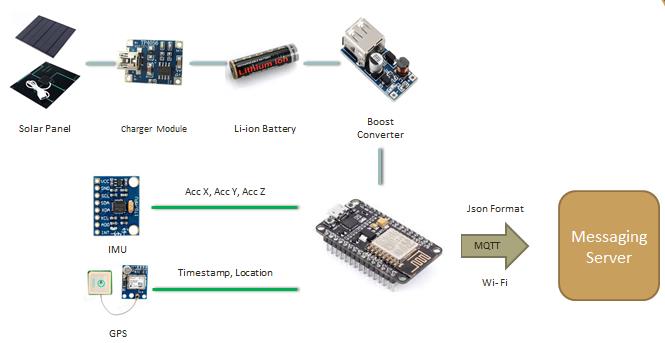
2.1 Sensor Earthquake Catcher Network



Gambar Diagram Blok Sensor ECN

Pada sistem yang kita buat, Sensor ECN berfungsi untuk membaca data seismik, timestamp, dan lokasi dari sensor tersebut. Seperti yang sudah dijabarkan pada dokumen-dokumen sebelumnya, sensor kami memiliki beberapa spesifikasi yang harus dipenuhi yaitu harga sensor yang jauh lebih rendah dari harga seismometer(kisaran 1 juta), peforma sensor yang tidak terganggu dengan kondisi lingkungan sekitarnya, sensor dapat bekerja selama 24 jam tanpa henti, daya yang digunakan sensor rendah (<1W), dan akurasi dari pembacaan sensor getaran tinggi (ADC 14 bit).

Desain sensor ECN terdiri dari tiga bagian, yaitu modul utama, modul charge controller, dan modem Wi-Fi untuk komunikasi. Modul utama berfungsi untuk membaca data-data yang diperlukan seperti data getaran (percepatan), Timestamp, dan Lokasi dari sensor dan mengirimkannya menggunakan protokol MQTT ke Messaging server (cloudAMQP) dengan menggunakan Wi-Fi, Modul Charge Controller berfungsi sebagai penyedia sumber daya yang terus-menerus bekerja untuk modul utama, sedangkan modem Wi-Fi digunakan untuk menyediakan Wi-Fi access point untuk modul utama yang bekerja secara terus menerus.

2.1.1 Main Module

2.1.1.1 Elektrikal

Terdapat tiga komponen utama pada main module, yaitu NodeMCU v2, MPU6050, dan GPS Ublox Neo 6M.



Gambar NodeMCU v2

NodeMCU merupakan mikrokontroller yang sudah built-in modul Wi-Fi sehingga tidak diperlukan modul Wi-Fi tambahan. NodeMCU digunakan sebagai mikrokontroler yang bertugas untuk melakukan proses sampling data percepatan sebanyak 20 kali selama 0,5 detik, membaca data lokasi dan timestamp yang diberikan oleh GPS, lalu mengirimkan data-data tersebut ke server dengan menggunakan protokol MQTT melalui jaringan Wi-Fi.



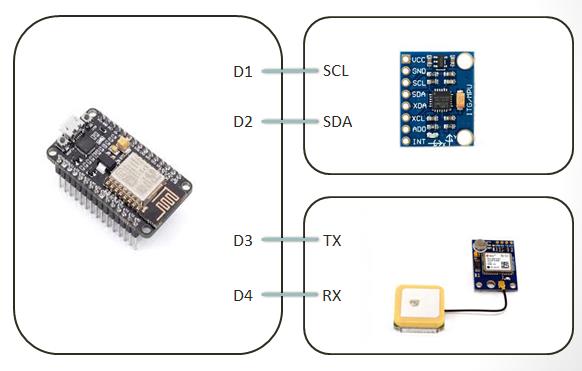
Gambar GPS Ublox Neo 6M

Untuk mendapatkan lokasi sensor dan data waktu yang dapat sinkron dengan sensor-sensor yang lain, digunakan modul GPS Ublox Neo 6M. Modul GPS ini terus menerus mendapat informasi dari satelit dan informasi-informasi tersebut dapat diperoleh NodeMCU dengan menggunakan komunikasi serial. Gelombang yang diterima oleh antena GPS ini memiliki daya tembus yang rendah sehingga modul GPS ini harus diletakkan di tempat yang tidak terhalang oleh tembok/atap. Data–data yang dapat diperoleh dari modul GPS ini adalah data latitude dan longitude serta data-data waktu yang tersinkronisasi seperti detik, menit, jam, tanggal, bulan, dan tahun



Gambar Sensor IMU

Modul yang digunakan untuk mendapatkan data getaran adalah modul MPU6050. Pada modul MPU6050 ini terdapat accelerometer sehingga dapat diperoleh data percepatan dari getaran. Komunikasi yang digunakan modul MPU6050 ini adalah I2C. Inisisasi pembacaan data dari MPU6050 dilakukan dengan cara wake-up MPU6050 atau dengan menuliskan bit 0 ke register 0x6B. Setelah dilakukan inisiasi, data percepatan x, y, dan z dapat diperoleh dengan membaca register 0x3B sampai 0x3F.

