Politecnico di Torino

Digital Systems Electronics

04OIHNX - A.A. 2024/2025

Laboratory assignment n°. 8:

Square waveform generation and potentiometer

Due date: 20/05/2025 Delivery date: 20/05/2025

Group 08 - Contributions:

310779, Simone Viola - 33,3%
300061, Viorel Ionut Bohotici - 33,3%
308299, Daniele Becchero - 33,3%

The members of the group listed above declare under their own responsibility that no part of this document has been copied from other documents and that the associated code is original and has been developed expressly for the assigned project.

Contents

1 Square waveform generator

The generation of a waveform based on a software loop is never a reliable approach. Although the system may appear to function correctly, this is only true when the MCU is solely dedicated to this task. When additional processes need to be executed, the waveform generation becomes unpredictable, particularly in terms of signal frequency. For completeness, a picture with the Nucleo board pinout is shown below.

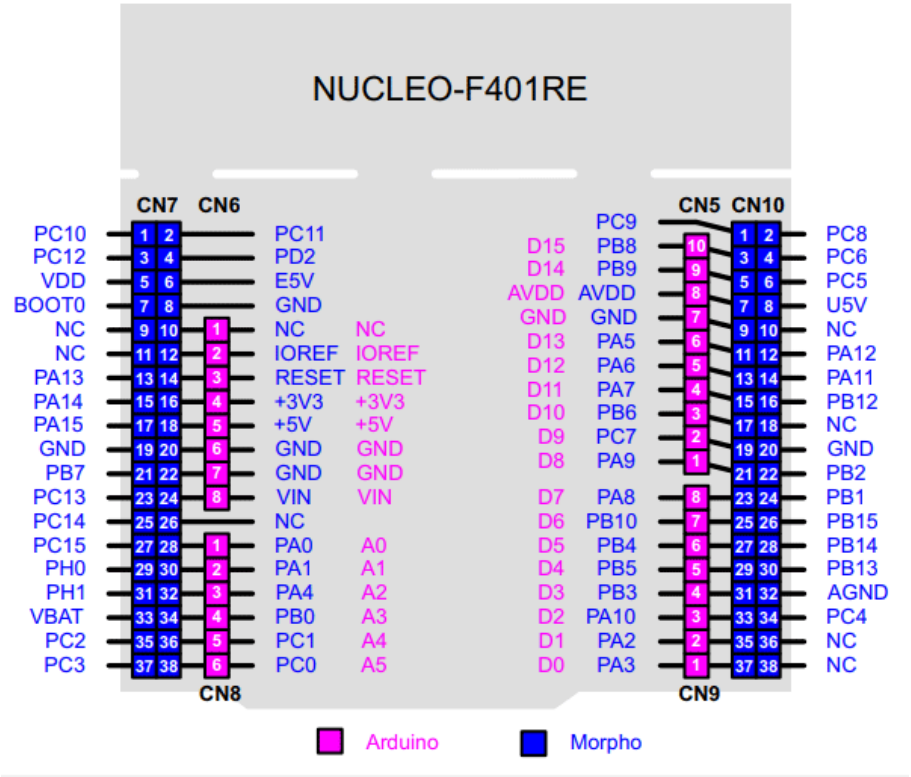

Figure 1: Nucleo-F401RE headers connections

1.1 Timer, Counter, Register Polling

1.1.1 Application Description

A basic timer-based implementation operates by polling the timer's counter register and toggling the logical state of an output pin.

1.1.2 Code Implementation

First, the MCU peripherals are configured using CubeMX. TIM3 is enabled with an internal clock source set to 84 MHz (refer to the clock configuration for timer clock settings in CubeMX). Channel 1 of the timer is configured as "output compare, no output", meaning the timer does not directly control any pin of the MCU, while channel's mode is set to "frozen". Pin `PA10` is defined as GPIO_Output. The application utilizes LL drivers for all configured peripherals.

