MODUL 4
OPERATING SYSTEM

Ahmad Aziz (13220034)
Asisten: Theodore Maximillan Jonathan (13219021)
Tanggal Percobaan: 31/3/2023
EL3215 - Praktikum Sistem Mikroprosesor
Laboratorium Sistem Kendali dan Komputer - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Abstrak *Praktikum sistem mikroprosesor modul ini adalah percobaan tentang operating system. Pada praktikum ini aka nada 2 bagian percobaan yang akan dilakukan yaitu percobaan MPU6050 dan RTOS.*

Kata kunci: ESP32, RTOS, MPU6050, kernel.

1. PENDAHULUAN

Operating system (OS) pada mikroprosesor berperan sebagai pengatur dan pengelola sumber daya sistem, seperti memori, CPU, perangkat masukan/keluaran, dan jaringan. OS membantu aplikasi dalam mengatur sumber daya dan menangani interaksi dengan perangkat keras, sehingga aplikasi dapat dijalankan dengan efisien dan terstruktur.

Namun, OS pada mikroprosesor biasanya tidak didesain khusus untuk menjalankan aplikasi real-time atau sistem yang memerlukan waktu respon yang sangat cepat. Oleh karena itu, OS pada mikroprosesor seringkali disebut sebagai General Purpose Operating System (GPOS) atau Desktop Operating System (DOS).

Real- Time Operating System (biasa di sebut RTOS) adalah sebuah Operating System (OS) yang digunakan untuk memenuhi kebutuhan aplikasi secara Real Time pada Embedded Device yang memproses data secara langsung tanpa ada nya penundaan (Buffer). RTOS dirancang khusus untuk memenuhi kebutuhan tersebut dan menjamin bahwa aplikasi real-time dapat dijalankan dengan waktu respon yang terukur dan sesuai dengan kebutuhan.

Sementara OS pada mikroprosesor biasanya memiliki fitur-fitur umum seperti manajemen memori dan file, pengaturan tugas, serta sistem keamanan dan jaringan, RTOS memperluas kemampuan ini dengan menambahkan fitur-fitur yang dibutuhkan untuk mengoptimalkan waktu respon dan menjamin determinisme. Sebagai contoh, RTOS dapat memprioritaskan tugas-tugas yang penting, memungkinkan aplikasi untuk mengakses sumber daya dengan waktu yang konsisten, dan menangani kesalahan sistem dengan lebih baik.

Pada perobaan praktikum modul ini dilakukan percobaan dengan RTOS dan juga modul inertial mesarurement unit (IMU) MPU6050. Percobaan ini bertujuan sebagai berikut:

1. Praktikan memahami penggunaan MPU6050 yang memanfaatkan protokol komunikasi I2C.
2. Praktikan mampu membuat produk sederhana dengan memanfaatkan Operating system.

2. STUDI PUSTAKA

2.1 REAL- TIME OPERATING SYSTEM (RTOS)

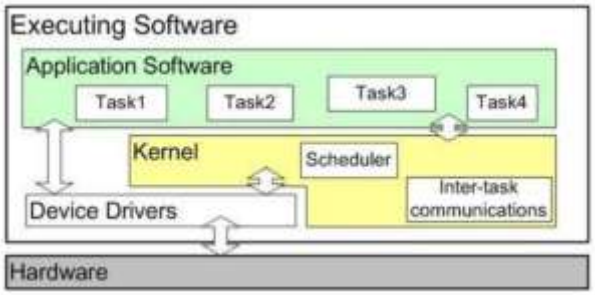
Real- Time Operating System (biasa di sebut RTOS) adalah sebuah Operating System (OS) yang digunakan untuk memenuhi kebutuhan aplikasi secara Real Time pada Embedded Device yang memproses data secara langsung tanpa ada nya penundaan (Buffer). Real Time karena system ini hamper bekerja setiap saat dimana ia dibutuhkan saat itu juga. Salah satu kelebihan Operating System RTOS adalah kemampuan nya untuk melakukan kerja secara konsisten baik secara waktu yang ia butuhkan maupun secara task aplikasi yang mampu ia kerjakan.

