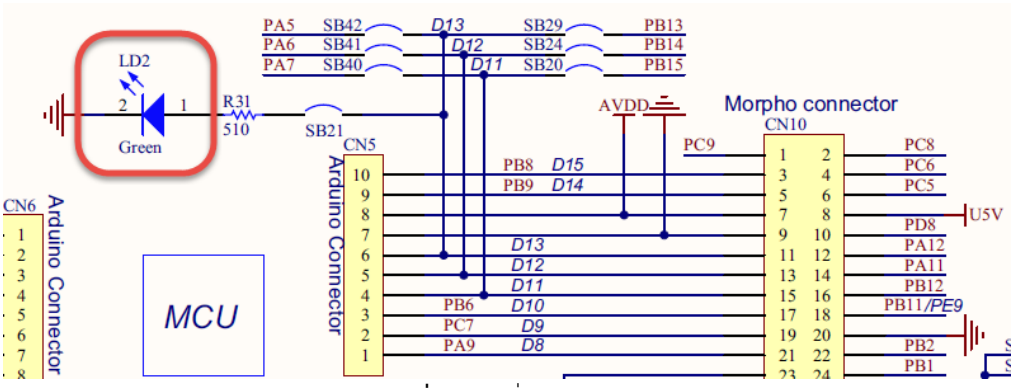
<u>การทดลองที่ 1</u> การใช้งาน LED บนบอร์ดและการใช้งาน Keil Logic Analyzer ในโหมดดีบัก

วัตถุประสงค์

1) สามารถเขียนโปรแกรมควบคุม LED บนบอร์ดได้
2) สามารถใช้โปรแกรม Keil Logic Analyzer โหมดดีบักได้
3) เข้าใจว่าการกำหนด Optimization level ให้กับคอมไพเลอร์ส่งผลต่อโปรแกรมอย่างไร

1. โครงสร้าง LED บนบอร์ด

บนบอร์ด Nucleo-F411RE มี User LED 1 ดวง เชื่อมต่อกับ GPIO พอร์ต A ขา 5 LED จะติดเมื่อป้อนลอจิก “1” (3.3 v) และจะดับเมื่อป้อนลอจิก “0” (0V) ดังรูปที่ 1



รูปที่ 1 การเชื่อมต่อ LED

2. เขียนโปรแกรมเพิ่มเติม

เปิด Project จาก Lab 0 แล้วแก้ไขโปรแกรม ดังนี้

- ประกาศตัวแปรโกลบอล num ดังรูปที่ 2
- แก้ไข while loop ดังรูปที่ 3
- **<u>เพิ่มฟังก์ชัน delay ลงในไฟล์ main.c ดังรูปที่ 4 พร้อมประกาศฟังก์ชัน Prototype ดังรูปที่ 5</u>**

การเขียนโปรแกรมเพิ่มเติมลงในไฟล์ main.c ควรเขียนให้อยู่ระหว่าง comment **/* USER CODE BEGIN x */** และ **/* USER CODE END x */** เพื่อป้องกันไม่ให้โปรแกรมที่เขียนเพิ่มนั้นโดนลบในกรณีที่สั่ง Generate code ทับ Project เดิม

```
/* Includes ------------------------
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------

/* USER CODE BEGIN PV */
/* Private variables ---------------
uint8_t num=0;
/* USER CODE END PV */
```

รูปที่ 2 ประกาศตัวแปร Global Variable

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
  if (num<=7)
    num++;
  else
    num = 0;

  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  delay(500);
}
/* USER CODE END 3 */
```

รูปที่ 3 แก้ไขโปรแกรมใน while loop

```
/* USER CODE BEGIN 4 */
void delay (uint32_t ms)
{
  volatile uint32_t i,j;

  for(i=0; i<=ms; i++)
    for(j=0; j<=6600; j++)
      ;

  return;