Before entering the infinite loop, several instructions initialize the timer and define its operational parameters. The prescaler is set to 83, and the auto-reload register is assigned a value of 65535, so that the timer's counter register increments its value at a rate that is calculated using the formula 1.

$$f_{increment} = \frac{\text{Timer clock}}{(1 + \text{PSC})} \tag{1}$$

Given these parameters, the timer increments its counter register at a frequency of 1 MHz. The application code continuously monitors `TIM3_CNT` register, toggling the output value for `PA10` when the counter reaches the specified threshold, which is configured initially at 249 (meaning 250 increments). With that value, a matching event is generated each 4 kHz, which is twice the output frequency. When a matching event is detected, the output pin is toggled and the threshold is updated by adding the interval specified using a define directive.

$$\frac{f_{\text{increment}}}{2 \cdot 2\,\text{kHz}} = \frac{1\,\text{MHz}}{4\,\text{kHz}} = 250 \tag{2}$$

Variable `threshold` is declared as `uint16_t` to match the number of bits allocated for `TIM3_CNT`. Consequently, handling an overflow condition in both `threshold` and `TIM3_CNT` is unnecessary, as discussed in the assignment.

1.1.3 Verification and testing

This application has been tested using an oscilloscope, and its behavior exhibits very poor stability. The frequency of the waveform fluctuates continuously due to the nature of the testing approach, which is dependent on the execution of the code. When using the test condition with "equal to", the frequency variation is more pronounced than with the "greater or equal to" condition. This discrepancy is dependent on the strictly synchronism that is required when evaluating the "equal to" condition.

