|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

MODUL 1

BLINK AND ANALOG READ

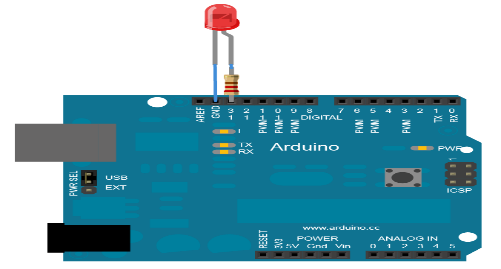
1. TUJUAN
2. Peserta mengerti cara membangun program untuk Free-RTOS
3. Peserta dapat melakukan proses instalasi source-code atau library freeRTOS dalam IDE (Integrated Development Environment) Arduino.
4. Peserta dapat memahami Memahami cara kerja FreeRTOS untuk sebuah kasus sederhana, yaitu mematikan dan menghidupkan lampu LED.
5. ALAT DAN BAHAN
6. Arduino
7. LED
8. Project Board
9. Kabel Jumper
10. DASAR TEORI

FreeRTOS adalah sistem operasi real-time (Real Time Operating System - RTOS)Perbedaannya dengan siustem operasi biasa (Windows & Linux ) adalah dalam RTOS jadwal waktu eksekusi setiap aplikasi menjadi pertimbangan utama. Jika ada aplikasi yang dieksekusi tidak sesuai jadwal berartti sistem RTOS gagal.

Dalam RTOS setiap aplikasi memiliki jadwal ekseskusi, contoh aplikasi A dieksekusi setiap 100ms, aplikasi B dieksekusi setiap 50ms dst. Setiap aplikasi tidak ada kaitan atau terpisah, independent. Jika ekseskusi aplikasi tidak sesuai dengan jadwal maka dikatakan perfomance RTOS kurang bagus, semakinb besar selisih waktu semakin tidak bagus. Selisih waktu ekseskusi dengan jadwal yang ditentukan disebut dengan JITTER. Idealnya JITTER = 0, dalam implementasi JITTER > 0.

1. HASIL PRAKTIKUM

D. 1. PERCOBAAN 1

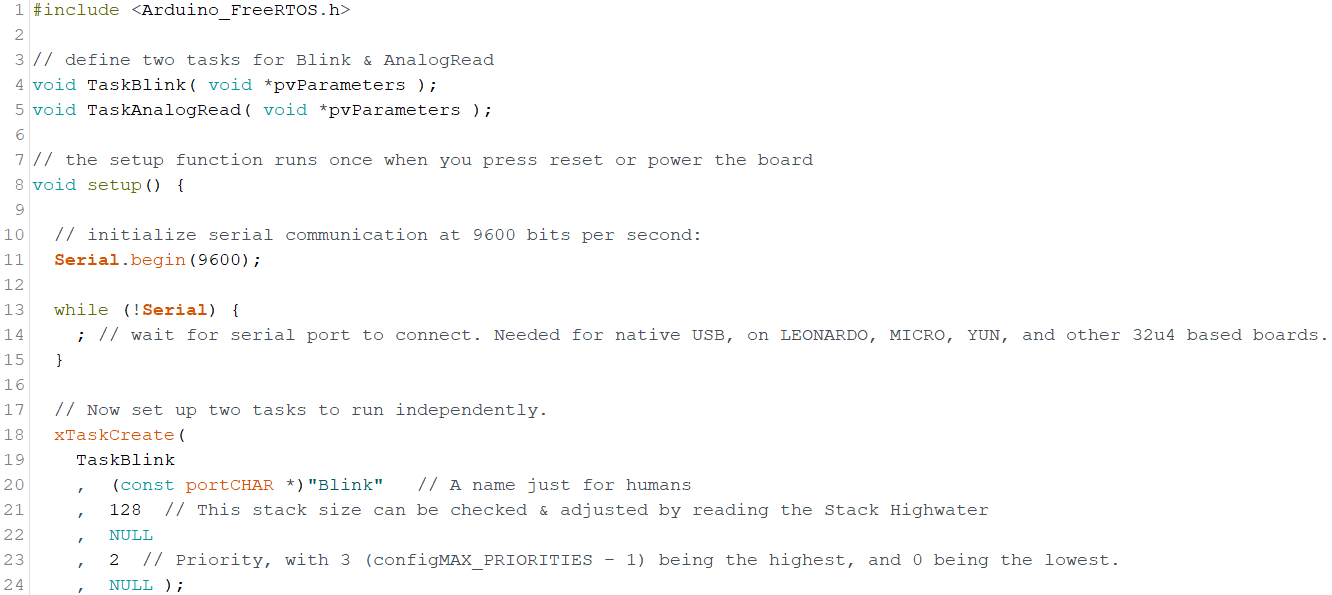
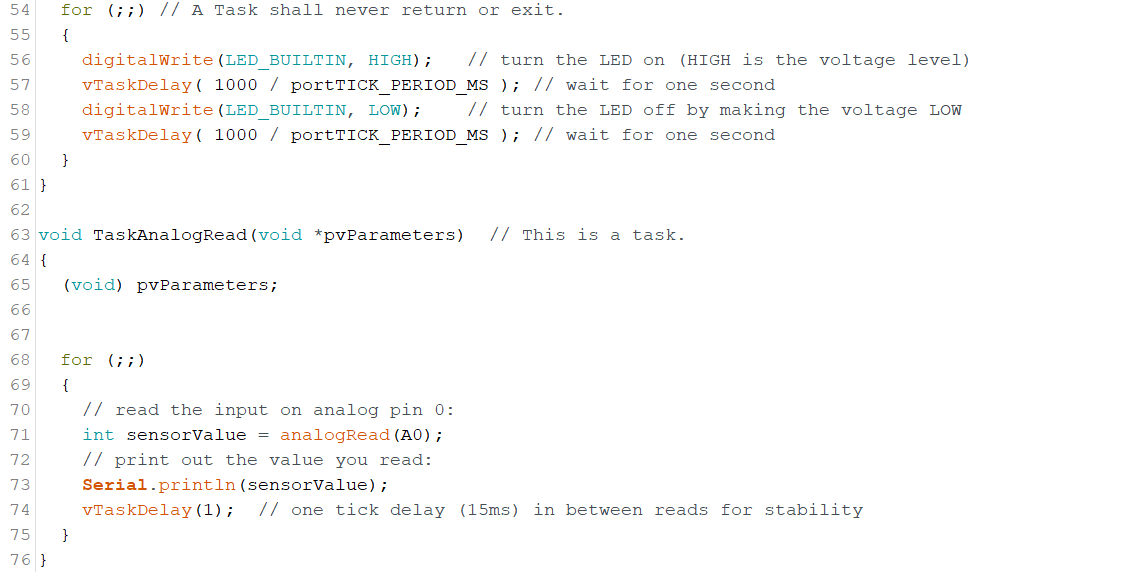
 D.1.1. SKEMA RANGKAIAN



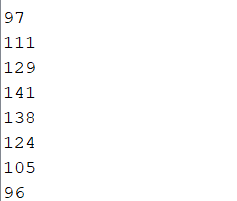
D.1.2. FOTO RANGKAIAN



D.1.3. SCRIPT PROGRAM



D.1.4. FOTO HASIL



1. ANALISA

E.1. PERCOBAAN 1

Pada percobaan ini terdapat dua task, yaitu task Blink dan Analog Read. Keduanya bekerja berdasarkan berdasarkan prioritas, semakin besar angka prioritas maka semakin tinggi prioritas sebuah task. Task Blink mempunyai kinerja akan menghidupkan dan mematikan led, dengan delay dalam task ini 1000/portTick. Sedangkan untun task analog read ini merupakan task yang bekerja berdasarkan sensor analog yang terdapat pin Analog A0.

Dalam FreeRTOS tahap awal yaitu terdiri dari pembuatan task dan berikutnya isi task tersebut.

1. KESIMPULAN
2. FreeRTOS adalah sistem operasi real-time (Real Time Operating System - RTOS)
3. Dalam multitasking FreeRTOS bekerja berdasarkan prioritas, semakin besar angka prioritas, semakin tinggi priorritas.
4. Dalam pemograman FreeRTOS terdiri dua tahap yaitu, tahap awal yaitu terdiri dari pembuatan task dan berikutnya isi task tersebut.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

PROJECT 1

BLINK, DIGITAL READ AND ANALOG READ

1. TUJUAN
2. Peserta mengerti cara membangun program untuk Free-RTOS
3. Peserta dapat memahami Memahami cara kerja FreeRTOS untuk sebuah kasus sederhana, yaitu mematikan dan menghidupkan lampu LED.
4. Peserta dapat mengembangkan task
5. ALAT DAN BAHAN
6. Arduino
7. LED
8. Project Board
9. Kabel Jumper
10. DASAR TEORI

FreeRTOS adalah sistem operasi real-time (Real Time Operating System - RTOS)Perbedaannya dengan siustem operasi biasa (Windows & Linux ) adalah dalam RTOS jadwal waktu eksekusi setiap aplikasi menjadi pertimbangan utama. Jika ada aplikasi yang dieksekusi tidak sesuai jadwal berartti sistem RTOS gagal.

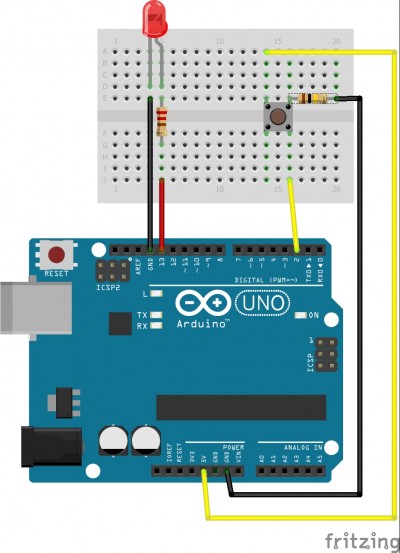
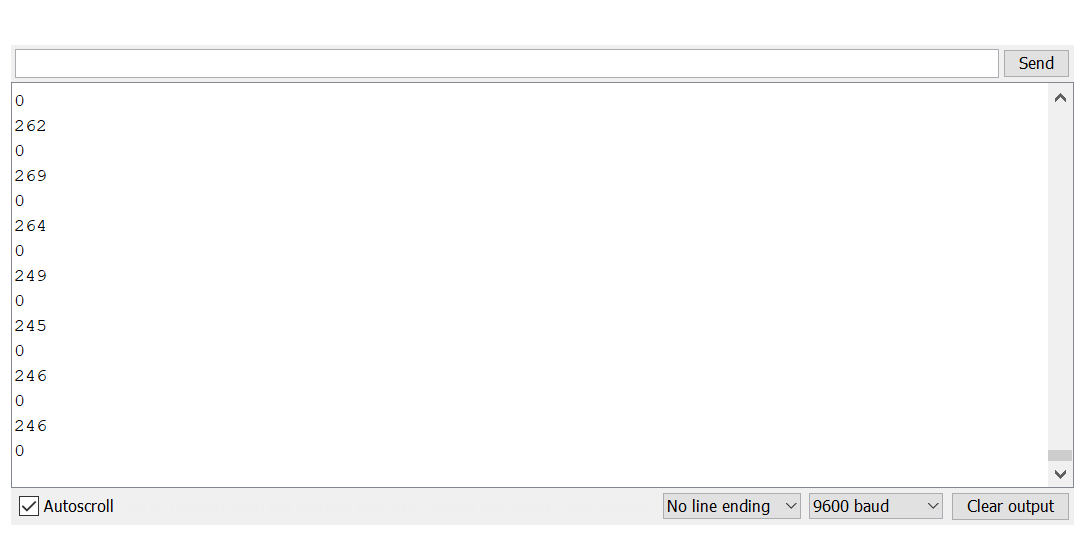
LED atau singkatan dari Light Emitting Diode adalah salah satu komponen elektronik yang tidak asing lagi di kehidupan manusia saat ini. LED saat ini sudah banyak dipakai, seperti untuk penggunaan lampu permainan anak-anak, untuk rambu-rambu lalu lintas, lampu indikator peralatan elektronik hingga ke industri, untuk lampu emergency, untuk televisi, komputer, pengeras suara (speaker), hard disk eksternal, proyektor, LCD, dan berbagai perangkat elektronik lainnya sebagai indikator bahwa sistem sedang berada dalam proses kerja, dan biasanya berwarna merah atau kuning. LED ini banyak digunakan karena komsumsi daya yang dibutuhkan tidak terlalu besar dan beragam warna yang ada dapat memperjelas bentuk atau huruf yang akan ditampilkan. dan banyak lagi.

