April,2023 Semester

ICT 6641
Advanced Embedded System Design

Lecture#5: Timers

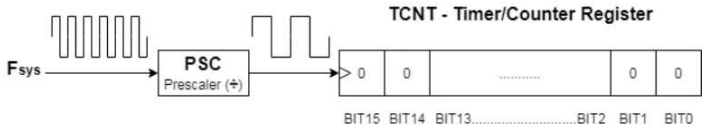
S. M. Lutful Kabir, *PhD*
Professor, BUET

STM32 Timers

- In this lecture, we'll be discussing the STM32 timers modules in STM32 microcontrollers.
- There are different hardware timers in STM32 microcontrollers each can operate in multiple modes and perform many tasks.
- You'll get to know these different hardware variants and their application use cases.

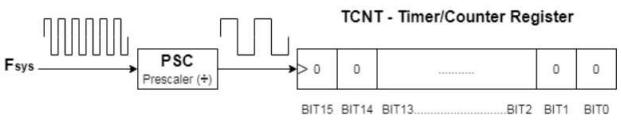
Introduction To Timers Modules

- A Timer Module in its most basic form is a digital logic circuit that counts up every clock cycle.
- It can have a Pre-scaler to divide the input clock frequency by a selectable value.
- It can also have circuitry for input capture, PWM signal generation, and much more.
- Let's consider a basic 16-Bit timer like the one shown below.
- As a 16-Bit timer, it can count from 0 up to 65535.
- Every clock cycle, the value of the timer is incremented by 1.
- And as you can see, the Fsys is not the frequency that is incrementing the timer module, but it gets divided by the Pre-scaler, then it is fed to the timer.



Introduction To Timers Modules (continued)

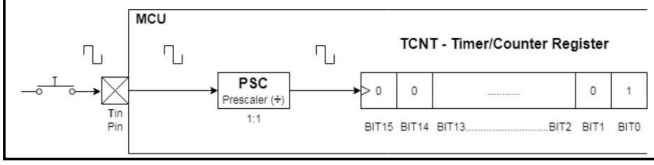
- This means if the Fsys is 72MHz & PSC is 1:2048, the TCNT gets incremented by 1 every 28.44µSec. $f_{timer} = \frac{72MHz}{2048} = 35.156\ kHz, So, Time\ Period = \frac{1}{35.156X10^3} S = 28.44\mu Sec$
- Therefore, if you start this timer to count from 0 until it reaches overflow (at 65535), it will give you an interrupt signal once every 1.8641 Second.
- What if I need to set up this timer to give me an interrupt signal once per 1 second?
- Well, for this reason, you change the maximum value for overflow.
- For generating interrupts at 1 second interval, the value of that maximum number should be 35156 (instead of 65535), this comes from the following calculations.



$$Time\ of\ overflow = max.value\ X\ \frac{Prescaler}{72X10^6}$$

Introduction to Timers (continued)

- A timer module can also operate in a counter mode where the clock source is not known, it's actually an external signal.
- Maybe from a push button, the counter gets incremented every rising or falling edge from the button press.
- This mode can be advantageous in numerous applications as we'll discuss hereafter.
- But for now, consider the following diagram. You can see, the clock signal is now driven from the push button and gets to the timer clock input through the Prescaler.



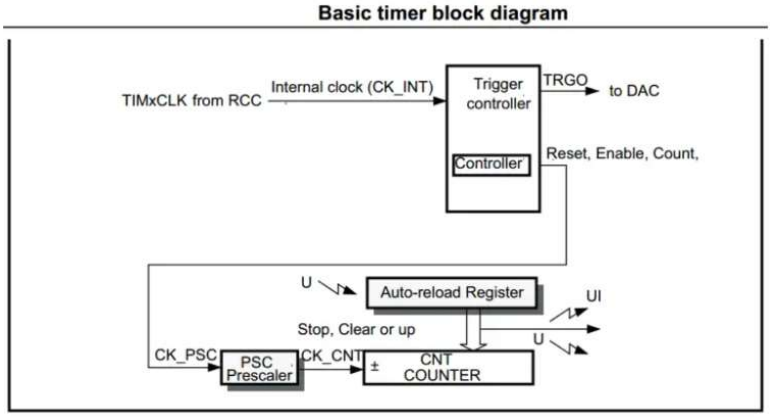
- And you can capture the information of how many times the button is pressed by simply reading the TCNT register's value.

STM32 Timers Hardware

- STMicroelectronics provides some different versions or variants for the hardware timer modules.
- STM32 microcontrollers usually have a handful of each type, however, some parts may lack one or more of these hardware timers.
- An overview of the different available hardware timers in STM32 microcontrollers will be presented.
- You have to read the datasheets to have a better understanding on which type fits in which kind of applications.
- Which in turn helps you better pick the right MCU part for your project.

