3HB08

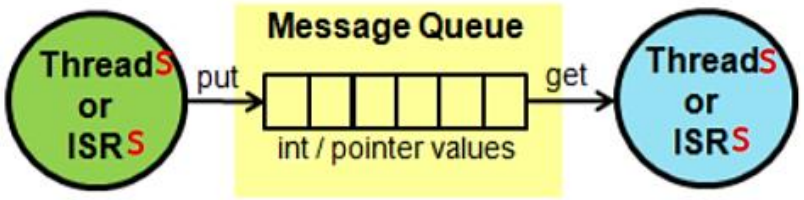
FreeRTOS Queue Management

วัตถุประสงค์

- สามารถใช้งาน Real-time os บน ระบบ Embedded ขนาดเล็กในการสื่อสารตอบโต้กันระหว่างอุปกรณ์ได้

- เข้าใจการจัดล าดับคิวงานในระบบ Multi-Tasking

Queue on RTOS



รูป 1 Message Queue

จากรูปที่ 1 แสดงให้เห็นถึง การจัดลำดับงานโดยใช้ Massage queue ใน OS เพื่อรองรับการจัดการงานจากแหล่งกำเนิดงาน (Thread/ISR) จากหลายแหล่ง และมี Service routine สำหรับการจัดการงานที่ได้รับจาก Message queue สำหรับ FreeRTOS ใน Arudino platform นั้นมีการออกแบบรองรับในการ Message queue (www.freertos.org/a00116.html) โดย ใช้คำสั่งในการสร้าง Message queue box - xQueueCreate(XX, sizeof(struct XXX)); XX = จำนวนคิวที่ queue box รับได้สูงสุด, XXX=structure ที่จะใส่ลงใน Massage queue box - xQueueSend(QueueHandle, pvItemToQueue, TicksToWait); TicksToWait = ระยะเวลาที่Task/Thread นั้น ๆ รอได้ใน queue box - xQueueReceive(QueueHandle,pvItemToQueue,TicksToWait);

การทดลอง

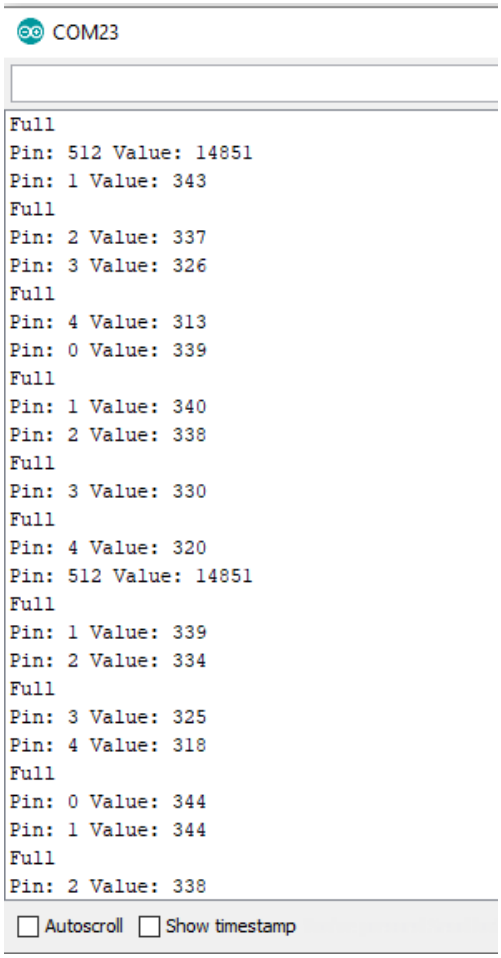
กำหนดให้ใช้ Function xQueueIsQueueFullFromISR เพื่อแสดงสถานะขนาดของ Message queue box ว่า เต็มหรือไม่ โดยมีการ Update การแสดงผลผ่าน Serial ทุก ๆ 150 ms

Code

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>
struct pinRead {
  int pin;
  int value;
};
QueueHandle_t structQueue;
void setup() {
  structQueue = xQueueCreate(10, sizeof(struct pinRead));

  if (structQueue != NULL) {
    xTaskCreate(TaskSerial, "Serial", 64,  NULL, 2, NULL);
    xTaskCreate(TaskAnalogReadPin0, "AnalogReadPin0", 64, NULL, 1, NULL);
    xTaskCreate(TaskAnalogReadPin1, "AnalogReadPin1", 64, NULL, 1, NULL);
    xTaskCreate(TaskAnalogReadPin2, "AnalogReadPin1", 64, NULL, 1, NULL);
    xTaskCreate(TaskAnalogReadPin3, "AnalogReadPin1", 64, NULL, 1, NULL);
    xTaskCreate(TaskAnalogReadPin4, "AnalogReadPin1", 64, NULL, 1, NULL);
    }

    xTaskCreate(TaskBlink,"Blink", 64, NULL, 0, NULL );

