PERANCANGAN KONTROL GERAK *LEADSCREW* UNTUK *PROGRAMMABLE LOGIC CONTROLLER* BERBASIS MIKROKONTROLER

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

NAUFAL AWANDA PUTRA
NIM. 115060301111015 - 63

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG

2015

LEMBAR PERSETUJUAN

PERANCANGAN KONTROL GERAK *LEADSCREW* UNTUK *PROGRAMMABLE LOGIC CONTROLLER* BERBASIS MIKROKONTROLER

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

NAUFAL AWANDA PUTRA
NIM. 115060301111015 - 63

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing :

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Nanang Sulistiyanto, M.T.	**Akhmad Zainuri, S.T., M.T.**
NIP. 19700113 199403 1 002	**NIP. 19840120 201212 1 003**

LEMBAR PENGESAHAN

PERANCANGAN KONTROL GERAK *LEADSCREW* UNTUK *PROGRAMMABLE LOGIC CONTROLLER* BERBASIS MIKROKONTROLER

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

NAUFAL AWANDA PUTRA
NIM. 115060301111015 - 63

Skripsi ini telah diuji dan dinyatakan lulus sidang skripsi
pada tanggal 1 Juli 2015

Dosen Penguji I	Dosen Penguji II
Mochammad Rif'an, S.T., M.T.	**Dr. Eng. Panca Mudjirahardjo, S.T., M.T.**
NIP. 19710301 200012 1 001	**NIP. 19700329 200012 1 001**

Dosen Penguji III

Dr. Ir. Ponco Siwindarto, M.Eng, S.c
NIP. 19590304 198903 1 001

Mengetahui
Ketua Jurusan Teknik Elektro

M. Aziz Muslim, S.T., M.T., Ph.D.
NIP. 19741203 200012 1 001

RINGKASAN

Naufal Awanda Putra, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Juli 2015, *Perancangan Kontrol Gerak Leadscrew untuk Programmable Logic Controller berbasis Mikrokontroler*, Dosen Pembimbing : Nanang Sulistiyanto, Akhmad Zainuri.

Penggerak-penggerak dalam proses manufaktur lebih banyak menggunakan perangkat yang dikendalikan oleh sebuah sistem mikroprosesor. Mikroprosesor dengan pesat menggantikan perangkat kontrol operasi mekanikal dan secara umum digunakan sebagai fungsi kontrol. Jenis mikroprosesor yang lebih banyak diadaptasi adalah *Programmable Logic Controller* atau disebut PLC. Aktuator linier berupa gabungan motor stepper dan *leadscrew* banyak digunakan dalam proses industri. Dengan kombinasi ini, posisi dan pergerakan repetisi dari *shaft leadscrew* dapat ditentukan secara presisi.

PLC dapat mengendalikan motor stepper dengan bantuan sebuah *driver*. Pada penelitian ini, mikrokontroler digunakan sebagai sebuah rangkaian pengatur *driver* motor stepper yang menyimpan parameter posisi dan kecepatan dari gerakan *leadscrew.* Dengan menggunakan mikrokontroler sebagai pengatur *driver*, maka agar pemrograman pada sistem PLC akan lebih mudah. Data setter digunakan sebagai perangkat eksternal untuk menyimpan parameter. *Data setter* dapat menyimpan dan mengirimkan parameter posisi dan kecepatan untuk mempermudah sistem PLC dalam menggerakan *leadscrew* pada banyak jalur manufaktur. Dengan begitu, perintah yang dikirimkan PLC hanya berupa sinyal kontrol dan register posisi.

Hasil pengujian menunjukkan *leadscrew* dapat bergerak sesuai dengan perintah dari PLC. Dari keseluruhan *range* frekuensi yang diuji, dapat disimpulkan semakin jauh perpindahan *shaft leadscrew* maka semakin besar *error* yang dihasilkan. Rata-rata *error* terkecil berada pada *range* frekuensi antara 125 - 250 Hz. E*rror* maksimum atau ketelitian adalah 2,2 mm pada perpindahan *shaft leadscrew* sejauh 6 inci (152,4 mm) dan resolusi dari alat adalah 87 pulsa.

Kata Kunci : aktuator linier, mikrokontroler, motor stepper, PLC

SUMMARY

Naufal Awanda Putra, Electrical Engineering Department, Engineering Faculty Brawijaya University, July 2015, *Microcontroller-Based Leadscrew Motion Control System for Programmable Logic Controller*, academic supervisor: Nanang Sulistiyanto, Akhmad Zainuri.

Propulsion system in manufacturing process at this time mostly use devices that controlled by microprocessor system. Microprocessor rapidly change mechanical operation control device and in general used as control function. One type of microprocessor that mostly adapted is Programmable Logic Controller (PLC). Linear actuator with combination of stepper motor and leadscrew are common use in industrial process. With this combination, position and repetitive movement of leadscrew shaft can be determine accurately.

PLC can drive stepper motor using controller circuit. In this research, microcontroller is used as controlling circuit to store position and speed parameter of leadscrew movement. By using microcontroler as controlling circuit, then PLC programming will be easier. Data setter used as external device to store parameter. Data setter can store and send position and speed parameter to make PLC system easier when it should drive leadscrew in several manufacture line. Thereby, the instruction from PLC just a control signal and position register.

The result of examination show that leadscrew could move properly by instruction from PLC. In conclusion, from all frequency range that tested, the farther the distance that leadscrew shaft move, the larger error that occured. On average the smallest error occured in frequency range 125 – 250 Hz. The biggest error or accuracy is 2,2 mm on displacement of 6 inch (152,4 mm) and resolution is 87 pulse.

Keywords : liniear actuator, microcontroller, stepper motor, PLC

PENGANTAR

Segala puji hanya milik Allah, hanya berkat rahmat Allah skripsi ini dapat terselesaikan tepat pada waktunya. Skripsi yang berjudul Perancangan Kontrol Gerak *Leadscrew* untuk *Programmable Logic Controller* berbasis Mikrokontroler disusun sebagai syarat memperoleh gelar Sarjana Teknik dari Jurusan Teknik Elektro Universitas Brawijaya. Skripsi ini membahas mengenai aplikasi mikrokontroler pada proses industri manufaktur yang penulis amati selama melaksanakan Kuliah Kerja Nyata Praktik. Selama pembuatan skripsi ini banyak pihak yang telah membantu penulis, oleh karena itu dengan bersungguh-sungguh penulis mengucapkan terima kasih yang sebesar-besarnya kepada :

1. Orangtua, bude dan pakde serta seluruh keluarga besar yang telah memberikan doa, nasihat, dan dukungan selama penulis berkuliah.
2. Bapak M. Aziz Muslim selaku Ketua Jurusan, Ibu Nurussa'adah selaku Ketua Kelompok Dosen Keahlian Teknik Elektronika, Bapak Wijono selaku dosen pembimbing, serta semua Bapak Ibu dosen dan staf Jurusan Teknik Elektro Universitas Brawijaya.
3. Bapak Nanang Sulistiyanto dan Bapak Akhmad Zainuri selaku Dosen Pembimbing 1 dan 2 atas segala bimbingan, motivasi, dan kesempatan bertemu yang telah diberikan.
4. Rekan-rekan seperjuangan selama pembuatan skripsi, Swaraka, Rizal, Tegar, Agung, Nurdin, April, Bustanul, Dimas, Azri, Rozi, serta seluruh teman-teman lain di Jurusan atas bantuan yang telah diberikan.
5. Asisten Laboratorium Sistem Digital, Sistem Kontrol, dan Elektronika yang telah memberikan fasilitas selama pelaksanaan pembuatan skripsi.
6. Sahabat, rekan-rekan dan semua pihak yang tidak dapat disebut satu persatu namanya atas semua bantuan yang sangat membantu penulis.

Penulis menyadari skripsi ini masih jauh dari sempurna karena keterbatasan waktu dan ilmu yang ada, segala kritik dan saran yang membangun sangat diharapkan. Semoga skripsi ini dapat bermanfaat untuk pengembangan lebih lanjut.

Malang, Juni 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan ilmu pengetahuan dan teknologi di dunia membawa dampak positif pada dunia industri, salah satunya pada industri manufaktur. Proses pembuatan produk pada industri manufaktur berskala besar membutuhkan kecepatan dan ketepatan yang tinggi. Oleh karena itu, pelaku industri manufaktur beralih menggunakan mesin-mesin serta robot-robot yang dapat melakukan pekerjaan dengan meminimumkan kesalahan manusia. Penggerak-penggerak dalam proses manufaktur lebih banyak menggunakan perangkat yang dikendalikan oleh sebuah sistem mikroprosesor. Mikroprosesor dengan pesat menggantikan perangkat kontrol operasi mekanikal dan secara umum digunakan sebagai fungsi kontrol. Mikroprosesor memiliki keuntungan besar dikarenakan banyaknya kemungkinan jenis program. Di banyak sistem yang sederhana biasanya hanya terdapat sistem mikrokontroler yang berupa mikroprosesor dengan memori terintregasi dalam satu *chip* untuk digunakan dalam tugas tertentu. Jenis mikroprosesor yang lebih banyak diadaptasi adalah *Programmable Logic Controller* (selanjutnya disebut PLC) (Bolton, 2004: 11).

Pada proses industri yang memerlukan ketepatan yang tinggi, pemilihan penggunaan aktuator akan sangat berpengaruh. Aktuator berupa motor stepper banyak digunakan dalam proses industri. Pada sebuah proses tertentu, motor stepper memiliki beberapa kelebihan dibandingkan motor DC biasa di antaranya adalah posisi dan pergerakan repetisinya dapat ditentukan secara presisi (Kenjo, 1984). Motor stepper dapat dimanfaatkan sebagai penggerak pada aktuator linier yang membutuhkan ketepatan posisi. Motor stepper pada umumnya tidak memiliki umpan balik. Motor stepper memiliki torsi yang besar pada kecepatan yang rendah dan torsi yang kecil pada kecepatan yang tinggi. Jika kecepatan motor stepper terlalu besar, motor akan kehilangan torsinya yang mengakibatkan motor kehilangan posisinya.

Pemrograman pada PLC pada dasarnya berfungsi untuk fungsi logika, penghitung, pewaktu, dan aritmatika untuk mengontrol mesin dan proses-proses industri dan pada dasarnya dibuat khusus untuk pemrograman yang mudah (Bolton, 2004: 444). Sebuah PLC dapat mengontrol gerakan motor stepper dengan bantuan sebuah rangkaian pengendali. Rangkaian pengendali berguna untuk menerjemahkan sinyal perintah menjadi gerakan motor dengan karakteristik arus dan tegangan yang sesuai dengan motor

tersebut. Sinyal perintah dari PLC memungkinkan agar motor mulai berputar, berhenti, bergerak searah jarum jam dan sebaliknya. Akan tetapi, kontrol gerak seperti banyaknya putaran motor dan kecepatan motor akan menjadi hambatan untuk pemrograman pada PLC dikarenakan rumitnya perintah yang harus dibuat untuk satu tugas. Dengan membuat rangkaian pengendali yang dapat menerjemahkan sedikit perintah dari PLC akan tetapi dapat memerhatikan kontrol gerak tersebut, maka pemrograman PLC akan menjadi lebih mudah.

Salah satu jenis aktuator linier yang banyak digunakan ialah gabungan antara *leadscrew* dan motor stepper. Kombinasi ini menawarkan pemosisian yang presisi tergantung dari jenis motor stepper dan banyaknya *Turns per Inch* (TPI) dari *leadscrew* yang digunakan. Penggunaan PLC untuk mengontrol gerakan aktuator linier ini memerlukan kontrol gerak berupa posisi dan kecepatan dari *shaft leadscrew* tersebut. Rangkaian pengendali yang digunakan selain memerlukan perhitungan yang tepat dari karaterisitik tegangan dan arus motor yang digunakan, juga perlu menerjemahkan instruksi dari PLC agar *shaft leadscrew* dapat bergerak ke posisi tertentu dengan kecepatan tertentu, sesuai dengan parameter yang telah diatur sebelumnya. Dengan menggunakan sinyal perintah berupa register posisi yang terdiri atas sinyal logika digital dapat disusun sebuah rangkaian pengendali yang akan mengakses data operasi gerakan *shaft leadscrew.* Register posisi berisikan parameter kontrol gerak berupa posisi dan kecepatan dari *shaft leadscrew.*

Rangkaian pengendali yang dirancang berfungsi untuk menerjemahkan sinyal dari PLC berupa sinyal untuk menggerakkan motor stepper searah atau berlawanan arah jarum jam, menghentikan gerakan, memulai gerakan dengan mempertimbangkan sinyal register posisi dan sinyal untuk kembali ke titik referensi. Rangkaian pengendali kemudian akan menerjemahkan sinyal register posisi menjadi gerakan motor yang sesuai dengan parameter kontrol gerak yaitu posisi *shaft leadscrew* relatif dari titik referensi dan kecepatan *shaft leadscrew* untuk mencapai posisi tersebut. Selain itu rangkaian pengendali juga dirancang agar mengeluarkan sinyal status berupa status gerakan motor, sinyal penanda motor siap dioperasikan, dan sinyal penanda bahaya. Sinyal status tersebut berguna sebagai sinyal bagi PLC untuk melihat posisi *shaft leadscrew* dan juga untuk berkomunikasi dengan perangkat lainnya dalam industri.

Penggunaan peralatan yang praktis memudahkan pelaku industri dalam mengontrol sebuah proses industri berskala besar. Proses industri manufaktur yang memiliki banyak jalur proses seringkali memiliki beberapa mesin yang melakukan hal yang sama.

Penggunaan rangkaian pengendali aktuator linier yang dikendalikan oleh PLC akan menjadi lebih mudah apabila terdapat fitur penyimpanan parameter kontrol gerak yang terpisah. Oleh karena itu diperlukan sebuah peralatan yang dapat berfungsi sebagai penyimpan parameter kontrol gerak yang disebut *data setter*.

Dengan mempertimbangkan kebutuhan untuk menggerakan *shaft leadscrew* untuk menuju posisi tertentu dengan kecepatan tertentu, maka pilihan yang tepat ialah menggunakan perantara berupa mikrokontroler antara PLC dengan rangkaian pengendali. Mikrokontroler bertugas untuk menerima sinyal perintah dari PLC kemudian menerjemahkannya menjadi perintah untuk menggerakan *shaft leadscrew* sesuai dengan parameter kontrol gerak yang telah diatur sebelumnya menggunakan *data setter*. Mikrokontroler akan mengeluarkan sinyal status dari gerakan *shaft leadscrew*. Motor stepper bergerak dengan memanfaatkan pulsa *step*. Aktuator linier yang berupa gabungan dari motor stepper dan *leadscrew* memungkinkan perubahan posisi dari *shaft leadscrew* dapat ditentukan dengan memberikan jumlah pulsa *step* yang sesuai dan juga frekuensi pulsa *step* akan memengaruhi kecepatan dari *shaft leadscrew* tersebut. Mikrokontroler merupakan solusi yang tepat untuk permasalahan tersebut.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, dapat disusun rumusan masalah sebagai berikut :

1. Bagaimana merancang sistem aktuator linier menggunakan motor stepper dan *leadscrew.*
2. Bagaimana menghubungkan sistem mikrokontroler dengan PLC.
3. Bagaimana merancang sistem mikrokontroler *data setter*.
4. Bagaimana merancang sistem mikrokontroler pengatur *driver* motor stepper.

1.3 Batasan Masalah

Mengacu pada rumusan masalah yang telah dirumuskan, maka hal-hal yang berkaitan dengan sistem akan diberi batasan sebagai berikut :

1. PLC yang digunakan adalah merk OMRON tipe CP1L-L20D dengan tegangan pin masukkan 24 V sejumlah 12 pin dengan *common* negatif dan tegangan pin keluaran 5 V sejumlah 8 pin dengan *common* positif.
2. Motor stepper yang digunakan berjenis unipolar dengan tegangan tiap fasa 5,4 VDC dan jumlah *step* per revolusi adalah 200 SPR *(Step per Revolution)*.

3. Mekanik aktuator linier berupa *leadscrew* yang memiliki diameter 8 mm, panjang efektif 6 inci dan jumlah putaran per inci sebesar 5 TPI *(Turns per Inch)*.
4. Frekuensi pulsa *step* motor stepper dibatasi untuk tiga frekuensi yaitu 166,67 Hz, 200 Hz dan 250 Hz, ketelitian maksimum alat dibatasi untuk jarak perpindahan 0,1 inci (2,54 mm) atau sebanyak 100 pulsa *step*.

1.4 Tujuan

Tujuan penelitian ini adalah merancang sebuah sistem kontrol gerak aktuator linier berupa gabungan *leadscrew* dan motor stepper berbasis mikrokontroler yang dikendalikan oleh PLC. Parameter kontrol gerak yaitu posisi dan kecepatan, diatur dan disimpan pada *data setter* yang kemudian ditransfer ke mikrokontroler pengendali motor stepper. Sistem diharapkan mampu bekerja sesuai parameter yang telah ditentukan dengan ketelitian 0,1 inci (2,45 mm).

1.5 Manfaat

Kontribusi utama dalam perancangan dan pembuatan sistem ini ialah memenuhi kebutuhan pembelajaran masalah-masalah yang terjadi dalam aplikasi mikrokontroler pada proses industri manufaktur. Aplikasi mikrokontroler diperlukan untuk mengontrol sistem yang lebih kecil dimana PLC mengendalikan proses yang lebih luas. Dengan aplikasi mikrokontroler pada pengontrolan motor stepper, maka sistem yang dikendalikan oleh PLC akan lebih mudah.

BAB II
TINJAUAN PUSTAKA

Tinjauan pustaka memuat teori-teori penunjang yang digunakan dalam penelitian. Untuk dapat memahami prinsip kerja maupun dasar-dasar perancangan alat atau sistem yang akan dibuat, diperlukan penjelasan dan uraian dari teori yang sudah ada dan telah diuji kebenarannya. Teori penunjang yang akan dijelaskan dalam bab ini adalah , aktuator linier, *Programmable Logic Controller*, mikrokontroler, motor stepper, dan EEPROM.

2.1 Aktuator Linier

Dalam mendesain peralatan otomatis, banyak faktor yang perlu diperhatikan, mulai dari *layout* jalur produksi, lingkungan instalasi, kemudahan perawatan, konfigurasi dari pengkabelan dan sistem kontrol, dan sebagainya. Diperlukan waktu yang banyak untuk memilih motor dan komponen mekanik lain, pembuatan *part list*, *drawing*, prosedur operasi, dan lainnya. Penggunaan aktuator linier dapat mengurangi waktu untuk membuat desain ini. Aktuator linier yang menggunakan motor stepper menawarkan kelebihan dalam pengontrolan sistem dibandingkan sistem aktuator hidraulik dan *pneumatic* (Oriental Motor, 2015), kelebihan tersebut antara lain :

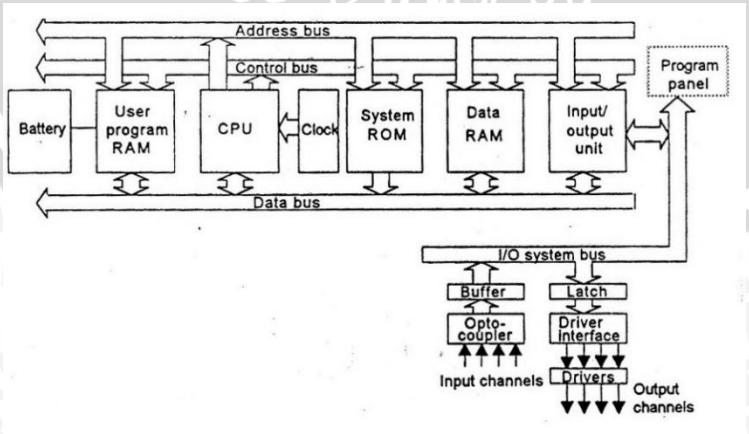
1. Aktuator linier dapat dioperasikan dengan sangat stabil, bahkan pada kecepatan yang rendah. Aktuator linier juga menawarkan operasi akselerasi dan deselerasi yang halus.
2. Pergerakan dapat diprogram dengan titik berhenti yang banyak.
3. Dengan menggunakan motor stepper sebagai aktuator linier, pengaturan posisi dan kecepatan dapat diatur dengan mudah menggunakan data. Konfigurasi juga mudah dengan adanya pengaturan data.

Leadscrew merupakan salah satu elemen transmisi daya yang paling sederhana dan banyak digunakan dalam mesin presisi. Sebagian besar dari mesin perkakas menggunakan *leadscrew* untuk mengubah gerakan putaran motor menjadi gerak linier (Callister, 2007). Motor stepper bersama dengan *leadscrew* akan membentuk sebuah sistem aktuator linier yang presisi. Banyaknya putaran yang diperlukan untuk menggerakan *shaft leadscrew* sejauh 1 inci dinyatakan dalam *Turns per Inch* (disingkat TPI). Jadi resolusi dan tingkat presisi satu *step* pemberian energi pada motor stepper dipengaruhi oleh TPI. Gambar 2.1 menunjukkan aktuator linier menggunakan *leadscrew.*

Gambar 2.1 Aktuator Linier menggunakan *Leadscrew*
Sumber: DNC Labs, 2013

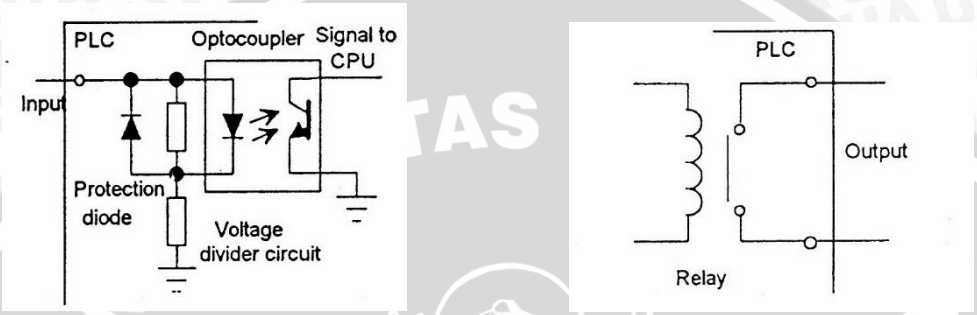
2.2 *Programmable Logic Controller*

Programmable Logic Controller (disingkat PLC) adalah sebuah perangkat kontrol berbasis mikrokontroler yang menggunakan memori yang diprogram untuk menyimpan instruksi dan fungsi implementasi seperti logika, sekuensial, penghitung, pewaktu, dan fungsi aritmatika untuk mengontrol mesin dan proses dan didesain untuk menjadikan pemrograman mudah. Kata logika digunakan karena pemrograman difokuskan untuk mengimplementasikan logika dan operasi pensaklaran. Devais masukkan, contoh saklar, dan devais keluaran, contoh motor, dikendalikan dan dihubungkan ke PLC dan pengontrol memonitor masukkan dan keluaran sesuai dengan program yang disimpan pada PLC oleh operator yang kemudian mengontrol mesin dan proses. Pada mulanya, PLC digunakan untuk menggantikan *relay* dan *timer* pada sistem kontrol logika. PLC memiliki keuntungan besar yaitu dimungkinkan untuk memodifikasi sistem kontrol tanpa harus mengulang pengkabelan antara masukkan dan keluaran sistem, satu-satunya yang diperlukan yaitu operator harus memiliki set instruksi. Arsitektur dari PLC ditunjukkan pada Gambar 2.2.



Gambar 2.2 Arsitektur PLC
Sumber: Bolton, 2004 : 445

Program pada RAM dapat diganti oleh pengguna. Setelah program dituliskan pada RAM, program kemudian ditransfer ke EEPROM untuk menjadi permanen. Spesifikasi dari PLC biasanya ditentukan oleh besarnya memori untuk menyimpan program. Unit masukkan/keluaran memberikan antarmuka antara sistem dan dunia luar. Kanal masukkan/keluaran biasanya diisolasi agar sensor atau aktuator dapat dihubungkan tanpa ada rangkaian tambahan. Gambar 2.3 menunjukkan contoh rangkaian kanal masukkan dan keluaran.

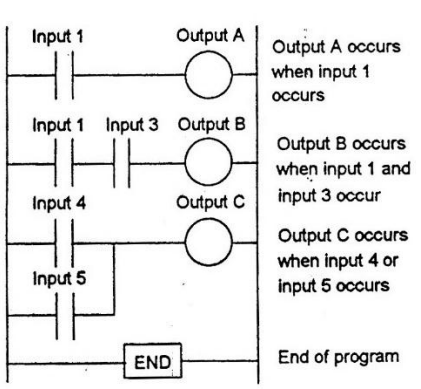


(a) (b)

Gambar 2.3 Antarmuka Kanal (a) masukkan, (b) keluaran PLC
Sumber: Bolton, 2004: 446

Pemrograman dalam PLC menggunakan *ladder diagram* yang merupakan penulisan program yang menyerupai cara menggambar rangkaian pensaklaran. *Ladder diagram* terdiri dari dua garis vertikal yang merepresentasikan rel daya. Rangkaian dihubungkan pada garis horizontal di antara garis vertikal yang disebut rung. Gambar 2.4 menujukkan contoh *ladder diagram*. Pada pemrograman *ladder diagram*, paling tidak terdapat satu masukkan dan satu keluaran pada setiap *rung*. Setiap rung harus diawali dengan masukkan dan diakhiri dengan keluaran. Selain operasi logika, PLC juga dapat mengolah instruksi lain yaitu pewaktu *(timer)* dan penghitung *(counter).* Set instruksi dari jenis dan pabrik PLC satu sama lain berbeda-beda. (Bolton, 2004: 444 – 460). PLC serupa dengan komputer hanya saja memiliki spesifikasi sebagai pengontrol yaitu :

1. Tahan terhadap lingkungan kasar seperti getaran, temperatur, kelembaban dan gangguan.
2. Antarmuka antara masukkan dan keluaran ada di dalam pengontrol.
3. Mudah diprogram dan bahasanya mudah dipahami. Pemrograman difokuskan kepada operasi logika dan pensaklaran.



Gambar 2.4 *Ladder Diagram*
Sumber: Bolton, 2004: 449

PLC yang digunakan adalah merk Omron tipe CP1L-L20DT1-D yang memiliki blok terminal masukkan dan keluaran sebanyak 20 titik yang terdiri dari 12 titik masukkan dan 8 titik keluaran. Rangkaian internal PLC diisolasi dari sistem luar, terminal masukkan dan keluaran PLC memiliki *common* sendiri. Rangkaian internal dari terminal masukkan memiliki spesifikasi yang berbeda tergantung dari fungsi yang diberikan. Terminal masukkan memiliki tegangan masukkan sebesar 24 VDC dengan tegangan ON minimum 17 VDC untuk CIO 0.00 – 0.09 dan 14,4 VDC untuk CIO 0.10 – 0.11. Terminal masukkan memiliki sebuah *common* yang dihubungkan ke *ground*. Rangkaian internal dari terminal keluaran bertipe transistor tipe *sourcing* sehingga diperlukan *common* berupa tegangan positif.

2.3 Mikrokontroler

Mikrokontroler adalah sebuah sistem mikroprosesor di mana di dalamnya sudah terdapat CPU, ROM, RAM, I/O, Clock, dan peralatan internal lainnya yang sudah saling terhubung dan terorganisasi (teralamati) dengan baik oleh pabrik pembuatnya dan dikemas dalam satu chip yang siap pakai. Pengguna hanya perlu memprogram isi ROM sesuai aturan penggunaan oleh pabrik pembuatnya (Winoto, 2010 : 3). Arsitektur mikrokontroler keluarga AVR secara luas terdiri atas ALU *(Arithmetic Logic Unit)*, program memori, *program counter*, *general purpose working register*, *static random access memory*, dan *internal peripheral*.

1. ALU *(Arithmetic Logic Unit)* adalah *processor* yang bertugas mengeksekusi (eksekutor) kode program yang ditunjuk oleh programn counter.
2. Program memori adalah memori FLASH EPROM yang bertugas menyimpan program (software) yang kita buat dalam bentuk kode-kode program (berisi

alamat memori beserta kode program dalam ruangan memori alamat tersebut) yang telah kita compile berupa bilangan heksa atau biner.

3. *Program Counter* (PC) adalah komponen yang bertugas menunjukkan ke ALU alamat program memori yang harus diterjemahkan kode programnya dan dieksekusi. Sifat dari PC adalah linier artinya dia menghitung naik satu bilangan yang bergantung alamat awalnya.
4. 32 *General Purpose Working Register* (GPR) adalah register file atau register kerja yang mempunyai ruangan 8-bit. Tugas GPR adalah tempat ALU mengeksekusi kode-kode program, setiap instruksi dalam ALU melibatkan GPR.
5. *Static Random Access Memory* (SRAM) adalah RAM yang bertugas menyimpan data sementara sama seperti RAM pada umumnya mempunyai alamat dan ruangan data. Alamat terakhir dari SRAM bergantung pada kapasitas SRAM. Dalam bahasa c, pembuatan stack menjadi tanggungan compiler.
6. *Internal Pheripheral* adalah peralatan/modul internal yang ada dalam mikrokontroler seperti saluran I/O, Interupsi eksternal, Timer/Counter, USART, EEPROM, dan lain-lain. Tiap peralatan internal mempunyai register port (register I/O) yang mengendalikan peralatan internal tersebut.

AVR memiliki arsitektur harvard di mana bus memori program dan bus memori data terpisah, sehingga dapat mengakses memori data dan memori program dalam satu waktu. Memori dalam mikrokontroler sangat terbatas sehingga diperlukan manajemen memori dengan baik. Memori ATmega terbagi tiga yaitu :

1. Memori Flash – memori ROM tempat kode-kode program berada. Kata flash menunjukkan jenis ROM yang dapat ditulis dan dihapus secara elektrik. Memori flash terbagia dua bagian, yaitu bagian aplikasi dan bagian boot. Bagian aplikasi adalah bagian kode-kode program aplikasi berada. Bagian boot adalah bagian yang digunakan khusus untuk booting awal yang dapat diprogram untuk menulis bagian aplikasi tanpa melalui programmer/downloader, misalnya melalui USART.
2. Memori Data – memori RAM yang digunakan untuk keperluan program. Memori data terbagi tiga bagian yaitu GPR, I/O register, dan Aditional I/O register.
3. EEPROM – memori data yang dapat mengendap ketika chip mati, digunakan untuk keperluan penyimpanan data yang tahan terhadap gangguan catu daya.

Eksekusi aliran program mikrokontroler dengan bahasa C dimulai dari fungsi main dan terus ke bawah, adapun lompat/panggil fungsi lain di luar fungsi main tergantung pada instruksi/pernyataan dalam fungsi main. Setiap peripheral mikrokontroler dilengkapi flag/bit status kejadian tertentu. Flag tersebut dapat digunakan untuk membangkitkan sistem interupsi. Sistem interupsi menghentikan aliran program akibat terjadinya triger tertentu dan memaksa eksekusi rutin/fungsi layanan interupsi, setelah selesai maka aliran program akan kembali ke pernyataan program sebelum terjadinya interupsi. Urutan prioritas interupsi yang paling tinggi adalah RESET, jika terjadi interupsi bersamaan, maka urutan prioritas tertinggi didahulukan sedangkan sisanya menunggu hingga interupsi prioritas tinggi selesai.

Interupsi eksternal adalah sebuah pheripheral dalam chip yang bertugas mendeteksi triger dari luar yang akan membangkitkan interupsi yang bersangkutan. Interupsi eksternal chip disediakan tiga buah pada pin INT0, INT1, dan INT2 dimana triger yang digunakan dapat dipilih sendiri (tepi turun, tepi naik atau logika rendah). Pada mikrokontroler ATmega1284P terdapat interupsi eksternal bernama *Pin Change Interrupt* (PCINT) sejumlah 32 buah yang terdapat pada setiap port I/O. PCINT akan dibangkitkan ketika pin yang ditunjuk *toggle*.

2.4 Motor Stepper

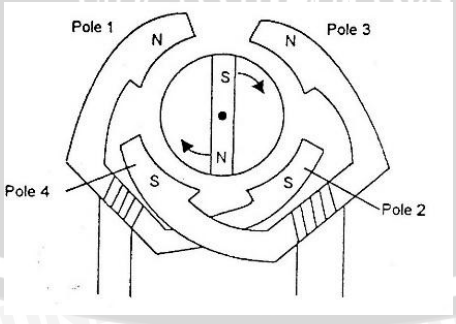
Motor stepper merupakan sebuah devais yang dapat menghasilkan gerak rotasi dengan sudut yang sama, yang disebut *step*, untuk setiap pulsa digital yang diberikan ke masukkan. Sebagai contoh, jika sebuah motor dalam 1 pulsa menghasilkan gerak rotasi sebanyak 6^o maka dalam 60 pulsa akan dihasilkan gerak rotasi sebanyak 360^o (Bolton, 2004 : 174). Motor stepper banyak digunakan pada aplikasi pengukuran dan kontrol. Berapa fungsi yang dimiliki motor stepper menjadikannya sesuai untuk aplikasi tersebut, yaitu :

1. *Brushless* – motor stepper merupakan motor berjenis brushless yang memiliki keunggulan daripada motor bersikat biasa.
2. Bebas beban – motor stepper akan bergerak pada kecepatan yang sama selama tidak melebihi torsi maksimum motor.
3. Pemosisi *loop* terbuka – motor stepper bergerak dengan *step* yang terukur. Selama motor bergerak dengan torsi yang sesuai spesfikasi, posisi dari *shaft* dapat diketahui tanpa menggunakan mekanisme umpan balik.
4. *Holding Torque* – motor stepper dapat menahan *shaft* tanpa bergerak.

5. Respon yang baik – untuk memulai gerakan, berhenti dan berputar ke arah sebaliknya.

Terdapat tiga jenis dasar motor stepper, yaitu *permanent magnet, variable reluctance,* dan *hybrid*. Motor *permanent magnet* memiliki rotor magnet, sedangkan motor *variable reluctance* memiliki rotor gigi besi lunak. Motor *hybrid* merupakan gabungan dari teknologi permanent magnet dan *variable reluctance*. Pada pemilihan jenis motor, ada beberapa keputusan yang perlu dipikirkan diantaranya adalah torsi, lingkungan pengoperasian, usia pakai, dimensi fisik, besar langkah, RPM (Rotation per Minute) maksimum. Salah satu yang paling penting sekali pada pemilihan motor yaitu besar langkah (Condit & Jones, 2004 :6).

Gambar 2.5 menunjukkan bentuk dasar dari motor stepper magnet permanen. Motor memiliki stator dengan 4 kutub. Masing-masing kutub dililit oleh medan lilit, koil pasangan yang berlawanan kutub dipasang secara seri. Arus disuplai dari sumber DC ke lilitan melalui saklar. Rotor adalah magnet permanen dan ketika arus berganti pada pasangan stator, rotor akan dengan sejajar mengikuti. Ketika arus diberikan, rotor akan bergerak sejauh 45^0. Jika arus menjadi sebaliknya lalu berganti, maka rotor akan bergerak sejauh 45^0 untuk kembali sejajar lagi. Maka dengan melewatkan arus ke koil maka rotor akan bergerak sejauh 45^0. Dengan jenis motor seperti ini, maka besar langkah biasanya bervariasi $1{,}8^0$, $7{,}5^0$, $1{,}5^0$, 30^0, 34^0 atau 90^0 (Bolton, 2004: 178). Untuk bergerak secara linier, banyak motor stepper dihubungkan dengan leadscrew (dikenal sebagai aktuator linier). dengan begitu akan didapatkan gerakan yang halus walaupun besar langkah yang digunakan tergolong besar.



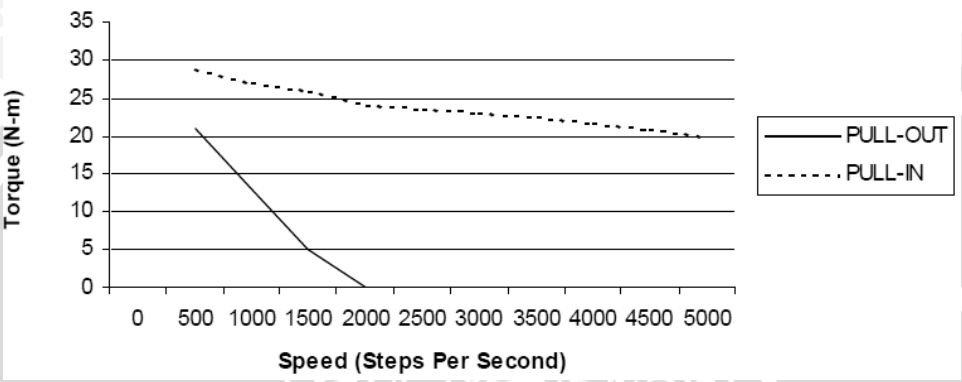
Gambar 2.5 Motor Stepper Magnet Permanen
Sumber: Bolton, 2004: 179

Motor stepper memiliki beberapa jenis nilai torsi, yaitu :

1. *Holding Torque* – torsi yang dibutuhkan untuk memutar *shaft* motor ketika lilitan diberikan energi.

2. *Pull-in Torque* – torsi yang bisa diberikan motor untuk berakselerasi dari berhenti sampai bergerak tanpa kehilangan langkah.
3. *Pull-out Torque* – beban yang dapat digerakan motor ketika beroperasi.
4. *Detent Torque* – torsi yang dibutuhkan untuk memutar *shaft* motor ketika lilitan tidak diberikan energi.

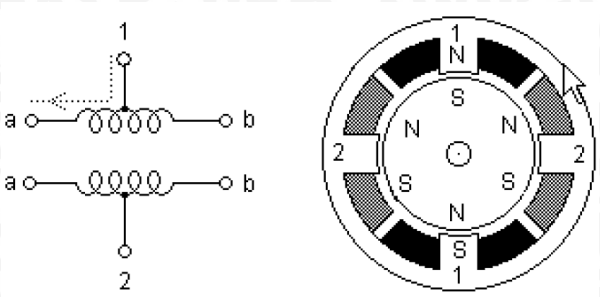
Gambar 2.6 menujukkan kurva torsi-kecepatan yang biasa digunakan dalam menentukan spesifikasi motor stepper yang ingin digunakan. Garis putus-putus menunjukkan besar *Pull-in Torque*, dan garis tebal menunjukkan besar *Pull-out Torque*.