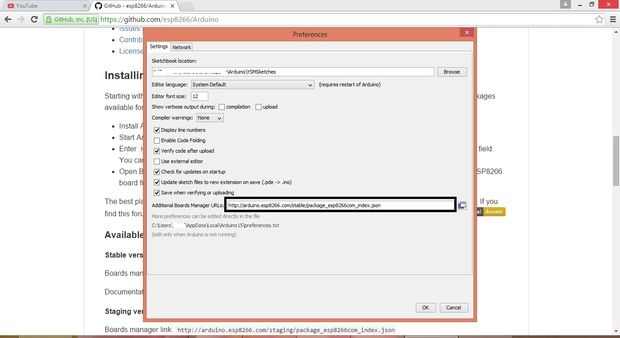


Gambar Skematik Modul Utama Sensor ECN

Untuk interkoneksi antar komponen dapat dilihat pada gambar diatas. Pin SCL pada MPU6050 dihubungkan dengan pin D1 pada NodeMCU dan pin SDA pada MPU6050 dihubungkan dengan pin D2 pada NodeMCU agar komunikasi I2C dapat dijalankan. Sedangkan pin Tx pada GPS dihubungkan dengan pin D3 pada NodeMCU dan pin Rx pada GPS dihubungkan dengan pin D4 pada NodeMCU. Pin D3 dan D4 merupakan pin yang dibuat menjadi pin Rx dan Tx dengan menggunakan software serial. Tidak digunakannya pin Rx dan Tx yang sudah ada pada NodeMCU karena pin Rx dan Tx tersebut sudah digunakan untuk serial monitor sehingga proses debugging lebih mudah dilakukan.

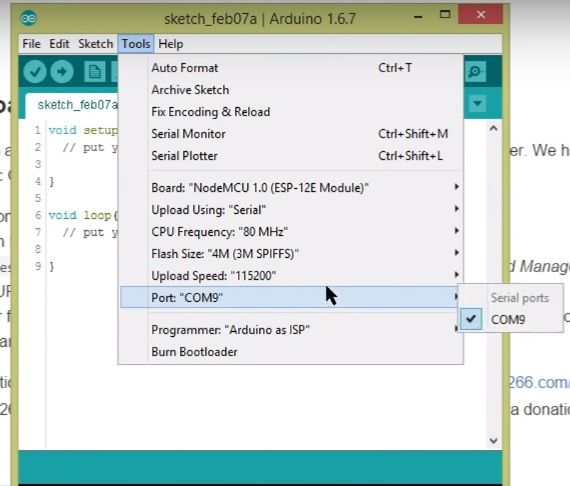
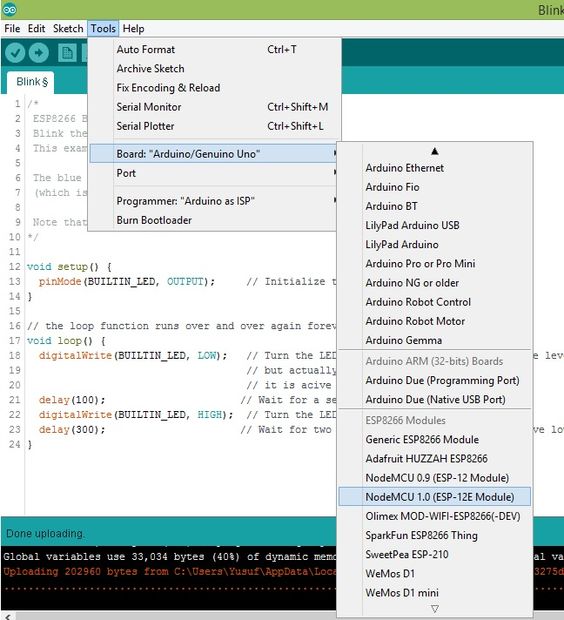
2.1.1.2 Algoritma

Digunakan IDE Arduino untuk mempermudah proses debugging dan upload program ke NodeMCU. Langkah pertama yang dilakukan adalah pada bagian preferences pada IDE Arduino, kolom Additional Board Manager URLs diisi dengan <http://arduino.esp8266.com/stable/package_esp8266com_index.json>



Gambar Tampilan untuk Menambahkan Board pada IDE Arduino

Setelah itu IDE Arduino harus di-restart. Setelah IDE Arduino dibuka kembali, board NodeMCU sudah terinstall pada IDE Arduino. Langkah selanjutnya yang dilakukan adalah memilih board pada bagian Tools -> Board.



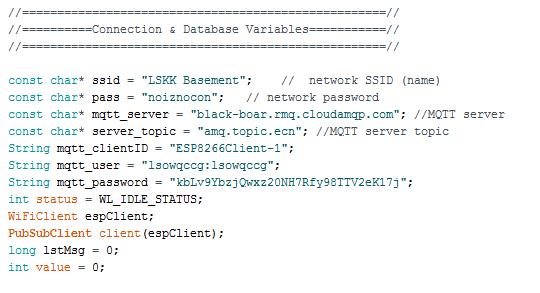
Gambar Tampilan untuk Setting Board dan Port

Setelah itu pilih Port pada bagian Tools -> Port. Setelah langkah-langkah tersebut dilakukan, proses kompilasi dan upload program ke NodeMCU dapat dilakukan pada IDE Arduino.



