

BAB IV

PEMBUATAN ALAT

4.1 Pembuatan Perangkat Keras

4.1.1 Alat dan Bahan

Proses pertama dalam pembuatan sistem ini adalah mempersiapkan alat dan bahan yang digunakan. Daftar alat yang digunakan untuk membangun rancang bangun sistem *Monitoring* baterai LiFePO₄ pada PLTS berbasis IoT menggunakan komunikasi LoRa dapat dilihat pada Tabel 4.1.

Tabel 4-1 Bahan Pembuatan Tugas Akhir

No	Nama Bahan	Jumlah
1	Panel Surya 50Wp	1 Buah
2	<i>Solar Charge Controller</i> 12V	1 Buah
3	Baterai Lithium Ion 12V 12AH	2 Buah
4	Buck Converter MP1584EN	1 Buah
5	Sensor INA219	1 Buah
6	Sensor DS18B20	1 Buah
7	ESP 32	2 Buah
8	Lora SX1278	2 Buah
9	Adaptor 5V 1A	1 Buah
10	LCD 16x2	1 Buah
11	Wifi Antenna 433 Mhz 5DBi	2 Buah
12	Switch ON/OFF	1 Buah
13	Kabel JST XH 2,54 mm	2 Buah
14	Kapasitor 470 uF	2 buah
15	Dioda Rectifier 1N5822	2 buah
16	Konektor Jack DC	2 buah
17	Konektor XH2 2.54mm	2 buah
18	Panel Listrik 35x25x12cm	1 Buah
19	<i>Box</i> X5 145x95x50mm	1 Buah

No	Nama Bahan	Jumlah
20	Tiang Penyangga	1 Buah
21	Lampu LED DC 20 Watt	1 Buah

Pada Tabel 4.2 merupakan bahan yang digunakan untuk membangun rancang bangun sistem *Monitoring* baterai LiFePO4 pada PLTS berbasis IoT menggunakan komunikasi LoRa.

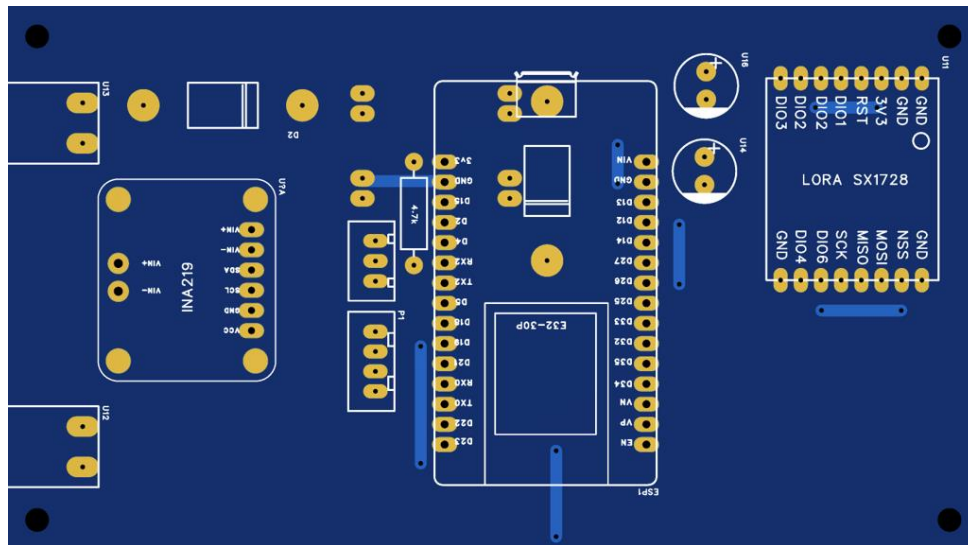
Tabel 4-2 Alat yang Digunakan

No	Nama Alat	Jumlah
1	Multimeter	1 Buah
2	Obeng Set	1 Buah
3	Gunting	1 Buah
4	Tang Potong	1 Buah
5	Tang kupas	1 Buah
6	Solder	1 Buah
7	Bor Tangan Set	1 Buah
8	Janga Sorong	1 Buah
9	Gerinda	1 Set
10	Alat Tulis	1 Set

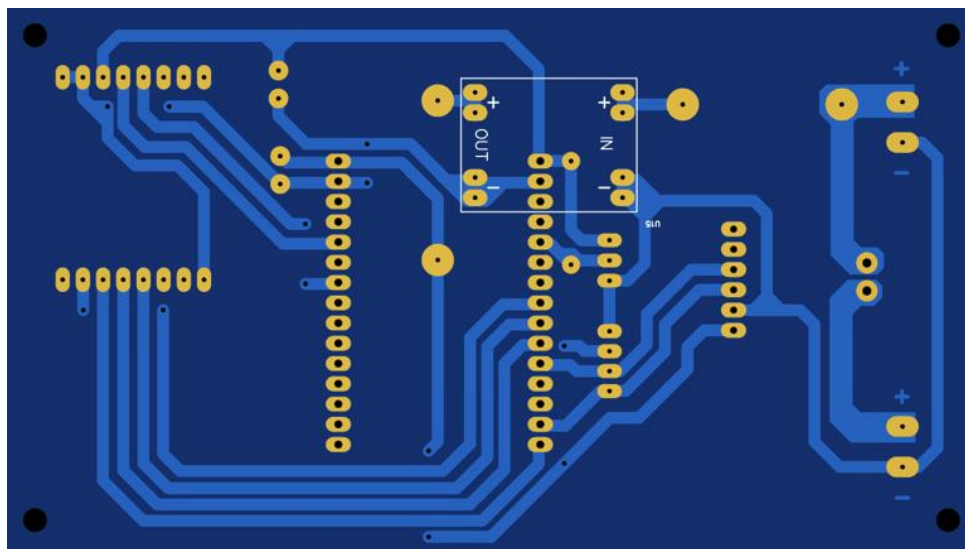
4.2 Pembuatan Perangkat Elektrik

Proses pembuatan perangkat elektrik tahapan penting dalam proses perakitan dalam rancangan alat ini. Tahap ini, melakukan perancangan dan perakitan seluruh komponen elektronik seperti mikrokontroller, sensor dan sistem komunikasi yang berfungsi mengendalikan sistem serta mendeteksi keberhasilan keluaran yang diinginkan. Seluruh komponen elektronik dirangkai dalam satu sistem berbasis mikrokontroller. Berikut langkah-langkah pembuatan perangkat elektrik:

4.2.1 Perancangan Skematik dan Layout PCB



Gambar 4.1 Layout PCB (*Transmitter*) Tampak Depan



Gambar 4.2 Layout PCB (*Transmitter*) Tampak Belakang

Langkah pertama melakukan pembuatan skematik rangkaian elektronik menggunakan perangkat lunak EasyEDA seperti yang ditunjukkan pada gambar di atas. Dalam skematik ini, terbagi menjadi dua modul yaitu modul *transmitter* dan modul *receiver*. Pada modul *transmitter*, seluruh komponen utama seperti mikrokontroler ESP32, sensor INA219, sensor DS18B20, *buck converter* MP1584EN, dan modul LoRa SX1278 Ra-02 dirancang dengan sistematis agar komponen dapat terhubung dengan benar sesuai kebutuhan sistem. Pada modul

receiver, komponen yang dirancang terdiri dari ESP32, modul LoRa SX1278 Ra-02, LCD I2C 16x2, serta konektor daya dari adaptor *switching* 5V 1A. Penyusunan skematik ini bertujuan untuk memastikan jalur catu daya, jalur sinyal, dan koneksi *input-output* terhubung sesuai dengan rancangan kebutuhan sistem, termasuk pemasangan dioda Schottky 1N5822 dan SS14 sebagai proteksi polaritas terbalik, serta kapasitor filter 470 μ F 25V pada jalur tegangan +5V dan +3,3V.

Setelah rangkaian skematik selesai dirancang, selanjutnya adalah pembuatan layout pada PCB berdasarkan jalur dan koneksi yang telah ditentukan. Proses ini meliputi peletakan komponen pada papan rangkaian, pengaturan jalur penghantar (*routing*), penyesuaian ukuran papan PCB, serta penataan pin komunikasi untuk modul LoRa SX1278 Ra-02 dan LCD I2C 16x2 agar tidak saling bersilangan. Dengan *layout* PCB yang tersusun dan sesuai dengan rancangan, proses perakitan rangkaian dapat dilakukan dengan mudah, sekaligus meminimalisir kesalahan pada jalur dan koneksi pada sistem *Monitoring* baterai LiFePO₄ yang dirancang.

4.2.2 Proses pencetakan dan penyolderan PCB

Setelah proses perancangan skematik dan *layout* selesai dilakukan menggunakan perangkat lunak EasyEDA, desain PCB dicetak menggunakan metode CNC Laser. Tahap pertama, menyiapkan PCB polos dan membersihkan papan menggunakan amplas untuk menghilangkan oksidasi agar cat pilox dapat menempel dengan rata. Selanjutnya, mengecat PCB dengan pilox semprot secara tipis dan merata, kemudian ditunggu hingga mengering. Setelah itu, papan PCB yang sudah dilapisi cat pilox ditempatkan pada meja atau bracket CNC laser, dan laser akan membuat jalur rangkaian sesuai dengan rancangan skematik yang telah dibuat untuk kedua modul, yaitu modul *transmitter* dan modul *receiver*.

Setelah proses pembuatan jalur berhasil, dilakukan proses pelarutan tembaga (*etching*) dengan merendam papan PCB dalam larutan *ferric chloride* (FeCl₃). Larutan ini berfungsi menghilangkan bagian tembaga yang tidak dilapisi oleh cat pilox, sehingga menyisakan jalur tembaga sesuai dengan

rancangan layout PCB yang telah dibuat. Setelah tahap etching selesai, papan PCB dibersihkan menggunakan cairan thinner untuk menghilangkan sisa cat pilox dan memperlihatkan jalur tembaga, kemudian dikeringkan.

Selanjutnya adalah proses pelubangan (*drilling*) pada papan PCB menggunakan alat bor mini untuk membuat lubang pemasangan komponen elektronik seperti terminal konektor, pin header, dan modul pendukung lainnya. Setelah lubang selesai dibuat, komponen-komponen utama dipasang dan disolder pada masing-masing papan PCB sesuai modulnya. Pada PCB *transmitter* dipasang mikrokontroler ESP32, sensor INA219, sensor DS18B20, *buck converter* MP1584EN, modul LoRa SX1278 Ra-02, dua dioda Schottky 1N5822, serta kapasitor filter 470 μ F 25V. Pada PCB *receiver* dipasang mikrokontroler ESP32, modul LoRa SX1278 Ra-02, dioda Schottky SS14, kapasitor filter 470 μ F 25V, serta konektor untuk LCD I2C 16x2.

Konektor untuk sumber daya menggunakan terminal *blok screw* untuk memudahkan penyambungan pada sumber tegangan baterai LiFePO4 dan adaptor secara aman dan efisien, sedangkan konektor XH2.54-4P digunakan untuk komponen sensor INA219 agar proses peletakan dan perawatan sistem dapat dilakukan dengan mudah. Penempatan komponen yang sesuai struktur dan jalur rangkaian yang rapi dapat membuat sistem *Monitoring* baterai LiFePO4 pada Pembangkit Listrik Tenaga Surya (PLTS) berbasis IoT menggunakan komunikasi LoRa dapat bekerja dengan maksimal dan meminimalisir risiko kerusakan yang terjadi.