}
/* USER CODE END 4 */
```

รูปที่ 4 ฟังก์ชัน delay

```
/* USER CODE BEGIN PFP */
/* Private function prototypes -----
void delay (uint32_t);
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
```

รูปที่ 5 ประกาศฟังก์ชัน Prototype

3. อธิบายการทำงาน

โปรแกรมจะเริ่มต้นทำงานที่**ฟังก์ชัน main** โดยจะรันฟังก์ชันดังต่อไปนี้

- ฟังก์ชัน HAL_Init() เพื่อกำหนดค่าเริ่มต้นที่จำเป็นต่อการเริ่มการทำงานให้กับไมโครคอนโทรลเลอร์ โดยโค้ดของฟังก์ชันนี้จะอยู่ในไฟล์ stm32f4xx_hal.c
- ฟังก์ชัน SystemClock_Config() ทำงานต่อจากฟังก์ชัน HAL_Init() เพื่อตั้งค่าวงจรหารและคูณความถี่ภายในไมโครคอนโทรลเลอร์ให้ทำงานตามที่ตั้งค่าไว้จากโปรแกรม STM32CubeMX โดยรายละเอียดของชนิดตัวแปรแบบ Structure และโค้ดของฟังก์ชันที่เรียกใช้ภายในฟังก์ชัน SystemClock_Config() นั้นสามารถศึกษาเพิ่มเติมได้จากไฟล์ stm32f4xx_hal_rcc.h และ stm32f4xx_hal_rcc.c

- ฟังก์ชัน MX_GPIO_Init() ซึ่งมีรายละเอียดดังรูปที่ 6 เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นเพื่อกำหนดให้ขา PA5 ทำหน้าที่เป็นเอาต์พุตตามที่กำหนดไว้ในโปรแกรม ขา PA5 จะทำงานได้ต้องจ่ายสัญญาณนาฬิกาไปยังโมดูล GPIO พอร์ต A ด้วยฟังก์ชัน ฟังก์ชันนี้โดนเรียกใช้ในฟังก์ชัน main ไฟล์ main.c แต่ตัวฟังก์ชันอยู่ในไฟล์ gpio.c มีรายละเอียดการทำงาน ดังนี้

  __HAL_RCC_GPIOA_CLK_ENABLE();

- ตั้งค่าให้ขา PA5 มีระดับลอจิกเริ่มต้นเป็นลอจิก 0 ตามที่ได้กำหนดไว้ในโปรแกรม STM32CubeMX ด้วยคำสั่ง

  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

- จากนั้น enable ขาที่ต้องการใช้งานซึ่งได้แก่ขา 5 ของ Port A ผ่านตัวแปรแบบโครงสร้าง GPIO_InitStructure ด้วยคำสั่ง

  GPIO_InitStructure.Pin = GPIO_Pin_5;

  แล้วกำหนดให้ทั้งสองขาทำหน้าที่เป็นขาเอาต์พุตแบบ push pull ที่ความเร็วแบบ High ด้วยคำสั่ง

  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

  จากนั้นจึงทำให้การตั้งค่าเกิดผลด้วยการเรียกฟังก์ชัน

  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

```
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
```

รูปที่ 6 รายละเอียดของฟังก์ชัน MX_GPIO_Init

- ตัวแปร GPIO_InitStruct มีชนิดข้อมูลเป็น GPIO_InitTypeDef ซึ่งเป็นชนิดข้อมูลแบบโครงสร้าง มีรายละเอียดดังนี้

```
typedef struct
{
  uint32_t Pin;          //ระบุขาที่ต้องการตั้งค่า
  uint32_t Mode;         //ระบุโหมดการทำงานของขาที่ต้องการตั้งค่า
  uint32_t Pull;         //ระบุการทำงานแบบ Pull-Up หรือ Pull-Down
```

```
  uint32_t Speed;          //ระบุความเร็วเมื่อทำงานเป็นขาเอาต์พุต
}GPIO_InitTypeDef;
```

- GPIOA เป็น pointer ที่ถูกสร้างขึ้นด้วยมาโครในไฟล์ stm32f411xe.h

  #define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)

- GPIOA จึงมีสถานะเป็น pointer ที่ชี้ไปยังหน่วยความจำ ณ ตำแหน่งเริ่มต้นของพอร์ต A โดยมีชนิดของข้อมูลเป็น struct GPIO_TypeDef ซึ่งมีข้อมูลย่อยภายใน struct เป็นรีจิสเตอร์ทั้งหมดของพอร์ต A มีรายละเอียดดังนี้

