INSTITUT TEKNOLOGI BANDUNG
PROGRAM STUDI TEKNIK ELEKTRO
JALAN GANESHA NO. 10 Gedung Labtek V Lantai 2 ☎ (022)2508135-36, 🖷 (022)2500940
BANDUNG 40132

Dokumentasi Produk Tugas Akhir

Lembar Sampul Dokumen

Judul Dokumen	**TUGAS AKHIR TEKNIK ELEKTRO:** *e-Shrimp: Sistem Kontrol Pintar untuk Tambak Udang Vanamei dengan menggunakan Multi Sensor*
Jenis Dokumen	**IMPLEMENTASI**
	Catatan: Dokumen ini dikendalikan penyebarannya oleh Prodi Teknik Elektro ITB
Nomor Dokumen	**B400-02-TA161701060**
Nomor Revisi	**Versi 02**
Nama File	**B400-02-TA161701060.docx**
Tanggal Penerbitan	**4 Mei 2017**
Unit Penerbit	**Prodi Teknik Elektro - ITB**
Jumlah Halaman	**146** (termasuk lembar sampul ini)

Data Pemeriksaan dan Persetujuan				
Ditulis Oleh	Nama	Daniel Anugrah Wiranata	Jabatan	Ketua
	Tanggal	4 Mei 2017	Tanda Tangan	
	Nama	Edwin Sanjaya	Jabatan	Anggota
	Tanggal	4 Mei 2017	Tanda Tangan	
	Nama	Marcel	Jabatan	Anggota
	Tanggal	4 Mei 2017	Tanda Tangan	
Diperiksa Oleh	Nama	Elvayandri, S.Si, MT	Jabatan	Koordinator Pembimbing
	Tanggal	4 Mei 2017	Tanda Tangan	
	Nama	Ir. Farkhad Ihsan Hariadi, M.Sc.	Jabatan	Pembimbing
	Tanggal	4 Mei 2017	Tanda Tangan	

Disetujui	Nama	Elvayandri, S.Si, MT	Jabatan	Koordinator Pembimbing
Oleh	Tanggal	4 Mei 2017	Tanda Tangan	
	Nama	Ir. Farkhad Ihsan Hariadi, M.Sc.	Jabatan	Pembimbing
	Tanggal	4 Mei 2017	Tanda Tangan	

DAFTAR ISI

DAFTAR GAMBAR

Catatan Sejarah Perbaikan Dokumen

VERSI, TGL, OLEH	PERBAIKAN

Implementasi Riset e-Shrimp : Sistem Kontrol Pintar dengan Multisensor untuk Tambak Udang Vannamei

1 PENGANTAR

1.1 RINGKASAN ISI DOKUMEN

Dokumen isi berisi implementasi dari **e-Shrimp**: Sistem Kontrol Pintar dengan Multisensor untuk Tambak Udang Vannamei. Isi dari dokumen desain ini antara-lain *overview* dari dokumen B200 dan B300 mengenai spesifikasi dan desain dari alat yang dijadikan referensi sebagai dasar implementasi alat-alat yang akan dibuat pada riset.

Pada dokumen ini juga dilampirkan tinjauan dari spesifikasi dan desain dari setiap alat yang dibuat beserta hasil implementasi dari setiap alat dalam bentuk *hardware* dan *software*. Implementasi *hardware* dijabarkan dalam bentuk rangkaian atau skematik, sedangkan untuk *software,* dilampirkan dalam bentuk source code dalam bahasa Arduino.

Dokumen ini juga dilengkapi dengan beberapa kendala yang dihadapi dalam proses pembuatan dan implementasi alat di riset ini beserta solusi yang dilakukan untuk mengatasinya, sehingga diharapkan agar dokumen ini dapat dijadikan referensi untuk pembuatan alat, *troubleshooting*, dan tidak mengulangi kesalah-kesalahan yang dilakukan pada riset untuk e-Shrimp kedepannya.

1.2 TUJUAN PENULISAN DAN APLIKASI DOKUMEN

Tujuan dari penulisan dokumen ini adalah sebagai berikut :

1. Sebagai dokumen implementasi yang menjelaskan prosedur dan hasil pembuatan hardware dan software untuk riset Sistem Kontrol Pintar dengan Multisensor untuk Tambak Udang Vannamei
2. Sebagai landasan dalam membuat prosedur pengujian Sistem Kontrol Pintar dengan Multisensor untuk Tambak Udang Vannamei untuk kedepannya
3. Sebagai dokumen yang memaparkan beberapa kendala yang telah dihadapi dan solusi yang telah dilakukan selama proses implementasi, yang dapat dijadikan penelitian kedepannya

1.3 REFERENSI

Referensi yang digunakan dalam pembuatan dokumen spesifikasi riset ini adalah sebagai berikut :

1. BS EN 60529 *: Degrees of protection provided by enclosures (IP code)*, British Standards Institution, 1992
2. SNI 01-7246-2006 tentang *Produksi udang vaname (Litopenaeus vannamei) di tambak dengan teknologi intensif*, Badan Standarisasi Nasional, 2006

1.4 DAFTAR SINGKATAN

SINGKATAN	ARTI
MCU	*Microcontroller Unit*
HMI	*Human Machine Interface*

SINGKATAN	ARTI
DO	*Dissolved Oxygen*
pH	*Power of Hydrogen*
SMS	*Short Message Service*
GSM	*Global System for Mobile communication*
LED	*Light Emitting Diode*
LCD	*Liquid Crystal Display*
RPM	*Remote Pond Monitoring*

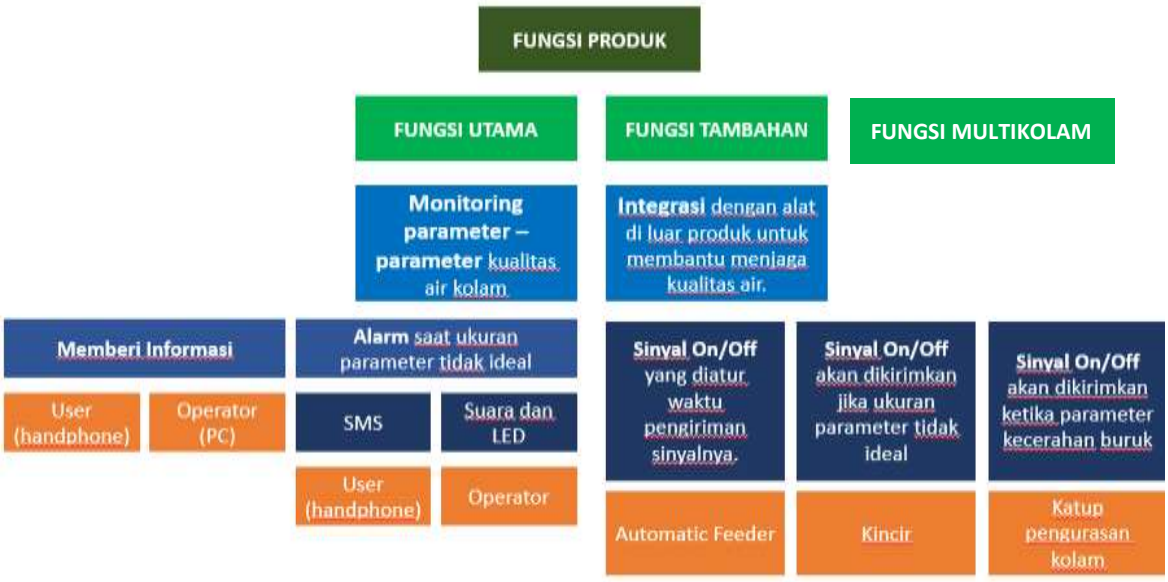
2 PERANCANGAN UMUM

2.1 Definisi, Fungsi dan Spesifikasi dari Solusi

2.1.1 Definisi, Fungsi dan Spesifikasi

e-Shrimp adalah alat sistem monitoring cerdas untuk tambak udang vannamei dengan multi-sensor. Alat ini memiliki kemampuan untuk melakukan monitoring terhadap parameter kualitas air kolam, mengolah data untuk memperingati penjaga kolam jika ada parameter kualitas air kolam yang berada di luar batas normal dan juga mampu untuk mengaktifkan dan mengnon-aktifkan kincir pada kolam tergantung kondisi parameter kualitas air kolam.

e-Shrimp memiliki beberapa fungsi yang dapat dilihat pada bagan dibawah ini:



Gambar 2. 1 Fungsi pada e-Shrimp

Keterangan:

User : *Stakeholder* tambak udang yang perlu untuk memantau kondisi kolamnya tanpa harus berada di lokasi.

Operator: Orang yang bertugas untuk mengurus tambak udang.

Fungsi dari produk ini dibagi menjadi fungsi utama dan tambahan. Fungsi utama yaitu untuk melakukan monitoring terhadap parameter – parameter yang menentukan kualitas air kolam udang. Hasil pengukuran parameter – parameter kualitas air tersebut akan dikirimkan ke user (handphone) dan operator yang akan melihat hasil pengukuran lewat PC. Hal ini bertujuan agar operator dan user dapat melakukan pemantauan terhadap kolam udangnya.

Kemudian hasil dari pengukuran parameter – parameter kualitas air ini juga akan memicu alarm ketika nilai pengukurannya tidak ideal. Alarm pada produk ini berupa sms yang akan dikirimkan ke user (handphone) dan dalam bentuk suara dan lampu LED untuk memberi peringatan bagi operator. Tujuannya agar user mengetahui kondisi kolamnya dan agar operator dapat langsung melakukan tindakan yang dapat mengubah parameter – parameter kualitas air menjadi ideal kembali.

Fungsi tambahan produk ini yaitu berupa integrasi dengan alat – alat lain yang akan membantu untuk menjaga kualitas air kolam udang yaitu *automatic feeder*, kincir dan katup

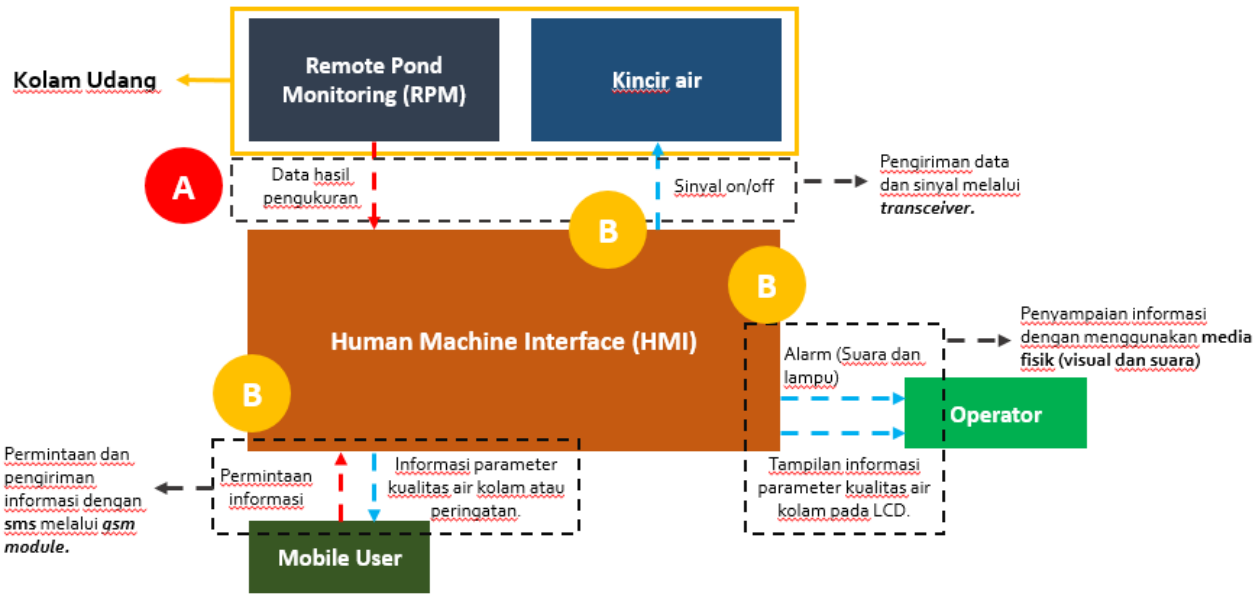
pengurasan kolam. Untuk *automatic feeder* akan dikirimkan sinyal on/off berdasarkan waktu, waktu pengiriman ini akan diatur berdasarkan jam makan udang. Untuk katup pengurasan kolam akan dikirimkan sinyal on/off ketika kecerahan air pada kolam udang sudah memburuk. Untuk kincir, sinyal on/off akan dikirimkan saat parameter – parameter kualitas air dalam keadaan yang tidak ideal. Hal ini dilakukan agar kincir bekerja dan memperbaiki kualitas air pada kolam udang.

Perlu diketahui bahwa fungsi tambahan yang akan diimplementasikan pada *e-Shrimp* saat ini baru satu bagian yaitu bagian otomasi untuk menyalakan kincir. Ada beberapa alasan mengapa hanya satu bagian yang diimplementasikan:

3. Untuk *Automatic Feeder*, pertimbangan harga menjadi salah satu kendala untuk mengimplementasikannya karena biaya sudah terkuras untuk pembelian sensor.
4. Untuk Katup Penguras, mempertimbangkan frekuensi penggunaannya yang hanya sekali dalam periode 1 – 2 minggu sehingga dinilai tidak akan signifikan manfaatnya untuk mengaplikasikan sistem otomasi pada bagian ini.
5. Secara umum, konsep dari pengaplikasian *automatic feeder* dan katup penguras sama dengan pengaplikasian otomasi kincir kolam. Dalam aplikasinya digunakan relay untuk mengontrol aktif dan non-aktifnya kincir kolam tersebut kemudian digunakan juga *transceiver* untuk menerima sinyal yang akan mengontrol aktif dan non-aktifnya relay.

Berdasarkan 3 alasan diatas, fungsi tambahan yang akan diaplikasikan pada *e-Shrimp* versi 2017 hanya pada bagian otomasi untuk kincir kolam.

Untuk memahami lebih jelas bagaimana sistem *e-Shrimp* bekerja, dapat dilihat terlebih dahulu ilustrasi dibawah ini:

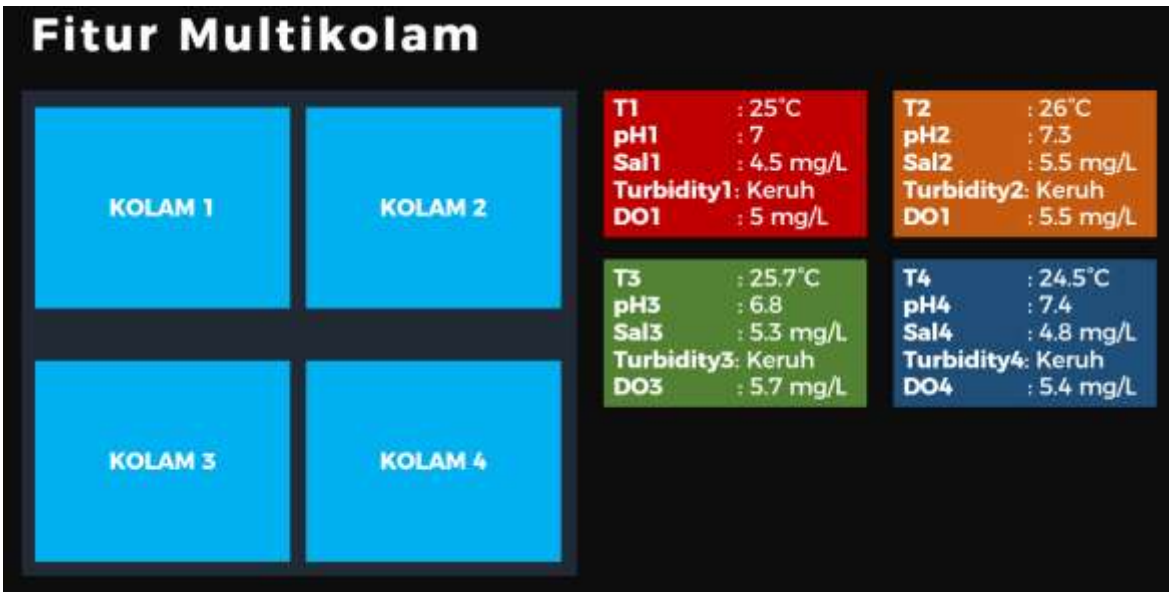


Gambar 2. 2 Sistem Kerja e-Shrimp

Secara umum ada 2 proses pada keseluruhan sistem *e-shrimp*, pertama adalah bagian pengambilan data (A) dan yang kedua adalah bagian pengolahan dan tindakan dari hasil pengolahan tersebut (B). Dapat dilihat dari ilustrasi diatas bahwa pada proses A, *e-Shrimp* akan mengambil data dari kelima sensor yang ada kemudian meng-*compile* data tersebut ke dalam paket *array* untuk kemudian dikirim ke modul HMI menggunakan *transceiver*.

Pada bagian B, seperti yang juga bisa dilihat pada ilustrasi diatas bahwa bagian B terdiri dari pengolahan data yang diterima dari modul RPM berserta tindakan berdasarkan data yang sudah diolah tersebut. Ada 3 tindakan yang dilakukan pada bagian B, pertama pengiriman informasi status parameter kualitas air kolam ke *mobile user*, sebaliknya *mobile user* juga bisa meminta informasi yang lebih detail mengenai nilai parameter kualitas air kolam menggunakan sms ke modul HMI. Kedua yaitu peringatan untuk operator kolam berupa alarm dan lampu yang akan menyala ketika ada parameter kolam yang nilainya tidak ideal. Tindakan atau *action* ketiga yang akan dilakukan adalah pengiriman sinyal untuk mengatur aktif dan non-aktifnya relay kincir. Relay kincir ini akan meneruskan arus ke kincir kolam ketika salah satu dari parameter temperature atau oksigen terlarut berada dalam kondisi tidak ideal.

Pada e-Shrimp juga terdapat fungsi multikolam dimana, e-Shrimp dapat menampilkan data dari maksimal 4 kolam udang. Berikut adalah ilustrasinya:



Gambar 2. 3 Fitur Multikolam e-Shrimp

5.1.1 Penjelasan Fitur

Fitur untuk Pengguna Sistem

Terdapat dua jenis pemakai dalam sistem ini, yaitu:

1. Operator, yaitu orang yang bertanggung jawab mengawasi kualitas air kolam di lokasi serta mengoperasikan sistem. Interaksi yang melibatkan operator adalah :
    - Melihat hasil *monitoring* pada display.
    - Melakukan aksi pengaturan kualitas air yang tidak dilakukan oleh sistem
2. *Mobile user*, yaitu orang yang dapat mengakses data yang telah disimpan dan diolah di HMI. Interaksi yang melibatkan *mobile user* adalah:
    - *Mobile user* menerima data dari sistem dengan dua pilihan, yaitu:

a. Menerima pada waktu tertentu, ketika *user* mengirimkan instruksi via SMS

b. Menerima pesan peringatan melalui SMS ketika ada parameter yang melewati batas normal.

Informasi yang didapat oleh pemakai yaitu beberapa parameter kualitas air yaitu:

a. Suhu
b. Perubahan pH
c. DO
d. Salinitas
e. Kecerahan air

Peringatan yang didapat oleh pemakai adalah:

1. Untuk operator di lapangan, akan mendapatkan peringatan berupa bunyi alarm dan LED yang menunjukkan adanya parameter yang melewati batas normal.
2. Pada *mobile user* (tidak berada di lapangan), akan mendapatkan peringatan berupa SMS yang berisi informasi mengenai data *monitoring.*

*SMS dikirim melalui GSM *module.*

Penjelasan Fungsi Perangkat Utama

Terdapat dua perangkat utama yang digunakan produk ini, yaitu:

1. RPM

   RPM (Remote Pond Monitoring) merupakan modul yang mengumpulkan seluruh data yang didapat dari sensor dan membawa seluruh data ke *transmitter* setelah dilakukan proses pengolahan data untuk ditransfer ke modul HMI

2. HMI

   HMI merupakan modul yang terdiri dari kumpulan data yang didapat dari modul RMP dan mengeluarkan keputusan berdasarkan kebutuhan. Keputusan-keputusan yang diberikan berupa
   - Memberi sinyal on/off kepada kincir air dan katup (fitur tambahan)
   - Mengirimkan paket data ke *automatic feeder* (fitur tambahan)
   - Mengirimkan informasi via SMS
   - Menampilkan tampilan data *monitoring.*

Penjelasan Verifikasi Sistem (Pengujian Sistem)

Pengujian produk ini dilakukan 2 kali. Pertama, dengan menggunakan model tambak udang yang dibuat dengan menggunakan wadah besar yang diisi air yang telah dicampur dengan garam untuk mensimulasikan keadaan pada tambak udang air asin. Alat *monitoring* akan diletakan pada permukaan air menggunakan pelampung, sehingga kontak yang terjadi antara air dan alat hanya terdapat pada sensor. Pengujian kedua dilakukan langsung di tambak udang vaname, untuk melakukan pengujian yang tidak bisa dilakukan pada model.

Pengujian pertama dilakukan pada modul RMP. Prosedur pengujian yang dilakukan adalah:

- Menempatkan modul RMP pada model tambak.
- Pengukuran kualitas air dengan menggunakan cara manual yaitu dengan menggunakan alat ukur masing-masing parameter. Pengukuran suhu menggunakan thermometer, pengukuran pH menggunakan pH meter, pengukuran DO menggunakan DO meter, dan pengukuran salinitas menggunakn salinometer.
- Melakukan pengukuran sebanyak 5 kali sehingga didapatkan data yang akurat.
- Mengukur kualitas air dengan menggunakan sensor-sensor yang terdapat pada modul RMP. Dan menampilkan hasil pengukuran pada PC.
- Membandingkan hasil pengukuruan secara manual dengan pengujian menggunakan sensor.

Pengujian berikutnya dilakukan pada modul HMI. Prosedur pengujian yang dilakukan adalah:

- Mengirimkan hasil pengukuran pada RMP melalui *transmitter* ke *receiver* pada modul HMI. Kemudian data diolah MCU untuk ditampilkan pada *display*. Kemudian membandingkan hasil pada *display* dengan nilai pengujian RMP dan pengukuran manual.
- Menguji proses pengiriman data dari lebih dari satu modul RMP ke satu *receiver*. Kemudian data diolah MCU dan ditampilkan pada *display*. Lalu membandingkan hasil *display* dengan hasil pengujian sebelumnya (menguji satu persatu RMP).
- Menghubungkan HMI dengan PC. Kemudian memasukkan nilai yang melebihi atau dibawah batas normal kualitas air, sehingga HMI akan mengaktifkan alarm dan LED.
- Menguji modul GSM dengan mengirimkan pesan secara langsung tanpa menunggu instruksi dari *mobile user*. Kemudian menguji modul GSM dengan memberikan instruksi dari *mobile user* melalui SMS, untuk meminta data *monitoring.*

Prosesur pengujian diatas dapat dilakukan dengan menggunakan model tambak udang. Sedangkan pengujian yang dilakukan di tambak udang vaname adalah daya tahan sensor, daya tahan baterai RMP, komunikasi jarak jauh antar *transmitter* RMP dan *receiver* HMI, komunikasi antar kolam, pengukuran kecerahan air serta ketinggian air serta integrasi dengan alat-alat lainnya seperti kincir air dan katup.

5.1.2 Spesifikasi Tugas Akhir

Untuk memaksimalkan produktivitas dari tambak udang, maka diperlukan monitoring terhadap parameter – parameter kualitas air. Pada tabel berikut, dapat dilihat 5 parameter beserta nilai idealnya yang akan diukur oleh sistem monitoring cerdas kolam tambak udang vannamei:

Parameter	**Nilai Ideal**

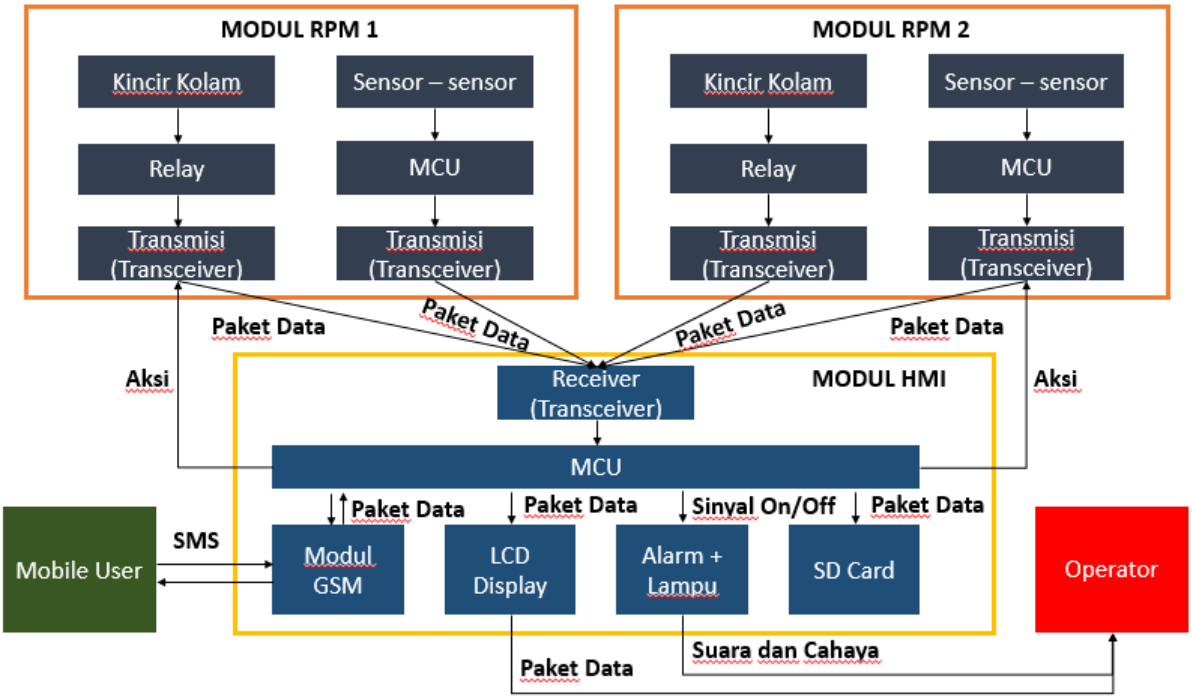
Salinitas	10 - 35 mg/L
Temperatur	28 – 31.5°C
pH	7.5 – 8.5
Oksigen Terlarut	3.5 mg/L (minimal)
Kekeruhan air	30 – 45 cm

Tabel 2.1 Parameter dan nilai idealnya sesuai dengan SNI 7772:2013

2.2 Perancangan (Design)

2.2.1 Spesifikasi dan Daya Kerja (Performance) Fungsi

Berikut merupakan diagram blok sistem keseluruhan yang memaparkan modul-modul beserta hubungan dari setiap modul yang ada pada e-Shrimp :

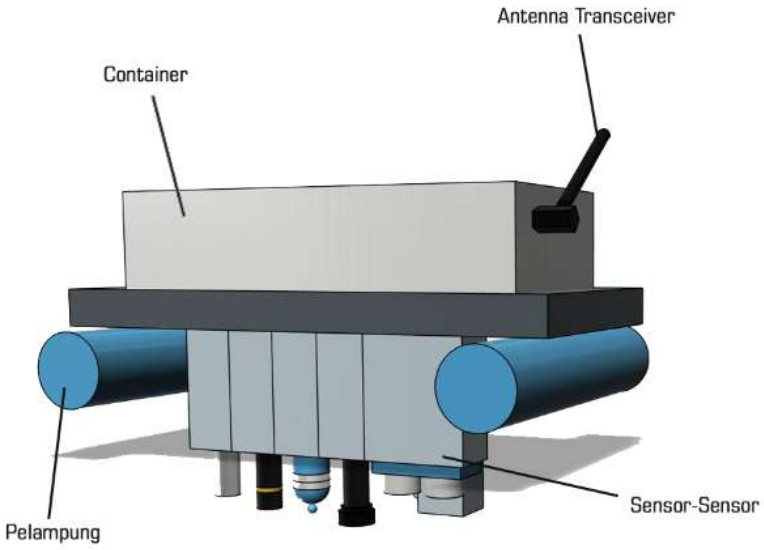


Gambar 2. 4 Desain Produk Versi 2

2.2.2 Desain Fisik Perangkat Keras

Berikut merupakan desain untuk tampilan fisik modul-modul perangkat keras pada **e-Shrimp** : Sistem Kontrol Pintar dengan Multisensor untuk Tambak Udang Vannamei, sebagai spesifikasi dan gambaran fisik untuk alat yang dirancang pada riset ini :

Modul RPM :



Gambar 2. 4 Desain Tampilan Fisik Modul RPM (*Remote Pond Monitoring*)

Container pada modul RPM berisi beberapa perangkat keras yang tidak tahan air seperti board mikroprosessor, receiver, LCD, board-board probe dan lain-lain. Untuk bahan RPM, terutama untuk container digunakan bahan dengan standar *International Protection Marking* atau *Ingress Protection Marking* yang setara dengan barang dengan sertifikasi IP64 (Tidak ada jalan masuk untuk debu dan tahan terhadap cipratan air).

RPM juga memiliki pelampung yang berfungsi untuk membuat RPM dapat mengambang pada tambak udang ketika beroperasi, hal ini juga diperlukan untuk mencegah terendamnya perangkat-perangkat keras yang tidak tahan air pada tambak udang.

Sensor-sensor yang digunakan pada RPM antara lain :

- Sensor temperatur
- Sensor salinitas
- Sensor pH
- Sensor DO (*Dissolved Oxygen*)
- Sensor Kecerahan Air

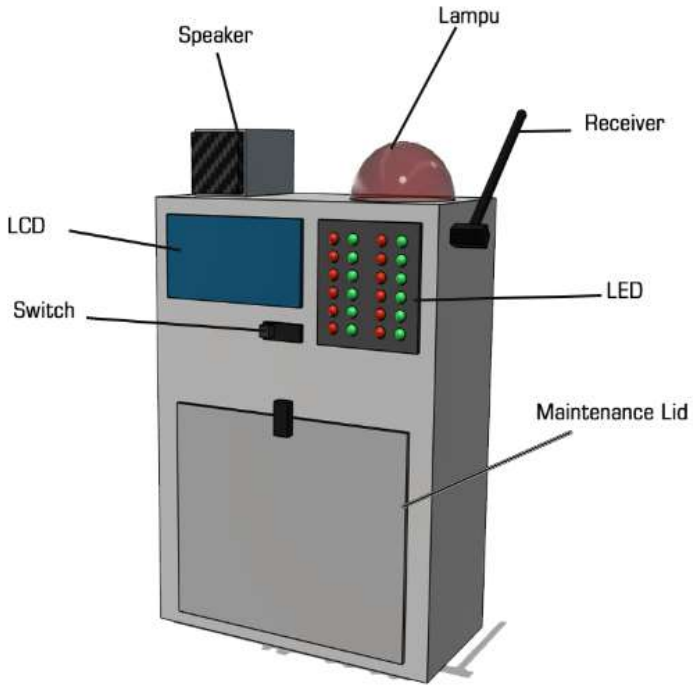
Mayoritas sensor yang diigunakan memiliki bentuk fisik probe yang waterproof sehingga, diletakan pada bagian bawah RPM yang akan terendam oleh air agar setiap sensor dapat melakukan pengukuran parameter kualitas air tambak secara langsung. Untuk sensor kecerahan air, karena menggunakan LED dan photodiode yang tidak tahan air, maka akan dibuat sebuah lapisan kedap air transparan pada pipa berisi LED dan photodiode untuk mencegah masuknya air.

Antenna transceiver berfungsi untuk melakukan penerimaan dan pengiriman data antara modul RPM dengan modul HMI. Diletakan bersebelahan dengan container untuk mengurangi jarak antara board transceiver dengan mikroprosessor sehingga bisa menghemat space yang ada

Dimensi : 50cm x 30cm x 50cm (Panjang x Lebar x Tinggi)

Massa : 2,5 kg

Modul HMI :



Gambar 2. 5 Desain Tampilan Fisik Modul HMI (*Human Machine Interface*)

Sama seperti RPM, bahan untuk HMI menggunakan casing dengan bahan yang setara dengan sertifikasi IP64.

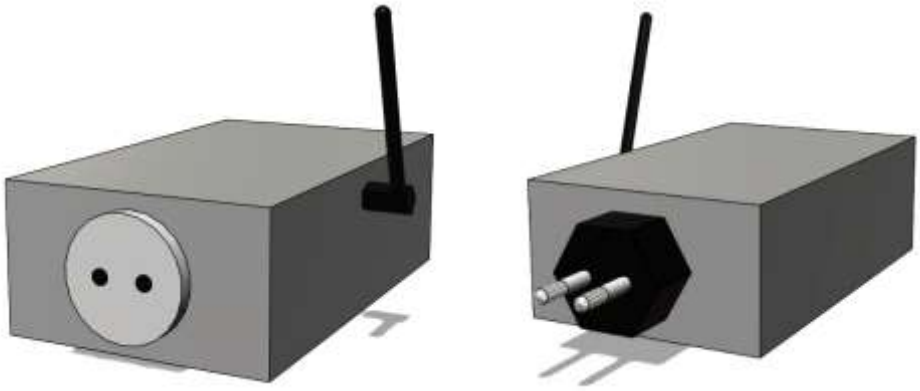
Ada beberapa komponen yang diletakan diluar *casing* atau tembus pandang dari luar antara lain :

- LCD untuk menampilkan nilai-nilai parameter kualitas air tambak yang sedang dideteksi oleh RPM
- LED sebagai indikator apakah parameter kualitas air pada suatu tambak berada dinilai yang normal atau tidak (hijau untuk nilai normal dan merah untuk nilai yang tidak normal)
- Speaker untuk memberitahukan operator dijarak yang cukup jauh ketika ada parameter kualitas air tambak dalam range yang tidak normal
- Lampu untuk memberitahukan operator lokasi HMI yang RPMnya mendeteksi parameter kualitas air tambak dalam range yang tidak norma;
- Switch : Untuk menyalakan dan mematikan HMI
- Receiver : untuk melakukan komunikasi secara wireless antara perngkat HMI dengan perangkat-perangkat RPM lainnya

Dimensi : 30cm x 15cm x 55cm (Panjang x Lebar x Tinggi)

Massa : 2 Kg

Modul Kincir Air :



Gambar 2. 6 Desain Tampilan Fisik Modul Kincir Air

Modul ini berfungsi sebagai penghubung antara sumber listrik dengan kincir. Modul ini berisi sebuah rangkaian *switching* yang berfungsi sebagai sakelar yang dapat menyalakan dan mematikan kincir secara otomatis. Konsep *switching* ini diperlukan ketika terdapat parameter air kolam yang berada diluar kondisi normal yang dapat dikoreksi dengan menggunakan kincir.

Modul ini juga memiliki *receiver* yang berfungsi untuk menerima sebuah sinyal dari HMI ketika ada parameter kualitas air yang bisa dikoreksi oleh kincir, sinyal ini akan membuat *switch* menjadi terhubung dan membuat kincir berjalan.

Dimensi : 25cm x 10cm x 10cm (Panjang x Lebar x Tinggi)

Massa : 0,25 Kg

3 IMPLEMENTASI

3.1 Implementasi Modul RPM

3.1.1 Pengantar

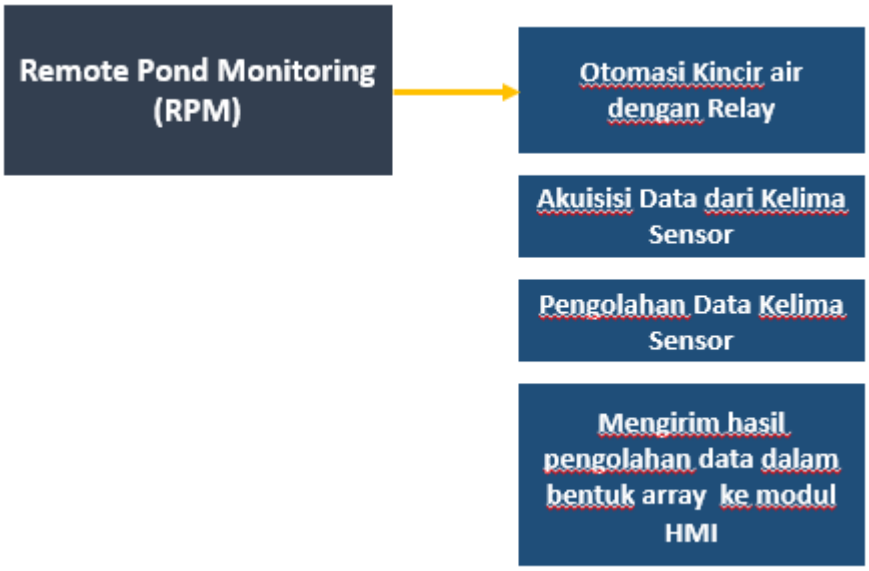
Remote Pond Monitoring (RPM) adalah salah salah satu modul dari sistem monitoring cerdas kolam tambak udang vannamei. Secara umum pada modul RPM terdapat 5 sensor untuk mengukur 5 parameter kualitas air kolam. Kelima parameter tersebut diantaranya adalah:

1. Sensor Temperatur
2. Sensor DO (*Dissolved Oxygen*)
3. Sensor pH
4. Sensor Salinitas (Konduktivitas)
5. Sensor Kekeruhan Air

Selain terdapat sensor, pada modul RPM juga terdapat relay yang akan digunakan untuk otomasi operasional kincir. Kincir disini akan berfungsi sebagai respons pertama ketika parameter temperature dan DO dalam kondisi tidak normal. (diluar batas yang sudah ditentukan)

3.1.1.1 Tinjauan Spesifikasi dan Desain

Modul RPM memiliki beberapa bagian diantaranya:



Gambar 3. 1 Fungsi Modul RPM

Terdapat 4 fungsi utama pada modul RPM seperti yang sudah perlihatkan oleh gambar diatas.

Spesifikasi modul RPM adalah sebagai berikut:

1. Sensor

   Sensor pada modul RPM dapat melakukan pengukuran parameter kualitas air kolam dengan range pengukuran sebagai berikut:

Parameter	Nilai Ideal
Salinitas	1 – 35 mg/L
Temperatur	28 – 31.5°C
pH	7.5 – 8.5
Oksigen Terlarut	3.5 mg/L (minimal)
Kecerahan air	30 – 45 cm

Table 3.1 Spesifikasi Sensor Modul RPM

2. Akuisisi Data

   Dalam akuisisi data ada beberapa hal yang harus diperhatikan diantara lain adalah batasan – batasan yang dimiliki oleh sensor, spesifikasi yang dibutuhkan dan kondisi lapangan yang dihadapi. Dalam proses akuisisi data, kita harus menentukan frekuensi pengambilan data. Pada modul RPM ini ditentukan bahwa frekuensi pengambilan data yaitu setiap 30 menit sekali. Angka 30 menit ini diambil dari pertimbangan:

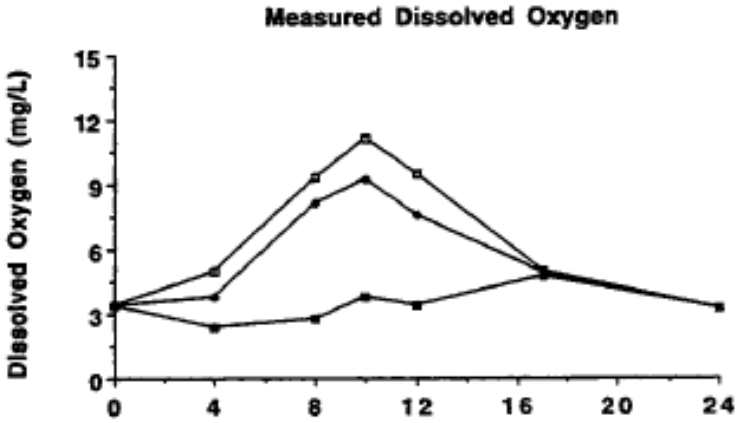
   a. Batasan Sensor

      Sensor harus menyala bergantian agar tidak saling mengganggu. Jika sensor dinyalakan bersama maka kebocoran arus listrik dengan *magnitude* sangat kecil dapat mengganggu pembacaan sensor lainnya. Kemudian harus ada interval atau jeda dari pengaktifan 1 sensor ke sensor lainnya. Dalam percobaan yang telah dilakukan, rata – rata interval waktu ideal adalah 6 menit.

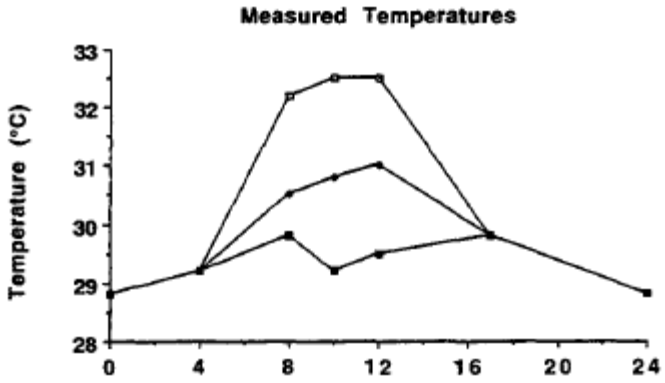
   b. Spesifikasi

      Jika dilihat dari karakteristik kelima parameter kualitas air kolam, tidak ada satupun parameter yang dapat berubah nilainya secara signifikan dalam waktu kurang dari 30 menit. Maka dari itu angka 30 menit diasumsikan aman.

   c. Kondisi



Gambar 3. 2 Hasil Pengukuran DO selama 24 jam



Gambar 3. 3 Hasil Pengukuran Temperatur selama 24 jam

Berdasarkan *paper* karya Steven D. Culberson dan Raul H. Piedrahita, ada 2 parameter air kolam yang mengalami perubahan sangat drastis pada periode 24 jam yaitu temperature dan DO (*Dissolved Oxygen*) atau kelarutan oksigen dalam air. Namun jika kita lihat pada gambar 2, perubahan maksimum terdapat pada titik 3 mg/L – 11 mg/L dalam waktu 10 jam. Jika kenaikan tersebut kita asumsikan linear seperti pada gambar maka rata – rata perubahannya hanya +0.8 mg/L per jam. Perubahan ini tentu saja masih bisa dideteksi meskipun frekuensi pengukuran kita atur setiap 30 menit sekali.

