SISTEM KENDALI KECEPATAN MOTOR DC BERBASIS PWM (*Pulse Width Modulation*)

DC motor speed Control System based PWM (Pulse Width Modulation)

BAHARUDDIN
P2700209031

TEKNIK ELEKTRO
PROGRAM PASCASARJANA
UNIVERSITAS HASANUDDIN
MAKASSAR
2012

LEMBAR PENGESAHAN
UJIAN AKHIR MAGISTER

Judul : Sistem Kendali Kecepatan Motor DC Berbasis PWM
(Pulse Width Modulation)

Nama : Baharuddin

Nomor Pokok : P2700209031

Konsentrasi : Teknik Konmputer, Kendali dan Elektronika

Mengatahui

Komisi Penasehat

Makassar, 16 , Juli 2012

Ketua Penasehat	Anggota Penasehat
Dr. Ir. H. Rhiza S. Sadjad,MSEE	Prof. Dr. Ir. H. Muhammad Tola, M. Eng.

Mengatahui

Ketua Program Studi

Prof. Dr. Ir. H. Salama Manjang,M.T

Nip. 19621231 199003 1 024

PERNYATAAN KEASLIAN TESIS

Yang bertanda tangan di bawah ini :

Nama : Baharuddin

Nomor Mahasiswa : P2700209031

Program Studi : Teknik Elektro

Menyatakan dengan sebenarnya bahwa tesis yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan mewrupakan pengambilalihan tulisan atau pemikiran orang lain. Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebahagian atau keseluruhan tesis ini hasil karya orang lain, saya bersedia menerima sangsi atas pebuatan tersebut.

Makassar, 16 Juli 2012

Yang menyatakan

Baharuddin

KATA PENGANTAR

Alhamdulillah, puji dan syukur penulis panjatkan ke hadirat Allah SWT atas rahmat dan ksrunia-Nya sehingga penulis dapat menyelesaikan penelitian ini.

Ide yang melatar belakangi penelitian ini adalah adanya berbagai penelitian sebelumya yang terkait dengan sistem kecepatan dc yang masih menggunakan mikroprosesser yang dihubungkan ke alat yang dikendalikan melalui programmable peripherial interface 8255 yang memiliki tiga port yang mampu dialamati . Untuk mengembanghkan sistem pengendalian kecepatan motor tersebut maka penulis mengangkat suatu penelitian yang berjudul " Sistem Kendali Kecepatan Motor DC Berbasis PWM " .

Banyak kendala yang dihadapi penulis dalam rangka penyusunan tesis ini, namun berkat bantuan dari berbagai pihak , maka tesis ini akhirnya dapat terselesaikan. Penulis dengan tulus menyampaikan ucapan terima kasih kepada :

1. Dr. Ir. H. Rhiza S.Sadja,MSEE, selaku Ketua Komisi Penasehat.
2. Prof.Dr.Ir. H.Muhammad Tola, sebagai Anggota Komisi Penasehat atas bantuan dan bimbingan yang telah diberikan mulai dari awal penyusunan proposal hingga selesainya penelitian ini.
3. Prof.Dr.Ir.H. Muhammad Arif,Dipl.Ing, Prof.Dr.Ir. H.Najamuddin,MS, Porf.Dr.Ir. H.Salama Manjang,MT selaku Dosen Penguji yang telah memberikan saran dalam perbaikan tesis ini.

4. Terima kasih kepada istriku serta saudara-saudaraku yang senantiasa memberikan motivasi serta mendo'akan penulis sehingga dapat menyelesaikan penelitian ini.
5. Terima kasih juga penulis sampaikan kepada teman-teman Pasca Sarjana di Fakultas Teknik Elektro Universitas Hasanuddin, khususnya kepada teman-teman Program Studi Teknik Komputer, Kendali dan Elektronika angkatan 2009 yang telah banyak membantu penulis dalam menyelesaikan penelitian ini.

Makassar, 16 Juli 2012

Penulis

Baharuddin

ABSTRAK

BAHARUDDIN, Sistem Kendali Kecepatan Motor DC Berbasis PWM (Pulse Width Modulation) (dibimbing oleh Rhiza S.Sadjad dan Muhammad Tola).

Penelitian ini bertujuan meningkatkan kinewrja motor DC dalam menjaga kecepatan agar tetap pada set point ketika terjadi perubahan beban serta kecepatan motor DC ketika diberi tegangan melalui PWM (Pulse Width Modulation).

Metode yang digunakan adalah methode mode phase corrent yaitu nilai registrasi counter CTNTn yang mencacah naik dan turun secara terus-menerus akan selalu dibandingkan dengan register OCRn. Hasil perbandingan register TCNTn dengan OCRn digunakan untuk membangkitkan sinyal PWM yang dikeluarkan melalui sebuah pin OCn, dan metode pre skalar yang merupakan faktor pengali clock (scala clock) sehingga frekuensi register TCNT dapat diatur .

Hasil penelitian menunjukkan bahwa dengan menggunakan teknologi mikrokontroller khususnya pengendalian dengan PWM dapat meningkatkan efesiensi daya karena tidak terbuang ke transistor , transistor bekerja dengan mode on atau off yang diatur periodenya secara PWM, ketika sinyal dalam kondisi high motor diberi tegangan dan kondisi low tegangan 0 diberikan tetapi motor tetap bergerak.

Kata kunci : Kendali kecepatan motor DC, mikrokontroller ATMega 85354, PWM.

ABSTRACT

BAHARUDDIN, DC Motor Speed Control System with PWM (Pulse Width Modulation) (Supevised by Rhiza S.Sadjad and Muhammad Tola).

The rersearch aims to inprove the performance of DC motors in maintaining speed to keep the set point when there are changes of load and DC motor speed when they are given voltage thtought PWM (Pulse Width Modulation).

The study used the corrent phase mode method in which the counter register value TCNTn chopping up and down was continuously be compared to the comparator register OCRn. The results of the comparison were used to generate PWM signal released through a pin OCn. Another method was the pre scalar method as the clock multiflier factor (clock scala) so that the frequency of TCNT register can be set.

The results reveal that the use of microcontroller technology, especially the control the control with PWM can inprove the efficiency of power becouse the power is not wasted to the transistor. The transistor works with mode on or off and the period is regulated with PWM. When signal in the motor was given voltage condition of low voltage 0 was given, the motor still moved.

Keywords : DC motor speed control, microcontroller ATMega 8535. PWM.

DAFTAR ISI

Halaman

BAB III METODOLOGI PENELITIAN

DAFTAR GAMBAR

Nomor **Halaman**

DAFTAR TABEL

Nomor **Halaman**

DAFTAR LAMPIRAN

Nomor

BAB I

PENDAHULUAN

I.1 Latar belakang

Dewasa ini, kemajuan dunia industri tidak lepas dari peranan kendali automatik baik dalam mengendalikan gerak, kecepatan, tekanan, suhu, kelembaban, viskositas, dan besaran-besaran lainnya. Kendali automatik ini diperlukan dalam operasi mesin, penanganan dan perakitan bagian-bagian mekanik dalam industri manufaktur.

Kendali automatik memberikan kemudahan dalam:

- mendapatkan performansi dari sistem dinamik
- mempertinggi kualitas dan laju produksi
- mengurangi biaya produksi
- meniadakan pekerjaan rutin yang berulang yang membosankan untuk dilakukan manusia.

Sebagai contoh perusahaan yang bergerak dalam bidang industri tekstil , kecepatan dan ketepatan menghasilkan suatu barang yang banyak dengan pengeluaran biaya yang sedikit merupakan ukuran suatu perusahaan yang berhasil.

PWM (Pulse Width Modulation) adalah kebutuhan standar di dunia industri manufaktur. Perangkat teknologi informasi dan komunikasi yang membangun PWM berfungsi sebagai pembangkit sinyal keluaran yang periodenya berulang antara *high* dan *low* . PWM adalah bagian utama dari sistem-sistem teknologi kendali proses *(Process Control Technology)* di dunia industri manufaktur.

Dalam penelitian ini akan dibangun suatu sistem yang akan membangkitkan catudaya motor DC. Yang akan diteliti adalah pengendalian kecepatan motor DC. Motor akan dikendalikan oleh PWM yang merupakan fitur dari mikrokontroler ATMega 8535.

Selain permasalahan kualitas hasil pengendalian, permasalahan energi juga tidak kalah penting, terlebih lagi disaat krisis energi seperti pada saat ini. Kemampuan menjaga kecepatan agar tetap sesuai dengan *set point* ketika terjadi perubahan beban maupun kemampuan sistem untuk mengejar kecepatan agar mencapai *set point* ketika motor mulai berjalan menjadi faktor yang sangat penting sebagai ukuran kinerja pengendali kecepatan motor, oleh karena itu penulis bertujuan untuk meningkatkan kinerja motor tersebut.

Berdasarkan dengan permasalahan yang ada maka diperlukan suatu penelitian khusus terhadap kinerja kecepatan motor DC , sehingga akan diperoleh suatu pengetahuan terhadap proses yang tarjadi dalam sistem kendali kecepatan motor DC berbasis PWM.

I.2 Rumusan masalah

Beberapa permasalahan yang dihadapi adalah :

1. Bagaimana meningkatkan kinerja motor DC dalam menjaga kecepatan agar tetap pada set point ketika terjadi perubahan beban serta kecepatan sistem dalam mengejar set point.
2. Bagaimana pengaturan kecepatan motor DC ketika diberi tegangan melalui PWM (Pulse Width Modulation).

I.3 Tujuan penelitian

1. Meningkatkan kinerja motor DC dalam menjaga kecepatan agar tetap pada set point ketika terjadi perubahan beban serta kecepatan sistem dalam mengejar set point.
2. Mengatur kecepatan motor DC ketika diberi tegangan melalui PWM (Pulse Width Modulation).

I.4. Manfaat penelitian

1. Mengetahui sistem kendali kecepatan motor DC berbasis PWM.
2. Mengetahui kinerja motor Dc dalam menjaga kecepatan agar tetap set point ketika terjadi gangguan serta kecepatan sistem dalam mengejar set point.
3. Mengetahui pengaturan kecepatan motor DC ketika diberi tegangan melalui PWM.

I.5. Batasan Penelitian

Batasan penelitian ini meliputi pemanfaatan fitur PWM pada mikrokontroller ATMega8535 dengan masukan berupa *set point* sedangkan motor yang digunakan berupa motor DC. Masukan nilai *set point* melalui PC/laptop yang nilainya dikonversikan ke besaran *digital* melalui mikrokontroller pengaturan
keluaran tegangan memanfaatkan fitur PWM pada Atmega 8535 yang kemudian
dialirkan ke driver motor dc.

I.6. Sistimatika Penulisan

Sistimatika prnulisan proposal ini di bagi menjadi 4 bab :

I. Bab I. Pendahuluan

Berisi tentang latar belakang, rumusan masalah, tujuan penelitian dan batasan penelitian.

II. Bab II. Tinjauan Pustaka

Berisi tentang tepri-teori sistem kendali, sensor kecepatan,PWM, motor DC dan mikrokontroller .

III. Bab III. Metode Penelitian

Berisi tentang rancangan penelitian, waktu dan lokasi penelitian, bahan dan alat penelitian, Definisi operasional dan teknik analisis.

IV. Bab IV. Hasil Penelitian Dan Pembahasan

Berisi tentang pengujian dan analisis hasil rancangan

V. Bab V. Kesimpulan dan Saran

Berisi kesimpulan dan saran – saran

VI. Daftar Pustaka

VII. Lampiran

BAB II
TINJAUAN PUSTAKA

II.1. Dasar- dasar sistem kendali

Sejarah perkembangan sistem kendali diawali dengan revulusi industri I dan II sedangkan perang Dunia I dan II mempercepat perkembangannya. James Watts menemukan generator sintrifugal (1767) ; Minovzky menemukan pengendalian kapal laut (1922); metode analisis kepekaan di temukan oleh Nyquist (1932); Hazen pada tahun 1934 menemukan sistem kendali posisi; metode analisis fungsi frekuensi masukan-tunggal keluaran tunggal *(single* masukan *single* keluaran = SISO) dikembangkan oleh Bode pada tahun 1940; sedangkan W.R. Evans pada tahun 1950 mengembangkan metode analisis tempat kedudukan akar (*root locus*).

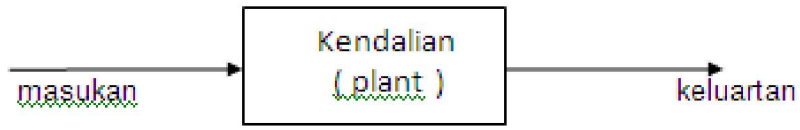
Sistem kendali modern diawali pada tahun 1960, yaitu untuk masukan jamak keluaran jamak (*multi* masukan *multi* keluartan = MIMO), yang berdasarkan pada teorema vector matriks. Disusul dengan perkembangannya sistem kendali optimal (yang memenuhi kriteria-kriteria tambahan) sistem stokastik (tak tentu, yang berdasarkan pada teorima probalitas), sistem kendali adaktif (yang mempunyai kemampuan untuk beradaptasi), sistem kendali yang mampu untuk belajar (*learning control system)* yang merupakan metode dalam intelegensia buatan; sampai dengan sistem kendali komputer *(computer control system*).

II.1.1.Defisi dan pengertian sistem kendali

Sistem kendali adalah hubungan antara komponen yang membentuk konfigurasi sistem yang akan menghasilkan tanggapan sistem yang diharapkan. (Killian, (1997)

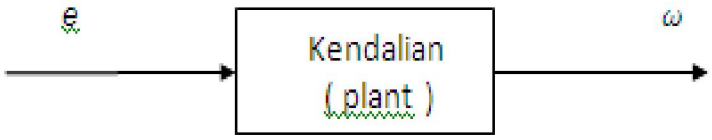
Modern Control Technology, Componen and Systems,2nd edition, Prentice Hall – International, Inc.).

Diagram blok sistem kendali secara umum digambarkan sebagai berikut:



Gambar II.1. Diagram blok sistem kendali secara umum

Masukan dan keluaran merupakan variable atau besaran fisik. Keluaran merupakan hal yang dihasilkan oleh kendalian (yang dikendalikan) sedangkan masukan adalah yang mempengaruhi kendalian (keluaran). Kedua dimensi masukan dan keluaran tidak harus sama. Sebagai contoh sistem pengendali kecepatan motor DC berikut ini :



Gambar II.2. Diagram blok sistem kendali kecepatan pada motor DC

Masukan e berupa tegangan sementara keluarannya berupa kecepatan sudut ω sehingga pengaturan kecepatan sudut diatur dengan pengaturan pemberian tegangan yang diberikan ke motor.

