(ARR+1) = 84MHz/8400 = 10kHz
8     TIM_TimeBaseInitStructtrue.TIM_Period=8399;
9     TIM_TimeBaseInitStructtrue.TIM_CounterMode=TIM_CounterMode_Up;
10    // default: DIV2
11    TIM_TimeBaseInitStructtrue.TIM_ClockDivision=TIM_CKD_DIV1;
12    // SET TIM4 PARAM
13    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseInitStructtrue);
14 }
```

For the CounterMode, there are mainly 2 kinds of mode: Counter-aligned mode and Edge-aligned mode. More detailed can be found in [Timing control and PWM](#).

**Attention** The CenterAlignedMode1 will use the driver frequency **twice** than CenterAligned-Mode3 if they realize the same frequency. And it will **delay a quarter of a periode**.

To **initialize** the PWM, to calculate the parameter of Pluse, we use

$$Pulse = \frac{(ARR + 1) \times DutyCycle}{100} - 1$$

and then we will use these functions:

```

1 typedef struct
2 {
3     uint16_t TIM_OCMode;
4     uint16_t TIM_OutputState;    // Normally we SET TIM_OutputState_Enable
5     uint16_t TIM_OutputNState;
6     uint32_t TIM_Pulse;
7     uint16_t TIM_OCPolarity;    // Normally we SET TIM_OCPolarity_High
8     uint16_t TIM_OCNPolarity;
9     uint16_t TIM_OCIdleState;
10    uint16_t TIM_OCNIdleState;
11 } TIM_OCInitTypeDef;
12
13 /**
14  * @brief  Initializes the TIMx Channel1 according to the specified
15  *         parameters in
16  *         the TIM_OCInitStruct.
```

```

16 * @param TIMx: where x can be 1 to 14 except 6 and 7, to select the TIM
    peripheral.
17 * @param TIM_OCInitStruct: pointer to a TIM_OCInitTypeDef structure that
    contains
18 *         the configuration information for the specified TIM peripheral.
19 * @retval None
20 */
21 void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct);
22
23 /**
24 * @brief Enables or disables the TIMx peripheral Preload register on
    CCR1.
25 * @param TIMx: where x can be 1 to 14 except 6 and 7, to select the TIM
    peripheral.
26 * @param TIM_OCPreload: new state of the TIMx peripheral Preload
    register
27 *         This parameter can be one of the following values:
28 *             @arg TIM_OCPreload_Enable
29 *             @arg TIM_OCPreload_Disable
30 * @retval None
31 */
32 void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, uint16_t TIM_OCPreload);

```

Finally, we configure the GPIO.

```

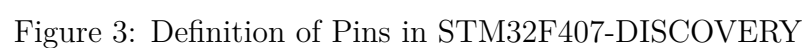
1 void GPIO_AF_Initialize(void){
2     // ENABLE RCC AHB1
3     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD,ENABLE);
4     // ENABLE mode alternate function / TIM4
5     GPIO_PinAF(GPIOD,12,GPIO_AF_TIM4);