RTOS dibutuhkan karena pada system Embedded karena biasa nya pada system Embedded , digunakan sebuah mikrokontroler dengan prosesor tunggal (Single Processor), sehingga pada pekerjaan Embedded System yang membutuhkan system secara Real Time dan melakukan lebih dari satu pekerjaan, dibutuhkan sebuah Operating System yang dapat melakukan penjadwalan (Scheduling) beberapa pekerjaan sehingga dapat dilakukan dalam sebuah prosesor tunggal, dan mudah dimodifikasi untuk melakukan berbagai pekerjaan. Karakter dasar dari Operating System RTOS adalah sebuah sistem yang mempunyai beberapa konsekuensi yang akan berpengaruh

pada sistem apabila deadline (batas akhir waktu pelaksanaan sebuah pekerjaan) 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 88 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 system 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.

Diagram arsitektur cara kerja RTOS dapat dilihat sebagai berikut:



Terdapat 2 komponen utama dalam RTOS, yaitu Tugas (atau Task) dan Kernel (atau Scheduler).

2.1.1 TUGAS (TASK)

Tugas (atau biasa di sebut Task) adalah 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 yaitu: (i) Running, merupakan keadaan di mana sebuah task dengan prioritas tertinggi berjalan, (ii) Ready, merupakan keadaan yang dialami sebuah task jika terdapat sebuah task lain sedang running dan task yang berada pada State ready akan melanjutkan pengerjaan task yang sempat tertunda oleh task yang lebih tinggi prioritasnya. (iii) 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.

2.1.2 KERNEL

Kernel merupakan salah satu bagian dari sistem multitasking yang mempunyai fungsi sebagai manajemen dari seluruh task, mengatur komunikasi tiap task dan yang terpenting adalah mengatur pewaktuan (Clock) untuk CPU sehingga tidak terjadi tabrakan (Crash) Task pada CPU. Terdapat 2 jenis Kernel:

a. Non- Preempetive Kernel

   Non-preemptive scheduling biasa dikenal dengan nama lain cooperative multitasking, di mana task bekerja sama satu sama lain untuk berbagi CPU. ISR bias membuat sebuah task dengan prioritas tertinggi menjadi siap untuk dieksekusi, tetapi kemudian ISR akan kembali ke task yang sebelumnya mendapat interupsi. Task yang sudah siap tadi akan berjalan apabila task yang mendapat interupsi tadi sudah selesai berjalan atau dengan kata lain task yang sudah selesai berjalan akan menyerahkan CPU kepada task dengan prioritas tertinggi.

b. Preempetive Kernel

   Preemptive kernel banyak digunakan untuk membuat aplikasi dengan RTOS. Hal ini karena preemptive kernel mempunyai respons yang lebih bagus daripada non-preemptive kernel. Dari gambar 2.3 dapat dijelaskan prinsip kerja dari preemptive kernel. Di mana task dengan prioritas tertinggi yang sudah siap dieksekusi akan langsung berjalan. Dan jika pada saat itu sedang ada task dengan prioritas yang lebih rendah berjalan maka task dengan prioritas rendah tersebut akan ditunda. Jadi dapat 90 disimpulkan bahwa preemptive kernel selalu mendahulukan

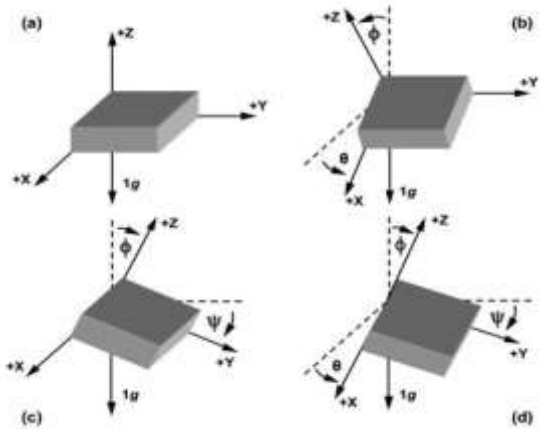
task dengan prioritas tertinggi yang siap untuk dieksekusi.

