POLITECNICO DI TORINO

– Digital Systems Electronics –
Lab#9

Interrupts

The purpose of this laboratory session is to learn how to manage several tasks on the NUCLEO processors by the means of interrupts. Detailed information on how to program the board could be found on document [1] available on the website, and on the user [2] and reference manuals [3] .

Projects:

1 - Interrupt-based variable frequency square waveform generator
2.1 - Handling of three interrupts
2.2 - Handling of four interrupts
2.3 - Handling of five interrupts

Some of these projects require the availability of a few elementary measurement instruments, such as an oscilloscope and a waveform generator. For these projects, the LAB7 accompanying document describes an alternative option, which opens the possibility to complete the assigned projects with no needs for laboratory instruments. The guidelines and specifications useful to complete the alternative assignments are highlighted in bright green. See the Appendix in LAB7 accompanying document for details on the related viewer application.

Abbreviations and acronyms:

LED – Light Emitting Diode
MCU – Microcontroller Unit
TIM – Timer
ADC – Analog-to-Digital Converter

1 - Interrupt-based variable frequency square waveform generator

Polling is not always the best way to make things. Interrupts, differently from polling, do not occupy the processor in waiting loops, but *interrupt* the execution when an event occurs. In this project, you have to implement the same functions as in the last project of the previous lab; however, this time, you have to use an interrupt to trigger the toggling of the output pin. Do not use Toggle on match functionality, instead toggle the PA10 pin by means of a proper statement in the Interrupt Service Routine (ISR). The required steps are the following:

1. **Create a new project** using STM32CubeIDE. Select the proper configuration for
   a. the pin PA10, which provides the generated waveform,
   b. the counter TIM3, which works in Output Compare mode and triggers the ISR that toggles the pin,

c. the ADC, which converts the analog level provided by the potentiometer into a proper delay. Use the polling on the EOC (end of conversion) flag to read a new sample. Set preemptive priority for TIM3 interrupt.

2. **Fully understand the code generated** by STM32Cube.
3. **Add the instruction** given at the end of this assignment, and related to the SysTick function.
4. **Write a C program** that polls the flag of the end of conversion of the ADC and updates the correct register of the timer. You may need to perform mathematical operations on the converted value to reach the desired interval of frequencies. The TIM3 is configured in OC mode with no output generated and it is used in interrupt mode.
5. **Write the interrupt service routine (ISR)** for TIM3 (TIM3_IRQHandler).
6. **Compile** the project.
7. **Download** the compiled code on the Nucleo board.
8. Use a breadboard and the discrete potentiometer to **build the needed circuitry**. You have to use the NUCLEO board to give the supply voltage to the potentiometer. Follow the connections given in Figure *1*.
9. **Test** the functionality of the circuit by using the oscilloscope and varying the potentiometer.

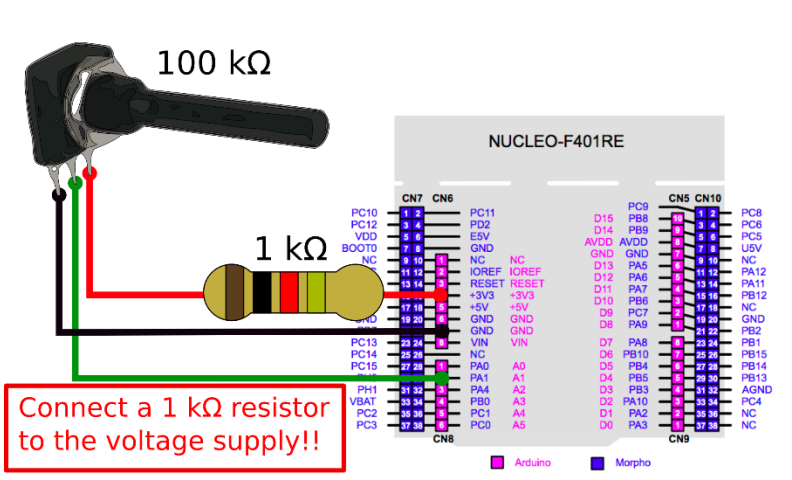

Figure 1: Connection of the potentiometer to the Nucleo board.

Measure the minimum and maximum period/frequency of the waveform; are they the same as in the previous laboratory session? Are the measures affected by any jitter?

Now we discover that the template provided for the previous lab already contained the management of an interrupt. This interrupt was intended to emulate the real behavior of an MCU, where the execution of a task can be interrupted by other tasks.
Looking at the code, you will discover that the *SysTick* was used to this purpose. In particular, there was a routine to simulate some tasks called periodically. Insert again the following function in your code, just as in the previous laboratory session.

1. Insert the following line of code in your *main* file, just before the "while(1)"

```
SysTick_Config(SystemCoreClock / 1000);
```

2. Locate file named "stm32f4xx_it.c" in your project
3. Copy the code below in the body of the function called "void SysTick_Handler(void)"

```
static int x=0x12c0; // What is this number ?
for(int i=0;i<x;i++);
x = (x >> 2) | (((x & 1)^(x & 2)) << 4);
```

Now try to remove the pre-emptive priority for the TIM3 interrupt. Do you see the same behavior on the oscilloscope? Is the waveform stable?

Project version with no need of measuring instruments.