Gambar 2.6 Kurva Torsi-Kecepatan
Sumber: Microchip Technology : 7

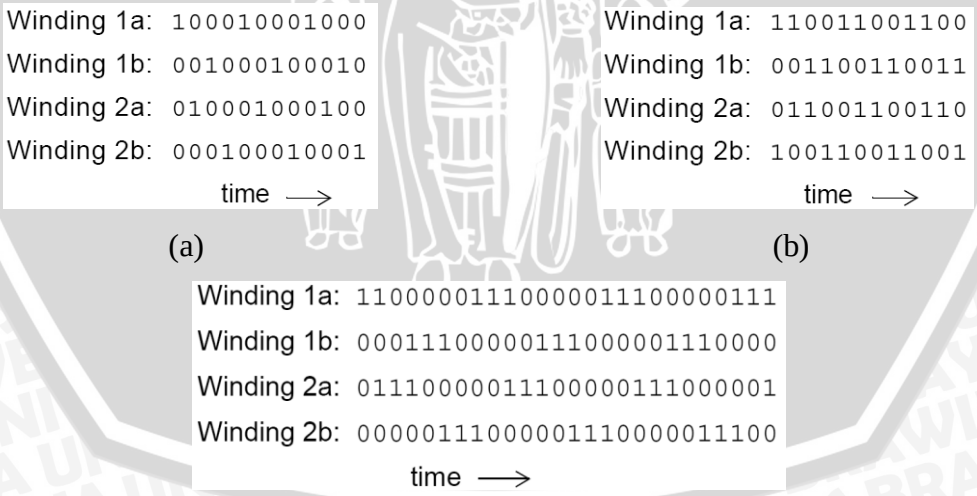
2.4.1 Motor Stepper Unipolar

Motor stepper unipolar terdiri atas dua lilitan yang masing-masing terhubung dengan *center tap*. *Center tap* dihubungkan keluar atau dihubungkan secara internal satu sama lain kemudian dihubungkan keluar menjadi satu kabel. Sebagai hasilnya, motor stepper unipolar terdiri atas 5 atau 6 kabel. *Center tap* terhubung ke *supply* dan ujung lain dari koil terhubung ke *ground* secara bergantian. Motor stepper unipolar seperti motor *permanent magnet* yang lain beroperasi dengan menarik kutub utara atau selatan dari magnet permanen pada rotor ke kutub stator. Maka pada motor ini, arah dari arus yang melewati lilitan stator menentukan kutub rotor yang mana yang akan dipengaruhi kutub stator. Pemberian arus dari motor unipolar bergantung dari setengah lilitan mana yang diberikan energi. Secara fisik, setengah lilitan dililit paralel terhadap satu sama lain, sehingga satu lilitan bereaksi sebagai kutub utara atau selatan tergantung dari setengah lilitan mana yang diberikan energi. Gambar 2.7 menunjukkan lilitan motor stepper jenis unipolar.



Gambar 2.7 Motor Stepper Lilitan Unipolar.
Sumber: Microchip Technology : 2

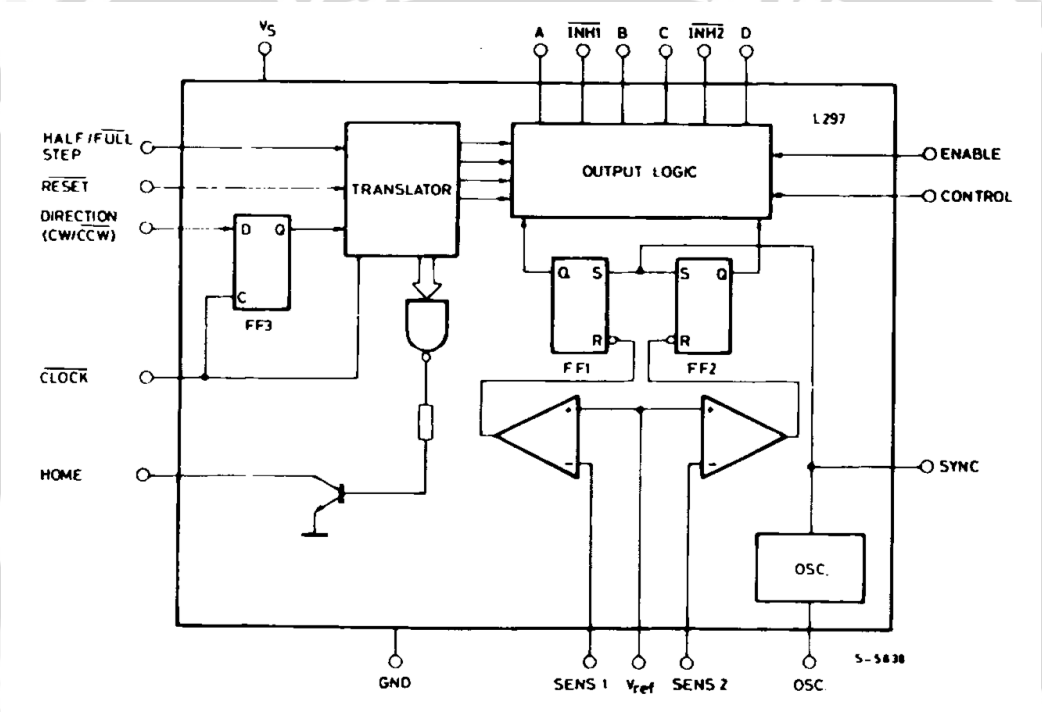
Untuk menggerakan motor stepper unipolar diperlukan urutan pemberian energi lilitan secara bergantian. Terdapat tiga mode yaitu mode *wave drive*, *normal drive*, dan *half step*. Mode *wave drive* memberikan satu fasa energi secara bergantian, mode *normal drive* memberikan dua fasa energi secara bergantian. Mode *normal drive* memberikan torsi yang lebih besar akan tetapi menggunakan energi yang lebih banyak. Mode *half step* menggabungkan mode *normal drive* dan *wave drive* sehingga didapatkan besar langkah setengah dari mode lainnya. Mode *half step* menggunakan torsi dan arus yang tidak konstan karena memberikan energi kepada satu fasa dan dua fasa secara bergantian. Urutan pemberian energi pada lilitan motor dengan mode *normal drive, wave drive* dan *half step* dapat dilihat pada Gambar 2.8



Gambar 2.8 Urutan Pemberian Energi Pada Lilitan
(a) *Wave Drive*, (b) *Normal Drive*, dan (c) *Half Step*
Sumber: Microchip Technology : 2 – 3

2.4.2 IC *Driver* Motor Stepper

IC *driver* motor stepper L297 digunakan untuk mengontrol motor stepper bipolar 2 fasa atau unipolar 4 fasa pada aplikasi menggunakan mikrokontroler. IC ini menghasilkan urutan pengendali fasa motor stepper sehingga memudahkan dalam pemrograman mikrokontroler. Motor dapat dikendalikan dengan mode *half step, normal drive*, dan *wave drive* dengan mengatur pin masukkan yang bersangkutan. Sinyal yang perlu diberikan oleh mikrokontroler untuk mengendalikan motor stepper berupa sinyal *Clock, Direction,* dan Mode *Half/Full step* yang masuk ke IC *driver* motor stepper tersebut. Blok diagram dari IC *driver* motor stepper L297 ditunjukkan pada Gambar 2.9. Fungsi dari masing-masing pin masukkan dan keluaran ditunjukkan pada Tabel 2.1.



Gambar 2.9 Blok Diagram IC *Driver* Motor Stepper L297
Sumber : STMicroelectronics, 2003: 7

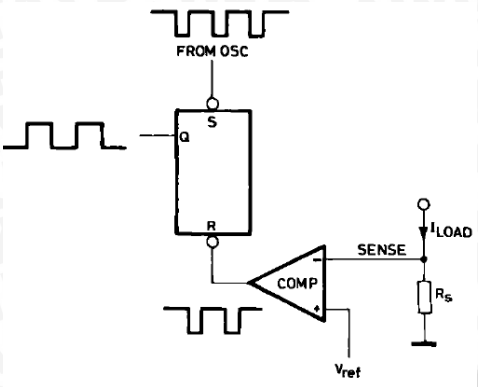
Tabel 2.1 Fungsi Pin *Out* IC *Driver* Motor Stepper L97
Sumber : STMicroelectronics, 2003 : 16

No. Pin	Nama	Fungsi
1	SYNC	sinyal keluaran dari osilator internal, digunakan untuk menyinkronkan sumber pulsa jika menggunakan lebih dari satu IC L297
2	GND	*Ground*
3	HOME	sinyal penanda posisi translator berada pada posisi awal
4	A	sinyal keluaran untuk fasa A

5	INH1	sinyal keluaran untuk mengontrol arus beban pada lilitan motor pada fasa A dan B
6	B	sinyal keluaran untuk fasa B
7	C	sinyal keluaran untuk fasa C
8	INH2	sinyal keluaran untuk mengontrol arus beban pada lilitan motor pada fasa C dan D.
9	D	sinyal keluaran untuk fasa D
10	ENABLE	pin masukkan untuk mengaktifkan keluaran A, B, C, D, INH1, dan INH2
11	CONTROL	pin masukkan untuk mengendalikan aksi *chopper*, jika berlogika rendah, *chopper* aktif pada INH1 dan INH2, jika berlogika tinggi, *chopper* aktif pada A, B, C, dan D
12	Vs	Pin *supply*
13	SENS2	pin masukkan untuk tegangan dari arus beban pada fasa C dan D
14	SENS1	pin masukkan untuk tegangan dari arus beban pada fasa A dan B
15	Vref	tegangan referensi untuk rangkaian chopper yang akan menentukan batas arus maksimal
16	OSC	pin masukkan untuk rangkaian RC penentu besar *chopper* yang digunakan
17	CW/CCW	pin masukkan untuk mengontrol arah gerakan motor stepper
18	CLOCK	pin masukkan pulsa *step*
19	HALF/FULL	pin masukkan untuk mengatur mode operasi motor stepper
20	RESET	pin masukkan untuk mengatur ulang posisi translator ke posisi awal

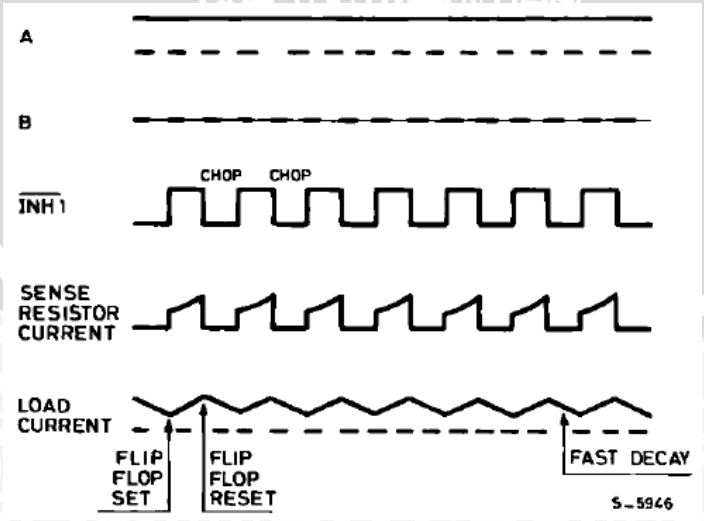
Kelebihan dari penggunaan IC *driver* motor stepper adalah IC ini memilki rangkaian *chopper Pulse Width Modulation* (PWM) yang terintegrasi dalam chip yang digunakan untuk mengendalikan arus dalam lilitan motor. *Chopper* PWM berasal dari flip-flop FF1 dan FF2 (Gambar 2.10) yang akan set dari pulsa osilator, mengakibatkan arus beban meningkat yang akan meningkatkan tegangan yang melewati resistor *sense*, dan flip-flop akan reset ketika tegangan melampaui Vref, mengakibatkan keluaran mati sampai pulsa osilator berikutnya set. Besarnya Rs yang diperlukan untuk menerjemahkan arus yang melewati lilitan motor menjadi tegangan untuk kemudian dibandingkan dengan tegangan Vref sesuai dengan persamaan (2-1) :

$$R_S = \frac{V_{ref}}{I_{Load}} \tag{2-1}$$



Gambar 2.10 Rangkaian *Chopper* PWM
Sumber : STMicroelectronics, 2003: 10

Untuk motor stepper unipolar digunakan *chopper* PWM pada pin INH1 dan INH2 dengan mengatur pin masukkan CONTROL berlogika rendah. Hal ini dikarenakan INH1 dan INH2 akan berlogika rendah (Gambar 2.11) jika *chopper* PWM aktif ketika flip-flop reset, yang mengakibatkan arus sirkulasi ulang dari *ground* menuju Vs. *Chopper* PWM ini diperlukan agar didapatkan pembuangan arus yang berguna untuk mendapatkan kecepatan dan torsi yang diperlukan yang terhambat akibat gaya elektromagnetik pada rotor dari motor magnet permanen. Dengan demikian, setelah mengatur besarnya arus maksimal yang dialirkan ke motor, mengatur mode operasi motor, dan menentukan arah putaran motor, maka devais lain hanya perlu mengubah pin masukkan *Clock* sesuai dengan frekuensi yang diinginkan. IC *driver* kemudian akan mengatur urutan pemberian energi ke motor melalui pin keluaran A, B, C, dan D dan besarnya arus yang sesuai.



Gambar 2.11 Gelombang *Chopper* pada INH1
Sumber : STMicroelectronics, 2003: 12

2.5 EEPROM 24LC64

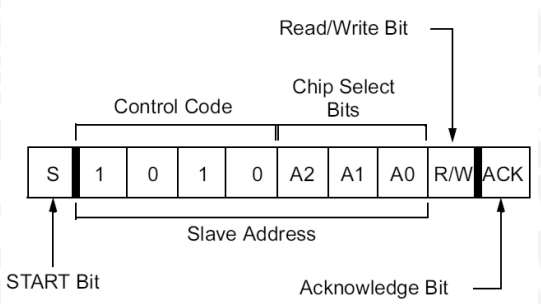
Perkembangan teknologi yang sangat pesat saat ini membuat garis batas antara RAM *(Random Access Memory)* dan ROM *(Read Only Memory)* menjadi kabur. Beberapa jenis memori yang ada saat ini merupakan gabungan antara keduanya. Sebuah memori dapat dibaca dan tulis sesuai dengan keinginan seperti RAM, akan tetapi memelihara data tanpa daya elektrik seperti ROM. EEPROM *(Electrically Erasable Programmable Read Only Memory)* dapat melakukan operasi penghapusan secara elektrik bukan dengan dipaparkan cahaya ultraviolet. Setiap *byte* dari EEPROM dapat dihapus dan ditulis kembali. Sekali ditulis, data baru akan tetap untuk selamanya atau hingga dihapus secara elektris. Kelebihan fungsi ini harus dibayar mahal dikarenakan proses penulisan berlangsung sangat lamban dibandingkan dengan proses penulisan pada RAM (Barr, 2001: 103 - 104).

Jenis EEPROM yang digunakan pada perancangan adalah 24LC64 buatan Microchip yang merupakan EEPROM 64 Kbit. Devais ini tersusun dari delapan unit memori 1 K x 8 bit dengan antarmuka I2C *(Inter Integrated Circuit)*. Desain tegangan rendah mengizinkan beroperasi pada tegangan 1,8 V dengan kondisi standby dan aktif berturut-turut memerlukan arus sebesar 1 µA dan 1 mA. Fungsi tiap pin dari 24LC64 terlihat pada Tabel 2.2.

Tabel 2.2 Fungsi *Pin Out* 24LC64
Sumber: Microchip Technology, 2002: 1

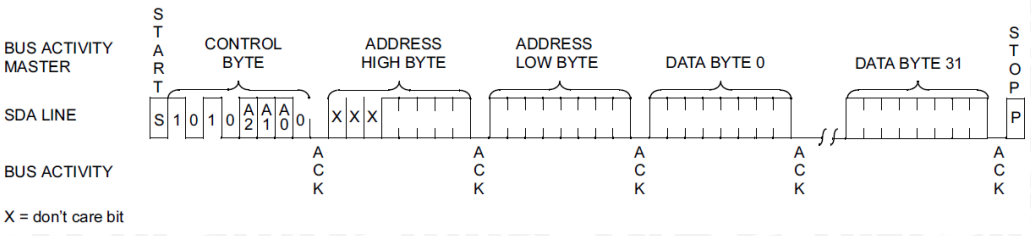
No. Pin	Nama	Fungsi
1	A0	
2	A1	*Input* Alamat Chip
3	A2	
4	V_{SS}	*Ground*
5	SDA	*Serial Address/Data IO*
6	SCL	*Serial Clock*
7	WP	*Write Protect Input*
8	V_{CC}	+1,8 – 5,5 V Power Supply

Control byte merupakan *byte* pertama yang dikirim setelah kondisi start diberikan devais utama. *control byte* terdiri dari empat *bit* kode kontrol, untuk 24LC64, '1010' merupakan kode biner untuk operasi baca dan tulis. Tiga *bit* terakhir adalah selektor alamat chip (A2, A1, A0). Selektor alamat chip memungkinkan penggunaan hingga delapan devais 24LC64 pada bus yang sama dan untuk memilih devais mana yang dialamatkan. Gambar 2.12 menujukkan format data *control byte*.



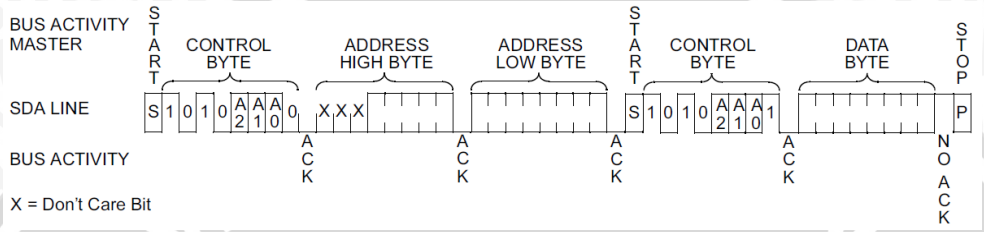
Gambar 2.12 *Control Byte*
Sumber: Microchip Technology, 2002: 6

Proses penulisan data dari devais *master* ke EEPROM ialah sebagai berikut: SDA adalah jalur yang dipakai untuk transmisi data, digunakan bersama oleh *master* dan *slave*. SCL adalah jalur untuk *clock*, yang dihasilkan oleh *master*. Proses penulisan data dimulai dengan *master* mengirim start *bit* diikuti *control byte* yang terdiri dari kode kontrol (empat *bit*), *chip select* (tiga *bit*) dan *bit* R/$\overline{W}$ (yang diset logika rendah untuk proses penulisan), kemudian 24LC64 akan mengirimkan sinyal *acknowledge* yang berlogika rendah pada jalur yang sama pada siklus pulsa ke sembilan. Oleh karena itu pada pulsa ke sembilan, jalur SDA tidak boleh diisi oleh *master*. *Byte* berikutnya digunakan untuk alamat *byte* tinggi, dan *byte* berikutnya digunakan untuk alamat *byte* rendah setelah sinyal *acknowledge* dari 24LC64 pada siklus pulsa ke sembilan. Pada *byte* berikutnya barulah data dapat dituliskan pada alamat yang dituju. Jika proses penulisan berakhir, *master* akan mengirimkan stop *bit* ketika jalur SCL ditahan logika tinggi. Jika tidak, *master* boleh melanjutkan hingga 31 *byte* data tambahan yang akan disimpan sementara pada *buffer* dan ditulis ketika *master* mengirimkan stop *bit*. Gambar 2.13 menujukkan proses penulisan data pada jalur SDA. Pada 24LC64 terdapat pin WP *(write protection)* yang akan mencegah data ditulis ketika pin ini terhubung dengan VCC. Jika pin ini dihubungkan ke VDD maka *write protection* tidak diaktifkan. Proses pembacaan tetap bisa berlangsung walaupun *write protection* diaktifkan.



Gambar 2.13 Proses Penulisan Data pada Jalur SDA
Sumber: Microchip Technology, 2002: 8

Proses pembacaan 24LC64 hampir sama dengan proses penulisan. Hanya saja setelah proses penunjukkan alamat rendah selesai, *master* mengirimkan start *bit* lagi diikuti *control byte*, *chip select* dan *bit* R/$\overline{W}$ yang diset logika tinggi. Setelah itu 24LC64 akan mengirinkan sinyal *acknowledge* diikuti dengan data 8 *bit*. Proses pembacaan berakhir ketika 24LC64 tidak mengirimkan sinyal *acknowledge* yang membuat *master* mengirimkan stop *bit* (Microchip Technology, 2002: 1 – 12). Gambar 2.14 menujukkan proses pembacaan data pada jalur SDA.



Gambar 2.14 Proses Pembacaan Data Pada Jalur SDA
Sumber: Microchip Technology, 2002: 11

BAB III
METODE PENELITIAN

Penyusunan penelitian didasarkan pada masalah yang bersifat aplikatif yaitu perancangan dan implementasi sistem. Untuk menyelesaikan rumusan masalah dan merealisasikan tujuan penelitian yang terdapat pada pendahuluan maka diperlukan metode penelitian. Metode penelitian yang digunakan adalah perancangan dan pembuatan alat, dan pengujian dan analisis.

3.1 Perancangan dan Pembuatan Alat

Perancangan dan pembuatan alat didasari oleh teori yang ada dan telah dibahas pada Bab II. Perancangan dan pembuatan alat ditentukan sesuai dengan batasan masalah untuk kemudian ditentukan spesifikasi alat. Perancangan dan pembuatan alat diawali dengan membuat blok diagram rancangan sistem secara keseluruhan sesuai dengan spesifikasi alat. Dari rancangan sistem akan diperoleh perangkat keras dan perangkat lunak yang diperlukan dalam penelitian. Perancangan dan pembuatan alat dalam penelitian ini dibagi menjadi dua bagian, yaitu perancangan dan pembuatan perangkat keras, dan perangkat lunak.

3.1.1 Perancangan dan Pembuatan Perangkat Keras

Perancangan dan pembuatan perangkat keras terdiri atas dua bagian yaitu perancangan mekanik alat dan perancangan elektrik alat. Perancangan mekanik terdiri atas pembuatan aktuator linier menggunakan *leadscrew* dan motor stepper dan juga peletakan sensor pada aktuator linier. Perancangan elektrik meliputi perancangan rangkaian konversi level tegangan mikrokontroler dengan PLC, rangkaian *driver* motor stepper, rangkaian elektrik mikrokontroler *data setter*, dan rangkaian elektrik mikrokontroler pengatur *driver* motor stepper. Papan rangkaian tercetak *(Printed Circuit Board* disingkat PCB) dirancang dengan menggunakan perangkat lunak EAGLE *(Easily Applicable Graphical Layout Editor)* versi 6.5.

3.1.2 Perancangan dan Pembuatan Perangkat Lunak

Perangkat lunak digunakan untuk memfungsikan perangkat keras sesuai agar didapatkan sistem yang sesuai dengan perancangan alat. Perangkat lunak dirancang melalui pembuatan diagram alir sistem secara keseluruhan dan kemudian masing-masing subsistem. Perancangan perangkat lunak pada terdiri atas tiga bagian, yaitu perancangan perangkat lunak sistem PLC, mikrokontroler *data setter*, dan mikrokontroler pengatur *driver* motor stepper. Perangkat lunak pada sistem PLC dibuat menggunakan program

CX-Programmer versi 8.10. Perangkat lunak pada mikrokontroler dibuat menggunakan program CV AVR *(Code Vision AVR C Compiler)* versi 2.05.0.

3.2 Pengujian Alat

Pengujian alat dilakukan untuk memastikan apakah kinerja alat telah sesuai dengan perancangan. Pengujian dilakukan secara bertahap mulai dari pengujian masing-masing subsistem hingga pengujian sistem secara keseluruhan. Dari hasil pengujian masing-masing subsistem akan diperoleh data yang dapat memengaruhi perancangan alat secara keseluruhan. Secara garis besar pengujian yang dilakukan adalah sebagai berikut :

1. Pengujian aktuator linier.
2. Pengujian rangkaian konversi level tegangan mikrokontroler dengan PLC.
3. Pengujian rangkaian *driver* motor stepper.
4. Pengujian sistem mikrokontroler *data setter.*
5. Pengujian sistem secara keseluruhan.

BAB IV
PERANCANGAN DAN PEMBUATAN ALAT

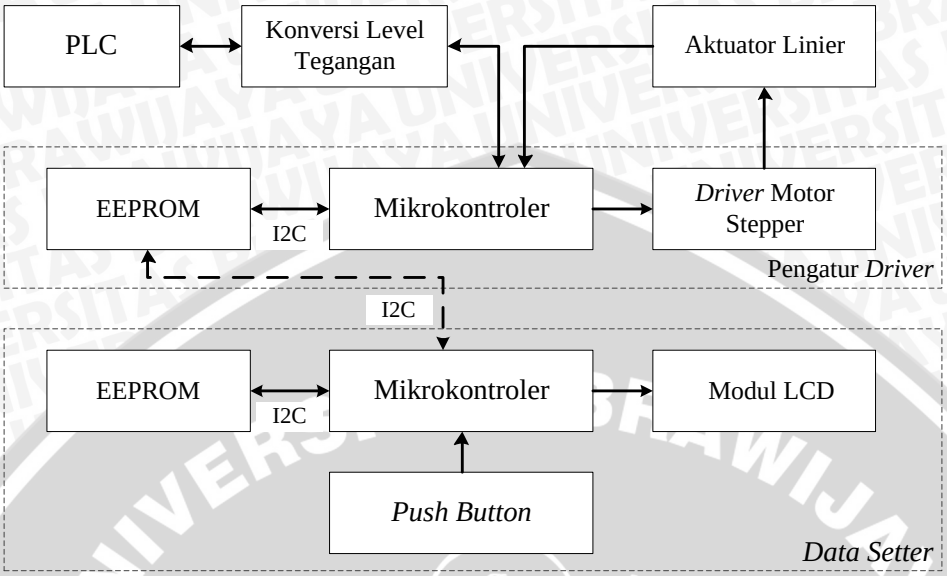
4.1 Perancangan Sistem

Perancangan dan pembuatan alat dilakukan secara bertahap dan dilakukan perbagian agar mudah untuk dianalisis. Perancangan alat terdiri atas perancangan perangkat keras dan perancangan perangkat lunak. Spesifikasi alat ditentukan terlebih dahulu sebagai acuan dalam perancangan alat selanjutnya. Spesifikasi alat yang digunakan adalah sebagai berikut :

1. PLC yang digunakan adalah merk OMRON tipe CP1L-L20D dengan tegangan pin masukkan 24 V sejumlah 12 pin dengan *common* negatif dan tegangan pin keluaran 5 V sejumlah 8 pin dengan *common* positif.
2. Motor stepper yang digunakan berjenis unipolar dengan tegangan tiap fasa 5,4 VDC dan jumlah *step* per revolusi adalah 200 SPR *(Step per Revolution)*.
3. Mekanik aktuator linier berupa *leadscrew* yang memiliki diameter 8 mm, panjang efektif 6 inci (152,4 mm) dan jumlah putaran per inci sebesar 5 TPI *(Turns per Inch)*.
4. Frekuensi pulsa *step* motor stepper dibatasi untuk tiga frekuensi yaitu 166,67 Hz, 200 Hz dan 250 Hz, ketelitian maksimum alat dibatasi untuk jarak perpindahan 0,1 inci (2,54 mm) atau sebanyak 100 pulsa *step*.

Perancangan alat diawali dengan membuat blok diagram sistem secara keseluruhan. Diagram blok sistem secara keseluruhan ditunjukkan dalam Gambar 4.1. Blok sistem yang dirancang terdiri atas tiga bagian utama yaitu rangkaian konversi level tegangan, rangkaian *data setter*, dan rangkaian mikrokontroler pengatur *driver* motor stepper. Rangkaian *data setter* berfungsi sebagai peralatan pengatur dan penyimpan parameter, dan peralatan pengontrol. *Data setter* berkomunikasi dengan sistem mikrokontroler pengatur *driver* motor stepper dengan menggunakan jalur I2C untuk mengubah parameter yang tersimpan pada EEPROM. Parameter yang diatur adalah posisi dan kecepatan antar posisi. PLC mengontrol gerakan *leadscrew* dengan memberikan sinyal ke rangkaian mikrokontroler pengatur *driver* motor stepper. PLC juga menerima sinyal dari mikrokontroler tentang status dari gerakan *leadscrew.* Sinyal yang dikirimkan dan diterima oleh mikrokontroler dari dan ke PLC melewati rangkaian untuk menyesuaikan level tegangan PLC. Dengan demikian PLC dapat mengendalikan leadscrew agar

bergerak sesuai dengan program pada PLC dan PLC dapat mengontrol peralatan lain dengan memanfaatkan sinyal status dari gerakan *leadscrew.*



Gambar 4.1 Blok Diagram Sistem Secara Keseluruhan

Fungsi dari masing-masing bagian pada blok diagram sistem adalah sebagai berikut:

1. PLC berfungsi sebagai pengontrol bagi mikrokontroler pengatur *driver* motor stepper untuk memberikan perintah ke motor stepper yang mengendalikan *leadscrew.*
2. Rangkaian konversi level tegangan berfungsi menerjemahkan sinyal dari level tegangan PLC ke level tegangan mikrokontroler dan sebaliknya.
3. Mikrokontroler pengatur *driver* motor stepper berfungsi sebagai pemroses sinyal kendali motor stepper.
4. *Driver* motor stepper berfungsi sebagai rangkaian pengendali motor stepper agar dapat dikendalikan oleh mikrokontroler.
5. Motor stepper bersama dengan *leadscrew* membentuk aktuator linier, pada aktuator linier terdapat *limit switch* yang berfungsi sebagai umpan balik posisi *shaft leadscrew* agar diketahui oleh mikrokontroler.
6. IC EEPROM berfungsi sebagai media penyimpanan parameter pengendalian motor stepper.
7. Mikrokontroler *data setter* berfungsi sebagai pemroses untuk mengatur parameter.
8. *Push button* berfungsi sebagai perangkat masukkan untuk memudahkan operator dalam menentukan parameter pengendalian.

9. Modul *Liquid Crystal Display* (LCD) berfungsi sebagai perangkat penampil.

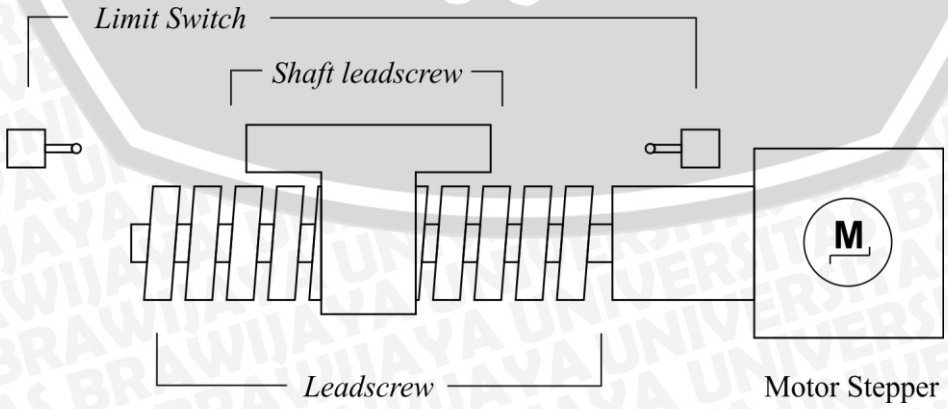
4.2 Perancangan Perangkat Keras

Perancangan dan pembuatan perangkat keras terdiri atas dua bagian yakni perancangan mekanik dan perancangan elektrik. Perancangan mekanik meliputi perancangan aktuator linier. Perancangan elektrik meliputi perancangan rangkaian konversi level tegangan mikrokontroler dengan PLC, perancangan rangkaian *driver* motor stepper, perancangan rangkaian elektrik mikrokontroler data setter, dan perancangan rangkaian elektrik mikrokontroler pengatur *driver*.

4.2.1 Perancangan Aktuator Linier

Aktuator linier yang digunakan dalam perancangan adalah motor stepper yang dihubungkan dengan *leadscrew* yang memiliki panjang efektif 6 inci dengan jumlah putaran yang diperlukan untuk menggerakan *shaft leadscrew* sejauh 1 inci adalah 5 putaran (5 TPI). Untuk menggerakan motor stepper sebanyak 1 putaran penuh diperlukan 200 pulsa *step* sehingga untuk menggerakan *shaft leadscrew* sejauh 1 inci diperlukan 1000 pulsa *step*. Panjang efektif dari *leadscrew* adalah 6 inci, maka dapat dikatakan terdapat maksimum 6000 pulsa *step* yang dapat diberikan. Dalam teorinya, setiap satu pulsa *step* akan menggerakan *shaft leadscrew* sejauh 0,001 inci (0,0254 mm) oleh karena itu ketelitian dari alat ini dibatasi hingga 100 pulsa *step* atau 0,1 inci (2,54 mm) agar dapat dianalisis menggunakan alat ukur.

Perancangan mekanik aktuator linier ditunjukkan pada Gambar 4.2. Pada aktuator digunakan *limit switch* sebagai umpan balik posisi *shaft leadscrew*. *Limit switch* digunakan untuk menentukan posisi minimum dan maksimum dari *shaft leadscrew* agar tidak terjadi kerusakan sistem.



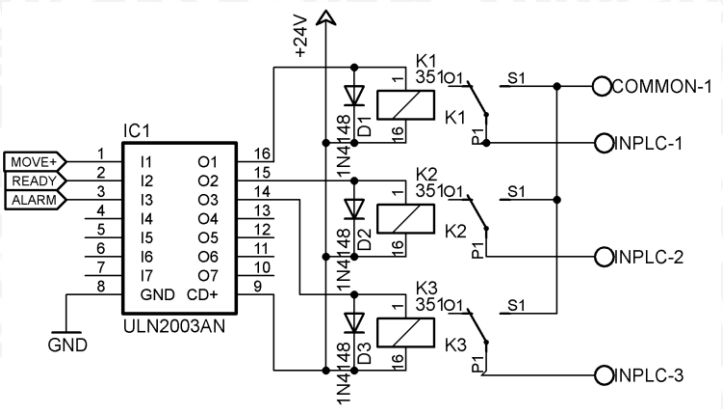
Gambar 4.2 Perancangan Aktuator Linier

Kecepatan dari motor stepper dipengaruhi oleh frekuensi pemberian pulsa *step* ke lilitan motor. Berdasarkan *datasheet*, motor bisa berputar pada *range* frekuensi 30 Hz – 1000 Hz. Akan tetapi, torsi terbesar diperoleh motor pada frekuensi 200 Hz. Oleh karena itu ditentukan frekuensi pemberian pulsa *step* pada frekuensi 166,67 Hz (1 periode pulsa *step* selama 6 ms), 200 Hz (1 periode pulsa *step* selama 5 ms), dan 250 Hz (1 periode pulsa *step* selama 4 ms). Dengan mengalikan periode pemberian 1 pulsa *step* tersebut dengan jumlah pulsa *step* yang diperlukan per inci, maka dapat diperoleh waktu yang diperlukan untuk *shaft leadscrew* bergerak sejauh 1 inci. Untuk frekuensi 166,67 Hz, didapatkan untuk menempuh jarak 1 inci diperlukan 6 detik (0,16667 $^{inci}/_{s}$), untuk frekuensi 200 Hz, 5 detik (0,2 $^{inci}/_{s}$) dan untuk frekuensi 250 Hz, 4 detik (0,25 $^{inci}/_{s}$).

4.2.2 Perancangan Rangkaian Konversi Level Tegangan Mikrokontroler dengan PLC

Rangkaian konversi level tegangan mikrokontroler dengan PLC berfungsi untuk menerjemahkan sinyal yang diberikan oleh PLC agar dapat direspon oleh mikrokontroler dan sebaliknya. Untuk dapat mengirimkan sinyal ke PLC, mikrokontroler memerlukan rangkaian transistor dan *relay* agar dapat menyuplai kebutuhan tegangan dan arus yang diperlukan oleh terminal masukkan PLC untuk keadaan logika tinggi. Terminal masukkan PLC untuk keadaan logika tinggi memutuhkan tegangan sebesar 24 V, oleh karena itu digunakan *relay* yang pada kontaknya digunakan tegangan sebesar 24 V. *Relay* yang digunakan membutuhkan arus minimum untuk mengaktifkan koil sebesar 15 mA. Arus keluaran untuk pin mikrokontroler maksimum sebesar 20 mA dan untuk setiap port tidak boleh melebihi 100 mA. Dikarenakan terdapat tiga sinyal keluaran yang dibutuhkan, maka dibutuhkan transistor untuk mengurangi kebutuhan arus agar tidak membebani mikrokontroler. Transistor yang digunakan adalah IC ULN2003 yang dapat menyuplai arus pada masing-masing pin hingga 500 mA. Untuk mengaktifkan transistor diperlukan arus sebesar 0,35 mA sesuai dengan *datasheet.*

Perancangan rangkaian konversi level tegangan keluaran mikrokontroler ke PLC ditunjukkan pada Gambar 4.3. Tegangan pada *common* IC transistor diberikan sesuai kebutuhan koil *relay* yaitu 24 V. Masukkan PLC diberikan pada kontak *normally open*. Ketika pin masukkan diberikan tegangan 5V dari mikrokontroler, maka transistor aktif yang mengakibatkan koil *relay* aktif dan arus mengalir dari *common* PLC ke pin input PLC.

