Sistem Inkubator
Menggunakan Mikrokontroler 16-bit H8/3069F

Skripsi diajukan sebagai salah satu syarat
untuk memperoleh gelar Sarjana Fisika

Oleh :

Muchtar Suhari Putra

0302020593

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS INDONESIA

DEPOK

2008

LEMBAR PENGESAHAN

Skripsi : Sistem Inkubator Menggunakan Mikrokontroler 16-bit H8/3069F

Nama : Muchtar Suhari Putra

NPM : 0302020593

Jurusan : Fisika

Peminatan : Instrumentasi Elektronika

TELAH DIPERIKSA DAN DISETUJUI :

Dr. Supriyanto
PEMBIMBING

Dr. Sastra Kusuma W.
PENGUJI I

Lingga Hermanto, M.Si
PENGUJI II

KATA PENGANTAR

Alhamdulillah, puji syukur penulis panjatkan ke hadirat Allah SWT, Yang Maha Sempurna Lagi Maha Kaya, karena berkat segala rahmat maupun nikmat-Nya, penulis dapat menyelesaikan laporan tugas akhir ini dengan baik. Shalawat serta salam senantiasa kita sanjungkan kepada Nabi Besar Muhammad SAW, serta kepada keluarga dan para sahabatnya. Semoga kita semua termasuk umat yang mendapatkan syafaat di hari kiamat nanti. Amin.

Laporan tugas akhir ini berjudul **Sistem Inkubator Menggunakan Mikrokontroler 16-bit H8/3069F** disusun sebagai salah satu syarat kelulusan dalam menyelesaikan studi S-1 Reguler di departemen Fisika, FMIPA, Universitas Indonesia. Dengan segala daya upaya dan kerja keras, serta berbagai tantangan dan rintangan yang penulis hadapi, maka inilah persembahan dari penulis sebuah laporan tugas akhir yang tentu saja masih jauh dari harapan dan sempurna. Namun demikian, penulis tidak lupa mengucapkan terima kasih kepada:

1. Orang tua penulis, Bapak Suhari dan Ibu Kumari, semoga Allah SWT selalu memberikan kesehatan dan rezeki yang melimpah ke Bapak dan Ibu.
2. Dr. Supriyanto – sebagai pembimbing dan penasehat – yang telah memberikan saran dan inspirasi kepada penulis, semoga Allah SWT membalas amal kebaikan yang telah Bapak lakukan.
3. Dr. Terry Mart – dosen mata kuliah Pendahuluan Mekanika Kuantum – terima kasih atas kalimat bijaknya, semoga Allah SWT meridhoi cita-cita Bapak.
4. Alfa & Tyo – *My forever best friend* – rezeki dan jodoh selalu di tangan Allah SWT, semoga Allah SWT selalu memberikan yang terbaik untuk kalian.
5. Mas Karno & Suhendro (2006) – *Important friend in important event* – jangan sungkan untuk meminta bantuan penulis, semoga Allah SWT memberikan ilmu yang luas kepada kalian.
6. Tante Sumijati Bawi & Tante Lili – *Support me since I was child untill now* – yang selalu membantu penulis baik materi maupun non-materi, semoga Allah SWT memberikan rezeki yang banyak untuk kalian.

7. Rekan-rekan seperjuangan pendukung penulis, Putu (02), Popo (02), Rizky (06), Firdy (Ekstensi 05), Ilham (02), Allan (03), Welly (04), Santiko (04) dan lain-lain yang tidak bisa penulis sebutkan satu persatu, karena memang penulis tidak sedang mengabsen.
8. Mba Ratna (*ucapan sangat terima kasih sekali ya, Mba!!*), Pak Dono (*Anda sangat baik sekali Pak!!*), Pak Katman (*terima kasih untuk pinjaman* black box-*nya*).
9. Rekan-rekan Lab CISCO Fisika, Mas Arif, Mas Dwi Seno, Bang Cepy, Gindo, terima kasih atas kesediaannya menyediakan tempat yang layak di lab (*akhirnya…!!!*).
10. Terakhir, untuk B-6287-BNK (*I love you*), Fluke, L-8400, Lab Cisco, H8/3069F, *Alfa Digital Oscilloscope*, dan peralatan maupun komponen lainnya yang tidak bisa disebutkan satu persatu (*ga cukup niy halamannya…!!*).

Akhir kata, semoga laporan tugas akhir ini dapat menjadi pembuka jalan maupun inspirasi bagi rekan-rekan mahasiswa Fisika Instrumentasi lainnya yang akan melaksanakan tugas akhir baik serupa seluruhnya atau sebagian dengan penulis. Jangan ragu dan sungkan untuk bertanya kepada penulis, karena dengan senang hati penulis akan membantu kalian tanpa dipungut biaya sepeser pun (*yah…minimal transport juga boleh lah…*^_^")

Depok, Juni 2008

Penulis

Sistem Inkubator Menggunakan Mikrokontroler 16-bit H8/3069F

(*Incubator System Using 16-bit Microcontroller H8/3069F*)

Muchtar Suhari Putra*

Departemen Fisika, FMIPA, Universitas Indonesia

Depok 16424

ABSTRAK

Suatu sistem inkubator dibuat untuk mengatasi masalah keterbatasan jumlah petugas yang memantau dengan cara menghubungkan beberapa inkubator ke dalam suatu. Sistem ini dilengkapi dengan sensor temperatur, *heater*, *blower*, serta mikrokontroler 16-bit H8/3069F. Kestabilan temperatur di dalam inkubator dikontrol dengan metode PID (*Proportional Integral Derrivative*). Agar dapat menggunakan program yang cukup banyak, perlu menggunakan sistem operasi. Sistem operasi yang dipakai menggunakan kernel linux sehingga pengoperasiannya mirip dengan komputer biasa.

Kata Kunci: PID, mikrokontroler 16-bit, sistem operasi, kernel

ABSTRACT

The incubator system made to solve problem of number medical staff which to monitor some incubator in local area network. This system was design with temperature sensors, heater, blower, and 16-bit microcontroller H8/3069F. Stability of incubator were control with PID method (Proportional Integral Derrivative). Since we use more task than one program, we must download an operating system. Operating system that we use include linux kernel, so the operation is similar with common personal computer.

Keywords: PID, microcontroller 16-bit, operating system, kernel

*email: muchtarsputra@gmail.com

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Hingga saat ini, berbagai peralatan elektronika mengalami perkembangan yang cukup pesat. Salah satu bidang yang mengalami kemajuan yang cukup berarti antara lain bidang elektronika medis/kedokteran. Seperti diketahui, bahwa peralatan medis merupakan peralatan yang memiliki teknologi relatif paling canggih di dunia selain peralatan militer. Alat-alat medis seperti MRI (*Magnetic Resonance Imaging*), CT (*Computed Tomography*) Scan, CMR (*Comprehensive Microbial Resource*), dsb, merupakan salah satu contoh peralatan berteknologi canggih.

Inkubator bayi (selanjutnya disebut inkubator saja) merupakan salah satu peralatan medis yang berfungsi untuk menjaga bayi berada dalam kondisi seperti pada saat bayi berada dalam kandungan. Namun, perkembangan teknologi inkubator tidak mengalami kemajuan yang berarti. Salah satu bentuk teknologi yang belum berkembang antara lain seperti pengendalian alat medis dari jarak jauh. Peralatan seperti MRI dapat dikendalikan menggunakan komputer dari jarak jauh – artinya, dokter tidak perlu berada terlalu dekat dengan pasien – melalui suatu jaringan komputer. Sedangkan inkubator masih dikendalikan dari jarak dekat, yaitu petugas medis melihat kondisi bayi secara langsung dari jarak dekat.

Masalah ini mungkin tidak terlalu berarti bila jumlah inkubator kurang dari 5 unit. Namun bagaimana jika jumlah inkubator lebih dari 5 unit? Apakah seorang

petugas medis dapat memantau seluruh inkubator secara bersamaan? Akan kurang efektif apabila seorang petugas medis bekerja untuk satu inkubator saja. Bagaimana bila inkubator berjumlah 10 unit? Selain membutuhkan jumlah petugas medis yang lebih banyak, juga akan menambah biaya sewa bagi orang tua bayi.

Untuk penelitian ini, dibuat sistem inkubator yang memiliki kemampuan untuk dapat dikendalikan melalui jarak jauh serta dapat mengendalikan inkubator lebih banyak. Sistem inkubator ini memiliki pengontrol suhu dengan menggunakan *PID Controller*, terhubung ke jaringan komputer secara langsung, serta memungkinkan untuk terhubung ke internet agar orang tua atau kerabat lainnya dapat melihat kondisi bayi dari mana saja tanpa harus datang ke rumah sakit.

Perkembangan mikrokontroller saat ini semakin mendekati kemampuan komputer *desktop*. Karena pada perkembangannya saat ini kapasitas memori, RAM, EEPROM, serta jumlah bit semakin meningkat sehingga dapat dimasukkan sistem operasi seperti pada komputer. Salah satu hasil dari perkembangan ini antara lain mikrokontroller 16-bit H8/3069F yang termasuk ke dalam keluarga H8/300S yang memiliki kapasitas ROM sebesar 512 kb dan RAM 16 kb. Karena memiliki ROM berkapasitas 512 kb sehingga kita dapat memasukkan sistem operasi yang berukuran file sangat kecil seperti MicroCLinux atau RTOS yang berbasiskan bahasa C.

1.2 Tujuan

Tujuan penelitian ini antara lain:

a. Membuat sistem inkubator yang lebih baik serta terhubung ke jaringan komputer, sehingga pemantauan dapat dilakukan secara *online*.

b. Memahami dan mempelajari cara kerja H8/3069F baik *hardware* maupun *software*.

c. Mempelajari PID (*Proportional Integral Derivative*) sebagai pengatur temperatur dalam inkubator.

d. Menghubungkan beberapa mikrokontroler dalam suatu jaringan LAN (*local area network*) menggunakan protokol TCP/IP.

e. Mempelajari *operating system* yang dapat dipergunakan pada mikrokontroler.

1.3 Pembatasan Masalah

Pada inkubator terdapat *heater* dan *blower*. *Heater* terdiri atas beberapa buah lampu yang mampu menghasilkan panas. Sedangkan *blower* terdiri atas beberapa *fan* berukuran kecil yang mampu menghasilkan angin yang cukup untuk mengalirkan dan mengurangi panas di dalam inkubator. Penelitian inkubator ini difokuskan pada teknik menghubungkan beberapa mikrokontroler dengan sebuah komputer yang berfungsi sebagai server pemantau data. Sehingga, untuk dapat terhubung melalui *ethernet*, mikrokontroler harus berisi *operating system* yang berisi program untuk dapat berkomunikasi dalam LAN, dengan bahasa pemrograman yang digunakan pada mikrokontroler ini adalah bahasa C.

1.4 Metodologi Penelitian

Metode penelitian yang telah dilakukan terdiri atas beberapa tahap antara lain :

a. Studi Peralatan

Studi peralatan bertujuan untuk mempelajari karakteristik dan spesifikasi alat yang akan digunakan dalam pembuatan inkubator agar diperoleh teori-teori dasar sebagai sumber penulisan skripsi.

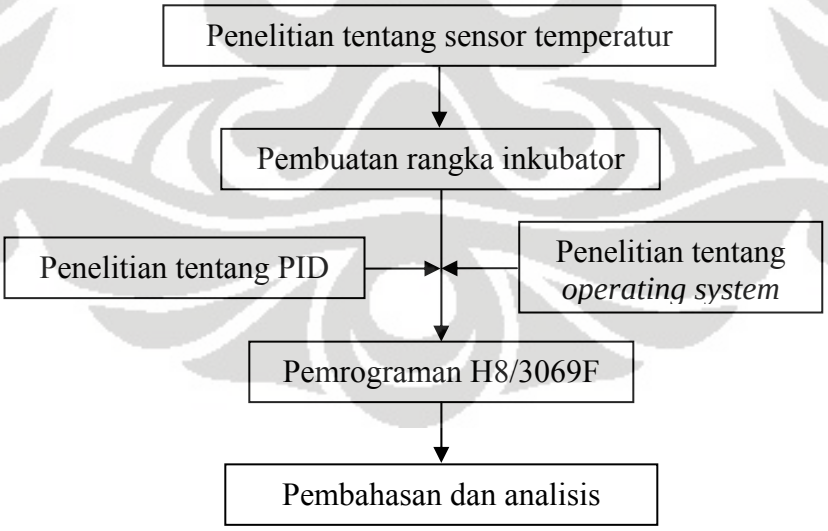
b. Studi kepustakaan

Studi kepustakaan dilakukan untuk memperoleh informasi dan pustaka yang berkaitan dengan masalah ini baik dari literatur, penjelasan yang diberikan dosen pembimbing, internet dan buku-buku yang berhubungan dengan tugas akhir penulis.

c. Penelitian Laboratorium

Penelitian laboratorium dilakukan untuk merakit, membuat alat dan meneliti kerja alat yang telah dibuat.

Berikut ini *flowchart* kegiatan penelitian:



Gambar 1.1 Flowchart langkah-langkah penelitian

Penjelasan diagram alir penelitian :

1. Penelitian tentang sensor temperatur

   Sensor temperatur yang digunakan adalah LM35 keluaran *National Semiconductor*. Sensor yang digunakan sebanyak 2 (dua) unit.

2. Pembuatan rangka inkubator

   Pembuatan rangka inkubator berbahan *acrylic* transparan yang bertujuan agar inkubator lebih ringan, tahan benturan, dan tahan pada temperatur hingga 60°C serta harganya yang lebih terjangkau.

3. Penelitian tentang PID

   Penelitian tentang PID berguna untuk mencari persamaan matematika yang tepat dengan sistem inkubator, sehingga akan didapatkan nilai kontanta maupun variabel yang lebih baik agar sistem dapat bekerja sesuai dengan keinginan.

4. Penelitian tentang *operating system*

   Penelitian tentang *operating system* bertujuan untuk memilih sistem operasi yang cocok dan ukuran file yang lebih kecil sehingga sesuai dengan kapasitas memori yang tersedia. Selain itu, penggunaan sistem operasi antara lain agar mempermudah sistem komunikasi melalui *ethernet* sehingga tidak memerlukan peralatan tambahan pada mikrokontroler.

5. Pemrograman H8/3069F

   Pemrograman yang dilakukan meliputi keseluruhan program yang akan digunakan, seperti program pembacaan temperatur, dan sistem operasi.

6. Pembahasan dan analisis

Hasil dari seluruh kegiatan penelitian yang telah dilakukan akan dibahas dan dianalisis untuk mendapatkan hasil yang lebih baik.

1.5 Sistematika Penulisan

Sistematika penulisan skripsi ini terdiri atas 5 bab yang secara garis besar dapat diuraikan sebagai berikut:

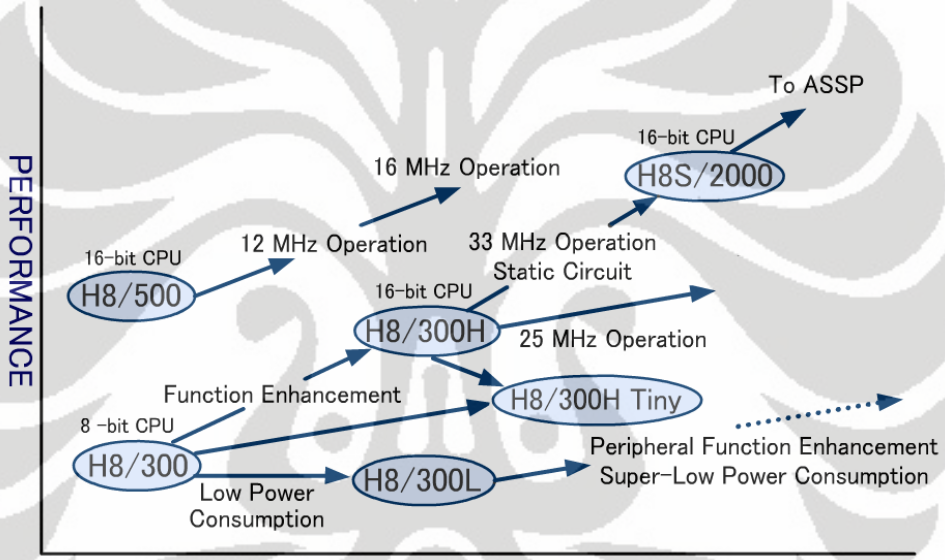
- ✓ Bab 1 Pendahuluan, membahas tentang latar belakang dari penelitian, tujuan, metode penelitian yang digunakan, dan juga pembatasan masalah pada penelitian yang dilakukan.
- ✓ Bab 2 Teori Dasar, membahas secara garis besar teori dasar yang berhubungan dengan penelitian.
- ✓ Bab 3 Perangkat Keras dan Perangkat Lunak, membahas khusus pada perangkat keras dan perangkat lunak yang digunakan untuk membuat sistem inkubator bayi.
- ✓ Bab 4 Analisis Hasil Penelitian, berisi penjelasan mengenai hasil penelitian mikrokontroler H8/3069F dan juga analisis dari sistem kontrol inkubator yang telah dibuat.
- ✓ Bab 5 Kesimpulan dan Saran, berisi kesimpulan-kesimpulan penelitian, dan juga saran-saran yang mendukung penelitian ini agar memberikan hasil yang lebih baik lagi di masa yang akan datang.

BAB 2

TEORI DASAR

2.1 Perkembangan Mikrokontroler H8/3069F

Seri H8/300H adalah *single-chip microcomputer* (memiliki kemampuan dasar seperti komputer yang terintegrasi dalam satu chip tunggal) berkemampuan tinggi yang mempunyai sebuah CPU *(Central Processing Unit)* 16-bit sebagai prosessornya.

