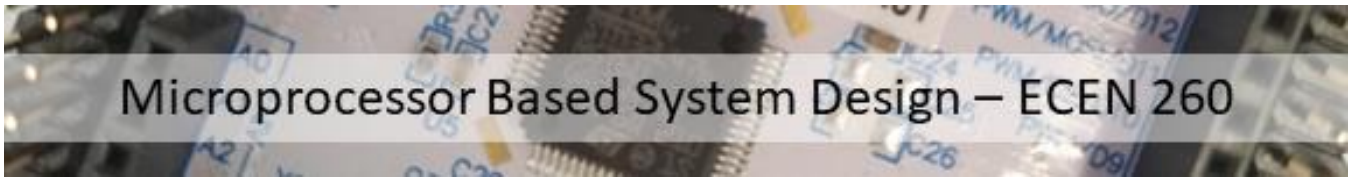


ECEN 260 Lab 09

Analog I/O

Lab Objectives

- Learn how to use built-in **clock drivers** to set up a timer.
- Learn how to use a timer and built-in **PWM drivers** to create a PWM output signal.
- Learn how to use built-in **ADC drivers** to read an analog input signal.
- Write code that interprets an ADC input signal.

Required Parts

- 1 Nucleo-L476RG development board
- 1 USB cable
- 1 knob potentiometer
- 1 joystick
- 1 breadboard (protoboard)
- 1 tri-color RGB LED
- 3 single-color LEDs
- 3 resistors
- several jumper wires

ST account: The code generation requires an ST account. Make sure you are logged in with your ST account in the IDE.

Overview

For this lab, you will separately test the timer (Part 1), PWM (Part 2), and ADC (Part 3) features of the STM microcontroller. Then, you combine all three concepts (Part 4), having a timer trigger the ADC sampling from three analog inputs and using those input values as PWM outputs to the tri-color RGB LED. You will also have three single-color LEDS showing the top 3-bits of one of the ADC values.

There is **no lab partner portion** for this lab. It may be completed on your own. Working with a lab partner is allowed, but not required. If you work with a lab partner, you may help each other, but you must each write your own code, wire your own circuits, and write your own report.

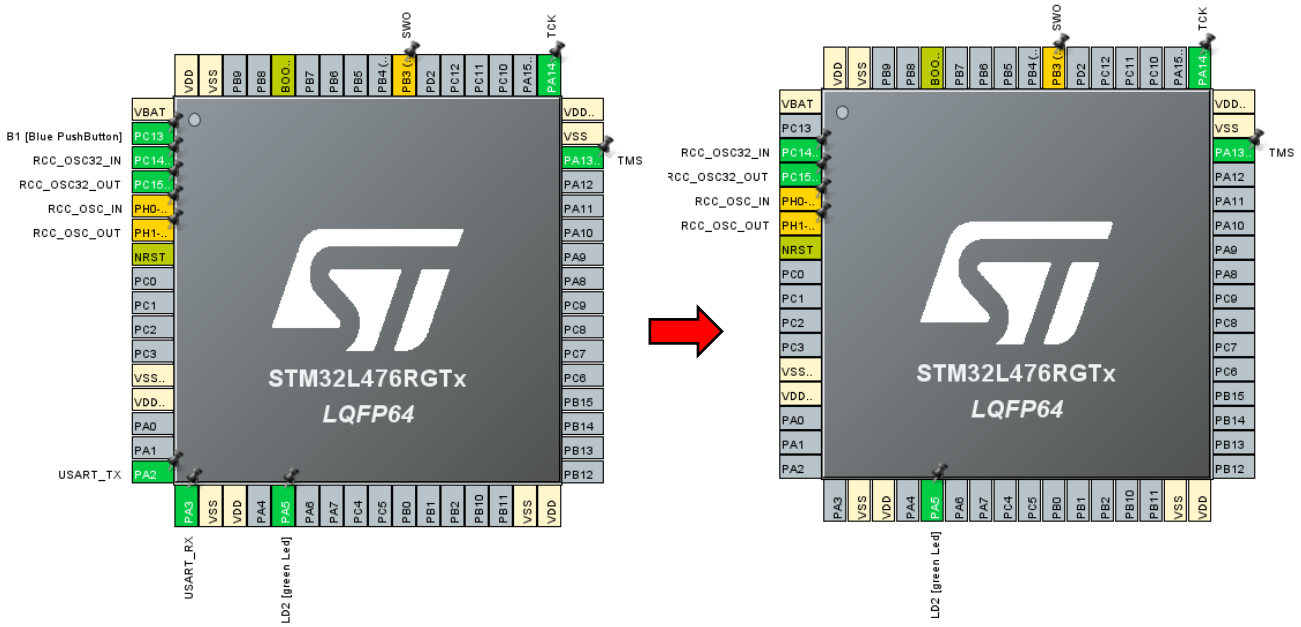
Part 1 (Timers)

For this part of the lab, we will set up a timer to toggle an LED once per second.
Note: The instructions for this part of the lab are adapted from a tutorial by DigiKey (link).

1. Open STM32CubeIDE.
2. Create a new project:
    a. File -> New -> STM32 Project.
    b. In the "Board Selector" tab, select the "NUCLEO-L476RG" board.
    c. Click "Next."
    d. Name the project "Lab9_PartA"
    e. Under "Targeted Project Type," do **not** select "Empty" – leave it on "STM32Cube."
    f. Click "Finish."
    g. When asked "Initialize all peripherals with their default Mode?" select "Yes."
    h. If asked to switch perspectives, click "Yes."