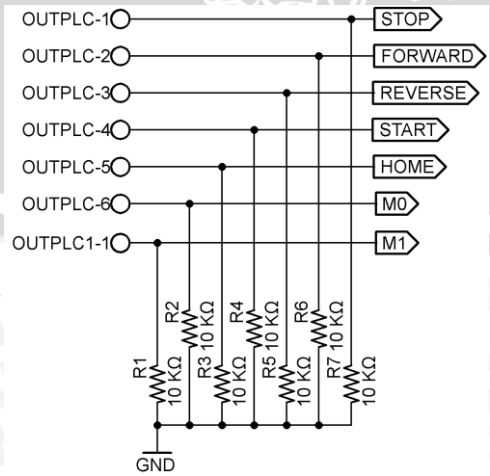


Gambar 4.3 Perancangan Rangkaian Konversi Level Tegangan Mikrokontroler ke PLC

Sinyal masukkan mikrokontroler dari PLC langsung dihubungkan ke pin mikrokontroler dikarenakan tegangan yang dikeluarkan sudah sesuai dengan kemampuan mikrokontroler. Resistor sebesar 10 kΩ digunakan untuk menjaga agar kondisi pin tetap berlogika rendah ketika tidak ada masukkan dari PLC. Dengan menggunakan resistor sebesar 10 kΩ, maka arus yang mengalir pada pin mikrokontroler ketika ada sinyal dari PLC ditunjukkan oleh persamaan (4 – 1) :

$$I = \frac{V_{outPLC}}{R_{pull-down}} = \frac{5\ V}{10\ \text{k}\Omega} = 0{,}5\ mA \quad (4 - 1)$$

Maka ketika ada sinyal dari PLC, arus yang mengalir ke mikrokontroler adalah sebesar 0,5 mA. Rangkaian konversi level tegangan PLC ke mikrokontroler ditunjukkan pada Gambar 4.4.



Gambar 4.4 Perancangan Rangkaian Konversi Level Tegangan PLC ke Mikrokontroler

4.2.3 Perancangan Rangkaian *Driver* Motor Stepper

Rangkaian *driver* motor stepper terdiri atas IC *driver* motor stepper L297 dan MOSFET IRF540N. Pin masukkan IC *driver* motor stepper yang diperlukan untuk menggerakan motor stepper yang berasal dari mikrokontroler adalah pin *Clock*, dan *Direction*. Pin *Control* diberikan logika rendah dikarenakan motor stepper yang digunakan adalah jenis unipolar. Mode pengoperasian motor stepper ditetapkan menggunakan mode *full step* sehingga pin *Half/Full* diberikan logika rendah. Pin *Enable* diberikan logika tinggi untuk mengaktifkan chip.

Motor stepper yang digunakan memiliki arus maksimum tiap fasa sebesar 1,5 A. Untuk mengendalikan *chopper* PWM agar dapat membatasi arus sebesar 1,5 A, digunakan resistansi Rsense sesuai dengan persamaan (2-1). I_{Load} yang mengalir sebesar 1,5 A, jika V_{ref} ditentukan sebesar 2 V, maka besar R_s yang diperlukan ditunjukkan oleh persamaan (4 – 2) :

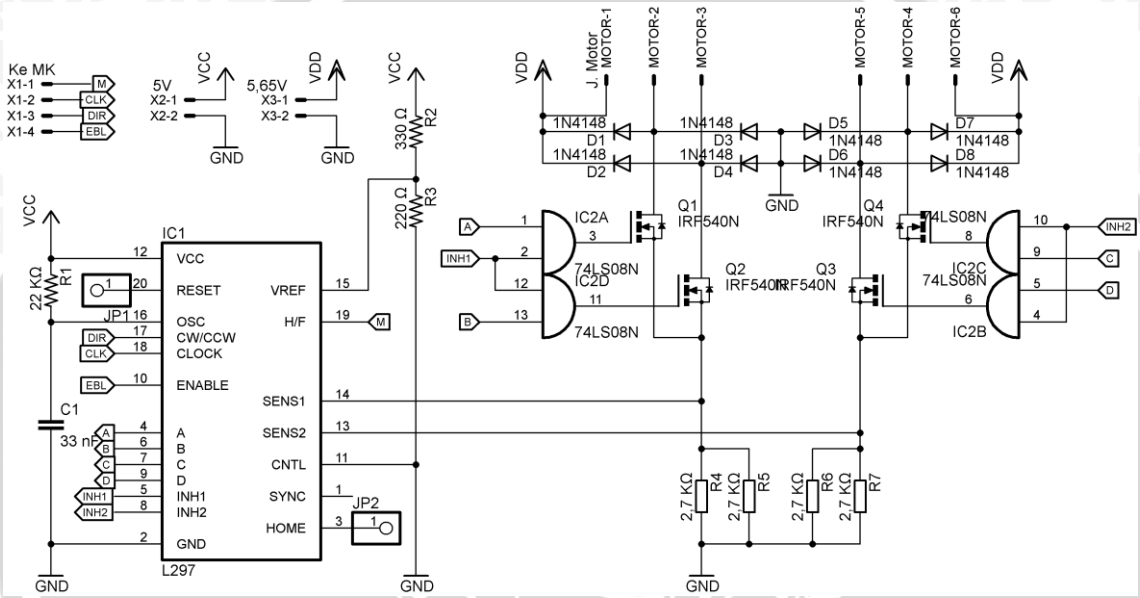
$$R_S = \frac{V_{ref}}{I_{Load}} = \frac{2\ V}{1{,}5\ A} = 1{,}333\ \Omega \quad (4 - 2)$$

Dari perhitungan didapatkan besar R_s yang diperlukan sebesar 1,333 Ω, oleh karena itu digunakan dua buah resistor 2,7 Ω yang disusun secara paralel agar didapatkan nilai yang mendekati yaitu 1,35 Ω. Dengan begitu, besar arus maksimum yang dapat dialirkan ke motor ditunjukkan oleh persamaan (4 – 3) :

$$I_{Load} = \frac{V_{ref}}{R_S} = \frac{2\ V}{1{,}35\ \Omega} = 1{,}4815\ A \quad (4 - 3)$$

Sinyal keluaran A dan B harus di AND kan dengan sinyal INH1 begitu juga sinyal keluaran C dan D di AND kan dengan sinyal INH2. Hal ini diperlukan agar didapatkan periode singkat untuk arus sirkulasi ulang ketika arus yang mengalir lebih dari 1,5 A. Oleh karena itu diperlukan gerbang AND 74LS08 yang sesuai dengan level logika keluaran IC *driver* motor stepper. Keluaran dari gerbang AND masuk ke MOSFET IRF540N yang memiliki V_{th} 3 V sehingga MOSFET akan bekerja dengan tegangan

keluaran gerbang AND yaitu 5 V. Pada *drain* MOSFET 1 yang dihubungkan fasa 1 motor stepper, *drain* MOSFET 2 ke fasa 2, dan seterusnya sampai fasa 4 motor stepper. Penghubungan ini harus dilakukan secara berurutan agar urutan pemberian energi pada fasa motor stepper yang dikendalikan oleh IC *driver* motor stepper sesuai dengan urutan fasa motor stepper. *Center tap* dari motor stepper masing-masing dihubungkan ke sumber tegangan 5,4 V sesuai dengan karakteristik motor. Keseluruhan rangkaian *driver* motor stepper ditunjukkan pada Gambar 4.5.



Gambar 4.5 Rangkaian Driver Motor Stepper

4.2.4 Perancangan Rangkaian Elektrik Mikrokontroler *Data Setter*

Rangkaian elektrik mikrokontroler *data setter* terdiri atas rangkaian sistem mikrokontroler, IC EEPROM, LCD, dan *push button.* Untuk berkomunikasi dengan pengguna, mikrokontroler dihubungkan dengan *push button* sebagai unit masukkan dan LCD sebagai unit keluaran data. Empat buah *push button* yang digunakan masuk ke pin mikrokontroler dengan menggunakan kapasitor yang disusun secara paralel untuk mengkompensasi *bouncing* pada *push button.* Untuk memfungsikan LCD, selain diperlukan sumber daya juga digunakan 7 pin yang akan dihubungkan ke mikrokontroler. Sumber daya masuk ke pin VCC dan VDD pada LCD, dan ke pin untuk menyalakan *backlight* yaitu Anoda dan Katoda. Pin VSS digunakan untuk mengontrol ketajaman tampilan LCD. Pin RS *(Register Select)*, R/W *(Read and Write)*, E *(Enable)*, dan D4 –

D7 dihubungkan ke mikrokontroler agar mikrokontroler dapat menampilkan data ke LCD.

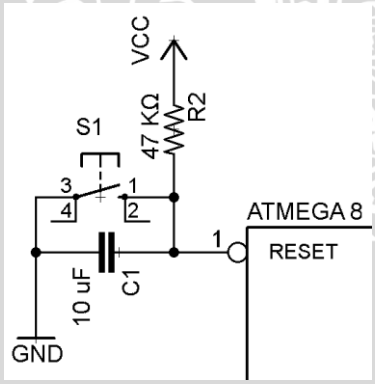
IC EEPROM digunakan mikrokontroler untuk menyimpan parameter yang telah diatur oleh pengguna. Mikrokontroler berkumunikasi dengan IC EEPROM dengan menggunakan jalur I2C oleh karena itu diperlukan jalur SDA *(Serial Data)* dan SCL *(Serial Clock)* dari mikrokontroler untuk dihubungkan ke pin SDA dan SCL pada IC EEPROM. Untuk dapat menggunakan jalur SDA dan SCL diperlukan resistor *pull-up* pada kedua pin tersebut. Dari datasheeet IC EEPROM, nilai resistor yang disarankan adalah sebesar 2 kΩ akan tetapi dikarenakan keterbatasan komponen yang ada maka nilai resistor yang digunakan dalam perancangan sebesar 2,2 kΩ. Pada rangkaian *data setter* terdapat konektor untuk berkomunikasi dengan mikrokontroler pada rangkaian driver. Mikrokontroler pada *data setter* berkomunikasi dengan IC EEPROM pada rangkaian mikrokontroler pengatur *driver* motor stepper menggunakan jalur I2C, sehingga pin SDA dan SCL juga perlu disambungkan menggunakan konektor.

Mikrokontroler Atmega 8 memiliki 3 port masukkan dan keluaran dua arah yang dapat diprogram yaitu port B, port C, dan port D. Secara keseluruhan, konfigurasi pin rangkaian mikrokontroler pada *data setter* adalah sebagai berikut :

1. Konfigurasi pin untuk antarmuka LCD :
    - Pin RS *(Register Select)* pada LCD yang berfungsi untuk memilih mode pengiriman data atau instruksi dihubungkan ke pin PB.0 pada mikrokontroler.
    - Pin RW *(Read/Write)* pada LCD yang berfungsi untuk memilih mode operasi baca atau tulis dihubungkan ke pin PD.7 pada mikrokontroler.
    - Pin E *(Enable)* pada LCD yang berfungsi untuk mengaktifkan akses LCD dihubungkan ke pin PB.1 pada mikrokontroler.
    - Pin D4 – D7 pada LCD yang berfungsi sebagai jalur data dihubungkan ke pin PB.2 – PB.5 pada mikrokontroler.
2. Konfigurasi pin untuk antarmuka IC EEPROM :
    - Pin SDA pada IC EEPROM yang berfungsi sebagai jalur data komunikasi I2C dihubungkan ke pin SDA mikrokontroler yaitu PC.4.
    - Pin SCL pada IC EEPROM yang berfungsi sebagai jalur *clock* komunikasi I2C dihubungkan ke pin SCL mikrokontroler yaitu PC.5.

3. Konfigurasi pin untuk *push button* :
   - Pin PD.2 – PD.5 pada mikrokontroler diatur sebagai masukkan untuk push button dengan nama berturut-turut PB. Kanan, PB. Kiri, PB. Bawah, dan PB. Atas.

Pada rangkaian sistem mikrokontroler terdapat rangkaian *oscillator* eksternal sebagai pembangkit *clock* mikrokontroler dan rangkaian reset eksternal sebagai pemicu reset mikrokontroler. Rangkaian *oscillator* terdiri dari dua buah kapasitor dan kristal, pada perancangan digunakan kristal sebesar 16 MHz, sedangkan nilai kapasitor disesuaikan dengan datasheet dimana nilai yang digunakan adalah 22 pF. Nilai komponen pada rangkaian reset mikrokontroler (Gambar 4.6) ditentukan berdasarkan perhitungan waktu minimum yang diperlukan agar mikrokontroler dapat reset, pada datasheet waktu minimum reset mikrokontroler yang dibutuhkan adalah 1,5 us. Komponen yang digunakan dalam perancangan adalah kapasitor yang bernilai 10 µF dan resistor yang bernilai 47 kΩ, kapasitor digunakan untuk mengkompensasi *bouncing* ketika reset terjadi sedangkan resistor digunakan agar pin RESET mikrokontroler tidak berada pada kondisi ambang. Untuk mendapatkan nilai waktu reset perlu diketahui terlebih dahulu nilai tegangan minimum kapasitor, nilai tegangan kapasitor didapatkan dari persamaan (4 - 4). Setelah tegangan minimal kapasitor diketahui, nilai waktu reset mikrokontroler dapat dihitung menggunakan persamaan (4 - 5).



Gambar 4.6 Rangkaian Reset Mikrokontroler

$$V_{C(\min)} = 0{,}2\ x\ V_{cc} \qquad (4-4)$$

$$= 0{,}2\ x\ 5 = 1\ V$$

$$V_{C(\min)} = V_s\ x\ e^{-\frac{1}{R}\ x\ C(t)} \qquad (4-5)$$

$$1 = 5\ x\ e^{-\frac{1}{47\ k\Omega}\ x\ 10\ \mu F}$$

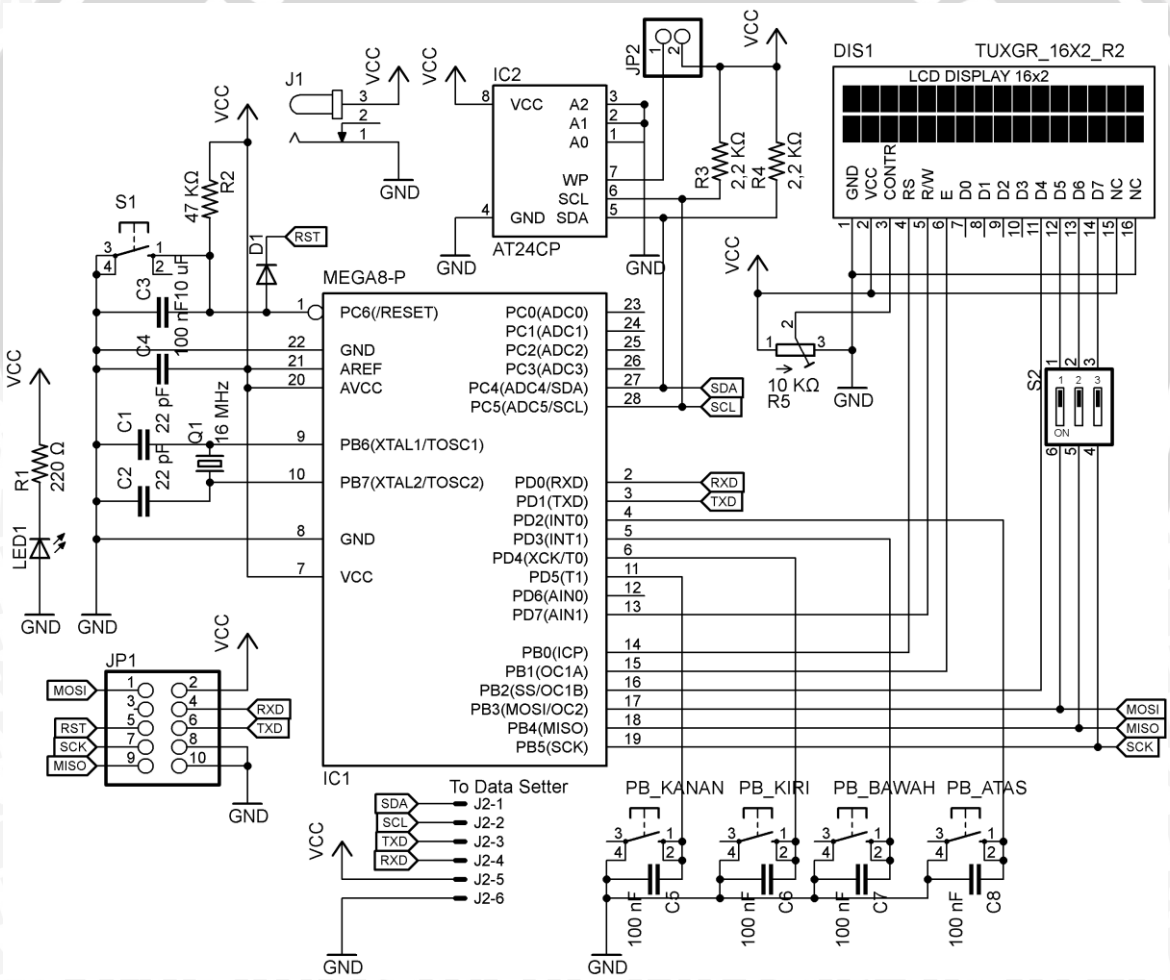
$$0{,}6 = e^{-\frac{t}{0{,}1}}$$

$$\ln 0{,}6 = \frac{-t}{0{,}1}$$

$$-0{,}51 = \frac{-t}{0{,}1}$$

$$t = 0{,}05\ s$$

Dari hasil perhitungan, nilai waktu reset (t) yang didapatkan sebesar 0,05 s sudah melebihi waktu minimum reset mikrokontroler sehingga pemilihan komponen kapasitor dan resistor yang digunakan sudah tepat. Mikrokontroler akan reset ketika tombol ditekan, dan kapasitor dan resistor akan menahan pin RESET berlogika rendah selama 0,05 s. Secara keseluruhan, rangkaian elektrik mikrokontroler *data setter* ditunjukkan pada Gambar 4.7.



Gambar 4.7 Perancangan Rangkaian Elektrik Mikrokontroler *Data Setter*

4.2.5 Perancangan Rangkaian Elektrik Mikrokontroler Pengatur *Driver* Motor Stepper

Rangkaian elektrik mikrokontroler pengatur *driver* motor stepper terdiri atas rangkaian sistem mikrokontroler, dan IC EEPROM, antarmuka *driver* motor stepper, dan antarmuka dengan umpan balik sensor. Pada rangkaian sistem mikrokontroler terdapat rangkaian *oscillator* eksternal sebagai pembangkit *clock* mikrokontroler dan rangkaian reset eksternal sebagai pemicu reset mikrokontroler. Rangkaian *oscillator* terdiri dari dua buah kapasitor dan kristal, pada perancangan digunakan kristal sebesar 16 MHz, sedangkan nilai kapasitor disesuaikan dengan datasheet dimana nilai yang digunakan adalah 22 pF. Waktu minimum yang diperlukan agar mikrokontroler dapat reset adalah 2,5 ns, dengan menggunakan perhitungan pada persamaan (4 – 4) dan (4 – 5), maka nilai kapasitor dan resistor yang digunakan masing-masing sebesar 10 uF dan 47 kΩ Mikrokontroler pengatur *driver* motor stepper menggunakan jalur I2C yang sama dengan IC EEPROM pada *data setter* oleh karena itu, perlu diatur penggunaan jalur ini agar ketika konektor dihubungkan tidak ada dua pasang resistor *pull-up* yang digunakan. Pengaturan ini menggunakan *DIP Switch* yang berfungsi memutus penggunaan *pull-up* jika *data setter* sedang dihubungkan.

Antarmuka mikrokontroler dengan PLC terdiri atas antarmuka masukkan mikrokontroler dari PLC dan keluaran mikrokontroler ke PLC. Sinyal yang masuk dari PLC dirancang untuk masuk ke pin interupsi eksternal dan PCINT sesuai dengan prioritas interupsi dari sinyal yang memiliki prioritas tertinggi sampai terendah. Prioritas interupsi dari yang paling tinggi ke rendah adalah sinyal *Stop, Forward, Reverse, Start,* dan *Home*. Pin interupsi eksternal yang tersedia hanya 3 pin sehingga untuk sinyal *Start* dan *Home* dihubungkan ke pin PCINT. Sinyal yang dikeluarkan mikrokontroler dan masuk ke PLC berupa sinyal status yang tidak memerlukan perlakuan khusus selain sebagai sinyal keluaran biasa. Sinyal tersebut adalah sinyal *Move, Ready,* dan *Alarm*.

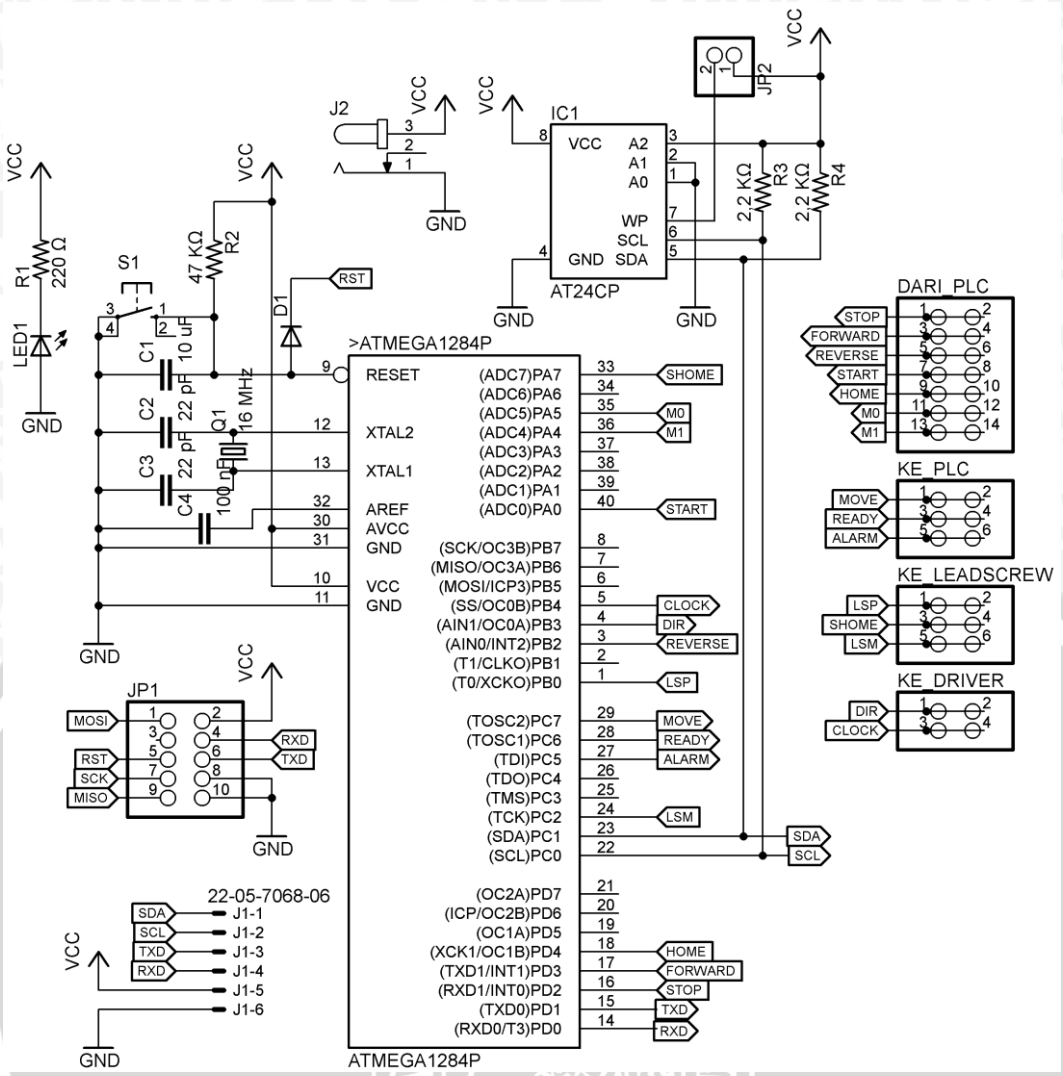
Antarmuka mikrokontroler dengan umpan balik sensor menggunakan pin mikrokontroler yang difungsikan sebagai masukkan. Sensor yang digunakan untuk mendeteksi posisi *shaft* pada *leadscrew* terdiri atas 3 sensor yaitu *LSP* (*Limit Switch* pada posisi *plus*), *LSM* (*Limit Switch* pada posisi *negatif*), dan *Shome* (*Push button* untuk titik referensi). Sinyal yang dikirimkan oleh sensor LSP dan LSM juga adalah interupsi sehingga dihubungkan ke pin PCINT.

Antarmuka mikrokontroler dengan *driver* motor stepper menggunakan pin mikrokontroler yang difungsikan sebagai keluaran. Sinyal yang diberikan oleh

mikrokontroler akan menentukan langkah apa yang perlu dilakukan oleh *driver* motor. Sinyal yang dikirimkan adalah sinyal *Clock*, dan *Direction.*

Mikrokontroler Atmega1284P memiliki 4 port masukkan dan keluaran dua arah yang dapat diprogram yaitu Port A, Port B, Port C, dan Port D. Konfigurasi pin rangkaian mikrokontroler pengatur *driver* motor stepper (Gambar 4.8) adalah sebagai berikut :

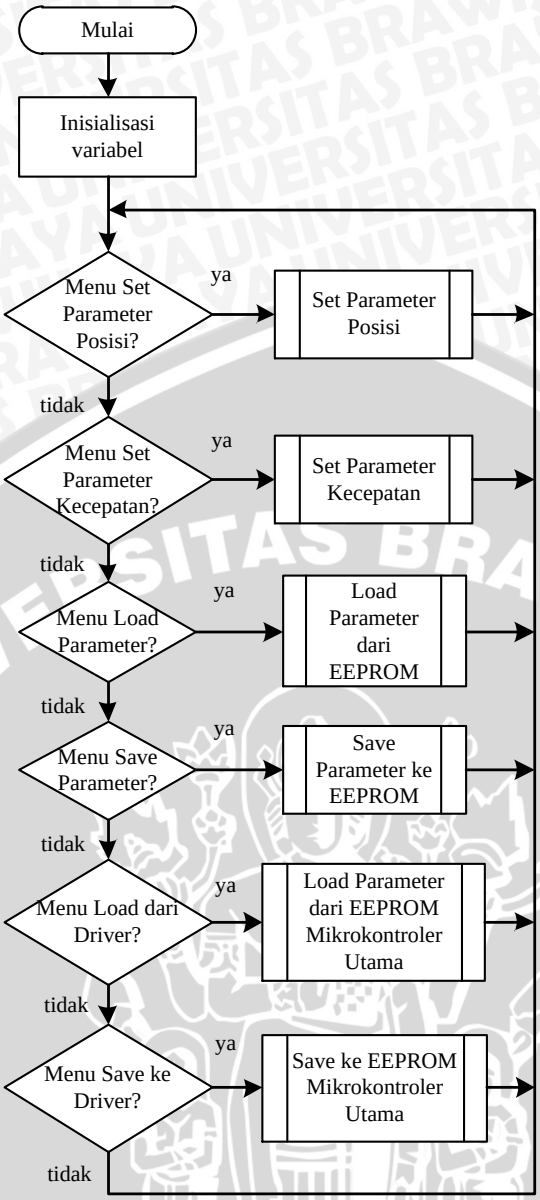
1. Konfigurasi pin untuk antarmuka IC EEPROM :
   - Pin SDA pada IC EEPROM yang berfungsi sebagai jalur data komunikasi I2C dihubungkan ke pin SDA mikrokontroler yaitu PC.1.
   - Pin SCL pada IC EEPROM yang berfungsi sebagai jalur *clock* komunikasi I2C dihubungkan ke pin SCL mikrokontroler yaitu PC.0.
2. Konfigurasi pin untuk antarmuka dengan PLC :
   - Sinyal *Stop* dihubungkan ke pin INT0 pada PD.2 yang diatur sebagai pin interupsi masukkan.
   - Sinyal *Forward* dihubungkan ke pin INT1 pada PD.3 yang diatur sebagai pin interupsi masukkan.
   - Sinyal *Reverse* dihubungkan ke pin INT2 pada PB.2 yang diatur sebagai pin interupsi masukkan.
   - Sinyal *Start* dihubungkan ke pin PCINT0 pada PA.0 yang diatur sebagai pin interupsi masukkan.
   - Sinyal *Home* dihubungkan ke pin PCINT24 pada PD.0 yang diatur sebagai pin interupsi masukkan.
   - Sinyal *Move* dihubungkan ke port PC.7 yang diatur sebagai pin keluaran.
   - Sinyal *Ready* dihubungkan ke port PC.6 yang diatur sebagai pin keluaran.
   - Sinyal *Alarm* dihubungkan ke port PC.5 yang diatur sebagai pin keluaran.
3. Konfigurasi pin untuk antarmuka dengan *driver* motor stepper.
   - Sinyal *Clock* dihubungkan ke port PB.4 yang diatur sebagai pin keluaran.
   - Sinyal *Direction* dihubungkan ke port PB.3 yang diatur sebagai pin keluaran.
4. Konfigurasi pin untuk antarmuka dengan umpan balik sensor.
   - Sinyal *LSP* dihubungkan ke pin PCINT8 pada PB.0 yang diatur sebagai pin masukkan.
   - Sinyal *LSM* dihubungkan ke pin PCINT18 pada PC.2 yang diatur sebagai pin masukkan.
   - Sinyal *SHome* dihubungkan ke pin PA.7 yang diatur sebagai pin masukkan.



Gambar 4.8 Perancangan Rangkaian Mikrokontroler Pengatur *Driver* Motor Stepper

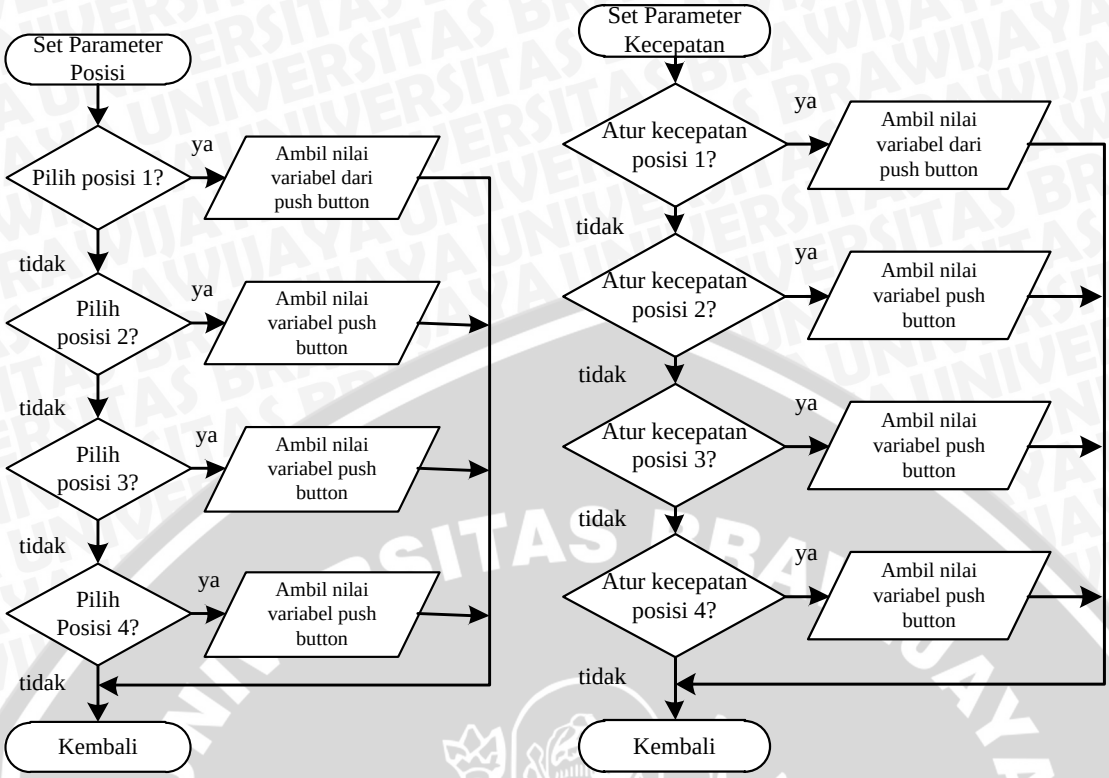
4.3 Perancangan Perangkat Lunak Mikrokontroler *Data Setter*

Data setter berfungsi sebagai peralatan pengatur dan penyimpan parameter. Parameter yang diatur oleh *data setter* adalah posisi dan kecepatan antar posisi dari gerakan *leadscrew*. Posisi yang dapat diatur ditentukan sebanyak 4 posisi. Kecepatan dari leadscrew ditentukan sejumlah 3 mode kecepatan yang berbeda. Parameter yang telah diatur disimpan ke lokasi memori yang berbeda sehingga parameter yang dapat disimpan ditentukan sebanyak 3 parameter. Posisi menunjuk kepada jumlah pulsa *step* yang diberikan, sedangkan kecepatan menujuk kepada periode pemberian pulsa *step* yang dinyatakan dalam *milisecond*. Diagram alir perangkat lunak fungsi utama *data setter* ditunjukkan dalam Gambar 4.9.



Gambar 4.9 Diagram Alir Perangkat Lunak Fungsi Utama *Data Setter*

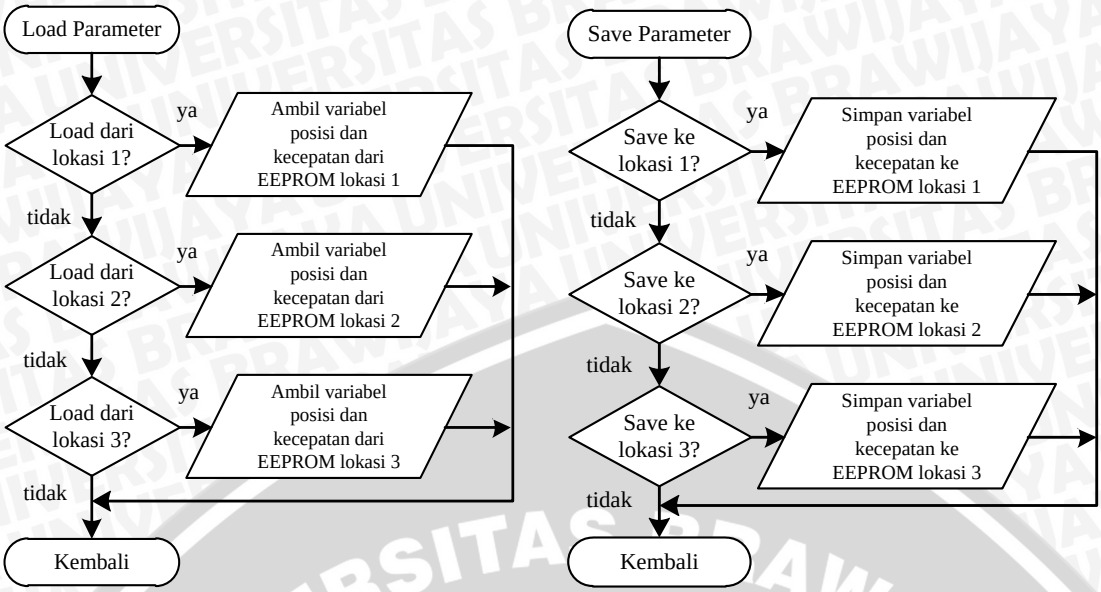
Terdapat 2 menu untuk mengubah parameter, yaitu menu "Set Parameter Posisi", dan "Set Parameter Kecepatan", menu "Set Parameter Posisi" digunakan untuk mengubah 4 nilai posisi menjadi nilai antara 0 – 6000, menu "Set Parameter Kecepatan" digunakan untuk mengubah kecepatan antar posisi menjadi 3 nilai kecepatan yaitu pemberian 1 periode pulsa *step* selama 4 ms (250 Hz), 5 ms (200 Hz), dan 6 ms (166,67 Hz). Nilai dari variabel tersebut diatur dengan menggunakan push button PB Kanan dan PB Kiri. Diagram alir subfungsi "Set Parameter Kecepatan", dan "Set Parameter Posisi" ditunjukkan dalam Gambar 4.10.



(a) (b)

Gambar 4.10 Diagram Alir Subfungsi (a)"Set Parameter Posisi" dan (b)"Set Parameter Kecepatan"

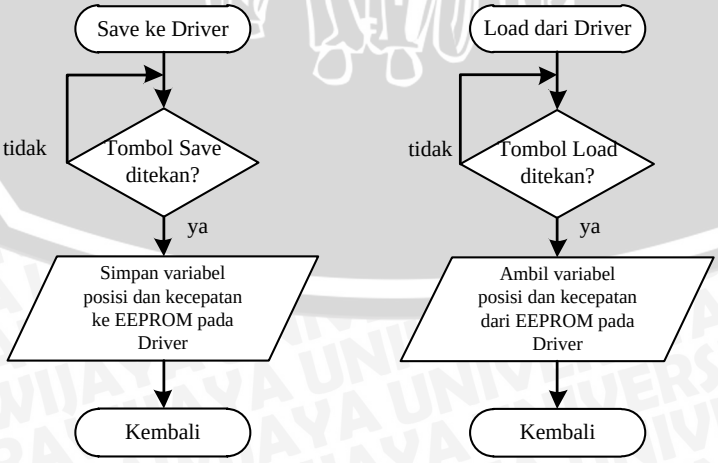
Menu "*Load* Parameter" dan "*Save* Parameter" digunakan untuk menyimpan nilai parameter posisi dan kecepatan ke dalam EEPROM. Parameter tersebut dapat disimpan di 3 alamat yang berbeda pada EEPROM. EEPROM yang digunakan adalah EEPROM pada rangkaian *data setter* yang diatur memiliki alamat kontrol 0xA0 untuk proses tulis dan 0xA1 untuk proses baca. Nilai parameter disimpan pada lokasi ke 1 dimulai dari alamat 0x00, lokasi ke 2 dimulai dari alamat 0x14, dan lokasi ke 3 dimulai dari alamat 0x28. Masing-masing lokasi kemudian menyimpan variabel posisi dan kecepatan pada alamat yang berbeda-beda. Sehingga lokasi ke 1, 2, dan 3 tidak saling berhubungan dan dapat dengan bebas diubah-ubah tanpa mengubah variabel pada lokasi lainnya. Hal lain yang perlu diperhatikan adalah program pada mikrokontroler *data setter* dan pengatur *driver* harus menunjuk alamat yang sama agar nilai posisi dan kecepatan yang disimpan akan sama dengan nilai yang sudah dimasukkan sebelumnya. Diagram alir subfungsi "*Load* Parameter", dan "*Save* Parameter" ditunjukkan dalam Gambar 4.11.