Basic Timers Modules

- The basic timers consist of a 16-bit counter driven by a programmable Prescaler.
- Auto-Reload Register (ARR) is loaded with a given value (max^{m} 65535 for 16 bit).
- As an incoming pulse enters, the CNT COUNTER register is increased.



- When The CNT value overflows the ARR value, an interrupt is generated.
- The CNT value resets and the process is repeated.
- The frequency of income pulse is the internal frequency divided by the pre-scalar.

Different Timer Modules

- The **TIM** peripheral is a multi-channel timer unit, available in various configurations, depending on the instance used.
- There are basically following categories: advanced-control timers, general-purpose timers and basic timers.
- The TIM can provide:
  - PWM with complementary output and dead-time insertion,
  - break detection,
  - input capture,
  - quadrature encoder interface (typically used for rotary encoders),
  - trigger source for other internal peripherals like: ADC.
  - The full list can be found in Peripherals Interconnect matrix in the reference manual.

Different Timers

- The TIM peripheral is available in different configurations, depending on the selected instance :
  - TIM1 and TIM8 are advanced-control timers, with 6 independent channels.
  - TIM2, TIM3, TIM4 and TIM5 are general-purpose timers, with 4 independent channels.
  - TIM12, TIM13 and TIM14 are general-purpose timers, with 2 (TIM12) or 1 (TIM13 and TIM14) independent channels.
  - TIM15, TIM16 and TIM17 are also general-purpose timers, with 2 (TIM15) or 1 (TIM16 and TIM17) independent channels.
  - TIM6 and TIM7 are basic timers

TIMERS in Blue Pill

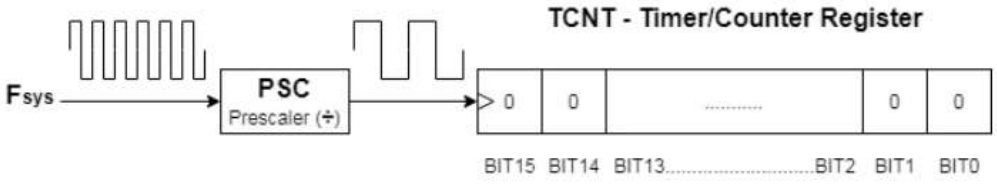
- Blue Pill contains,
  - TIM1
  - TIM2
  - TIM3
  - TIM4

STM32 Timers Modes OF Operation

- An STM32 timer module can operate in any of the following modes, namely
  - Timer mode
  - Counter mode
  - Input Capture mode
  - PWM mode
  - Advanced mode,
  - Output Compare mode
  - One pulse mode
  - Encoder mode
  - Timer DMA burst mode
- However, you should not assume that a given timer does support all of these modes.
- Instead, you'll have to check the datasheet to figure out which modes are supported by which timers.

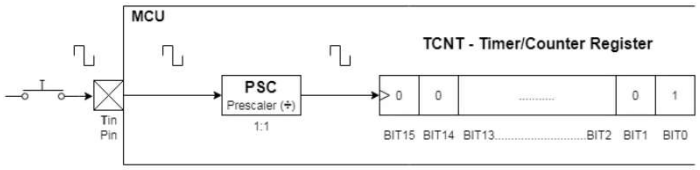
Timer Modes

- In timer mode, the timer module gets clocked from an internal clock source with a known frequency.
- Hence the clocking frequency is known, the overflow time can also be calculated and controlled by the preload register to get any arbitrarily chosen time interval.
- When a timer overflows, the timer signals the CPU with an interrupt that indicates the end of the specified time interval.
- This mode of operation is usually used to get a specific operation done each specific time interval.
- And to achieve timing & sync between various tasks and events in the system.
- It can also replace delays in various situations for better system response.



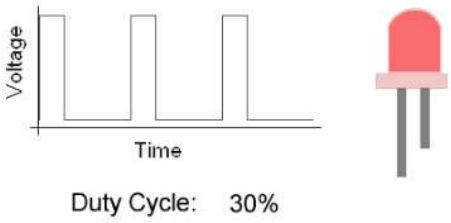
Counter Mode

- In counter mode, the timer module gets clocked from an external source (timer input pin).
- So, the timer counts up or down on each rising or falling edge of the external input.
- This mode is really helpful in numerous situations when you need to implement a digital counter without polling input pins or periodically reading a GPIO or continuously interrupt the CPU if you've chosen to hook it up to an EXTI pin.
- You can actually monitor the counter value to tell how many pulses did occur or what was the frequency of it.
- Such a mode can be advantageous in many situations like this.



