240-302 Computer Engineering Hardware Lab II ภาคเรียนที่ 2 ปีการศึกษา 2562

Lab 2HB06 RTOS for Arduino

ผู้สอน ผู้ช่วยศาสตราจารย์ ดร. ปัญญยศ ไชยกาฬ

1. วัตถุประสงค์

เพื่อให้นักศึกษาสามารถเข้าใจหลักการทำงานของระบบปฏิบัติการ RTOS และสามารถเขียนโปรแกรมบนบอร์ด Arduino เพื่อทำงานบนระบบปฏิบัติการ RTOS ได้

2. กำหนดส่งงานและวิธีการส่งงาน

- การตรวจ Checkpoint

การทดลองนี้มี 1 Checkpoint เมื่อทำการทดลองใน Checkpoint ให้เสร็จและเรียกผู้ช่วยคุมแลบตรวจในวันลงปฏิบัติการ และให้ตัดลายเซ็นของผู้คุมแล็บหรืออาจารย์แปะลงในสมุด Log book ด้วย

- การส่งสรุปผลการทดลองและคำถามท้ายการทดลอง

ให้บันทึกผลการทดลอง เขียนสรุปผลการทดลองและตอบคำถามท้ายการทดลองลงในสมุด Log book และให้ส่งสมุด Log book ในตู้ส่งงานหน้าธุรการภาควิชาฯ ภายในวันเวลาที่กำหนด

3. การให้สัดส่วนคะแนน

คะแนนของการทดลองนี้แบ่งออกเป็น 4 ส่วน ได้แก่

- Checkpoint	30 %
- ตอบคำถามท้ายการทดลอง	10 %
- สรุปผลการทดลอง	10 %
- สอบภาคทฤษฎี	50 %

4. แนะนำ FreeRTOS

FreeRTOS (Free Real-Time Operating System) เป็นระบบปฏิบัติการตัวหนึ่ง ซึ่งออกแบบมาสนับสนุนการทำงานแบบมัลติทาสกิ้ง โดยมีวัตถุประสงค์เพื่อใช้กับงานที่ต้องการการตอบสนองต่ออินพุตภายในระยะเวลาไม่เกินค่าที่กำหนด ระบบปฏิบัติการตัวนี้ถูกพัฒนาโดยบริษัท Real Time Engineer นิยมใช้งานในระบบสมองกลฝังตัว (Embedded System) โดยสามารถรองรับตัวประมวลผลได้หลากหลายสถาปัตยกรรม ยกตัวอย่างเช่น AVR, MCS-51, ARM, PowerPC, ESP-32, MicroBlaze และ RISC-V เป็นต้น

4.1 Task ใน FreeRTOS

การทำงานของระบบที่ใช้ FreeRTOS จะต้องมีการสร้าง Task ขึ้นมาอย่างน้อย 1 ตัว และตัว Scheduler จะถูกเรียกขึ้นมาทำงานทุกครั้งที่มีการสิ้นสุดของแต่ละ time slice โดยที่ตัวประมวลผลจะถูกขัดจังหวะทุก ๆ ค่าเวลาช่วงหนึ่งซึ่งเรียกว่า tick interrupt ซึ่งโดยทั่วไปมักตั้งให้มีค่าเท่ากับ 10 มิลลิวินาที ซึ่งหมายความว่า ทุก ๆ 10 มิลลิวินาทีจะเกิดการขัดจังหวะตัวประมวลผลหนึ่งครั้ง และ Scheduler จะเข้ามาทำหน้าที่ตัดสินใจว่าจะอนุญาตให้ Task ได้รับการบริการจากตัวประมวลผล ในการพัฒนาระบบ ผู้เขียนโปรแกรมจะสร้าง Task ขึ้นมาในรูปแบบของฟังก์ชันภาษาซี โดยมีโครงสร้าง

ดังแสดงในรูปที่ 1 ในข้อกำหนดของ FreeRTOS นั้นฟังก์ชันที่ผู้เขียนโปรแกรมจะสามารถนำมาประกาศเป็น Task ได้จะต้องเป็นฟังก์ชันที่มีการทำงานแบบไม่รู้จบ และไม่มีการส่งค่ากลับมายังผู้เรียก หรือไม่มีการรีเทอร์นค่า

```
 1  void ATaskFunction( void *pvParameters )
 2  {
 3
 4    int32_t local_Variable_Example = 0;
 5    for( ;; )
 6    {
 7       ...   //write your code here
 8       ...
 9    }
10  }
```