(a) (b)

Gambar 4.11 Diagram Alir Subfungsi (a)"Load Parameter", dan (b)"Save Parameter"

Parameter yang telah diatur kemudian ditransfer ke alamat 0x00 EEPROM pada mikrokontroler pengatur *driver* motor stepper dengan menggunakan jalur I2C menggunakan menu "*Save* ke *Driver*". EEPROM tersebut memiliki alamat kontrol 0xA8 untuk proses tulis dan alamat kontrol 0xA9 untuk proses baca. Selain menyimpan parameter, data setter juga bisa mengambil parameter yang telah ada pada mikrokontroler pengatur *driver* motor stepper menggunakan menu "*Load* dari *Driver*" untuk dapat dimodifikasi. Diagram alir subfungsi "*Save* ke *Driver*", dan "*Load* dari *Driver*" ditunjukkan dalam Gambar 4.12.



(a) (b)

Gambar 4.12 Diagram Alir Subfungsi (a)"*Save* ke *Driver*", dan (b)"*Load* dari *Driver*"

4.4 Perancangan Perangkat Lunak Mikrokontroler Pengatur *Driver* Motor Stepper

Fungsi dari mikrokontroler pengatur *driver* motor stepper adalah menerima sinyal perintah *Start, Stop, Forward, Reverse*, dan *Home* dari PLC, menerima sinyal dari *limit switch*, menerima parameter dari *data setter*, dan memberikan perintah *Clock* dan *Direction* kepada driver motor stepper. Mikrokontroler bertugas untuk menggerakan *leadscrew* ke posisi tertentu dengan kecepatan tertentu sesuai dengan perintah dari PLC. Pemilihan posisi dan kecepatan ditentukan berdasarkan sinyal masukan M0 dan M1 dengan nilai yang sudah ditetapkan sebelumnya oleh *data setter* (Tabel 4.1). Perintah dari PLC disusun berdasarkan prioritas interupsi pada mikrokontroler. Urutan prioritas interupsi dari yang tertinggi sampai yang terendah yaitu perintah *Stop, Forward, Reverse, Start,* dan *Home*. Prioritas tertinggi yaitu *Stop* dapat menghentikan gerakan *leadscrew* secara langsung dan mencegah perintah lain. Mikrokontroler juga perlu mengeluarkan sinyal penanda agar PLC dapat mengetahui status dari gerakan *leadscrew*. Sinyal penanda ini juga digunakan PLC untuk berkomunikasi dengan perangkat lainnya. Tabel 4.2 menunjukkan sinyal masukkan dan keluaran mikrokontroler dari dan ke PLC.

Tabel 4.1 Parameter di dalam Register Posisi

Pengaturan Register Posisi		Parameter Posisi	Parameter Kecepatan
M1	M0		
0	0	Posisi 1	Kecepatan 1
0	1	Posisi 2	Kecepatan 2
1	0	Posisi 3	Kecepatan 3
1	1	Posisi 4	Kecepatan 4

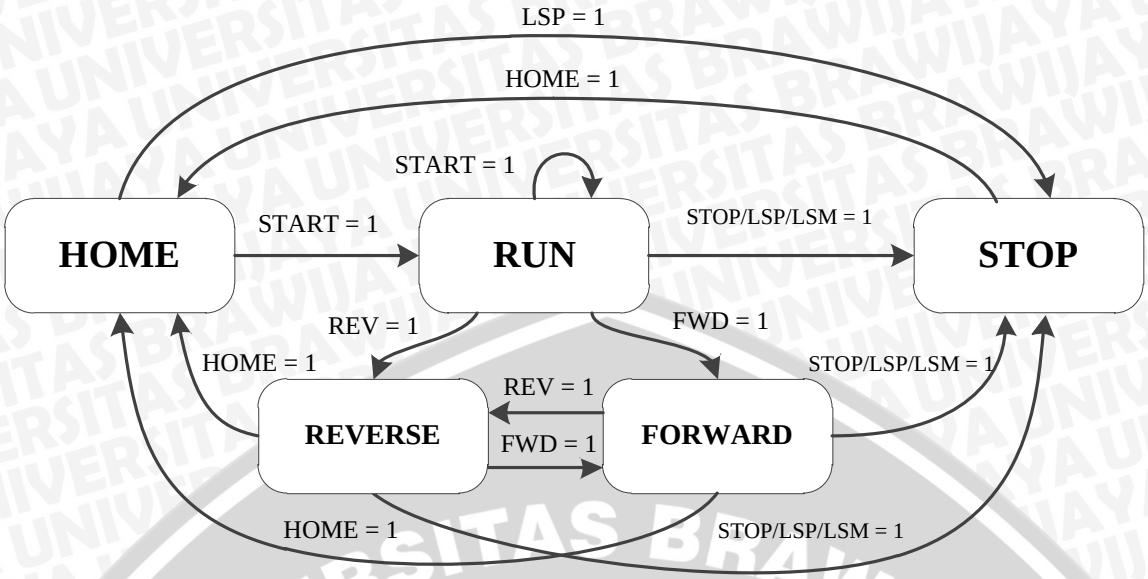
Tabel 4.2 Sinyal Masukkan dan Keluaran Mikrokontroler dari dan ke PLC

Sinyal Masukkan Mikrokontroler dari PLC	Fungsi
Stop	Sinyal untuk menghentikan gerakan.
Forward	Sinyal untuk menggerakan motor searah jarum jam.
Reverse	Sinyal untuk menggerakan motor berlawanan arah jarum jam.
Start	Sinyal untuk memulai gerakan.
Home	Sinyal untuk menggerakan leadscrew ke posisi *Home*.
M0 M1	Register posisi.

Sinyal Keluaran Mikrokontroler ke PLC	Fungsi
Move	Sinyal penanda motor sedang bergerak
Ready	Sinyal penanda mikrokontroler siap menerima perintah gerakan berikutnya
Alarm	Sinyal penanda *leadscrew* dalam kondisi berbahaya

Nilai dari parameter posisi diatur ada dalam rentang 0 – 6000 sesuai dengan jumlah pemberian pulsa *(step)* yang perlu diberikan ke *driver* motor stepper yang untuk menggerakan *shaft leadscrew* sejauh 6 inci. Nilai dari masing-masing parameter posisi diatur berpatokan dari titik referensi Dari *datasheet* motor, diketahui torsi terbesar dari motor didapatkan jika motor diberikan kecepatan pemberian pulsa sebesar 250 Hz, oleh karena itu nilai dari parameter kecepatan dibatasi 3 nilai kecepatan yang berbeda yaitu dengan memberikan 1 periode pulsa step selama 4 ms (250 Hz), 5 ms (200 Hz), dan 6 ms (166,67 Hz) ke pin *Clock* IC *driver*. Parameter posisi dan kecepatan disimpan pada IC EEPROM.

Perangkat lunak mikrokontroler pengatur *driver* motor stepper dibuat berdasarkan *state diagram*. Ketika sistem pertama kali diaktifkan, *state Home* akan dimulai sampai sinyal *Ready* aktif dan akan dilanjutkan ke *state Run*. Jika *shaft leadscrew* tidak mendeteksi titik referensi, maka sinyal *Alarm* aktif dan akan dilanjutkan ke *state Stop*. *State Run* akan dimulai ketika ada perintah *Start* dari PLC. *State Run* mengambil sinyal posisi M0 dan M1 dari PLC lalu mengambil parameter posisi dan kecepatan yang sesuai dari IC EEPROM. Sistem akan tetap berada pada *state Run* sampai perintah selesai dilaksanakan atau akan pindah ke *state Stop* ketika ada perintah *Stop* dari PLC atau ada sinyal dari *limit switch LSP* dan *LSM*. Program juga akan berpindah dari *state Run* ke *state Forward* atau *Reverse* atau *Home* jika ada perintah dari PLC. Program dapat berpindah dari *state Forward* atau *Reverse* ke *state Stop* ketika ada perintah *Stop* dari PLC atau ada sinyal dari *limit switch LSP* dan *LSM*. Akan tetapi, untuk kembali ke *state Run* ketika sebelumnya program berada pada *state Forward, Reverse,* atau *Stop,* program dirancang agar perlu kembali ke *state Home* menggunakan perintah *Home* dari PLC. Hal ini diperlukan agar untuk kembali ke posisi M0 dan M1 sistem perlu kembali ke titik referensi yaitu posissi nol *shaft leadscrew.* *State Diagram* dari perangkat lunak mikrokontroler pengatur *driver* motor stepper ditunjukkan pada Gambar 4.13.



Gambar 4.13 *State Diagram* Program Utama Mikrokontroler Pengatur *Driver* Motor Stepper

BAB V
PENGUJIAN DAN ANALISIS

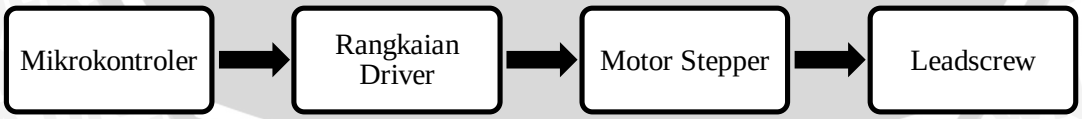
Pengujian alat dilakukan untuk memastikan apakah kinerja alat telah sesuai dengan perancangan. Pengujian dilakukan secara bertahap mulai dari pengujian masing-masing subsistem hingga pengujian sistem secara keseluruhan. Pengujian yang dilakukan terdiri atas :

1. Pengujian aktuator linier.
2. Pengujian rangkaian konversi level tegangan mikrokontroler dengan PLC.
3. Pengujian rangkaian *driver* motor stepper.
4. Pengujian sistem mikrokontroler *data setter.*
5. Pengujian sistem secara keseluruhan

5.1 Pengujian Aktuator Linier

Pengujian aktuator linier bertujuan untuk memastikan sinyal *Clock* yang diberikan oleh mikrokontroler kepada rangkaian driver motor stepper sesuai dengan perpindahan *shaft leadscrew* dari posisi tertentu. Diagram blok pengujian ditunjukkan pada Gambar 5.1. Pada pengujian digunakan kecepatan yang berbeda yang sesuai dengan frekuensi sinyal *Clock* yang diberikan mikrokontroler kepada IC driver motor stepper. Pengujian dilakukan dengan prosedur sebagai berikut :

1. Melakukan pemrograman pada mikrokontroler untuk menghasilkan sinyal *Clock* sebanyak 6000 kali dengan frekuensi 50 Hz.
2. Mengatur posisi *shaft leadscrew* ke titik referensi.
3. Mengambil data perpindahan *shaft leadscrew* setiap 1 inci.
4. Mengulangi prosedur 1, 2 dan 3 dengan frekuensi sinyal *Clock* diubah hingga motor tidak dapat berputar.

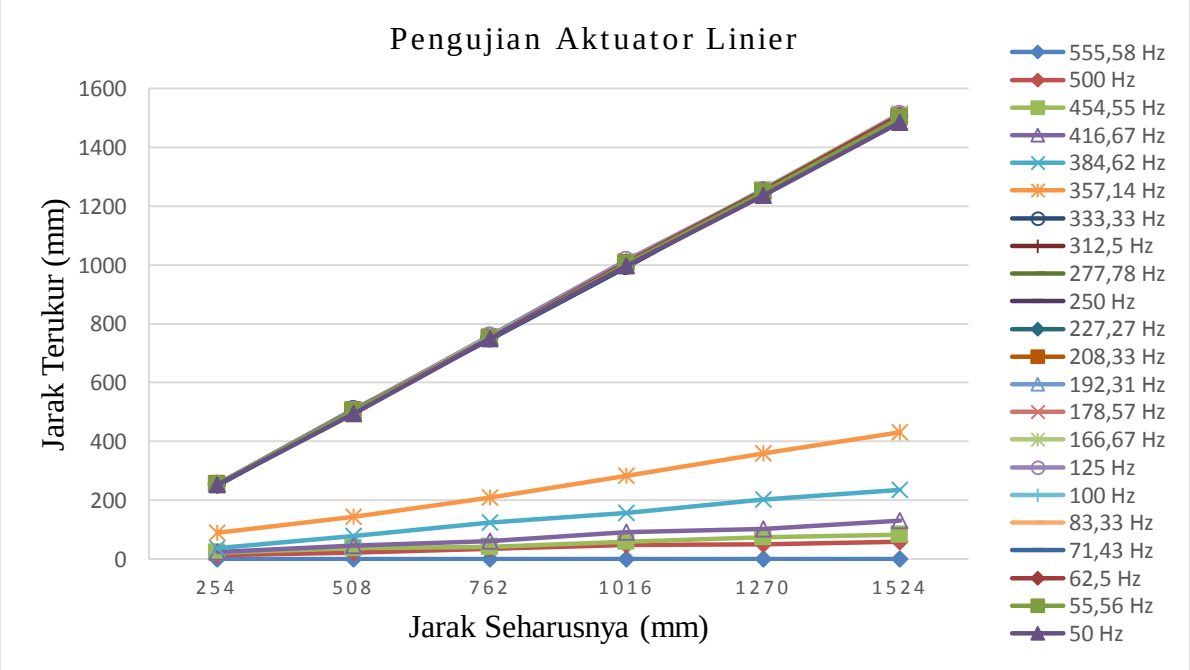


Gambar 5.1 Diagram Blok Pengujian Aktuator Linier

Hasil pengujian yang diharapkan adalah *shaft leadscrew* dapat bergerak sejauh 5 inci dengan ketelitian sebesar 0,1 inci (2,54 mm). Hasil pengujian diharapkan tidak jauh berbeda ketika frekuensi sinyal *Clock* diubah. Data hasil pengujian ditunjukkan pada Tabel 5.1 dan Gambar 5.2.

Tabel 5.1 Data Hasil Pengujian Aktuator Linier

Pulsa *Step*		Jarak Terukur (mm)					
Periode (ms)	Frekuensi (Hz)	Jarak 25,4 mm	Jarak 50,8 mm	Jarak 76,2 mm	Jarak 101,6 mm	Jarak 127,0 mm	Jarak 152,4 mm
20	50	25,3	49,4	74,8	99,8	123,8	148,5
18	55,56	25,3	50,2	75,0	100,3	124,7	150,0
16	62,5	25,2	50,3	75,2	100,8	125,4	150,9
14	71,43	25,1	50,2	75,1	100,6	125,3	150,8
12	83,33	25,2	50,4	75,3	100,2	125,5	150,3
10	100	25,4	50,4	75,8	100,1	125,8	150,2
8	125	25,4	50,8	76,1	101,6	125,5	151,3
6	166,67	25,4	50,8	75,7	101,1	125,8	151,5
5.6	178,57	25,4	50,5	75,5	101,2	125,2	151,1
5.2	192,31	25,4	50,4	75,6	101,2	125,4	151,2
4.8	208,33	25,4	50,5	75,3	100,6	125,2	150,4
4.4	227,27	25,1	50,5	75,4	100,1	125,6	150,2
4	250	25,4	50,1	75,2	100,1	125,4	150,2
3.6	277,78	25,1	50,3	75,9	100,1	125,2	150,3
3.2	312,5	25,3	50,1	75,4	100,1	125,1	150,9
3	333,33	25,5	50,9	74,7	99,3	124,8	149,2
2.8	357,14	8,9	14,3	20,8	28,4	35,9	43,1
2.6	384,62	3,6	7,9	12,3	15,6	20,2	23,4
2.4	416,67	2,3	4,6	6,1	9,1	10,3	13,1
2.2	454,55	2,2	3,7	4,2	5,9	7,3	8,3
2	500	1,3	2,1	3,4	4,7	4,9	5,8
1.8	555,56	0	0	0	0	0	0



Gambar 5.2 Grafik Hasil Pengujian Aktuator Linier

Dari keseluruhan *range* frekuensi, dapat terlihat semakin jauh perpindahan *shaft leadscrew* dari titik referensi, semakin besar *error* yang terjadi. Dari pengujian didapatkan hasil jarak terukur sesuai dengan jarak seharusnya dengan *error* maksimum 3,9 mm pada frekuensi pemberian *Clock* di atas 333,33 Hz. Pada frekuensi di atas 333,33 Hz, motor mulai tidak dapat berputar hingga pada frekuensi 500 Hz motor tidak dapat berputar lagi dikarenakan motor tidak dapat menghasilkan torsi awal yang cukup untuk menggerakan *shaft leadscrew.* Rata-rata *error* terkecil berada pada *range* frekuensi 125 - 250 Hz, sesuai dengan besarnya torsi maksimum yang dapat diberikan motor yang tertera pada *datasheet*. *Error* maksimum dari range frekuensi 125 – 250 Hz adalah 2,2 mm pada perpindahan *shaft leadscrew* sejauh 6 inci (152,4 mm). Oleh karena itu dapat disimpulkan, ketelitian dari alat ini adalah 2,2 mm pada frekuensi pemberian *Clock* antara 125 – 250 Hz. Berdasarkan ketelitian tersebut, untuk menggerakan *shaft leadscrew* sejauh 2,2 mm diperlukan sekitar 87 pulsa, maka resolusi dari alat ini adalah 87 pulsa.

5.2 Pengujian Rangkaian Konversi Level Tegangan Mikrokontroler dengan PLC

Pengujian rangkaian konversi level tegangan mikrokontroler dengan PLC bertujuan untuk mengetahui apakah instruksi dari PLC dapat masuk dan diolah oleh mikrokontroler dan sebaliknya apakah instruksi yang diberikan mikrokontroler dapat masuk dan diolah oleh PLC. Diagram blok pengujian ditunjukkan pada Gambar 5.3. Pengujian dilakukan dengan cara melakukan pemrograman pada PLC dan mikrokontroler dan kemudian dilihat respon keluaran PLC dan mikrokontroler menggunakan osiloskop. Pengujian dilakukan dengan menggunakan rangkaian pada Gambar 4.3 dan Gambar 4.4.

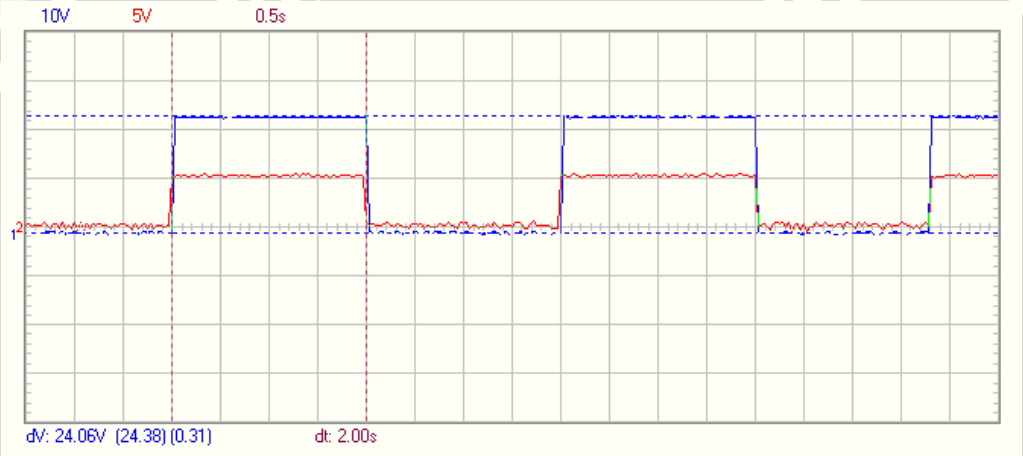
Gambar 5.3 Diagram Blok Pengujian Rangkaian Konversi Level Tegangan

Pengujian dilakukan dengan prosedur sebagai berikut :

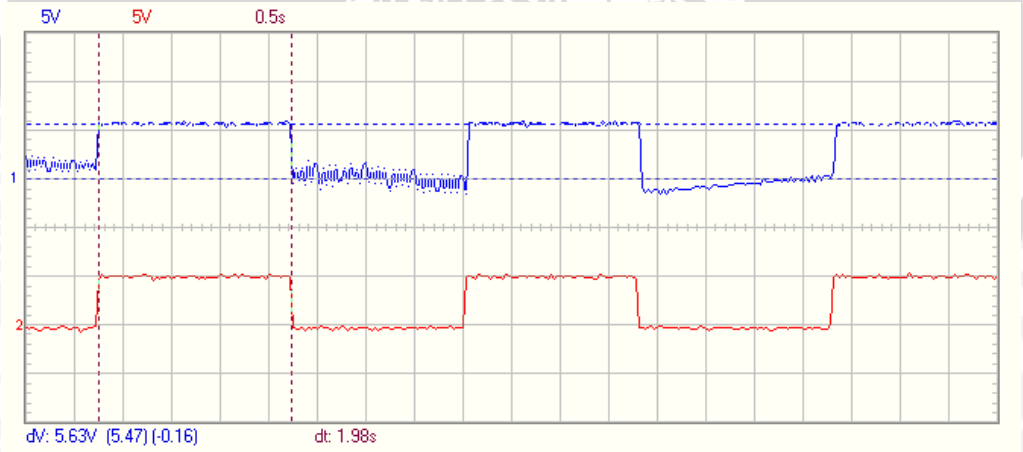
1. Melakukan pemrograman pada mikrokontroler agar dapat mengirimkan sinyal periodik selama 2 detik pada PORTA.0, kemudian melakukan pemrograman pada PLC untuk menerima sinyal dari mikrokontroler pada pin 0.00 kemudian mengeluarkan sinyal pada port 1.00 yang sesuai dengan sinyal pada pin 0.00.
2. Menghubungkan sistem PLC dan mikrokontroler sesuai dengan rangkaian pada Gambar 4.3 dan Gambar 4.4.

3. Mentransfer program ke PLC kemudian menjalankan menu *Work Online* pada perangkat lunak untuk melihat kerja PLC
4. Melihat respon keluaran PLC dan mikrokontroler menggunakan osiloskop.
5. Melakukan pemrograman pada PLC agar PLC dapat mengirimkan sinyal periodik selama 2 detik pada *port* 1.00, kemudian melakukan pemrograman pada mikrokontroler agar dapat menerima sinyal dari PINB.0 kemudian meneruskan sinyal yang diterima untuk langsung dikeluarkan di PORTA.0.
6. Mengulangi langkah 2, 3, dan 4.

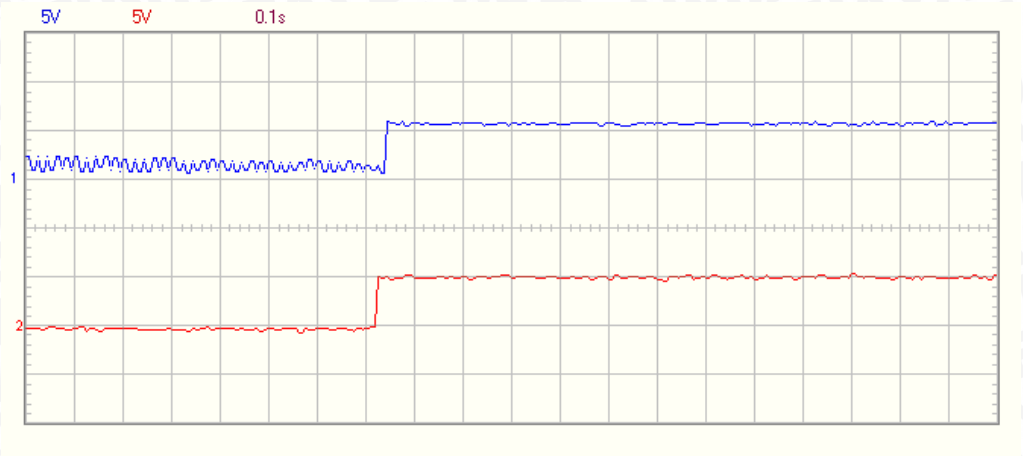
Hasil pengujian yang diharapkan adalah mikrokontroler dan PLC dapat menerima dan meneruskan sinyal dengan respon yang sesuai dengan level logika terminal masukkan dan keluaran PLC. Data hasil pengujian ditunjukkan pada Gambar 5.4.



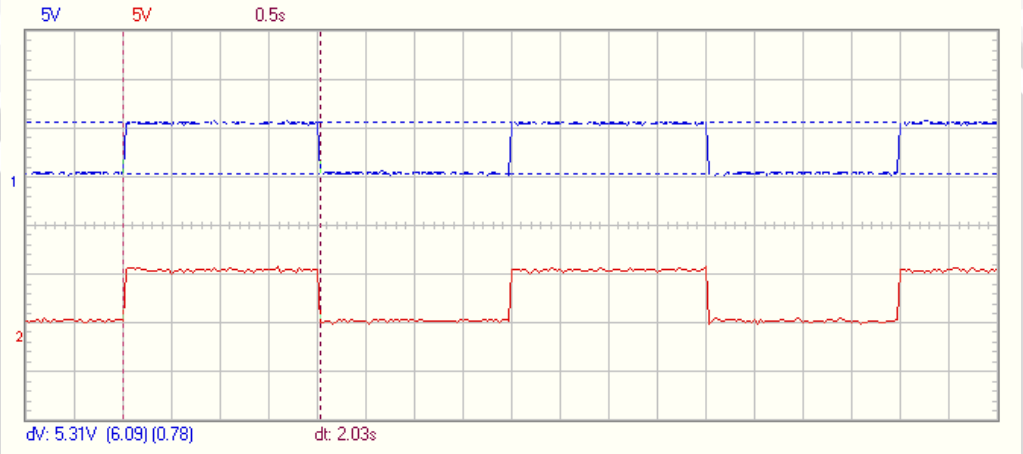
(a)



(b)



(c)



(d)

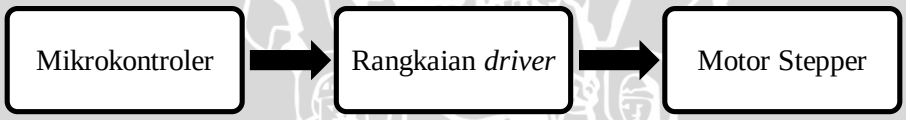
Gambar 5.4 Data Hasil Pengujian Rangkaian Konversi Level Tegangan (a) Sinyal Keluaran Mikrokontroler dan Sinyal Keluaran *Relay*, (b) Sinyal Keluaran PLC (atas) dan Sinyal Keluaran Mikrokontroler (bawah), (c) Perbandingan Sinyal Keluaran Mikrokontroler (bawah) dan Sinyal Keluaran PLC (atas), dan (d) Sinyal Masukan Mikrokontroler dari PLC (atas) dan Sinyal Keluaran Mikrokontroler (bawah)

Dari pengujian didapatkan hasil rangkaian *relay* (Gambar 4.3) telah berfungsi dengan baik dengan mengeluarkan tegangan 24 V ketika mikrokontroler mengeluarkan tegangan 5 V dan sebaliknya dengan respon yang sesuai (Gambar 5.5 (a)). Sinyal yang diterima pada pin 0.00 langsung dikeluarkan pada port 1.00 (Gambar 5.5 (b)). Pada port 1.00 terlihat adanya *noise* tegangan yang apabila dihubungkan ke sistem mikrokontroler tidak akan memengaruhi level logika mikrokontroler. Pada Gambar 5.5 (c) terlihat adanya jeda yang diperlukan sinyal mikrokontroler (bawah) untuk dapat diproses oleh PLC (sinyal keluaran PLC pada bagian atas. Jeda ini diakibatkan oleh rangkaian konversi level tegangan dan kecepatan pemroses sinyal pada sistem PLC. Pada pengujian berikutnya ketika PLC mengirimkan sinyal kepada mikrokontroler untuk diproses dan dikeluarkan

langsung pada PORTA.0 terlihat bahwa mikrokontroler dapat memroses sinyal masukkan. Noise tegangan yang terlihat pada Gambar 5.5 (b) hilang dikarenakan sinyal keluaran PLC dijaga agar tetap berlogika rendah (mengacu pada rangkaian masukkan pada Gambar 4.4). Dari pengujian dapat disimpulkan rangkaian konversi level tegangan mikrokontroler dengan PLC telah dapat mengirimkan sinyal dan menerima sinyal dengan jeda waktu singkat dan level logika tegangan yang terjaga.

5.3 Pengujian Rangkaian *Driver* Motor Stepper

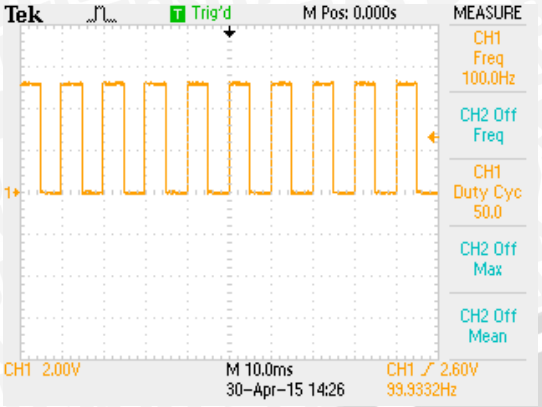
Pengujian rangkaian elektrik *driver* motor stepper bertujuan untuk mengetahui apakah perintah yang diberikan oleh mikrokontroler sesuai dengan gerakan motor stepper. Perintah yang diberikan oleh mikrokontroler berupa sinyal *Clock* dan *Direction* kepada IC *driver* motor stepper. Jika perintah sesuai, maka motor akan bergerak dengan kecepatan tertentu tergantung dari frekuensi sinyal *Clock* yang diberikan. Sinyal *Direction* menentukan arah putaran motor. Pengujian dilakukan dengan cara melihat bentuk gelombang pada pin-pin masukkan dan keluaran IC *driver* motor stepper. Bentuk gelombang kemudian dianalisis apakah sesuai dengan *datasheet*. Pengujian rangkaian *driver* motor stepper sesuai dengan rancangan pada Gambar 4.5. Diagram blok pengujian ditunjukkan pada Gambar 5.5.



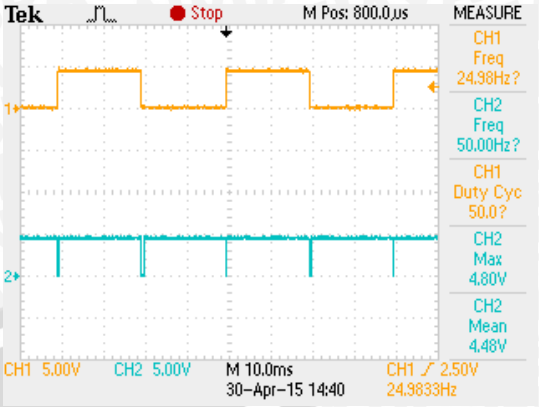
Gambar 5.5 Diagram Blok Pengujian Rangkaian *Driver* Motor Stepper

Hasil pengujian ditunjukkan pada Gambar 5.6. Pengujian dilakukan dengan prosedur sebagai berikut :

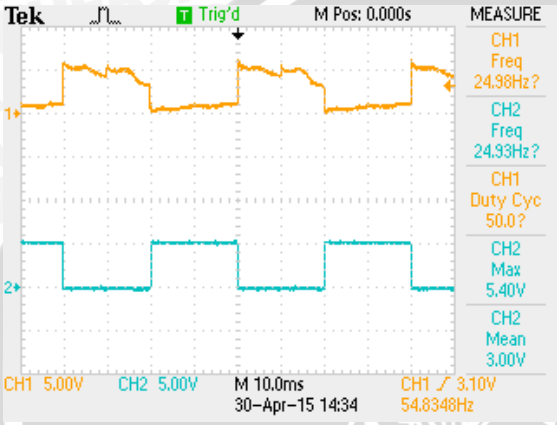
1. Melakukan pemrograman mikrokontroler dengan memberikan pulsa dengan frekuensi 100 Hz ke pin *Clock* pada rangkaian *driver*.
2. Melihat sinyal keluaran A dan INH1 pada IC *driver* motor stepper.
3. Melihat sinyal masukkan *gate* MOSFET 1 dan sinyal pada *drain* MOSFET1.
4. Melihat sinyal pada *drain* MOSFET 1 dan *drain* MOSFET 2, dan juga *drain* MOSFET 1 dan *drain* MOSFET 3.



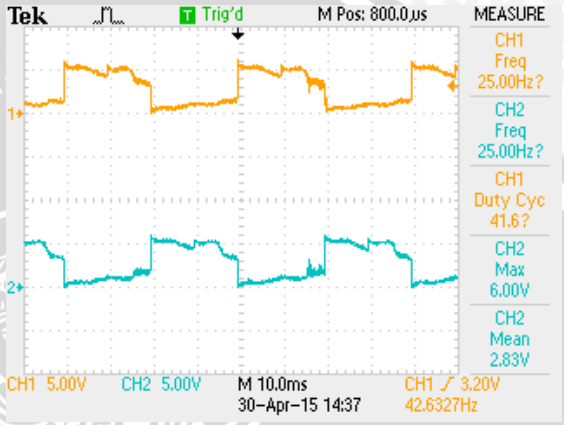
(a)



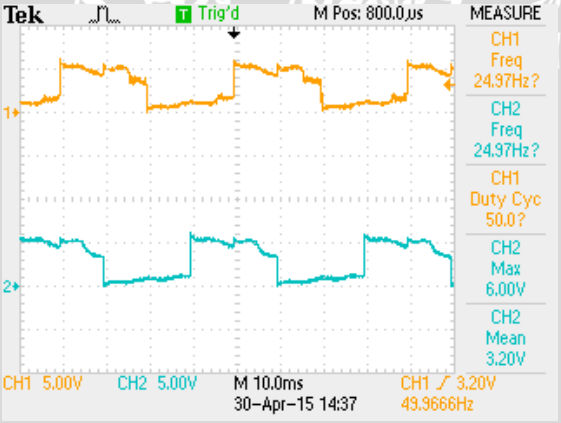
(b)



(c)



(d)



(e)

Gambar 5.6 Hasil Pengujian Rangkaian *Driver* Motor Stepper, (a) Sinyal Keluaran Clock, (b) Sinyal Keluaran A dan INH1, (c) Sinyal *Gate* MOSFET 1 dan Sinyal *Drain* Mosfet 1, (d) Tegangan *Drain* MOSFET 1 dan MOSFET 2, (e) Tegangan Drain MOSFET 1 dan MOSFET 3

Pada Gambar 5.6 (a) terlihat bahwa pemrograman mikrokontroler telah menghasilkan frekuensi sinyal yang ditentukan yaitu 100 Hz. Sinyal ini masuk ke pin *Clock* pada IC *driver* motor stepper. Gambar 5.6 (b) terlihat bahwa ada dua keluaran dari IC *driver* motor stepper yang masuk ke gerbang AND yaitu sinyal A dan INH1, terlihat

bahwa sinyal *chopper* PWM bekerja yang membuat sinyal INH1 berlogika rendah sesaat. Gambar 5.6 (c) menunjukkan masukkan *gate* MOSFET 1 mengakibatkan arus mengalir dari *drain* menuju *source*. Ketika gate diberikan tegangan 5 V, arus mengalir melewati lilitan motor, MOSFET dan Rs, tegangan yang terlihat adalah tegangan dari Vds MOSFET. Ketika *gate* diberikan tegangan 0 V, arus mengalir melewati lilitan motor dan diarahkan langsung ke *supply* melewati dioda (Gambar 4.5), tegangan yang terlihat adalah tegangan *supply* dikurangi tegangan dari pengisian arus pada koil. Gambar 5.6 (d) menunjukkan tegangan pada *drain* MOSFET 1 (atas) dan tegangan pada *drain* MOSFET 2 (bawah) yang berbeda fasa 180^0 bersama dengan Gambar 5.6 (e) tegangan pada *drain* MOSFET 1 (atas) dan tegangan pada *drain* MOSFET 3 (bawah) yang berbeda fasa 90^0. Hal ini menunjukkan mode pengoperasian motor stepper secara *full drive* sesuai dengan perancangan.

5.4 Pengujian Sistem Mikrokontroler *Data Setter*

Pengujian sistem mikrokontroler *data setter* bertujuan untuk memastikan apakah mikrokontroler dapat membaca dan menulis parameter yang diperlukan dari dan ke IC EEPROM. Pengujian dilakukan dengan cara memasukkan parameter posisi dan kecepatan pada masing-masing lokasi memori kemudian ditampilkan pada modul LCD. Diagram blok pengujian ditunjukkan pada Gambar 5.7.

Gambar 5.7 Diagram Blok Pengujian Sistem Mikrokontroler *Data Setter*

Hasil pengujian ditunjukkan pada Tabel 5.2. Pengujian dilakukan dengan prosedur sebagai berikut :

1. Melakukan penulisan parameter posisi dan kecepatan pada lokasi 1, 2, dan 3 EEPROM pada data setter.
2. Mematikan catu daya kemudian melakukan pembacaan parameter posisi dan kecepatan pada lokasi 1, 2, dan 3.
3. Mengulangi langkah 2 sampai 3 kali percobaan.
4. Mentransfer parameter posisi dan kecepatan pada lokasi 1 ke EEPROM rangkaian mikrokontroler pengatur driver motor stepper.

5. Mematikan catu daya kemudian membaca parameter posisi dan kecepatan pada EEPROM rangkaian mikrokontroler pengatur *driver* motor stepper.
6. Mengulangi langkah 4 dan 5 dengan parameter pada lokasi 2 dan 3.

