<u>การทดลองที่ 6</u> การใช้งาน Timer

วัตถุประสงค์

1) เข้าใจการทำงานของ Timer
2) สามารถเขียนโปรแกรมควบคุมการทำงานของ Timer

1. Timer

STM32F767 มี Timer จำนวน 18 โมดูล โดยมี TIM1 และ TIM 8 เป็น advanced-control timers และ TIM2 – TIM5 เป็น general-purpose timers เป็นต้น โดย Timer มีคุณสมบัติดังนี้

- counter มีขนาด 16/32 บิต
- prescaler ขนาด 16 บิต
- สามารถสร้างสัญญาณ interrupt เมื่อ timer นับครบตามค่าที่กำหนด
- Timer ส่วนใหญ่สามารถแยกใช้งานได้ 4 ช่องสัญญาณอิสระจากกัน โดยสามารถนำไปใช้ในโหมดต่างๆ ดังนี้ โหมดตรวจจับสัญญาณอินพุต (input capture), โหมดเปรียบเทียบข้อมูล (output capture), โหมดสร้างสัญญาณ PWM (pulse width modulation generation), โหมดสร้างสัญญาณพัลส์เดี่ยว (one pulse mode output)

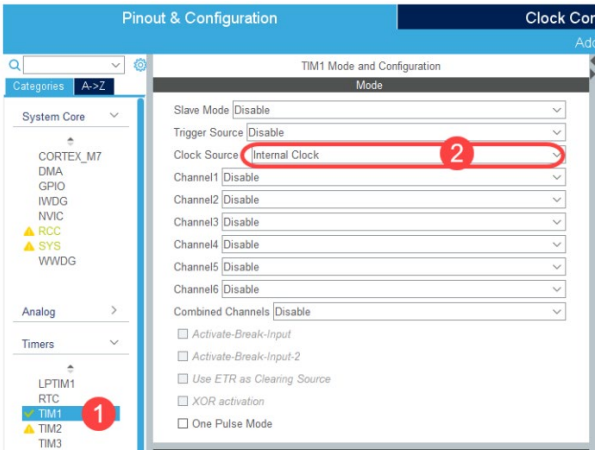
โดยสามารถที่จะตั้งค่า Timer ให้นับขึ้นหรือนับลงได้ Timer แต่ละโมดูลนั้นเชื่อมต่ออยู่กับบัสที่ต่างกันดังตารางที่ 1.1

ตารางที่ 1.1 แสดงการเชื่อมต่อ Timer กับ APB

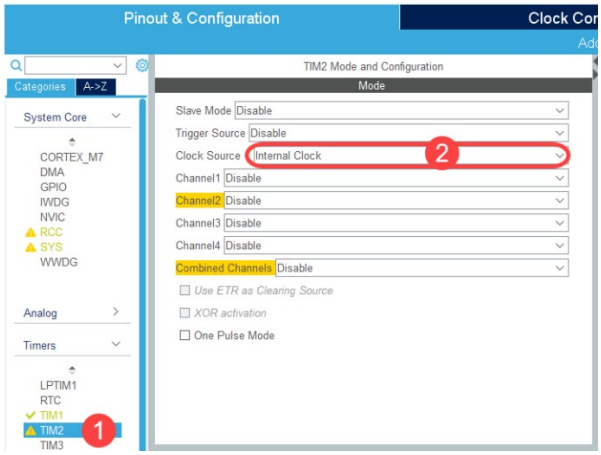
Bus	MAX Bus Frequency (MHz)	MAX Timer Frequency (MHz)	Module
APB1	54	108	TIM2 TIM3 TIM4 TIM5 TIM12 TIM13 TIM14
APB2	108	216	TIM1 TIM8 TIM9 TIM10 TIM11

2. การตั้งค่าในโปรแกรม STM32CubeMX

2.1 Enable Timer ที่ต้องการด้วยการกำหนดแหล่งจ่ายสัญญาณนาฬิกา (Clock Source) ของวงจร เช่น ต้องการใช้งานโมดูล TIM1 และ TIM2 ให้ตั้งค่า Clock Source เป็น Internal Clock ดังรูปที่ 2.1



(a)



(b)