Gambar 4.3 Proses Pencetakan & Penyolderan PCB

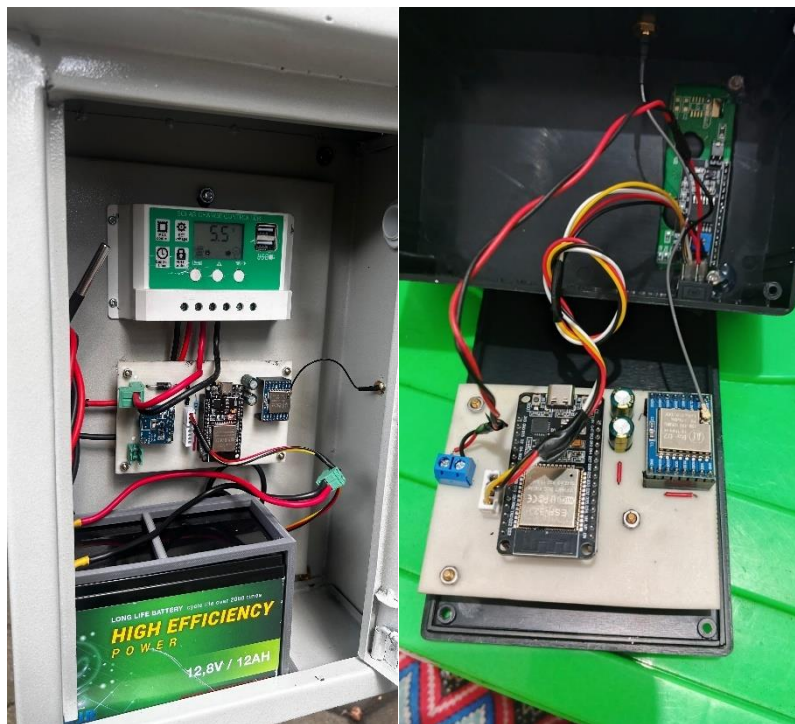
4.2.3 Integrasi Pada Sistem

Beberapa komponen pada sistem *Monitoring* baterai LiFePO₄ pada Pembangkit Listrik Tenaga Surya (PLTS) berbasis IoT menggunakan komunikasi LoRa, seperti sensor DS18B20, sensor INA219, LCD I2C 16x2, dan sumber catu daya, tidak disolder secara langsung pada papan PCB utama. Komponen-komponen tersebut dihubungkan menggunakan kabel *jumper*, konektor XH2.54-4P, terminal konektor *screw*, dan *pin header socket*.

Tujuan penggunaan konektor pada komponen tersebut adalah untuk memberikan kemudahan dan fleksibilitas dalam peletakan komponen saat proses integrasi sistem ke dalam bentuk *prototype*. Sensor DS18B20 yang dipasang menempel pada permukaan baterai LiFePO₄ dihubungkan ke PCB *transmitter* menggunakan kabel dengan resistor *pull-up* 4,7k Ω , sehingga memungkinkan sensor ditempatkan pada jarak tertentu dari PCB sesuai kebutuhan pengukuran suhu yang akurat. Sensor INA219 yang dipasang pada jalur antara baterai dan SCC dihubungkan menggunakan konektor XH2.54-4P dan *terminal screw*, sehingga memudahkan pemasangan tanpa perlu memotong kabel secara permanen. LCD I2C 16x2 pada unit *receiver* dihubungkan menggunakan kabel *jumper* melalui jalur SDA dan SCL, sehingga memungkinkan LCD ditempatkan pada bagian depan *box* panel terpisah dari

posisi PCB utama. Sumber catu daya baik dari baterai LiFePO₄ maupun adaptor *switching* 5V 1A dihubungkan menggunakan terminal konektor *screw* dan konektor DC *Jack* agar proses penyambungan dan pemutusan daya dapat dilakukan dengan aman dan praktis.

Selain meningkatkan fleksibilitas peletakan, penggunaan konektor juga memudahkan proses perawatan dan penggantian komponen apabila terjadi kerusakan pada modul sensor, modul komunikasi, maupun modul penampilan. Komponen dapat diganti tanpa perlu melakukan proses penyolderan ulang pada PCB, Hal ini sangat efisien pada tahap pengembangan prototipe sistem *Monitoring* baterai LiFePO₄ yang memungkinkan adanya perubahan sistem atau perbaikan desain selama proses pengujian berlangsung.



Gambar 4.4 Integrasi Sistem Pada Modul *Transmitter* dan *Receiver*

4.3 Pembuatan Perangkat Mekanik

Setelah pengembangan perangkat selesai dan komponen elektronik terhubung dengan baik, selanjutnya adalah melakukan rancangan mekanik. Perancangan mekanik bertujuan merancang struktur fisik yang dapat melindungi perangkat keras dari gangguan lingkungan, memudahkan pemasangan komponen mikrokontroller, sensor serta sistem komunikasi dan memastikan stabilitas sistem secara keseluruhan. Dengan demikian, integrasi perangkat keras elektronik dan perancangan mekanik yang benar dapat menghasilkan rancangan yang tidak hanya fungsional tapi tahan lama dan mudah digunakan dalam aplikasi nyata.

Desain komponen dilakukan dengan menggunakan perangkat lunak SolidWorks. Sistem mekanik pada rancangan alat ini difokuskan pada peletakan dan kestabilan sensor INA219 dan sensor DS18B20 dan sistem komunikasi LoRa SX1278 agar data yang diperoleh akurat dan pengiriman data hasil *Monitoring* baterai Lithium Ion dapat dilakukan tidak ada gangguan.

Sistem rancangan ini tersusun dari beberapa komponen mekanik utama yang dirancang menggunakan material besi hollow karena memiliki kelebihan pada ketahanan yang baik terhadap kondisi lapangan. Komponen-komponen tersebut terdiri dari:

1. Tiang Penyangga Utama

Berfungsi sebagai kerangka utama dan sebagai penghubung antara bagian atas dan bawah, dan untuk menempatkan panel *box* listrik

2. Penyangga Bawah

Berfungsi sebagai penopang fondasi alat yang menopang seluruh fondasi struktur agar tetap stabil dan aman saat digunakan

3. *Box* Panel Listrik

Berfungsi sebagai tempat untuk menyimpan komponen-komponen elektronik, *Solar Charge Controller* dan baterai Lithium Ion

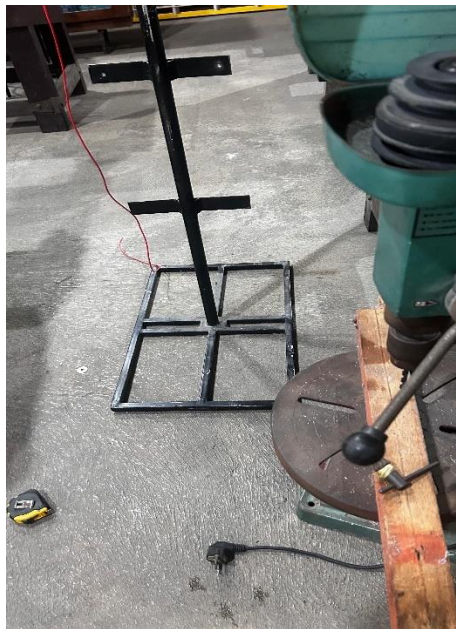
4. Dudukan Panel Surya

Berfungsi sebagai wadah untuk meletakkan panel surya agar panel surya berada pada posisi yang sesuai dengan pancaran sinar matahari.