Clock tick merupakan interupsi special yang muncul secara periodik. Clock tick bias dianggap sebagai detak jantung dari sistem yang berfungsi sebagai dasar untuk menentukan timer pada sistem Real Time dengan RTOS. Waktu untuk tiap munculnya clock tick bisa ditentukan pada saat merancang sistem RTOS. Selain itu dalam penggunaan RTOS terdapat istilah Semaphore. Semaphore dalam sistem RTOS adalah penanda yang menandakan kapan suatu Task dapat dilakukan dan kapan suatu Task tidak dapat dilakukan. Terdapat beberapa jenis RTOS, seperti SafeRTOS dan FreeRTOS. Jenis Operating System RTOS yang paling sering digunakan adalah FreeRTOS, karena jenis RTOS yang bersifat Open Source, gratis dan mudah digunakan.

2.2 MPU6050

Sensor MPU6050 adalah sensor yang mampu membaca kemiringan sudut berdasarkan data dari sensor Accelerometer dan sensor Gyroscop. Sensor ini juga dilengkapi oleh sensor suhu yang dapat digunakan untuk mengukur suhu di keadaan sekitar. Jalur data yang digunakan pada sensor ini adalah jalur data I2C dan mampu berjalan pada tegangan sumber sebesar 3 sampai 5 V. Penampilan Sensor Accelerometer dan Gyroscope MPU6050 dapat dilihat sebagai berikut.

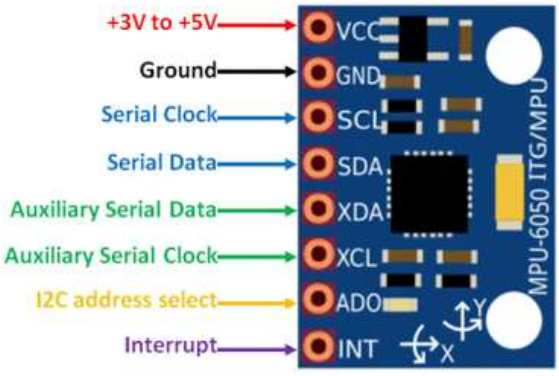
Sensor MPU6050 memiliki sensor Keseimbangan (Gyroscope) dan sensor kecepatan (Accelerometer) bersama dengan sensor suhu. Modul ini berukuran sangat kecil, memiliki konsumsi daya yang rendah, sangat akurat, toleransi guncangan yang tinggi, dan memiliki harga yang murah. Selain itu Sensor MPU6050 juga memiliki modul bawaan yang dapat digunakan dalam berbagai mikrokontroler yang sering digunakan, seperti Arduino dan ESP32. MPU6050 dapat dengan mudah dihubungkan dengan sensor lain seperti Magnetometer . Giroskop yang ada di Sensor MPU6050 dapat mendeteksi rotasi pada tiga sumbu yaitu sumbu- x , sumbu - y, dan sumbu- Z. Efek Coriolis menyebabkan getaran saat Gyros diputar di sekitar sumbu mana pun. Getaran ini ditangkap oleh kapasitor, yang kemudian sinyal yang dihasilkan kemudian diperkuat, didemodulasi dan di- Filter untuk menghasilkan tegangan yang sebanding dengan kecepatan sudut. Tegangan ini kemudian didigitalisasi menggunakan modul Analog to Digital Converter (ADC). Pergerakan pada sumbu- x , sumbu y dan sumbu- z yang dapat diukur oleh sensor MPU6050 dapat dilihat sebagai berikut.