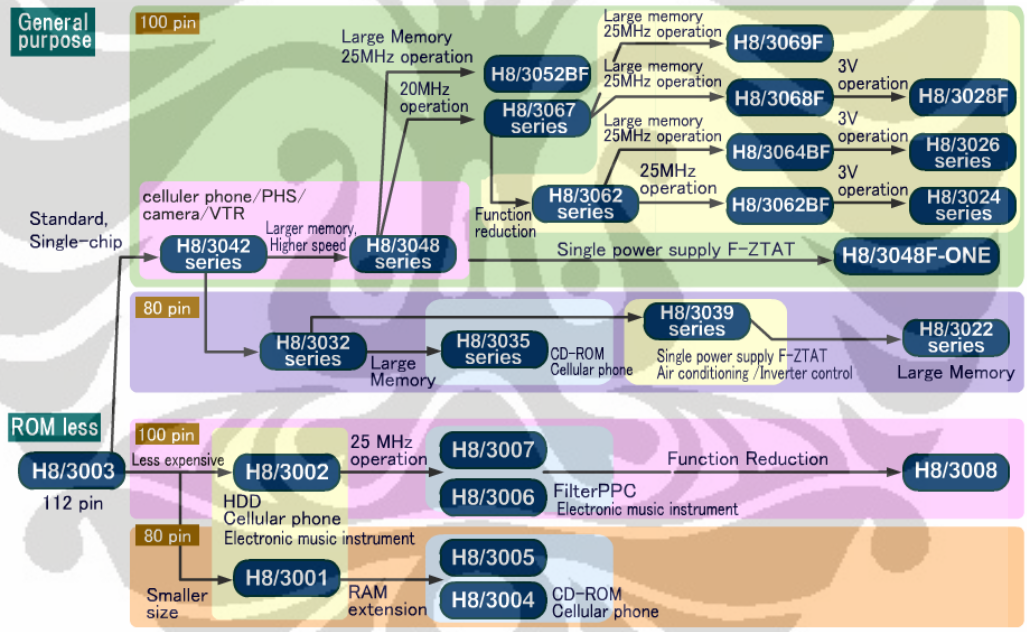


Gambar 2.1 Perkembangan mikrokontroler H8

Gambar 2.1 menunjukan perkembangan seri H8 serta performansinya. Seri H8 dibagi menjadi dua grup. Grup pertama adalah seri H8/500, yang menggunakan CPU 16-bit. Grup kedua adalah seri H8/300, yang menggunakan CPU 8-bit. Produk dari seri ini umumnya digunakan sebagai *single-chip microcomputer*. Seri H8/300H yang merupakan pengembangan dari H8/300, menggunakan CPU 16-bit dan

memungkinkan koneksi memori hingga 16 MByte. Produk dari seri ini tidak hanya digunakan sebagai *single-chip microcomputer*, tetapi juga dapat digunakan sebagai *multi-chip microcomputer* yang memungkinkan penambahan memori eksternal.

Seri H8S/2000 dikembangkan dengan menambah instruksi pada seri H8/300H. H8S/2000 lebih stabil dan memungkinkan operasi berkecepatan tinggi hingga 33 MHz. Pada daftar H8 juga terdapat seri H8/300L yang merupakan versi *low power consumption* dari seri H8/300. Selain itu, juga terdapat seri H8/300H Tiny yang merupakan versi *compact* dari seri H8/300H. Mikrokontroler seri H8/300H ini umumnya digunakan sebagai sarana edukasi perguruan tinggi di Jepang.



Gambar 2.2 Perkembangan mikrokontroler yang menggunakan mikroprosesor H8/300H

Gambar 2.2 menunjukkan perkembangan dari seri H8/300H dan bidang aplikasi serta fitur-fiturnya. Produk pertama dari seri H8/300H adalah H8/3003.

H8/3003 tidak bisa digunakan sebagai *single-chip microcomputer* karena tidak memiliki ROM *(Read Only Memory)* di dalamnya. H8/3042 mempunyai ROM dan bisa digunakan sebagai *single-chip microcomputer*. H8/3069F adalah versi pengembangan dari pendahulunya. H8/3069F telah memiliki memori yang diperbesar untuk peningkatan kecepatan. FZTAT mengindikasikan bahwa *flash* memori telah terintegrasi di dalamnya. Tipe *flash* memori ini adalah EEPROM yang berarti dapat dihapus secara elektrik dan dapat diprogram kembali. *Register-register* seperti pada telepon selular atau alat-alat lainnya telah tertulis di *flash* memori. Karena *flash* memori adalah ROM, maka data yang disimpan tidak akan hilang walaupun catu dayanya telah dimatikan. Pada pengembangannya terdapat berbagai macam produk dengan fungsi, kapasitas memori, harga dan kecepatan operasi yang beragam.

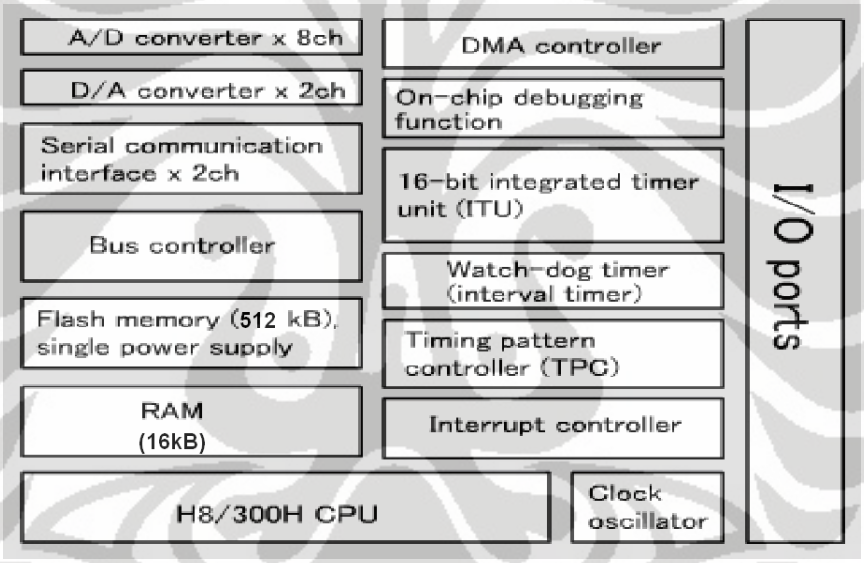
H8/3069F adalah salah satu mikrokontroler yang menggunakan prosesor H8/300H. H8/3069F mempunyai sebuah *writable flash* memori internal yang menggunakan catu daya tunggal 5 V. Perbedaan dengan keluarga mikrokontroler H8 yang memakai prosesor H8/300H lainnya adalah kapasitas ROM, RAM dan fitur-fiturnya. Namun, pada umumnya fitur-fitur pada keluarga mikrokontroler H8 serupa. Gambar 2.3 menunjukkan fitur-fitur pada mikrokontroler H8/3069F.

FITUR-FITUR CPU

CPU 16-bit yang berperan sebagai *general-purpose register* dilengkapi dengan 16-bit x 16 *general-purpose register* dan tersedia juga dalam 8-bit x 8 + 16-bit x 16 atau 32-bit x 8. Mikrokontroler H8/3069F memiliki CPU berkecepatan tinggi. Frekuensi maksimumnya adalah 25 MHz. Waktu eksekusi perintah

penambahan atau pengurangan dilakukan dalam waktu 80 ns (*nano-second*) dan perkalian atau pembagian dieksekusi dalam waktu 560 ns.

Operasi CPU berbasis pada sinyal *clock,* semakin tinggi frekuensi sinyal *clock* maka semakin cepat operasinya. Waktu dari pulsa sinyal *clock* 25 Mhz adalah 0,04 μs (40 ns), yang disebut "1 *state*". Penambahan atau pengurangan diselesaikan dalam "2 *state*" sedangkan perkalian atau pembagian diselesaikan dalam "14 *state*". Selain itu CPU ini juga dilengkapi dengan ruang *address* (terdapat 8 area, yaitu CS0 – CS7) maksimum sebesar 16 MB.



Gambar 2.3 Fitur-fitur mikrokontroler H8/3069F

FUNGSI SEBAGAI *SINGLE-CHIP* DAN *MULTI-CHIP*

Mikrokontroler ini dapat difungsikan sebagai *single-chip microcomputer* karena tersedia internal ROM, RAM dan fungsi I/O pada CPU. Mikrokontroler ini juga dapat difungsikan sebagai *multi-chip microcomputer* saat terjadi penambahan memori.

ROM INTERNAL

Mikrokontroler ini mempunyai *flash memory* 384 KB, namun jika kita mengubah mode pengoperasian dapat dicapai hingga 512 KB yang dapat di operasikan dengan sebuah catu daya 5 V.

RAM INTERNAL

Mikrokontroler ini mempunyai RAM internal 16 KB dan pada mode pengoperasian tertentu dapat memcapai kapasitas hingga 2 MB.

I/O PORT

I/O *port* dapat digunakan sebagai masukan status *on*/*off* atau masukan sinyal dari berbagai sensor. Saat I/O *port* digunakan sebagai keluaran, mikrokontroler dapat diatur untuk mengontrol kedipan lampu atau mengontrol saklar *on*/*off* dari motor atau pemanas.

SCI (*SERIAL COMMUNICATION INTERFACE*) INTERNAL

Terdapat 3 kanal SCI internal dan ketiganya mempunyai fungsi yang sama. Mode dari SCI ini adalah sinkron dan asinkron. SCI mikrokontroler ini juga mempunyai komunikasi multiprosesor dengan dua atau lebih prosesor. SCI juga dapat dihubungkan dengan *smart card interface* dengan mengubah *setting* pada *register* CPU.

TIMER INTERNAL

Terdapat 3 kanal *timer* internal 16-bit dan 4 kanal *timer* internal 8-bit. Kanal 0 dan kanal 1 pada *timer* 16-bit mempunyai fungsi yang sama, sedangkan kanal 2 mempunyai *register* sendiri pada CPU. *Timer* 8-bit dibagi menjadi dua grup dengan masing-masing dua kanal. Grup 0 terdiri dari kanal 0 dan kanal 1, dan grup 1 terdiri dari kanal 2 dan kanal 3.

TPC (*TIMMING PATTERN CONTROLLER*) INTERNAL

H8/3069F mempunyai TPC yang menyediakan keluaran pulsa dengan berbasis *timer* 16-bit. Pulsa keluaran dari TPC dibagi menjadi grup 4-bit (grup 3 sampai grup 0) yang dapat beroperasi secara serempak dan independen.

WDT (*WATCH DOG TIMER*) INTERNAL

WDT dapat dioperasikan untuk mengawasi jalannya program, atau hanya sebagai interval *timer*. Ketika WDT digunakan, WDT akan membangkitkan sinyal *reset* pada *chip* H8/3069F bila sistem *crash.*

ADC (*ANALOG TO DIGITAL CONVERTER)* INTERNAL

H8/3069F memiliki ADC internal 8 kanal dengan resolusi 10-bit. 8 kanal masukan analog dibagi menjadi dua grup yaitu grup 0 dan grup 1. AV_{CC} dan AV_{SS} adalah catu daya sirkuit analog pada ADC, dan V_{REF} adalah tegangan referensi.

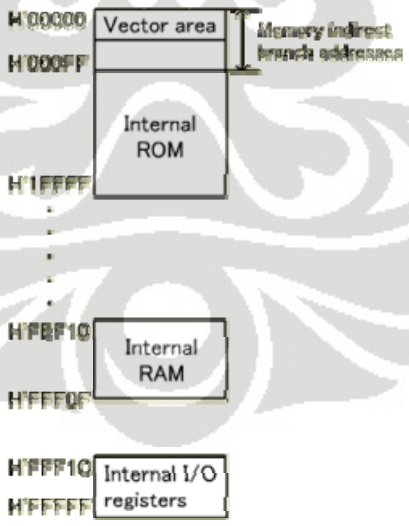
DAC (*DIGITAL TO ANALOG CONVERTER*) INTERNAL

H8/3069F memiliki DAC internal 2 kanal dengan resolusi 8-bit. Tegangan keluarannya berkisar antara 0 V sampai V_{REF}. Pengaturan DAC ini diatur pada sebuah *register* di CPU.

DMAC INTERNAL

Empat kanal DMAC digunakan untuk transfer data berkecepatan tinggi. DMAC memungkinkan transfer data lebih cepat dari penggunaan CPU. Umumnya digunakan dengan sebuah *timer* dan fungsi komunikasi lainnya.

H8/3069F dapat digunakan sebagai mikrokomputer *single-chip*. Pada kondisi ini, hanya memori internal yang dapat digunakan. Gambar 2.4 menunjukkan memori map pada mode *single-chip*. Pada mode ini alamat memori diekspresikan dengan notasi 5 digit heksadesimal.



Gambar 2.4 Memori *map* mode *single-chip*

2.2 Sistem Operasi

Sebuah sistem operasi (disingkat OS – *Operating System*) merupakan komponen software dari sistem komputer yang bertanggung jawab untuk mengatur dan mengkoordinasikan segala aktifitas dan berbagi *resources* komputer. OS bekerja seperti host untuk program aplikasi yang berjalan pada mesin. Sebagai host, salah satu tujuan dari OS adalah untuk menangani secara detil pengoperasian *hardware*. Hal ini dapat mengurangi beban program aplikasi untuk menangani secara detil pengoperasian *hardware* dan menjadi lebih mudah dalam membuat aplikasi. Hampir semua komputer, termasuk komputer jenis *hand-held*, komputer *desktop*, superkomputer, dan beberapa konsol *video game* modern, menggunakan OS jenis tertentu.

Embedded System

Embedded system adalah tujuan khusus sistem komputer yang didesain untuk melakukan satu atau lebih fungsi tertentu, yang bergantung pada komputasi *real-time*. Biasanya *embedded system* merupakan bagian peralatan yang lengkap termasuk hardware dan bagian mekanik. Jelasnya, komputer untuk tujuan umum, seperti *personal computer* (PC), dapat melakukan tugas yang berbeda tergantung dari program yang tersedia. *Embedded system* dapat mengatur lebih banyak peralatan yang biasa dipakai saat ini.

Sejak *embedded system* digunakan secara resmi untuk tugas tertentu, para engineer optimis, hal ini dapat mengurangi ukuran dan biaya produksi, atau

meningkatkan daya tahan dan performa sistem. Beberapa *embedded system* diproduksi massal, sesuatu hal yang menguntungkan bila dilihat dari skala ekonomi.

Secara fisik, *embedded system* digunakan mulai dari peralatan *portable* seperti jam digital dan MP3 *Player*, hingga instalasi besar seperti lampu lintas, *controller* pabrik, atau sistem kontrol pada *plant* reaktor nuklir. Kompleksitasnya bervariasi mulai dari *single chip microcontroller* hingga multiple unit, peralatan dan jaringan yang terletak di dalam chassis besar.

Secara umum, "*embedded system*" bukan arti secara pasti, seperti kebanyakan sistem yang memiliki beberapa elemen yang dapat diprogram. Sebagai contoh, komputer *hand-held* membagi beberapa elemen dengan *embedded system* – seperti sistem operasi dan mikroprosesor yang mendukungnya – tetapi tidak benar-benar *embedded system*, karena hanya mengizinkan aplikasi yang berbeda untuk dijalankan dan peralatan yang dihubungkan.

Embedded system menjangkau segala aspek di kehidupan modern dan ada banyak contoh yang mereka gunakan.

Sistem telekomunikasi menggunakan banyak *embedded system* mulai dari *switch* telepon untuk jaringan hingga *mobile phone* untuk tingkat pengguna. Jaringan komputer menggunakan router *dedicated* dan *network bridge* untuk me-route data.

Peralatan elektronik termasuk di dalamnya PDA, MP3 *Player*, *mobile phone*, konsol *video game*, kamera digital, DVD Player, penerima GPS, dan printer. Banyak peralatan rumah tangga, seperti *oven microwave*, mesin cuci dan mesin cuci piring, sudah menyertakan *embedded system* untuk menyediakan fleksibilitas, *feature-feature* dan efisiensi. Sistem HVAC modern menggunakan jaringan thermostat untuk

menghasilkan kontrol temperatur yang lebih akurat dan efisien yang dapat berubah setiap hari dan musim. *Home automation* menggunakan kabel dan jaringan *wireless* yang digunakan untuk mengontrol cahaya, iklim, keamaan, audio/visual, dsb, semua yang menggunakan peralatan *embedded* untuk sensor dan kontrol.

Penggunaan *embedded system* pada sistem transportasi mulai dari pesawat hingga *automobile* mengalami peningkatan. Pesawat baru terdiri atas peralatan *avionic* modern seperti *inertial guidance system* dan penerima GPS yang juga amat dibutuhkan untuk masalah keselamatan. Berbagai motor listrik – DC motor *brushless*, motor induksi, dan motor DC – menggunakan kontroler motor listrik/elektronik. Automobil, kendaraan listrik, dan kendaraan *hybrid* mengalami peningkatan penggunaan *embedded system* untuk memaksimalkan efisiensi dan mengurangi polusi. Sistem keamaan otomotif lainnya seperti *anti-lock braking system* (ABS), *Electronic Stability Control* (ESC/ESP), and *automatic four-wheel drive*.

Peningkatan peralatan medis berlanjut dengan menggunakan lebih banyak *embedded system* untuk monitoring tanda-tanda vital, stestoskop elektronik untuk memperkuat bunyi, dan berbagai pencitraan medis (PET, SPECT, CT, MRI) untuk pemeriksaan internal tanpa gangguan.

