|  |  |  |  |
| --- | --- | --- | --- |
| **KELOMPOK** | C | **ACC** | |
| **Tanggal Praktikum** | 02 Juni 2018 |  |  |
|  | Nugroho Nanda Styanto |
|  | Kharisma Aji Satria Tama |
|  | Puji Hanung Winarko | **Tanggal ACC :** | |
|  | Ahmad Sujarwo | **Revisi Tanggal :** | |

**MODUL 3**

**Flexible and Absolute Timing On**

1. **TUJUAN**
2. Praktikan dapat memahami dan mengetahui fungsi flexible dan absolute timing dalam FreeRTOS.
3. Praktikan dapat membuat program dengan fungsi flexible dan absolute timing dalam FreeRTOS.
4. **ALAT DAN BAHAN**
5. Laptop / PC yang telah diinstal software Arduino
6. Arduino Uno
7. **DASAR TEORI**

Timing atau delay adalah suatu jeda waktu yang digunakan untuk mengeksekusi task selanjutnya. Dalam FreeRTOS, delay terbagi menjadi 2 jenis yaitu flexible dan absolute.

1. **Flexible timing ( vTaskDelay() )**

vTaskDelay () menentukan waktu di mana tugas ingin membuka blokir **relatif terhadap** waktu di mana vTaskDelay () dipanggil. Misalnya, menentukan jangka waktu pemblokiran 100 tick akan menyebabkan tugas untuk membuka blokir 100 ticks setelah vTaskDelay () dipanggil. vTaskDelay () oleh karena itu tidak menyediakan metode yang baik untuk mengendalikan frekuensi tugas periodik karena jalur yang diambil melalui kode, serta tugas dan aktivitas interupsi lainnya, akan mempengaruhi frekuensi di mana vTaskDelay () dipanggil

1. **Absolute timing ( vTaskDelayUntil() )**

Fungsi ini berbeda dari vTaskDelay() dalam satu aspek penting: vTaskDelay() menentukan waktu di mana tugas ingin membuka blokir relatif terhadap waktu di mana vTaskDelay() dipanggil, sedangkan vTaskDelayUntil() menentukan waktu absolut di mana tugas ingin untuk membuka blokir atau mengaktifkan task selanjutnya

vTaskDelay() akan menyebabkan delay aktif dari task selama sejumlah ticks yang ditentukan dari waktu vTaskDelay() dipanggil. Oleh karena itu sulit untuk menggunakan vTaskDelay() dengan sendirinya untuk menghasilkan frekuensi eksekusi tetap sebagai waktu antara task membuka blokir setelah panggilan ke vTaskDelay() dan task berikutnya.

Sedangkan vTaskDelay() menentukan waktu bangun relatif terhadap waktu saat fungsi dipanggil, vTaskDelayUntil() menentukan waktu absolut (tepat) di mana ia ingin membuka blokir atau mengkatifkan task selanjutnya.

Parameters:

pxPreviusWakeTime : Penunjuk ke variabel yang menyimpan waktu saat task terakhir diblokir. Variabel harus diinisialisasi dengan waktu saat ini sebelum penggunaan pertama (lihat contoh di bawah). Setelah ini, variabel secara otomatis diperbarui dalam vTaskDelayUntil ().

xTimeIncrement : Periode waktu siklus. Task akan dibebaskan pada waktunya (\* pxPreviousWakeTime + xTimeIncrement). Memanggil vTaskDelayUntil dengan nilai parameter xTimeIncrement yang sama akan menyebabkan task dijalankan dengan periode interval tetap.

1. **HASIL PRAKTIKUM**

#include "FreeRTOS\_AVR.h"

#include "basic\_io\_AVR.h"

void vContinousProcessingTask (void \*pvParameters);

void vPeriodicTask (void \*pvParameters);

const char \*pcTextForTask1 = "task 1 is running\r\n";

const char \*pcTextForTask2 = "task2 is running\r\n";

const char \*pcTextForPeriodicTask3 = "task3 is running\r\n";

const char \*pcTextForPeriodicTask4 = "task4 is running\r\n";

void setup(void)

{

Serial.begin(9600);

xTaskCreate(vContinousProcessingTask, "Task 1", 200, (void\*)pcTextForTask1, 2, NULL);

xTaskCreate(vContinousProcessingTask, "Task 2", 200, (void\*)pcTextForTask2, 2, NULL);

xTaskCreate(vPeriodicTask3, "Task 3", 200, (void\*)pcTextForPeriodicTask3, 3, NULL);

xTaskCreate(vPeriodicTask4, "Task 4", 200, (void\*)pcTextForPeriodicTask4, 3, NULL);

vTaskStartScheduler();

for(;;);

}

void vContinousProcessingTask (void \*pvParameters)

{

char \*pcTaskname;

pcTaskname = (char\*)pvParameters;

for(;;)

{

vPrintString(pcTaskname);

}

vTaskDelay(500);

}

void vPeriodicTask3 (void \*pvParameters)

{

TickType\_t xLastWakeTime3;

xLastWakeTime3 = xTaskGetTickCount ();

for(;;)

{

vPrintString("Periodic task3 is running------------------\r\n");

vTaskDelayUntil(&xLastWakeTime3, (500/portTICK\_PERIOD\_MS));

}

}

void vPeriodicTask4 (void \*pvParameters)

{

TickType\_t xLastWakeTime4;

xLastWakeTime4 = xTaskGetTickCount ();

for(;;)

{

vTaskDelayUntil(&xLastWakeTime4, (500/portTICK\_PERIOD\_MS));

vPrintString("Periodic task4 is running------------------\r\n");

vTaskDelayUntil(&xLastWakeTime4, (200/portTICK\_PERIOD\_MS));

}

}

void loop() {

// put your main code here, to run repeatedly:

}

1. **ANALISA**

Analisa Percobaan

Pada praktikum ini menggunakan vTaskdelay(), vTaskDelayUntil()pada program .Dengan membuat program dapat berjalan secara terus menerus. vTaskDelay() digunakan pada void vContinousProcessingTask sedangkan vTaskDelayUntil() digunakan pada void vPeriodicTask. Task 1, 2 terdapat pada void vContinousProcessingTask sedangkan Task 3 terdapat pada void vPeriodicTask3 dan Task 4 terdapat pada void vPeriodicTask4

1. **KESIMPULAN**
2. vTaskDelay di mana tugas membuka blokir relatif terhadap waktu di mana vTaskDelay() ini dipanggil.
3. vTaskDelayUntil() menentukan waktu absolut yang membuka blokir atau mengaktifkan task selanjutnya.