```
typedef struct
{
  __IO uint32_t CRL;   //Control Register Low
  __IO uint32_t CRH;   //Control Register High
  __IO uint32_t IDR;   //Input Data Register
  __IO uint32_t ODR;   //Output Data Register
  __IO uint32_t BSRR;  //Bit Set/Reset Register
  __IO uint32_t BRR;   //Bit Reset Register
  __IO uint32_t LCKR;  //Configuration Lock Register
} GPIO_TypeDef;
```

- เมื่อเข้า Infinite Loop คำสั่งแรกจะเป็นการเปลี่ยนค่าตัวแปรโกลบอล num ให้มีค่าอยู่ในช่วง 0 – 7 ซึ่งตัวแปร num จะถูกใช้เพื่อสาธิตการใช้งาน Keil Logic Analyzer
- ฟังก์ชัน HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5) ใน while loop คือการกลับลอจิกของขา PA5 เช่น ถ้าแต่เดิมขา PA5 มี**ลอจิก 0** ภายหลัง execute คำสั่งนี้จะทำให้ขา PA5 จะมี**ลอจิก 1**
- ฟังก์ชัน HAL_GPIO_TogglePin และฟังก์ชันอื่นๆ ที่เกี่ยวข้องกับโมดูล GPIO สามารถศึกษาเพิ่มเติมได้จากไฟล์ stm32f4xx_hal_gpio.h และ stm32f4xx_hal_gpio.c หรือศึกษาจากเอกสารคู่มือจากไฟล์ UM1725 ซึ่งมีรายละเอียดของฟังก์ชันนี้อยู่ที่หน้า 406 ดังรูปที่ 7

HAL_GPIO_TogglePin

Function name	**void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)**
Function description	Toggles the specified GPIO pins.
Parameters	• **GPIOx:** Where x can be (A..K) to select the GPIO peripheral for STM32F429X device or x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices. • **GPIO_Pin:** Specifies the pins to be toggled.
Return values	• **None:**

รูปที่ 7 รายละเอียดของฟังก์ชัน `HAL_GPIO_TogglePin`

- **ฟังก์ชัน void delay (uint32_t ms)** เป็นฟังก์ชันหน่วงเวลาเพื่อหยุดการทำงานของไมโครโพรเซสเซอร์ชั่วคราวด้วยการไม่ให้ไป execute คำสั่งอื่น มีการทำงานเป็นการวนลูป 2 ลูปซ้อนกัน โดยลูปในเป็นการวนลูปเพื่อหน่วงเวลา 1 ms ดังนั้นการวนลูปนอกจึงเป็นการกำหนดว่าต้องการหน่วงเวลากี่ ms ซึ่งถูกกำหนดค่าผ่านตัวแปร ms และการกำหนด optimization level ตอนคอมไพล์โปรแกรม

