Table of Contents

1. STM32 Control of build-in LEDs with PWM 2
1.1. Prerequisites 2
1.1.1. Tasks to be implemented 2
1.1.2. Task 1 2
1.1.3. Task 2 3
1.1.3.1. Hardware we use 3
1.1.3.2. Software library 3
1.1.3.3. Step 1 3
1.1.3.4. Step 2 4
1.1.3.5. Step 3 4
1.1.3.6. Step 4 5
1.1.3.7. Step 5 5
1.1.4. Result validation 5
1.2. FAQ 5

1. STM32 Control of build-in LEDs with PWM

This exercise presents how to handle the brightness control of three LEDs mounted on the Nucleo STM32WB55 board. LEDs are electrically connected to microcontroller's pins. Those LEDs have 3 different colours (Red, Green, and Blue). The red LED is connected to the pin PB1, the green LED to the pin PB0, and the blue LED to the 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 or control brightness using a PWM signal.

1.1. Prerequisites

A good understanding of the digital GPIO pins operation, PWM signal and duty cycle is necessary. We also use built-in timers to control the PWM hardware channels of the STM32WB55 chip. In this case, we do not use an external library; instead, we use the hardware timer library built in the Arduino Core STM32 framework (stm32duino)[1] for STM32WB55 so that no additional external libraries will be included in the project.

1.1.1. 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. This part is to recall the knowledge from the first STM laboratory, be familiar with the equipment, and confirm the proper operation of laboratory elements.
2. Implement a program that will light LEDs consecutively with red, green, and blue with a PWM signal. Every LED should start from dark, go up to the maximum brightness and then go down to dark. This must be implemented with the use of a timer.

1.1.2. 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.

1.1.3. Task 2

To do the second task it is necessary to understand the timer operation. It is more complex task than the previous so some detailed explanation is provided below.

1.1.3.1. Hardware we use

In the STM32WB55 there are two timers capable of generating the PWM signal for up to four channels per timer. Unfortunately, all three LEDs mounted on the Nucleo board are connected to pins which are not controlled directly by a timer. This situation requires using the interrupts to turn the LEDs on and off in the callback functions connected to interrupts signalled by the chosen timer instance. If we use one timer for all three LEDs they will operate with the same frequency. The duty cycle can be different for every LED if we use a separate channel for each of them.

1.1.3.2. Software library

The Arduino framework for STM implements the Hardware Timer library. The library implements a set of functions which allow us to choose the working mode of the timer, enable or disable its operation, and control the duty cycle of the PWM signal expressing it in different formats, including percentages.

1.1.3.3. Step 1

In the first step, we include the library, define PIN assignments to channels, and specify the PWM frequency (100Hz). It is not possible to define a PWM channel without pin assignment, so here we assign channels to pins capable of operating as PWM outputs (although we are not able to control on-board LEDs with these pins).



We use pins which are not connected to on-board LEDs, so the pins here are dummies. In the case of our own PCB design, it is possible to connect proper outputs to LEDs directly and control them without the need to use interrupts.

```
#include <HardwareTimer.h>

// Pins definition for RGB LED
#define LED_PWM_GREEN D3  //Arduino numbering D3, STM numbering PA_10
#define LED_PWM_BLUE  D6  //Arduino numbering D6, STM numbering PA_8
#define LED_PWM_RED   D9  //Arduino numbering D9, STM numbering PA_9

#define PWM_freq 100
```

GPIO dummy pins for PWM channels are D9 (Red), D3 (Green) and D6 (Blue). They are internally connected at the outputs of timer 1.