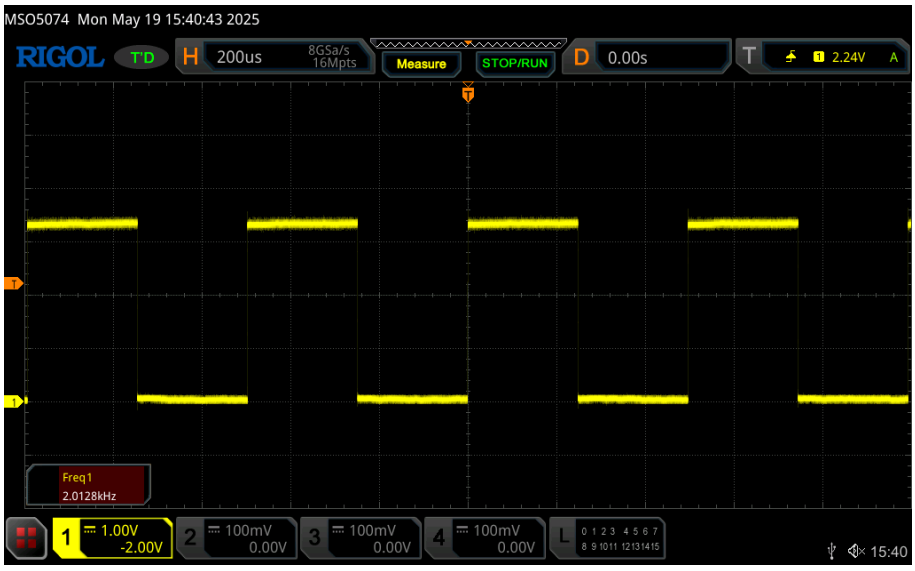

Figure 2: Output waveform with "equal to" condition

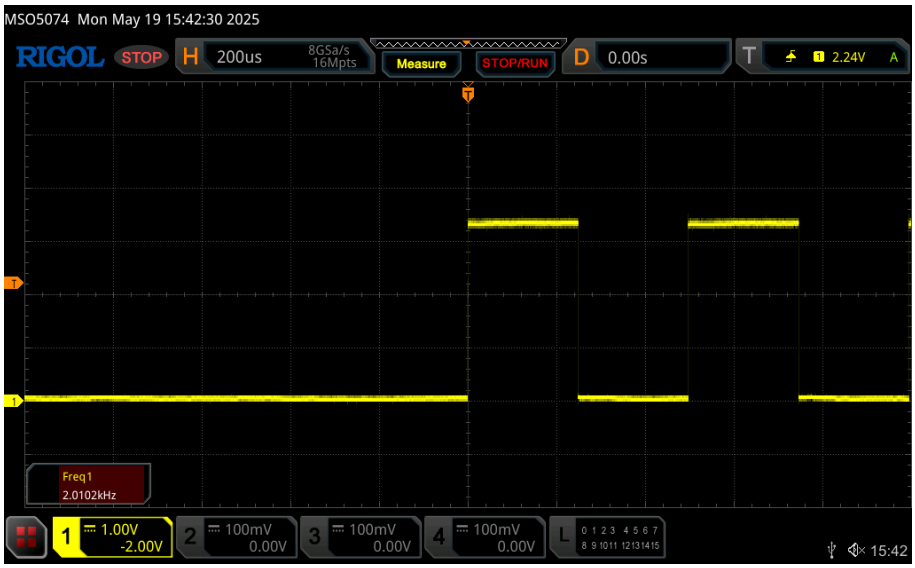

Figure 3: Output waveform with "greater or equal to" condition

The waveforms appear quite stable when generated with "greater or equal to" condition. Different combinations of PSC and ARR lead to the same result, if condition 4 is respected.

1.2 Timer, Compare Flag Polling

1.2.1 Application description

This project represents the first improved version of the previous implementation. The toggle operation of `PA10` is now driven by polling the `UIF` flag of the timer, ensuring a more stable execution compared to the earlier approach.

1.2.2 Code implementation

The project first configure the MCU peripherals in CubeMX like in the previous section. In the generated code, there is an initialization part, in which prescaler and ARR registers are configured with values that are defined by the following formula.

$$\frac{\text{Timer clock}}{(1+\text{PSC})(1+\text{ARR})} = 4\,\text{kHz} \tag{3}$$

From 3, `PSC` and `ARR` values can be determined as

$$(1+\text{PSC})(1+\text{ARR}) = \frac{\text{Timer clock}}{4\,\text{kHz}} = 21000 \tag{4}$$

For this project, the prescaler is set to 10499 and ARR is set to 1.

In the infinite loop, bit 0 (`UIF`) of `TIM3_SR` is checked to determine whether an update event has occurred in `TIM3_CNT`. When `UIF` is `HIGH`, `PA10` is toggled, and the `UIF` flag is cleared. There is no need to manually reset the CNT register of TIM3, as this operation is handled automatically by the timer.

1.2.3 Verification and testing

An oscilloscope has been used to check the correct behaviour of the system. The result is better than the previous one, but not yet optimal. If multiple waveforms have to be generated from the same timer, this approach is unsuitable. Overall, the waveform is stable and shows no unusual behavior.

The trace on the oscilloscope is more stable and accurate with respect to the previous implementation, and no strange behaviour has been observed. If the frequency needs to be raised to 4 MHz, just the `PSC` value must be changed to 4.

