2024

EEE3095/6S: PRACTICAL 1

1. OVERVIEW

From your previous working with the STM board, you should already be aware of how to interface with some of the basic board components, namely the LEDs and pushbuttons. This practical will recap both briefly and extend your knowledge of embedded systems to another major topic: timers!

The use of timers is one of the key features of many embedded systems as timers are useful for many things — from simply displaying time or a countdown (like most microwaves ovens) to using it to schedule operations as needed (such as in household appliances like ovens, dishwashers, etc). In this practical, you will thus cover how to create delays with a timer and how to invoke a timer interrupt to run a portion of code at regular intervals.

This practical will also introduce you to the **STM32CubeIDE** software as well as the Hardware Abstraction Layer (**HAL**) libraries — both of which will allow you to program your development board using C with significantly less hassle than ever before! Additionally, by the end of this practical, you should also have a basic understanding of **Git** (more information is available here).

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 flash LEDs in a pattern, and this will change at a specified interval defined by your timer settings. You will also be using your pushbuttons to alternate the delay timing.

You will learn about the following aspects:

- STM32CubeIDE and the HAL libraries (documentation is available here)
- LEDs and pushbuttons
- Timers and timer interrupts
- Git

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/Prac1/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, 2pg, report documenting the 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 1 Hand-in STDNUM001 STDNUM002.pdf
  Note: Your code (and GitHub link) should be copy-pasted into your short report, as an appendix, 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.
- Note: Ensure "STDNUM001 STDNUM002" is in the correct order, according to the group partner sheet.

4. GETTING STARTED

1. Follow the instructions at http://wiki.ee.uct.ac.za/Git to configure Git on your computer
2. Clone or download the Git repository:
   $ git clone https://github.com/gdhjam001/EEE3096S-2024
   From this, the project folder that you will be using is /EEE3096S-2024/Practical1/Prac1_student
3. You will also need to download and install STMCubeIDE (our IDE for this course).
4. After installing and opening STMCubeIDE, go to File --> Import --> Existing Code as Makefile Project -->> Next. Then Browse to the above project folder, 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.
5. 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. TASK OUTCOMES

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. On a continuous cycle, display the following patterns on the LED array. The pattern must change every one second.

PAT No.	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8
1	1	1	1	0	1	0	0	1
2	1	1	0	1	0	0	1	0
3	1	0	1	0	0	1	0	0
4	0	1	0	0	1	0	0	0
5	1	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0	0
7	0	1	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0

2. Write code to change the timer cycle delay to a:
    a. 0.5 s delay when Pushbutton 0 (pin PA0) is pressed
    b. 2 s delay when Pushbutton 1 (pin PA1) is pressed
    c. 1 s delay when Pushbutton 2 (pin PA2) is pressed
3. When Pushbutton 3 (pin PA3) is pressed the sequence is reset back to pattern 1.
4. Throughout the code development maintain a good version control history on Git. After significant milestones have been achieved a commit should be made. The commit history should include contributions from both members of the group.

6. TASK DIRECTION

1) Define any input variables you may need. Consider how you will store and keep track of the LED pattern and pattern number.
   A timer (TIM16) has been configured and initialised for you using the *.ioc* file, and an interrupt handler called "TIM16_IRQHandler" has been made available to you. As part of your main function, start the TIM16 process in interrupt (IT) mode. Use the TIM16_IRQHandler function (which triggers every 1 second) to change the LED pattern.

2) Use TIM16's ARR value in your implementation.
3) Questions to think about:

   You may notice that your button presses are not always registered, and sometimes the pattern changes will only occur after a delay. Think about why this may be.

   What is the fundamental clock speed of your micro controller?