Begitu juga dengan temperature, dimana perubahan maksimum terjadi pada titik 29◦C ke titik 32.5◦C dalam waktu 6 jam (dari jam 4 ke 10). Jika diasumsikan perubahan tersebut terjadi secara linear seperti pada gambar, maka rata – rata perubahannya adalah +0.583◦C per jam. Perubahan ini tentu saja sangat kecil dan dengan frekuensi pengambilan data yang 30 menit sekali masih dapat terdeteksi jika terjadi perubahan.

Maksut dari tinjauan ini adalah untuk menjawab pertanyaan frekuensi ‘aman’ untuk pengukuran parameter kualitas air kolam. ‘Aman’ disini berarti jika ada perubahan nilai parameter air kolam maka masih dapat terdeteksi dan dapat segera ditanggulangi (tidak ada

keterlambatan akibat frekuensi pengambilan data yang lama sehingga parameter yang sudah dalam kondisi bahaya tidak terdeteksi). Dari tinjauan pada dua parameter yang perubahannya paling drastis ini dapat diyakini bahwa frekuensi pengambilan data 30 menit sekali dapat mencukupi spesifikasi.

Sumber:
Aquaculture pond ecosystem model: temperature and dissolved oxygen prediction - mechanism and application
Steven D. Culberson, Raul H. Piedrahita *
Department of Biological and Agricultural Engineering, University of California, Davis, CA 95616, USA
Received 15 August 1994; accepted 26 April 1995

3. Pengolahan Data

   Pada modul RPM juga akan dilakukan pengolahan data yang telah diakuisisi oleh sensor. Pengolahan data ini untuk menentukan apakah nilai parameter air kolam yang terukur tersebut berada dalam kondisi normal atau tidak.

4. Relay untuk Kincir

   Pada modul RPM juga terdapat relay yang akan berfungsi untuk menghidupkan dan mematikan kincir.

Komponen yang digunakan pada modul RPM diantara lainnya adalah:

1. Arduino Mega 2560
2. Sensor Temperatur DS18S20
3. Sensor DO Atlas Scientific
4. Sensor Salinitas (Konduktivitas) Atlas Scientific
5. Sensor Kecerahan Air
6. Sensor DF Robot pH
7. Relay
8. Transceiver

3.1.1.2 Lingkungan Implementasi

Lingkungan implementasi dari modul RPM ini akan dibagi menjadi 3 bagian, yaitu:

1. Temperatur dan Kelembapan Lingkungan

   Modul RPM akan diletakkan mengapung pada permukaan air dengan temperature sekitar 20 – 35◦C. Namun *casing* modul RPM ini akan dibuat untuk dapat bertahan pada lingkungan dengan temperature 10 – 50◦C. Kemudian casing modul RPM ini juga akan dibuat untuk dapat bertahan pada lingkungan yang mempunyai kadar kelembapan sebesar 50 – 95%.

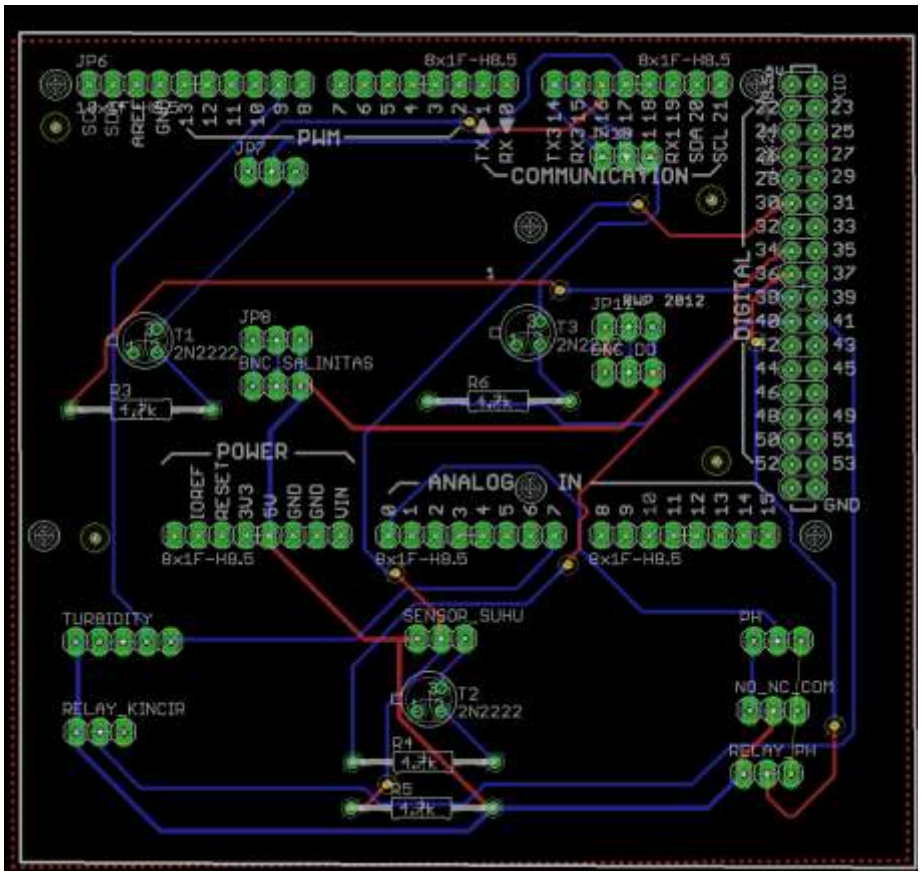
2. Ketahanan terhadap Getaran

   Modul RPM akan diletakkan mengapung pada permukaan air kolam tambak yang relative tenang. Namun pada beberapa kondisi, permukaan air dapat menjadi tidak tenang akibat faktor alam (cuaca) dan manusia (pengurasan, kincir, dll). Sehingga modul RPM harus mampu menahan getaran dan kejut sesuai standar IP66.

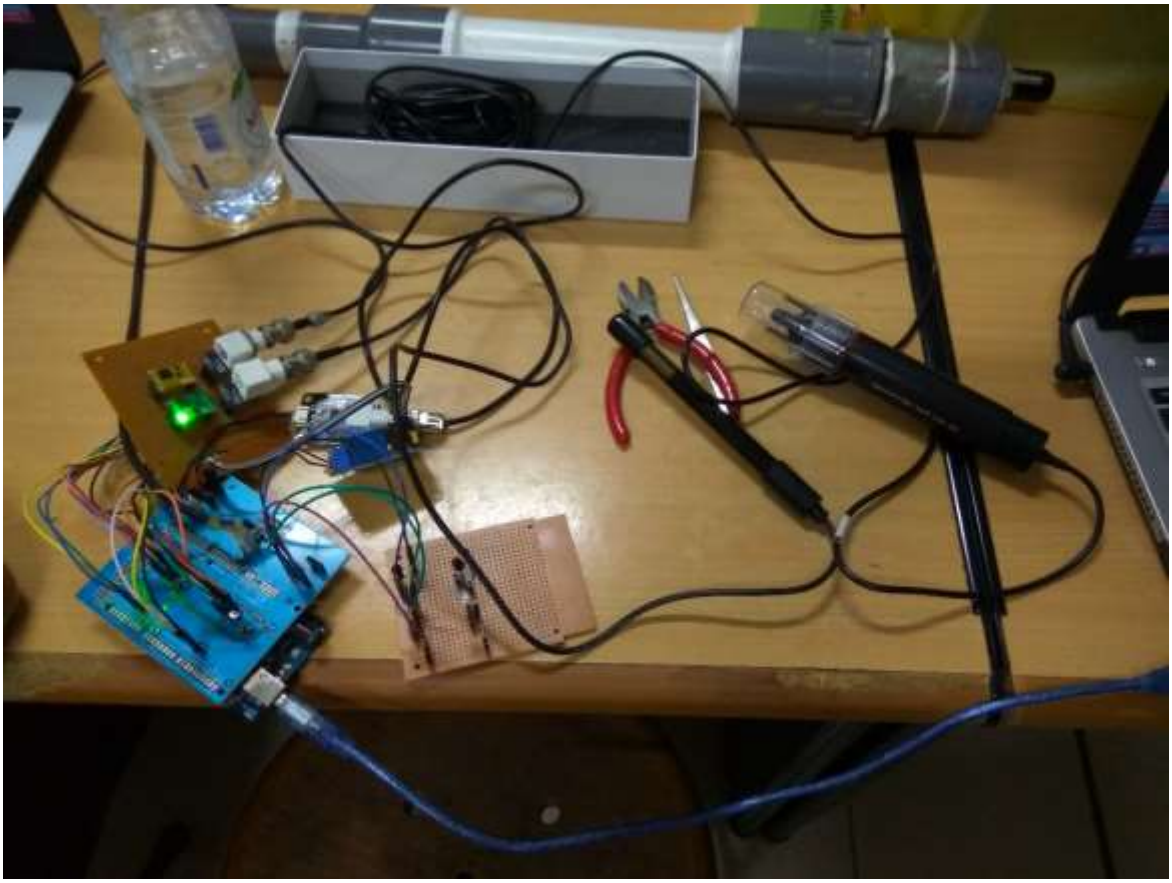
3. Ketahanan terhadap Debu, Abu dan Benda cair

   Untuk bertahan terhadap debu, abu dan benda cair, *casing* modul RPM akan dibuat tertutup rapat dan dilapisi oleh lapisan anti air pada bagian – bagian yang berpotensi dapat mengalami kebocoran. Sementara kelima sensor yang dipilih mempunyai spesifikasi untuk dapat bertahan di dalam air.

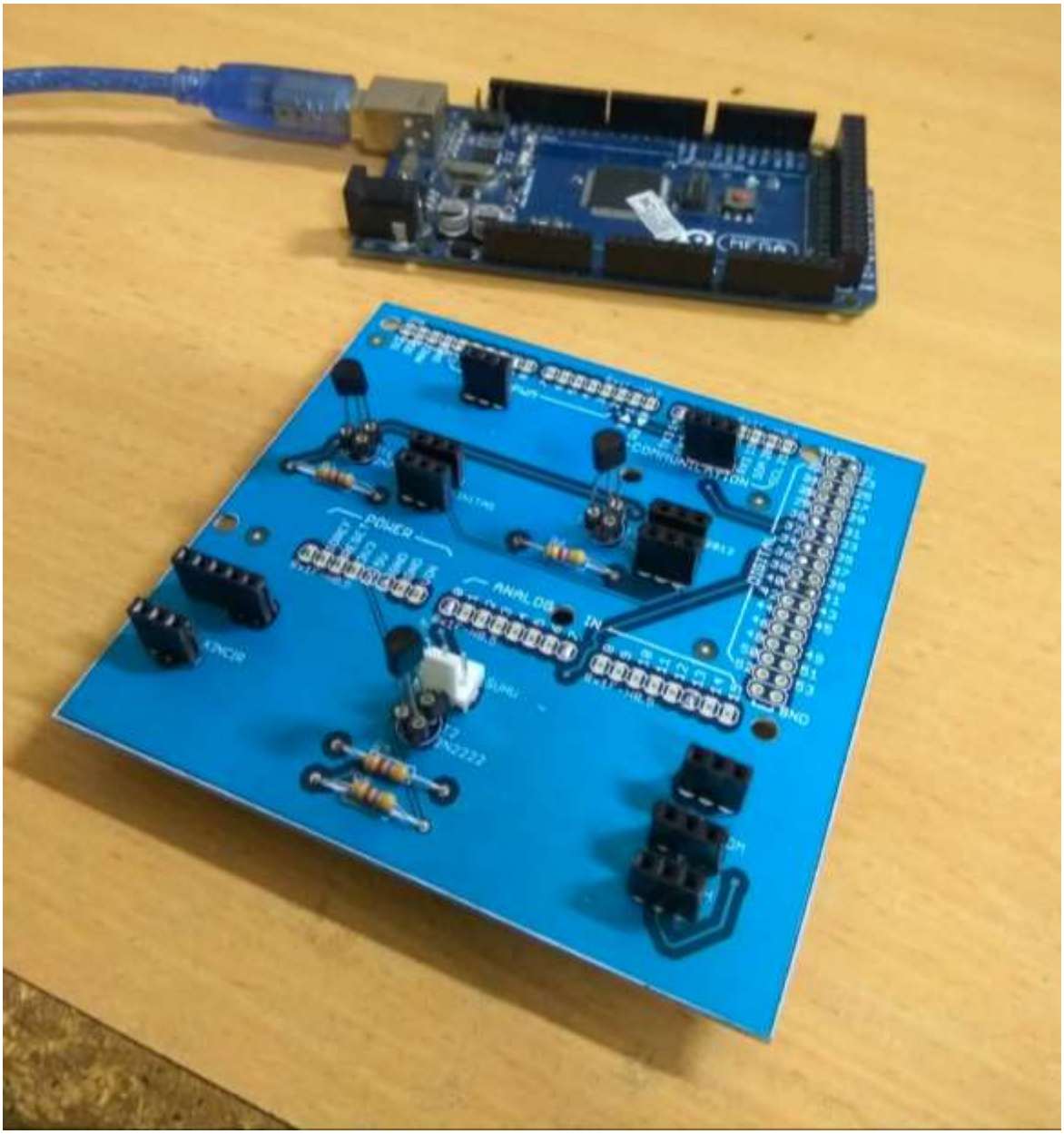
3.1.1.3 Hasil Implementasi *Hardware*



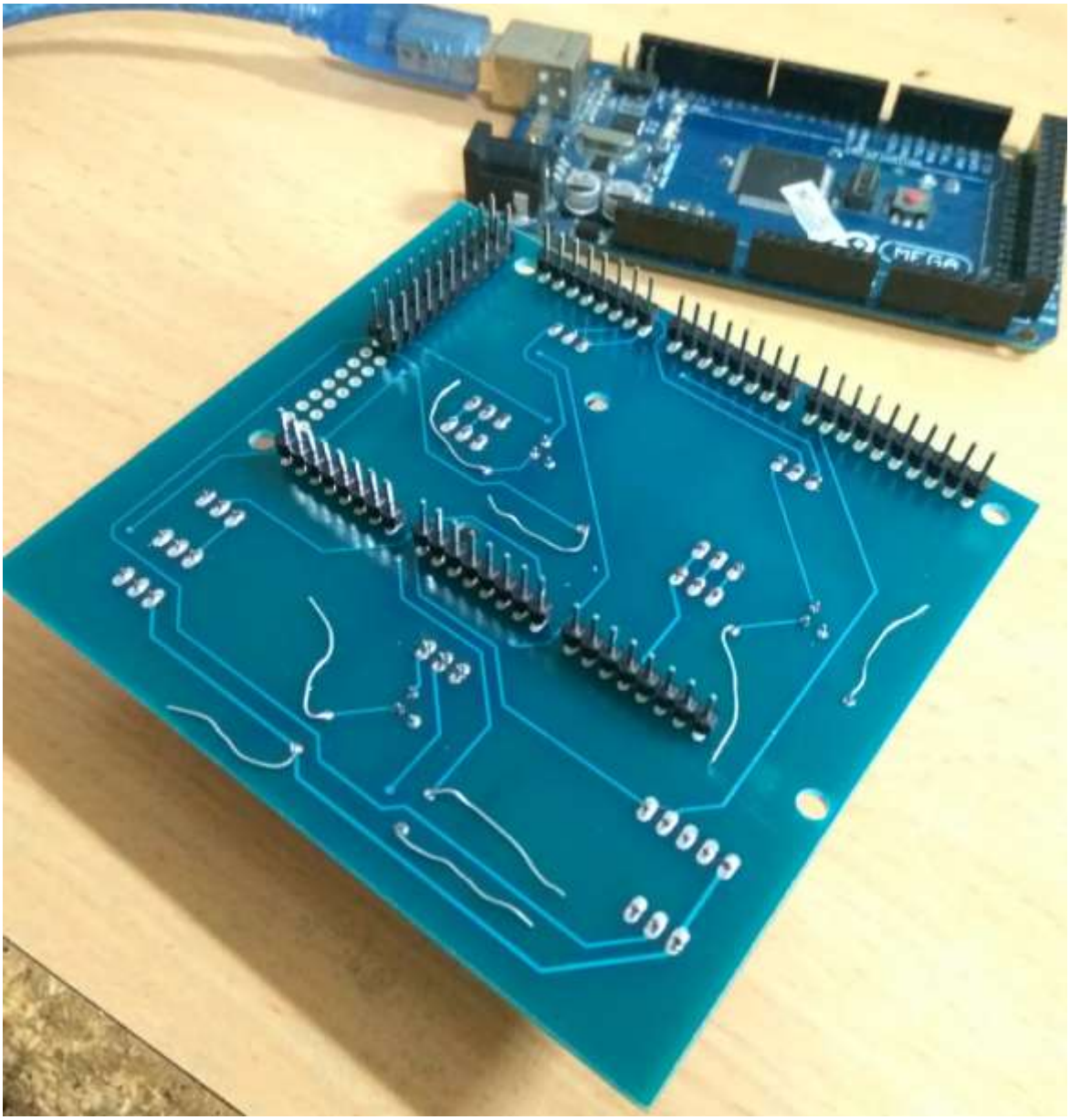
Gambar 3. 4 Desain Board Modul RPM



Gambar 3. 5 Hasil Implementasi Sensor dan Relay untuk Kincir pada RPM



Gambar 3. 6 PCB untuk modul RPM (Tampak atas)

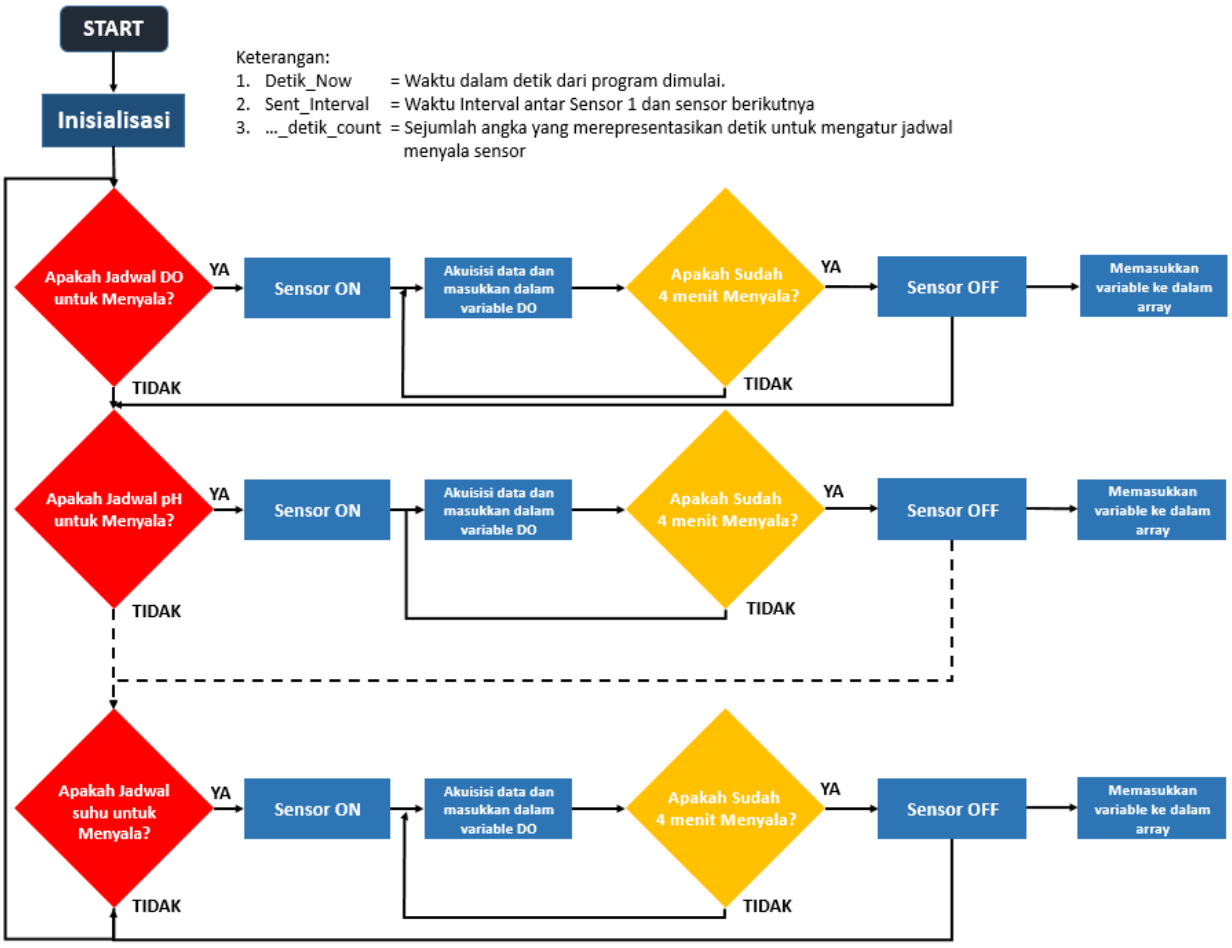


Gambar 3. 7 PCB untuk modul RPM (Tampak bawah)

3.1.1.4 Hasil Implementasi *Software*

Secara sederhana modul RPM akan mengatur kelima sensor yang ada untuk secara bergantian menyala dan dalam jangka waktu tertentu. Masing – masing sensor memiliki jadwal untuk menyala kemudian melakukan pengambilan data selama 4 menit dan kemudian mati selama 2 menit. Jadi setiap sensor memiliki interval selama 6 menit dari jadwal sensor tersebut menyala sampai jadwal sensor berikutnya untuk menyala. Waktu 2 menit ketika sensor dalam keadaan mati memiliki fungsi agar area sekitar *probe* sensor dapat memiliki waktu untuk menetralkan diri akibat adanya aliran listrik kecil ketika sensor dalam keadaan menyala. Jika aliran listrik tersebut belum dinetralkan tentu saja hal ini dapat berpotensi mengganggu bacaan sensor selanjutnya. Dibawah ini dapat dilihat *flowchart* untuk modul RPM, untuk menghemat tempat hanya ditunjukkan *flowchart* untuk 3 sensor. Selain menghemat tempat, juga terjadi pengulangan di dalam *flowchart* sehingga 3 sensor

sudah dianggap bisa merepresentasikan kelima sensor yang ada di modul RPM. Berikut adalah *flowchart*-nya:



Gambar 3. 8 *Flowchart* untuk Modul RPM

3.1.1.5 Permasalahan dan Solusi Implementasi Keseluruhan

Dari hasil implementasi modul RPM yang sudah dikerjakan terdapat beberapa permasalahan, diantaranya adalah:

1. Implementasi masih dilakukan menggunakan breadboard belum pada PCB. Hal ini menghambat proses desain *casing* karena masih menunggu ukuran PCB.
2. Implementasi perangkat lunak untuk sistem RPM secara keseluruhan masih belum diuji coba. Pengujian masih sebatas pada proses akuisisi dan pengolahan data, untuk sistem pengambilan data dengan frekuensi 30 menit, transmisi data ke HMI dan relay untuk kincir belum diuji coba.

Solusi untuk 2 permasalahan diatas adalah:

1. Pada minggu ke-9 dan 10 akan dilakukan desain, pencetakan dan implementasi rangkaian yang sudah dibuat pada PCB. Setelah ukuran PCB diketahui, maka akan di-desain juga casing untuk modul RPM.
2. Pada minggu ke-9 implementasi perangkat lunak untuk keseluruhan sistem RPM akan diselesaikan dan ditest dengan rangkaian menggunakan breadboard.

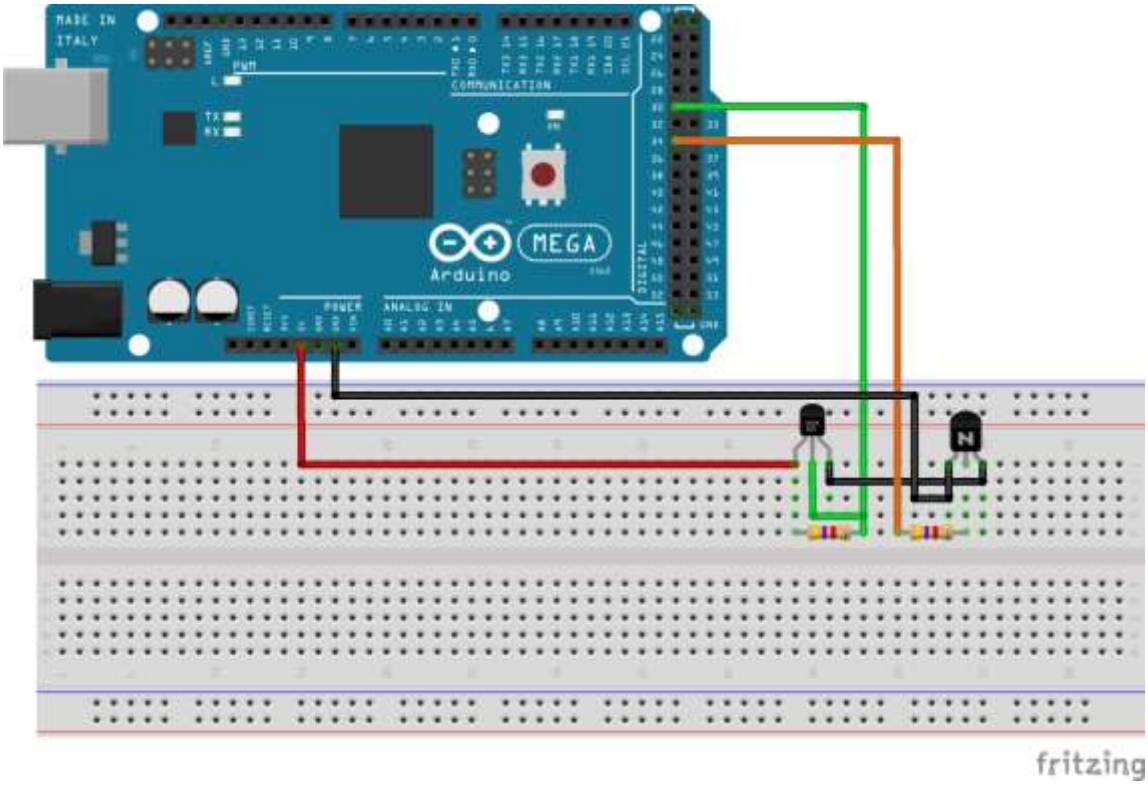
3.1.2 Implementasi Sensor Temperatur Modul RPM

3.1.2.1 Tinjauan Spesifikasi dan Desain

Pada ujicoba ini digunakan sensor DS18S20 keluaran MAXIM. Untuk uji coba sensor temperatur ini digunakan air keran gedung labtek VIII lantai 3 ITB yang ditempatkan pada sebuah ember.

Gambar 3. 9 Sensor Temperatur DS18S20

Sensor temperature DS18S20 mempunyai kemampuan untuk mengukur suhu kolam tambak udang yang mempunyai rentang ideal antara 28 – 31.5◦C. Sensor ini juga mempunyai resolusi yang sangat baik yaitu 0.1◦C dimana resolusi ini sangat mencukupi spesifikasi dari sistem monitoring cerdas tambak udang vannamei ini. Salah satu pertimbangan dalam memilih sensor ini adalah kemampuannya untuk bertahan di dalam air dalam periode waktu yang lama. Selain itu sensor temperature DS18S20 juga sudah *compatible* dengan arduino



Gambar 3. 10 Skematik Rangkaian Sensor Suhu DS18S20 pada Modul RPM

Salah satu fitur dari sensor temperature DS18S20 adalah kemampuannya untuk beroperasi tanpa memerlukan *external power supply*. Daya yang diperlukan oleh sensor di-*supply* melalui pull-up resistor 4.7kΩ yang ditempatkan pada VCC dan pin input pada sensor temperature DS18S20. Kemudian untuk mengontrol kondisi on-off sensor temperature DS18S20 digunakan transistor BJT yang akan berfungsi sebagai switch. Transistor BJT ditempatkan pada pin ground (GND) milik sensor temperature DS18S20. Resistor 4.7 kΩ dipasang pada base transistor npn tipe 2n2222 dengan asumsi arus pada *collector* 150 mA dan nilai β 150. (Nilai asumsi mengacu pada datasheet transistor 2n2222: http://www.farnell.com/datasheets/296640.pdf).

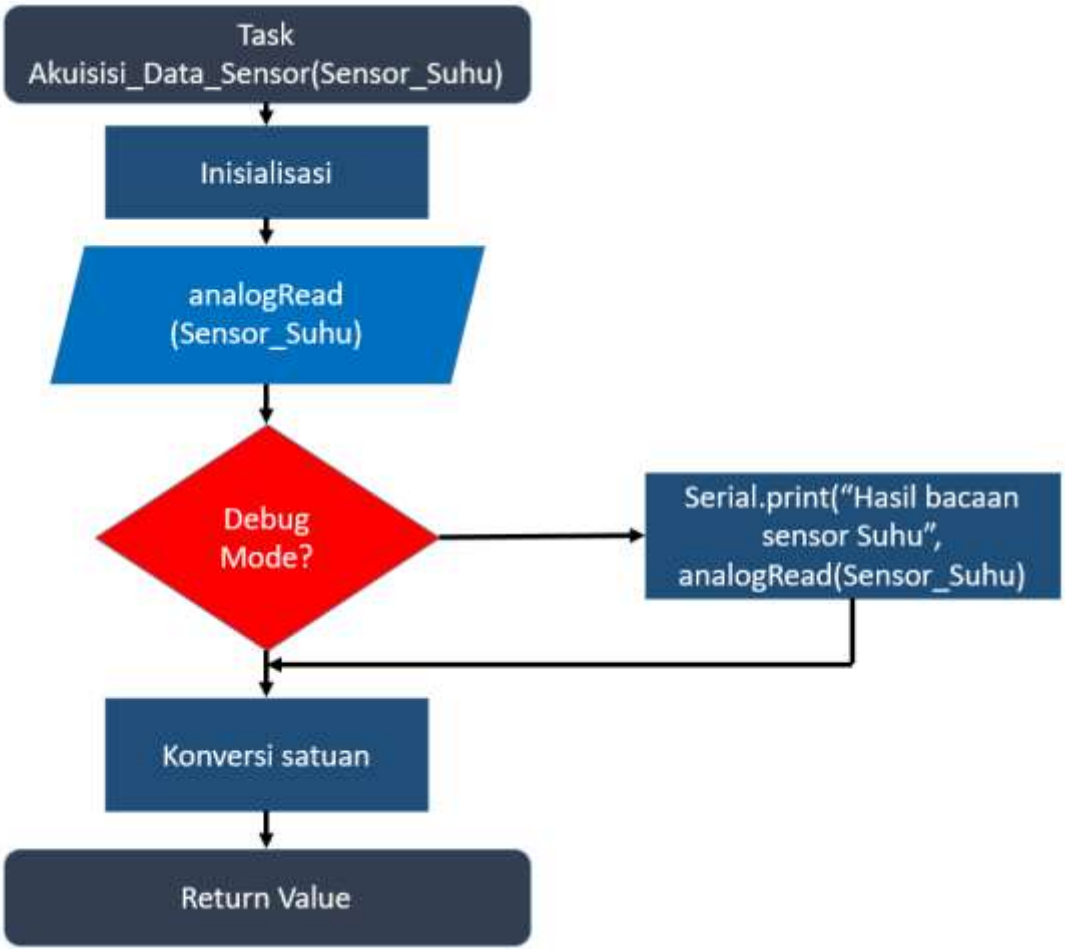
Fungsi on-off ini dibutuhkan agar sensor temperature pada modul RPM dapat beroperasi secara bergantian karena probe dari sensor temperature dapat mengeluarkan arus listrik sangat kecil yang dapat mengganggu pembacaan dari sensor lainnya. Maka dari itu diperlukan sistem on-off agar sensor temperature bisa dibuat dalam kondisi off ketika tidak diperlukan datanya. Pada tahap ini, implementasi rangkaian pada gambar diatas belum dilakukan.

3.1.2.2 Hasil Implementasi Perangkat Keras

Gambar 3. 11 Implementasi Skematik Rangkaian Uji Coba Sensor Temperatur

Pada gambar di atas dapat dilihat bahwa sensor suhu (dalam lingkaran merah) telah berhasil dihubungkan ke modul RPM (*board* berwarna biru).

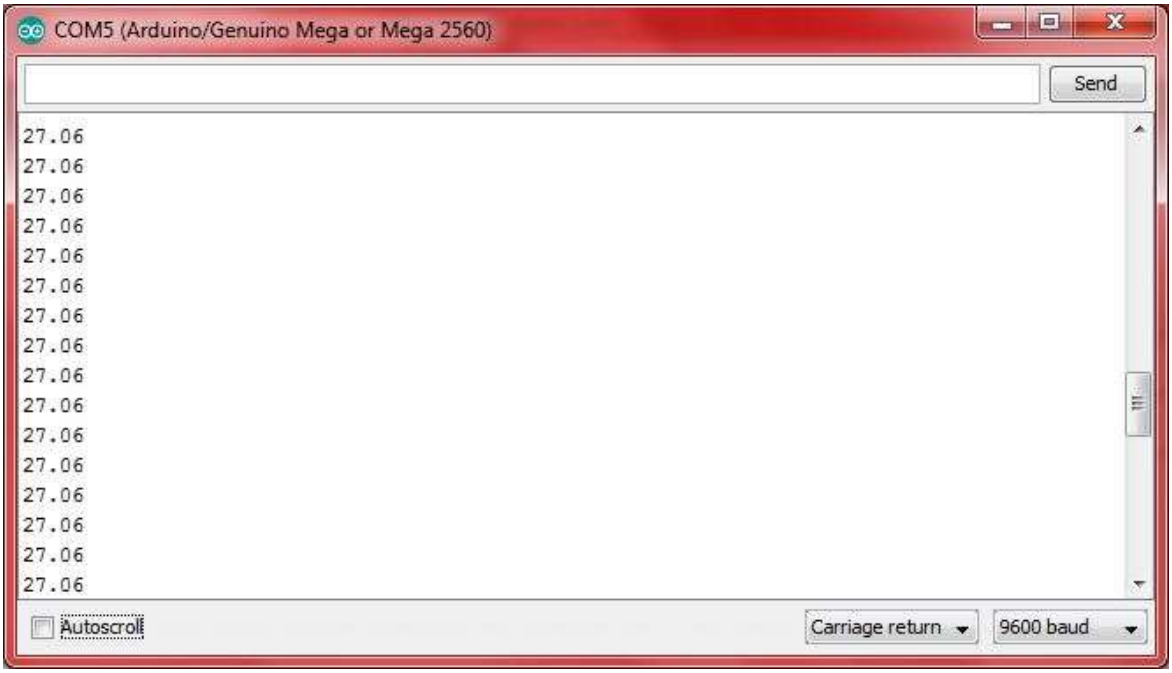
3.1.2.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 12 *Flowchart* untuk Akuisisi Data Sensor Suhu

Pada implementasi software dari *flowchart* diatas, software akan mempunyai fungsi getTemp() yang berfungsi untuk mengambil data temperature dalam bentuk bit dan mengkonversinya ke satuan derajat celcius. Fungsi ini akan mempunyai output bacaan data dari sensor temperature dalam satuan derajat celcius dengan variable Temperature_Sum.

Software diatas telah berhasil di-*compile* dan diimplementasikan pada PCB modul RPM yang sudah dibuat. Berikut menrupakan hasil *screenshoot* dari serial monitor yang menunjukkan bacaan sensor suhu:



Gambar 3. 13 Hasil Bacaan Sensor Suhu pada Serial Monitor

3.1.2.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi sensor Temperatur adalah sebagai berikut :

- Hasil pembacaan belum dapat dipastikan sesuai dengan suhu yang sebenarnya dari air / sampel. Masalah ini dapat diatasi dengan melakukan kalibrasi dan perbandingan hasil ukur dengan menggunakan sensor dan thermometer.
- Transistor BJT belum dapat berfungsi sebagai switch, untuk mengatasi masalah ini akan diteliti lagi 2 kemungkinan permasalahannya. Kedua kemungkinan tersebut adalah transistornya yang mengalami kerusakan atau arus yang dihasilkan tidak mencukupi sehingga nilai resistor di base transistor harus diubah.

Update kondisi kedua permasalahan tersebut:

- Untuk masalah akurasi sensor suhu akan dibahas pada dokumen pengujian B500.
- Transistor BJT sudah berhasil untuk berfungsi sebagai switch yang akan menyalakan dan mematikan sensor suhu setelah dilakukan *debugging* terhadap software yang digunakan.

3.1.3 Implementasi Sensor pH Modul RPM

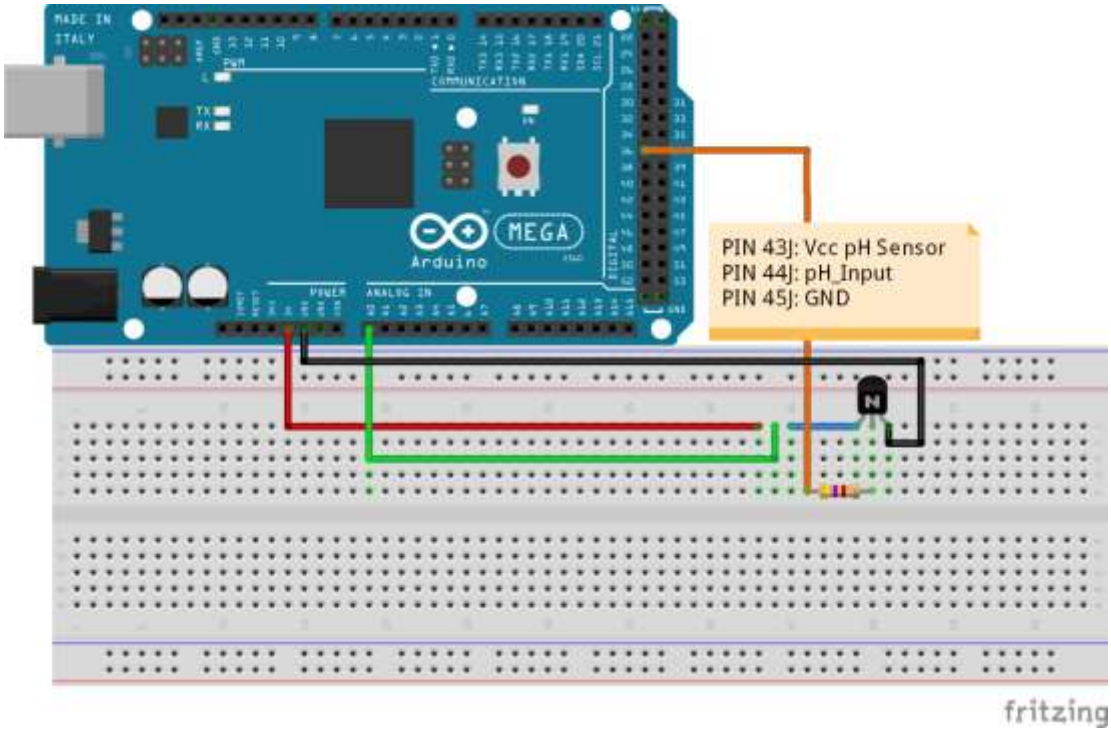
3.1.3.1 Tinjauan Spesifikasi dan Desain

Pada uji coba ini digunakan sensor *pH meter (sen:0169)* keluaran DF Robot. Untuk uji coba sensor pH ini digunakan air keran gedung labtek VIII lantai 3 ITB yang ditempatkan pada sebuah ember.

Gambar 3. 14 Sensor pH DF Robot (SEN: 0169)

Sensor DF Robot (sen:0169) digunakan karena mampu untuk mengukur pH pada rentang pH 7.5 – 8.5 yang merupakan kadar pH ideal untuk kolam tambak udang vannamei. Sensor ini juga memiliki resolusi yang ideal dan sesuai spesifikasi yang diinginkan yaitu 0.1. Kemudian salah satu pertimbangan yang dipikirkan juga mengenai kemampuan sensor untuk berada terus menerus di dalam air, sensor DF robot ini memiliki kemampuan tersebut. Terakhir beberapa faktor pertimbangan lainnya dalam memilih sensor ini adalah harga dan ketersediaan di pasaran Indonesia.