II.1.2. Sistem kendali *loop* terbuka dan *loop* tertutup

Sistem kendali *loop* terbuka (*loop* terbuka sistem) adalah sistem kendali yang tidak terdapat elemen yang mengamati keluaran yang terjadi untuk dibandingkan dengan masukannya (yang diinginkan) meskipun menggunakan sebuah pengandali (*controller)* untuk memperoleh tanggapan yang diinginkan. Sebagai contoh pada

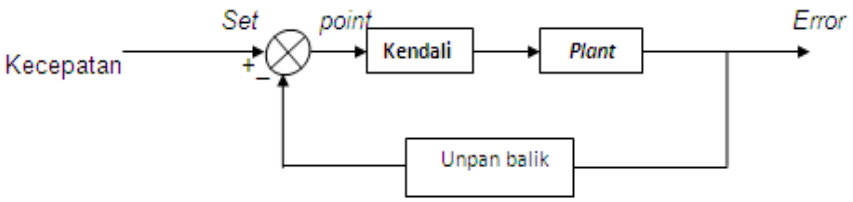
pengendalian kecepatan putaran motor DC dengan *loop* terbuka digambarkan sebagai berikut :

Gambar II.3. Diagram sistem kendali kecepatan secara *loop* terbuka

:

Adapun sistem kendali *loop* tertutup atau disebut juga siatem kendali umpan balik (feedback control system) adalah sistem kendali yang mempunyai elemen umpan balik yang berfungsi untuk mengamati keluaran yang terjadi untuk dibandingkan dengan masukannya (yang diinginkan). Selisih antara *set point* dengan umpan balik digunakan sebagai masukan komputasi untuk menentukan seberapa besar nilai sinyal kendali yang diberikan ke *plant* sehingga parameter-parameter kendali yang diharapkan dapat diimplementasikan . (Killian, (1997) Modern Control Technology, Componen and Systems, 2nd edition. Prentice - Hall International, Inc).

Diagram blok sistem kendali secara *loop* tertutup digambarkan sebagai berikut :

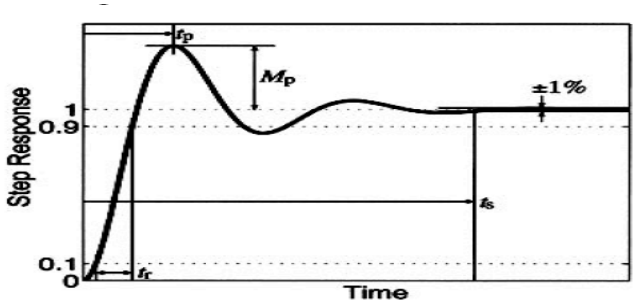


Gambar II.4. Diagram sistem kendali kecepatan secara *loop* tertutup

Pada sistem loop tertutup, kemampuan regulasi dapat ditingkatkan akan tetapi isu kestabilan menjadi hal yang penting karena mungkin saja koreksi terhadap error yang terjadi dapat menyebabkan osilasi sehingga sistem tidak stabil.

II.1.3. Kinerja sistem kendali

Kinerja sistem kendali sangatlah penting sehingga pada proses pencampuran dan pengembangan sistem, parameter-parameter kinerja menjadi acuan hasil yang diharapkan. Kinerja sistem dengan masukan *step* merupakan acuan yang umum digunakan. (Killian, (1997) Modern Control Technology, Componen and Systems, 2nd edition, Prentice – Hall International, Inc). Kinerja sistem dengan masukan *step* dalam domain waktu,dinyatakan dalam grafik berikut ini :



Gambar II.5. Kinerja sistem kendali dalam domain waktu

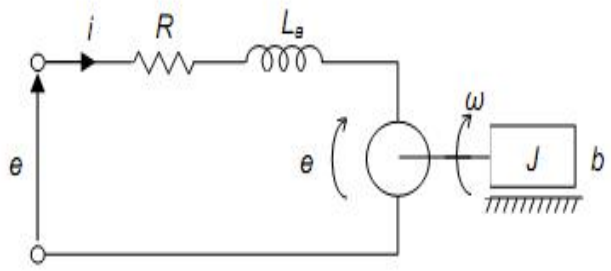
Secara umum, karakteristik sistem tersebut dinyatakan :

- Waktu tunda (*delay time*), *td* adalah waktu yang diperlukan respon mencapai 50 % nilai akhir pada pertama kali.
- Waktu naik (*rise time*), tr adalah waktu yang diperlukan respon untuk naik 0% sampai 100% dari nilai akhir (untuk redaman kurang) atau dari 10% sampai dengan 90% dari nilai akhir (untuk redaman lebih).
- Waktu puncak (*peak time*), tp adalah waktu yang diperlukan respon untuk mencapai puncak simpangan yang pertama kali.
- Lewatan maksimum (*maximum overshoot),* mp adalah perbandingan antara puncak tertinggi dari kurva respon terhadap nilai akhir respon.

- Waktu penetapan (*settling time)*, ts adalah waktu yang diperlukan agar kurva respon mencapai dan tetap berada di dalam batas-batas yang dekat dengan nilai akhir.

II.2. Motor DC

Motor dc adalah perangkat listrik yang mengubah energi listrik menjadi energi mekanik. (Muslimin Marappung, (1979) Teknik Tenaga Listrik , Armico Bandung). Model motor DC yang disederhanakan terlihat pada gambar II.10 :



Gambar.II.10. Sistem Motor dc

Sistem motor DC yang mempunyai medan konstan dengan pengendalian pada armature diperlihatkan pada gambar II.10 . Resistansi dan induktansi armature dinotasikan oleh Ra dan La. Tegangan balik motor e_m (t) dinyatakan sebagai :

$$E_m (t) = Kb \, \omega (t)$$

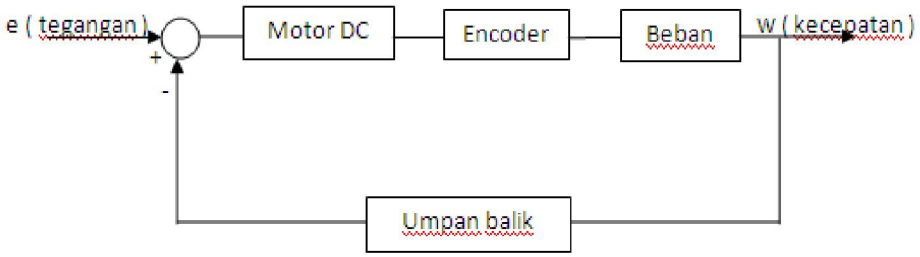
Dimana θ merupakan posisi shaft motor, ω adalah kecepatan sudut shaft, dan Kb adalah kontanta motor, jumlah momen inersia yang dihubungkan ke shaft motor adalah J, dan b adalah gesekan rekatan (viocous). Dengan memisahkan T sebagai torsi yang dihasilkan oleh motor, maka kita tuliskan :

$$T = J\frac{d^2\theta}{dt^2} + b\frac{d\theta}{dt} \quad \text{atau} \quad T = J\frac{d\omega}{dt} + b\omega$$

Torsi yang dihasilkan untuk motor ini diberikan oleh :

$$T = K_T\ Ia$$

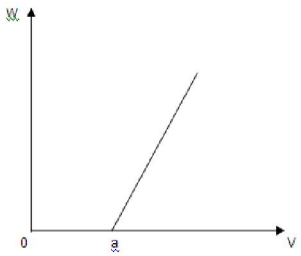
Diagram blok motor DC



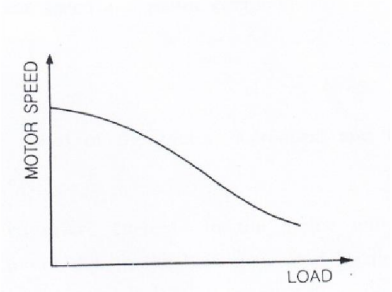
Gambar II.11. Diagram blok motor DC dengan Umpan balik

Masukan e berupa tegangan sementara keluasrannya w berupa kecepatan sudut pengaturan kecepatan sudut yang diamati untuk dibandingkan tegangan masukan kemudian diatur dengan pengaturan pemberian tegangan yang diberikan ke motor.

II.2.1. Karakteristik motor DC tak berbeban



II.2.2. Karakteristik motor DC berbeban



II.2.3. Sensor

Sensor adalah suatu peralatan yang berfungsi untuk mendeteksi gejala-gejala atau sinyal-sinyal yang berasal dari perubahan suatu energi seperti energi listrik, energi fisika, energi kimia, energi biologi, energi mekanik dan sebagainya. (Jacob Fraden. 2004. Handbook of Modern Sensors Physics, Designs, and Applications. Springer-Verlag New York, Inc)

Contoh; Camera sebagai sensor penglihatan, telinga sebagai sensor pendengaran, kulit sebagai sensor peraba, LDR (*light dependent resistance*) sebagai sensor cahaya, dan lainnya.

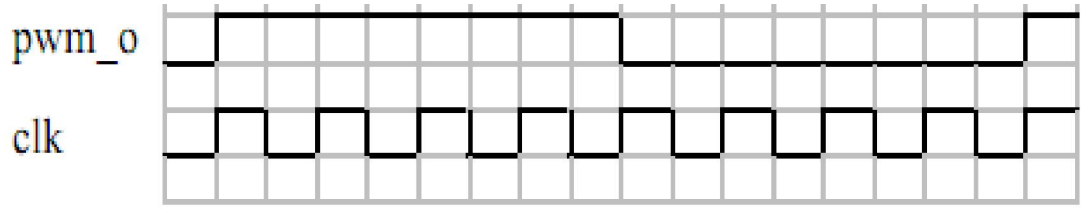
II.2.4. Sensor Kecepatan (*Motion Sensor*)

Pengukuran kecepatan dapat dilakukan dengan cara analog dan cara digital. Secara umum pengukuran kecepatan terbagi dua cara yaitu: cara angular dan cara translasi. Untuk mengukur kecepatan translasi dapat diturunkan dari cara pengukuran angular. Yang dimaksud dengan pengukuran angular adalah pengukuran kecepatan rotasi (berputar), sedangkan pengukuran kecepatan translasi adalah kecepatan gerak lurus beraturan dan kecepatan gerak lurus tidak beraturan.

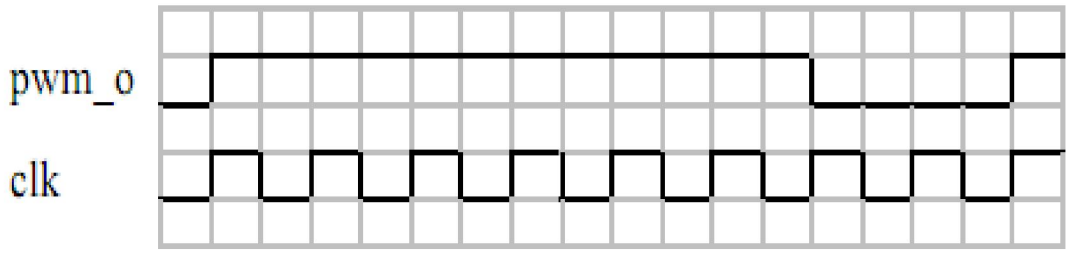
II.3. PWM (Pulse width modulation)

PWM merupakan mekanisme untuk membangkitkan sinyal keluaran yang periodenya berulang antara ***high*** dan ***low,*** dimana kita dapat mengontrol durasi sinyal ***high*** dan ***low*** sesuai dengan yang kita inginkan.***Duty cycle*** merupakan prosentase periode sinyal ***high*** dan periode sinyal ***low***, prosentase ***duty cycle*** akan berbanding lurus dengan tegangan rata-rata yang dihasilkan. (Wikipedia, PWM (Pulse Width

Modulation) en.Wikipedia org/ wiki / pulse – width – modulation (diakses : Januari 2012) Berikut gambaran sinyal PWM, misalkan kondisi *high* 5 Volt dan kondisi *low* 0 Volt



a. Duty cycle 50% . V rata-rata 2,5 V



b. Duty cycle 75% , V rata-rata 3.75 V

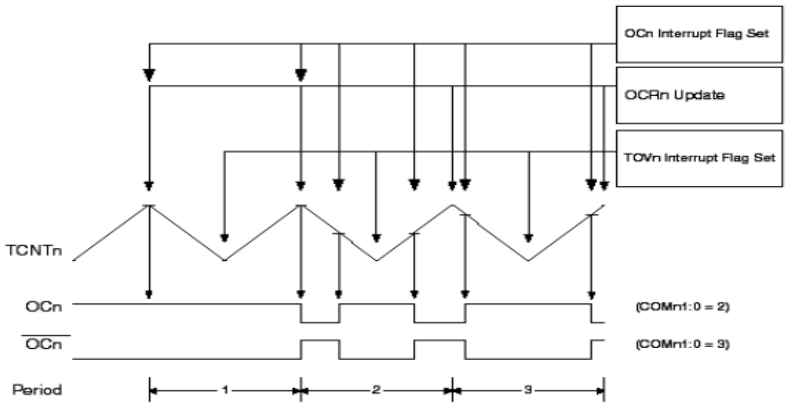
Gambar II.6. PWM dengan variasi duty cycle

Pembangkit sinyal PWM menggunakan fungsi *timer/counter* yang dibandingkan nilainya dengan sebuah register tertentu. Beberapa mikrokontroller, yang penulis gunakan (ATMega 8535) telah menyediakan bebeapa fasilitas sehingga mempermudah pengembangannya.