Dalam percobaan ini mencoba membuat tiga task sekaligus, dimana ketiganya akan bekerja secara bersamaan, LED akan bekerja secara on off, di sisi lain Arduino UNO akan membaca data analog dan digital. Data digital di dapat berdasarkan inputan pada pin digital Arduino UNO.

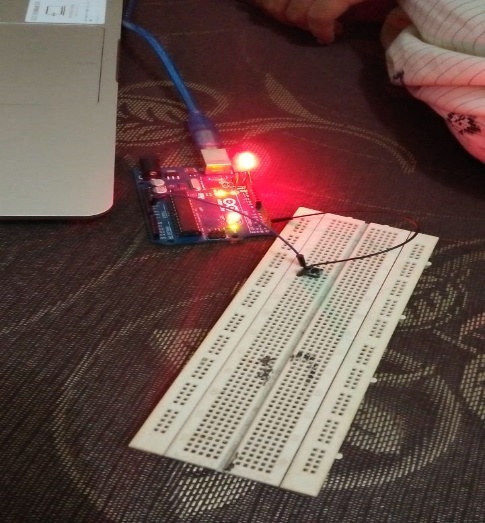
1. HASIL PRAKTIKUM

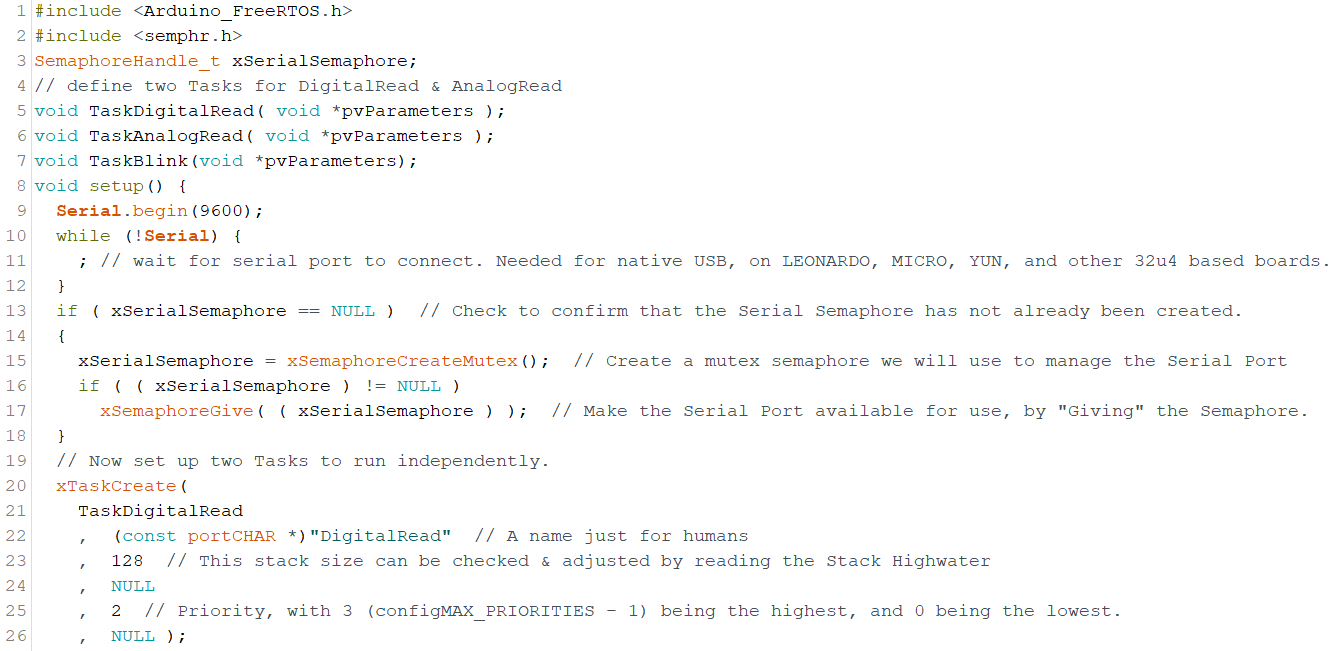
D.1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN

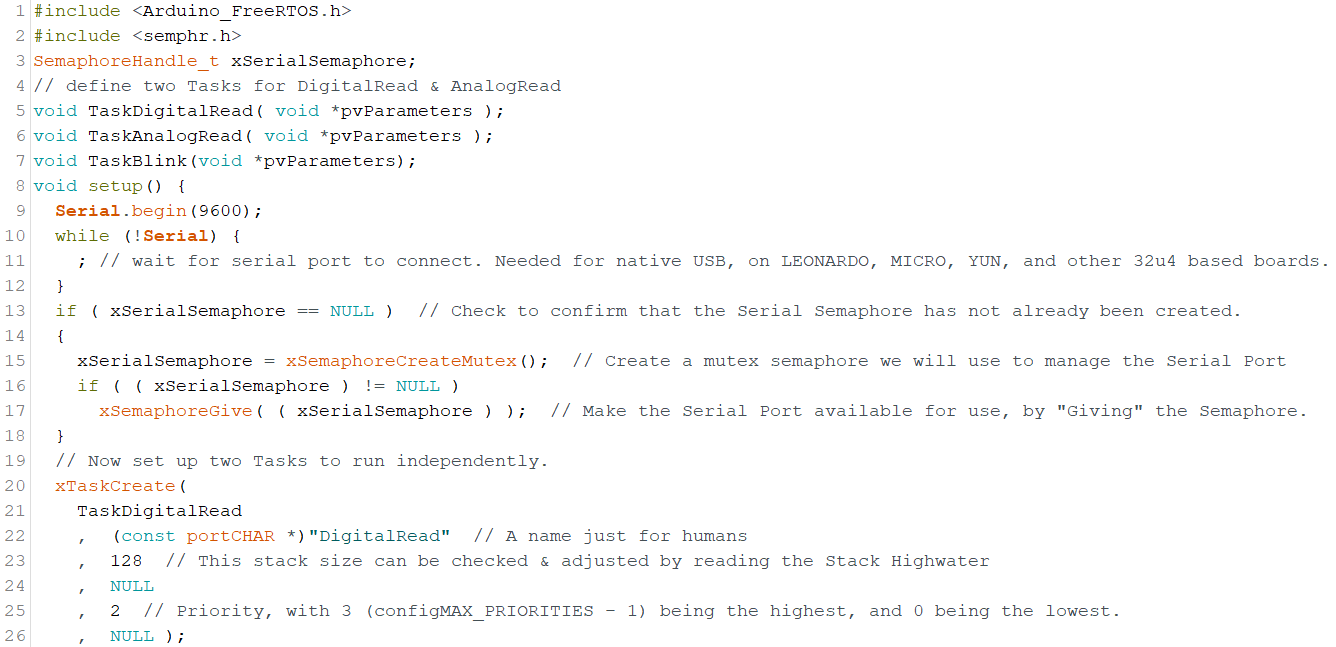


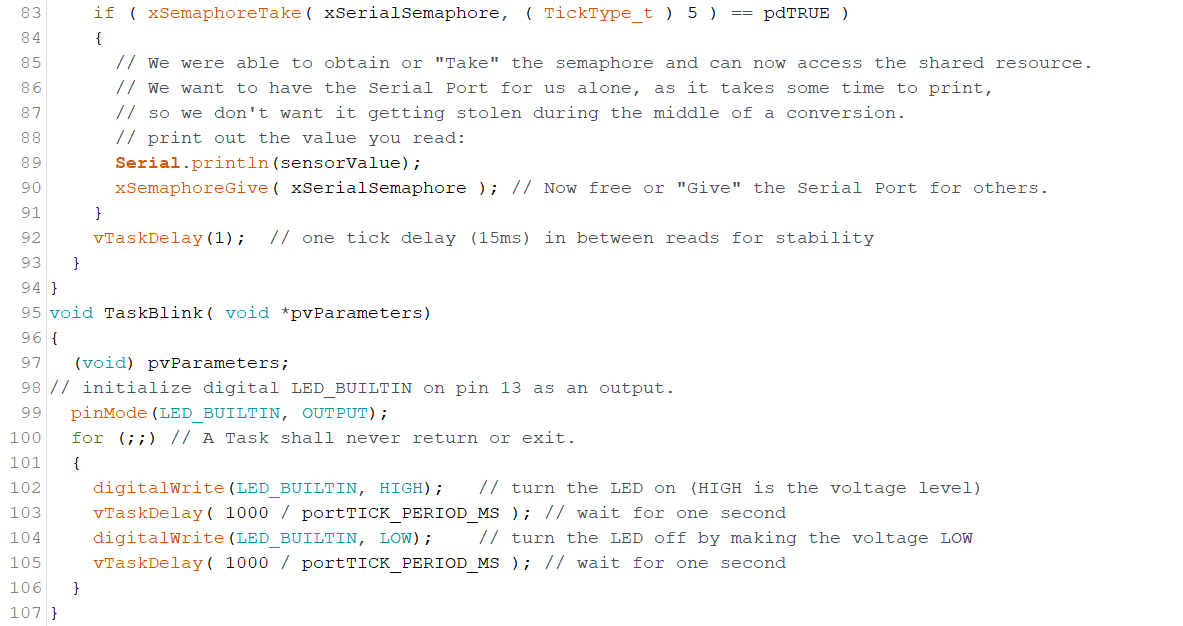
D.1.2. FOTO RANGKAIAN



D.1.3. SCRIPT PROGRAM

‘





1. ANALISA

E.1. PERCOBAAN 1

Pada percobaan ini terdapat tiga task, yaitu task Blink, Analog Read, dan Digital Read. Keduanya bekerja berdasarkan berdasarkan prioritas, semakin besar angka prioritas maka semakin tinggi prioritas sebuah task. Task Blink mempunyai kinerja akan menghidupkan dan mematikan led, dengan delay dalam task ini 1000/portTick. Sedangkan untuk task analog read ini merupakan task yang bekerja berdasarkan sensor analog yang terdapat pin Analog A0. Dan push button yang kita jadikan sebaga inputan data digital terletak pada pin 2.

Dalam FreeRTOS tahap awal yaitu terdiri dari pembuatan task dan berikutnya isi task tersebut.

1. KESIMPULAN
   1. Dalam multitasking FreeRTOS bekerja berdasarkan prioritas, semakin besar angka prioritas, semakin tinggi priorritas.
   2. Untuk membaca data analog maka menempatkan sensor pada analog pin, dan untuk membaca data digital maka menempatkan sensor pada digital pin.
   3. Dalam penulisan script program, task terdiri dari dua bagian yaitu tahap awal yaitu terdiri dari pembuatan task dan berikutnya isi task tersebut.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

MODUL 2

SEMAPHORE

1. TUJUAN
2. Peserta mengerti cara membangun program semaphore untuk Free-RTOS
3. Peserta dapat memahami Memahami cara kerja semaphore dalam FreeRTOS untuk sebuah kasus sederhana.
4. Peserta mengerti kegunaan semaphore dalam sebuah program.
5. ALAT DAN BAHAN
6. Arduino
7. LED
8. Project Board
9. Kabel Jumper
10. DASAR TEORI

Jika ada resources (variable, hardware) yang akan diakses lebih dari satu aplikasi (task) maka diperlukan manajemen resource, agar tidak terjadi konflik antar aplikasi. Dalam sebuah mikrokontroler yang menjadi pusat pengolahan maka sangatlah penting kegiatan manajemen resource.

FreeRTOS telah menyediakan sebuah fitur untuk mengatur sebuah resource ketika penggunaannya diakses lebih dari satu aplikasa (task), salah satunya menggunakan Semaphore. Dalam skenario penggunaannya, nilai hitungan semaphore mewakili jumlah sumber daya yang tersedia.

