– Elettronica dei Sistemi Digitali –

Lab#8

Square waveform generation and Potentiometer

The purpose of this laboratory session is to generate and measure a square waveform. Detailed information on how to program the board could be found either on the lab documentation available on the website or on the reference manual of the board. For each exercise, you have to follow all the assignment.
-If some questions are present at the end of the assignments, be ready to answer them during the lab interview.

Contents:

Abbreviations and acronyms:

MCU	– Microcontroller Unit
TCR	– Timer Count Register
ADC	– Analog-to-Digital Converter

1.0- Square waveform generator (8)

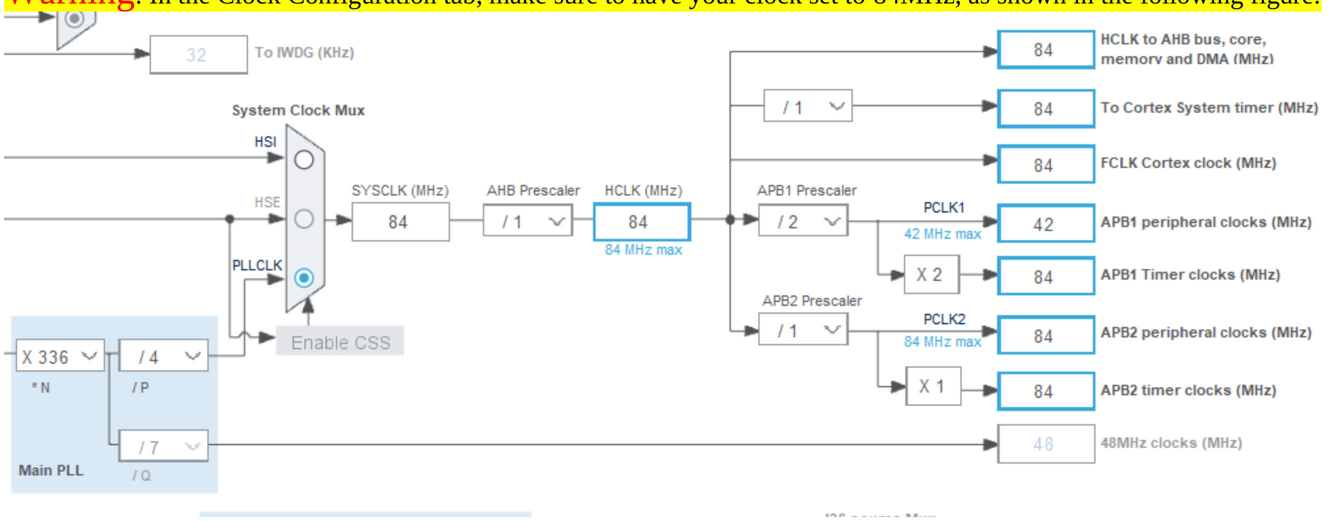
There are several ways to implement a square wave generator with a microcontroller. In the following, we will analyze some of the possible solutions. The precision of the obtained waveform depends on the technique used to generate it. In the first lab experience, you used a software loop to generate the waveform. This is not a good idea since MCUs usually perform different tasks and the main loop could be interrupted during execution. A specific peripheral is used to “count” time intervals: the free-running counter. In this lab experience, you are required to use one of the counters of the NUCLEO MCU to generate more stable square waves.

1.1-Timer Counter Register Polling (8.1)

In the first exercise we are going to use the TIM3 Counter to generate a 2.5 kHz square waveform on the PA10 pin. You have to poll the counter register and toggle the output pin only when the correct value is read. Detailed information on the counter configuration and usage can be found in the documentation. The required steps are the following:

1. **Create a new project** using the “STM32CubeIDE”.
2. **Configure the hardware** using “STM32Cube” selecting the proper configuration for the counter TIM3 and PA10. **Configure the source clock of TIM3 to 84MHz.**
3. **Fully understand the code generated** by the “STM32Cube”.
4. **Follow the instruction** described at the end of this document.
5. **Write a C program** that toggles the output pin and reset the counter when the counter value is equal to the *target*.
6. **Compile** the project.
7. **Download** the compiled code on the Nucleo board.
8. **Test** the functionality of the circuit by using the oscilloscope.