PWM Mode

- In PWM mode, the timer module is clocked from an internal clock source and produces a digital waveform on the output channel pin called the PWM signal.
- By using capture and compare registers (CCR), the incrementing timer's register value (CNT) is constantly compared against this CCR register.
- When a match occurs the output pin state is flipped until the end of the period and the whole process is repeated.
- The timer in PWM mode will produce a PWM signal at the specified the user's chosen frequency.
- The duty cycle is also programmatically controlled by its register.



Input Capture Mode

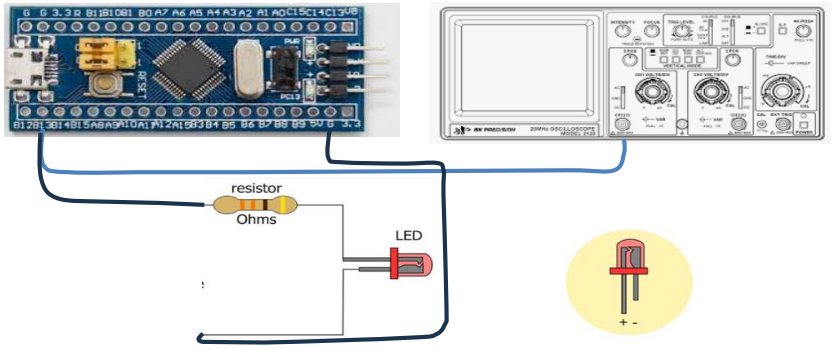
- In Input capture mode, the Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding input signal.
- When a capture occurs, the corresponding CCXIF flag (TIMx_SR register) is set and an interrupt can be sent if they are enabled.
- This mode is extremely important for external signal measurement or external event timing detection.
- The current value of the timer counts (CNT) is captured when an external event occurs and an interrupt is fired.
- So, we can use this feature for a wide range of measurement applications.
- An application example is an ultrasonic sensor that measures the distance and sends the information as a pulse to your microcontroller.
- By measuring the pulse width time, you can find out the distance reading. This can be achieved by using the input capture unit (ICU) within the timer module.

STM32 Timers – Timer Mode

- After this overview of the STM32 timers hardware variants and the timers' possible modes of operations, we'll just focus on one of them for the rest of this tutorial.
- Which is going to be the very basic one, the "timer mode".
- So, we'll discuss the timer mode operation in-depth and we'll follow this up by a practical exercise.

Exercise 1: An Example of Time Mode

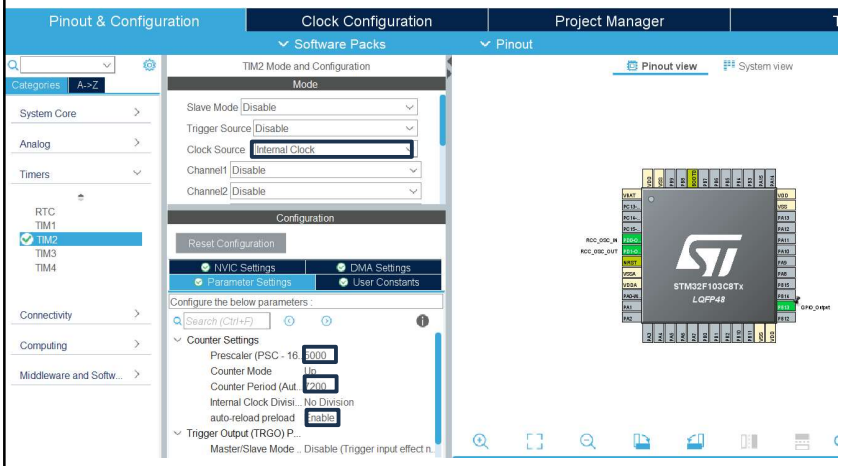
- The objective of the first exercise is to use Timer Overflow interrupt and toggle a pin at every interrupt.
- In consequence a square wave will be generated with 50% duty cycle.
- We shall connect our Blue pill module according to the following diagram.



Steps to be followed

1. We shall toggle a pin at an interval (set by the timer).
2. Let us choose Pin B13 as an Output pin.
3. We shall choose external crystal by selecting RCC and set the internal frequency as 72 MHz.
4. First of all, we have to choose pre-scalar and the clock period counts value.
5. It needs some calculations.
    - If we want to generate a square wave of 1 kHz, Tout=1/1000=1mS, So, Tout/2=500mS.
    - fsys=72 MHz, Say, Pre-scalar=5000, effective internal frequency = 72000kHz/5000=14.4kHz, i.e. one clock period = (1/14.4)mS=0.069mS.
    - So, number of clock period required to make 500 mS is 500/0.069 = 7200.
    - Therefore let us set prescalar = 5000 and clock period counts = 7200.

Timer Settings



In Timer menu, choose TIM2 and set the followings:

- Clock Source->”Internal Clock”,
- Prescalar->5000,
- Counts Period->7200 and
- auto-reload->“Enable”.