Spesifikasi umum dari sensor MPU6050 dapat dilihat sebagai berikut:

Chip model	MPU-6050
Power supply	3–5 V
Communication protocol	I2C
Gyroscope range	±250, ±500, ±1000, ±2000° /s
Accelerometer range	±2, ±4, ±8, ±16g
16-bit AD converter/16-bit data output	

Diagram Pin Input dan Output dari sensor MPU6050 dapat dilihat sebagai berikut. Pada sensor MPU6050, terdapat 8 buah pin, yaitu sebuah pin sumber tegangan (VCC), pin Ground , pin Serial Clock (SCL) , pin Serial Data (SDA), pin Auxillary Serial Data (XDA), pin Auxillary Serial Clock (XCL), pin I2C Address Select (AD0), dan pin interupsi (INT). Diagram pin MPU6050 dan keterangan detail nya dapat dilihat sebagai berikut.



MPU6050 Pinout		
Pin#	Pin Name	Description
01	Vcc	This pin used for Supply Voltage. Its input voltage is +3 to +5V
02	GND	This pin use for ground
03	SCL	This pin is used for clock pulse for I2C compunction
04	SDA	This pin is used for transferring of data through I2C communication
05	Auxiliary Serial Data (XDA)	It can be used for other interfaced other I2C module with MPU6050.
06	Auxiliary Serial Clock (XCL)	It can also be used for other interfaced other I2C module with MPU6050.
07	AD0	If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address
08	Interrupt (int)	This pin is used to indicate that data is available for MCU to read.

3. METODOLOGI

Dalam percobaan pada modul ini, ada beberapa peralatan yang digunakan yaitu sebagai berikut:

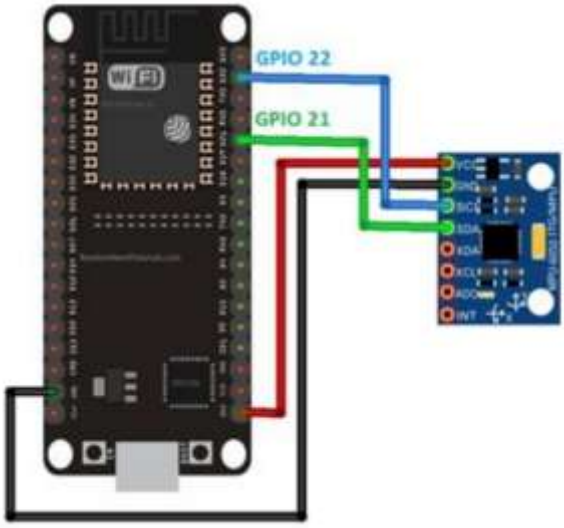
1. ESP32
2. LED (1 buah)
3. Resistor 150 Ohm (8 buah)
4. Sensor MPU6050
5. LCD(16x2) plus I2C
6. Jumper
7. Busur derajat

Langkah umum penyettingan alat dalam melakukan percobaan pada modul ini adalah sebagai berikut:

1. Buatlah rangkaian percobaan pada breadboard.
2. Buat kode yang akan digunakan pada percobaan.
3. Compile kode yang sudah dibuat.
4. Jika kode berhasil compile, hubungkan ESP32 ke komputer dengan kabel USB.
5. Pastikan ESP32 terhubung ke komputer dengan mengecek port pada pengelola perangkat.
6. Flash kode yang sudah berhasil dikompilasi ke ESP32.
7. Amati hasil percobaan dan catat pada buku catatan laboratorium.

