Lab1 Sensors/Actuators

Polytech Nice Sophia

You will be asked to answer the following questions on the statement R1, R2, …
You will have to write, test and then show the teacher the following codes C1, C2, …

Open the STM32CubeMX software, in File select "Load project" and select the provided `EDU32F103_v0.4.ioc` project.
Generate the code, and start editing the `main.c` file in the `Core/Src/` directory
The application can be compiled and flashed to the board either from VS Code or from the command line.
If the microcontroller cannot be flashed, press the BOOT0 button while connecting the card to put it into bootloader mode and be able to flash a fresh firmware.

1) Management of peripherals

Create a first program to manage the following peripherals :

- 8× LEDs,
- 4× 7-segment displays,
- 8× switches.
- The purpose of this first code is to find the display table of the 10 decimal digits on the 7-segment displays. You will write the configuration read on the switches on the LEDs and on the first 7-segment display (on the right), the others remaining at 0.

To do this, use the two functions to read and write a single GPIO pin:

- `HAL_GPIO_WritePin()`
- `HAL_GPIO_ReadPin()`

Or for an entire GPIO port, use its output and input registers directly, for example:

- `SEVENSEG0_A_GPIO_Port->ODR`
- `SW0_GPIO_Port->IDR`

R1. Pins configuration is set from STM32CubeMX and main.h, list all GPIOs of this microcontroller and the devices associated.

GPIO	Device	Protocol	Direction

C1. Test the values on the switches.

R2. Fill in the following display table (Appendix 2).

Digit	Binary configuration	Hex configuration
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

This table should be used to define a translation table or LUT (Look Up Table) :
int LUT [10] = {config0, config1, config2, ... config9} ;
where each configuration will be written in hexadecimal, for example: 0xff.

C2. Test by sending the value of a counter from 0 to 9999 on the 4 7-segment displays.
The wait between 2 read/write will be done by the `HAL_Delay()` function

2) Programmation of interrupts

Now that the peripherals are properly configured, we will program the processor interrupts to read the push button configuration.
As for any sensor or input device reading, two methods of reading are possible:
- by polling,
- by interruption.

R3. Recall the advantages and disadvantages of each method.

Access method	Pros and cons
Polling	
Interruptions	

Before programming the interruptions, some modifications must be done. Buttons are configured as standard GPIO Input by default, which does not give access to interruptions. This can be modified in the pinout view of STM32CubeMX. The GPIOs that need to trigger an interrupt should be changed to GPIO_EXTI*. Note that the pin user label must stay the same. The corresponding interrupt lines should then be enabled in the NVIC tab of the GPIO panel in System Core.

Programming of interrupts can be done using the interrupt handler:

```
void HAL_GPIO_EXTI_Callback()
```

R4. On your current application, LED3 and LED4 are disabled, find the reason using Pinout & Configuration view. How can this be solved and what are the pros and cons of this solution ?

C3. Write a code that indicates the number of the button pressed on the 7-segment displays. The LEDs should light up to indicate the switches in the up position. This switch configuration will be used to program the waiting time in the main loop (the more switches the user lifts, the longer the maximum waiting time is).

3) Reflex game

C4. Now that the peripherals are programmable, create a code that meets the following specifications:

- to start the test, the user must press button 1,
- the LEDs start blinking regularly (even and then odd LEDs...),
- all LEDs will then light up after a random time,
- the user should press button 4 (left) as quickly as possible,
- the system measures the user's reaction time between turning on all the LEDs and pressing the button
- the 7-segment displays shows the seconds and milliseconds. Use the default timer for this purpose (appendix 1).
- Switches are used to provide the maximum waiting time before ignition,
- a new reflex test is started by pressing button 1,
- by pressing button 2, the system displays the average reaction time since the beginning of the tests.

Time (s,ms)	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
User 1					
User 2					