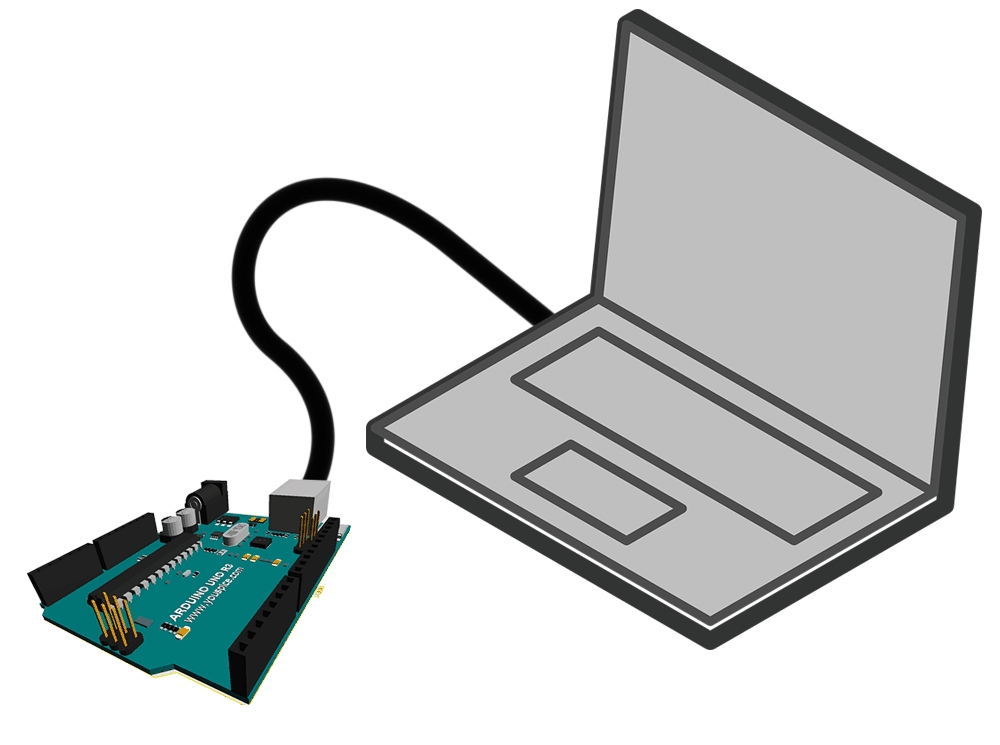
Untuk mendapatkan kontrol sumber daya, tugas pertama-tama harus berhasil "mengambil" semaphore. Tindakan “mengambil” semaphore akan menurunkan nilai hitungan semaphore. Ketika nilai hitungan mencapai nol, tidak ada lagi sumber daya yang tersedia, dan upaya lebih lanjut untuk 'mengambil' semaphore akan gagal. Ketika tugas selesai dengan sumber daya, tugas itu harus 'memberi' semaphore. Tindakan 'memberi' semaphore akan meningkatkan nilai hitungan semaphore, menunjukkan bahwa sumber daya tersedia, dan memungkinkan upaya masa depan untuk 'mengambil' semaphore agar berhasil.

Semaphore yang dibuat untuk mengelola sumber daya harus dibuat dengan nilai hitungan awal sama dengan jumlah sumber daya yang tersedia.

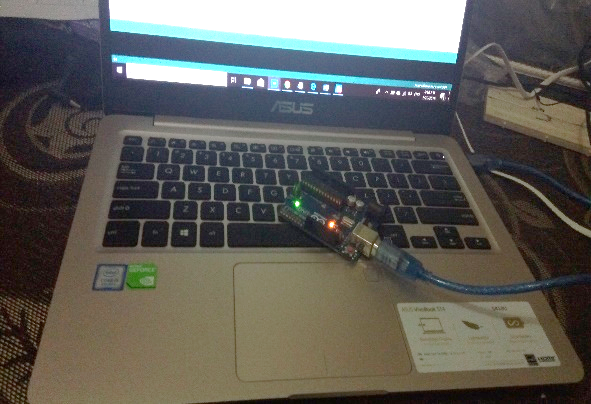
1. HASIL PRAKTIKUM

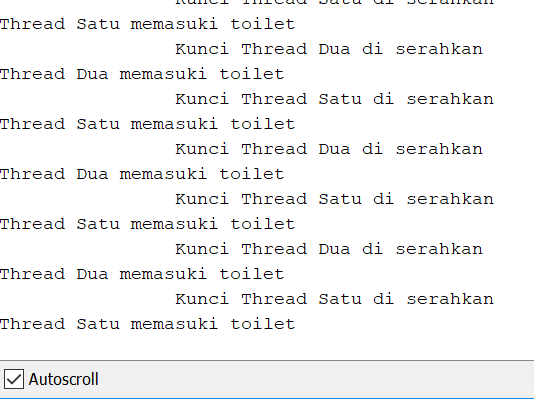
D. 1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN



D.1.2. FOTO RANGKAIAN





D.1.3. SCRIPT PROGRAM



1. ANALISA

E.1. PERCOBAAN 1

Dalam percobaan ini manajemen resources diibaratkan seperti pemakaian toilet, dimana ada dua thread yang akan menggunakan dua toilet secara bergantian. Ketika thread 1 masuk ke dalam toilet artinya thread satu membawa kunci toilet, maka thread dua harus menunggu, begitu pula ketika thread dua memasuki toilet maka thread satu menunggu. Kunci inilah yang merupakan resources yang dipakai secara bersamaan.

Instruksi Fungsi xSemaphoreGive(sem) pada Void Setup ini artinya bahwa sudah ada kunci yang tersedia misalnya dalam box kunci. Sedangkan sem = xSemaphoreCreateCounting(1,0) artinya jumlah kunci.

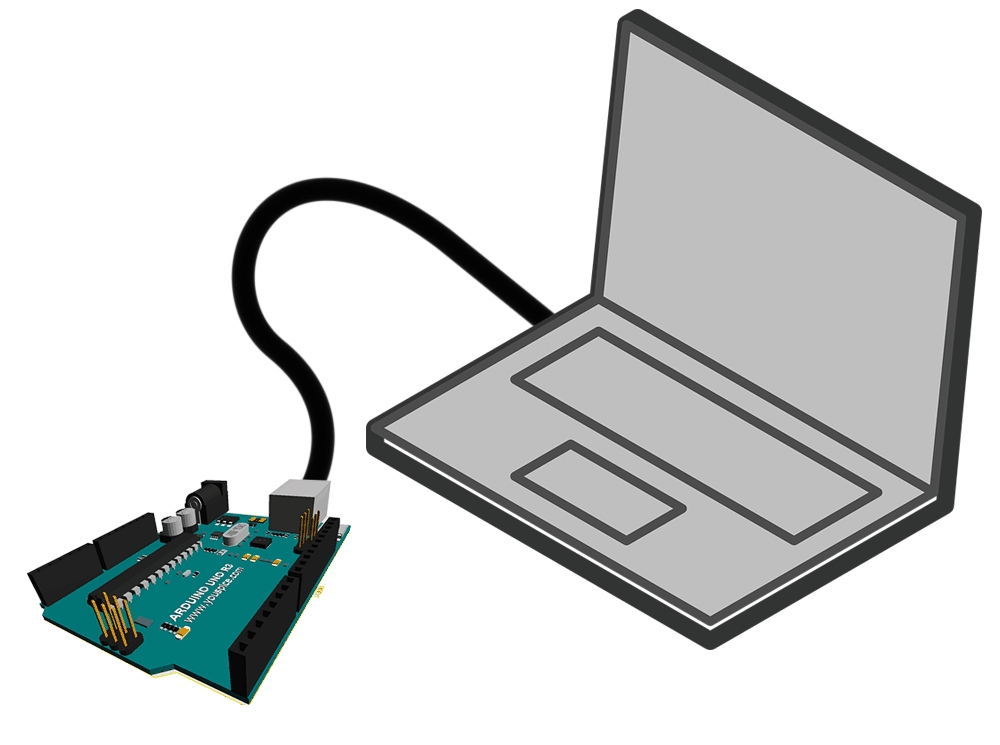
if (xSemaphoreTake (sem, portMAX\_DELAY)){ xSemaphoreGive(sem);

instruksi ini terdapat pada Task satu dan Task dua, fungsi instruksi ini yaitu mengambil dan mengembalikan kembali kunci (resources) agar dapat dipakai bersamaan secara bergantian.

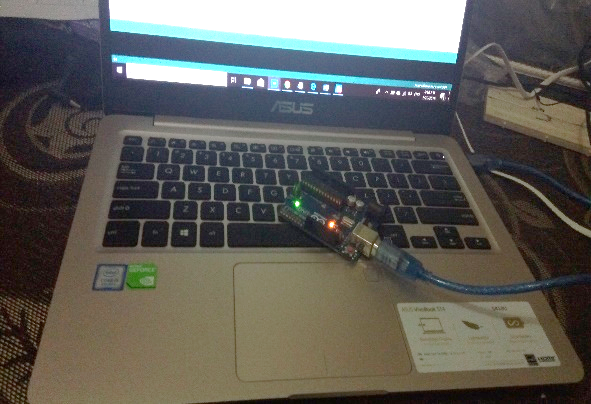
1. TUGAS

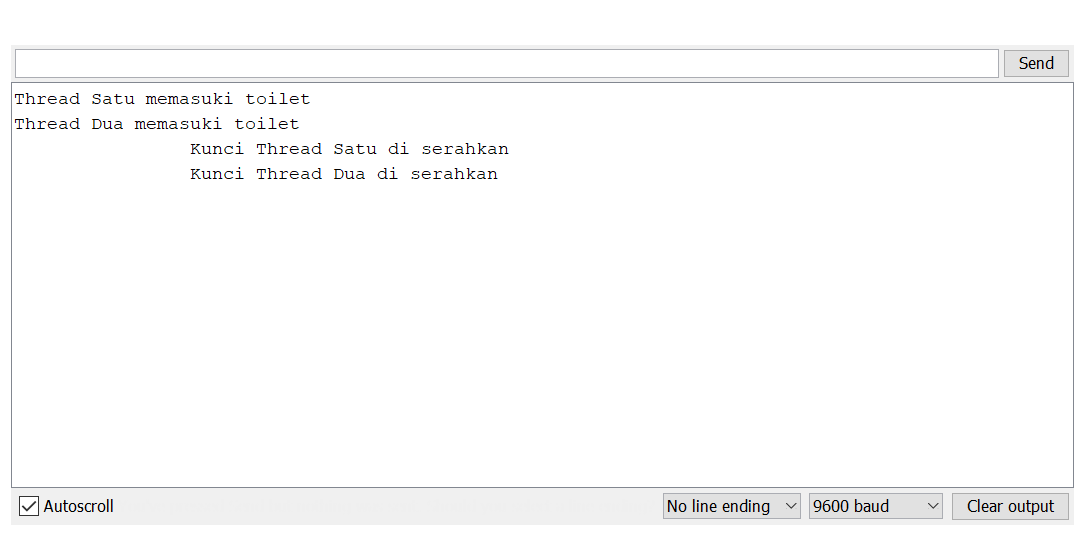
Kelompok kami memilih sebuah kasus bagaimana jika Thread Satu masuk bersama dengan Thread dua tanpa Thread Satu menyerahkan kunci toilet yang telah diambil.

F.1.1. SKEMA RANGKAIAN



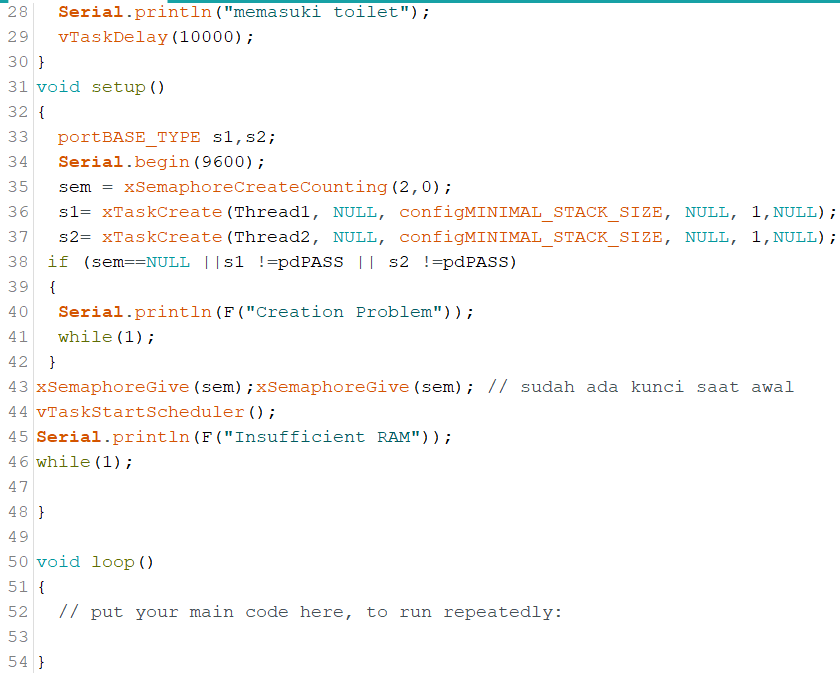
F.1.2. FOTO RANGKAIAN

W



F.1.3. SCRIPT PROGRAM





F.1.4. ANALISA

Dalam kasus ini tidak jauh beda dengan percobaan, hanya saja kita membuat bahwa kunci yang tersedia di box kunci ada dua, sehingga thread satu dan thread dua dapat menggunakan toilet secara bersamaan tanpa kunci dikembalikan terlebih dahulu. Mengembalikan kunci kita non akifkan dengan menghapus xSemaphoreGive(sem) pada masing masing task. Namun menambah xSemaphoreGive(sem) agar jumlah kunci dalam box sesuai jumlah kunci yang tersedia yaiu pada sem = xSemaphoreCreateCounting(2,0).