รูปที่ 2.1 แสดงการตั้งค่าโมดูล (a) TIM1 และ (b) TIM2

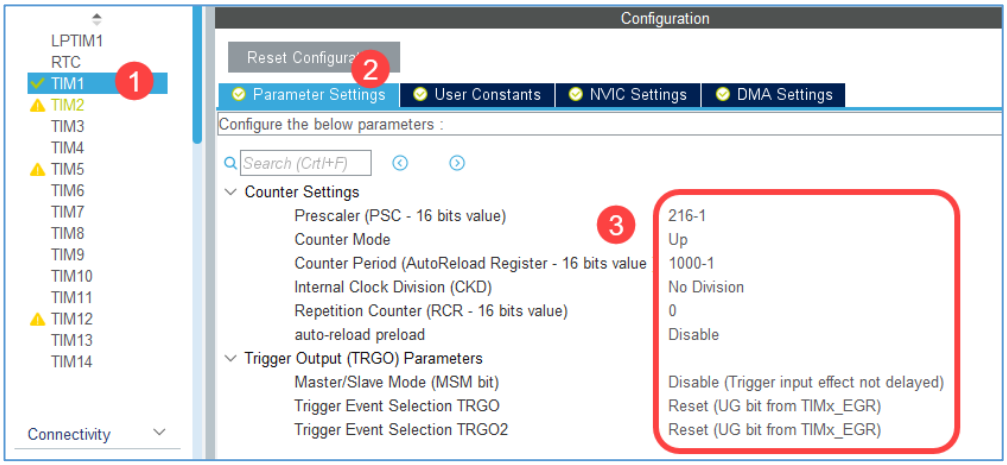
2.2 จากนั้นต้องตั้งค่าให้โมดูล Timer นับตามระยะเวลาที่ต้องการ โดยการกำหนด Prescaler, Counter Mode และ Counter Period โดยคำนวณได้จาก

ระยะเวลาที่ต้องการนับ (Time Interval) = (Clock Division x Prescaler x Period) / APBx Bus Speed

หากต้องการให้โมดูล TIM1 นับเป็นระยะเวลา 1 ms สามารถตั้งค่าได้ดังตัวอย่างต่อไปนี้และตั้งค่าในโปรแกรม STM32CubeMX ได้ดังรูปที่ 2.2 ซึ่งต้องลบค่าที่ต้องการออกด้วย 1 เสมอ

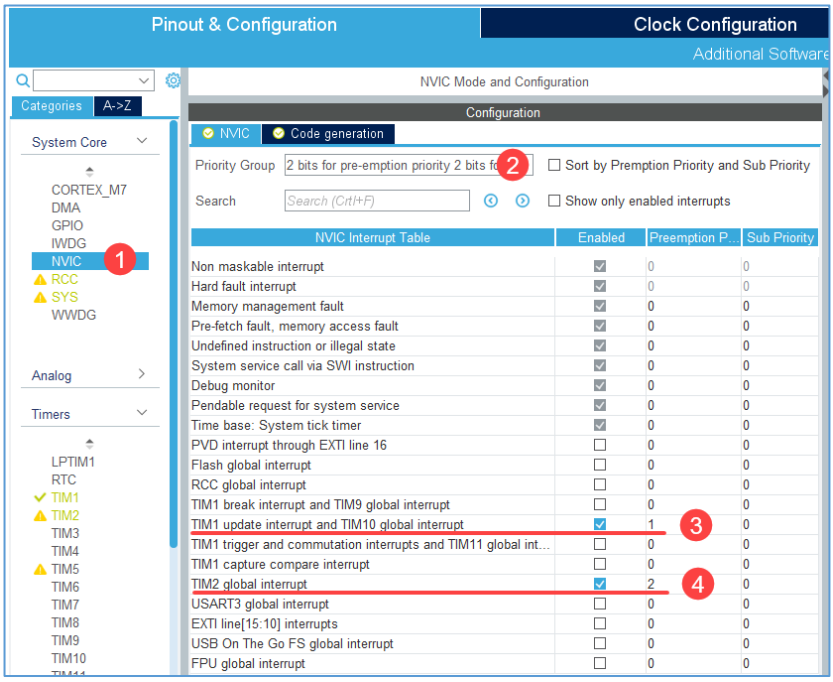
(1 x 216 x 1000) / 216 MHz = 1 ms

โดย Prescaler และ Period ของ TIM1 มีขนาด 16 บิต ClockDivision มีค่า 1, 2 หรือ 4



รูปที่ 2.2 แสดงการตั้งค่าโมดูล TIM1 ให้นับเป็นระยะเวลา 1 ms

2.3 เมื่อ Enable โมดูล Timer และได้ตั้งค่าให้ Timer นับตามระยะเวลาที่กำหนดแล้ว เมื่อ Timer นับครบระยะเวลาที่ตั้งไว้จะสร้างสัญญาณ Interrupt ขึ้นมา ดังนั้นขั้นตอนต่อไปคือกำหนด Priority ให้กับ Timer Interrupt โดย TIM1 เป็นโมดูลที่มีความซับซ้อนกว่า Timer โมดูลอื่นๆ จึงมีสัญญาณ Interrupt หลายประเภท หากต้องการให้ TIM1 เกิดสัญญาณ Interrupt เมื่อนับครบ ต้องเลือกใช้ TIM1 Update Interrupt ส่วน TIM2 มีสัญญาณ Interrupt แบบเดียวซึ่งจะถูกสร้างขึ้นเมื่อ TIM2 นับครบระยะเวลา ได้แก่ TIM2 Global Interrupt สามารถกำหนด Priority ของสัญญาณ Timer Interrupt ที่โมดูล NVIC ในโปรแกรม STM32CubeMX ได้ดังรูปที่ 2.3



