Lab 2: GPIOs and DAC

ECSE 444: Microprocessors

Winter 2020

Introduction

This lab introduces the STM32L4 Discovery kit IoT node development board that will be used for the remainder of the labs. This board features an STM32L4 series micro-controller based on the ARM Cortex-M4 CPU core, and includes an impressive list of sensors and peripherals.

You will also be introduced to the STM32 Hardware Abstraction Layer (HAL) drivers for the family of STM32L4 devices. The HAL drivers are a collection of embedded software that provides a simple set of APIs to the programmer, thereby abstracting the hardware specific details for each device.

You will use the HAL drivers to program the General Purpose Input/Output (GPIO) pins on the board in both digital and analog mode, and to control the on-chip Digital to Analog (DAC) converter. In particular, you will:

- Learn how to use the HAL drivers to configure a GPIO as a digital output to light up an on board LED.
- Configure the GPIO associated with the blue push-button as a digital input to control the LED.
- Learn how to use the HAL drivers to configure and initialize the on-chip DAC.
- Generate a square wave on the DAC output GPIO via software.
- Generate a sinusoid on the DAC output GPIO via software.

Changelog

- 19-Sep Initial release.

Grading

- 40 % Using a push-button to control an LED
- 30 % Square wave generation and display on oscilloscope
- 30 % Sine wave generation and display on oscilloscope

Background

From this lab onward, you will need to become proficient with searching through user manuals and documentation to achieve a desired task. This handout will provide links to all information that is necessary to succeed in this lab, and it is highly recommended that you familiarize yourself with the various documents so that you are able to navigate through them yourself in future labs.

Understanding the hardware

- **ARM Cortex-M4:**

  This is a CPU core that has been designed by ARM, and supports the ARM assembly ISA that was used in Lab 0 and Lab 1. STM has provided a programming manual which gives a full description of the processor programming model, instruction set and core peripherals

- **STM32L475VG:**

  The is the Micro-controller Unit (MCU) that STM has built around the ARM Cortex-M4 core. The MCU chip includes useful and essential peripherals such as clock sources, external memory, additional timers, ADC's, DAC's, communication interfaces (SPI, I2C, USART, USB, etc.) and much more. STM has provided a datasheet and reference manual for the MCU. The datasheet provides a brief overview of the MCU's capabilities, while the reference manual provides complete information about the MCU and its peripherals.

- **STM32L4 IoT node (B-L475E-IOT01A):**

  This is the development board that STM has built around the STM32L475VG MCU. This board provides all the necessary peripherals for the MCU (crystal oscillators, power supply, USB serial interface for programming etc.). The board also includes several useful peripherals and sensors (sensors for temperature, pressure, acceleration, etc., microphones, and much more). STM has provided a user manual for the board, which contains a description of the peripherals as well as board schematics.

Base-project

A base-project **GXX_Lab2.7z** has been provided for this lab. The Keil project can be found in the *GXX_Lab2/MDK-ARM/* directory. The project contains code that initializes the MCU and the board, and also includes code to initialize the componenets used in the lab. In particular, the *main()* function consists of 5 function calls. The *HAL_Init()* and *SystemClock_Config()* functions initialize the MCU and on chip hardware, and set up the system clock. The *MX_GPIO_Init()* initializes the LED GPIO in output mode and the push-button GPIO in input mode. *MX_DAC1_Init()* initializes both channels of the DAC. Finally, *HAL_GPIO_WritePin()* turns on the LED. This information will be discussed in more detail in the following sections.

IMPORTANT: If you are using the lab computers, please ensure that you save the project in C:\Users\<your_name_goes_here> otherwise it might not work in Keil.

Using GPIO's in digital mode: Push-button and LED

In this first task you will learn how to configure a GPIO pin to act as a digital input or output pin. The task is to use the blue push-button **B2** to control the green LED **LD2**.

The first step is to identify the correct GPIO pins associated with these peripherals. The GPIO for **LD2** is found readily in Section 7.14, pp. 28-29 of the board user manual, and again in Appendix A, Table 11, p. 38. Verify that the pin you have identified for **LD2** is *PB14*.

Identifying the GPIO for the blue push-button **B2** involves a slightly more complicated trajectory through the same manual. Begin in Section 7.14, pp. 28-29, then visit Appendix B, Figure 31, p. 51, and finally end up in Appendix A, Table 11, p. 37. Verify that the GPIO you have identified for **B2** matches the one in the *MX_GPIO_Init()* function provided in **main.c**

Upon studying the *MX_GPIO_Init()* function, you will see that the desired pins have been set in input and output mode using the HAL drivers. Study the material in Section 28, pp.387-393 of the HAL driver user manual to ensure that you understand how this function works.

You should now be equipped to control **LD2** using **B2**. Write code within the *while* loop in the *main()* function to check if the push-button is pressed. If so, turn the LED on, otherwise turn it off.

NOTE: If the code is not written in the *while* loop, it will only execute once and terminate.

You may find that your code causes the LED to turn off when the button is pressed, and turn on when the button is released. If this is the case, studying Figure 31 in Appendix B, p. 51 of the board user manual and make sure you understand why this is happening.

Refer to Chapter 8, pp. 294 to 312 in the STM32L475VG reference manual to learn a great deal about the GPIO pins. While it is not essential to succeed in this lab, it will give you an excellent understanding of how they work.

Using GPIO's in analog mode: DAC

In this task, you will configure both channels of the on-chip DAC to generate a square wave on two GPIO's. You will then probe the GPIO's using the oscilloscopes in the lab. The peak-to-peak voltage of the waveform shape must reach the maximum and minimum of the DAC (0V to 3V)
In this section you will be pointed to all the documentation you need to refer to, as well as the specific driver functions you will need. The code itself is at most 10 lines and very simple. The actual work involves digging through the documentation to understand what driver functions you need and why, as well as what arguments to pass to them.

To begin, refer to Section 14, pp. 217-242 of the HAL driver user manual. There, you will find information that explains the *MX_DAC1_Init()* function. More importantly, the only two driver functions you will need in the task are found here: *HAL_DAC_Start()* and *HAL_DAC_SetValue()*. ***Hint:*** You need to use *HAL_DAC_Start()* before the *while* loop, and *HAL_DAC_SetValue()* within the *while* loop.

The value you will write to the DAC is either 0 or 255/4095, by comparing to a simple unsigned integer that is incremented on every iteration of the *while* loop. The max value should be 4095 or 255, depending on whether you have chosen 8 or 12 bit precision (the choice is yours).

Chapter 4, Table 16, p. 62 of the STM32L475xx datasheet provides the GPIO pins associated with the DAC. Make a note of the pins associated with both channels. To locate where these pins appear on the board (so that you are able to scope them), first refer to Appendix A, Table 11, p. 37 and then to Appendix B, Figure 30, p. 50 of the board user manual.

Refer to Chapter 19, pp. 619 to 653 in the STM32L475VG reference manual to learn a great deal about the DAC. While it is not essential to succeed in this lab, it will give you an excellent understanding of how the DAC works.

Sine Wave Generation

Next, generate a sinusoidal wave. You can try the sine/cos functions from ”math.h”, the ”arm_sin_f32” from the DSP libraries, or simply copy-pasting the values into an array declaration. The peak to peak value should be equal to the DAC limits of the DAC (3V). Other characteristics of the wave can be set as you choose.