Experiment 5: Timers

In this experiment, student will learn how use Timer modules. Timer (sometimes referred to as a counter) is a special piece of hardware inside many microcontrollers. Their function is simple: they count (up or down, depending on the configuration). A clock signal is sent into the Timer module, which causes the Timer to count as shown in Figure 1. According to Figure 1, the Timer will count up with frequency of 2 MHz or count up every 0.5 µsec. Timer can be used to generate true reference time in the real world because this clock signal has a specific frequency or period. Therefore, Timer is usually used for counting event, generating time-specific signal and other applications. There are two types of Timers in an STM32F103C8T6: 1. Advanced Timers and 2. General Purpose Timers. For simplicity, in this experiment, all Timers will be used in General purpose Timer application.

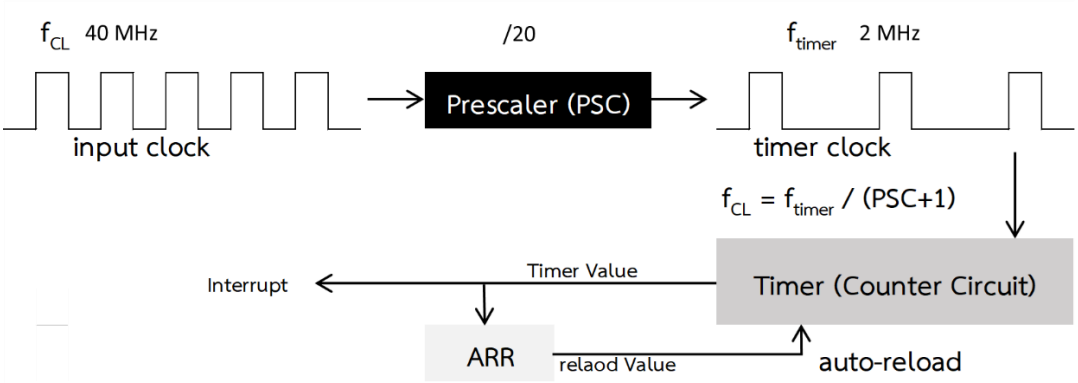

Figure 1. Basic Timer block diagram.

1. Timer Modules

General purpose Timers have all the features of a typical timer-counter module. They can be used for any timer-counter-related purpose such as PWM generation, input capture, time-base generation and output compare. These are the basic uses of General-purpose Timers. Basically, Timer module outputs can be assigned to GPIO pins. Input clock of Timers can be internal clock signal or external signal from GPIO pins. Therefore, the common functions for Timers are:

1. Output compare (OC): toggle a pin when a timer reaches a certain value.
2. Input capture (IC): measure the number of counts of a timer between events on a pin.

3. Pulse width modulation (PWM): toggle a pin when a timer reaches a certain value and on rollover. This event generates a duty cycle.

In this experiment, the Output Capture (OC) function is tested. The timer 3 will be used to generate exactly 50 msec delay time. If system clock is set to 72MHz and Prescaler is set to /72 then timer needs to count 50,000 times in order to achieve 50 msec exactly (50,000/1MHz = 50,000 μsec = 50 msec). This delay time from Timer3 will be used to create blink LED action, which turns LED on and off continuously, thus, the period of LED is 2 x 50msec = 100 msec = 0.1 sec.

Experiment 5.1

1. First, new project according to experiment 0 and assign project name as Lab5.
2. Write down the **timer3_delay( )** function to the **main.c** file in the project before main( ) function as below:

```
void timer3_delay()
{
     TIM_Cmd(TIM3,ENABLE);
     // wait until timer 3 done counting //
     while(TIM_GetFlagStatus(TIM3, TIM_FLAG_Update) != SET);
     TIM_ClearFlag(TIM3,TIM_FLAG_Update);
     TIM_Cmd(TIM3,DISABLE);
}
```

3. Write down the **GPIO_configuration** and **Timer3Setup( )** function to the **main.c** file in the project before main( ) function as below:

```
void GPIO_Configuration()
{
   GPIO_InitTypeDef GPIO_InitStruct;
   // Enable clock for GPIOB and GPIOC and GPIOA and AFIO
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC,ENABLE);
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO,ENABLE);
   // Configure PA0 and PA1 as Analog input
   GPIO_InitStruct.GPIO_Pin= GPIO_Pin_0 | GPIO_Pin_1;
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOA, &GPIO_InitStruct);
   // Configure PC13 as output push pull 50MHz
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;
   GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOC, &GPIO_InitStruct);