CPU platforms

Embedded prosesor dapat ditentukan menjadi 2 kategori: mikroprosesor biasa (μP) dan mikrokontroler (μC), yang memiliki lebih banyak pendukung pada chip, mengurangi biaya dan ukuran. Perbedaan dengan pasar PC dan server, penggunaan arsitektur CPU hampir mendekati angka yang cukup besar; diantaranya Von

Neumann yang sederajat dengan arsitektur Harvard, RISC yang sama baiknya dengan yang non-RISC dan VLIW, dengan panjang kata bervariasi mulai dari 4-bit hingga 64-bit dan diluar itu (terutama untuk prosesor DSP) meskipun jenis umumnya adalah 8/16-bit.

Beberapa daftar arsitektur pada umumnya, meskipun tidak banyak namun cukup mewakili diantaranya: 65816, 65C02, 68HC08, 68HC11, 68k, 8051, ARM, AVR, Blackfin, C167, Coldfire, COP8, eZ8, eZ80, FR-V, H8, HT48, M16C, M32C, MIPS, MSP430, PIC, PowerPC, R8C, SHARC, ST6, SuperH, TLCS-47, TLCS-870, TLCS-900, Tricore, V850, x86, XE8000, Z80, dsb.

PC/104 dan PC/104+ merupakan contoh *board* komputer yang tersedia untuk kebutuhan kecil-kecilan. Biasanya menggunakan DOS, Linux, NetBSD, atau sistem operasi real-time seperti MicroC/OS-II, QNX or VxWorks.

Embedded system berhubungan dengan dunia luar menggunakan jalur komunikasi, seperti:

- ✓ *Serial Communication Interfaces* (SCI): RS-232, RS-422, RS-485, dll.
- ✓ *Synchronous Serial Communication Interface*: I2C, JTAG, SPI, SSC dan ESSI.
- ✓ *Universal Serial Bus* (USB).
- ✓ Jaringan: Ethernet, Controller Area Network, LonWorks, dsb.
- ✓ Timers: PLL, *Capture/Compare and Time Processing Units*
- ✓ Discrete IO: aka *General Purpose Input/Output* (GPIO)
- ✓ *Analog to Digital/Digital to Analog* (ADC/DAC)

Seperti software yang lainnya, desainer *embedded system* menggunakan *compiler*, *assembler*, and *debugger* untuk membangun software *embedded system*.

Arsitektur Embedded software

Ada beberapa tipe arsitektur software yang umum digunakan, antara lain:

1. *Simple control loop*

   Untuk desain jenis ini, secara sederhana software memiliki *loop*. Loop memanggil subrutin, yang masing-masing mengatur bagian dari hardware atau software.

2. Sistem Kontrol *Interrupt*

   Beberapa embedded system terutama memiliki kontrol *interrupt*. Artinya tugas tersebut dilakukan oleh sistem yang dipicu pada jenis kejadian yang berbeda-beda. Sebuah *interrupt* akan dihasilkan salah satunya oleh timer yang frekuensinya telah diatur sebelumnya, atau oleh port serial saat menerima data.

   Sistem jenis ini digunakan jika penanganan kejadian membutuhkan *latency* yang rendah dan penanganan kejadian yang mudah dan pendek.

   Biasanya sistem ini menjalankan tugas yang sederhana pada bagian loop, tetapi tugas ini tidak sensitif pada *delay* yang tidak diinginkan.

   Kadang-kadang penanganan *interrupt* yang ditambahkan cukup panjang pada struktur antrian. Kemudian, setelah penanganan *interrupt* selesai, tugas ini selanjutnya dieksekusi oleh loop utama. Metode ini membuat sistem mendekati kernel *multitasking* dengan proses diskrit.

3. *Cooperative multitasking*

   Sistem *multitasking nonpreemptive* sangat mirip dengan skema *simple control loop*, kecuali bagian loop-nya disembunyikan pada sebuah API.

Programer mendefinisikan urutan tugas, dan masing-masing tugas mendapatkan lingkungannya masing-masing untuk dijalankan. Kemudian, ketika tugas ini tidak jalan, biasanya akan dipanggil rutin tertentu (biasa disebut "*pause*", "*wait*", "*yield*", "*nop*" (*Stands for no operation*), etc.).

Keuntungan dan kerugiannya mirip dengan *control loop*, kecuali pada saat penambahan software yang menjadi lebih mudah, mudah dalam menulis tugas baru atau menambahkan *queue-interpreter*.

4. *Preemptive multitasking* atau *multi-threading*

Sistem jenis ini, bagian level rendah dari kode diganti antara tugas atau *thread* yang berdasarkan pada timer (yang terhubung ke sebuah *interrupt*). Level inilah yang secara umum dianggap memiliki kernel sistem operasi. Tergantung dari seberapa banyak fungsi yang dibutuhkan, hal ini memperkenalkan lebih banyak atau sedikit konsep paralel yang berjalan dalam mengatur lebih banyak tugas yang kompleks.

Karena kode apapun dapat berpotensi untuk merusak data pada *task* lain (kecuali pada sistem lebih besar yang menggunakan MMU) program harus didesain dan diuji dengan hati-hati, dan akses ke data yang dibagi harus dikontrol oleh suatu strategi sinkronisasi, seperti antrian pesan, *semaphore* atau skema sinkronisasi tanpa penghalang.

Karena kompleksitas ini, adalah hal biasa untuk organisasi membeli sistem operasi *real-time*, memungkinkan programer aplikasi untuk berkonsentrasi pada fungsi alat daripada layanan sistem operasi, paling tidak untuk sistem skala besar; sistem dengan skala lebih kecil seringkali tidak

mampu mengatasi masalah yang berhubungan dengan sistem *real-time* generik, karena pembatasan yang berhubungan ukuran memori, performa, dan atau umur baterai.

5. *Microkernel* dan *exokernel*

Mikrokernel adalah langkah maju dari sistem operasi *real-time*. Pengaturan umumnya adalah sistem operasi kernel mengalokasikan memori dan memindahkan CPU ke pelaksanaan *thread* yang berbeda. Mode pengguna memproses implementasi fungsi utama seperti sistem file, antarmuka jaringan, dan lain-lain.

Secara umum, mikrokernel berhasil ketika perpindahan *task* dan komunikasi antar *task* cukup cepat, dan gagal ketika mereka berjalan dengan lambat.

Eksokernel berkomunikasi secara efisien oleh *normal subroutine calls.* Perangkat keras, dan semua perangkat lunak dalam sistem juga tersedia, dan dapat diperluas oleh programer aplikasi.

6. *Monolithic kernel*

Pada kasus ini, kernel dengan ukuran relatif besar dengan kemampuan khusus diadaptasi untuk menyesuaikan dengan lingkungan *embedded.* Hal ini memberi programer lingkungan yang serupa dengan sistem operasi *desktop* seperti Linux atau Microsoft Windows, dan oleh karena itu sangat produktif untuk pembangunan; tapi di sisi lain membutuhkan sumber perangkat keras lebih banyak, seringkali lebih mahal, dan karena kompleksitas dari kernel ini maka menjadi tidak mudah diprediksi dan tidak dapat diandalkan.

Contoh umum dari *embedded monolithic kernels* adalah Embedded Linux dan Windows CE. Walaupun ada peningkatan biaya dari segi perangkat keras, jenis sistem *embedded* ini popularitasnya semakin meningkat, khususnya pada alat *embedded* yang lebih bertenaga seperti router *wireless* dan sistem navigasi GPS. Berikut ini adalah beberapa alasannya: *Port* untuk *chip embedded* yang umum sudah tersedia. *Port* ini membolehkan penggunaan ulang kode publikasi untuk *driver* alat, *web server, firewall,* dan kode lainnya.

Pembangunan sistem dapat dimulai dengan set fitur yang lebar, dan kemudian distribusi dapat dikonfigurasi untuk meniadakan fungsi yang tidak diperlukan, dan menyimpan penggunaan memori yang mungkin akan dikonsumsi.

Banyak insinyur percaya bahwa menjalankan kode aplikasi dalam mode pengguna akan lebih handal, lebih mudah untuk di-*debug* dan oleh karena itu proses pembangunan lebih mudah dan kode akan semakin *portable.*

Banyak sistem *embedded* kekurangan kebutuhan *real-time* yang ketat dari sistem kontrol. Sistem seperti *Embedded Linux* mempunyai respon yang cukup cepat untuk banyak aplikasi.

Fitur yang membutuhkan respon lebih cepat yang dapat dijamin seringkali dapat diletakan di perangkat keras.

Banyak sistem RTOS memiliki biaya per unit. Ketika menggunakan produk berupa komoditas, maka biaya akan menjadi signifikan.

7. Customisasi Sistem Operasi

Bagian kacil dari *embedded system* membutuhkan keamanan, tepat waktu, dan dapat dipercaya atau justru tidak mendapatkan efisiensi sama sekali pada salah satu arsitektur yang disebutkan sebelumnya. Pada kasus ini, suatu organisasi perlu membangun sistem untuk meregulasi.. Pada beberapa keadaan, sistem mungkin akan dipartisi ke dalam "mekanisme kontroler" menggunakan teknik khusus, dan "*display controller*" dengan sistem operasi konvensional.

8. Komponen Software Tambahan

Sebagai tambahan untuk inti sistem operasi, banyak *embedded system* yang memiliki tambahan komponen software pada lapisan atas layer. Komponen ini terdiri atas protokol jariangan seperti TCP/IP, FTP, HTTP, and HTTPS, dan juga termasuk kemampuan penyimpanan seperti FAT dan sistem manajemen *flash memory*. Jika peralatan *embedded* memiliki kemampuan audio dan video, maka driver dan codec yang tepat perlu dipasang pada sistem. Pada kategori *monolithic kernel*, feature-feature sudah disertakan. Pada kategori RTOS, software tambahan tersedia tergantung pada penawaran secara komersial.

RTOS

Real-time operating system (RTOS) adalah sistem operasi multitasking yang ditujukan untuk aplikasi real-time. Contoh aplikasinya termasuk *embedded system* (termostat yang dapat diprogram, pengendali alat-alat rumah tangga, telepon selular), robot industri, pesawat luar angkasa, alat pengendali industri dan alat-alat riset sains.

RTOS memudahkan pembuatan sistem *real-time,* tapi tidak menjamin hasil akhirnya akan menjadi *real-time*; hal ini membutuhkan penyusunan program yang tepat. RTOS tidak perlu memiliki *throughput* yang tinggi, tapi RTOS menyediakan fasilitas yang jika digunakan dengan baik, menjamin dapat mencapai *deadline* secara umum *(soft real-time* atau *hard real-time).* RTOS biasanya menggunakan algoritma penjadwalan khusus dengan tujuan memberikan *real-time developer* alat-alat yang diperlukan untuk menghasilkan *deterministic behavior* pada sistem akhir. RTOS mendapat nilai lebih untuk seberapa cepat dapat merespon kejadian khusus daripada jumlah kerja yang dapat dilakukan. Faktor kunci RTOS adalah *interrupt latency* minimal dan *thread switching latency* minimal.

Contoh awal dari sistem operasi *real-time* skala besar adalah Fasilitas Pemroses Transaksi *(Transaction Processing Facility)* yang dibangun oleh American Airlines dan IBM untuk sistem reservasi Sabre Airline.

Filosofi desain

Dua desain dasar yang ada:

- *Event-driven* (penjadwalan prioritas) mendesain tugas penggantian hanya jika *event* dengan prioritas lebih tinggi butuh pelayanan, disebut dengan prioritas *pre-emptive.*
- *Time-sharing* mendesain tugas penggantian pada *clock interrupt,* dan pada *events,* disebut *round robin.*

Desain CPU awal membutuhkan banyak siklus untuk berpindah tugas dan selama itu CPU tidak dapat melakukan hal yang bermanfaat. Jadi sistem operasi awal mencoba untuk meminimalisasi waktu CPU yang terbuang dengan memaksimalkan pencegahan tugas penggantian yang tidak perlu.

CPU saat ini membutuhkan waktu yang lebih sedikit untuk berpindah dari satu *task* ke *task* lainnya; kasus ekstrim adalah prosesor *barrel* yang dapat berpindah dari satu *task* ke *task* lainnya dalam nol siklus. RTOS yang baru hampir tanpa kecuali mengimplementasikan penjadwalan *time-sharing* dengan prioritas penjadwalan *driven pre-emptive.*

1. Penjadwalan

Dalam desain umum, *task* memiliki tiga keadaan: 1) *running,* 2) *ready,* 3) *blocked.* Kebanyakan *task* adalah *blocked.* Hanya satu *task* yang berjalan tiap CPU. Dalam sistem yang lebih sederhana, *ready list* biasanya pendek, paling banyak hanya dua atau tiga *task.*

Kuncinya adalah mendesain *scheduler.* Biasanya struktur data *ready list* dalam *scheduler* didesain untuk meminimalisasi panjangnya waktu yang dihabiskan dalam kondisi kritis *scheduler.* Tapi, pemilihan struktur data juga bergantung pada jumlah maksimum *task* yang dapat berada pada *ready list.*

Jika tidak pernah ada lebih dari beberapa *task* pada *ready list*, maka *list* sederhana yang terhubung pada *ready task* mungkin akan optimal. Jika *ready list* biasanya hanya berisi beberapa *task* tapi terkadang berisi lebih, maka *list* harus diurutkan berdasarkan prioritas, jadi untuk menemukan *task* dengan prioritas tertinggi tidak perlu mencari di seluruh *list.* Untuk memasukan sebuah *task* perlu menjalankan

ready list hingga mencapai akhir dari *list,* atau *task* dengan prioritas lebih rendah dari *task* yang akan dimasukan.

Waktu respon kritis, terkadang disebut waktu *flyback,* adalah waktu yang dibutuhkan untuk mengantri *ready task* yang baru dan mengembalikan kondisi *task* dengan prioritas tertinggi. Pada RTOS yang didesain dengan baik, mempersiapkan *task* yang baru akan membutuhkan 3-20 instruksi tiap antrian, dan pemulihan *task* dengan prioritas tertinggi akan membutuhkan 5-30 instruksi. Pada prosesor 68000 20MHz, perpindahan *task* berjalan sekitar 20 mikro detik dengan dua *task* yang siap. CPU ARM 100MHz berpindah *task* dalam beberapa mikro detik.

Pada sistem *real-time* tingkat tinggi, *real-time task* berbagi sumber komputasi dengan banyak *task* yang tidak *real-time,* dan *ready list* dapat berubah panjangnya. Dalam sistem seperti itu, suatu *scheduler ready list* yang diimplementasikan sebagai *list* yang terhubung tidak akan cukup.

2. Algoritma

Beberapa algoritma penjadwalan RTOS yang umum adalah:

- *Cooperative scheduling*
    - *Round-robin scheduling*
    - *Fixed priority pre-emptive scheduling*
- *Pre-emptive scheduling*
    - *Pre-emptive time slicing*
    - *Critical section pre-emptive scheduling*
    - *Static time scheduling*
- *Earliest deadline first approach*

- *Advanced scheduling using the stochastic and MTG*

3. Komunikasi *intertask* dan pembagian sumber

Sistem *multitasking* harus mengatur pembagian data dan sumber perangkat keras di antara berbagai *task.* Biasanya ”tidak aman” untuk dua *task* untuk mengakses data spesifik yang sama atau sumber perangkat keras secara bersamaan. (”Tidak aman” berarti hasilnya tidak konsisten atau tidak dapat diprediksi, khususnya ketika *task* berada di tengah-tengah perubahan koleksi data. Pandangan oleh *task* yang lain terbaik dilakukan sebelum perubahan dimulai atau sesudah perubahan selesai.) Terdapat tiga pendekatan yang umum untuk mengatasi problem ini:

- Menonaktifkan *interrupt* untuk sementara waktu
- *Binary semaphores*
- *Message passing*

Kegunaan sistem operasi biasanya tidak membolehkan program pengguna untuk menonaktifkan *interrupt,* karena program pengguna dapat mengendalikan CPU selama yang diinginkan. CPU modern membuat bit kontrol non-aktif *interrupt* tidak dapat diakses pada mode pengguna untuk mengizinkan sistem operasi mencegah pengguna melakukan hal ini. Banyak *embedded system* dan RTO, mengizinkan aplikasi itu sendiri untuk beroperasi dalam mode kernel untuk efisiensi sistem yang lebih baik dan juga untuk memberi izin pada aplikasi untuk mendapat kontrol yang lebih besar pada lingkungan operasi tanpa membutuhkan intervensi sistem operasi.

Pada sistem dengan prosesor tunggal, jika aplikasi dijalankan pada mode kernel dan menutup *interrupt,* seringkali solusi terbaik adalah untuk mencegah akses bersamaan ke sumber yang dibagi. Sementara *interrupt* tertutup, *task* yang sedang

berjalan mendapat hak ekslusif penggunaan CPU; tidak ada *task* atau *interrupt* lain yang pegang kendali, jadi bagian kritis efektif terlindungi. Ketika *task* keluar dari bagian kritisnya, *task* itu harus membuka *interrupt;* menunda *interrupt*, jika ada, lalu kemudian mengeksekusinya. Penutup *interrupt* sementara seharusnya hanya dilakukan ketika jalan terpanjang melalui bagian kritis lebih pendek daripada *interrupt latency* maksimum yang diinginkan, atau kalau tidak metode ini akan meningkatkan *interrupt latency* maksimum sistem. Biasanya metode proteksi ini digunakan ketika bagian kritis hanya beberapa baris *source code* panjangnya dan tidak berisi *loop.* Metode ini ideal untuk perlindungan perangkat keras *bitmapped registers* ketika bit dikontrol oleh *task* yang berbeda.