รูปที่ 1 โครงสร้างของฟังก์ชันที่จะถูกสร้างเป็น Task ใน FreeRTOS

ผู้ใช้สามารถเปลี่ยนค่า Tick interrupt ได้โดยการกำหนดค่า TICK_RATE_HZ ซึ่งอยู่ในไฟล์ FreeRTOSConfig.h ยกตัวอย่างเช่น หากเราตั้งให้ค่า TICK_RATE_HZ เท่ากับ 100 จะหมายความว่าค่า time slice มีค่าเท่ากับ 10 มิลลิวินาที อย่างไรก็ดี ในการทดลองนี้จะใช้ค่าโดยปริยายที่โปรแกรมตั้งเอาไว้แล้วโดยที่ผู้ทดลองไม่ต้องเข้าไปยุ่งในส่วนนี้

4.2 การสร้าง Task

การสร้าง Task สามารถทำได้ในฟังก์ชันหลัก (main) ด้วยการเรียกใช้ฟังก์ชัน xTaskCreate ซึ่งมีค่าพารามิเตอร์ดังต่อไปนี้

```
BaseType_t  xTaskCreate( TaskFunction_t pvTaskCode,
                         const char * const pcName,
                         uint16_t usStackDepth,
                         void *pvParameters,
                         UBaseType_t uxPriority,
                         TaskHandle_t *pxCreatedTask );
```

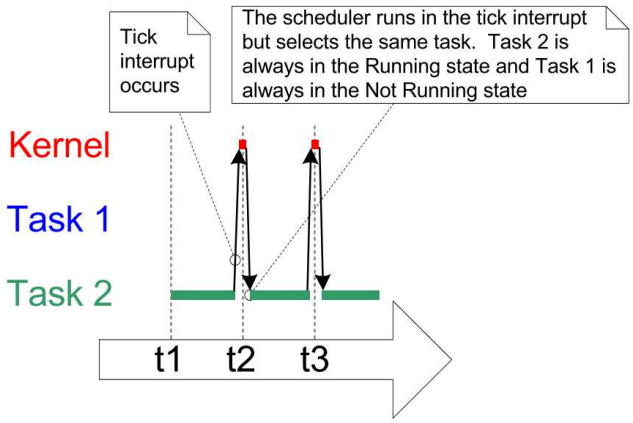
- พารามิเตอร์ pvTaskCode คือ ชื่อของฟังก์ชันที่จะถูกเรียกขึ้นมาสร้างเป็น Task
- พารามิเตอร์ pcName ใช้ในการดีบักโปรแกรม (ผู้ทดลองไม่ต้องทำอะไรในส่วนนี้ก็ได้)
- พารามิเตอร์ usStackDepth คือ ขนาดของสแต็กที่จองให้กับ Task
- พารามิเตอร์ *pvParameter คือ ค่าพารามิเตอร์ที่ผู้เขียนโปรแกรมต้องการส่งให้ Task นั้น ๆ รับเข้ามาใช้งาน
- พารามิเตอร์ uxPriority คือ ค่าลำดับความสำคัญ (Priority) ของ Task
- พารามิเตอร์ *pxCreatedTask คือ Task handle (ผู้ทดลองไม่ต้องทำอะไรในส่วนนี้ ให้ตั้งค่าให้เป็น NULL)

รูปที่ 2 แสดงตัวอย่างการสร้าง Task ขึ้นมา 2 ตัว ภายในฟังก์ชันหลัก (main) โดยมีการกำหนดให้ Task 1 และ Task 2 มีค่าลำดับความสำคัญเท่ากับ 1 และ 2 ตามลำดับ ซึ่งจะเห็นว่าในกรณีนี้ Task 2 มีค่าลำดับความสำคัญสูงกว่า Task 1

```
/* Define the strings that will be passed in as the task parameters. These are
defined const and not on the stack to ensure they remain valid when the tasks are
executing. */
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\r\n";
int main( void )
{
	/* Create the first task at priority 1. The priority is the second to last
	parameter. */
	xTaskCreate( vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL );
	/* Create the second task at priority 2, which is higher than a priority of 1.
	The priority is the second to last parameter. */
	xTaskCreate( vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 2, NULL );
	/* Start the scheduler so the tasks start executing. */
	vTaskStartScheduler();
	/* Will not reach here. */
	return 0;
}
```

รูปที่ 2 ตัวอย่างการสร้าง Task สองตัวซึ่งมี priority ไม่เท่ากัน

จากโค้ดโปรแกรมในรูปที่ 2 จะเห็นว่า Task 2 มีค่าลำดับความสำคัญสูงกว่า Task1 ดังนั้น Scheduler จะตัดสินใจให้ Task2 ทำงานเสมอ นั่นหมายความว่าในกรณีนี้ Task ที่ได้เข้าสู่ Running state ก็จะมีเพียง Task2 เท่านั้น ส่วน Task1 จะไม่ถูกเลือกโดย Scheduler ให้เข้าสู่ Running state ได้เลย หรือเรียกอีกอย่างหนึ่งว่า Task1 อยู่ในภาวะอดอยาก (Starved) ดังรูปที่ 3



รูปที่ 3 รูปแบบการเอกซีคิวต์เมื่อ Task หนึ่งมีค่าลำดับความสำคัญสูงกว่าอีก Task หนึ่ง