II.3.1 PWM mode Phase corrent

Dalam ATMega 8535 dapat di hasilkan PWM mode *Phase corrent* dimana nilai register counter TCNTn yang mencacah naik dan turun secara terus menerus akan selalu dibandingkan dengan register pembanding OCRn. Hasil perbandingan register TCNTn dengan OCRn digunakan untuk membangkitkan sinyal PWM yang dikeluarkan

melalui sebuah pin OCn seperti gambar di bawah ini :



Gambar II.7. PWM mode *phase corrent*

Pada PWM 8 bit,maka frekuensi dan ***duty cycle*** pada mode ***phase corrent*** dirumuskan :

$f_{PWM} = \frac{fosc}{Nx512}$ $f_{DPWM} = \frac{OCRx}{255}$ x 100

Dimana : $_{PWM}$ = Frekuensi PWM

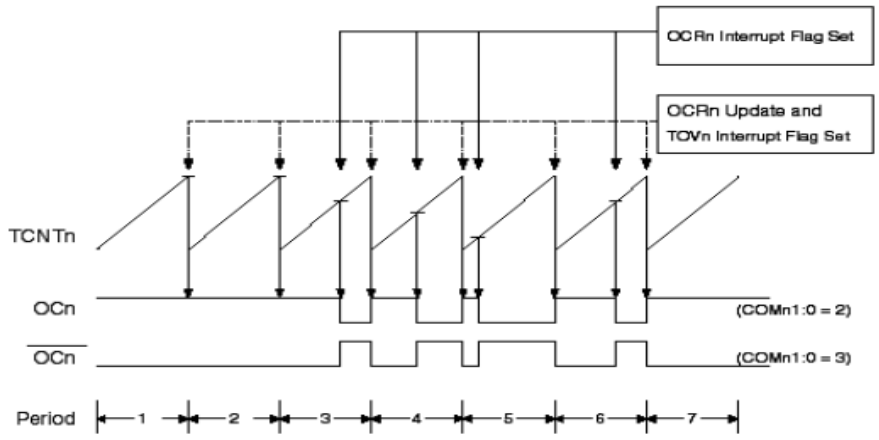
$_{DPWM}$ = Frekuensi osilator

N = Skala *clock*

D = *Duty cycle*

II.3.2 PWM mode *fast*

Pada mode ***fash*** hampir sama dengan mode ***phase correct,*** hanya register TCNTx mencacah naik tanpa mencacah turun seperti terlihat dalam gambar di bawah ini :



Gambar II.8. PWM *mode fash*

Pada PWM 8 bit, maka frekuensi dan ***duty cycle*** pada mode ***phase fash*** dirumuskan :

$f_{PWM} = \frac{fosc}{Nx512}$ $_{DPWM} = \frac{OCRx}{255}$ x 100

Dimana : f_{PWM} = Frekuensi PWM

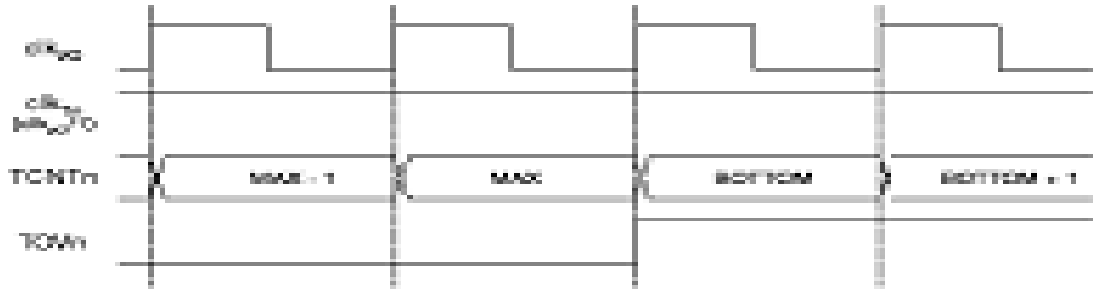
$_{DPWM}$ = Frekuensi osilator

N = Skala ***clock***

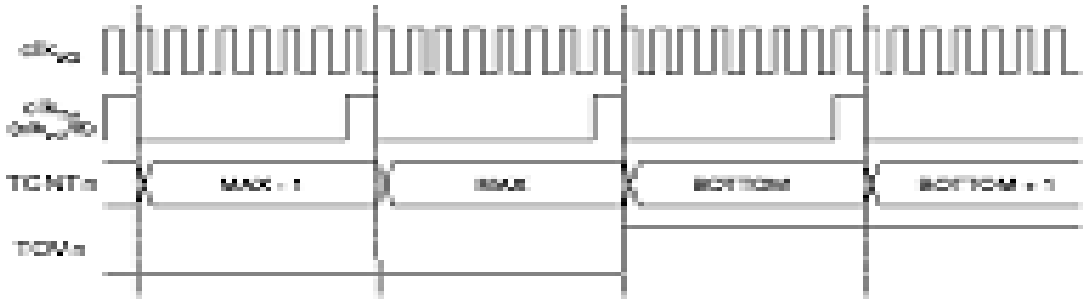
D = ***Duty cycle***

II.3.3. Pre skalar pada PWM

Preskalar merupakan faktor pengali *clock* (skalar *clock*) sehingga frekuensi register TCNT dapat diatur. Misalkan *preskalar* 8 berarti bahwa TCNT akan naik setiap 8 siklus *clock.* Perbedaan antara tanpa dan dengan *preskalar* di gambarkan berikut ini:



Gambar II.9 (a) *Timer/counter* tanpa *preskalar*



Gambar II.9 (b) *Timer/counter* dengan *preskalar*

Dengan memasukkan *preskalar* dalam menentukan frekuensi PWM, maka kita dapat memberi frekuensi PWM yang beragam,tidak harus sama dengan frekuensi osilator.

Motor dc dilengkapi dengan *encoder* sebagai sensor kecepatan dengan ketelitian 116 pulsa / rotasi. Jika motor berputar, maka beban berupa generator juga akan berputar sehingga menghasilkan tegangan pada generator .

II.4. Mikrokontroller

Merupakan sebagai sebuah " one chip solution " pada dasarnya adalah rangkaian terintegrasi (*Integrated Circuit* – IC) yang telah mengandung secara lengkap berbagai komponen pembentuk sebuah komputer . (Widodo Budiharto, (2005) Perancangan Sistem dan Aplikasi Mikrokontroller, PT. Elex Media Komputindo, Jakarta).

Berbeda dengan mikroprosessor yang masih memerlukan komponen luar tambahan seperti seperti RAM, ROM, Timer dan sebagainya.Untuk sistem mikrokontroller tambahan komponen di atas secara praktis hampir tidak dibutuhkan tidak dibutuhkan lagi. Hal ini disebabkan semua komponen penting tersebut telah ditanam bersama dengan sistem prosesssor ke dalam IC tunggal mikrokontroller bersangkutan. Mikrokontroller merupakan contoh suatu sistem komputer sederhana yang masuk dalam kategori ***embedded computer***.

Berdasarkan fungsinya, mikrokontroller secara umum digunakan untuk menjalankan program yang bersifat permanen pada sebuah aplikasi yang spesifik (misal aplikasi yang berkaitan dengan pengontrolan dan monitoring). Sedangkan program aplikasi yang dijalankan pada sistem mikroprosessor biasanya bersifat sementara dan berorientasi pada pengolahan data.

II.4.1. Mikrokontroller Seri AVR

Mikrokontroller Seri AVR pertama kali diperkenalkan ke pasaran sekitar tahun 1997 oleh perusahaan Atmel, yaitu sebuah perusahaan yang sangat terkenal dengan produk mikrokontroller seri MCS51 yang sampai sekarang masih banyak digunakan. Tidak seperti mikrokontroller seri MCS51 yang masih mempertahankan arsitektur dan set instruksi dasar mikrokontroller 8051 dari perusahaan INTEL. Mikrokontroller AVR ini memiliki arsitektur dan set instruksi yang berbeda dengan arsitektur mikrokontroller sebelumnya yang diproduksi oleh perusahaan tersebut. Namun demikian istilah – istilah dasarnya hamper sama, pemrograman level assembler – nya pun relatif tidak jauh berbeda. (Lingga Wardhana. 2006. Belajar Sendiri Mikrokontroler AVR Seri ATMega 8535 Simulasi, Hardware dan Aplikasi. Andi,Yogyakarta.).

Berdasarkan arsitekturnya, AVR merupakan mikrokontroller RISC (*Reduce Intruction Set Computer*) dengan lebar bus data 8 bit. Berbeda dengan sistem MCS51 yang memiliki frekuensi kerja seperduabelas kali frekuensi osilator, sehingga hal tersebut menyebabkan kecepatan kerja AVR untuk frekuensi osilator yang sama adalah dua belas kali lebih cepat dibandingkan dengan mikrokontroller keluarga MCS51.

Dengan instruksi yang sangat variatif (mirip dengan system CISC – *Complex Insnstruction Set Computer)* serta jumlah register serbaguna (*General Purpose Register*) sebanyak 32 buah yang semuanya terhubung secara langsung ke ALU (*Arithmetic Logic Unit*), kecepatan operasi mikrokontroller AVR ini dapat mencapai 16 MIPS (enam belas juta instruksi per detik) sebuah kecepatan yang sangat tinggi untuk ukuran mikrokontroller 8 bit yang di pasaran saat ini.

Untuk memenuhi kebutuhan dan aplikasi industri yang sangat beragam, mikrokontroller keluarga AVR ini muncul di pasaran dengan tiga seri utama yaitu : tinyAVR, ClasicAVR (AVR), megaAVR. Berikut ini beberapa seri yang dapat dijumpai di pasaran.

Tabel 1. Seri AVR yang ada di pasaran

Seri tinyAVR	Seri ClasicAVR	SerimegaAVR
ATtiny13	AT90S2313	ATMega 103
ATtiny22L	AT90S2323	ATMega 128
ATtiny22	AT90S2333	ATMega 16
ATtiny2313	AT90S4414	ATMega 162
ATtiny26	AT90S4433	ATMega 168
ATtiny2313V	AT90S8515	ATMega 8535

Keseluruhan seri AVR ini pada dasarnya memiliki organisasi memori dan set instruksi yang sama (sehingga dengan demikian jika kita telah mahir menggunakan salah satu seri AVR, untuk beralih ke seri yang lain akan relative mudah). Perbedaaan antara tinyAVR, AVR dsn megaAVR pada kenyataannya hanya terletak pada fitur yang ditawarkan (misal adanya tambahan ADC internal pada seri AVR tertentu, jumlah Port I/O serta memori yang berbeda, dan sebagainya). Di antara ketiganya mega AVR umumnya memiliki fitur yang paling lengkap, disusul oleh ClasicAVR dan terakhir tinyAVR.

Tabel 2. memperlihatkan perbedaan ketiga seri AVR dilihat dari jumlah memori yang dimilikinya. Seperti terlihat pada tabel 2 tersebut. Semua jenis AVR ini telah dilengkapi dengan dengan memori flash sebagai memori program . Tergantung serinya, kapasitas memori flash dimiliki bervariasi dari 1K sampai 128 KB. Secara teknis, memori jenis ini dapat diprogram melalui salueran antarmuka yang dikenal dengan nama *Serial Peripherial Interface* (SPI) yang terdapat pada setiap seri AVR tersebut.

Dengan menggunakan perangkat lunak *programmer* (*downloader*) yang tepat, pengisian memori flash dengan menggunakan saluran SPI dapat dilakukan bahkan ketika chip AVR telah terpasang pada sistem akhir (*and system*), sehingga dengan demikian pemrogramannya sangat fleksibel dan tidsak merepotkan pengguna. Secara praktis metode ini dikenal dengan istilah ISP- In System Programming, sedangkan perangkat lunaknya dinamakan In System Programmer .

Tabel 2. Perbedaan seri AVR berdasarkan jumlah memori

Mikrokontroller AVR		**Memori (byte)**		
Jenis	Paket IC	Flash	EEPROM	SRAM
TinyAVR	8 – 32 pin	1 - 2 K	64 - 128	0 – 128
AVR (Clasic AVR)	20 – 44 pin	1 – 8 K	128 - 512	0 – 1 K
MegaAVR	32 – 64 pin	8 - 128	512 – 4 K	512 – 4 K

Untuk menyimpan data, mikrokontroller AVR menyediakan dua jenis memori yang berbeda yaitu EEPROM (*Electrically Erasable Programmable Read Only Memory*) dan SRAM (*Static Random Access memory).* EEPROM umumnya digunakan untuk menyimpan data-data program yang bersifat permanen sedangkan SRAM digunakan untuk menyimpan data variabel yang dimungkinkan berubah setiap saat.Kapasitas simpan data kedua memori ini bervariasi tergantung pada jenis AVR-nya.Untuk seri AVR yang tidak memiliki SRAM, penyimpanan data Variabel dapat dilakukan pada register serbaguna yang terdapat pada CPU mikrokontroller tersebut .

Selain seri-seri di atas yang sifatnya lebih umum, perusahaan Atmel juga memproduksi beberapa jenis mikrokontroller AVR untuk tujuan yang lebih khusus dan terbatas, seperti seri AT86RF401 yang khusus digunakan untuk aplikasi *wireless remote control* dengan menggunakan gelombang radio (RF), seri AT90SC yang khusus digunakan untuk peralatan sistem-sistem keamanan kartu SIM,GSM dan lain sebagainya.

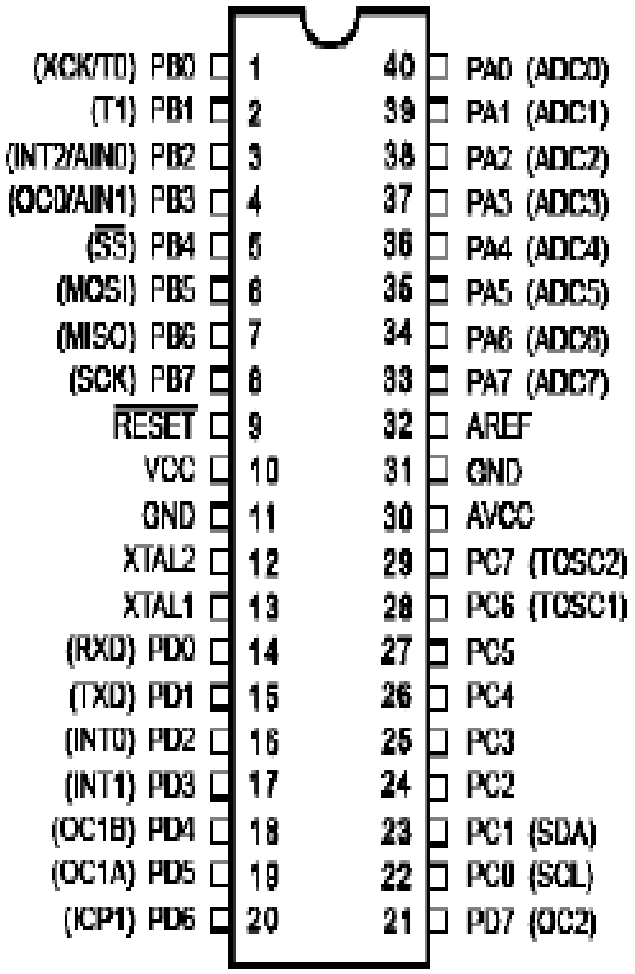
II.4.2. Mikrokontroller AVR ATMega 8535

ATMega 8535 merupakan mikrokontroller keluarga AVR produksi Atmel. Mikrokontroller AVR ini memiliki arsitektur RISC 8 bit, dimana semua instruksi dikemas dalam kode 16 bit dan sebagian besar konstruksi dalam 1 (satu) Siklus clock, (Ardi Winanto,(2010) Mikrokontroller AVR ATmega 8535 dan Pemrogramannya dengan Bahasa C pada Win AVR . Informatika , Bandung), berbeda dengan instruksi MCS51 yang membutuhkan 12 siklus clock. Hal ini terjadi karena kedua jenis mikrokontroller tersebut memiliki arsitektur yang berbeda . AVR berteknologi RISC (*Reduced Instruction Set Computing*) sedangkan seri MCS51 berteknologi CISC (*Complex Intruction aset Computing)* . Mikrokontroller ATMega 8535 termasuk keluarga RISC produksi Atmel.Fitur-fitur pada ATMega 8535 sebagai berikut :