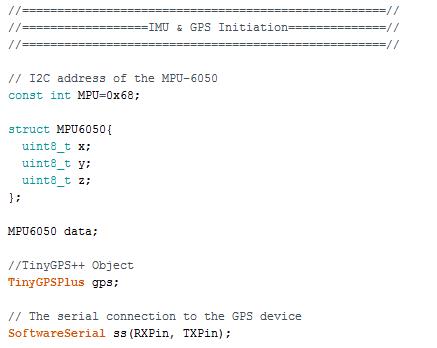
Gambar Potongan Source Code Include Library & Deklarasi Konstanta

Pada algoritma awal, dilakukan beberapa inisiasi seperti include library dan pendefinisian konstanta. Pada sensor ECN ini digunakan 5 library yaitu library ESP8266Wifi untuk koneksi Wi-Fi pada NodeMCU, PubSubClient untuk protokol MQTT, SoftwareSerial untuk deklarasi pin GPIO sebagai pin Serial tambahan yang digunakan untuk modul GPS, Wire untuk komunikasi I2C yang digunakan untuk memperoleh data dari MPU6050, dan Library TinyGPS++ untuk pembacaan data yang diperoleh dari GPS. Selain itu dilakukan juga deklarasi pin yang digunakan untuk komunikasi I2C dan Serial serta deklarasi register-register pada MPU6050. Konstanta IMU\_RES merupakan resolusi data percepatan pada sensor IMU, diberi nilai 1 karena data yang ingin dikirimkan memiliki tipe data integer pembacaan ADC pada IMU. Konstanta GPSBaud merupakan baudrate yang digunakan untuk berkomunikasi serial dengan modul GPS. Konstanta SendPeriod merupakan periode pengiriman message.



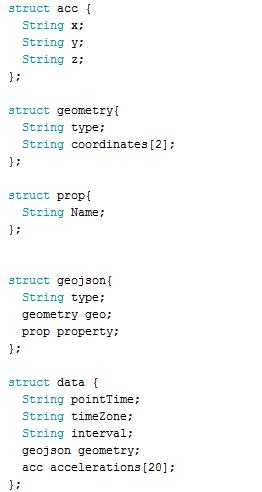
Gambar Potongan Source Code Deklarasi Variabel Koneksi dan MQTT

Untuk melakukan pengiriman diperlukan beberapa variabel yang harus digunakan. Untuk melakukan koneksi Wi-Fi dengan menggunakan NodeMCU hanya diperlukan dua variabel yaitu ssid dan pass yang bertipe array of constant character. Dua variabel ini menandakan access point mana yang ingin digunakan oleh NodeMCU. Sedangkan untuk pengaturan server MQTT digunakan variabel mqtt\_server, server\_topic, mqtt\_clientID, mqtt\_user, dan mqtt\_password. Variabel mqtt\_server digunakan untuk mendeklarasikan server manayang ingin digunakan pada CloudAMQP. Variabel server\_topic digunakan untuk mendeklarasikan topic mana yang ingin dikirimkan message pada server. Format dari variabel ini adalah “x.t” dimana “x” merupakan nama exchange yang digunakan dan “t” merupakan nama topic yang digunakan pada CloudAMQP. Sedangkan variabel mqtt\_clientID digunakan untuk mendeklarasikan nama client ketika melakukan koneksi ke server ClousAMQP. Nama yang digunakan boleh bebas asal tidak sama antara satu sensor dengan sensor yang lainnya. Variabel mqtt\_user dan mqtt\_password digunakan untuk mendeklarasikan username dari sensor beserta passwordnya. Format username yang digunakan adalah “v.u”, dimana “v” merupakan nama vhost yang digunakan dan “u” merupakan nama username yang digunakan. Dalam kasus dengan menggunakan CloudAMQP ini nama vhost dan username yang digunakan sama. Sedangkan espClient merupakan objek yang digunakan pada fungsi-fungsi pada library ESP8266WiFi dan objek PubSubClient merupakan hasil dari fungsi client pada library PubSubClient dengan input objek WiFiClient espClient.



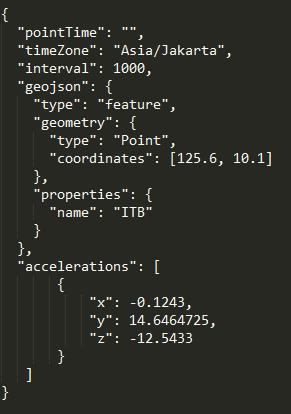
Gambar Potongan Source Code Deklarasi Variabel IMU

Pada bagian dari potongan source code yang digunakan diatas, dilakukan deklarasi beberapa variabel yang berhubungan dengan proses pembacaan data dari sensor IMU dan GPS. Konstanta MPU mengindikasikan alamat I2C dari MPU6050. Sedangkan struct MPU6050 dibuat untuk mempermudah parsing data ketika melakukan pembacaan dari sensor IMU. Lalu dilakukan juga inisiasi objek TinyGPSPlus dan SoftwareSerial yang digunakan pada library TinyGPS++ untuk proses pembacaan data yang diterima dari GPS.

