

การทดลองที่ 7 การใช้งาน Pulse-Width Modulation

วัตถุประสงค์

- 1) เข้าใจการทำงานของ Pulse-Width Modulation
- 2) สามารถเขียนโปรแกรมควบคุมการทำงานของ Timer เพื่อสร้างสัญญาณ Pulse-Width Modulation

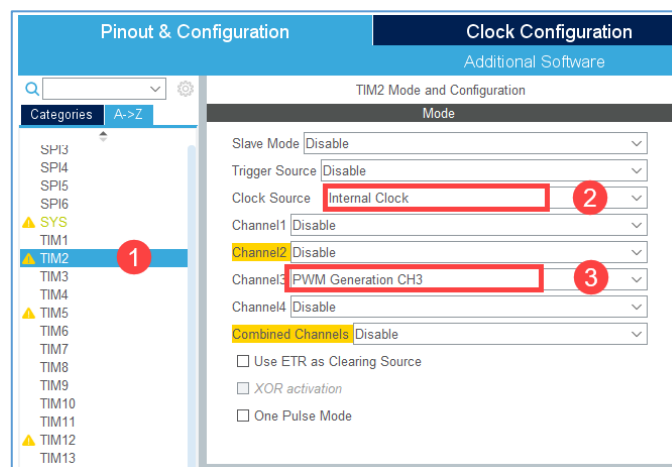
1. Pulse-Width Modulation Generation

วงจร Timer ภายในไมโครคอนโทรลเลอร์ STM32F767 จะมีโหมดการทำงานเพื่อสร้างสัญญาณ PWM สำหรับส่งออกไปควบคุมอุปกรณ์ภายนอก เช่น LED หรือมอเตอร์ได้ โดย**ความถี่**ของสัญญาณถูกควบคุมโดยรีจิสเตอร์ TIMx_ARR ส่วน Duty Cycle ถูกควบคุมโดยรีจิสเตอร์ TIMx_CCRx วงจร Timer 1 โมดูลประกอบไปด้วยช่องสัญญาณจำนวน 4 ช่องสัญญาณ ดังนั้นจึงสามารถสร้างสัญญาณ PWM จำนวน 4 สัญญาณได้จาก Timer 1 โมดูล โดยจะทำการเปรียบเทียบระหว่าง counter ของ Timer กับรีจิสเตอร์ TIMx_CCRx เพื่อสร้างสัญญาณ PWM ด้วยกัน 2 โหมด ดังนี้

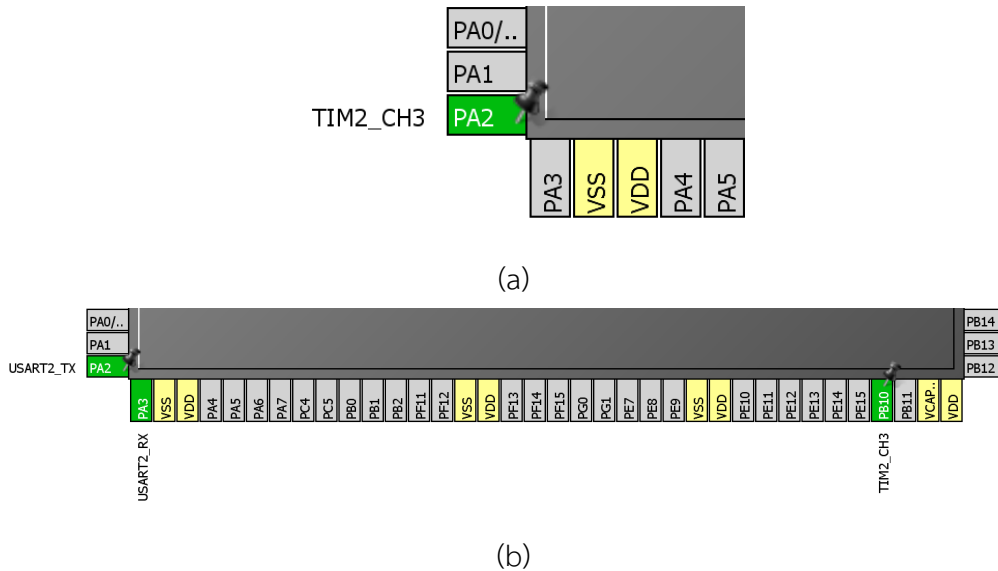
1. **โหมด 1** ในการนับขึ้นจะสร้างสัญญาณ PWM สถานะ SET เมื่อ counter ของโมดูล Timer มีค่าน้อยกว่าค่าในรีจิสเตอร์ TIMx_CCRx และเมื่อ counter มีค่ามากกว่ารีจิสเตอร์ดังกล่าวสัญญาณ PWM จะมีสถานะ RESET
2. **โหมด 2** ในการนับลงจะสร้างสัญญาณ PWM สถานะ SET เมื่อ counter ของโมดูล Timer มีค่ามากกว่าค่าในรีจิสเตอร์ TIMx_CCRx และเมื่อ counter มีค่าน้อยกว่ารีจิสเตอร์ดังกล่าวสัญญาณ PWM จะมีสถานะ RESET

