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. Steps 3
1.1.3.1. Step 1 3
1.1.3.2. Step 2 3
1.1.3.3. Step 3 4
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) and are 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 be familiar with the environment, GPIO controlling, and the STM32WB55 board.
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 in the same way as for the classic Arduino board. First, the pin should be configured as the output.

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

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”.

The hardware timer library implements functions which allow us to control the duty cycle of the PWM signal and express it in different formats, including percentages. In the laboratory equipment, all three LEDs are connected to one hardware timer instance. Thanks to this LEDs operate as three separate channels of a single timer. In the case of setting the PWM duty cycle expressed in percentages, the minimum value is 0, and the max (full brightness) is 100. Note that full brightness may be too bright for the observation camera, so consider using a range between 0 and 50%.

1.1.3. Steps

To use PWM in STM32WB55, it is best to use a built-in hardware timer library.

```
#include <HardwareTimer.h>
```

The hardware timer uses internal timer modules and allows us to define channels attached to the timer. We will use 1 channel per colour (R, G and B, so 3 in total). A PWM frequency is controlled with the timer to be shared for all 3 R, G and B channels. Channels control the PWM duty cycle.

1.1.3.1. Step 1

Include the library, and define PIN assignments to channels and PWM frequency (100Hz):

```
#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 pins controlling LEDS are D9 (Red), D3 (Green) and D6 (Blue), respectively.

1.1.3.2. 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. */
```

To modify the intensity of the chosen LED after initialisation use the setCaptureCompare function. 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
```

1.1.3.3. Step 3

Write a loop for each colour (R, G, then B) to light the colour from dark to max value.



Full duty cycle (100%) will be too bright for the remote access video camera to handle it. Use some reasonable range such as 0..50% is strongly advised.

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 RGB LED, 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 (connected to our RGB LED) and timer 2 with 4 PWM channels (with two of them connected to the servo and fan described in other scenarios). 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=1709584742

Last update: **2024/03/04 21:39**