3.1 MPU6050

Berikut ini adalah rangkain yang digunakan pada percobaan dengan MPU6050:

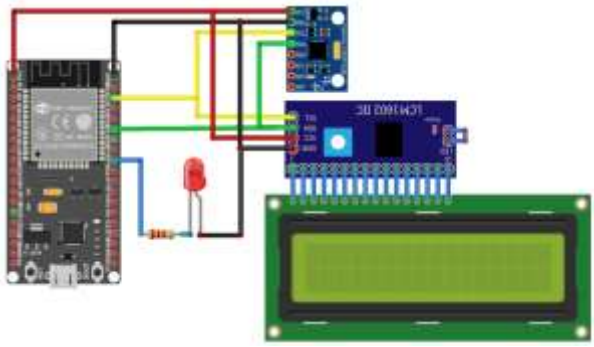


Koneksi yang digunakan adalah sebagai berikut:

MPU6050	ESP32
VCC	3.3V atau 5V
GND	GND
SCL	GPIO22
SDA	GPIO21

3.2 RTOS

Berikut ini adalah rangkaian yang digunakan pada percobaan RTOS:



4. HASIL DAN ANALISIS

Setelah dilakukan simulasi rangkaian percobaan pada praktikum ini, didapatkan data dan hasil analisis sebagai berikut:

4.1 MPU6050

Pada percobaan ini menggunakan modul MPU6050 yang menggunakan protokol komunikasi I2C.

Sensor MPU6050 dihubungkan dengan ESP32 dengan protokol komunikasi I2C dan hasil pembacaan sensor ditampilkan melalui serial monitor.

Berikut ini adalah penjelasan kode program yang digunakan untuk membaca sensor MPU6050

```
#include "Wire.h"

#define MPU_ADDR 0x68
```

Library Wire.h disini diperlukan untuk menghubungkan ESP dengan MPU melalui protokol komunikasi I2C. Protokol komunikasi I2C memiliki alamat untuk setiap devicenya sehingga perlu didefinisikan alamat device yang digunakan dalam hal ini MPU6050 menggunakan alamat 0x68.

Selanjutnya deklarasi variable yang akan digunakan dengan tipe data yang sesuai. Kemudian pada void setup, dilakukan inisiasi sebagai berikut:

```
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);

  Wire.endTransmission(true);
```

Pada bagian ini, program mengatur frekuensi komunikasi serial menjadi 9600 baud, kemudian menginisialisasi koneksi I2C dan menulis nilai 0 ke alamat register 0x6B pada sensor MPU6050 untuk mengaktifkan sensor.

Selanjutnya pada void loop akan dilakukan pembacaan sensor secara terus menerus dan menampilkannya ke serial monitor

```
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 14,
true);
```

Pertama memulai komunikasi dengan fungsi `beginTransmission(MPU_ADDR)` dengan alamat sebagai parameternya. Selanjutnya bagian `Wire.write(0x3B)` akan menuliskan alamat register yang akan dibaca pada MPU6050. Pada kasus ini, register 0x3B dipilih karena register tersebut berisi data akselerometer pada MPU6050.

Fungsi `endTransmission(false)` akan engirimkan sinyal STOP pada koneksi I2C setelah selesai menulis alamat register yang akan dibaca.

Pembacaan selanjutnya yaitu pada fungsi `requestFrom(MPU_ADDR, 14, true)` yang melakukan pengiriman sinyal START pada koneksi I2C dan menuliskan alamat slave device yang akan dibaca (dalam hal ini MPU6050). Parameter kedua adalah jumlah byte yang akan dibaca (dalam hal ini 14 byte), dan parameter ketiga adalah nilai true yang menandakan bahwa sinyal STOP akan dikirimkan setelah selesai membaca data.

```
  accX = Wire.read() << 8 |
Wire.read();
  accY = Wire.read() << 8 |
Wire.read();
  accZ = Wire.read() << 8 |
Wire.read();
```

Pembacaan data raw dari MPU6050 dilakukan dengan fungsi `accX = Wire.read() << 8 | Wire.read()`. Membaca nilai akselerasi sumbu X pada MPU6050 menggunakan fungsi read() dari koneksi I2C. Nilai akselerasi ini dikirimkan dalam dua byte, sehingga perlu dilakukan penggabungan byte tersebut menggunakan operator bitwise << (shift left) dan operator bitwise OR . Operasi << 8 digunakan untuk menggeser nilai byte pertama ke kiri sejauh 8 bit, kemudian nilai byte kedua ditambahkan dengan operator OR . Hasil akhirnya disimpan ke dalam variabel accX. Cara yang sama dilakukan untuk sumbu Y dan Z.

