Table of Contents

1. STM32 Control of built-in LEDs with buttons 2
1.1. Prerequisites 2
1.2. Background 2
1.2.1. LED connection 2
1.2.2. Buttons connection 3
1.2.3. Interrupts 3
1.2.4. Tasks to be implemented 4
1.2.5. Task 1 4
1.2.6. Task 2 4
1.2.7. Task 3 4
1.2.8. Task 4 5
1.2.9. Result validation 5
1.3. FAQ 5

1. STM32 Control of built-in LEDs with buttons

This exercise presents how to control three LEDs with buttons mounted on the Nucleo STM32WB55 board.

1.1. Prerequisites

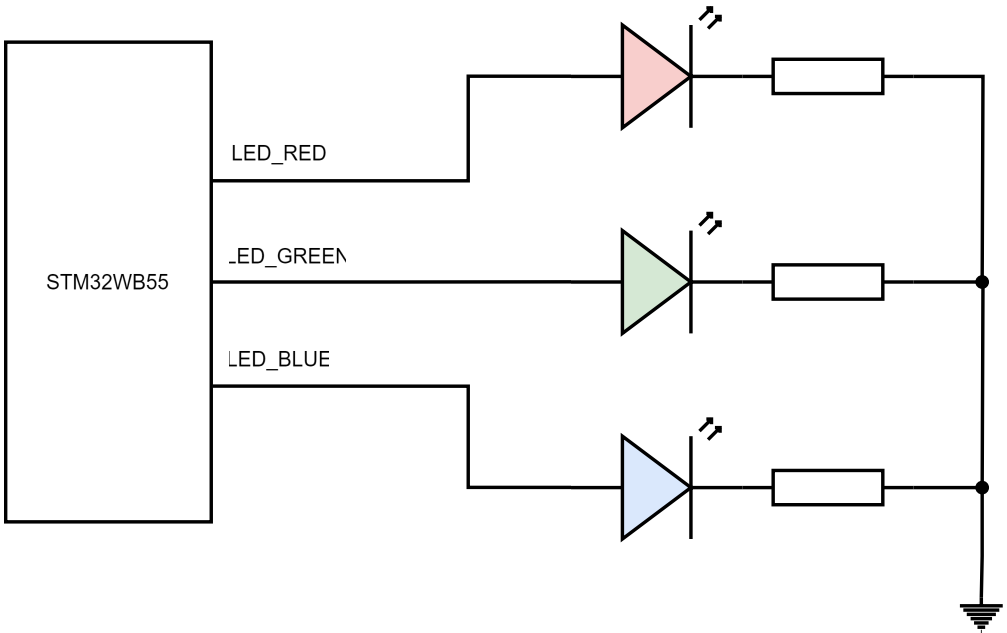
A good understanding of the digital GPIO pins operation and principles of interrupts is necessary. We use the Arduino Core STM32 framework (stm32duino)[1)] for STM32WB55. During this laboratory, no additional external libraries will be required for the project.

1.2. Background

LEDs and buttons are electrically connected to the microcontroller's pins. Below, we will show how they are connected because it determines their behaviour depending on the software.

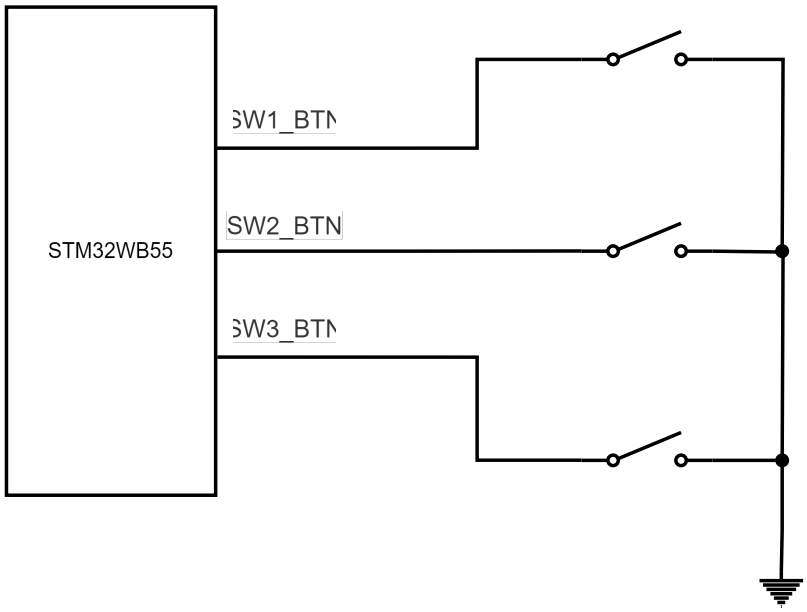
1.2.1. LED connection

LEDs are electrically connected to the microcontroller's pins. LEDs have 3 different colours (Red, Green, and Blue). The red LED is connected to pin PB1, the green LED is connected to pin PB0, and the blue LED is connected to pin PB5. LED is lit when the pin state is “1”, and dark when the pin state is “0”. LEDs are easily accessible in the software as digital pins named LED_RED, LED_GREEN, and LED_BLUE, respectively. Each LED can be controlled with the Arduino standard DigitalWrite() function to switch it on or off.