- 130 macam instruksi yang hampir semuanya dieksekusi dalam satu siklus *clock*
- 32 x 8 bit *register* serbaguna
- Kecepatan mencapai 16 MIPS dengan clock 16 MHz
- 8 Kbyte Flash Memori yang memiliki fasilitas *In – System Programming*
- 512 Byte internal EEPROM
- 512 Byte SRAM
- Programming clock, Fasilitas untuk mengamankan kode program
- 2 buah timer/counter ++8-Bit dan satu buah timer/counter 16- Bit
- 4 saluran keluaran PWM
- 8 channel ADC 10-Bit

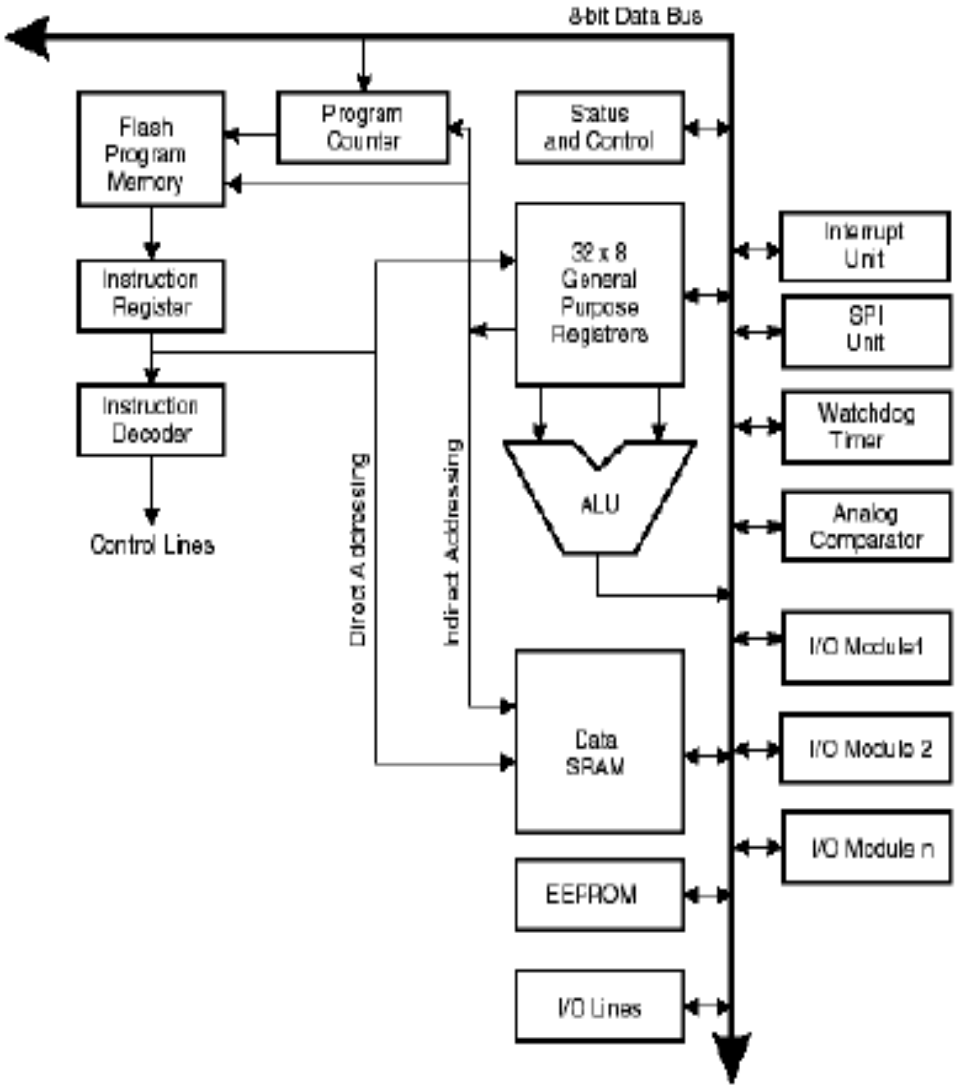
- Serial USART
- Master/Slave serial interface
- Serial TWI dan I2C
- On-Chip analog

Konfigurasi PIN TMega 8535 sebagai berkut :



Gambar II.11. Pin Atmega 8535

Arsitektur ATMega 8535 sebagai berikut :



Gambar II.12. Arsitektur ATmega 8535

II.4.2.1. Port Input/output ATMega 8535

Konfigurasi pin ATmega 8535 dengan kemasan 40 pin DIP (Dual Inline Package) dapat dilihat pada gambar II.4.2.1. penjelasan fungsi dari masing-masing pin ATmega 8535 sebagai berikut :

1. VCC merupakan pin yang berfungsi sebagai masukan catu daya.
2. GND merupakan pin ground.
3. RESET merupakan pin yang digunakan untuk me-reset mikrokontroller.
4. XTAL 1 dan XTAL 2 merupakan pin masukan clock eksternal.
5. AVCC merupakan pin masukan tegangan untuk ADC.
6. AREFF merupakan pin masukan tegangan referensi ADC.
7. Port. A (Port A.0 …….Port.A.7) merupakan pin input/output dua arah dengan fungsi alternatif sebagai pin masukan ADC (tabel 3)

Tabel 3. Fungsi alternatif pin port.A

Port pin	Alternatif function
PA7	ADC7 (ADC input cannel 7)
PA6	ADC6 (ADC input cannel 6)
PA5	ADC5 (ADC input cannel 5)
PA4	ADC4 (ADC input cannel 4)
PA3	ADC3 (ADC input cannel 3)
PA2	ADC2 (ADC input cannel 2)
PA1	ADC1 (ADC input cannel 1)
PA0	ADC0 (ADC input cannel 0)

8. Port B (PortB0 …… portB7) merupakan input /output dua arah dengan fungsi alternatif seperti pada tabel 4.

Tabel 4. Fungsi alternatif pin PortB

Port pin	Alternatif function
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/ Slave Output)
PB5	MOSI (SPI Master Output/Slave Input)
PB4	SS (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negatif Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positif Input) INT2 (External Interupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

9. Port C (PortC0 ……..PortC7) merupakan pin input /output dua arah dengan fungsi alternatif seperti pada tabel 5.
10. Port D (PortD0 …….PortD7) merupakan pin Input/output dua arah dengan fungsi alternatif seperti pada tabel 6.

Tabel 5. Fungsi alternatif pin PortC

Port pin	Alternatif function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC2 (Timer Oscillator Pin 1)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire erial Bus Clock Line)

Tabel 6. Fungsi Alternatif pin PortD

Port pin	Alternatif function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter 1 Input Capture Pin)
PD5	OC1A (Timer/Counter 1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter 1 Output Compare B Match Output)
PD3	INT1 (External Interupt 1 Input)
PD2	INT0 (External Interupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

Keempat port tersebut merupakan jalur *bidirectional* yang semuanya dapat diprogram sebagai input atau output dengan pilihan *internal pull-up*. Tiap port mempunyai tiga buah register bit, yaitu DDXn, PORTxn dan PINxn. Huruf ‘x’ mewakili nama huruf dari port sedangkan huruf ‘n’ mewakili nomor bit.Bit DDxn terdapat pada I/O address DDRx, bit PORTxn terdapat pada I/O address PORTx, dan bit PINxn terdapat pada I/O address PINx. Pengaturan konfigurasi Port ditunjukkan pada tabel 7.

Bit DDxn dalam register DDRx (*Data direction Register*) menetukan arah pin. Bila DDxn diset 1 maka PORTxn berfungsi sebagai pin output dan bila diset 0 maka PORTxn berfungsi sebagai input. Bila PORTxn diset 1 pada saat pin terkonfigurasi sebagai pin input, maka resistor *pull-up* akan diaktifkan. Untuk mematikan resistor *pull-up*, PORTxn harus diset 0 atau pin dikonfigurasi sebagai pin output. Pin port adalah tri state setelah kondisi reset.

Tabel 7. Konfigurasi pin port

DDxn	**PORTxn**	**PUD (In SFIOR)**	**I/O**	***Pull-up***	**Comment**

0	0	X	Input	No	Tri- State (Hi-Z)
0	1	0	Input	Yes	Pxn will sourse current it ext.pulled low
0	1	1	Input	No	Tri-State (Hi – Z)
1	0	X	Input	No	Output Low (Sink)
1	1	X	Input	No	Output High (Sourse)

Bila PORTxn diset 1 pada saat pin dikonfigurasi sebagai pin output maka pin port akan berlogika 1. dan bila PORTxn diset 0 pada saat pin dikonfigurasi sebagai pin output maka pin port akan berlogika 0 .

Bila bit 2 (PUD : *Pull-up Disable*) bernilai 1 maka *pull-up* pada port I/O akan dimatikan walaupun register DDxn dan PORTxn dikonfigurasi untuk menyalakan pull-up (DDxn = 0 , PORTxn = 1)

II.4.2.2. Komunikasi USART pada ATmega 8535

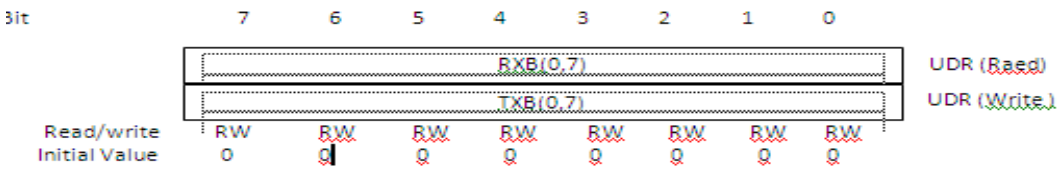
USART (*Universal Synchronous and asynchronous Serial Receiver and transmitter)* pada ATmega 8535 merupakan perangkat komunikasi serial sangat fleksibel (Wikipedia, USART. En.Wikipedia.org/Wiki/universal_asynchronous_receiver/ transmitter

(diakses : Januari 2012) dengan fitur utama sebagai berikut :

1. Operasi *full duplex* (register penerima dan pengirim serial dapat berdiri sendiri).
2. Operasi Asinchronous atau synchronous.
3. Operasi clocked syncrhronous pada master atau slave
4. Pembangkit baud rate dengan resolusi tinggi
5. Mendukung serial frames dengan 5,6,7,8 atau 9 data bit dan 1 atau 2 stop bit.
6. Pembangkit bit odd atau even varity dan varity check didukung oleh hardware.
7. Penyaringan gangguan (noice) meliputi pendeteksian bit false start dan pendeteksian digital low pass filter.
8. Tiga interrupt terdiri dari TX complete, TX data register empty dan RX complete.
9. Mendukung mode komunikasi multi prosessor (MPCM)
10. Mode komunikasi *double speed asynchronous*.

Untuk mengatur komunikasi serial pada ATmega 8535, ada beberapa register yang perlu dikonfigurasi yaitu Usart I/O Data register (UDR), *Usart Control and status Register A* (UCSRA), *Usart Control and Status Register B* (UCSRB), *Usart Control and Status Register C* (UCSRC) dan *Usart Baud Rate Register* (UBRRL dan UBRRH).

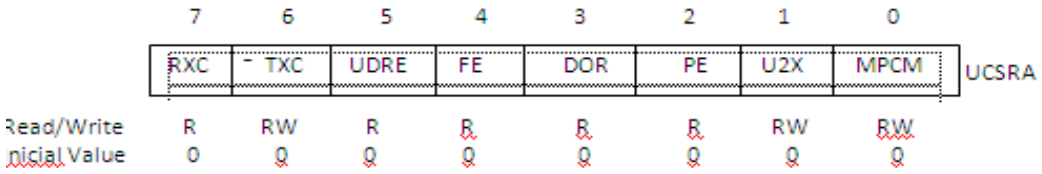
Register UDR adalah register penampung data kirim dan data terima. Data yang akan dikirim harus ditempatkan pada register ini, dan data yang diterima dari luar dibaca pada register ini pula.



Gambar II.13. Register UDR

UDR terdiri dari dua buah register terpisah, dengan alamat dan nama yang sama, yaitu UDR. Saat menulis data untuk dikirim pada UDR ini, maka yang dipakai adalah UDR (write). Saat menerima data setelah hadirnya stop bit, yang dibaca adalah UDR (Read). Instruksi *OUT* DAN *IN* dapat dipakai untuk menulis dan membaca register UDR ini.

Register UCSRA adalah register yang berisi status dari proses transfer data komunikasi serial. Susunan bit-bit pada register UCSRA ditunjuk pada gambar 15 :



Gambar II.14. Bit – bit pada register UCSRA

Penjelasan dari bit-bit pada register UCSRA adalah sebaga berikut :

1. **Bit 7 (RXC : USART Receive Complete).**

   Bit ini menjadi tinggi jika ada data di dalam buffer penerima (UDR – read) yang belum diambil atau dibaca dan akan otomatis rendah setelah buffer penerima dibaca. Jika unit penerima tiba-tiba dimatikan setelah diaktifkan, maka isi dalam buffer penerima akan langsung dibuang (flushed) dan bit RXC ini langsung dibuat rendah. Bit ini juga bisa mengaktifkan interupsi " *Receive Complete interrupt* " . Lihat penjelasan tentang bit RXCIE .

2. **Bit 6 (TXC : USART Transmit Complete).**

   Bit ini akan otomatis tinggi saat semua frame dalam shift – register pengiriman telah digeser semuanya keluar dan tidak ada data baru yang berada dalam buffer pengiriman (UDR – write). Bit TXC ini akan otomatis rendah setelah "

Transmit Complete interrupt " dijalankan atau dengan meng – clear – kan secara manual dengan cara menulis bit ini dengan nilai 1 (tinggi). Bit TXC ini dapat membangkitkan " *Transmit Complete interrupt* ". Lihat penjelasan tentang bit TXCIE .

3. **Bit 5 (UDRE : USART Data Register Empty)**

   Bit UDRE ini adalah untuk memberikan tanda jika buffer pengiriman (UDR – write) telah siap untuk menerima data baru. Bila bit ini bernilai 1 (tinggi), UDR dikosongkan dansiap ditulis. Bit dapat membangkitkan UDRIE atau " *Data Register Empty interrupt* ". Lihat penjelasan tentang bit UDRIE. Bit ini bernilai 1 setelah rerset, yang berarti siap untuk melakukan pengiriman.

4. **Bit 4 (FE : Frame Error).**

   Bit ini otomatis menjadi tinggi jika saat menerima data, ternyata ada kesalahan dari frame yang diterima. Misalnya saat unit penerima seharusnya menunggu sebuah bit stop, ternyata data yang ada adalah 0 (rendah) saat kita menulis UCSRA.