Tabel 5.2 Hasil Pengujian Penulisan dan Pembacaan Parameter

Lokasi 1					
Posisi ke-	Parameter yang ditulis	Data yang ditulis	Pengujian 1	Pengujian 2	Pengujian 3
1	Posisi (jumlah pulsa step)	100	100	100	100
	Kecepatan (periode pulsa dalam ms)	4	4	4	4
2	Posisi (jumlah pulsa step)	4000	4000	4000	4000
	Kecepatan (periode pulsa dalam ms)	4	4	4	4
3	Posisi (jumlah pulsa step)	5000	5000	5000	5000
	Kecepatan (periode pulsa dalam ms)	4	4	4	4
4	Posisi (jumlah pulsa step)	6000	6000	6000	6000
	Kecepatan (periode pulsa dalam ms)	4	4	4	4
Lokasi 2					
Posisi ke-	Parameter yang ditulis	Data yang ditulis	Pengujian 1	Pengujian 2	Pengujian 3
1	Posisi (jumlah pulsa step)	4000	4000	4000	4000
	Kecepatan (periode pulsa dalam ms)	4	4	4	4
2	Posisi (jumlah pulsa step)	3000	3000	3000	3000
	Kecepatan (periode pulsa dalam ms)	5	5	5	5
3	Posisi (jumlah pulsa step)	2300	2300	2300	2300
	Kecepatan (periode pulsa dalam ms)	6	6	6	6
4	Posisi (jumlah pulsa step)	1100	1100	1100	1100
	Kecepatan (periode pulsa dalam ms)	5	5	5	5
Lokasi 3					
Posisi ke-	Parameter yang ditulis	Data yang ditulis	Pengujian 1	Pengujian 2	Pengujian 3
1	Posisi (jumlah pulsa step)	5000	5000	5000	5000
	Kecepatan (periode pulsa dalam ms)	4	4	4	4

2	Posisi (jumlah pulsa step)	4900	4900	4900	4900
	Kecepatan (periode pulsa dalam ms)	5	5	5	5
3	Posisi (jumlah pulsa step)	6400	6400	6400	6400
	Kecepatan (periode pulsa dalam ms)	5	5	5	5
4	Posisi (jumlah pulsa step)	2100	2100	2100	2100
	Kecepatan (periode pulsa dalam ms)	5	5	5	5

Dari hasil pengujian didapatkan seluruh parameter posisi dan kecepatan yang ditulis pada posisi ke 1 sampai dengan 4 berhasil disimpan oleh EEPROM pada seluruh lokasi 1, 2, dan 3. Selanjutnya, parameter yang telah tersimpan perlu diuji untuk ditransfer ke EEPROM pada mikrokontroler pengatur *driver* motor stepper. Tabel 5.3 menunjukkan hasil pengujian transfer parameter dengan membaca data yang telah ditransfer.

Tabel 5.3 Hasil Pengujian Transfer Parameter

Lokasi 1			
Posisi ke-	Parameter yang ditransfer	Data yang ditransfer	Data yang terbaca
1	Posisi (jumlah pulsa step)	100	100
	Kecepatan (periode pulsa dalam ms)	4	4
2	Posisi (jumlah pulsa step)	4000	4000
	Kecepatan (periode pulsa dalam ms)	4	4
3	Posisi (jumlah pulsa step)	5000	5000
	Kecepatan (periode pulsa dalam ms)	4	4
4	Posisi (jumlah pulsa step)	6000	6000
	Kecepatan (periode pulsa dalam ms)	4	4
Lokasi 2			
Posisi ke-	Parameter yang ditransfer	Data yang ditransfer	Data yang terbaca
1	Posisi (jumlah pulsa step)	4000	4000
	Kecepatan (periode pulsa dalam ms)	4	4
2	Posisi (jumlah pulsa step)	3000	3000
	Kecepatan (periode pulsa dalam ms)	5	5
3	Posisi (jumlah pulsa step)	2300	2300
	Kecepatan (periode pulsa dalam ms)	6	6
4	Posisi (jumlah pulsa step)	1100	1100
	Kecepatan (periode pulsa dalam ms)	5	5
Lokasi 3			

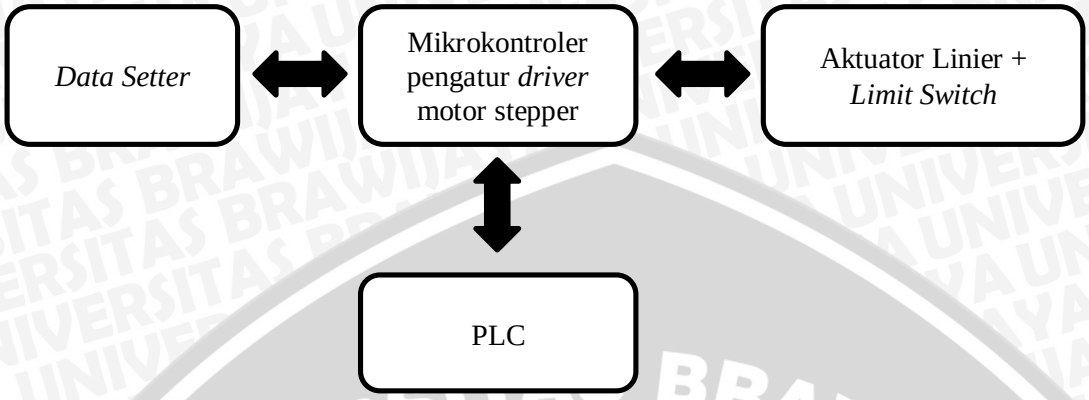
Posisi ke-	Parameter yang ditransfer	Data yang ditransfer	Data yang terbaca
1	Posisi (jumlah pulsa step)	5000	5000
	Kecepatan (periode pulsa dalam ms)	4	4
2	Posisi (jumlah pulsa step)	4900	4900
	Kecepatan (periode pulsa dalam ms)	5	5
3	Posisi (jumlah pulsa step)	6400	6400
	Kecepatan (periode pulsa dalam ms)	5	5
4	Posisi (jumlah pulsa step)	2100	2100
	Kecepatan (periode pulsa dalam ms)	5	5

Dari hasil pengujian didapatkan seluruh parameter posisi dan kecepatan yang ditransfer pada lokasi 1, 2, dan 3 berhasil ditransfer dan dapat dibaca kembali oleh mikrokontroler *data setter*. Hal ini membuktikan EEPROM telah dapat bekerja dengan baik untuk kedua sistem mikrokontroler. Poin terpenting dari pengujian ini ialah pengalamatan memori pada kedua sistem harus sama baik untuk proses tulis maupun proses baca, sehingga data yang ditransfer hasilnya akan sama seperti yang telah dimasukkan.

5.5 Pengujian Sistem Secara Keseluruhan

Pengujian sistem secara keseluruhan dimulai dengan memasukkan parameter posisi dan kecepatan pada *data setter*. Parameter yang telah dimasukkan kemudian ditransfer ke mikrokontroler pengatur *driver* motor stepper. Parameter yang dimasukkan untuk pengujian keseluruhan 3 dan 4 ditunjukkan pada Tabel 5.4 dan Tabel 5.5. Diagram blok pengujian sistem secara keseluruhan ditunjukkan pada Gambar 5.8. Pengujian dilakukan untuk menguji apakah masukkan dari PLC dan *limit switch* dapat direspon oleh mikrokonroler dan dapat memberikan respon yang sesuai dengan perancangan. Pengujian dilakukan dengan cara mengecek sinyal masukkan dan keluaran dari pin-pin yang bersangkutan pada mikrokontroler dengan menggunakan *logic analyzer*. Pengujian keseluruhan 1 dan 2 diuji ketika program ada pada *state Home* yang bertujuan untuk melihat apakah mikrokontroler dapat mengembalikan posisi *shaft leadscrew* ke titik referensi. Pengujian keseluruhan 3 dan 4 diuji ketika program ada pada *state Run* yang bertujuan untuk melihat apakah mikrokontroler dapat menggerakan *shaft leadscrew* sesuai dengan sinyal register M0 dan M1 dan sinyal kontrol *Start*. Pada pengujian keseluruhan 3, kecepatan dari setiap posisi dibuat sama sedangkan pada pengujian keseluruhan 4, keceptan dari posisi 2 dan 3 dibuat berbeda. Pengujian keseluruhan 4 diuji ketika program berada pada *state Run* lalu pindah ke *state Forward* dan *Reverse* sesuai

dengan sinyal dari PLC. Hasil pengujian ditunjukkan pada Gambar 5.9, Gambar 5.10, Gambar 5.11, Gambar 5.12, dan Gambar 5.13.



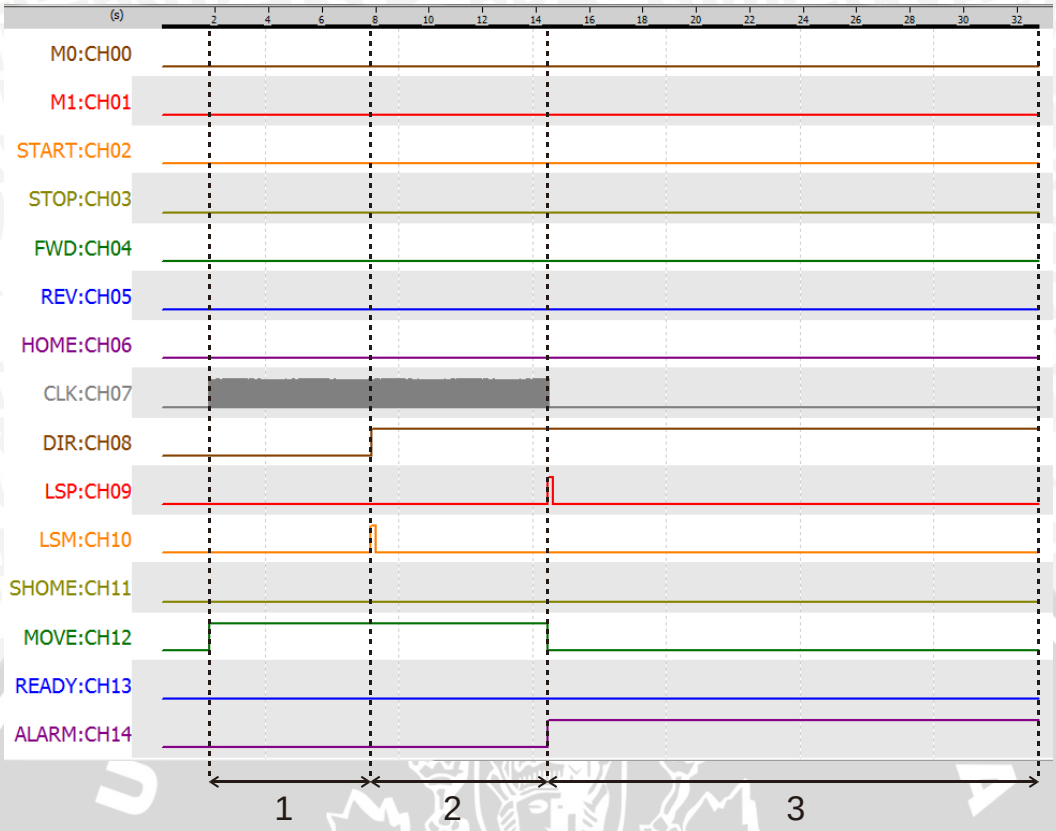
Gambar 5.8 Diagram Blok Pengujian Sistem Secara Keseluruhan

Tabel 5.4 Parameter yang digunakan pada Pengujian Keseluruhan 3

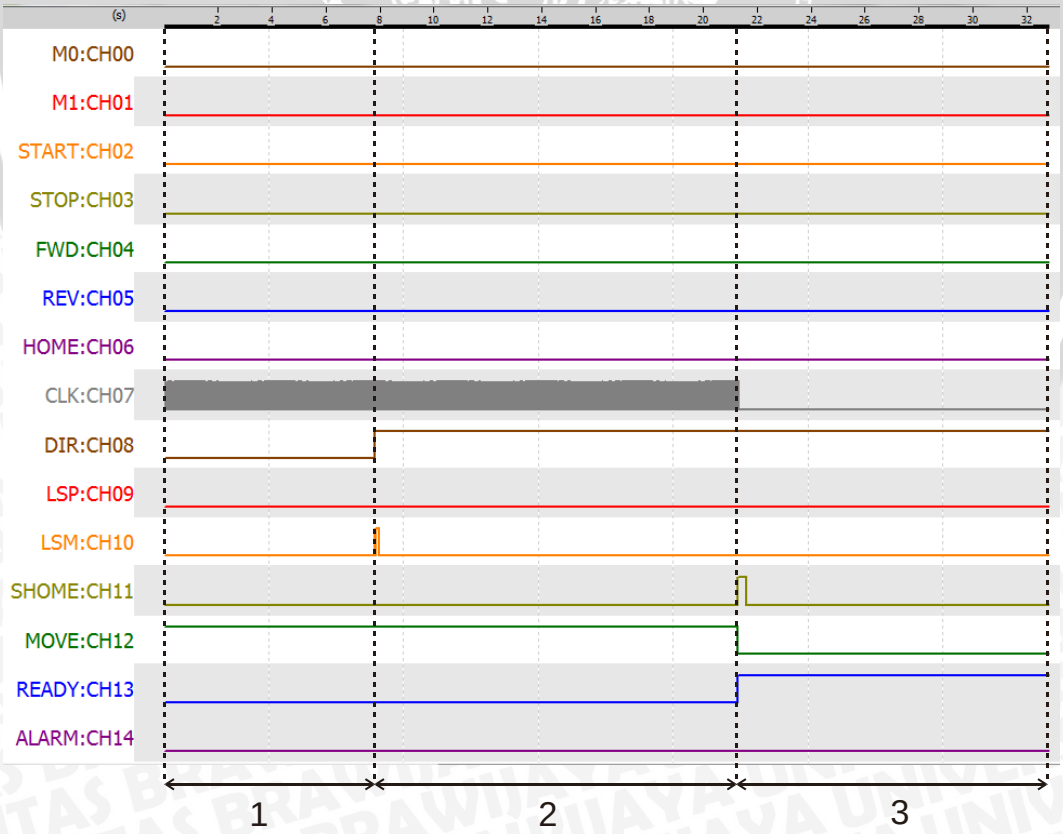
Pengaturan Register Posisi		Parameter yang ditransfer	Parameter yang diinginkan	Parameter yang dimasukkan
M1	M0			
0	0	Posisi 1	0 inci (0 mm)	0
		Kecepatan 1	4 ms (250 Hz)	4
1	0	Posisi 2	1 inci (25,4 mm)	1000
		Kecepatan 2	4 ms (250 Hz)	4
1	0	Posisi 3	1,5 inci (38,1 mm)	1500
		Kecepatan 3	4 ms (250 Hz)	4
1	1	Posisi 4	2 inci (50,8 mm)	2000
		Kecepatan 4	4 ms (250 Hz)	4

Tabel 5.5 Parameter yang Digunakan pada Pengujian Keseluruhan 4

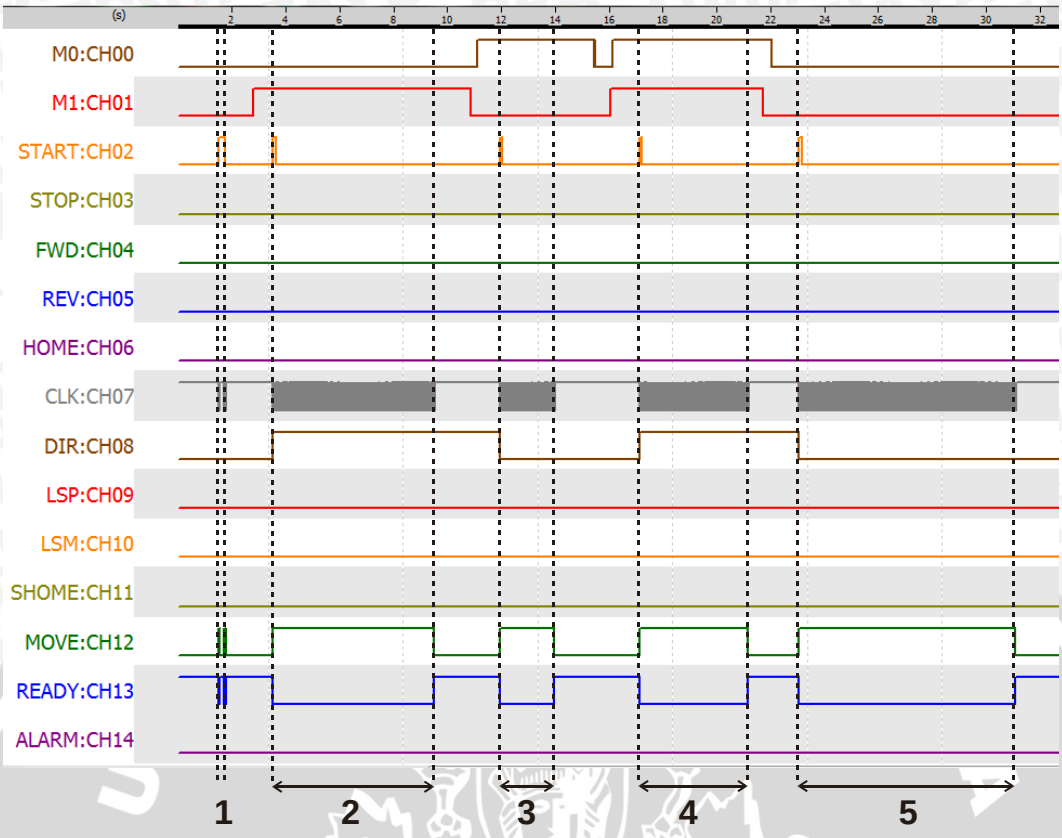
Pengaturan Register Posisi		Parameter yang ditransfer	Parameter yang diinginkan	Parameter yang dimasukkan
M1	M0			
0	0	Posisi 1	0 inci (0 mm)	0
		Kecepatan 1	4 ms (250 Hz)	4
1	0	Posisi 2	1 inci (25,4 mm)	1000
		Kecepatan 2	5 ms (200 Hz)	5
1	0	Posisi 3	1,5 inci (38,1 mm)	1500
		Kecepatan 3	5 ms (200 Hz)	5
1	1	Posisi 4	2 inci (50,8 mm)	2000
		Kecepatan 4	4 ms (250 Hz)	4



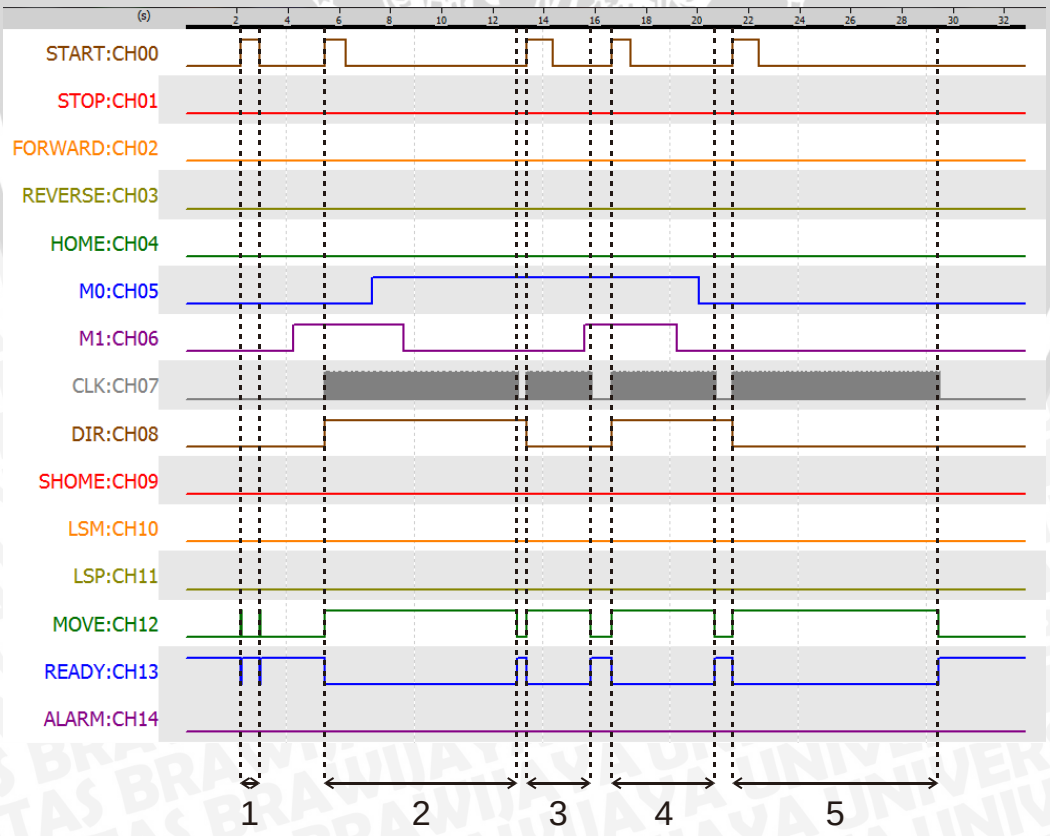
Gambar 5.9 Sinyal Mikrokontroler Hasil Pengujian Keseluruhan 1



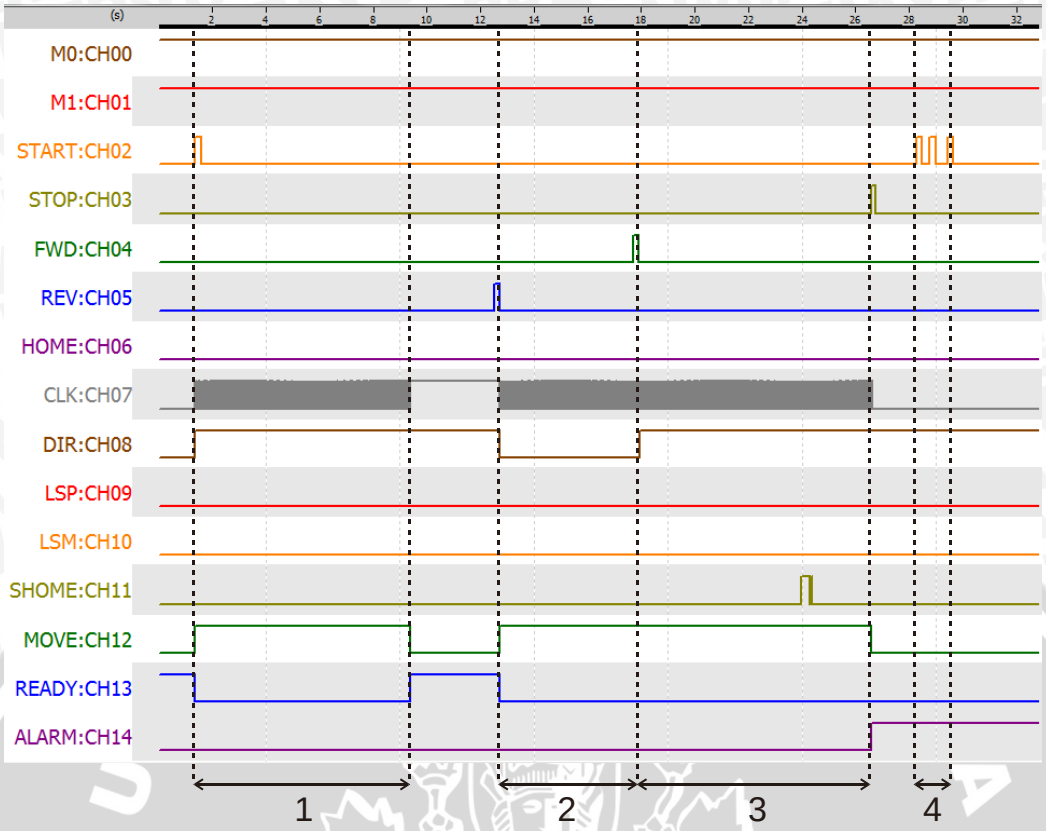
Gambar 5.10 Sinyal Mikrokontroler Hasil Pengujian Keseluruhan 2



Gambar 5.11 Sinyal Mikrokontroler Hasil Pengujian Keseluruhan 3



Gambar 5.12 Sinyal Mikrokontroler Hasil Pengujian Keseluruhan 4



Gambar 5.13 Sinyal Mikrokontroler Hasil Pengujian Keseluruhan 5

Dari Gambar 5.9 tampak pada waktu 1, mikrokontroler memberikan sinyal *Clock* kepada rangkaian *driver* motor stepper sampai *limit switch LSM* aktif dan mikrokontroler mengubah sinyal *Direction* menjadi berlogika 1 yang mengakibatkan motor berputar ke arah sebaliknya. Pada waktu 3, mikrokontroler menghentikan pemberian pulsa *Direction* akibat *limit switch LSP* aktif. Mikrokontroler kemudian menghentikan sinyal *Move* dan memberikan sinyal *Alarm*. Sistem kemudian berada pada *state Stop*. Dari Gambar 5.10 tampak pada waktu 3 *shaft leadscrew* mendeteksi titik referensi dan kemudian mikrokontroler memberikan sinyal *Ready*.

Dari Gambar 5.11 dan Gambar 5.12 tampak pada waktu 1, mikrokontroler sempat mengeluarkan sinyal *Move* akan tetapi langsung selesai dikarenakan posisi yang dituju berjarak 0 inci dari posisi *Home*. Pada waktu 2, 3, dan 4 terlihat mikrokontroler memberikan sinyal *Clock* dan *Move* dan menghentikan sinyal *Ready* kemudian memberikan sinyal *Ready* kembali ketika jumlah *Clock* yang diberikan telah tercapai. Pada waktu 3 dan 5 terlihat mikrokontroler mengubah sinyal *Direction* ke logika 0 yang mengakibatkan motor berputar ke arah sebaliknya. Hal ini dikarenakan parameter posisi yang diberikan adalah posisi yang relatif dari titik *Home*. Analisis pengujian keseluruhan 3 dan 4 ditunjukkan pada Tabel 5.6 dan Tabel 5.7.

Tabel 5.6 Analisis Pengujian Keseluruhan 3

Posisi		Waktu Pemberian	Perpindahan Terukur	Perpindahan	*Error*
Sebelum	Sesudah	Pulsa *Clock* (ms)	(mm)	Seharusnya (mm)	(mm)
Home	Posisi 1	0	0	0	0
Posisi 1	Posisi 3	6023	37,5	38,1	0,6
Posisi 3	Posisi 2	2007	24,4	25,4	1
Posisi 2	Posisi 4	4016	49,5	50,8	1,3
Posisi 4	Posisi 1	8030	2,2	0	2,2

Tabel 5.7 Analisis Pengujian Keseluruhan 4

Posisi		Waktu Pemberian	Perpindahan Terukur	Perpindahan	*Error*
Sebelum	Sesudah	Pulsa *Clock* (ms)	(mm)	Seharusnya (mm)	(mm)
Home	Posisi 1	0	0	0	0
Posisi 1	Posisi 3	7532	36,2	38,1	1,9
Posisi 3	Posisi 2	2510	23	25,4	2,4
Posisi 2	Posisi 4	4016	48,7	50,8	2,1
Posisi 4	Posisi 1	8035	3	0	3

Dari Tabel 5.6 dan Tabel 5.7 masing-masing terlihat untuk menggerakan *shaft leadscrew* ke posisi 3 dari posisi 1, mikrokontroler memberikan sinyal *Clock* kepada rangkaian *driver* selama 6023 dan 7532 ms untuk menggerakan shaft sejauh ±1,5 inci (38,1 mm). Untuk menggerakan *shaft* sejauh ±1 inci (25,4 mm) ke posisi 2 dari posisi 3, masing-masing selama 2007 dan 2510 ms. Hal ini dikarenakan frekuensi yang diberikan ke motor adalah 250 Hz dan 200 Hz sesuai dengan parameter yang diberikan ke *data setter* pada Tabel 5.4 dan Tabel 5.5. Untuk menggerakan *shaft* dari posisi 2 ke posisi 4 dan posisi 4 ke posisi 1 terlihat waktu pemberian pulsa kurang lebih sama. Jarak yang ditempuh *shaft* relatif dari titik *Home* mengalami kesalahan terbesar sejauh 3 mm atau 0,118 inci. Hal ini dikarenakan motor tidak dapat berputar dengan sempurna sesuai dengan pulsa *step* yang diberikan *driver* motor ketika torsi yang diberikan untuk menggerakan *shaft* kurang yang mengakibatkan terjadi kesalahan.

Dari Gambar 5.13 dapat terlihat pada waktu 1 mikrokrontroler memberikan sinyal *Clock* dan *Direction* kepada *driver* dengan perintah dari sinyal *Start*. Pada waktu 2 dan 3 terlihat PLC memberikan sinyal *Reverse* dan *Forward* yang mengakibatkan sistem berpindah ke *state Reverse* dan *Forward*. *State* Forward selesai ketika ada sinyal *Stop* dan sistem pindah ke *state Stop* ditandai dengan mikrokontroler memberikan sinyal *Alarm*. Pada waktu 4 terlihat PLC memberikan sinyal *Start* akan tetapi sistem tidak pindah ke *state Run* ditandai dengan sinyal *Clock* yang tidak berubah.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan analisis dan pengujian dari perancangan kontrol gerak *leadscrew* menggunakan PLC berbasis mikrokontroler dapat ditarik kesimpulan sebagai berikut :

1. Mekanik aktuator linier yang dibuat memiliki panjang efektif 6 inci dan untuk menggerakan *shaft leadscrew* sejauh 1 inci diperlukan 5 putaran (5 TPI). Motor stepper yang digunakan memiliki jumlah *step* per revolusi sebesar 200 SPR sehingga untuk menggerakan *shaft leadscrew* sejauh 1 inci akan diperlukan 1000 pulsa *step* (urutan pemberian energi) pada lilitan motor. Dari pengujian diketahui semakin jauh perpindahan *shaft leadscrew* dari titik referensi maka akan semakin besar *error* yang terjadi. Rata-rata *error* terkecil berada pada *range* frekuensi 125 - 250 Hz, e*rror* maksimum atau ketelitian alat ini pada range frekuensi tersebut adalah 2,2 mm pada perpindahan *shaft leadscrew* sejauh 6 inci (152,4 mm), berdasarkan hasil tersebut, maka resolusi dari alat ini adalah 87 pulsa.
2. PLC dapat dihubungkan dengan mikrokontroler menggunakan rangkaian konversi level tegangan. Sinyal yang diberikan oleh PLC sudah sesuai dengan level tegangan mikrokontroler sehingga dapat langsung dihubungkan. Sinyal yang diberikan mikrokontroler untuk PLC perlu dikonversi terlebih dahulu ke level tegangan PLC yaitu 24 V menggunakan rangkaian *relay*. Pada pengujian didapatkan terdapat jeda sinyal yang singkat (kurang dari 100 ms) akibat rangkaian konversi level tegangan dari mikrokontroler ke PLC.
3. Rangkaian *data setter* memiliki fungsi sebagai penyimpan parameter kontrol gerak berupa posisi dan kecepatan dari tiap-tiap register posisi. Parameter posisi yang diisikan adalah posisi relatif dari titik referensi. Parameter posisi yang diberikan dibatasi dengan ketelitian sebesar 100 pulsa dan parameter kecepatan yang diberikan dibatasi terdapat 3 mode yaitu dengan kecepatan 250 Hz, 200 Hz dan 166,67 Hz. Dari pengujian didapatkan rangkaian *data setter* dapat menyimpan dan membaca parameter ke rangkaian mikrokontroler pengatur *driver*.
4. Mikrokontroler dapat menggerakan *leadscrew* ke posisi tertentu dengan menggunakan parameter posisi dan kecepatan yang ditentukan oleh *data setter*.

Sinyal yang diberikan oleh PLC dapat direspon oleh mikrokontroler sesuai dengan *state diagram* yang telah dirancang.

6.2 Saran

Beberapa hal yang direkomendasikan untuk penelitian dan pengembangan berikutnya ialah sebagai berikut :

1. Penambahan sistem *loop* tertutup menggunakan *rotary encoder* agar posisi *shaft leadscrew* dapat dikoreksi ketika pergerakan *shaft* tidak sesuai dengan sinyal dari mikrokontroler.
2. Penambahan parameter kontrol gerak yang lain seperti akselerasi dan deselerasi untuk sistem dengan kecepatan tinggi dikarenakan motor stepper memiliki *detent torque* atau torsi minimum yang diperlukan untuk menggerakan motor saat mulai berputar dari keadaan berhenti.

DAFTAR PUSTAKA

Atmel. 2009. *ATmega1284P Preliminary*. San Jose: Atmel.

Barr, M. May, 2001. Memory Types. *Embedded Systems Programming*, 103 - 104.

Bolton, W. 2004. *Mechatronics : Electronic Control Systems in Mechanical Engineering / Edition 3*. New Jersey: Prentice Hall.

Callister, W. D. 1997. *Material Science and Engineering, an Introduction*. New York: John Wiley & Sons.

Condit, R., & Jones, D. W. 2004. *Stepping Motor Fundamentals*. Chandler: Microchip Technology.

Kenjo, T. 1984. *Stepping Motors and Their Microprocessor Controls*. Oxford: Oxford University Press.

Microchip Technology. 2012. *24AA64/24LC64/24FC64 64K I2C Serial EEPROM*. Chandler: Microchip Technology.

Omron. 2009. *CP1L CPU Unit Operation Manual*. Kyoto: Omron.

Soloman, S. 1994. *Sensors and Control Systems in Manufacturing*. Singapore: McGraw-Hill.

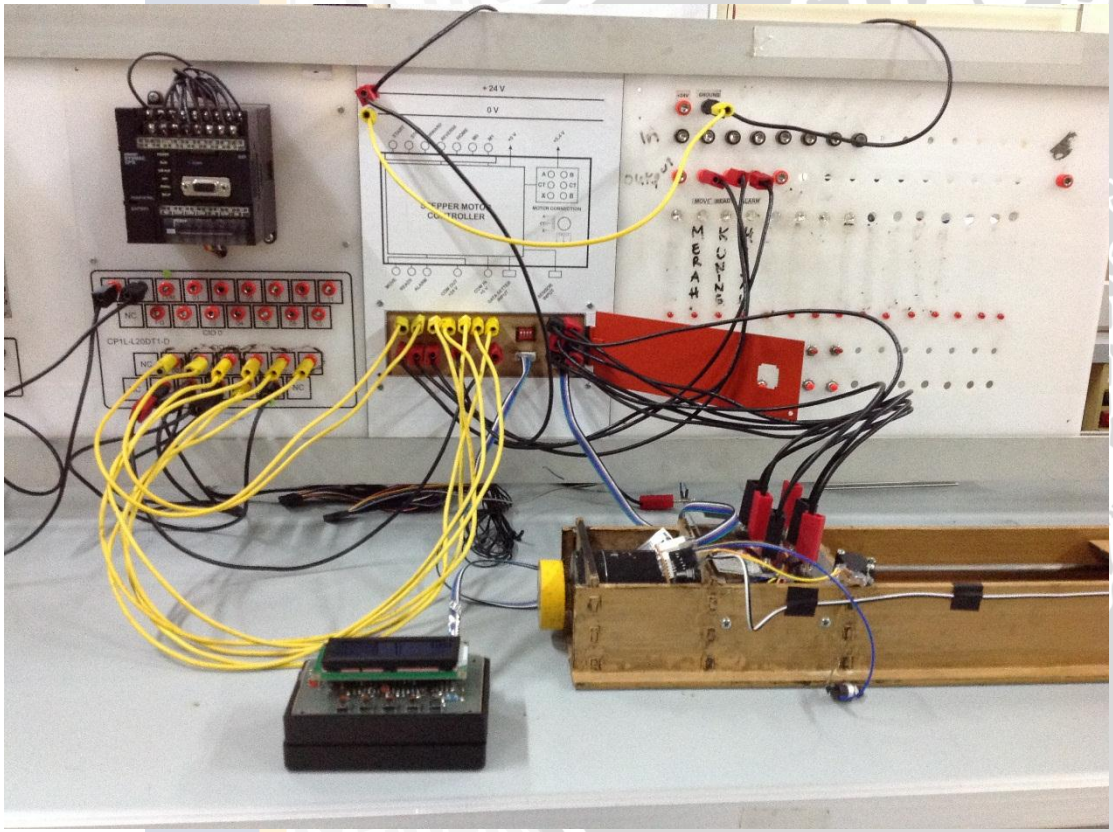
STMicroelectronics. 2003. *AN470 Aplication Note: The L297 Stepper Motor Controller* . Italy: STMicroelectronics.

Texas Instrument. 2013. *ULN2003A*. Dallas: Texas Instrument.

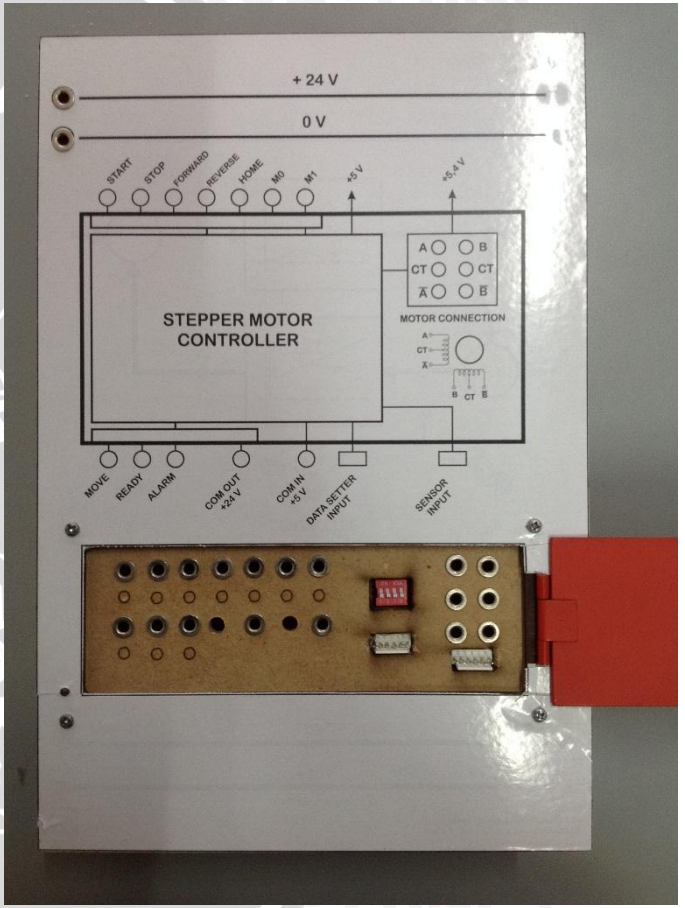
Winoto, A. 2010. *Mikrokontroler AVR ATmega8/32/16/8535 dan Pemrogramannya dengan Bahasa C pada WinAVR*. Bandung: Informatika.

