Experiment No 11

PWM generation & Motor Speed control using STM32

Objectives:

- PWM generation using Timers
- Motor Speed control by varying duty cycle of PWM signal

11.1 PWM mode of Timers:

Three general-purpose timers (TIM-2, TIM3 & TIM-4) having 4 channels each, can perform the function of PWM. This mode enables the timers to generate square waves of particular frequencies. This lab consists of an example in which a PWM output of 1 kHz is generated.

Pulse Width Modulation mode allows generating a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing „110“ (PWM mode 1) or „111“ (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. The corresponding preload register must be enabled by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register. As the preload registers are transferred to the shadow registers only when an update event occurs,before starting the counter, the user must initialize all the registers by setting the UG bit in the TIMx_EGR register.

OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSSI and OSSR bits (TIMx_CCER and TIMx_BDTR registers). Refer tothe TIMx_CCER register description for more details. In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx ≤TIMx_CNT or TIMx_CNT ≤TIMx_CCRx (depending on the direction of the counter). The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.

The STM32 hardware timers are separate hardware blocks that can count from 0 to a given valuetriggering some events in between. In the PWM mode the timer controls the output of 1 or more output channels. When the counter value reaches 0, maximum or a **compare value** defined for each channel, the output value of the channel can be changed. Various configuration optionsdefine which events change the value and how it is changed.

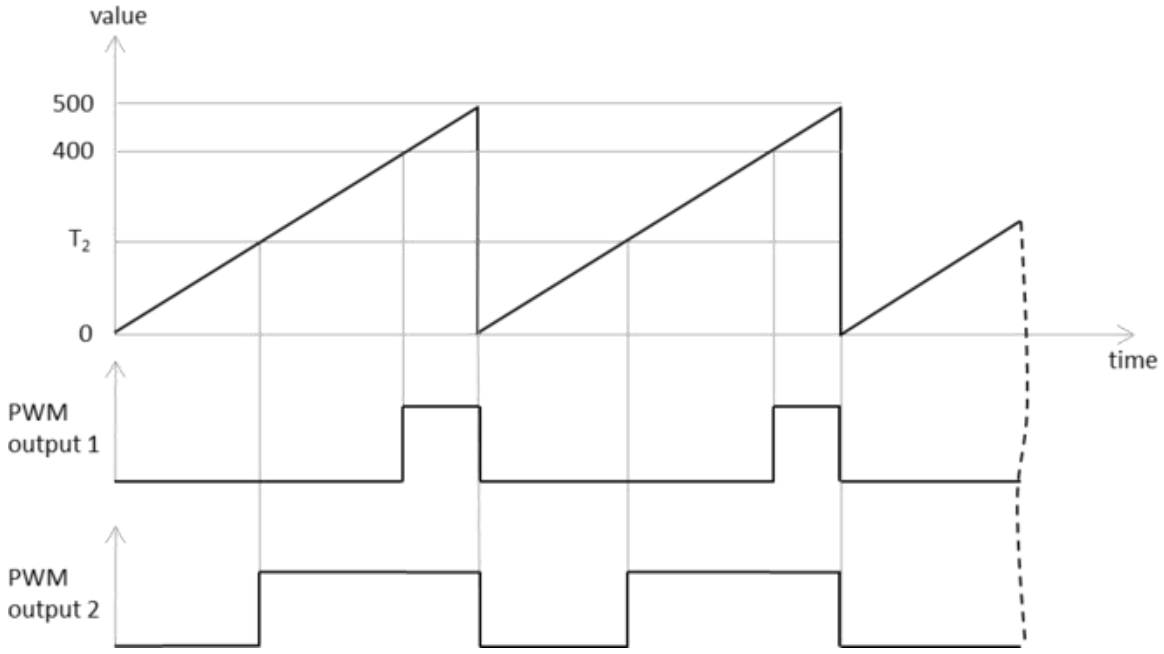


Figure 1: PWM Mode of Timers

Before we can use the PWM mode, we need to setup the timer so that it starts counting. We willconfigure it to count upwards with a period of 7199 and the prescaler to 10.

11.2 TIM2 to TIM4 main features:

General-purpose TIMx timer features include:

- 16-bit up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency byany factor between 1 and 65536.
- Up to 4 independent channels for:
  - – Input capture
  - – Output compare
  - – PWM generation (Edge- and Center-aligned modes)
  - – One-pulse mode output

- Synchronization circuit to control the timer with external signals and to interconnect severaltimers.
- Interrupt/DMA generation on the following events:
  - 11.2.1.1 Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - 11.2.1.2 Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - 11.2.1.3 Input capture
  - 11.2.1.4 Output compare
- Supports incremental (quadrature) encoder and hall-sensor circuitry for positioning purposes
- Trigger input for external clock or cycle-by-cycle current management