Ketika bagian kritis lebih panjang dari beberapa baris *source code* atau mencakup *looping* yang panjang, seorang *programmer* harus menggunakan mekanisme yang identik atau sama dengan yang tersedia pada sistem operasi kegunaan umum, seperti *semaphores* dan *OS-supervised interprocess messaging.* Mekanisme tersebut mencakup *system calls,* jadi mereka dapat menangani ratusan instruksi CPU untuk dilaksanakan, sementara penutup *interrupt* dapat mengambil sebanyak tiga instruksi pada beberapa prosesor. Tapi, untuk bagian kritis yang lebih panjang, tidak ada pilihan; *interrupt* tidak dapat ditutupi untuk waktu yang lama tanpa meningkatkan *interrupt katency* sistem.

Binary semaphore dapat dikunci atau tidak dikunci. Ketika dikunci, suatu antrian *task* dapat menunggu *semaphore.* Biasanya suatu *task* dapat mengatur *timeout* untuk waktu tunggu *semaphore.* Permasalahan pada desain dengan *semaphore* telah banyak dikenal: inversi prioritas dan *deadlock.*

Pada inversi prioritas, *task* dengan prioritas tinggi menunggu karena *task* dengan prioritas rendah memiliki *semaphore.* Solusi umum adalah dengan menjalankan *task* yang memiliki *semaphore* pada *task* yang memiliki prioritas menunggu tertinggi. Tapi pendekatan sederhana ini gagal ketika terdapat level menunggu yang berbeda-beda (A menunggu *binary semaphore* yang dikunci B, dimana B menunggu *binary semaphore* yang dikunci C).

Pada *deadlock,* dua atau lebih *task* mengunci sejumlah *binary semaphore* dan kemudian menunggu selamanya untuk *binary semaphore* lainnya, menghasilkan grafik siklus ketergantungan. Skenario *deadlock* paling sederhana terjadi ketika dua *task* mengunci dua *semaphore* dalam suatu *lockstep,* tapi dengan urutan yang berlawanan. *Deadlock* biasanya dicegah dengan desain yang hati-hati, atau dengan *floored semaphore* (dimana mengoper kontrol dari *semaphore* ke *task* dengan prioritas lebih tinggi pada kondisi yang telah ditentukan).

Pendekatan lain untuk pembagian sumber adalah dengan pengiriman pesan oleh *task.* Dalam paradigma ini, sumber secara langsung diatur hanya oleh satu *task;* ketika *task* lain ingin menggunakan sumber, *task* lain tersebut mengirimkan pesan kepada *task* pengatur. Paradigma ini mendapat masalah yang serupa dengan *binary semaphore*: inversi prioritas terjadi ketika suatu *task* bekerja pada pesan dengan prioritas rendah, dan mengabaikan pesan dengan prioritas lebih tinggi (atau pesan yang secara tidak langsung berasal dari *task* prioritas tinggi) pada *inbox*-nya. Protokol *deadlock* terjadi ketika dua atau lebih *task* saling menunggu satu sama lain untuk mengirim pesan respon.

Walaupun kelakuan *real-time* mereka lebih singkat daripada sistem *semaphore*, sistem dengan basis pesan sederhana biasanya tidak memiliki protokol bahaya *deadlock*, dan secara umum berlaku lebih baik daripada sistem *semaphore.*

4. Penanganan interrupt dan penjadwalan

Karena *interrupt handler* menghadang *task* prioritas tertinggi untuk berjalan, dan karena sistem operasi *real-time* didesain untuk menjaga *thread latency* tetap minimum, *interrupt handlers* biasanya dijaga sependek mungkin. *Interrupt handler* menangguhkan segala interaksi dengan perangkat keras selama mungkin; biasanya yang penting adalah mengakui atau menonaktifkan *interrupt* (jadi hal tersebut tidak terjadi lagi ketika *interrupt handler* kembali). *Interrupt handler* kemudian mengantri kerja yang perlu diselesaikan pada level prioritas yang lebih rendah, seringkali dengan membuka penghalang *driver task* (dengan melepaskan *semaphore* atau mengirimkan pesan). *Scheduler* seringkali menyediakan kemampuan untuk membuka penghalang suatu *task* dari konteks *interrupt handler.*

5. Alokasi memori

Alokasi memori pada RTOS jauh lebih kritis daripada sistem operasi lain. Pertama, kecepatan dari alokasi adalah penting. Skema alokasi memori standar meneliti *list* untuk mencari blok memori kosong yang cocok; bagaimanapun, hal ini tidak dapat diterima karena alokasi memori harus terjadi dalam waktu yang telah ditentukan di RTOS.

Kedua, memori dapat menjadi ter-fragmen karena daerah bebas menjadi terpisah dengan daerah yang digunakan. Hal ini dapat menyebabkan program terhenti, tidak mendapatkan memori, walaupun secara teori masih tersedia. Algoritma

alokasi memori yang memperlambat pengumpulan *fragmentation* mungkin dapat berjalan baik untuk mesin *desktop* namun tidak dapat diterima untuk *embedded system* yang seringkali berjalan tahunan tanpa proses *reboot.*

Algoritma *fixed-size-blocks* bekerja dengan baik untuk sistem *embedded* sederhana.

Kekuatan lain dari blok ukuran tetap adalah untuk sistem DSP khususnya dimana satu inti menyelenggarakan satu bagian saluran dan bagian selanjutnya dikerjakan oleh inti lainnya. Dalam kasus ini, manajemen *buffer* ukuran tetap dengan satu inti mengisi *buffer* dan set inti lainnya mengembalikan *buffer* dengan sangat efisien. RTOS yang DSP-nya dioptimalkan seperti sistem operasi Unison atau DSPnano RTOS menyediakan fitur ini.

2.3 Sensor Temperature LM35

Sensor temperatur LM35 merupakan *integrated circuit temperature sensor* yang cukup presisi, mudah dikalibrasi, dimana output yang dihasilkan dari sensor ini adalah tegangan yang dapat dihubungkan langsung ke ADC atau *interface* lainnya untuk mendapatkan nilai yang diinginkan. Prinsip kerjanya hanya berdasarkan output yang dihasilkan oleh dua terminal zener.

Sensor ini memiliki tegangan *breakdown* yang sama dengan temperatur absolut pada 10 mV/°C. Sensor ini bekerja pada kisaran temperatur 0°C – 100°C, dengan harga yang cukup murah dan banyak di pasaran.

2.4 Analog to Digital Converter (ADC)

Sebelum masuk ke mikrokontroler, sinyal analog dari pengkondisi sinyal dikonversi terlebih dahulu menjadi data digital oleh ADC *(Analog to Digital Converter)*. ADC merupakan bagian yang cukup penting pada sistem inkubator ini karena menentukan nilai keluaran dari sensor LM35.

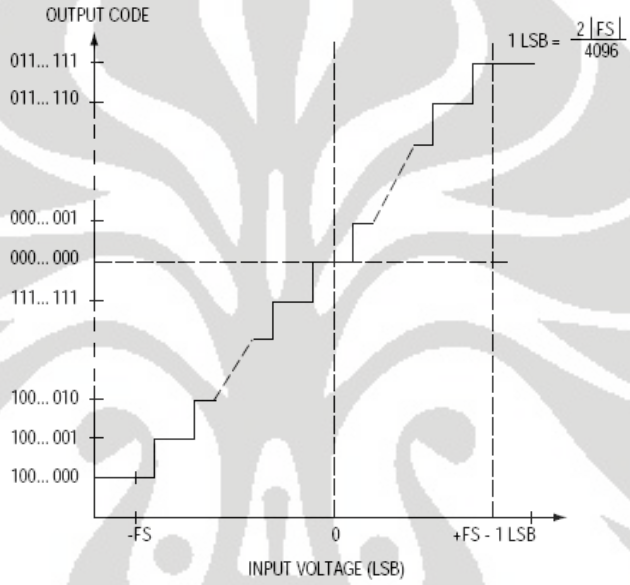
Mikrokontroler H8/3069F memiliki ADC internal 10-bit 8 kanal. Namun, ADC ini hanya memiliki jangkauan masukan dari 0 sampai 5 V. Karena pemrograman ADC internal belum berhasil dilakukan, maka ADC internal ini tidak digunakan. Sebagai gantinya digunakan ADC MAX128 buatan Maxim.

ADC MAX128 ini memiliki 8 kanal masukan dan resolusi sebesar 12-bit serta dapat menerima masukan tegangan positif dan negatif. Kanal ADC yang digunakan adalah kanal 0 dan kanal 1. Kanal 0 ADC dihubungkan dengan pengkondisi sinyal kanal 1 dan kanal 1 ADC dihubungkan dengan pengkondisi sinyal kanal 2. Sebagai tegangan referensi ADC digunakan rangkaian *buffer* yang menghasilkan tegangan referensi sebesar 5,12 V yang dihubungkan dengan pin REF yaitu pin 23 pada IC ADC.

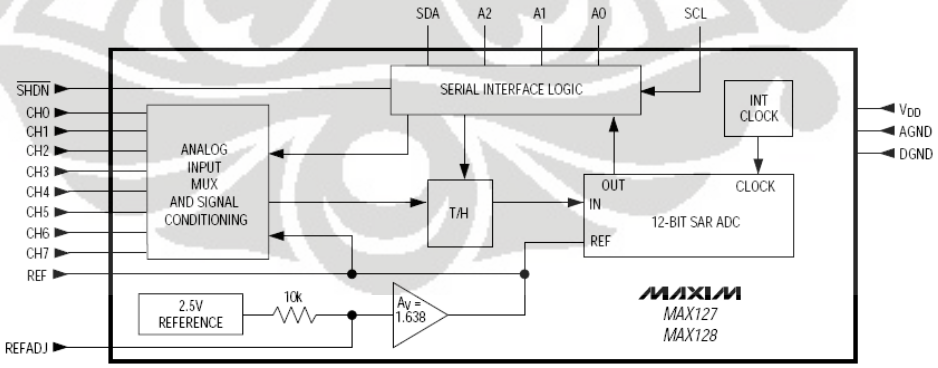
$$LSB = \frac{2|FS|}{4096} \qquad (2.1)$$

Dari persamaan 2.1 di atas dapat diketahui resolusi ADC sebesar 2,5 mV setiap *step*-nya. Artinya, bila ADC menunjukkan angka 4, maka angka tersebut merupakan hasil konversi dari sinyal analog yang besarnya 10 mV. Nilai tegangan positif akan muncul dengan angka 0 sampai dengan 2047 dan nilai tegangan negatif akan muncul dengan angka 2048 sampai dengan 4095.

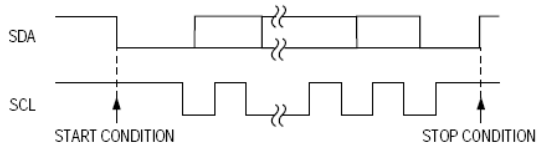
Komunikasi antara ADC dengan mikrokontroler dilakukan dengan metode I^2C. I^2C adalah kependekan dari *Inter-Integrated Circuit*. Metode yang ditemukan oleh Philips ini digunakan untuk menghubungkan komponen berkecepatan rendah ke mikrokontroler atau *motherboard.*



Gambar 2.14 Fungsi transfer bipolar



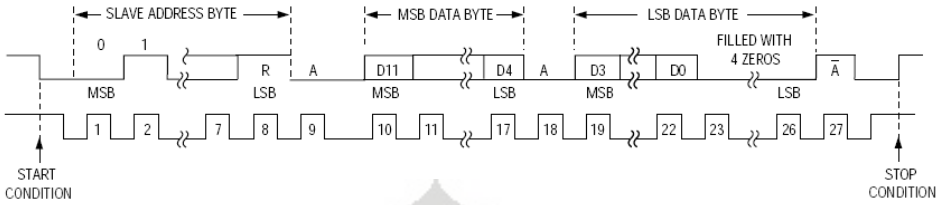
Gambar 2.15 Blok diagram ADC MAX128



Gambar 2.16 Kondisi *start* dan *stop*

Gambar 2.16 menunjukkan kondisi *start* dan *stop* transmisi data dengan metode I^2C. Kondisi *start* terjadi ketika SDA bertransisi dari kondisi *high* ke *low* ketika SCL dalam kondisi *high.* Kondisi *stop* terjadi ketika SDA bertransisi dari *low* ke *high* ketika SCL dalam kondisi *high.* Kondisi SDA hanya dapat berubah ketika SCL sedang *low*, kecuali untuk kondisi *start* dan *stop.*

Akses data dimulai dengan *master* memberikan kondisi START yang diikuti dengan 7-bit *address* dan sebuah bit *read*. Setelah bit ke delapan diterima dan alamatnya cocok, maka *slave* mengeluarkan *acknowledge* dengan mengubah SDA menjadi *low* untuk satu siklus *clock* (A=0) diikuti dengan *byte* pertama dari data serial (D11-D4, MSB terlebih dahulu). Setelah mengirim *byte* pertama, *slave* menunggu *master* untuk mengeluarkan *acknowledge* (A=0). Setelah menerima *acknowledge*, *slave* mengirim *byte* kedua (D3-D0 dan empat nol) diikuti dengan *NOT acknowledge* dari *master* yang mengindikasikan bahwa *byte* data terakhir telah diterima. Pada akhirnya, *master* mengeluarkan kondisi STOP untuk menghentikan siklus baca.



Gambar 2.17 *Timing Waveforms*

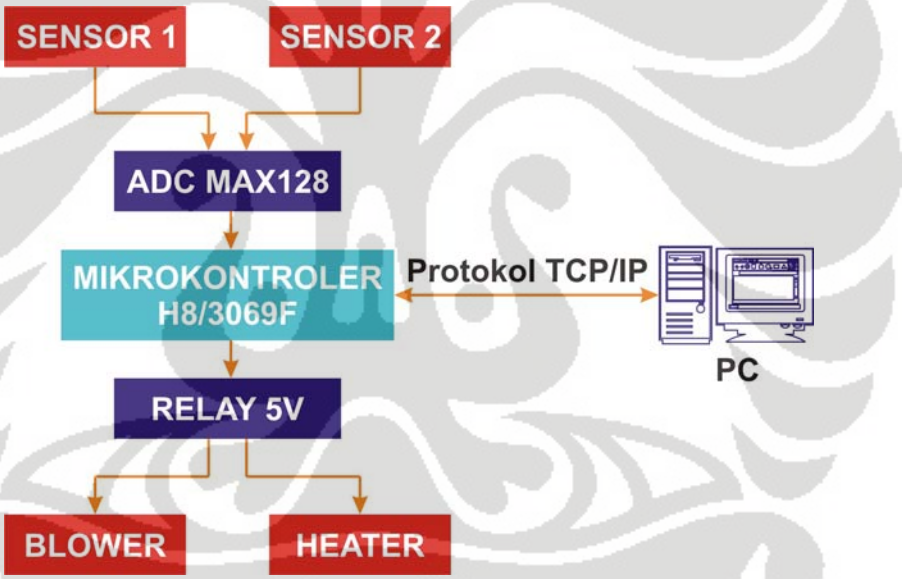
Salah satu spesifikasi ADC yang menentukan adalah waktu konversi, yaitu waktu yang dibutuhkan untuk mengkonversi sinyal analog menjadi digital. Sebuah ADC dengan waktu konversi yang singkat akan lebih baik untuk digunakan walaupun harganya menjadi lebih mahal. ADC MAX128 memiliki waktu konversi maksimum sebesar 10 μs.

BAB 3

PERANCANGAN ALAT

3.1 Perangkat Keras

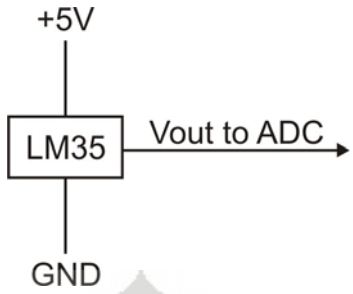
Dalam pembuatan alat secara keseluruhan, sebelumnya setiap rangkaian telah dirancang dalam bentuk beberapa blok. Hal ini bertujuan agar lebih mudah dalam merancang sistem, mudah dalam mencari kesalahan yang mungkin terjadi, serta lebih mudah menyesuaikan bila terjadi pergantian atau penambahan blok rangkaian. Gambar 3.1 menunjukkan blok diagram sistem inkubator.



Gambar 3.1 Blok diagram sistem inkubator

3.1.1 Rangkaian Sensor

Untuk mendeteksi temperatur di dalam inkubator, sistem ini menggunakan 2 buah sensor LM35 yang masing-masing diletakkan di bagian tengah atas inkubator. Hasil pembacaan dari sensor ini selanjutnya dikonversi ke ADC dengan tegangan referensi 5 V.



Gambar 3.2 Rangkaian dasar sensor temperatur

Kemudian hasil konversi dari ADC tadi akan diolah dengan menggunakan metode kontrol PID (*proportional integral derivative*), agar dapat mengatur input dari sensor dan dikeluarkan pada *heater* dan *blower*.

3.1.2 Rangkaian ADC

Komponen ADC yang digunakan adalah ADC MAX128 yang memiliki resolusi 12-bit. Dengan tegangan referensi 5 V, maka tegangan yang mampu dibaca tiap bitnya adalah

$$Vout = \frac{5}{2^{12}} = 0{,}001 \ volt$$

ADC MAX 128 memiliki 8 channel input yang dapat digunakan secara bersamaan. Pada sistem inkubator ini, hanya digunakan channel 0 dan 1.

Output dari ADC merupakan rangkaian I^2C, sehingga terdapat 2 port yang digunakan untuk berkomunikasi yaitu port untuk sinyal *clock* yang dihubungkan dengan port 1.0 pada mikrokontroler dan port untuk sinyal data yang dihubungkan dengan port 1.1 pada mikrokontoler.

3.1.3 Rangkaian Mikrokontroler

Sesuai dengan yang ditulis pada bab 2, mikrokontroler yang digunakan yaitu H8/3069F buatan Renesas Hitachi.

Port yang digunakan antara lain port 1 dan 2. Port 1 digunakan khusus untuk input dari ADC dan port 2 khusus digunakan untuk *output* ke *heater* dan *blower*.