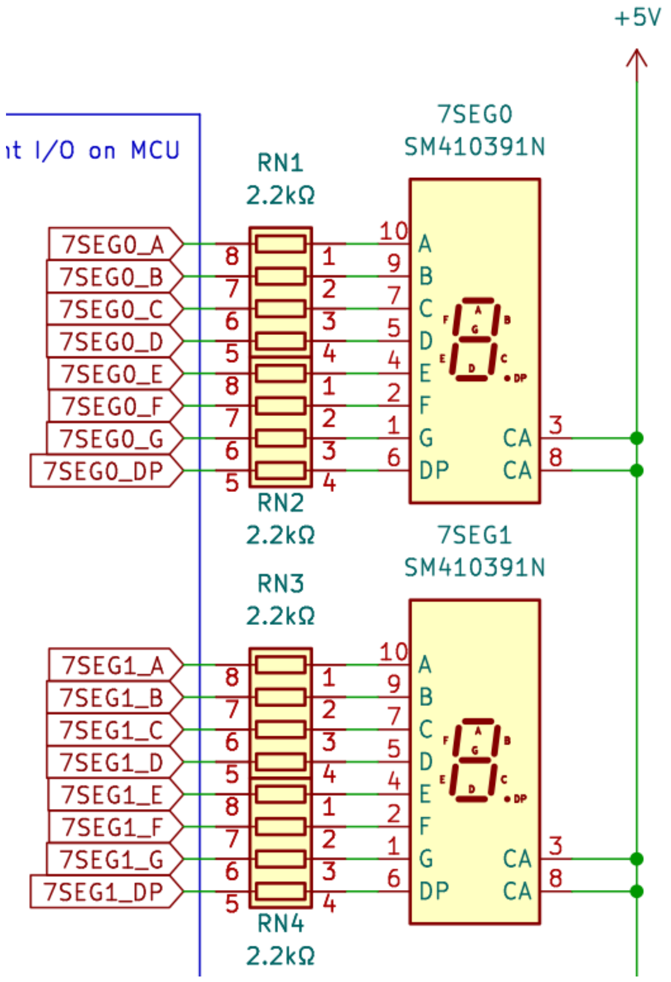
Appendix 1 – Timer

Use the existing SysTick timer

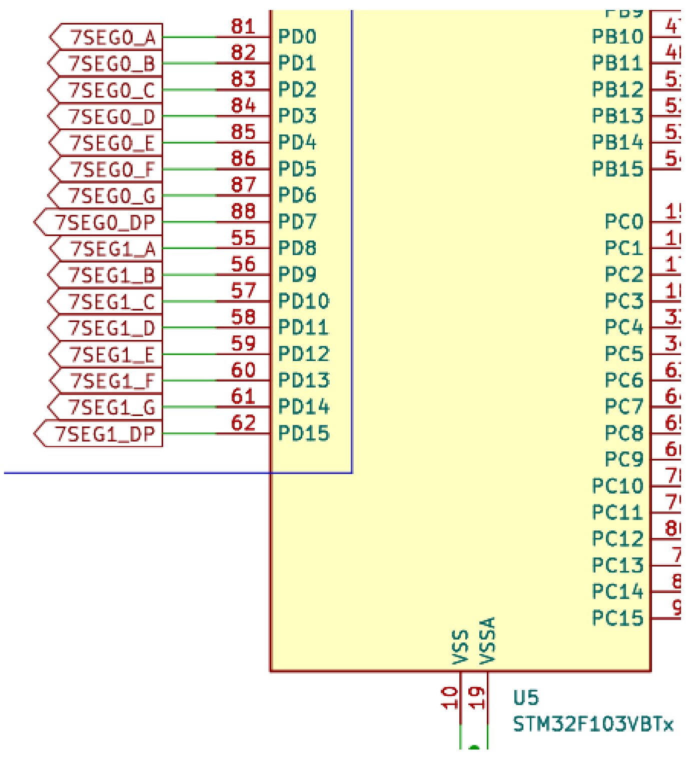
SysTick counts the number of "ticks" since startup.
The value of the timer ticks can be retrieved using the `HAL_GetTick()` function.
The **period** in ms (not frequency!) of the timer ticks can be retrieved using the `HAL_GetTickFreq()` function

Appendix 2 – 7-segment displays

Routing of 7-segment displays on the PCB (example on 7SEG0 and 7SEG1) :



+5V
t I/O on MCU
RN1
2.2kΩ
7SEG0
SM410391N
7SEG0_A
7SEG0_B
7SEG0_C
7SEG0_D
7SEG0_E
7SEG0_F
7SEG0_G
7SEG0_DP
A
B
C
D
E
F
G
DP
CA
CA
RN2
2.2kΩ
RN3
2.2kΩ
7SEG1
SM410391N
7SEG1_A
7SEG1_B
7SEG1_C
7SEG1_D
7SEG1_E
7SEG1_F
7SEG1_G
7SEG1_DP
RN4
2.2kΩ



7SEG0_A 81 PD0
7SEG0_B 82 PD1
7SEG0_C 83 PD2
7SEG0_D 84 PD3
7SEG0_E 85 PD4
7SEG0_F 86 PD5
7SEG0_G 87 PD6
7SEG0_DP 88 PD7
7SEG1_A 55 PD8
7SEG1_B 56 PD9
7SEG1_C 57 PD10
7SEG1_D 58 PD11
7SEG1_E 59 PD12
7SEG1_F 60 PD13
7SEG1_G 61 PD14
7SEG1_DP 62 PD15
PB10
PB11
PB12
PB13
PB14
PB15
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13
PC14
PC15
VSS 10
VSSA 19
U5
STM32F103VBTx