```

```
17.    // Configure PB15 as output push pull 50MHz
18.    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_15;
19.    GPIO_Init(GPIOB, &GPIO_InitStruct);
20.    // Configure PB1 as PWM output Timer 3 OC 4
21.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
22.    GPIO_Init(GPIOB, &GPIO_InitStruct);
23. }
24.
```

```
1.  void Timer3Setup()
2.  {
3.     TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
4.     /*
5.     TIM3 is connected to APB1 bus,
6.     if 72MHz clock is used
7.     Timer count frequency is set with
8.     timer_tick_frequency = Timer_default_frequency / (prescaller + 1)
9.     timer_tick_frequency = 72,000,000 / (71 + 1) = 72,000,000/72 = 1,000,000 Hz
10.    1,000,000 Hz -> 1 MHz -> T = 1 usec
11.    Timer count 50,000 time -> Timer period -> 50,000 usec = 50 msec
12.    */
13.    /*Set RCC for Timer 3*/
14.    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
15.    /*Timer Base Initialization for timer3  */
16.    TIM_TimeBaseInitStructure.TIM_Prescaler=0x47; // 0x47 = 71 //
17.    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
18.    TIM_TimeBaseInitStructure.TIM_Period = 49999;
19.    TIM_TimeBaseInitStructure.TIM_ClockDivision     = TIM_CKD_DIV1;
20.    TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);
21. }
```

4. Edit the **main( )** function in **main.c** as follow:

```
1.  int main(void)
2.  {
3.    unsigned char state = 0;
4.    GPIO_Configuration();
5.    GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
6.    Timer3Setup();
7.    while(1)
8.    {
9.       if (state == 0) {
10.          GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
11.          state = 1;     }
12.      else {
13.          GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
14.          state = 0;     }
15.      timer3_delay();
16. }
```

5. Compile and program into the board. Reset the STM32F103C8T6 and use the oscilloscope to find the real frequency of LED, compare the measured frequency with the theory and why they are difference. How to generate 1 sec delay using the code in this experiment?

Real LED frequency is _____ Hz

_____

_____

_____

_____

_____

_____

Lab instructor / TA signature

2. Pulse Width Modulation (PWM)

From the previous experiment, the C code cannot create exact time delay due to the compiler and header. Signal that generated from experiment 5.1 is called Pulse Width Modulation (PWM). This signal is common in robot and control applications, such as motor speed controllers. PWM has two important parameters which are frequency and duty cycle. Unlike the analog signal that contains amplitude, the PWM signal is digital signal that information is embedded into duty cycle of the signal. Duty cycle is expressed as a percentage of ON time respective to period of the digital signal (ON time + OFF time) as shown in Figure 2 which has duty cycle = ON/(ON+OFF) = 33.33%

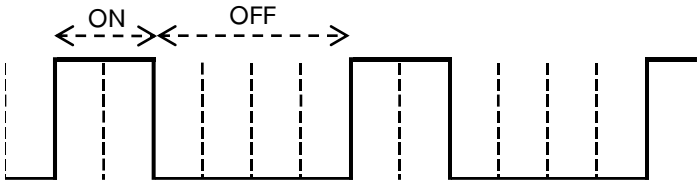

Figure 2. Duty Cycle

In order to generate PWM signal, Timer Output Compare mode is used. Timer will count until it reaches its set maximum counting number as usual but when it counts, the counted number is monitored by comparator and if the counted number greater than the desired value then the output logic will be toggled. The diagram of this Timer Output Compare mode is shown in Figure 3. The output logic from this mode propagates to assigned GPIO pin called channel programmed via software. In this experiment, Timer 3 output channel 4 is selected and it is PB1 as shown in Figure 4.