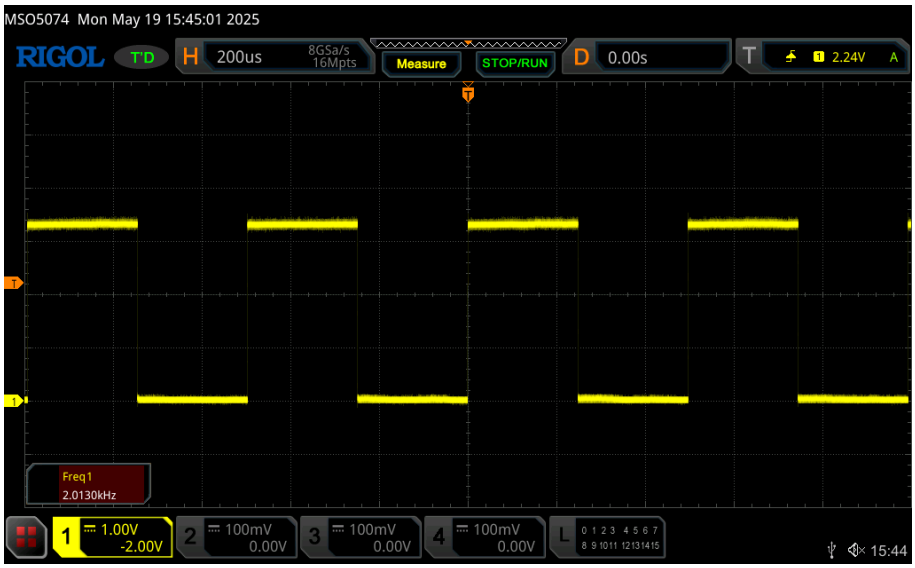

Figure 4: "Timer, Compare Flag Polling" output waveform

1.3 Timer OC function

1.3.1 Application description

This section uses the output compare capability of the STM32 to generate two distinct waveforms from the same timer. This functionality relies on some hardware that continuously monitor whether the value in the CNT register matches a predefined reference value stored in another register. When a match condition between the two registers occurs, the corresponding output channel, configured via CubeMX, can be set to high, low, toggled, or driven in PWM mode, depending on its specific configuration, and a flag in the status register of the timer is driven HIGH.

In this implementation, `TIM3` employs two output compare channels: CC1 and CC2, with `PA10` and `PB10` designated as GPIO output pins. To enhance time precision, the timer clock has been set to 50 MHz in the clock configuration window of CubeMX.

1.3.2 Code implementation

The initialization of the timer registers follows the same steps described in previous sections. To utilize the output compare capability, additional registers must be configured. Registers `CCR1` and `CCR2` contain the reference values used for comparison during the output compare operation. `CCR1` is set to 999 (1000 - 1), meaning that the `CC1IF` flag is raised every 5 kHz, which is twice the output frequency of the first waveform. Similarly, `CCR2` is assigned 199, causing the `CC2IF` flag to be set `HIGH` every 25 kHz, again reflecting twice the frequency of the second waveform.

These values were determined by taking the increment frequency of the counter and dividing it by twice the waveform frequency. This accounts for the toggling operation, which must occur at double the output frequency to achieve the desired waveform characteristics.

The final register to configure is `CCMR1` - the capture/compare mode register 1. The

first 8 bits refers to capture/compare channel 1, while the second 8 bits apply to capture/compare channel 2. For `CC1`, bits 6:4 must be modified to configure the channel to toggle on match. For `CC2`, same operation must be performed on bits 14:12.

The infinite loop relies on two conditional checks, each evaluating whether the corresponding `CCxIF` bit in `TIM3_SR` is `HIGH`, indicating that the respective output must be toggled. Additionally, register `CCRx` needs to be updated by adding an amount proportional to half the output period. Before exiting each conditional block, the `CCxIF` flag must be cleared to handle the next toggling operation.

1.3.3 Verification and testing

With the oscilloscope, we obtain the result shown in the next figure.