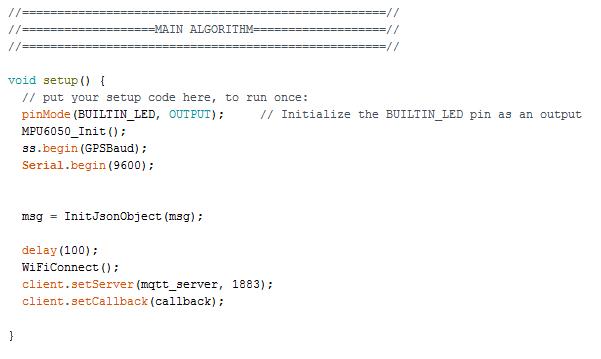


Gambar Potongan Source Code Deklarasi Objek Json

Dilakukan juga deklarasi objek yang berkolerasi dengan format message json yang ingin dikirimkan ke server. Format Json yang digunakan dapat dilihat pada gambar dibawah ini.

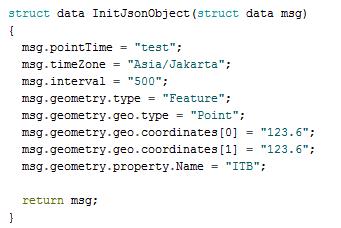


Gambar Format Message Json yang Digunakan



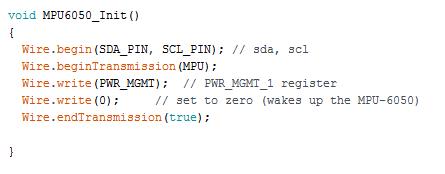
Gambar Potongan Source Code Setup

Pada bagian setup Arduino, dilakukan beberapa hal. Dilakukan deklarasi port built in LED sebagai port output. Setelah itu dilakukan inisiasi proses pembacaan data dari MPU6050 dan deklarasi baudrate pada port serial tambahan dan serial pada NodeMCU, dan deklarasi nilai dari setiap objek Json. Setelah itu dilakukan proses inisiasi koneksi Wi-Fi dan pengesetan server dan port pada protokol MQTT. Fungsi setCallback digunakan ketika device men-subscribe message dari server, karena sensor ECN tidak men-subscribe apapun dari server sehingga fungsi callback dapat dihiraukan.



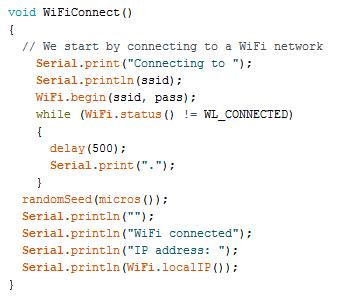
Gambar Potongan Source Code Fungsi InitJsonObject

Dapat dilihat dari gambar diatas, fungsi InitJsonObject memiliki input objek Json dan output objek Json. Pada fungsi ini dilakukan deklarasi awal beberapa variabel dari objek tersebut. Yang dideklarasikan antara lain zona waktu yang digunakan, interval pengiriman, dan lokasi dari sensor. Latitude dan Longitude perlu dideklarasikan asal terlebih dahulu untuk menghindari error pada fungsi pembentukan string dengan format Json.



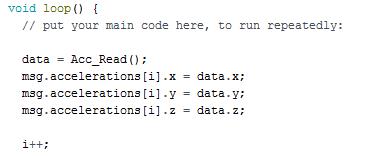
Gambar Potongan Source Code Fungsi MPU6050\_Init

Pada fungsi MPU6050\_Init dilakukan deklarasi port SDA dan SCL yang digunakan pada NodeMCU. Setelah itu dimulai transmisi pada alamat register I2C MPU6050. Lalu dilakukan penulisan bit 0 pada register 0x6B. Lalu koneksi I2C antara NodeMCU dengan MPU6050 dihentikan.



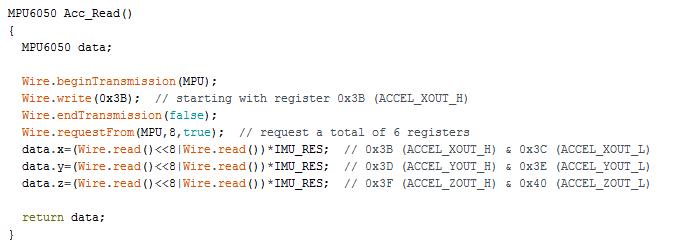
Gambar Potongan Source Code Fungsi WiFiConnect

Pada fungsi WiFiConnect dilakukan proses inisiasi koneksi antara NodeMCU dengan Access Point. Fungsi ini akan terus menerus berjalan sampai ada Access Point yang terhubung. Status dari proses ini dapat dilihat pada serial monitor pada IDE Arduino.



Gambar Potongan Source Code Loop

Gambar diatas merupakan implementasi algoritma loop dari sensor ECN. Hal yang pertama kali dilakukan pada bagian loop adalah pembacaan data percepatan dari IMU dengan menggunakan fungsi Acc\_Read. Jumlah sampling yang diambil berjumlah 20 dengan delay selama 25 ms setiap proses sampling dilakukan. Ketika proses sampling dilakukan, selalu dilakukan pengecekan apakah server masih terhubung dengan sensor ECN. Ketika sudah tidak terhubung maka proses sampling akan berhenti dan akan dilanjutkan untuk melakukan koneksi ulang ke server dengan menggunakan fungsi reconnect\_server. Ketika proses sampling telah selesai atau i==20, sensorECN akan mengirimkan message ke server dengan menggunakan fungsi publish pada library PubSubClient. Sebelum dilakukan pengiriman, dilakukan dulu pembentukan format Json dari objek Json yang telah dibuat pada sebelumnya (msg) dengan menggunakan fungsi JsonToString. Setelah itu string hasil pembentukan format Json tersebut diubah menjadi array of character dengan menggunakan fungsi bawaan dari Arduino toCharArray.



