Brodric Young

ECEN 260

Lab 09 Report: Analog I/O

03/08/2025

Overview

In this lab, I learned about timers, Pulse-Width Modulation (PWM), and Analog-to-Digital Converters (ADC). The primary objectives were to learn how to set up a timer, create a PWM output signal, and read an analog input signal using built-in drivers. The lab was divided into four parts, with the first three parts focusing on individual components and the final part integrating all concepts. By the end of the lab, I successfully combined these elements to control an RGB LED with an PWM analog signal and display binary values on single-color LEDs based on ADC inputs with a digital signal.

Specifications

In Part 4 of the lab, I integrated the concepts used in the previous parts to create a comprehensive system that combines timers, PWM outputs, and ADC inputs. The system uses a timer to trigger ADC sampling, reads analog inputs from a potentiometer and joystick, adjusts the brightness of an RGB LED and single-color LEDs based on these inputs, and lights up LEDs representing the three most significant bits of the digital sample from the analog signal.

System Functionality:

- **Timer Configuration:** The timer (TIM16) is set up to trigger an interrupt every 0.1 seconds. This interrupt initiates the ADC sampling process.
- **ADC Inputs:** Three ADC channels are used to read analog signals from a potentiometer (PA0) and a joystick (PA1 for X-axis, PC0 for Y-axis). These inputs are sampled every 0.1 seconds.
- **PWM Outputs:** The PWM signals are generated using TIM3 on three channels to control the red (PA6), green (PA7), and blue (PB0) components of the RGB LED. The duty cycles of these PWM signals are adjusted based on the ADC readings.
- **Binary Display:** The three most significant bits of the ADC value from the potentiometer are displayed on three single-color LEDs (PA10, PB5, PA8), showing a binary count from 000 to 111.

Parts List:

- 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
- 4 resistors
- Several jumper wires

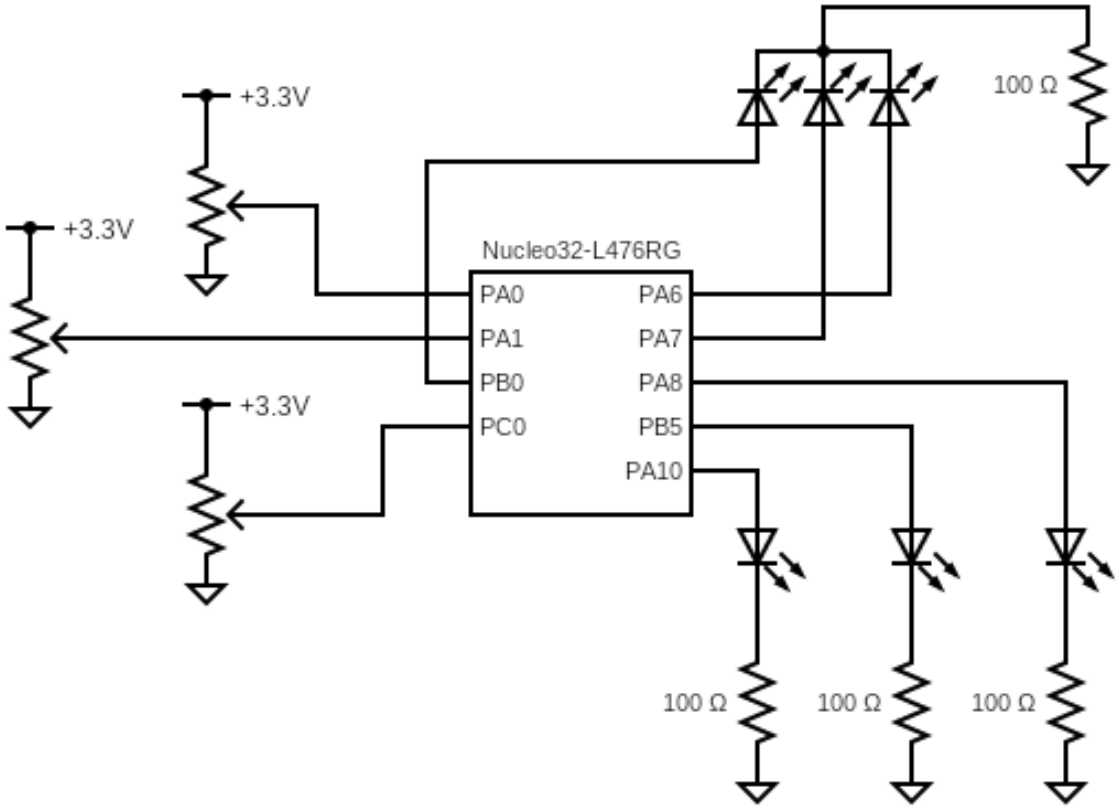
Limitations:

- The system relies on precise timing and ADC readings, which may be affected by noise or variations in component quality.
- The PWM resolution is limited by the timer's bit width, affecting the smoothness of LED brightness transitions.

How the Other Parts Worked:

- **Part 1 (Timers):** Used a timer to toggle an LED once per second.
- **Part 2 (PWM):** Created a PWM signal to control the brightness of an LED.
- **Part 3 (ADC):** Read an analog input signal using the ADC and displayed the value in debug mode.

Schematic


+3.3V
+3.3V
+3.3V
Nucleo32-L476RG
PA0
PA1
PB0
PC0
PA6
PA7
PA8
PB5
PA10
100 Ω
100 Ω
100 Ω
100 Ω

Test Plan and Results

Test Case	Test Scenario	Steps	Expected Results	Actual Results
2	Knob controls red LED intensity	1. Turn the knob fully left. 2. Gradually turn the knob right.	Red LED brightness changes smoothly from off to full brightness.	Red LED brightness changes smoothly from off to full brightness.
3	Joystick x-axis controls green LED intensity	1. Move the joystick fully left. 2. Gradually move the joystick to the right.	Green LED brightness changes smoothly from off to full brightness.	Green LED brightness changes smoothly from off to full brightness.
4	Joystick y-axis controls blue LED intensity	1. Move the joystick fully down. 2. Gradually move the joystick up.	Blue LED brightness changes smoothly from off to full brightness.	Blue LED brightness changes smoothly from off to full brightness.
5	Single-color LEDs display binary values of knob	1. Turn the knob fully left. 2. Gradually turn the knob right.	LEDs show binary count from 000 to 111 based on knob position.	LEDs show binary count from 000 to 111 based on knob position.
6	System handles rapid changes in knob input	1. Rapidly turn the knob back and forth.	Red LED and single-color LEDs respond correctly to rapid changes.	Difficult to tell, but it seemed that the red LED and single-color LEDs responded correctly to rapid changes.
7	System handles rapid changes in joystick input	1. Rapidly move the joystick in all directions.	Green and blue LEDs respond correctly to rapid changes.	Green and blue LEDs respond correctly to rapid changes.
8	Combined input changes	1. Adjust the knob and joystick simultaneously.	All LEDs respond correctly to simultaneous input changes.	All LEDs respond correctly to simultaneous input changes.

Code

```
// Callback: this ISR is triggered whenever a timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
   // Check which timer triggered this callback
   if (htim == &htim16) // if the triggered timer was Timer 16
   {
      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

      // get the three most significant bits
      char msb = (knob_measurement & (1 << 11)) >> 11;
      char mid = (knob_measurement & (1 << 10)) >> 10;
      char lsb = (knob_measurement & (1 << 9)) >> 9;

      // write the value of the corresponding bit to the LEDs
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, msb);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, mid);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, lsb);
   }
}
```

Conclusion

During this lab, I successfully configured timers, PWM outputs, and ADC inputs on the STM32 microcontroller. I learned how to generate PWM signals to control LED brightness and read analog signals using ADCs. By integrating these components, I created a system that dynamically adjusts LED colors and displays binary values based on analog inputs.

This lab builds on concepts learned in previous labs, such as GPIO configuration and digital signal manipulation. It also connects to class topics from this week about digital and analog signals, converting between them and interpreting them. The skills gained in this lab will be valuable for future projects involving analog devices like sensors.