Lab 6: Timers and PWM

I. Objectives

1. Use buzzers and ultrasonic sensors to experiment PWM (Pulse Width Modulation) output generation and input capture
2. Use STM32 timers to implement precise delay, generate PWM outputs, and measure input pulses
3. Use a passive buzzer along with PWM enabled IOs to produce audio with distinct musical notes
4. Use Ultrasound sensor to detect objects and their distances

II. PWM overview

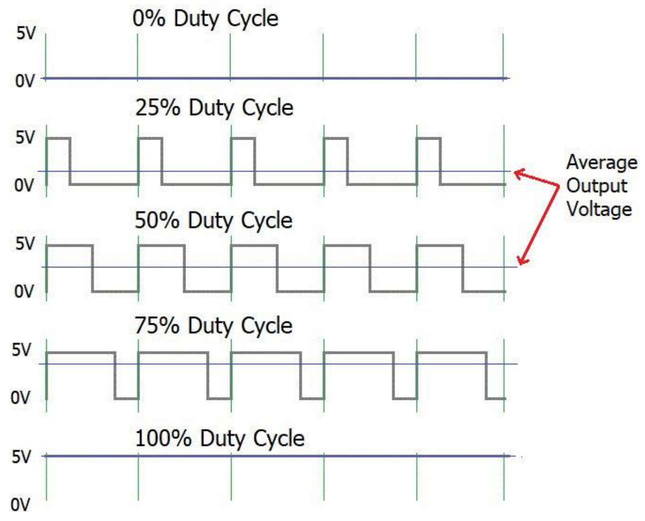
Pulse-width modulation (PWM) is a **powerful technique for controlling analog circuits with a microcontroller's digital outputs**. In other words, PWM is **used to produce analog signals from a digital device** just like our STM32 microcontroller. **The signal produced will have a train of pulses** in the form of a square wave. That is, at any given instance of time the wave will either be high or low. The duration at which the signal stays high is called the "on time" and the duration at which the signal stays low is called as the "off time". **Two important parameters are associated with a PWM signal, the PWM duty cycle and the PWM frequency**.

The **duty cycle** is the **percentage of time in which the PWM signal remains HIGH**. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle. The formula to calculate the duty cycle is:

```
Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)
```

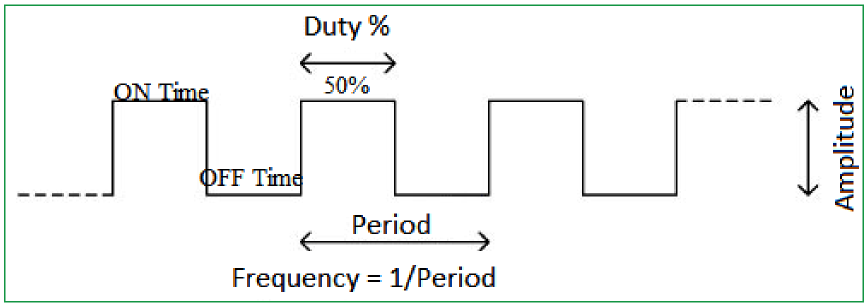
By controlling the Duty cycle from 0% to 100% we can control the "on time" of PWM signal and thus the width of signal. Since **we can modulate the width of the pulse, it got its iconic name "Pulse width Modulation"**.

To get varying analog values, you change (or modulate) that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED. The output voltage of a PWM signal after converting it to analog will be the percentage of duty cycle out of its operating voltage, as shown in the figure below.



The frequency of a PWM signal determines how fast a PWM completes one period. One Period is the complete ON and OFF time of a PWM signal as shown in the figure below. The formula to calculate the frequency is:

```
Frequency of PWM signal = 1/Time Period
Time Period = On time + Off time
```



Therefore, the frequency of the PWM signal decides how fast the PWM signal should turn on and off, and the duty cycle decides how long the PWM signal should remain turned on in that speed. Varying the duty cycle with a constant frequency can be used to control the brightness of RGB LEDs or to control the direction of a servo motor attached to a mechanical object like a robot arm.

III. PWM and Timers in STM32

The nucleo board has distinct pins that can be used for PWM. Micro-controllers, such as the STM32 utilize hardware timers to generate signals of various frequencies, generate pulse-width-modulated (PWM) outputs, measure input pulses, and trigger events at known frequencies or delays.