The Device Configuration Tool should now be open, showing a pinout image of the chip (see below). Notice that PA5 is already configured for LED 2 (LD2) and PC13 is already configured for Button 1 (B1). Other pins also have a default configuration, such as PA2 as the transmit (Tx) pin and PA3 as the receiver (Rx) pin. The Tx/Rx pins will be relevant for next week's lab.

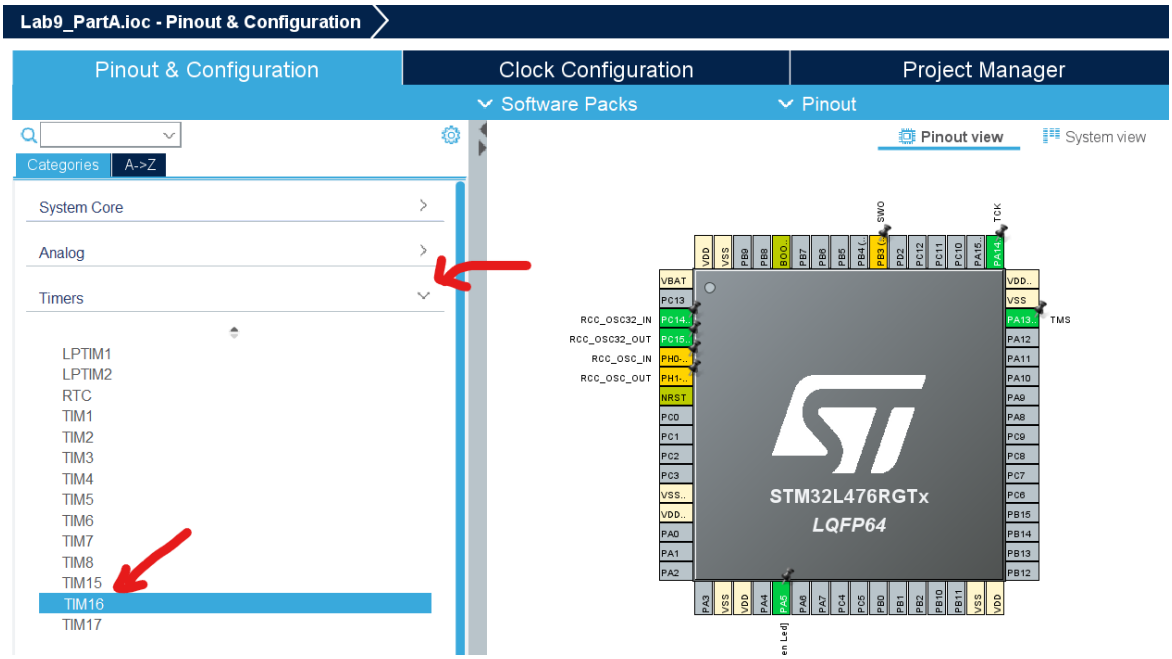
3. We won't use the button for this part of the lab. Click on PC13 and change it from GPIO_EXTI13 to Reset_State.
4. We also won't use the Tx/Rx pins. Click on PA2 to change it from USART2_TX to Reset_State and on PA3 to change it from USART2_RX to Reset_State.



Do not change any of the other default pins – they have to do with clocks and debug signals.

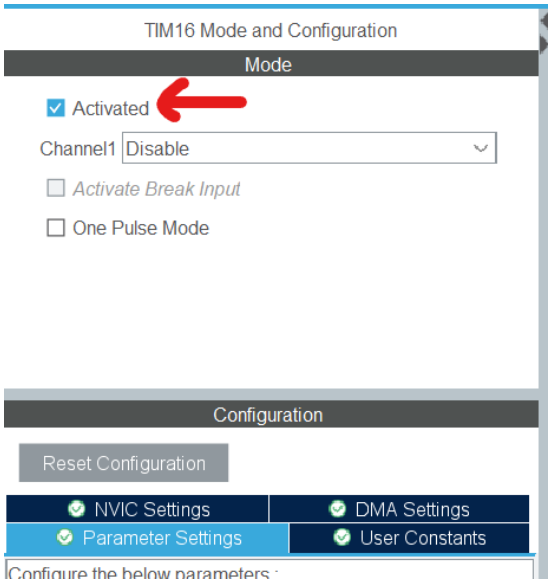
Next, we will use one of the existing clock signals to trigger one of the timers. For this part of the lab, we will use the main “High Speed Clock” (HCLK) that the CPU runs on as the source that triggers a specific timer to increment. There are several timers to choose from. We’ll use Timer16.

5. In the “Pinout & Configuration” tab and then under the “Categories” tab on the left, expand the “Timers” category and select TIM16.



The “TIM16 Mode and Configuration” pane will open.

