EEE3095/6S: Practical 1A

STM32 Developer Board Refresher(GPIOS, Timers & Timer Interrupts)

July 29, 2025

1 Overview

From working with the STM32 previously, you should know how to interface with some basic board components, namely the LEDs and pushbuttons. This practical will recap both of these 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. It is helpful for many things, from simply displaying time or a countdown (like most microwave ovens) to scheduling operations as needed, for example, in other household appliances such as ovens, dishwashers, etc. In this practical, you will thus cover how to create delays with a timer and invoke a timer interrupt to run some code at regular intervals.

In this practical session, you will learn how to use a timer to create delays, handle button inputs with debounce functionality, and implement multiple LED flashing modes. You will also improve your familiarity with the STM32CubeIDE software and the hardware abstraction layer (HAL) libraries. These tools simplify the programming of your C development board by abstracting low-level hardware interactions. By the end of this practical, you will have gained a deeper understanding of GPIO control, timer configuration.

2 Outcomes and Knowledge Areas

In this practical, you will be using C to write code to flash all eight LEDs in a pattern based on the mode corresponding to the respective button press, which will change at an interval defined by your timer settings. You will also enable one of your push buttons to alternate the delay timing.

You will learn about the following aspects:

- LEDs and pushbuttons (recap)
- Timers and timer interrupts
- STM32CubeIDE and HAL libraries

3 Deliverables

For this practical, you must:

- Develop the code required to meet all objectives specified in the Tasks section
- Push your completed code to a shared repository on GitHub
- 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.
- Submit your main.c file. This must be in **PDF** format and submitted on Gradescope with the naming convention(**If you do not adhere to the naming convention, there will be a penalty**):

  EEE3096S 2025 Practical 1A Hand-in STDNUM001 STDNUM002.pdf

- 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 for Prac 1B. Both you and your partner will receive the same mark.

4 Getting Started

The procedure is as follows:

1. Clone or download the Git repository(The practical folder of interest is Practical1/Practical1_A:

   `git clone https://github.com/EEE3096S-UCT/EEE3096S-2025.git`

2. - Open **STM32CubeIDE**, then navigate through the menus:

     `File` → `Import` → `Existing Code as Makefile Project`

   - Click `Next`.
   - In the dialog, click `Browse...` and select your project folder.
   - Under **Toolchain**, choose `MCU ARM GCC`.
   - Click `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.**

3. 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

You are required to complete the following tasks:

1. The first "TODO" requires you to define any necessary input variables eg array to hold the patterns maybe. You may also need to define other variables based on the Tasks below.
2. A timer (**TIM16**) has already been configured and initialised for you, including an interrupt called **TIM16_IRQHandler**. As part of your main function, start the **TIM16** process in interrupt mode.

   HINT: The **HAL** library will be useful here, particularly one of the **HAL_TIM** functions.
3. The **TIM16_IRQHandler** function is invoked whenever the timer interrupt occurs (approximately every 1 second). Write code to toggle between the LED modes on their respective button presses, i.e PA1 activates mode 1, PA2 activates mode 2, and PA3 activates mode 3. You will need away to keep track of the current LED Mode and/or current LED on/off, so add logic to handle that.
4. **TIM16** has been configured to trigger an interrupt every 1 second. As part of your main function's while loop, write code to alternate between a **1-second** delay and a **0.5-second** delay whenever Pushbutton 0 (pin PA0) is pressed.

   HINT: Use TIM16's ARR value in your implementation. **NOTE:** You may notice that your button presses are not always registered; think about why this may be — it will be covered in your lectures later.

The end result should be a system that does the following:

- On system startup up all LEDS should be off
- On every timer interrupt, display the LED pattern corresponding to the mode the system is in.
- Upon the press of PA1, the system should transition to mode 1:

  PA1: Mode 1 back/forth
  In this mode, only one LED is on, cycling between each of the 8 LEDs from LED0 to LED7, and then back from LED7 to LED0. However, when swapping round, it should right away go to the next LED, not put on the same LED that was on in the previous period. i.e. the display (in slow speed) is shown in Figure 1:

t	LEDS
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000
8	01000000
9	00100000