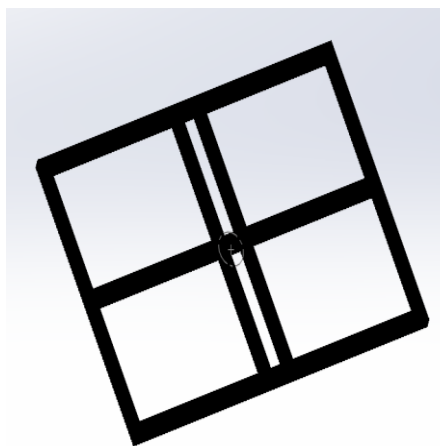
5. Box BX-X5

Berfungsi sebagai wadah untuk meletakan komponen-komponen elektronik pada modul *Receiver*.

Seluruh komponen dirancang menggunakan metode pengelasan maupun pengeboran , sesuai dengan kebutuhan dan kekuatan yang berfungsi untuk mendukung kinerja pada rancangan sistem ini. Desain ini memudahkan proses instalasi dan perawatan sehingga kinerja alat ini dapat maksimal.



Gambar 4.5 Tiang Penyangga Utama



Gambar 4.6 Penyangga Bawah



Gambar 4.7 *Box Panel Listrik*



Gambar 4.8 *Dudukan Panel Surya*



Gambar 4.9 *Box BX-B5*

4.4 Pembuatan Perangkat Lunak

Pemrograman mikrokontroler ESP32 dilakukan menggunakan perangkat lunak Arduino IDE dengan bahasa pemrograman C++. Program ini merupakan bagian utama dari sistem karena mengatur kerja seluruh komponen pada kedua unit ESP32, yaitu unit *transmitter* dan unit *receiver*. Pada unit *transmitter*, program mengatur kerja sensor INA219, sensor DS18B20, dan modul LoRa SX1278 Ra-02. Pada unit *receiver*, program mengatur kerja modul LoRa SX1278 Ra-02, LCD I2C 16x2, koneksi WiFi, dan *Google Sheets*. Struktur kode disusun secara modular untuk memudahkan pengembangan, pengujian, dan perawatan sistem.

Pembagian program yang dikembangkan meliputi beberapa bagian, yaitu:

1. Program untuk inisialisasi pin dan konfigurasi awal ESP32 *transmitter* dan *receiver*.
2. Program untuk membaca data sensor INA219 dan sensor DS18B20 pada unit *transmitter*.
3. Program untuk mengirim sinyal permintaan data "REQ" dari *receiver* ke *transmitter*.

4. Program untuk menghitung arus, daya, dan *Load Voltage* berdasarkan data sensor yang diperoleh.
5. Program untuk mengemas dan mengirimkan data sensor melalui komunikasi LoRa.
6. Program untuk menampilkan hasil pengukuran pada LCD I2C 16x2.
7. Program untuk mengirim dan menyimpan data hasil pengukuran ke *Google Sheets*.
8. Program *looping* untuk melakukan pembacaan dan pengiriman data secara berulang secara *real-time*.

4.4.1 Inisialisasi Pin dan Konfigurasi Awal ESP 32

ESP32 merupakan mikrokontroler utama yang digunakan sebagai pusat pengendali pada sistem *Monitoring* baterai LiFePO4. Pada tahap awal program dilakukan proses inisialisasi pin dan konfigurasi awal ESP32 agar seluruh komponen pada sistem dapat bekerja dengan baik sesuai fungsi yang telah dirancang.

Pada proses inisialisasi, program mendefinisikan pin-pin GPIO yang digunakan untuk menghubungkan modul LoRa SX1278 Ra-02, sensor DS18B20, sensor INA219, serta komponen pendukung lainnya. Pin SPI digunakan untuk komunikasi dengan modul LoRa, sedangkan pin GPIO4 digunakan untuk komunikasi 1-Wire dengan sensor DS18B20. Selain itu, dilakukan juga konfigurasi komunikasi serial pada baudrate 115200 sebagai media *Monitoring* dan debugging sistem selama proses pengujian berlangsung.

```
//-----INCLUDE SECTION-----//
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <Adafruit_INA219.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//-----DEFINE AND VARIABEL SECTION-----//
// LORA
#define SS      5
#define RST     14
```

```

#define DI00 26
const long frequency = 433E6;
// DS18B20
#define DS18B20_PIN 4

// SHUNT RESISTOR
const float R_SHUNT = 0.1; // Ohm

//-----SENSOR DECLARATION-----//
OneWire onewire(DS18B20_PIN);
DallasTemperature sensors(&onewire);
Adafruit_INA219 INA219(0x40);

```

Tahap konfigurasi awal ini menjadi dasar penting agar seluruh perangkat keras dapat saling terhubung dan bekerja secara sinkron dalam membaca parameter baterai serta mengirimkan data secara nirkabel melalui LoRa. Fungsi `setup()` pada ESP32 *transmitter* digunakan untuk menjalankan seluruh proses inisialisasi modul yang dibutuhkan, yaitu LoRa, DS18B20, dan INA219, seperti dijelaskan pada kode berikut:

```

//-----SETUP AND LOOP-----//
void setup() {
  Serial.begin(115200);
  setupLoRa();
  setupDS18B20();
  setupINA219();
}

```

4.4.2 Pembacaan Data Sensor INA219 dan DS18B20

ESP32 Sensor INA219 dan DS18B20 digunakan sebagai komponen utama untuk memperoleh data parameter baterai LiFePO₄ yang melintas pada jalur antara baterai dan SCC. Sistem bekerja dengan membaca nilai tegangan *shunt* dan *Bus Voltage* dari sensor INA219 melalui komunikasi I2C, serta nilai suhu dari sensor DS18B20 melalui protokol 1-Wire. Proses pembacaan sinyal dilakukan oleh ESP32 melalui pin yang telah dikonfigurasi sebelumnya.