6. Under “Mode” select “Activated” which will then open the “Configuration” options.

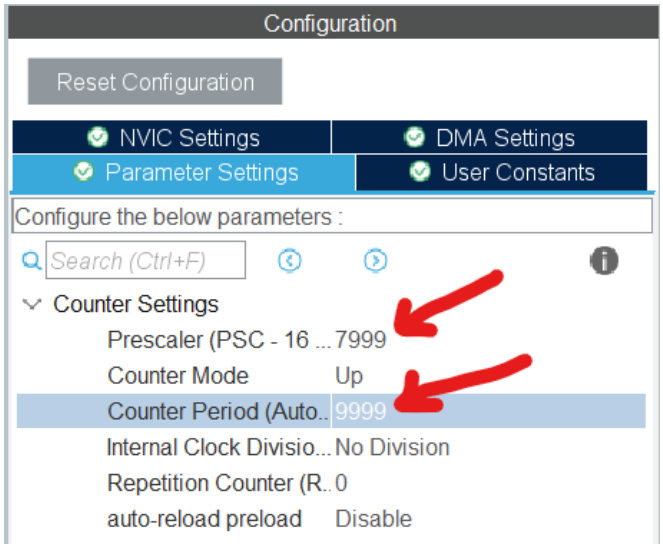


The HCLK default frequency is 80 MHz. We want to scale that down so our timer increments less frequently. If we scale the clock by 8000, we can have our timer increment once every 8000 clock periods. Since 80 million divided by 8000 is 10,000, our timer will increment at 10 kHz, or 10,000 times every second.

7. To scale the clock by 8000 for our timer, set the "prescaler" of the clock to 7999 (because it is zero-indexed). This option is found under "Configuration" in the "Parameter Settings" tab with the "Prescaler (PSC - 16 bits value) option.

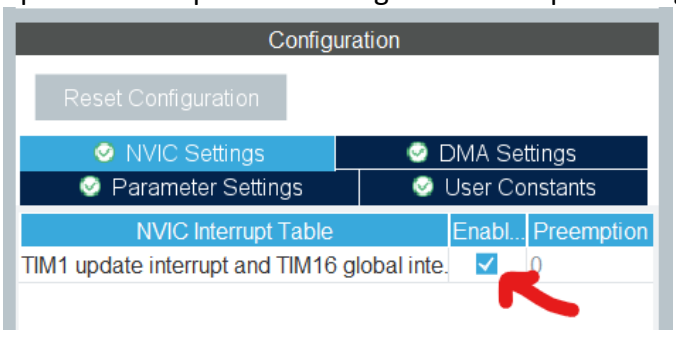
Next, we want to tell it when the timer should reset. If we have the timer reset after incrementing 10,000 times, then the timer will reset once every second.

8. Set the "Counter Period (AutoReload Register – 16 bits value) to 9999 (because it is zero-indexed). This option is also found under "Configuration" in the "Parameter Settings" tab.



Now, we want to enable an interrupt to trigger each time the timer resets by using the NVIC (Nested Vector Interrupt Controller). For bigger projects, you'll want your code to be able to do something else while it waits for a timer, which is why we use interrupts with timers.

9. Still under "Configuration," switch to the "NVIC Settings" tab.
10. Next to the "TIM1 update interrupt and TIM16 global interrupt" setting, select "Enable."



Now, we will use these settings we have changed to generate code that implements these settings using the built-in drivers.

11. Save the configuration: File -> Save.
12. When asked “Do you want to generate Code?” select “Yes.”
13. If asked to switch perspectives, select “Yes.”

A main.c file should now be open. If you change only the code sections between the “USER CODE BEGIN” and “USER CODE END” comments, then you can later make other changes with the Device Configuration Tool and regenerate new code without it changing the code you placed between the “USER CODE” tags.

Take a minute to look at the code. In main(), the code calls HAL_Init(), SystemClock_Config(), MX_GPIO_Init(), and MX_TIM16_Init() before going to an infinite loop. The code for the first of these functions is in an include file, while the definitions for the other three functions is generated and placed after main, setting up the system clock, the GPIO pin settings for LED2 (PA5), and the seetings for Timer16, respectively.

To use the timer, we need to start the timer in our code:

14. In the “USER CODE BEGIN 2” section, add the following line of code:

```
HAL_TIM_Base_Start_IT(&htim16); // Start timer
```

Finally, we need to add the code for the ISR that gets executed whenever the timer resets:

15. In the “USER CODE BEGIN 4” section, add the following ISR function definition:

```
// Callback: timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  // Check which version of the timer triggered this callback and toggle LED
  if (htim == &htim16)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }
}
```

This ISR will be executed when any of the timers trigger an interrupt, so it checks to see if it was Timer16 that triggered the interrupt. If it was, it will toggle PA5 (LED 2). Run the code, and it should now toggle the LED once every second.

Record a video of the LED toggling once every second for your report. For Part 1, you don’t need to submit any code, just the video. You are now finished with Part 1 (Timers). Continue to Part 2 (PWM).

Part 2 (PWM)

Pulse-Width Modulation (PWM) is the most common and cheapest way of producing an "analog" signal from a microcontroller. PWM isn't a true analog signal, but it mimics one by rapidly switching a digital signal from HIGH to LOW voltage, but with the ratio of time that it is HIGH:LOW determining the "average" output voltage supplied.