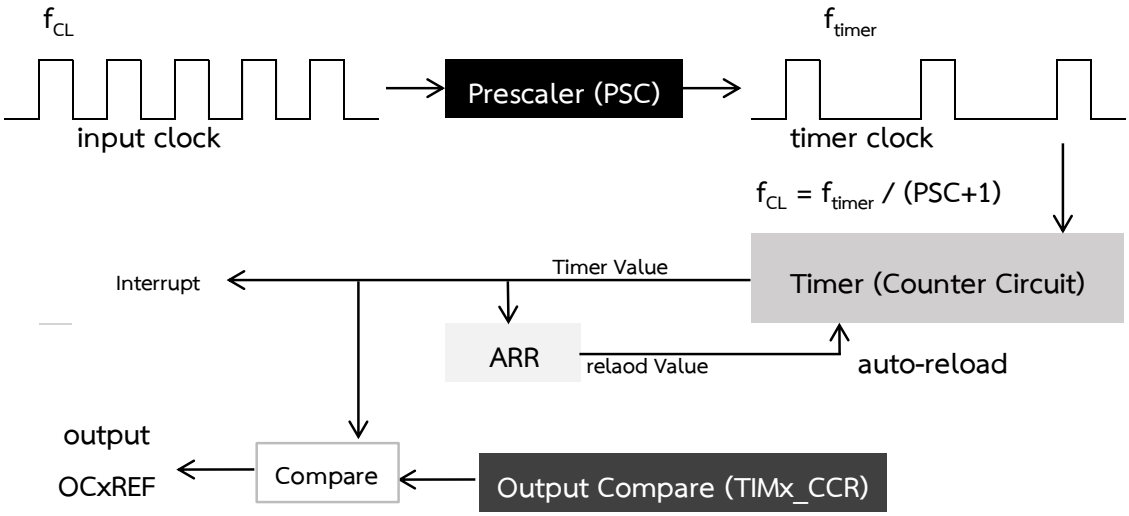

Figure 3. Output Compare mode.

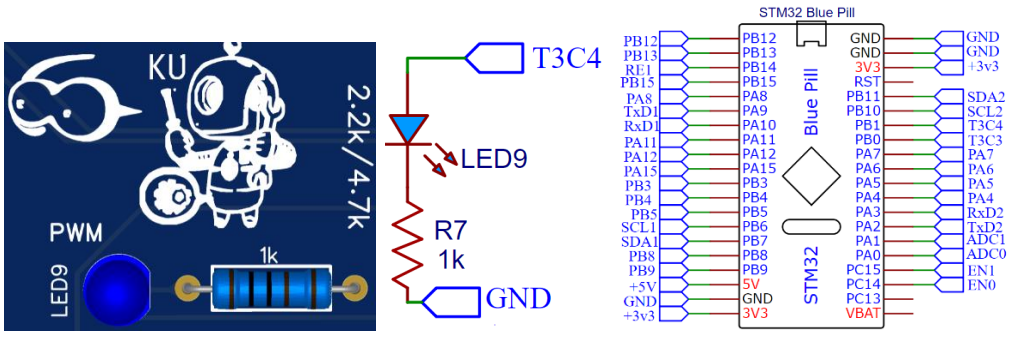

Figure 4. GPIO PB1 Pin is the Timer 3 output channel 4.

Experiment 5.2

1. Use the same project as experiment 5.1. Add the **DelayMC( )**, **ADC1_Init( )** and **readADC1( )** function from **Experiment 3** into the **main.c** .

2. Modify the **Timer3Setup( )** function from experiment 5.1 as follow:

```
void Timer3Setup()
{
   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
   TIM_OCInitTypeDef TIM_OCInitStructure;
   /*    if 72MHz clock is used
   timer_tick_frequency = Timer_default_frequency / (prescaller + 1)
   timer_tick_frequency = 72,000,000 / (143 + 1) = 72,000,000/144 = 500,000 Hz
   500,000 Hz -> 0.5 MHz -> T = 2 usec
   Timer count 50,000 time -> Timer period -> 100,000 usec = 100 msec
   PWM has 50% duty cycle from TIM_Pulse = 50% 0f Timer count period    */
   /*Set RCC for Timer 3*/
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
   /*Timer Base Initialization for timer3  */
   TIM_TimeBaseInitStructure.TIM_Prescaler =0x8F; // 0x47 = (72 x2) -1 =143  //
   TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
   TIM_TimeBaseInitStructure.TIM_Period = 50000 - 1;
   TIM_TimeBaseInitStructure.TIM_ClockDivision     = TIM_CKD_DIV1;
   TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);
   /* Timer3 Output Compare Setup */
   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
   TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable ;
   TIM_OCInitStructure.TIM_Pulse = 25000 -1 ;
   TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
   TIM_OC4Init(TIM3, &TIM_OCInitStructure); ///PB1 - Output PWM on channel 4
   /* Enable Timer3 */
   TIM_Cmd(TIM3,ENABLE);
}
```

3. Add **PWM_Set( )** function to **main.c** as below:

```
void PWM_Set(unsigned int Input)
{
    unsigned int Pulse = 0;
    TIM_OCInitTypeDef TIM_OCInitStructure;
    // 4095 = 50000 then Pulse = Input ADC *50000/4095
    Pulse = (unsigned int) (Input*50000/4095);
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable ;
    TIM_OCInitStructure.TIM_Pulse = Pulse-1;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
}
```

4. Edit the **main( )** function in **main.c** as follow:

```
int main(void)
{
    unsigned int ADCRead_value = 0;
    GPIO_Configuration();
    GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
    Timer3Setup();

```

```
8.      ADC1_Init();
9.      while(1)
10.     {
11.          ADCRead_value = readADC1(0);
12.          PWM_Set(ADCRead_value);
13.          DelayMC(100);
14.     }
15. }
```

5. Compile and program into the board. Rotate variable resistor VR0 on KU Lab board explain the result .Is the LED frequency is still the same for all resistor values?. Use oscilloscope to measure the frequency of the LED and compare the frequency with experiment 5.1. Which code can generate LED frequency closer to the theory?

