EED308 **Embedded System Hardware** **Spring 2021**

Experiment 8 Frequency Measurement and PWM Generation using Timer

Part A. Frequency Measurement

In this experiment we will measure the frequency of an external clock signal (Fast Clock available in the Digital Trainer Kit) using the Capture/Compare feature of Timers. The two possible methods for Frequency measurement using a Timer and the corresponding mathematical relationships are described below.

1. Counting cycles of the external signal for a time interval (GATE) set by the Timer –

   The GATE interval is generated by the Timer Counter through appropriate settings of the ARR, taking the default setting (zero) of the Repetition Counter Register RCR:

   GATE = ([ARR] + 1) * TCLK, where TCLK is the Time Period of the Counter Clock.

   If the number of cycles of the external signal counted in the GATE interval is COUNT, the frequency of the external signal FSIG = COUNT / GATE.

2. Counting cycles of Counter Clock for one time period of the external signal –

   If this count is given by COUNT, then the frequency FSIG = FCLK / COUNT, where FCLK is the Counter Clock frequency.

Thus for a measurement with an accuracy of n decimal digits, the first method would require 10^n pulses from the external signal to be counted over the GATE duration, which will therefore have to be equal to 10^n TSIG, while the second method would require 10^n pulses from the chosen internal clock to be counted over a duration TSIG. Hence the first method is normally preferred for HIGH-FREQUENCY signals, and the second method, for LOW-FREQUENCY signals, HIGH and LOW being in comparison with the range of Counter clock frequency employed in Timers, which may be up to 100MHz.

In this experiment, the external signal has a frequency slightly less than 1 kHz, implying TSIG > 1 ms, and hence we will use the second method for measuring its frequency. Maximum possible accuracy would be achieved in the measured frequency for a given RCC Clock (= 8MHz in our case) if the Counter Clock has the maximum possible frequency, and if the Counter is allowed to count to the highest possible COUNT value. So we will set [PSC] = 1 and [ARR] = 65000, giving FSIG = (4000 / COUNT) kHz.

Configure a Timer by following the sequence of steps through STM32CubeIDE as given below, and measure the frequency of the fast clock of the DE Kit. Display the value on live expression.

1. Open STM32CubeIDE and create a new project; follow the basic instructions for setting up the project for your respective microcontroller and Nucleo board.
2. Select RCC from the pinout pane, and choose
   High Speed Clock (HSE): Crystal/Ceramic Resonator.
3. Select a suitable Timer (TIM3) from the pinout pane, and make the following choices:
   a. Slave Mode: Disable
   b. Trigger Source: Disable
   c. Clock Source: Internal Clock.
   d. Channel 1: Input Capture Direct Mode.
4. Set the following parameters from the configuration pane:
   a. Pre-scalar: Appropriate value
   b. Counter Mode: UP.
   c. Counter Period (ARR): Appropriate value
   d. Repetition Counter: 0.
5. Set NVIC as follows:
   a. TIMx Global Interrupt: Enable

Generate the code and do the suitable modification

Changes at place 1

```
/* USER CODE BEGIN PV */
uint32_t COUNT1 = 0, COUNT2 = 0, COUNT_Ticks = 0, COUNT_TIM3_OVC = 0,
state = 0;
uint32_t F_CLK;
uint32_t signalFreq;
/* USER CODE END PV */
```

Changes at place 2

```
/* Private user code ---------------------------------------------------
-------*/
/* USER CODE BEGIN 0 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if (state == 0)
  {
        COUNT1 = TIM3->CCR1;
        COUNT_TIM3_OVC = 0;
        state = 1;
  }
  else if (state == 1)
  {
        COUNT2 = TIM3->CCR1;
        COUNT_Ticks = (COUNT2 + (COUNT_TIM3_OVC * 65536)) - COUNT1;
        F_CLK = HAL_RCC_GetPCLK2Freq()/(TIM3->PSC+1);
        signalFreq = (uint32_t) F_CLK/COUNT_Ticks;
        state = 0;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
      COUNT_TIM3_OVC++;
}
      /* USER CODE END 0 */
```

Changes at place 3

```
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
     /* USER CODE END 2 */
```

Finish

Part B. Pulse Width Modulation

In this experiment, we will generate ***two simultaneous PWM waveforms having the same time-period*** using the Capture/Compare feature of Timers. TIM1, which has special features for PWM generation, providing three PWM outputs, each with complementary outputs. This feature is very useful for controlling motors that need three-phase PWM.

A PWM waveform is generally specified in terms of the Time Period of the waveform and its' Duty Cycle. The latter is given by the ratio, expressed as a percentage, of the ON time to the Time Period of the waveform. We will call these parameters PERIOD and DUTY. A Timer is able to generate a PWM waveform with PERIOD determined by the value (Counter Period) loaded into the Automatic Reload Register (ARR), and DUTY determined by the value loaded into the Capture/Compare Register (CCR), in accordance with the following relationships:

DUTY = [CCR] / ([ARR] + 1) and PERIOD = [ARR] + 1.

Configure TIM1 by following the sequence of steps through STM32CubeIDE as given below.

1. Open CubeIDE and create a new project; follow the basic instructions for setting up the project for your respective microcontroller and Nucleo board.
2. Select RCC from the pinout pane, and choose
   High Speed Clock (HSE): Crystal/Ceramic Resonator.
3. Select TIM1 from the pinout pane, and make the following choices:
   a. Clock Source: Internal Clock.
   b. Channel 1: PWM Generation CH1.
   c. Channel 2: PWM Generation CH2.
4. Set the following parameters from the configuration pane:
   e. Pre-scalar: 1.
   f. Counter Mode: UP.
   g. Counter Period (ARR-16-bit Value): (as desired).
   h. Repetition Counter: 0.
5. Build the project to generate the code, and insert User codes as follows. The parameter values have to be entered as required.
   a. Define and assign appropriate value to Private Variable `int PERIOD`.
   b. Define and assign appropriate values to Variables *`int`* `DUTY1` and `int DUTY2`.
   c. Define the TIM1 Channels to be used and the computation of CCR values:

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
PERIOD = htim1.Init.Period + 1;
TIM1->CCR1= DUTY1*PERIOD/100;        // PWM with Duty Cycle DUTY1
TIM1->CCR2= DUTY2*PERIOD/100;        // PWM with Duty Cycle DUTY2
```

6. Display the PWM waveform on a DSO after the correct values are obtained on the SWV data trace timeline graph display.