Tegangan input untuk mikrokontroler sebesar +5 V, yang juga dihubungkan sebagai tegangan input untuk sensor LM35.

3.1.4 Rangkaian Pengatur Temperatur

Sistem inkubator ini menggunakan rangkaian *blower* yang terdiri atas 2 unit fan 9V/0,15 A dan rangkaian *heater* yang terdiri atas 3 unit lampu berdaya 100W. Temperatur akan dikontrol dengan metode PID, sehingga untuk mencapai temperatur yang diinginkan cukup dengan mengatur nyala *blower* dan *heater*.

3.1.4.1 Rangkaian *Blower*

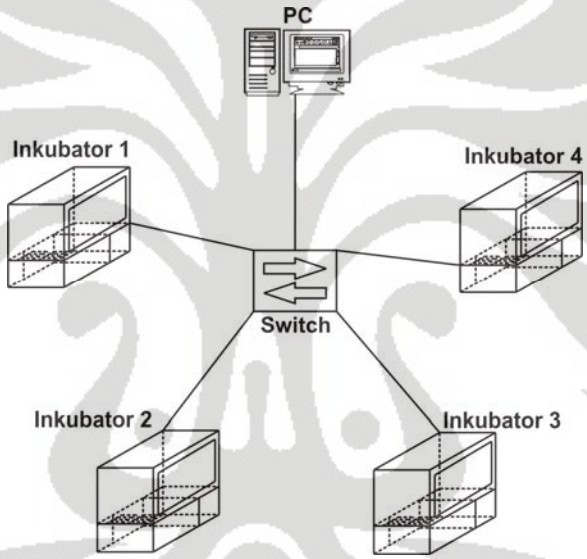
Pengaturan nyala blower menggunakan PWM (*Pulse Width Modulation*), yaitu dengan mengatur lebar pulsa *high* dan *low* sehingga kecepatan putar *blower* dapat diatur sedemikian rupa. Sinyal PWM memiliki frekuensi yang sama namun dengan *duty cycle* yang berubah-ubah. Kecepatan putaran ditentukan dengan mengatur *duty cycle* dari PWM tersebut.

3.1.4.2 Rangkaian *Heater*

Pengaturan nyala heater juga menggunakan metode PWM, sehingga lampu dapat diatur intensitas cahayanya, mulai dari mati, redup, sedang hingga menyala terang.

Untuk menghubungkan antara mikrokontroler yang menggunakan tegangan input +5V DC dengan lampu yang menggunakan input 220V AC, dibutuhkan rangkaian driver untuk menjembatani perbedaan tegangan tersebut. Untuk itu penulis menggunakan komponen *optocoupler* yang memiliki kemampuan memisahkan tegangan yang berbeda nilai yang cukup signifikan.

3.1.5 Rangkaian Komunikasi TCP/IP



Gambar 3.3 Skema Rancangan Jaringan LAN Sistem Inkubator

Pada mikrokontroler H8/3069F, terdapat modul *ethernet* sehingga tidak perlu penambahan modul lainnya seperti pada jenis mikrokontroler yang lain. Hal ini disebabkan karena mikrokontroler H8/3069F didesain agar mendukung port *ethernet* dengan menambahkan IC Realtek RTL8019AS, yang sudah umum digunakan pada teknologi *ethernet* baik pada komputer maupun peralatan lain yang mendukung penggunaan port *ethernet*.

3.2 Perangkat Lunak

3.2.1 Mikrokontroler H8/3069F

Perangkat lunak pada mikrokontroler menggunakan bahasa C, yang dikategorikan sebagai bahasa *mid-level*, karena mudah dipahami baik oleh mesin (dalam hal ini mikrokontroler) maupun oleh manusia (mudah untuk dipelajari).

Agar port *ethernet* pada mikrokontoler dapat digunakan, program yang digunakan haruslah berjenis *operating system* (OS) yang berbeda dengan pemrograman umum pada mikrokontroler. Sebagaimana diketahui, agar lebih efisien, program yang akan digunakan oleh mikrokontroler cukup program yang memang memiliki tujuan khusus. Misalnya, ketika kita memerlukan program untuk membaca sensor temperatur dan menampilkannya di LCD, maka cukup kita buat program yang memang hanya untuk membaca sensor dan menampilkannya di LCD.

Namun, ketika kita menggunakan port *ethernet*, hal ini berarti, selain program yang digunakan untuk membaca sensor temperatur, kemudian digunakan untuk mengendalikan *heater* dan *blower*, juga diperlukan untuk melakukan komunikasi melalui port *ethernet* secara *real-time*. Hal ini agak sulit dilakukan bila menggunakan program dengan tujuan khusus tadi (meskipun mungkin saja dapat dilakukan, namun cukup rumit).

Penambahan OS akan memudahkan penambahan program-program lain yang diperlukan. Cara kerja OS pada mikrokontroler sama dengan cara kerja OS pada komputer, yaitu menjalankan fungsi dasar *hardware* secara maksimal. Ketika kita membutuhkan program tertentu untuk melakukan tugas tertentu, cukup dengan

menambahkan program tambahan tersebut, tanpa perlu mengetahui cara kerja *hardware* secara lebih detail.

Sebelum memasukkan OS ke dalam mikrokontroler, terlebih dahulu dimasukkan program Redboot untuk melakukan inisialisasi alamat hardware mikrokontroler. Redboot sendiri berisi perintah untuk melakukan instalasi OS ke dalam mikrokontroler.

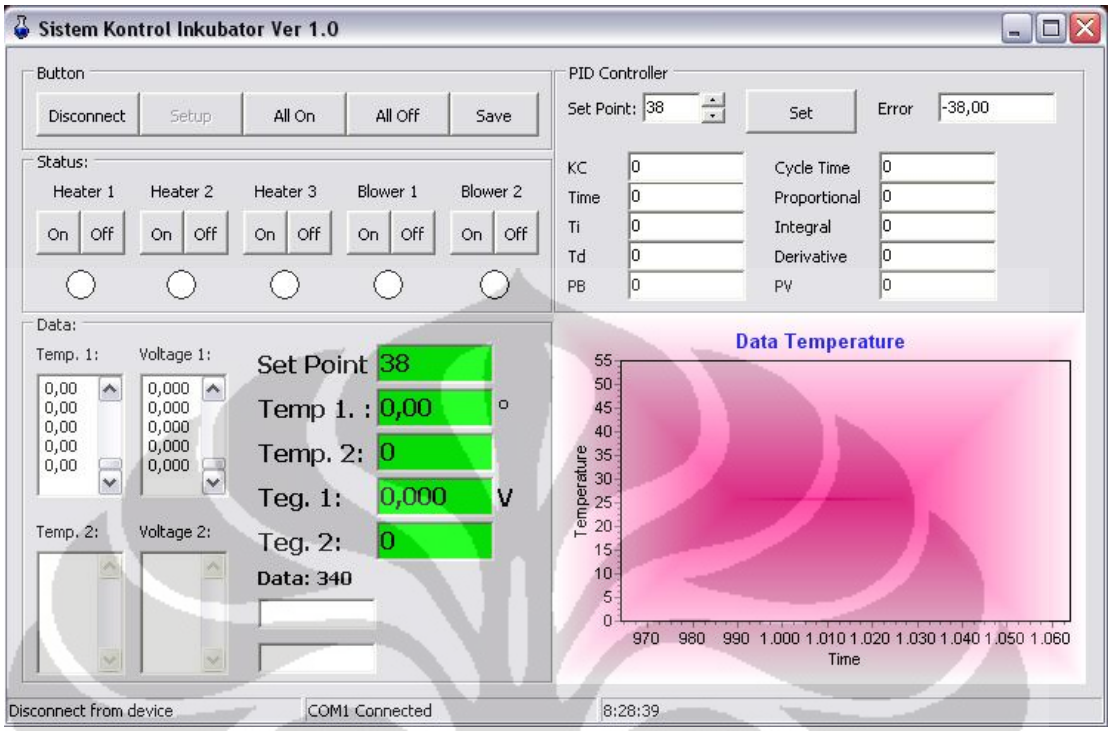
Beberapa feature yang disediakan oleh Redboot antara lain:

- ✓ Mendukung skrip *booting*,
- ✓ Mendukung *Command Line Interface* (CLI) untuk monitoring dan kontrol,
- ✓ Dapat diakses melalui port serial dan port ethernet,
- ✓ Mendukung GDB,
- ✓ Mendukung sistem *flash image*,
- ✓ Mendukung modem X/Y,
- ✓ Mendukung jaringan *bootstrap* menggunakan BOOTP atau konfigurasi alamat IP statis.

Sebagai catatan, saat menggunakan port ethernet, sangat penting untuk mengatur alamat IP yang sesuai. Alamat IP mikrokontroler harus disesuaikan dengan alamat IP sumber program, agar sistem dapat berjalan dengan baik.

3.2.2 Antarmuka Sistem Inkubator

Antarmuka sistem inkubator ini menggunakan bahasa pemrogaman Delphi 7. Software sistem inkubator ini selain digunakan untuk mengatur temperatur juga digunakan sebagai kontrol PID.



Gambar 3.4 Tampilan Antarmuka Sistem Inkubator

Fitur yang terdapat pada software ini antara lain pengendalian inkubator hingga 6 unit (pada pengembangan selanjutnya dapat lebih banyak sesuai dengan kapasitas jaringan), pengendalian temperatur secara manual maupun otomatis, simulasi kontrol temperatur, penyimpanan data dalam bentuk file berekstensi txt, excel, maupun PDF, serta pengendalian secara umum seperti ON/OFF sistem inkubator, dll.

BAB 4

ANALISIS

4.1 Perangkat Keras

Untuk memastikan alat kontrol temperatur yang telah dibuat bekerja dengan baik sesuai dengan yang diinginkan, maka perlu dilakukan beberapa pengujian dan pengambilan data.

4.1.1 Kalibrasi Sensor Temperatur

Untuk melakukan kalibrasi sensor LM35, peneliti menggunakan termometer air raksa sebagai referensi. Dengan memanaskan objek yang tadi digunakan, maka akan didapatkan nilai *output* tegangan dari sensor dan hasil konversi dalam bentuk digital dari ADC MAX128. Pengambilan data dilakukan pada kisaran 25 °C – 64 °C, karena memang sistem inkubator ini hanya pada menggunakan kisaran temperatur ini saja.

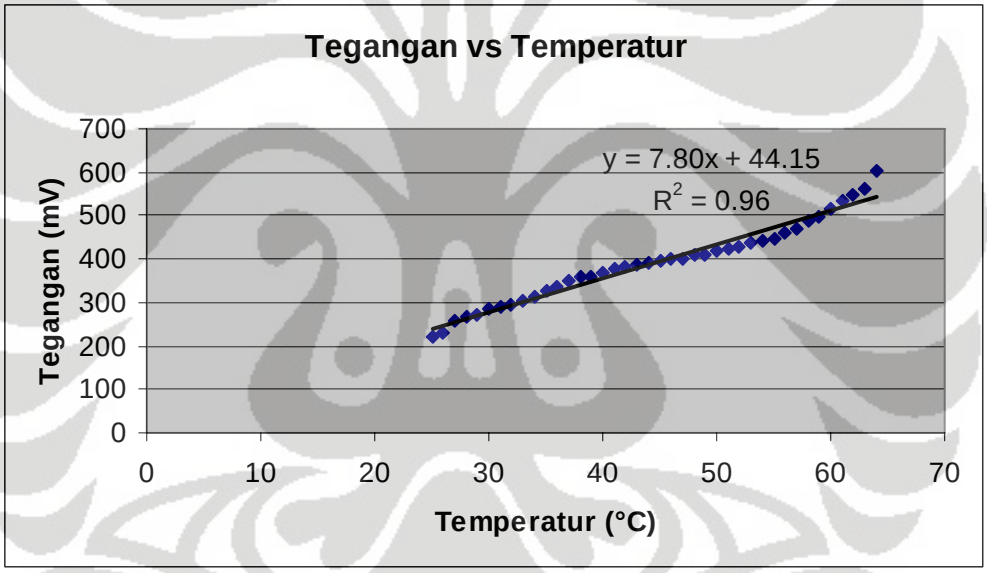
Tabel 4.1 merupakan data hasil pengamatan yang ditampilkan dalam bentuk grafik pada gambar 4.1.

Tabel 4.1 Data Pengamatan

No	Temp (°C)	Tegangan (mV)	No	Temp (°C)	Tegangan (mV)
1	25	223	21	45	395
2	26	230	22	46	399
3	27	257	23	47	402
4	28	266	24	48	408
5	29	271	25	49	412
6	30	285	26	50	417
7	31	291	27	51	422
8	32	296	28	52	428

9	33	302
10	34	314
11	35	329
12	36	336
13	37	349
14	38	357
15	39	361
16	40	370
17	41	376
18	42	381
19	43	386
20	44	390

29	53	438
30	54	443
31	55	449
32	56	460
33	57	470
34	58	487
35	59	498
36	60	518
37	61	532
38	62	548
39	63	560
40	64	602



Gambar 4.1 Hasil kalibrasi sensor LM35 dengan termometer air raksa (T-Hg)

Berdasarkan hasil kalibrasi yang didapat, terlihat bahwa data yang diplot berbentuk tidak lurus. Pada saat kalibrasi data diambil baik pada saat menaikkan temperatur dan menurunkan temperatur. Pada saat menaikkan temperatur, sensor LM35 menunjukkan nilai tegangan dengan perbedaan yang cukup besar, sedangkan ketika menurunkan temperatur, sensor LM35 menunjukkan nilai tegangan dengan

perbedaan yang kecil. Saat menaikkan temperatur, peneliti melakukan percobaan dengan memanaskan air sedangkan pada saat menurunkan temperatur peneliti menambahkan es ke dalam air.

4.1.2 Pengolahan data untuk kontrol *heater* dan *blower*

Pengaturan kecepatan motor dilakukan dengan mengatur *duty cycle* pada frekuensi yang telah ditentukan. Pada tabel 4.3 dapat dilihat bahwa periode sinyal sebesar 20000 ms. Jika kondisi yang diinginkan adalah kipas mati total maka status timer pada keadaan 100% untuk OFF dan 0% untuk ON, begitu pula sebaliknya bila kipas diinginkan kipas menyala dengan kecepatan maksimal maka keadaannya adalah 0% untuk OFF dan 100% untuk ON.

Tabel 4.2 Pengaturan Waktu Untuk Putaran Kipas

DUTY CYCLE PWM			
ON		OFF	
%	Time (ms)	%	Time (ms)
100	20000	0	0
90	18000	10	2000
80	16000	20	4000
70	14000	30	6000
60	12000	40	8000
50	10000	50	10000
40	8000	60	12000
30	6000	70	14000
20	4000	80	16000
10	2000	90	18000
0	0	100	20000

4.1.3 Respon kontrol sistem inkubator terhadap perubahan temperatur

Selain melakukan pengambilan data untuk sensor dan juga kontrol kipas, disini juga dilakukan pengambilan data untuk melihat respon sistem inkubator yang dibuat terhadap perubahan nilai temperatur yang diinginkan (*set point*). Berikut ini adalah gambar respon terhadap perubahan setting kenaikan dan juga penurunan. Semua data diambil pada temperatur ruangan berkisar pada temperatur 24°C. Untuk melakukan pengambilan data ini, penulis menggunakan software yang telah dibuat setelah sebelumnya memasukkan rumus hasil dari kalibrasi termometer air raksa dengan sensor LM35. Proses pengambilan data menggunakan port serial antara mikrokontroler dengan komputer dan menggunakan software sistem inkubator yang telah dibuat oleh penulis.

Pada gambar 4.2 dapat dilihat respon saat menaikkan temperatur dari 28°C menuju 30°C membutuhkan waktu sebesar 118 detik (1 menit 58 detik). Saat menuju temperatur di atasnya, sistem tidak langsung menaikkan temperaturnya, tetapi menuju temperatur yang diinginkan kemudian turun lagi dan naik lagi. Artinya, sistem membutuhkan waktu beberapa saat agar stabil pada suhu yang tetap.

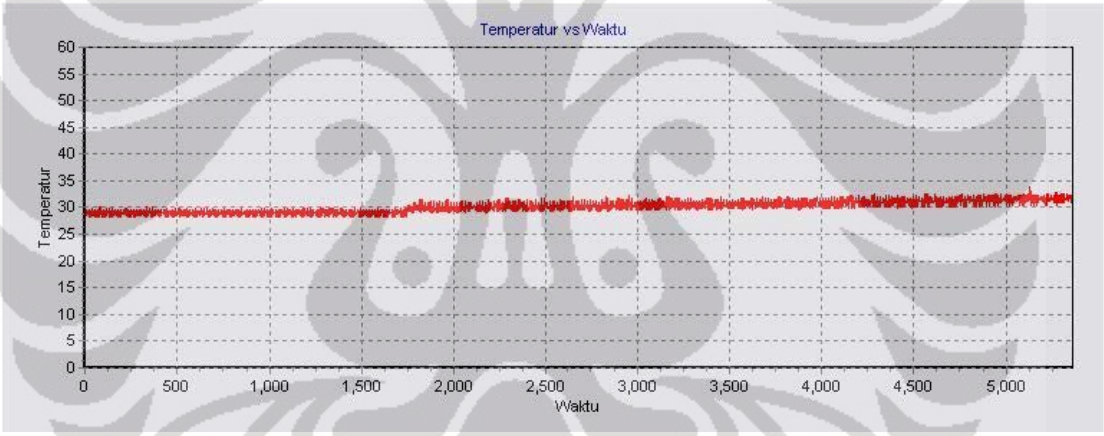
Pada gambar 4.3 dapat ditunjukkan waktu yang dibutuhkan untuk menaikkan temperatur dari 30°C menuju 35°C sebesar 272 detik (4 menit 32 detik). Sistem dengan cepat menaikkan temperatur dan seperti pada kenaikkan sebelumnya, sistem membutuhkan waktu beberapa saat untuk menjaga agar temperatur tetap stabil.