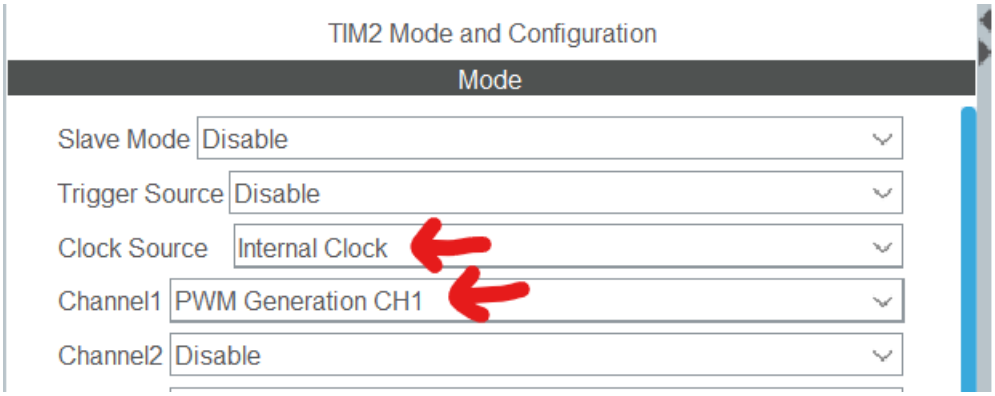
PWM signals are created using a timer to trigger when the signal should switch from HIGH to LOW or LOW to HIGH. Many microcontrollers, like our STM32L476RG chip, include drivers and circuitry to support this. We will use Timer2 and set up its Channel 1 to create a PWM signal.

Note: The instructions for this part are adapted from a tutorial by DeepBlue Embedded (link).

1. Create a new project as in Part 1, except with the name "Lab9_PartB." (Do **not** select "Empty.")
2. In the device configuration tool, change PA2 from Tx to "Reset_State," change PA3 from Rx to "Reset_State," and change PC13 (B2) from external interrupt to "Reset_State." We aren't using these pins for this part of the lab.
3. On PA5 (LED2), change it from GPIO_Output to "TIM2_CH1." This will attach PA5 in "alternate mode" to the channel 1 signal from Timer2, on which we will configure the PWM signal.
4. Under the "Timers" category, select "TIM2."

The "TIM2 Mode and Configuration" pane will appear.

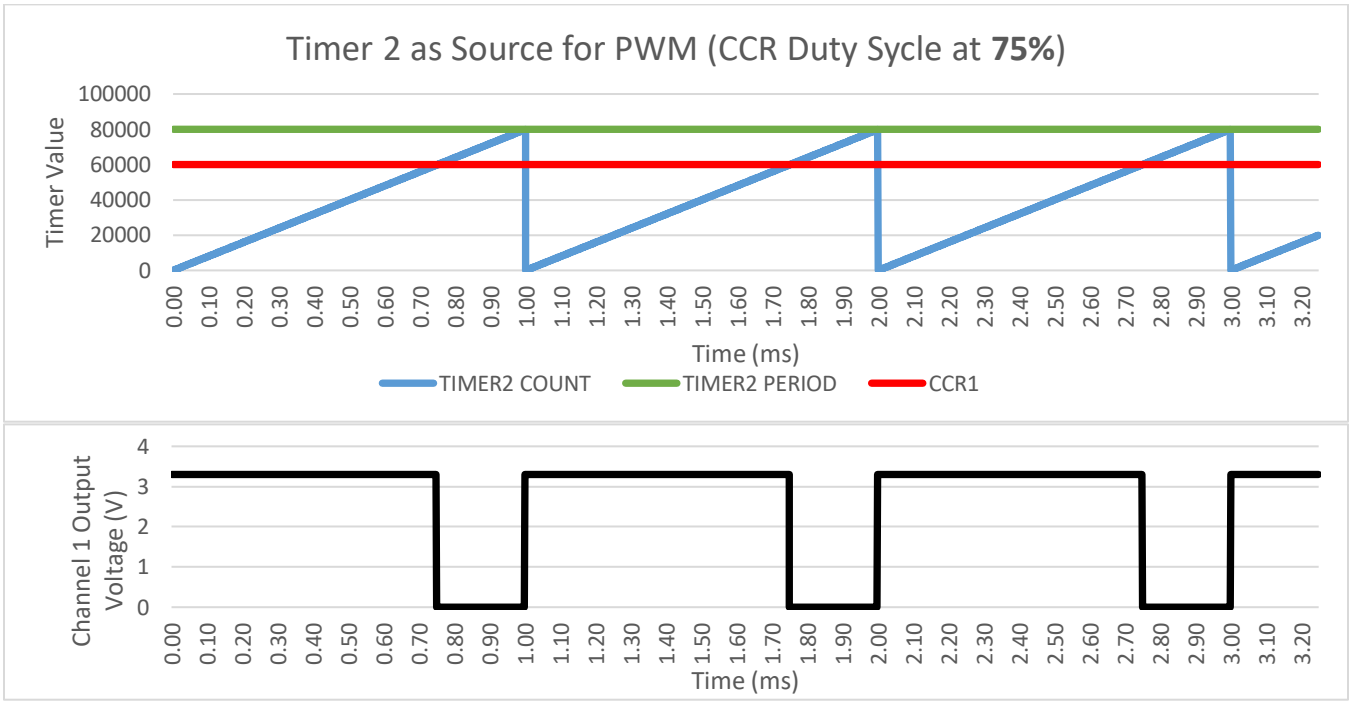
5. Under "Mode," change the "Clock Source" option to "Internal Clock."
6. Change the "Channel1" option to "PWM Generation CH1."