1.2.2. Buttons connection

The buttons on board are named SW1, SW2, and SW3, and are connected to pins PC4, PD0, and PD1, respectively. In the software, the names of the pushbuttons are SW1_BTN, SW2_BTN, and SW3_BTN. Pushbuttons are easily accessible in the software by their names using the Arduino function DigitalRead(). Buttons are connected to the ground, so the reading gives the “0” state result if the button is pressed and the “1” state otherwise.



1.2.3. Interrupts

The button can control the LED in a simple way with a pair of functions, digitalRead() and digitalWrite(). The drawback of this approach is the dependency of the reading and writing time on the current flow of the program. If the program calls the Delay() function, it's impossible to perform other actions, including controlling the LED state. The asynchronous approach would allow us to use the Delay() or other blocking functions together with the instant reaction on the button press. This can be achieved with interrupts. It is possible to call the interrupt handler function as the callback attached to the button press event. To define the attachment, you can use the attachInterrupt() function. It requires three parameters pin number, the address of the callback function, and the event type. You can use the event type as FALLING, which detects the pin state change from “1” to “0”, RISING, which detects the pin state change from “0” to “1”, and CHANGE, which reacts on both signal edges.



Except for CHANGE, RISING, and FALLING, there are also HIGH and LOW events. HIGH works the same as RISING, LOW the same as FALLING.

Inside the callback function, you can write any code until you keep the function short, and avoid using loops and delays. The function called at the interrupt level has a higher priority over the main program, so longer execution can disrupt the normal behaviour of the program.

1.2.4. Tasks to be implemented

1. Write the program that will switch the red LED for one second, then the green LED for one second, and the blue LED for one second repeatedly. Use the Delay() function. This part is to be familiar with the environment, GPIO controlling, and the STM32WB55 board.
2. Write the program that lights the blue LED when the SW1 button is pressed, green LED when SW2 is pressed, and red LED when SW3 is pressed.
3. Write the program that controls the LEDs with interrupts. One LED should blink periodically using the Delay() function, and the second LED should be controlled asynchronously with two buttons. One button press should turn the LED on, and the second button should turn the LED off.
4. Experiment with the program from task 3 using the long (few seconds) delay inside the callback function and observe the behaviour of periodical LED blinking.

1.2.5. Task 1

In the Arduino framework for STM32 handling the GPIO pins is done the same way as for the classic Arduino board. First, the pin should be configured as the output.

```
pinMode(LED_RED, OUTPUT);
```

To turn the LED on, we have to set the output pin to “1”.

```
DigitalWrite(LED_RED, 1);
```

To turn the LED off, we have to clear the output pin.

```
DigitalWrite(LED_RED, 0);
```

Use the delay function to be able to observe the behaviour of LEDs.

```
Delay(500); //Parameter is the delay time in microseconds
```

1.2.6. Task 2

From task 1, you are already familiar with turning the LED on and off. To read the button state, you can use the digitalRead() function, which requires the pin number as the parameter, and returns the state of the input pin at the time of the function call.

```
digitalRead(SW1_BTN);
```

1.2.7. Task 3

This task requires writing the callback functions triggered by the events of button pressing. One function should be attached to the chosen push button, and the second function should be attached to another push button. Attaching switch 1 with the SW1_callback_f function which is called every time the button is pressed, can look like the following line.

```
attachInterrupt (SW1_BTN, SW1_callback_f, FALLING);
```

SW1_BTN is the pin number of the chosen button. SW1_callback_f is the name of the function which works as the interrupt handler, FALLING means that our function will be started whenever the signal at the pin changes from “1” to “0”. Inside the interrupt handler, you can use the digitalWrite() function to turn the LED on or off.

```
void SW1_callback_f(){
  digitalWrite(LED_RED, 1);
}
```

1.2.8. Task 4

In this task, add the delay() function inside the interrupt handler. Experiment with the number of microseconds. Observe the results.

1.2.9. Result validation

You should be able to observe LED blinking in task 1 and control LEDs with push buttons for tasks 2 and 3. For task 4 you should observe the increase in LED timing after the button press.

1.3. FAQ

What if the time of interrupt handler execution is too long?: You need to avoid long interrupt handlers. They should be as short as possible. In some cases, if a handler is too long, it can block proper execution of the operating system and cause the system to crash. A good practice is to write in the handler code for setting the flag, which passes the information to the main code about the need to perform some task, and code for the whole task in the main program.

[1)]
“stm32duino”, https://github.com/stm32duino

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:macro:embedeed:lab:stm32-gpio

Last update: **2025/04/02 10:05**