4.3 สถานะของ Task ใน FreeRTOS

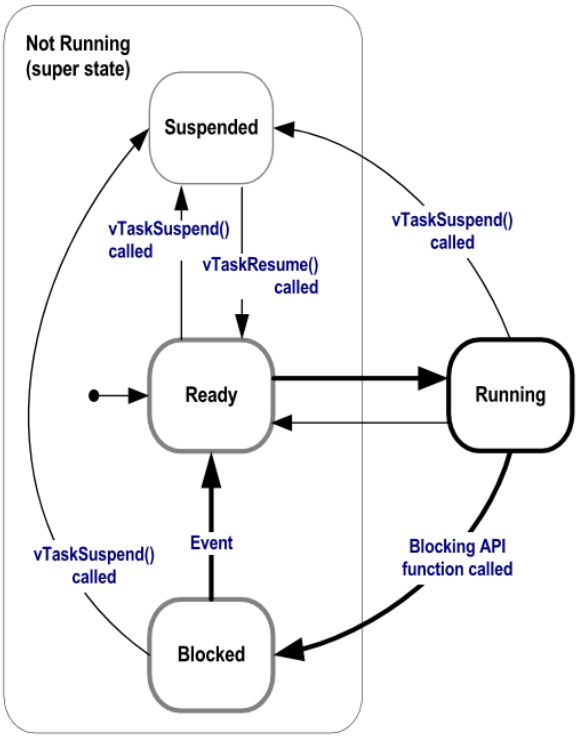
จากรูปที่ 4 Task แบ่งออกเป็นสองสถานะ ได้แก่ Not Running state และ Running state โดยที่สถานะ Not Running state จะแบ่งออกเป็น 3 สถานะย่อย ได้แก่ Suspended state, Ready State, และ Blocked state

- ภาวะ Ready state คือ ภาวะที่ Task พร้อมที่จะถูกเลือกให้เข้าสู่ภาวะ Running state
- ภาวะ Suspended state คือ ภาวะที่ Task ถูกหยุดการทำงานชั่วคราว ซึ่งเกิดขึ้นเมื่อมีการเรียกใช้ฟังก์ชัน vTaskSuspend() เมื่อมีการเรียกใช้ฟังก์ชัน vTaskResume() จะส่งผลให้ Task นั้นกลับเข้าสู่ภาวะ Ready state อีกครั้ง

- ภาวะ Blocked คือ ภาวะที่ Task จะต้องมีการรอการทำงาน เช่นจาก อินพุต หรือรอการเกิดของเหตุการณ์บางอย่าง เมื่อเหตุการณ์หรือ อินพุตนั้น ๆ เกิดขึ้นแล้ว Task นั้นจะกลับมาอยู่ในสถานะ Ready state อีกครั้ง