}

void loop() {}

void TaskAnalogReadPin0(void *pvParameters)
{
  (void) pvParameters;

  for (;;)
  {
    struct pinRead currentPinRead;
    currentPinRead.pin = 0;
    currentPinRead.value = analogRead(A0);

    xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
      vTaskDelay(1);
  }
}

void TaskAnalogReadPin1(void *pvParameters)
{
  (void) pvParameters;

  for (;;)
  {
    struct pinRead currentPinRead;
    currentPinRead.pin = 1;
    currentPinRead.value = analogRead(A1);
    xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
      vTaskDelay(1);
  }
}

void TaskAnalogReadPin2(void *pvParameters)
{
  (void) pvParameters;
```

```
58
59    for (;;)
60    {
61      struct pinRead currentPinRead;
62      currentPinRead.pin = 2;
63      currentPinRead.value = analogRead(A2);
64      xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
65        vTaskDelay(1);
66    }
67  }
68  void TaskAnalogReadPin3(void *pvParameters)
69  {
70    (void) pvParameters;
71
72    for (;;)
73    {
74      struct pinRead currentPinRead;
75      currentPinRead.pin = 3;
76      currentPinRead.value = analogRead(A3);
77      xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
78        vTaskDelay(1);
79    }
80  }
81  void TaskAnalogReadPin4(void *pvParameters)
82  {
83    (void) pvParameters;
85    for (;;)
86    {
87      struct pinRead currentPinRead;
88      currentPinRead.pin = 4;
89      currentPinRead.value = analogRead(A4);
90      xQueueSend(structQueue, &currentPinRead, portMAX_DELAY);
91        vTaskDelay(1);
92    }
93  }
94
95  void TaskSerial(void * pvParameters) {
96    (void) pvParameters;
97    Serial.begin(9600);
98
99    while (!Serial) {
100     vTaskDelay(1);
101   }
102
103   for (;;)
104   {
105
106     struct pinRead currentPinRead;
107
108     if (xQueueReceive(structQueue, &currentPinRead, portMAX_DELAY) == pdPASS) {
109       Serial.print("Pin: ");
110       Serial.print(currentPinRead.pin);
111       Serial.print(" Value: ");
112       Serial.println(currentPinRead.value);
113     }
114    vTaskDelay(pdMS_TO_TICKS(90));
115   }
116 }
117
118 void TaskBlink(void *pvParameters)
119 {
120   (void) pvParameters;
121
122   for (;;)
123   {
124    if(xQueueIsQueueFullFromISR(structQueue)==pdTRUE)
125    Serial.println("Full");
126    vTaskDelay(pdMS_TO_TICKS(150));
127   }
128 }
```

ผลลัพธ์



คำอธิบาย

จากผลลัพธ์ด้านบนอธิบายได้ว่าระบบมีการสร้าง Task ขึ้นมา 7 Tasks โดยแบ่งเป็น Task ที่ใช้สำหรับอ่านค่าสัญญาณ Analog ตั้งแต่ PIN 0-4 และ Task ที่ใช้สำหรับอ่านค่า PIN ปัจจุบันที่เข้ามาใน queue ว่าเป็น PIN ใด และมีขนาดของข้อมูลเท่าใด และสุดท้ายคือ Task Blink ใช้ตรวจสอบว่า PIN ปัจจุบันที่กำเข้ามาใน queue นั้นเต็มความจุแล้วหรือยัง หากใช่ก็จะแสดงข้อความว่า Full

สรุปผล

จากการทดลองสรุปได้ว่า Queues เป็นรูปแบบหลักของการสื่อสารระหว่างกัน สามารถใช้เพื่อส่งข้อความระหว่างงานและระหว่างการขัดจังหวะและงาน คือ queue ที่ใช้ในการสื่อสารระหว่าง task โดยเป็นแบบ first in first out (FIFO) ตัว message ที่ถูกส่งไปใน queue จะใช้ pointer สำหรับการ blocked ของ queue นั้น เป็นดังนี้ถ้า queue ว่างแล้วมี task มา get message ตัว task นั้นจะถูก blocked ถ้า queue เต็ม แล้วมี task ส่ง message เข้ามา task นั้นจะถูก blocked ถ้ามีหลาย task เข้ามาใช้ queue พร้อมกัน task ที่ priority จะได้ใช้ก่อน