5. **Bit 3 (DOR : Data Over Run)**

   Bit ini otomstis menjadi tinggi saat kondisi overrun terjadi. Kondisi ini terjadi saat buffer penerima sudah penuh dan berisi 2 data karakter , dimana data karakter terakhir tidak bisa dipindahkan ke UDR – read, karena tidak kunjung dibaca oleh user. Bit ini valid setelah kita membaca UDR . Harap selalu menuli bit dengan 0 (rendah) saat kita sedang menulis UCSRA.

6. **Bit 2 (PE : Pariry Error)**

Bit ini akan menjadi tinggi saat karakter yang sedang diterima ternyata menjadi formar parity yang salah. Tentu saja hal ini terjadi jika bit *parity checking diaktifkan* (UPM1 = 1). Bir ini valid setelah kita membaca UDR. Harap selalu menulis bit ini dengan 0 (rendah) saat kita sedang menulis UCSRA.

7. **Bit 1 (U2X : Double the USART transmission Speed)**

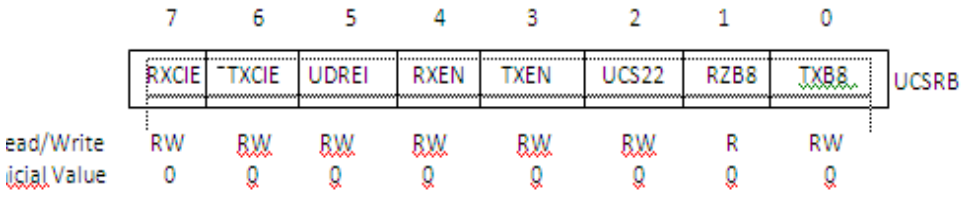
   Bit ini hanya berlaku umtuk operasi tak singkron (asynchronous). Jika bit ini kita tulis dengan 1's (tinggi) maka baut rate akan menjadi lebih cepat 2 kali. Hal itu terjadi karena pembagfi baud rate yang biasanya membagi 16 kemudian membagi menjadi dengan 8 saja.

   Tulis O's (rendah) untuk operasi sinkron (synchronous).

8. **Bit 0 (MPCM : Multi-proprosessor Comunication Mode)**

   Bit ini digunakan untuk mode komunikasi Multi-Prosessor. Saat bit ini dibuat menjadi tinggi maka setiap data yang diterima oleh unit penerima, yang tidak dilengkapi dengan informasi alamat,akan diabaikan. Bit ini hanya berguna untuk penerima, dan bukan untuk pengirim.

Register UCSRB adalah register yang juga berisi status dari proses transfer komunikasi serial dengan susunan bit-bit ditunjukkan pada gambar 16.



Gambar II.15.Bit-bit pada Register UCSRB

Penjelasan dari bit-bit pada register UCSRB adalah sbb :

1. **Bit 7(RXCIE : RX Complete Interrupt Enable)**

   Menulis bit ini menjadi tinggi akan mengakibatkan interupsi yang berkaitan dengan bit RXC. USART Receive Complete Interrupt akan terjadi hanya jika bit RXCIE ini, bit I (Global Interrupt) pada register SREG adalah 1(tinggi), dan bit RXC pada UCSRA juga di – set 1.

2. **Bit 6 (TXCIE : TX Complete Interrupt Enable)**

   Menulis bit ini menjadi tinggi akan mengaktifkan interupsi pada bit TXC,yakni *“ USART Transmisi Complete Interrupt* “ yang akan menjalankan interupsi sertiap frame dari data pengiriman selesai di kirim atau dengan kata lain terjadi interupsi setiap bit TXC menjadi tinggi. Interupsi hanya bisa terjadi jika sebelumnya bit TXCIE ini di set tinggi dan bit Global Interrupt (I) pada register SREG juga di set tinggi.

3. **Bit 5 (UDRIE : USART Data Register Empty Interrupt Enable)**

   Menulis bit ini menjadi tinggi maka akan mengaktifkan interupsi pada bit UDRE, yakni “ *Data Register Empty interrupt* “ yang akan menjalankan interupsi saat data pad buffer pengiriman sidah kosong atau dengan kata lain setiap bit UDRE menjadi tingg. Interupsi hanya bisa terjadi jika sebelumnya bit UDRE ini di set tinggi dan bit Global Interrupt (I) milik register UCSRA juga di set tinggi.

4. **Bit 4 (RXEN : Receiver Enable)**

   Agar unit penerima dari USART dapat bekerja, maka bit RXEN ini harus dibuat tinggi sebelumnya. Begitu dibuat tinggi, pin RxD akan diputus sebagai standar I/O dan dihubungkan dengan unit penerima USART ini, Namun jika tiba-tiba bit

RXEN ini dibuat rendah kembali setelah tadinya tinggi, maka unit penerima USART akan segera menghentikan kerjanya dan membatalkan proses penerimaan data, serta memberdihkan buffer penerima, termasuk juga bit FE, DOR, dan PE.

5. **Bit 3 (TXEN : Transmitter Enable)**

   Agar unit pengirim USART dapat bekerja, maka bit TXEN harus dibuat tinggi sebelumnya. Begitu dibuat tinggi, pin TxD akan diputus dari standar I/O dan dihubungkan dengan unit pengirim USART ini. Namun jika tiba-tiba bit TXEN ini dibuat rendah kembali setelah tadinya tinggi, maka unit pengirim USART masih harus menyelesaikan tugasnya yang terakhir, yakni mengirim data yang tersisa. Baru kemudian unit pegiriman USART akan berhanti dan mengembalikan port TxD menjadi I/O kembali.

6. **Bit 2 (UCSZ2 : Character Size)**

   Bit ini adalah pasangan dari bit UCSZ1 dsan bit UCSZ0 milik register UCSRC, untuk menetukan jumlah data yang hendak di transfer.

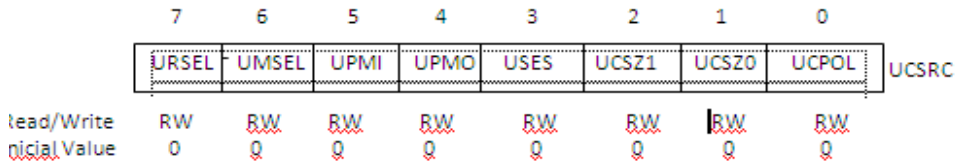
7. **Bit 1 (RXB9 : Receive Data Bit 8)**

   Jika kita menggunakan format penerimaan data 9-bit maka bit yang terakhir yang diterima akan ditempatkan pada bit RXB8 ini. Pabrik menyarankan untuk membaca bit ini terlebih dahulu sebelum membaca 8-bit data lainnya di UDR.

8. **Bit 0 (TXB8 ; Transmit Data Bit 8)**

   Jika kita menggunakan format pengiriman data 9-bit makabit yang terakhir yang hendak dikirim ditempatkan pada bit TXB8 ini. Pabrik menyarankan untuk menulis bit ini terlebih dahulu sebelum menulis 8-bit data lainnya di UDR.

UCSRC adalah register untuk melakukan kontrol pada peralatan USART. Namun jika kita tidak menggunakan fungsi USART, maka register ini boleh diabaikan seperti dalam keadaan resetnya.



Gambar II.16. Bit – bit pada Register UCSRC

Penjelasan bit – bit pada register UCSRC adalah :

1. **Bit 7 (URSEL : Register Select)**

   Saat kita hendak menulis port $20 maka ada dua register yang kita akses, yang ditentukan dari D7 dari data yang kita tuliskan. Jika D7 (MSB) adalah 1atau datanya di atas $80 maka kita sedang menuliskan data pada UCSRC. Sedang jika data yang hendak dituliskan adalah dibawah $80 , maka sedang menulis UBRRH. Sat membaca UCSRC, harus dipastikan bit ini dalam keadaan tinggi sebelum membaca lakasi $20 ini.

2. **Bit 6 (UMSEL :USART Mode Select)**

   Bit ini untuk memilih USART dijadikan mode Asynchronous atau Synchronous.

   Tabel 8. Pemilihan mode operasi asinkron atau sinkron

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

3. **Bit 5 dan 4 (UPM1 dan UPM0 : Parity Mode)**

Bit – bit ini adalah untuk menghidupkan pembangkit dan pemeriksa perity. Jika diaktifkan maka akan otomatis membangkitkan parity pada setiap data yang dikirimkan dan akan memeriksa validitas parity dari setiap data yang diterima. Pada unit penerima kita akan mendapatkan parity yang diterima, kemudian parity tersebut akan dibandingkan dengan status dari UMP0. Jika ternyata tidak cocok, maka bendera bit PE (Perity Error) pada UCSRA akan diaktifkan.

Tabel 9. Pengaturan Mode Parity

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled,Odd Parity

4. Bit 3 (USBS : Stop Bit Select)

Dengan membiarkan bit in menjadi 0 maka frame akan dilengkapi dengan Stop-bit selebar 1-bit. Sedangjika bit ini ditulis tinggi, maka bit stop menjadi 2-bit. Stop bit sepanjang 2-bit ini biasanya diperlkan bagi sistim lain yang terhubung dengan AVR akan memiliki waktu yang cukup untuk memproses data yang baru saja diterimanya, dan sudah benar-benar siap untuk menerima data berikutnya. Unit penerima USART tidak menggunakan bit ini.

Tabel 10.Jumlah bit sebagai bit stop

USBS	Stop Bit(s)
0	1-bit
1	2-bit

5. **Bit 2 (UCSZ1 dan UCSZ0 : Caracter Size)**

menetukan karakter dari data yang hendak dikirimkan dan diterima haruslah merujuk pada bit-bit ini. Yakni UCSZ1 dan bit UCSZ0 milik register ini di tambah dengan bit UCSZ2 pada register UCSBR.

TABEL 11. Pengaturan jumlah bit data yang ditransfer

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

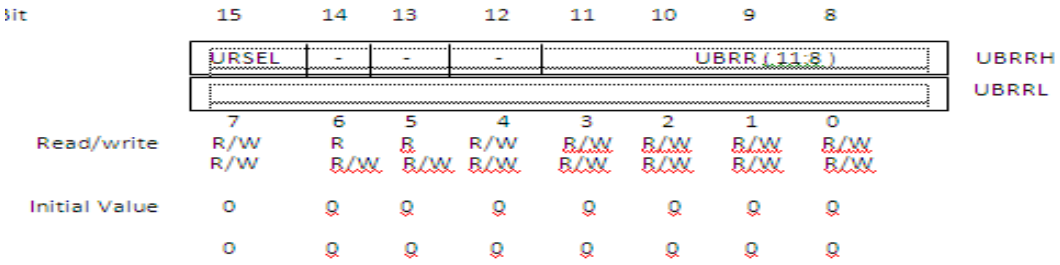
6. **Bit 0 (UCPOL : Clock Polarity)**

Bit ini hanya digunakan pada mode Synchronous. Dalam mode ini data yang diterima dan data yang dikirim disinkronkan dengan status dari XCX (synchronous clock). Tabel 13. menunjukkan bagaiman sinkronisasi terjadi untuk ujung XCX (edge) yang berbeda dengan mengatur bit UCPOL ini.

Tabel 12. Pengaturan Polaritas dalam modeSynchronous

UCPOL	Transmittered Data Changed (Output of TxD Pin)	Received Data Sampled (Input on Rxd Pin)
0	Falling XCK Edge	Rising XCK Edge
1	Rising XCK Edge	Falling XCK Edge

UBRR adalah register 16 bit yang terdiri dua register 8 bit UBBRL dan UBBRH yang berperang menetukan baud rate transfer data. Baud rate antara kedua perangkat yang berkomunikasi haruslah sama untuk mencegah kesalahan transfer data. Register UBRRH memiliki alamat I/O yang sama dengan register UCSRC. Jadi perlu berhati-hati ketika mengakses lokasi I/O ini.



Gambar II.17. Bit – bit pada Register UBRR

Penjelasan bit-bit pada register UBRR adalah sebagai berikut :

1. **Bit 15 (URSEL : Register Seletc)**

   Bit ini memilih dalam mengakses alamat port $20. Seperti yang dijelaskan sebelumnya bahwa alamat port ini dimiliki oleh 2 buah register yang berbeda., yakni UBRRH dan UCSRC. Saat kita hendak menulis UBBRH maka perlu dipastikan databit MSB atau D7 berlogika rendah.

2. **Bit 14,13 dan 12 (Reserved Bits)**

Bit ini tidak digunakan saat ini. Mungkin akan digunakan pada produksi berikutnya.Oleh karena itu kita disarankan untuk tidak menulisnya dengan nilai 1 pada bit-bit yang tidak digunakan.

3. **Bit 11… bit 0 (UBRR11 …. UBBR0 : USART Baud Rate Register)**

   Merupakan register 12-bit yang berisi pengatur baud rate USART.

   Register ini adalah pasangan dari register UBBRH dan UBRRL, dimana UBBRL berisi 8-bit LSB dan sisanya dimiliki oleh UBRRH. Mengubah nilai dari register ini saat transmisi berlangsung, akan menyebabkan kesalahan transmisi, karena baud rate akan langsung berubah seiring dengan berubahnya nilai dari register ini.

   Untuk mengatur baud rate dan nilai setting register UBRR dapat dilakukan perhitungan dengan menggunakan rumus dalam tabel 13.

   Tabel 13. Rumus untuk menetukan baud rate dan isi register UBRR

Operating Mode	**Equation for Caculating Baud Rate**	**Equation for Calculating UBRR Value**
Asynchronous Normal Mode (U2X = 0)	$\text{Baud} = \frac{f_{osc}}{16\,(\text{UBBR} + 1)}$	$\text{UBBR} = \frac{f_{osc}}{16\,\text{BAUD}} - 1$
Synchronous Double Speed Mode (U2X =0)	$\text{Baud} = \frac{f_{osc}}{8\,(\text{UBBR} + 1)}$	$\text{UBBR} = \frac{f_{osc}}{8\,\text{BAUD}} - 1$
Synchronous Master Mode	$\text{Baud} = \frac{f_{osc}}{2\,(\text{UBBR} + 1)}$	$\text{UBBR} = \frac{f_{osc}}{2\,\text{BAUD}} - 1$

Pada tabel tersebut, BAUD adalah *baud rate* (bit per detik atau bps). fosc adalah frekuensi clock isolator dan UBRR adalah nilai isian pada register UBRRH dan UBBRL

II.4.2.3. ADC pada ATmega 8535

ATmega 8535 menyediakan fasilitas ADC dengan resolusi 10 bit dengan waktu konversi 65-260 us. ADC ini dihubungkan dengan 8 channel analog Multiplexer yang memungkingkan terbentuknya 8 input tegangan single-ended yang masuk melalui pin padfa PortA. ADC memiliki pin supply tegangan analog yang terpisah yaitu AVCC. Tegangan referensi ADC dapat dipilih menggunakan tegangan referensi internal maupun eksternal. (Jika menggunakan tegangan referensi internal, bisa dipilih on-chip internal referensi voltage yaitu sebesar 2,56 V atau sebesar AVCC. Jika menggunakan tegangan referensi eksternal, dapat dihubungkan melalui pin ARFF. (Ardi Winanto, (2010) Mikrokontroller AVR ATmega 8535 dan Pemrogramannya dengan Bahasa C pada WinAVR. Informatika, Bandung).