Setelah data raw terbaca, data tersebut kemudian dinormalisasi pada bagian kode berikut

```
  NormAccX = accX * rangePerDigit *
9.80665f;
  NormAccY = accY * rangePerDigit *
9.80665f;
  NormAccZ = accZ * rangePerDigit *
9.80665f;

//Mengkalkulasi pitch dan roll
  pitch = -(atan2(NormAccX,
sqrt(NormAccY*NormAccY + NormAccZ*
NormAccZ)) * 180.0) / M_PI;
  roll = (atan2(NormAccY,
NormAccZ)*180.0)/M_PI;
```

Kode tersebut merupakan proses untuk mengubah nilai akselerasi yang sudah dibaca pada sumbu X, Y, dan Z menjadi nilai akselerasi yang telah dinormalisasi (dalam satuan m/s^2) dan menghitung nilai pitch dan roll.

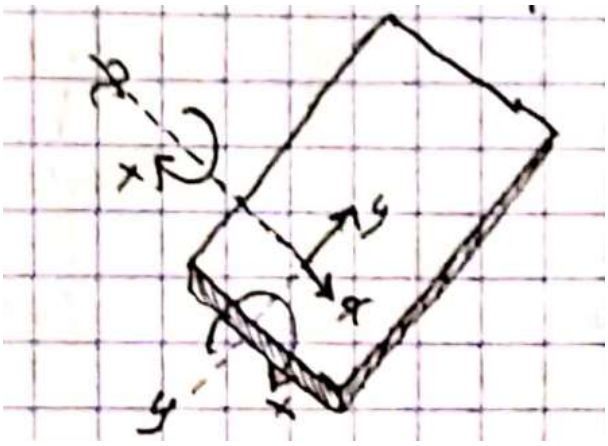
Selanjutnya nilai pitch dan roll yang sudah didapatkan ditampilkan ke serial monior.

```
  Serial.print("Pitch: ");
  Serial.print(pitch);
  Serial.print(" Roll: ");
  Serial.println(roll);
```

Hasil pembacaan sensor yang dibandingkan dengan pengukuran busur dapat diperhatikan pada tabel berikut:

Hasil Pengukuran Busur		Hasil Pembacaan Sensor	
Sumbu x	Sumbu y	Roll	Pitch
-90	0	-105	0
90	0	98	0
-45	0	-54	0
-12	13	-13	13
0	-90	0	-79
0	90	0	75
0	45	0	55

Arah sumbu dapat diperhatikan pada gambar berikut:



Dengan arah CW merupakan postif dan CCW negatif.

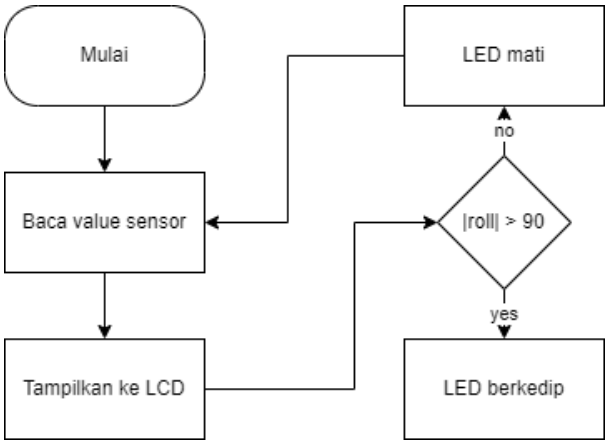
Dapat diperhatikan pada tabel tersebut bahwa hasil pembacaan untuk sensor sudah sesuai dalam arahnya yang mana ketika dimiringkan ke arah kanan akan pembacaan sensor bernilai positif, begitu pula sebaliknya.

