|  |  |  |
| --- | --- | --- |
| **KELOMPOK** | **E/5** | **ACC** |
| **Tanggal Praktikum** |  | Asisten 1 |
| Asisten 2 |
|  | 1. **IBNU S K (D400150093)** |  |
|  | 1. **TAUFIQ D (D400150102)** | **Tanggal ACC :** |
|  | 1. **AGUS YOGA (D400150135)** | **Revisi Tanggal :** |

**MODUL 1**

**Analog Digital Read dengan FreeRTOS**

1. **TUJUAN**
2. Praktikan dapat memahami salah satu sistem operasi yang ada pada Arduino yaitu FreeRTOS.
3. Praktikan dapat memahami program Analog Digital Read menggunakan sistem operasi FreeRTOS.
4. Praktikan dapat membuat basic program dengan menggunakan sistem oprasi FreeRTOS.
5. **ALAT DAN BAHAN**
6. Laptop / PC yang telah diinstal software Arduino
7. Arduino Uno
8. Ldr
9. Kabel jumper
10. Rotation sensor
11. Push button
12. **DASAR TEORI**

* FREERTOS

Karakter dasar dari RTOS adalah sebuah sistem yang mempunyai beberapa konsekuensi yang akan berpengaruh pada sistem apabila deadline (batas akhir waktu pelaksanaan task) tidak terpenuhi. RTOS sendiri terdiri dari 2 jenis yaitu, sistem soft RTOS dan sistem hard RTOS. Soft RTOS bisa dideskripsikan sebagai sistem yang hampir selalu menyelesaikan task dengan waktu yang telah ditentukan. Pada soft RTOS kemungkinan penyelesaian task melewati batas waktu pelaksanaan task masih bisa terjadi. Dan pada sistem soft RTOS, apabila terjadi kegagalan mencapai deadline dalam waktu yang telah ditentukan maka sistem akan mengalami efek yang tidak begitu berbahaya bagi sistem. Contohnya seperti penurunan performa sistem. Sedangkan hard RTOS merupakan sistem yang dipastikan selalu menyelesaikan task dalam waktu yang telah ditentukan. Dikatakan pasti selalu menyelesaikan task karena hard RTOS selalu menyelesaikan task sebelum deadline dan apabila terjadi kegagalan menyelesaikan task maka sistem akan mengalami efek berbahaya yang dapat merusak sistem secara keseluruhan.

* TASK

Sebuah task, merupakan sebuah objek/program yang dapat dieksekusi dan beranggapan mempunyai CPU untuk task itu sendiri. Salah satu proses perancangan aplikasi dengan RTOS yaitu membagi semua pekerjaan dalam aplikasi tersebut menjadi beberapa bagian task. Tiap task merupakan loop yang akan terus berulang. Dalam proses pengulangan tersebut, task akan mengalami tiga buah keadaan (gambar 2.1) yaitu:

1. Running, merupakan keadaan di mana sebuah task dengan prioritas tertinggi berjalan
2. Ready, merupakan keadaan yang dialami sebuah task jika terdapat sebuah task lain sedang running dan task yang berada pada ready akan melanjutkan pengerjaan task yang sempat tertunda oleh task yang lebih tinggi prioritasnya.
3. Blocked, merupakan keadaan di mana jika sebuah task membutuhkan event atau data maka akan masuk ke dalam blocked hingga event atau data yang dibutuhkan telah tersedia

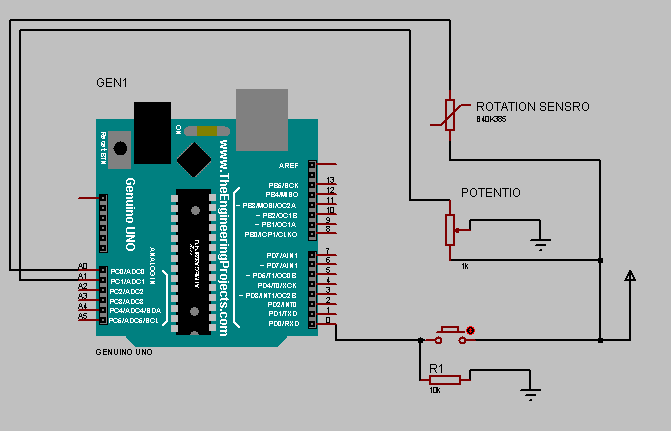
* LDR (Light Dependent Resistor)