1. KESIMPULAN
2. Instruksi Fungsi xSemaphoreGive(sem) pada Void Setup ini artinya bahwa sudah ada kunci yang tersedia misalnya dalam box kunci. Sedangkan sem = xSemaphoreCreateCounting(1,0) artinya jumlah kunci.
3. Mengembalikan kunci kita non akifkan dengan menghapus xSemaphoreGive(sem) pada masing masing task. Namun menambah xSemaphoreGive(sem) agar jumlah kunci dalam box sesuai jumlah kunci yang tersedia yaiu pada sem = xSemaphoreCreateCounting(2,0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

MODUL 3

FLEXIBLE AND TIMMING ON

1. TUJUAN
2. Peserta mengerti cara membagi resources dalam Free-RTOS
3. Peserta mengerti kegunaan manajemen resources dalam sebuah program.
4. ALAT DAN BAHAN
5. Arduino
6. LED
7. Project Board
8. Kabel Jumper
9. DASAR TEORI

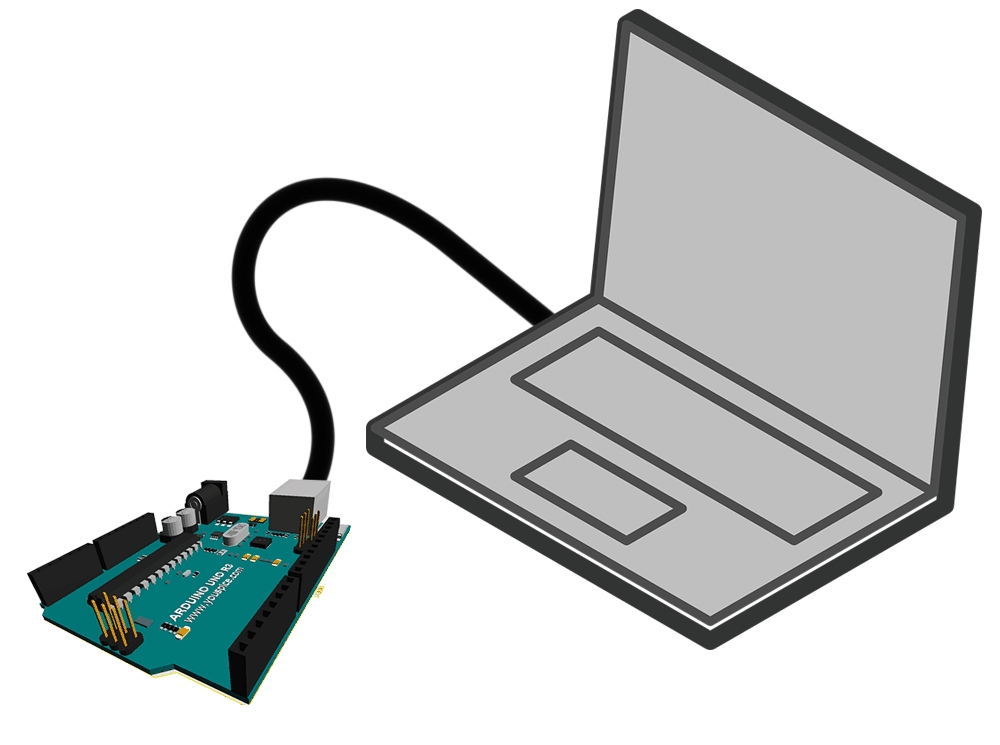
Jika ada resources (variable, hardware) yang akan diakses lebih dari satu aplikasi (task) maka diperlukan manajemen resource, agar tidak terjadi konflik antar aplikasi. Dalam sebuah mikrokontroler yang menjadi pusat pengolahan maka sangatlah penting kegiatan manajemen resource.

FreeRTOS telah menyediakan sebuah fitur untuk mengatur sebuah resource ketika penggunaannya diakses lebih dari satu aplikasa (task), namun berbeda dengan percobaan sebelumnya dalam manajemen ini tidak menggunakan resources. Dalam program ini merupakan program aplikasi untuk menjalankan dua buah task dengan prioritas yang sama. Task berupa aplikasi mencetak nama task ke ‘Serial Monitor’ dan menunggu untuk sementara waktu kemudian mencetak lagi dan seterusnya

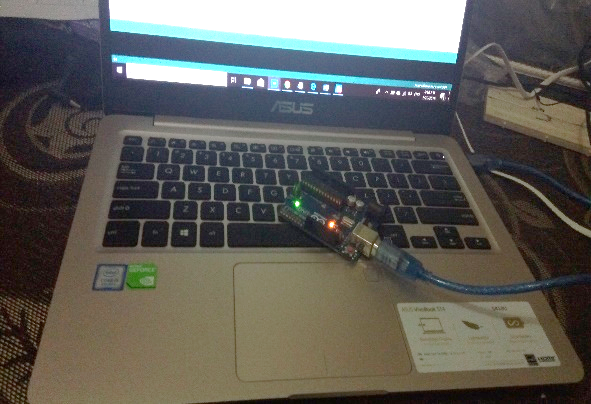
1. HASIL PRAKTIKUM

D.1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN



D.1.2. FOTO RANGKAIAN





D.1.3. SCRIPT PROGRAM



1. ANALISA

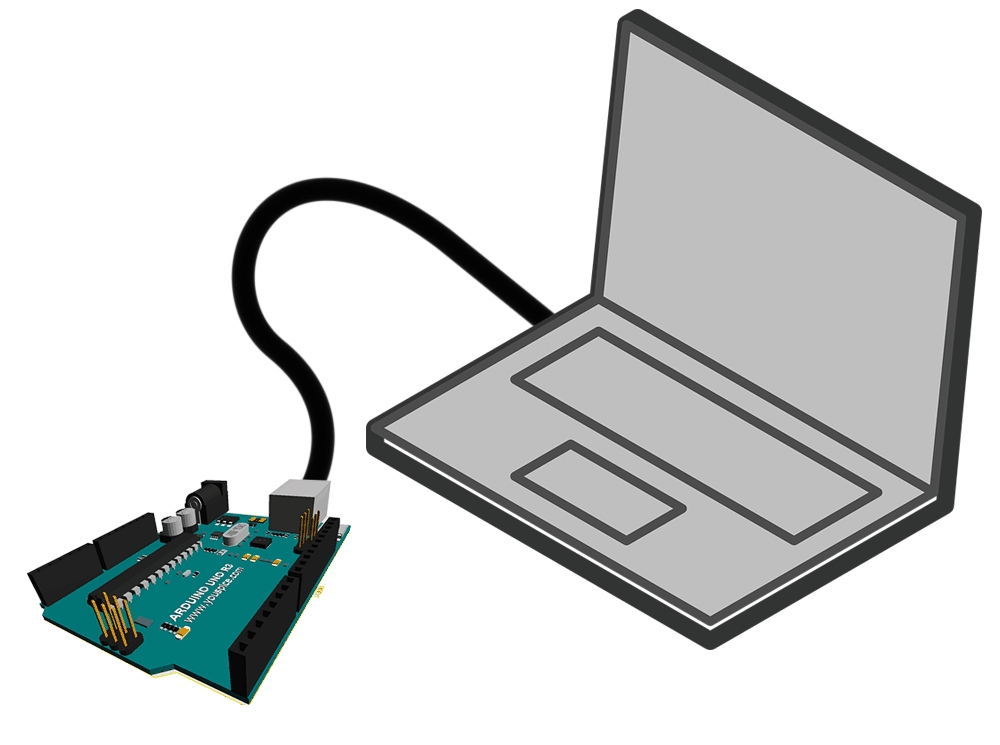
E.1. PERCOBAAN 1

Mengontrol task dengan menggunakan ‘vTaskDelay’ agar task dengan prioritas paling tinggi tidak mendominasi sistem operasi. Terdapat perintah ‘vTaskDelay( 1000 / portTICK\_PERIOD\_MS );’ sekarang dua buah task dengan prioritas yang sama dapat dieksekusi semua.

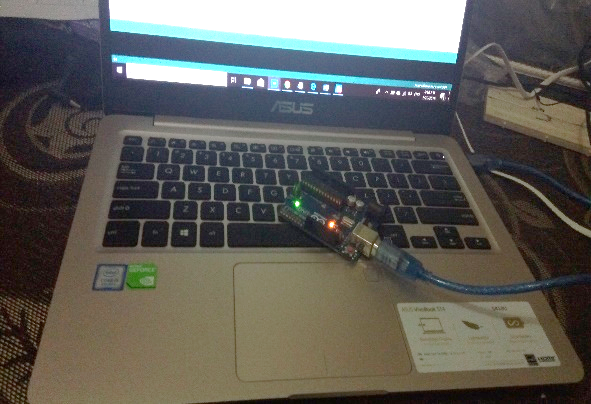
Perbedaan terletak pada cara menyusun program aplikasinya. Dalam contoh ini dua buah task tersebut juga memiliki nomor prioritas yang sama.

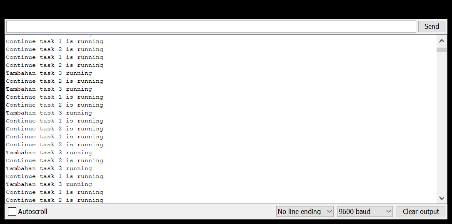
1. TUGAS

F.1.1. SKEMA RANGKAIAN



F.1.2. FOTO RANGKAIAN





F.1.3. SCRIPT PROGRAM



F.1.4. ANALISA

Penggunaan prioritas yang sama sehingga jalannya program berdasarkan tata cara penyusunan, vPeriodicTask digunakan untuk menampilkan Taskname. Mengontrol task dengan menggunakan ‘vTaskDelay’ agar task dengan prioritas paling tinggi tidak mendominasi sistem operasi. Terdapat perintah ‘vTaskDelay( 1000 / portTICK\_PERIOD\_MS )

G.KESIMPULAN

* + 1. Penggunaan prioritas yang sama sehingga jalannya program berdasarkan tata cara penyusunan.
    2. vPeriodicTask digunakan untuk menampilkan Taskname.
    3. Mengontrol task dengan menggunakan ‘vTaskDelay’ agar task dengan prioritas paling tinggi tidak mendominasi sistem operasi. Terdapat perintah ‘vTaskDelay( 1000 / portTICK\_PERIOD\_MS )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

MODUL 4

MANAJEMEN TASK

1. TUJUAN
2. Peserta mengerti cara membagi resources dalam Free-RTOS
3. Peserta mengerti kegunaan manajemen resources dalam sebuah program.
4. ALAT DAN BAHAN
5. Arduino
6. LED
7. Project Board
8. Kabel Jumper
9. DASAR TEORI