6     // CONFIG mode: Alternate Function, Output Push Pull, Output
    Speed 100MHz, et no Pull up / down
7     GPIO_PinConfigure(GPIOD,12, GPIO_MODE_AF,
8                       GPIO_OUTPUT_PUSH_PULL,
9                       GPIO_OUTPUT_SPEED_100MHz,
10                      GPIO_NO_PULL_UP_DOWN);
11 }

```

**Attention** Do not forget to **INITIALIZE** these peripherals in funtion main!

**Attention** If you want to verify the second problem **1.3 Application à la commande d'un moteur pas à pas**, you can configure the states Q1,Q2,Q3,Q4 to LEDs. And if you utilize TIM5, you must use two jump wires to connect the pins of TIM5 to two of the LEDs. Because the pins of TIM5 cannot configure to LEDs directly. The definition of the pins is shown as Table 1.

Table 1: Definition of Pins

## 4 UART

In this section, we will try to know a common communication protocol: **Serial Port**, also named **USART** (Universal Synchronous/Asynchronous Receiver-Transmitter). We take **UART** as an example.

### 4.1 Interface and Parameters of UART

The two signals of each UART<sup>3</sup> device are named:

- Transmitter (Tx);
- Receiver (Rx).

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication. The transmitting UART is connected to a controlling data bus that sends data in a parallel form. From this, the data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving UART. This, in turn, will convert the serial data into parallel for the receiving device.

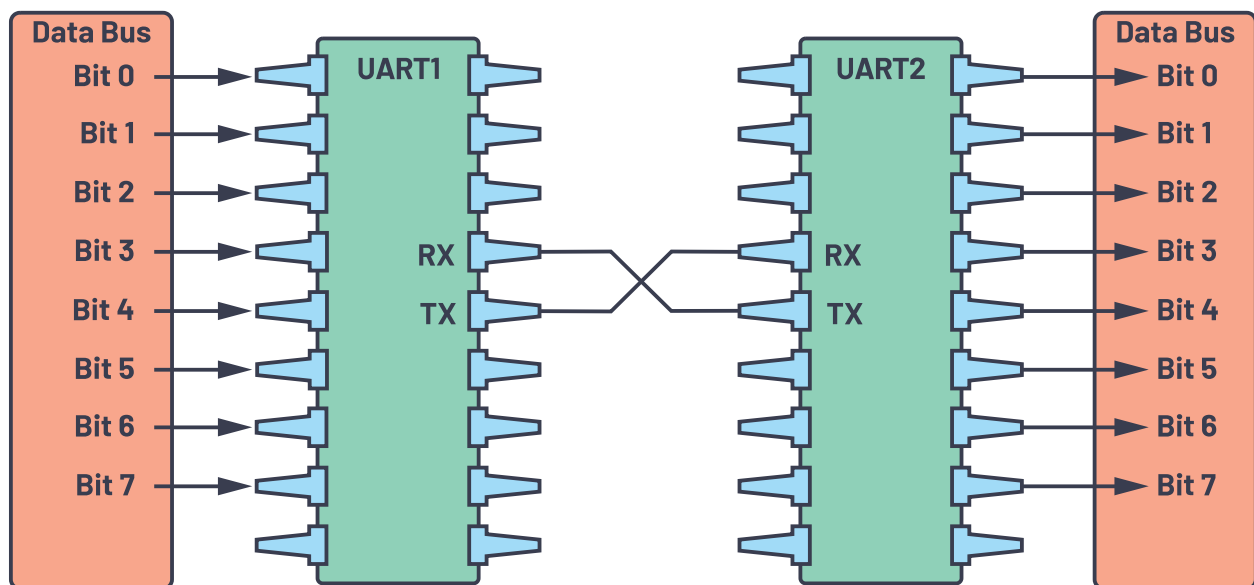


Figure 4: UART with data bus.

And there are several parameters of a UART in Table 2.

### 4.2 Use the Functions in CMSIS

First we need to **Initialize** the USART3. We will use these functions.

---

<sup>3</sup>[More Detailed Knowledge](#)

Parameters	Options
BaudRate	9600,115200,...
StartBit	1
WordLength	5 to 9
ParityBit	0, 1
StopBit	1, 2
...	...

Table 2: Parameters of A UART

```

1  /**
2  * @brief  Enables or disables the Low Speed APB (APB1) peripheral clock.
3  * @param  RCC_APB1Periph: specifies the APB1 peripheral to gates its
4  *          clock.
5  *          This parameter can be any combination of the following values:
6  *          @arg RCC_APB1Periph_USART3: USART3 clock
7  *          ... ..
8  * @param  NewState: new state of the specified peripheral clock.
9  *          This parameter can be: ENABLE or DISABLE.
10 * @retval None
11 */
12 void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState
13                               NewState);
14
15 typedef struct
16 {
17     uint32_t  USART_BaudRate;
18     uint16_t  USART_WordLength;
19     uint16_t  USART_StopBits;
20     uint16_t  USART_Parity;
21     uint16_t  USART_Mode;
22     uint16_t  USART_HardwareFlowControl;
23 } USART_InitTypeDef;
24
25 /**
26 * @brief  Initializes the USARTx peripheral according to the specified
27 *          parameters in the USART_InitStruct .
28 * @param  USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the
29 *          USART or UART peripheral.
30 * @param  USART_InitStruct: pointer to a USART_InitTypeDef structure that
31 *          contains the configuration information for the specified USART
32 *          peripheral.
33 * @retval None
34 */
35 void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef*
36                 USART_InitStruct);

```