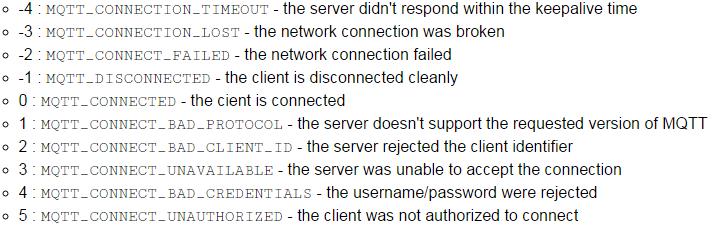
Gambar Potongan Source Code Fungsi Acc\_Read

Pada fungsi Acc\_Read dilakukan beberapa proses dalam pembacaan data percepatan dengan menggunakan MPU6050. Hal pertama yang dilakukan adalah memulai transmisi komunikasi I2C ke alamat I2C dari MPU6050. Setelah itu dilakukan pembacaaan dari register 0x3B sampai 0x40 dan disimpan pada objek data. Dapat dilihat karena satu register hanya dapat menyimpan 8 bit sedangkan pembacaan dari MPU6050 menggunakan 16 bit ADC sehingga untuk setiap salah satu axis percepatan diwakili dengan dua register pada MPU6050. Sehingga diperlukan proses shift left sebanyak 8 kali pada register yang menyimpan pembacaan MSB dari ADC MPU6050.

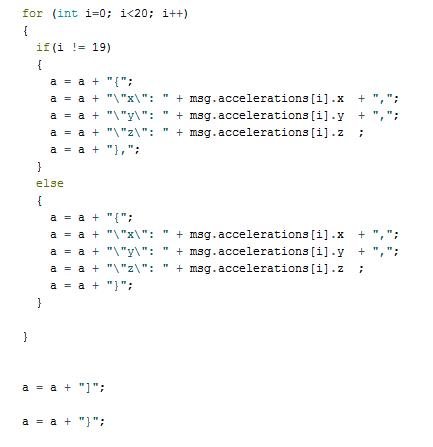
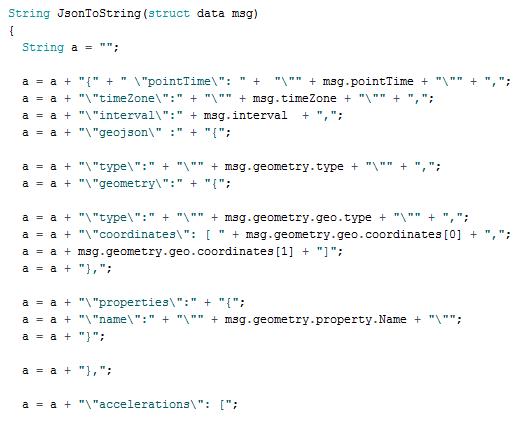


Gambar Potongan Source Code Fungsi reconnect\_server

Pada fungsi reconnect\_server dilakukan proses inisasi koneksi ulang lagi ke messaging server (CloudAMQP). Proses ini terus berlangsung sampai NodeMCU dapat terkoneksi ke server lagi. Status dari proses reconnect server ini dapat dilihar pada serial monitor. Penyebab error dalam percobaan koneksi ke server dapat dilihat dari nilai rc-nya. Informasi mengenai alasan error-nya sangat berguna untuk melakukan proses debugging. Keterangan mengenai nilai rc dapat dilihat pada gambar berikut.



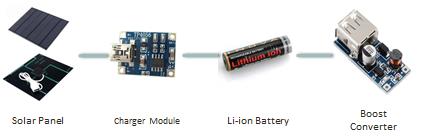
Gambar Informasi Mengenai Nilai RC dan Penjelasannya



Gambar Potongan Source Code Fungsi JsonToString

Fungsi JsonToString memiliki input objek Json yang sudah dideklarasikan sebelumnya dengan output dengan tipe data string. Tidak digunakannya library untuk encode dan decode string format Json yang tersedia dikarenakan tingkat kerumitan format Json yang kita gunakan tidak dapat di-handle oleh fungsi-fungsi tersebut, oleh karena itu dibuat fungsi encode format Json ini. Langkah yang dilakukan sangat sederhana, yaitu hanya menyusun string sesuai dengan format Json sehingga terbentuk string yang bersesuaian dengan objek Json yang kita gunakan. Dilakukan proses penambahan string a sesuai dengan format Json yang diinginkan. Perhatikan bahwa “\”” digunakan untuk mencetak string petik awal atau akhir.