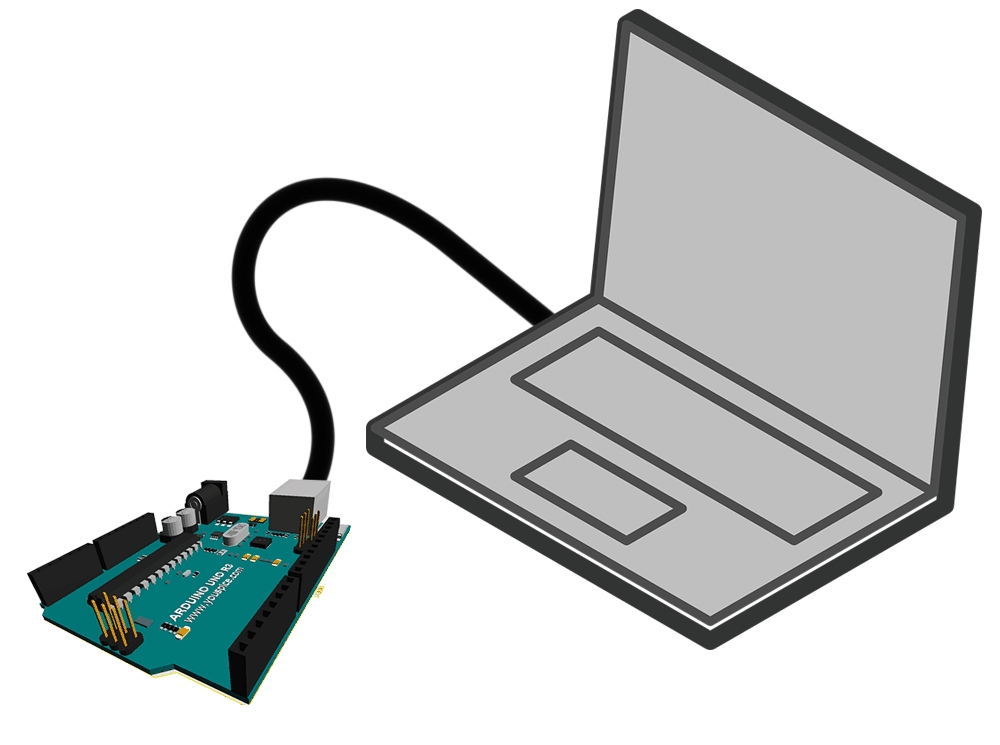
Pada Aktifitas lab ini telah dilakukan berbagai percobaan yang berkaitan dengan menejemen task. Pengendalian eksekusi task dapat dilakukan dengan cara mengatur nomor prioritas atau mengatur delay, atau dengan logika tertentu untuk meng-create dan menghapus task pada saat sistem sudah berjalan. IDLE TIME dapat dimanfaatkan untuk melakukan sesuatu yang membuat sistem menjadi lebih handal. Misalnya untuk mengatur mode sistem dalam POWER SAVE.

Contoh kasus, jika ada dua aplikasi yang sama-sama menggunakan Serial Monitor untuk menampilkan hasil pembacaan, maka diperlukan Semaphore (mutex) untuk mengatur akses.

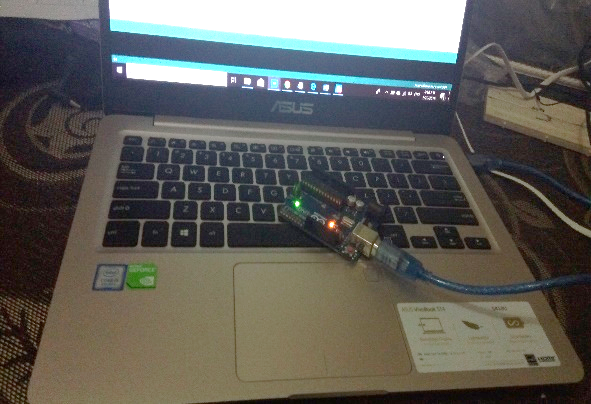
D.HASIL PRAKTIKUM

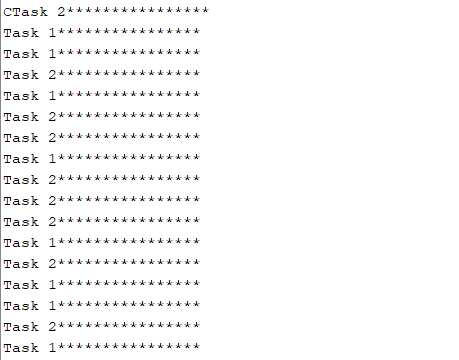
D.1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN

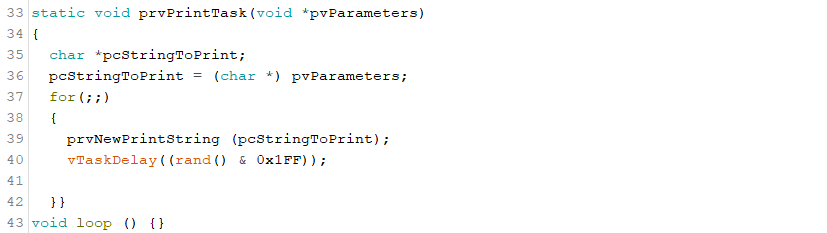


D.1.2. FOTO RANGKAIAN





D.1.3. SCRIPT PROGRAM

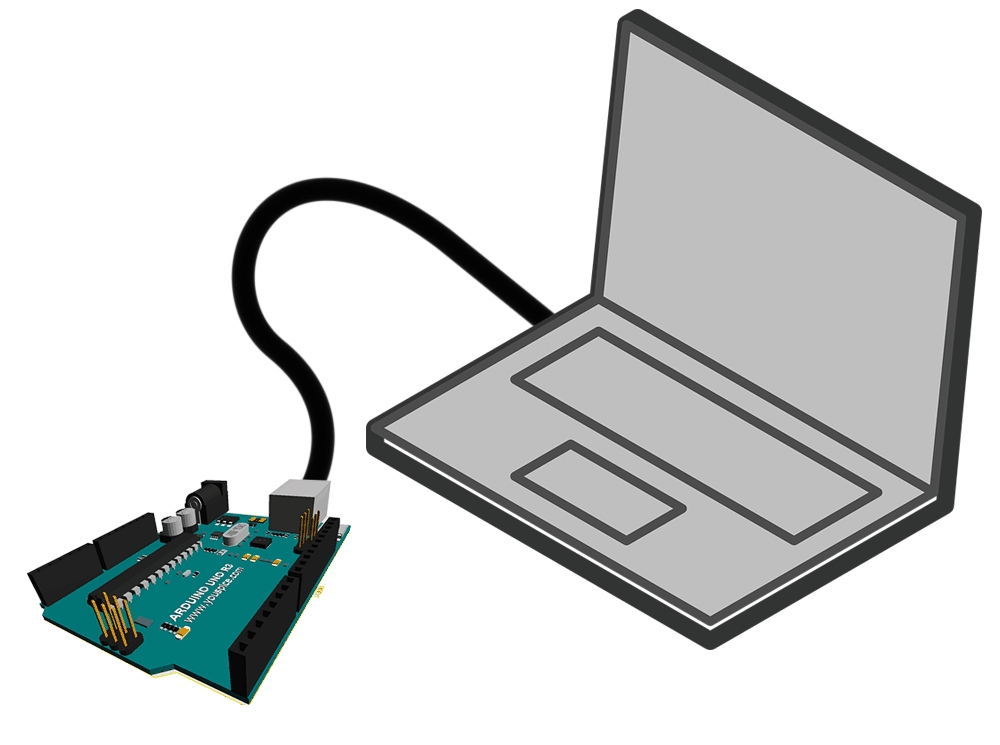


E. ANALISA

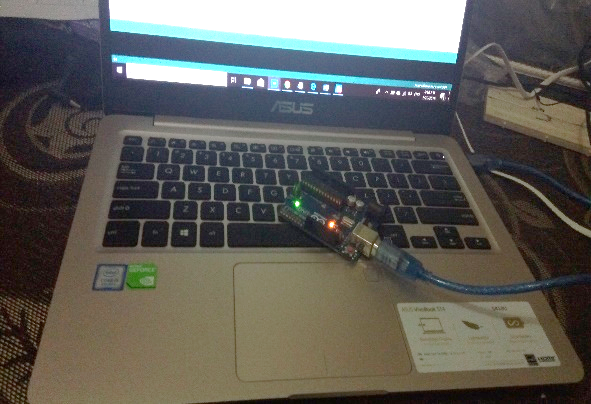
Semaphore (mutex) untuk mengatur akses, SemaphoreHandle\_t xSerialSemaphore; deklarasi variabel pengendali semaphore/mutex untuk mengatur penggunaan resource serial monitor. Hal ini hampir sama dengan percobaan pada modul 2 yaitu keduanya menggunakan fungsi Semaphore, yang berbeda dalam semaphore ini tidak menggunakan counting, karena menggunakan angka prioritas.

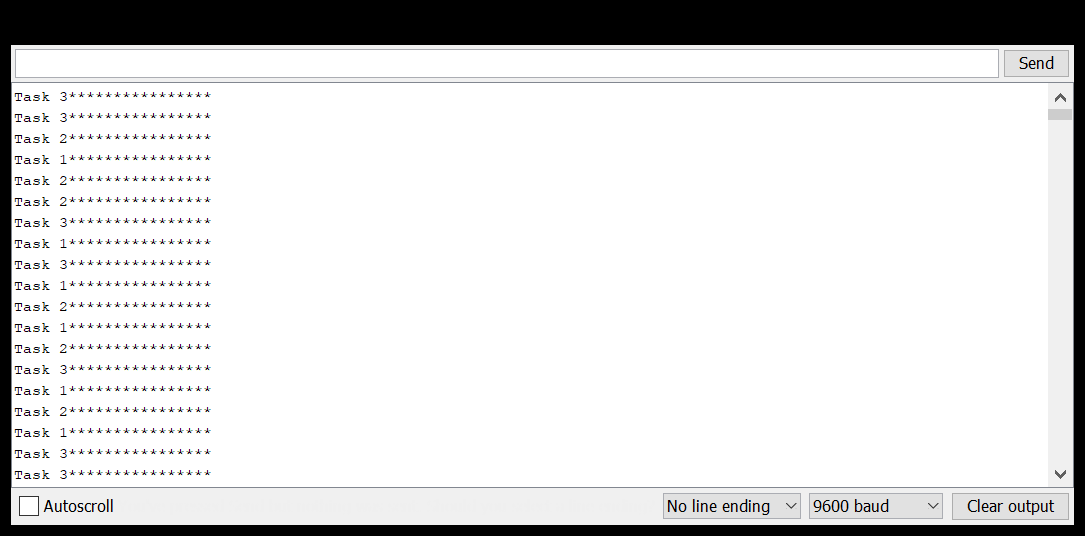
F. TUGAS

F.1.1. SKEMA RANGKAIAN

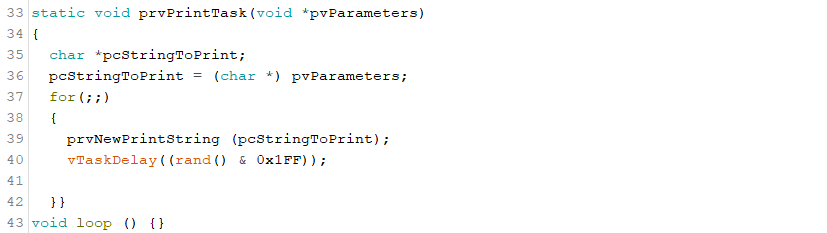
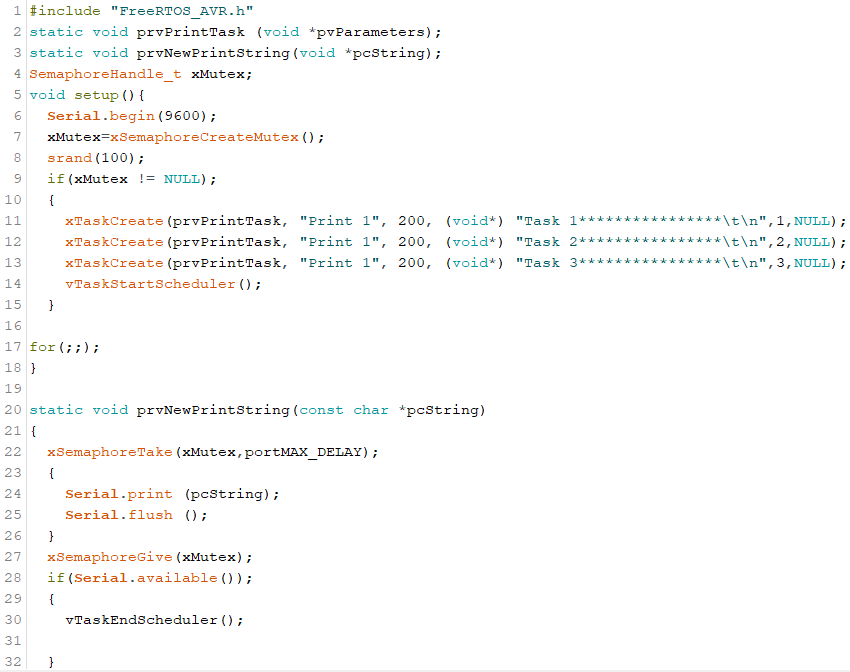