LAMPIRAN

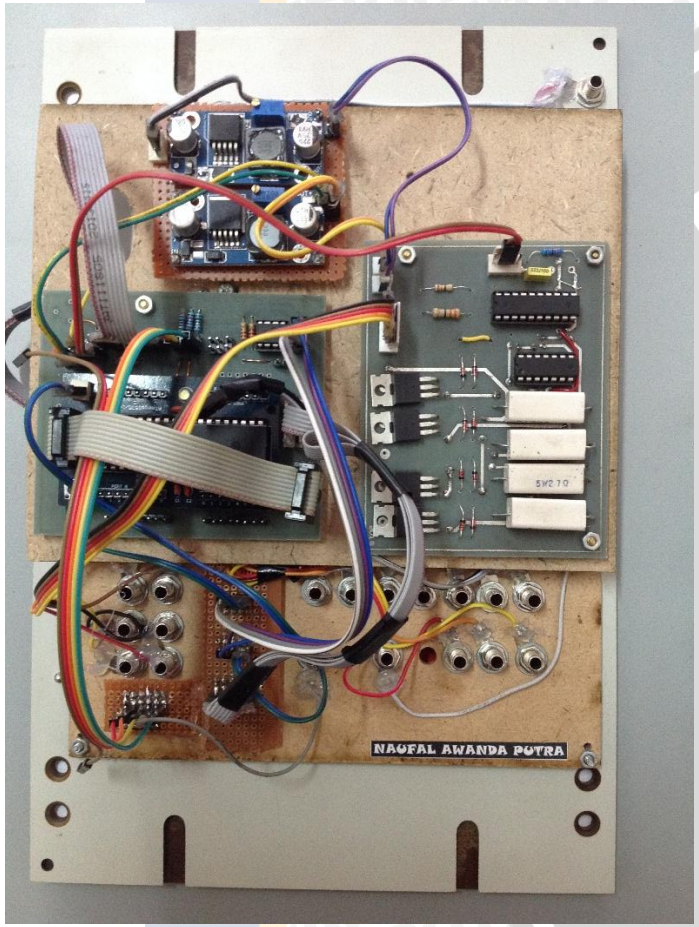
Lampiran 1 Foto Alat



Alat Keseluruhan

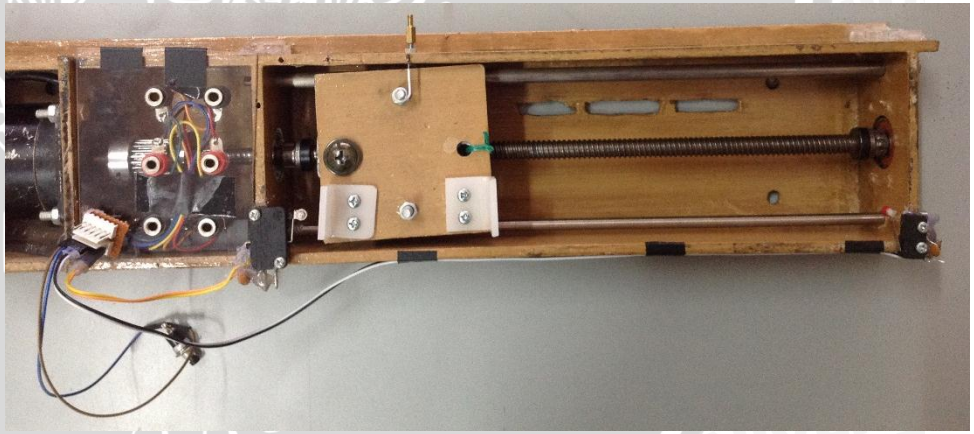


Modul Rangkaian Mikrokontroler



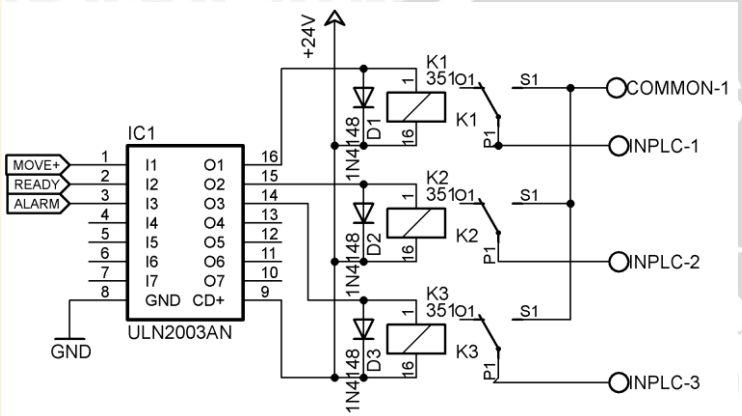
Modul Rangkaian Mikrokontroler

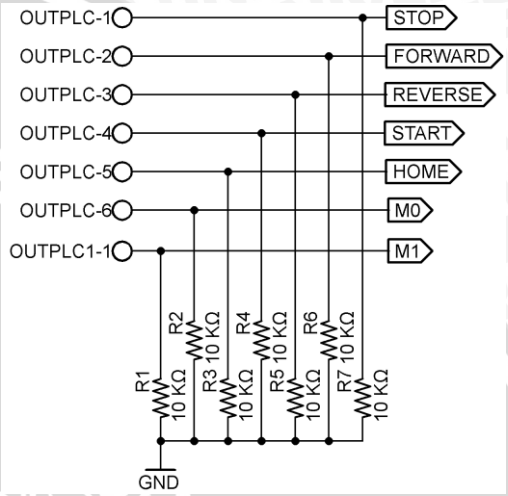
Data Setter

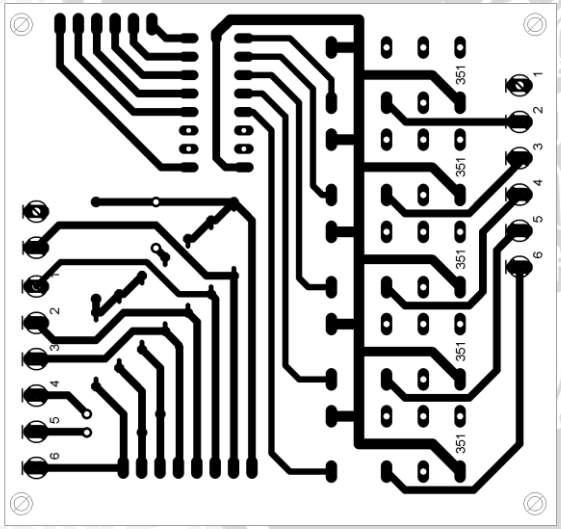


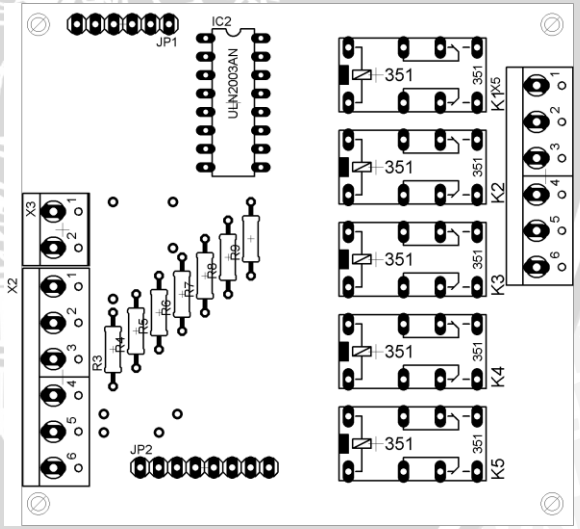
Aktuator Linier

Lampiran 2 Rangkaian Konversi Level Tegangan

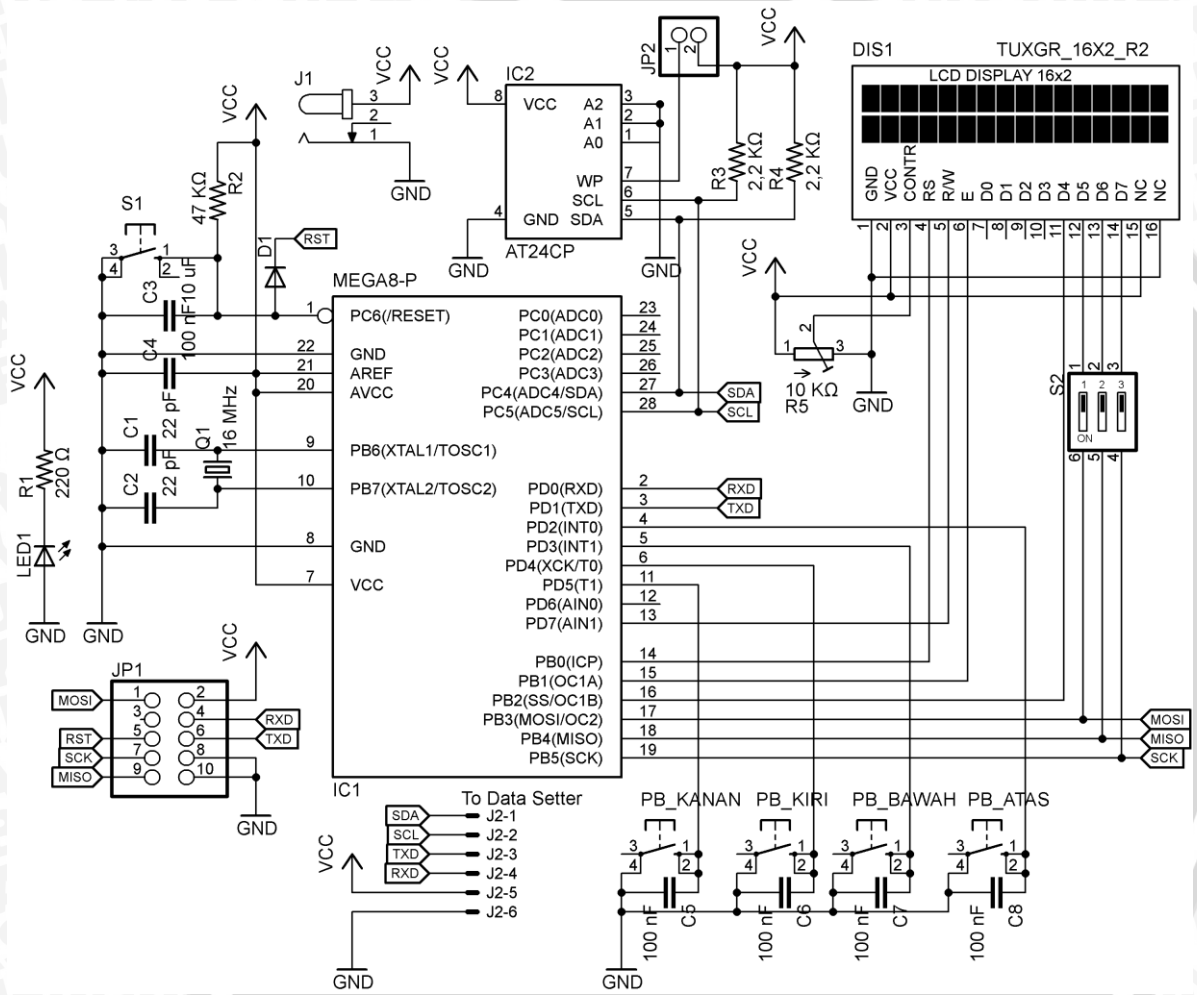


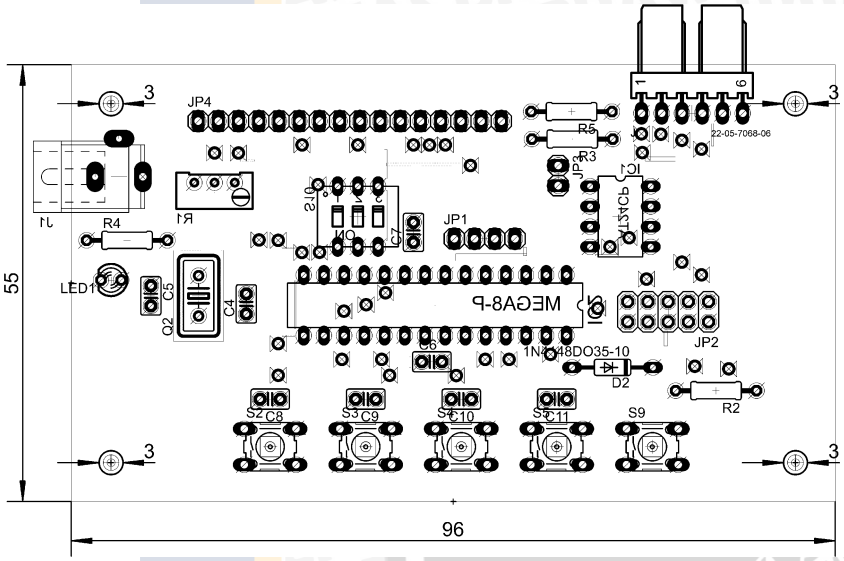


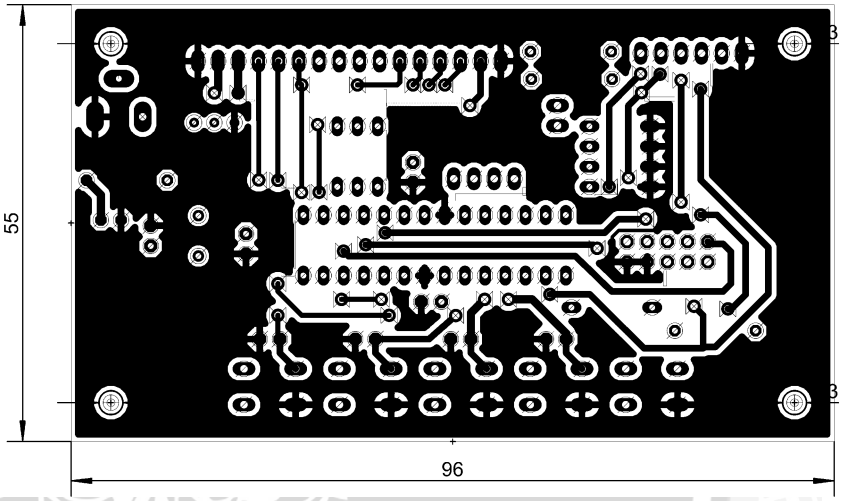


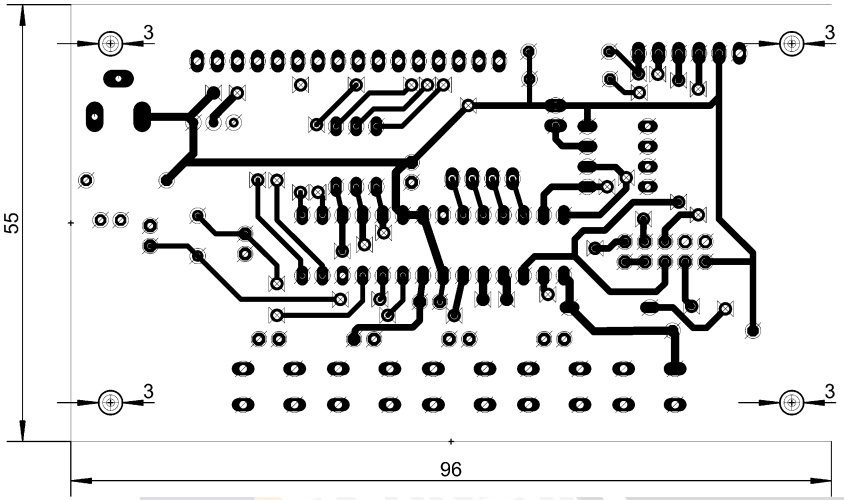


Lampiran 3 Rangkaian *Data Setter*

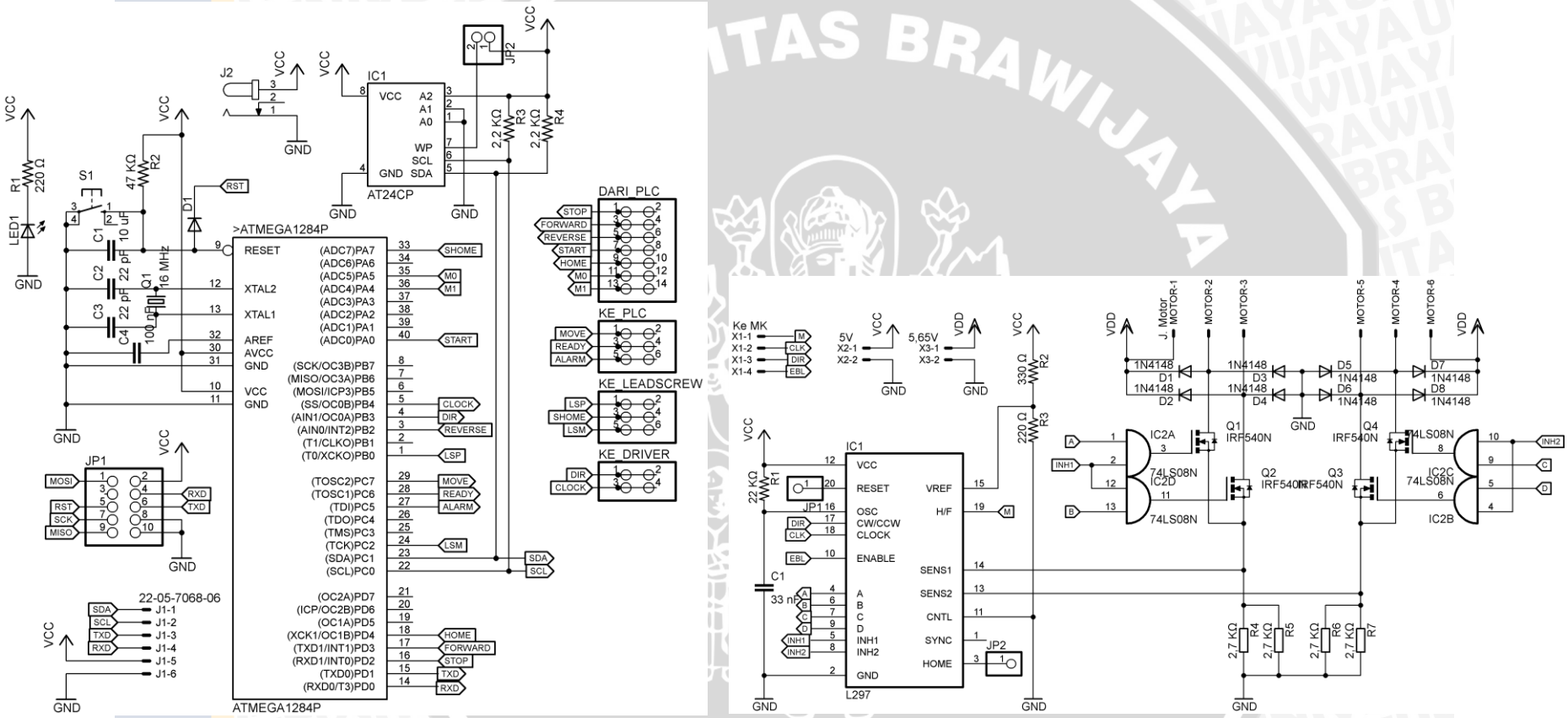


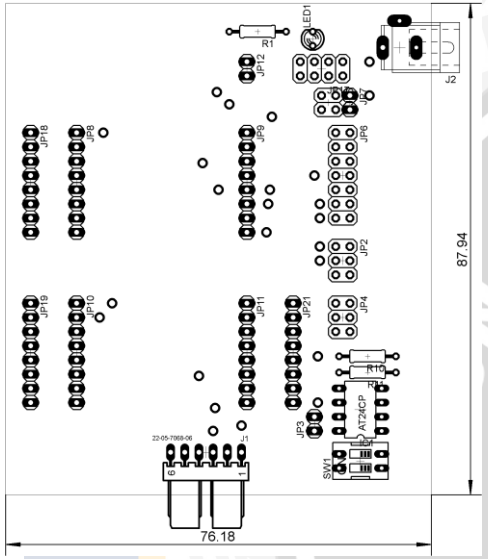

JP4
R5
R3
22-05-7068-06
JP1
R4
LED1
C5
C4
Q2
JP2
R2
D2
S2
S3
S4
S9
C8
C9
C10
C11
55
96
3

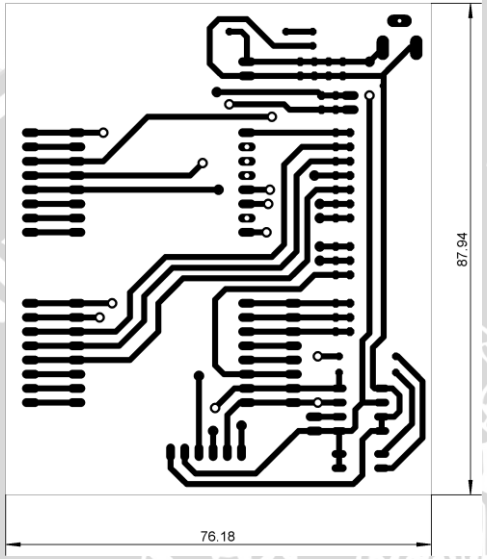

55
96
3

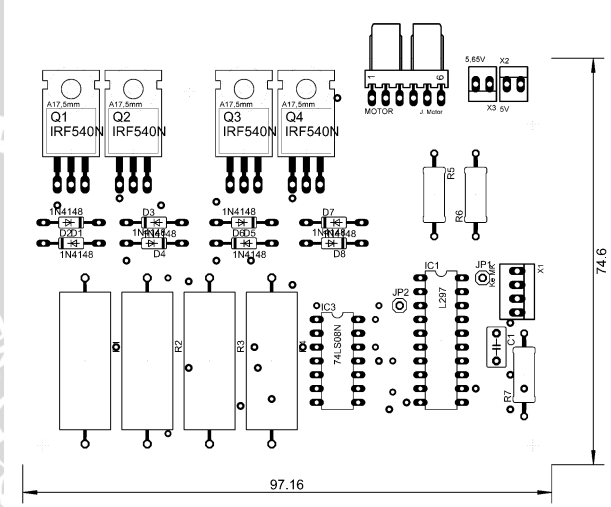

55
96
3

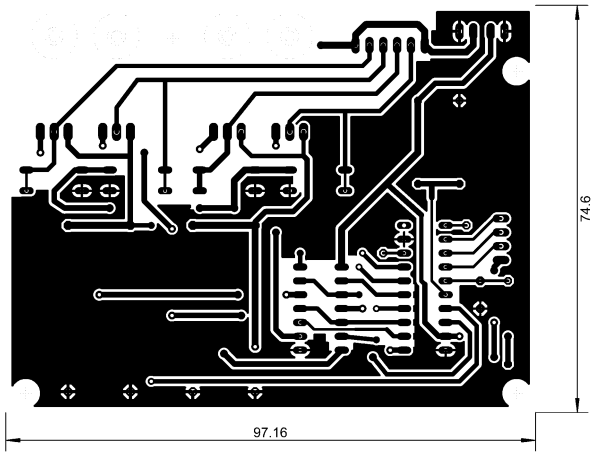
Lampiran 4 Rangkaian Mikrokontroler Pengatur *Driver* Motor Stepper

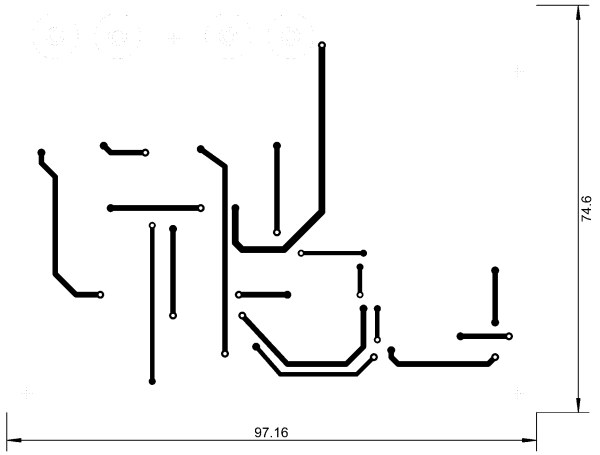



LED1
R1
JP12
J2
JP16
JP8
JP9
JP6
JP2
JP10
JP11
JP21
JP4
R16
JP3
SW1
J1
87.94
76.18


87.94
76.18


Q1
IRF540N
Q2
IRF540N
Q3
IRF540N
Q4
IRF540N
MOTOR
5.65V
X2
X3 5V
1N4148
D1
D2
D3
D4
D5
D6
D7
D8
R5
R6
IC1
L297
IC3
74LS08N
JP2
R2
R3
R7
C1
X1
74.6
97.16


74.6
97.16


74.6
97.16

Lampiran 5 Program *Data Setter*

```
/*****************************************
***********
This program was produced by the
CodeWizardAVR V2.05.0 Evaluation
Automatic Program Generator
© Copyright 1998-2010 Pavel Haiduc, HP
InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 11/4/2015
Author  : Freeware, for evaluation and non-
commercial use only
Company :
Comments:


Chip type               : ATmega8
Program type            : Application
AVR Core Clock frequency: 16.000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************
**********/

//=============== HEADER ===============//

#include <mega8.h>
#include <delay.h>
#include <i2c.h>
#include <alcd.h>
#include <stdio.h>
#include <stdlib.h>

//==== DEFINE VARIABEL ====//

#define UP PIND.5
#define DOWN PIND.4
#define LEFT PIND.3
#define RIGHT PIND.2

#define EEPROM1_WR 0xA0
#define EEPROM1_RD 0xA1
#define EEPROM2_WR 0xA8
#define EEPROM2_RD 0xA9

//==============I2C==============//

#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=4
   .equ __scl_bit=5
#endasm


//=============== EEPROM ==============//

unsigned char eeprom_read(unsigned int
address16, unsigned char address_write,
unsigned char address_read)
{
    unsigned char data;
    unsigned char address_high;
    unsigned char address_low;
    address_high=(address16 & 0xFF00)>>8;
    address_low=address16 & 0xFF;
    i2c_start();
    i2c_write(address_write);
    i2c_write(address_high);
    i2c_write(address_low);
    i2c_start();
    i2c_write(address_read);
    data=i2c_read(0);
    i2c_stop();

return data;
}

void eeprom_write(unsigned int address16,
unsigned char data, unsigned char
address_write)
{
    unsigned char address_high;
    unsigned char address_low;
    address_high=(address16 & 0xFF00)>>8;
    address_low=address16 & 0xFF;
    i2c_start();
    i2c_write(address_write);
    i2c_write(address_high);
    i2c_write(address_low);
    i2c_write(data);
    i2c_stop();
    delay_ms(10);
}


//======= LCD PUT INTEGER ===========//

unsigned char buff[33];
void lcd_putint(unsigned int dat)
{
    sprintf(buff,"%d ",dat);
    lcd_puts(buff);
}


void main(void)
{
// Declare your local variables here
```

```
PORTB=0x00;
DDRB=0x00;
PORTC=0x00;
DDRC=0x0F;
PORTD=0b00111100;
DDRD=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1
Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x67;

// I2C Bus initialization
i2c_init();

lcd_init(16);



while (1)
      {
        unsigned int alamat = 0;
        unsigned int pos_1, pos_2, pos_3,
pos_4 = 0;
        unsigned char acc_1, acc_2, acc_3,
acc_4 = 1;
        unsigned char pos1a, pos1b, pos2a,
pos2b, pos3a, pos3b, pos4a, pos4b;

char load, save;

m_1:
lcd_gotoxy(0,0);
lcd_puts("DATA SETTER");
lcd_gotoxy(0,1);
lcd_puts("SET PARAMETER");

if(!RIGHT)
{
    delay_ms(250);
    lcd_clear();
    m_2:
    lcd_gotoxy(0,0);
    lcd_puts("DATA SETTER");
    lcd_gotoxy(0,1);
    lcd_puts("LOAD PARAMETER");
    if(!RIGHT)
    {
       delay_ms(250);
       lcd_clear();
       goto m_3;
    }
    if(!LEFT)
    {
       delay_ms(250);
       lcd_clear();
       goto m_1;
    }
    if(!DOWN)
    {
       delay_ms(250);
       lcd_clear();
       load1:
       lcd_gotoxy(0,0);
       lcd_puts("LOAD PARAMETER");
       lcd_gotoxy(0,1);
       lcd_puts("LOAD-1");
       if(!RIGHT)
       {
            delay_ms(250);
            load = 0;
            goto loading;
       }
       if(!DOWN)
       {
            delay_ms(250);
            lcd_clear();
            goto load2;
       }
       if(!UP)
       {
            delay_ms(250);
            lcd_clear();
            goto m_2;
       }
       goto load1;

       load2:
       lcd_gotoxy(0,0);
       lcd_puts("LOAD PARAMETER");
       lcd_gotoxy(0,1);
       lcd_puts("LOAD-2");
       if(!RIGHT)
       {
            delay_ms(250);
            load = 1;
            goto loading;
       }
       if(!DOWN)
       {
            delay_ms(250);
            lcd_clear();
            goto load3;
       }
       if(!UP)
       {
            delay_ms(250);
            lcd_clear();
            goto load1;
```

```
}
goto load2;

load3:
lcd_gotoxy(0,0);
lcd_puts("LOAD PARAMETER");
lcd_gotoxy(0,1);
lcd_puts("LOAD-3");
if(!RIGHT)
{
    delay_ms(250);
    load = 2;
    goto loading;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto m_2;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto load2;
}
goto load3;

loading:

lcd_clear();
lcd_gotoxy(0,0);
lcd_puts("LOAD PARAMETER");
lcd_gotoxy(0,1);
lcd_puts("LOADING...");

switch(load)
{
    case 0: alamat = 0;
break;
                    case 1: alamat = 20;
break;
                    case 2: alamat = 40;
break;
                    default: alamat = 60;
break;
                }
pos1a=eeprom_read(alamat, EEPROM1_WR,
EEPROM1_RD);
pos1b=eeprom_read(alamat+1, EEPROM1_WR,
EEPROM1_RD);
pos2a=eeprom_read(alamat+2, EEPROM1_WR,
EEPROM1_RD);
pos2b=eeprom_read(alamat+3, EEPROM1_WR,
EEPROM1_RD);
pos3a=eeprom_read(alamat+4, EEPROM1_WR,
EEPROM1_RD);
pos3b=eeprom_read(alamat+5, EEPROM1_WR,
EEPROM1_RD);
pos4a=eeprom_read(alamat+6, EEPROM1_WR,
EEPROM1_RD);
pos4b=eeprom_read(alamat+7, EEPROM1_WR,
EEPROM1_RD);
acc_1=eeprom_read(alamat+10, EEPROM1_WR,
EEPROM1_RD);
acc_2=eeprom_read(alamat+11, EEPROM1_WR,
EEPROM1_RD);
acc_3=eeprom_read(alamat+12, EEPROM1_WR,
EEPROM1_RD);
acc_4=eeprom_read(alamat+13, EEPROM1_WR,
EEPROM1_RD);
pos_1 = pos1a + (pos1b << 8);
pos_2 = pos2a + (pos2b << 8);
pos_3 = pos3a + (pos3b << 8);
pos_4 = pos4a + (pos4b << 8);
                delay_ms(100);
                goto m_2;
            }

            goto m_2;
m_3:
lcd_gotoxy(0,0);
lcd_puts("DATA SETTER");
lcd_gotoxy(0,1);
lcd_puts("SAVE PARAMETER");

if(!RIGHT)
{
    delay_ms(250);
    lcd_clear();
    goto m_4;
}

if(!LEFT)
{
    delay_ms(250);
    lcd_clear();
    goto m_2;
}

if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    save1:
    lcd_gotoxy(0,0);
    lcd_puts("SAVE PARAMETER");
    lcd_gotoxy(0,1);
    lcd_puts("SAVE-1");
    if(!RIGHT)
    {
        delay_ms(250);
        save = 0;
        goto saving;
    }
    if(!DOWN)
    {
        delay_ms(250);
        lcd_clear();
```

```
    goto save2;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto m_3;
}
goto save1;

save2:
lcd_gotoxy(0,0);
lcd_puts("SAVE PARAMETER");
lcd_gotoxy(0,1);
lcd_puts("SAVE-2");
if(!RIGHT)
{
    delay_ms(250);
    save = 1;
    goto saving;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto save3;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto save1;
}
goto save2;

save3:
lcd_gotoxy(0,0);
lcd_puts("SAVE PARAMETER");
lcd_gotoxy(0,1);
lcd_puts("SAVE-3");
if(!RIGHT)
{
    delay_ms(250);
    save = 2;
    goto saving;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto m_3;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto save2;
}
goto save3;

saving:
lcd_gotoxy(0,0);
lcd_puts("SAVE PARAMETER");
lcd_gotoxy(0,1);
lcd_puts("SAVING...");

switch(save)
{
    case 0: alamat = 0;
break;
    case 1: alamat = 20;
break;
    case 2: alamat = 40;
break;
    default: alamat = 60;
break;
}
pos1a=pos_1 & 0xFF;
pos1b=(pos_1>>8) & 0xFF;
pos2a=pos_2 & 0xFF;
pos2b=(pos_2>>8) & 0xFF;
pos3a=pos_3 & 0xFF;
pos3b=(pos_3>>8) & 0xFF;
pos4a=pos_4 & 0xFF;
pos4b=(pos_4>>8) & 0xFF;
eeprom_write(alamat,pos1a,EEPROM1_WR);
eeprom_write(alamat+1,pos1b, EEPROM1_WR);
eeprom_write(alamat+2,pos2a, EEPROM1_WR);
eeprom_write(alamat+3,pos2b, EEPROM1_WR);
eeprom_write(alamat+4,pos3a, EEPROM1_WR);
eeprom_write(alamat+5,pos3b, EEPROM1_WR);
eeprom_write(alamat+6,pos4a, EEPROM1_WR);
eeprom_write(alamat+7,pos4b, EEPROM1_WR);
eeprom_write(alamat+10,acc_1, EEPROM1_WR);
eeprom_write(alamat+11,acc_2, EEPROM1_WR);
eeprom_write(alamat+12,acc_3, EEPROM1_WR);
eeprom_write(alamat+13,acc_4, EEPROM1_WR);

    delay_ms(250);

    goto m_3;
}


goto m_3;

m_4:
lcd_gotoxy(0,0);
lcd_puts("DATA SETTER");
lcd_gotoxy(0,1);
lcd_puts("SET KECEPATAN");

if(!RIGHT)
{
    delay_ms(250);
    lcd_clear();
    goto m_5;
}
```

```
if(!DOWN)
{
delay_ms(250);
lcd_clear();
acc1:
lcd_gotoxy(0,0);
lcd_puts("KEC-00 (1-3)");
lcd_gotoxy(0,1);
lcd_putint(acc_1);
if(!RIGHT)
{
    delay_ms(250);
    acc_1++;
}
if(!LEFT)
{
    delay_ms(250);
    acc_1--;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto m_4;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto acc2;
}
goto acc1;

acc2:
lcd_gotoxy(0,0);
lcd_puts("KEC-01 (1-3)");
lcd_gotoxy(0,1);
lcd_putint(acc_2);
if(!RIGHT)
{
    delay_ms(250);
    acc_2++;
}
if(!LEFT)
{
    delay_ms(250);
    acc_2--;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto acc1;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto acc3;
}
goto acc2;

acc3:
lcd_gotoxy(0,0);
lcd_puts("KEC-10 (1-3)");
lcd_gotoxy(0,1);
lcd_putint(acc_3);
if(!RIGHT)
{
    delay_ms(250);
    acc_3++;
}
if(!LEFT)
{
    delay_ms(250);
    acc_3--;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto acc2;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto acc4;
}
goto acc3;

acc4:
lcd_gotoxy(0,0);
lcd_puts("KEC-11 (1-3)");
lcd_gotoxy(0,1);
lcd_putint(acc_4);
if(!RIGHT)
{
    delay_ms(250);
    acc_4++;
}
if(!LEFT)
{
    delay_ms(250);
    acc_4--;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto acc3;
}

if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto m_4;
}
```

```
goto acc4;
}

if(!LEFT)
{
    delay_ms(250);
    lcd_clear();
    goto m_3;
}
goto m_4;

 m_5:
lcd_gotoxy(0,0);
lcd_puts("DATA SETTER");
lcd_gotoxy(0,1);
lcd_puts("LOAD DARI DRIVER");
if(!RIGHT)
{
    delay_ms(250);
    lcd_clear();
    goto m_6;
}
if(!LEFT)
{
    delay_ms(250);
    lcd_clear();
    goto m_4;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    load_menu:
    lcd_gotoxy(0,0);
    lcd_puts("LOAD DARI
DRIVER");
    lcd_gotoxy(0,1);
    lcd_puts("LOAD?");
                        if(!RIGHT)
                        {
                            delay_ms(250);
                            lcd_clear();
                            lcd_gotoxy(0,0);
                            lcd_puts("LOAD DARI
DRIVER");
                            lcd_gotoxy(0,1);
                            lcd_puts("LOADING...");

                            alamat=0;

pos1a=eeprom_read(alamat, EEPROM2_WR,
EEPROM2_RD);
pos1b=eeprom_read(alamat+1, EEPROM2_WR,
EEPROM2_RD);
pos2a=eeprom_read(alamat+2, EEPROM2_WR,
EEPROM2_RD);
pos2b=eeprom_read(alamat+3, EEPROM2_WR,
EEPROM2_RD);
pos3a=eeprom_read(alamat+4, EEPROM2_WR,
EEPROM2_RD);
pos3b=eeprom_read(alamat+5, EEPROM2_WR,
EEPROM2_RD);
pos4a=eeprom_read(alamat+6, EEPROM2_WR,
EEPROM2_RD);
pos4b=eeprom_read(alamat+7, EEPROM2_WR,
EEPROM2_RD);
acc_1=eeprom_read(alamat+10, EEPROM2_WR,
EEPROM2_RD);
acc_2=eeprom_read(alamat+11, EEPROM2_WR,
EEPROM2_RD);
acc_3=eeprom_read(alamat+12, EEPROM2_WR,
EEPROM2_RD);
acc_4=eeprom_read(alamat+13, EEPROM2_WR,
EEPROM2_RD);

pos_1 = pos1a + (pos1b << 8);
pos_2 = pos2a + (pos2b << 8);
pos_3 = pos3a + (pos3b << 8);
pos_4 = pos4a + (pos4b << 8);
                            delay_ms(100);
                            goto m_5;
                        }

                        if(!UP)
                        {
                            delay_ms(250);
                            lcd_clear();
                            goto m_5;
                        }
                        goto load_menu;
                    }

                    goto m_5;


                    m_6:
                    lcd_gotoxy(0,0);
                    lcd_puts("DATA SETTER");
                    lcd_gotoxy(0,1);
                    lcd_puts("SAVE KE DRIVER");
                    if(!LEFT)
                    {
                        delay_ms(250);
                        lcd_clear();
                        goto m_5;
                    }
                    if(!DOWN)
                    {
                        delay_ms(250);
                        lcd_clear();
                        save_menu:
                        lcd_gotoxy(0,0);
                        lcd_puts("SAVE KE DRIVER");
                        lcd_gotoxy(0,1);
                        lcd_puts("SAVE?");
                        if(!RIGHT)
                        {
                            delay_ms(250);
```

```
                    lcd_clear();
                    lcd_gotoxy(0,0);
                    lcd_puts("SAVE
PARAMETER");
                    lcd_gotoxy(0,1);
                    lcd_puts("SAVING...");

                    alamat=0;

                    pos1a=pos_1 & 0xFF;
                    pos1b=(pos_1>>8) &
0xFF;
                    pos2a=pos_2 & 0xFF;
                    pos2b=(pos_2>>8) &
0xFF;
                    pos3a=pos_3 & 0xFF;
                    pos3b=(pos_3>>8) &
0xFF;
                    pos4a=pos_4 & 0xFF;
                    pos4b=(pos_4>>8) &
0xFF;


eeprom_write(alamat,pos1a,EEPROM2_WR);
eeprom_write(alamat+1,pos1b, EEPROM2_WR);
eeprom_write(alamat+2,pos2a, EEPROM2_WR);
eeprom_write(alamat+3,pos2b, EEPROM2_WR);
eeprom_write(alamat+4,pos3a, EEPROM2_WR);
eeprom_write(alamat+5,pos3b, EEPROM2_WR);
eeprom_write(alamat+6,pos4a, EEPROM2_WR);
eeprom_write(alamat+7,pos4b, EEPROM2_WR);
eeprom_write(alamat+10,acc_1, EEPROM2_WR);
eeprom_write(alamat+11,acc_2, EEPROM2_WR);
eeprom_write(alamat+12,acc_3, EEPROM2_WR);
eeprom_write(alamat+13,acc_4, EEPROM2_WR);
                    delay_ms(250);
                goto m_6;
                }

                if(!UP)
                {
                    delay_ms(250);
                    lcd_clear();
                    goto m_6;
                }
                goto save_menu;
            }

            goto m_6;

            m_7:
            lcd_gotoxy(0,0);
            lcd_puts("DATA SETTER");
            lcd_gotoxy(0,1);
            lcd_puts("MANUAL DRIVE");
            if(!LEFT)
            {
                delay_ms(250);
                lcd_clear();
                goto m_6;
            }
            if(!DOWN)
            {
                delay_ms(250);
                lcd_clear();
                drive_menu:
                lcd_gotoxy(0,0);
                lcd_puts("MANUAL DRIVE");

                if(!RIGHT)
                {
                    delay_ms(250);



                //goto m_6;
                }
                if(!LEFT)
                {
                    delay_ms(250);
                //goto m_6;
                }

                if(!UP)
                {
                    delay_ms(250);
                    lcd_clear();
                    goto m_7;
                }
                goto drive_menu;
            }

            goto m_7;

            //TAMBAH
        }

        if(!DOWN)
        {
            delay_ms(250);
            lcd_clear();
            pos1:
            lcd_gotoxy(0,0);
            lcd_puts("POS-00 (0-8000)");
            lcd_gotoxy(0,1);
            lcd_putint(pos_1);
            if(!RIGHT)
            {
                delay_ms(75);
                pos_1=pos_1+100;
            }
            if(!LEFT)
            {
                delay_ms(75);
                pos_1=pos_1-100;
```

```
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto m_1;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto pos2;
}
goto pos1;

pos2:
lcd_gotoxy(0,0);
lcd_puts("POS-01 (0-8000)");
lcd_gotoxy(0,1);
lcd_putint(pos_2);
if(!RIGHT)
{
    delay_ms(75);
    pos_2=pos_2+100;
}
if(!LEFT)
{
    delay_ms(75);
    pos_2=pos_2-100;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto pos1;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto pos3;
}
goto pos2;

pos3:
lcd_gotoxy(0,0);
lcd_puts("POS-10 (0-8000)");
lcd_gotoxy(0,1);
lcd_putint(pos_3);
if(!RIGHT)
{
    delay_ms(75);
    pos_3=pos_3+100;
}
if(!LEFT)
{
    delay_ms(75);
    pos_3=pos_3-100;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto pos2;
}
if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto pos4;
}
goto pos3;

pos4:
lcd_gotoxy(0,0);
lcd_puts("POS-11 (0-8000)");
lcd_gotoxy(0,1);
lcd_putint(pos_4);
if(!RIGHT)
{
    delay_ms(75);
    pos_4=pos_4+100;
}
if(!LEFT)
{
    delay_ms(75);
    pos_4=pos_4-100;
}
if(!UP)
{
    delay_ms(250);
    lcd_clear();
    goto pos3;
}

if(!DOWN)
{
    delay_ms(250);
    lcd_clear();
    goto m_1;
}

goto pos4;

}
goto m_1;

}
}
```