ADC mengkonversi tegangan input analog menjadi data digital 8 bit atau 10 bit. Data digital tersebut akan disimpan di dalam ADC . Data register yaitu ADCH dan ADCL. Sekali ADCL dibaca maka akses ke data register tidak bisa dilakukan, dan ketika ADCH dibaca, maka akses ke data register kembali enable.

ADC memiliki dua mode operasi yaitu Konversi Tunggal (*Single Convertion*) dan Konversi Kontinu (*Free Ranning*).

1. Mode konversi tunggal (*Single Convertion*)

Dalam mode ini konversi dilakukan untuk sekali pembacaan sampel tegangan input. Konversi dimulai ketika bit ADSC di-set dan bit ini tetap set sampai satu kali konversi selesai, setelau itu bit ini otomatis di-clear.

2. Mode konversi kontinu (*Free Ranning*)

   Dalam mode ini, konversi dilakukan secara terus menerus (ADC membaca samel tegangan input lalu dikonversi dan hasilnya ditampung di register ADCH dan ADCL secara kontinu).

Register pengendali ADC terdiri dari ADC *Multilexer Selection Register* (ADMUX), ADC *Control and Status Register A* (ADCSRA), ADC *Data Register* (ADCH-ADCL) dan *Special Function I/O Register* (SFIOR).

II.5. Bahasa Assembly

Salah satu bahasa yang dipakai untuk mengembangkan sistem mikrokontroller AVR adalah bahasa assembly yang dapat dibuat dengan menggunakan software aplikasi AVR CodeVision. AVR CodeVision merupakan software khusus untuk bahasa assembly yang mempunyai fungsi sangat lengkap yaitu digunakan untuk menulis program, kompilasi, simulasi dan download program ke IC mikrokontroller AVR.

Bahasa assembly adalah bahasa pemrograman tingkat rendah. Dalam pemrograman computer dikenal dua jenis tingkatan bahasa, jenis yang pertama adalah bahasa pemrograman tingkat tinggi (hight level language) dan jenis yang ke dua adalah pemrograman tingkat rendah (low level language) .

Bahasa pemrograman tingkat tinggi lebih berorientasi kepada manusia yaitu bagaimana agar pernyataan-pernyataan yang ada dalam program mudah ditulis dan dimengerti oleh manusia. Sedangkan bahasa tingkat rendah lebih berorientasi ke

mesin, yaitu bagaimana agar komputer dapat langsung mengintrepretasikan pernyataan-pernyataan program.

Kelebihan bahasa assembly :

1. Ketika di compile lebih kecil ukurannya / hemat memori.
2. Lebih cepat dieksekusi.
3. Memungkinkan kita untuk melakukan kontrol secara langsung pada register, sehingga program dapat menjadi sangat ramping dan efisien.

Kelemahan bahasa assembly :

1. Dalam melakukan suatu pekerjaan, baris program relatif lebih panjang dibanding bahasa tingkat itnggi.
2. Relatif lebih sulit untuk dipahami terutama jika jumlah baris sudah terlalu banyak.
3. Lebih sulit dalam melakukan pekerjaan rumit, misal operasi matematika.

Sebuah program assembly terdiri dari dua bagian : yaitu program inisialisasi dan program utama. Bagian inisialisasi harus mendeklarasikan *definisi chip* yang dipakai, mendefinikan nama *variabel, konstanta alamat*, awal program dan *stack pointer* agar program utama dapat berjalan.

II.6. Kerangka Pikir

Pengembangan sistem kendali kecepatan motor dc berbasis PWM (Pulse Width Modulation) berawal dari berbagai sistem pengendali yang telah dikembangkan sebelumnya dan selanjutnya dilakukan identifikasi sistem kendali kecepatan motor dc untuk mendapatkan suatu sistem. Pemilihan jenis mikrokontroller, bentuk antar muka dan metode koneksi dengan computer dan mempertimbangkan sensor dan objek

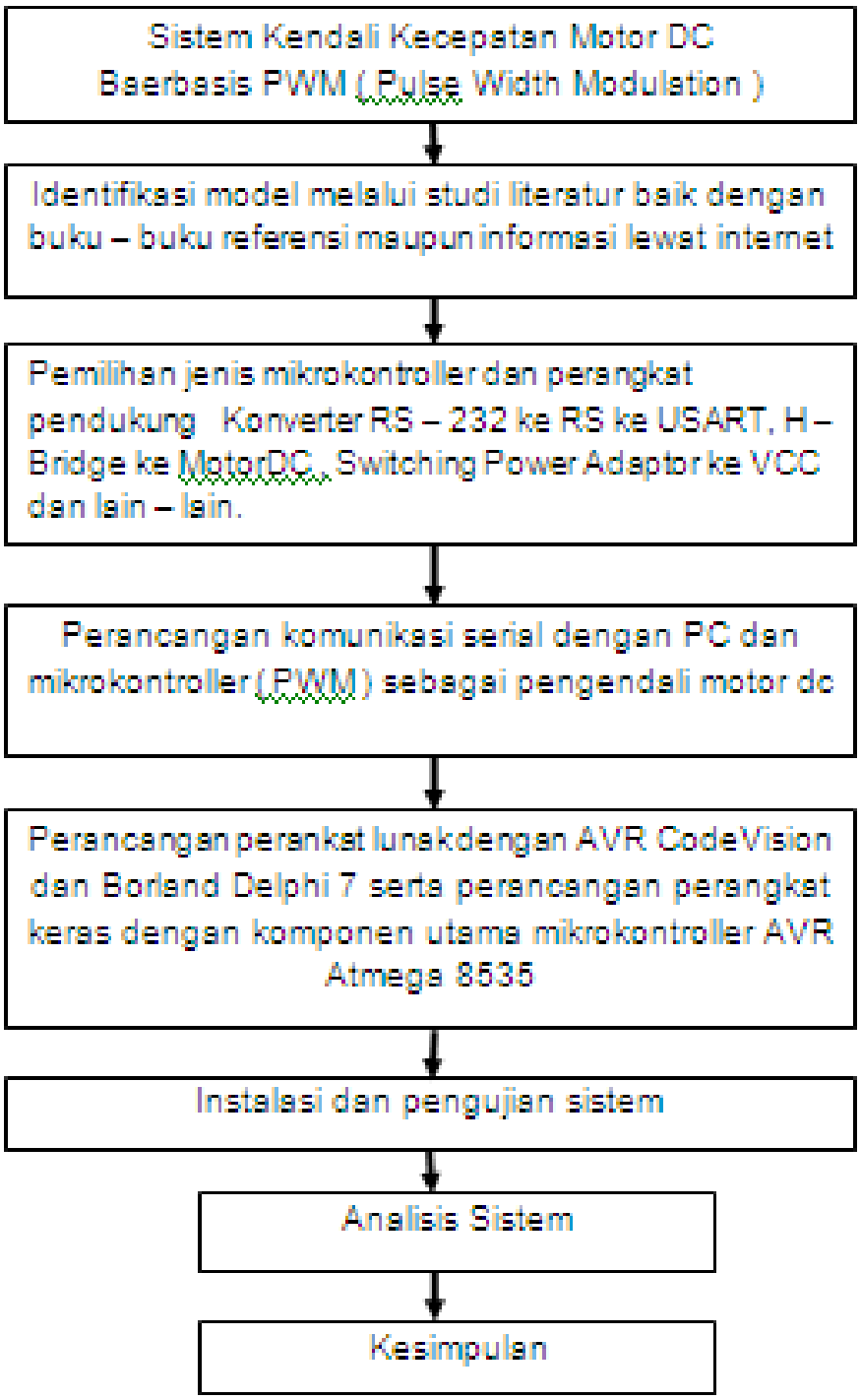
kendalian, implementasi sistem kendali motor dc berbasis PWM merupakan pertimbangan awal dalam perancangan sistem .

Selain perancangan perangkat keras, tahapan yang tidak kalah pentingnya dalam penelitian ini adalah perancangan perangkat lunak, baik pada minimum sistem mikrokontroller maupun pada PC . Pembuatan perangkat lunak dapat dilakukan dengan menggunakan bahasa pemrograman tingkat rendah yaitu bahasa assembly malalui software grafis AVR Studio yang telah disediakan oleh Atmel, atau menggunakan bahasa pemrograman tingkat tinggi misalnya bahasa C melalui software CodeVision buatan Pavel Haiduc and HP Info Tech.Sedangkan perangkat lunak pada PC dibuat dengan bahasa pemrograman visual berorientasi onjek misalnya Visual Basic atau Borland Delphi.

Perangkat lunak pada PC dibuat platform Windows XP berfungsi untuk mengambil data dan member perintah pada mikrokontroller melalui interface saluran komunikasi RS232 inverter hasilnya akan dilihat pada motor dc yang akan berputar sesuai set point yang kita berikan. Perangkat lunat mikrokontroller berfungsi menyalurkan data dari PC ke mikrokonreoller kemudian dikirim ke motor dc melalui H – Bridge.

Dalam sistem ini, PC dan mikrokontroller membentuk komunikasi mono – drop dimana PC berfungsi memberi perintah melalui mikrokontrller dalam hal ini fitur PWM ke motor dc untuk mengatur kecepatannya agar sesuai set point yang telah ditentukan .

Dari uraian diatas dapatlah disusun keangka pikir dalam bentuk bagan sebagaimana diperlihatkan pada gambar II.18.



Gambar II.18. Bagan Kerangka Pikir

BAB III
METODOLOGI PENELITIAN

A. Kerangka Konsep

Dalam penelitian ini penulis menggunakan metodologi yaitu :

III.1. Studi literature

1.1. Mengumpulkan berbagai informasi dar buku-buku, skripsi, tesis maupun jurnal yang berkaitan dengan sistem kendali kecepatan motor dc dari berbagai sumber baik perpustakaan maupun internet.

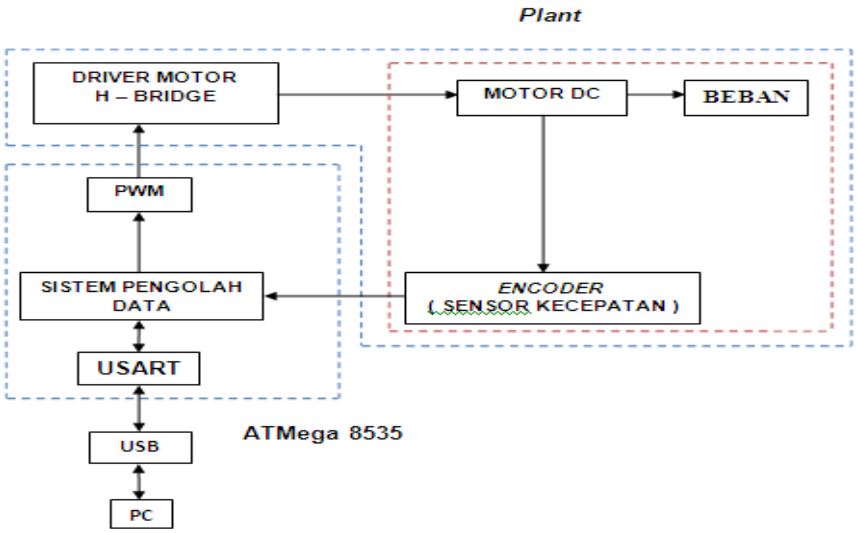
1.2. Mempelajari cara sistem kendalian kecepatan motor dc menggunakan perangkat lunak berupa software yang berorientasi objek.

1.3. Mempelajari sistem komunikasi menggunakan interface RS 232 yang dapat diimplementasikan dalam sistem kendali kecepatan motor dc.

1.4. Mempelajari tentang konsep dasar yang berhubungan dengan prinsip aplikasi PWM pada mikrokontroller khususnya mikrokontroller AVR.

III.2. Rancangan Umum Sistem

Desain Perancangan Sistem



Gambar III.1.Diagram blok sistem kendali kecepatan motor DC Berbasis PWM (Pulse Width Modulation)

Penjelasan fungsi masing-masing elemen sistem di atas sebagai berikut :

1. PC/laptop berfungsi memasukkan nilai *set point* dan akuisisi data, hubungan PC dengan mikrokontroller sebagai pusat pengendali yang memberi perintah ke mikrokontroller melalui USB dan Usart dan data ditampilkan dalam bentuk angka yaitu nilai *set point* dan nilai kecepatan motor pada monitor PC.
2. USB RS – 232 Comverter [4] merupakan penghubung anatara PC dengan mikrokontroller melalui port serial , dimana port serial merupakan fasilitas yang disediakan ole Atmega 8535, dan sebelum dihubungkan maka PC harus diinstal terlebih dahulu supaya PC dan mikrokontroller dapat terhubung .

3. USART merupakan bagian dari mikrokontroller Atmega 8535 yang berfungsi sebagai penghubung antara PC dengan Mikrokontroller melalui USB, dimana usart juga sebagai perangkat komunikasi serial yang mempunyai register penerima dan pengirim serial yang dapat berdiri sendiri.
4. Sistem pengolahan data merupakan bagian utama mikrokontroller Atmega 8535 untuk melakukan komputasi dan sebagai tempat
5. sebagai pemberi penguatan sinyal tegangan dan arus pada driver serta pembangkit sinyal keluaran yang periodenya berulang antara *high* dan *low,* dimana kita dapat mengontrol durasi sinyal *high* dan *low* sesuai pengisian program yang akan dijalankan pada sistem tersebut.
6. PWM merupakan bagian dari mikrokontroller Atmega 8535 yang berfungsi dengan yang kita inginkan dan sekaligus sebagai pengendali kecepatan motor dc.
7. Driver motor berfungsi meningkatkan tegangan dan arus dari PWM sebagai catu daya motor dc yang kemudian disalurkan ke motor dc sebagai objek yang dikendalikan. Proses penguatan sinyal menggunakan metode H – Bridge dengan H – Bridge driver L293D.
8. Motor yang digunakan adalah motor dc dengan jenis motor dc yang digunakan adalah *Yaskawa Minertia & Encoder Model # UGFMED – C9MRX11* tegangan 12 Vdc,arus maks. 0,36 amp. kecepatan motor 1930 rpm, dan torsi 22,2 oz – in , dilengkapi dengan *encoder* 116 pulsa / putaran dan berfungsi sebagai object yang di kendalikan kecepatannya.