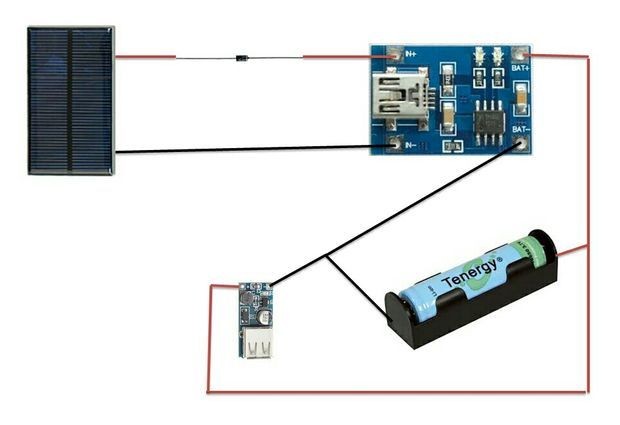
2.1.1 Charge Controller Module



Gambar Diagram Blok Charge Controller Module

Untuk membuat sensor ECN dapat berjalan secara terus menerus diperlukannya sumber daya yang dapat diperbaharui untuk menyuplai sensor ECN. Digunakan solar panel karena mudah untuk mengaksesnya serta cukup sederhana dalam proses implementasinya serta harga yang tidak terlalu mahal.

Digunakan solar panel dengan keluaran 5 Volt dan daya 4W sehingga solar panel memiliki kemampuan untuk menyuplai arus yang diperlukan NodeMCU (0,2A) dan charging baterai (0,6A) agar baterai dapat menyuplai sensor pada malam hari. Baterai yang digunakan untuk menyimpan daya adalah 2 baterai Li-Ion dengan kapasitas masing-masing adalah 2200 mAh. Pemilihan Li-Ion sendiri karena lebih tidak mudah rusaknya baterai ketika digunakan jika dibandingkan dengan baterai-baterai yang dapat di-charge lainnya seperti NiMH dan NiCD. Untuk mengatur tingkat charging baterai, digunakan modul Li-Ion battery charge module TP4056. Agar stabilnya tegangan yang diberikan oleh charge module ini, diperlukan modul boost converter. Digunakan modul boost converter dengan range input dari 0,9V – 5V dan arus keluaran antara 0,2 – 0,3 A. Untuk koneksi antar modulnya dapat dilihat pada gambar dibawah ini.