4. การใช้งานโหมดดีบักและ Keil Logic Analyzer

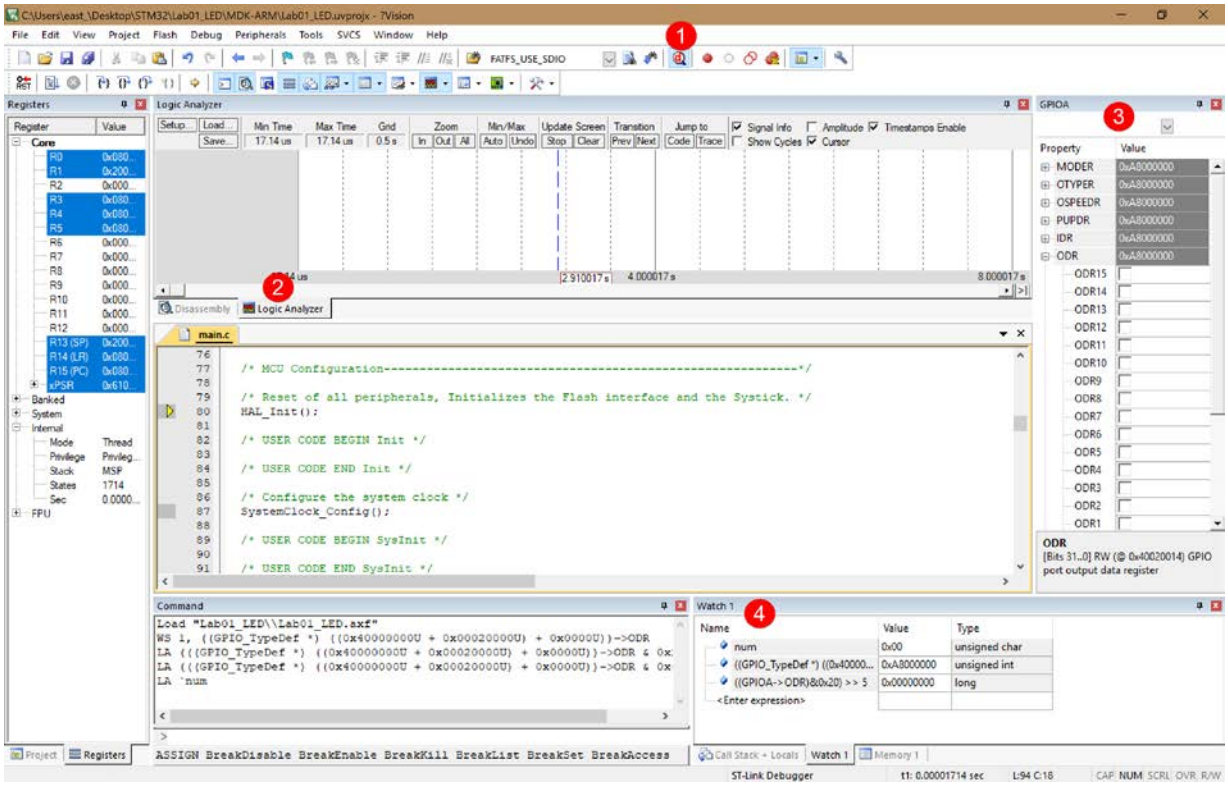
การพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์นั้นสามารถใช้งานในโหมดดีบักได้เช่นเดียวกับการพัฒนาโปรแกรมโดยทั่วไป เพียงแต่การประมวลผลข้อมูลเกิดขึ้นที่ไมโครคอนโทรลเลอร์ แล้วจึงนำผลลัพธ์จากการประมวลผลมาแสดงภายในโปรแกรมที่ใช้พัฒนาบนเครื่องคอมพิวเตอร์

โปรแกรมสำหรับพัฒนาไมโครคอนโทรลเลอร์หลายๆ โปรแกรมมีฟังก์ชันสำหรับการดีบักหลายอย่างด้วยกัน เช่น โปรแกรม Keil µVision สามารถตั้งค่า Breakpoint, อ่านค่าตัวแปร, อ่านค่าจากหน่วยความจำที่ตำแหน่งต่างๆ, รีจิสเตอร์ทั่วไป และแสดงค่าที่อ่านได้แบบ Timing Diagram บน Logic Analyzer เป็นต้น

Logic Analyzer เป็นเครื่องมือวัดสำหรับวัดสัญญาณดิจิตอล ทำหน้าที่คล้ายกับ Oscilloscope ที่ใช้วัดสัญญาณแอนาล็อก โดยโปรแกรม Keil มีฟังก์ชัน Logic Analyzer อย่างง่ายอยู่ด้วย สามารถใช้**ตรวจสอบค่าของตัวแปรโกลบอลขาสัญญาณหรือรีจิสเตอร์บางตัวได้** ช่วยให้สามารถหาจุดผิดพลาดในโปรแกรมได้ง่ายขึ้น

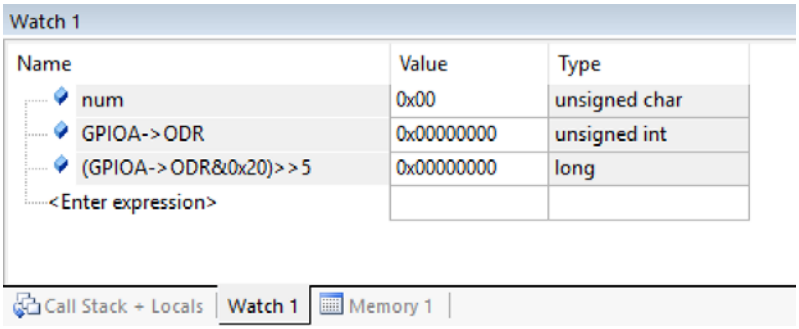
สำหรับการเข้าสู่โหมดดีบักเพื่ออ่านค่าตัวแปร num ซึ่งมีค่าอยู่ในช่วง 0 – 7 และสถานะลอจิกของขา PA5 สามารถทำได้ดังรูปที่ 8 โดยมีรายละเอียดดังนี้

1) กดปุ่มเพื่อเข้าสู่โหมดดีบัก
2) เรียกหน้าต่าง Logic Analyzer โดยไปที่เมนู View -> Analysis Windows -> Logic Analyzer
3) เรียกหน้าต่าง GPIOA โดยไปที่เมนู View -> System Viewer -> GPIO -> GPIOA หรับอ่านค่ารีจิสเตอร์ต่างๆ ที่อยู่ภายใน GPIOA
4) เรียกหน้าต่างดูค่าตัวแปร (Watch) โดยไปที่เมนู View -> Watch Windows -> Watch 1



รูปที่ 8 การปรับแต่งหน้าจอในโหมดดีบัก

สำหรับการอ่านค่าตัวแปร num และสถานะลอจิกจากขา PA5 **แบบเรียลไทม์** สามารถตั้งค่าได้ดังรูปที่ 9 โดยคลิกที่ <Enter expression> จากนั้นพิมพ์ num กด Enter แล้วทำซ้ำอีก 2 ครั้ง โดยพิมพ์ GPIOA->ODR และ (GPIOA->ODR&0x20)>>5 เพื่ออ่านค่ารีจิสเตอร์ ODR ของ GPIOA ทั้งหมดและอ่านค่าจากรีจิสเตอร์ ODR ของ GPIOA เฉพาะที่เกี่ยวข้องกับพิน 5 ตามลำดับ