Lampiran 6 Program Mikrokontroler Pengatur *Driver* Motor Stepper

```
/************************************
This program was produced by the
CodeWizardAVR V2.05.0 Professional
Automatic Program Generator
© Copyright 1998-2010 Pavel Haiduc, HP
InfoTech s.r.l.
http://www.hpinfotech.com

Project : PERANCANGAN KONTROL GERAK
LEADSCREW MENGGUNAKAN PROGRAMMABLE
LOGIC CONTROLLER BERBASIS
MIKROKONTROLER
Version : FINAL
Date    : 1/5/2015
Author  : NAUFAL AWANDA PUTRA
Company : TEUB 2015
Comments:
Chip type               : ATmega1284P
Program type            : Application
AVR Core Clock frequency: 16.000000
MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 4096
************************************/
//========== HEADER =========//
#include <mega1284p.h>
#include <delay.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

//========== PIN ==========//

//INPUT PLC DAN SENSOR
#define STOP PIND.2         //INT0
#define FORWARD PIND.3      //INT1
#define REVERSE PINB.2      //INT2
#define START PINA.0        //PCINT0
#define LSP PINB.0          //PCINT1
#define LSM PINC.2          //PCINT2
#define HOME PIND.4         //PCINT3
#define SHOME PINA.7        //SENSOR
JARAK HOME
#define M0 PIND.7           //REGISTER
POSISI
#define M1 PINA.5

//SINYAL KELUAR UNTUK PLC
#define MOVE PORTC.7
#define READY PORTC.6
#define ALARM PORTC.5
//SINYAL KELUAR KE L297
#define CLOCK PORTB.4
#define DIR PORTB.3

//========== VARIABEL ==========//
//VARIABEL EEPROM
unsigned int alamat = 0;
unsigned int pos;
unsigned char acc;
unsigned int pos_1, pos_2, pos_3,
pos_4 = 0;
unsigned char acc_1, acc_2, acc_3,
acc_4 = 1;
unsigned char pos1a, pos1b, pos2a,
pos2b, pos3a, pos3b, pos4a, pos4b;
int pos_sebelum = 0;
int pos_sekarang;
int pos_sesudah;
unsigned int temp;

volatile int flagls=0;
volatile int flagstart = 0;
volatile int flaghome = 0;
volatile char state;

volatile bit kondisi=0;
volatile unsigned int x=0;
```

```
//========== I2C ==========//

#asm
   .equ __i2c_port=0x08 ;PORTC
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>

//======== EEPROM =========//

#define EEPROM2_WR 0xA8  //alamat ic
#define EEPROM2_RD 0xA9

unsigned char eeprom_read(unsigned int
address16)
{
    unsigned char data;
    unsigned char address_high;
    unsigned char address_low;
    address_high=(address16 &
0xFF00)>>8;
    address_low=address16 & 0xFF;
    i2c_start();
    i2c_write(0xA8);
    i2c_write(address_high);
    i2c_write(address_low);
    i2c_start();
    i2c_write(0xA9);
    data=i2c_read(0);
    i2c_stop();

return data;
}

// Timer 0 overflow interrupt service
routine
interrupt [TIM0_OVF] void
timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x83;
x++;
}

void tunda( unsigned int waktu)
{
    TIMSK0=0x01;
    while (x<=waktu)
        {
        }
    x=0;
    TIMSK0=0x00;
}
// Declare your global variables here

void inisialisasi(void)
{
    pos1a=eeprom_read(alamat);
    pos1b=eeprom_read(alamat+1);
    pos2a=eeprom_read(alamat+2);
    pos2b=eeprom_read(alamat+3);
    pos3a=eeprom_read(alamat+4);
    pos3b=eeprom_read(alamat+5);
    pos4a=eeprom_read(alamat+6);
    pos4b=eeprom_read(alamat+7);

    acc_1=eeprom_read(alamat+10);
    acc_2=eeprom_read(alamat+11);
    acc_3=eeprom_read(alamat+12);
    acc_4=eeprom_read(alamat+13);

    pos_1 = pos1a + (pos1b << 8);
    pos_2 = pos2a + (pos2b << 8);
    pos_3 = pos3a + (pos3b << 8);
    pos_4 = pos4a + (pos4b << 8);
}

void selektor_pos(void)
{
   switch ((PIND & 0xC0) >> 6)
    {
        case 0b00 :
```

```
        pos = pos_1;
        acc = acc_1;

            break;
        case 0b01 :
        pos = pos_2;
        acc = acc_2;

            break;
        case 0b10 :
        pos = pos_3;
        acc = acc_3;
            break;
        case 0b11 :
        pos = pos_4;
        acc = acc_4;
            break;
    }
}

/*
void selektor_pos(void)
{
   switch ((PINA & 0x30) >> 4)
    {
        case 0b00 :     M = 0; break;
        case 0b01 :     M = 1000;
break;
        case 0b10 :     M = 1500;
break;
        case 0b11 :     M = 2000;
break;
    }
}
*/

void gerak_fwd(void)
{
    DIR = 1;
    CLOCK=1;
    tunda(3);
    CLOCK=0;
    tunda(3);
}

void gerak_rev(void)
{
    DIR =0;
    CLOCK=1;
    tunda(3);
    CLOCK=0;
    tunda(3);
}

void baca3sensor(void)
{
flagls=0;
while (SHOME==0&&LSM==0)
{
    gerak_rev();
        if(state!='g')
        {
            break;
        }
}
while(SHOME==0&&LSP==0)
{
    gerak_fwd();
        if(state!='g')
        {
            break;
        }
}
if(SHOME==1)
    {
        kondisi=1;
    }
    else if (LSP==1)
    {
        kondisi=0;
    }
switch (kondisi)
{
    case 0:
```

```
        state = 'e';//ke kondisi ALARM
        flagls=1;
            break;
        case 1:
        state = 'f';//ke kondisi READY
        flagls=1;
            break;
        default:
        state = 'e';
        flagls=1;
            break;
    }
}

//========== PCINT ==========//

interrupt [EXT_INT0] void
ext_int0_isr(void)    //STOP
{
    state = 'a';
}
interrupt [EXT_INT1] void
ext_int1_isr(void)    //FWD
{
    state = 'b';
}
interrupt [EXT_INT2] void
ext_int2_isr(void)    //REV
{
    state = 'c';
}
interrupt [PC_INT0] void
pin_change_isr0(void)  //START
{
    if (flagstart ==1)
    {
        state = 'd';
    }
}
interrupt [PC_INT1] void
pin_change_isr1(void)  //LSP
{
    if (flagls==1)
    {
        state = 'a';
    }
}
interrupt [PC_INT2] void
pin_change_isr2(void)  //LSM
{
    if (flagls==1)
    {
        state = 'a';
    }
}
interrupt [PC_INT3] void
pin_change_isr3(void)  //HOME
{
    if (flagls==1 && flaghome ==0)
    {
        state = 'g';
    }
}

void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0b11111010;

PORTC=0x00;
DDRC=0xF0;

PORTD=0x00;
DDRD=0x00;

//==== INISIALISASI TIMER 1 MS ====//
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 250.000 kHz
```

```
// Mode: Normal top=0xFF
// OC0A output: Disconnected
// OC0B output: Disconnected
TCCR0A=0x00;
TCCR0B=0x03;
TCNT0=0x83;
OCR0A=0x00;
OCR0B=0x00;

//=============== INISIALISASI
EXTERNAL INTERRUPT DAN PCINTERRUPT
===============//
// External Interrupt(s)
initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: On
// INT1 Mode: Falling Edge
// INT2: On
// INT2 Mode: Falling Edge
// Interrupt on any change on pins
PCINT0-7: On
// Interrupt on any change on pins
PCINT8-15: On
// Interrupt on any change on pins
PCINT16-23: On
// Interrupt on any change on pins
PCINT24-31: On
EICRA=0x2A;
EIMSK=0x07;
EIFR=0x07;
PCMSK0=0x01;
PCMSK1=0x01;
PCMSK2=0x04;
PCMSK3=0x10;
PCICR=0x0F;
PCIFR=0x0F;


// Timer/Counter 0 Interrupt(s)
initialization
TIMSK0=0x01;

// I2C Bus initialization
i2c_init();

#asm("sei")

inisialisasi();
state = 'g';

while (1)
      {

       switch (state)
       {
                case 'a':
//STATE STOP; START HARUS KE HOME
                MOVE    = 0;
                READY   = 0;
                ALARM   = 1;
                flagstart = 0;
                    break;

                case 'b':
//STATE FWD; START HARUS KE HOME
                MOVE    = 1;
                READY   = 0;
                ALARM   = 0;
                flaghome = 0;
                flagstart = 0;
                gerak_fwd();
                    break;

                case 'c':
//STATE REV; START HARUS KE HOME
                MOVE    = 1;
                READY   = 0;
                ALARM   = 0;
                flaghome = 0;
                flagstart = 0;
                gerak_rev();
                    break;
```

```
            case 'd':
//STATE START; FLAGSTART HARUS = 1
            MOVE    = 1;
            READY   = 0;
            ALARM   = 0;
            flaghome = 1;
            selektor_pos();
//AMBIL DATA PADA REGISTER
            temp = pos;
            pos_sekarang = temp -
pos_sebelum;
            pos_sesudah=0;

            if (pos_sekarang < 0)
            {
                pos_sekarang =
abs(pos_sekarang);
                DIR = 0;
            }

            else if (pos_sekarang > 0)
            {
                DIR = 1;
            }

            else if ( pos_sekarang ==
0)
                                    {
                                        pos_sesudah =
                                pos_sekarang;
                                    }

                                     for ( ; pos_sesudah<=
                                pos_sekarang ; pos_sesudah++)
                                        {
                                            CLOCK = 1;
                                            tunda(acc);
                                            CLOCK = 0;
                                            tunda(acc);
                                            if(state!='d')
                                            {
                                                break;
                                            }
                                        }
                                    pos_sebelum = temp;
                                    state = 'f';
                                //KEMBALI KE STATE READY
                                        break;

                                    case 'e' :
                                //STATE ALARM; START HARUS KE HOME
                                    MOVE    = 0;
                                    READY   = 0;
                                    ALARM   = 1;
                                                        flagstart = 0;
                                                            break;

                                                        case 'f' :
                                                //STATE READY; DO NOTHING
                                                        MOVE    = 0;
                                                        READY   = 1;
                                                        ALARM   = 0;
                                                        flagstart = 1;
                                                        flaghome = 1;
                                                            break;

                                                        case 'g' :
                                                //STATE HOME; KEMBALI KE HOME
                                                        MOVE    = 1;
                                                        READY   = 0;
                                                        ALARM   = 0;
                                                        flagstart = 0;
                                                        baca3sensor();
                                                            break;
                                                        default: state='a';
                                                            break;
                                                }
                                                }}
```

Lampiran 7 Program PLC

Lampiran 8 *Datasheet* Mikrokontroler

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 128K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 4K Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 16K Bytes Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - Two 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel, 10-bit ADC
    Differential mode with selectable gain at 1x, 10x or 200x
  - Byte-oriented Two-wire Serial Interface
  - Two Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 1.8 - 5.5V for ATmega1284P
- Speed Grades
  - 0 - 4 MHz @ 1.8 - 5.5V
  - 0 - 10 MHz @ 2.7 - 5.5V
  - 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active: 0.4 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.7 µA (Including 32 kHz RTC)

8-bit AVR® Microcontroller with 128K Bytes In-System Programmable Flash

ATmega1284P

Preliminary

8059D–AVR–11/09

1. Pin Configurations

Figure 1-1. Pinout ATmega1284P

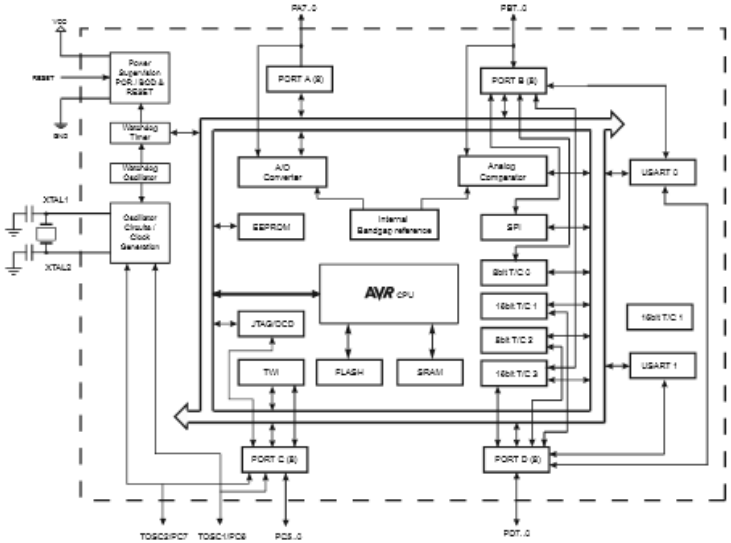
Note: The large center pad underneath the QFN/MLF package should be soldered to ground on the board to ensure good mechanical stability.

2. Overview

The ATmega1284P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega1284P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega1284P provides the following features: 128K bytes of In-System Programmable Flash with Read-While-Write capabilities, 4K bytes EEPROM, 16K bytes SRAM, 32 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 2 USARTs, a byte oriented 2-wire Serial Interface, a 8-channel, 10-bit ADC with optional differential input stage with programmable gain, programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the Crystal/Resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega1284P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega1284P AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

2.2 Pin Descriptions

2.2.1 VCC

Digital supply voltage.

2.2.2 GND

Ground.

2.2.3 Port A (PA7:PA0)

Port A serves as analog inputs to the Analog-to-digital Converter.

Port A also serves as an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port A pins that are externally pulled low will source current if the pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port A also serves the functions of various special features of the ATmega1284P as listed on page 78.

2.2.4 Port B (PB7:PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega1284P as listed on page 80.

2.2.5 Port C (PC7:PC0)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port C also serves the functions of the JTAG interface, along with special features of the ATmega1284P as listed on page 83.

2.2.6 Port D (PD7:PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega1284P as listed on page 86.

2.2.7 $\overline{\text{RESET}}$

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in "System and Reset Characteristics" on page 327. Shorter pulses are not guaranteed to generate a reset.

2.2.8 XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

2.2.9 XTAL2

Output from the inverting Oscillator amplifier.

2.2.10 AVCC

AVCC is the supply voltage pin for Port F and the Analog-to-digital Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.

2.2.11 AREF

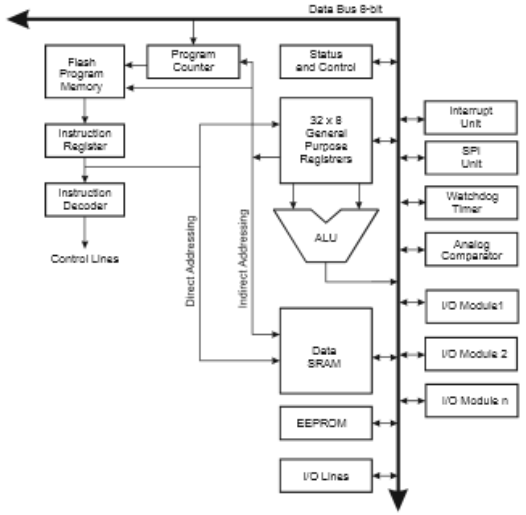
This is the analog reference pin for the Analog-to-digital Converter.

5. AVR CPU Core

5.1 Overview

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

Figure 5-1. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typ-

interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence..