Gambar Skematik Modul Charge Controller

**LAMPIRAN**

|  |
| --- |
| /\*  \* Earthquake Catcher Network v1.0  \* Capstone Design by  \* Christoporus Deo Putratama  \* Kevin Shidqi  \* Bramantio Yuwono  \*  \* Read seismic waves using IMU sensor  \* Read location and exact time using GPS  \* Sending those data using Wi-Fi using MQTT  \*  \*/  //====================================================//  //=================Library & Constant=================//  //====================================================//  #include <ESP8266WiFi.h>  #include <PubSubClient.h>  #include <SoftwareSerial.h>  #include <Wire.h>  #include <TinyGPS++.h>  #define SDA\_PIN D2  #define SCL\_PIN D1  #define PWR\_MGMT 0x6B  #define IMU\_RES 1  #define RXPin D3  #define TXPin D4  #define GPSBaud 9600  #define SendPeriod 500 //in ms  //====================================================//  //==========Connection & Database Variables===========//  //====================================================//  const char**\*** ssid **=** "LSKK Basement"**;** // network SSID (name)  const char**\*** pass **=** "noiznocon"**;** // network password  const char**\*** mqtt\_server **=** "black-boar.rmq.cloudamqp.com"**;** //MQTT server  const char**\*** server\_topic **=** "amq.topic.ecn"**;** //MQTT server topic  String mqtt\_clientID **=** "ESP8266Client-1"**;**  String mqtt\_user **=** "lsowqccg:lsowqccg"**;**  String mqtt\_password **=** "kbLv9YbzjQwxz20NH7Rfy98TTV2eK17j"**;**  int status **=** WL\_IDLE\_STATUS**;**  WiFiClient espClient**;**  PubSubClient client**(**espClient**);**  long lstMsg **=** 0**;**  int value **=** 0**;**  //====================================================//  //==================IMU & GPS Initiation==============//  //====================================================//  // I2C address of the MPU-6050  const int MPU**=**0x68**;**  struct MPU6050**{**  uint8\_t x**;**  uint8\_t y**;**  uint8\_t z**;**  **};**  MPU6050 data**;**  //TinyGPS++ Object  TinyGPSPlus gps**;**  // The serial connection to the GPS device  SoftwareSerial ss**(**RXPin**,** TXPin**);**  //===================================================//  //===================JSON OBJECT=====================//  //===================================================//  struct acc **{**  String x**;**  String y**;**  String z**;**  **};**  struct geometry**{**  String type**;**  String coordinates**[**2**];**  **};**  struct prop**{**  String Name**;**  **};**  struct geojson**{**  String type**;**  geometry geo**;**  prop property**;**  **};**  struct data **{**  String pointTime**;**  String timeZone**;**  String interval**;**  geojson geometry**;**  acc accelerations**[**20**];**  **};**  struct data msg**;**  struct data InitJsonObject**(**struct data msg**);**  String JsonToString**(**struct data msg**);**  int i **=** 0**;**  //====================================================//  //===================MAIN ALGORITHM===================//  //====================================================//  void setup**()** **{**  // put your setup code here, to run once:  pinMode**(**BUILTIN\_LED**,** OUTPUT**);** // Initialize the BUILTIN\_LED pin as an output  MPU6050\_Init**();**  ss**.**begin**(**GPSBaud**);**  Serial**.**begin**(**9600**);**  msg **=** InitJsonObject**(**msg**);**    delay**(**100**);**  WiFiConnect**();**  client**.**setServer**(**mqtt\_server**,** 1883**);**  client**.**setCallback**(**callback**);**    **}**  void loop**()** **{**  // put your main code here, to run repeatedly:    data **=** Acc\_Read**();**  msg**.**accelerations**[**i**].**x **=** data**.**x**;**  msg**.**accelerations**[**i**].**y **=** data**.**y**;**  msg**.**accelerations**[**i**].**z **=** data**.**z**;**    i**++;**  **if** **(!**client**.**connected**())** **{**  reconnect\_server**();**  **}**  client**.**loop**();**  long now **=** millis**();**  //if (now - lstMsg > SendPeriod)  **if(**i**==**20**)**  **{**  i **=** 0**;**  **while** **(**ss**.**available**()** **>** 0**)**  **if** **(**gps**.**encode**(**ss**.**read**()))**  displayInfo**();**  **else**  Serial**.**println**(**"INVALID"**);**    //lstMsg = now;  String message **=** JsonToString**(**msg**);**  //msg.printTo(message);  char message\_t**[**800**];**  message**.**toCharArray**(**message\_t**,**800**);**  //publish sensor data to MQTT broker  bool test **=** client**.**publish**(**server\_topic**,** message\_t**);**  **if(**test**)**  Serial**.**println**(**"publish success"**);**  **}**  delay**(**25**);**  **}**  //====================================================//  //======Wi-Fi Connection & MQTT Function Procedure====//  //====================================================//  void WiFiConnect**()**  **{**  // We start by connecting to a WiFi network  Serial**.**print**(**"Connecting to "**);**  Serial**.**println**(**ssid**);**  WiFi**.**begin**(**ssid**,** pass**);**  **while** **(**WiFi**.**status**()** **!=** WL\_CONNECTED**)**  **{**  delay**(**500**);**  Serial**.**print**(**"."**);**  **}**  randomSeed**(**micros**());**  Serial**.**println**(**""**);**  Serial**.**println**(**"WiFi connected"**);**  Serial**.**println**(**"IP address: "**);**  Serial**.**println**(**WiFi**.**localIP**());**  **}**  void reconnect\_server**()** **{**  // Loop until we're reconnected  **while** **(!**client**.**connected**())**  **{**  Serial**.**print**(**"Attempting MQTT connection..."**);**  // Create a random client ID  String clientId **=** "ESP8266Client-"**;**  clientId **+=** String**(**random**(**0xffff**),** HEX**);**  // Attempt to connect  //if you MQTT broker has clientID,username and password  //please change following line to if (client.connect(clientId,userName,passWord))  **if** **(**client**.**connect**(**mqtt\_clientID**.**c\_str**(),** mqtt\_user**.**c\_str**(),**mqtt\_password**.**c\_str**()))**  **{**  Serial**.**println**(**"connected"**);**  **}** **else** **{**  Serial**.**print**(**"failed, rc="**);**  Serial**.