F.1.2. FOTO RANGKAIAN





F.1.3. SCRIPT PROGRAM



G. KESIMPULAN

1. Semaphore (mutex) untuk mengatur akses, SemaphoreHandle\_t xSerialSemaphore; deklarasi variabel pengendali semaphore/mutex untuk mengatur penggunaan resource serial monitor

2. Pengendalian eksekusi task dapat dilakukan dengan cara mengatur nomor prioritas atau mengatur delay, atau dengan logika tertentu untuk meng-create dan menghapus task pada saat sistem sudah berjalan.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

PROJECT 3

FLEXIBLE AND TIMMING ON 3TASK

1. TUJUAN
2. Peserta mengerti cara membagi resources dalam Free-RTOS
3. Peserta mengerti kegunaan manajemen resources dalam sebuah program.
4. ALAT DAN BAHAN
5. Arduino
6. LED
7. Project Board
8. Kabel Jumper
9. DASAR TEORI

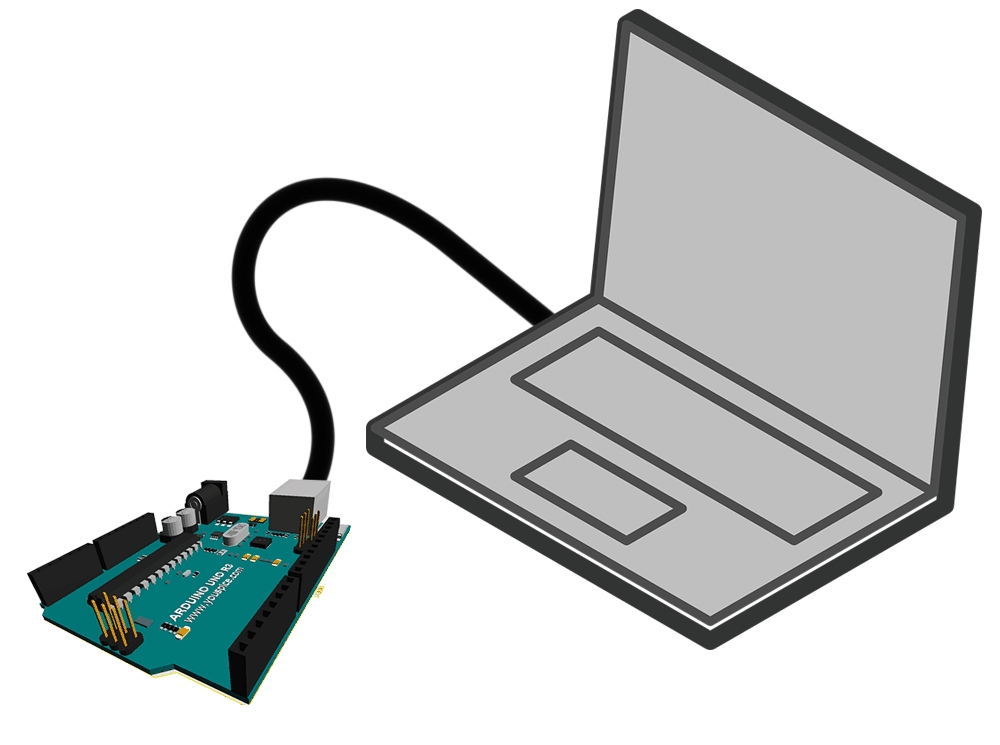
Jika ada resources (variable, hardware) yang akan diakses lebih dari satu aplikasi (task) maka diperlukan manajemen resource, agar tidak terjadi konflik antar aplikasi. Dalam sebuah mikrokontroler yang menjadi pusat pengolahan maka sangatlah penting kegiatan manajemen resource.

FreeRTOS telah menyediakan sebuah fitur untuk mengatur sebuah resource ketika penggunaannya diakses lebih dari satu aplikasa (task), namun berbeda dengan percobaan sebelumnya dalam manajemen ini tidak menggunakan resources. Dalam program ini merupakan program aplikasi untuk menjalankan dua buah task dengan prioritas yang sama. Task berupa aplikasi mencetak nama task ke ‘Serial Monitor’ dan menunggu untuk sementara waktu kemudian mencetak lagi dan seterusnya

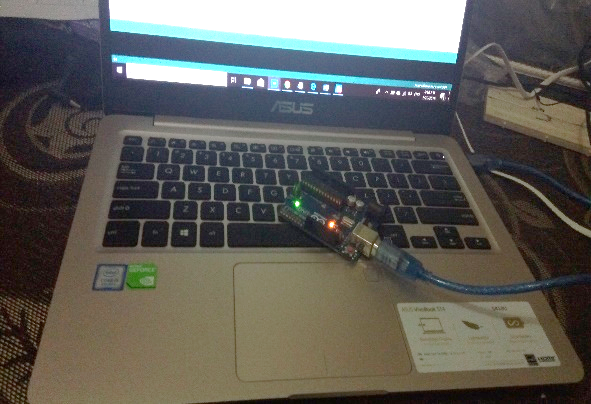
1. HASIL PRAKTIKUM

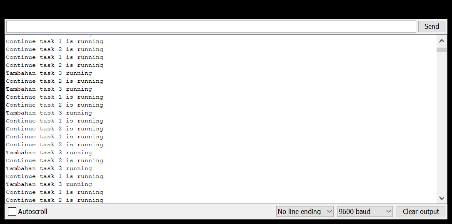
D.1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN



D.1.2. FOTO RANGKAIAN





D.1.3. SCRIPT PROGRAM



E. ANALISA

Penggunaan prioritas yang sama sehingga jalannya program berdasarkan tata cara penyusunan, vPeriodicTask digunakan untuk menampilkan Taskname. Mengontrol task dengan menggunakan ‘vTaskDelay’ agar task dengan prioritas paling tinggi tidak mendominasi sistem operasi. Terdapat perintah ‘vTaskDelay( 1000 / portTICK\_PERIOD\_MS )

F. .KESIMPULAN

* + 1. Penggunaan prioritas yang sama sehingga jalannya program berdasarkan tata cara penyusunan.
    2. vPeriodicTask digunakan untuk menampilkan Taskname.
    3. Mengontrol task dengan menggunakan ‘vTaskDelay’ agar task dengan prioritas paling tinggi tidak mendominasi sistem operasi. Terdapat perintah ‘vTaskDelay( 1000 / portTICK\_PERIOD\_MS )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

PROJECT 2

SEMAPHORE 3 TASK

1. TUJUAN
2. Peserta mengerti cara membangun program semaphore untuk Free-RTOS
3. Peserta dapat memahami Memahami cara kerja semaphore dalam FreeRTOS untuk sebuah kasus sederhana.
4. Peserta mengerti kegunaan semaphore dalam sebuah program.
5. ALAT DAN BAHAN
6. Arduino
7. LED
8. Project Board
9. Kabel Jumper
10. DASAR TEORI

Jika ada resources (variable, hardware) yang akan diakses lebih dari satu aplikasi (task) maka diperlukan manajemen resource, agar tidak terjadi konflik antar aplikasi. Dalam sebuah mikrokontroler yang menjadi pusat pengolahan maka sangatlah penting kegiatan manajemen resource.

FreeRTOS telah menyediakan sebuah fitur untuk mengatur sebuah resource ketika penggunaannya diakses lebih dari satu aplikasa (task), salah satunya menggunakan Semaphore. Dalam skenario penggunaannya, nilai hitungan semaphore mewakili jumlah sumber daya yang tersedia.

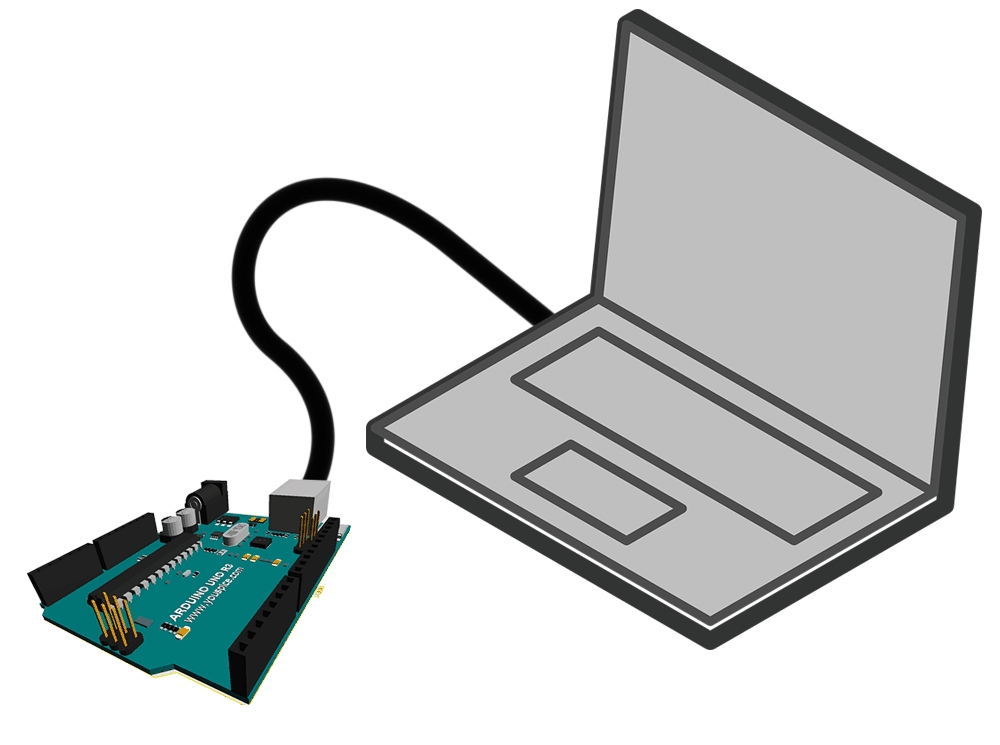
Untuk mendapatkan kontrol sumber daya, tugas pertama-tama harus berhasil "mengambil" semaphore. Tindakan “mengambil” semaphore akan menurunkan nilai hitungan semaphore. Ketika nilai hitungan mencapai nol, tidak ada lagi sumber daya yang tersedia, dan upaya lebih lanjut untuk 'mengambil' semaphore akan gagal. Ketika tugas selesai dengan sumber daya, tugas itu harus 'memberi' semaphore. Tindakan 'memberi' semaphore akan meningkatkan nilai hitungan semaphore, menunjukkan bahwa sumber daya tersedia, dan memungkinkan upaya masa depan untuk 'mengambil' semaphore agar berhasil.

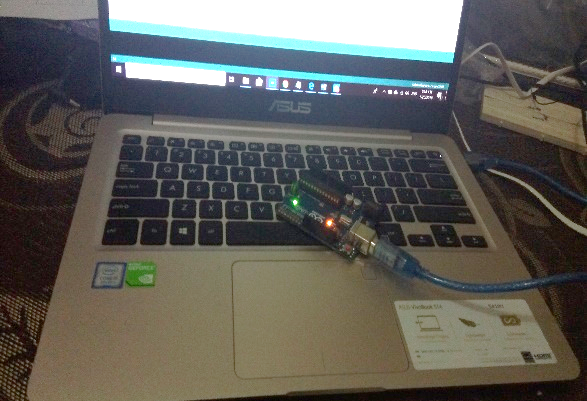
Semaphore yang dibuat untuk mengelola sumber daya harus dibuat dengan nilai hitungan awal sama dengan jumlah sumber daya yang tersedia.