_____

_____

_____

_____

_____

_____

_____

6. Why we need to set `TIM_TimeBaseInitStructure.TIM_Prescaler =0x8F` in this experiment.

_____

_____

Lab instructor / TA signature

3. Incremental Encoder Reader

An incremental encoder is a linear or rotary electromechanical device that has two output signals, A and B, which issue pulses when the device is moved. Together, the A and B signals indicate both the occurrence of and direction of movement. Unlike an absolute encoder, an incremental encoder does not indicate absolute position, it

only reports changes in position and, for each reported position change, the direction of movement. The output A, B signals are shown in Figure 5. Therefore, the incremental encoder requires software to collect the position changes and convert to the real position. Timers in STM32F103C8T6 can be configured for reading the incremental encoder. Timer will capture signal A and B and count up or down according to phase of signal A and B as shown in Figure 6. From Figure 6, Timer uses rising edge and falling edge to indicate the phase difference. Thus, Timer will increase the resolution of the Encoder by 4 times. To use Encoder in KU Lab board, jumper J0 and J1 must be set as shown in Figure 7.

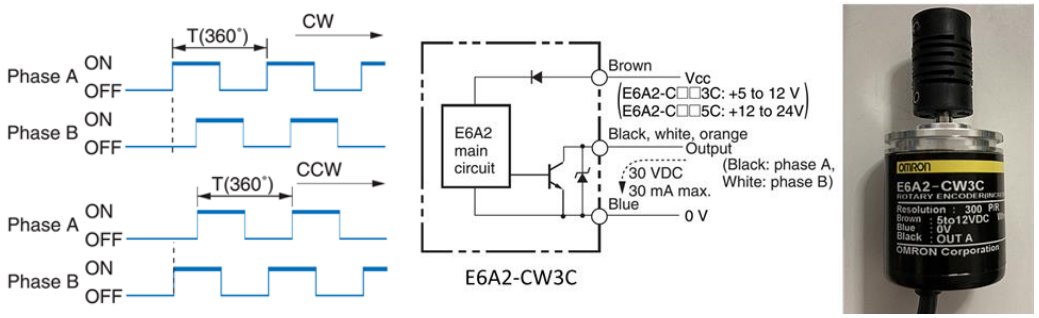

Figure 5. (Left) Encoder Signals and (Right) Real Encoder and wires detail.

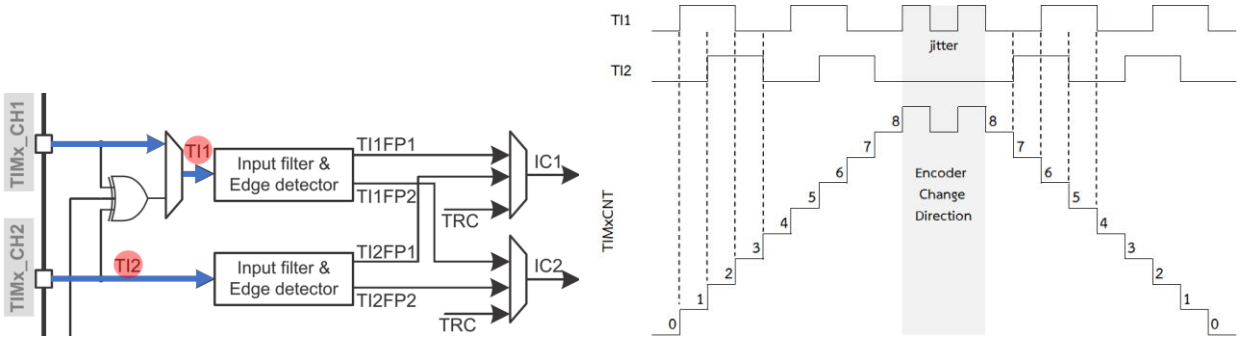

Figure 6. Encoder mode.