9. *Encoder* adalah sebuah generator kecil yang membangkitkan tegangan dc yang berfungsi sebagai sensor kecepatan yang menyatu dengan motor dc sekaligus sebagai pengukur kecepatan motor dc.

Penjelasan proses sistem kendali kecepatan motor DC berbasis PWM :

Pada saat PC diberi nilai kemudian disalurkan ke mikrokontrollere melalui USB dan port serial diproses oleh mikrokontroller dan di salurkan ke PWM sebagai pencacah nilai yang kita berikan kemudian diberi penguatan oleh driver motor , setelah diberi penguatan kemudian di salurkan ke motor dan motor berputar.

III.2.1. Analisis Perancangan

III.2.1. 1. Perancangan Hardware

Hardware yang dibutuhkan dalam perancangan yaitu : Analisis pengaturan kecepatan motor dc,

III.2.1.1.1. Motor DC

Motor yang digunakan dalam penelitian ini adalah motor dc dengan jenis motor dc yang digunakan adalah *Yaskawa Minertia & Encoder Model # UGFMED – C9MRX11* tegangan 12 Vdc,arus maks. 0,36 amp. kecepatan motor 1930 rpm, dan torsi 2,22 lb – in , dilengkapi dengan *encoder* 116 pulsa / putaran

Gambar III.2. Motor DC

III.2.1.1.2. Pengaturan kecepatan motor dc

Pengaturan kecepatan motor dc dapat dilakukan dengan mengatur besar tegangan masukan. Kecepatan putar motor dc (ω) dapat dirumuskan pada persamaan di bawah ini :

$$\omega = \frac{Vt - Ra\ Ia}{Kb} \text{ (rad/sec)}$$

Vt merupakan tegangan masukan motor dalam volt, Ia adalah arus masukan motor dalam amp., Ra adalah hambatan jangkar motor dalam ohm, Kb adalah pluks magnetic dalam Volt.dt/rad, ω kecepatan motor dalam rad/dt, Eb ggl lawan dari jangkar dan T adalah torsi dalam N.m.

Kecepatan motor dc berbanding lurus dengan suplai tegangan, sehingga pengurangan suplai tegangan akan menurunkan kecepatan motor dan penambahan suplai tegangan akan menambah kecepatan motor.

III.2.1.1.2.a. Pengukuran kecepatan motor dc tanpa beban dengan Menggunakan Tachometer.

Tachometer merupakan alat yang digunakan untuk mengukur putaran motor dengan sensor kecepatan. Untuk lebih akuratnya data tersebut maka terlebih dahulu dianalisis pengukuran kecepatan motor Contoh pengaturan kecepatan motor dc, dimana kecepatan motor yang diberikan 1000 rpm, arus jangkar Ia = 0 pada beban nol, berapa tegangan pada beban nol ?

Ra = 7 Ohm hasil pengukuran

jadi :

ω = 1930 * 2 π / 60 = 202 rad / dt

$$Kb = \frac{Vt - Ia\,Ra}{\omega}$$

$$Kb = \frac{12 - 0{,}36\,(7)}{202} = \frac{12 - 2{,}52}{202} = 0{,}047 \text{ Volt dt /rad}$$

Pada kondisi tak berbeban kecepatan 1000 rpm :

ω = 1000 * 2 π / 60 = 104,7 rad / dt

$$\omega = \frac{Vt - IaRa}{Kb}$$

$$\omega = \frac{Vt - 0}{Kb}$$

$$104{,}7 = \frac{Vt - 0}{0{,}048}$$

Vt = 104,7 * 0,047

Vt = 4,92 Volt

Jadi misal Vt = 6 Volt , maka ω = 1219,68 rpm.

Tabel 14. Perbandingan hasil perhitungan dan pengukuran dengan Tachometer dalam keadaan tak berbeban.

Hasil Perhitungan		Hasil pengukuran dengan Tachometer	
Vt	ω (rpm)	Vt	ω (rpm)
3	609,84	3	610
4,5	914,71	4,5	914
6	1219,68	6	1219
7,5	1524,59	7,5	1524
9	1824,52	9	1830
12	2439,35	12	2440

Dari penyelesaian perhitungan contoh soal dapat disimpulkan bahwa penambahan tegangan dapat mempercepat kecepatan motor dc dan pengurangan tegangan dapat memperlambat kecepatan motor dc, dan hasil perhitungan dan pengukuran dengan tachometer mempunyai selisih.

Adaptor yang digunakan adalah Mitoyouri MTY – 999 AC/DC Adaptor. Input 220 V / AC 50 Hz, output 3: 4,5 : 6 : 7,5 : 9 : 12 VDC, arus 1200 mA dan kekuatan 18 Watt.

Tachometer yang digunakan adalah tachometer sensor Pravo RM 1000, Range 10 to 100,000 RPM, basic accuracy +/- 0,01 % +/ - dgt, rosulation 0,1 RPM, Sample Rate 1 Sec.Measuring distance 50 mm – 200 mm, made in Taiwan.

III.2.1.1.2.b. Pengukuran kecepatan motor dc berbeban dengan Menggunakan Tachometer.

Tachometer merupakan alat yang digunakan untuk mengukur putaran motor dengan sensor kecepatan. Untuk lebih akuratnya data tersebut maka terlebih dahulu dianalisis pengukuran kecepatan motor Contoh pengaturan kecepatan motor dc, dimana kecepatan motor yang diberikan 1000 rpm, arus jangkar Ia = 0,36 Amp. Ra = 7 Ohm pada beban penuh , berapa tegangan pada beban nol ?

Ra = 7 Ohm hasil pengukuran

jadi :

ω = 1930 * 2 π / 60 = 202 rad / dt

Kb = $\frac{Vt - Ia\,Ra}{\omega}$

Kb = $\frac{12 - 0{,}36\,(7)}{202}$ = $\frac{12 - 2{,}52}{202}$ = 0,047 Volt dt /rad

Pada kondisi berbeban kecepatan 1000 rpm :

ω = 1000 * 2 π / 60 = 104,7 rad / dt

ω = $\frac{Vt - IaRa}{Kb}$

ω = $\frac{Vt - 0{,}36\,(7)}{Kb}$

104,7 = $\frac{Vt - 2{,}52}{0{,}048}$

Vt – 2,52 = 104,7 * 0,047

Vt = 4,92 + 2,52 = 7,44 Volt

Jadi misal Vt = 7,5 Volt , maka ω = 1012,328 rpm.

Tabel 15. Perbandingan hasil perhitungan dan pengukuran dengan Tachometer dalam keadaan berbeban.

Hasil Perhitungan		**Hasil pengukuran dengan Tachometer**	
Vt	**ω (rpm)**	**Vt**	**ω (rpm)**
3	97,548	3	98
4,5	402,516	4,5	402
6	707,388	6	708
7,5	1012,328	7,5	1012
9	1317,248	9	1318
12	1927,089	12	1930

Dari penyelesaian perhitungan contoh soal dapat disimpulkan bahwa penambahan tegangan dapat mempercepat kecepatan motor dc dan pengurangan tegangan dapat memperlambat kecepatan motor dc, dan hasil perhitungan dan pengukuran dengan tachometer mempunyai selisih.

III.2.1.1.2.c. Pengaturan kecepatan motor dc dengan Menggunakan encoder.

Encoder yang digunakan sudah menyatu dengan motor DC sehingga penggunaannya lebih mudah dan praktis. Encoder akan memberikan 116 pulsa / putaran jika motor diputar 360 ^{0}, jika waktu pencuplikan 50 ms maka kecepatan motor dalam rpm dinyatakan :

$$rpm = \frac{\text{jumlah pulsa}}{116} \times \frac{1000}{Ts} \times 60$$

Dimana :

rpm = kecepatan putaran motor permenit

Ts = waktu pencuplikan dalam ms

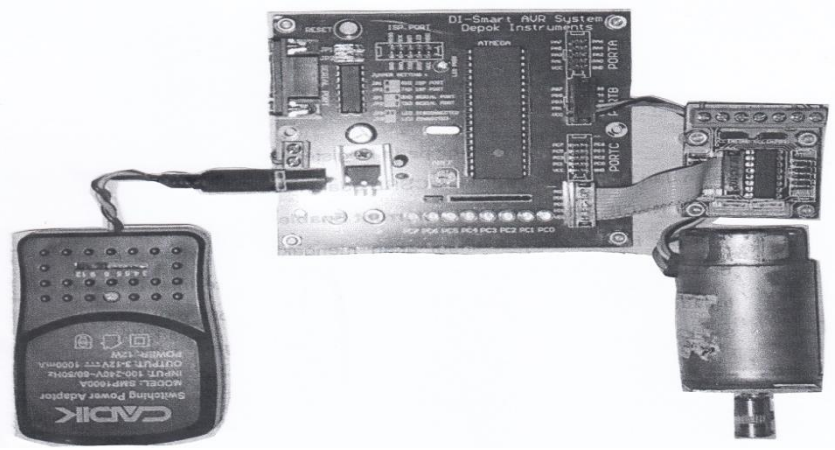
Contoh soal pengukuran dengan encoder, dimana jumlah pulsa yang diberikan 60 pulsa dengan waktu pencuplikan 50 ms, sehingga kecepatan motor dalam rpm adalah :

$$rpm = \frac{\text{jumlah pulsa}}{116} \times \frac{1000}{50} \times 60$$

$$rpm = \frac{60}{116} \times 20 \times 60$$

$$rpm = \frac{60 \; x \; 20 \; x \; 60}{116} = \frac{72000}{116} = 620{,}689$$

Rangkaian pengukuran dari sistem di atas dapat dilihat pada gambar III.1..



Gambar III.3. Pengukuran dengan encoder

Adaptor merupakan pemberi tegangan pada motor dc sehingga berputar sesuai dengan tegangan yang diberikan oleh adaptor, seed encoder mempunyai tiga kabel yaitu kabel input (+) berwarna orange yang dihubungkan dengan VCC pada port B sebagai

tegangan masukan ke encoder,kabel input (-) berwarna hitam yang dihubungkan dengan GND (ground) dan warna kuning merupakan output encoder dengan menghasilkan 116 pulsa / putaran yang dfihubungkan dengan port B.0. dan driver motor DC adalah pembangkit tegangan dan arus yang disalurkan ke motor DC.

Data output encoder yang dikirim ke mikrokontroller melalui port B.0. diterima dan diolah oleh mikrokontroller kemudian disalurkan ke PC melalui port serial dan USB RS – 232 dan hasil olahan yang diterima PC dapat kita lihat pada layar PC.

Tabel 16. Perbandingan hasil perhitungan dan hasil pengukuran dengan encoder

Hasil Perhitungan			Hasil Pengukuran dengan encoder		
Vt	ω (rpm)	Jumlah pulsa	Vt	ω (rpm)	Jumlah pulsa
3	609,84	58,95	3	620	60
4,5	914,71	88,42	4,5	920	89
6	1219,68	117,7	6	1220	118
7,5	1524,59	147,32	7,5	1531	148
9	1829,52	176,76	9	1831	177
12	2439,35	235,8	12	2440	236

Dari penyelesaian perhitungan contoh soal, perhitungan dan pengukuran dengan encoder dapat disimpulkan bahwa hasil pengukuran dan perhitungan mempunyai selisih.

Proses mikrokontroller dalam mengolah data dari encoder yaitu :

Misal : Sebuah motor dc berputar dengan kecepatan 1000 rpm, encoder memberi pulsa 116 pulsa / putaran, jadi jumlah pulsa yang dikirim ke PC adalah :

Jika kecepatan motor dc 1000 rpm, maka :

1 menit = 1000 putaran

1 putaran = 116 pulsa

Jumlah pulsa = 1000* 116

= 116.000 pulsa/menit

Maka jumlah pulsa yang dikirim ke PC = 116.000 pulsa/menit dalam bentuk hexadecimal.

Port B.0. mendapat input dari encoder yaitu nilai jumlah pulsa dengan jumla 116000 pulsa yang disalurkan ke mikrokontroller dan diterima oleh timer/counter0 untuk dihitung jumlah pulsa eksternal tersebut kemudian disimpan dalam register timer yaitu timer high dan timer low. Kemudian dibandingkan dengan jumlah pulsa clock, kalau jumlah pulsa yang ada dalam register tiner TH dan TL sama dengan jumlah pulsa clock maka sebuah interrupt akan terjadi sebagai tanda bahwa timer telah overflow (menjadi nol), maka sebuah timer flag akan bernilai 1 yang menandakan bahwa counter telah selesai menghitung dan flag tersebut bisa digunakan untuk meng-interrupt program.

Kemudian PC menerima data dari mikrokontroller yaitu jumlah pulsa. Misal jumlah pulsa yang diterima PC adalah 116000 pulsa, encoder memberi pulsa 116 pulsa / putaran, maka kecepatan yang dihasilkan PC adalah :

$$\omega = \frac{jumlah\ pulsa}{116} \text{ rpm}$$

$$\omega = \frac{116000}{116} = 1000 \text{ rpm}$$

Jadi kecepatan yang dihaasilkan PC adalah **1000** rpm

III.2.1.2. Perancangan Software

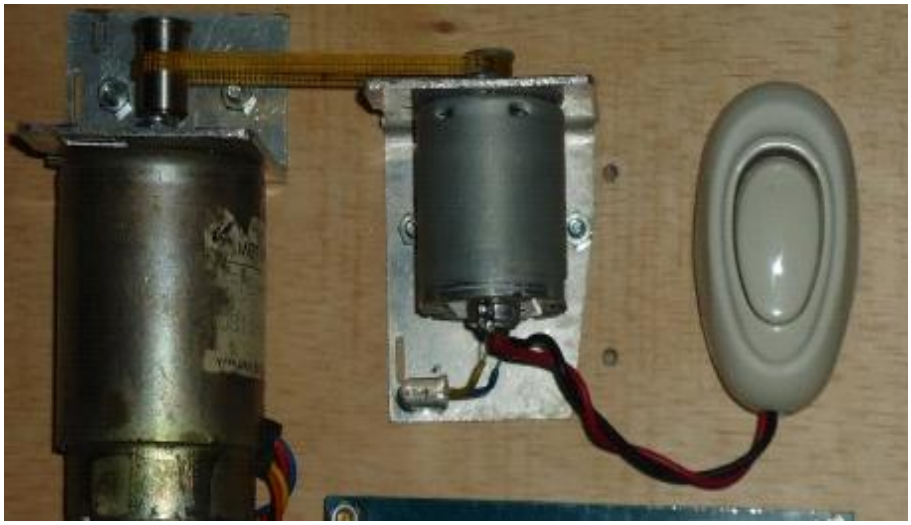
Perancangan perangkat lunak sistem kendali kecepatan motor dc berbasis PWM menggunakan dua bahasa pemrograman yaitu :

a. Bahasa untuk pengembangan programnya menggunakan Code Vision AVR, untuk proses *download* ke mikrokontroller penulis menggunakan IC USB to Serial TTL sehingga dapat pula digunakan sebagai anatarmuka komunikasi antar perangkat elektronika berlevel TTL.Proses *download* dengan menggunakan file hexa yang kemudian di *download* ke mikrokontroller melalui port USB. Setelah program di *download k*e mikrokontroller, maka mikrokontroller akan berfungsi sesuai program yang kita masukkan..
b. Bahasa pemrograman delphi 7, yang merupakan program utama pada sistem ini, yang dugunakan pada PC/laptop berfungsi menerima data dari mikrokontroller untuk mengatur kecepatan motor dc.