Namun, sensor ini tidak cukup akurat. Hanya akurat pada sudut kecil, keakuratan sensor akan berkurang ketika sudur bacaan yang lebih dari 45 derajat.

4.2 RTOS

Percobaan ini adalah percobaan implementasi penggunaan RTOS pada ESP32 dengan pecobaan menggunakan sensor MPU6050 dan menampilkan display ke LCD.

Berikut ini adalah FC hasil percobaan MPU6050 dengan RTOS



Hasil pembacaan sensor dengan RTOS sama dengan percobaan sebelumnya karena menggunakan komunikasi, pembacaan dan algoritma yang sama.

5. KESIMPULAN

a. Interuput dapat ditrigger pada rising atau falling edge.

b. ESP32 memiliki internal resistor pullup dan pulldown yang dapat digunakan.

c. Penggunaan output digital dengan arus yang tidak melebihi batas secara bersamaan tidak menurunkan tegangan pada 5V esp secara signifikan

DAFTAR PUSTAKA

[1] Adijarto, Waskita dkk , *Petunjuk Praktikum Sistem Mikropeosesor*, Institut Teknologi Bandung, Bandung, 2023.

LAMPIRAN

1. Source code MPU6050

```
// Praktikum EL3116 - Sistem Microprosesor
// Modul : 4
// Percobaan : 1
// Tanggal : 31 Maret 2023
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Emmanuella Pramudita Rumanti (13220031)
// Nama File : main.ino
// Deskripsi : code for reading MPU6050 sensors

#include "Wire.h"
#define MPU_ADDR 0x68

int16_t accX, accY, accZ;
float rangePerDigit = .000061f;
float NormAccX,NormAccY,NormAccZ;
int pitch, roll;

void setup() {
   Serial.begin(9600);
   Wire.begin();
   Wire.beginTransmission(MPU_ADDR);
   Wire.write(0x6B);
   Wire.write(0);
   Wire.endTransmission(true);
}

void loop() {
   Wire.beginTransmission(MPU_ADDR);
   Wire.write(0x3B);
   Wire.endTransmission(false);
   Wire.requestFrom(MPU_ADDR, 14, true);

   //Pembacaan urut dari alamat accX
   accX = Wire.read() << 8 | Wire.read();
   accY = Wire.read() << 8 | Wire.read();
   accZ = Wire.read() << 8 | Wire.read();

   //Normalisasi Raw Data tersebut
   NormAccX = accX * rangePerDigit * 9.80665f;
   NormAccY = accY * rangePerDigit * 9.80665f;
   NormAccZ = accZ * rangePerDigit * 9.80665f;

   //Mengkalkulasi pitch dan roll
   pitch = -(atan2(NormAccX, sqrt(NormAccY*NormAccY + NormAccZ*
NormAccZ)) * 180.0) / M_PI;
   roll = (atan2(NormAccY, NormAccZ)*180.0)/M_PI;

   //Output Serial
   Serial.print("Pitch: ");
   Serial.print(pitch);
   Serial.print(" Roll: ");
   Serial.println(roll);
   delay(100);
}
```

