ELEC335 Lab3

November 23, 2020

The objective of this lab is to give you a thorough understanding of *exception operation, fault generation and handling, external interrupt and managing multiple sources, and priority*. You will use C language for the problems unless some parts require *inline assembly*. An example about how to add assembly instructions to your code is given in the explanations.

Use *blinky* project from stm32g0 repo as the starting point for your problems. You will need to examine *startup_xxx* file to figure out exception handler function names, and implement those functions in your code.

Submission

You should submit the following items organized in a folder:

- **Lab report** - Written in English. Proper cover page, intro, problems, flow charts, block diagrams, schematic diagrams, comments, and theoretical, mathematical work, simulation vs.
- **Source files** - should have proper comments and pin connections if there are any external components.
- **Video link** - A link to video demonstration of your lab (this video also should show yourself.)
- **Elf files** - generated elf file with **debug information**.

Compress the folder to a zip file, and submit that way. Each problem should be in a separate folder. Example folder structure is given below.

```
name_lastname_number_labX/
  name_lastname_report_labX.pdf
  video_link.txt
  problem1/
    problem1.s/c
    problem1.elf
    ...
  problem2/
    problem2.s/c
    problem2.elf
    ...
```

Problems

Problem 1

In this problem, you will work on fault generation and handling.

- Create a **HardFault Handler** that will detect possible faults. The handler should properly restore stack pointer, and call reset handler afterwards. After compiling, examine the routine to make sure there are no overheads. The exact name of this function can be seen in **startup_xxx** file under *include* directory.

- Explain the type of faults that we discussed in the class and implement a routine for each case. Observe the hardfault operation, check the context after a fault occur, and explain the behavior.
- Attach an external button to your board, and using polling method execute these hardfaults randomly. For this you can use `rand()` C library function.
- Optionally you can implement a simple toggle LED mechanism in the beginning of your code to see the reset operation.

You might need to use inline assembly for handling routine which can be used as

```
1  __asm("movs r0, r0");
```

in your C functions.

Problem 2

In this problem, you will work with external interrupts.

- Setup and implement **an external interrupt routine** with highest possible priority, and tie it to your external button. Upon pressing, your interrupt routine should execute the previously implemented hardfaults randomly.
- When interrupt happens, check out the context and explain each section.
- Your main while loop should be empty for this problem since everything will be taken care of in the interrupt handler routine.

External interrupts are used to detect edge transitions on external signal. Both rising and falling edge transitions can be configured and detected if the hardware supports it. For Cortex-M0+, external interrupts are connected to an external interrupt MUX, and each pin can be configured with the desired edge trigger mechanism.

First make sure to read **Section 12 on RM0444**, and understand external interrupt operation. (You can skip the parts about wakeup) Overall, to enable an external interrupt:

1. Depending on your external button connection pin and configuration (active-high or -low) enable rising or falling edge trigger from `EXTI_RTSR` or `EXTI_FTSR` registers. Your connected pin will be the bit that you should enable (i.e. write 1). For example, if you connect the button to PC8 pin in an active-high configuration, you should write `(1 << 8)` to `EXTI_RTSR`.
2. Write the source port to the appropriate register `EXTI_EXTICRx`. There are a total of 4 `EXTI_EXTICRx` registers, and each register is used for selecting the source port for 4 pins. Check Table 55 on RM0444 and relevant register definition for specific configurations. For PC8 example, from Table 55 on RM0444, pin 8 can be configured to PortC by writing `0x02` the first 8 bits of `EXTI_EXTICR3` register.
3. Enable external interrupt from NVIC, and set priority.
4. Create external interrupt handler.

Problem 3

In this problem, you will work with multiple sources for external interrupts and priorities.

- Connect two buttons and two LEDs to your board. Enable external interrupt for both of them. Each button should toggle one LED.
    - Connect one button in one of the following groups: {0,1}, {2,3}, {4-15} and the other in the remainin group.
- You will use two external interrupt handlers. Make sure to check using the pending bit if the correct interrupt is received.

- Assign different priorities to your buttons, and inside the handlers create a long delay that will last for at least 5-10 seconds)
    - You can implement this behavior by turning on the associated LED in the beginning, then an empty for loop, then turning off the associated LED.
- Observe the preemption by trying to interrupt the current interrupt routine. (i.e. while one of the LEDs is on, press the other button to interrupt.) Explain your observations.