Light Dependent Resistor atau disingkat dengan LDR adalah jenis Resistor yang nilai hambatan atau nilai resistansinya tergantung pada intensitas cahaya yang diterimanya. Nilai Hambatan LDR akan menurun pada saat cahaya terang dan nilai Hambatannya akan menjadi tinggi jika dalam kondisi gelap. Dengan kata lain, fungsi LDR (Light Dependent Resistor) adalah untuk menghantarkan arus listrik jika menerima sejumlah intensitas cahaya (Kondisi Terang) dan menghambat arus listrik dalam kondisi gelap.

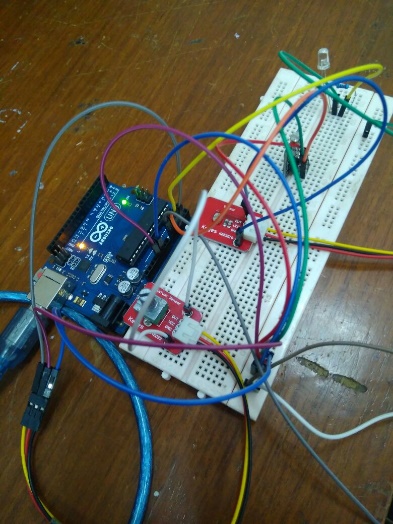
1. **HASIL PRAKTIKUM**

**D.1 Hasil Percobaan 1**

**D.1.1 Skema Rangkaian**

****

**D.1.2 Foto Rangkaian**



**D.1.3 Script Program**

#include <Arduino\_FreeRTOS.h>

#include <semphr.h> // add the FreeRTOS functions for Semaphores (or Flags).

// Declare a mutex Semaphore Handle which we will use to manage the Serial Port.

// It will be used to ensure only only one Task is accessing this resource at any time.

SemaphoreHandle\_t xSerialSemaphore;

// define two Tasks for DigitalRead & AnalogRead

void TaskDigitalRead( void \*pvParameters );

void TaskAnalogRead0( void \*pvParameters );

void TaskAnalogRead1( void \*pvParameters );

void TaskAnalogRead2( void \*pvParameters );

// the setup function runs once when you press reset or power the board

void setup() {

// initialize serial communication at 9600 bits per second:

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4 based boards.

}

// Semaphores are useful to stop a Task proceeding, where it should be paused to wait,

// because it is sharing a resource, such as the Serial port.

// Semaphores should only be used whilst the scheduler is running, but we can set it up here.

if ( xSerialSemaphore == NULL ) // Check to confirm that the Serial Semaphore has not already been created.

{

xSerialSemaphore = xSemaphoreCreateMutex(); // Create a mutex semaphore we will use to manage the Serial Port

if ( ( xSerialSemaphore ) != NULL )

xSemaphoreGive( ( xSerialSemaphore ) ); // Make the Serial Port available for use, by "Giving" the Semaphore.

}

// Now set up two Tasks to run independently.

xTaskCreate(

TaskDigitalRead

, (const portCHAR \*)"DigitalRead" // A name just for humans

, 128 // This stack size can be checked & adjusted by reading the Stack Highwater

, NULL

, 5 // Priority, with 3 (configMAX\_PRIORITIES - 1) being the highest, and 0 being the lowest.

, NULL );

xTaskCreate(

TaskAnalogRead0

, (const portCHAR \*) "AnalogRead"

, 128 // Stack size

, NULL

, 2 // Priority

, NULL );

xTaskCreate(

TaskAnalogRead1

, (const portCHAR \*) "AnalogRead"

, 128 // Stack size

, NULL

, 3 // Priority

, NULL );

xTaskCreate(

TaskAnalogRead2

, (const portCHAR \*) "AnalogRead\_Potentio"

, 128 // Stack size

, NULL

, 4 // Priority

, NULL );

// Now the Task scheduler, which takes over control of scheduling individual Tasks, is automatically started.

}

void loop()

{

// Empty. Things are done in Tasks.

}

/\*--------------------------------------------------\*/

/\*---------------------- Tasks ---------------------\*/

/\*--------------------------------------------------\*/

void TaskDigitalRead( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.