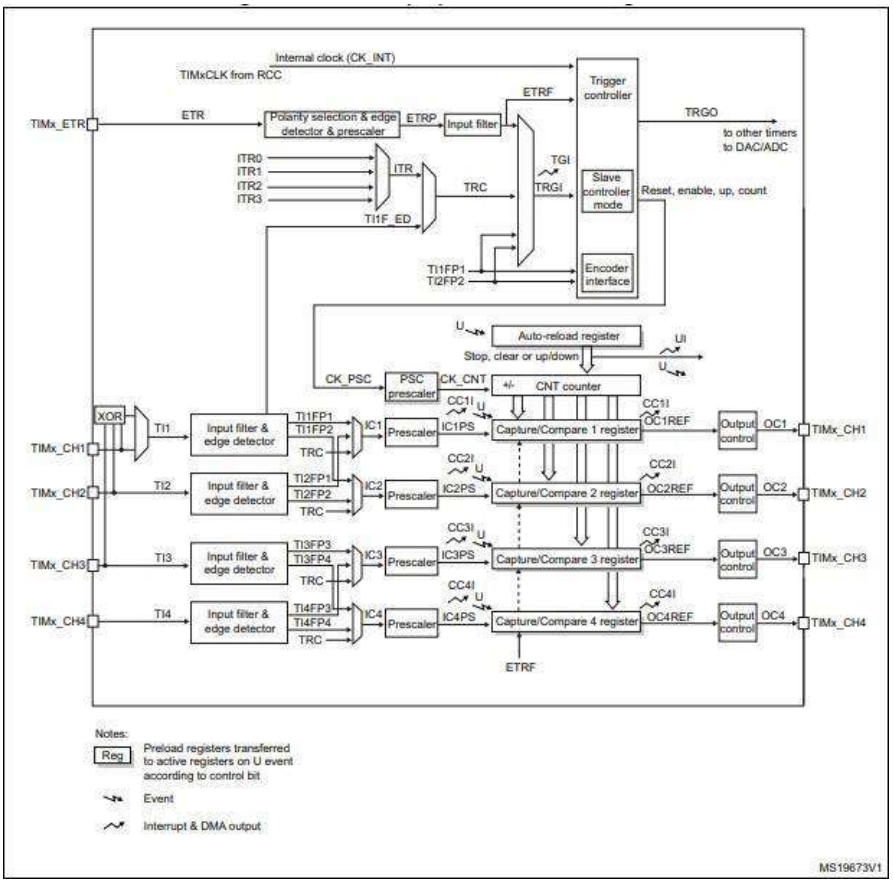


Figure 2: General Purpose Timer Block Diagram

11.3 Required Configurations for Timer:

```
void PWM_Init()
{
        // Initialization struct
        TIM_TimeBaseInitTypeDef
        TIM_TimeBaseInitStruct; TIM_OCInitTypeDef
        TIM_OCInitStruct; GPIO_InitTypeDef
        GPIO_InitStruct;

        // Step 1: Initialize TIM2
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,
        ENABLE);
        // Create 1kHz PWM
        // TIM2 is connected to APB1 bus that have default clock 72MHz
        // So, the frequency of TIM2 is 72MHz
        // We use prescaler 10 here
        // So, the frequency of TIM2 now is 7.2MHz

        TIM_TimeBaseInitStruct.TIM_Prescaler = 10;

        // TIM_Period determine the PWM frequency by this equation:
        // PWM_frequency = timer_clock / (TIM_Period + 1)
        // If we want 1kHz PWM we can calculate:
        // TIM_Period = (timer_clock / PWM_frequency) - 1
        // TIM_Period = (7.2MHz / 1kHz) - 1 = 7199

        TIM_TimeBaseInitStruct.TIM_Period = 7199;

        TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
        TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStruct);
        // Start TIM2
        TIM_Cmd(TIM2,
        ENABLE);

        //Continued
```

```
// Step 2: Initialize PWM

// Common PWM settings TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM2;
TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
// Duty cycle calculation equation:
// TIM_Pulse = (((TIM_Period + 1) * duty_cycle) / 100) - 1
// Ex. 25% duty cycle:
//                 TIM_Pulse = (((7199 + 1) * 25) / 100) - 1 = 1799
//                 TIM_Pulse = (((7199 + 1) * 75) / 100) - 1 = 5399
// We initialize PWM value with duty cycle of 0% TIM_OCInitStruct.TIM_Pulse = 0;
TIM_OC2Init(TIM2, &TIM_OCInitStruct);
TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);

// Step 3: Initialize GPIOA (PA0) RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,
ENABLE);
// Initialize PA0 as push-pull alternate function (PWM output) for
LEDGPIO_InitStruct.GPIO_Pin = GPIO_Pin_1; GPIO_InitStruct.GPIO_Mode =
GPIO_Mode_AF_OD; GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

11. 4 Libraries to include:

PWM_Generation.c

```
1  #include "stm32f10x.h"
2  #include "stm32f10x_rcc.h"
3  #include "stm32f10x_gpio.h"
4  #include "stm32f10x_tim.h"
5
6  void PWM_Init(void);
7  void delay(unsigned int nCount);
8
9  int brightness;
10
```

11.5 Main Function:

```
11  int main(void)
12  {
13
14    // Initialize PWM
15    PWM_Init();
16
17    while (1)
18    {
19      // Set LED brightness from 0 to max
20      for (brightness = 0; brightness <= 7199; brightness += 10)
21      {
22        TIM2->CCR2 = brightness;
23      }
24      // Set LED brightness from max to 0
25      for (brightness = 7199; brightness >= 0; brightness -= 10)
26      {
27        TIM2->CCR2 = brightness;
28      }
29    }
30  }
```

11.6 Delay Function:

```
92  // Delay function
93  void delay(unsigned int nCount)
94  {
95    unsigned int i, j;
96      for (i = 0; i < nCount; i++)
97        for (j = 0; j < 0x2AFF; j++);
98  }
```

11.7 Programming in KEIL:

To program the Timer in PWM mode in Keil, the first step is to create a new uVision Project andselecting the device which is STM32F103C8 just like we already did in the previous labs.

After saving the project in your desired directory, the device will be selected in the „SelectDevice for Target" dialogue box.

After that, include the libraries of Core, GPIO, Startup, Framework, RCC & TIM as shown in thefigure below:

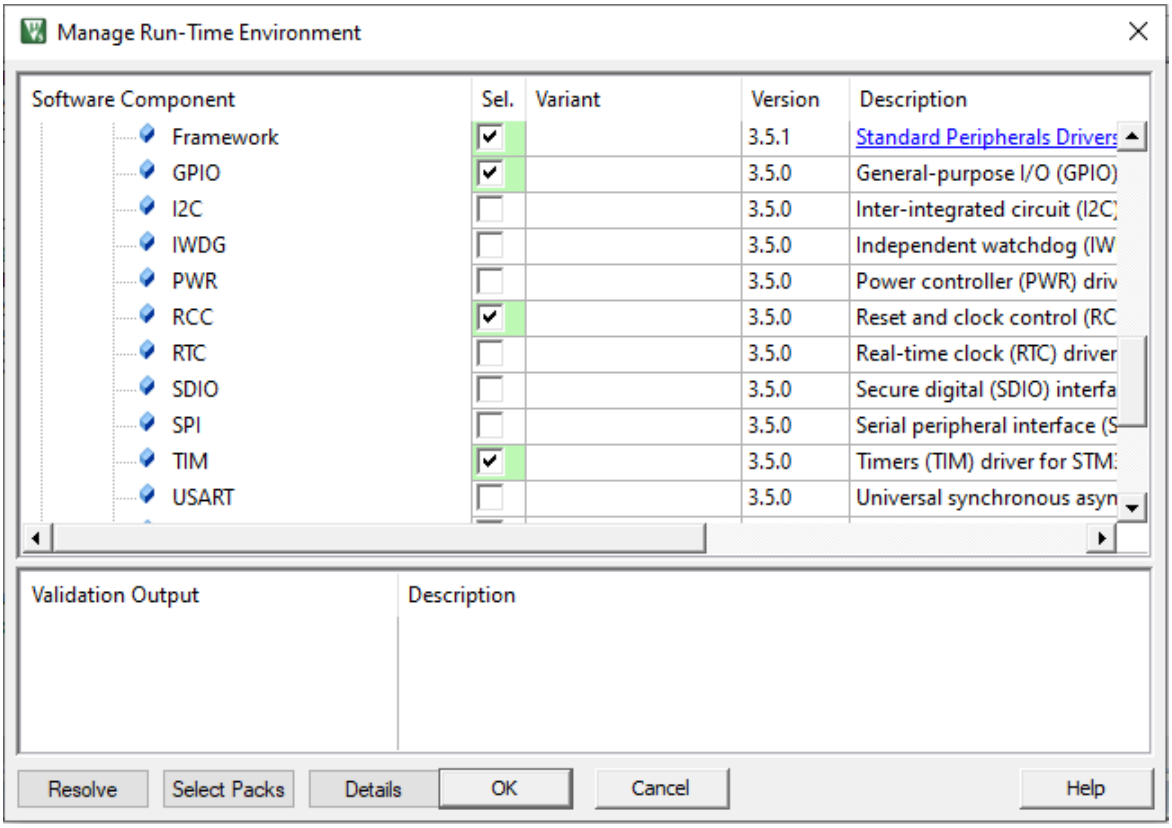


Figure 3: Libraries to Include

Then add a new item into your source group. Select the "C File (.c)" option, name your C fileand save it in your desired directory.

Lab Task (Psychomotor)

Activity 01:

Write a C-Program that starts a Timer in PWM mode to generate a PWM of 1 kHz. The program should increase the brightness of an LED to the maximum and then decreases it to the minimum by varying the Dutycycle of the generated PWM.

Activity 02:

Write a C-Program that can control a DC motor's Speed & Direction using the PWM mode of Timers. Speed and Direction can be controlled by varying the Dutycycle of the generated PWM accordingly.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date:** _____