- Enable The Timer Interrupt Signal from NVIC Tab.
- Generate the Project Initialization Code.

Necessary Starting and Callback function

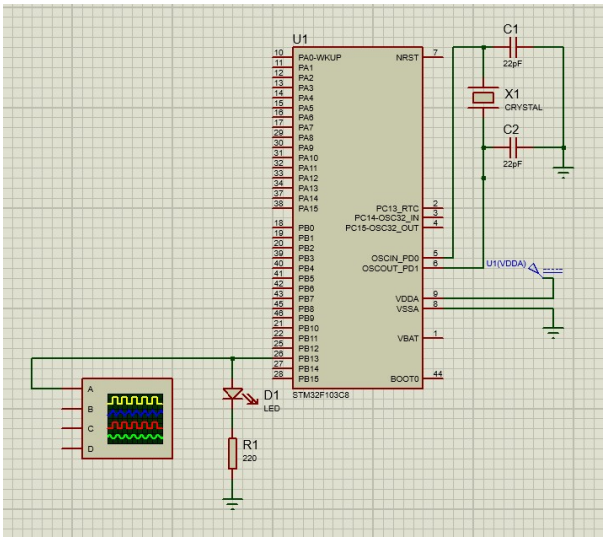
Starting the timer

```
HAL_TIM_Base_Start_IT(&htim2);
```