Pada program, ESP32 akan membaca seluruh sensor secara hampir bersamaan dalam satu waktu untuk menjaga konsistensi data. Pembacaan sensor dilakukan melalui fungsi `getShuntVoltage_mV()` dan `getBusVoltage_V()` pada

sensor INA219, serta fungsi requestTemperatures() dan getTempCByIndex(0) pada sensor DS18B20.

```
// DS18B20
void setupDS18B20() {
  Serial.println("Preparing DS18B20 Sensor");
  sensors.begin();
}

float getTemperature() {
  sensors.requestTemperatures();
  return sensors.getTempCByIndex(0);
}

// INA219
void setupINA219() {
  Serial.println("Preparing INA219 Sensor");
  if (!INA219.begin()) {
    Serial.println("Failed to find INA219 chip");
    while (1) { delay(10); }
  }
  INA219.setCalibration_32V_2A();
  Serial.println("INA219 Sensor Ready (Calibrated 32V/2A)");
}
```

Hasil pembacaan sensor kemudian digunakan untuk menentukan kondisi operasi baterai. Ketika nilai arus bernilai positif, sistem akan menganggap baterai sedang dalam kondisi *charging*. Sebaliknya, jika nilai arus bernilai negatif maka baterai dianggap sedang dalam kondisi *discharging*.

```
if (Current > 10) Serial.println("Status: CHARGING");
else if (Current < -10) Serial.println("Status:
DISCHARGING");
else Serial.println("Status: IDLE");
```

4.4.3 Perhitungan Arus, Daya, dan Load Voltage

Setelah seluruh sensor berhasil dibaca, sistem akan menghitung parameter turunan berdasarkan data mentah yang diperoleh dari sensor INA219. Perhitungan dilakukan secara manual menggunakan rumus yang telah ditentukan sebelumnya, yaitu arus dihitung dari pembagian tegangan *shunt* dengan nilai resistor *shunt* 0,1Ω, daya dihitung dari perkalian arus dengan *Bus*

Volatge, serta *Load Voltage* dihitung dari penjumlahan *Bus Volatge* dengan tegangan *shunt* yang dikonversi ke satuan Volt.

```
String sendDataToReceiver() {
    // Baca suhu, shunt, dan bus SEKALI di waktu yang sama
    float temp    = getTemperature();
    float shunt    = INA219.getShuntVoltage_mV();
    float bus      = INA219.getBusVoltage_V();

    // Hitung Current, Power, Load SEMUA dari nilai raw yang sama
    float Current = shunt / R_SHUNT; // mA, (+) Charging, Discharging
    float Power   = Current * bus;   // mW, tanda mengikuti Current
    float Load    = bus + (shunt / 1000.0); // V

    String data = "DATA,";
    data += String(temp, 2) + ",";
    data += String(shunt, 2) + ",";
    data += String(bus, 2) + ",";
    data += String(Current, 2) + ",";
    data += String(Power, 2) + ",";
    data += String(Load, 2);
    return data;
}
```

4.4.4 Pengiriman dan Penerimaan Data Melalui LoRa

Pengiriman dan penerimaan data dilakukan berdasarkan mekanisme request-driven antara unit *transmitter* dan unit *receiver*. Sistem memanfaatkan modul LoRa SX1278 Ra-02 untuk mengirimkan sinyal permintaan data dan membalas dengan data sensor yang telah diolah. Pada program, nilai data dikemas dalam format string CSV yang dipisahkan menggunakan tanda koma. Konfigurasi LoRa ditetapkan pada frekuensi 433 MHz dengan Spreading Factor SF11 dan daya pancar 20 dBm sehingga jangkauan komunikasi dapat mencapai ± 80 meter antara unit *transmitter* dan *receiver*

```
void setupLoRa() {
    Serial.println("Preparing LoRa SX1278");
    LoRa.setPins(SS, RST, DIO0);
    if (!LoRa.begin(frequency)) {
        Serial.println("LoRa init failed. Check your connections.");
        while (true);
    }
    // === SETTING JANGKAUAN (HARUS SAMA PERSIS DENGAN RECEIVER) ===
}
```

```

LoRa.setSpreadingFactor(11);    // SF11: jangkauan jauh dengan
margin aman (~1km)
LoRa.setSignalBandwidth(125E3); // 125 kHz: standar, seimbang
LoRa.setTxPower(20);           // 20 dBm: daya pancar maksimal
LoRa.setCodingRate4(5);        // 4/5: error correction
LoRa.setPreambleLength(8);     // panjang preamble standar
LoRa.setSyncWord(0x12);        // sync word (harus sama dengan
receiver)
// =====
Serial.println("LoRa init success! (SF11, BW125kHz, TX20dBm)");
}

void sendLoRa(String dataToSend) {
LoRa.beginPacket();
LoRa.print(dataToSend);
LoRa.endPacket();
Serial.println("Sending: " + dataToSend);
}

```

4.4.5 Penerimaan Permintaan Data Dari *Receiver*

Penerimaan permintaan data dilakukan untuk mengetahui adanya sinyal "REQ" yang dikirimkan oleh unit *receiver* setiap interval 5 menit. Sistem memanfaatkan fungsi `LoRa.parsePacket()` pada ESP32 *transmitter* untuk memantau kanal komunikasi secara terus menerus selama proses operasi berlangsung. Nilai data yang diterima nantinya digunakan sebagai dasar dalam memicu proses pengambilan data sensor. Ketika modul LoRa menerima paket data, sistem akan membaca isi paket tersebut karakter demi karakter dan menyimpannya ke dalam variabel `incoming` untuk divalidasi.

```

void checkIncoming() {
int packetSize = LoRa.parsePacket();
if (packetSize) {
String incoming = "";
while (LoRa.available()) {
incoming += (char)LoRa.read();
}
Serial.println("Get Request: " + incoming);
delay(1000);
handleRequest(incoming);
}
}

```

Setelah paket data berhasil diterima, sistem akan memvalidasi apakah data tersebut benar-benar berupa sinyal "REQ" yang valid sebelum melanjutkan proses pengambilan data sensor.

```
void handleRequest(String incoming) {
  if (incoming == "REQ") {
    String data = sendDataToReceiver();
    sendLoRa(data);
  }
}
```

4.4.6 Penerimaan Permintaan Data Dari *Receiver*

LCD I2C digunakan sebagai media tampilan untuk menampilkan informasi hasil pembacaan sistem secara *real-time* pada unit *receiver*. Informasi yang ditampilkan meliputi suhu baterai, tegangan *shunt*, *Bus Volatge*, arus, daya, dan *Load Volatge* yang telah dihitung oleh ESP32 *transmitter*. Penggunaan LCD I2C bertujuan agar pengguna dapat memantau kondisi sistem secara langsung tanpa perlu menggunakan perangkat tambahan. Pada tahap awal program, LCD diinisialisasi menggunakan library `LiquidCrystal_I2C.h`. Sistem kemudian mengaktifkan backlight dan menampilkan tampilan awal sebagai indikator bahwa sistem telah berhasil dijalankan.

```
#define LCD_ADDRESS 0x27
#define LCD_COLUMNS 16
#define LCD_ROWS 2
LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLUMNS, LCD_ROWS)
void setupLCD() {
  lcd.init();
  lcd.backlight();
  lcd.clear();
}
```