3.1.3.2 Hasil Implementasi Perangkat Keras



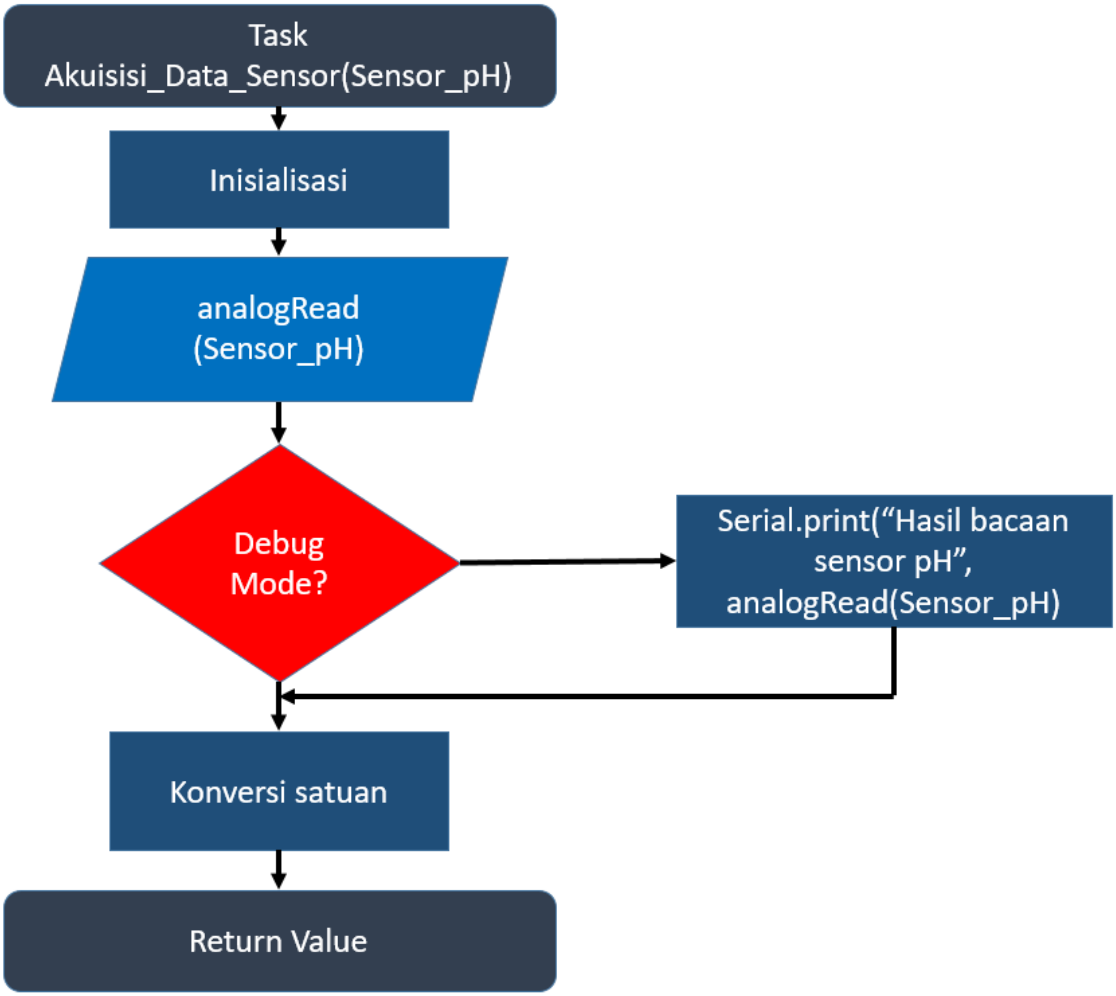
Gambar 3. 15 Skematik Rangkaian Sensor pH pada Modul RPM

Gambar 3. 16 Implementasi Skematik Rangkaian Sensor pH pada Modul RPM

Sensor pH menggunakan digital pin untuk mengirim informasi/data hasil pengukuran ke arduino. Pada sensor pH tidak dibutuhkan pull-up resistor seperti yang digunakan pada sensor temperature. Namun transistor npn 2n222 tetap digunakan sebagai *switch.*

Hasil implementasi dari rangkaian sensor pH pada modul RPM dapat dilihat pada gambar 3.13. Seperti terlihat pada gambar, rangkaian sensor pH menggunakan relay 2 channel untuk mengatur nyala-mati dari sensor tersebut.

3.1.3.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 17 *Flowchart* untuk Sensor pH

Pada implementasi dari *flowchart* diatas, software tersebut akan mempunyai fungsi getpH() yang berfungsi untuk mengambil data pH dalam bentuk bit dan mengkonversinya ke mV, untuk kemudian dikonversi lagi menjadi satuan pH. Dalam fungsi ini juga akan diberikan angka kalibrasi sesuai dengan hasil percobaan yang nantinya akan dilakukan. Dalam percobaan tersebut, nantinya akan dibandingkan hasil bacaan sensor dengan larutan kalibrasi, kemudian selisih dari hasil bacaan dan pH pada larutan sensor tersebut akan

dijadikan nilai kalibrasi. Fungsi ini akan mempunyai output bacaan data dari sensor pH dalam satuan pH dengan variable pHValue.

Software diatas telah berhasil di-*compile* dan diimplementasikan pada PCB modul RPM yang sudah dibuat. Berikut menrupakan hasil *screenshoot* dari serial monitor yang menunjukkan bacaan sensor pH:

Gambar 3. 18 Hasil Bacaan Sensor pH pada Serial Monitor

3.1.3.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi sensor pH adalah sebagai berikut :

- Hasil pembacaan belum dapat dipastikan sesuai dengan kadar pH yang sebenarnya dari air / sampel. Masalah ini dapat diatasi dengan melakukan kalibrasi dan perbandingan hasil ukur antara sensor dengan pH-meter.
- Transistor BJT belum dapat berfungsi sebagai switch, untuk mengatasi masalah ini akan diteliti lagi 2 kemungkinan permasalahannya. Kedua kemungkinan tersebut adalah transistornya yang mengalami kerusakan atau arus yang dihasilkan tidak mencukupi sehingga nilai resistor di base transistor harus diubah.

Update kondisi kedua permasalahan tersebut:

- Untuk masalah akurasi sensor suhu akan dibahas pada dokumen pengujian B500.
- Transistor BJT tidak berhasil berfungsi sebagai switch karena tidak bisa mengalirkan arus listrik yang cukup untuk sensor pH menyala. Untuk itu transistor BJT digantikan oleh relay 1 channel yang juga akan berfungsi sebagai switch.

3.1.4 Implementasi Sensor Salinitas Modul RPM

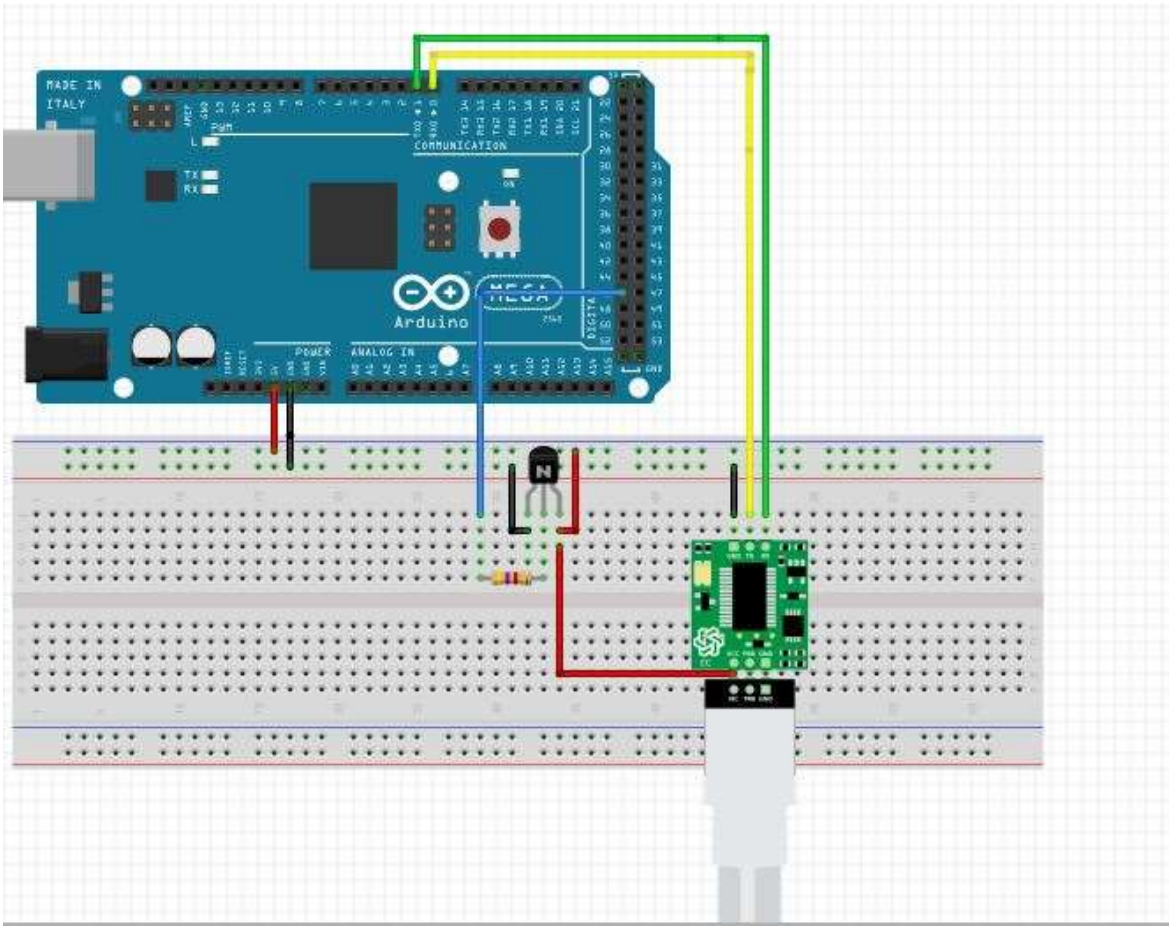
3.1.4.1 Tinjauan Spesifikasi dan Desain

Pada ujicoba ini digunakan sensor *Conductivity K0.1* keluaran Atlas Scientific. Untuk uji coba sensor salinitas(konduktivitas) ini digunakan air keran gedung labtek VIII lantai 3 ITB yang ditempatkan pada sebuah ember.

Gambar 3. 19 Sensor Salinitas (Konduktivitas) Atlas Scientific

Sensor Salinitas (Konduktivitas) dari Atlas Scientific mempunyai spesifikasi yang sesuai untuk melakukan pengukuran konduktivitas dalam rentan 0.07-50000µS/cm. Pemilihan penggunaan sensor ini didasarkan juga dengan alasan sensor dapat bertahan lama di dalam air, selain itu sensor memiliki lifespan yang cukup lama yaitu 10 tahun. Terakhir pada proses pemilihan sensor ini juga dipertimbangkan harga dan ketersedian sensor di pasaran Indonesia.

3.1.4.2 Hasil Implementasi Perangkat Keras

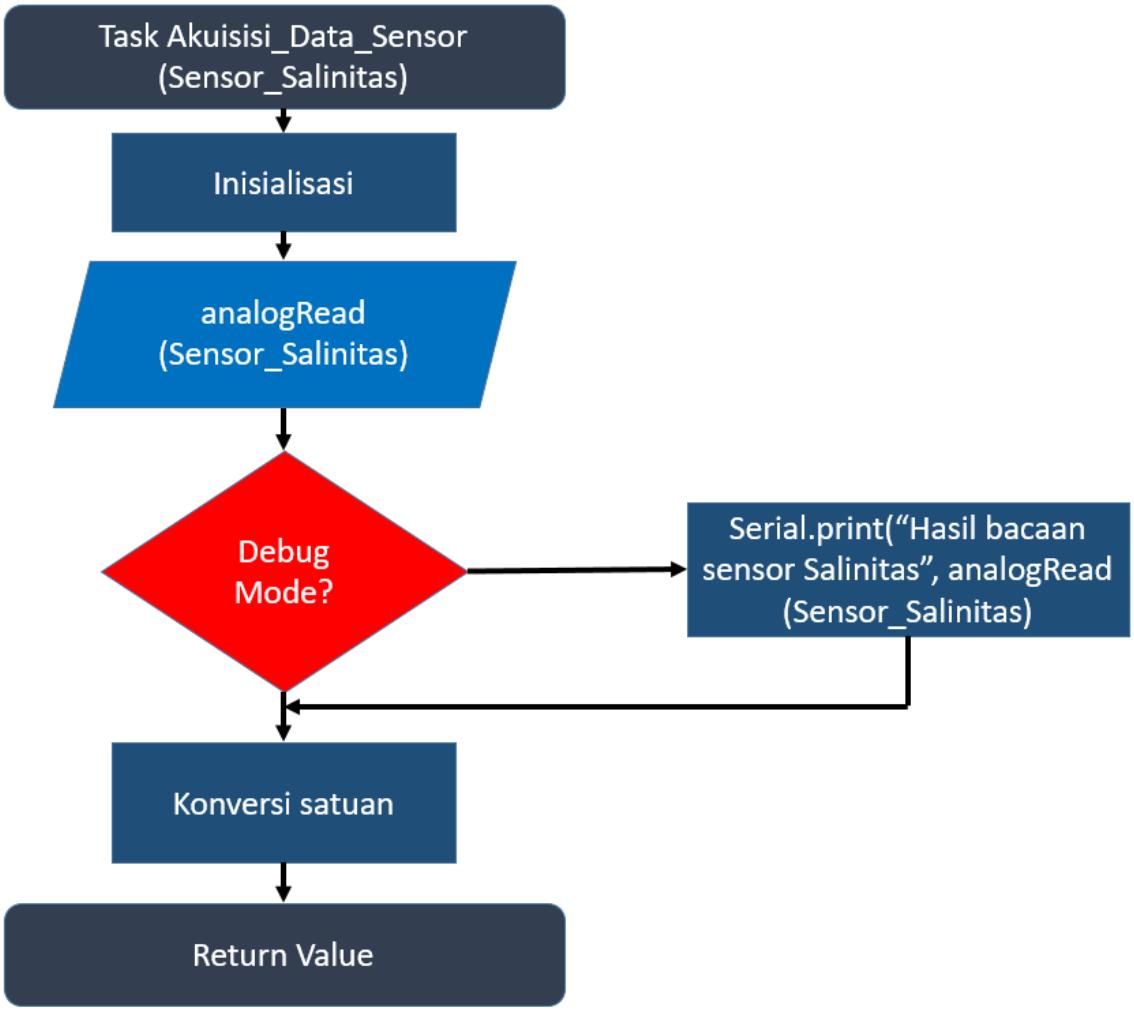


Gambar 3. 20 Skematik Rangkaian Sensor Salinitas(Konduktivitas)

Gambar 3. 21 Implementasi Skematik Rangkaian Sensor Salinitas(Konduktivitas)

Sensor Salinitas menggunakan serial komunikasi rx tx untuk mengirim dan menerima data. Dengan komunikasi serial ini dapat diatur kapan arduino mega sebagai microcontroller akan mengirim sinyal kepada sensor DO agar sensor melakukan pengambilan data dan kemudian menghentikan pengambilan data ketika sudah tidak diperlukan. Sensor DO juga harus beroperasi ketika keempat sensor lainnya dalam keadaan tidak aktif karena bila sensor DO beroperasi ketika sensor lainnya aktif maka pembacaan dari sensor DO akan terganggu. Gangguan ini datang dari adanya arus listrik kecil yang bocor ke air sekitar *probe.* Maka dari itu untuk menanggulangi permasalahan tersebut akan digunakan transistor BJT npn sebagai switch on-off. Sehingga kelima sensor akan diatur untuk menyala satu per satu.

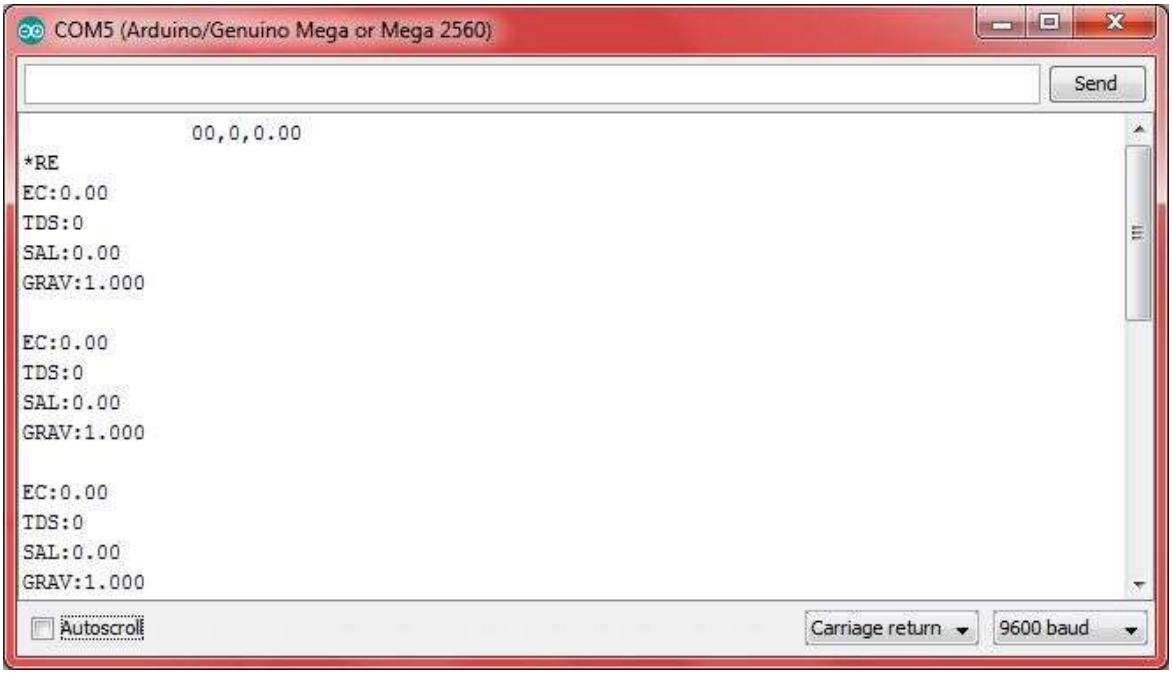
3.1.4.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 22 *Flowchart* untuk Sensor Salinitas

Hasil dari implementasi perangkat lunak sensor salinitas akan memiliki output hasil bacaan dalam satuan mg/L. Pada dasarnya, sensor ini akan melakukan pembacaan dalam bit kemudian software dari sensor salinitas akan mengkonversi nilai dalam bit tersebut ke dalam 4 variabel seperti yang dapat dilihat pada *screenshoot serial monitor* dibawah ini. Software diatas telah berhasil di-*compile* dan diimplementasikan pada PCB modul RPM

yang sudah dibuat. Berikut menrupakan hasil *screenshoot* dari serial monitor yang menunjukkan bacaan sensor salinitas:



Gambar 3. 23 Hasil Bacaan Sensor salinitas pada Serial Monitor

3.1.4.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi sensor Salinitas adalah sebagai berikut :

- Hasil pembacaan belum dapat dipastikan sesuai dengan kadar salinitas yang sebenarnya dari air / sampel. Masalah ini dapat diatasi dengan melakukan kalibrasi dan perbandingan hasil ukur sensor dengan alat ukur salinitas yang sudah terkalibrasi.
- Transistor BJT belum dapat berfungsi sebagai switch, untuk mengatasi masalah ini akan diteliti lagi 2 kemungkinan permasalahannya. Kedua kemungkinan tersebut adalah transistornya yang mengalami kerusakan atau arus yang dihasilkan tidak mencukupi sehingga nilai resistor di base transistor harus diubah.

Update kondisi kedua permasalahan tersebut:

- Untuk masalah akurasi sensor suhu akan dibahas pada dokumen pengujian B500.
- Transistor BJT sudah berhasil untuk berfungsi sebagai switch yang akan menyalakan dan mematikan sensor suhu setelah dilakukan *debugging* terhadap software yang digunakan.

3.1.5 Implementasi Sensor Oksigen Terlarut (DO) Modul RPM

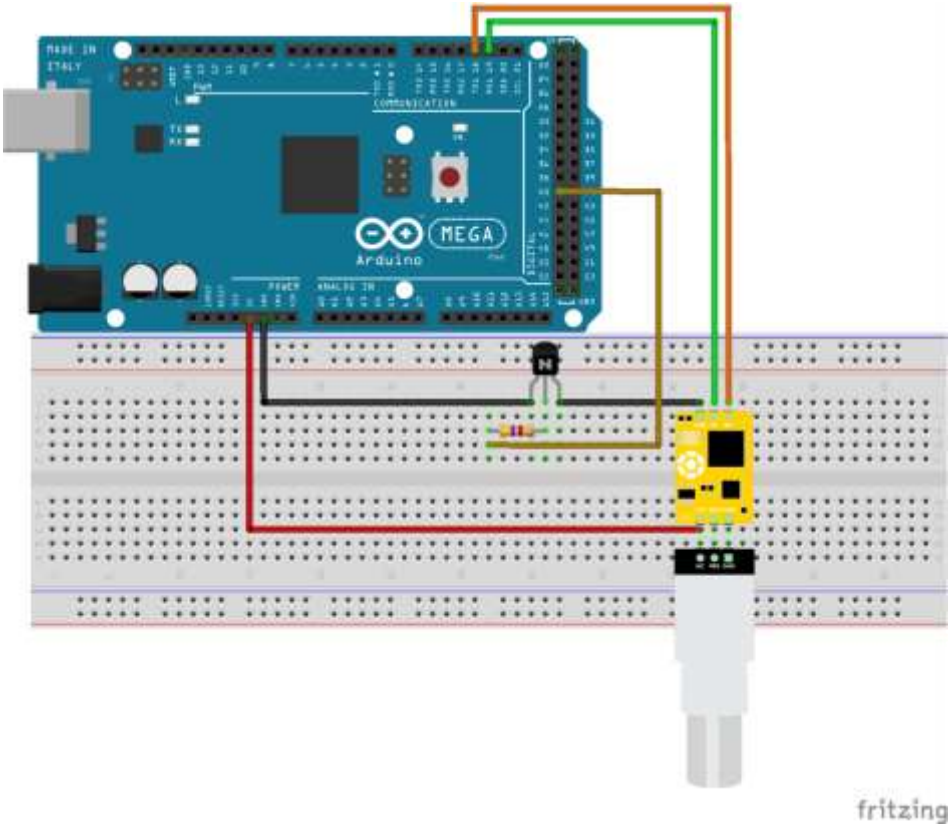
3.1.5.1 Tinjauan Spesifikasi dan Desain

Pada ujicoba ini digunakan sensor *dissolved oxygen* (DO) keluaran Atlas Scientific. Untuk uji coba sensor DO ini digunakan air keran gedung labtek VIII lantai 3 ITB yang ditempatkan pada sebuah ember.

Gambar 3. 24 Sensor DO Atlas Scientific

DO (*Dissolved Oxygen)* atau oksigen terlarut dalam air merupakan salah satu parameter kualitas air kolam yang paling penting. Selain penting, parameter ini juga sangat rawan akibat nilainya yang dapat naik dan turun secara drastic dalam kurun waktu 24 jam. Untuk itu dipilih sensor DO dari Atlas Scientific yang sudah mempunyai spesifikasi yang sesuai untuk melakukan pengukuran DO pada rentang nilai 0.01 – 35.99 mg/L dan dengan tingkat akurasi ±0.05 mg/L. Dengan keakuratan pengukuran tersebut diharapkan sistem monitoring cerdas tambak udang vannamei dapat mendeteksi perubahan nilai DO yang membahayakan udang pada tambak. Selain itu, pemilihan sensor ini juga berdasarkan kemampuan sensor DO Atlas Scientific yang mampu digunakan untuk pengukuran di dalam air dalam jangka waktu yang lama. Terakhir pada proses pemilihan sensor ini juga dipertimbangkan harga dan ketersedian sensor di pasaran Indonesia.

3.1.5.2 Hasil Implementasi Perangkat Keras

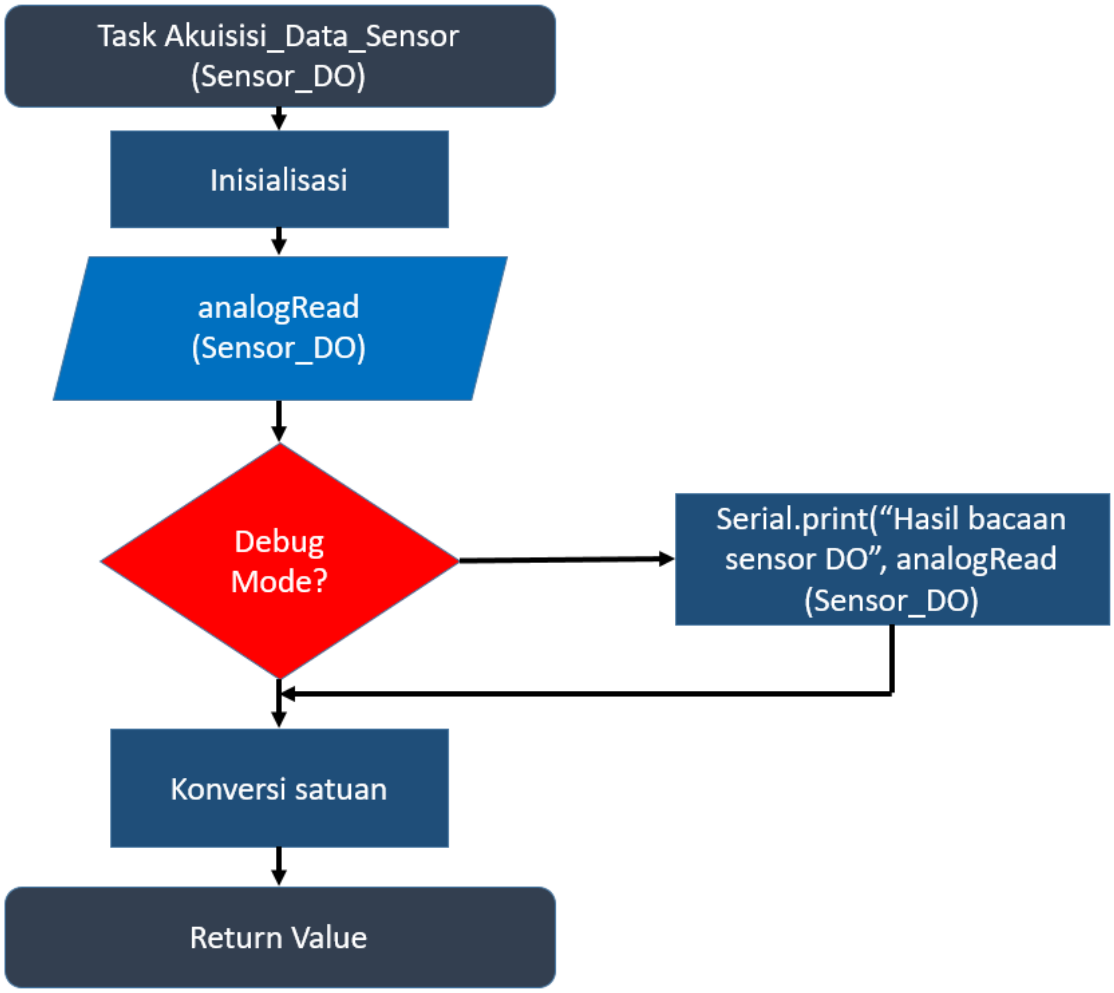


Gambar 3. 25 Skematik Rangkaian Sensor DO

Gambar 3. 26 Implementasi Skematik Rangkaian Sensor DO

Sensor DO menggunakan serial komunikasi rx tx untuk mengirim dan menerima data. Dengan kemampuan untuk mengirim dan menerima data, sensor DO dapat memanfaatkan serial komunikasi ini untuk fungsi on-off sensor. Dengan fungsi ini dapat diatur kapan arduino mega sebagai microcontroller akan mengirim sinyal kepada sensor DO agar sensor Melakukan pengambilan data dan kemudian menghentikan pengambilan data ketika sudah tidak diperlukan.

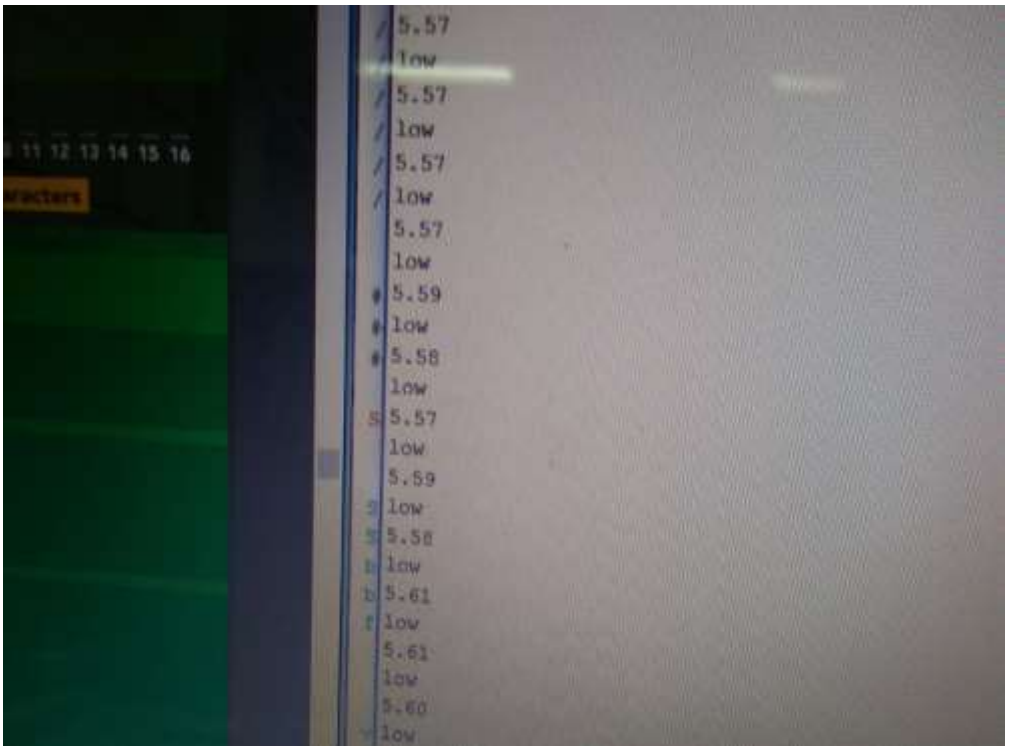
3.1.5.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 27 *Flowchart* untuk Sensor DO

Hasil implementasi dari perangkat lunak sensor DO akan memiliki output hasil bacaan sensor DO dalam satuan mg/L. Pada implementasi software ini, sensor DO menggunakan komunikasi serial untuk komunikasi ke board arduino. Kemudian untuk menghidupkan dan mematikan sensor DO digunakan fungsi digitalWrite(). Pada perangkat lunak ini juga diberi fungsi untuk mengkompensasi hasil bacaan DO setiap 50 kali bacaan. Nilai DO akan dikompensasi oleh nilai temperature dari hasil bacaaan sensor temperature.

Software diatas telah berhasil di-*compile* dan diimplementasikan pada PCB modul RPM yang sudah dibuat. Berikut menrupakan hasil *screenshoot* dari serial monitor yang menunjukkan bacaan sensor DO:



Gambar 3. 28 Hasil Bacaan Sensor DO pada Serial Monitor

3.1.5.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi sensor DO adalah sebagai berikut :

- Hasil pembacaan belum dapat dipastikan sesuai dengan kadar DO yang sebenarnya dari air / sampel. Masalah ini dapat diatasi dengan melakukan kalibrasi dan perbandingan hasil ukur sensor dengan alat ukur DO yang sudah terkalibrasi
- Transistor BJT belum dapat berfungsi sebagai switch, untuk mengatasi masalah ini akan diteliti lagi 2 kemungkinan permasalahannya. Kedua kemungkinan tersebut adalah transistornya yang mengalami kerusakan atau arus yang dihasilkan tidak mencukupi sehingga nilai resistor di base transistor harus diubah.

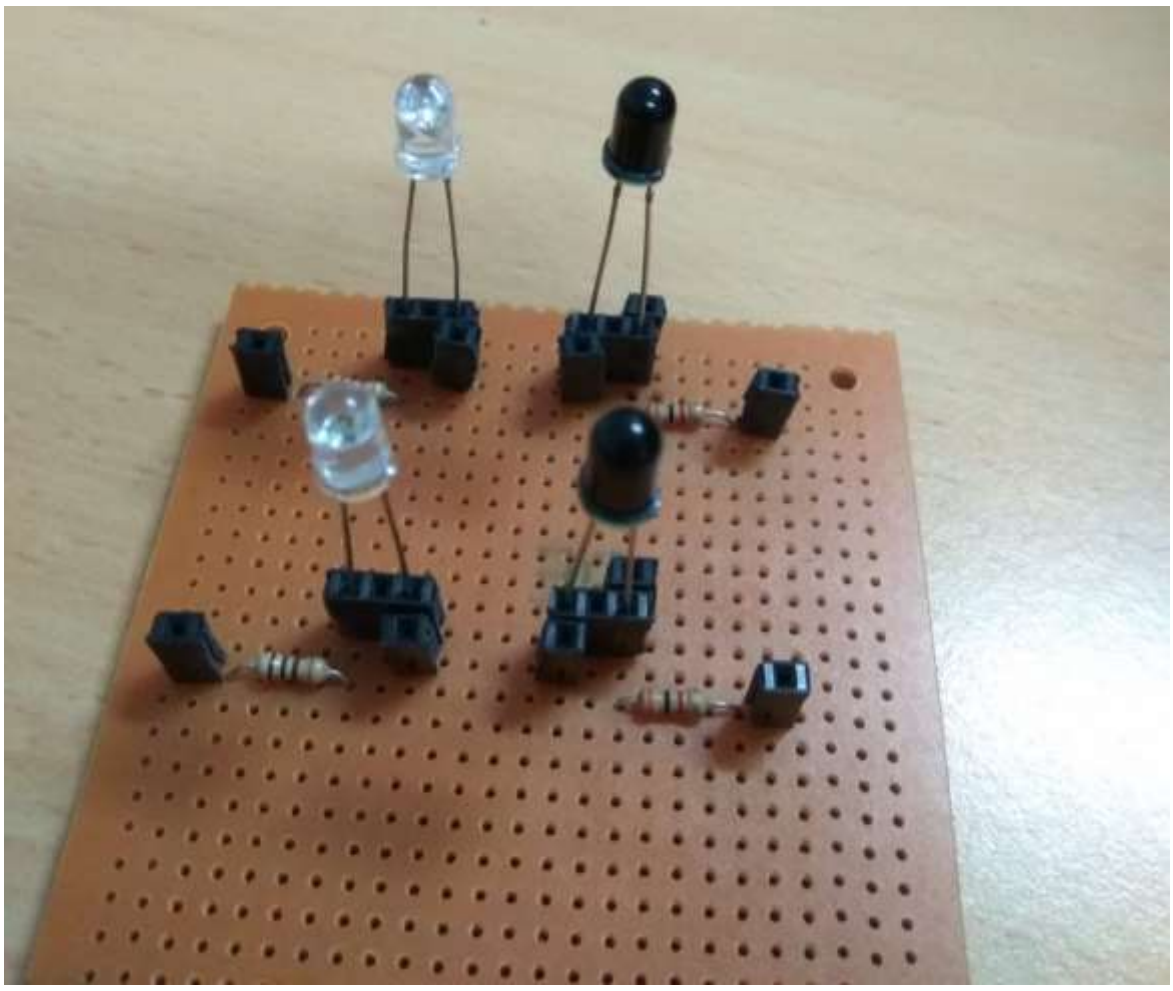
Update kondisi kedua permasalahan tersebut:

- Untuk masalah akurasi sensor suhu akan dibahas pada dokumen pengujian B500.
- Transistor BJT sudah berhasil untuk berfungsi sebagai switch yang akan menyalakan dan mematikan sensor suhu setelah dilakukan *debugging* terhadap software yang digunakan.

3.1.6 Implementasi Sensor Kekeruhan Air Modul RPM

3.1.6.1 Tinjauan Spesifikasi dan Desain

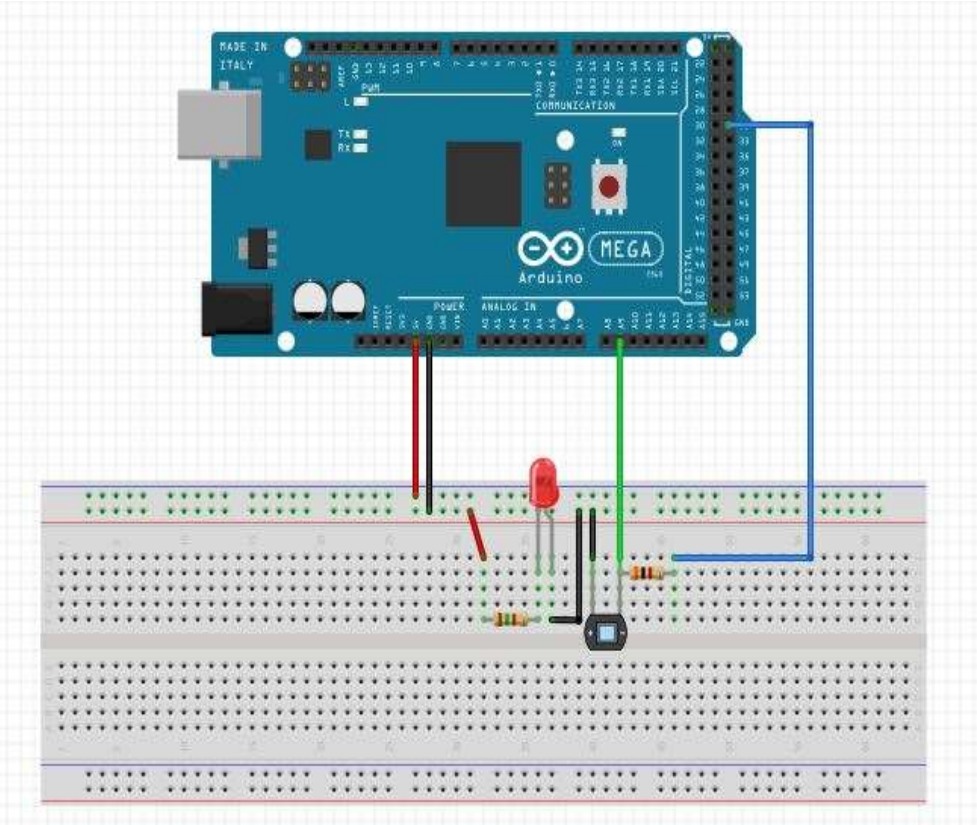
Sensor kekeruhan air ini dibuat dengan memanfaatkan LED IR dan fotodioda. Konsep kerja dari sensor ini adalah mengukur banyaknya cahaya inframerah yang masuk dan dideteksi oleh fotodioda. Semakin banyak cahaya yang masuk artinya semakin banyak partikel yang memantulkan cahaya inframerah ke fotodioda. Semakin banyak partikel berarti airnya semakin keruh. Untuk uji coba sensor pH ini digunakan air keran gedung labtek VIII lantai 3 ITB yang ditempatkan pada sebuah ember



Gambar 3. 29 Sensor Kekeruhan Air (Turbidity Meter)

Sensor ini menggunakan cahaya inframerah karena cahaya inframerah memiliki panjang gelombang yang lebih panjang dibandingkan cahaya tampak. Pemasangan inframerah dan photodiode akan mengikuti standar ISO 7027, yaitu transmitter dan receiver akan dipasang dengan derajat pemasangan 90^0

3.1.6.2 Hasil Implementasi Perangkat Keras

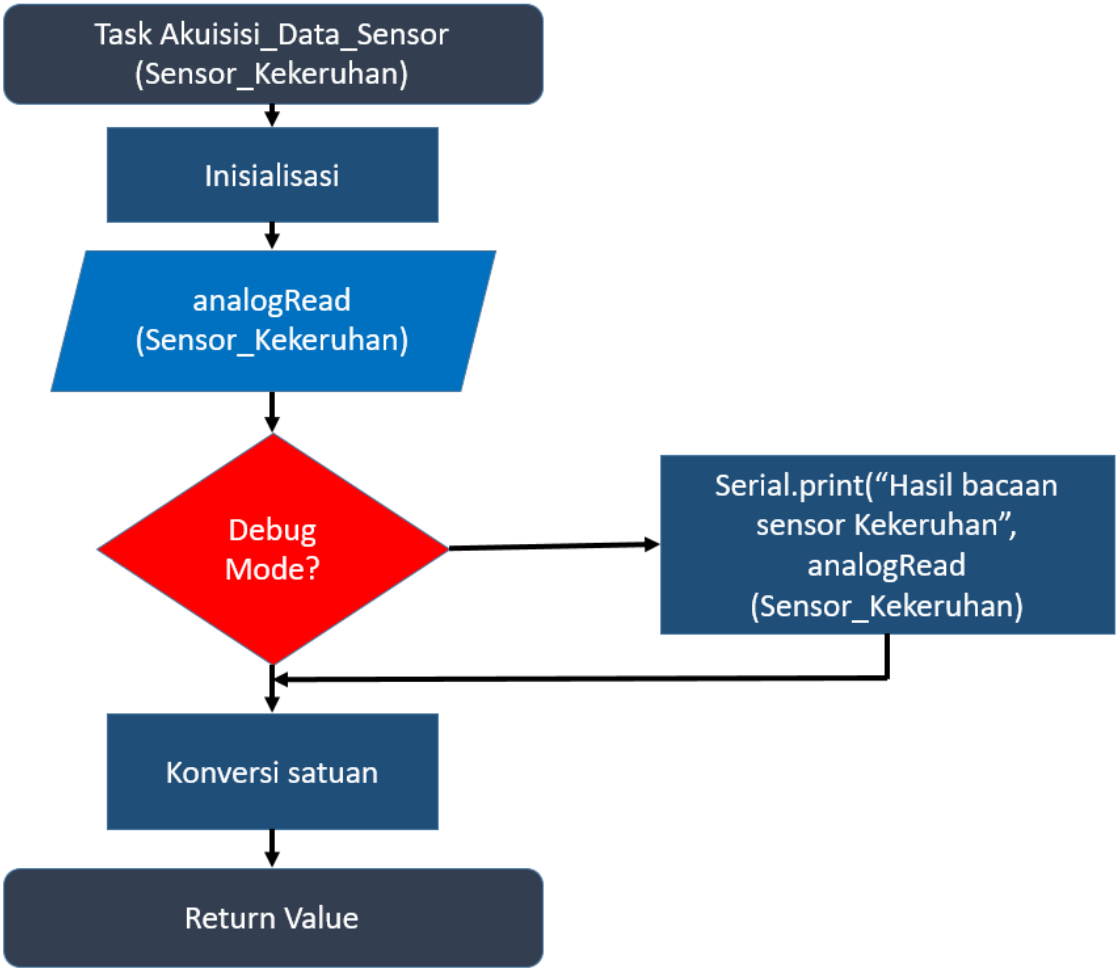


Gambar 3. 30 Skematik Rangkaian Sensor Kekeruhan Air pada Modul RPM

Gambar 3. 31 Implementasi Skematik Rangkaian Sensor Kekeruhan Air pada Modul RPM

Rangkaian sensor kekeruhan akan dibuat dalam casing yang tahan air sehingga dapat diletakan di dalam tambak. Transmitter dan receiver akan dipasang membentuk sudut 90^0 dimana receiver menghadap ke dasar tambak supaya dapat mengurangi pengaruh sinar matahari ketika pembacaan sensor di siang hari.