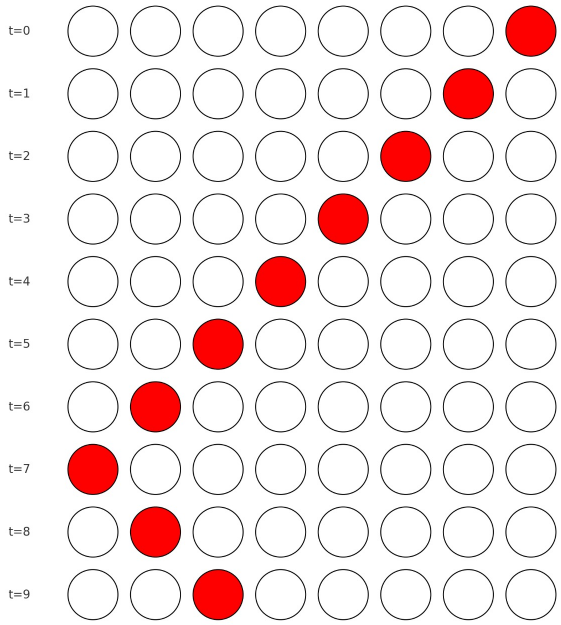

Figure 1: Mode 1 back/forth

- Upon the press of PA2, the system should transition to mode 2:

  PA2: Mode 2 inverse back/forth
  This mode is the opposite of Mode 1, where all LEDs are on except one, the LED that is off cycles between LED0 to LED7, and then back from LED7 to LED0. i.e. the display (in slow speed) is shown in Figure 2:

t	LEDS
0	11111110
1	11111101
2	11111011

3	11110111
4	11101111
5	11011111
6	10111111
7	01111111
8	10111111
9	11011111

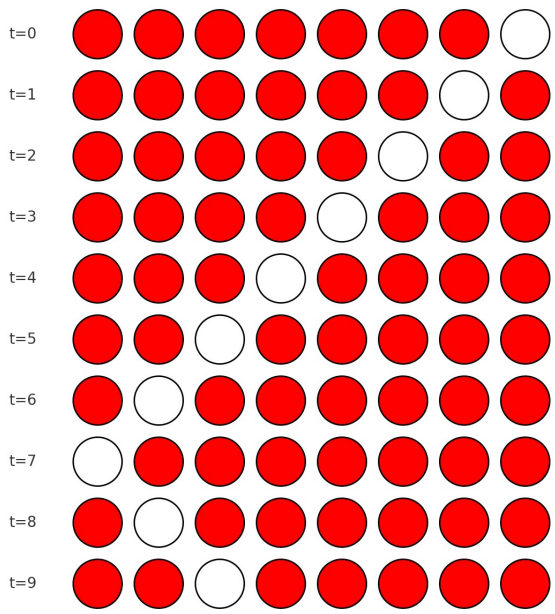

Figure 2: Mode 2 inverse back/forth

- Upon the press of PA3 should transition to mode 3:

  PA3: Mode 3 Sparkle
  In this mode, we want the LEDs 'sparkling', basically a random number between 0 and 255 displayed, and then held for a random delay (100 to 1500ms) and then each LED turned off one at a time with a delay of random(100)ms between each being turned off. So it looks something like shown in Figure 3:

```
0 0 0 0 0 0 0 0   starts all dark
1 0 1 0 0 0 1 0   various ones come on, i.e. LEDs=random(255)
1 0 1 0 0 0 1 0   held for random(100)
1 0 0 0 0 0 1 0   one of LEDs goes off, held for random(100)
0 0 0 0 0 0 1 0   one of LEDs goes off, held for random(100)
0 0 0 0 0 0 0 0   last one off, held for random(100)
... and repeats ...
```

6 Helpful Tips

1. Read through the documentation for the HAL libraries.
2. The source code provided to you already has a lot of implementation. Ensure you read and understand it before embarking on the practical.
3. Some helpful information is available here, if you need help with understanding timer basics, such as the ARR and PSC

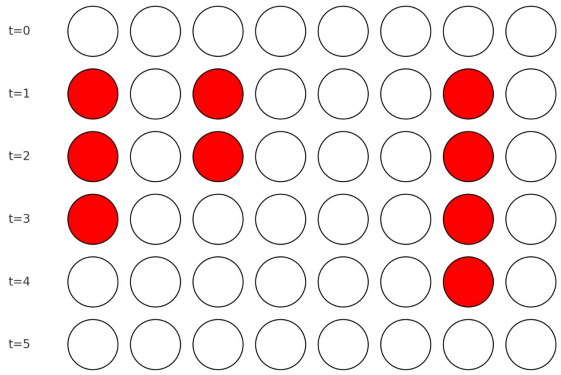

Figure 3: Mode 3 Sparkle