การเลือก Task ขึ้นมาทำงานของ Scheduler จะมีกลไกการทำงานดังนี้

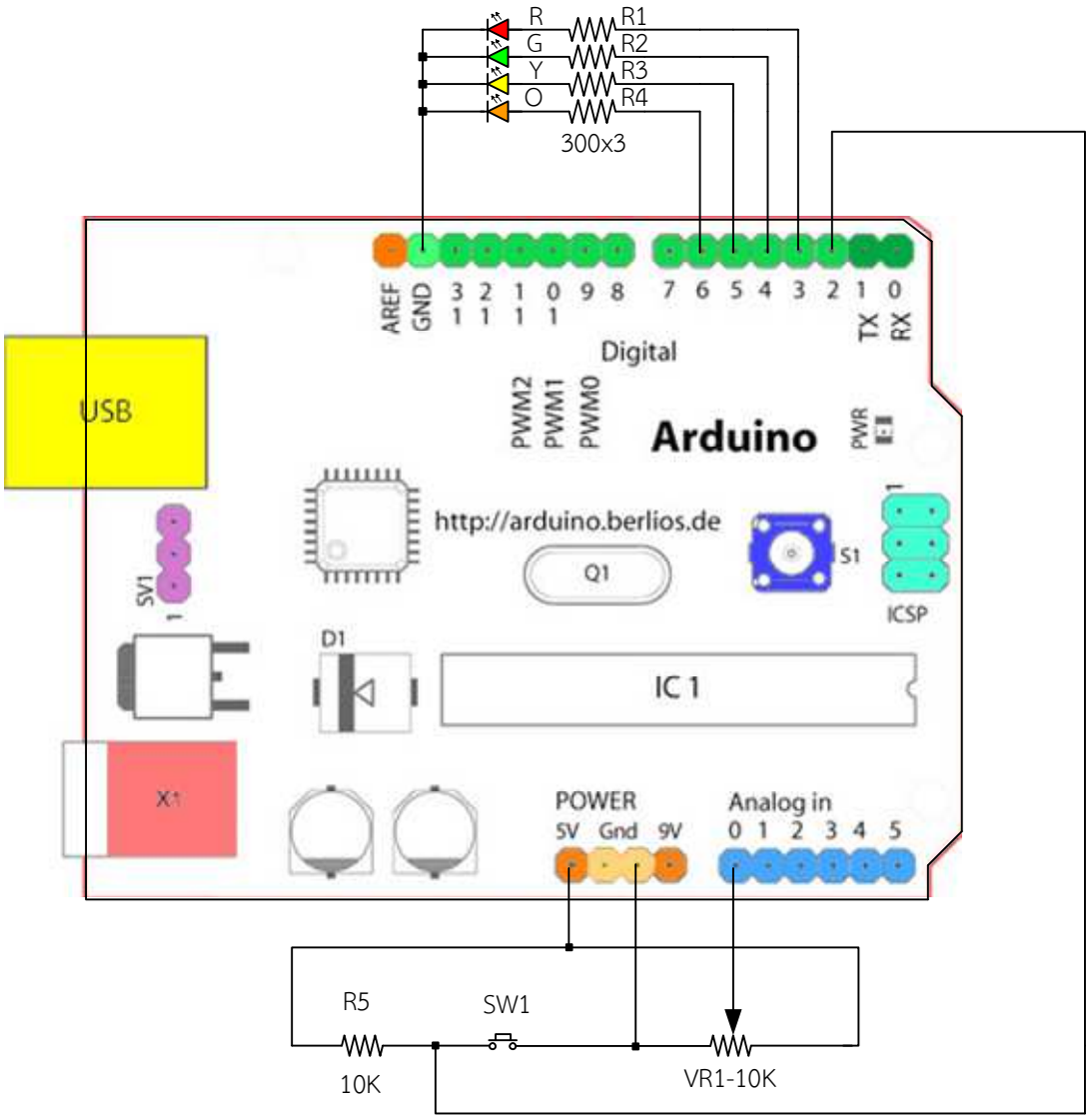
- เลือก Task ที่อยู่ในสถานะ Ready state ที่มี Priority สูงสุดขึ้นมาทำงาน
- หากมี Task ที่อยู่ในสถานะ Ready state หลายตัวแต่มีค่า Priority เท่ากันจะใช้วิธีการวนเลือก (Round Robin)
- Task ที่อยู่ในภาวะ Suspend และ Blocked state จะไม่ถูกเลือกโดย Scheduler แม้จะมีค่า Priority สูงกว่า Task ที่อยู่ในภาวะ Ready state



รูปที่ 4 สถานะทั้งสี่รูปแบบของ Task ใน FreeRTOS

5. วัสดุและอุปกรณ์ที่ใช้ในการทดลอง

- บอร์ด Arduino Uno	1	บอร์ด
- สายดาวน์โหลดโปรแกรม	1	เส้น
- ตัวต้านทาน 300 โอห์ม	4	ตัว
- ตัวต้านทาน 1 กิโลโอห์ม	1	ตัว
- ตัวต้านทานปรับค่าได้ 10 กิโลโอห์ม	1	ตัว
- สวิตช์กดติดปล่อยดับ	1	ตัว
- สายไฟสำหรับเชื่อมต่อวงจร	20	เส้น



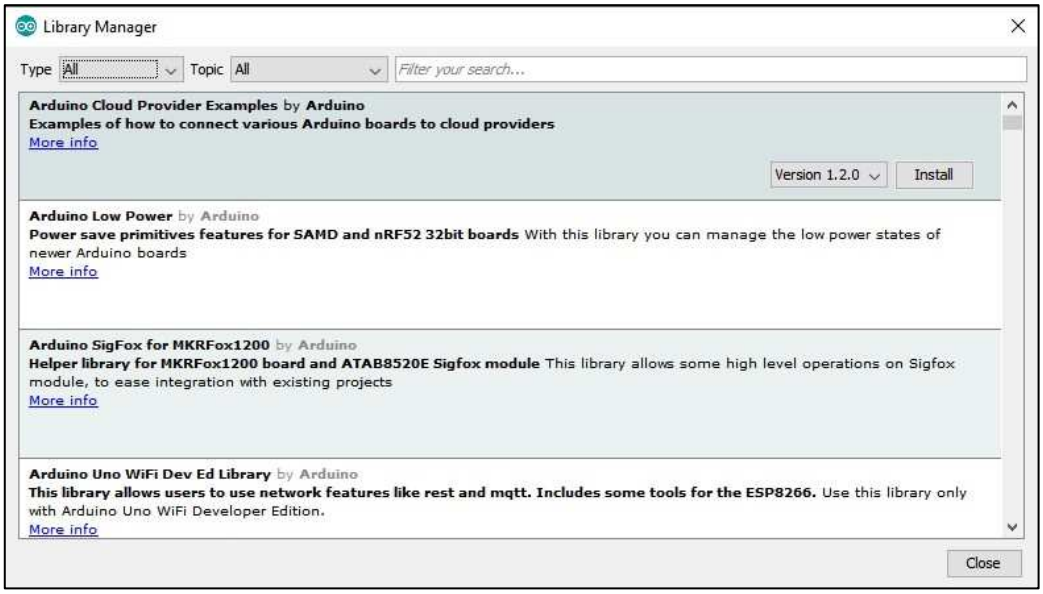
รูปที่ 5 การต่อวงจรทดลองระหว่างบอร์ด Arduino กับตัวต้านทานปรับค่าได้ สวิตช์ และแอลดีดี