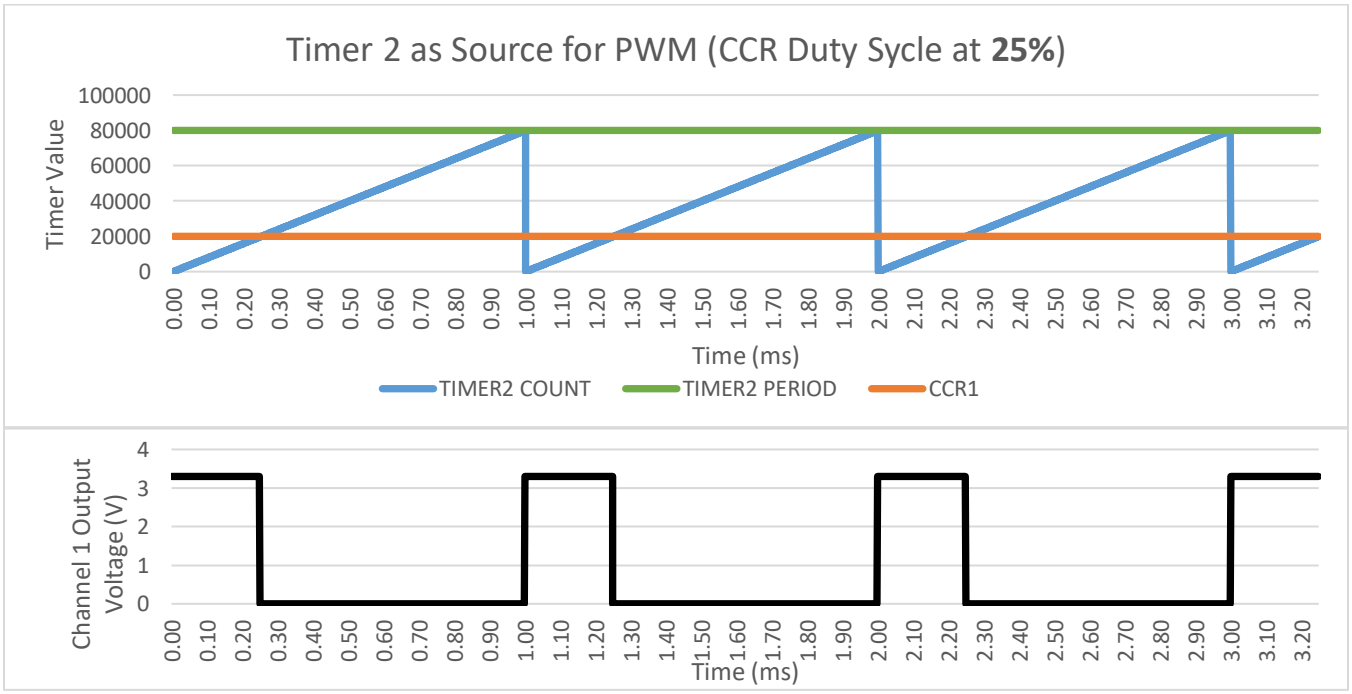
Let's set up the PWM to have a period of 1 ms – meaning it will complete a HIGH period and a LOW period (1 complete cycle) once every millisecond, or 1000 times per second. Since our PWM signal will be created using Timer2, let's change the frequency of Timer2 to be 1/80,000th of the HCLK frequency (80 MHz / 80,000 = 1 KHz).

7. Below "Configuration," switch to the "Parameter Settings" tab.
8. Change the "Counter Period" option to 79999 (because it is zero-indexed).
9. Change the "auto-reload preload" option to "Enable."

Timer 2 will now reset once every millisecond as it counts from 0 to 79999. The Capture/Control Register (CCR1) holds a value we choose in our code (any number between 0 and 79999) that will tell the PWM signal when to switch from HIGH to LOW along the way from 0 to 79999. Then, when it reaches 79999, it will go back HIGH again. See this example with CCR set to 75% the value of the Timer period, creating a 75% duty cycle:



Compare the above with the example below with CCR at **25%** of the period, creating a **75%** duty cycle:



10. Save and generate the code based on the above configurations.

In the generated main.c file:

11. Add the following line of code to “USER CODE BEGIN 2” to start Timer2 as PWM on Channel 1:

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // start Timer2 PWM on Channel 1
```

12. Add the following lines of code immediately after the “USER CODE BEGIN 3” comment, but still inside the main while loop:

```
int PWM_PERIOD = 80000;
float duty_cycle = 0;

// gradually increase duty cycle
while(duty_cycle < 1.0)
{
  TIM2->CCR1 = (int) (duty_cycle * PWM_PERIOD);
  duty_cycle += 0.01;
  HAL_Delay(10); // wait 10 ms
}