Is the waveform stable? Do you see any strange behavior?
What happens if you change the "==" with a ">="?
Now follow the pass described at the end of this exercise and perform again the test with "==" and ">=". Are there any differences? Why, in your opinion?
Warning: In the Clock Configuration tab, make sure to have your clock set to 84MHz, as shown in the following figure:



Some parts of this code could be unclear now, in particular the body of the function must be taken as it is without introducing any modification. The understanding of these portions is the topic of the next labs.

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)"

```
int x = (rand()%10 + 50)*5;
   for(int i=0;i<x;i++);
```

1.2-Timer Output Compare Flag Polling (8.2)

You should have noticed that the repeated comparison between the content of the TIM3 counter and a defined constant, may lead to completely wrong behavior and this kind of approach should not be adopted in any similar application. Furthermore, it is possible that you can't reset the counter value in any moment, so the approach of the previous exercise is discouraged. **You should never reset manually the counter since it could be used for more than one task.** The Output Compare (OC) function is exactly what we need. Each timer has multiple channels that could be configured differently. In this exercise you have to exploit the OC function to generate a square wave at 2.5 kHz using channel 1 of TIM3 and a 12.5 kHz square wave using channel 2 of the same timer, polling the corresponding flags. Use PA10 for the 2.5 kHz and PB10 for the other one.
The required steps are the following:

1. **Create a new project** using the "STM32CubeIDE".
2. **Configure the hardware** using "STM32Cube" selecting the proper configuration for pins PA10, PB10 and counter TIM3. **Configure the source clock of TIM3 to 84MHz.**
3. **Fully understand the code generated** by the "STM32Cube".
4. **Follow the instruction** described at the end of this document.
5. **Write a C program** that exploits the OC to generate the square waves, updating the value in the compare registers and toggling the output pins.

6. **Compile** the project.
7. **Download** the compiled code on the Nucleo board.
8. **Build the necessary circuitry**
9. **Test** the functionality of the circuit.

Are the waveforms stable? Do you see any strange behavior?
Now follow the pass described at the end of this exercise and perform again the tests. Is there any difference? Why, in your opinion?

Some parts of this code could be unclear now, in particular the body of the function must be taken as it is without introducing any modification. The understanding of these portions is the topic of the next labs.

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)"

```
int x = (rand()%10 + 50)*5;
   for(int i=0;i<x;i++);
```

1.3-Timer Output Compare Function (8.1, 8.2)

The OC Function of the timer peripheral has another useful property. You can set it in order to automatically toggle a specific pin in hardware when the match occurs. The only thing that you have to manage is the update of the target value for the OC. In this exercise you have to exploit the OC *Toggle on match* function to generate a square wave at 2.5 kHz using channel 1 of TIM3 and a 12.5 kHz square wave using channel 2 of the same timer, polling the corresponding flags to update the target values.
The required steps are the following:

1. **Create a new project** using the "STM32CubeIDE".
2. **Configure the hardware** using "STM32Cube" selecting the proper configuration for counter TIM3. **Configure the source clock of TIM3 to 84MHz**.
3. **Fully understand the code generated** by the "STM32Cube".
4. **Follow the instruction** described at the end of this document.
5. **Write a C program** that update the target values when each OC flag is set.
6. **Compile** the project.
7. **Download** the compiled code on the Nucleo board.
8. **Build the necessaire circuitry**
9. **Test** the functionality of the circuit.

Is the waveform stable? Do you see any strange behavior on the oscilloscope?
Now try to follow the pass described at the end of this exercise and perform again the tests. Are there any differences? Why, in your opinion?

Some parts of this code could be unclear now, in particular the body of the function must be taken as it is without introducing any modification. The understanding of these portions is the topic of the next labs.