5. วิธีการทดลอง

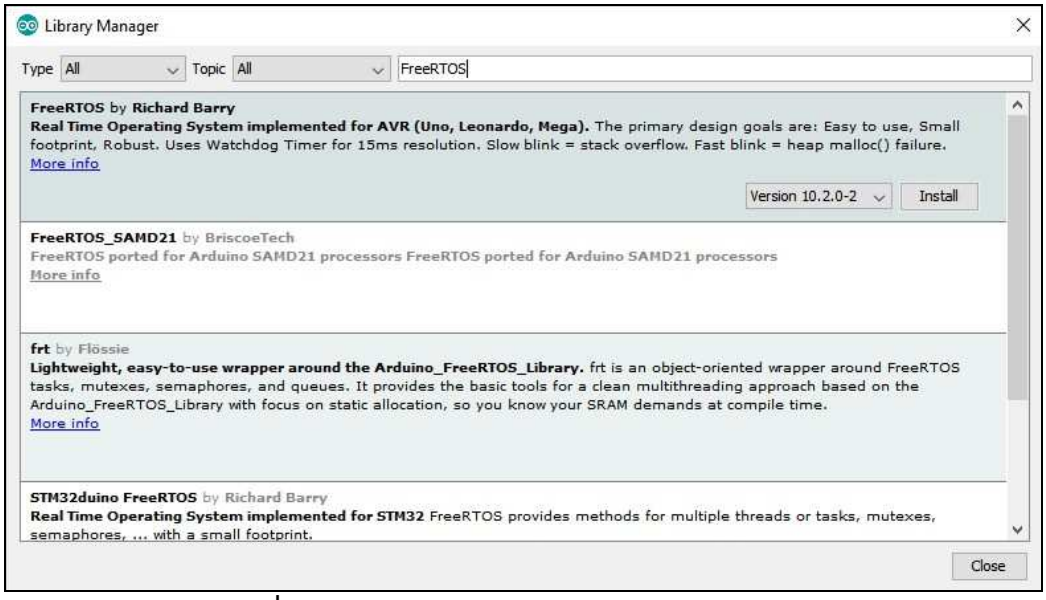
การทดลองในแล็บนี้ประกอบไปด้วย 4 การทดลองย่อย ให้นักศึกษาต่อวงจรตามรูปที่ 5 ระหว่างบอร์ด Arduino Uno กับสวิตช์และแอลอีดี ซึ่งจะเห็นว่าขา A0 ซึ่งรับสัญญาณแอนะล็อกจากตัวต้านทานปรับค่าได้ (VR1) ซึ่งต่อกับแหล่งจ่าย 5 โวลต์ในลักษณะวงจรแบ่งแรงดันไฟฟ้า ส่วนสวิตช์กดติดปล่อยดับ (SW1) ต่ออยู่กับขา D2 ของบอร์ด Arduino และมีแอลอีดีจำนวนสี่ตัว ต่ออยู่กับขา D3-D6

5.1 การทดลองที่ 1

เปิดโปรแกรม Arduino จากนั้นไปที่เมนู Sketch -> Include Library -> Manage Libraries จะปรากฏหน้าต่าง Library Manager ขึ้นมาดังรูปที่ 6 ในช่องว่างด้านบนขวาของ Topic ให้พิมพ์คำว่า FreeRTOS จะปรากฏคลังโปรแกรมที่ใกล้เคียงให้เลือก ดังรูปที่ 7