Setelah proses penerimaan data selesai dilakukan, nilai parameter baterai akan ditampilkan pada LCD secara bergantian setiap 3 detik untuk masing-masing parameter.

```
void dataDisplay(int8_t displayData) {
  switch (displayData) {
    case DISPLAY_TEMP:
      sprintf(lcdbuff, "%0.2f C", temp);
      lcd.setCursor(0, 0); lcd.print("Temperature:");
      lcd.setCursor(0, 1); lcd.print(lcdbuff);
  }
}
```

```

        break;
    case DISPLAY_SHUNT_VOLTAGE:
        sprintf(lcdbuff, "%0.2f mV", shuntVoltage);
        lcd.setCursor(0, 0); lcd.print("Shunt Voltage:");
        lcd.setCursor(0, 1); lcd.print(lcdbuff);
        break;
    case DISPLAY_BUS_VOLTAGE:
        sprintf(lcdbuff, "%0.2f V", busVoltage);
        lcd.setCursor(0, 0); lcd.print("Bus Volatge:");
        lcd.setCursor(0, 1); lcd.print(lcdbuff);
        break;
    case DISPLAY_CURRENT:
        sprintf(lcdbuff, "%0.2f mA", Current_mA);
        lcd.setCursor(0, 0); lcd.print("Current:");
        lcd.setCursor(0, 1); lcd.print(lcdbuff);
        break;
    case DISPLAY_POWER:
        sprintf(lcdbuff, "%0.2f mW", Power_mW);
        lcd.setCursor(0, 0); lcd.print("Power:");
        lcd.setCursor(0, 1); lcd.print(lcdbuff);
        break;
    case DISPLAY_LOAD_VOLTAGE:
        sprintf(lcdbuff, "%0.2f V", LoadVoltage);
        lcd.setCursor(0, 0); lcd.print("Load Voltage:");
        lcd.setCursor(0, 1); lcd.print(lcdbuff);

```

Selain menampilkan nilai parameter baterai, LCD juga digunakan untuk menampilkan kondisi status sistem seperti proses koneksi WiFi, proses pengiriman data, maupun kondisi timeout. Tampilan tersebut akan diperbarui secara *real-time* sesuai hasil penerimaan data yang diterima oleh ESP32 *receiver*.

4.4.7 Penerimaan Permintaan Data Dari Receiver

Sistem *Monitoring* yang dirancang dilengkapi dengan fitur *Internet of Things* (IoT) yang memungkinkan data hasil pembacaan baterai dapat dikirim dan disimpan secara otomatis ke *Google Sheets* melalui koneksi internet. Fitur ini bertujuan untuk memudahkan proses *Monitoring* dan penyimpanan data baterai LiFePO4 secara *real-time*. Pada tahap awal, ESP32 *receiver* akan terhubung ke jaringan Wi-Fi menggunakan SSID dan *password* yang telah

ditentukan pada program. Setelah koneksi berhasil dilakukan, sistem akan melakukan inisialisasi *Google Sheets* menggunakan *library* ESP_Google_Sheet_Client.h

```
#define WIFI_SSID    "ALVIANO"
#define WIFI_PASSWD  "02242024"

void setupWiFi();
WiFi.setAutoReconnect(true);
WiFi.begin(WIFI_SSID, WIFI_PASSWD);
lcd.clear();
displayCentered("Connect To WiFi", 0);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(1000);
  Serial.println("Wi-Fi connected successfully");
  Serial.print("Connected with IP: ");
```

Setelah koneksi internet berhasil, sistem melakukan autentikasi akun *Google Sheets* menggunakan PROJECT_ID, CLIENT_EMAIL, dan PRIVATE_KEY. Tahap ini diperlukan agar ESP32 dapat mengakses dan mengirim data menuju spreadsheet yang telah ditentukan.

```
#define PROJECT_ID    "Monitoring-baterai-pada-plts"
#define CLIENT_EMAIL  "Monitoring-baterai-lithium-ion@Monitoring-
baterai-pada-plts.iam.gserviceaccount.com"
const char PRIVATE_KEY[] PROGMEM = "BEGIN PRIVATE KEY
const char spreadsheetId[] = "1SNl6d-8xp0lHA_bUBu0

void GsheetSetup();
GSheet.printf("ESP Google Sheet Client v%s\n\n",
ESP_GOOGLE_SHEET_CLIENT_VERSION);
GSheet.setTokenCallback(tokenStatusCallback);
GSheet.setPrereshSeconds(10 * 60);
GSheet.begin(CLIENT_EMAIL, PROJECT_ID, PRIVATE_KEY);
```

Data yang dikirim ke *Google Sheets* meliputi timestamp, suhu, *Shunt Voltage*, *Bus Voltage*, arus, daya, dan *Load Voltage*. Proses pengiriman data dilakukan menggunakan fungsi `append()` sehingga setiap data baru akan tersimpan pada baris berikutnya secara otomatis.

```
void sendToGoogleSheets();
timeToString() {
  valueRange.add("majorDimension", "COLUMNS");
  valueRange.set("values/[0]/[0]", waktuStr);
  valueRange.set("values/[1]/[0]", temp);
  valueRange.set("values/[2]/[0]", shuntVoltage);
  valueRange.set("values/[3]/[0]", busVoltage);
  valueRange.set("values/[4]/[0]", Current_mA);
  valueRange.set("values/[5]/[0]", Power_mW);
  valueRange.set("values/[6]/[0]", LoadVoltage);
  bool success = GSheet.values.append
  if (success) {
    valueRange.clear(); Serial.println("Data berhasil dikirim ke
    Google Sheets"); } }
```

4.4.8 Pembuatan Perangkat Lunak Blynk

Sistem Blynk digunakan sebagai interface monitoring pada alat tugas akhir ini. Untuk proses pembuatannya melalui PC/Laptop. Pada PC ketik "blynk.clouds" hingga muncul tampilan seperti pada gambar di bawah. Dikarenakan belum memiliki akun, selanjutnya adalah membuat akun dengan memasukkan email dan password yang akan digunakan.