รูปที่ 2.3 แสดงการตั้งค่าโมดูล NVIC

3. อธิบายการตั้งค่า Timer

โปรแกรม STM32CubeMX ตั้งค่า TIM1 และ TIM2 ด้วยฟังก์ชัน `MX_TIM1_Init` และ `MX_TIM2_Init` ตามลำดับในไฟล์ `tim.c` ดังรูปที่ 3.1 และรูปที่ 3.2 เนื่องจากโมดูล Timer ที่เลือกใช้ไม่ได้รับหรือส่งข้อมูลใดๆ จึงไม่ต้องตั้งค่าให้กับ GPIO ดังนั้นจึงไม่มีโค้ดใดๆ อยู่ในฟังก์ชัน `MX_GPIO_Init`

ฟังก์ชัน `MX_TIM1_Init`

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า TIM1 บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับที่กำหนดไว้ในโปรแกรม
- เริ่มต้นด้วยการประกาศตัวแปร Global `htim1` สำหรับตั้งค่า TIM1 ที่ส่วนต้นของไฟล์ `tim.c`

```
TIM_HandleTypeDef htim1;
```

- ส่วนฟังก์ชันเริ่มต้นด้วยการประกาศตัวแปร สำหรับตั้งค่าภายในโมดูล Timer

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

- จากนั้นตั้งค่า Clock Division ซึ่งทำหน้าที่หารความถี่ของสัญญาณนาฬิกาที่จ่ายให้โมดูล TIM1

```
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

- แล้วตั้งค่า Prescaler ซึ่งเป็น Counter ตัวแรกที่ได้รับสัญญาณนาฬิกาจาก Clock Division หากต้องการให้ Prescaler นับ 216 ค่า (0 - 215) ต้องตั้งค่า Prescaler ด้วย 215 และเพื่อป้องกันความสับสนสามารถเขียนเป็น 216 - 1

```
htim1.Init.Prescaler = 216-1;
```

- จากนั้นตั้งค่า Period ซึ่งเป็น Counter ตัวสุดท้ายในโมดูล รับสัญญาณนาฬิกาจาก Prescale การตั้งค่า Period เหมือนกับกรณีการตั้งค่า Prescaler

```
htim1.Init.Period = 1000-1;
```

- แล้วจึงตั้งค่าให้ TIM1 นับขึ้นหรือนับลง

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

```
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

/* TIM1 init function */
void MX_TIM1_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 216-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 1000-1;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

รูปที่ 3.1 แสดงการตั้งค่า TIM1 ในฟังก์ชัน `MX_TIM1_Init()`

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

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 0;
  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();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

รูปที่ 3.2 แสดงการตั้งค่า TIM2 ในฟังก์ชัน `MX_TIM2_Init()`

ฟังก์ชัน `HAL_TIM_Base_MspInit`

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา ในไฟล์ `tim.c` เพื่อตั้งค่า Priority ให้กับสัญญาณ Interrupt ของโมดูล Timer ตามที่ได้กำหนดไว้ในโปรแกรม ดังรูปที่ 3.3

```c
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
{

  if(tim_baseHandle->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspInit 0 */

  /* USER CODE END TIM1_MspInit 0 */
    /* TIM1 clock enable */
    __HAL_RCC_TIM1_CLK_ENABLE();

    /* TIM1 interrupt Init */
    HAL_NVIC_SetPriority(TIM1_UP_TIM10_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);
  /* USER CODE BEGIN TIM1_MspInit 1 */

  /* USER CODE END TIM1_MspInit 1 */
  }
```

รูปที่ 3.3 แสดงการตั้งค่า Interrupt Priority ให้กับ TIM1

4. Interrupt Service Routine ของ Timer

หากต้องการให้โมดูล Timer เริ่มต้นทำงาน (เริ่มต้นนับ) แล้วสร้างสัญญาณ Interrupt เมื่อนับครบตามที่ได้ตั้งค่าไว้ ต้องเรียกใช้ฟังก์ชัน `HAL_TIM_Base_Start_IT` ภายหลังการตั้งค่า เช่น