Assembly Code Example

```
in r16, SREG     ; store SREG value
cli    ; disable interrupts during timed sequence
sbi EECR, EEMPE  ; start EEPROM write
sbi EECR, EEPE
out SREG, r16    ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
__disable_interrupt();
EECR |= (1<<EEMPE); /* start EEPROM write */
EECR |= (1<<EEPE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

10. Interrupts

10.1 Overview

This section describes the specifics of the interrupt handling as performed in ATmega1284P. For a general explanation of the AVR interrupt handling, refer to "Reset and Interrupt Handling" on page 13.

10.2 Interrupt Vectors in ATmega1284P

Table 10-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	PCINT0	Pin Change Interrupt Request 0
6	$000A	PCINT1	Pin Change Interrupt Request 1
7	$000C	PCINT2	Pin Change Interrupt Request 2
8	$000E	PCINT3	Pin Change Interrupt Request 3
9	$0010	WDT	Watchdog Time-out Interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 Compare Match A
11	$0014	TIMER2_COMPB	Timer/Counter2 Compare Match B
12	$0016	TIMER2_OVF	Timer/Counter2 Overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 Capture Event
14	$001A	TIMER1_COMPA	Timer/Counter1 Compare Match A
15	$001C	TIMER1_COMPB	Timer/Counter1 Compare Match B
16	$001E	TIMER1_OVF	Timer/Counter1 Overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 Compare Match A
18	$0022	TIMER0_COMPB	Timer/Counter0 Compare match B
19	$0024	TIMER0_OVF	Timer/Counter0 Overflow
20	$0026	SPI_STC	SPI Serial Transfer Complete
21	$0028	USART0_RX	USART0 Rx Complete
22	$002A	USART0_UDRE	USART0 Data Register Empty
23	$002C	USART0_TX	USART0 Tx Complete
24	$002E	ANALOG_COMP	Analog Comparator
25	$0030	ADC	ADC Conversion Complete
26	$0032	EE_READY	EEPROM Ready
27	$0034	TWI	2-wire Serial Interface

11. External Interrupts

11.1 Overview

The External Interrupts are triggered by the INT2:0 pin or any of the PCINT31:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT2:0 or PCINT31:0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

The Pin change interrupt PCI3 will trigger if any enabled PCINT31:24 pin toggle, Pin change interrupt PCI2 will trigger if any enabled PCINT23:16 pin toggles, Pin change interrupt PCI1 if any enabled PCINT15:8 toggles and Pin change interrupts PCI0 will trigger if any enabled PCINT7:0 pin toggles. PCMSK3, PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT31:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Registers – EICRA (INT2:0). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Low level interrupts and the edge interrupt on INT2:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from Power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the Start-up Time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL Fuses as described in "System Clock and Clock Options" on page 27.

11.2 Register Description

11.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – Reserved**

These bits are reserved in the ATmega1284P, and will always read as zero.

- **Bits 5:0 – ISC21, ISC20 – ISC00, ISC00: External Interrupt 2 - 0 Sense Control Bits**

The External Interrupts 2 - 0 are activated by the external pins INT2:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 11-1. Edges on INT2..INT0 are registered asynchronously. Pulses on INT2:0 pins wider than the minimum pulse width given in "External Interrupts Characteristics" on page 327 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.

Table 11-1. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Note: 1. n = 3, 2, 1or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

11.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	INT2	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 2:0 – INT2:0: External Interrupt Request 2 - 0 Enable**

When an INT2:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Register, EICRA, defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

11.2.3 EIFR –External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	INTF2	INTF1	INTF0	EIFR
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 2:0 – INTF2:0: External Interrupt Flags 2 - 0**

When an edge or logic change on the INT2:0 pin triggers an interrupt request, INTF2:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT2:0 in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT2:0 are configured as level interrupt. Note that when entering sleep mode with the INT2:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF2:0 flags. See "Digital Input Enable and Sleep Modes" on page 74 for more information.

11.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – PCIE3: Pin Change Interrupt Enable 3**

When the PCIE3 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 3 is enabled. Any change on any enabled PCINT31..24 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI3 Interrupt Vector. PCINT31..24 pins are enabled individually by the PCMSK3 Register.

- **Bit 2 – PCIE2: Pin Change Interrupt Enable 2**

When the PCIE2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT23..16 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI2 Interrupt Vector. PCINT23..16 pins are enabled individually by the PCMSK2 Register.

- **Bit 1 – PCIE1: Pin Change Interrupt Enable 1**

When the PCIE1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT15..8 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI1 Interrupt Vector. PCINT15..8 pins are enabled individually by the PCMSK1 Register.

- **Bit 0 – PCIE0: Pin Change Interrupt Enable 0**

When the PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI0 Interrupt Vector. PCINT7..0 pins are enabled individually by the PCMSK0 Register.

11.2.5 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)			–	–	PCIF3	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3– PCIF3: Pin Change Interrupt Flag 3**

When a logic change on any PCINT31..24 pin triggers an interrupt request, PCIF3 becomes set (one). If the I-bit in SREG and the PCIE3 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 2 – PCIF2: Pin Change Interrupt Flag 2**

When a logic change on any PCINT23..16 pin triggers an interrupt request, PCIF2 becomes set (one). If the I-bit in SREG and the PCIE2 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 1 – PCIF1: Pin Change Interrupt Flag 1**

When a logic change on any PCINT15..8 pin triggers an interrupt request, PCIF1 becomes set (one). If the I-bit in SREG and the PCIE1 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 0 – PCIF0: Pin Change Interrupt Flag 0**

When a logic change on any PCINT7..0 pin triggers an interrupt request, PCIF0 becomes set (one). If the I-bit in SREG and the PCIE0 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

11.2.6 PCMSK3 – Pin Change Mask Register 3

Bit	7	6	5	4	3	2	1	0	
(0x73)	PCINT31	PCINT30	PCINT29	PCINT28	PCINT27	PCINT26	PCINT25	PCINT24	PCMSK3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT31:24: Pin Change Enable Mask 31:24**

Each PCINT31:24-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT31:24 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT31..24 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

11.2.7 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT23:16: Pin Change Enable Mask 23..16**

Each PCINT23:16-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT23:16 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

11.2.8 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT15:8: Pin Change Enable Mask 15..8**

Each PCINT15:8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 is set and the PCIE1 bit in EIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

11.2.9 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x68)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – PCINT7:0: Pin Change Enable Mask 7..0**

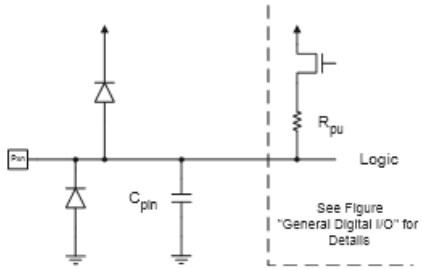
Each PCINT7:0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

12. I/O-Ports

12.1 Overview

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and Ground as indicated in Figure 12-1. Refer to "Electrical Characteristics" on page 323 for a complete list of parameters.

Figure 12-1. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case "x" represents the numbering letter for the port, and a lower case "n" represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in "Register Description" on page 90.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. However, writing a logic one to a bit in the PINx Register, will result in a toggle in the corresponding bit in the Data Register. In addition, the Pull-up Disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

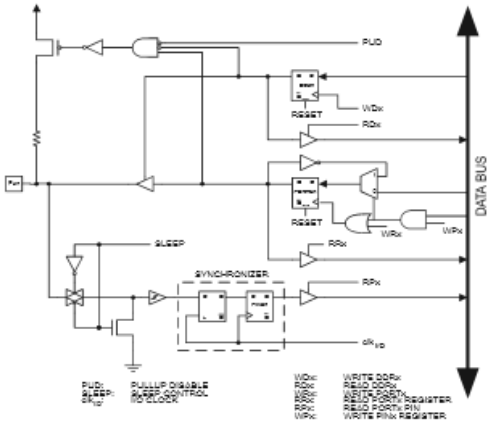
Using the I/O port as General Digital I/O is described in "Ports as General Digital I/O" on page 71. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in "Alternate Port Functions" on page 76. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

12.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 12-2 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 12-2. General Digital I/O(1)



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

12.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description" on page 90, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

12.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

12.2.3 Switching Between Input and Output

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

Table 12-1 summarizes the control signals for the pin value.

Table 12-1. Port Pin Configurations

DDxn	PORTxn	PUD (In MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

12.2.4 Reading the Pin Value

Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit. As shown in Figure 12-2, the PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 12-3 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

13. 8-bit Timer/Counter0 with PWM

13.1 Features

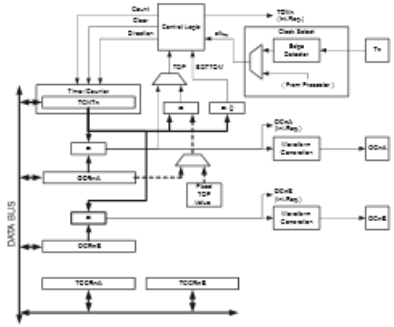
- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

13.2 Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 13-1. For the actual placement of I/O pins, see "Pin Configurations" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "Register Description" on page 103.

Figure 13-1. 8-bit Timer/Counter Block Diagram



13.2.1 Registers

The Timer/Counter (TCNT0) and Output Compare Registers (OCR0A and OCR0B) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T0}).

The double buffered Output Compare Registers (OCR0A and OCR0B) are compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pins (OC0A and OC0B). See Section "13.5" on page 94. for details. The Compare Match event will also set the Compare Flag (OCF0A or OCF0B) which can be used to generate an Output Compare interrupt request.

13.2.2 Definitions

Many register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, in this case 0. A lower case "x" replaces the Output Compare Unit, in this case Compare Unit A or Compare Unit B. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 13-1 are also used extensively throughout the document.

Table 13-1. Definitions

BOTTOM	The counter reaches the BOTTOM when it becomes 0x00.
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255).
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A Register. The assignment is dependent on the mode of operation.

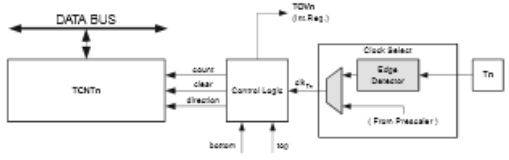
13.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0B). For details on clock sources and prescaler, see "Timer/Counter Prescaler" on page 153.

13.4 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 13-2 shows a block diagram of the counter and its surroundings.

Figure 13-2. Counter Unit Block Diagram



Signal description (internal signals):

count	Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{T0}). clk_{T0} can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether clk_{T0} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC0A and OC0B. For more details about advanced counting sequences and waveform generation, see "Modes of Operation" on page 97.

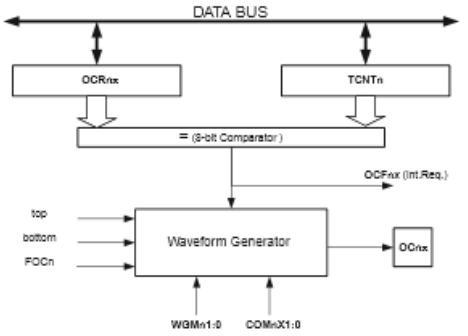
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM02:0 bits. TOV0 can be used for generating a CPU interrupt.

13.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the Output Compare Flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt. The Output Compare Flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM02:0 bits and Compare Output mode (COM0x1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation ("Modes of Operation" on page 97).

Figure 13-3 shows a block diagram of the Output Compare unit.

Figure 13-3. Output Compare Unit, Block Diagram



The OCR0x Registers are double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0x Compare Registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0x Buffer Register, and if double buffering is disabled the CPU will access the OCR0x directly.

13.5.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC0x) bit. Forcing Compare Match will not set the OCF0x Flag or reload/clear the timer, but the OC0x pin will be updated as if a real Compare Match had occurred (the COM0x1:0 bits settings define whether the OC0x pin is set, cleared or toggled).

13.5.2 Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 Register will block any Compare Match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0x to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

13.5.3 Using the Output Compare Unit

Since writing TCNT0 in any mode of operation will block all Compare Matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the Output Compare Unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0x value, the Compare Match will be missed, resulting in incorrect waveform

generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is down-counting.

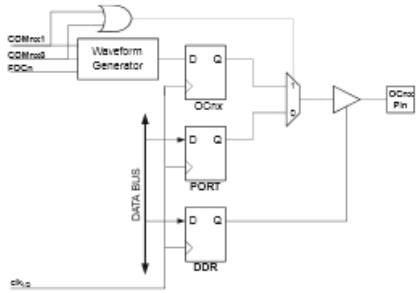
The setup of the OC0x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0x value is to use the Force Output Compare (FOC0x) strobe bits in Normal mode. The OC0x Registers keep their values even when changing between Waveform Generation modes.

Be aware that the COM0x1:0 bits are not double buffered together with the compare value. Changing the COM0x1:0 bits will take effect immediately.

13.6 Compare Match Output Unit

The Compare Output mode (COM0x1:0) bits have two functions. The Waveform Generator uses the COM0x1:0 bits for defining the Output Compare (OC0x) state at the next Compare Match. Also, the COM0x1:0 bits control the OC0x pin output source. Figure 13-4 shows a simplified schematic of the logic affected by the COM0x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM0x1:0 bits are shown. When referring to the OC0x state, the reference is for the internal OC0x Register, not the OC0x pin. If a system reset occur, the OC0x Register is reset to "0".

Figure 13-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0x) from the Waveform Generator if either of the COM0x1:0 bits are set. However, the OC0x pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0x pin (DDR_OC0x) must be set as output before the OC0x value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0x state before the output is enabled. Note that some COM0x1:0 bit settings are reserved for certain modes of operation. See Section "13.9" on page 103.

13.6.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0x1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0x1:0 = 0 tells the Waveform Generator that no action on the OC0x Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 13-2 on page 103. For fast PWM mode, refer to Table 13-3 on page 103, and for phase correct PWM refer to Table 13-4 on page 104.

A change of the COM0x1:0 bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC0x strobe bits.

13.7 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM02:0) and Compare Output mode (COM0x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x1:0 bits control whether the output should be set, cleared, or toggled at a Compare Match (See Section "14.8" on page 121.).

For detailed timing information see "Timer/Counter Timing Diagrams" on page 101.

13.7.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM02:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

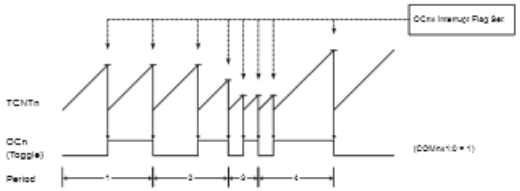
The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

13.7.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM02:0 = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 13-5. The counter value (TCNT0) increases until a Compare Match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

Figure 13-5. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR0A is lower than the current value of TCNT0, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the Compare Match can occur.

For generating a waveform output in CTC mode, the OC0A output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM0A1:0 = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk\_I/O}/2$ when OCR0A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

The *N* variable represents the prescale factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

13.7.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM2:0 = 3, and OCR0A when WGM2:0 = 7. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the Compare Match between TCNT0 and OCR0x, and set at BOTTOM. In inverting Compare Output mode, the output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast

17. USART

17.1 Features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

17.2 USART1 and USART0

The ATmega1284P has two USART's, USART0 and USART1.

The functionality for all USART's is described below, most register and bit references in this section are written in general form. A lower case "n" replaces the USART number.

USART0 and USART1 have different I/O registers as shown in "Register Summary" on page 360.

17.3 Overview

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device.

A simplified block diagram of the USART Transmitter is shown in Figure 17-1 on page 172. CPU accessible I/O Registers and I/O pins are shown in bold.

The Power Reducion USART0 bit, PRUSART0, in "PRR0 – Power Reduction Register 0" on page 46 must be disabled by writing a logical zero to it.

The Power Reducion USART1 bit, PRUSART1, in "PRR0 – Power Reduction Register 0" on page 46 must be disabled by writing a logical zero to it.

19. 2-wire Serial Interface

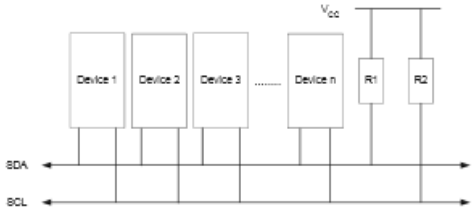
19.1 Features

- Simple Yet Powerful and Flexible Communication Interface, only two Bus Lines Needed
- Both Master and Slave Operation Supported
- Device can Operate as Transmitter or Receiver
- 7-bit Address Space Allows up to 128 Different Slave Addresses
- Multi-master Arbitration Support
- Up to 400 kHz Data Transfer Speed
- Slew-rate Limited Output Drivers
- Noise Suppression Circuitry Rejects Spikes on Bus Lines
- Fully Programmable Slave Address with General Call Support
- Address Recognition Causes Wake-up When AVR is in Sleep Mode

19.2 2-wire Serial Interface Bus Definition

The 2-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 19-1. TWI Bus Interconnection



19.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 19-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

The Power Reduction TWI bit, PRTWI bit in "PRR0 – Power Reduction Register 0" on page 46 must be written to zero to enable the 2-wire Serial Interface.

19.2.2 Electrical Interconnection

As depicted in Figure 19-1, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices trim-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

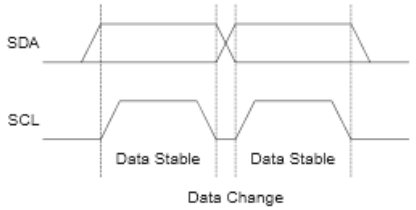
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in "SPI Timing Characteristics" on page 328. Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

19.3 Data Transfer and Frame Format

19.3.1 Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

Figure 19-2. Data Validity

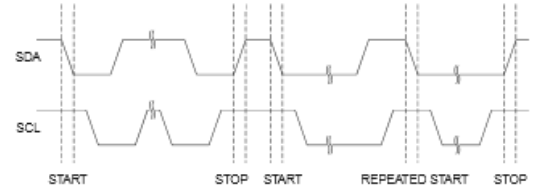


19.3.2 START and STOP Conditions

The Master initiates and terminates a data transmission. The transmission is initiated when the Master issues a START condition on the bus, and it is terminated when the Master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As

depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

Figure 19-3. START, REPEATED START and STOP conditions



19.3.3 Address Packet Format

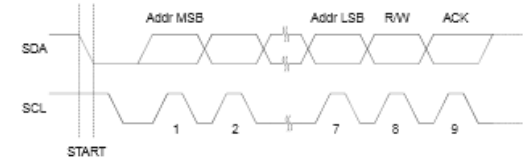
All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed Slave is busy, or for some other reason can not service the Master's request, the SDA line should be left high in the ACK clock cycle. The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a Master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

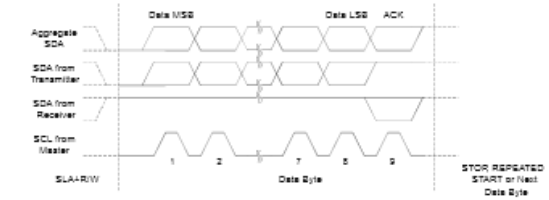
Figure 19-4. Address Packet Format



19.3.4 Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the Master generates the clock and the START and STOP conditions, while the Receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signalled. When the Receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the Transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 19-5. Data Packet Format

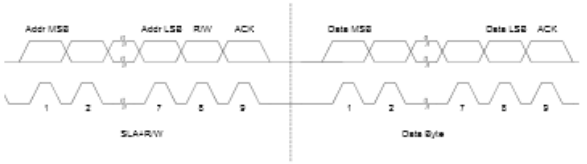


19.3.5 Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement handshaking between the Master and the Slave. The Slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the Master is too fast for the Slave, or the Slave needs extra time for processing between the data transmissions. The Slave extending the SCL low period will not affect the SCL high period, which is determined by the Master. As a consequence, the Slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 19-6 on page 210 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 19-6. Typical Data Transmission



Lampiran 9 *Datasheet* MOSFET

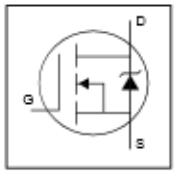
International
IOR Rectifier

PD - 91341B

IRF540N

HEXFET® Power MOSFET

- Advanced Process Technology
- Ultra Low On-Resistance
- Dynamic dv/dt Rating
- 175°C Operating Temperature
- Fast Switching
- Fully Avalanche Rated



V_{DSS} = 100V

$R_{DS(on)}$ = 44mΩ

I_D = 33A

TO-220AB

Description

Advanced HEXFET® Power MOSFETs from International Rectifier utilize advanced processing techniques to achieve extremely low on-resistance per silicon area. This benefit, combined with the fast switching speed and ruggedized device design that HEXFET power MOSFETs are well known for, provides the designer with an extremely efficient and reliable device for use in a wide variety of applications.

The TO-220 package is universally preferred for all commercial-industrial applications at power dissipation levels to approximately 50 watts. The low thermal resistance and low package cost of the TO-220 contribute to its wide acceptance throughout the industry.

Absolute Maximum Ratings

	Parameter	Max.	Units
I_D @ T_C = 25°C	Continuous Drain Current, V_{GS} @ 10V	33	A
I_D @ T_C = 100°C	Continuous Drain Current, V_{GS} @ 10V	23	
I_{DM}	Pulsed Drain Current ①	110	
P_D @T_C = 25°C	Power Dissipation	130	W
	Linear Derating Factor	0.87	W/°C
V_{GS}	Gate-to-Source Voltage	± 20	V
I_{AR}	Avalanche Current①	16	A
E_{AR}	Repetitive Avalanche Energy①	13	mJ
dv/dt	Peak Diode Recovery dv/dt ③	7.0	V/ns
T_J T_{STG}	Operating Junction and Storage Temperature Range	-55 to + 175	°C
	Soldering Temperature, for 10 seconds	300 (1.6mm from case)	
	Mounting torque, 6-32 or M3 srew	10 lbf•in (1.1N•m)	

Thermal Resistance

	Parameter	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	—	1.15	°C/W
$R_{\theta CS}$	Case-to-Sink, Flat, Greased Surface	0.50	—	
$R_{\theta JA}$	Junction-to-Ambient	—	62	

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	100	—	—	V	V_{GS} = 0V, I_D = 250µA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.12	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	—	44	mΩ	V_{GS} = 10V, I_D = 16A ④
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	V_{DS} = V_{GS}, I_D = 250µA
g_{fs}	Forward Transconductance	21	—	—	S	V_{DS} = 50V, I_D = 16A④
I_{DSS}	Drain-to-Source Leakage Current	—	—	25	µA	V_{DS} = 100V, V_{GS} = 0V
		—	—	250		V_{DS} = 80V, V_{GS} = 0V, T_J = 150°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	100	nA	V_{GS} = 20V
	Gate-to-Source Reverse Leakage	—	—	-100		V_{GS} = -20V
Q_g	Total Gate Charge	—	—	71	nC	I_D = 16A
Q_{gs}	Gate-to-Source Charge	—	—	14		V_{DS} = 80V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	—	21		V_{GS} = 10V, See Fig. 6 and 13
$t_{d(on)}$	Turn-On Delay Time	—	11	—	ns	V_{DD} = 50V
t_r	Rise Time	—	35	—		I_D = 16A
$t_{d(off)}$	Turn-Off Delay Time	—	39	—		R_G = 5.1Ω
t_f	Fall Time	—	35	—		V_{GS} = 10V, See Fig. 10 ④
L_D	Internal Drain Inductance	—	4.5	—	nH	Between lead, 6mm (0.25in.) from package and center of die contact
L_S	Internal Source Inductance	—	7.5	—		
C_{iss}	Input Capacitance	—	1960	—	pF	V_{GS} = 0V
C_{oss}	Output Capacitance	—	250	—		V_{DS} = 25V
C_{rss}	Reverse Transfer Capacitance	—	40	—		f = 1.0MHz, See Fig. 5
E_{AS}	Single Pulse Avalanche Energy②	—	700⑤	185⑥	mJ	I_{AS} = 16A, L = 1.5mH

Source-Drain Ratings and Characteristics

	Parameter	Min.	Typ.	Max.	Units	Conditions
I_S	Continuous Source Current (Body Diode)	—	—	33	A	MOSFET symbol showing the integral reverse p-n junction diode.
I_{SM}	Pulsed Source Current (Body Diode)①	—	—	110		
V_{SD}	Diode Forward Voltage	—	—	1.2	V	T_J = 25°C, I_S = 16A, V_{GS} = 0V ④
t_{rr}	Reverse Recovery Time	—	115	170	ns	T_J = 25°C, I_F = 16A
Q_{rr}	Reverse Recovery Charge	—	505	760	nC	di/dt = 100A/µs ④
t_{on}	Forward Turn-On Time	Intrinsic turn-on time is negligible (turn-on is dominated by L_S+L_D)				

Notes:

① Repetitive rating; pulse width limited by max. junction temperature. (See fig. 11)

② Starting T_J = 25°C, L = 1.5mH
R_G = 25Ω, I_{AS} = 16A. (See Figure 12)

③ I_{SD} ≤ 16A, di/dt ≤ 340A/µs, V_{DD} ≤ $V_{(BR)DSS}$,
T_J ≤ 175°C

④ Pulse width ≤ 400µs; duty cycle ≤ 2%.

⑤ This is a typical value at device destruction and represents operation outside rated limits.

⑥ This is a calculated value limited to T_J = 175°C .

Lampiran 10 *Datasheet* IC *Driver* Motor Stepper

L297

STEPPER MOTOR CONTROLLERS

- NORMAL/WAVE DRIVE
- HALF/FULL STEP MODES
- CLOCKWISE/ANTICLOCKWISE DIRECTION
- SWITCHMODE LOAD CURRENT REGULATION
- PROGRAMMABLE LOAD CURRENT
- FEW EXTERNAL COMPONENTS
- RESET INPUT & HOME OUTPUT
- ENABLE INPUT

DIP20

SO20

ORDERING NUMBERS : L297/1 (DIP20)
L297D (SO20)

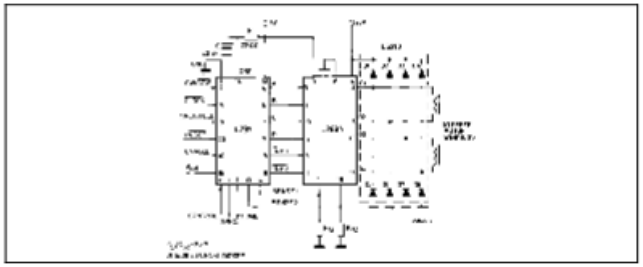
DESCRIPTION

The L297 Stepper Motor Controller IC generates four phase drive signals for two phase bipolar and four phase unipolar step motors in microcomputer-controlled applications. The motor can be driven in half step, normal and wave drive modes and on-chip PWM chopper circuits permit switch-mode control of the current in the windings. A feature of this device is that it requires only clock, direction and mode input signals. Since the phase are generated internally the burden on the microprocessor, and the programmer, is greatly reduced. Mounted in DIP20 and SO20 packages, the L297 can be used with monolithic bridge drives such as the L298N or L293E, or with discrete transistors and darlingtons.

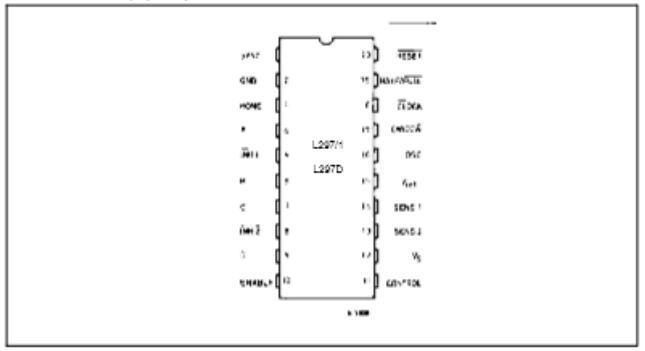
ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_s	Supply voltage	10	V
V_i	Input signals	7	V
P_{tot}	Total power dissipation (T_{amb} = 70°C)	1	W
T_{stg}, T_j	Storage and junction temperature	-40 to + 150	°C

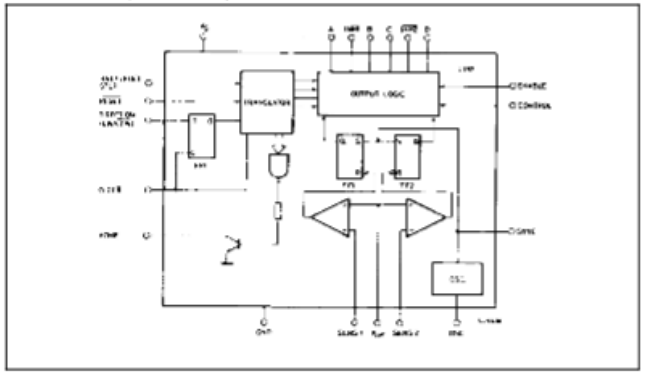
TWO PHASE BIPOLAR STEPPER MOTOR CONTROL CIRCUIT



PIN CONNECTION (Top view)



BLOCK DIAGRAM (L297/1 - L297D)



PIN FUNCTIONS - L297/1 - L297D

N°	NAME	FUNCTION
1	SYNC	Output of the on-chip chopper oscillator. The SYNC connections The SYNC connections of all L297s to be synchronized are connected together and the oscillator components are omitted on all but one. If an external clock source is used it is injected at this terminal.
2	GND	Ground connection.
3	HOME	Open collector output that indicates when the L297 is in its initial state (ABCD = 0101). The transistor is open when this signal is active.
4	A	Motor phase A drive signal for power stage.
5	$\overline{INH1}$	Active low inhibit control for driver stage of A and B phases. When a bipolar bridge is used this signal can be used to ensure fast decay of load current when a winding is de-energized. Also used by chopper to regulate load current if CONTROL input is low.
6	B	Motor phase B drive signal for power stage.
7	C	Motor phase C drive signal for power stage.
8	$\overline{INH2}$	Active low inhibit control for drive stages of C and D phases. Same functions as INH1.
9	D	Motor phase D drive signal for power stage.
10	ENABLE	Chip enable input. When low (inactive) INH1, INH2, A, B, C and D are brought low.
11	CONTROL	Control input that defines action of chopper. When low chopper acts on INH1 and INH2; when high chopper acts on phase lines ABCD.
12	V_s	5V supply input.
13	$SENS_2$	Input for load current sense voltage from power stages of phases C and D.
14	$SENS_1$	Input for load current sense voltage from power stages of phases A and B.
15	V_{ref}	Reference voltage for chopper circuit. A voltage applied to this pin determines the peak load current.
16	OSC	An RC network (R to V_{CC}, C to ground) connected to this terminal determines the chopper rate. This terminal is connected to ground on all but one device in synchronized multi - L297 configurations. f ≅ 1/0.69 RC
17	CW/$\overline{CCW}$	Clockwise/counterclockwise direction control input. Physical direction of motor rotation also depends on connection of windings. Synchronized internally therefore direction can be changed at any time.
18	$\overline{CLOCK}$	Step clock. An active low pulse on this input advances the motor one increment. The step occurs on the rising edge of this signal.

PIN FUNCTIONS - L297/1 - L297D (continued)

N°	NAME	FUNCTION
19	HALF/$\overline{FULL}$	Half/full step select input. When high selects half step operation, when low selects full step operation. One-phase-on full step mode is obtained by selecting FULL when the L297's translator is at an even-numbered state. Two-phase-on full step mode is set by selecting FULL when the translator is at an odd numbered position. (The home position is designate state 1).
20	$\overline{RESET}$	Reset input. An active low pulse on this input restores the translator to the home position (state 1, ABCD = 0101).

THERMAL DATA

Symbol	Parameter		DIP20	SO20	Unit
$R_{th\text{-}j\text{-}amb}$	Thermal resistance junction-ambient	max	80	100	°C/W

CIRCUIT OPERATION

The L297 is intended for use with a dual bridge driver, quad darlington array or discrete power devices in step motor driving applications. It receives step clock, direction and mode signals from the systems controller (usually a microcomputer chip) and generates control signals for the power stage.

The principal functions are a translator, which generates the motor phase sequences, and a dual PWM chopper circuit which regulates the current in the motor windings. The translator generates three different sequences, selected by the HALF/FULL input. These are normal (two phases energised), wave drive (one phase energised) and half-step (alternately one phase energised/two phases energised). Two inhibit signals are also generated by the L297 in half step and wave drive modes. These signals, which connect directly to the L298's enable inputs, are intended to speed current decay when a winding is de-energised. When the L297 is used to drive a unipolar motor the chopper acts on these lines.

An input called CONTROL determines whether the chopper will act on the phase lines ABCD or the inhibit lines INH1 and INH2. When the phase lines are chopped the non-active phase line of each pair (AB or CD) is activated (rather than interrupting the line then active). In L297 + L298 configurations this technique reduces dissipation in the load current sense resistors.

A common on-chip oscillator drives the dual chopper. It supplies pulses at the chopper rate which set the two flip-flops FF1 and FF2. When the current in a winding reaches the programmed peak value the voltage across the sense resistor (connected to one of the sense inputs $SENS_1$ or $SENS_2$) equals V_{ref} and the corresponding comparator resets its flip flop, interrupting the drive current until the next oscillator pulse arrives. The peak current for both windings is programmed by a voltage divider on the V_{ref} input.

Ground noise problems in multiple configurations can be avoided by synchronising the chopper oscillators. This is done by connecting all the SYNC pins together, mounting the oscillator RC network on one device only and grounding the OSC pin on all other devices.

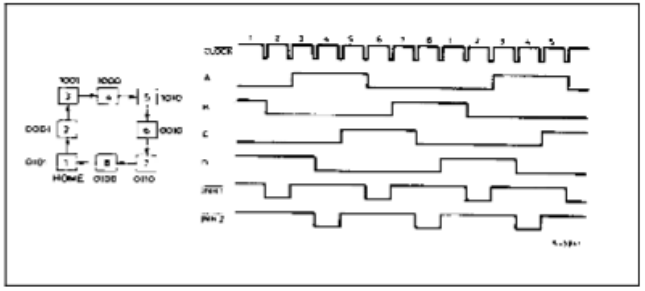
MOTOR DRIVING PHASE SEQUENCES

The L297's translator generates phase sequences for normal drive, wave drive and half step modes. The state sequences and output waveforms for these three modes are shown below. In all cases the translator advances on the low to high transition of CLOCK.

Clockwise rotation is indicate; for anticlockwise rotation the sequences are simply reversed RESET restores the translator to state 1, where ABCD = 0101.

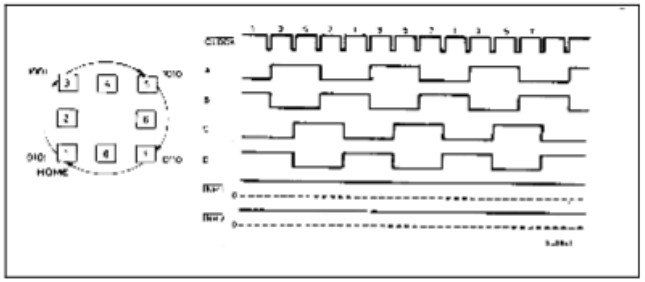
HALF STEP MODE

Half step mode is selected by a high level on the HALF/FULL input.



NORMAL DRIVE MODE

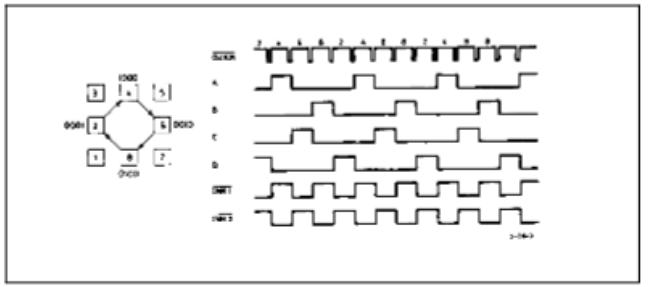
Normal drive mode (also called "two-phase-on" drive) is selected by a low level on the HALF/FULL input when the translator is at an odd numbered state (1, 3, 5 or 7). In this mode the INH1 and INH2 outputs remain high throughout.



MOTOR DRIVING PHASE SEQUENCES (continued)

WAVE DRIVE MODE

Wave drive mode (also called "one-phase-on" drive) is selected by a low level on the HALF/FULL input when the translator is at an even numbered state (2, 4, 6 or 8).



ELECTRICAL CHARACTERISTICS (Refer to the block diagram T_{amb} = 25°C, V_s = 5V unless otherwise specified)

Symbol	Parameter	Test conditions		Min.	Typ	Max.	Unit
V_s	Supply voltage (pin 12)			4.75		7	V
I_s	Quiescent supply current (pin 12)	Outputs floating			50	80	mA
V_i	Input voltage (pin 11, 17, 18, 19, 20)		Low			0.6	V
			High	2		V_s	V
I_i	Input current (pin 11, 17, 18, 19, 20)		V_i = L		100		µA
			V_i = H			10	µA
V_{en}	Enable input voltage (pin 10)		Low			1.3	V
			High	2		V_s	V
I_{en}	Enable input current (pin 10)		V_{en} = L			100	µA
			V_{en} = H			10	µA
V_o	Phase output voltage (pins 4, 6, 7, 9)	I_o = 10mA	V_{OL}			0.4	V
		I_o = 5mA	V_{OH}	3.9			V
V_{inh}	Inhibit output voltage (pins 5, 8)	I_o = 10mA	$V_{inh\ L}$			0.4	V
		I_o = 5mA	$V_{inh\ H}$	3.9			V
V_{SYNC}	Sync Output Voltage	I_o = 5mA	$V_{SYNC\ H}$	3.3			V
		I_o = 5mA	$V_{SYNC\ V}$			0.8	

Lampiran 11 *Datasheet* Motor Stepper

ORIGINAL ITEM **PH268-21B**

Obsolete - Review Options Below; $112

PH268-21B **SPECIFICATIONS**

http://store.yahoo.com/yhst-47568356416153/ph268-21b.html

Motor		Voltage	Current per Phase		Holding Torque		Resistance per Phase (Winding)	Inductance per Phase (Winding)
Single Shaft	Double Shaft	V. DC	A/Phase		OZ-in	N-cm	ohm/Phase (Winding)	mH/Phase (Winding)
PH266-01	PH266-01B	6	1.2		83.3	58.8	5	6
PH266-02	PH266-02B	12	0.6		83.3	58.8	20	32
PH266-03	PH266-03B	24	0.3		83.3	58.8	80	128
PH266-E1.2	PH266-E1.2B	6	1.2*1	1.7*2	83.3	58.8	(5)	(8)
PH268-21	PH268-21B	5.4	1.5		125	88.2	3.6	6
PH268-22	PH268-22B	12	0.66		125	88.2	17.7	30
PH268-23	PH268-23B	24	0.34		125	88.2	70.6	120
PH268-E2.3	PH268-E2.3B	3.9	2.3*1	3.3*2	125	88.2	(1.7)	(3)
PH268-E1.6	PH268-E1.6B	5.1	1.6*1	2.3*2	125	88.2	(3.2)	(5)

*1 Unipolar *2 Bipolar parallel

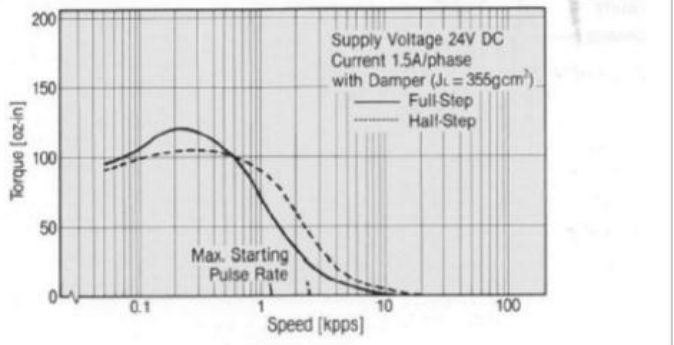
ORIGINAL ITEM **PH268-21B**

Obsolete - Review Options Below; $112

PH268-21B **SPEED-TORQUE CURVE**

http://store.yahoo.com/yhst-47568356416153/ph268-21b.html

● **PH268-21B** * Measured by unipolar constant current chopper drive



Lampiran 12 *Datasheet* IC EEPROM

MICROCHIP

24AA64/24LC64

64K I²C™ Serial EEPROM

Device Selection Table

Part Number	Vcc Range	Max Clock Frequency	Temp Ranges
24AA64	1.8-5.5	400 kHz(1)	I
24LC64	2.5-5.5	400 kHz	I, E

Note 1: 100 kHz for Vcc <2.5V

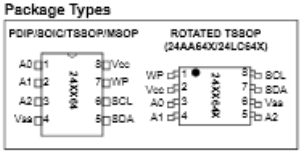
Features

- Single supply with operation down to 1.8V
- Low power CMOS technology
  - 1 mA active current typical
  - 1 µA standby current (max.) (I-temp)
- Organized as 8 blocks of 8K bit (64K bit)
- 2-wire serial interface bus, I²C™ compatible
- Cascadable for up to eight devices
- Schmitt Trigger inputs for noise suppression
- Output slope control to eliminate ground bounce
- 100 kHz (24AA64) and 400 kHz (24LC64) compatibility
- Self-timed write cycle (including auto-erase)
- Page-write buffer for up to 32 bytes
- 2 ms typical write cycle time for page-write
- Hardware write protect for entire memory
- Can be operated as a serial ROM
- Factory programming (QTP) available
- ESD protection > 4,000V
- 1,000,000 erase/write cycles
- Data retention > 200 years
- 8-lead PDIP, SOIC, TSSOP, and MSOP package
- Available temperature ranges:
  - Industrial (I): -40°C to +85°C
  - Automotive (E): -40°C to +125°C

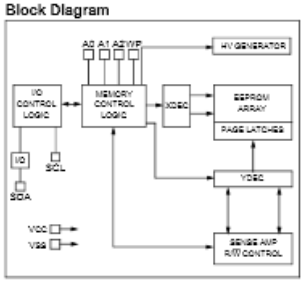
Description

The Microchip Technology Inc. 24AA64/24LC64 (24XX64*) is a 64 Kbit Electrically Erasable PROM. The device is organized as eight blocks of 1K x 8-bit memory with a 2-wire serial interface. Low voltage design permits operation down to 1.8V with standby and active currents of only 1 µA and 1 mA respectively. It has been developed for advanced, low power applications such as personal communications or data acquisition. The 24XX64 also has a page-write capability for up to 32 bytes of data. Functional address lines allow up to eight devices on the same bus, for up to 512 Kbits address space. The 24XX64 is available in the standard 8-pin PDIP, surface mount SOIC, TSSOP and MSOP packages.

Package Types



Block Diagram



*24XX64 is used in this document as a generic part number for the 24AA64/24LC64 devices.

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings†

Vcc 6.5V
All inputs and outputs w.r.t. Vss ..... -0.3V to Vcc +1.0V
Storage temperature ..... -65°C to +150°C
Ambient temp. with power applied ..... -40°C to +125°C
ESD protection on all pins ..... ≥ 4 kV

† NOTICE: Stresses above those listed under "Maximum ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

1.1 DC Characteristics

DC CHARACTERISTICS			Vcc = +1.8V to +5.5V Industrial (I): TAMB = -40°C to +85°C Automotive (E): TAMB = -40°C to +125°C				
Param. No.	Sym	Characteristic	Min	Typ	Max	Units	Conditions
D1	VIH	**WP, SCL and SDA pins**	—	—	—	—	—
D2	—	High level input voltage	0.7 Vcc	—	—	V	—
D3	VIL	Low level input voltage	—	—	0.3 Vcc	V	—
D4	VHYS	Hysteresis of Schmitt Trigger inputs	0.05 Vcc	—	—	V	**(Note 1)**
D5	VOL	Low level output voltage	—	—	0.40	V	IOL = 3.0 mA, Vcc = 2.5V
D6	ILI	**Input leakage current**	—	—	±10	µA	VIN =.1V to Vcc
D7	ILO	**Output leakage current**	—	—	±10	µA	VOUT =.1V to Vcc
D8	CIN, COUT	Pin capacitance (all inputs/outputs)	—	—	10	pF	Vcc = 5.0V **(Note 1)** TAMB = 25°C, FCLK = 1 MHz
D9	ICC write	**Operating current**	—	0.1	3	mA	Vcc = 5.5V, SCL = 400 kHz
D10	ICC read		—	0.05	1	mA	—
D11	ICCS	**Standby current**	— —	.01 —	1 5	µA µA	Industrial Automotive SDA = SCL = Vcc WP = Vss

Note 1: This parameter is periodically sampled and not 100% tested.
2: Typical measurements taken at room temperature.

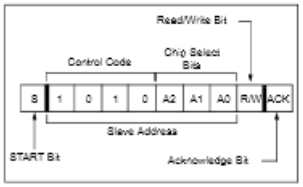
3.6 Device Addressing

A control byte is the first byte received following the START condition from the master device (Figure 3-2). The control byte consists of a four bit control code; for the 24XX64 this is set as 1010 binary for read and write operations. The next three bits of the control byte are the chip select bits (A2, A1, A0). The chip select bits allow the use of up to eight 24XX64 devices on the same bus and are used to select which device is accessed. The chip select bits in the control byte must correspond to the logic levels on the corresponding A2, A1, and A0 pins for the device to respond. These bits are in effect the three Most Significant bits of the word address.

The last bit of the control byte defines the operation to be performed. When set to a one a read operation is selected, and when set to a zero a write operation is selected. The next two bytes received define the address of the first data byte (Figure 3-3). Because only A12...A0 are used, the upper three address bits are don't care bits. The upper address bits are transferred first, followed by the less significant bits.

Following the START condition, the 24XX64 monitors the SDA bus checking the device type identifier being transmitted. Upon receiving a 1010 code and appropriate device select bits, the slave device outputs an Acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24XX64 will select a read or write operation.

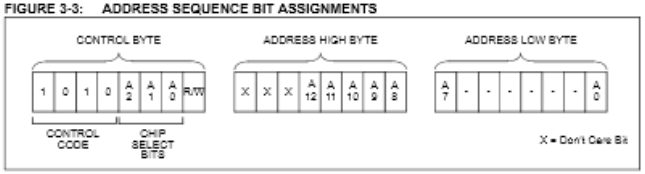
FIGURE 3-2: CONTROL BYTE FORMAT



3.7 Contiguous Addressing Across Multiple Devices

The chip select bits A2, A1, A0 can be used to expand the contiguous address space for up to 512K bits by adding up to eight 24XX64's on the same bus. In this case, software can use A0 of the control byte as address bit A13, A1 as address bit A14, and A2 as address bit A15. It is not possible to sequentially read across device boundaries.

FIGURE 3-3: ADDRESS SEQUENCE BIT ASSIGNMENTS



4.0 WRITE OPERATIONS

4.1 Byte Write

Following the START condition from the master, the control code (four bits), the chip select (three bits), and the R/W bit (which is a logic low) are clocked onto the bus by the master transmitter. This indicates to the addressed slave receiver that the address high byte will follow after it has generated an Acknowledge bit during the ninth clock cycle. Therefore, the next byte transmitted by the master is the high-order byte of the word address and will be written into the address pointer of the 24XX64. The next byte is the Least Significant Address Byte. After receiving another Acknowledge signal from the 24XX64 the master device will transmit the data word to be written into the addressed memory location. The 24XX64 acknowledges again and the master generates a STOP condition. This initiates the internal write cycle, and during this time the 24XX64 will not generate Acknowledge signals (Figure 4-1). If an attempt is made to write to the array with the WP pin held high, the device will acknowledge the command but no write cycle will occur, no data will be written and the device will immediately accept a new command. After a byte write command, the internal address counter will point to the address location following the one that was just written.

4.2 Page Write

The write control byte, word address and the first data byte are transmitted to the 24XX64 in the same way as in a byte write. But instead of generating a STOP condition, the master transmits up to 31 additional bytes which are temporarily stored in the on-chip page buffer and will be written into memory after the master has transmitted a STOP condition. After receipt of each word, the five lower address pointer bits are internally incremented by one. If the master should transmit more than 32 bytes prior to generating the STOP condition, the address counter will roll over and the previously received data will be overwritten. As with the byte write operation, once the STOP condition is received, an internal write cycle will begin (Figure 4-2). If an attempt is made to write to the array with the WP pin held high, the device will acknowledge the command but no write cycle will occur, no data will be written and the device will immediately accept a new command.

Note: Page write operations are limited to writing bytes within a single physical page, **regardless** of the number of bytes actually being written. Physical page boundaries start at addresses that are integer multiples of the page buffer size (or 'page size') and end at addresses that are integer multiples of [page size - 1]. If a page write command attempts to write across a physical page boundary, the result is that the data wraps around to the beginning of the current page (overwriting data previously stored there), instead of being written to the next page as might be expected. It is therefore necessary for the application software to prevent page write operations that would attempt to cross a page boundary.

4.3 Write Protection

The WP pin allows the user to write protect the entire array (0000-1FFF) when the pin is tied to Vcc. If tied to Vss or left floating, the write protection is disabled. The WP pin is sampled at the STOP bit for every write command (Figure 3-1) Toggling the WP pin after the STOP bit will have no effect on the execution of the write cycle.

FIGURE 4-1: BYTE WRITE

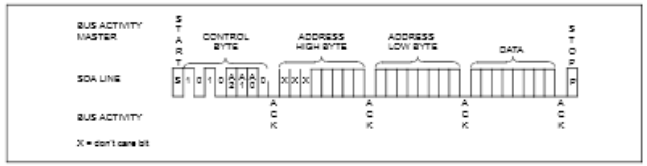
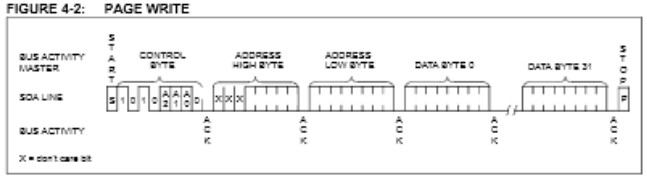


FIGURE 4-2: PAGE WRITE



6.0 READ OPERATION

Read operations are initiated in the same way as write operations with the exception that the R/$\overline{W}$ bit of the control byte is set to one. There are three basic types of read operations: current address read, random read, and sequential read.

6.1 Current Address Read

The 24XX64 contains an address counter that maintains the address of the last word accessed, internally incremented by one. Therefore, if the previous read access was to address n (n is any legal address), the next current address read operation would access data from address n + 1.

Upon receipt of the control byte with R/$\overline{W}$ bit set to one, the 24XX64 issues an acknowledge and transmits the eight bit data word. The master will not acknowledge the transfer but does generate a STOP condition and the 24XX64 discontinues transmission (Figure 6-1).

6.2 Random Read

Random read operations allow the master to access any memory location in a random manner. To perform this type of read operation, first the word address must be set. This is done by sending the word address to the 24XX64 as part of a write operation (R/$\overline{W}$ bit set to 0). After the word address is sent, the master generates a START condition following the acknowledge. This terminates the write operation, but not before the internal address pointer is set. Then the master issues the control byte again but with the R/$\overline{W}$ bit set to a one. The 24XX64 will then issue an acknowledge and transmit the 8-bit data word. The master will not acknowledge the transfer but does generate a STOP condition which causes the 24XX64 to discontinue transmission (Figure 6-2). After a random read command, the internal address counter will point to the address location following the one that was just read.

6.3 Sequential Read

Sequential reads are initiated in the same way as a random read except that after the 24XX64 transmits the first data byte, the master issues an acknowledge as opposed to the STOP condition used in a random read. This acknowledge directs the 24XX64 to transmit the next sequentially addressed 8-bit word (Figure 6-3). Following the final byte transmitted to the master, the master will NOT generate an acknowledge but will generate a STOP condition. To provide sequential reads the 24XX64 contains an internal address pointer which is incremented by one at the completion of each operation. This address pointer allows the entire memory contents to be serially read during one operation. The internal address pointer will automatically roll over from address 1FFF to address 0000 if the master acknowledges the byte received from the array address 1FFF.

FIGURE 6-1: CURRENT ADDRESS READ

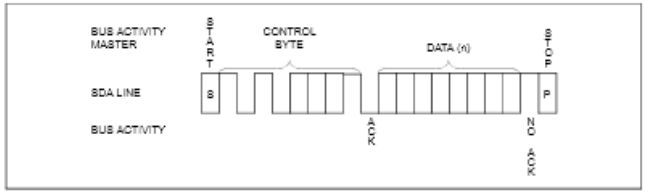


FIGURE 6-2: RANDOM READ

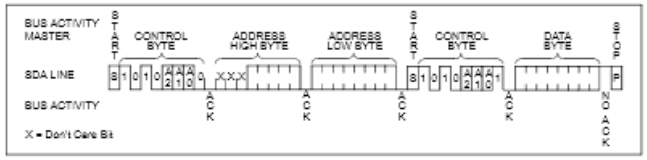
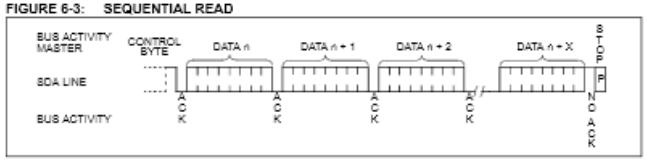


FIGURE 6-3: SEQUENTIAL READ



7.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 7-1.

TABLE 7-1: PIN FUNCTION TABLE

Name	PDIP	SOIC	TSSOP	MSOP	ROTATED TSSOP	Description
A0	1	1	1	1	3	Chip Address Input
A1	2	2	2	2	4	Chip Address Input
A2	3	3	3	3	5	Chip Address Input
Vss	4	4	4	4	6	Ground
SDA	5	5	5	5	7	Serial Address/Data I/O
SCL	6	6	6	6	8	Serial Clock
WP	7	7	7	7	1	Write Protect Input
Vcc	8	8	8	8	2	+1.8V to 5.5V Power Supply

7.1 A0, A1, A2 Chip Address Inputs

The A0, A1, A2 inputs are used by the 24XX64 for multiple device operation. The levels on these inputs are compared with the corresponding bits in the slave address. The chip is selected if the compare is true.

Up to eight devices may be connected to the same bus by using different chip select bit combinations. These inputs must be connected to either Vcc or Vss.

7.2 Serial Data (SDA)

This is a bi-directional pin used to transfer addresses and data into and data out of the device. It is an open-drain terminal, therefore, the SDA bus requires a pull-up resistor to Vcc (typical 10 kΩ for 100 kHz, 2 kΩ for 400 kHz)

For normal data transfer SDA is allowed to change only during SCL low. Changes during SCL high are reserved for indicating the START and STOP conditions.

7.3 Serial Clock (SCL)

This input is used to synchronize the data transfer from and to the device.

7.4 Write Protect (WP)

This pin can be connected to either Vss, Vcc or left floating. An internal pull-down resistor on this pin will keep the device in the unprotected state if left floating. If tied to Vss or left floating, normal memory operation is enabled (read/write the entire memory 0000-1FFF).

If tied to Vcc, WRITE operations are inhibited. Read operations are not affected.