Gambar 4.4 menunjukkan respon untuk kenaikan temperatur dari 35°C menuju 40°C. Waktu yang dibutuhkan untuk menaikkan temperatur tersebut sebesar 458

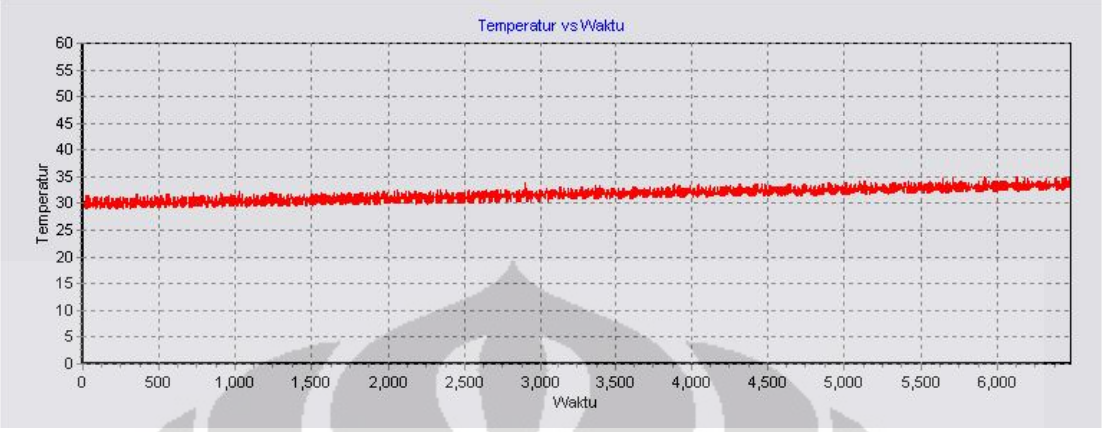
detik (7 menit 38 detik). Sistem dapat bertahan pada temperatur tersebut, dan penelitian ini dipertahankan selama 1180 detik (19 menit 40 detik).

Gambar 4.5 menunjukkan respon sistem inkubator pada penurunan temperatur dari 40°C menuju 35°C yang membutuhkan waktu selama 295 detik (3 menit 55 detik). Berbeda sekitar 23 detik antara menaikkan temperatur dan menurunkannya.

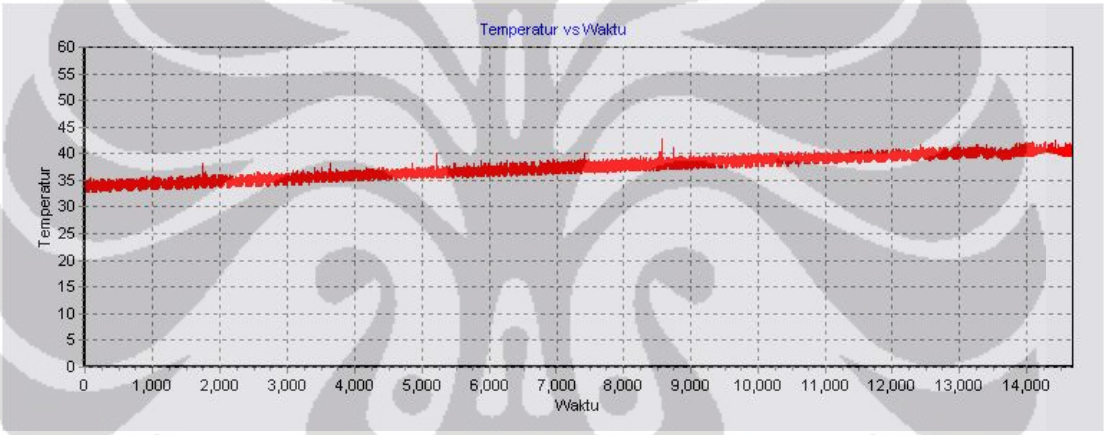
Respon untuk penurunan temperatur dari 35°C menuju 30°C sekitar 1085 detik (18 menit 5 detik) ditunjukkan pada gambar 4.6. Hal ini jauh berbeda dengan saat kenaikan temperatur.



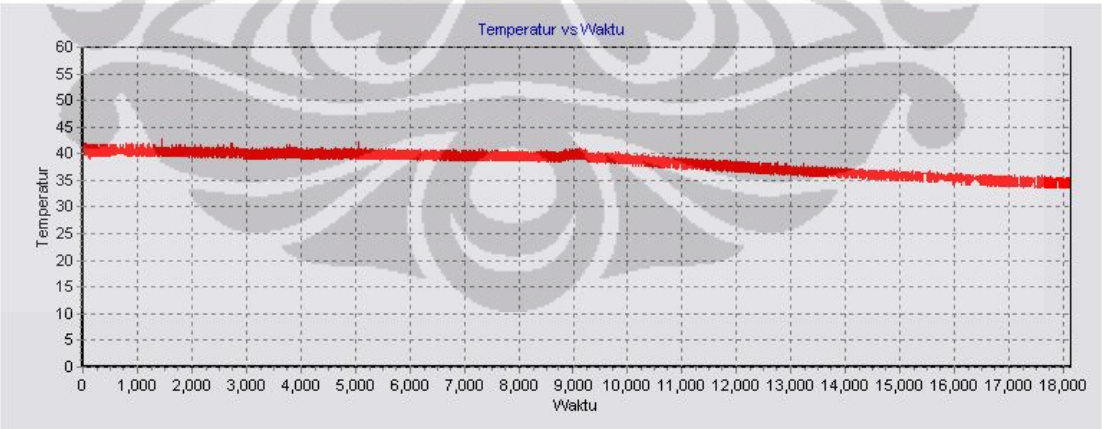
Gambar 4.2 Grafik respon sistem inkubator pada temperatur 28°C - 30°C



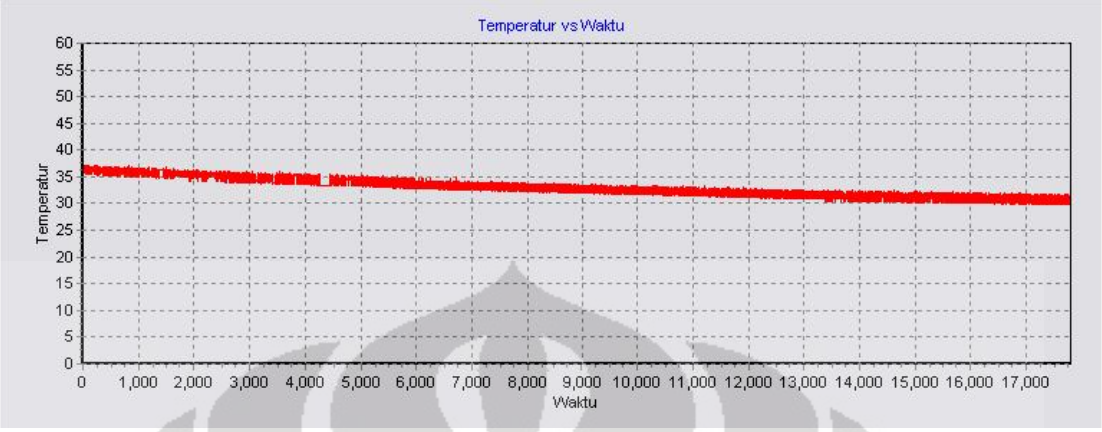
Gambar 4.3 Grafik respon sistem inkubator pada temperatur 30°C – 35°C



Gambar 4.4 Grafik respon sistem inkubator pada temperatur 35°C – 40°C



Gambar 4.5 Grafik respon sistem inkubator pada temperatur 40°C – 35°C



Gambar 4.6 Grafik respon sistem inkubator pada temperatur 35°C - 30°C

4.1.4 Proses transmisi data

Data dikirim dalam melalui komunikasi port serial RS232. Metode pengiriman menggunakan komponen Comport pada pemrograman Delphi 7.0.

4.2 Perangkat Lunak

Software inkubator telah bekerja dengan baik terhadap respon sistem inkubator saat menggunakan komunikasi serial.

Namun gagal dalam menghubungkan mikrokontroler ke jaringan yang disebabkan oleh definisi port mikrokontroler yang tidak tersedia pada kernel Linux. Sebab definisi yang tersedia pada kernel linux hanya sebatas pada feature-feature komunikasi seperti *serial communication interface* (SCI), dan port ethernet. Sedangkan input/output (I/O) yang didefinisikan merupakan I/O milik komputer, bukan untuk jenis mikrokontroler H8/3069F.

Karena pendefinisian alamat port pada kernel linux cukup rumit, hingga skripsi ini dibuat belum berhasil dilakukan. Sehingga dengan demikian, hanya penggunaan port ethernet dan port serial saja yang berhasil, sedangkan penggunaan I/O belum berhasil.

Berikut ini adalah tampilan yang dihasilkan oleh Redboot pada saat inisialisasi:

```
1 Ethernet eth0: MAC address 00:d0:b7:92:9d:d2
2 IP: 192.168.0.10, Default server: 192.168.0.1, DNS server IP: 0.0.0.0
3
4 RedBoot(tm) bootstrap and debug environment [FLOPPY]
5 Non-certified release, version UNKNOWN - built 17:55:00, Apr 21 2002
6
7 Platform: PC (I386)
8 Copyright (C) 2000, 2001, 2002, Red Hat, Inc.
9
10 RAM: 0x00000000-0x000a0000, 0x00088870-0x000a0000 available
11 RedBoot>
```

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan pada hasil yang didapat selama melakukan penelitian, terdapat beberapa kesimpulan sebagai berikut:

- ✓ Kontrol temperatur pada sistem inkubator dapat berjalan dengan baik sesuai dengan harapan dengan menggunakan komunikasi serial.
- ✓ Sensor temperatur bekerja dengan baik dan mendekati linier pada kisaran temperatur 30°C – 60°C.
- ✓ Temperatur pada inkubator yang digunakan berkisar antara 30°C – 40°C.
- ✓ Metode pengontrolan temperatur menggunakan metode PID (*proportional integral derivative*).
- ✓ Mikrokontroler belum terhubung ke jaringan LAN, karena disebabkan oleh kegagalan definisi alamat port pada kernel Linux.

5.2 Saran

Berdasarkan kesimpulan yang disebutkan di atas, terdapat beberapa saran sebagai berikut:

- ✓ Sensor temperatur yang digunakan sebaiknya tidak perlu menggunakan ADC atau dapat langsung dikonversi dari pengukuran temperatur agar dapat mengurangi kesalahan pengukuran temperatur.

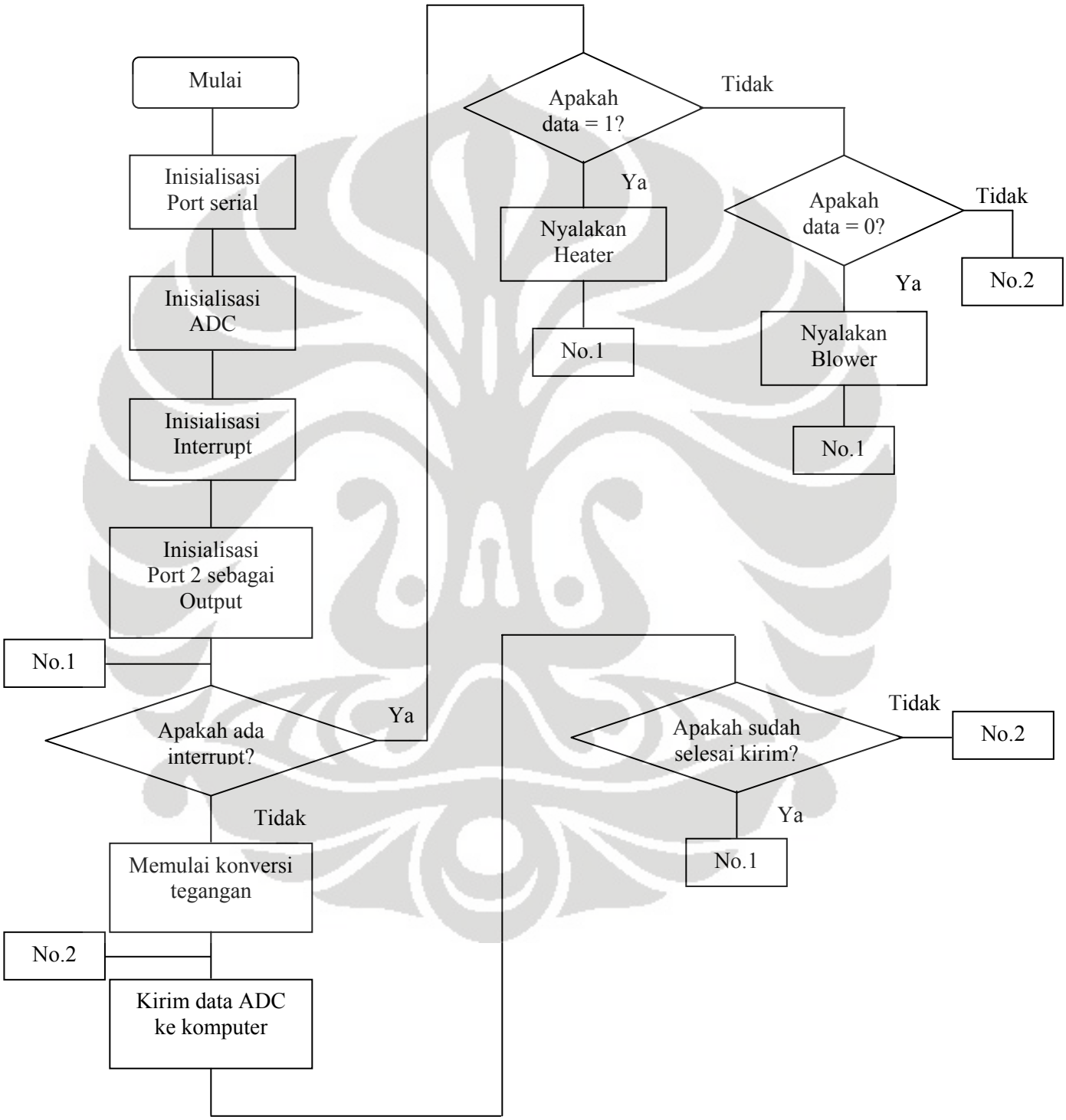
- ✓ Memaksimalkan penggunaan mikrokontroler H8/3069F, baik kapasitas ROM/RAM, jumlah input dan output, maupun feature-feature yang tersedia.
- ✓ Untuk pengembangan selanjutnya, agar mikrokontroler dapat langsung digunakan sebagai server sehingga dapat melayani input/output yang lebih banyak lagi.

DAFTAR ACUAN

1. Febriarso, Singgih. 2004. *Sistem Kontrol dan Monitoring melalui LAN Untuk Temperatur*. Depok: Skripsi Sarjana Fisika Universitas Indonesia.
2. Alfa JK, MAS. 2007. *Osiloskop Digital Dua Kanal Menggunakan Mikrokontroler 16-bit H8/3069F*. Depok: Skripsi Sarjana Fisika Universitas Indonesia.
3. Joni, I Made & Budi Raharjo. 2006 *Pemrograman C dan Implementasinya*. Informatika, Bandung: 400 hlm.
4. Kleitz, William. 1996. *Digital Electronics 4th Edition.* Prentice-Hall, Inc., New Jersey: xxv + 726 hlm.
5. Malvino, Albert Paul. 1999. *Electronics Principles 6th Edition.* Tata McGraw-Hill, New Delhi: xi + 1012 hlm.
6. Renesas Solutions Corp. 2005. *H8/3068F-ZTAT™ Hardware Manual*. Renesas Technology Corp., Japan: 935 hlm.
7. Sjachriyanto, Wawan, Kusnassriyanto Saiful Bahri. 2005. *Pemrograman Delphi*. Informatika, Bandung.
8. Sunarto, Rumono. 2004. *Membangun Sistem Akuisisi Data Berbasis Database dengan Delphi*. Elex Media Komputindo, Jakarta.