```

34
35 /**
36 * @brief Enables or disables the specified USART peripheral.
37 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the
38 *           USART or UART peripheral.
39 * @param NewState: new state of the USARTx peripheral.
40 *           This parameter can be: ENABLE or DISABLE.
41 * @retval None
42 */
43 void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);

```

Because the ports of USART3 (Tx,Rx) are connected to the GPIO, we need to **Initialize** the GPIO, and set the mode of GPIO to the Alternate Function of USART3. These steps we have already known.

**Attention** For the STM32F407 DISC Board, the pins of USART3 are setted: Tx->PB10, Rx->PB11.

As for the transmit and receive data:

```

1 /**
2 * @brief Transmits single data through the USARTx peripheral.
3 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the
4 *           USART or UART peripheral.
5 * @param Data: the data to transmit.
6 * @retval None
7 */
8 void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
9
10 /**
11 * @brief Returns the most recent received data by the USARTx peripheral.
12 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the
13 *           USART or UART peripheral.
14 * @retval The received data.
15 */
16 uint16_t USART_ReceiveData(USART_TypeDef* USARTx)

```

**Attention** These functions are used to transmit or receive **A** data.

**Attention** When we transmit and receive data, we must check the status and do the "hold on" function.

```

1 /**
2 * @brief Checks whether the specified USART flag is set or not.
3 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the
4 *           USART or UART peripheral.
5 * @param USART_FLAG: specifies the flag to check.
6 *           This parameter can be one of the following values:
7 *           @arg USART_FLAG_TXE: Transmit data register empty flag
8 *           @arg USART_FLAG_TC: Transmission Complete flag

```