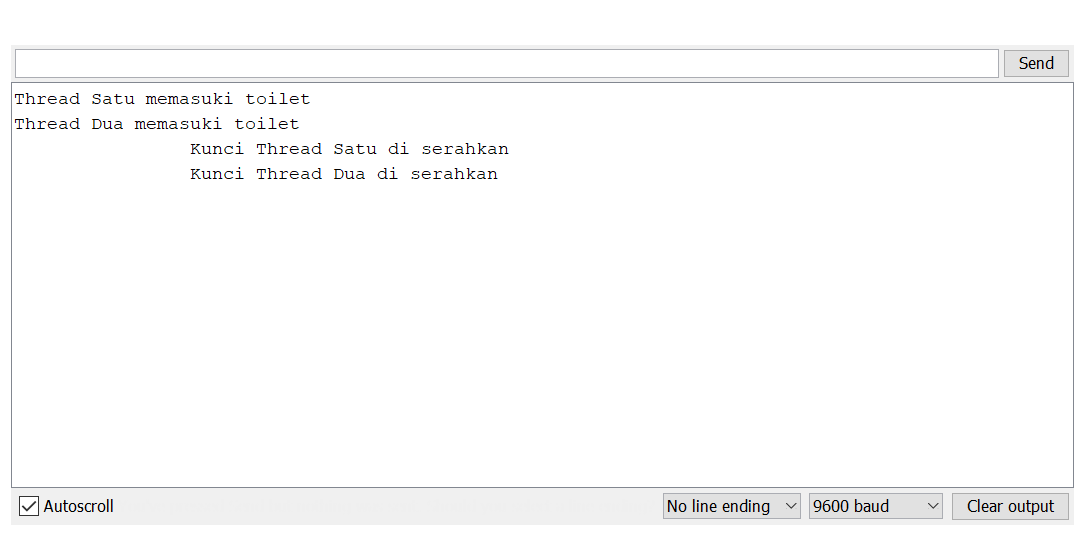
D.HASIL PRAKTIKUM

D. 1. PERCOBAAN 1

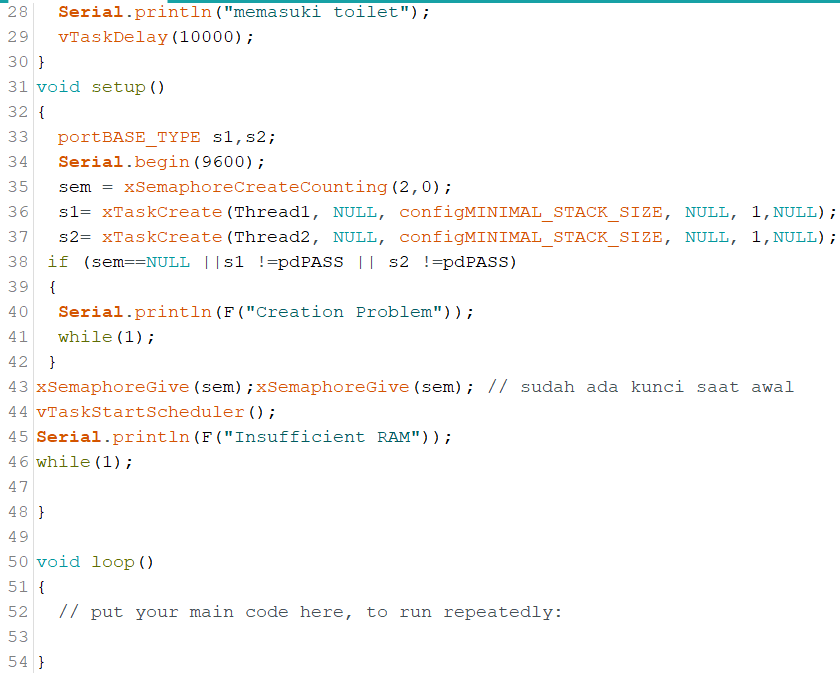
D.1.1. SKEMA RANGKAIAN



 D.1.2. FOTO RANGKAIAN



D.1.3. SCRIPT PROGRAM



E. ANALISA

Dalam kasus ini tidak jauh beda dengan percobaan, hanya saja kita membuat bahwa kunci yang tersedia di box kunci ada dua, sehingga thread satu dan thread dua dapat menggunakan toilet secara bersamaan tanpa kunci dikembalikan terlebih dahulu. Mengembalikan kunci kita non akifkan dengan menghapus xSemaphoreGive(sem) pada masing masing task. Namun menambah xSemaphoreGive(sem) agar jumlah kunci dalam box sesuai jumlah kunci yang tersedia yaiu pada sem = xSemaphoreCreateCounting(2,0).

F. KESIMPULAN

1. Instruksi Fungsi xSemaphoreGive(sem) pada Void Setup ini artinya bahwa sudah ada kunci yang tersedia misalnya dalam box kunci. Sedangkan sem = xSemaphoreCreateCounting(1,0) artinya jumlah kunci.
2. Mengembalikan kunci kita non akifkan dengan menghapus xSemaphoreGive(sem) pada masing masing task. Namun menambah xSemaphoreGive(sem) agar jumlah kunci dalam box sesuai jumlah kunci yang tersedia yaiu pada sem = xSemaphoreCreateCounting(2,0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kelompok |  |  | | ACC |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

PROJECT 4

MANAJEMEN 3 TASK

1. TUJUAN
2. Peserta mengerti cara membagi resources dalam Free-RTOS
3. Peserta mengerti kegunaan manajemen resources dalam sebuah program.
4. ALAT DAN BAHAN
5. Arduino
6. LED
7. Project Board
8. Kabel Jumper
9. DASAR TEORI

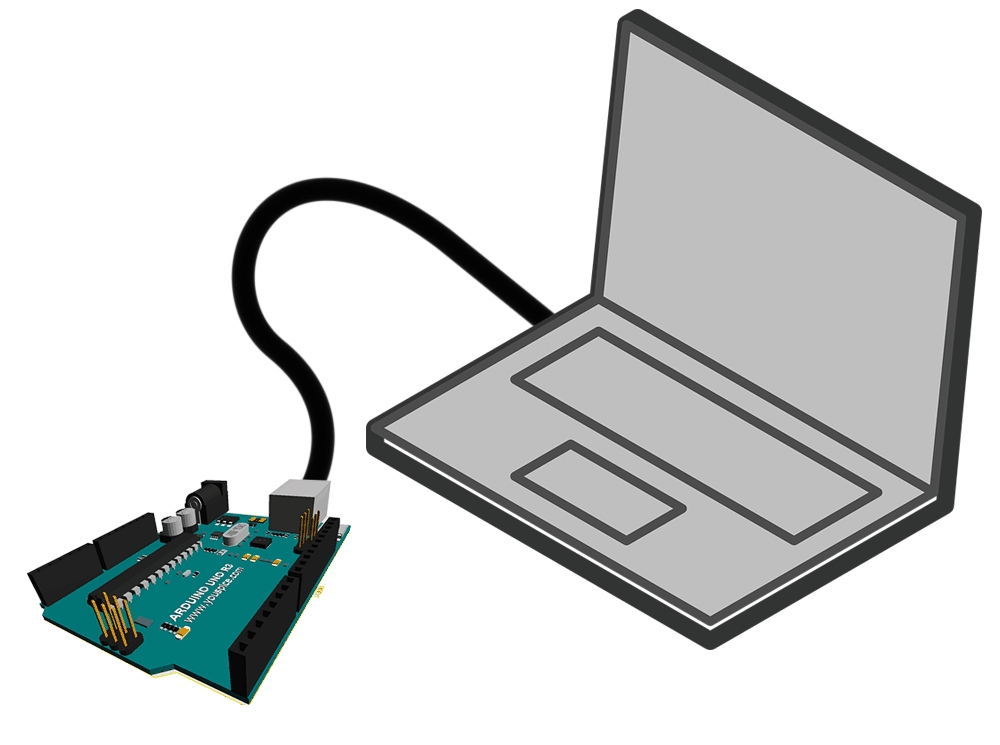
Pada Aktifitas lab ini telah dilakukan berbagai percobaan yang berkaitan dengan menejemen task. Pengendalian eksekusi task dapat dilakukan dengan cara mengatur nomor prioritas atau mengatur delay, atau dengan logika tertentu untuk meng-create dan menghapus task pada saat sistem sudah berjalan. IDLE TIME dapat dimanfaatkan untuk melakukan sesuatu yang membuat sistem menjadi lebih handal. Misalnya untuk mengatur mode sistem dalam POWER SAVE.

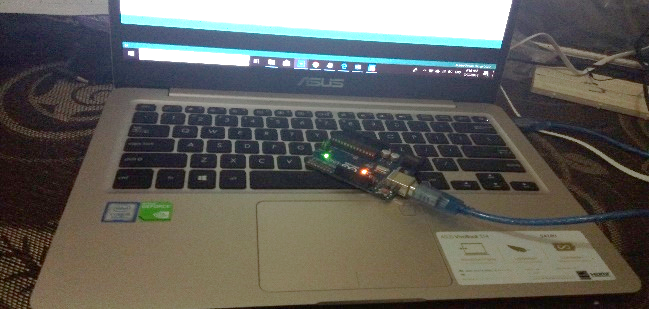
Contoh kasus, jika ada dua aplikasi yang sama-sama menggunakan Serial Monitor untuk menampilkan hasil pembacaan, maka diperlukan Semaphore (mutex) untuk mengatur akses.

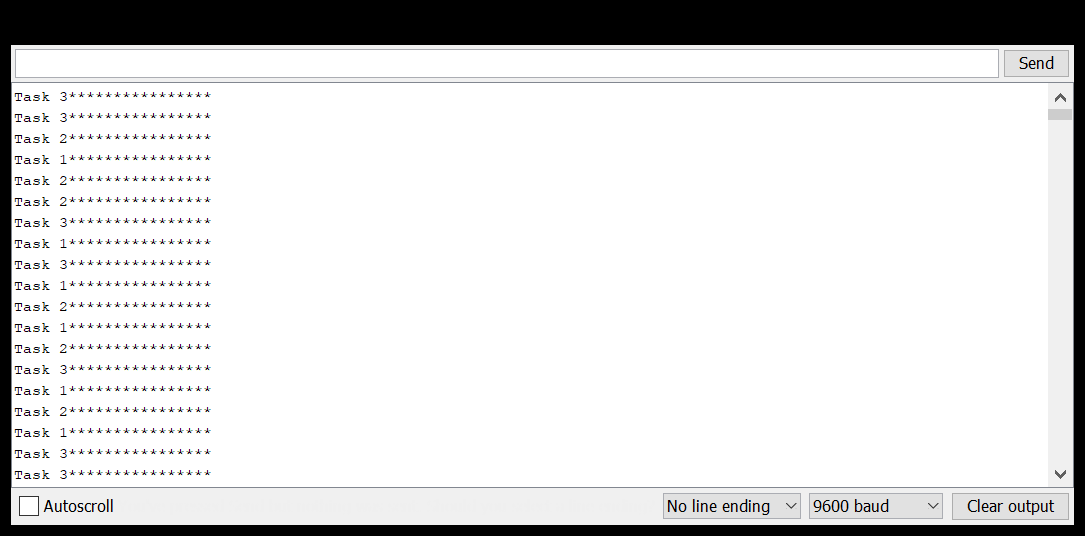
D.HASIL PRAKTIKUM

D.1. PERCOBAAN 1

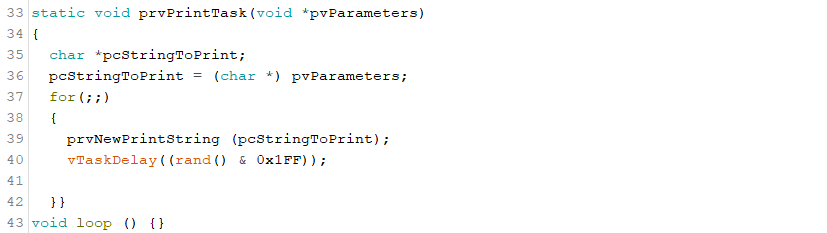
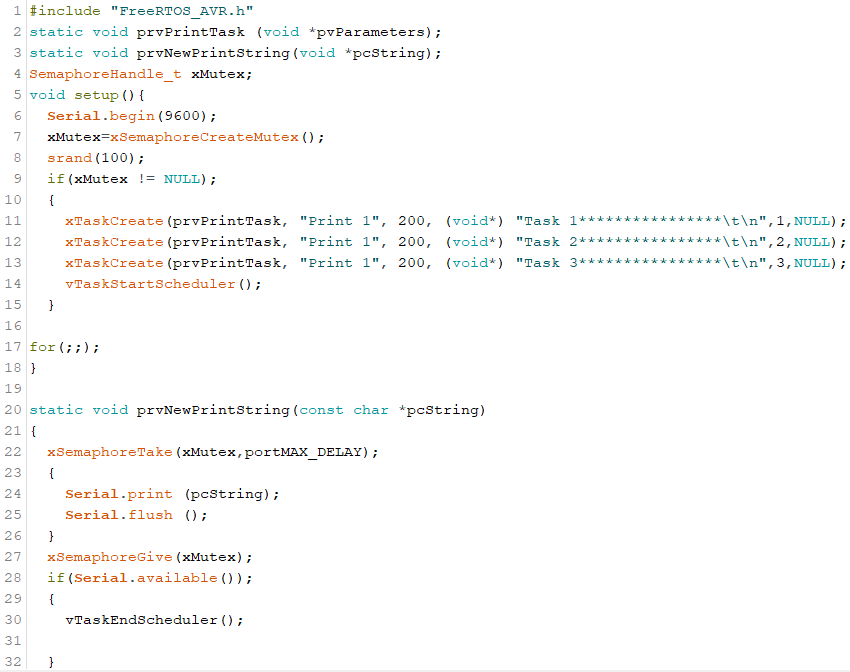
D.1.1. SKEMA RANGKAIAN