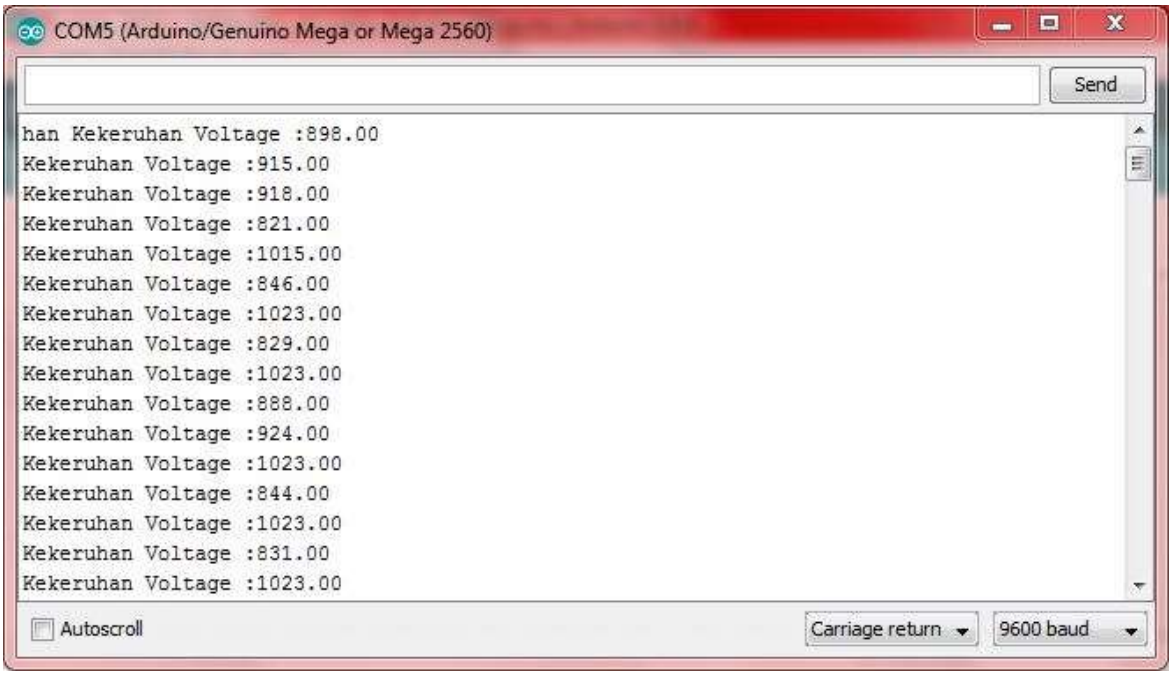
3.1.6.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 32 Flowchart untuk Sensor Kekeruhan Air

Implementasi perangkat lunak ini akan mengkonversi nilai analog yang diperoleh menjadi tegangan. Kemudian nilai tegangan tersebut dapat digunakan untuk menjadi batas tingkat kekeruhan air.

Software diatas telah berhasil di-*compile* dan diimplementasikan pada PCB modul RPM yang sudah dibuat. Berikut menrupakan hasil *screenshoot* dari serial monitor yang menunjukkan bacaan sensor kekeruhan air:



Gambar 3. 33 Hasil Bacaan Sensor kekeruhan air pada Serial Monitor

3.1.6.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi sensor kekeruhan air adalah sebagai berikut :

- Hasil pembacaan tingkat kekeruhan air tidak memiliki standar / satuan cerah atau keruh nya air. Untuk mengatasi masalah ini akan ditentukan batas-batas keadaan air tergolong cerah atau keruh dengan menggunakan secchi disk sebagai acuan tingkat kekeruhan air.

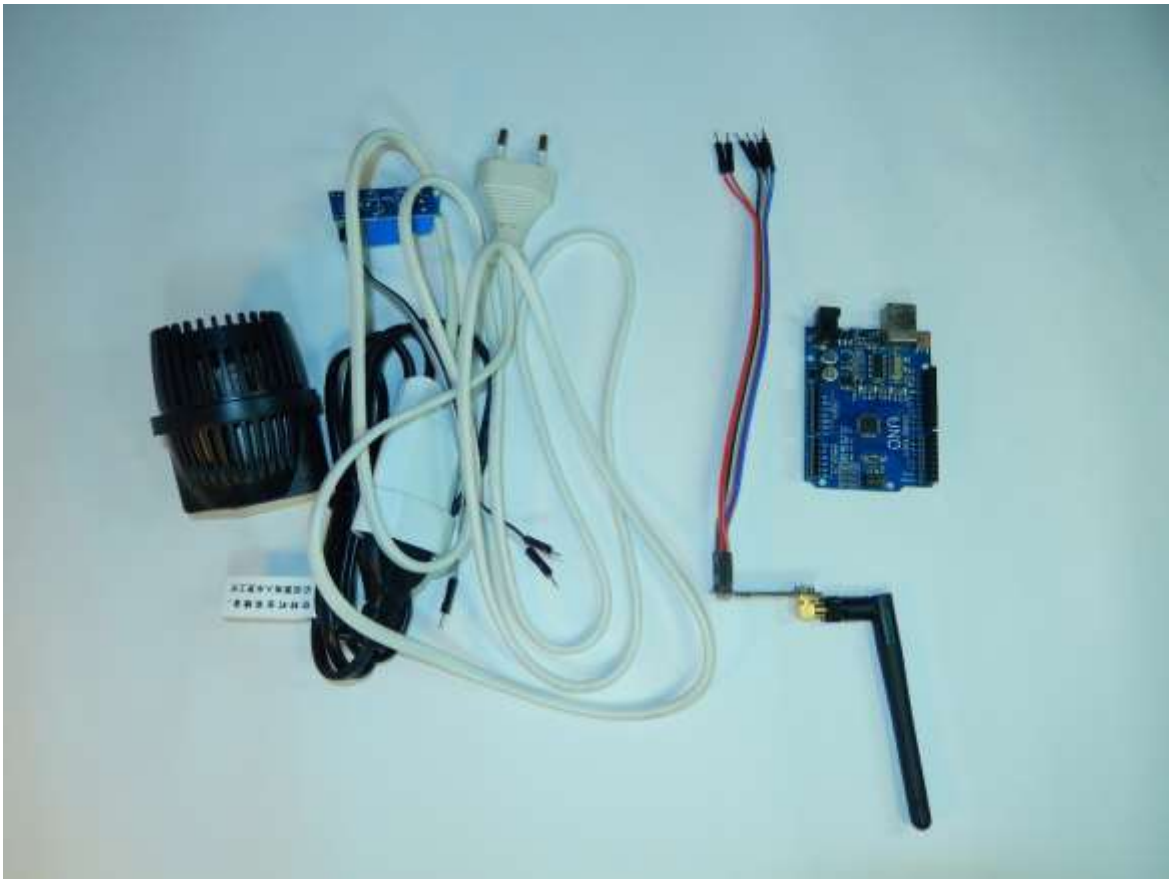
Update:

- Sensor kekeruhan air akan diuji dan dikalibrasi nilainya sehingga diketahui batas – batas keadaan air yang tergolong cerah atau keruh pada dokumen pengujian B500.

3.1.7 Implementasi Relay Kincir pada RPM

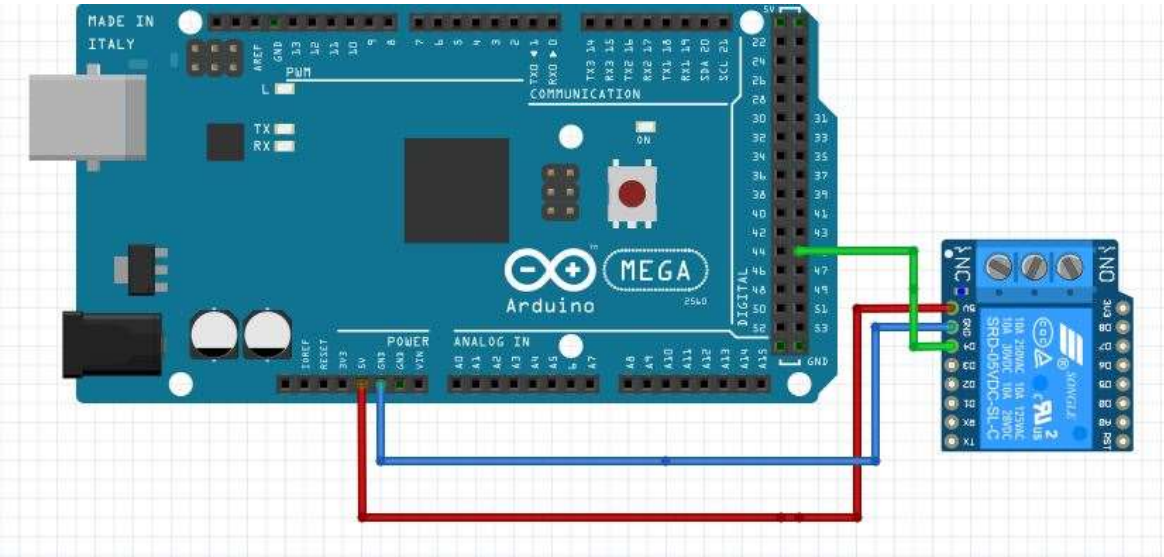
3.1.7.1 Tinjauan Spesifikasi dan Desain

Relay pada modul kincir air ini menggunakan Relay 1 channel yang dihubungkan pada kabel / socket yang menjadi sumber daya dari model kincir air yang digunakan. Tegangan input yang digunakan adalah 5V (sesuai dengan tegangan keluaran dari Arduino).



Gambar 3. 34 Relay dan Kincir Air pada Modul RPM

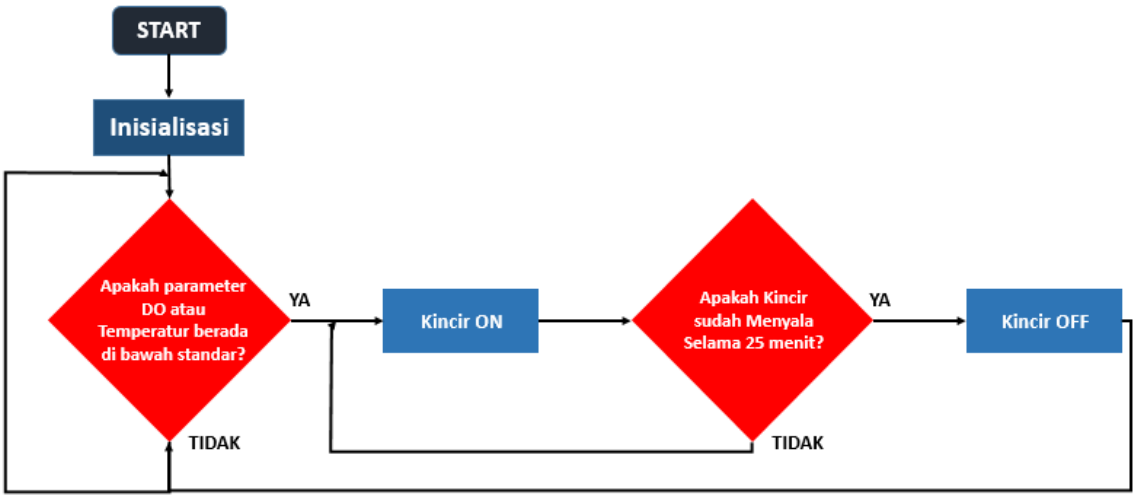
3.1.7.2 Hasil Implementasi Perangkat Keras



Gambar 3. 35 Skematik Rangkaian Relay untuk Kincir Air pada Modul RPM

Relay disambungkan ke sambungan kabel / socket sehingga dapat berperan sebagai sumber listrik untuk kincir air.

3.1.7.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 36 *Flowchart* untuk Modul Kincir Air

Modul kincir air akan mengatur secara otomatis kapan kincir akan menyala dan mati. Kincir akan menyala ketika keadaan parameter DO tidak sesuai dengan standar. Ketika keadaan tidak sesuai dengan keadaan normal, kincir akan menyala selama 25 menit. Kemudian modul kincir akan memeriksa kembali keadaan DO. Sinyal on / off yang diterima modul Kincir, dikirim dari modul HMI. Sinyal yang dikirim berupa trigger yang bernilai 1 untuk menyalakan relay pada modul kincir air sehingga kincir menyala. Dibawah ini dapat dilihat *flowchart* untuk modul kincir air

3.1.7.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi relay pada kincir adalah sebagai berikut :

- Penggunaan model kincir air dengan menggunakan kipas tidak dapat digunakan di dalam air, sehingga untuk mengatasi masalah ini dipilih kincir air khusus akuarium sehingga kincir dapat digunakan didalam air (*waterproof).*

3.2 Implementasi Modul HMI

3.2.1 Pengantar

Human Machine Interface (HMI) adalah salah satu modul dari e-Shrimp : sistem monitoring cerdas untuk tambak udang vannamei yang berfungsi sebagai alat berupa tampilan yang berfungsi sebagai penghubung antara manusia pengguna (*user*) dengan e-Shrimp.

3.2.1.1 Tinjauan Spesifikasi dan Desain

HMI dirancang untuk memenuhi spesifikasi sebagai berikut :

- Melakukan penerimaan data dari modul sensor
- Menampilkan kondisi kualitas air kolam pada *user*
- Melakukan penyimpanan data pengukuran pada *storage device*
- Melakukan aksi / komunikasi ke user ketika ada kondisi air tambak yang tidak normal
- Melakukan komunikasi ke relay untuk membangkitkan kincir ketika ada kondisi air tambak yang tidak normal
- Memiliki menu untuk mengatur HMI dan dapat diinput oleh *user*

Dari spesifikasi yang telah ditentukan, modul HMI dirancang dengan menggunakan komponen dan modul sebagai berikut :

- Mikrokontroller Arduino Mega : sebagai sistem kontroller untuk menerima input dan menghasilkan output untuk komunikasi antara perangkat-perangkat keras yang ada pada HMI.
- Transceiver nRF24L01+ untuk penerimaan data dari RPM dan mengirimkan sinyal ke relay kincir
- LCD 20x4 untuk sebagai layar menampilkan waktu, kondisi kualitas air kolam dan navigasi menu HMI
- LED untuk menampilkan kondisi kualitas air kolam (hijau : normal dan merah : tidak normal)
- SD Card Module dan SD Card untuk penyimpanan data yang telah diperoleh RPM
- Modul GSM SIM900A Mini dan SIM Card untuk melakukan *broadcasting* SMS
- Relay 2-Channel : untuk melakukan switching sirine horn
- Sirine Horn SK-103 untuk menghasilkan suara notifikasi ketika ada air kolam yang berada di kondisi tidak normal
- Keypad sebagai modul untuk komunikasi *user* ke mesin dalam menentukan menu yang ingin ditampilkan dan penggantian nomor HP tujuan SMS

3.2.1.2 Lingkungan Implementasi

Lingkungan implementasi dari modul HMI dibagi menjadi 2 bagian antara lain :

1. Spesifikasi Hardware

   Spesifikasi hardware yang diperlukan untuk membangun sistem adalah sebagai berikut :

   - Frekuensi RF *unlicensed* dengan standar ISM : 2,4 GHz
   - SD Card dengan kapasitas 4 GB
   - LCD 20 x 4
   - Keypad 4 x 4
   - GSM
   - RTC
   - Adapter 220V AC ke 12V DC
   - DC Step Down 12V ke 5V
   - DC Step Down 12V ke 3,3V
   - 14 LED
   - Speaker + Lampu Sirine

2. Spesifikasi Sofware

   Spesifikasi software yang digunakan dalam pembuatan sistem adalah sebagai berikut :

   - Sistem Operasi : Windows 10
   - Program Aplikasi : Arduino 1.6.7
   - Program Tambahan : Notepad++, Microsoft Office, Microsoft Excel, Eagle CAD 7.6.0

3.2.1.3 Hasil Implementasi Keseluruhan

Gambar 3. 37 Hasil Implementasi Keseluruhan Modul HMI

3.2.1.4 Permasalahan dan Solusi Implementasi Keseluruhan

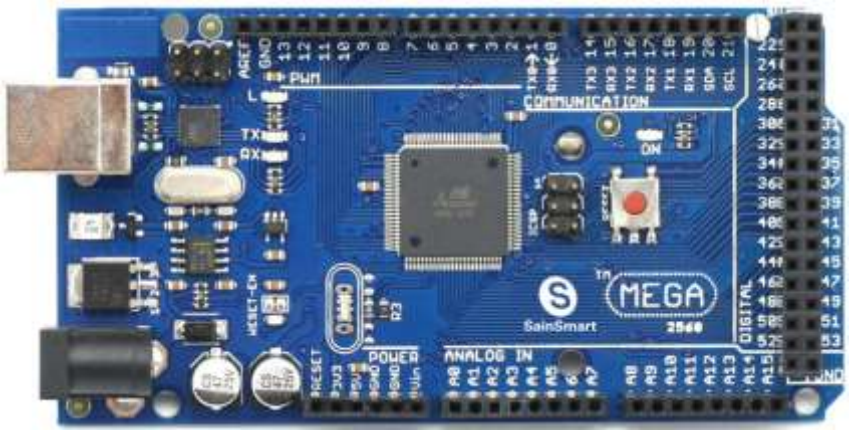
Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul HMI secara keseluruhan adalah sebagai berikut :

- Implementasi awal masih dilakukan tanpa menggunakan respon langsung dari RPM. Namun untuk transmisi sudah dapat diverifikasi dengan menggunakan dummy (berupa mikrokontroller Arduino Uno yang dihubungkan dengan RF24) yang mengirimkan data berupa *array of float*
- Implementasi masih dilakukan didalam breadboard, untuk kedepannya akan dilakukan implementasi dengan pembuatan PCB tambahan dan Arduino Mega shield agar hubungan modul dan kabel pada HMI bisa lebih rapi dan membutuhkan ruang yang seminimal mungkin
- Sampai saat ini belum dilakukan implementasi untuk melakukan koneksi transmisi sumber listrik dari jala-jala 220V ke setiap perangkat. Implementasi transmisi sumber listrik akan dilakukan pada minggu ke-9

3.2.2 Implementasi Mikrokontroller pada Modul HMI

3.2.2.1 Tinjauan Spesifikasi dan Desain

HMI memerlukan sebuah mikrokontroller untuk dapat mengendalikan rangkaian listrik, komponen dan modul pada HMI untuk melakukan kinerja yang diinginkan. Dalam implementasi pada riset ini, digunakan Arduino Mega 2560 sebagai mikrokontroller HMI sesuai dengan kebutuhan yang telah disebutkan pada desain sebelumnya.

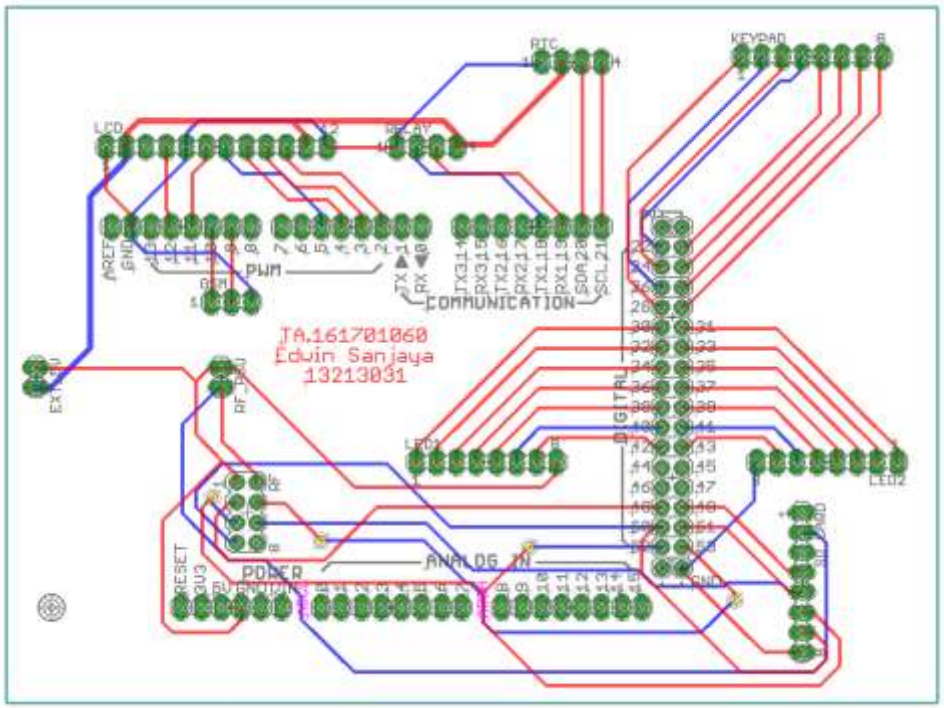


Gambar 3. 38 Arduino Mega 2560

Ketika melakukan *wiring* antara Arduino Mega 2560 dengan komponen lainnya, karena beberapa komponen memerlukan PIN yang spesifik (seperti *interrupt*, I2C dan SPI), maka *wiring* yang dihasilkan cenderung tidak rapi. Oleh karena itu dibuat sebuah Arduino Mega Shield, yang berfungsi untuk menyatukan PIN yang dibutuhkan pada setiap komponen sehingga *wiring* dapat dilakukan dengan lebih rapi.

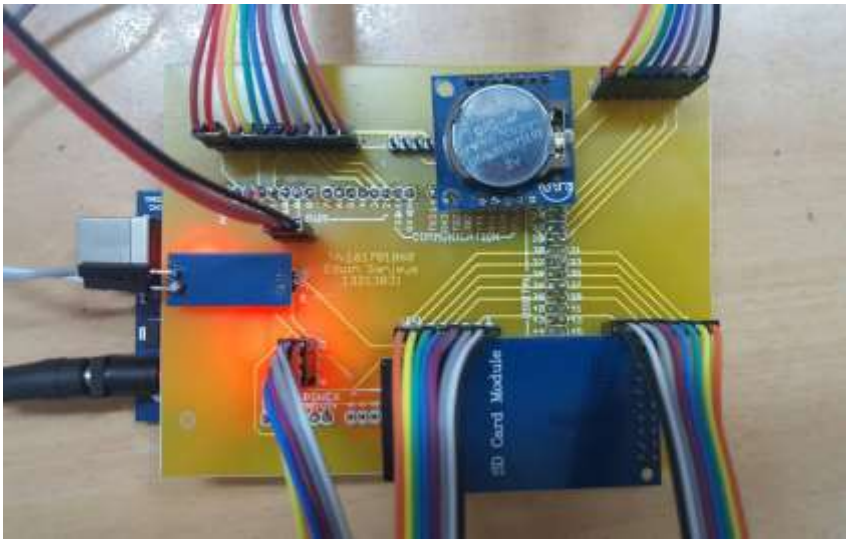
3.2.2.2 Hasil Implementasi Perangkat Keras

Dalam proses pembuatan board ini, digunakan software Eagle CAD 7.6.0 untuk melakukan desain PCB dan proses pencetakan dilakukan di Spectra Bandung. Berikut merupakan desain board PCB yang dibuat :



Gambar 3. 39 Desain Board PCB Arduino Mega Shield

Berikut merupakan hasil implementasi dari board yang telah dibuat dan diintegrasi dengan komponen HMI lainnya :



Gambar 3. 40 Hasil Impelementasi Hardware Arduino Mega Shield

3.2.2.3 Hasil Implementasi Perangkat Lunak

Seluruh source-code Arduino yang dikompilasi pada mikrokontroller dapat dilihat pada lampiran. Source code juga dibagi menjadi beberapa prosedur yang dibagi berdasarkan komponen lain yang diprogram.

3.2.2.4 Permasalahan dan Solusi Implementasi

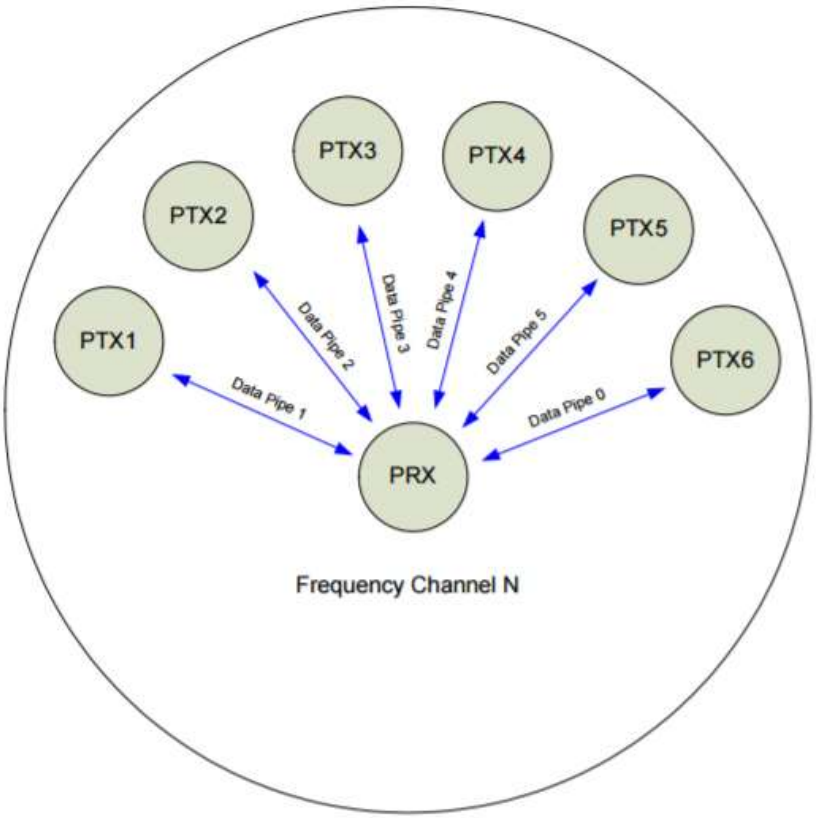
Permasalahan utama yang dihadapi ketika menggunakan Arduino Mega 2560 sebagai mikrokontroller adalah *wiring* yang tidak rapi, karena beberapa komponen membutuhkan PIN dengan kebutuhan yang spesifik, sedangkan beberapa PIN tersebut terkadang memiliki jarak yang cukup jauh. Oleh karena itu Arduino Mega Shield merupakan solusi yang digunakan pada riset ini untuk mengatasi masalah *wiring*.

3.2.3 Implementasi Transceiver pada Modul HMI

3.2.3.1 Tinjauan Spesifikasi dan Desain

Transceiver pada HMI berfungsi untuk menerima data-data nilai parameter yang telah diukur oleh modul RPM, sebagai pengembangan dari riset sebelumnya diharapkan agar HMI memiliki kemampuan untuk melakukan penerimaan data dari beberapa modul RPM sehingga pemilihan transceiver sebagai pengirim data harus dipertimbangkan dengan baik. Transceiver yang digunakan adalah nRF24L01+ yang memiliki kemampuan untuk melakukan pengiriman dan penerimaan data secara wireless dengan menggunakan pita frekuensi di 2,4 GHz. Dimana frekuensi 2,4 GHz ini merupakan pita frekuensi tanpa lisensi dengan standar ISM (*Industrial, Scientific* dan *Medical*) sehingga bisa digunakan untuk melakukan implementasi pengiriman data antara HMI dengan RPM.

Kemudian salah satu pertimbangan besar penggunaan nRF24L01+ pada riset ini adalah kemampuan modul untuk melakukan penerimaan data yang dikirim dari beberapa modul nRF24L01+ yang lain, sehingga dapat mendukung sistem komunikasi paralel HMI pada e-Shrimp untuk dapat menerima data-data maksimal dari 6 buah RPM (Jumlah *data pipe* dari nRF24L01+ sebanyak 6 buah) yang beroperasi bersamaan. Namun pada riset ini, hanya digunakan dua buah RPM yang akan melakukan pengiriman data pada HMI.



Gambar 3. 41 Ilustrasi Komunikasi Pararel pada nRF24L01+

Data-data yang diterima dari HMI merupakan *array of float* dengan ukuran sebesar 5 yang berisi nilai parameter kualitas air kolam yang diukur oleh RPM. Karena HMI terhubung dengan sumber tegangan, maka HMI akan melakukan *reading* untuk menerima dapat dari RPM dalam periode yang singkat, untuk mencegah adanya data yang telah di *write* oleh RPM namun, gagal diterima oleh HMI.

0	1	2	3	4
DO	Suhu	pH	Kekeruhan	Salinitas

3.2.3.2 Hasil Implementasi Perangkat Keras

nRF24L01+ dioperasi dan dikonfigurasi melalui *Serial Peripheral Interface*, sehingga PIN pada modul nRF24L01+ ini harus disesuaikan dengan PIN mikrokontroler yang digunakan. Berikut merupakan konfigurasi PIN yang dilakukan untuk menghubungkan modul nRF24L01+ dengan mikrokontroler Arduino Mega 2560.

PIN RF24	Arduino Mega
V+	3,3 V
GND	GND

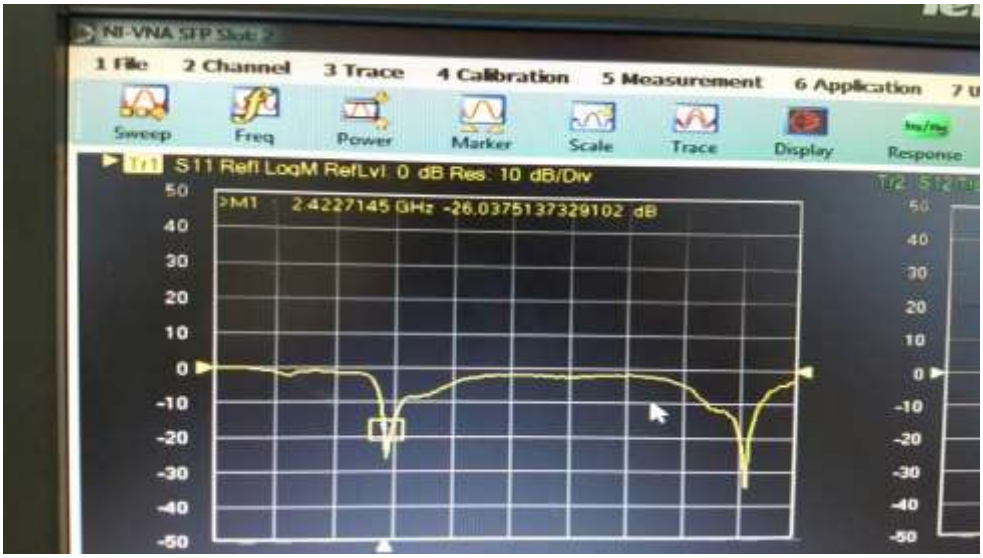
CSN	PIN 53
CE	PIN 48
MOSI	PIN 51
SCK	PIN 52
IRQ	-
MISO	PIN 50

Gambar 3. 42 Implementasi Hardware nRF24L01+

Modul nRF24L01+ memerlukan sumber tegangan sebesar 3,3V (penggunaan sumber daya 5V dapat menyebabkan modul menjadi rusak), pemasangan PIN pada mikrokontroler dilakukan karena PIN 50, 51 dan 52 merupakan PIN SPI untuk Arduino Mega (SCK, MOSI dan MISO). Bersamaan dengan modul SD Card, nRF24L01+ ini menjadi *SPI Slave* dengan Arduino sebagai *SPI Master*.

Pada implementasi sampai saat ini, sumber tegangan 3,3 yang digunakan berasal dari mikrokontroler Arduino Mega, namun pada implementasi kedepannya akan digunakan sumber tegangan yang berasal dari sumber tegangan yang lebih baik (220V AC yang dikonversi menjadi 3,3V DC) untuk meningkatkan jarak transmisi (karena ditakutkan arus yang diperlukan oleh nRF24L01+ ini tidak cukup karena Arduino harus membagi sumber tegangan ke beban lainnya).

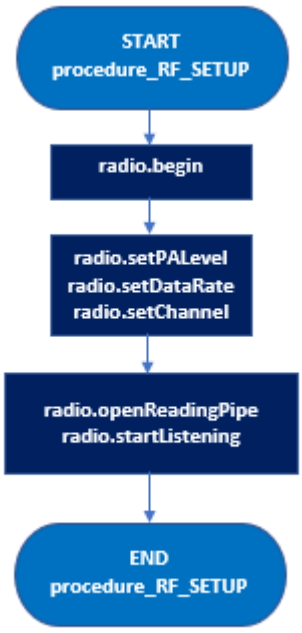
Selain melakukan konfigurasi pada modul, dilakukan juga verifikasi apakah antenna yang telah disediakan bersamaan dengan modul cocok untuk diaplikasikan untuk komunikasi RF. Verifikasi dilakukan dengan menggunakan *National Instrument – Vector Network Analzyer* untuk melihat koefisien refleksi dari antenna dan frekuensi yang paling optimal. Berikut merupakan hasil pengukuran yang telah diperoleh :



Gambar 3. 43 Verifikasi Antenna nRF24L01+ dengan NI-VNA

Dari hasil pengukuran yang telah diperoleh, dapat terlihat bahwa koefisien refleksi antenna yang rendah memiliki nilai di -26 dB atau sebesar 0,05 dengan frekuensi operasi di 2,422 GHz. Sehingga antenna telah diverifikasi untuk beroperasi di frekuensi sekitar 2,4 GHz. Dari pengukuran yang diperoleh didapatkan juga channel operasi yang terbaik adalah di channel 22 (untuk membuat frekuensi menjadi 2,422 GHz).

3.2.3.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 44 *Flowchart* procedure_RF_SETUP

`procedure_RF_SETUP` digunakan untuk melakukan pengesetan awal pada modul RF24. Prosedur ini melakukan inisialisasi modul RF24, pengesetan *range* & *channel*, membuka *data pipe* sebanyak 6 buah dan memulai penerimaan data :

- `radio.begin()` digunakan untuk menjalankan fungsi modul RF24 secara keseluruhan
- `radio.setPALevel(RF24_PA_MAX)` ; digunakan untuk mengatur besar *Power Amplifier* (PA) dalam melakukan transmisi, semakin besar nilai PA akan membutuhkan arus yang lebih besar (dalam implementasi ini, digunakan PA maksimal karena menggunakan *power supply* 12 V yang dikonversi dari sumber tegangan jala-jala)

- `radio.setDataRate(RF24_250KBPS);` digunakan untuk menentukan kecepatan pengiriman data pada modul RF. semakin rendah nilai *datarate* akan meningkatkan jarak transmisi oleh karena itu digunakan *datarate* minimum yaitu 2
- `radio.setChannel(X);` digunakan untuk menentukan *channel* komunikasi RF yang juga menentukan frekuensi operasi yang digunakan dengan persamaan berikut (dalam satuan MHz) :

$$Frekuensi\ Operasi : 2400 + X$$

Sehingga bila channel yang ditulis merupakan 108 (sesuai dengan implementasi software) dapat diketahui frekuensi operasi RF24 yang digunakan terletak pada 2,508 GHz.

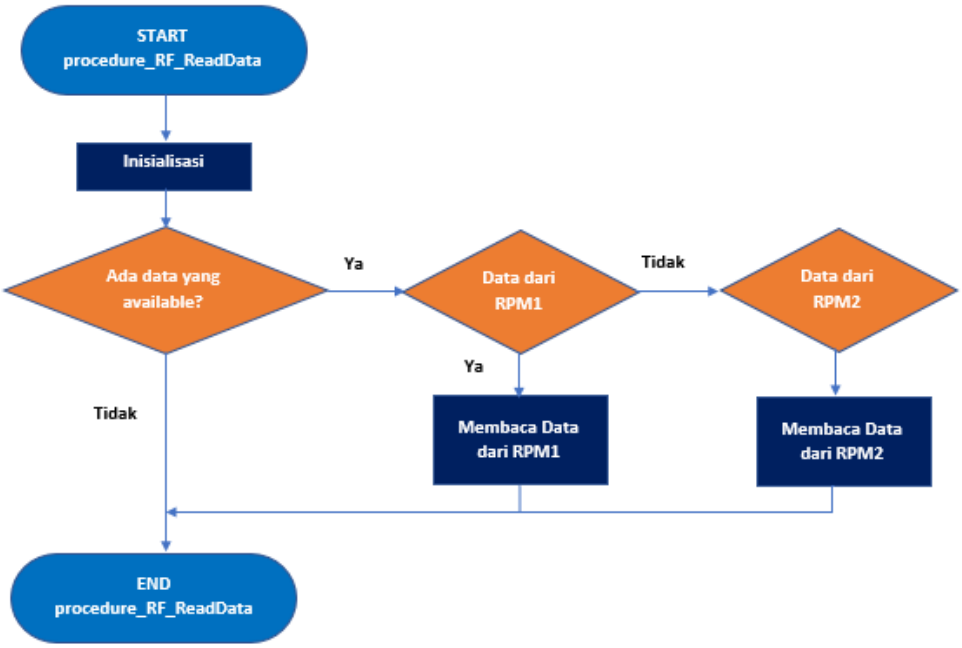
- `radio.startListening();` digunakan untuk membuat modul RF24 menjadi mode penerimaan data (*receiver*);
- `radio.setChannel(X);` digunakan untuk menentukan *channel* komunikasi RF yang juga menentukan frekuensi operasi yang digunakan dengan persamaan berikut (dalam satuan MHz) :

$$Frekuensi\ Operasi : 2400 + X$$

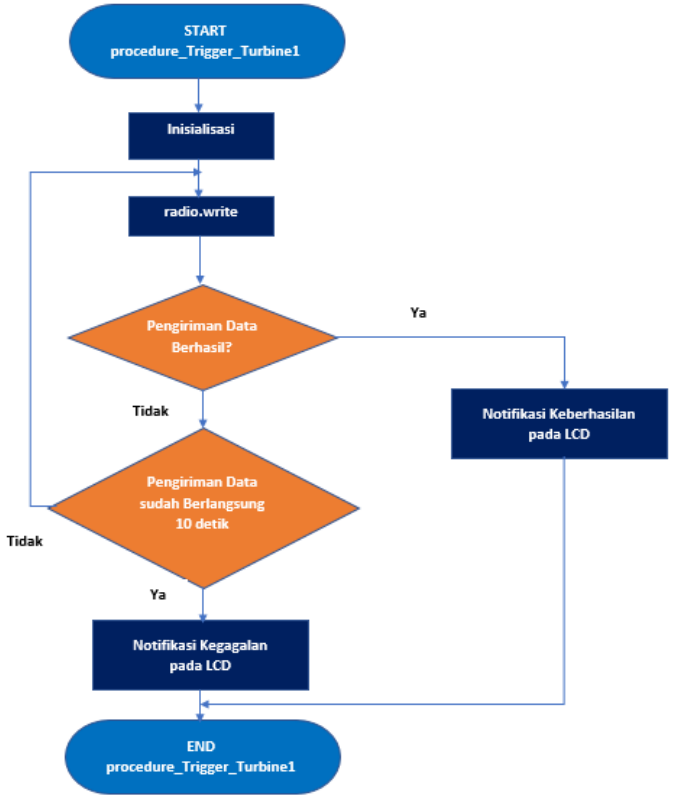
Sehingga bila channel yang ditulis merupakan 108 (sesuai dengan implementasi software) dapat diketahui frekuensi operasi RF24 yang digunakan terletak pada 2,508 GHz.

`procedure_RF_ReadData` digunakan untuk melakukan penerimaan data yang diterima dari transmitter, setelah menerima data dari suatu transmitter, program akan mendeteksi *data pipe* yang digunakan oleh transmitter dalam mengirim data, sehingga HMI dapat mengetahui RPM yang mengirim *data tersebut*. Pada riset ini hanya menggunakan dua buah RPM, sehingga hanya digunakan dua buah *data pipe* dengan nomor 0 dan 1, jika HMI menerima data dari RPM 0, maka data-data yang diterima disimpan kedalam *array* `data_RPM1` dan jika HMI menerima data dari RPM 1, maka data-data yang diterima disimpan kedalam *array* `data_RPM2`.

Selain itu melakukan penerimaan data, ketika data diterima dari RPM tersebut maka dilakukan update waktu koneksi pada setiap RPM untuk mengetahui apakah RPM masih aktif melakukan pengiriman data atau dalam keadaan mati / gagal mengirim data. Kemudian digunakan prosedur `procedure_RF_Condition` untuk menyimpan kondisi setiap RPM1 dalam bentuk string.



Gambar 3. 45 *Flowchart* procedure_RF_ReadData



`procedure_Trigger_Turbine1` digunakan untuk melakukan transmisi sinyal *trigger* berupa data 1 byte untuk menggerakan kincir sebagai pemenuhan fitur kendali cerdas pada riset e-Shrimp. Proses diawali dengan melakukan inisialisasi variabel dan tampilan LCD, kemudian melakukan penulisan data selama 10 detik yang ditunjukan kepada RF24 sebagai *receiver* modul kincir. Kemudian prosedur diakhiri dengan melakukan konfirmasi apakah sinyal *trigger* kekincir berhasil atau gagal dikirim.