**print**(**client**.**state**());**  Serial**.**println**(**" try again in 5 seconds"**);**  // Wait 6 seconds before retrying  delay**(**6000**);**  **}**  **}**  **}** //end reconnect()  void callback**(**char**\*** topic**,** byte**\*** payload**,** unsigned int length**)**  **{**  Serial**.**print**(**"Message arrived ["**);**  Serial**.**print**(**topic**);**  Serial**.**print**(**"] "**);**  **for** **(**int i **=** 0**;** i **<** length**;** i**++)** **{**  Serial**.**print**((**char**)**payload**[**i**]);**  **}**  Serial**.**println**();**  // Switch on the LED if an 1 was received as first character  **if** **((**char**)**payload**[**0**]** **==** '1'**)** **{**  digitalWrite**(**BUILTIN\_LED**,** LOW**);** // Turn the LED on (Note that LOW is the voltage level  // but actually the LED is on; this is because  // it is acive low on the ESP-01)  **}** **else** **{**  digitalWrite**(**BUILTIN\_LED**,** HIGH**);** // Turn the LED off by making the voltage HIGH  **}**  **}**  //====================================================//  //==============IMU & GPS Function Procedure==========//  //====================================================//  void MPU6050\_Init**()**  **{**  Wire**.**begin**(**SDA\_PIN**,** SCL\_PIN**);** // sda, scl  Wire**.**beginTransmission**(**MPU**);**  Wire**.**write**(**PWR\_MGMT**);** // PWR\_MGMT\_1 register  Wire**.**write**(**0**);** // set to zero (wakes up the MPU-6050)  Wire**.**endTransmission**(true);**  **}**  MPU6050 Acc\_Read**()**  **{**  MPU6050 data**;**    Wire**.**beginTransmission**(**MPU**);**  Wire**.**write**(**0x3B**);** // starting with register 0x3B (ACCEL\_XOUT\_H)  Wire**.**endTransmission**(false);**  Wire**.**requestFrom**(**MPU**,**8**,true);** // request a total of 6 registers  data**.**x**=(**Wire**.**read**()<<**8**|**Wire**.**read**())\***IMU\_RES**;** // 0x3B (ACCEL\_XOUT\_H) & 0x3C (ACCEL\_XOUT\_L)  data**.**y**=(**Wire**.**read**()<<**8**|**Wire**.**read**())\***IMU\_RES**;** // 0x3D (ACCEL\_YOUT\_H) & 0x3E (ACCEL\_YOUT\_L)  data**.**z**=(**Wire**.**read**()<<**8**|**Wire**.**read**())\***IMU\_RES**;** // 0x3F (ACCEL\_ZOUT\_H) & 0x40 (ACCEL\_ZOUT\_L)    **return** data**;**  **}**  void displayInfo**()**  **{**  **if(**gps**.**location**.**isValid**())**  **{**  msg**.**geometry**.**geo**.**coordinates**[**0**]** **=** String**(**gps**.**location**.**lat**(),**3**);**  msg**.**geometry**.**geo**.**coordinates**[**1**]** **=** String**(**gps**.**location**.**lng**(),**3**);**  Serial**.**println**(**gps**.**location**.**lat**(),**3**);**  Serial**.**println**(**gps**.**location**.**lng**(),**3**);**  **}**  **else**  **{**  Serial**.**println**(**"INVALID"**);**  **}**  **if(**gps**.**date**.**isValid**()** **&&** gps**.**time**.**isValid**())**  **{**  String YEAR **=** String**(**gps**.**date**.**year**());**  String MONTH **=** String**(**gps**.**date**.**month**());**  String DATE **=** String**(**gps**.**date**.**day**());**  String HOUR **=** String**(**gps**.**time**.**hour**());**  String MINUTE **=** String**(**gps**.**time**.**minute**());**  String SECOND **=** String**(**gps**.**time**.**second**());**  msg**.**pointTime **=** YEAR **+** "-" **+** MONTH **+** "-" **+** DATE **+** "T" **+** HOUR **+** ":" **+** MINUTE **+** ":"**+** SECOND **+** "Z"**;**  **}**  **else**  **{**  Serial**.**println**(**"INVALID"**);**  **}**  **}**  //====================================================//  //==================ENCODE JSON FUNCTION==============//  //====================================================//  struct data InitJsonObject**(**struct data msg**)**  **{**  msg**.**pointTime **=** "test"**;**  msg**.**timeZone **=** "Asia/Jakarta"**;**  msg**.**interval **=** "500"**;**  msg**.**geometry**.**type **=** "Feature"**;**  msg**.**geometry**.**geo**.**type **=** "Point"**;**  msg**.**geometry**.**geo**.**coordinates**[**0**]** **=** "123.6"**;**  msg**.**geometry**.**geo**.**coordinates**[**1**]** **=** "123.6"**;**  msg**.**geometry**.**property**.**Name **=** "ITB"**;**  **return** msg**;**  **}**  String JsonToString**(**struct data msg**)**  **{**  String a **=** ""**;**  a **=** a **+** "{" **+** " \"pointTime\": " **+** "\"" **+** msg**.**pointTime **+** "\"" **+** ","**;**  a **=** a **+** "\"timeZone\":" **+** "\"" **+** msg**.**timeZone **+** "\"" **+** ","**;**  a **=** a **+** "\"interval\":" **+** msg**.**interval **+** ","**;**  a **=** a **+** "\"geojson\" :" **+** "{"**;**  a **=** a **+** "\"type\":" **+** "\"" **+** msg**.**geometry**.**type **+** "\"" **+** ","**;**  a **=** a **+** "\"geometry\":" **+** "{"**;**  a **=** a **+** "\"type\":" **+** "\"" **+** msg**.**geometry**.**geo**.**type **+** "\"" **+** ","**;**  a **=** a **+** "\"coordinates\": [ " **+** msg**.**geometry**.**geo**.**coordinates**[**0**]** **+** ","**;**  a **=** a **+** msg**.**geometry**.**geo**.**coordinates**[**1**]** **+** "]"**;**  a **=** a **+** "},"**;**  a **=** a **+** "\"properties\":" **+** "{"**;**  a **=** a **+** "\"name\":" **+** "\"" **+** msg**.**geometry**.**property**.**Name **+** "\""**;**  a **=** a **+** "}"**;**  a **=** a **+** "},"**;**  a **=** a **+** "\"accelerations\": ["**;**  **for** **(**int i**=**0**;** i**<**20**;** i**++)**  **{**  **if(**i **!=** 19**)**  **{**  a **=** a **+** "{"**;**  a **=** a **+** "\"x\": " **+** msg**.**accelerations**[**i**].**x **+** ","**;**  a **=** a **+** "\"y\": " **+** msg**.**accelerations**[**i**].**y **+** ","**;**  a **=** a **+** "\"z\": " **+** msg**.**accelerations**[**i**].**z **;**  a **=** a **+** "},"**;**  **}**  **else**  **{**  a **=** a **+** "{"**;**  a **=** a **+** "\"x\": " **+** msg**.**accelerations**[**i**].**x **+** ","**;**  a **=** a **+** "\"y\": " **+** msg**.**accelerations**[**i**].**y **+** ","**;**  a **=** a **+** "\"z\": " **+** msg**.**accelerations**[**i**].**z **;**  a **=** a **+** "}"**;**  **}**    **}**      a **=** a **+** "]"**;**  a **=** a **+** "}"**;**      **return** a**;**  **}** |