If a laboratory oscilloscope is not available, you can complete the assigned project and display on the screen of your PC the generated waveform by exploiting the **waveform viewer** application described in the Appendix of the LAB7 accompanying document. The details of the viewer may be partially unclear at this time, because you did not yet study all peripherals that are involved in this application. However, you should be able to follow the described procedure. In addition, scale down the signal frequency by a factor 10 and use the 80 – 450 Hz range.
Finally, as the viewer application heavily uses the ADC, it is not recommended to connect the potentiometer to the ADC input to control the frequency of the waveform. Therefore, you have to exploit the user pushbutton to determine the frequency. Initially, just after launching the application, the generated waveform must have the lowest frequency (80 Hz); then, when you press the pushbutton, the frequency must grow to the next level in the table above (120 Hz) and at any further pushing of the button the frequency must update the waveform with the next frequency value in the table, up to 450 Hz. After reaching this frequency level, the next frequency update action must lead back to the 80 Hz.

2 - Multiple interrupts

In the previous project, you were asked to write the code to manage just one interrupt (plus the one connected to *SysTick*) . In the following, you will run several tasks on the MCU. Each task will be managed as an interrupt event through a specific routine.

2.1 - Three interrupts

Prepare a project and develop the C code to implement a 3-bit clock process working in interrupt mode. The 3-bit clock process relies on three channels of the Timer Count Register, namely channel 0, channel 1 and channel 2. Exploit the OC mode that automatically toggles an hardwired pin of the MCU. The interrupts produced by the output compare channels are used as follows:

1. channel 0 has to produce a square waveform with period T_0=1 ms,
2. channel 1 has to produce a square waveform with period T_1=2 ms,
3. channel 2 has to produce a square waveform with period T_2=4 ms.

The required steps are:

1. **Create a new project** using STM32CubeIDE; select the proper configuration for counter channels 1, 2 and 3 of TIM3.
2. **Fully understand the code generated** by STM32Cube.
3. **Write a C program** that starts the counter and then enter an infinite loop.
4. **Write the interrupt service routines (ISR)** for the three channels of TIM3.
5. **Compile** the project.
6. **Download** the compiled code on the Nucleo board.
7. **Test** the functionality of the circuit by using the oscilloscope.

Are you able to see all the waveform stable on the oscilloscope? Explain.

Project version with no need of measuring instruments.

If a laboratory oscilloscope is not available, you can complete the assigned project and display on the screen of your PC the generated waveform by exploiting the **waveform viewer** application described in the Appendix of the LAB7 accompanying document. The details of the viewer may be partially unclear at this time, because you did not yet study all peripherals that are involved in this application. However, you should be able to follow the described procedure.

2.2 - Four interrupts

Modify the previous project to add the following task: toggle LED status (from on to off and vice versa) every time the pushbutton is pressed. Implement this feature as an interrupt.

The required steps are:

1. **Create a new project** using STM32CubeIDE, and select the proper configuration for counter channel 1,2 and 3 of TIM3, the LED and the pushbutton.
2. **Fully understand the code generated** by STM32Cube.
3. **Write a C program** that starts the counter and then enter an infinite loop.
4. **Write the interrupt service routines (ISR)** for all the channels of TIM3 and the pushbutton (EXTI15_10_IRQHandler).
5. **Compile** the project.
6. **Download** the compiled code on the Nucleo board.
7. **Test** the functionality of the circuit by using the oscilloscope and pressing the pushbutton.

Is the pushbutton affecting the square wave frequency? Does it cause any jitter? Explain.

Project version with no need of measuring instruments.

If a laboratory oscilloscope is not available, you can complete the assigned project and display on the screen of your PC the generated waveform by exploiting the **waveform viewer** application described in the Appendix of the LAB7 accompanying document. The details of the viewer may be partially unclear at this time, because you did not yet study all peripherals that are involved in this application. However, you should be able to follow the described procedure.

2.3 - Five interrupts

Just skip this project if you do not have an oscilloscope.

Modify the previous project to add the following task: use the potentiometer to change the frequency of the three square waves. The range should be:

1. channel 0 → 0.1-1 ms
2. channel 1 → 0.2-2 ms
3. channel 2 → 0.4-4 ms.

Furthermore, use channel 1 of the ADC as in the previous experience, but in single conversion mode. This implies that the ADC completes the conversion of a single sample, then it stops and waits for a new conversion trigger. You have to use channel 2 of TIM4 in OC mode to trigger a new conversion of the ADC every 500 ms. In addition to set the appropriate

prescaler and autoreload values, you have to also set the trigger output parameters. Moreover, the TIM4 must be configured in interrupt mode, to start a new conversion each time the counter value is equal to the CCR2. The interrupt must also be enabled for the ADC, but use the polling approach for mapping the new converted value into the corresponding frequency of each waveform.
The required steps are:

1. **Create a new project** using STM32Cube, and select the proper configuration for counter channel 1,2 and 3 of TIM3, channel 4 of TIM4, the LED, the pushbutton and the ADC.
2. **Fully understand the code generated** by STM32Cube.
3. **Write a C program** that starts the counters and then polls the ADC status (**in the main function**).
4. **Write the interrupt service routines (ISR)** for the three TIM3 channels, the ADC and channel 2 of TIM4.
5. **Compile** the project.
6. **Download** the compiled code on the Nucleo board.
7. **Test** the functionality of the circuit by using the oscilloscope, pressing the pushbutton and changing the value of the potentiometer.

Are you able to see all the waveforms stable on the oscilloscope? Do you see a different behavior when you change the potentiometer? Modify your code to trigger a new conversion every 100 ms and look again at the waveforms. Do you see any difference? Explain.

3 - References

[1] Lab9-STM32-document, Interrupts
[2] UM1724 User manual (STM32 Nucleo-64 board), Nov. 2016 (description of the Nucleo board).
[3] RM0368 Reference manual, May 2015 (guide to the use of memory and peripherals in the STM32).