Gambar 3. 46 *Flowchart* procedure_Trigger_Turbine1

3.2.3.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul RF24adalah sebagai berikut :

- Pada awal implementasi modul, interaksi antara pengirim dan penerima data tidak dapat berjalan dengan baik, dengan melakukan *troubleshooting*, hal ini disebabkan karena penentuan PIN yang salah, sehingga digunakan referensi SPI Arduino untuk melakukan pemasangan PIN antara modul nRF24L01+ dengan Arduino Mega yang sesuai dengan datasheet

3.2.4 Implementasi GSM pada Modul RPM

3.2.4.1 Tinjauan Spesifikasi dan Desain

Modul GSM digunakan sebagai perangkat pada HMI yang berfungsi untuk melakukan pengiriman data nilai parameter kualitas air kolam yang diukur oleh RPM dalam interval waktu tertentu. Modul GSM yang digunakan sama dengan modul GSM yang digunakan pada riset sebelumnya yaitu SIM900A Mini v3.8.2 .

Pada prinsip kerjanya ketika HMI dinyalakan GSM akan memberikan SMS awal yang menandakan bahwa HMI sudah terhubung dengan nomor handphone yang sudah disetting. Setelah itu dalam interval waktu tertentu akan dilakukan pengiriman data RPM pada e-Shrimp

Data-data yang dikirimkan pada SMS dibentuk dengan format berikut :

```
DD/MM/YYYY hh:mm

RPM1
DO :
Suhu :
pH :
Turbidity :
Salinitas :

RPM2
Suhu :
Turbidity :
```

Salah satu kendala yang dihadapi pada riset sebelumnya adalah adanya kegagalan dalam menerima paket SMS ketika modul GSM diperintah untuk mengirimkan SMS. Oleh karena itu dilakukan modifikasi ulang program GSM secara keseluruhan untuk mengatasi permasalahan ini.

Modifikasi yang dilakukan antara lain, library SIM900.h yang digunakan pada riset sebelumnya tidak digunakan, karena tidak dipublikasikan secara resmi sehingga ditakutkan ada permasalahan pada source code library yang digunakan pada riset sebelumnya. Kemudian dilakukan pengiriman *AT Command* secara manual yang dikirimkan melalui komunikasi *Serial Software* dari Arduino ke Modul GSM.

3.2.4.2 Hasil Implementasi Perangkat Keras

Berikut merupakan konfigurasi PIN pada modul GSM untuk dioperasikan pada HMI :

Pin SIM900A	Arduino Mega	Fungi
VCC5	5V	Power Supply
GND	GND	Ground
5VT	10	5V TX
5VR	11	5V RX

Untuk Pin TX dan RX pada modul GSM, digunakan PIN 5VT dan 5VR, karena Arduino Mega yang digunakan memiliki TTL Logic Level dengan tegangan 5V. Selain itu ground pada modul GSM juga harus dihubungkan dengan Arduino Mega.

Untuk memastikan kondisi kerja modul SIM900A, LED D6 dapat digunakan sebagai indikator apakah GSM siap untuk mengirimkan SMS. LED D6 akan berkedip setiap satu detik bila modul GSM tidak terhubung dengan jaringan seluler, sedangkan jika koneksi ke jaringan seluler berjalan dengan baik, LED akan berkedip setiap tiga detik. LED D5 merupakan indikator panggilan telepon, karena pada riset ini kita tidak menggunakan panggilan telepon untuk komunikasi maka tidak perlu diperhatikan (komunikasi dengan panggilan telepon memerlukan *cost* yang cukup besar, terutama bila dilakukan panggilan ke nomor dengan operator yang berbeda).

Gambar 3. 47 Implementasi Hardware Modul GSM

3.2.4.3 Hasil Implementasi Perangkat Lunak

Pada riset sebelumnya implementasi modul GSM dilakukan dengan menggunakan bantuan header `SIM900.h` dan `sms.h` . Oleh karena itu pada implementasi *software* kami menghindari penggunaan header tersebut dan menggunakan metode pengiriman SMS dengan *AT Command*.

Gambar 3. 48 *Flowchart* procedure_GSM_SETUP

`procedure_GSM_SETUP` dijalankan pada setup program HMI. Prosedur ini berfungsi untuk melakukan pengaturan awal pada GSM sebagai berikut :

- `GSM.begin(2400);` digunakan pada tahap inisialisasi untuk melakukan pengaturan *baud rate* sebesar 2400
- `GSM.println("AT+CMGD=1,4");` digunakan pada tahap inisalisasi untuk menghapus SMS yang masih disimpan pada inbox modul GSM
- `GSM.println("AT+CNMI=2,2,0,0,0");` digunakan agar modul GSM selalu standby untuk menerima SMS baru

- `sprintf` digunakan untuk melakukan pengisian nomor telepon yang akan menjadi target SMS dalam bentuk string (pengiriman SMS dilakukan dengan *serial communication* dengan perintah dalam bentuk string)

`procedure_GSM_SendMessage` dijalankan setelah kita telah melakukan setup GSM. Prosedur ini berfungsi untuk mengirimkan SMS berisi data parameter kualitas air kolam, Prosedur ini pertama kali akan melakukan penerimaan data waktu yang diperoleh dari modul RTC, kemudian dari data waktu yang diperoleh, dibuat sebuah string `SMS_Header` dengan format sebagai berikut :

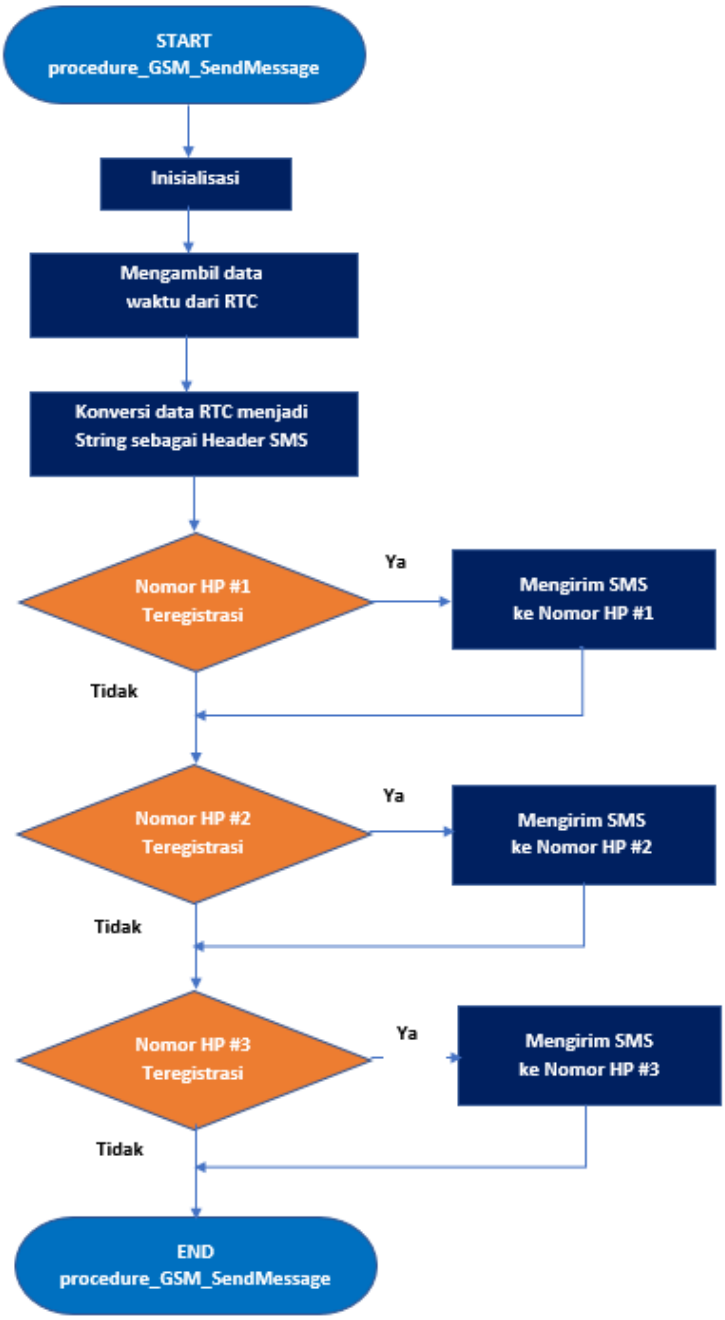
```
DD/MM/YYYY hh:mm
```

Dimana :

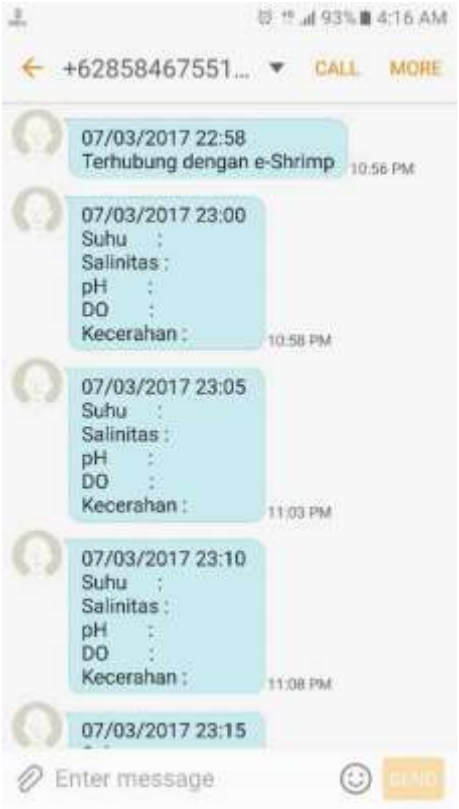
- `DD` = Tanggal
- `MM` = Bulan
- `YYYY` = Tahun
- `hh` = Jam
- `mm` = Menit

Setelah membuat headet tanggal, modul GSM diset menjadi mode SMS dengan menggunakan perintah serial `GSM.println("AT+CMGF=1");` dan memulai operasi pengiriman SMS dengan menggunakan perintah `AT+CMGS`. Pada awal dari perintah pengiriman SMS ini, diperlukan string dari nomor handphone tujuan pengiriman SMS, sehingga perintah serial untuk pengiriman SMS dimodifikasi menjadi `GSM.println("AT+CMGS=\"+628111700373\"\r");` .

Setelah itu mengisi karakter-karakter yang dikirim didalam SMS sudah dapat dilakukan dengan menggunakan `GSM.println` dan `GSM.print` dan untuk mengakhiri isi dari SMS digunakan `GSM.println((char)26);`. Pada implementasi riset ini, modul GSM mampu melakukan SMS *broadcasting* maksimum sampai 3 nomor *Handphone*.



Gambar 3. 49 *Flowchart* procedure_GSM_SendMessage



Gambar 3. 50 Hasil Tampilan SMS

Dari hasil pengubahan pemrograman, telah diperoleh hasil yang lebih lebih baik dimana dilakukan pengiriman SMS mulai dari inisialisasi dan melakukan pengiriman SMS setiap menit ke-5 dan kelipatannya selama 30 menit. Hasilnya adalah HMI telah sukses untuk melakukan pengiriman semua SMS, tanpa ada kegagalan (1 sms inisalisasi dan 7 sms pengiriman data dari 23:00 sampai dengan 23:30),

Oleh karena itu permasalahan yang dihadapi pada riset sebelumnya sudah berhasil diselesaikan.

3.2.4.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul GSM adalah sebagai berikut :

- Pada saat testing modul GSM tidak dapat mengirimkan pesan. Setelah melakukan beberapa *troubleshooting*, didapatkan bahwa modul GSM telah di-set untuk beroperasi di *baud rate* 2400 pada riset sebelumnya, sehingga pada deklarasi *Serial* digunakan nilai baud rate sebesar 2400 (sebelumnya 9600)
- Pada saat menampilkan hasil tampilan SMS yang dikirim oleh modul GSM, header tanggal pengiriman memiliki format yang tidak sesuai harapan (jam dan menit pengiriman SMS terulang dua kali dan tidak ada spasi antara tanggal pengiriman dan jam pengiriman). Solusi dari masalah ini adalah dengan melakukan deklarasi jumlah array yang tepat untuk setiap variabel pada header waktu di prosedur.
- Sempat terjadi masalah dimana GSM tidak terhubung dengan operator (ditandai dengan D6 yang berkedip setiap satu detik), masalah ini diselesaikan dengan mematikan HMI, mengambil SIM card, membersihkannya dan mencaoba pemasangan serta menjalankan HMI kembali.
- Modul GSM tidak dapat menampilkan hasil AT Command atau menerima SMS dari luar, setelah melakukan studi, diperoleh bahwa modul GSM SIM900A memerlukan pin *interrupt* untuk melakukan fungsinya dan konfigurasi awal GSM adalah PIN 9 Arduino sebagai RX yang dihubungkan dengan TX modul GSM dan PIN 10

Arduino sebagai TX yang dihubungkan dengan RX modul GSM. PIN 9 tidak memiliki fitur *interrupt* sehingga menganggu komunikasi GSM dengan mikrokontroller. Solusi yang dilakukan adalah *enabler* LCD di PIN 11 diubah ke PIN 9 karena tidak memerlukan pin khusus. Sedangkan PIN 10 Arduino diubah menjadi RX dan PIN 11 Arduino menjadi TX yang dihubungkan dengan RX Arduino

3.2.5 Implementasi Keypad pada Modul RPM

3.2.5.1 Tinjauan Spesifikasi dan Desain

Keypad merupakan salah satu fitur baru yang dilakukan pada riset ini dibanding riset sebelumnya. Salah satu pertimbangan dibutuhkannnya sebuah *keypad* adalah agar komunikasi antara manusia dengan e-Shrimp dapat berjalan dalam dua arah (riset sebelumnya hanya terjadi komunikasi satu arah dari mesin ke pengguna). Keypad yang digunakan pada riset ini adalah keypad 4x4 yang memiliki karakter A-B-C-D dibanding keypad 3x4 yang karakternya menyerupai karakter yang umum pada telepon.

Fungsi utama yang dilakukan pada *keypad* adalah melakukan navigasi ke menu-menu yang disediakan oleh HMI antara lain :

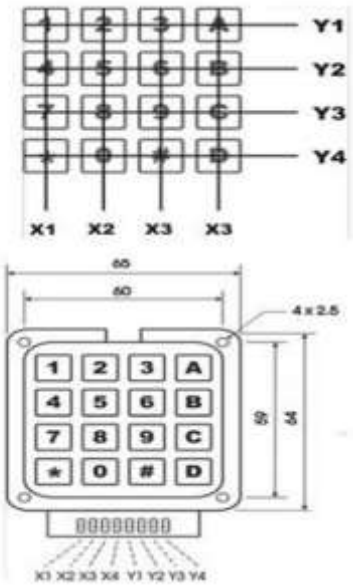
- Menu A : Menu display data dari RPM1
- Menu B : Menu display data dari RPM2
- Menu C : Menu nomor handphone GSM
    - Menampilkan nomor handphone tujuan SMS
    - Mengisi atau mengganti nomor handphone tujuan SMS
    - Menghapus nomor handphone tujuan SMS
- Menu D : Menu Lain-Lain
    - Mengatur Waktu HMI
    - Mengirimkan SMS Manual
    - Mengirim Data ke SD Card Manual

Untuk melakukan perpindahan menu digunakan karakter A-B-C-D yang mewakili setiap menu. Karakter angka digunakan untuk input nomor *handphone* (pengecualian untuk angka 1,2 dan 3 yang digunakan untuk pemilihan *sub-menu*). Karakter * digunakan untuk kembali ke menu sebelumnya / pembatalan dan karakter # digunakan untuk subtitusi tombol enter.

3.2.5.2 Hasil Implementasi Perangkat Keras

Secara hardware, Keypad 4x4 memiliki jumlah PIN sebanyak 8 buah, dimana setiap PIN ini mewakili 4 baris dan 4 kolom pada tombol-tombol di Keypad. Sehingga, pemasangan keypad pada mikrokontroller, akan memerlukan 8 buah pin digital pada mikrokontroller, sehingga pada implementasi ini digunakan pin digital yang tidak mempunyai fungsi khusus. Berikut merupakan konfigurasi PIN yang dilakukan :

PIN Keypad	Arduino Mega	Fungsi
8	22	Kolom 1
7	24	Kolom 2
6	26	Kolom 3
5	28	Kolom 4
4	29	Baris 1
3	27	Baris 2
2	25	Baris 3
1	23	Baris 4



Gambar 3. 51 Jalur Pin Keypad

Gambar 3. 52 Implementasi Hardware Keypad pada HMI

3.2.5.3 Hasil Implementasi Perangkat Lunak

Catatan : Beberapa implementasi keypad juga dilakukan diprosedur program HMI yang lain, namun tidak seluruhnya dilampirkan pada bagian ini (Bagian implementasi software keypad ini hanya mengambil program untuk penggunaan keypad untuk mengatur menu pada HMI)

```
/* MENU BY KEYPAD */
  keypressed = myKeypad.getKey();
  if (keypressed != NO_KEY)
```

```
  {
    if (keypressed == 'A' || keypressed == 'B' || keypressed == 'C' || keypressed == 'D'
)
    {
      if (mode != keypressed)
      {
        lcd.clear();
      }
      mode = keypressed;
    }
  }

  // Mode A : Display RPM1 Data
  // Mode B : Display RPM2 Data
  // Mode C : Edit no HP untuk target GSM
  // Mode D : Request SMS manually
  if (mode == 'A')
  {
    procedure_LCD_SETUP_SensorData1();
  }
  else if (mode == 'B')
  {

    procedure_LCD_SETUP_SensorData2();
  }
  else if (mode == 'C')
  {
    procedure_GSM_Menu();
    if(GSMkey == '1')
    {
      procedure_GSM_PhoneNumber_Display();
    }
    else if(GSMkey == '2')
    {
      procedure_GSM_PhoneNumber_SETUP_Menu();
      procedure_GSM_PhoneNumber_SETUP();
    }
    else if(GSMkey == '3')
    {
      //procedure_GSM_PhoneNumber_Delete();
    }
  }
  else if (mode == 'D')
  {
      procedure_GSM_SendMessage();

  }
```

Program diatas merupakan program yang digunakan untuk melakukan pengaturan menu pada HMI dengan menggunakan input berupa *keypad*, langkah yang dilakukan oleh program ini adalah sebagai berikut :

- `keypressed = myKeypad.getKey();` akan memasukan sebuah nilai / karakter sesuai dengan tombol yang telah diinput ke variabel `keypressed`.
- Program percabangan pertama akan mendeteksi apakah ada input dari keypad, jika iya akan diperiksa apakah input merupakan salah satu dari karakter A, B, C, D.
- Jika karakter yang diinput berbeda dengan kondisi menu, maka terjadi perubahan menu dan diperlukan `lcd.clear();` untuk menghapus semua tampilan terlebih dahulu sebelum penulisan untuk mencegah karakter yang sudah tidak diinginkan masih muncul pada layar LCD.
- Melakukan prosedur dan penampilan LCD sesuai dengan menu yang dipilih

3.2.5.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul Keypad adalah sebagai berikut :

- Pada awal implementasi *keypad*, hasil testing pemberian input dengan menekan tombol akan menghasilkan output yang tidak sama dengan karakter input yang ditekan. Kesalahan yang dilakukan adalah adanya kesalahan pemasangan PIN pada mikrokontroler Arduino Mega, oleh karena itu untuk kedepannya pemasangan PIN *keypad* ke mikrokontroler harus dipastikan sesuai dengan *keymapping* yang dilakukan diawal program.
- Salah satu permasalahan yang terjadi pada riset ini adalah, keluaran data *keypad* yang mengalami *error* ketika diolah lebih lanjut. Setelah melakukan penelitian, diperoleh bahwa keluaran yang dihasilkan merupakan data berbentuk *char,* sehingga dilakukan penyesuaian untuk mengolah data *char* pada program kedepannya.

3.2.6 Implementasi RTC pada Modul HMI

3.2.6.1 Tinjauan Spesifikasi dan Desain

Modul RTC (*Real Time Clock*) merupakan perangkat penting dalam komputer yang berfungsi untuk mendeteksi waktu yang ada saat ini. Pada HMI, modul RTC juga digunakan untuk mendeteksi waktu saat ini, sehingga dapat ditampilkan pada monitor LCD, selain itu waktu yang diperoleh digunakan untuk menentukan waktu atau interval pengiriman SMS, menentukan waktu pengiriman SMS / penulisan data di SD Card dan memberi label waktu di header SMS / nama file dan text untuk file yang dibuat di SD Card.

3.2.6.2 Hasil Implementasi Perangkat Keras

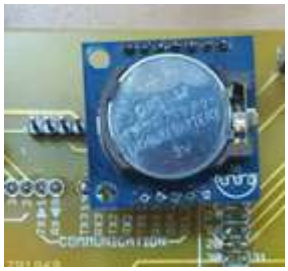
RTC yang digunakan pake riset ini untuk implementasi HMI adalah DS1307 yang sama dengan modul yang digunakan pada riset sebelumnya. DS1307 menggunakan komunikasi serial I2C dalam melakukan pengiriman data. Oleh karena itu pin I2C pada modul RTC harus dihubungkan dengan pin 12C mikrokontroller juga. Berikut merupakan konfigurasi pemasangan pin pada modul RTC DS1307 :

Pin RTC	Arduino Mega	Fungi
BAT	-	Backup Supply Input
VCC	5V	Power Supply
GND	GND	Ground
SDA	20	I2C Data
SCL	21	I2C Clock
DS	-	Temp. Sensor Output (DS18B20)
SQ	-	Square Wave Output

Modul DS1307 disupply dengan menggunakan tegangan sebesar 5V, PIN BAT sebagai *power supply* cadangan tidak perlu digunakan karena RTC selalu disupply dengan

sumber daya 5V yang selalu menyala ketika HMI dijalankan. SDA dan SCL merupakan PIN I2C *data* dan *clock* yang dihubungkan dengan PIN SDA dan SCL pada mikrokontroler Arduino yaitu PIN 20 dan 21.

Modul DS1307 juga dilengkapi dengan sensor DS18B20 yang outputnya dapat diperoleh dari PIN DS. Namun pengukuran temperatur tidak dibutuhkan pada HMI sehingga PIN DS tidak digunakan. *Square Wave Output* , digunakan untuk menghasilkan gelombang kotak dengan frekuensi 1 Hz, namun tidak diperlukan sehingga PIN SQ tidak digunakan.



Gambar 3. 53 Implementasi Hardware Modul RTC pada HMI

3.2.6.3 Hasil Implementasi Perangkat Lunak

Gambar 3. 54 *Flowchart* procedure_RTC

`procedure_RTC` merupakan prosedur utama RTC yang berfungsi untuk membaca data waktu yang dideteksi oleh RTC, serta menampilkannya pada layar LCD :

- `RTC.read(tm)` berfungsi untuk melakukan pembacaan waktu yang dideteksi. Dengan menggunakan header `DS1307RTC.h` dan `Time.h` . Semua variabel waktu (Tahun, Bulan, Tanggal, dll) disimpan didalam sebuah struktur dengan nama

`tm` berikut merupakan data-data pada struktur yang dibaca dan digunakan untuk HMI :

- `tm.Day`
- `tm.Month`
- `tm.Year`
- `tm.Hour`
- `tm. Minute`

- `lcd.setCursor` dan `lcd.print` merupakan perintah yang digunakan untuk menampilkan text pada HMI (akan dibahas pada bagian LCD)

3.2.6.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul RTC adalah sebagai berikut :

- Data yang ditampilkan oleh perangkat RTC ketika pertama kali digunakan menghasilkan data waktu yang tidak tepat, serta penambahan interval waktu yang tidak akurat. Solusi dari permasalahan ini adalah dengan melakukan kaliberasi pada RTC terlebih dahulu dengan menggunakan program "Setime.ino" untuk menyesuaikan waktu pada RTC dengan waktu pada *compiler*.

3.2.7 Implementasi LCD pada Modul HMI

3.2.7.1 Tinjauan Spesifikasi dan Desain

Modul LCD berfungsi sebagai layar monitor pada modul HMI, fungsi utama dari modul ini adalah untuk menampilkan data-data parameter kualitas air kolam yang telah diukur pada RPM . Selain itu bersama dengan keypad, LCD berfungsi untuk membantu interaksi user dengan HMI untuk dapat menampilkan dan melakukan modifikasi nomor *handphone* yang menjadi target pengiriman SMS oleh modul GSM.

Berbeda dengan LCD sebelumnya yang berukuran 16x4, pada riset ini HMI diimplementasikan dengan menggunakan LCD 20x4 untuk meningkatkan jumlah karakter yang bisa ditulis, sehingga diharapkan agar informasi yang diberikan oleh HMI bisa lebih banyak.

3.2.7.2 Hasil Implementasi Perangkat Keras

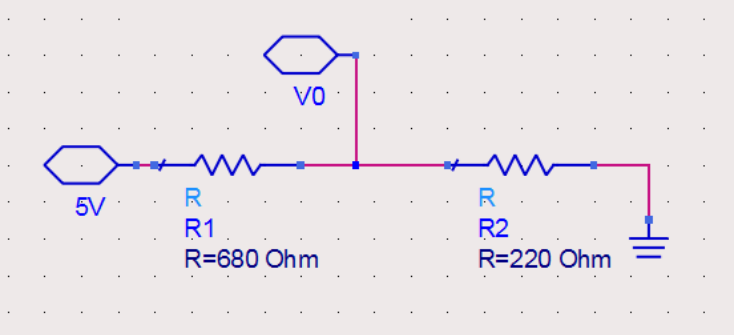
Konfigurasi PIN untuk melakukan implementasi LCD sebagai berikut :

PIN LCD 20x4	PIN Arduino Mega	Fungsi
VSS	GND	Ground
VDD	VSS	Tegangan Sumber untuk Logic
V0	*Output Voltage Divider	Tegangan Operasi LCD
RS	PIN 12	Register Select High : Character Data

RW	GND	Read/Write Low : Write Data to LCD
E	PIN 9	Chip Enable
D0	-	Data Pin : Hanya menggunakan D4-D7 untuk beroperasi dalam 4-bit mode (Jika ingin menggunakan operasi 8-bit mode bisa menggunakan D0-D7)
D1	-	
D2	-	
D3	-	
D4	PIN 5	
D5	PIN 4	
D6	PIN 3	
D7	PIN 2	
A	5V	Backlight Anode (Tegangan sumber)
K	GND	Backlight Cathode (Ground)

V0 merupakan tegangan yang mempengaruhi kontras layer, untuk mendapatkan nilai V0 yang paling baik maka dilakukan percobaan sebagai berikut :

- Dilakukan pemasangan potensiometer 3 kΩ untuk mendapatkan tegangan V0. Hasil untuk memperoleh kontras yang baik adalah sebesar 1,25V
- Rasio R1 dan R2 ditentukan menjadi 3:1 karena tegangan supply bernilai 5V
- Digunakan resistor dengan nilai yang mendekati perbandingan yaitu 680Ω dan 220Ω

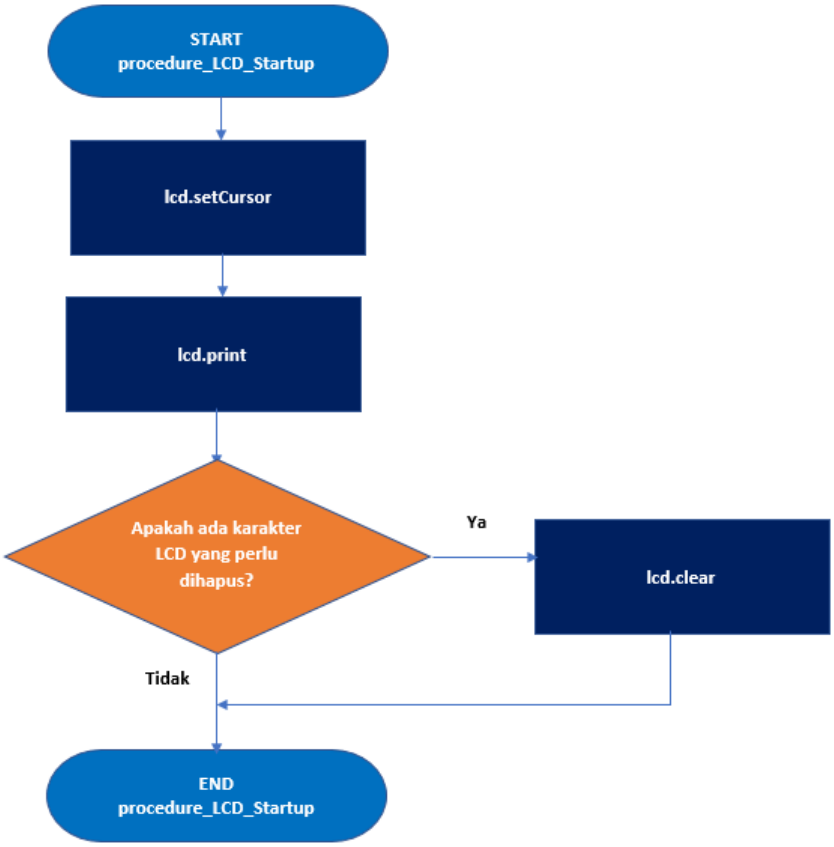


Gambar 3. 55 Rangkaian Pembagi Tegangan untuk Mengatur Kontras Layar

Gambar 3. 56 Implemenasi Hardware Modul LCD

3.2.7.3 Hasil Implementasi Perangkat Lunak

Catatan : Beberapa implementasi LCD juga dilakukan diprosedur program HMI yang lain, namun tidak dilampirkan pada bagian ini (Bagian implementasi software LCD hanya mengambil flowchart program untuk menampilkan display data yang diperoleh RPM)



Gambar 3. 57 *Flowchart* untuk Implementasi LCD

Dalam melakukan implementasi software untuk perangkat LCD pada HMI, ada fungsi utama yang digunakan antara lain :

- `lcd.setCursor(x,y)` berfungsi untuk mengatur posisi penulisan karakter pada LCD, `x` merupakan kolom (range 0-3) dan `y` merupakan baris (range 0-19)
- `lcd.print` berfungsi untuk melakukan pengisian karakter di variabel `z` pada layer di lokasi yang sudah diatur oleh `lcd.setCursor` format penulisan yang digunakan antara lain :
  - `lcd.print("…")` : menulis karakter yang diapit tanda petik dua (“)
  - `lcd.print(z)` : menulis karakter yang tersimpan di variabel z
  - `lcd.print(i[j],k);` : menulis karakter yang tersimpan di array `i` dengan alamat `j` dan jika `i[j]` merupakan bentuk float bisa ditambahkan `,k` untuk membatasi jumlah angka dibelakang koma sebanyak `k`
- `lcd.clear` berfungsi untuk menghapus seluruh tampilan karakter yang ada pada LCD, sering digunakan ketika kita mengubah tampilan secara signifikan (ketika kita mengubah layar teks, beberapa karakter yang tidak diganti akan tetap ada)

3.2.7.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul Keypad adalah sebagai berikut :

- Salah satu permasalahan yang sering terjadi adalah LCD *blinking* dan tampilan LCD tidak sesuai dengan hasil yang diharapkan. Solusi yang umum dilakukan adalah memastikan nilai delay yang diberikan, memastikan program dan prosedur sudah berjalan dengan benar dan memastikan penggunaan `lcd.clear()` untuk menghapus karakter yang tidak ditimpa

3.2.8 Implementasi SD Card pada Modul HMI

3.2.8.1 Tinjauan Spesifikasi dan Desain

Modul SD Card berfungsi sebagai tempat penyimpanan data-data yang telah diterima oleh modul RPM, penyimpanan data dapat dilakukan setiap interval waktu tertentu yang telah ditetapkan. Pada riset ini, digunakan pemrograman yang sama dengan riset sebelumnya dengan menggunakan SPI bersamaan dengan modul RF24, sehingga dilakukan sharing PIN SPI untuk modul SD Card dengan RF24.

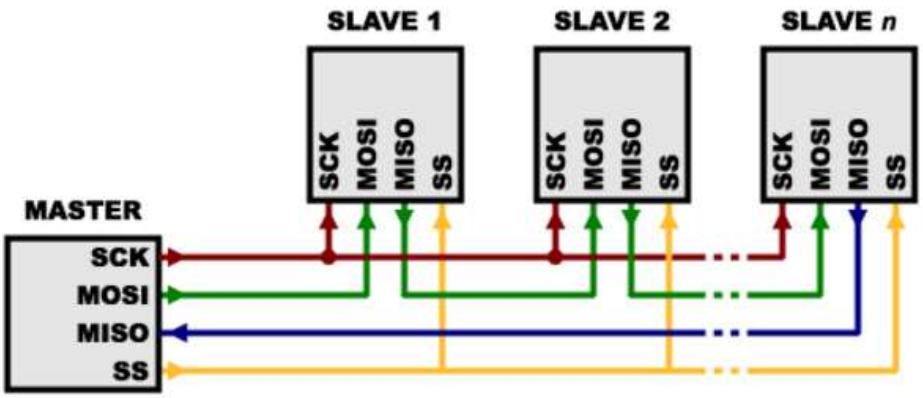
3.2.8.2 Hasil Implementasi Perangkat Keras

Seperti yang telah dijelaskan sebelumnya, SD Card module bekerja dengan menggunakan SPI sehingga memiliki konfigurasi yang identik dengan RF24 kecuali untuk bagian *Chip Select,* berikut merupakan konfigurasi PIN untuk melakukan implementasi LCD modul SD Card :

SD Card Module	PIN Arduino Mega	Fungsi

3V3	-	Sumber Tegangan
5V	5V	Sumber Tegangan
CS	49	Chip Select
MOSI	51	Master Out Slave In
SCK	52	Serial Clock
MISO	50	Master In Slave Out
GND	GND	Ground

Bersamaan dengan modul RF24, modul SD Card disini berpersan sebagai *SPI Slave* dengan Arduino Mega sebagai *SPI Master*, dalam melakukan komunikasi, *SPI Master* hanya bisa terhubung dengan salah satu dari *SPI Slave* saja, oleh karena itu digunakan *Chip Select / Chip Select Not* yang berfungsi untuk menunjuk *Slave* yang berkomunikasi dengan master. *Slave* akan berkomunikasi dengan *Master* ketika pin CS/CSN bernilai LOW dan pin CS/CSN dari *Slave* lain harus HIGH.



Gambar 3. 58 Ilustrasi Sistem Master-Slave pada Komunikasi SPI

Dalam beroperasi SD Card memerlukan tegangan sebesar 3,3V, namun modul SD Card sudah dilengkapi dengan IC AMS1117-3.3 yang berperan untuk melakukan DC *step down* dari tegangan 5V yang kemudian dikonversi menjadi tegangan 3,3V.

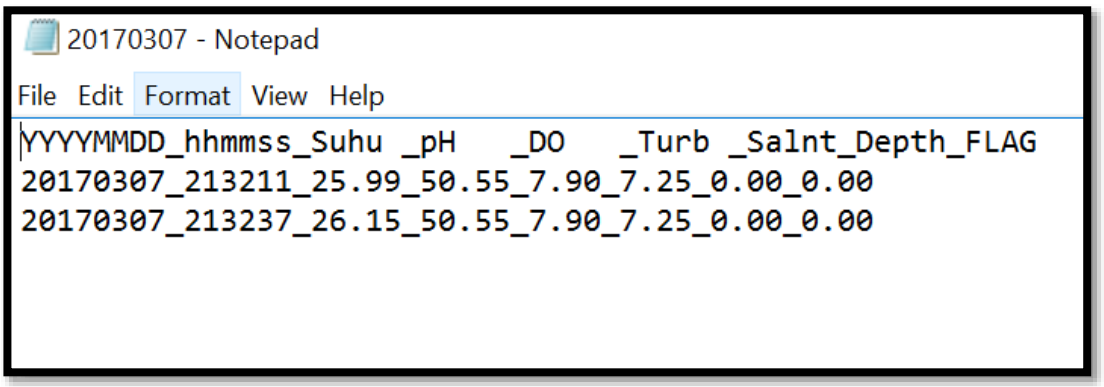
Gambar 3. 59 Implementasi Hardware SD Card

3.2.8.3 Hasil Implementasi Perangkat Lunak

Berikut merupakan *flowchart* dari prosedur untuk melakukan *data logging* dengan menggunakan modul SD Card :

Gambar 3. 60 *Flowchart* procedure_SDCard_DataLogging

Berikut merupakan hasil contoh implementasi dari penulisan file text yang dilakukan oleh HMI dan disimpan kedalam SD Card :



Gambar 3. 61 Hasil File Text Ditulis pada SD Card

3.2.8.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul Keypad adalah sebagai berikut :

- Pembacaan micro SD Card tidak dapat berjalan dengan baik, masalah ini diselesaikan dengan menggunakan micro SD Card yang lain, sehingga diasumsikan bahwa micro SD Card sebelumnya (yang berasal dari riset sebelumnya) dalam keadaan rusak.

3.2.9 Implementasi LED pada Modul HMI

3.2.9.1 Tinjauan Spesifikasi dan Desain

Modul LED berfungsi sebagai indikator untuk kondisi kualitas air tambak dan indikator untuk kondisi modul-modul lain yang ada di HMI. Pada riset ini warna LED yang digunakan hanya dua buah yaitu LED berwarna merah (untuk menandakan kondisi kualitas air tambak / modul memiliki masalah) dan LED berwarna hijau (untuk menandakan kondisi kualitas air tambak aman dan modul berjalan dengan baik).

3.2.9.2 Hasil Implementasi Perangkat Keras

Untuk melakukan implementasi LED, langkah pertama yang dilakukan adalah menentukan besar resistor. Penentuan nilai resistor dapat dilakukan dengan menggunakan hukum ohm sebagai berikut :

$$R = \frac{V}{I}$$

Dimana nilai dari tegangan (V) merupakan tegangan sumber (Vcc) yang dikurangi oleh tegangan jatuh LED (Vled), sehingga persamaannya menjadi :

$$R = \frac{Vcc - Vled}{I}$$

Arus yang mengalir dalam PIN Arduino diinginkan sebesar 5 mA untuk menghasilkan cahaya yang terang, namun tidak merusak LED itu sendiri. Sedangkan LED memiliki tegangan jatuh bervariasi di nilai 1.8V (Merah) sampai dengan 3.3V (biru), untuk mendapatkan nilai yang akurat maka dilakukan pengukuran terhadap dua buah

- Setelah melakukan pengukuran dengan multimeter didapatkan bahwa LED Merah yang digunakan memiliki tegangan jatuh kurang lebih 1,8V
- Setelah melakukan pengukuran dengan multimeter didapatkan bahwa LED Hijau yang digunakan memiliki tegangan jatuh kurang lebih 2,5V

Gambar 3. 62 Pengukuran Tegangan Jatuh pada LED

Oleh karena itu, dilakukan perhitungan nilai resistor untuk LED merah sebagai berikut :

$$R = \frac{5 - 1{,}8}{5m} = \frac{3{,}2}{5m} = 640\Omega$$

Namun, sangat sulit untuk mencari resistor seharga 640Ω yang komersial, untuk membuat agar arus yang mengalir tetap diatas 5mA maka digunakan harga resistor dibawah 640Ω yang paling dekat yaitu 560Ω.

Sedangkan untuk, resistor LED hijau dilakukan perhitungan sebagai berikut :

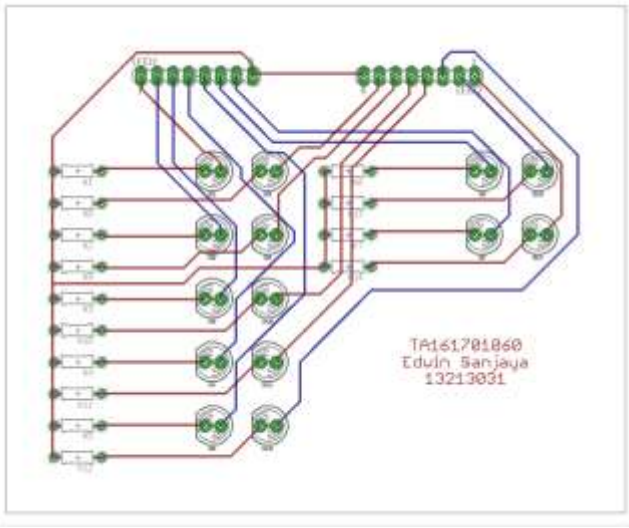
$$R = \frac{5 - 2{,}5}{5m} = \frac{2{,}5}{5m} = 500\Omega$$

Sama dengan kasus sebelumnya, sangat sulit untuk mencari resistor seharga 500Ω yang komersial, untuk membuat agar arus yang mengalir tetap diatas 5mA maka digunakan harga resistor dibawah 500Ω yang paling dekat yaitu 470Ω

Berikut merupakan hasil pengukuran arus setelah menggunakan resistor yang telah dipertimbangkan :

Gambar 3. 63 Pengukuran Arus yang Mengalir pada LED

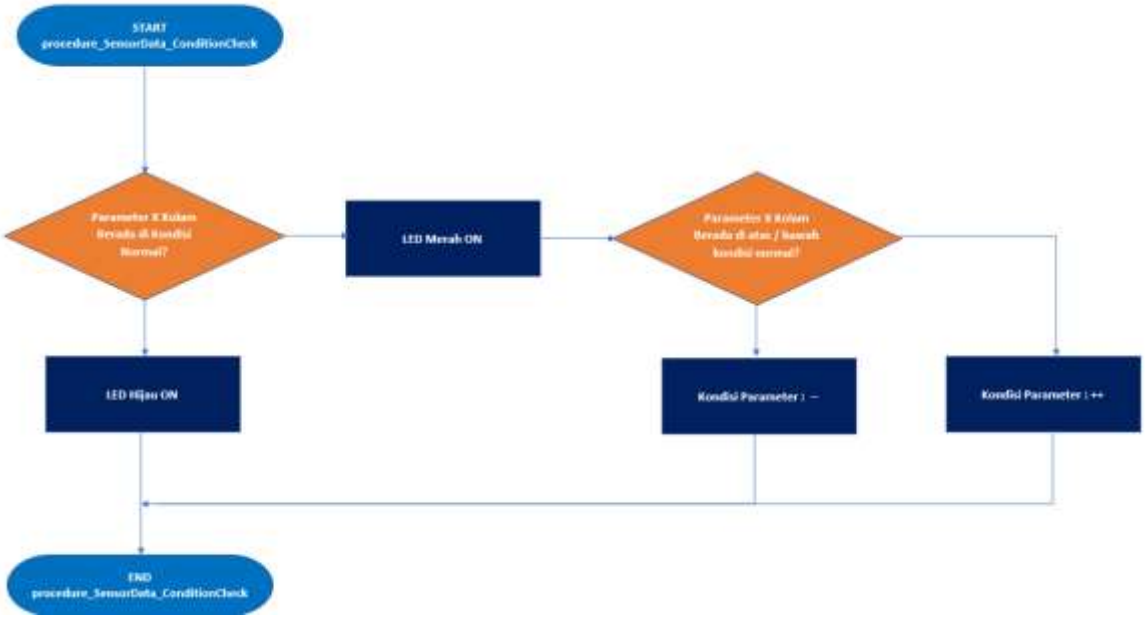
Dari hasil pengukuran arus, dapat diperoleh bahwa arus yang mengalir di LED sudah sesuai dengan harapan yaitu memiliki arus yang memiliki nilai disekitar 5mA. Sehingga untuk implementasi LED yang dilakukan hanya dilakukan pergantian pada nilai resistor untuk LED hijau. Hasil implementasi board LED :



Gambar 3. 64 Desain Board PCB LED pada Modul HMI

Gambar 3. 65 Implementasi Hardware LED pada Modul HMI

3.2.9.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 66 *Flowchart* procedure_SensorData_ConditionCheck

Dasar dari implementasi LED adalah dengan menggunakan digital PIN yang akan memberikan sinyal LOW dan HIGH. Pertama-data akan dilakukan pengecekan terhadap suatu nilai parameter air kolam untuk menentukan LED warna apa yang menyala. LED berwarna hijau menyala sebagai indikator bahwa parameter air kolam berada dikondisi normal, sedangkan LED warna merah menyala sebagai indikator bahwa parameter air kolam berada dikondisi tidak normal

LED akan menyala ketika diberi sinyal HIGH dan akan mati ketika diberi sinyal LOW. Pertama-tama dilakukan penentuan PIN-PIN yang akan digunakan sebagai sinyal dengan menggunakan `pinMode`. Setelah itu kita bisa melakukan penentuan sinyal yang diberikan dengan menggunakan `digitalWrite` . Nilai 1 diberikan untuk membuat PIN memberikan sinyal HIGH dan Nilai 0 untuk sinyal LOW.