รูปที่ 6 หน้าต่าง Library Manager ของโปรแกรม Arduino



รูปที่ 7 ตัวอย่างผลการค้นหา FreeRTOS ใน Library Manager

ให้กดปุ่ม Install เพื่อติดตั้งคลังโปรแกรมของ Richard Barry จากนั้นเขียนโปรแกรมในสภาพแวดล้อม Arduino โดยใช้โค้ดโปรแกรมซึ่งแสดงในรูปที่ 8 ทำการคอมไพล์โปรแกรมและอัพโหลดโปรแกรมลงบอร์ด Arduino สังเกตผลการทดลองที่แอลดีดีทั้งสองดวงซึ่งต่ออยู่กับบอร์ด Arduino

```
1   #include <Arduino_FreeRTOS.h>
2   #define R_LED  3
3   #define G_LED  4
4
5   void setup() {
6     xTaskCreate(Display_R_LED, "Red LED Task", 128, NULL, 1, NULL);
7     xTaskCreate(Display_G_LED, "Green LED Task", 128, NULL, 1, NULL);
8     vTaskStartScheduler();
9   }
10
11  void Display_R_LED(void *pvParameters)
12  {
13    pinMode(R_LED, OUTPUT);
14    while (1)
15    {
16      vTaskDelay(pdMS_TO_TICKS(500));                  //หน่วงเวลา 500 mS
17      digitalWrite(R_LED, digitalRead(R_LED) ^ 1);
18    }
19  }
20
21  void Display_G_LED(void *pvParameters)
22  {
23    pinMode(G_LED, OUTPUT);
24    while (1)
25    {
26      vTaskDelay(pdMS_TO_TICKS(1000));                 //หน่วงเวลา 1000 mS
27      digitalWrite(G_LED, digitalRead(G_LED) ^ 1);
28    }
29  }
30
31  void loop()
32  {
33  }
```

รูปที่ 8 ตัวอย่างโปรแกรมบน RTOS สำหรับสั่งกระพริบแอลอีดี 2 ดวง

จากรูปที่ 8 จะเห็นว่ามีการสร้าง Task จำนวน 2 ตัว สำหรับควบคุมการกระพริบแอลอีดีจำนวนสองดวง โดยที่ Task Display_R_LED ทำหน้าที่กลับค่าตรรกะของพอร์ตซึ่งต่ออยู่กับแอลอีดีสีแดงให้มีค่าตรงกันข้ามทุก ๆ 500 มิลลิวินาที การหน่วงเวลาทำได้โดยใช้ฟังก์ชัน vTaskDelay ซึ่งทำหน้าที่สั่งให้ Task ย้ายไปอยู่ในสถานะ Blocked เป็นค่าเวลาที่ระบุในค่าอากิวเมนต์ที่ได้รับ ให้สังเกตว่าค่าที่ส่งให้กับ vTaskDelay จะต้องใช้ฟังก์ชัน pdMS_TO_TICK เพื่อแปลงค่าจากหน่วยมิลลิวินาทีอีกต่อหนึ่ง

5.2 การทดลองที่ 2

ให้สร้างโปรเจกต์ใหม่ในโปรแกรม Arduino จากนั้นเขียนโปรแกรมที่แสดงในรูปที่ 9 จากนั้นจึงทำการคอมไพล์โปรแกรมและอัพโหลดโปรแกรมลงบอร์ด Arduino สังเกตผลการทดลองที่แอลดีดีทั้งสองดวงซึ่งต่ออยู่กับบอร์ด Arduino ทำการกดปุ่มที่สวิตช์กดติดปล่อยดับ สังเกตการเปลี่ยนแปลงของแอลอีดีว่าตัวใดมีการเปลี่ยนแปลง และบันทึกผลการทดลองลงใน logbook

เปิดโปรแกรม Serial Monitor ซึ่งอยู่ในเมนู Tools ของโปรแกรม Arduino จากนั้นทำการหมุนปรับค่าความต้านทานของ VR1 และสังเกตการเปลี่ยนแปลงในโปรแกรม Serial Monitor บันทึกผลการทดลองลงใน logbook

```
#include <Arduino_FreeRTOS.h>
#define R_LED 3
#define G_LED 4
#define PUSH_SW 2

int v_delay;

void setup() {
  Serial.begin(9600);           //  setup serial
  xTaskCreate(Read_Poten, "Read Potentiometer", 128, NULL, 1, NULL);
  xTaskCreate(Read_Switch, "Read push button switch", 128, NULL, 1, NULL);
  xTaskCreate(Display_R_LED, "Red LED Task", 128, NULL, 1, NULL);
  xTaskCreate(Display_G_LED, "Green LED Task", 128, NULL, 1, NULL);
  vTaskStartScheduler();
}

void Display_R_LED(void *pvParameters)
{
  pinMode(R_LED, OUTPUT);
  while (1)
  {
    vTaskDelay(pdMS_TO_TICKS(500));
    digitalWrite(R_LED, digitalRead(R_LED) ^ 1);
  }
}

void Display_G_LED(void *pvParameters)
{
  pinMode(G_LED, OUTPUT);
  while (1)
  {
    vTaskDelay(v_delay);
    digitalWrite(G_LED, digitalRead(G_LED) ^ 1);
  }
}

void Read_Poten(void *pvParameters)
{
  while (1)
  {
    int val = analogRead(A0);  // read the input pin
    Serial.println(val);
  }
}

void Read_Switch(void *pvParameters)
{
  pinMode(PUSH_SW, INPUT);
  const TickType_t delay_0300ms = pdMS_TO_TICKS(300);
  const TickType_t delay_1000ms = pdMS_TO_TICKS(1000);
  while (1)
  {
    int sw_status = digitalRead(PUSH_SW);
    if (sw_status==LOW)
      v_delay = delay_0300ms;
    else
      v_delay = delay_1000ms;
  }
}

void loop() {}
```