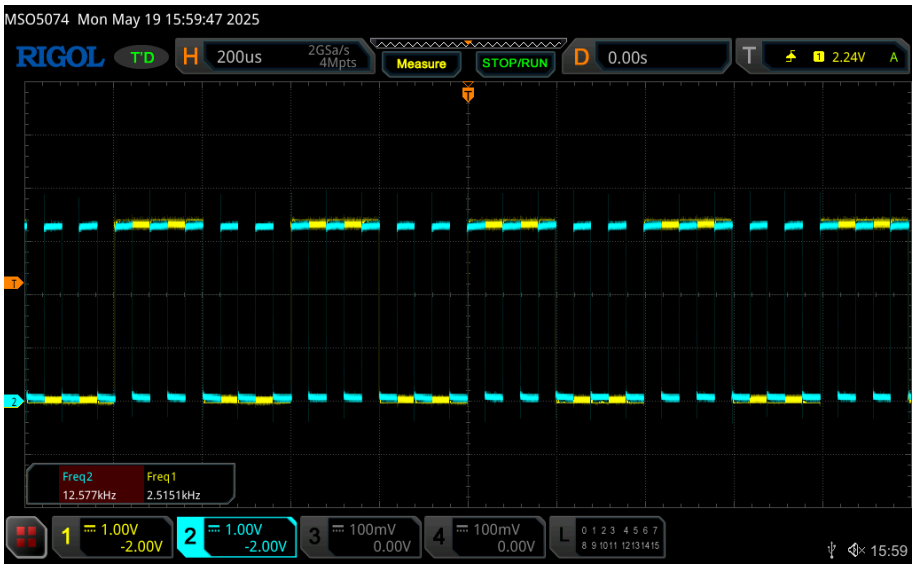

Figure 5: "Timer OC function" output waveform

1.4 Timer OC function, with automatic pin toggling

1.4.1 Application description

The objective of the application is to generate two square waves by using the timer output compare function. The first must be at 2.5 kHz, the second must be at 12.5 kHz. The difference with the previous section is in the approach used to drive the output pins. In this case, the automatic pin toggling function have to be used.

1.4.2 Code implementation

The first part of the implementation is equal to the previous ones. Channel 1 and channel 2 have been configured as "Output Compare Channel CHx" in CubeMX. The clock tree of the MCU have been edited in the part of the timers' clock, which has been set at 50 MHz to handle the two frequencies without any approximation due to non-integer values in the timer's registers. The content of all the registers has been set via software. The prescaler have been configured at 9, so that the timer will increase its content each 5 MHz and the auto-reload register have been set at its maximum value. Then, the `CCRx` registers were assigned values of 999 and 199, consistent with the configuration described in the previous section. The two output compare channels have been enabled to drive their own output pin, which cannot be chosen by the user. In the previous exercise, `PA10` and `PB10` have been used as GPIO pins set as output pin. In this section, the channel CH1 of TIM3 is one of the alternate functions of the GPIO pin `PA6` (hence, CH1 is connected to `PA6`), while CH2 of TIM3 is connected to `PA7`. The last operation to perform to initialize correctly the timer is to configure CH1 and CH2 as output, and their operation mode. In this case, the two pins have been configured to toggle on match.

Once the configuration part is completed, the MCU enters an infinite loop where the interrupt flag of CC1 and CC2 is continously red with a polling approach. When the flag is HIGH, it indicates that a match condition has occurred and that the corresponding pin has already been toggled. The toggling operation happens automatically due to the timer's configuration. However, the flag must be read to determine when a match has taken place because the CCRx register has to be incremented by a predefined amount, and the CCx interrupt flag must be cleared to ensure proper handling of the next match event.

1.4.3 Verification and testing

This project has been verified using an oscilloscope with two probes connected to the pins `PA6` and `PA7`. The screen capture is shown below.The waveform remains consistently stable, with no signs of abnormal behavior.