The Timers generic features include:

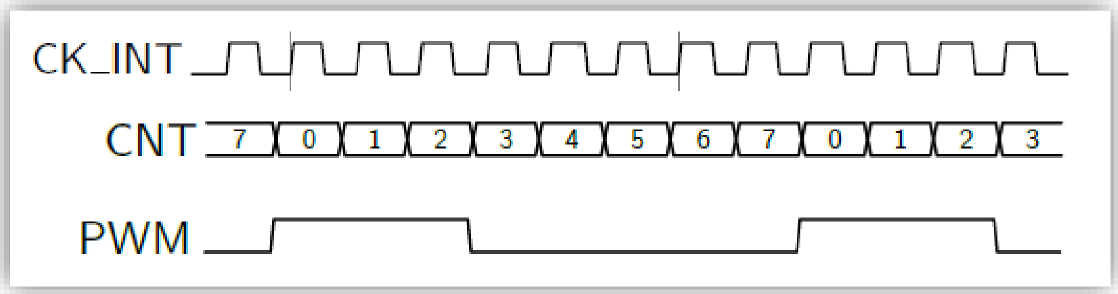
- Up, down, up/down auto-reload counter
- Programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
- Up to 4 independent channels for:
  - Input Capture
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - One-pulse mode output

A timer has four components – a controller, a prescaler (PSC), an "auto-reload" register (ARR) and a counter (CNT).

The function of the **prescaler (PSC)** is to divide a reference clock to lower frequency. The STM32 timers have 16-bit prescaler registers and can divide the reference clock by any value 1 → 65535. For example, a 24 MHz system clock could be used to generate a 1 MHz count frequency with a prescaler of 23.

The **counter register (CNT)** can be configured to count up, down, or up/down and to be reloaded from the **auto reload register (ARR)** whenever it wraps around (an "update event") or to stop when it wraps around.

An important use of counter channels is the generation of precisely timed pulses. There are two variations of this use – **“one-pulse” pulses**, in which a single pulse is generated, and **pulse width modulation**, in which a series of pulses is generated with the counter UEV period. The pulse width is controlled by the **Capture/Compare Register (CCR)**. **In PWM mode, ARR controls the period, and CCR controls the pulse width (and hence the duty cycle).**



Hence in order to implement a PWM signal using STM32 timers, the frequency of a whole counter period can be calculated as follows:

```
Frequency of one counter period (one PWM signal) = SYSCLK/( (Prescaler+1)X(ARR+1) )
```

<u>**Note:**</u> More on timers can be found in “PWM output & input capture - ultrasonic Sensor with STM32.pdf” supporting document.

IV. PWM Sound Synthesis

PWM is sometimes used in sound (music) synthesis. The varying pulses that we send out via PWM can be used to generate analog audio signals.

Every sound humans encounter consists of one or more frequencies, and the way the ear interprets those sounds is called pitch. The pitch of a sound may be its true frequency or may be a result of overtones which are perceived as a particular pitch. To produce a variety of pitches, a digital signal needs to convey the frequency of sound it is trying to reproduce. The simplest approach is to generate a 50% duty cycle pulse stream and set the frequency to the desired pitch. To generate a specific musical note, its frequency needs to be found.

Therefore in order to produce audio, the duty cycle should be kept at a constant 50% while varying the frequency through producing a variable frequency PWM signal. A constant duty cycle of 50% allows for a nice-sounding, clean audio signal, whereas a lower or higher duty cycle causes the sound to be sharp and have varying volume.

Passive buzzers are simple modules that can produce different tones according to the sound control input signal, unlike active buzzers which produce a fixed tone when given power.

V. Experiment 1

Develop an application on the nucleo board that uses a passive buzzer to continuously produce distinct musical notes, each separated by 0.5 second delay. The notes should range from 100 to 2000 Hz with a step of 100.

Notes:

- If you're trying to produce tones for the human ear, then values in the range from few hundreds to just a few thousands of Hz are where our ears are most tuned.
- Adjust your STM32 clock system on CubeMX. STM32L4xx microcontrollers support a SYSCLK frequency of up to 80MHz. The constant `SystemCoreClock` is defined in the firmware library to be the number of SYSCLK cycles per second.
- Enable one of the STM32 timers TIM on CubeMX, specify its clock source, adjust its prescaler and ARR, then enable one of its channels to generate an output PWM signal to drive the passive buzzer.
- Check the STM32L4 HAL reference for the TIM APIs needed.