LAMPIRAN A

FLOWCHART



LAMPIRAN B

PROGRAM KONTROL INKUBATOR

Program Mikrokontroler

```
#include "io306x.h"
#include "mydef.h"
#include "uart.h"
#include "inthandler.h"
#include "max128.h"

#define waktu 100000


unsigned char rxdata = 0;
int i, pilih;
short ch0, ch1;
short heater1, heater2, heater3, blower1, blower2;

//*********************************************

int main ()
{
        P2DDR = 0xFF;
        P2DR.BYTE = 0;
        heater1 = 0;
        heater2 = 0;
        blower1 = 0;
        blower2 = 0;

        uart_init();
        set_interrupt_mask(0);


        while (1){
        //Looping trus

        start_conv(CH0);
        usdelay(10);
        ch0 = get_value();
        uart_tx("1");
        uart_num((long)ch0);
        uart_tx("\n\r");

        start_conv(CH1);
        usdelay(10);
        ch1 = get_value();
        uart_tx("2");
        uart_num((long)ch1);
        uart_tx("\n\r");

        }

        return 0;
}

//Interrupt handler
void INT_RXI1(void)
{
        /*add your code here*/
        rxdata = uart_rd();
        //P2DR.BYTE = rxdata;
```

```
        //Nyalakan Heater1
        if ((rxdata&0xFE) == 4){
                pilih = 0x01;
                heater1 = pilih;
                goto lompat;
        }
        //Nyalakan Heater2
        if ((rxdata&0xFE) == 6){
                pilih = 0x02;
                heater2 = pilih;
                goto lompat;
        }
        //Nyalakan Heater3
        if ((rxdata&0xFE) == 8){
                pilih = 0x04;
                heater3 = pilih;
                goto lompat;
        }
        //Nyalakan Blower1
        if ((rxdata&0xFE) == 10){
                pilih = 0x80;
                blower1 = pilih;
                goto lompat;
        }
        //Nyalakan Blower2
        if ((rxdata&0xFE) == 12){
                pilih = 0x40;
                blower2 = pilih;
                goto lompat;
        }
        //Mematikan Heater1
        if ((rxdata&0xFE) == 16){
                pilih = 0x00;
                heater1 = pilih;
                goto lompat;
        }
        //Mematikan Heater2
        if ((rxdata&0xFE) == 18){
                pilih = 0x00;
                heater2 = pilih;
                goto lompat;
        }
        //Mematikan Heater3
        if ((rxdata&0xFE) == 20){
                pilih = 0x00;
                heater3 = pilih;
                goto lompat;
        }
        //Mematikan Blower1
        if ((rxdata&0xFE) == 22){
                pilih = 0x00;
                blower1 = pilih;
                goto lompat;
        }
        //Mematikan Blower2
        if ((rxdata&0xFE) == 24){
                pilih = 0x00;
                blower2 = pilih;
                goto lompat;
        }
        //Menyalakan semuanya
        if ((rxdata&0xFE) == 14){
                pilih = 0x00;
                heater1 = 0x01; heater2 = 0x02; heater3 = 0x04; blower1 = 0x80; blower2
= 0x40;
                goto lompat;
        }
```

```
        //Mematikan semaunya
        if ((rxdata&0xFE) == 26){
                pilih = 0x00;
                heater1 = pilih; heater2 = pilih; heater3 = pilih; blower1 = pilih;
blower2 = pilih;
                goto lompat;
        }
        lompat:
        P2DR.BYTE = heater1 | heater2 | heater3 | blower1 | blower2;
}
```

==========

Program Interface Sistem Inkubator Ver. 1.0

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, CPortCtl, CPort, StdCtrls, Buttons, ExtCtrls,
  TeEngine, Series, TeeProcs, Chart, Math, IdBaseComponent, IdComponent,
  IdTCPConnection, IdTCPClient;

type
  TForm1 = class(TForm)
    ComPort: TComPort;
    StatusBar: TStatusBar;
    Chart1: TChart;
    Series1: TFastLineSeries;
    TimerCount: TTimer;
    GBStatus: TGroupBox;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Shape1: TShape;
    Shape2: TShape;
    Shape4: TShape;
    Shape5: TShape;
    Label3: TLabel;
    Shape3: TShape;
    BtnOn1: TBitBtn;
    BtnOff1: TBitBtn;
    BtnOn2: TBitBtn;
    BtnOff2: TBitBtn;
    BtnOn4: TBitBtn;
    BtnOff4: TBitBtn;
    BtnOn5: TBitBtn;
    BtnOff5: TBitBtn;
    BtnOn3: TBitBtn;
    BtnOff3: TBitBtn;
    TimerData1: TTimer;
    Series2: TFastLineSeries;
    MemoData: TMemo;
    GroupBox1: TGroupBox;
    Label10: TLabel;
    MemoCh1: TMemo;
    MemoCh2: TMemo;
    Label2: TLabel;
    Label12: TLabel;
    MemoVolt1: TMemo;
    MemoVolt2: TMemo;
    Label11: TLabel;
    LblSensor2: TLabel;
```

```
LblSensor1: TLabel;
LblVolt1: TLabel;
LblVolt2: TLabel;
LabelData1: TLabel;
Edit1: TEdit;
Edit3: TEdit;
GroupBox2: TGroupBox;
BtnConnect: TBitBtn;
BtnDisconnect: TBitBtn;
BtnSetup: TBitBtn;
BitBtn9: TBitBtn;
BitBtn10: TBitBtn;
Memo1: TMemo;
EditTemp1: TEdit;
EditTemp2: TEdit;
EditVolt1: TEdit;
EditVolt2: TEdit;
GBPID: TGroupBox;
Label16: TLabel;
EditSP: TEdit;
UpDown1: TUpDown;
Label1: TLabel;
Label4: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
EditTime: TEdit;
EditTi: TEdit;
EditTd: TEdit;
EditPB: TEdit;
EditCycle: TEdit;
EditProp: TEdit;
EditInt: TEdit;
EditDiff: TEdit;
EditPV: TEdit;
Label21: TLabel;
EditKC: TEdit;
BtnSP: TBitBtn;
Label22: TLabel;
EditSPDisp: TEdit;
Label23: TLabel;
EditError: TEdit;
BtnSave: TBitBtn;
Timer1: TTimer;
Timer2: TTimer;
Label9: TLabel;
BtnStartTime: TButton;
BtnStopTime: TButton;
BtnReset: TButton;
procedure PID;
procedure inisialisasi;
procedure TampilData;
procedure AfterAdd(Sender: TChartSeries; ValueIndex: integer);
procedure AmbilData;
procedure KirimData;
procedure Kondisi(Cek: word);
procedure AturKondisi;
function ConvertTemp(Const valTemp: integer): Double;
function ConvertVolt(Const valVolt: integer): Double;
function LeftStr(Const Str: String; Size: Word): String;
function CekData(Const Data: String): String;
//procedure untuk file excel
procedure XlsBeginStream(XlsStream: TStream; const BuildNumber: Word);
procedure XlsEndStream(XlsStream: TStream);
```

```
    procedure XlsWriteCellRk(XlsStream: TStream; const ACol, ARow: Word; const AValue:
Integer);
    procedure XlsWriteCellNumber(XlsStream: TStream; const ACol, ARow: Word; const
AValue: Double);
    procedure XlsWriteCellLabel(XlsStream: TStream; const ACol, ARow: Word; const
AValue: string);

    procedure ComPortRxChar(Sender: TObject; Count: Integer);
    procedure FormActivate(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure TimerCountTimer(Sender: TObject);
    procedure TimerData1Timer(Sender: TObject);
    procedure BtnConnectClick(Sender: TObject);
    procedure BtnDisconnectClick(Sender: TObject);
    procedure BtnSetupClick(Sender: TObject);
    procedure BtnOn1Click(Sender: TObject);
    procedure BtnOff1Click(Sender: TObject);
    procedure BtnOn2Click(Sender: TObject);
    procedure BtnOff2Click(Sender: TObject);
    procedure BtnOn3Click(Sender: TObject);
    procedure BtnOff3Click(Sender: TObject);
    procedure BtnOn4Click(Sender: TObject);
    procedure BtnOff4Click(Sender: TObject);
    procedure BtnOn5Click(Sender: TObject);
    procedure BtnOff5Click(Sender: TObject);
    procedure BitBtn9Click(Sender: TObject);
    procedure BitBtn10Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);
    procedure CBH1Click(Sender: TObject);
    procedure CBH2Click(Sender: TObject);
    procedure CBH3Click(Sender: TObject);
    procedure CBB1Click(Sender: TObject);
    procedure CBB2Click(Sender: TObject);
    procedure BtnSPClick(Sender: TObject);
    procedure BtnSaveClick(Sender: TObject);
    procedure BtnStartTimeClick(Sender: TObject);
    procedure BtnStopTimeClick(Sender: TObject);
    procedure BtnResetClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  //Variabel untuk PID
  PB: integer; //variabel untuk menentukan perbesaran gain
  Ti: integer; //variabel untuk menentukan nilai time integral
  Td: integer; //variabel untuk menentukan nilai time differensial
  Time_val: integer;
  SP, temp_SP: integer; //variabel untuk menentukan nilai set point
  MV, temp_MV: integer; //variabel untuk menampung hasil pengolahan data PID
  PV, temp_PV: integer; //variabel untuk menampung hasil sementara dari alat
  Err_val: integer; //variabel untuk menampung nilai error
  Prop_value: integer; //variabel untuk menampung hasil perhitungan proportional
  Int_value: integer;  //variabel untuk menampung hasil perhitungan integral
  Diff_value: integer; //variabel untuk menampung hasil perhitungan differential
  KC_value: integer; //variabel untuk menampung hasil perhitungan KC(gain)
  First_data: integer; //variabel untuk menampung hasil perhitungan awal
  Last_data: integer; //variabel untuk menampung hasil perhitungan akhir
  Paralel_value: integer;
  Seri_value: integer;
  Mix_value: integer;
  Cadangan, Hasil: integer; //Buat nampung nilai yang akan di kirim ke plant
```

```
  option: byte; //variabel untuk memilih struktur PID
  cacah, counter_pid: integer;
  cycletime, hitung: integer;
  hsl_sp, hsl_asli, hsl_temp: double;

  //Variabel untuk excel
  CXlsBof: array[0..5] of Word = ($809, 8, 00, $10, 0, 0);
  CXlsEof: array[0..1] of Word = ($0A, 00);
  CXlsLabel: array[0..5] of Word = ($204, 0, 0, 0, 0, 0);
  CXlsNumber: array[0..4] of Word = ($203, 14, 0, 0, 0);
  CXlsRk: array[0..4] of Word = ($27E, 10, 0, 0, 0);

  Status, StatusHeater1, StatusHeater2, StatusHeater3, StatusBlower1, StatusBlower2,
StatusAll: word;
  counter, counterdata, dataku: integer;
  MyFile1: TextFile;
  FFile: String; //Untuk jenis file text
  FStream: TFileStream; //Untuk jenis file stream
  Str, StrTemp, ReadData: String; //Untuk membaca data dari mikrokontroler
  WriteData: String; //Untuk menulis data ke mikrokontroler
  DataTemp, d: Double;
  DataBuff: Array[0..4000] of double;
  k, ax, ay: integer;
  TempCh1, TempCh2, voltch1, voltch2: double;
  Tanggal, Waktu: String;
  Jam, Menit, Detik, Mdetik: Word;
  StrWaktu, StrTime: String;
  f: double;
  int_f: integer;
const
  ComHeater1On = 4; ComHeater1Off = 16;
  ComHeater2On = 6; ComHeater2Off = 18;
  ComHeater3On = 8; ComHeater3Off = 20;
  ComBlower1On = 10; ComBlower1Off = 22;
  ComBlower2On = 12; ComBlower2Off = 24;
  ComAllOn = 14; ComAllOff = 26;

implementation

{$R *.dfm}

function Proportional(data: integer; gain: integer): integer;
//Fungsi untuk mengitung besar gain dari PB
begin
  Result := gain * data;
end;

function Integral(data: integer; jumlah: integer): integer;
//Fungsi untuk menghitung jumlah dari beberapa masukan
begin
  Result := jumlah + data;
end;

function Differential(data: integer; data_1: integer): integer;
//Fungsi untuk mengurangi data baru dengan data lama
begin
  Result := data_1 - data;
end;

function KC(input: integer): integer;
//Fungsi untuk mengitung besar gain dari PB
begin
  Result := 100 div input;
end;

function  Paralel(time_d:  integer;  time_i:  integer;  in_prop:  integer;  in_integ:
integer;
                in_tur: integer; T: integer): integer;
```

```
//Fungsi untuk menghitung PID dengan metode paralel
begin
  Result := in_prop + ((T div time_i) * in_integ) + ((time_d div T) * in_tur);
end;

function Seri(time_d: integer; time_i: integer; in_prop: integer; in_integ: integer;
              in_tur: integer; T: integer; KC: integer; data: integer): integer;
//Fungsi untuk menghitung PID dengan metode seri
begin
  Result := in_prop + ((time_d div time_i) * data) + (((KC * T) div time_i)* in_integ)
+
            (((KC * time_d) div t) * in_tur);
end;

function Mix(time_d: integer; time_i: integer; in_prop: integer; in_integ: integer;
             in_tur: integer; T: integer; KC: integer): integer;
//Fungsi untuk menghitung PID dengan metode campuran
begin
  Result := in_prop + (((KC * T) div time_i) * in_integ) + (((KC * time_d) div T)*
in_tur);
end;

procedure TForm1.PID;
//Prosedur utama untuk menghitung nilai PID
begin
  KC_value      := KC(StrToInt(EditPB.Text));
  Prop_value    := Proportional(Last_data, KC_value);
  Int_value     := Integral(Last_data, Prop_value);
  Diff_value    := Differential(Last_data, First_data);
  Paralel_value := Paralel(Td, Ti, Prop_value, Int_value, Diff_value, Time_val);
  Seri_value    := Seri(Td, Ti, Prop_value, Int_value, Diff_value, Time_val, KC_value,
Last_data);
  Mix_value     := Mix(Td, Ti, Prop_value, Int_value, Diff_value, Time_val, KC_value);
end;

procedure TForm1.Inisialisasi;
//Prosedur untuk melakukan inisialisasi
begin
  PV := 0;
  MV := 0;
  SP := 0;
end;

procedure TForm1.TampilData;
begin
  //Prosedur untuk menampilkan data
  EditKC.Text      := IntToStr(KC_value);
  EditProp.Text    := IntToStr(Prop_value);
  EditInt.Text     := IntToStr(Int_value);
  EditDiff.Text    := IntToStr(Diff_value);
  //EditParalel.Text := IntToStr(Paralel_value);
  //EditSeri.Text    := IntToStr(Seri_value);
  //EditMix.Text     := IntToStr(Mix_value);
end;

procedure TForm1.KirimData;
begin
//Kirim data
end;


procedure TForm1.AfterAdd(Sender: TChartSeries; ValueIndex: integer);
begin
  //
  with sender.GetHorizAxis do
  begin
    Automatic := False;
    Minimum := 0;
```

```
    Maximum := Sender.XValues.MaxValue;
    Minimum := Maximum - 100;
  end;
end;

procedure TForm1.AmbilData();
begin
  //Ambil data
  ReadData := StrTemp;
end;

procedure TForm1.Kondisi(Cek: word);
begin
  //Melakukan kondisi alat
  {
  Kondisi 0: Heater1 on  |  Kondisi 6: Heater1 off
  Kondisi 1: Heater2 on  |  Kondisi 7: Heater2 off
  Kondisi 2: Heater3 on  |  Kondisi 8: Heater3 off
  Kondisi 3: Blower1 on  |  Kondisi 9: Blower1 off
  Kondisi 4: Blower2 on  |  Kondisi 10: Blower2 off
  Kondisi 5: Semua on    |  Kondisi 11: Semua off
  }

  case Cek of
  0:  //Heater 1 On
  begin
    StatusHeater1 := 1;
    Shape1.Brush.Color := clRed;
    If Comport.Connected then
    begin
      Comport.WriteStr(chr(ComHeater1On));
    end;
  end;
  1:  //Heater 2 On
  begin
    StatusHeater2 := 1;
    Shape2.Brush.Color := clRed;
    If Comport.Connected then
    begin
      Comport.WriteStr(chr(ComHeater2On));
    end;
  end;
  2:  //Heater 3 On
  begin
    StatusHeater3 := 1;
    Shape3.Brush.Color := clRed;
    If Comport.Connected then
    begin
      Comport.WriteStr(chr(ComHeater3On));
    end;
  end;
  3:  //Blower 1 On
  begin
    StatusBlower1 := 1;
    Shape4.Brush.Color := clRed;
    If Comport.Connected then
    begin
      Comport.WriteStr(chr(ComBlower1On));
    end;
  end;
  4:  //Blower 2 On
  begin
    StatusBlower2 := 1;
    Shape5.Brush.Color := clRed;
    If Comport.Connected then
    begin
      Comport.WriteStr(chr(ComBlower2On));
    end;
```

```
end;
5:  //Semuanya On
begin
  StatusAll := 1;
  Shape1.Brush.Color := clRed;
  Shape2.Brush.Color := clRed;
  Shape3.Brush.Color := clRed;
  Shape4.Brush.Color := clRed;
  Shape5.Brush.Color := clRed;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComAllOn));
  end;
end;
6:  //Heater 1 off
begin
  StatusHeater1 := 0;
  Shape1.Brush.Color := clWhite;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComHeater1Off));
  end;
end;
7:  //Heater 2 off
begin
  StatusHeater2 := 0;
  Shape2.Brush.Color := clWhite;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComHeater2Off));
  end;
end;
8:  //Heater 3 off
begin
  StatusHeater3 := 0;
  Shape3.Brush.Color := clWhite;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComHeater3Off));
  end;
end;
9:  //Blower 1 off
begin
  StatusBlower1 := 0;
  Shape4.Brush.Color := clWhite;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComBlower1Off));
  end;
end;
10:  //Blower 2 off
begin
  StatusBlower1 := 0;
  Shape5.Brush.Color := clWhite;
  If Comport.Connected then
  begin
    Comport.WriteStr(chr(ComBlower2Off));
  end;
end;
11:  //Semuanya Off
begin
  StatusAll := 0;
  Shape1.Brush.Color := clWhite;
  Shape2.Brush.Color := clWhite;
  Shape3.Brush.Color := clWhite;
  Shape4.Brush.Color := clWhite;
  Shape5.Brush.Color := clWhite;
  If Comport.Connected then
```

```
    begin
      Comport.WriteStr(chr(ComAllOff));
    end;
  end;
  end;

end;

procedure TForm1.AturKondisi;
begin
  //Mengatur kondisi heater dan blower
  SP := StrToInt(EditSP.Text);  //mendapatkan nilai set point
  f := round(TempCh1); //untuk mendapatkan nilai dari sensor
  int_f := StrToInt(FloatToStr(f));  //mengubah variabel f menjadi integer
  PV := int_f; //masukkan nilai dari sensor ke PV
  Err_val := SP - PV;  //hasil error dari SP - PV
  EditError.Text := IntToStr(Err_val);