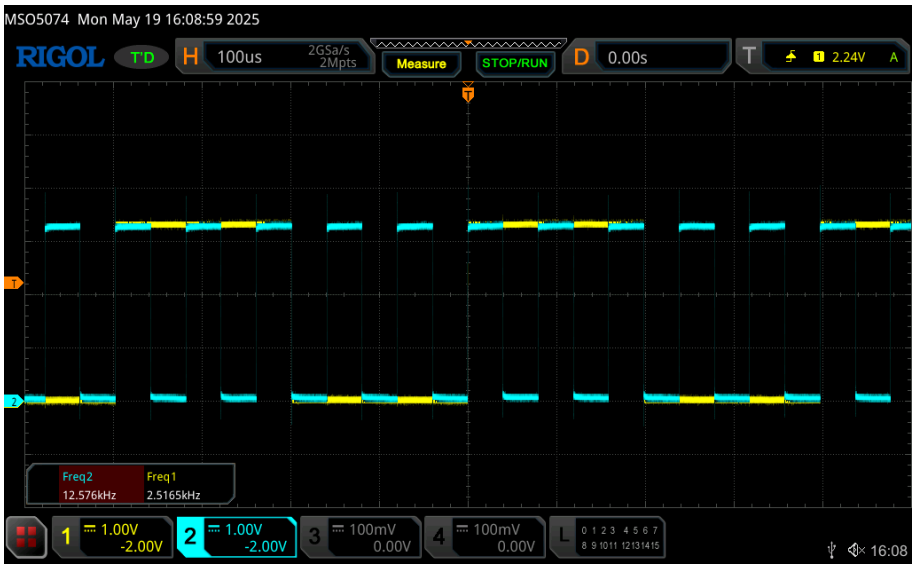

Figure 6: Timer OC function with automatic pin toggling

1.5 Timer OC function with variable frequency

1.5.1 Application description

Similarly to the previous section, this project aims to generate a square wave. This time, the frequency may be selected by the user in a range between 800 Hz and 1.6 kHz by means of a potentiometer. This component provides an analog voltage proportional to the position of the shaft, so the ADC of the MCU must be turned on and configured in order to convert the analog voltage into a digital value.

1.5.2 Code implementation

In CubeMX, TIM3 and ADC1 have been enabled and configured as follow. TIM3 is driven by the internal clock and the output compare channel is connected to `PA6`. The ADC enabled input is IN0, which is connected to `PA0`, and has been configured to have a resolution of 8 bits. The scan mode is disabled, while the continuous conversion mode is enabled, meaning that the ADC sequentially converts the analog voltage applied to `PA0`.

The prescaler of TIM3 have been set to 83, so that the TIM3 increments its value at a rate of 1 MHz. The other registers of the timer have been configured to toggle the pin of CH1 at a rate specified by a define directive. Then, in the ADC register `CR2`, the `ADON` bit have been configured to '1' to enable the ADC and `SWSTART` bit is set to '1' to start the conversion.

In the infinite loop, the `EOC` bit in `ADC_SR` register is checked to be '1', meaning that the conversion has ended, and if true, the digital value is stored in a variable. A mathematical operation is done to compute the time interval of the toggling of `PA6` based on the voltage read by the ADC, and the `EOC` bit is cleared. Also `CC1IF` bit of `TIM3_SR` is checked to be '1', meaning that a match condition occurred. If true, the value in the output compare register is updated by adding the value in time interval variable and the `CC1IF` flag is cleared.

1.5.3 Verification and testing

This project had required two different testing condition. In the first part, just the ADC is tested. The auxiliary circuit has been built on the breadboard and all the connection were done. A debugging session has initiated, and by moving the shaft of the potentiometer, we checked for the converted value to change accordingly to the position of the cursor. The second part comes after the first, and involves the use of the scope to capture the generated waveform.

Using the debugger, an approximate value of OC increment is viewed. The values are shown in 1.

Frequency	**OC Increment**
800 Hz	625
1.2 kHz	417
3.0 kHz	167
4.5 kHz	111

Table 1: Computed OC Register increments for various output frequencies

2 Conclusion

This laboratory experience has been crucial in developing a deeper understanding of the features and configuration process of timers in STM32. Beginning with a simple implementation involving a timer, we progressively enhanced precision in time-based events, such as waveform generation.

Next exercises introduced additional functionalities, including the `UIF` flag and output compare capability, significantly improving the accuracy and efficiency of time-based event handling. By employing hardware-driven operations instead of software-based control, the computational overhead was minimized, making the approach both precise and resource-efficient.

Finally, the integration of ADC in the last exercise further expanded our knowledge of STM32 peripherals, demonstrating its fundamental role in various embedded systems. These advancements enrich the practical application of STM32 timers while reinforcing essential concepts in embedded system design.

3 Sources and notes

All the projects in this document have been developed from scratch using STM32CubeIDE software, with reference to the STM32 MCU's Reference Manual and User Manual, as well as the Nucleo board's User Manual—all provided by STMicroelectronics. These specific resources can be found on `https://www.st.com/` by searching for the Nucleo-F401RE board and navigating to the documentation section. They are also available within the Digital System Electronics course material.

The members of the group have used GitHub to collaborate in the developing of the delivered codes. For each project, only the essential files have been provided. However, the project does not always work with just these files alone. For exercises requiring the STM32Cube hardware configuration, it is necessary to start a new project in STM32CubeIDE using the delivered .ioc file. After generating the C code, the `main.c` file should be replaced with the delivered version. Only after completing these steps will the project be ready to be built.