The user code inside main function should be as follows:

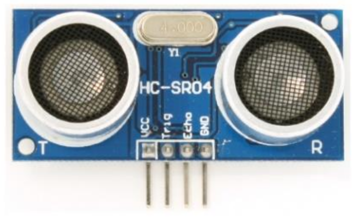
```
// start the PWM signal generation on the intended timer channel

// set the Capture/Compare Register (CCR) for adjusting the pulse width (duty cycle)

//vary frequency to produce 10 sounds
while (1)
{
        for (……)
        {
                // set a prescaler value for the intended timer
                // wait for 0.5 sec
        }
}
```

VI. Ultrasonic Sensors

Timers input capture can be used in conjunction with PWM to control commonly available ultrasonic ranging devices such as that illustrated in the figure below.



The HC-SR04 ultrasonic ranging module emits an ultrasound which travels through the air. If there is an object or obstacle on its path, the ultrasound will bounce back to the module. Considering the travel time and the speed of sound you can calculate the distance in between.

The sensor generates the ultrasound by setting the `Trig` pin on a High State for 10 µs. The Module will then send out eight 40 kHz pulses which will travel at the speed of sound and will be received by the `Echo` pin. The `Echo` pin will output the time the sound wave traveled in microseconds.

`Distance covered = (Duration of high level Echo pin * speed of sound)/2`

Note: More info can be found in "HC-SR04 User Guide.pdf" supporting document.

VII. Experiment 2

Develop an application on the nucleo that uses the ultrasonic sensor to measure its distance from nearby moving obstacles and sends the distance over a UART interface to be displayed on TeraTerm.

Notes for CubeMX code generation:

- Use a timer to implement precise microseconds delay, and to measure the input `Echo` pulse
- Provide microseconds delay in order to set the `Trig` pin on a High State for an exact 10 µs. Use one of the general purpose timers to produce this delay by adjusting its prescaler and ARR and measuring the number of counter rounds needed to produce a 1 µs delay. For instance, if SYSCLK is 8 MHz, prescaler = 7, ARR = 65535, the SYSCLK frequency is lowered to 1 MHz with counter counting 0 → 65535 continuously, with each tick happening every 1/1M = 1 µs.
- Enable the input capture mode on one of the timer channels to capture the input `Echo` pulse and measure its time.
- Enable the capture compare interrupt handler for that timer.
- Enable a pin as GPIO output to be used as `Trig.`

Notes for Keil µVision:

- Use the API `HAL_TIM_IC_Start_IT` to start the TIM Input Capture measurement in interrupt mode for capturing and measurement of our `Echo` input.
- Use the timer capture compare interrupt handler in `stm32l4xx_it.c` file to handle the fired interrupt and measure the time whenever the `Echo` input goes up and whenever it goes down and calculate the distance accordingly.
- Use the API `HAL_TIM_ReadCapturedValue` to read the timer captured value.
- Use the API `__HAL_TIM_SET_CAPTUREPOLARITY` to change the input polarity on runtime to be on the rising or falling edge of our `ECHO` input.

The following user code creates a delay:

```
// Start the TIM Base generation

// Set the TIM Counter Register value to 0
while (/*TIM Counter Register value is less than the intended value */);
```

The user code inside main function should be as follows:

```

// Start the TIM Base generation
// Pull Trig output low for a little time
while (1)
{
        // pull TRIG high
        //create 10µS delay by setting the TIM Counter Register value to 0
        while (/*TIM Counter Register value is less than the intended value */);
        // pull TRIG low
        // start TIM Input Capture measurement in interrupt mode
        // wait for a little time
}
```

The user code inside the timer capture compare interrupt handler should be as follows:

```
// if you're waiting for the rising edge
//      Capture the timer current value
//      Set the polarity to wait for the next falling edge

// if you're waiting for the falling edge
//      Capture the timer current value
//      Set the polarity to wait for the next rising edge
//      calculate the distance between the two edges
//      Transmit distance over UART

HAL_TIM_IRQHandler(&htim1);
```

VIII. References:

- PWM output & input capture - ultrasonic Sensor with STM32.pdf
- HC-SR04 User Guide.pdf
- UM1884_Description of STM32L4 HAL and LL drivers (dm00173145).pdf

Lab Report [10 pts] (Individual submission)

1. [1 pts] Provide your C code of experiments 1 and 2.
2. [4.5 pts] Modify experiment 1 code to output the precise musical tones of "Do, Re, Mi, Fa, Sol, La, Ti" in a certain sequence that matches a song of your choice. You need to generate 15-20 tones for the song. Provide your C code along with a small sized video of the running program.
3. [4.5 pts] Develop an application that uses PWM to control the brightness of an external LED. The LED's brightness should continuously be increasing from 0v to its maximum then decreasing till 0v and so on. Provide your C code along with a small sized video of the running program.