2. Source code RTOS

```
// Praktikum EL3116 - Sistem Microprosesor
// Modul : 4
// Percobaan : 2
// Tanggal : 31 Maret 2023
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Emmanuella Pramudita Rumanti (13220031)
// Nama File : main.ino
// Deskripsi : code for reading MPU6050 and diplay to LCD with RTOS

#include "Wire.h"
#define MPU_ADDR 0x68
#define rangePerDigit .000061f
#include <LiquidCrystal_I2C.h>
#define LED 19

//Tasks Priority
#define priorityTask1 2
#define priorityTask2 2
#define priorityTask3 2

int pitch, roll;

//Mutex Definition
SemaphoreHandle_t xMutex;

//Task Display
int lcdColumns = 16;
int lcdRows = 2;

//Membuat objek LCD
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

//Sensing Task
void SensingTask(void *pvParam);

//LED Task
void BlinkTask(void *pvParam);

//Display Task
void DisplayTask(void *pvParam);

void setup() {

   Serial.begin(9600);

   //Inisiasi LED
   pinMode(LED, OUTPUT);

   //Inisiasi MPU
   Wire.begin();
   Wire.beginTransmission(MPU_ADDR); //Inisialisasi komunikasi I2C
dengan MPU
   Wire.write(0x6B); //Power Management untuk MPU6050
   Wire.write(0); //Membangunkan MPU
   Wire.endTransmission(true);

   //Inisiasi LCD
   lcd.init();
```

```
    lcd.backlight();

    //Mutex
    xMutex = xSemaphoreCreateMutex();

    //Task Start
    xTaskCreatePinnedToCore(SensingTask, "Task 1", 2048,
NULL,priorityTask1, NULL, 0);
    xTaskCreatePinnedToCore(BlinkTask, "Task 2", 2048, NULL,
    priorityTask2, NULL, 1);
    xTaskCreatePinnedToCore(DisplayTask, "Task 3", 2048, NULL,
    priorityTask3, NULL, 0);
}


void SensingTask(void *pvParam) {
    (void) pvParam;

    int16_t accX, accY, accZ;
    float NormAccX, NormAccY, NormAccZ;

    while (1) {
          xSemaphoreTake(xMutex, portMAX_DELAY);
          {
                Wire.beginTransmission(MPU_ADDR);
                Wire.write(0x3B);
                Wire.endTransmission(false); //Agar transimisi tetap
berjalan
                Wire.requestFrom(MPU_ADDR, 6, true); //Akan mengakses 6
register

                //Pembacaan urut dari alamat awal
                accX = (Wire.read()<<8)|(Wire.read());
                accY = (Wire.read()<<8)|(Wire.read());
                accZ = (Wire.read()<<8)|(Wire.read());
          }

          xSemaphoreGive(xMutex);

          //Normalisasi Raw Data tersebut
          NormAccX = accX * rangePerDigit * 9.80665f;
          NormAccY = accY * rangePerDigit * 9.80665f;
          NormAccZ = accZ * rangePerDigit * 9.80665f;

          //Mengkalkulasi pitch dan roll
          pitch = -(atan2(NormAccX, sqrt(NormAccY * NormAccY + NormAccZ
* NormAccZ)) * 180.0) / M_PI;
          roll = (atan2(NormAccY, NormAccZ) * 180.0) / M_PI;

          //Periode 100ms
          vTaskDelay(pdMS_TO_TICKS(100));
    }
}


void BlinkTask(void *pvParam) {
    (void) pvParam;

    while (1) {

          //Ketika roll di luar rentang -90 hingga 90 derajat
          if (roll > 90 || roll < -90) {
                digitalWrite(LED,HIGH); //LED dinyalakan
```

```
            } else {
                  digitalWrite(LED,LOW);
            }

            //Periode 500ms
            vTaskDelay(pdMS_TO_TICKS(500));
            digitalWrite(LED,LOW);
            vTaskDelay(pdMS_TO_TICKS(500));
      }
   }

   void DisplayTask(void *pvParam) {
      (void) pvParam;

      while (1) {
            xSemaphoreTake(xMutex, portMAX_DELAY);

            {
                  lcd.clear();
                  lcd.setCursor(0, 0);
                  lcd.print("Pitch : ");
                  lcd.print(pitch);
                  lcd.setCursor(0, 1);
                  lcd.print("Roll : ");
                  lcd.print(roll);
            }

            xSemaphoreGive(xMutex);

            //Refresh 1 detik
            vTaskDelay(pdMS_TO_TICKS(1000));
      }
   }

   void loop() {  } //kosong
```