Klik "Create New Template" pada menu Template. Kemudian masukkan nama projek dan hardware seperti pada gambar. Kemudian pada menu "Datastreams", klik pilihan "New Datastreams". Kemudian pilih "Virtual Pin" seperti pada Gambar. Masukkan virtual pin "Suhu" untuk nilai Tingkat suhu pada baterai. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.10

Suhu

General

Semantic

Units

History

Expiration

Sync

Automations

Visibility

Notes

API reference

General

DATASTREAM NAME: Suhu → DISPLAY NAME: Suhu
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V0 → COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location ●

MIN VALUE: 0 MAX VALUE: 100 DECIMALS: #.## DEFAULT VALUE: 0

Semantic
Semantic types enable Plotly to understand the meaning of the datapoints. This will be used in data associations.

Cancel Save

Gambar 4.10 Memasukkan Virtual Pin “Suhu”

Memasukkan virtual pin "Battery Voltage" untuk nilai tegangan baterai
Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.11

Battery Voltage

General

Semantic

Units

History

Expiration

Sync

Automations

Visibility

Notes

API reference

General

DATASTREAM NAME: Battery Voltage → DISPLAY NAME: Battery Voltage
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V1 → COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location ●

MIN VALUE: 0 MAX VALUE: 20 DECIMALS: #.## DEFAULT VALUE: 0

Semantic
Semantic types enable Plotly to understand the meaning of the datapoints. This will be used in data associations.

Cancel Save

Gambar 4.11 Memasukkan Virtual Pin “*Battery Voltage*”

Memasukkan virtual pin "Shunt Voltage" untuk nilai tegangan shunt pada
baterai Kemudian melakukan setting satuan serta nilai min dan maks seperti
Gambar 4.12

Shunt Voltage

General
 Semantic
 Units
 History
 Expiration
 Sync
 Automations
 Visibility
 Notes
 API reference

General

DATASTREAM NAME: → DISPLAY NAME:
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: COLOR: ICON:

DATA TYPE: ☐ Integer ☐ Double ☐ String ☐ Enum ☒ Location

MIN VALUE: MAX VALUE: DECIMALS: DEFAULT VALUE:

Semantic
Semantic tags enable Dash to understand the meaning of the datastream. This will be used in data operations.

⌵ Battery saver

Gambar 4.12 Memasukkan Virtual Pin “*Shunt Voltage*”

Memasukkan virtual pin "Arus" untuk nilai arus pada baterai Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.13

Arus

General
 Semantic
 Units
 History
 Expiration
 Sync
 Automations
 Visibility
 Notes
 API reference

General

DATASTREAM NAME: → DISPLAY NAME:
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: COLOR: ICON:

DATA TYPE: ☐ Integer ☐ Double ☐ String ☐ Enum ☒ Location

MIN VALUE: MAX VALUE: DECIMALS: DEFAULT VALUE:

Semantic
Semantic tags enable Dash to understand the meaning of the datastream. This will be used in data operations.

Cancel Save

Gambar 4.13 Memasukkan Virtual Pin “Arus”

Memasukkan virtual pin "Daya" untuk nilai daya pada baterai Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.14

Daya

General
Semantic
Units
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: Daya → DISPLAY NAME: Daya
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V4 → COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location •

MIN VALUE: -50000 MAX VALUE: 50000 DECIMALS: ### → DEFAULT VALUE: 0

Semantic

Semantic types enable Daplo to understand the meaning of the datastream. This will be used in data associations.

Cancel Save

Gambar 4.14 Memasukkan Virtual Pin “Daya”

Memasukkan virtual pin "Load Voltage" untuk nilai tegangan pada Solar Charge Controller Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.14

Load Voltage

General
Semantic
Units
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: Load Voltage → DISPLAY NAME: Load Voltage
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V5 → COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location •

MIN VALUE: 0 MAX VALUE: 20 DECIMALS: ### → DEFAULT VALUE: 0

Semantic

Semantic types enable Daplo to understand the meaning of the datastream. This will be used in data associations.

Cancel Save

Gambar 4.15 Memasukkan Virtual Pin “Load Voltage”

Memasukkan virtual pin "RSSI" untuk nilai kekuatan sinyal pengiriman LoRa, Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.16

RSSI

General
Semantic
Units
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: → DISPLAY NAME:
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: COLOR: ICON:

DATA TYPE: ☐ Integer ☐ Double ☐ String ☐ Enum ☒ Location

MIN VALUE: MAX VALUE: DECIMALS: DEFAULT VALUE:

Semantic
Semantic tags enable Dukt to understand the meaning of the datapoint. This will be used in data associations.

Gambar 4.16 Memasukkan Virtual Pin “RSSI”

Memasukkan virtual pin "SoC" untuk nilai presentase baterai, Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.17

SoC

General
Semantic
Units
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: → DISPLAY NAME:
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: COLOR: ICON:

DATA TYPE: ☐ Integer ☐ Double ☐ String ☐ Enum ☒ Location

MIN VALUE: MAX VALUE: DECIMALS: DEFAULT VALUE:

Semantic
Semantic tags enable Dukt to understand the meaning of the datapoint. This will be used in data associations.

Gambar 4.17 Memasukkan Virtual Pin “SoC”

Memasukkan virtual pin "SoH" untuk nilai kondisi kesehatan baterai, Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.18

SoH

General
Semantic
Units
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: SoH → DISPLAY NAME: SoH
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V8 COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location ●

MIN VALUE: 0 MAX VALUE: 100 DECIMALS: ### DEFAULT VALUE: 0

Semantic
Semantic tags enable Dukt to understand the meaning of the datapoint. This will be used in data associations.

Cancel Save

Gambar 4.18 Memasukkan Virtual Pin “SoH”

Memasukkan virtual pin "Status" untuk mengetahui kondisi Charging dan Discharging baterai, Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.19

Status

General
Semantic
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: Status → DISPLAY NAME: Status
For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V9 COLOR: ICON: 

DATA TYPE: Integer Double String Enum Location ●

DEFAULT VALUE: IDLE

Semantic
Semantic tags enable Dukt to understand the meaning of the datapoint. This will be used in data associations.