รูปที่ 9 ตัวอย่างโปรแกรมบน RTOS สำหรับสั่งกระพริบแอลอีดี 2 ดวง อ่านค่าจากสวิตช์และ VR1

5.3 การทดลองที่ 3

การทดให้สร้างโปรเจกต์ใหม่ในโปรแกรม Arduino จากนั้นเขียนโปรแกรมที่แสดงในรูปที่ 10 จากนั้นจึงทำการคอมไพล์โปรแกรมและอัพโหลดโปรแกรมลงบอร์ด Arduino สังเกตผลการทดลองที่แอลดีดีทั้งสามดวงซึ่งต่ออยู่กับบอร์ด Arduino ทำการกดปุ่มที่สวิตช์กดติดปล่อยดับ สังเกตการเปลี่ยนแปลงของแอลอีดีว่าตัวใดมีการเปลี่ยนแปลง และบันทึกผลการทดลองลงใน logbook

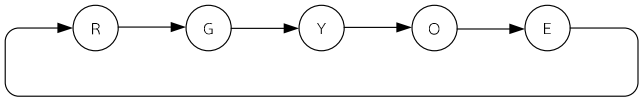
```
#include <Arduino_FreeRTOS.h>
#define R_LED 3
#define G_LED 4
#define Y_LED 5
#define PUSH_SW 2
int display=0; //if display=1 then turn on only Red LED
               //if display=2 then turn on only Green LED
               //if display=3 then turn on only Yellow LED

void setup() {
  xTaskCreate(Read_Switch, "Read push button switch", 128, NULL, 1, NULL);
  xTaskCreate(Display_R_LED, "Red LED Task", 128, NULL, 1, NULL);
  xTaskCreate(Display_G_LED, "Green LED Task", 128, NULL, 1, NULL);
  xTaskCreate(Display_Y_LED, "Yellow LED Task", 128, NULL, 1, NULL);
  vTaskStartScheduler();
}

void Display_R_LED(void *pvParameters)
{
  pinMode(R_LED, OUTPUT);
  while (1)  {
    if (display ==1)
      digitalWrite(R_LED, HIGH);
    else
      digitalWrite(R_LED, LOW);
  }
}

void Display_G_LED(void *pvParameters)
{
  pinMode(G_LED, OUTPUT);
  while (1)  {
    if (display ==2)
      digitalWrite(G_LED, HIGH);
    else
      digitalWrite(G_LED, LOW);
  }
}


void Display_Y_LED(void *pvParameters)
{
  pinMode(Y_LED, OUTPUT);
  while (1)  {
    if (display ==3)
      digitalWrite(Y_LED, HIGH);
    else
      digitalWrite(Y_LED, LOW);
  }
}

void Read_Switch(void *pvParameters)
{
  pinMode(PUSH_SW, INPUT);
  while (1)  {
    int sw_status = digitalRead(PUSH_SW);
    if (sw_status==LOW)
    {
      vTaskDelay(pdMS_TO_TICKS(10));
      sw_status = digitalRead(PUSH_SW);
      if (sw_status==LOW)
      {
        while(sw_status==LOW)
        {
          sw_status = digitalRead(PUSH_SW);
        }
        display++;
        if (display>3)
          display=0;
      }
    }
  }
}
void loop() {}
```

รูปที่ 10 ตัวอย่างโปรแกรมรับค่าจากดิปสวิตช์เพื่อควบคุมการทำงานของแอลอีดีสามตัว

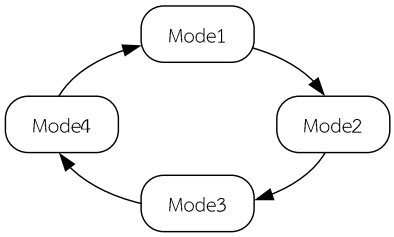
5.4 การทดลองที่ 4 (มี checkpoint)

จงเขียนโปรแกรมบน RTOS เพื่อให้ระบบมีคุณสมบัติดังนี้

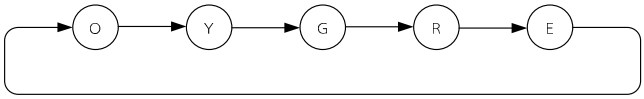
- เมื่อระบบเริ่มทำงาน แอลอีดีจะแสดงผลใน Mode1 ดังรูปที่ 11 โดยกำหนดให้ R, G, Y, O คือภาวะที่แอลอีดีสีแดง(Red) สีเขียว(Green) สีเหลือง(Yellow) และ สีส้ม (Orange) ติดตามลำดับ ส่วน E หมายถึงภาวะที่ไม่มีแอลอีดีดวงใดติดเลย
- เมื่อมีการกดสวิตช์ SW1 จำนวน 1 ครั้ง ให้ระบบเปลี่ยนมาแสดงผลใน Mode2 และเมื่อมีการกดสวิตช์ครั้งต่อไปก็ให้มีการเปลี่ยนโหมดการแสดงผลไปเรื่อย ๆ ตามสเตทไดอะแกรมในรูปที่ 12