Selain itu pada prosedur ini dilakukan juga deklarasi data string untuk menyatakan simbol kondisi parameter air kolam agar data bisa diolah lebih lanjut. Keadaan normal akan diwakili string “OK”, keadaan dibawah parameter normal diwakili string “--“ dan keadaan diatas parameter normal diwakili string “++”

3.2.9.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul Relay adalah sebagai berikut :

- LED yang masih digunakan berasal dari riset sebelumnya dimana hanya terdapat 5 indikator untuk 1 modul RPM. Untuk kedepannya akan dibuat modul LED yang baru dengan 2 indikator untuk RPM ke-2.

3.2.10 Implementasi Relay untuk Speaker pada Modul HMI

3.2.10.1 Tinjauan Spesifikasi dan Desain

Modul relay berfungsi untuk melakukan switching dengan menggunakan sinyal dari mikrokontroller untuk menyalakan / mematikan speaker pada HMI. Relay perlu digunakan karena memiliki kapabilitas yang lebih tinggi untuk mengatur switching pada listrik dengan tegangan dan arus yang cukup besar. Speaker yang digunakan memiliki tegangan operasi sebesar 12V dengan arus yang dapat ditarik sampai dengan 300mA.

Karena pada umumnya speaker hanya menyala ketika kondisi kolam berada diluar parameter yang normal, maka speaker lebih sering berada dikondisi mati. Oleh karena itu, digunakan hubungan *Normally Open* pada implementasi relay dimana hubungan sumber tegangan dengan speaker dalam keadaan open ketika HMI mati. Speaker hanya akan menyala ketika e-Shrimp dinyalakan dan relay diberi input LOW (input LOW diberikan ketika ada parameter kualitas air tambak berada di kondisi tidak normal)

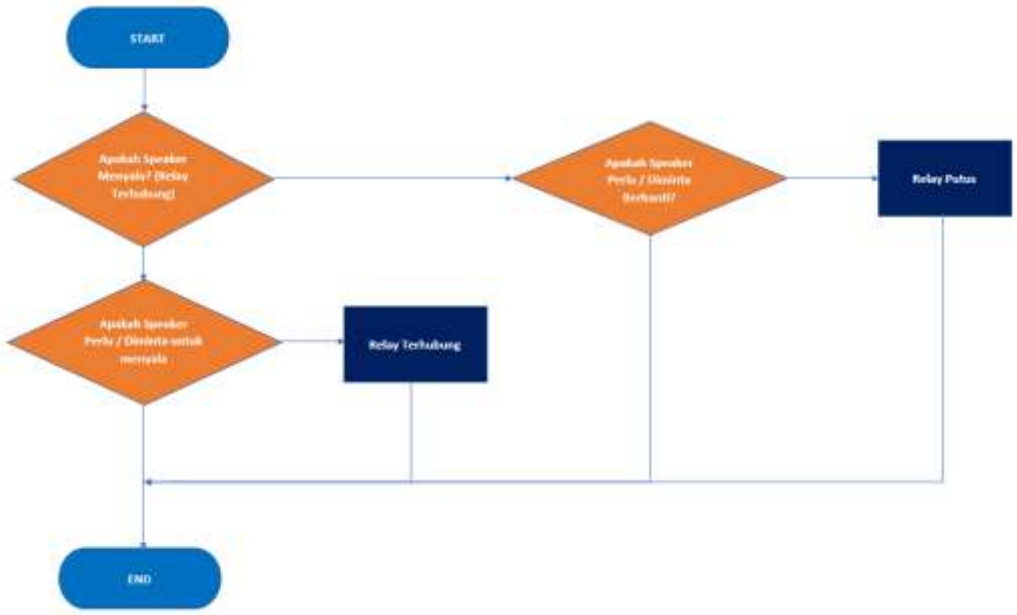
3.2.10.2 Hasil Implementasi Perangkat Keras

Berikut merupakan konfigurasi yang dilakukan untuk melakukan implementasi 2-Relay Module pada modul HMI :

Modul Relay	Dihubungkan Dengan	Keterangan
NC1	-	Normally Closed Relay 1
COM1	V+ Baterai	Common Relay 1
NO1	V+ Speaker	Normally Open Relay 1
NC2	-	Normally Closed Relay 2
COM2	-	Common Relay 2
NO2	-	Normally Open Relay 2
VCC	5 V	Sumber Tegangan Opto Coupler
GND	GND	Ground
IN1	PIN 18	Sinyal Switching Relay 1
IN2	PIN 19	Sinyal Switching Relay 2

Gambar 3. 67 Implementasi Hardware Relay

3.2.10.3 Hasil Implementasi Perangkat Lunak



Gambar 3. 68 *Flowchart* Implementasi Software Relay

Secara sederhana, program untuk menentukan apakah relay aktif atau putus ditentukan pada bagian *loop* pada program. Implementasi software pada modul relay relatif sama dengan modul LED dimana akan digunakan PIN digital yang menjadi sinyal untuk mengatur apakah relay akan terhubung / putus. Sinyal LOW akan membuat relay dengan koneksi *Normally Open* menjadi terhubung sehingga aliran listrik dari sumber tegangan dapat terhubung ke komponen. Sedangkan sinyal HIGH akan membuat relay menjadi terputus

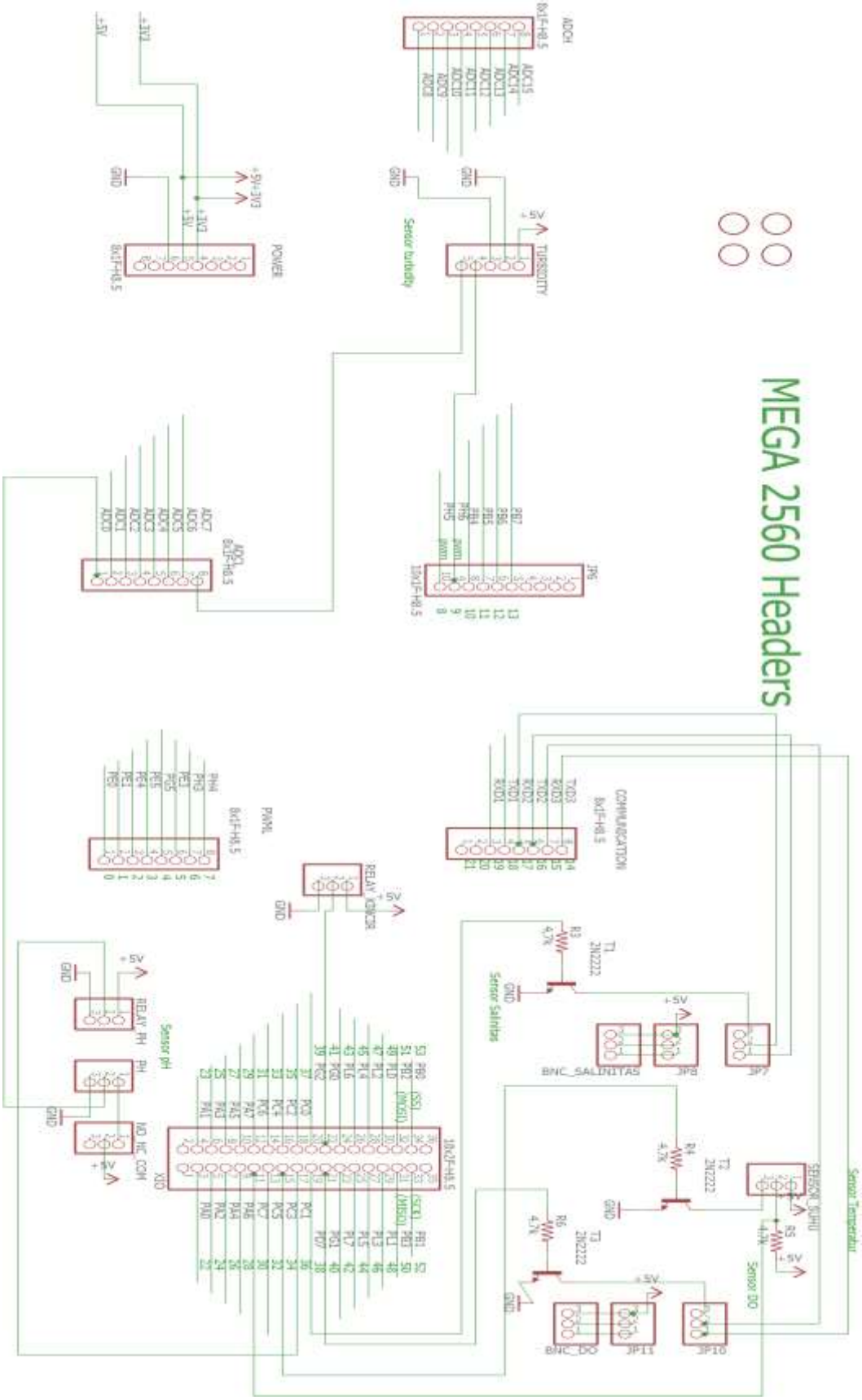
3.2.10.4 Permasalahan dan Solusi Implementasi

Permasalahan dalam riset yang dihadapi ketika melakukan implementasi modul Relay adalah sebagai berikut :

- Penggunaan relay sebelumnya mengalami masalah dimana relay tidak dapat memberikan sumber tegangan ke komponen, hal ini ternyata disebabkan karena logika sinyal ke relay bernilai HIGH (untuk membuat relay terhubung, diperlukan logika sinyal LOW).
- Penggunaan relay masih disubtitusikan oleh tegangan sumber baterai 9V untuk menggantikan adapter 12V dan lampu LED untuk menggantikan speaker. Penggunaan speaker dan adapter akan dilakukan kedepannya.

4 Lampiran

- Skematik untuk Modul RPM



- Software Modul RPM 1

```
/*
 * Project        : Firmware Prototype for Remote Monitoring Platform
 *                  Sistem Monitoring Kualitas Air Tambak Udang Vaname
 * Version        : 0.1
 * Date Created   : January 13, 2017
 * Date Modified  : April 6, 2017
 * Author         : Daniel Anugrah Wiranata
 * Company        : Department of Electrical Engineering
 *                  School of Electrical Engineering and Informatics
 *                  Bandung Institute of Technology (ITB)
 * Summary        :
 */

#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>


//---------------------------------------- Pin Configuration ------------
--------------------------------
// Temperature
const int DS18S20_Pin = 30;               // DS18S20 Signal pin on
digital pin 30
const int control_BJT_temperature = 34;   // Temperature probe BJT
controller/base on digital pin 34

// LCD
/*const int Button_Pin = 22;              // LCD pushbutton pin on
digital pin 22
const int LCDbacklight_Pin = 9;           // LCD Backlight controller
using Enable pin connected to digital pin 9
*/

// pH
#define RELAY_ON 0
#define RELAY_OFF 1
#define RELAY_1  36
#define Sensor_pH_Pin 0                   // pH meter Analog output to
Arduino Analog Input 0
#define Offset 0.15                       // deviation compensate
#define samplingInterval 20
#define printInterval 800
#define ArrayLength  40                   // times of collection

// Salinity
const int control_BJT_salinity = 38;      // Salinity probe BJT
controller/base on digital pin 38
// DO
const int control_BJT_DO = 40;            // Salinity probe BJT
controller/base on digital pin 40
// Turbidiy
```

```
#define Sensor_turbidity_Pin 7             // Output turbidity sensor to
Analog Input 7
int pd = 9;                                // photodiode to digital pin 9

// Transceiver
const int pinCE = 48; //This pin is used to set the nRF24 to standby (0)
or active mode (1)
const int pinCSN = 53; //This pin is used to tell the nRF24 whether the
SPI communication is a command or message to send out

//======================================== Constant Variables
==========================================
// Temperature
OneWire ds(DS18S20_Pin);                    // Onewire library used by
DS1820 on digital pin 30
int Sensor_interval = 360;                  // in seconds
int Sensor_active_interval = 300;           // in seconds

//======================================== Constant Variables End
==========================================


//======================================== Changing Variables
==========================================

//---------------------------------------- MISC Variable ---------------
---------------------------------
unsigned long Detik_Now = 0;             // Variable used to show how
many seconds passed since the last reset
unsigned long Detik_Previous = 0;        // Used for rollover prevention
(Latest time)

//---------------------------------------- Transmitter Variable ---------
---------------------------------
RF24 radio(pinCE, pinCSN); // Create your nRF24 object or wireless SPI
connection
#define WHICH_NODE 1     // must be a number from 1 - 6 identifying the
PTX node
const uint64_t wAddress[] = {0x7878787878LL, 0xB3B4B5B6F1LL,
0xB3B4B5B6CDLL, 0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL};
const uint64_t PTXpipe = wAddress[ WHICH_NODE - 1 ];   // Pulls the
address from the above array for this node's pipe
byte counter = 1; //used to count the packets sent


int sent_ready_flag = 0;         // Flag used to tell if the receiver is
ready to sent another packet to the HMI
int Sent_interval = 1800;        // Time interval between every packet
transmission, in seconds

//---------------------------------------- Temperature Variable ---
---------------------------------------
// Temperature
float temperature = 0;                   // Temperature measurement
result
float temperature_measurement = 0;       // Temperature measurement
result used to compensate DO measurement
int temperature_loop_count = 0;

//---------------------------------------- DO Variable ------------
------------------------------
```

```
String input_DO_string = "";                    // String to hold incoming
data from the PC
String sensor_DO_string = "";                   // String to hold the data
from the Atlas Scientific probe
boolean input_DO_string_complete = false;       // Check if received all
the data from the PC
boolean sensor_DO_string_complete = false;      // Check if received all
the data from the Atlas Scientific probe
float DO_measurement;                           // DO probe measurement
result
int DO_loop_count = 0;

//------------------------------------------- Salinity Variable ------
-----------------------------------
String input_Salinity_string = "";                // String to hold
incoming data from the PC
String sensor_Salinity_string = "";               // String to hold the
data from the Atlas Scientific probe
boolean input_Salinity_string_complete = false;   // Check if received
all the data from the PC
boolean sensor_Salinity_string_complete = false;  // Check if received
all the data from the Atlas Scientific probe
float f_ec;
float f_sal;
float f_salinity;
int salinity_loop_count = 0;

//------------------------------------------- pH Variable ------------
-----------------------------
unsigned long int Average_Value_pH;             // Store the average
value of the pH probe result
int buf_pH[10],temp_pH;                         // Buffer and temporary
variable used to sort and averaging pH measurement result
float pH_measurement = 0;
int pH_loop_count = 0;


int pHArray[ArrayLength];   //Store the average value of the sensor
feedback
int pHArrayIndex=0;
//---------------------------------------- Turbidity Variable ---------
-----------------------------
unsigned long int Average_Value_turbidity;      // Store the average
value of the turbidity probe result
int buf_turbidity[10],temp_turbidity;           // Buffer and temporary
variable used to sort and averaging turbidity measurement result
float KekeruhanVoltage;

//------------------------------------------------ Flag ---------------
-----------------------------
// Salinity flag
boolean salinity_read_ready_flag = false;    // Flag to toggle the
salinity probe
// pH flag
boolean ph_read_ready_flag = false;          // Flag to toggle the pH
probe
// temperature flag
boolean temperature_read_ready_flag = false; // Flag to toggle the
temperature probe
// DO flag
```

```
boolean DO_read_ready_flag = false;          // Flag to toggle the DO
probe
boolean DO_data_flag = false;                // Flag to tell if DO probe
ready to sent data to RMP

int DO_detik_count = 0;                 // Inisialisasi DO_detik_count
int temperature_detik_count = 0;        // Inisialisasi
temperature_detik_count
int pH_detik_count = 0;                 // Inisialisasi pH_detik_count
int salinity_detik_count = 0;           // Inisialisasi
salinity_detik_count
int turbidity_detik_count = 0;          // Inisialisasi
turbidity_detik_count
//====================================== Changing Variables End
==========================================


//============================================ SETUP
================================================
void setup(void)
{
  analogReference(DEFAULT);              // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  // DO
  Serial.begin(9600);                       // Serial communication baud
rate set to 9600
  Serial3.begin(9600);
  input_DO_string.reserve(10);              // String reserve for input
command from PC set to 10
  sensor_DO_string.reserve(30);             // String reserve for DO probe
result set to 30

  // Salinity
  Serial.begin(9600);
  Serial2.begin(38400);                      // set baud rate for software
serial port_2 to 9600
  input_Salinity_string.reserve(10);        // set aside some bytes for
receiving data from the PC
  sensor_Salinity_string.reserve(30);       // set aside some bytes for
receiving data from Atlas Scientific product

  //pH
  pinMode(13,OUTPUT);

  // BJT as switch for Temperature sensor
  pinMode(control_BJT_temperature, OUTPUT);     // Initialize pin BJT
control to output (as a switch)
  // Relay as switch for pH sensor
  pinMode(RELAY_1, OUTPUT);
  // BJT as switch for Salinity
  pinMode(control_BJT_salinity, OUTPUT);        // Initialize pin BJT
control to output (as a switch)
  // BJT as switch for DO
  pinMode(control_BJT_DO, OUTPUT);              // Initialize pin BJT
control to output (as a switch)
  // Turbidity
  pinMode(pd,OUTPUT);

//============================================ BJT Initialization
================================================
```

```
  digitalWrite(control_BJT_DO, LOW);              // Initialize DO Sensor
at OFF Condition
  digitalWrite(control_BJT_temperature, LOW);     // Initialize
temperature Sensor at OFF Condition
  digitalWrite(RELAY_1, RELAY_OFF);               // Initialize pH Sensor
at OFF Condition
  digitalWrite(control_BJT_salinity, LOW);        // Initialize Salinity
Sensor at OFF Condition
  digitalWrite(pd,HIGH);                          // supply 5 volts to
photodiode

  // Transceiver
  procedure_RF_Setup();


  DO_detik_count = 0 * Sensor_interval;           // Setting up DO
measurement schedule
  temperature_detik_count = 1 * Sensor_interval;  // Setting up
Temperature measurement schedule
  pH_detik_count = 2 * Sensor_interval;           // Setting up pH
measurement schedule
  salinity_detik_count = 3 * Sensor_interval;     // Setting up Salinity
measurement schedule
  turbidity_detik_count = 4 * Sensor_interval;    // Setting up
turbidity measurement schedule

  Serial.println("Ready");                        // Test the serial
monitor
}
//========================================== SETUP END
==================================================

//=================================== Additional Procedure and
Function ================================

 // ---------------------------------------- DO ------------------------
------------------

void serialEvent() {                                        // if the
hardware serial port_0 receives a char
  input_DO_string = Serial.readStringUntil(13);             // read the
string until we see a <CR>
  input_DO_string_complete = true;                          // set the
flag used to tell if we have received a completed string from the PC
}

void serialEvent3() {                                       // if the
hardware serial port_3 receives a char
  sensor_DO_string = Serial3.readStringUntil(13);           // read the
string until we see a <CR>
  sensor_DO_string_complete = true;                         // set the
flag used to tell if we have received a completed string from the PC
}

// ---------------------------------------- Salinity -------------------
-----------------------
 void serialEvent1() {                                      // if the
hardware serial port_0 receives a char
  input_Salinity_string = Serial.readStringUntil(13);       // read the
string until we see a <CR>
```

```
  input_Salinity_string_complete = true;                  // set the
flag used to tell if we have received a completed string from the PC
}

void serialEvent2() {                                     // if the
hardware serial port_3 receives a char
  sensor_Salinity_string = Serial2.readStringUntil(13);   // read the
string until we see a <CR>
  sensor_Salinity_string_complete = true;                 // set the
flag used to tell if we have received a completed string from the PC
}
//---------------------------------------- temperature ------------------
---------------------
// this algorithm is acquired from dallas semiconductor DS1820 library -
- Copyright (c) 2010 bildr community

float getTemp()                                           // Returns the
temperature from one DS18S20 in DEG Celsius
{
  byte probe_data[12];
  byte addr[8];
  if ( !ds.search(addr)) {
      ds.reset_search();                                  // No more sensors
on chain, reset search
      return -9;
  }
  if ( OneWire::crc8( addr, 7) != addr[7]) {              // 8 bit Dallas
Semiconductor Cyclic Redundancy Check
      Serial.println("CRC is not valid!");
      return -9;
  }
  if ( addr[0] != 0x10 && addr[0] != 0x28) {              // Device signature
recognization from Onewire Livrary, used for multi probe measurement
      Serial.print("Device is not recognized");
      return -9;
  }
  ds.reset();
  ds.select(addr);
  ds.write(0x44,1);                                       // start conversion,
with parasite power on at the end
  byte present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);                                         // Read from the
Scratchpad
  for (int i = 0; i < 9; i++) {
    probe_data[i] = ds.read();
  }
  ds.reset_search();
  byte MSB = probe_data[1];
  byte LSB = probe_data[0];
  float Temp_Read = ((MSB << 8) | LSB);                   // 2's complement
  float Temperature_Sum = Temp_Read / 16;
  return Temperature_Sum;                                 // Temperature
Value: Temperature_Sum
}

// ------------------------------------------------ pH ----------------
---------------------------
// This code is an adaptation of the sample test code given from dfrobot
site for dfrobot pH probe -- Copyright (c) 2013 dfrobot
```

```
double averagearray(int* arr, int number){
  int i;
  int max,min;
  double avg;
  long amount=0;
  if(number<=0){
    Serial.println("Error number for the array to avraging!/n");
    return 0;
  }
  if(number<5){   //less than 5, calculated directly statistics
    for(i=0;i<number;i++){
      amount+=arr[i];
    }
    avg = amount/number;
    return avg;
  }else{
    if(arr[0]<arr[1]){
      min = arr[0];max=arr[1];
    }
    else{
      min=arr[1];max=arr[0];
    }
    for(i=2;i<number;i++){
      if(arr[i]<min){
        amount+=min;        //arr<min
        min=arr[i];
      }else {
        if(arr[i]>max){
          amount+=max;    //arr>max
          max=arr[i];
        }else{
          amount+=arr[i]; //min<=arr<=max
        }
      }//if
    }//for
    avg = (double)amount/(number-2);
  }//if
  return avg;
}

//---------------------------------------------- turbidity ----------
---------------------------------------

float getTurbidity()
{
  Average_Value_turbidity = analogRead(Sensor_turbidity_Pin);
  float turbidityValue = Average_Value_turbidity;
  turbidityValue = turbidityValue * 4.88758553;                 // Nilai
4.88758553 didapat dari kalibrasi, nilai tersebut dapat berubah.

  return turbidityValue;                                        //
Turbidity Value: turbidityValue
}

//---------------------------------------------- Salinity -----------
---------------------------------------
float print_EC_data()
{
// this function will pars the string
  char sensor_Salinity_string_array[30];
// we make a char array
```

```
  char *EC;
// char pointer used in string parsing
  char *TDS;
// char pointer used in string parsing
  char *SAL;
// char pointer used in string parsing
  char *GRAV;
// char pointer used in string parsing

  float f_ec;
// used to hold a floating point number that is the EC
  float f_sal;
// used to hold a floating point number that is the Sal

  sensor_Salinity_string.toCharArray(sensor_Salinity_string_array, 30);
// convert the string to a char array

  EC = strtok(sensor_Salinity_string_array, ",");
// let's pars the array at each comma
  TDS = strtok(NULL, ",");
// let's pars the array at each comma
  SAL = strtok(NULL, ",");
// let's pars the array at each comma
  GRAV = strtok(NULL, ",");
// let's pars the array at each comma

  f_ec=atof(EC);                                          // uncomment this
line to convert the char to a float
  f_sal=atof(SAL);

  return f_sal;
}

//------------------------------------------------ Transceiver --------
----------------------------------------
void procedure_RF_Setup()
{
  Serial.begin(9600);   //start serial to communicate process
  radio.begin();            //Start the nRF24 module
  radio.setPALevel(RF24_PA_LOW);  // "short range setting" - increase if
you want more range AND have a good power supply
  radio.setDataRate(RF24_250KBPS);
  radio.setChannel(22);          // the higher channels tend to be more
"open"
  radio.openReadingPipe(0,PTXpipe);  //open reading or receive pipe
  radio.stopListening(); //go into transmit mode
  radio.openWritingPipe(PTXpipe);        //open writing or transmit pipe
}

//================================ Additional Procedure and Function End
================================

//********************************************* Main Loop
*********************************************
void loop(void)
{
  Detik_Now = (millis()/1000);
  // ============================================== Sensor Measurement
=========================================
```

```
  // ------------------------------------------------ DO Measurement ---
-------------------------------------
  if (Detik_Now % Sent_interval == DO_detik_count)
  {
    DO_read_ready_flag = true;
    digitalWrite(control_BJT_DO, HIGH);                              // Turn
DO Sensor ON
  }

  if (input_DO_string_complete == true) {                            // if a
string from the PC has been received in its entirety
    Serial3.print(input_DO_string);                                  // send
that string to the Atlas Scientific product
    Serial3.print('\r');                                             // add a
<CR> to the end of the string
    input_DO_string = "";                                            // clear
the string
    input_DO_string_complete = false;                                // reset
the flag used to tell if we have received a completed string from the PC
  }

  if ((DO_read_ready_flag == true)&&(DO_loop_count < 500))
  {
    if (sensor_DO_string_complete == true)                           // if a
string from the Atlas Scientific product has been received in its
entirety
    {
        Serial.println(sensor_DO_string);                            // send
that string to the PC's serial monitor
        if (isdigit(sensor_DO_string[0]))                            // if
the first character in the string is a digit
        {
            DO_measurement = sensor_DO_string.toFloat();             //
convert the string to a floating point number so it can be evaluated by
the Arduino
        }
    }
    sensor_DO_string = "";                                           // clear
the string:
    sensor_DO_string_complete = false;                               // reset
the flag used to tell if we have received a completed string from the
Atlas Scientific product
    DO_loop_count++;
  }


  if (DO_loop_count == 500)                    // At 41 loop, the power to
the sensor are cut out
  {
    digitalWrite(control_BJT_DO, LOW);         // Turn DO Sensor OFF
    DO_loop_count = 0;
    DO_read_ready_flag = false;

  }

  //---------------------------------------- Temperature Measurement --
----------------------------------//
  if (Detik_Now % Sent_interval == temperature_detik_count)
  {
    temperature_read_ready_flag = true;
```

```
    digitalWrite(control_BJT_temperature, HIGH);
// Turn Temperature Sensor ON
  }
  if ((temperature_read_ready_flag == true)&&(temperature_loop_count <
500))
  {
      temperature = getTemp();
// Call the function to measure temperature
      temperature_loop_count++;
  }

  if(temperature_loop_count == 500)
  {
    digitalWrite(control_BJT_temperature, LOW);
// Turn DO Sensor OFF
    temperature_loop_count = 0;
    temperature_read_ready_flag = false;
  }

  //------------------------------------- pH measurement -------------
----------------------//
  if (Detik_Now % Sent_interval == pH_detik_count)
  {
    ph_read_ready_flag = true;
    digitalWrite(RELAY_1, RELAY_ON);
// Turn DO Sensor OFF
  }

  if ((ph_read_ready_flag == true)&&(temperature_loop_count < 500))
  {
      float pHValue;
      float voltage;

          pHArray[pHArrayIndex++]=analogRead(Sensor_pH_Pin);
          if(pHArrayIndex==ArrayLength)
          {
            pHArrayIndex=0;
          }
          voltage = averagearray(pHArray, ArrayLength)*5.0/1024;
          pHValue = 3.5*voltage+Offset;
          pH_measurement = pHValue;
      pH_loop_count++;
 }

  if (pH_loop_count == 500)
  {
    digitalWrite(RELAY_1, RELAY_OFF);
    pH_loop_count = 0;
    ph_read_ready_flag = false;
  }

  //------------------------------------- Turbidity Measurement ------
--------------------------------//
    float KekeruhanVoltage = getTurbidity();          // Call
getTurbidity Function

  // ---------------------------------------------- Salinity
Measurement ----------------------------------------
  if (Detik_Now % Sent_interval == salinity_detik_count)
  {
    salinity_read_ready_flag = true;
```

```
    digitalWrite(control_BJT_salinity, HIGH);               // Turn
Salinity Sensor ON
  }

  if (input_Salinity_string_complete == true)               // if a
string from the PC has been received in its entirety
  {
    Serial2.print(input_Salinity_string);                   // send that
string to the Atlas Scientific product
    Serial2.print('\r');                                    // add a
<CR> to the end of the string
    input_Salinity_string = "";                             // clear the
string
    input_Salinity_string_complete = false;                 // reset the
flag used to tell if we have received a completed string from the PC
  }

  if ((salinity_read_ready_flag == true)&&(salinity_loop_count < 500))
  {
  if (sensor_Salinity_string_complete == true) {            // if a
string from the Atlas Scientific product has been received in its
entirety
    if (isdigit(sensor_Salinity_string[0]) == false) {      // if the
first character in the string is a digit
      Serial.println(sensor_Salinity_string);               // send that
string to the PC's serial monitor
    }
    else                                                    // if the
first character in the string is NOT a digit
    {
      f_salinity = print_EC_data();                                //
then call this function
    }
    sensor_Salinity_string = "";                            // clear the
string
    sensor_Salinity_string_complete = false;                // reset the
flag used to tell if we have received a completed string from the Atlas
Scientific product
  }
  salinity_loop_count++;
  }

  if (salinity_loop_count == 500)                           // At 60
loop, the power to the sensor are cut out
  {
    digitalWrite(control_BJT_salinity, LOW);                // Turn
Salinity Sensor OFF
    salinity_loop_count = 0;
    salinity_read_ready_flag = false;
  }

  //============================================ Sensor Measurement End
==========================================//


  //============================================ Array of Float
========================================
  // Declaring the array of float
  float Packet_Total[5];

  // Assign a value to the array
```

```
  Packet_Total[0] = DO_measurement;
  Packet_Total[1] = temperature;
  Packet_Total[2] = pH_measurement;
  Packet_Total[3] = KekeruhanVoltage;
  Packet_Total[4] = f_salinity;
  //Print the value to serial monitor for debugging
  for (int i=0; i < 5; i++)
    {
          Serial.println("Read Data");
          Serial.println(Packet_Total[i]);
    }
  Serial.println();
  //========================================== Array of Float END
==========================================

  //========================================== Transceiver
==============================================
  radio.write( Packet_Total, sizeof(Packet_Total) );        //open
writing or transmit pipe

  if (!radio.write( Packet_Total, sizeof(Packet_Total) ))
  {  //if the write fails let the user know over serial monitor
    Serial.println("Send Failed");
  }
  delay(1000);

  //======================================== flag management
==============================================
  if (Detik_Now % Sent_interval == 1)                        // To
ensure only 1 transmission made in 1 second interval
  {
    sent_ready_flag = 0;
  }

  if(Detik_Now % Sent_interval == (temperature_detik_count +
Sensor_active_interval))
  {
    temperature_read_ready_flag = false;
    //salinity_read_ready_flag = false;
    digitalWrite(control_BJT_temperature, LOW);
    //digitalWrite(control_BJT_salinity, LOW);
  }

  if(Detik_Now % Sent_interval == (DO_detik_count +
Sensor_active_interval))
  {
    DO_read_ready_flag = false;
    digitalWrite(control_BJT_DO, LOW);
  }

  if(Detik_Now % Sent_interval == (pH_detik_count +
Sensor_active_interval))
  {
    ph_read_ready_flag = false;
    digitalWrite(RELAY_1, RELAY_OFF);
  }

  if(Detik_Now % Sent_interval == (salinity_detik_count +
Sensor_active_interval))
  {
    salinity_read_ready_flag = false;
```

```
    digitalWrite(control_BJT_salinity, LOW);
  }
}
//********************************************** Main Loop End
**********************************************
```

- Software Modul RPM 2

```
/*
 * Project        : Firmware Prototype for Remote Monitoring Platform
 *                  Sistem Monitoring Kualitas Air Tambak Udang Vaname
 * Version        : 0.1
 * Date Created   : January 13, 2017
 * Date Modified  : April 6, 2017
 * Author         : Daniel Anugrah Wiranata
 * Company        : Department of Electrical Engineering
 *                  School of Electrical Engineering and Informatics
 *                  Bandung Institute of Technology (ITB)
 * Summary        :
 */

#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
#include <SPI.h> //Call SPI library so you can communicate with the
nRF24L01+
#include <nRF24L01.h> //nRF2401 library found at
https://github.com/tmrh20/RF24/
#include <RF24.h>

//-------------------------------------- Pin Configuration ------------
--------------------------------
// Temperature
const int DS18S20_Pin = 7;               // DS18S20 Signal pin on
digital pin 7
const int control_BJT_temperature = 6;   // Temperature probe BJT
controller/base on digital pin 6

// Turbidity
#define Sensor_turbidity_Pin 0            // Output turbidity sensor to
Analog Input 7
int pd = 4;                               // photodiode to digital pin 4

//Transceiver
const int pinCE = 9; //This pin is used to set the nRF24 to standby (0)
or active mode (1)
const int pinCSN = 10; //This pin is used to tell the nRF24 whether the
SPI communication is a command or message to send out

//====================================== Constant Variables
==========================================
// Temperature
OneWire ds(DS18S20_Pin);                    // Onewire library used by
DS1820 on digital pin 30
int Sensor_interval = 360;                  // in seconds
int Sensor_active_interval = 300;           // in seconds
```

```
//======================================== Constant Variables End
==========================================

//========================================= Changing Variables
==========================================

//---------------------------------------- MISC Variable ---------------
---------------------------------
unsigned long Detik_Now = 0;            // Variable used to show how
many seconds passed since the last reset
unsigned long Detik_Previous = 0;       // Used for rollover prevention
(Latest time)

//---------------------------------------- Transmitter Variable ---------
---------------------------------
int sent_ready_flag = 0;        // Flag used to tell if the receiver is
ready to sent another packet to the HMI
int Sent_interval = 1800;       // Time interval between every packet
transmission, in seconds

//---------------------------------------------- Temperature Variable ---
---------------------------------------
// Temperature
float temperature = 0;                  // Temperature measurement
result
float temperature_measurement = 0;      // Temperature measurement
result used to compensate DO measurement
int temperature_loop_count = 0;

//---------------------------------------- Turbidity Variable ---------
---------------------------------
unsigned long int Average_Value_turbidity;      // Store the average
value of the turbidity probe result
int buf_turbidity[10],temp_turbidity;           // Buffer and temporary
variable used to sort and averaging turbidity measurement result
float KekeruhanVoltage;

//------------------------------------------------ Flag ---------------
---------------------------------
// temperature flag
boolean temperature_read_ready_flag = false; // Flag to toggle the
temperature probe

int temperature_detik_count = 0;        // Inisialisasi
temperature_detik_count
int turbidity_detik_count = 0;          // Inisialisasi
turbidity_detik_count

//------------------------------------------------ Transceiver --------
---------------------------------------
RF24 radio(pinCE, pinCSN); // Create your nRF24 object or wireless SPI
connection
#define WHICH_NODE 2     // must be a number from 1 - 6 identifying the
PTX node
const uint64_t wAddress[] = {0x7878787878LL, 0xB3B4B5B6F1LL,
0xB3B4B5B6CDLL, 0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL};
const uint64_t PTXpipe = wAddress[ WHICH_NODE - 1 ];   // Pulls the
address from the above array for this node's pipe
byte counter = 1; //used to count the packets sent
```

```
//====================================== Changing Variables End
=========================================


//============================================ SETUP
================================================
void setup(void)
{
  analogReference(DEFAULT);              // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)

  // BJT as switch for Temperature sensor
  pinMode(control_BJT_temperature, OUTPUT);     // Initialize pin BJT
control to output (as a switch)
  // Turbidity
  pinMode(pd,OUTPUT);

//============================================ BJT Initialization
================================================
  digitalWrite(control_BJT_temperature, LOW);    // Initialize
temperature Sensor at OFF Condition
  digitalWrite(pd,HIGH);                         // supply 5 volts to
photodiode

  temperature_detik_count = 0 * Sensor_interval;  // Setting up
Temperature measurement schedule
  turbidity_detik_count = 1 * Sensor_interval;    // Setting up
turbidity measurement schedule

  Serial.println("Ready");                        // Test the serial
monitor

//============================================ Transceiver
================================================
  procedure_RF_Setup();

}
//========================================== SETUP END
================================================

//==================================== Additional Procedure and
Function ================================


//------------------------------------ temperature -----------------
---------------------
// this algorithm is acquired from dallas semiconductor DS1820 library -
- Copyright (c) 2010 bildr community

float getTemp()                                       // Returns the
temperature from one DS18S20 in DEG Celsius
{
  byte probe_data[12];
  byte addr[8];
  if ( !ds.search(addr)) {
      ds.reset_search();                              // No more sensors
on chain, reset search
      return -9;
  }
  if ( OneWire::crc8( addr, 7) != addr[7]) {          // 8 bit Dallas
Semiconductor Cyclic Redundancy Check
```

```
      Serial.println("CRC is not valid!");
      return -9;
  }
  if ( addr[0] != 0x10 && addr[0] != 0x28) {        // Device signature
recognization from Onewire Livrary, used for multi probe measurement
      Serial.print("Device is not recognized");
      return -9;
  }
  ds.reset();
  ds.select(addr);
  ds.write(0x44,1);                                 // start conversion,
with parasite power on at the end
  byte present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);                                   // Read from the
Scratchpad
  for (int i = 0; i < 9; i++) {
    probe_data[i] = ds.read();
  }
  ds.reset_search();
  byte MSB = probe_data[1];
  byte LSB = probe_data[0];
  float Temp_Read = ((MSB << 8) | LSB);             // 2's complement
  float Temperature_Sum = Temp_Read / 16;
  return Temperature_Sum;                           // Temperature
Value: Temperature_Sum
}

//------------------------------------------------ turbidity ----------
---------------------------------------

float getTurbidity()
{
  Average_Value_turbidity = analogRead(Sensor_turbidity_Pin);
  float turbidityValue = Average_Value_turbidity;
  turbidityValue = turbidityValue * 4.88758553;                 // Nilai
4.88758553 didapat dari kalibrasi, nilai tersebut dapat berubah.
  return turbidityValue;                                        //
Turbidity Value: turbidityValue
}

//------------------------------------------------ Transceiver --------
----------------------------------------
void procedure_RF_Setup()
{
  Serial.begin(9600);   //start serial to communicate process
  radio.begin();            //Start the nRF24 module
  radio.setPALevel(RF24_PA_LOW);  // "short range setting" - increase if
you want more range AND have a good power supply
  radio.setDataRate(RF24_250KBPS);
  radio.setChannel(22);          // the higher channels tend to be more
"open"
  radio.openReadingPipe(0,PTXpipe);  //open reading or receive pipe
  radio.stopListening(); //go into transmit mode
  radio.openWritingPipe(PTXpipe);        //open writing or transmit pipe
}


//================================ Additional Procedure and Function End
================================
```

```
//****************************************** Main Loop
******************************************
void loop(void)
{
  Detik_Now = (millis()/1000);
  // ============================================== Sensor Measurement
==========================================
  //--------------------------------------- Temperature Measurement --
----------------------------------//
  if (Detik_Now % Sent_interval == temperature_detik_count)
  {
    temperature_read_ready_flag = true;
    digitalWrite(control_BJT_temperature, HIGH);
// Turn Temperature Sensor ON
  }
  if ((temperature_read_ready_flag == true)&&(temperature_loop_count <
500))
  {
      temperature = getTemp();
// Call the function to measure temperature
      temperature_loop_count++;
  }

  if(temperature_loop_count == 500)
  {
    digitalWrite(control_BJT_temperature, LOW);
// Turn DO Sensor OFF
    temperature_loop_count = 0;
    temperature_read_ready_flag = false;
  }
  //------------------------------------- Turbidity Measurement ------
--------------------------------//
  float KekeruhanVoltage = getTurbidity();          // Call getTurbidity
Function


  //============================================== Sensor Measurement End
==========================================//


  //============================================== Array of Float
==========================================
  // Declaring the array of float
  float Packet_Total[2];

  // Assign a value to the array
  Packet_Total[0] = temperature;
  Packet_Total[1] = KekeruhanVoltage;

  //Print the value to serial monitor for debugging
  for (int i=0; i < 2; i++)
    {
          Serial.println("Read Data");
          Serial.println(Packet_Total[i]);
    }
  Serial.println();
  //============================================== Array of Float END
==========================================

  //============================================== Transceiver
==========================================
```

```
  radio.write( Packet_Total, sizeof(Packet_Total) );              //open
writing or transmit pipe

  if (!radio.write( Packet_Total, sizeof(Packet_Total) ))
  {  //if the write fails let the user know over serial monitor
    Serial.println("Send Failed");
  }
  delay(1000);

  //================================================ flag management
================================================
  if (Detik_Now % Sent_interval == 1)                              // To
ensure only 1 transmission made in 1 second interval
  {
    sent_ready_flag = 0;
  }

  if(Detik_Now % Sent_interval == (temperature_detik_count +
Sensor_active_interval))
  {
    temperature_read_ready_flag = false;
    digitalWrite(control_BJT_temperature, LOW);
  }
}

//************************************************ Main Loop End
***********************************************
```