2. การตั้งค่าในโปรแกรม STM32CubeMX

หากต้องการใช้งานช่องสัญญาณ 3 ของ TIM2 เพื่อสร้างสัญญาณ PWM สามารถตั้งค่า TIM2 ได้ดังรูปที่ 2.1 จากนั้นโปรแกรมจะกำหนดให้ค่า PA2 ทำหน้าที่จ่ายสัญญาณ PWM ดังรูปที่ 2.2 (a) แต่หากมีความจำเป็นต้องใช้งานขา PA2 ในโหมด UART2 สามารถเลือกให้ขา PB10 ทำหน้าที่จ่ายสัญญาณ PWM ของ TIM2_CH3 แทนได้ ซึ่งเป็น Additional Function ของขา PB10 ดังรูปที่ 2.2 (b)

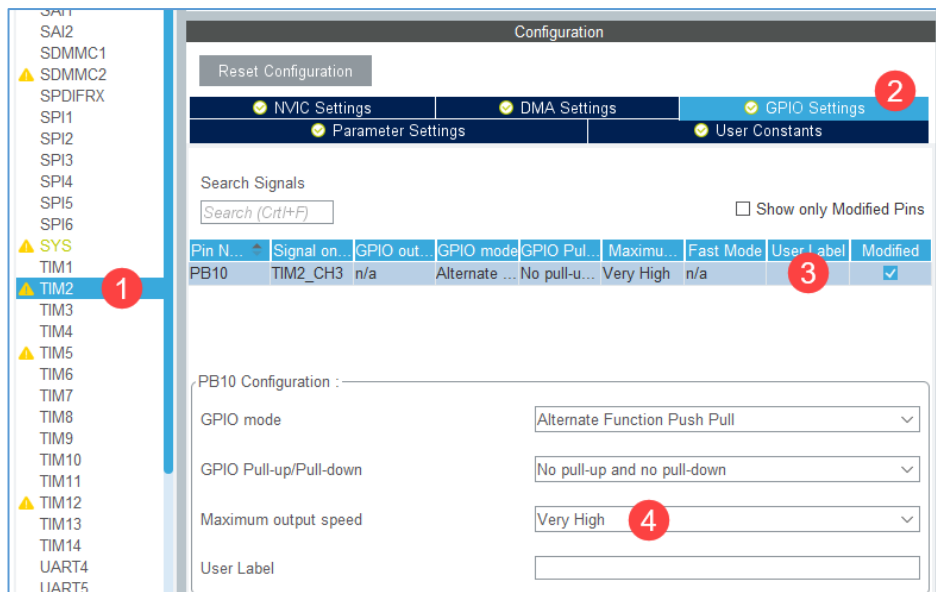


รูปที่ 2.1 แสดงการตั้งค่าให้โมดูล TIM2 Channel 3 จ่ายสัญญาณ PWM

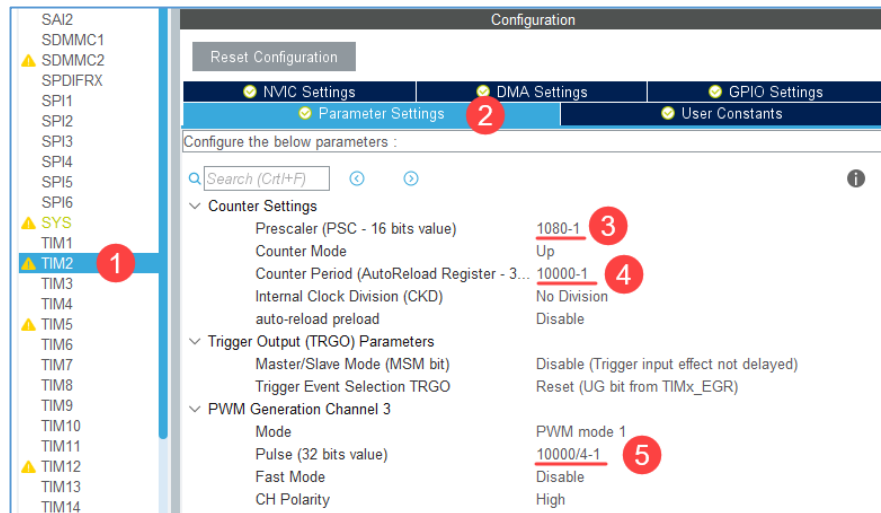


รูปที่ 2.2 แสดงขาเอาต์พุตที่สามารถนำสัญญาณ PWM จาก TIM2_CH3 ไปใช้งานได้

จากนั้นตั้งค่าให้ขา PB10 ทำงานที่ความเร็วสูงสุดดังรูปที่ 2.3 และตั้งค่าโมดูล TIM2 ได้ดังรูปที่ 2.4 ซึ่งตั้งค่าให้สัญญาณ PWM มีคาบเวลา 100 ms และมี Duty Cycle 25%



รูปที่ 2.3 แสดงการตั้งค่าขา PB10



รูปที่ 2.4 แสดงการตั้งค่าคาบเวลาและ Duty Cycle ของ TIM2_CH3

3. อธิบายการตั้งค่า

โปรแกรม STM32CubeMX ตั้งค่า TIM2 เพื่อสร้างสัญญาณ PWM ด้วยฟังก์ชัน MX_TIM2_Init และฟังก์ชัน HAL_TIM_MspPostInit ในไฟล์ tim.c และฟังก์ชัน MX_GPIO_Init ในไฟล์ gpio.c ดังรูปที่ 3.1 และรูปที่ 3.2

ฟังก์ชัน MX_TIM2_Init

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่าคาบเวลาและ Duty Cycle ของสัญญาณ PWM บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับที่กำหนดไว้ในโปรแกรม
- เริ่มต้นด้วยการประกาศตัวแปร Global htim2 สำหรับตั้งค่า TIM2 ที่ส่วนต้นของไฟล์ main.c

```
TIM_HandleTypeDef htim2;
```

- ส่วนฟังก์ชันเริ่มต้นด้วยการประกาศตัวแปร สำหรับตั้งค่าภายในโมดูล Timer

```
TIM_ClockConfigTypeDef sClockSourceConfig;  