  {if Err_val < 0 then
  begin
    kondisi(6); kondisi(7); kondisi(8); kondisi(3); kondisi(4);
  end else
  if Err_val > 0 then
  begin
    kondisi(0); kondisi(1); kondisi(2); kondisi(3); kondisi(4);
  end else
  begin
  end;}

  {case Err_val of
  -8..-4 : begin kondisi(6); kondisi(7); kondisi(8); kondisi(3); kondisi(4); end;
  -3..-1 : begin kondisi(6); kondisi(3); kondisi(4); end;
  1..3 : begin kondisi(0); kondisi(3); kondisi(4); end;
  4..8 : begin kondisi(0); kondisi(1); kondisi(2); kondisi(3); kondisi(4); end;
  else
  begin kondisi(3); kondisi(4); end;
  end;}
end;

function TForm1.ConvertTemp(Const valTemp: integer): Double;
const
  a = 44.15;
  b = 7.8045;
  a1 = 0.5323;
  b1 = 0.441;
  //mili = 1e-3;
var
  hasil: double;
begin
  //Mengkonversi temperature
  case valTemp of
  0..2047    :
  begin
  hasil := (((valTemp+a1)/b1)-a)/b;
  //hasil := (valTemp - a) / b;
  end;
  2048..4095 :
  begin
  hasil := ((valTemp-4096) - a) / b;
  end;
  else
  hasil := 0;
  end;

  Result :=  hasil;
end;

function TForm1.ConvertVolt(Const valVolt: integer): Double;
```

```
const
  a = 0.5323;
  b = 0.441;
  mili = 1e-3;
var
  hasil: double;
begin
  //Mengkonversi
  case valVolt of
  0..2047    :
  begin
  hasil := mili * (valVolt + a) / b;
  end;
  2048..4095 :
  begin
  hasil := mili * ((valVolt-4096) + a) / b;
  end;
  else
  hasil := 0;
  end;

  Result :=  hasil;
end;

function TForm1.LeftStr(Const Str: String; Size: Word): String;
begin
     LeftStr := Copy(Str,1,Size)
end; {LeftStr}

function TForm1.CekData(Const Data: String): String;
//Digunakan untuk meghilangkan karakter yang tidak digunakan
var
  tempdata: string;
  panjang: integer;
begin
  panjang := Length(Data);

If Pos(char(254),Data) = 0 then
  tempdata := Data else
  tempdata := LeftStr(Data,panjang-1);

  Result:= tempdata;
end;

procedure TForm1.XlsBeginStream(XlsStream: TStream; const BuildNumber: Word);
begin
  CXlsBof[4] := BuildNumber;
  XlsStream.WriteBuffer(CXlsBof, SizeOf(CXlsBof));
end;

procedure TForm1.XlsEndStream(XlsStream: TStream);
begin
  XlsStream.WriteBuffer(CXlsEof, SizeOf(CXlsEof));
end;

procedure  TForm1.XlsWriteCellRk(XlsStream:  TStream;  const  ACol,  ARow:  Word;  const
AValue: Integer);
var
  V: Integer;
begin
  CXlsRk[2] := ARow;
  CXlsRk[3] := ACol;
  XlsStream.WriteBuffer(CXlsRk, SizeOf(CXlsRk));
  V := (AValue shl 2) or 2;
  XlsStream.WriteBuffer(V, 4);
end;
```

```
procedure TForm1.XlsWriteCellNumber(XlsStream: TStream; const ACol, ARow: Word; const
AValue: Double);
begin
  CXlsNumber[2] := ARow;
  CXlsNumber[3] := ACol;
  XlsStream.WriteBuffer(CXlsNumber, SizeOf(CXlsNumber));
  XlsStream.WriteBuffer(AValue, 8);
end;

procedure TForm1.XlsWriteCellLabel(XlsStream: TStream; const ACol, ARow: Word; const
AValue: string);
var
  L: Word;
begin
  L := Length(AValue);
  CXlsLabel[1] := 8 + L;
  CXlsLabel[2] := ARow;
  CXlsLabel[3] := ACol;
  CXlsLabel[5] := L;
  XlsStream.WriteBuffer(CXlsLabel, SizeOf(CXlsLabel));
  XlsStream.WriteBuffer(Pointer(AValue)^, L);
end;

procedure TForm1.ComPortRxChar(Sender: TObject; Count: Integer);
var
  a, b, c: integer;
  MyText, temp: string;
  TempText: string;
  Banding, Panjang: integer;
  Data: word;
  Simpan, Pangkat: real;
begin
  //Membaca data dari mikrokontroler
  ComPort.ReadStr(Str,Count); //Membaca dari buffer serial dan disimpan di Str
  Edit1.Text := Trim(Str); //Menghilangkan semua karakter spasi
  Edit3.Text := Str;

  If (Edit1.Text <> '') OR (Edit1.Text = #13) then //Melakukan pengecekan apabila ada
data yang kosong
  begin
    MyText := CekData(Edit1.Text);
    a := Pos('@',MyText);
    b := Pos(' ',MyText);
    If (a=0) OR (b=0) then
    begin
      Temp := Copy(MyText,1,4);
    end else
      Temp := Copy(MyText,a+1,a+b+3); //Copy(S,index,count)

    Banding := StrToInt(LeftStr(MyText,1)); //untuk mendapatkan 1 karakter pertama
    Panjang := Length(MyText); //untuk mendapatkan panjang text seluruhnya
    Data := Panjang-1; //untuk menentukan basis pangkat
    TempText := LeftStr(MyText,panjang);
    Pangkat := Power(10,data); //Pangkat = 10^data

    case Banding of
    1:
    begin
    Simpan := StrToFloat(TempText) - Pangkat;
    ax := StrToInt(FloatToStr(Simpan));
    end;
    2:
    begin
    Simpan := StrToFloat(temptext) - (2 * Pangkat);
    ay := StrToInt(FloatToStr(Simpan));
    end;
    else
    text := '0';
```

```
    end;

    c := StrToInt(Trim(Temp));
    voltch1 := ConvertVolt(ax);
    voltch2 := ConvertVolt(ay);
    TempCh1 := ConvertTemp(ax);
    TempCh2 := ConvertTemp(ay);
    d := ConvertTemp(c);
    Memo1.Lines.Add(Str);
  end else
  Memo1.ClearSelection;

end;

procedure TForm1.TimerData1Timer(Sender: TObject);
begin
  //Untuk menampilkan series1 pada chart
  Ambildata;
  Aturkondisi;
  //f := round(TempCh1);
  //int_f := StrToInt(FloatToStr(f));

  With Series1 Do
  begin
    //Data untuk chart1 dari channel 1
    addXY(counterdata, TempCh1, '', clTeeColor);
  end;

  With Series2 Do
  begin
    //Data untuk chart1 dari channel 2
    addXY(counterdata, TempCh2, '', clTeeColor);
  end;

  MemoCh1.Lines.Add(FormatFloat('0.00',TempCh1));
  MemoCh2.Lines.Add(FormatFloat('0.00',TempCh2));
  MemoVolt1.Lines.Add(FormatFloat('0.000',voltch1));
  MemoVolt2.Lines.Add(FormatFloat('0.000',voltch2));
  MemoData.Lines.Add(IntToStr(counterdata-
7)+#9+FormatFloat('0.00',TempCh1)+#9+FormatFloat('0.000',voltch1)+#9+FormatFloat('0.00
',TempCh2)+#9+FormatFloat('0.000',voltch2));
  LblSensor1.Caption := 'Temp 1. :                '+char(176);
  LblSensor2.Caption := 'Temp 2. :                '+char(176);
  LblVolt1.Caption := 'Teg. 1:                    V';
  LblVolt2.Caption := 'Teg. 2:                    V';
  EditTemp1.Text := FormatFloat('0.00',TempCh1);
  EditTemp2.Text := FormatFloat('0.00',TempCh2);
  EditVolt1.Text := FormatFloat('0.000',voltch1);
  EditVolt2.Text := FormatFloat('0.000',voltch2);
  LabelData1.Caption := 'Data: '+IntToStr(MemoCh1.Lines.Count);
  hsl_sp := StrToFloat(EditSP.Text);
  hsl_asli := StrToFloat(EditTemp1.Text);
  hsl_temp := hsl_asli - hsl_sp;
  EditError.Text := FormatFloat('0.00',hsl_temp);

  Tanggal := FormatDateTime('dd mmmm yyyy',date);

  //Save File to myfile
  AssignFile(Myfile1,Tanggal+' '+Waktu+'.txt');
  ReWrite(Myfile1);
  Write(MyFile1,'Hasil data temperature');
  Writeln(MyFile1);
  Write(MyFile1,Tanggal+' '+Waktu);
  Write(MyFile1,'. Banyak data: '+IntToStr(MemoCh1.Lines.Count));
  Writeln(MyFile1);
  Writeln(MyFile1);
  Write(MyFile1, 'Data'+#9+'Temp. 1'+#9+'Teg. 1'+#9+'Temp. 2'+#9+'Teg. 2');
  Writeln(MyFile1);
```

```
  Write(MyFile1,MemoData.Text);
  Writeln(MyFile1);
  CloseFile(MyFile1);

  //Save File to untuk parameter sensor 1
  AssignFile(Myfile1,'Temp1-'+Tanggal+' '+Waktu+'.txt');
  ReWrite(Myfile1);
  Write(MyFile1,MemoCh1.Text);
  //Writeln(MyFile1);
  CloseFile(MyFile1);

end;

procedure TForm1.TimerCountTimer(Sender: TObject);
begin
  //Untuk mengcounter data
  inc(counterdata);
end;

procedure TForm1.BtnConnectClick(Sender: TObject);
//var
//InputString: string;
begin
  //InputString:= InputBox('Sistem Inkubator', 'Masukan Nilai Set Point?', '37');
  //SP := StrToInt(InputString);
  //EditSP.Text := InputString;
  //EditSPDisp.Text := InputString;
  TimerData1.Enabled := True;
  BtnConnect.Visible := False;
  BtnDisconnect.Visible := True;
  BtnSetup.Enabled := False;
  GBStatus.Enabled := True;
  ComPort.Connected := True;
  StatusBar.Panels[1].Text := ComPort.Port + ' Connected';
end;

procedure TForm1.BtnDisconnectClick(Sender: TObject);
begin
  TimerData1.Enabled := False;
  BtnConnect.Visible := True;
  BtnDisconnect.Visible := False;
  BtnSetup.Enabled := True;
  GBStatus.Enabled := False;
  ComPort.Connected := False;
  StatusBar.Panels[1].Text := ComPort.Port + ' Disconnected';
end;

procedure TForm1.BtnSetupClick(Sender: TObject);
begin
  ComPort.ShowSetupDialog;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  ComPort.Connected := False;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DecodeTime(Time, Jam, Menit, Detik, MDetik);
  Waktu := IntToStr(Jam)+'-'+IntToStr(Menit)+'-'+IntToStr(Detik);
  EditSPDisp.Text := EditSP.Text;
  StatusBar.Panels[1].Text := ComPort.Port + ' Disconnected';
end;

procedure TForm1.BtnOn1Click(Sender: TObject);
begin
  //Heater 1 On
```

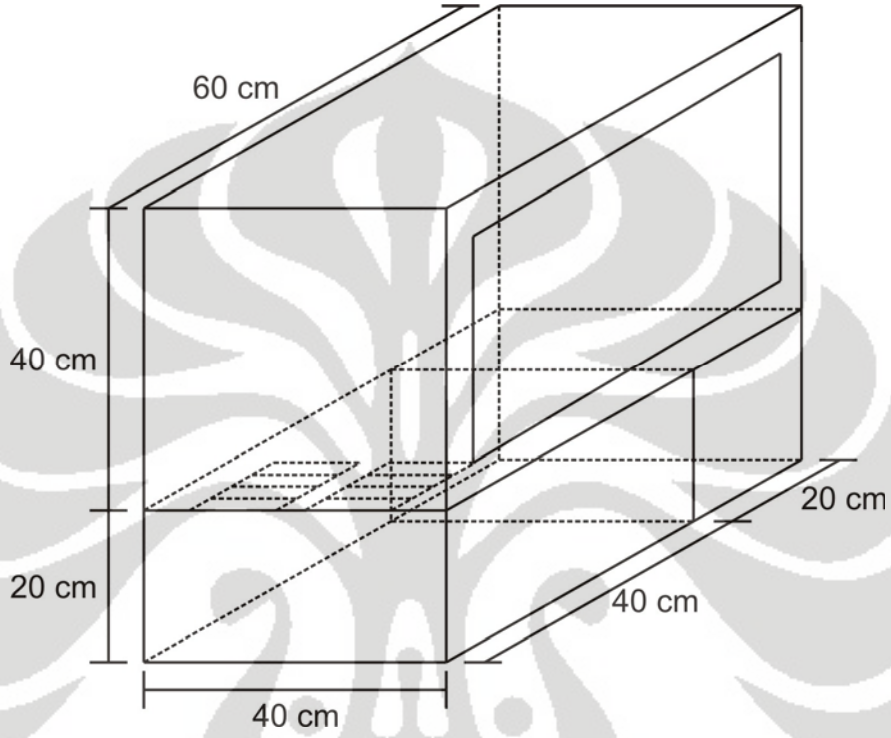
```
  Kondisi(0);
end;

procedure TForm1.BtnOff1Click(Sender: TObject);
begin
  //Heater 1 Off
  Kondisi(6);
end;

procedure TForm1.BtnOn2Click(Sender: TObject);
begin
  //Heater 2 On
  Kondisi(1);
end;

procedure TForm1.BtnOff2Click(Sender: TObject);
begin
  //Heater 2 Off
  Kondisi(7);
end;

procedure TForm1.BtnOn3Click(Sender: TObject);
begin
  //Heater 3 On
  Kondisi(2);
end;

procedure TForm1.BtnOff3Click(Sender: TObject);
begin
  //Heater 3 Off
  Kondisi(8);
end;

procedure TForm1.BtnOn4Click(Sender: TObject);
begin
  //Blower 1 On
  Kondisi(3);
end;

procedure TForm1.BtnOff4Click(Sender: TObject);
begin
  //Blower 1 Off
  Kondisi(9);
end;

procedure TForm1.BtnOn5Click(Sender: TObject);
begin
  //Blower 2 On
  Kondisi(4);
end;

procedure TForm1.BtnOff5Click(Sender: TObject);
begin
  //Blower 2 Off
  Kondisi(10);
end;

procedure TForm1.BitBtn9Click(Sender: TObject);
begin
  //Semuanya On
  Kondisi(5);
end;

procedure TForm1.BitBtn10Click(Sender: TObject);
begin
  //Semuanya Off
  Kondisi(11);
end;
```

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  //Timer1.Interval := StrToInt(EditCount.Text);
end;

procedure TForm1.Timer2Timer(Sender: TObject);
begin
  //Untuk melakukan simulasi
  inc(k);
  Label9.Caption := IntToStr(k);
end;

procedure TForm1.CBH1Click(Sender: TObject);
begin
  //
end;

procedure TForm1.CBH2Click(Sender: TObject);
begin
  //
end;

procedure TForm1.CBH3Click(Sender: TObject);
begin
  //
end;

procedure TForm1.CBB1Click(Sender: TObject);
begin
  //
end;

procedure TForm1.CBB2Click(Sender: TObject);
begin
  //
end;

procedure TForm1.BtnSPClick(Sender: TObject);
begin
  //Setting nilai SP
  EditSPDisp.Text := EditSP.Text;
  SP := StrToInt(EditSP.Text);
  AturKondisi;
end;

procedure TForm1.BtnSaveClick(Sender: TObject);
var
  I, x: integer;
begin
  //Save command
  //Save File to myfile
  //Save file to excel
  x := MemoCh1.Lines.Count;
  FStream := TFileStream.Create('data\'+Tanggal+' '+Waktu+'.xls', fmCreate);
  try
    XlsBeginStream(FStream, 0);
    XlsWriteCellLabel(FStream,    0,    0,    'Hasil    data    pengamatan    sebanyak
'+IntToStr(CounterData)+' data.');
    XlsWriteCellLabel(FStream, 0, 1, Tanggal+' '+Waktu);
    For I := 0 to x do
    begin
    XlsWriteCellLabel(FStream,  0,  2,  'Data');  //XlsWriteCellLabel(FStream,  Column,
Row, 'No')
    XlsWriteCellNumber(FStream, 0, I+2, I);
    XlsWriteCellLabel(FStream, 1, 2, 'Temp. 1');
    XlsWriteCellLabel(FStream, 1, I+2, MemoCh1.Lines.Strings[x]);
    XlsWriteCellLabel(FStream, 2, 2, 'Temp. 2');
```

```
    XlsWriteCellLabel(FStream, 2, I+2, MemoCh2.Lines.Strings[x]);
    XlsWriteCellLabel(FStream, 3, 2, 'Teg. 1');
    XlsWriteCellNumber(FStream, 3, I+2, VoltCh1);
    XlsWriteCellLabel(FStream, 4, 2, 'Teg. 2');
    XlsWriteCellNumber(FStream, 4, I+2, VoltCh2);
    end;

    XlsEndStream(FStream);
  finally
    FStream.Free;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  StatusBar.Panels[2].Text := TimeToStr(Time);

end;

procedure TForm1.BtnStartTimeClick(Sender: TObject);
begin
  Timer2.Enabled := True;
end;

procedure TForm1.BtnStopTimeClick(Sender: TObject);
begin
  TImer2.Enabled := False;
end;

procedure TForm1.BtnResetClick(Sender: TObject);
begin
  k := 0;
  Label9.Caption := '0';
end;

end.
```

LAMPIRAN C

DIMENSI INKUBATOR



BAHAN:

- ✓ *Heater* menggunakan 3 unit lampu Phillips 100 W.
- ✓ *Blower* menggunakan 2 unit kipas 12 V/0,15 A.
- ✓ Bahan terbuat dari *acrilyc* dengan tebal 3 mm.