1.1.3.4. Step 2

Define variables for the timer object and handlers of channels.

```
// PWM variables definitions
HardwareTimer *MyTimLED;               //Hardware Timer for PWM
uint32_t channelR, channelG, channelB; //RGB channels
```

Instantiate the timer object and initialise the 3 channels for PWM and make them dark (1% duty cycle):

```
//Create Timer instance type based on the chosen LED pin (the same timer is
used for all LEDs).
TIM_TypeDef *Instance = (TIM_TypeDef
*)pinmap_peripheral(digitalPinToPinName(LED_PWM_RED), PinMap_PWM);

//Define three separate channels for LEDs
channelR = STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(LED_PWM_RED),
PinMap_PWM));
channelG =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(LED_PWM_GREEN),
PinMap_PWM));
channelB =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(LED_PWM_BLUE),
PinMap_PWM));

// Instantiate HardwareTimer object. Thanks to 'new' instantiation,
HardwareTimer is not destructed when setup() function is finished.
MyTimLED = new HardwareTimer(Instance);

// Configure and start PWM
MyTimLED->setPWM(channelR, LED_PWM_RED, PWM_freq, 1);   /* 100 Hertz, 1%
dutycycle. */
MyTimLED->setPWM(channelG, LED_PWM_GREEN, PWM_freq, 1); /* 100 Hertz, 1%
dutycycle. */
MyTimLED->setPWM(channelB, LED_PWM_BLUE, PWM_freq, 1);  /* 100 Hertz, 1%
dutycycle. */
```

1.1.3.5. Step 3

Define callback functions to control real on-board LEDs. There will be one common callback to turn on all LEDs at the beginning of every cycle, and three separate callbacks for three channels to turn LEDs off. This enables different brightness of specific LEDs.

```
MyTimLED->attachInterrupt(Update_on_callback);
MyTimLED->attachInterrupt(channelR, Update_off_callback);
```

1.1.3.6. Step 4

To modify the intensity of the chosen LED in the loop() function use the setCaptureCompare function. It changes the content of the capture compare register responsible for the duration of the duty cycle. In the following example, the duty_cycle_value can vary between 0 and 100.

```
MyTimLED->setCaptureCompare(channelR, duty_cycle_value,
PERCENT_COMPARE_FORMAT);   //modify duty cycle
```

1.1.3.7. Step 5

Write a loop for each colour (R, G, then B) to light the colour from dark to max value. Mind to compose code to increase and decrease each colour. A hint is below (for channelR):

```
  // Increase brightness
  for (int duty_cycle_value = 0; duty_cycle_value <= 50; duty_cycle_value++)
{
    // Gradually increase duty cycle for Red LED
    MyTimLED->setCaptureCompare(channelR, duty_cycle_value,
PERCENT_COMPARE_FORMAT);
    delay(20); // Delay for smooth transition
  }

  delay(100);

  // Decrease brightness
  for (int duty_cycle_value = 50; duty_cycle_value >= 0; duty_cycle_value--)
{
    // Gradually decrease duty cycle for Red LED
    MyTimLED->setCaptureCompare(channelR, duty_cycle_value,
PERCENT_COMPARE_FORMAT);
    delay(20); // Delay for smooth transition
  }
```

1.1.4. Result validation

You should be able to observe the pulsing colours of the LEDs, increasing and decreasing brightness linearly.

1.2. FAQ

What is the maximum number of channels?: the MCU we use here is STM32WB55 on the Nucleo board. The pins available are connected to timer 1 with three PWM channels and timer 2 with 4 PWM channels. A single timer can generate PWM signals with independent duty cycles and identical frequency.
What is the maximum bit resolution for PWM?: Maximum resolution is 16 bits. Note that we can

express the duty cycle in a variety of formats. In the presented example we expressed it in percentage so it varies between 1 and 100. It can be also expressed in ticks, Hertz, microseconds, and 1-16 bits numbers.
What PWM frequency should I use?: there is no straightforward answer to this question: assuming you observe LED remotely with a camera, even 50Hz would be enough. But it would give a severe flickering experience to the live user, on the other hand. In the example above, we propose 100Hz, but this MCU can easily handle higher frequencies.

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-pwm?rev=1710745575

Last update: **2024/03/18 08:06**