- `HAL_TIM_Base_Start_IT (&htim1)`
- `HAL_TIM_Base_Start_IT (&htim2)`

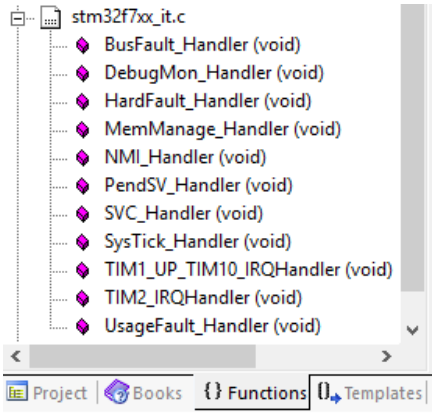
โดยมีรายละเอียดดังรูปที่ 4.1

HAL_TIM_Base_Start_IT

Function Name	**HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)**
Function Description	Starts the TIM Base generation in interrupt mode.
Parameters	• **htim:** : TIM handle
Return values	• HAL status

รูปที่ 4.1 แสดงรายละเอียดของฟังก์ชัน `HAL_TIM_Base_Start_IT`

สำหรับ ISR ของ TIM1 และ TIM2 ที่ได้ตั้งค่าไว้ดังรูปที่ 2.3 ได้แก่ ฟังก์ชัน `TIM1_UP_TIM10_IRQHandler` และ `TIM2_IRQHandler` ในไฟล์ `stm32f7xx_it.c` ดังรูปที่ 4.2



รูปที่ 4.2 แสดง Interrupt Service Routine ของ TIM1 และ TIM2

5. การทดลอง

1. การใช้งาน TIM1 และ ISR ของ TIM1

จงเขียนโปรแกรมควบคุม TIM1 เพื่อให้เพิ่มค่าตัวแปรขนาด 32 บิต ครั้งละ 1 ค่าทุกๆ 1 ms และแสดงค่าตัวแปรออกทาง UART3 ทุกๆ 400 ms

- โดยตั้งค่า TIM 1 ให้นับเป็นระยะเวลา 1 ms ได้ดังรูปที่ 2.1, รูปที่ 2.2 และรูปที่ 2.3
- จากนั้นประกาศตัวแปร **`uint32_t count`** ให้เป็นตัวแปรชนิด Global ในไฟล์ `main.c` และประกาศซ้ำในไฟล์ `stm32f7xx_it.c` โดยใช้คีย์เวิร์ด extern นำหน้า
- ในฟังก์ชัน `TIM1_UP_TIM10_IRQHandler` ซึ่งเป็น ISR ของ TIM1 ให้เขียนโปรแกรมเพิ่มค่าตัวแปร count ครั้งละ 1 ค่าเพิ่มลงไป `(count++)`
- สั่งให้ TIM1 เริ่มต้นนับและสร้างสัญญาณ Interrupt เมื่อนับครบด้วยการเรียกใช้ฟังก์ชัน `HAL_TIM_Base_Start_IT (&htim1)` หลังจากการตั้งค่า TIM1 แต่**ก่อนเข้า Infinite Loop** ในฟังก์ชัน `main`
- ให้สร้างฟังก์ชัน `displayNumber` ในไฟล์ `main.c` เพื่อรับค่าจำนวนเต็มแบบไม่มีเครื่องหมายขนาด 32 บิต แล้วแสดงค่าของตัวแปรในรูปของเลขฐาน 10 ทาง UART3
- เรียกใช้ฟังก์ชัน `displayNumber(count)` และ `HAL_Delay(400)` ภายใน Infinite Loop ในฟังก์ชัน `main`

2. ใช้ TIM1 และ TIM2 สร้างนาฬิกาโดยแสดงผลผ่าน UART3

จงเขียนโปรแกรมเพื่อแสดงเวลาในรูปของ นาที:วินาที (MM:SS) ผ่านทาง UART3 ที่เดินเท่ากับเวลาจริงโดยใช้สัญญาณ Interrupt จาก TIM1 (การส่งข้อมูลทาง UART ให้ปิดท้ายข้อความ (string) ด้วย carriage return (`'\r'`) โดยไม่ใช้ line feed (`'\n'`) เพื่อให้สามารถเขียนทับที่ตำแหน่งเดิม)

กำหนดให้แสดงผลผ่าน UART3 ทุกๆ 400 ms โดยให้ใช้ TIM2 เพื่อจับเวลาแทนฟังก์ชัน `HAL_Delay` พร้อมทั้งบันทึกการตั้งค่าของ TIM2 และแสดงการคำนวณ

ระยะเวลาที่ TIM2 นับ (Time Interval) .....

ClockDivision .....

Prescaler .....

Period .....

แสดงการคำนวณ .....

.....

.....

.....

.....

ใบตรวจการทดลองที่ 6

Microcontroller Application and Development 2563

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

3. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 1 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

การทดลองข้อ 2 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. การใช้โมดูล Timer เพื่อ Toggle LED ทุกๆ 500 ms ส่งผลต่อลักษณะการทำงานของไมโครคอนโทรลเลอร์หรือไม่อย่างไร หากเปรียบเทียบกับการใช้**ฟังก์ชัน `delay` แบบ Nested For-Loop**