Cancel Save

Gambar 4.19 Memasukkan Virtual Pin “Status”

Memasukkan virtual pin "Waktu" untuk mengetahui waktu terkini, Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.20

Waktu

General
Semantic
History
Expiration
Sync
Automations
Visibility
Notes
API reference

General

DATASTREAM NAME: Waktu → DISPLAY NAME: Waktu

For your reference and for MQTT topic naming Shown in widgets, dashboards and reports.

VIRTUAL PIN: V10 COLOR: [Dark Grey] ICON: [Icon]

DATA TYPE: Integer Double **String** Enum Location

DEFAULT VALUE: Enter...

Semantic

Specify the semantic. Click to understand the meaning of the datapoint. This will be used in data operations.

Cancel Save

Gambar 4.20 Memasukkan Virtual Pin “Waktu”

Memasukkan memasukkan semua virtual pin yang dibutuhkan, tampilan datastreams akan muncul seperti gambar 4.21. dibawah ini

All 11

Integer

Double

String

Enum

Location

Digital

Analog

Download

Share

☐	Pin	Name	Data type	Semantics	Min	Max	Default	Units	Automations	Actions
	V0	Suhu	Double	+ Add	0	100	0	°C	 None	
	V1	Battery Voltage	Double	+ Add	0	20	0	V	 None	
	V2	Shunt Voltage	Double	+ Add	-500	500	0	mV	 None	
	V3	Arus	Double	+ Add	-3000	3000	0	mA	 None	
	V4	Daya	Double	+ Add	-50000	50000	0	mW	 None	
	V5	Load Voltage	Double	+ Add	0	20	0	V	 None	

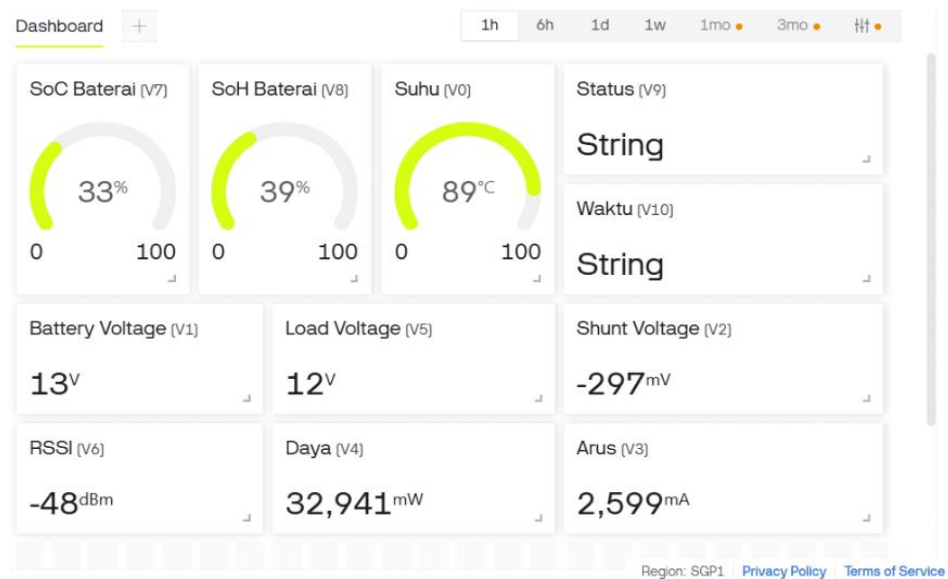
Region: SGP1

Privacy Policy

Terms of Service

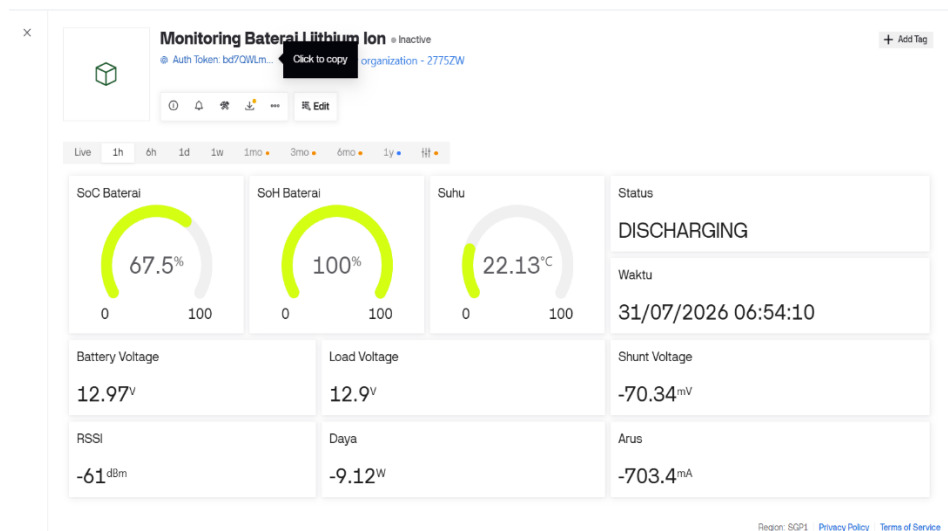
Gambar 4.21 Hasil Datastreams setelah memasukkan Virtual Pin

Lalu setelah datastreams muncul, dilanjutkan dengan pembuatan dashboard untuk melihat parameter seperti gambar 4.22. dibawah ini.



Gambar 4.22 Pembuatan dashboard untuk melihat parameter pada baterai

Setelah dashboard selesai dibuat, dilanjutkan untuk mengcopy code Blynk pada menu device untuk di masukan ke pemograman.4.23. dibawah ini.



Gambar 4.23 Pembuatan code Blynk pada menu device untuk di masukan ke pemograman

Lalu setelah di copy, masukan ke dalam pemograman.

```
1 //-----BLYNK DEFINE (HARUS SEBELUM INCLUDE BLYNK)-----//
2 #define BLYNK_TEMPLATE_ID "TMPL5VDRW2bUYZ8uUdK"
3 #define BLYNK_TEMPLATE_NAME (const char [33])"bd7QWLmp-_1f-QdkQvQDRW2buyZ8uUdK"
4 #define BLYNK_AUTH_TOKEN "bd7QWLmp-_1f-QdkQvQDRW2buyZ8uUdK"
5 #define BLYNK_PRINT Serial
6
```

Gambar 4.24 memasukan code ke dalam pemograman