```

9  *           @arg USART_FLAG_RXNE: Receive data register not empty flag
10 * @retval The new state of USART_FLAG (SET or RESET).
11 */
12 FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG
    );

```

**Attention** Do not forget to **INITIALIZE** these peripherals in function main!

### 4.3 Verify on Board

To test the program, you need to use a **CONVERTER**. And maybe you have to **INSTALL** a **DRIVER** of CH341/CH340.

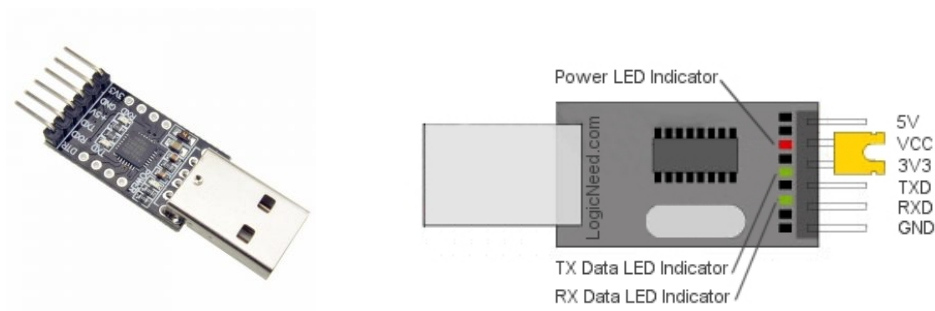


Figure 5: Converter

**Attention** The connection between the board and the converter is shown as Table 3.

Board PIN	Converter PIN
Tx(PB10)	Rx
Rx(PB11)	Tx
GND	GND

Table 3: Connection

## 5 NVIC

In this section, we will try to use button to control the LEDs. We will use two different interrupts, one is for the button and another is for the LEDs(control the period). Therefore, firstly we need to know NVIC.

### 5.1 NVIC

**Nested vectored interrupt controller**<sup>4</sup>, also called NVIC, is used to control and set the priority of the interrupts, and it also receives interrupt requests from ports through an additional hardware called **External interrupt/event Controller**(EXTI).

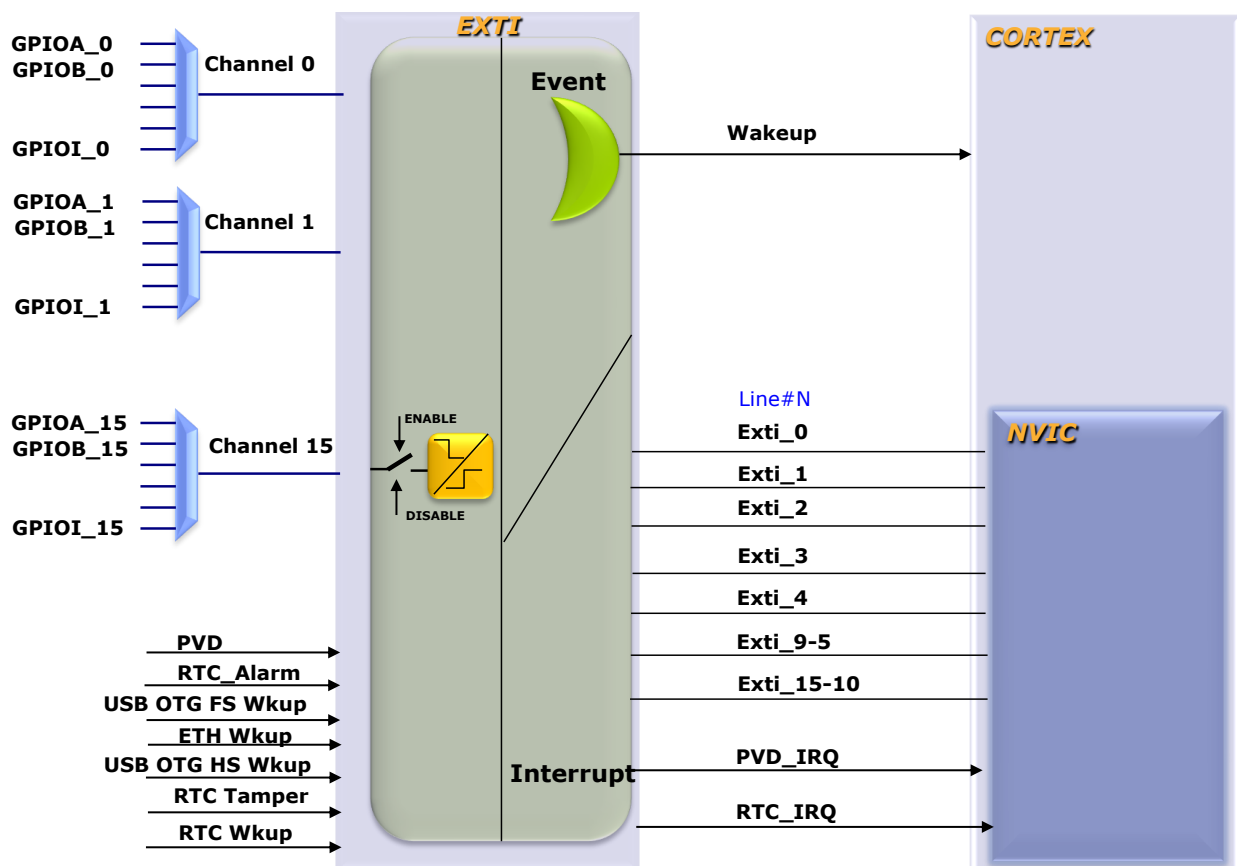


Figure 6: The chain involved in the processing of interrupt requests

In order to utilize the interrupt capabilities of the microcontroller the following must be fulfilled:

- The interrupt function must be prepared;
- The interrupt controller NVIC must be initialized and enabled;
- The controller EXTI must be initialized and enabled to sort-out the signals at ports and interpret them as interrupt requests.

<sup>4</sup>More information can be found in [STM32 Interrupts Tutorial](#).



Once the controller NVIC receives the interrupt request it validates it and notifies the processor. The processor then requests the code of the interrupt request source and uses it to find the pointer to the corresponding interrupt function from the interrupt vector table.

**Attention** You can not use two pins on one line simultaneously, for example:

- PA0 and PB0 and PC0 and so on, are connected to Line0, so you can use only one pin at one time to handle interrupt;
- PA0 and PA5 are connected to different lines, they can be used at the same time.

## 5.2 Use the Functions in CMSIS

Firstly, write the function LED\_shift with the help of TIM4 to set the period to be 0.5 second. You can simply consider to finish two actions in the same time (one period):

- light off this LED;
- light on the next LED.

Next, **initialize** the EXTI0, and **configure** the GPIO.

**Attention** User Button(the **BLUE** one)->PA0!

```

1  /**
2  * @brief  Enables or disables the peripheral clock.
3  * @param  RCC_AxBxPeriph: specifies the AxBx peripheral to gates its
4  *                      clock.
5  *          This parameter can be any combination of the following values:
6  *          @arg RCC_APB2Periph_SYSCFG: SYSCFG clock
7  *          @arg RCC_AHB1Periph_GPIOx:  GPIOx clock
8  * @param  NewState: new state of the specified peripheral clock.
9  *          This parameter can be: ENABLE or DISABLE.
10 * @retval None
11 */
12 void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState
    NewState);
13 void RCC_AHB1PeriphClockCmd(uint32_t RCC_AHB1Periph, FunctionalState
    NewState);
14 /**
15 \brief      Configure port pin
16 \param[in]  GPIOx      Pointer to GPIO peripheral
17 \param[in]  num         Port pin number
18 \param[in]  mode        \ref GPIO_MODE
19 \param[in]  output_type  \ref GPIO_OUTPUT_TYPE
20 \param[in]  output_speed \ref GPIO_OUTPUT_SPEED
21 \param[in]  pull_up_down \ref GPIO_PULL_UP_DOWN
22 \return     true  - success
23             false - error
24 */

```