- Software Sensor Temperatur pada Modul RPM

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into pin 2 on the Arduino
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices
// (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

void setup(void)
{
  // start serial port
  Serial.begin(9600);
  Serial.println("Dallas Temperature IC Control Library Demo");

  // Start up the library
  sensors.begin();
}


void loop(void)
```

```
{
  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  Serial.print(" Requesting temperatures...");
  sensors.requestTemperatures(); // Send the command to get temperatures
  Serial.println("DONE");

  Serial.print("Temperature is: ");
  Serial.print(sensors.getTempCByIndex(0)); // Why "byIndex"?
    // You can have more than one IC on the same bus.
    // 0 refers to the first IC on the wire
    delay(1000);
}
```

- Implementasi Software Sensor pH pada Modul PRM

```
#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
//#include <NewPing.h>

// pH
#define RELAY_ON 0
#define RELAY_OFF 1
#define RELAY_1  36
#define Sensor_pH_Pin 0                    // pH meter Analog output to
Arduino Analog Input 0
#define Offset 0.15                        //deviation compensate
#define samplingInterval 20
#define printInterval 800
#define ArrayLength  40                    //times of collection

//------------------------------------------- pH Variable ------------
------------------------------
unsigned long int Average_Value_pH;              // Store the average
value of the pH probe result
int buf_pH[10],temp_pH;                          // Buffer and temporary
variable used to sort and averaging pH measurement result
float pH_measurement = 0;
int pH_loop_count = 0;


int pHArray[ArrayLength];   //Store the average value of the sensor
feedback
int pHArrayIndex=0;

void setup(void)
{
  Serial.begin(9600);
  analogReference(DEFAULT);              // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  //pH
  pinMode(13,OUTPUT);
  // Relay as switch for pH sensor
  pinMode(RELAY_1, OUTPUT);
```

```
  digitalWrite(RELAY_1, RELAY_ON);              // Initialize pH Sensor
at OFF Condition
}

// ------------------------------------------------- pH ----------------
--------------------------
// This code is an adaptation of the sample test code given from dfrobot
site for dfrobot pH probe -- Copyright (c) 2013 dfrobot

double averagearray(int* arr, int number)
{
  int i;
  int max,min;
  double avg;
  long amount=0;
  if(number<=0){
    Serial.println("Error number for the array to avraging!/n");
    return 0;
  }
  if(number<5){   //less than 5, calculated directly statistics
    for(i=0;i<number;i++){
      amount+=arr[i];
    }
    avg = amount/number;
    return avg;
  }else{
    if(arr[0]<arr[1]){
      min = arr[0];max=arr[1];
    }
    else{
      min=arr[1];max=arr[0];
    }
    for(i=2;i<number;i++){
      if(arr[i]<min){
        amount+=min;        //arr<min
        min=arr[i];
      }else {
        if(arr[i]>max){
          amount+=max;    //arr>max
          max=arr[i];
        }else{
          amount+=arr[i]; //min<=arr<=max
        }
      }//if
    }//for
    avg = (double)amount/(number-2);
  }//if
  return avg;
}

void loop(void)
{
  float pHValue;
  float voltage;
  pHArray[pHArrayIndex++]=analogRead(Sensor_pH_Pin);
          if(pHArrayIndex==ArrayLength)
          {
            pHArrayIndex=0;
          }
          voltage = averagearray(pHArray, ArrayLength)*5.0/1024;
          pHValue = 3.5*voltage+Offset;
```

```
            pH_measurement = pHValue;
            Serial.print("    pH value: ");
            Serial.println(pHValue,2);
}
```

- Implementasi Software Sensor Salinitas pada Modul RPM

```
#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
//#include <NewPing.h>

// Salinity
const int control_BJT_salinity = 38;      // Salinity probe BJT
controller/base on digital pin 38

//------------------------------------------- Salinity Variable ------
-----------------------------------
String input_Salinity_string = "";                // String to hold
incoming data from the PC
String sensor_Salinity_string = "";               // String to hold the
data from the Atlas Scientific probe
boolean input_Salinity_string_complete = false;   // Check if received
all the data from the PC
boolean sensor_Salinity_string_complete = false;  // Check if received
all the data from the Atlas Scientific probe
//float Salinity_measurement;                     // DO probe
measurement result
//float Salinity_measurement_last;                // DO probe
measurement result, before the current result
//int Salinity_receive_data_count = 0;
float f_ec;
int salinity_loop_count = 0;

void setup(void)
{
   analogReference(DEFAULT);              // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
   // Salinity
  Serial.begin(9600);
  Serial2.begin(38400);                    // set baud rate for software
serial port_2 to 9600
  input_Salinity_string.reserve(10);       // set aside some bytes for
receiving data from the PC
  sensor_Salinity_string.reserve(30);      // set aside some bytes for
receiving data from Atlas Scientific product

  // BJT as switch for Salinity
  pinMode(control_BJT_salinity, OUTPUT);         // Initialize pin BJT
control to output (as a switch)

  digitalWrite(control_BJT_salinity, HIGH);       // Initialize Salinity
Sensor at OFF Condition

}
```

```
// ---------------------------------------- Salinity ------------------
-----------------------
 void serialEvent1() {                                          // if the
hardware serial port_0 receives a char
  input_Salinity_string = Serial.readStringUntil(13);          // read the
string until we see a <CR>
  input_Salinity_string_complete = true;                       // set the
flag used to tell if we have received a completed string from the PC
}

void serialEvent2() {                                          // if the
hardware serial port_3 receives a char
  sensor_Salinity_string = Serial2.readStringUntil(13);        // read the
string until we see a <CR>
  sensor_Salinity_string_complete = true;                      // set the
flag used to tell if we have received a completed string from the PC
}

//------------------------------------------------ Salinity -----------
-----------------------------------
void print_EC_data(void)
{
// this function will pars the string
  char sensor_Salinity_string_array[30];
// we make a char array

  char *EC;
// char pointer used in string parsing
  char *TDS;
// char pointer used in string parsing
  char *SAL;
// char pointer used in string parsing
  char *GRAV;
// char pointer used in string parsing

  float f_ec;
// used to hold a floating point number that is the EC

  sensor_Salinity_string.toCharArray(sensor_Salinity_string_array, 30);
// convert the string to a char array

  EC = strtok(sensor_Salinity_string_array, ",");
// let's pars the array at each comma
  TDS = strtok(NULL, ",");
// let's pars the array at each comma
  SAL = strtok(NULL, ",");
// let's pars the array at each comma
  GRAV = strtok(NULL, ",");
// let's pars the array at each comma


  Serial.print("EC:");                                  //we now print
each value we parsed separately
  Serial.println(EC);                                  //this is the EC
value

  Serial.print("TDS:");                                 //we now print
each value we parsed separately
  Serial.println(TDS);                                 //this is the TDS
value
```

```
  Serial.print("SAL:");                                    //we now print
each value we parsed separately
  Serial.println(SAL);                                    //this is the
salinity value

  Serial.print("GRAV:");                                   //we now print
each value we parsed separately
  Serial.println(GRAV);                                   //this is the
specific gravity
  Serial.println();                                       //this just makes
the output easer to read


  //f_ec=atof(EC);                                          // uncomment
this line to convert the char to a float
  //Serial.println(f_ec);
}

void loop(void)
{
  if (input_Salinity_string_complete == true)               // if a
string from the PC has been received in its entirety
  {
    Serial2.print(input_Salinity_string);                   // send that
string to the Atlas Scientific product
    Serial2.print('\r');                                    // add a
<CR> to the end of the string
    input_Salinity_string = "";                             // clear the
string
    input_Salinity_string_complete = false;                 // reset the
flag used to tell if we have received a completed string from the PC
  }

  if (sensor_Salinity_string_complete == true) {            // if a
string from the Atlas Scientific product has been received in its
entirety
    if (isdigit(sensor_Salinity_string[0]) == false) {      // if the
first character in the string is a digit
      Serial.println(sensor_Salinity_string);               // send that
string to the PC's serial monitor
    }
    else                                                    // if the
first character in the string is NOT a digit
    {
      print_EC_data();                                      // then call
this function
    }
    sensor_Salinity_string = "";                            // clear the
string
    sensor_Salinity_string_complete = false;                // reset the
flag used to tell if we have received a completed string from the Atlas
Scientific product
  }
}
```

- Implementasi Software untuk Sensor DO pada Modul RPM

```
#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
//#include <NewPing.h>

// DO
const int control_BJT_DO = 40;            // Salinity probe BJT
controller/base on digital pin 40

//------------------------------------------- DO Variable ------------
-----------------------------
//char DO_Temp_Compensation_variable[13];     // Latest temperature
measurement result that used for DO compensation
//int DO_Temp_Compensation_Counter = 0;       // Count number of loop
since the last temperatur compensation for DO probe

String input_DO_string = "";                  // String to hold incoming
data from the PC
String sensor_DO_string = "";                 // String to hold the data
from the Atlas Scientific probe
boolean input_DO_string_complete = false;     // Check if received all
the data from the PC
boolean sensor_DO_string_complete = false;    // Check if received all
the data from the Atlas Scientific probe
float DO_measurement;                         // DO probe measurement
result
// float DO_measurement_last;                 // DO probe measurement
result, before the current result
int DO_loop_count = 0;
//int DO_receive_data_count = 0;

void setup(void)
{
  analogReference(DEFAULT);              // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  // DO
  Serial.begin(9600);                     // Serial communication baud
rate set to 9600
  Serial3.begin(9600);
  input_DO_string.reserve(10);            // String reserve for input
command from PC set to 10
  sensor_DO_string.reserve(30);           // String reserve for DO probe
result set to 30

  // BJT as switch for DO
  pinMode(control_BJT_DO, OUTPUT);              // Initialize pin BJT
control to output (as a switch)

  digitalWrite(control_BJT_DO, HIGH);             // Initialize DO
Sensor at OFF Condition

}

 // ---------------------------------------- DO ----------------------
------------------

void serialEvent() {                                        // if the
hardware serial port_0 receives a char
  input_DO_string = Serial.readStringUntil(13);             // read the
string until we see a <CR>
```

```
  input_DO_string_complete = true;                                  // set the
flag used to tell if we have received a completed string from the PC
}

void serialEvent3() {                                               // if the
hardware serial port_3 receives a char
  sensor_DO_string = Serial3.readStringUntil(13);                   // read the
string until we see a <CR>
  sensor_DO_string_complete = true;                                 // set the
flag used to tell if we have received a completed string from the PC
}

void loop(void)
{
   if (input_DO_string_complete == true) {                           // if a
string from the PC has been received in its entirety
    Serial3.print(input_DO_string);                                 // send
that string to the Atlas Scientific product
    Serial3.print('\r');                                            // add a
<CR> to the end of the string
    input_DO_string = "";                                           // clear
the string
    input_DO_string_complete = false;                               // reset
the flag used to tell if we have received a completed string from the PC
  }

  if (sensor_DO_string_complete == true)                           // if a
string from the Atlas Scientific product has been received in its
entirety
    {
        Serial.println(sensor_DO_string);                           // send
that string to the PC's serial monitor
        if (isdigit(sensor_DO_string[0]))                           // if
the first character in the string is a digit
        {
            DO_measurement = sensor_DO_string.toFloat();            //
convert the string to a floating point number so it can be evaluated by
the Arduino
        }
    }
    sensor_DO_string = "";                                          // clear
the string:
    sensor_DO_string_complete = false;                              // reset
the flag used to tell if we have received a completed string from the
Atlas Scientific product
    Serial.print("    DO value: ");
    Serial.println(DO_measurement,2);
}
```

- Implementasi Software untuk Sensor Kekeruhan Air pada Modul RPM

```
#include <OneWire.h>
#include <LiquidCrystal.h>
#include <VirtualWire.h>
#include <string.h>
//#include <NewPing.h>

// Turbidiy
```

```
#define Sensor_turbidity_Pin 7            // Output turbidity sensor to
Analog Input 7
int pd = 9;                               // photodiode to digital pin 9

//---------------------------------------- Turbidity Variable ---------
-------------------------------
unsigned long int Average_Value_turbidity;      // Store the average
value of the turbidity probe result
int buf_turbidity[10],temp_turbidity;           // Buffer and temporary
variable used to sort and averaging turbidity measurement result
float KekeruhanVoltage;

void setup(void)
{
  Serial.begin(9600);
  analogReference(DEFAULT);             // the default analog reference
of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  // Turbidity
  pinMode(pd,OUTPUT);
  digitalWrite(pd,HIGH);                         // supply 5 volts to
photodiode
}

//--------------------------------------------------- turbidity ----------
---------------------------------------

float getTurbidity()
{
  Average_Value_turbidity = analogRead(Sensor_turbidity_Pin);
  float turbidityValue = Average_Value_turbidity;
  turbidityValue = turbidityValue * 4.88758553;                  // Nilai
4.88758553 didapat dari kalibrasi, nilai tersebut dapat berubah.
  return turbidityValue;                                         //
Turbidity Value: turbidityValue
}


void loop(void)
{
  float KekeruhanVoltage = getTurbidity();          // Call getTurbidity
Function
  Serial.print("Kekeruhan value: ");
  Serial.println(KekeruhanVoltage,2);
}
```

- Software untuk Modul Kincir Air

```
#define RELAY_ON 0
#define RELAY_OFF 1
#define RELAY_1  2

void setup() {
  pinMode(RELAY_1, OUTPUT);
  digitalWrite(RELAY_1, RELAY_OFF);
}

void loop()
{
  // Turn on and wait 3 seconds.
  digitalWrite(RELAY_1, RELAY_ON);
  delay(3000);
```

```
  // Turn off and wait 3 seconds.
  digitalWrite(RELAY_1, RELAY_OFF);
  delay(3000);
}
```

- Source Code Modul HMI

```
/*************************************************************************
=========================================================================
Project Name    : Development
                  Prototype of Final Project: Human Machine Interface (HMI)
Revision        : 0.7
Date Created    : August 27, 2015
Date Revised    : April, 14 2017
Developer       : Edwin Sanjaya
Dev Mail        : sanjayaedwin@gmail.com
Original Author : Baharuddin Aziz
Author Mail     : mail@baha.web.id
Company         : Department of Electrical Engineering
                  School of Electrical Engineering and Informatics
                  Bandung Institute of Technology (ITB)
/*************************************************************************
> Microcontroller Board :
   Arduino Mega 2560 R3 (IDE used is Arduino 1.6.4)
> Pin Configuration (see PinConfiguration v20151107.pdf) :
   Keypad 4x4 :
     Row 0    = pin 29
     Row 1    = pin 28
     Row 2    = pin 27
     Row 3    = pin 26
     Column 0 = pin 25
     Column 1 = pin 24
     Column 2 = pin 23
     Column 3 = pin 22
   LCD 20x4 :
     1  (VSS)  = -
     2  (VDD)  = 5V
     3  (V0)   = GND
     4  (RS)   = pin 12
     5  (RW)   = GND
     6  (E)    = pin 9 (Using GSM Pin Because of Mistakes)
     7  (DB0)  = -
     8  (DB1)  = -
     9  (DB2)  = -
     10 (DB3)  = -
     11 (DB4)  = pin 5
     12 (DB5)  = pin 4
     13 (DB6)  = pin 3
     14 (DB7)  = pin 2
     15 (LEDA) = 3.3V
     16 (LEDK) = GND
   Tiny RTC I2C Modules :
     SQ  = -
     DS  = -
     SCL = pin 21 (SCL)
     SDA = pin 20 (SDA)
     VCC = 5V
     GND = GND
*************************************************************************/

#include <LiquidCrystal.h>  // include the LCD library code
#include <SPI.h> //Call SPI library so you can communicate with the nRF24L01+
#include <nRF24L01.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/
#include <RF24.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/
#include <Keypad.h>
#include <Wire.h>
#include <Time.h>           // include time library code
#include <DS1307RTC.h>      // include the DS1307 RTC library code
#include <SoftwareSerial.h>
#include <SD.h>             // include the SD library


/* LCD 20x4 */
LiquidCrystal lcd(12, 9, 5, 4, 3, 2);  // pin number that used to interface LCD 20x4

// status
const int row_SystemStatus = 1;                                    // row that used
to display status of system
const int column_SystemStatus_title = 0;                           // column that
used to display status of system (title)
```

```
const int column_SystemStatus_content = column_SystemStatus_title + 8;  // column that
used to display content of status of system

// date & clock
const int row_date = 0;          // row that used to display the date
const int row_clock = row_date;  // row that used to display the clock
const int first_date_cursor = 0;                          // first column for date cursor
const int first_clock_cursor = first_date_cursor + 11;  // first column for clock cursor

// sensor data
const int row_Sensor_DO          = 2;               // row that used to display DO value
const int row_Sensor_Temperature = 3;               // row that used to display
temperature value
const int row_Sensor_pH          = 1;               // row that used to display pH value
const int row_Sensor_Turbidity   = 2;               // row that used to display
turbidity value
const int row_Sensor_Salinity    = 3;               // row that used to display salinity
value

const int column_Sensor_DO_title          = 0;      // column that used to display DO
title
const int column_Sensor_Temperature_title = 0;      // column that used to display
temperature title
const int column_Sensor_pH_title          = 10;     // column that used to display pH
title
const int column_Sensor_Turbidity_title   = 10;     // column that used to display
turbidity title
const int column_Sensor_Salinity_title    = 10;     // column that used to display
salinity title

const int column_Sensor_DO_value          = 5;      // column that used to display DO
value
const int column_Sensor_Temperature_value = 5;      // column that used to display
temperature value
const int column_Sensor_pH_value          = 14;     // column that used to display pH
value
const int column_Sensor_Turbidity_value   = 14;     // column that used to display
turbidity value
const int column_Sensor_Salinity_value    = 14;     // column that used to display
salinity value

/* KEYPAD */
const int row_keypress = 1;                                    // row that used for
keypress
const int first_keypress = 6;                                  // first column for
keypress cursor
const int length_of_LCD_keypress_value = 16 - first_keypress;  // length of LCD keypress
value

char mode = 'A'; // display 1st RPM data in start-up
char GSMkey;
char ETCkey;

/* RF MODULE*/
float data_RPM1[5]; //Array of data received from 1st RPM
//data_RPM1[0] : DO measurement value
//data_RPM1[1] : temperature measurement value
//data_RPM1[2] : pH measurement value
//data_RPM1[3] : turbidity measurement value
//data_RPM1[4] : salinity measurement value

float data_RPM2[2]; //Array of data received from 2nd RPM
//data_RPM2[0] : temperature measurement value
//data_RPM2[1] : turbidity measurement value

const int pinCE  = 48;          //CHIP ENABLER : This pin is used to set the nRF24 to
standby (0) or active mode (1)
const int pinCSN = 53;          //CHIP SELECT NOT : This pin is used to tell the nRF24
whether the SPI communication is a command or message to send out
RF24 radio(pinCE, pinCSN);      //Declare object from nRF24 library (Create your
wireless SPI)
//Create up to 6 pipe addresses P0 - P5;  the "LL" is for LongLong type

#define WHICH_NODE 3     // must be a number from 1 - 6 identifying the PTX node
const uint64_t rAddress[] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL,
0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL };
```

```
const uint64_t PTXpipe = rAddress[ WHICH_NODE - 1 ];   // Pulls the address from the
above array for this node's pipe

/* KEYPAD */
// constants for rows and columns on the keypad
const byte numRows = 4;  // number of rows on the keypad
const byte numCols = 4;  // number of columns on the keypad
// keymap defines the key pressed according to the row and columns just as appears on
the keypad
char keymap[numRows][numCols] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

// code that shows the the keypad connections to the arduino terminals
byte rowPins[numRows] = {22, 24, 26, 28};  // rows 0 to 3    --- row    0..3
byte colPins[numCols] = {29, 27, 25, 23};  // columns 0 to 3 --- column 0..3

// initializes an instance of the keypad class
Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);

/* GSM */
SoftwareSerial GSM(10, 11);           //GSM Receive and Transmission PIN

/* SD Card */
const int SDCS = 49; // SD Card Chip Select (Using SPI)

/* LED RPM1 (status) */
const int LED_1R  = 30;  // pin number of indicator LED (Red) of DO status
const int LED_2R  = 32;  // pin number of indicator LED (Red) of temperature status
const int LED_3R  = 34;  // pin number of indicator LED (Red) of pH status
const int LED_4R  = 36;  // pin number of indicator LED (Red) of Turbidity status
const int LED_5R  = 38;  // pin number of indicator LED (Red) of Salinity status
const int LED_1G  = 43;  // pin number of indicator LED (Green) of DO status
const int LED_2G  = 41;  // pin number of indicator LED (Green) of temperature status
const int LED_3G  = 39;  // pin number of indicator LED (Green) of pH status
const int LED_4G  = 37;  // pin number of indicator LED (Green) of Turbidity status
const int LED_5G  = 35;  // pin number of indicator LED (Green) of Salinity status

/* LED RPM2 (status) */
const int LED_6R  = 40;  // pin number of indicator LED (Red) of temperature status
const int LED_7R  = 42;  // pin number of indicator LED (Red) of Turbidity status
const int LED_6G  = 33;  // pin number of indicator LED (Green) of temperature status
const int LED_7G  = 31;  // pin number of indicator LED (Green) of Turbidity status

/* Relay */
const int Relay1  = 18;  // Relay to control speaker
const int Relay2  = 19;  // Additional unused relay that can be used for future
development

/*=============================================================================*/


/*============================( DECLARE VARIABLES )============================*/
/* General */
int MainCounter;

/* RTC */
// set up variables using the RTC utility library functions
tmElements_t tm;
// common variable for RTC
int RTC ThisDay;  // variable for checking and removing old file
int RTC_NextDay;  // variable for checking and removing old file

/* GSM Module */
boolean GSM_started = false;   // as flag for check the status of GSM Module, started or
not yet
char    GSM_PhoneNumber[20];   // used to store Phone Number of destination
char    GSM_PhoneNumber1[20];  // used to store Phone Number of destination
char    GSM_PhoneNumber2[20];  // used to store Phone Number of destination
char    GSM_CMGS[30];          // used to store string for sending message to Phone
Number
```

```
char    GSM_CMGS1[30];          // used to store string for sending message to Phone
Number
char    GSM_CMGS2[30];          // used to store string for sending message to Phone
Number
char    PhoneIndex = '0';       // used as index to select phone number for edit and
delete
int     SMS_Flag = 0;
int     SD_Flag = 0;
int     Number_Flag = 1;
int     Number_Flag1 = 0;
int     Number_Flag2 = 0;
int     Reply_Flag = 0;


/* KEYPAD */
char keypressed;
int ETC_Flag = 0;

/* SD CARD MODULE */
// set up variables using the SD utility library functions
SdVolume  volume;
Sd2Card   card;
SdFile    root;
// variable for create, write, and delete file
File myFile1, myFile2;
// file name for data logging
char SDCard_FileName[12];  // file name: YYYYMMDD.TXT
// for make sure only one data will be written
int SDCard DataLogging Write;  // flag for writting process to SD card
// for marking the data that already received from RMP
int SDCard Marker;
// for making new file for NEXT DAY
int SDCard_NewFile_NewDay_FLAG; // flag for creating new file on NEXT DAY

// RPM1
String Status Temperature;  // status of temperature
String Status pH;           // status of pH
String Status_DO;           // status of DO
String Status_Turbidity;    // status of turbidity
String Status_Salinity;     // status of salinity
unsigned long    RPM1_Time = 0;       // lastest time when receive data from RPM1
String           RPM1_Cond = "NC";    // condition of RPM1 transmission

// RPM2
String Status_Temperature2; // status of temperature
String Status Turbidity2;   // status of turbidity
unsigned long    RPM2_Time = 0;       // lastest time when receive data from RPM1
String           RPM2_Cond = "NC";    // condition of RPM1 transmission

/* Status Notes
OK : Normal range
-- : Under normal range
++ : Above normal range
*/

void setup()
{
  /* Serial Monitor */
  Serial.begin(9600);  //start serial to communication
  Serial.println(">> START()");

  /* LCD 20x4 */
  lcd.begin(20, 4);                                 // sum of column = 20, sum of row =
4


  /* RF MODULE */
  procedure_RF_SETUP();

  /* GSM */
  procedure_GSM_SETUP();
  //procedure_GSM_SendMessage_Initialize();

  /* SD CARD */
  procedure_SDCard_Initialization();                // check the presence of SD card
  procedure_SDCard_FileAvailableCheck();
```

```
  /* LCD 20x4 */
  lcd.begin(20, 4);                                    // sum of column = 20, sum of row =
4
  procedure_LCD_Startup();

  /* LED */
  procedure_LED_SETUP();

  /* Relay */
  procedure_Relay_SETUP();
}

void loop()
{
  // Reading Data from RPM with RF24
  procedure_RF_ReadData();
  procedure_RF_Condition();

  // Waiting for Keypad Input to Change Mode
  keypressed = myKeypad.getKey();
  if (keypressed != NO_KEY)
  {
    if (keypressed == 'A' || keypressed == 'B' || keypressed == 'C' || keypressed == 'D'
)
    {
      if (mode != keypressed)
      {
        lcd.clear();
      }
      mode = keypressed;
    }
  }

  // Mode A : Display RPM1 Data
  // Mode B : Display RPM2 Data
  // Mode C : Edit no HP untuk target GSM
  // Mode D : ETC Menu
  if (mode == 'A')
  {
    procedure_LCD_SETUP_SensorData1();
  }
  else if (mode == 'B')
  {
    Serial.println("Mode B");
    procedure_LCD_SETUP_SensorData2();
  }
  else if (mode == 'C')
  {
    procedure_GSM_Menu();
    if(GSMkey == '1')
    {
      procedure_GSM_PhoneNumber_Display();
    }
    else if(GSMkey == '2')
    {
      procedure_GSM_PhoneNumber_SETUP_Menu();
      if (GSMkey != '0')
      {
        procedure_GSM_PhoneNumber_SETUP();
      }
    }
    else if(GSMkey == '3')
    {
      procedure_GSM_PhoneNumber_Delete_Menu();
      if (GSMkey != '0')
      {
        procedure_GSM_PhoneNumber_Delete();
      }
    }
  }
  else if (mode == 'D')
  {
    while (ETC_Flag == 0)
    {
      procedure_ETC_Menu1();
```

```
      if (ETC_Flag == 0)
      {
        procedure_ETC_Menu2();
      }
    }
    ETC_Flag = 0;
    if(ETCkey == '1')
    {
      procedure_Setime();
    }
    else if(ETCkey == '2')
    {
      digitalWrite(Relay1, 0);
      ETCkey == '0';
      mode = 'A';
    }
    else if(ETCkey == '3')
    {
      procedure_GSM_SendMessage();
    }
    else if(ETCkey == '4') //Write SD Card
    {
      procedure_SDCard_FileName();                                    // SDCard_FileName
variable is assigned with the date of today
      procedure_SDCard_DataLogging();
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Menulis ke SD Card");
      delay (1500);
      lcd.clear();
      mode = 'A';
    }
    else if(ETCkey == '5') //Start Kincir 1
    {
      radio.stopListening(); //go into transmit mode
      radio.openWritingPipe(PTXpipe);        //open writing or transmit pipe
      procedure_Trigger_Turbine1();
      radio.startListening(); //Switch back to a receiver
      mode = 'A';
    }
  }

  // Transfer data to SD Card every 30 Minutes
  if((tm.Minute % 30) == 0 && SD_Flag == 1)
  {
    procedure_SDCard_FileName();                                    // SDCard_FileName
variable is assigned with the date of today
    procedure_SDCard_DataLogging();
    SD_Flag = 0;
  }
  if((tm.Minute % 30) == 29)
  {
    SD_Flag = 1;
  }

   /* RTC */
  procedure_RTC();
  procedure_SensorData_ConditionCheck();

  /* GSM SMS Reading */
  procedure_GSM_WaitingSMS();
  if (Reply_Flag == 1)
  {
    Serial.println(Reply_Flag);
    procedure_GSM_SendValue();
    Serial.println("Done Reply");
    Reply_Flag = 0;
    lcd.clear();
    mode = 'A';
  }

  /*
  if ((millis() > 12000) && (millis()<15000))
  {
    radio.stopListening(); //go into transmit mode
    radio.openWritingPipe(PTXpipe);        //open writing or transmit pipe
```

```
    procedure Trigger Turbine1();
    radio.startListening(); //Switch back to a receiver
  }
  */




  /* TESTING!!!!!
  if (!strcmp(GSM_PhoneNumber1,"Kosong") == 0)
  {
    Serial.println("No");
  }
  */

  /*
  if((tm.Minute % 60) == 0 && SMS Flag == 1)
  {
    procedure_GSM_SendMessage();
    SMS_Flag = 0;
  }
  if((tm.Minute % 60) == 29)
  {
    SMS Flag = 1;
  }
  */


}

void procedure_Trigger_Turbine1()
{
  int deliver = 0;
  int counter = 0;
  byte Trigger1 = 1;

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Transmit to Kincir 1");

  while (deliver == 0)
  {
    radio.write( &Trigger1, 1 );        //open writing or transmit pipe

    if (!radio.write( &Trigger1, 1 ))
    { //if the write fails let the user know over serial monitor
      counter = counter + 1;
    }
    else
    { //if the write was successful
      lcd.clear();
      lcd.print("Sent to Kincir 1");
      delay(2000);
      deliver = 1;
    }
    if (counter == 10)
    {
      lcd.clear();
      lcd.print("Fail");
      deliver = 1;
    }
    delay(1000);
  }
  lcd.clear();
}

void procedure_Counter()
// input   : N/A
// output  : counter in integer
// process : 1. counting every 1 second
//           2. counting is reseted when reach 3600 seconds
{
  Serial.println(">> procedure_Counter()");

  if (MainCounter < 60)
```

```
  {
    MainCounter = millis() / 1000;
  }
  else
  {
    MainCounter = 0;
  }
  Serial.println(MainCounter);
}

/*----------------------------------( LCD )----------------------------------*/

void procedure_LCD_Startup()
// input   : N/A
// output  : display the text when e-Shrimp doing start-up
// process : 1. print text "e-Shrimp v0.7"
//           2. print text "Developed by : TA.161701060"
{
  lcd.setCursor(6,1);
  lcd.print("e-Shrimp");
  lcd.setCursor(8,2);
  lcd.print("v0.7");
  delay(2500);
  lcd.clear();
  lcd.setCursor(3,1);
  lcd.print("Developed By :");
  lcd.setCursor(4,2);
  lcd.print("TA.161701060");
  delay(2500);
  lcd.clear();
}

void procedure_LCD_SETUP_SensorData1()
// input   : N/A
// output  : display the text "Suhu  =" and "pH    =" on LCD
// process : 1. print text "Suhu  ="
//           2. print text "pH    ="
{
  //Serial.println(">> procedure_LCD_SETUP_SensorData()");

  // RPM
  lcd.setCursor(0,1);
  lcd.print("RPM1");
  lcd.setCursor(6,1);
  lcd.print(RPM1_Cond);

  // Temperature
  lcd.setCursor(column_Sensor_Temperature_title, row_Sensor_Temperature);  // column =
column_Sensor_Temperature_title, row = row_Sensor_Temperature
  lcd.print("Suhu");
  lcd.setCursor(column_Sensor_Temperature_value, row_Sensor_Temperature);      // column
= column Sensor Temperature value, row = row Sensor Temperature
  lcd.print(data_RPM1[1],1);

  // salinity
  lcd.setCursor(column_Sensor_Salinity_title, row_Sensor_Salinity);            // column
= column_Sensor_Salinity_title, row = row_Sensor_Salinity
  lcd.print("Sal ");                                                          // display
text "Garam =" on LCD
  lcd.setCursor(column_Sensor_Salinity_value, row_Sensor_Salinity);            // column
= column_Sensor_Salinity_value, row = row_Sensor_Salinity
  lcd.print(data_RPM1[4],1);

  // pH
  lcd.setCursor(column_Sensor_pH_title, row_Sensor_pH);  // column =
column_Sensor_Temperature_title, row = row_Sensor_Temperature
  lcd.print("pH");
  lcd.setCursor(column_Sensor_pH_value, row_Sensor_pH);      // column =
column Sensor Temperature value, row = row Sensor Temperature
  lcd.print(data_RPM1[2],1);                                                   //
display text "pH    =" on LCD

  // DO
  lcd.setCursor(column_Sensor_DO_title, row_Sensor_DO);                        // column
= column_Sensor_DO_title, row = row_Sensor_DO
```

```
  lcd.print("DO");                                                    // display
text "DO    =" on LCD
  lcd.setCursor(column_Sensor_DO_value, row_Sensor_DO);                    // column
= column_Sensor_DO_value, row = row_Sensor_DO
  lcd.print(data_RPM1[0],1);                                               //
display value of DO

  // turbidity
  lcd.setCursor(column_Sensor_Turbidity_title, row_Sensor_Turbidity);      // column
= column_Sensor_Turbidity_title, row = row_Sensor_Turbidity
  lcd.print("Tur");                                                  // display
text "Keruh =" on LCD
  lcd.setCursor(column_Sensor_Turbidity_value, row_Sensor_Turbidity);      // column
= column Sensor Turbidity value, row = row Sensor Turbidity
  lcd.print(data_RPM1[3],1);

}

void procedure_LCD_SETUP_SensorData2()
// input   : N/A
// output  : display the text "Suhu  =" and "pH    =" on LCD
// process : 1. print text "Suhu         ="
//           2. print text "Turbidity    ="
{
  //Serial.println(">> procedure LCD SETUP SensorData()");

  // RPM
  lcd.setCursor(0,1);
  lcd.print("RPM2");
  lcd.setCursor(6,1);
  lcd.print(RPM2_Cond);

  // Temperature
  lcd.setCursor(column_Sensor_DO_title, row_Sensor_DO);      // using DO position
  lcd.print("Suhu");
  lcd.setCursor(column_Sensor_DO_value, row_Sensor_DO);      // using DO position
  lcd.print(data_RPM2[0],1);

  // turbidity
  lcd.setCursor(column_Sensor_Turbidity_title, row_Sensor_Turbidity);      // column
= column Sensor Turbidity title, row = row Sensor Turbidity
  lcd.print("Tur");                                                  // display
text "Keruh =" on LCD
  lcd.setCursor(column_Sensor_Turbidity_value, row_Sensor_Turbidity);      // column
= column_Sensor_Turbidity_value, row = row_Sensor_Turbidity
  lcd.print(data_RPM2[1],1);

}

/*---------------------------------( RF MODULE )---------------------------------*/

void procedure_RF_SETUP()
// input   : N/A
// output  : N/A
// process : Set pipe address of RPM to start receiving data
{
  radio.begin();  //Start the nRF24 module
  radio.setPALevel(RF24_PA_MAX);  // "short range setting" - increase if you want more
range AND have a good power supply
  radio.setDataRate(RF24_250KBPS);
  radio.setChannel(22);          // the higher channels tend to be more "open"
  // Open up to four pipes for PRX to receive data
  radio.openReadingPipe(0,rAddress[0]);
  radio.openReadingPipe(1,rAddress[1]);
  radio.openReadingPipe(2,rAddress[2]);
  radio.openReadingPipe(3,rAddress[3]);
  radio.openReadingPipe(4,rAddress[4]);
  radio.openReadingPipe(5,rAddress[5]);
  radio.startListening();                 // Start listening for messages
}

void procedure_RF_ReadData()
// input   : N/A
// output  : data of water quality parameter
// process : Receive data sent by RPM modules
{
```

```
  byte pipeNum = 0; //variable to hold which reading pipe sent data

  /* RF MODULE */
  if(radio.available(&pipeNum))
  { //Check if received data
    switch (pipeNum)
    {
    case 0:
      radio.read( &data_RPM1, sizeof(data_RPM1) );
      RPM1_Time = millis() + 10000;
      break;
    case 1:
      radio.read( &data_RPM2, sizeof(data_RPM2) );
      RPM2_Time = millis() + 10000;
      break;
    break;
    }
  }
}

void procedure_RF_Condition()
// input   : RF1_Time and RF2_Time as last connectivity time
// output  : data of water quality parameter
// process : Receive data sent by RPM modules
{
  unsigned long current_time = millis();
  if(current_time < RPM1_Time)
  {
    RPM1_Cond = "OK";
  }
  else
  {
    RPM1_Cond = "NC";
  }

  if(current_time < RPM2_Time)
  {
    RPM2_Cond = "OK";
  }
  else
  {
    RPM2_Cond = "NC";
  }
}

/*---------------------------------( RTC )---------------------------------*/
void procedure_RTC()
// input   : N/A
// output  : display of clock and date
// process : 1. check clock and date from RTC module
//           2. display the clock and date from RTC module
{
  // Serial.println(">> procedure RTC()");

  if (RTC.read(tm))                                   // check the condition of RTC
function
  {
    /* Print DATE to LCD */
    // Day [1..31]
    if (tm.Day < 10)                                  // condition for date number <
10
    {
      lcd.setCursor(first_date_cursor, row_date);     // column = first_date_cursor,
row = row date
      lcd.print('0');                                 // add text '0' on LCD
      lcd.setCursor(first_date_cursor + 1, row_date); // column = first_date_cursor +
1, row = row_date
      lcd.print(tm.Day);                              // display the date number
[1..9] on LCD
    }
    else                                              // condition for date number >=
10
    {
      lcd.setCursor(first_date_cursor, row_date);     // column = first_date_cursor,
row = row date
```

```
      lcd.print(tm.Day);                                    // display the date number
[10..31] on LCD
    }

    lcd.print("/");

    // Month [1..12]
    if (tm.Month < 10)                                      // condition for month number <
10
    {
      lcd.setCursor(first_date_cursor + 3, row_date);       // column = first_date_cursor +
3, row = row date
      lcd.print('0');                                      // add text '0' on LCD
      lcd.setCursor(first_date_cursor + 4, row_date);       // column = first_date_cursor +
4, row = row_date
      lcd.print(tm.Month);                                  // display the month number
[1..9] on LCD
    }
    else                                                    // condition for month number
>= 10
    {
      lcd.setCursor(first_date_cursor + 3, row_date);       // column = first_date_cursor +
3, row = row date
      lcd.print(tm.Month);                                  // display the month number
[10..12] on LCD
    }

    lcd.print("/");

    // Year [xxxx]
    lcd.setCursor(first_date_cursor + 6, row_date);         // column = first_date_cursor +
6, row = row_date
    lcd.print(tmYearToCalendar(tm.Year));                   // display the year number
[xxxx] on LCD

    /* Print CLOCK to LCD */
    // Hour [0..23]
    if (tm.Hour < 10)                                       // condition for hour number <
10
    {
      lcd.setCursor(first_clock_cursor, row_clock);         // column = first_clock_cursor,
row = row clock
      lcd.print('0');                                      // add text '0' on LCD
      lcd.setCursor(first_clock_cursor + 1, row_clock);     // column = first_clock_cursor
+ 1, row = row_clock
      lcd.print(tm.Hour);                                   // display the hour number
[0..9] on LCD
    }
    else                                                    // condition for hour number >=
10
    {
      lcd.setCursor(first_clock_cursor, row_clock);         // column = first_clock_cursor,
row = row clock
      lcd.print(tm.Hour);                                   // display the hour number
[10..23] on LCD
    }

    lcd.print(":");

    // Minute [0..59]
    if (tm.Minute < 10)                                     // condition for minute number
< 10
    {
      lcd.setCursor(first_clock_cursor + 3, row_clock);     // column = first_clock_cursor
+ 3, row 3 = row clock
      lcd.print('0');                                      // add text '0' on LCD
      lcd.setCursor(first_clock_cursor + 4, row_clock);     // column = first_clock_cursor
+ 4, row 3 = row_clock
      lcd.print(tm.Minute);                                 // display the minute number
[0..9] on LCD
    }
    else                                                    // condition for minute number
>= 10
    {
      lcd.setCursor(first_clock_cursor + 3, row_clock);     // column = first_clock_cursor
+ 3, row = row_clock
```