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)"

```
int x = (rand()%10 + 50)*5;
   for(int i=0;i<x;i++);
```

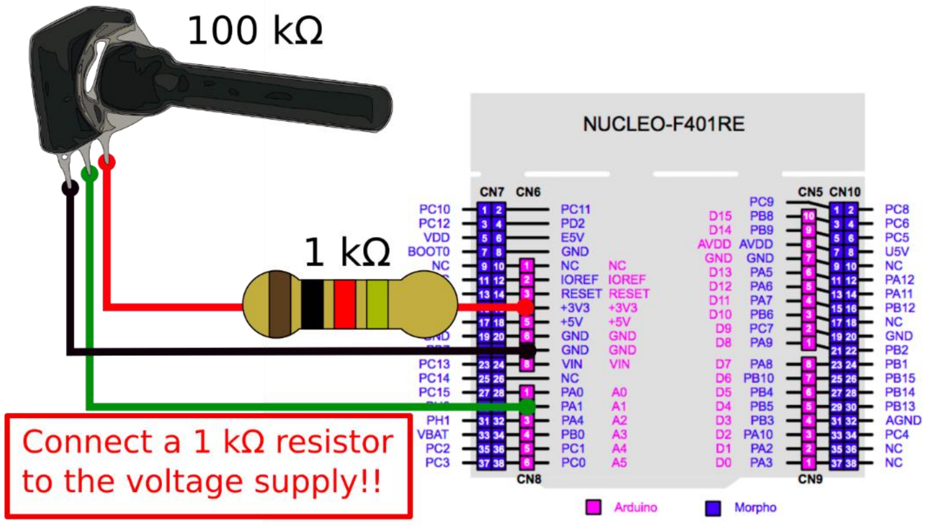
1.4-Timer Output Compare Function with Variable frequency (8.2, 8.4)

The objective of this project is changing the frequency of the generated square wave by means of a potentiometer. The potentiometer is connected to a GPIO pin configured as an analog input and the internal analog to digital converter (ADC) is used to sample and convert the input signal into a digital value. Finally, the obtained digital sample is exploited to tune the frequency in the interval 800 Hz to 4.5 kHz. We acquire samples from the ADC using the polling approach, which means that we check the end of conversion flag to know when the conversion is complete. On the other hand, to generate the square wave, we exploit the channel 1 of TIM3 timer configured with the toggle on match option. The required steps are the following:

1. **Create a new project** using the "STM32CubeIDE".
2. **Configure the hardware** using "STM32Cube" selecting the proper configuration for the counter TIM3 and the ADC. **Configure the source clock of TIM3 to 84MHz**.
3. **Fully understand the code generated** by the "STM32Cube".
4. **Follow the instruction** described at the end of this document.
5. **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 operation on the converted value to reach the desired interval of frequencies.
6. **Compile** the project.
7. **Download** the compiled code on the NUCLEO board.
8. Use a breadboard and the discrete potentiometer to **build the necessaire circuitry**. You have to use the NUCLEO board to give the supply voltage to the potentiometer. See image below.
9. **Test** the functionality of the circuit by using the oscilloscope and by varying the potentiometer.
10. **Use the debugger** to report the value of the variable that you use to update OC register for the following frequencies:

Frequency	**OC Register update value**
800 Hz	
1.2 kHz	
3.0 kHz	
4.5 kHz	

Is the waveform stable? Do you see any strange behavior on the oscilloscope? **Now try to follow the pass described at the end of this exercise and perform again the tests. Are there any differences? Why, in your opinion?**



Some parts of this code could be unclear now, in particular the body of the function must be taken as it is without introducing any modification. The understanding of these portions is the topic of the next labs.

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

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

5. Locate file named “stm32f4xx_it.c” in your project
6. Copy the code below in the body of the function called “void SysTick_Handler(void)”

```
int x = (rand()%10 + 50)*5;
   for(int i=0;i<x;i++);
```