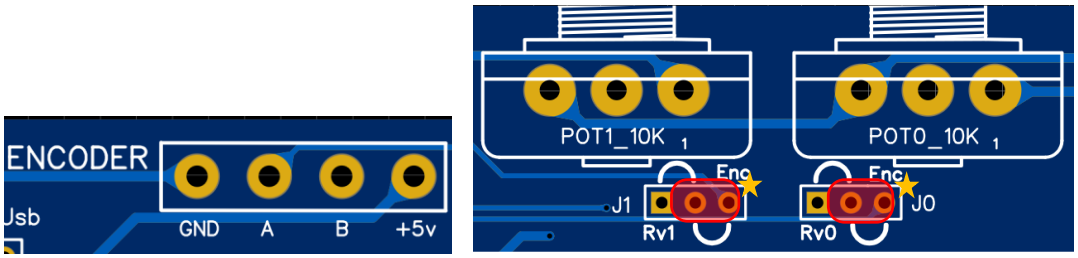

Figure 7. Jumpers setting.

Experiment 5.3

1. Use the same project as experiment 5.2. Add **USART_setup( )**, **USART1_sendC( )** and **USART1_putS( )** functions from **Experiment 2** into the **main.c** Add **Timer_Encoder_Setup( )** and **Encoder_Update( )** functions into **main.c** as follow:

```
int encoder_val=0;
#define MIDDLE  32767

void Timer_Encoder_Setup()
{
   GPIO_InitTypeDef GPIO_InitStructure;
   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
   TIM_OCInitTypeDef TIM_OCInitStructure;
   /*Set RCC for Timer 2*/
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
   /*Timer Base Initialization for timer2  */
   TIM_TimeBaseInitStructure.TIM_Prescaler=0x00;
   TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up ;
   TIM_TimeBaseInitStructure.TIM_Period = 65535;
   TIM_TimeBaseInitStructure.TIM_ClockDivision     = TIM_CKD_DIV1;
   TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);

   // Configures the encoder mode TI12 for TIM2
   TIM_EncoderInterfaceConfig(TIM2, TIM_EncoderMode_TI12,TIM_ICPolarity_Rising,
TIM_ICPolarity_Rising);
   /*Start timer 2 */
   TIM_Cmd(TIM2, ENABLE);
   /*set MIDDLE value*/
   TIM_SetCounter(TIM2, MIDDLE);
   /*Set RCC for GPIOA */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
   /*------------------ GPIO for Encoder --------------------*/
   /* PA0 : Timer2 Channel1 for Encoder phase A              */
   /* PA1 : Timer2 Channel2 for Encoder phase B              */
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_0  |  GPIO_Pin_1  ;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

```
void Encoder_Update(void)
{
      unsigned int ReadEncoder = 0;
      int dummy = 0;
      int Last_encoder =0;
      Last_encoder = encoder_val;
      // For Encoder Update
      ReadEncoder = TIM_GetCounter(TIM2);
```

```
9.      TIM_SetCounter(TIM2, MIDDLE);
10.     dummy = ReadEncoder - MIDDLE;
11.     encoder_val = Last_encoder + dummy;
12.     // one round is 300 pulses but the timer multiply it by 4 //
13.     // therefore one round is 300x4 = 1200 pulses/round //
14. }
```

2. Add function **DelayMC( )** from **Experiment 1**. Add function **__reverse ( )**, **itoa( )** from **Experiment 4**. Edit the **main( )** function in **main.c** as follow:

```
1.  int main(void)
2.  {
3.      unsigned int state = 0;
4.      unsigned char sbuf[20];
5.      GPIO_Configuration();
6.      GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
7.      Timer_Encoder_Setup();
8.      USART_setup();
9.      USART1_putS("Reading Encoder: \n\r\0");
10.     while(1)
11.     {
1.        if (state == 0) {
2.            GPIO_WriteBit(GPIOC,GPIO_Pin_13,1);
3.            state = 1;
4.        }
5.        if (state == 1) {
6.            GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
7.            state = 0;
8.        }
9.        Encoder_Update();
10.       itoa(encoder_val,sbuf,10);
11.       USART1_putS(sbuf);
12.       USART1_putS("\n\r");
12.       DelayMC(100);
13.     }
14. }
```

3. Compile and program into the board. Open ProcessingGrapher program, set to com port number and Baud rate to 115200 and click connect. Reset the STM32F103C8T6 and explain what's happen when turn the encoder clockwise and counter clockwise.

_____

_____

_____

_____

4. Write down a program for converting encoder value to the real angle.

Lab instructor / TA signature

Homework

1. Write a program to generate PWM signal with frequency of 100 Hz. The duty cycle of the PWM signal can be controlled by VR1 on KU Lab board via ADC value. The duty cycle has range between 1 msec (minimun) to 2 msec (maximum). The output of the PWM signal is Channel 3 of Timer 3.

Lab instructor / TA signature