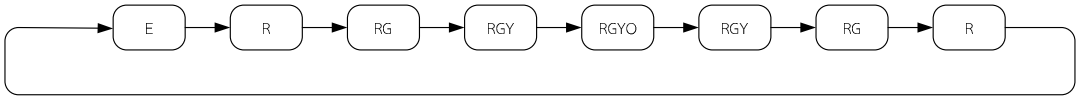
รูปที่ 11 การแสดงผลใน Mode1



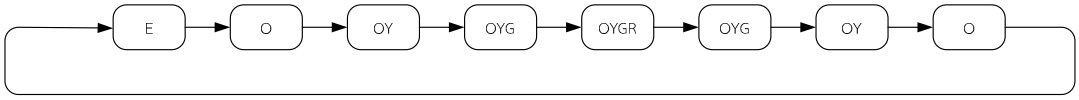
รูปที่ 12 การเปลี่ยนโหมดการแสดงผลเมื่อมีการกดสวิตช์ SW1 ในแต่ละครั้ง



รูปที่ 13 การแสดงผลใน Mode2



รูปที่ 14 การแสดงผลใน Mode3



รูปที่ 15 การแสดงผลใน Mode4

จากรูปที่ 14 และ 15 สภาวะการแสดงผล R หมายถึงสภาวะที่แอลอีดีสีแดงติดเพียงตัวเดียว สภาวะ RG หมายถึงสภาวะที่แอลอีดีสีแดงและสีเขียวติดพร้อมกัน สภาวะ RGY หมายถึงสภาวะที่แอลอีดีสีแดง สีเขียว และสีเหลืองติดพร้อมกัน

- ในสภาวะเริ่มต้น ให้การเปลี่ยนสภาวะการแสดงผลของแอลอีดีเกิดทุก ๆ 500 มิลลิวินาที (ยกตัวอย่างเช่น หากแสดงผลที่ Mode2 จะมีการสั่งให้แอลอีดีสีส้มสว่าง 500 มิลลิวินาที จากนั้นก็จะเปลี่ยนไปสั่งให้แอลอีดีสีเหลืองสว่าง 500 มิลลิวินาที และลำดับถัดไปจึงสั่งให้แอลอีดีสีเขียวสว่างเป็นเวลา 500 มิลลิวินาที และเป็นเช่นนี้ไปเรื่อย ๆ)

- กำหนดให้ผู้ใช้สามารถปรับความเร็วในการเปลี่ยนสถานะของแอลอีดีได้ด้วยการหมุนปรับค่าตัวต้านทาน VR1 โดยสามารถปรับสเต็ปของความเร็วได้ไม่น้อยกว่า 8 ระดับ โดยเมื่อหมุนค่า VR1 ทวนเข็มนาฬิกาไปจนสุดจะส่งผลให้ค่าหน่วงเวลาของการเปลี่ยนสถานะการแสดงผลแอลอีดีเท่ากับ 1.5 วินาที แต่หากหมุนปรับค่า VR1 ตามเข็มนาฬิกาจะส่งผลให้ค่าหน่วงเวลาลดลง และเมื่อหมุนปรับค่า VR1 ตามเข็มนาฬิกาไปจนสุดจะส่งผลให้ค่าหน่วงเวลามีค่าเท่ากับ 50 มิลลิวินาที

	ลายเซ็น	วันเดือนปี
Checkpoint 1 - Lab 3HB06		

6. คำถามท้ายการทดลอง

จงเขียนโปรแกรมให้มีการทำงานเช่นเดียวกับโปรแกรมที่ส่งใน Checkpoint 1 แต่เปลี่ยนมาใช้โปรแกรมภาษาซีบนสภาพแวดล้อม Arduino ปรกติซึ่งไม่ได้ใช้ FreeRTOS (สำหรับคนที่ถนัดภาษาแอสเซมบลีอนุญาตให้ใช้ภาษาแอสเซมบลีแทนภาษาซีได้) และทำส่งใน logbook

7. เอกสารอ้างอิง

- ปัญญยศ ไชยกาฬ, ไมโครคอนโทรลเลอร์และการเชื่อมต่อ, สงขลา, 2562
- ปัญญยศ ไชยกาฬ, เอกสารคำสอนรายวิชา 240-309 สถาปัตยกรรมไมโครคอนโทรลเลอร์และภาษาแอสเซมบลี. https://lms2.psu.ac.th/course/view.php?id=1585