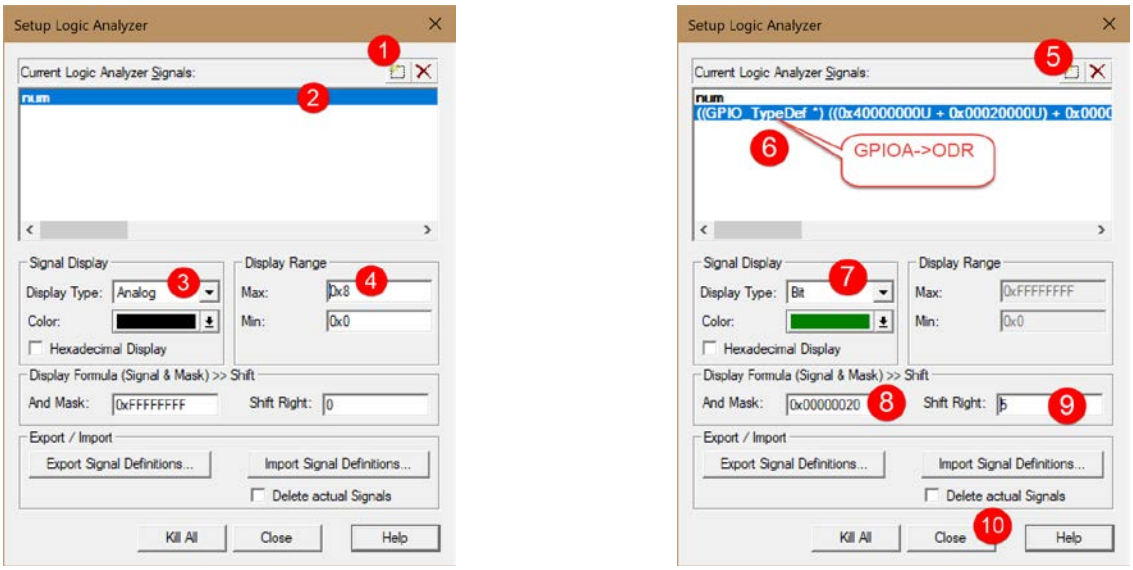


รูปที่ 9 การปรับแต่งหน้าจอ Watch 1 สำหรับการอ่านค่าตัวแปรแบบเรียลไทม์

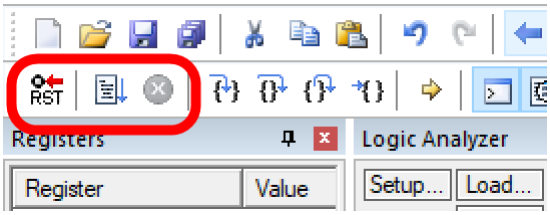
การตั้งค่า Logic Analyzer เพื่ออ่านค่าตัวแปร num และสถานะลอจิกที่ขา PA5 แล้วแสดงผลแบบ Timing Diagram ให้กดปุ่ม Setup ในหน้าต่าง Logic Analyzer แล้วตั้งค่า ดังรูปที่ 10 (a) และ (b)

รูปที่ 11 แสดงปุ่ม Reset ปุ่ม Run และปุ่ม Stop โดยทั้งสามปุ่มทำหน้าที่ดังนี้

- ปุ่ม Run ใช้สำหรับให้โปรแกรมเริ่มต้นการทำงาน หรือทำงานต่อจากจุดที่กดปุ่ม Stop เอาไว้
- ปุ่ม Stop ใช้สำหรับหยุดการรันโปรแกรมชั่วคราว
- ปุ่ม Reset ใช้สำหรับให้โปรแกรมกลับไปเริ่มต้นทำงานที่คำสั่งแรกสุด



รูปที่ 10 การตั้งค่า Logic Analyzer



รูปที่ 11 ปุ่ม Reset ปุ่ม Run และปุ่ม Stop

การทดลอง

1. ให้ตรวจสอบการหน่วงเวลาจากการเรียกฟังก์ชัน delay(500) ว่าหน่วงเวลาเป็นระยะเวลา 500 ms หรือไม่ เมื่อใช้ **optimization level 0** โดยใช้ Logic Analyzer ของโปรแกรม Keil เป็นเครื่องมือวัด **ถ้าไม่ใช่ให้เปลี่ยนเงื่อนไขของลูปข้างใน (inner loop)** ให้สามารถหน่วงเวลาได้ 500 ms แล้วบันทึกผล

จากการเรียกใช้ฟังก์ชัน delay(500) ปรากฏว่า หน่วงเวลาที่ประมาณ 495 ms ถ้าแก้ไขค่า j <= 6600 ให้เป็น j <= 6660 เพื่อให้ฟังก์ชัน delay(500) จะหน่วงเวลาได้ใกล้เคียง 500 ms ที่สุด

2. **เปลี่ยน optimization level ให้เป็น level 3** แล้วตรวจสอบดูว่าผลของการเรียกฟังก์ชัน delay(500) เปลี่ยนแปลงหรือไม่ อย่างไร เพราะสาเหตุใด ถ้ามีการเปลี่ยนแปลงแล้วเงื่อนไขของลูปในควรเปลี่ยนแปลงอย่างไร เพื่อให้ผลลัพธ์ของการเรียกฟังก์ชันเหมือนกับการทดลองที่ 1

จากการเปลี่ยน optimization level ให้เป็น level 3 ปรากฏว่า การหน่วงเวลาจากประมาณ 500 ms ลดลงเหลือประมาณ 400 ms เนื่องจากว่า optimization level 3 สามารถ generated code ได้เร็วกว่า level 0 ถ้าแก้ไขค่า j <= 6660 ให้เป็น j <= 8325 จะได้ผลลัพท์ หน่วงเวลาที่ประมาณ 500 ms