Toggling the output pin

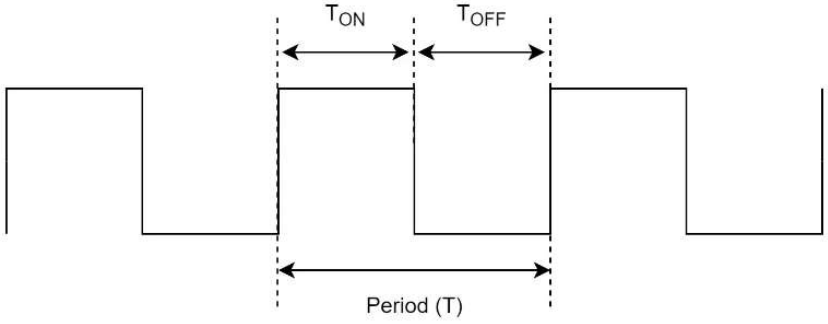
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
}
```

Simulation in Proteus



Timer in PWM Mode

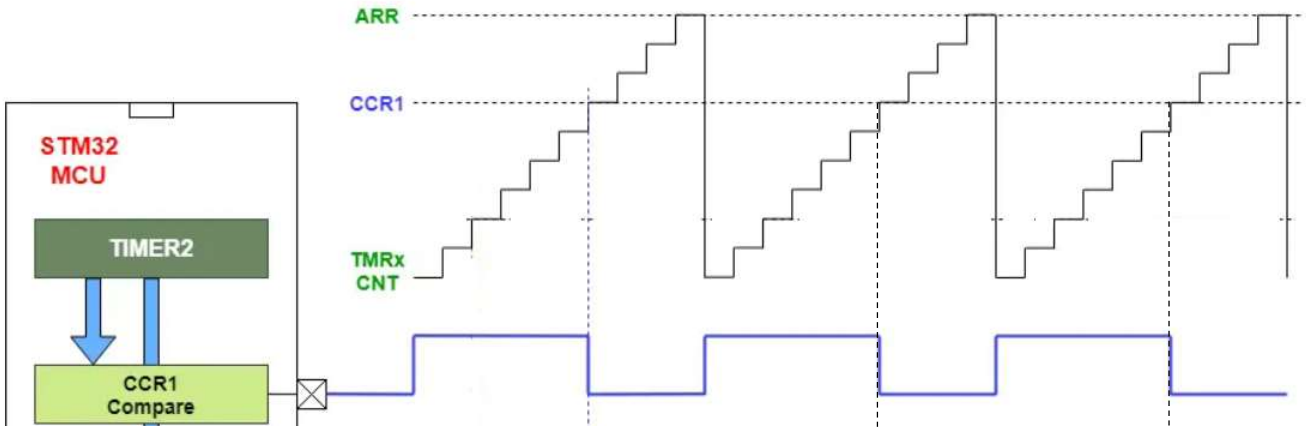
- What is PWM?
- What is Duty Cycle?



$$Frequency[PWM] = \frac{1}{Period(T)} Hz$$

$$DutyCycle[PWM] = \frac{T_{ON}}{T_{ON}+T_{OFF}} \times 100 = \frac{T_{ON}}{Period(T)} \times 100[\%]$$

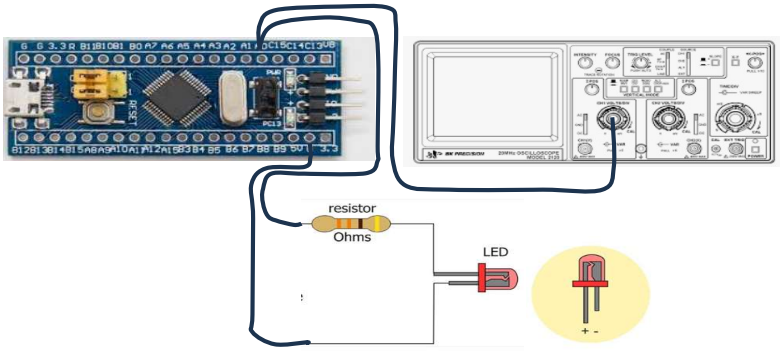
How PWM works in STM32



When CNT exceed ARR, output goes HIGH.
When CNT exceeds CCRx, output goes LOW.

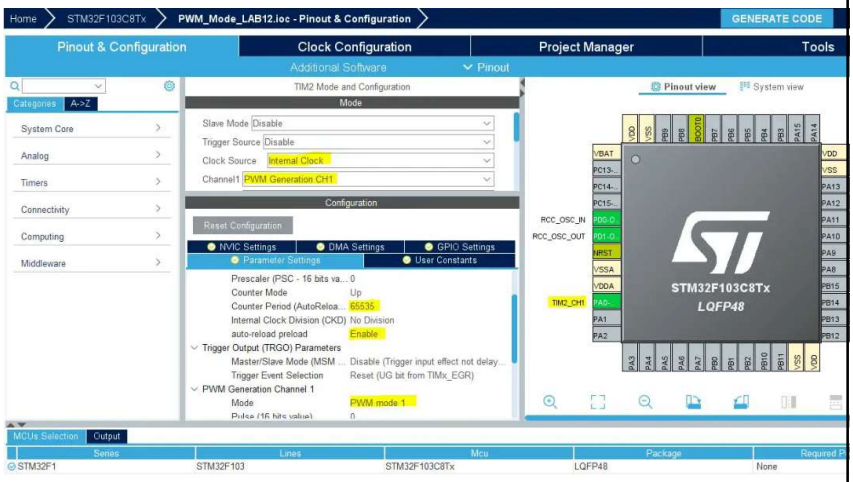
Exercise 2: PWM Mode

- The objective of this exercise is to generate Pulse Width Modulated (PWM) signal using Timer.
