2024

EEE3095/6S: PRACTICAL 3

1. OVERVIEW

Many embedded systems rely on the use of analog devices for various reasons. Certain sensors, for example, output an analog voltage or current that the digital brain of the system must then be able to read. Similarly, embedded devices often allow for users to adjust parameters by turning the knob of a rotary potentiometer. For us to be able to process these analog signals, we thus need to make use of an Analog-to-Digital Converter (ADC).

In this practical, you will be using C to read the voltage on a potentiometer's wiper using the on-board ADC on your STM32 board. You will then use that to calculate the duty cycle of a PWM signal to control the brightness of a built-in LED (pin PB0), and a pushbutton interrupt will be used to control the change the toggling frequency of LED PB7 with precise timing.

Finally, this practical will dive into a bit of embedded communications — namely by using Serial Peripheral Interface (SPI) to write data to the EEPROM (non-volatile memory that retains its contents after power cycling) chip on your UCT STM board, and then read this data back later to write to the LCD. Fun!

2. OUTCOMES AND KNOWLEDGE AREAS

In this practical, you will be using C code (and the HAL libraries) to interface with your microcontroller board to read data from your ADC, output it to the LCD, and use it to adjust the duty cycle of LED PB0. Another LED, PB7, must also toggle on/off every 1 second by default, and this can be adjusted using a pushbutton interrupt on PA0 to alternate the toggling frequency.

Additionally, you will be using SPI to interact with the EEPROM chip on your dev board — namely, to write data to EEPROM and then read from it. You will also have your remaining seven LEDs (PB0 to PB6) flashing in a pattern based on the value being read by SPI, and this will change at an interval defined by your timer settings (with a timer interrupt being used).

You will learn about the following aspects:

- ADCs
- SPI
- Interfacing with the LCD

3. DELIVERABLES

For this practical, you must:

- Develop the code required to meet all objectives specified in the Tasks section
- Push your completed main.c code to a shared GitHub repository for you and your partner. Your folder and file structure should follow the format:
  STDNUM001_STDNUM002_EEE3096S/Prac3/main.c
- Demonstrate your working implementation to a tutor in the lab. You will be allowed to conduct your demo during any lab session before the practical submission deadline.
- Write a short report (max 2 pages) documenting your main.c code, GitHub repo link, and a brief description of the implementation of your solutions. This must be in PDF format and submitted on Amathuba/Gradescope with the naming convention:
  EEE3096S 2024 Practical 3 Hand-in STDNUM001 STDNUM002.pdf
  Note: Your code (and GitHub link) should be copy-pasted into your short report so that the text is fully highlightable and searchable; do **NOT** submit screenshots of your code (or repository link) or you will be penalised.
- Your practical mark will be based both on your demo to the tutor (i.e., completing the below tasks correctly) as well as your short report. Both you and your partner will receive the same mark.

4. GETTING STARTED

As before:

1. Clone or download the (updated) Git repository:
   $ git clone https://github.com/eee3096s/2024
2. The project folder that you will be using for this practical is /eee3096s/2024/Prac3
3. Open STMCubeIDE, then go to File --> Import --> Existing Code as Makefile Project -->> Next. Then Browse to the project folder above, select it, and select "MCU ARM GCC" as the Toolchain --> Finish.
   Note: This IDE provides a GUI to set up clocks and peripherals (GPIO, UART, SPI, etc.) and then automatically generates the code required to enable them in the main.c file. The setup for this is stored in an *.ioc* file, which we have provided in the project folder if you would ever like to see how the pins are configured. However, it is crucial that you do **NOT** make/save any changes to this *.ioc* file as it would re-generate the code in your main.c file and may **delete** code that you have added.
4. In the IDE, navigate and open the main.c file under the Core/src folder, and then complete the Tasks below.

Note: All code that you need to write/add in the main.c file is marked with a "TODO" comment; do not edit any part of the other code that is provided.

5. TASKS

Complete the following tasks using the main.c file in STM32CubeIDE with the HAL libraries, and then demonstrate the working execution of each task to a tutor:

1. The leftmost LED, pin PB7, has been set to flash at a default frequency of 2 Hz using a simple delay function. Use a pushbutton interrupt to switch the frequency of PB7 by toggling between 1 Hz and 2 Hz whenever the pushbutton PA0 is pressed. Remember to debounce your button presses on PA0!
2. With our potentiometer (POT1, pin PA6) and the ADC already configured, complete the function *pollADC* to read the output ADC value from the potentiometer's wiper.
3. The function *ADCtoCCR* converts an ADC value to a CCR (Capture/Compare Register) value, which sets the PWM duty cycle. The ADC is configured in 12-bit mode, and TIM3 is configured to use Channel 3 (i.e., LED PB0) with a prescaler of 0 and an ARR (Auto Reload Register) value of 47999; this corresponds to a 1 kHz frequency for the PWM signal, and *Duty Cycle* = *CCR /ARR*. Using this information, update the CCR based on the ADC value from POT1, such that turning POT1 adjusts the brightness of LED PB0.
4. Initialise an array of 8-bit integers that holds the following six binary values:
   10101010, 01010101, 11001100, 00110011, 11110000, 00001111
5. SPI has been initialised for you. Write code that takes the above array of 8-bit values and writes them all to EEPROM using SPI; use the provided *write_to_address* function for this.
6. TIM16 has been initialised for you with a 1-second timer; in the interrupt handler, use the provided *read_from_address* function to read the next binary value of your array from EEPROM and then prints this value to the LCD as a **decimal** number (call it **`X`**) using the two-line format:
   `EEPROM byte:`
   `X`
   Hint: You will need to complete the *writeLCD* function to accomplish this. The LCD library's implementation (C file included under folder "Core/Inc") should show you how to interface with the LCD.
7. In your TIM16 interrupt handler, add failsafe code that checks whether the 8-bit value read from EEPROM corresponds to the correct value in your array; if not, output the following to your LCD to indicate SPI failure:
   `EEPROM byte:`
   `SPI ERROR!`

You will need some kind of counting mechanism to keep track of the expected array value.

6. GRADESCOPE SUBMISSION

1. Submit a single report on Gradescope for your group
2. Remember to link the submission to your partner. (The same report must not be submitted by each member of the group)
3. The first page of the report document should be a copy of the practical demonstration marksheet
4. Link the page of your report to the appropriate section ie.
   i. The copy of the practical demonstration marksheet to the “Practical Demonstration” section
   ii. The first page of the report to the “Report” section