// gradually decrease duty cycle
while(duty_cycle > 0.0)
{
  TIM2->CCR1 = (int) (duty_cycle * PWM_PERIOD);
  duty_cycle -= 0.01;
  HAL_Delay(10); // wait 10 ms
}
```

The above code will gradually increase the duty cycle from 0 to 100% and then decrease it from 100% down to 0% by adjusting the CCR value to be that portion of the PWM_Period.

Run the code. You should see LED2 (PA5) gradually getting brighter as the duty cycle increases and then gradually getting dimmer as the duty cycle decreases.

Record a video of the LED changing brightness for your report. For Part 2, you don’t need to submit any code, just the video. You are now finished with Part 2 (PWM). Continue to Part 3 (ADC) on the next page.

Part 3 (ADC)

Note: This part of the lab is adapted from an example in *Programming with STM32* (link).

1. Create a new project as before, except with the name "Lab9_PartC." (Do **not** select "Empty.")
2. In the device configuration tool, change PA2 from Tx to "Reset_State," change PA3 from Rx to "Reset_State," change PC13 (B2) from external interrupt to "Reset_State," and change PA5 from output to "Reset_State." We aren't using these pins for this part of the lab.

Our microcontroller comes with three ADCs: ADC1, ADC2, and ADC3, each of which may be attached to one of several input channels. For this part of the lab, we will measure an analog input value on PA0 using ADC1. PA0 can connect to "Channel 5" on ADC1.

3. Change PA0 to "ADC1_IN5."
4. Under "Categories", expand the "Analog" category.
5. Select "ADC1."

That should open the "ADC1 Mode and Configuration" pane.

6. Under "Mode," find the "IN5" option and change it to "IN5 Single-ended."
7. Save and generate the code.

In the generated main.c file, add the following code immediately after the "USER CODE BEGIN 3" comment, but still inside the main while loop:

```
// Start ADC Conversion
HAL_ADC_Start(&hadc1);

// Wait for ADC conversion to complete
HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

// Read ADC value
uint16_t analog_measurement = HAL_ADC_GetValue(&hadc1);