- The variation of duty cycle of the PWM signal can be varied by varying CCRx register.
- We shall learn how to generate PWM signal having variable duty cycle.
- First of all, we have to connect your Blue Pill as per the following diagram.



Timer Setting in PWM mode

- The settings that have to be made for operating in PWM mode are:
  - Clock Source->"Internal Clock",
  - Channel1->PWM Generation CH1,
  - Counts Period->65535
  - auto-reload->"Enable" and
  - Mode-> PWM mode 1



1. Then we have to generate the Project Initialization Code and the following parts.

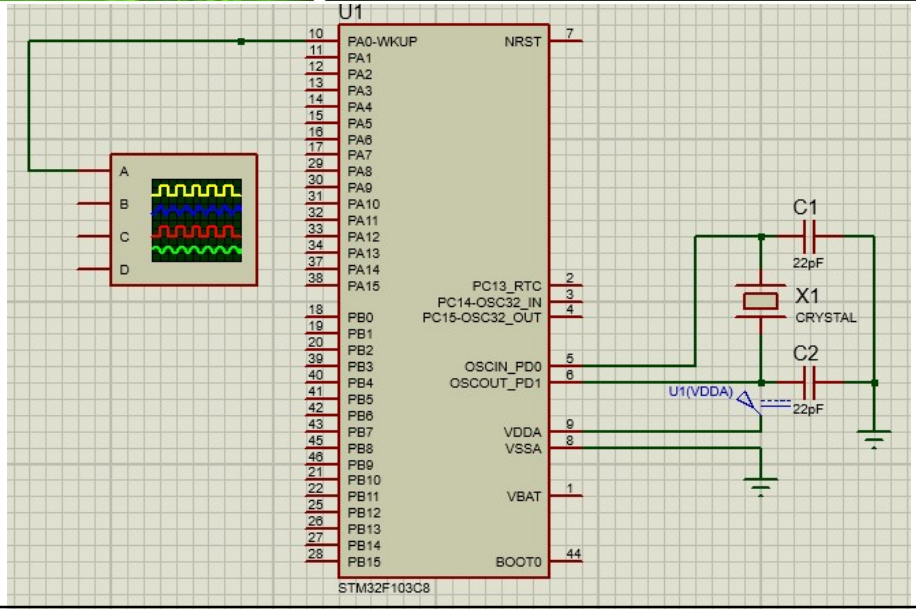
Start the timer in PWM mode

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

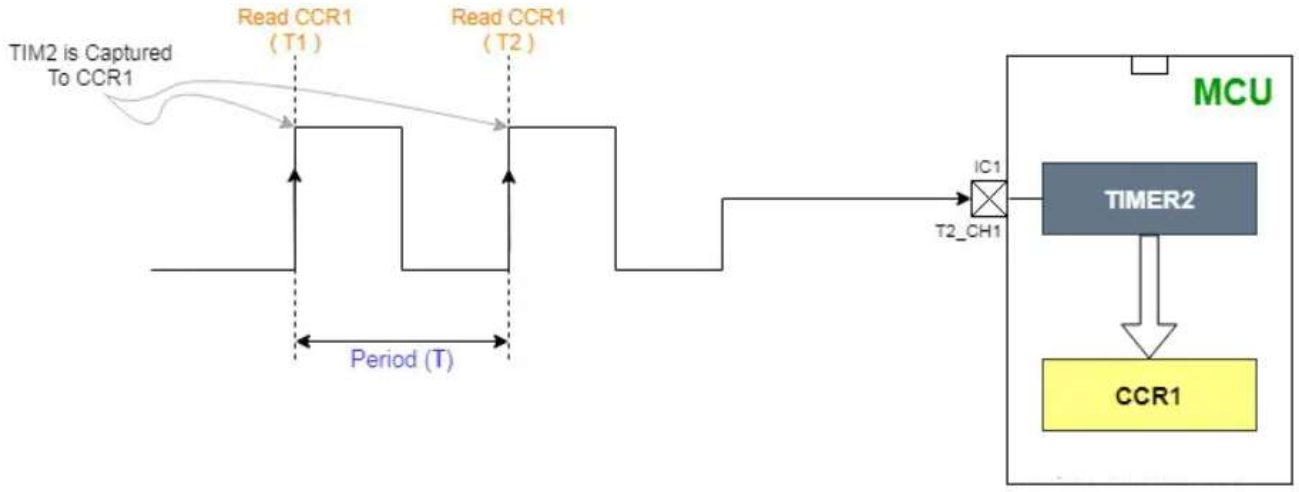
Code for varying PWM's DC →

