EEE3096S - Practical 2
2023

Timers and SPI

1 Overview

From working with the STM32 previously, 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 of these briefly, and will 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 and is useful for many things, from simply displaying time or a countdown (like most microwaves ovens) to using it to schedule operations as is 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 how to invoke a timer interrupt to run some code at regular intervals.

Finally, this practical will dive into a bit of embedded communications — namely through SPI, of which you should have a decent understanding by now. We will be using SPI to write data to the EEPROM (non-volatile, user-modifiable ROM that retains its contents during power cycling) on your UCT STM board, and then this data will be read back into another variable later; then these values will be displayed on your LEDs to create pretty patterns!

2 Outcomes and Knowledge Areas

In this prac, you will be using C to write to EEPROM and read from it using SPI. You will also have all eight LEDs flashing in a pattern based on the value being read by SPI, and this will change at an interval defined by your timer settings. You will also be enabling one of your pushuttons to alternate the delay timing.

You will learn about the following aspects:

- LEDs and pushbuttons (recap)
- Timers and timer interrupts
- SPI
- You should acquaint yourself with the STM32 HAL/LL documentation, available here.

3 Deliverables

For this practical, you must:

- Submit a **PDF** of your main.c code on Amathuba/Gradescope. Do this by copy-pasting your code into a document and saving it as a PDF; do NOT submit screenshots of your code as the text needs to be searchable.
- After you have submitted your PDF, push your code to your shared repository on **GitHub** and then **demonstrate** your working implementation to a tutor in the lab; they will open your submission on Amathuba and mark your demo in real time.
- You will have **two weeks** to complete this practical, and as such, you will be allowed to conduct your demo during any lab session within these two weeks. By the deadline, you should have submitted your PDF, pushed your code to GitHub, and had your demo marked.

4 Hardware Required

- UCT STM Board

5 Walkthrough

1. Clone or download the git repository.
   ```
   $ git clone https://github.com/UCT-EE-OCW/EEE3096S-2023
   ```
2. /EEE3096S-2023/WorkPackage2/Prac2_student is the project folder that you will need. You will also need to download and install STMCubeIDE (our IDE for this course).
3. After opening STMCubeIDE, go to File --> Import --> Existing Code as Makefile Project --> Next --> Browse to the project folder and then 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 is stored in a .ioc file, which we have provided if you want to see how the pins are configured — but do NOT make/save any changes to this as it would re-generate the code in your main.c file.
4. Open up the main.c file under the Core/src folder and complete the Tasks below. Note: All code that you need to complete is marked with a TODO comment; do not edit any of the other provided code.

5. **TASK 1:** The first "TODO" requires you to define any input variables you may need. One of these should be an array of 8-bit integers that holds the following binary values:
10101010, 01010101, 11001100, 00110011, 11110000, 00001111
You may also need to define other variables here based on the Tasks below.

6. **TASK 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 (IT) mode. HINT: The HAL library will be useful here, particularly one of the HAL_TIM functions.

7. **TASK 3**: SPI has already been initialised for you as part of the *main* function. Write code that takes your array of binary values and writes them all to EEPROM using SPI; use the provided write_to_address function as part of your work, taking note of the variable types involved.

8. **TASK 4**: The TIM16_IRQHandler function is invoked whenever the timer interrupt occurs (approximately every 1 second). Use the provided read_from_address function to read the next binary value of your array from EEPROM. HINT: You will need some way to keep track of the binary array index (i.e. the address input required by this function), so include logic to cycle to the next binary value each time the interrupt occurs. Write the returned binary value to the LEDs.

9. **TASK 5**: Include failsafe code in your interrupt that checks whether the 8-bit value read from EEPROM corresponds to the correct value in your array; if not, output 0b00000001 to your LEDs to indicate an SPI failure.

10. **TASK 6**: 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 this 1-second delay and a half-second delay whenever Pushbutton 0 (pin PA0) is pressed. 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 Practical 3!

11. The end result of all this should be code that achieves the following:

12. On system start-up, SPI is used to write the aforementioned array of 8-bit integers to EEPROM (once off).

13. On every timer interrupt, one of these 8-bit values must be read from EEPROM (using SPI) and then written to the eight red LEDs to produce a pattern. By default, the delay between interrupts should be 1 second.

14. Pressing PA0 should switch the timer delay from 1 second to 0.5 seconds, or vice-versa.

Demonstrate the above to a tutor.

11. Upload your main.c file to your shared repository. Your file structure should resemble this: STDNUM001_STDNUM002_EEE3096S/Prac2/main.c (assuming your shared repository is called STDNUM001_STDNUM002_EEE3096S).

6 Some Hints

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

7 Submission

A PDF submission is required for this practical:

1. The PDF document must comprise your main.c code in **text form**, not screenshots.
2. After submitting the PDF, demo the aforementioned functionalities to a tutor in the lab. Your PDF **must already be submitted** by the time you call a tutor to your bench for the demo, as they will then be able to open your submission on Amathuba/Gradescope and assign marks accordingly.
3. All marks for this practical will come from your demo and your submitted code.