TIM_MasterConfigTypeDef sMasterConfig;
```

- จากนั้นตั้งค่าคาบเวลาของสัญญาณ PWM โดยใช้ Clock Division, Prescaler และ Period เหมือนกับการทดลองที่ 6

```
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;  
htim1.Init.Prescaler = 1080-1;  
htim1.Init.Period = 10000-1;
```

- แล้วจึงตั้งค่าให้ TIM2 นับขึ้น

```
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
```

- จากนั้นตั้งค่าให้สัญญาณ PWM แบบ Active High มี Duty Cycle 25% ให้กับช่องสัญญาณที่ 3

```
sConfigOC.OCMode = TIM_OCMode_PWM1;  
sConfigOC.Pulse = 10000/4 -1;  
HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3);
```

```

TIM_HandleTypeDef htim2;

/* TIM2 init function */
void MX_TIM2_Init(void)
{
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};

    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 1080-1;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 10000-1;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = 10000/4-1;
    sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_TIM_MspPostInit(&htim2);
}

```

รูปที่ 3.1 แสดงการตั้งค่า TIM2 ในฟังก์ชัน MX_TIM2_Init()

ฟังก์ชัน MX_GPIO_Init

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า GPIO โดยจ่ายสัญญาณนาฬิกาให้กับ GPIO พอร์ต B

ฟังก์ชัน HAL_TIM_MspPostInit

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมาในไฟล์ stm32f7xx_hal_msp.c เพื่อเปลี่ยนการทำงานของขา PB10 จาก GPIO เป็น Additional function

```
static void MX_GPIO_Init(void)
{
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOB_CLK_ENABLE();
}
```