```
while(CH1_DC < 65535)
        {
            TIM2->CCR1 = CH1_DC;
            CH1_DC += 70;
            HAL_Delay(50);
        }
  while(CH1_DC > 0)
        {
            TIM2->CCR1 = CH1_DC;
            CH1_DC -= 70;
            HAL_Delay(50);
        }
```

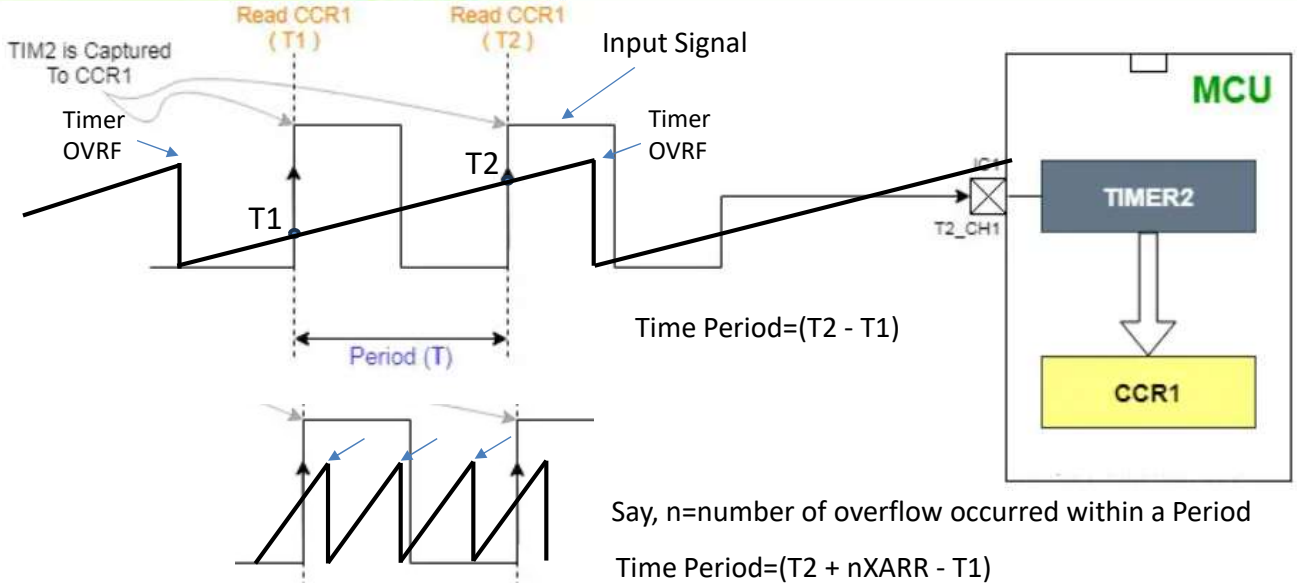
Simulation in Proteus



Principle of Input Capture



How the Time Period is Calculated?



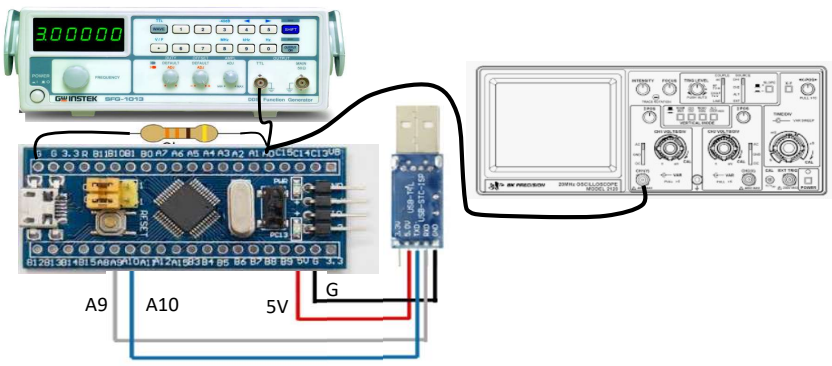
Time Period=(T2 - T1)

Say, n=number of overflow occurred within a Period

Time Period=(T2 + nXARR - T1)

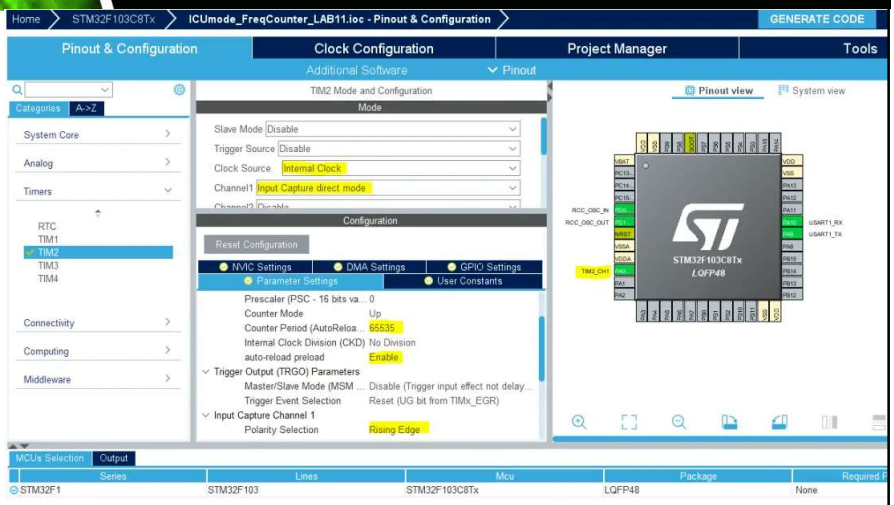
Exercise 3: Input Capture Mode

- The objective of this exercise is to determine the frequency of a square wave using Timer.
- The feature of the timer is called 'Input Capture'.
- We shall learn how to set the parameter and logic applied the determine the time period of square wave and hence the frequency.



Timer Settings

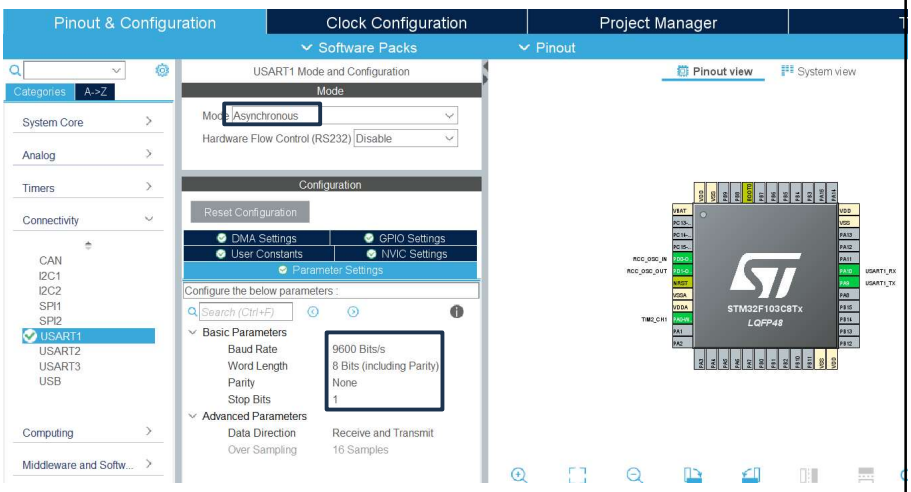
- Configure TIM2 peripheral to operate in the input capture mode.
  - Clock source->internal Clock,
  - Channel1->Input Capture direct mode,
  - Counter Period->65535,
  - auto-reload preload->Enable,
  - Polarity Selection->Rising Edge.



- Note that A0 is the input pin for the input capture signal.
- Enable Timer2 interrupt from NVIC control tab.

UART Setting

- Mode, asynchronous
- Baud rate, 9600 for Proteus, 115200 for PC



New Addition in the Code