```

25 extern bool GPIO_PinConfigure(GPIO_TypeDef *GPIOx,
26                               uint32_t      num,
27                               GPIO_MODE      mode,
28                               GPIO_OUTPUT_TYPE output_type,
29                               GPIO_OUTPUT_SPEED output_speed,
30                               GPIO_PULL_UP_DOWN pull_up_down);
31
32 /**
33  * @brief Selects the GPIO pin used as EXTI Line.
34  * @param EXTI_PortSourceGPIOx : selects the GPIO port to be used as
35  * source for EXTI lines where x can be (A..I) for
36  * STM32F405xx/407xx.
37  *
38  * @param EXTI_PinSourcex: specifies the EXTI line to be configured.
39  * This parameter can be EXTI_PinSourcex where x can be (0..15),
40  * except for EXTI_PortSourceGPIOI x can be (0..11) for
41  * ST32F405xx/407xx devices.
42  *
43  * @retval None
44  */
45 void SYSCFG_EXTILineConfig(uint8_t EXTI_PortSourceGPIOx, uint8_t
    EXTI_PinSourcex);
46
47 /**
48  \brief Configure EXTI line
49  \param[in] line Line number
50  \param[in] mode \ref EXTI_MODE
51                EXTI_MODE_INTERRUPT
52                EXTI_MODE_EVENT
53  \param[in] trigger \ref EXTI_TRIGGER
54                  EXTI_TRIGGER_NONE
55                  EXTI_TRIGGER_RISING
56                  EXTI_TRIGGER_FALLING
57                  EXTI_TRIGGER_RISING_FALLING
58  */
59 void EXTI_ConfigureLine (uint32_t line ,
60                          EXTI_MODE mode,
61                          EXTI_TRIGGER trigger);

```

And to **initialize** the NVIC, we need SET the interrupt of both EXTI0\_IRQn and TIM4\_IRQn, while EXTI0\_IRQn has a **higher priority**.

```

1 typedef struct
2 {
3     uint8_t NVIC_IRQChannel;
4     /*!< Specifies the IRQ channel to be enabled or disabled.
5     This parameter can be an enumerator of @ref IRQn_Type
6     enumeration (For the complete STM32 Devices IRQ Channels

```

```

7      list, please refer to stm32f4xx.h file) */
8
9      uint8_t NVIC_IRQChannelPreemptionPriority;
10     /*!< Specifies the pre-emption priority for the IRQ channel
11     specified in NVIC_IRQChannel. This parameter can be a value
12     between 0 and 15 as described in the table @ref
13         MISC_NVIC_Priority_Table
14     A lower priority value indicates a higher priority */
15
16     uint8_t NVIC_IRQChannelSubPriority;
17     /*!< Specifies the subpriority level for the IRQ channel specified
18     in NVIC_IRQChannel. This parameter can be a value
19     between 0 and 15 as described in the table @ref
20         MISC_NVIC_Priority_Table
21     A lower priority value indicates a higher priority */
22
23     FunctionalState NVIC_IRQChannelCmd;
24     /*!< Specifies whether the IRQ channel defined in NVIC_IRQChannel
25     will be enabled or disabled.
26     This parameter can be set either to ENABLE or DISABLE */
27 } NVIC_InitTypeDef;
28
29 /**
30 * @brief   Initializes the NVIC peripheral according to the specified
31 *          parameters in the NVIC_InitStruct.
32 * @param   NVIC_InitStruct: pointer to a NVIC_InitTypeDef structure that
33 *          contains the configuration information for the specified NVIC
34 *          peripheral.
35 * @retval  None
36 */
37 void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct);
38
39 void NVIC_EnableIRQ (IRQn_Type IRQn);

```

In `EXTI0_IRQHandler`, we detect the button and in `TIM4_IRQHandler` change the direction to clockwise/counterclockwise by modifying the function `LED_shift`.

**Attention** Do not forget to **INITIALIZE** these peripherals in function `main`!