D.1.2. FOTO RANGKAIAN



D.1.3. SCRIPT PROGRAM



E. KESIMPULAN

1. Semaphore (mutex) untuk mengatur akses, SemaphoreHandle\_t xSerialSemaphore; deklarasi variabel pengendali semaphore/mutex untuk mengatur penggunaan resource serial monitor

2. Pengendalian eksekusi task dapat dilakukan dengan cara mengatur nomor prioritas atau mengatur delay, atau dengan logika tertentu untuk meng-create dan menghapus task pada saat sistem sudah berjalan.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tanggal Praktikum |  | ` | |  |
| Nama | 1 | Aldi Majid | D400150121 |
|  | 2 | Fizal Chofarul | D400150152 |
|  | 3 | Usman Bimantoro | D400150104 | Tgl ACC : |
|  |  |  |  | Revisi Tgl : |

PROJECT TUGAS BESAR

SEMAPHORE DALAM PENGUKUR KANDUNGAN UDARA DAN LDR

1. TUJUAN
2. Peserta mengerti cara membangun program semaphore untuk Free-RTOS
3. Peserta dapat memahami Memahami cara kerja semaphore dalam FreeRTOS untuk sebuah kasus sederhana.
4. Peserta mengerti kegunaan semaphore dalam sebuah program.
5. ALAT DAN BAHAN
6. Arduino
7. LED
8. Project Board
9. Kabel Jumper
10. DASAR TEORI

Jika ada resources (variable, hardware) yang akan diakses lebih dari satu aplikasi (task) maka diperlukan manajemen resource, agar tidak terjadi konflik antar aplikasi. Dalam sebuah mikrokontroler yang menjadi pusat pengolahan maka sangatlah penting kegiatan manajemen resource.

FreeRTOS telah menyediakan sebuah fitur untuk mengatur sebuah resource ketika penggunaannya diakses lebih dari satu aplikasa (task), salah satunya menggunakan Semaphore. Dalam skenario penggunaannya, nilai hitungan semaphore mewakili jumlah sumber daya yang tersedia.

Untuk mendapatkan kontrol sumber daya, tugas pertama-tama harus berhasil "mengambil" semaphore. Tindakan “mengambil” semaphore akan menurunkan nilai hitungan semaphore. Ketika nilai hitungan mencapai nol, tidak ada lagi sumber daya yang tersedia, dan upaya lebih lanjut untuk 'mengambil' semaphore akan gagal. Ketika tugas selesai dengan sumber daya, tugas itu harus 'memberi' semaphore. Tindakan 'memberi' semaphore akan meningkatkan nilai hitungan semaphore, menunjukkan bahwa sumber daya tersedia, dan memungkinkan upaya masa depan untuk 'mengambil' semaphore agar berhasil.

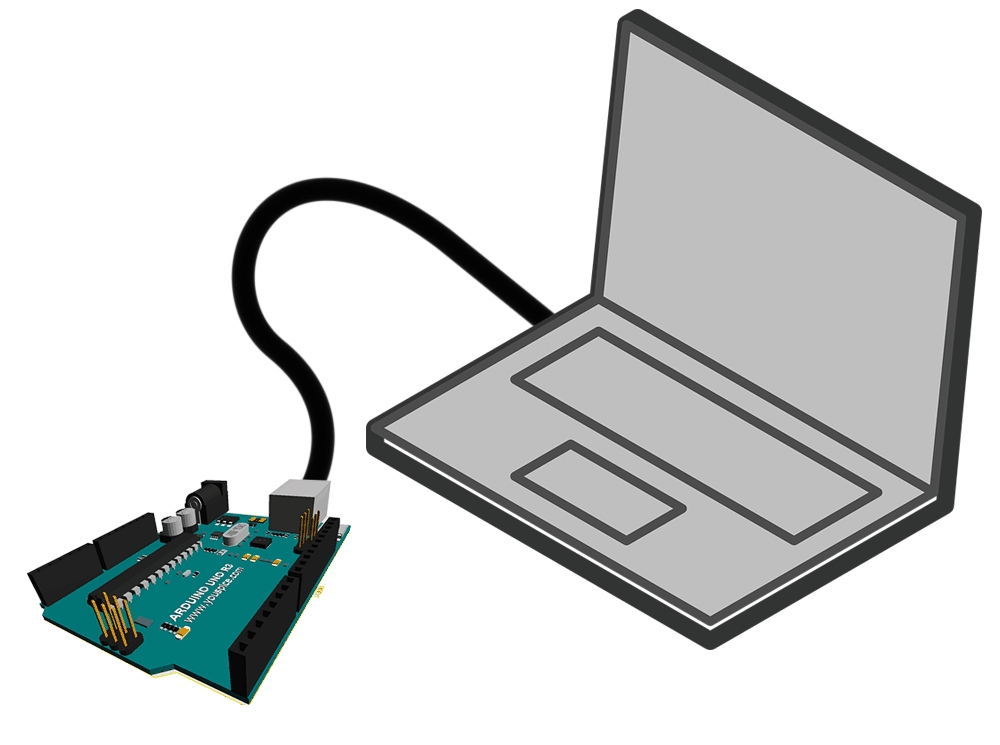
Semaphore yang dibuat untuk mengelola sumber daya harus dibuat dengan nilai hitungan awal sama dengan jumlah sumber daya yang tersedia.

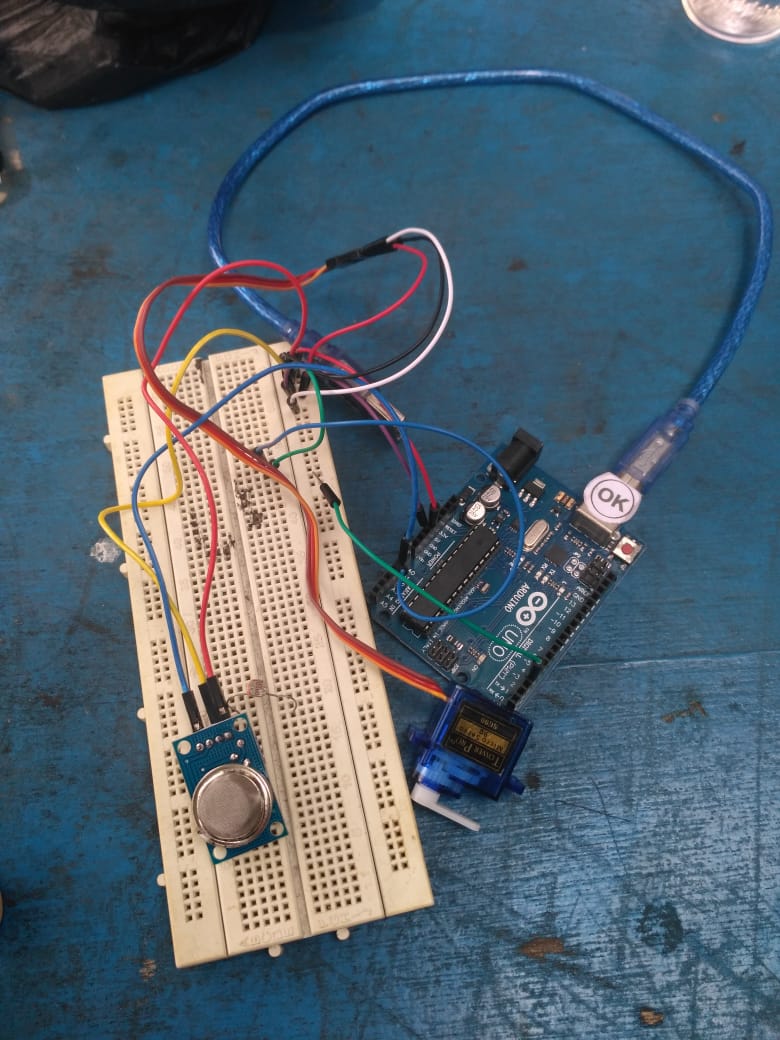
Semaphore dalam percobaan ini membagi peran dua task yaitu sensor MQ135 dan counting.

D.HASIL PRAKTIKUM

D. 1. PERCOBAAN 1

D.1.1. SKEMA RANGKAIAN



 D.1.2. FOTO RANGKAIAN

D.1.3. SCRIPT PROGRAm

#define LEDPIN1 13

#include <FreeRTOS\_AVR.h>

#include<Servo.h>

#include "MQ135.h"

#include "utility/FreeRTOS.h"

#include "utility/task.h"

#include "utility/queue.h"

#include "utility/semphr.h"

void TaskAnalogRead( void \*pvParameters );

void my\_task(void \*pvParameters);

int sensorValue;

SemaphoreHandle\_t sem;

static void AnalogRead(void\* arg){

while (1){

vTaskDelay(100);

if (xSemaphoreTake (sem, portMAX\_DELAY)){ //ambil kunci

xSemaphoreGive(sem);

Serial.println("-------");

sensorValue=analogRead(A0);

Serial.print("PPM= ");

Serial.println(sensorValue);

kamar();

ldr();

Serial.println("\t\tMembaca Sensor dulu");

vTaskDelay(100);

}

}

}

static void hitung(void\* arg){

while (1){

for(int i=0;i<=100;i++){

vTaskDelay(1000);

if (xSemaphoreTake (sem, portMAX\_DELAY)){

xSemaphoreGive(sem);

Serial.println("HITUNG;");

Serial.print(i);

vTaskDelay(100);

kamar();

servo();

ldr();

Serial.println("\t\tLanjut ke lain");

vTaskDelay(100);

}

}

}

}

void kamar(){

Serial.println("tunggu proses selesai");

vTaskDelay(10000);

}

void servo(){

Servo myservo; // create servo object to control a servo

// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

myservo.attach(3); // attaches the servo on pin 9 to the servo object

for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees

// in steps of 1 degree

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

void ldr (){

int LDR\_Pin = A1; //analog pin 0

Serial.begin(9600);

int LDRReading = analogRead(LDR\_Pin);

Serial.println(LDRReading);

delay(250);

}

void setup()

{

portBASE\_TYPE s1,s2;

Serial.begin(9600);

sem = xSemaphoreCreateCounting(1,0);

s1= xTaskCreate(AnalogRead, NULL, configMINIMAL\_STACK\_SIZE, NULL, 1,NULL);

s2= xTaskCreate(hitung, NULL, configMINIMAL\_STACK\_SIZE, NULL, 2,NULL);

if (sem==NULL ||s1 !=pdPASS || s2 !=pdPASS)

{

Serial.println(F("Creation Problem"));

while(1);

}

xSemaphoreGive(sem); // sudah ada kunci saat awal

vTaskStartScheduler();

Serial.println(F("Insufficient RAM"));

while(1);

}

void loop()

{

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

}

E. ANALISA

Semaphore dalam percobaan ini membagi peran dua task yaitu sensor MQ135 dan counting. Untuk mendapatkan kontrol sumber daya, tugas pertama-tama harus berhasil "mengambil" semaphore. Tindakan “mengambil” semaphore akan menurunkan nilai hitungan semaphore.

F. KESIMPULAN

1. Pengendalian eksekusi task dapat dilakukan dengan cara mengatur nomor prioritas atau mengatur delay, atau dengan logika tertentu untuk meng-create dan menghapus task pada saat sistem sudah berjalan.