// Delay 1 ms
HAL_Delay(1);
```

When you later build and run this code, the compiler will give you a warning that the variable "analog_measurement" is set but never used. That's ok. We're just going to look at the value in debug mode.

Place a breakpoint on the line with "HAL_Delay(1)" by double clicking on the line number to the left of the code. Then, enter debug mode by clicking on the bug in menu at the top of the IDE. Press the "Resume" button to start running the code in debug mode. The code should stop at your breakpoint.

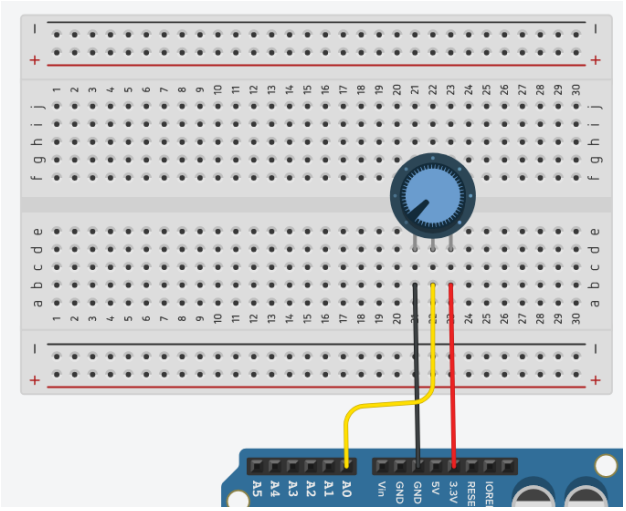
If you can't see your "Variables" window, go to Window -> Show View -> Variables.

Now, we will use a potentiometer (variable resistor) as a voltage divider to create an adjustable analog signal that we can read with PA0 on the microcontroller. Turning the knob on the potentiometer will change the voltage.

Wiring Instructions:
Find your potentiometer knob. It should have three pins. Note that if you switch left and right, it won't matter – that will just change the direction you turn the knob. The middle pin, however, must be the pin that goes to the analog input.

1. Connect the left pin to Ground.
2. Connect the right pin to 3.3V.
3. Connect the middle pin to PA0 (your analog input).

See the image below for help. Note that your potentiometer may have its three pins space farther apart, so the columns you connect them into the breadboard may be farther apart. If your potentiometer has wires soldered to it, you can connect those directly to the Nucleo board rather than plugging your potentiometer into a breadboard.



Observe the analog input:
1. Turn the knob all the way to the left. Then press resume. You should see that your analog_measurement variable is a value close to zero. **Take a screenshot.**
2. Turn the knob all the way to the right. Then press resume. You should see that your analog_measurement variable is a value a little below 4096 (which is the max for a 12-bit ADC, which is what we have). **Take a screenshot.**
3. Turn the knob to the middle. Then press resume. You should see that your analog_measurement variable is a value somewhere in the middle. **Take a screenshot.**

For Part 3, you don't need to submit any code, just the three screenshots. You are now finished with Part 3 (ADC). Continue to Part 4 (Timer + ADC + PWM) on the next page.

Part 4 (Timer + ADC + PWM)

For this part of the lab, we will combine all the concepts together. Rather than continuously polling to get a new ADC sample, we will have a timer trigger an interrupt when it is time to get the next sample.

We will sample 10 times every second. In the timer's ISR, we will read three analog input values with the ADCs (one from the knob, and two from the joystick – X&Y). Then, based on those values, we will control LEDs – PWM will control the intensity of the RGB values in a tri-color RGB LED, and normal digital outputs will be used to turn on/off three other single-color LEDs.

More specifically, the analog value read from the knob will control the intensity of the "red" hue of the tri-color RGB LED, the analog value from the x-coordinate of the joystick will control the intensity of the "green" hue of the tri-color RGB LED, and the analog value from the y-coordinate of the joystick will control the intensity of the "blue" hue of the tri-color RGB LED. The code for this is provided for you.

In addition, you will take the 3 most-significant bits of the 12-bit ADC result of reading from the knob and have them control the three other single-color LEDs. As you turn the knob, you should see these three lights count in binary from 000 -> 001 -> 010 -> … -> 110 -> 111. This code is *not* provided for you.

Setting up the Sampling Timer (Timer16)

1. Create a new project called "Lab9_PartD" (not empty).
2. In the device configuration tool, change PA2 from Tx to "Reset_State," change PA3 from Rx to "Reset_State," change PC13 (B2) from external interrupt to "Reset_State," and change PA5 from output to "Reset_State." We aren't using these pins.
3. Under "Categories," expand the "Timers" category, and select TIM16.
4. In the "TIM16 Mode and Configuration" page, under "Mode," select "Activated."
5. Under "Configuration" on the "Parameter Settings" tab, change "Prescaler" to 7999 and "Counter Period" to 999. This will have the timer increment at 10 kHz (80 MHz / 8000) and reset every 1000 times, or in other words, once every 1/10th of a second. This is how often we will sample with the ADCs.
6. Still under "Configuration," switch to the "NVIC Settings" tab. Next to the "TIM1 update interrupt and TIM16 global interrupt" setting, select "Enable."

Setting up 3 ADC Input Channels

While, we can have a single ADC switch back and forth between multiple input channels, to make the code simpler, let's have each of our three analog input channels go to a different ADC – PA0 to ADC1's channel 5. PA1 to ADC2's channel 6, and PC0's ADC3's channel 1:

1. In the device configuration tool, change PA0 to "ADC**1**_IN5," change PA1 to "ADC**2**_IN6," and change PC0 to "ADC**3**_IN1."
2. Under "Categories," expand the "Analog" category.
3. Select "ADC1," then under "Mode" change "IN5" to "IN5 Single-ended."
4. Select "ADC2," then under "Mode" change "IN6" to "IN6 Single-ended."
5. Select "ADC3," then under "Mode" change "IN1" to "IN1 Single-ended."

To wire up the 3 ADC inputs:

6. On the knob, connect the middle pin to PA0, the left pin to ground, and the right pin to 3.3V.
7. On the joystick:
    a. connect the GND pin of the joystick to ground.
    b. connect the +5V pin of the joystick to 3.3V (we're using a V_ref of 3.3V for the ADC).
    c. connect the VRx pin (x-coordinate) to PA1.
    d. connect the VRy pin (y-coordinate) to PC0.
    e. leave the SW pin disconnected.

Setting up the 3 PWM Output Signals

We will have three different PWM outputs for the RGB lights in the tri-color RGB LED. To simplify the code, we'll have them all use the same timer, just on different channels. Each channel will have a different CCR value for the duty cycle. TIM3 has 3 available channels that can go to these 3 available pins: PA6 (channel 1) to the red pin, PA7 (channel 2) to green pin, and PB0 (channel 3) to blue pin.

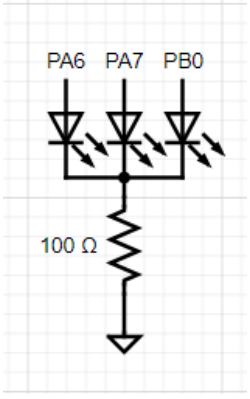
1. In the device configuration tool, change PA6 to "TIM3_CH1," change PA7 to "TIM3_CH2," and change PB0 to "TIM3_CH3."
2. Under the "Timers" category, select "TIM3."
3. Under "Mode," change the clock source to "Internal Clock," change Channel 1 to "PWM Generation CH1," change Channel 2 to "PWM Generation CH2," and change Channel 3 to "PWM Generation CH3."

Note: Because TIM3 is a 16-bit timer, we can't have it count up to 80,000 like we did with TIM2 in Part(B) because 2^16 = 65,536. But, if we scale the clock by 2 and count to 40,000, we'll have the same PWM period of once every millisecond (80 MHz / (2*40,000)).

4. Under "Configuration," select the "Parameter Settings" tab.
5. Change the Prescaler to 1 (zero-indexed).
6. Change the Counter Period to 39999 (zero-indexed).
7. Change the "auto-reload preload" option to "Enable."

To wire up the 3 PWM outputs to the tri-color RGB LED:

8. Connect the ground pin of the tri-color RGB LED (the longest pin) to ground via a current-limiting resistor.
9. Connect the blue pin of the tri-color RGB LED (the shortest pin) to PB0.
10. Connect the green pin (the pin between the ground & blue pins) to PA7.
11. Connect the red pin (on the other side of the ground pin) to PA6.



Setting up the 3 Digital LED Outputs

We will have three of the single-color LEDs show the 3 most-significant bits of the ADC value from the knob. These are digital on/off outputs. We'll use PA10, PB5, and PA8.

1. In the device configuration tool, change PA10, PB5, and PA8 each to "GPIO_OUTPUT."
2. Choose one color LED for PA10 (the most significant bit), another for PB5 (the 2nd most significant bit), and another for PA8 (the 3rd most significant bit). Use current limiting resistors.

Generate and Edit the Code

1. Save and generate the code.
2. In the "USER CODE BEGIN 2" section, add the following code to start the PWM signals:

```
HAL_TIM_Base_Start_IT(&htim16); // Start Timer16
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1); // start Timer3 PWM on Channel 1
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2); // start Timer3 PWM on Channel 2
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // start Timer3 PWM on Channel 3
```

3. In the "USER CODE BEGIN 4" section, add the following ISR function definition:

```
// Callback: timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  // Check which version of the timer triggered this callback and toggle LED
  if (htim == &htim16)
  {
    int PWM_PERIOD = 40000;
    int ADC_RANGE = 4096; // 2^12 (12-bit resolution)

    // Start ADC Conversions
    HAL_ADC_Start(&hadc1);
    HAL_ADC_Start(&hadc2);
    HAL_ADC_Start(&hadc3);

    // Wait for ADC conversions to complete
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY);
    HAL_ADC_PollForConversion(&hadc3, HAL_MAX_DELAY);

    // Read ADC values
    uint16_t knob_measurement = HAL_ADC_GetValue(&hadc1);
    uint16_t xjoy_measurement = HAL_ADC_GetValue(&hadc2);
    uint16_t yjoy_measurement = HAL_ADC_GetValue(&hadc3);

    // Convert ADC levels to a fraction of total
    float knob_value = ((float) knob_measurement) / ADC_RANGE;
    float xjoy_value = ((float) xjoy_measurement) / ADC_RANGE;
    float yjoy_value = ((float) yjoy_measurement) / ADC_RANGE;

    // Write the PWM duty cycle values for the tri-color RGB LED
    TIM3->CCR1 = (int) (knob_value * PWM_PERIOD); // red
    TIM3->CCR2 = (int) (xjoy_value * PWM_PERIOD); // green
    TIM3->CCR3 = (int) (yjoy_value * PWM_PERIOD); // blue

    // Use the three single-color LEDs as the three
    // most-significant bits of the knob measurement
    ???;
  }
}
```

4. Edit the above code where the "???" is to add some lines of code that will check the top three most significant bits of `knob_measurement` and turn on/off PA10, PB5, and PA8 respectively with those bit values. We did a 12-bit conversion, so you are looking at bits 11, 10, and 9.

Hint: This is how to use the built-in drivers to set/reset output pin PA10:

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
```

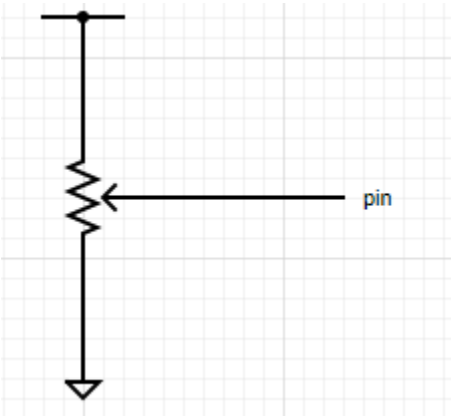
Your final circuit should have the knob control the intensity of the red hue of the tri-color LED, the x-coordinate of the joystick control the intensity of the green hue, and the y-coordinate control the intensity of the blue hue. At the same time, the three single-color digital LEDs in your circuit should show the binary representations of the "step" of the knob, based on 3-bit resolution (8 steps total – 000 through 111).

Note: Many of the joysticks are inverted with "up" being negative Y and "down" being positive Y.

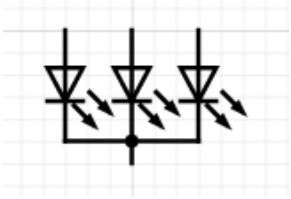
Take a video demonstration for Part 4. For this part of the lab, you do need to include your code – but just the code in the **`HAL_TIM_PeriodElapsedCallback`** Timer ISR definition.

Schematic for your report

Create a schematic showing the final circuit for Part 4. Your potentiometers (variable resistors) should look like the image below. If you use circuit-diagram.org, it is the symbol called "Potentiometer (US)." You should have two potentiometers for the joystick (one for the X direction and another for the Y direction) and a third potentiometer for the knob. While the X and Y are in a single device, show them as separate potentiometers on the schematic. Each potentiometer should have +3.3V on the top and 0V on the bottom.



For the tri-color RGB LED, you can search for and add the RGB LED component, shown below. When you add it, it may give you the option for common anode or common cathode. Choose common cathode.



‘A’-level versus ‘B’-level Project

If you complete Parts 1-4 successfully, that is a ‘B’-level project. For an ‘A’-level project, you should record and upload another video of you explaining Part 4, including an explanation of the ADC inputs, the PWM outputs, and the digital outputs. Also explain the HAL_TIM_PeriodElapsedCallback Timer ISR, including when it gets called and why it has the code that it does.

Demonstration of Correct Operation

Submit the following videos/screenshots and code to the “Lab 09 Demo & Code” assignment in I-Learn:

1. For Part 1, attach the video of LED2 blinking.
2. For Part 2, attach the video of LED2 brightening and dimming.
3. For Part 3, include the three screen shots.
4. For Part 4, attach both:
    - a video demo of Part 4 working, and
    - and a code file with just the code in the HAL_TIM_PeriodElapsedCallback Timer ISR definition from Part 4.
5. For an ‘A’-level project, also upload the video explanation of Part 4.

Lab Report Instructions

For this week’s lab report, you should create your report completely on your own without using a template. **Create your own title page – make it professional.** Be sure to include the following sections:

- Overview
- Specifications
- Schematic
- Test Plan and Results
- Code
- Conclusion

Submit your report as a pdf to the “Lab 09 Report” assignment in I-Learn.

Don’t forget that I love you! :)