(a)

```
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    if(timHandle->Instance==TIM2)
    {
        /* USER CODE BEGIN TIM2_MspPostInit 0 */

        /* USER CODE END TIM2_MspPostInit 0 */

        __HAL_RCC_GPIOB_CLK_ENABLE();
        /**TIM2 GPIO Configuration
        PB10 -----> TIM2_CH3
        */
        GPIO_InitStruct.Pin = GPIO_PIN_10;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
        GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

        /* USER CODE BEGIN TIM2_MspPostInit 1 */

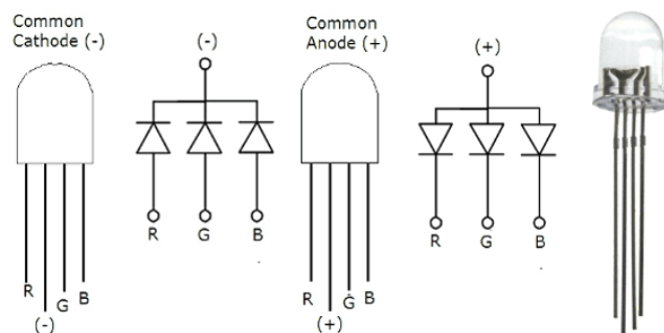
        /* USER CODE END TIM2_MspPostInit 1 */
    }
}
```

(b)

รูปที่ 3.2 แสดงการตั้งค่า PB10 ให้จ่ายสัญญาณ PWM จาก TIM2_CH3

4. การควบคุม RGB LED ด้วยสัญญาณ PWM

RGB LED คือ LED ที่ภายในประกอบไปด้วย LED ที่เป็นแม่สีจำนวน 3 สี ได้แก่ สีแดง สีเขียว และสีน้ำเงิน ทำให้สามารถนำมาสร้างเป็นแสงสีต่างๆ ได้ตามต้องการ โดยการกำหนดความเข้มแสงให้ LED แต่ละสีด้วยสัญญาณ PWM โดย LED ทั้งสามมีขา Common ร่วมกัน 1 ขา ซึ่งมีทั้งแบบ Common Anode และ Common Cathode ดังรูปที่ 4.1 และการใช้งานจะต้องต่อตัวต้านเพื่อจำกัดกระแสจำนวน 3 ตัวด้วย



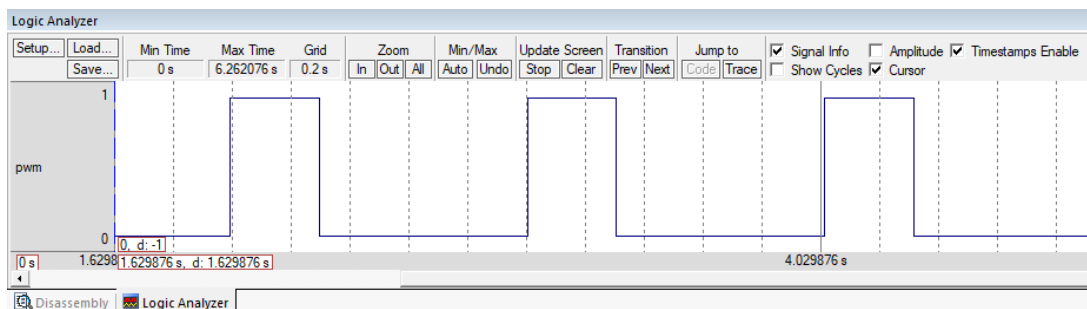
รูปที่ 4.1 แสดงรายละเอียดของขาสัญญาณ RGB LED แบบ Common Cathode และ Common Anode

5. การทดลอง

1. การสร้างสัญญาณ PWM จาก TIM2_CH3

จงเขียนโปรแกรมควบคุม TIM2 ช่องสัญญาณ 3 เพื่อสร้างสัญญาณ PWM ที่มีคาบเวลา 100 ms และมี Duty Cycle 25% ที่ขา PB10