{

/\*

DigitalReadSerial

Reads a digital input on pin 2, prints the result to the serial monitor

This example code is in the public domain.

\*/

// digital pin 2 has a pushbutton attached to it. Give it a name:

uint8\_t pushButton = 2;

// make the pushbutton's pin an input:

pinMode(pushButton, INPUT);

for (;;) // A Task shall never return or exit.

{

// read the input pin:

Serial.print("digitalread :");

int buttonState = digitalRead(pushButton);

// See if we can obtain or "Take" the Serial Semaphore.

// If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.

if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 5 ) == pdTRUE )

{

// We were able to obtain or "Take" the semaphore and can now access the shared resource.

// We want to have the Serial Port for us alone, as it takes some time to print,

// so we don't want it getting stolen during the middle of a conversion.

// print out the state of the button:

Serial.println(buttonState);

xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.

}

vTaskDelay(100); // one tick delay (15ms) in between reads for stability

}

}

void TaskAnalogRead0( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.

{

for (;;)

{

// read the input on analog pin 0:

int sensorValue = analogRead(A0);

// See if we can obtain or "Take" the Serial Semaphore.

// If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.

if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 6 ) == pdTRUE )

{

// We were able to obtain or "Take" the semaphore and can now access the shared resource.

// We want to have the Serial Port for us alone, as it takes some time to print,

// so we don't want it getting stolen during the middle of a conversion.

// print out the value you read:

Serial.print("task0:");

Serial.println(sensorValue);

xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.

}

vTaskDelay(200); // one tick delay (15ms) in between reads for stability

}

}

void TaskAnalogRead1( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.

{

for (;;)

{

// read the input on analog pin 0:

int sensorValue = analogRead(A1);

// See if we can obtain or "Take" the Serial Semaphore.

// If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.

if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 7 ) == pdTRUE )

{

// We were able to obtain or "Take" the semaphore and can now access the shared resource.

// We want to have the Serial Port for us alone, as it takes some time to print,

// so we don't want it getting stolen during the middle of a conversion.

// print out the value you read:

Serial.print("task1:");

Serial.println(sensorValue);

xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.

}

vTaskDelay(300); // one tick delay (15ms) in between reads for stability

}

}

void TaskAnalogRead2( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.

{

for (;;)

{

// read the input on analog pin 0:

int sensorValue = analogRead(A3);

// See if we can obtain or "Take" the Serial Semaphore.

// If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.

if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 8 ) == pdTRUE )

{

// We were able to obtain or "Take" the semaphore and can now access the shared resource.

// We want to have the Serial Port for us alone, as it takes some time to print,

// so we don't want it getting stolen during the middle of a conversion.

// print out the value you read:

Serial.print("task2:");

Serial.println(sensorValue);

xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.

}

vTaskDelay(300); // one tick delay (15ms) in between reads for stability

}

}

1. **ANALISA**

Pada projek ini terdapat 3 task, yaitu task membaca kondisi tombol, task membaca sensor rotasi dan task yang ketiga adalah membaca variabel resistor, pada script dimulai dari pendaftaran task yang meliputi alokasi memory, prioritas, dan lain-lain, kemudian terdapat resource yaf digunakan secara bergantian oleh sebab itu digunakanlah smaphore yang berfungsi memanajemen resourse yang akan digunakan secara bergantian, nilai dari masing-masing task akan dikirimkan secara bergantian ke serial monitor. Kemudian yang terakhir adalah pembuatan dari fungsi task yang berada pada lokasi paling bawah script program.

1. **KESIMPULAN**
2. Tingkat prioritas berfungsi agar task yang mempunyai tugas lebih penting dari pada task lain didahulukan pengerjaannya.
3. Penentuan tingkat prioritas tertinggi bisa dilakukan dengan melihat periode dari seberapa seringnya muncul task tersebut. Semakin sering task tersebut muncul maka semakin tinggi prioritas dai task tersebut
4. FreeRTOS (Free Real-time operating systems) adalah sebuah real time operating system pada Arduino yang banyak digunakan oleh mikrokontroler untuk kebutuhan sistem operasinya.
5. Sistem operasi RTOS dalam program dibagi dalam beberapa task yang dapat diatur urutannya sesuai dengan kebutuhan.