```
#include <stdio.h>
#define IDLE   0
#define DONE   1
#define F_CLK  72000000UL

volatile uint8_t gu8_State = IDLE;
volatile uint8_t gu8_MSG[35] = {'\0'};
volatile uint32_t gu32_T1 = 0;
volatile uint32_t gu32_T2 = 0;
volatile uint32_t gu32_Ticks = 0;
volatile uint16_t gu16_TIM2_OVC = 0;
volatile uint32_t gu32_Freq = 0;
```

Declaration

```
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
```

Starting Timer

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
  gu16_TIM2_OVC++;
}
```

First Callback function

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
  if(gu8_State == IDLE) {
    gu32_T1 = TIM2->CCR1;
    gu16_TIM2_OVC = 0;
    gu8_State = DONE;
  }
  else if(gu8_State == DONE) {
    gu32_T2 = TIM2->CCR1;
    gu32_Ticks = (gu32_T2 + (gu16_TIM2_OVC * 65536)) - gu32_T1;
    gu32_Freq = (uint32_t)(F_CLK/gu32_Ticks);
    if(gu32_Freq != 0)
    {
      sprintf(gu8_MSG, "Frequency = %lu Hz\n\r", gu32_Freq);
      HAL_UART_Transmit(&huart1, gu8_MSG, sizeof(gu8_MSG), 100);
    }
    gu8_State = IDLE;
  }
}
```

Second Callback function

Lab Experiments on Timers

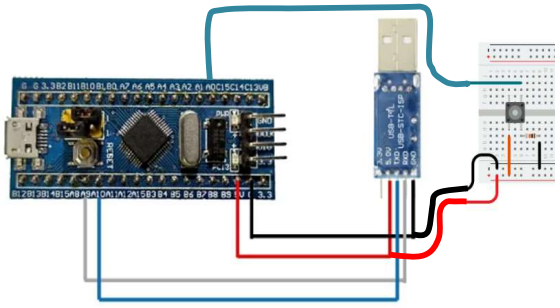
- Part2A- Timer in Timer mode
- Part2B- Timer in PWM mode
- Part2C- Timer in Input Capture Mode

Assignment-4

- Develop a system based on STM32 which will count an external event, and after an completion of a certain number of events, it will start counting afresh. The counting value will be monitored in the console.

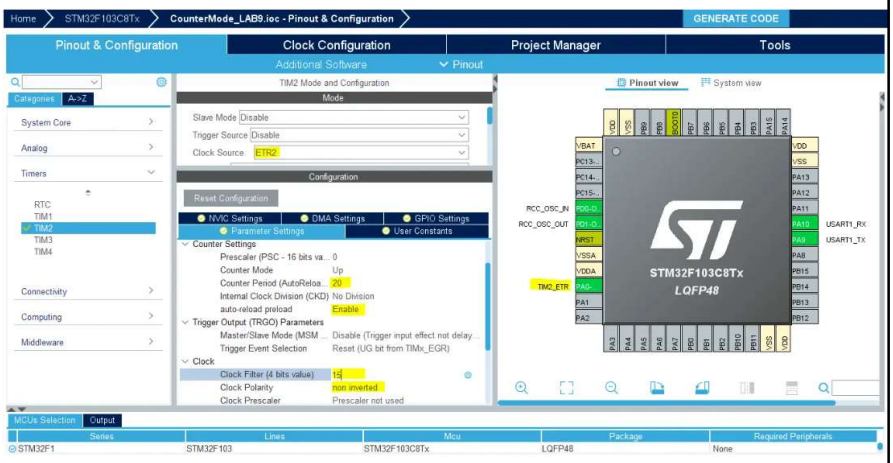
Circuit Diagram for the Assignment

- The objective of this assignment is to use timer as a counter.
- A program is written and a set up is proposed where, number of push button switch pressed will be counted
- And the value of the counting will reset to a preassigned max^{m} value.
- The counting value will be displayed in the computer console using serial print (which requires a USB-TTL converter).



Timer Setting in Counter Mode

- In TIM2 setting, the source of the clock is external, ETR2.
- The other parameter settings should be for configuring a digital filter for this input channel in order to reject noise. Set it as 15.
- Set also the clock polarity, non-inverted and the counter period, 20, auto-reload preload, Enable.



Thanks