- ตั้งค่า TIM2 ดังรูปที่ 2.1 รูปที่ 2.3 และรูปที่ 2.4
- ประกาศตัวแปรโกลบอล `uint8_t pwm`
- ต่อ LED ที่ขา PB10
- เพิ่มคำสั่งใน while loop เพื่อให้ไมโครคอนโทรลเลอร์สร้างสัญญาณ PWM ที่มีคาบเวลา 100 ms มี Duty Cycle 25% ตามที่ตั้งค่าไว้ เป็นระยะเวลานาน 100 ms แล้วจึงหยุดสร้างสัญญาณ PWM ได้ดังนี้
 - `HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);`
 - `HAL_Delay(100);`
 - `HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_3);`
 - `pwm = (GPIOB->IDR & 0xGPIO_PIN_10) >>10;`
- สังเกตความสว่างของ LED ที่ขา PB10 เมื่อเทียบกับการป้อนลอจิก 1
- ตรวจสอบคาบเวลาและ Duty Cycle ของสัญญาณ PWM ที่สร้างขึ้น โดยใช้ Logic Analyzer ของ Keil และ BitScope เพื่อดูค่าตัวแปร `pwm` แบบบิต ดังรูปที่ 5.1



รูปที่ 5.1 แสดงสัญญาณ PWM ที่วัด โดยใช้ Logic Analyzer ของ Keil

2. การกำหนด Duty Cycle ให้กับสัญญาณ PWM

จงเขียนโปรแกรมควบคุม TIM2 ช่องสัญญาณ 3 เพื่อสร้างสัญญาณ PWM ที่มี Duty Cycle แตกต่างกันโดยการแก้ไขค่าในรีจิสเตอร์ TIM2_CCR3

- สร้างตัวแปร `float dutyCycle = 0.5` ในฟังก์ชัน main เพื่อใช้ปรับเปลี่ยนค่า Duty Cycle
- เปลี่ยนคำสั่งใน while loop ให้เป็นคำสั่งดังต่อไปนี้แทน
 - `htim2.Instance -> CCR3 = (10000-1) * dutyCycle;`
 - `HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);`
 - `HAL_Delay(100);`
 - `HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_3);`
 - `pwm = (GPIOB->IDR & 0xGPIO_PIN_10) >>10;`

- ใช้ Logic Analyzer ของ Keil และ BitScope เพื่อตรวจสอบผลลัพธ์ดังการทดลองที่ 1 แล้วบันทึกรูปสัญญาณที่ได้ลงในตารางที่ 5.1

ตารางที่ 5.1 สำหรับบันทึกผลการทดลองที่ 2

ค่าตัวแปร dutyCycle	รูปสัญญาณ PWM ที่ได้	Duty cycle ของ PWM
0.5		
0.25		
0.75		
1.0		
2.0		

3. การผสมสีจากหลอดไฟ RGB LED

จงสร้างสัญญาณ PWM จำนวน 3 สัญญาณเพื่อควบคุมความเข้มของสีแดง สีเขียว และสีน้ำเงิน โดยกดปุ่ม r ที่แป้นพิมพ์เพื่อปรับความเข้มสีแดง ปุ่ม g เพื่อปรับความเข้มสีเขียว และปุ่ม b เพื่อปรับความเข้มสีน้ำเงิน ในการกดแต่ละครั้งจะเพิ่มความเข้มขึ้น 20% เริ่มจาก 0% – 100% แล้ววนมาที่ 0% อีกครั้ง โดยสามารถเลือกหาสัญญาณ PWM ได้ตามต้องการ และตอนตรวจการทดลองให้ใช้ Logic Analyzer ของ BitScope เป็นเครื่องมือวัด

PWM สำหรับสีแดง Timer Channel GPIO ขา

PWM สำหรับสีเขียว Timer Channel GPIO ขา

PWM สำหรับสีน้ำเงิน Timer Channel GPIO ขา

ใบตรวจการทดลองที่ 7

Microcontroller Application and Development 2563

วัน/เดือน/ปี _____ กลุ่มที่ _____

1. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____
2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____
3. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองที่ 3 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. ในฟังก์ชัน `MX_TIM2_Init` ของการทดลองที่ 1 หากเปลี่ยนคำสั่ง
`sConfigOC.OCMode = TIM_OCMode_PWM1;`
เป็น `sConfigOC.OCMode = TIM_OCMode_PWM2;`
ให้ผลเหมือนเดิมหรือต่างจากเดิมอย่างไร

.....

.....

.....

.....

.....

.....