3. ให้**เปลี่ยน optimization level กลับมาเป็น level 0** พร้อมกับใช้เงื่อนไขของลูปข้างในตามผลการทดลองที่ 1 จากนั้นจงเขียนโปรแกรมเพื่อแสดงระดับสัญญาณของวงจรนับขึ้น 3 บิต ใน Logic Analyzer จากโปรแกรม Keil โดยใช้ขา PA7 PA6 และ PA5 (LSB) โดยให้หน่วงเวลาที่ 300 ms

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
const unsigned int count_c = 1; // 1 = count up (0->7) , 0 = count down (7->0)
int8_t count = count_c;
while (1)
{
	/* USER CODE END WHILE */
	/* USER CODE BEGIN 3 */
	/* HAL_Delay(300); // For standard delay */
	delay(300);
	HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
	if(count % 2 == 0) HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_6);
	if(count % 4 == 0) HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_7);
	if((++count) == (8 + count_c)) count = count_c;
}
/* USER CODE END 3 */
```

ใบตรวจการทดลองที่ 1

KU CSC Embedded System

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนิสิต _____ ชื่อ-นามสกุล _____

2. รหัสนิสิต _____ ชื่อ-นามสกุล _____

3. รหัสนิสิต _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 1&2 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

การทดลองข้อ 3 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. GPIOA เป็นมาโคร pointer เพื่อชี้ไปยังตำแหน่งเริ่มต้นในหน่วยความจำของโมดูล GPIO พอร์ต A จงหาตำแหน่งเริ่มต้นนี้ว่าอยู่ที่ตำแหน่งที่เท่าไร (ตอบเป็นตัวเลข) และถูกจัดเป็นส่วนไหนใน memory space โดยศึกษาจากไฟล์ stm32f411xe.h

   PERIPH_BASE = 0x40000000U

   AHB1PERIPH_BASE = (PERIPH_BASE + 0x00020000U)

   GPIOA_BASE = (AHB1PERIPH_BASE + 0x000U)

   ดังนั้น GPIOA มีตำแหน่งเริ่มต้นที่ 0x40020000U

   และจัดอยู่ในพื้นที่ Peripheral (อุปกรณ์ต่อพ่วง) ใช้ bus AHB1