```
      lcd.print(tm.Minute);                                  // display the minute number
[10..59] on LCD
    }
  }
  else
  {
    /*procedure LED SystemStatus RTC Error();              // red LED turned on when RTC
do not work properly*/
  }
}

void procedure_Setime()
{
  // local variable
  boolean entry = false;
  char localkey;
  int Time_Temp[12];
  int Pointer1 = 0;
  int Pointer2 = 0;
  int NumCount = 0;
  int Y_temp;
  int M_temp;
  int D temp;
  int h temp;
  int m temp;
  char Newtime[15];

  lcd.setCursor(0,0);
  lcd.print("Ubah Waktu");
  lcd.setCursor(0,1);
  lcd.print("DD/MM/YYYY");
  lcd.setCursor(0,2);
  lcd.print("hh:mm");
  while(entry == false)
  {
    localkey = myKeypad.getKey();
    if (NumCount<8)
    {
      if (Pointer1 == 2 || Pointer1 == 5)
      {
        Pointer1 = Pointer1 + 1;
      }
      lcd.setCursor(Pointer1,1);
      lcd.print(char(95));
    }
    else if (NumCount>7)
    {
      if (Pointer2 == 2)
      {
        Pointer2 = Pointer2 + 1;
      }
      lcd.setCursor(Pointer2,2);
      lcd.print(char(95));
    }
    if (localkey != 'A' && localkey != 'B' && localkey != 'C' && localkey != 'D' &&
localkey != '*' && localkey != '#')
    {
      if (localkey != NO_KEY)
      {
        Time_Temp[NumCount] = localkey - '0';
        if (NumCount<8)
        {
          lcd.setCursor(Pointer1,1);
        }
        else if (NumCount>7)
        {
          lcd.setCursor(Pointer2,2);
        }
        lcd.print(localkey);
        NumCount = NumCount + 1;
        Pointer1 = Pointer1 +1;
        if (Pointer1 > 10)
        {
          Pointer2 = Pointer2 +1;
        }
      }
```

```
    }
    if (localkey == '*' || NumCount==12)
    {
      entry = true;
    }
  }
  if (NumCount==12)
  {
    D_temp = Time_Temp[0]*10 + Time_Temp[1];
    M_temp = Time_Temp[2]*10 + Time_Temp[3];
    Y_temp = Time_Temp[4]*1000 + Time_Temp[5]*100 + Time_Temp[6]*10 + Time_Temp[7];
    h_temp = Time_Temp[8]*10 + Time_Temp[9];
    m_temp = Time_Temp[10]*10 + Time_Temp[11];


    tm.Day = D_temp;
    tm.Month = M_temp;
    tm.Year = CalendarYrToTm(Y_temp);
    tm.Hour = h_temp;
    tm.Minute = m_temp;
    RTC.write(tm);

    lcd.setCursor(0,3);
    lcd.print("Sukses Ganti Waktu");
    delay (2500);
    sprintf(Newtime,"%d/%d/%d %d:%d",D_temp,M_temp,Y_temp,h_temp,m_temp);
    Serial.println(Newtime);
  }
  lcd.clear();
  mode = 'A';

}

/*------------------------------( GSM Module )------------------------------*/

void procedure_GSM_SETUP()
// input   : N/A
// output  : ...
// process : ...
{
  Serial.println(">> procedure_GSM_SETUP()");
  GSM.begin(2400);   // Setting the baud rate of GSM Module
  GSM.println("AT+CMGD=1,4");
  GSM.println("AT+CNMI=2,2,0,0,0"); // Listening to incoming SMS
  sprintf(GSM_PhoneNumber, "+628111700373");
  sprintf(GSM_PhoneNumber1, "Kosong\0");
  sprintf(GSM_PhoneNumber2, "Kosong\0");
  sprintf(GSM_CMGS,"AT+CMGS=\"%s\"\r", GSM_PhoneNumber);
}

void procedure_GSM_SendMessage_Initialize()
// input   : N/A
// output  : alarm ON
// process : ...
{
  Serial.println(">> procedure_GSM_SendMessage_Initialize()");

  // local variable
  int YYYY;
  int MM;
  int DD;
  int hh;
  int mm;
  char SMS_Header[16];
  char SMS_Header_Date[11];
  char SMS_Header_Time[5];

  /* SMS Header */
  // initializing
  YYYY = tmYearToCalendar(tm.Year);                    // year of today
  MM = tm.Month;                                       // month of today
  DD = tm.Day;                                         // date of today
  hh = tm.Hour;                                        // hour of now
  mm = tm.Minute;                                      // minute of now

  // date
```

```
  if ((MM < 10) && (DD < 10))                                  // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/0%d/%d", DD, MM, YYYY);  // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM < 10) && (DD >= 10))                            // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/0%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM >= 10) && (DD < 10))                            // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else                                                         // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/%d/%d", DD, MM, YYYY);     // assign SMS_Header_Date
variable with YYYYMMDD
  }


  // time
  if ((hh < 10) && (mm < 10))                                  // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:0%d", hh, mm);          // assign SMS_Header_Time
variable with hhmm
  }
  else if ((hh < 10) && (mm >= 10))                            // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:%d", hh, mm);           // assign SMS_Header_Time
variable with hhmm
  }
  else if ((hh >= 10) && (mm < 10))                            // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:0%d", hh, mm);           // assign SMS_Header_Time
variable with hhmm
  }
  else                                                         // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:%d", hh, mm);                  // assign SMS_Header_Time
variable with hhmm
  }


  // Header: date time (DD/MM/YYYY hh:mm)
  sprintf(SMS_Header, "%s%c%s",SMS_Header_Date,char(32),SMS_Header_Time);
  Serial.println(SMS_Header);


  GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
  delay(1000);  // Delay of 1000 milli seconds or 1 second
  GSM.println("AT+CMGS=\"+628111700373\"\r"); // Replace x with mobile number
  delay(1000);
  GSM.println(SMS_Header);// The SMS text you want to send
  GSM.println("Terhubung dengan e-Shrimp");// The SMS text you want to send
  delay(100);
  GSM.println((char)26);// ASCII code of CTRL+Z
  delay(1000);
}

void procedure_GSM_SendMessage()
// input   : N/A
// output  :
// process : ...
{
  Serial.println(">> procedure_GSM_SendMessage_Initialize()");
```

```
  // local variable
  int YYYY;
  int MM;
  int DD;
  int hh;
  int mm;
  char SMS_Header[16];
  char SMS_Header_Date[11];
  char SMS_Header_Time[5];

  /* SMS Header */
  // initializing
  YYYY = tmYearToCalendar(tm.Year);                    // year of today
  MM = tm.Month;                                       // month of today
  DD = tm.Day;                                         // date of today
  hh = tm.Hour;                                        // hour of now
  mm = tm.Minute;                                      // minute of now

  // date
  if ((MM < 10) && (DD < 10))                          // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/0%d/%d", DD, MM, YYYY);  // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM < 10) && (DD >= 10))                    // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/0%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM >= 10) && (DD < 10))                    // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else                                                 // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/%d/%d", DD, MM, YYYY);     // assign SMS_Header_Date
variable with YYYYMMDD
  }


  // time
  if ((hh < 10) && (mm < 10))                          // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:0%d", hh, mm);          // assign SMS_Header_Time
variable with hhmm
  }
  else if ((hh < 10) && (mm >= 10))                    // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:%d", hh, mm);           // assign SMS_Header_Time
variable with hhmm
  }
  else if ((hh >= 10) && (mm < 10))                    // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:0%d", hh, mm);           // assign SMS_Header_Time
variable with hhmm
  }
  else                                                 // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:%d", hh, mm);                // assign SMS_Header_Time
variable with hhmm
  }


  // Header: date_time (DD/MM/YYYY hh:mm)
  sprintf(SMS_Header, "%s%c%s",SMS_Header_Date,char(32),SMS_Header_Time);
  Serial.println(SMS_Header);
  lcd.clear();
```

```
  lcd.setCursor(0,0);
  lcd.print("Mengirim SMS...");


  if (Number_Flag == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);
    procedure_SendMessage_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(3000);
  }
  if (Number_Flag1 == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS1); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);// The SMS text you want to send
    procedure_SendMessage_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(3000);
  }
  if (Number_Flag2 == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS2); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);// The SMS text you want to send
    procedure_SendMessage_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(1500);
    //GSM.println("AT+CNMI=2,2,0,0,0"); // Listening to incoming SMS
    delay(1500);
  }
  lcd.clear();
  mode = 'A';
}


// Function to get SMS contents


void procedure_GSM_WaitingSMS()
{
  String buffer = readGSM();
  if (buffer.startsWith("\r\n+CMT: "))
  {
      Serial.println("*** RECEIVED SMS ***");

      // Remove first 51 characters
      buffer.remove(0, 49);
      int len = buffer.length();

      // Remove \r\n from tail
      buffer.remove(len - 2, 2);
      while (buffer[0] != 'H' && buffer[0] != '\0')
      {
        buffer.remove(0,1);
      }
      Serial.println(buffer);
      Serial.println("*** END SMS ***");
  }
  if (buffer=="HMI GETVALUE")
  {
    Reply_Flag = 1;
  }
}
```

```
String readGSM()
{
    String buffer;
    while (GSM.available()>0)
    {
        char c = GSM.read();
        buffer.concat(c);
        delay(10);
    }
    return buffer;
}

void procedure_GSM_SendValue()
// input   : N/A
// output  :
// process : ...
{
  Serial.println(">> procedure_GSM_SendValue()");

  // local variable
  int YYYY;
  int MM;
  int DD;
  int hh;
  int mm;
  char SMS_Header[16];
  char SMS_Header_Date[11];
  char SMS_Header_Time[5];

  /* SMS Header */
  // initializing
  YYYY = tmYearToCalendar(tm.Year);                       // year of today
  MM = tm.Month;                                          // month of today
  DD = tm.Day;                                            // date of today
  hh = tm.Hour;                                           // hour of now
  mm = tm.Minute;                                         // minute of now

  // date
  if ((MM < 10) && (DD < 10))                             // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/0%d/%d", DD, MM, YYYY);  // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM < 10) && (DD >= 10))                       // when (number of month < 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/0%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else if ((MM >= 10) && (DD < 10))                       // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "0%d/%d/%d", DD, MM, YYYY);   // assign SMS_Header_Date
variable with YYYYMMDD
  }
  else                                                    // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(SMS_Header_Date, "%d/%d/%d", DD, MM, YYYY);     // assign SMS_Header_Date
variable with YYYYMMDD
  }


  // time
  if ((hh < 10) && (mm < 10))                             // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:0%d", hh, mm);          // assign SMS_Header_Time
variable with hhmm
  }
  else if ((hh < 10) && (mm >= 10))                       // when (number of hour < 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "0%d:%d", hh, mm);           // assign SMS_Header_Time
variable with hhmm
```

```
  }
  else if ((hh >= 10) && (mm < 10))                        // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:0%d", hh, mm);            // assign SMS_Header_Time
variable with hhmm
  }
  else                                                     // when (number of hour >= 10)
AND (number of minute) < 10
  {
    sprintf(SMS_Header_Time, "%d:%d", hh, mm);                  // assign SMS_Header_Time
variable with hhmm
  }


  // Header: date_time (DD/MM/YYYY hh:mm)
  sprintf(SMS_Header, "%s%c%s",SMS_Header_Date,char(32),SMS_Header_Time);
  Serial.println(SMS_Header);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Mengirim SMS...");


  if (Number_Flag == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);
    procedure_SendValue_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(3000);
  }
  if (Number_Flag1 == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS1); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);// The SMS text you want to send
    procedure_SendValue_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(3000);
  }
  if (Number_Flag2 == 1)
  {
    GSM.println("AT+CMGF=1");    //Sets the GSM Module in Text Mode
    delay(1000);  // Delay of 1000 milli seconds or 1 second
    GSM.println(GSM_CMGS2); // Replace x with mobile number
    delay(1000);
    GSM.println(SMS_Header);// The SMS text you want to send
    procedure_SendValue_Content();
    delay(100);
    GSM.println((char)26);// ASCII code of CTRL+Z
    delay(3000);
  }

}

void procedure_GSM_Menu()
{
  boolean entry = false;
  GSMkey ='0';
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("MENU SMS");
  lcd.setCursor(0,1);
  lcd.print("1. Daftar No. HP");
  lcd.setCursor(0,2);
  lcd.print("2. Isi/Ubah No. HP");
  lcd.setCursor(0,3);
  lcd.print("3. Hapus No. HP");
  while (entry == false)
```

```
  {
    GSMkey = myKeypad.getKey();
    if (GSMkey == '1' || GSMkey == '2' || GSMkey == '3' || GSMkey == '*')
    {
      entry = true;
    }
  }
  if(GSMkey == '*')
  {
    GSMkey = '0';
    mode = 'A';
  }
  lcd.clear();
}

void procedure_GSM_PhoneNumber_Display()
{
  boolean entry = false;
  char localkey;
  while(entry == false)
  {
    lcd.setCursor(0,0);
    lcd.print("Daftar No. HP");
    lcd.setCursor(0,1);
    lcd.print("1. ");
    lcd.setCursor(0,2);
    lcd.print("2. ");
    lcd.setCursor(0,3);
    lcd.print("3. ");
    lcd.setCursor(3,1);
    lcd.print(GSM_PhoneNumber);
    lcd.setCursor(3,2);
    lcd.print(GSM_PhoneNumber1);
    lcd.setCursor(3,3);
    lcd.print(GSM_PhoneNumber2);
    localkey = myKeypad.getKey();
    if (localkey == '*' || localkey == '#')
    {
      entry = true;
    }
  }
 lcd.clear();
 mode = 'C';
}

void procedure_GSM_PhoneNumber_SETUP_Menu()
{
  boolean entry = false;
  while (entry == false)
  {
    lcd.setCursor(0,0);
    lcd.print("Isi/Ubah No. HP");
    lcd.setCursor(0,1);
    lcd.print("1. ");
    lcd.setCursor(0,2);
    lcd.print("2. ");
    lcd.setCursor(0,3);
    lcd.print("3. ");
    lcd.setCursor(3,1);
    lcd.print(GSM_PhoneNumber);
    lcd.setCursor(3,2);
    lcd.print(GSM_PhoneNumber1);
    lcd.setCursor(3,3);
    lcd.print(GSM_PhoneNumber2);
    PhoneIndex = myKeypad.getKey();
    if (PhoneIndex == '1' || PhoneIndex == '2' || PhoneIndex == '3' || PhoneIndex ==
'*')
    {
      entry = true;
    }
  }
  if(PhoneIndex == '*')
  {
    PhoneIndex = '0';
    GSMkey = '0';
    mode = 'C';
```

```
  }
  lcd.clear();
}

void procedure_GSM_PhoneNumber_SETUP()
{
  // local variable
  boolean entry = false;
  char localkey;
  int PhoneCount = 0;
  char GSM_Temp[15];
  char GSM_Final[20];

  while(entry == false)
  {
    lcd.setCursor(0,0);
    lcd.print("Isi/Ubah No. HP ");
    lcd.print(PhoneIndex);
    lcd.setCursor(0,1);
    lcd.print("+628");
    localkey = myKeypad.getKey();
    if (localkey != 'A' && localkey != 'B' && localkey != 'C' && localkey != 'D' &&
localkey != '*' && localkey != '#')
    {
      if (localkey != NO_KEY)
      {
        GSM_Temp[PhoneCount] = localkey;
        Serial.print(GSM_Temp[PhoneCount]);
        lcd.setCursor(PhoneCount+4,1);
        lcd.print(localkey);
        PhoneCount = PhoneCount + 1;
      }
    }
    if (localkey == '*' || localkey == '#')
    {
      entry = true;
    }
  }
  if (localkey == '#')
  {
    if (PhoneIndex == '1')
    {
      GSM_Temp[PhoneCount] = '\0';
      sprintf(GSM_PhoneNumber,"+628%s", GSM_Temp);
      sprintf(GSM_CMGS,"AT+CMGS=\"%s\"\r", GSM_PhoneNumber);
      sprintf(GSM_Final,"%s", GSM_PhoneNumber);
      Number_Flag = 1;
    }
    else if (PhoneIndex == '2')
    {
      GSM_Temp[PhoneCount] = '\0';
      sprintf(GSM_PhoneNumber1,"+628%s", GSM_Temp);
      sprintf(GSM_CMGS1,"AT+CMGS=\"%s\"\r", GSM_PhoneNumber1);
      sprintf(GSM_Final,"%s", GSM_PhoneNumber1);
      Number_Flag1 = 1;
    }
    else if (PhoneIndex == '3')
    {
      GSM_Temp[PhoneCount] = '\0';
      sprintf(GSM_PhoneNumber2,"+628%s", GSM_Temp);
      sprintf(GSM_CMGS2,"AT+CMGS=\"%s\"\r", GSM_PhoneNumber2);
      sprintf(GSM_Final,"%s", GSM_PhoneNumber2);
      Number_Flag2 = 1;
    }
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("No HP ");
    lcd.print(PhoneIndex);
    lcd.print(" Menjadi:");
    lcd.setCursor(0,1);
    lcd.print(GSM_Final);
    delay(3000);
  }
  PhoneIndex = '0';
  lcd.clear();
  mode = 'A';
```

```
}

void procedure_GSM_PhoneNumber_Delete_Menu()
{
  boolean entry = false;
  while (entry == false)
  {
    lcd.setCursor(0,0);
    lcd.print("Hapus No. HP");
    lcd.setCursor(0,1);
    lcd.print("1. ");
    lcd.setCursor(0,2);
    lcd.print("2. ");
    lcd.setCursor(0,3);
    lcd.print("3. ");
    lcd.setCursor(3,1);
    lcd.print(GSM_PhoneNumber);
    lcd.setCursor(3,2);
    lcd.print(GSM_PhoneNumber1);
    lcd.setCursor(3,3);
    lcd.print(GSM_PhoneNumber2);
    PhoneIndex = myKeypad.getKey();
    if (PhoneIndex == '1' || PhoneIndex == '2' || PhoneIndex == '3' || PhoneIndex ==
'*')
    {
      entry = true;
    }
  }
  if(PhoneIndex == '*')
  {
    PhoneIndex = '0';
    GSMkey = '0';
    mode = 'C';
  }
  lcd.clear();
}

void procedure_GSM_PhoneNumber_Delete()
{
  if (PhoneIndex == '1')
  {
    sprintf(GSM_PhoneNumber, "Kosong\0");
    Number_Flag = 0;
  }
  else if (PhoneIndex == '2')
  {
    sprintf(GSM_PhoneNumber1, "Kosong\0");
    Number_Flag1 = 0;
  }
  else if (PhoneIndex == '3')
  {
    sprintf(GSM_PhoneNumber2, "Kosong\0");
    Number_Flag2 = 0;
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("No HP ");
  lcd.print(PhoneIndex);
  lcd.setCursor(0,1);
  lcd.print("Telah Dihapus!");
  delay(3000);
  PhoneIndex = '0';
  lcd.clear();
  mode = 'A';
}


/*---------------------------------( SD CARD )---------------------------------*/

void procedure_SDCard_Initialization()
// input   : N/A
// output  : status of SD card
// process : 1. check the presence of SD card
//           2. display the status on serial monitor
{
  Serial.println(">> procedure_SDCard_Initialization()");
```

```
  /* Initializing */
  // serial monitor
  Serial.println("#2");
  Serial.print("Initializing SD card...");

  if (!SD.begin(SDCS))  // SD card is not available or not detected
  {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
}

void procedure_SDCard_FileName()
// input   : N/A
// output  : SDCard FileName variable is assigned with date of today
// process : 1. check date of today
//           2. assign SDCard FileName variable with date of today
{
  Serial.println(">> procedure_SDCard_FileName()");

  // local variable
  int YYYY;
  int MM;
  int DD;

  /* Create File Name */
  // initializing
  YYYY = tmYearToCalendar(tm.Year);                          // year of today
  MM = tm.Month;                                             // month of today
  DD = tm.Day;                                               // date of today

  if ((MM < 10) && (DD < 10))                                // when (number of month <
10) AND (number of date) < 10
  {
    sprintf(SDCard_FileName, "%d0%d0%d.TXT", YYYY, MM, DD);  // assign SDCard_FileName
variable with YYYYMMDD.TXT
  }
  else if ((MM < 10) && (DD >= 10))                          // when (number of month <
10) AND (number of date) < 10
  {
    sprintf(SDCard_FileName, "%d0%d%d.TXT", YYYY, MM, DD);   // assign SDCard_FileName
variable with YYYYMMDD.TXT
  }
  else if ((MM >= 10) && (DD < 10))                          // when (number of month >=
10) AND (number of date) < 10
  {
    sprintf(SDCard_FileName, "%d%d0%d.TXT", YYYY, MM, DD);   // assign SDCard_FileName
variable with YYYYMMDD.TXT
  }
  else                                                       // when (number of month >=
10) AND (number of date) < 10
  {
    sprintf(SDCard_FileName, "%d%d%d.TXT", YYYY, MM, DD);    // assign SDCard_FileName
variable with YYYYMMDD.TXT
  }
}

void procedure_SDCard_FileAvailableCheck()
// input   : N/A
// output  : done checking the availability of file name
//           that named by YYYYMMDD of today
// process : 1. assign SDCard FileName variable
//           2. check the presence of filename that named with YYYYMMDD of today
//           3. when that filename not exist,
//              delete old file, then create new file that named with YYYYMMDD of today
{
  Serial.println(">> procedure_SDCard_FileAvailableCheck()");

  procedure_SDCard_FileName();         // SDCard_FileName variable is assigned with the
date of today
  // check the presence of filename that named with YYYYMMDD of today
  if (!SD.exists(SDCard_FileName))     // filename that named with YYYYMMDD of today is
not exist
  {
```

```
    procedure_SDCard_RemoveOldFile();  // delete old file
    procedure_SDCard_NewFile();        // then, create new file that named with YYYYMMDD
of today
  }
}

void procedure_SDCard_DataLogging()
// input   : N/A
// output  : data sensor are written on file
// process : 1. print date of today to file
//           2. print clock at that time
//           3. print data sensor value of each parameter
//           4. print SDCard Marker value
{
  Serial.println(">> procedure_SDCard_DataLogging()");

  /* SENSOR DATA LOGGING - YYYYMMDD.TXT */
  // open the file. note that only one file can be open at a time,
  // so we have to close this one before opening another.

  // open YYYYMMDD.TXT file that already exist
  // if it isn't, then create YYYYMMDD.TXT as new file
  myFile1 = SD.open(SDCard_FileName, FILE_WRITE);

  // if the file opened okay, write to it:
  if (myFile1)
  {
    Serial.print("Writing to ");                    // display on serial monitor (Arduino
IDE)
    Serial.print(SDCard_FileName);
    Serial.print("...");

    /* date */
    // Year
    myFile1.print(tmYearToCalendar(tm.Year));
    // Month
    if (tm.Month < 10)                              // condition for month number < 10
    {
      myFile1.print('0');                           // add text '0' before month number
      myFile1.print(tm.Month);                      // print month number to file
    }
    else                                            // condition for month number >= 10
    {
      myFile1.print(tm.Month);                      // print month number to file
    }
    // Day
    if (tm.Day < 10)                                // condition for date number < 10
    {
      myFile1.print('0');                           // add text '0' before date number
      myFile1.print(tm.Day);                        // print date number to file
    }
    else                                            // condition for date number >= 10
    {
      myFile1.print(tm.Day);                        // print date number to file
    }

    /* separator */
    myFile1.print('_');                             // print separator with 'underline'

    /* clock */
    // Hour
    if (tm.Hour < 10)                               // condition for hour number < 10
    {
      myFile1.print('0');                           // add text '0' before hour number
      myFile1.print(tm.Hour);                       // print hour number to file
    }
    else                                            // condition for hour number >= 10
    {
      myFile1.print(tm.Hour);                       // print hour number to file
    }
    // Minute
    if (tm.Minute < 10)                             // condition for minute number < 10
    {
      myFile1.print('0');                           // add text '0' before minute number
      myFile1.print(tm.Minute);                     // print minute number to file
    }
```

```
    else                                          // condition for minute number >= 10
    {
      myFile1.print(tm.Minute);                   // print minute number to file
    }
    // Second
    if (tm.Second < 10)                           // condition for second number < 10
    {
      myFile1.print('0');                         // add text '0' before second number
      myFile1.print(tm.Second);                   // print second number to file
    }
    else                                          // condition for second number >= 10
    {
      myFile1.print(tm.Second);                   // print second number to file
    }

    /* separator */
    myFile1.print('_');                           // print separator with 'underline'

    //    /* keypress */
    //    for (int i = 1; i <= length_of_LCD_keypress_value; i++)
    //    {
    //      myFile1.print(LCD_keypress_value[i]);
    //    }

    /* sensor data */
    myFile1.print(data_RPM1[0]);      // print temperature value to file
    myFile1.print('_');                           // print separator with 'underline'
    myFile1.print(data_RPM1[1]);               // print pH value to file
    myFile1.print('_');                           // print separator with 'underline'
    myFile1.print(data_RPM1[2]);               // print DO value to file
    myFile1.print('_');                           // print separator with 'underline'
    myFile1.print(data_RPM1[3]);        // print turbidity value to file
    myFile1.print('_');                           // print separator with 'underline'
    myFile1.print(data_RPM1[4]);         // print salinity value to file
    myFile1.print('_');                           // print separator with 'underline'
    myFile1.print(data_RPM1[5]);                // print SDCard_Marker value to file
    myFile1.println("");                          // change line

    // close the file
    myFile1.close();                              // close file

    Serial.println("done.");                      // display on serial monitor (Arduino
IDE)


    // reset SDCard Marker
    SDCard_Marker = 0;                            // set SDCard_Marker value as 0
  }
  else
  {
    // if the file didn't open, print an error
    Serial.print("[DataLogging] ERROR opening ");  // display on serial monitor (Arduino
IDE)
    Serial.print(SDCard_FileName);
    Serial.println("!");
  }
}

void procedure_SDCard_NewFile()
// input   : N/A
// output  : 1. new file that named with YYYYMMDD of today is created
//           2. there is 1 line of title in that file
// process : 1. open YYYYMMDD.TXT file
//           2. write the title in 1st line of that file
{
  Serial.println(">> procedure_SDCard_NewFile()");

  /* NEW FILE for SENSOR DATA LOGGING - YYYYMMDD.TXT */
  // open the file
  // note that only one file can be open at a time,
  // so we have to close this one before opening another.

  // open YYYYMMDD.TXT file that already exist
  // if it isn't, then create YYYYMMDD.TXT as new file
  myFile1 = SD.open(SDCard_FileName, FILE_WRITE);
```

```
  // if the file opened okay, write to it:
  if (myFile1)                                        // file opened normally
  {
    // serial monitor
    Serial.print("Writing to ");
    Serial.print(SDCard_FileName);
    Serial.print("...");

    /* title */
    // title of the content of file
    myFile1.println("YYYYMMDD_hhmmss_Suhu _pH    _DO    _Turb _Salnt_Depth_FLAG ");

    // close the file
    myFile1.close();

    // serial monitor
    Serial.println("done.");
  }
  else                                                // the file do not opened normally
  {
    // if the file didn't open, print an error
    Serial.print("[NewFile] ERROR opening ");      // display on serial monitor (Arduino
IDE)
    Serial.print(SDCard_FileName);
    Serial.println("!");

  }
}

void procedure_SDCard_NewFile_NewDay()
// input   : N/A
// output  : 1. new file that named with YYYYMMDD of NEXT DAY is created
//           2. there is 1 line of title in that file
// process : 1. open YYYYMMDD.TXT file
//           2. write the title in 1st line of that file
{
  Serial.println(">> procedure_SDCard_NewFile_NewDay()");

  // local variable
  int YYYY;
  int MM;
  int DD;
  char FileName[12];

  /* Define File Name */
  // initializing
  YYYY = tmYearToCalendar(tm.Year);                   // year of today
  MM = tm.Month;                                      // month of today
  DD = tm.Day + 1;                                    // date of NEXT DAY

  // define the file name of OLD FILE
  if ((MM < 10) && (DD < 10))                           // when (number of month < 10) AND
(number of date) < 10
  {
    sprintf(FileName, "%d0%d0%d.TXT", YYYY, MM, DD);  // assign FileName variable with
YYYYMMDD.TXT
  }
  else if ((MM < 10) && (DD >= 10))                     // when (number of month < 10) AND
(number of date) < 10
  {
    sprintf(FileName, "%d0%d%d.TXT", YYYY, MM, DD);   // assign FileName variable with
YYYYMMDD.TXT
  }
  else if ((MM >= 10) && (DD < 10))                     // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(FileName, "%d%d0%d.TXT", YYYY, MM, DD);   // assign FileName variable with
YYYYMMDD.TXT
  }
  else                                                  // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(FileName, "%d%d%d.TXT", YYYY, MM, DD);    // assign FileName variable with
YYYYMMDD.TXT
  }
```

```
  /* NEW FILE for SENSOR DATA LOGGING - YYYYMMDD.TXT */
  // open the file
  // note that only one file can be open at a time,
  // so we have to close this one before opening another.

  // open YYYYMMDD.TXT file that already exist
  // if it isn't, then create YYYYMMDD.TXT as new file
  myFile1 = SD.open(FileName, FILE_WRITE);

  // if the file opened okay, write to it:
  if (myFile1)                                    // file opened normally
  {
    // serial monitor
    Serial.print("Writing to ");
    Serial.print(FileName);
    Serial.print("...");

    /* title */
    // title of the content of file
    myFile1.println("YYYYMMDD_hhmmss_Suhu _pH    _DO    _Turb _Salnt_FLAG ");

    // close the file
    myFile1.close();

    // serial monitor
    Serial.println("done.");
  }
  else                                            // the file do not opened normally
  {
    // if the file didn't open, print an error
    Serial.print("[NewFile] ERROR opening ");     // display on serial monitor (Arduino
IDE)
    Serial.print(FileName);
    Serial.println("!");

  }
}

void procedure_SDCard_RemoveOldFile()
// input   : N/A
// output  : old files removed
// process : 1. assign FileName variable with name of old file (1 year before)
//           2. removing the old files
{
  Serial.println(">> procedure_SDCard_RemoveOldFile()");

  // local variable
  int YYYY;
  int MM;
  int DD;
  char FileName[12];

  /* Delete File Name */
  // initializing
  YYYY = tmYearToCalendar(tm.Year) - 1;               // 1 year BEFORE of today
  MM = tm.Month;                                      // month of today
  DD = tm.Day;                                        // date of today

  // define the file name of OLD FILE
  if ((MM < 10) && (DD < 10))                         // when (number of month < 10) AND
(number of date) < 10
  {
    sprintf(FileName, "%d0%d0%d.TXT", YYYY, MM, DD);  // assign FileName variable with
YYYYMMDD.TXT
  }
  else if ((MM < 10) && (DD >= 10))                   // when (number of month < 10) AND
(number of date) < 10
  {
    sprintf(FileName, "%d0%d%d.TXT", YYYY, MM, DD);   // assign FileName variable with
YYYYMMDD.TXT
  }
  else if ((MM >= 10) && (DD < 10))                   // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(FileName, "%d%d0%d.TXT", YYYY, MM, DD);   // assign FileName variable with
YYYYMMDD.TXT
```

```
  }
  else                                                  // when (number of month >= 10)
AND (number of date) < 10
  {
    sprintf(FileName, "%d%d%d.TXT", YYYY, MM, DD);     // assign FileName variable with
YYYYMMDD.TXT
  }

  // check the file before removed
  if (SD.exists(FileName))                              // checking, the old file is exist
or not, delete if exist
  {
    Serial.print("File ");
    Serial.print(FileName);
    Serial.println(" must be deleted!");
    // remove the old file
    SD.remove(FileName);                                // removing the old file
    // check that the file has removed or not
    if (SD.exists(FileName))                            // checking, the old file is STILL
exist or not
    {
      Serial.print("Deleting ");
      Serial.print(FileName);
      Serial.println(" FAILED!");

    }
    else
    {
      Serial.print("Deleting ");
      Serial.print(FileName);
      Serial.println(" COMPLETED!");

    }
  }
  else
  {
    Serial.println("No file need to be deleted!");
  }
}

/*---------------------------------( LED )-------------------------------------*/
void procedure_LED_SETUP()
// input   : N/A
// output  : 1. pin of LED are set as OUTPUT
//           2. value of each pin are assigned with 0
// process : 1. set all of pin for LED as OUTPUT
//           2. assign value of each pin with 0
{
  Serial.println(">> procedure_LED_SystemStatus_SETUP_All()");

  /* LED for DATA SENSOR */
  // set pin as output
  pinMode(LED_1G, OUTPUT);
  pinMode(LED_2G, OUTPUT);
  pinMode(LED_3G, OUTPUT);
  pinMode(LED_4G, OUTPUT);
  pinMode(LED_5G, OUTPUT);
  pinMode(LED_6G, OUTPUT);
  pinMode(LED_7G, OUTPUT);
  pinMode(LED_1R, OUTPUT);
  pinMode(LED_2R, OUTPUT);
  pinMode(LED_3R, OUTPUT);
  pinMode(LED_4R, OUTPUT);
  pinMode(LED_5R, OUTPUT);
  pinMode(LED_6R, OUTPUT);
  pinMode(LED_7R, OUTPUT);

  // assign pin value with 0
  digitalWrite(LED_1G, 0);
  digitalWrite(LED_2G, 0);
  digitalWrite(LED_3G, 0);
  digitalWrite(LED_4G, 0);
  digitalWrite(LED_5G, 0);
  digitalWrite(LED_6G, 0);
  digitalWrite(LED_7G, 0);
  digitalWrite(LED_1R, 0);
```

```
  digitalWrite(LED_2R, 0);
  digitalWrite(LED_3R, 0);
  digitalWrite(LED_4R, 0);
  digitalWrite(LED_5R, 0);
  digitalWrite(LED_6R, 0);
  digitalWrite(LED_7R, 0);
}



/*---------------------------------( LED )-----------------------------------*/
void procedure_Relay_SETUP()
// input   : N/A
// output  : 1. pin of LED are set as OUTPUT
//           2. value of each pin are assigned with 0
// process : 1. set all of pin for LED as OUTPUT
//           2. assign value of each pin with 0
{
  Serial.println(">> procedure_LED_SystemStatus_SETUP_All()");

  /* LED for DATA SENSOR */
  // set pin as output
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);

  // assign pin value with 0
  digitalWrite(Relay1, 1);
  digitalWrite(Relay2, 1);

}



/*---------------------------------( ETC )-----------------------------------*/

void procedure_ETC_Menu1()
{
  boolean entry = false;
  unsigned long stop_menu1 = millis() + 5000;
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("MENU LAIN-LAIN");
  lcd.setCursor(0,1);
  lcd.print("1. Ubah Waktu");
  lcd.setCursor(0,2);
  lcd.print("2. Mematikan Speaker");
  lcd.setCursor(0,3);
  lcd.print("3. Mengirim SMS");
  while ((entry == false)&&(stop_menu1 > millis()))
  {
    ETCkey = myKeypad.getKey();
    if (ETCkey == '1' || ETCkey == '2' || ETCkey == '3' || ETCkey == '4' || ETCkey ==
'5' || ETCkey == '*')
    {
      entry = true;
      ETC_Flag = 1;
    }
  }
  if(ETCkey == '*')
  {
    ETCkey = '0';
    mode = 'A';
  }
  lcd.clear();
}

void procedure_ETC_Menu2()
{
  boolean entry = false;
  unsigned long stop_menu2 = millis() + 5000;
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("MENU LAIN-LAIN");
  lcd.setCursor(0,1);
  lcd.print("4. Mengisi SD Card");
  lcd.setCursor(0,2);
```

```
  lcd.print("5. Start Kincir 1");
  lcd.setCursor(0,3);
  lcd.print("6. N/A");
  while ((entry == false)&&(stop_menu2 > millis()))
  {
    ETCkey = myKeypad.getKey();
    if (ETCkey == '1' || ETCkey == '2' || ETCkey == '3' || ETCkey == '4' || ETCkey ==
'5' || ETCkey == '*')
    {
      entry = true;
      ETC_Flag = 1;
    }
  }
  if(ETCkey == '*')
  {
    ETCkey = '0';
    mode = 'A';
  }
  lcd.clear();
}


void procedure_SensorData_ConditionCheck()
// input   : N/A
// output  : 1. NORMAL value when condition is normal, OR
//           2. NOT NORMAL value when condition is not normal
// process : 1. check every data sensor that received from RMP module
//           2. compare those value with standard value (SNI)
{
  //Serial.println(">> procedure SensorData ConditionCheck()");

  /* Standard-Value Based on Standar Nasional Indonesia (SNI) */
  // temperature (Celcius degree)
  float Temperature_Low  = 28.5;
  float Temperature_High = 31.5;

  // pH
  float pH_Low  = 7.5;
  float pH_High = 8.5;

  // DO (miligram per liter)
  float DO_Low = 3.5;

  // turbidity --- non SNI
  float Turbidity_Low  = 100;
  float Turbidity_High = 500;

  // salinity (gram per liter)
  float Salinity_Low  = 15;
  float Salinity_High = 25;


  /* Condition Check */
  // #1
  // sensor data of DO RPM1
  if (data_RPM1[0] >= DO_Low)
  {
    Status_DO = "OK";               // FLAG when DO NORMAL
    digitalWrite(LED_1G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_1R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM1[0] < DO_Low)
  {
    Status_DO = "--";
    digitalWrite(LED_1G, 0);  // turn OFF the GREEN LED
    digitalWrite(LED_1R, 1);  // turn ON the RED LED
  }

  // #2
    // sensor data of TEMPERATURE RPM 1
  if ((data_RPM1[1] >= Temperature_Low) && (data_RPM1[1] <= Temperature_High))
  {
    Status_Temperature = "OK";
    digitalWrite(LED_2G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_2R, 0);  // turn OFF the RED LED
```

```
  }
  // when NOT normal
  else if (data_RPM1[1] < Temperature_Low)
  {
    Status_Temperature = "--";
    digitalWrite(LED_2G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_2R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM1[1] > Temperature_High)
  {
    Status_Temperature = "++";
    digitalWrite(LED_2G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_2R, 1);  // turn OFF the RED LED
  }

  // #3
  // sensor data of PH
  if ((data_RPM1[2] >= pH_Low) && (data_RPM1[2] <= pH_High))
  {
    Status_pH = "OK";
    digitalWrite(LED_3G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_3R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM1[2] < pH_Low)
  {
    Status_pH = "--";
    digitalWrite(LED_3G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_3R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM1[2] > pH_High)
  {
    Status_pH = "++";
    digitalWrite(LED_3G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_3R, 1);  // turn OFF the RED LED
  }

  // #4
  // sensor data of TURBIDITY
  if ((data_RPM1[3] >= Turbidity_Low) && (data_RPM1[3] <= Turbidity_High))
  {
    Status_Turbidity = "OK";
    digitalWrite(LED_4G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_4R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM1[3] < Turbidity_Low)
  {
    Status_Turbidity = "--";
    digitalWrite(LED_4G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_4R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM1[3] > Turbidity_High)
  {
    Status_Turbidity = "++";
    digitalWrite(LED_4G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_4R, 1);  // turn OFF the RED LED
  }

  // #5
  // sensor data of SALINITY
  if ((data_RPM1[4] >= Salinity_Low) && (data_RPM1[4] <= Salinity_High))
  {
    Status_Salinity = "OK";
    digitalWrite(LED_5G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_5R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM1[4] < Salinity_Low)
  {
    Status_Salinity = "--";
    digitalWrite(LED_5G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_5R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM1[4] > Salinity_High)
  {
```

```
    Status_Salinity = "++";
    digitalWrite(LED_5G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_5R, 1);  // turn OFF the RED LED
  }

  // #6
  // sensor data of Temperature RPM2
  if ((data_RPM2[0] >= Temperature_Low) && (data_RPM2[0] <= Temperature_High))
  {
    Status_Temperature2 = "OK";
    digitalWrite(LED_6G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_6R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM2[0] < Temperature_Low)
  {
    Status_Temperature2 = "--";
    digitalWrite(LED_6G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_6R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM2[0] > Temperature_High)
  {
    Status_Temperature2 = "++";
    digitalWrite(LED_6G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_6R, 1);  // turn OFF the RED LED
  }

  // #7
  // sensor data of TURBIDITY RPM2
  if ((data_RPM2[1] >= Turbidity_Low) && (data_RPM2[1] <= Turbidity_High))
  {
    Status_Turbidity2 = "OK";
    digitalWrite(LED_7G, 1);  // turn ON the GREEN LED
    digitalWrite(LED_7R, 0);  // turn OFF the RED LED
  }
  // when NOT normal
  else if (data_RPM2[1] < Turbidity_Low)
  {
    Status_Turbidity2 = "--";
    digitalWrite(LED_7G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_7R, 1);  // turn OFF the RED LED
  }
  else if (data_RPM2[1] > Turbidity_High)
  {
    Status_Turbidity2 = "++";
    digitalWrite(LED_7G, 0);  // turn ON the GREEN LED
    digitalWrite(LED_7R, 1);  // turn OFF the RED LED
  }
}

void procedure_SendMessage_Content()
{
  // The SMS text you want to send
  GSM.println("RPM1");
  GSM.print("DO         : ");
  GSM.println(Status_DO);
  GSM.print("Suhu       : ");
  GSM.println(Status_Temperature);
  GSM.print("pH         : ");
  GSM.println(Status_pH);
  GSM.print("Kecerahan  : ");
  GSM.println(Status_Turbidity);
  GSM.print("Salinitas  : ");
  GSM.println(Status_Salinity);
  GSM.println("");
  GSM.println("RPM2");
  GSM.print("Suhu       : ");
  GSM.println(Status_Temperature2);
  GSM.print("Kecerahan  : ");
  GSM.println(Status_Turbidity2);
}

void procedure_SendValue_Content()
{
  // The SMS text you want to send
  GSM.println("RPM1");
```

```
  GSM.print("DO        : ");
  GSM.println(data_RPM1[0],1);
  GSM.print("Suhu      : ");
  GSM.println(data_RPM1[1],1);
  GSM.print("pH        : ");
  GSM.println(data_RPM1[2],1);
  GSM.print("Kecerahan : ");
  GSM.println(data_RPM1[3],1);
  GSM.print("Salinitas : ");
  GSM.println(data_RPM1[4],1);
  GSM.println("");
  GSM.println("RPM2");
  GSM.print("Suhu      : ");
  GSM.println(data_RPM2[0],1);
  GSM.print("Kecerahan : ");
  GSM.println(data_RPM2[1],1);
}
```