III.2.1.2.1. Perancangan perubahan beban

Untuk perancangan perubahan beban ketika motor berputar, pada terminal generator (beban) dihubungkan ke beban berupa led sehingga kecepatan motor turun akan tetapi masih mampu berputar.

Sistem tersebut digambarkan sebagai berikut



Gambar.III.4. motor dc dan beban serta led yang digunakan

III.2.1.2.2. Perancangan *Driver* motor dc

Untuk memberi catu daya ke motor dc diperlukan *driver* yang menerima masukan dari mikrokontroller dan keluarannya ke motor dc. *driver* motor dc yang dibangun menggunakan IC L293D yang didalamnya merupakan rangkaian *H – Bridge* yang menerima masukan level TTL dan mampu memberikan arus maksimum 600 mA dan tegangan maksimum 2 – 6,8 Vdc .Dengan IC ini maka keluaran dari mikrokontroller dapat langsung diberikan ke pin masukan untuk mengatur polaritas pencatudayaan motor dan sinyal PWM melalui pin *enable* di L293D. Input *driver* ini berupa tegangan 0 – 5 Vdc sedangkan keluaran berupa tegangan yang besarnya sesuai dengan tegangan referensi dan *duty cyclenya*.

Ketika diimplementasikan sebagai *driver* motor dc, konpigurasi pin yang digunakan dalam penelitian ini sebagai berikut :

III.2.3. Perancangan modul pencacah pulsa *encoder*

Modul pencacah pulsa *encoder* memanfaatkan pasilitas *timer* pada Atmega 8535. Modul pencacah pulsa *encoder* menggunakan *timer* 0. Cara kerja *timer* yang difungsikan adalah mengkonfigurasikan sebagai pencacah agar sumber *clock* dari eksternal yakni dengan mengkonfigurasikan nilai TCCR0, berikut nilai TCCR0 :

Bit	7	6	5	4	3	2	1	0	
$33 ($53)	–	–	–	–	–	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar.III.5. Register TCCR Timer 0

Dengan memberikan nilai pada TCCR0 = 0 x 07 maka timer 0 menggunakan sumber clock eksternal dan counter akan naik ketika sinyal naik (*rising edge)* secara fisik, sumber clock timer 0 berasal dari luar yakni melalui PB.0.

Encoder yang digunakan dalam penelitian ini mempunyai ketelitian 116 pulsa / rotasi sehingga jika waktu pencuplikan 50 ms, maka kecepatan motor dalam rpm dinyatakan :

$$RPM = \frac{\text{jumlah pulsa}}{116} \times \frac{1000}{Ts} \times 60 \text{ rpm}$$

RPM = kecepatan motor dalam rpm

Ts = waktu pencuplikan dalam milli detik

III.2.2.4. Perancangan *Timer*

Timer digunakan sebagai pewaktu proses pencacah jumlah pulsa *encoder* (*time sampling*) proses pewaktu ini dengan memanfaatkan *timer* 2. Konfigurasi *timer* 2

sebagai *timer dengan cara* mengkonfigurasi register TCCR2 dan TCNT2. Register TCNT2 adalah register yang digunakan untuk mengkonfigurasi *pre scalar timer.*

Bit	7	6	5	4	3	2	1	0	
$25 ($45)	–	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20	TCCR2
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar.III.6. Register TCCR2 Timer 2

Dengan memberikan nilai 0 x 07 pada TCCR2 berarti *pre scalar* yang dipilih 1024, yakni counter register TCNT2 akan naik setiap 1024 siklus clock. Besar timer sampling yang dipilih harus memperhatikan beberpa factor sehingga didapatkan desain sistem kendali yang optimal, Faktor-faktor untuk menentukan penetuan besar timer sampling tersebut adalah :

- Ketelitian *encoder*
- Besar *rise time*
- Besar *sampling time*

Semakin kecil *sampling time* menyebabkan ketelitian penentuan nilai set point menjadi berkurang. Hal ini dikarenakan pada penghitungan pulsa *encoder* pada *time sampling* rendah, kesalahan atau pembulatan 1 pulsa akan signifikan karena pulsa yang terkumpul sedikit, lain halnya pada time sampling yang besar maka jumlah pulsa yang terkumpul banyak sehingga kesalahan atau pembulatan 1 pulsa tidak terlalu signifikan terhadap hasil pengukuran. Jadi penetuan *time sampling* merupakan kompromi ketiga variabel besaran yang berpengaruh di atas. Berikut tabel data penetuan besar *time sampling* dengan ketelitian yang dihasilkan dalam penetua *set point .*

Tabel 17. Pengaruh t*ime sampling* terhadap ketelitian *set point.*

Time sampling (ms)	Ketelitian (rpm)
5	10,34
10	11,49
15	12,93
20	14,78
25	17,24
30	20,69
35	25,56
40	34,48
45	51,32
50	103,45

Berdasar pengukuran karakteristik motor dalam domain waktu dengan dengan masukan step pada set point 80 dimana besar rise time yakni waktu yang dibutuhkan untuk mencapai nilai sebesar study stste yang pertama kali kurang lebih sebesar 160 ms dan besar nilai setting time sebesar 200 ms, maka dipilih time sampling sebesar 50 ms dengan pertimbangan :

- Ketelitian *set point 103,45* rpm
- Terdapat 10 kali pengambilan data kecepatan sebelum motor mencapai range study state .

Jika waktu pencuplikan yang diinginkan sebesar 50 ms, maka harus ditentukan mulai dari nilai berapa register TCNT2 bertambah sampai terjadi *overflow.* Dengan *pre scalar* 1024 dan *clock osilator* sebesar 11059200 berarti bahwa register TCNT2 akan naik 1024 x (1 / 11059200) atau sekitar 9,26 x 10^{-5} detik, sehingga dibutuhkan 216 siklus untuk mendapatkan nilai *timer* 50 ms. Karena nilai maksimum TCNT2 256 maka harus di set bahwa nilai terendahnya adalah 240 (256 – 216) atau 0 x 28. jadi secara keseluruhan konfigurasi *timer* 2 adalah : TCCR2 = 0 x 07 dan TCCR2 = 0x28

III.2.1.2.3. Perancangan PWM

Untuk mengkonfigurasikan *timer* 1 sebagai fash PWM 8 bit maka register TCCR1A diberi nilai 0 x A1, sedangkan untuk menjalankan pada *pre scalar 64,* maka TCCR1B diberi nilai 0 x 0B. Kemudian TCNT1 juga perlu diinisialisasi dengan nilai 0x00 sehingga secara keseluruhan konfigurasi PWM sebagai berikut :

TCCR1A = 0 x A1, TCCR1B = 0 x 0B, TCNT1 = 0 x 000. Besar kecilnya duty cycle PWM adalah perbandingan nilai TCNT1 dengan nilai maksimum *counter* sehingga pada mode fash PWM 8 bit maka duty cycle adalah TCNT1/255 .

B. Lokasi dan Waktu Penelitian

Penelitian ini dilaksanakan selama enam bulan yakni bulan Januari sampai dengan bulan juni 2011. Lokasi penelitian dilakukan pada Lab. Teknik Elektro Universitas Hasanuddin Makassar.

C. Pengujian Sistem

Pengujian sistem dilakukan untuk mengetahui tingkat keberhasilan dari sistem yang dibangun, dalam penelitian ini pengujian akan dilakukan dengan

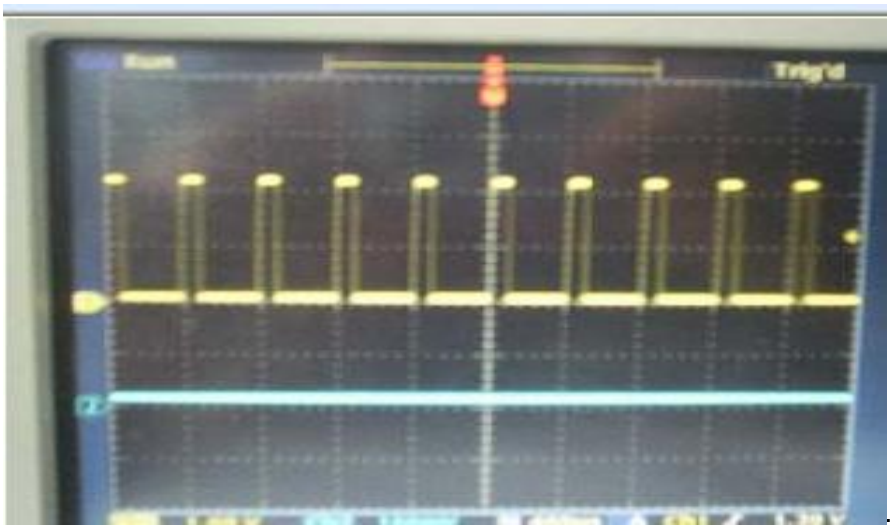
menggunakan mekanisme PWM (Pulse Width Modulation) yaitu pembangkit sinyal keluaran.

BAB IV
HASIL DAN PEMBAHASAN

Setelah proses perancangan, dilakukan analisis dan pengujian untuk mengukur tingkat keberhasilan perancangan yang telah dilakukan. Pengujian dilakukan permodul, setelah modul-modul diuji, kemudian diintegrasikan dan dilakukan pengujian sistem secara keseluruhan.

IV.1. Pengujian pulsa *encoder*

pengujian pulsa *encoder* digunakan untuk mengamati pulsa yang dihasilkan oleh *encoder* yang kemudian dihitung jumlahnya. Amplitudo pulsa *encoder* sebesar 2 Volt. Semakin cepat putaran motor, maka frekuensi pulsa akan semakin tinggi. Pada saat pulsa *encoder rising edge* akan mengakibatkan register di TCNTO bertambah, kemudian setelah selang 50 ms kecepatan dihitung dengan melihat jumlah pulsa yang nilainya sama dengan nilai yang ada di register TCNTO. Nilai TNCTO inilah yang diambil sebagai kecepatan motor dan diambil sebagai nilai umpan balik ke sistem kendali. *Encoder* yang digunakan sudah menyatu dengan motor sehingga penggunaannya lebih mudah dan praktis. *Encoder* akan memberikan 116 pulsa/ putaran jika motor diputar 360^0 .

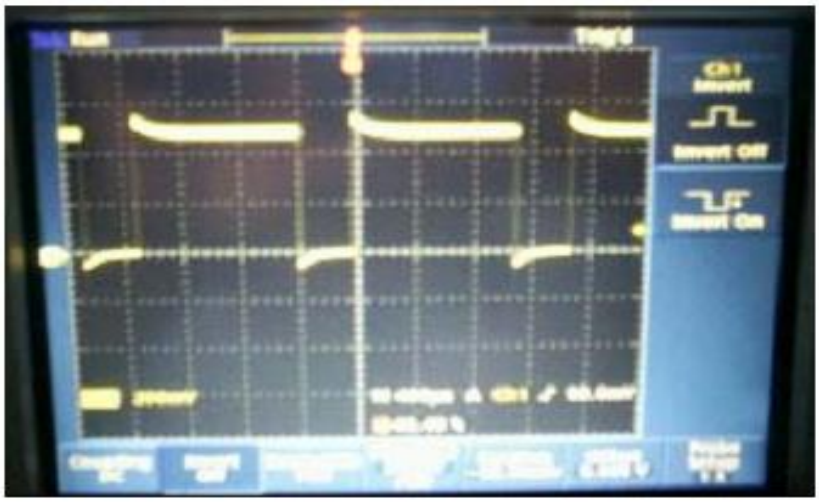


Gambar IV.1. Pengujian pulsa *encoder*

IV.2. Pengujian pulsa PWM dari Mikrontroller

Pulsa PWM yang keluar dari mikrokontroller mendekati ideal, yakni berupa sinyal kotak yang prosentase sinyal high dengan keseluruhan sinyal merupakan besar *duty cycle* yang dihasilkan.

Gambar IV.2. merupakan sinyal bentuk PWM dengan duty cycle 75%, dari gambar terlihat perbandingan antara sinyal high dan sinyal low adalah 3 : 1.

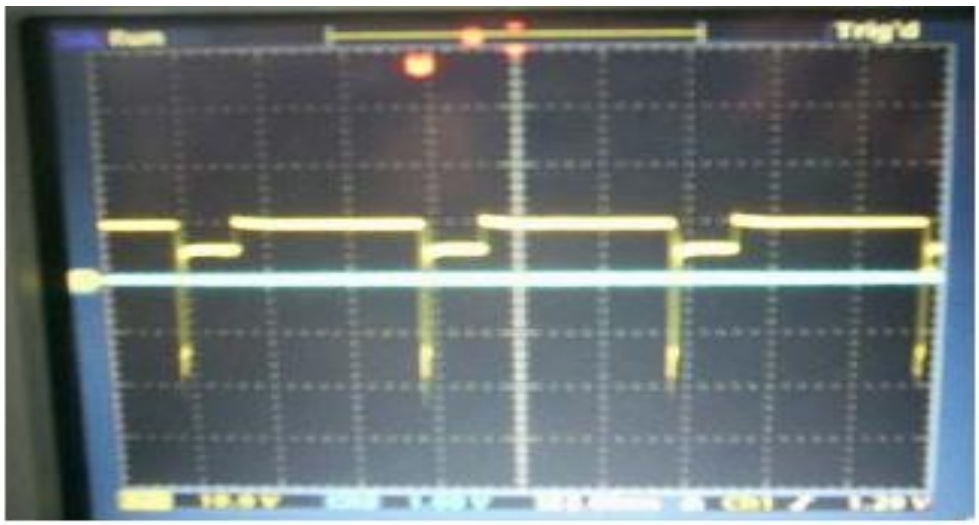


Gambar IV.2. Pengujian pulsa PWM dari mikrokontroller

IV.3. Pengujian pulsa PWM dari *Driver Motor*

Bentuk pulsa PWM yang keluar dari *driver motor* DC tidak semulus yang keluar dari mikrokontroller, hal ini karena pengaruh *driver motor* dan motor ketika kondisi high ke low tidak ideal. Tegangan rata-rata yang dihasilkan sebesar :

(duty cycke x V referensi} – Drop tegangan



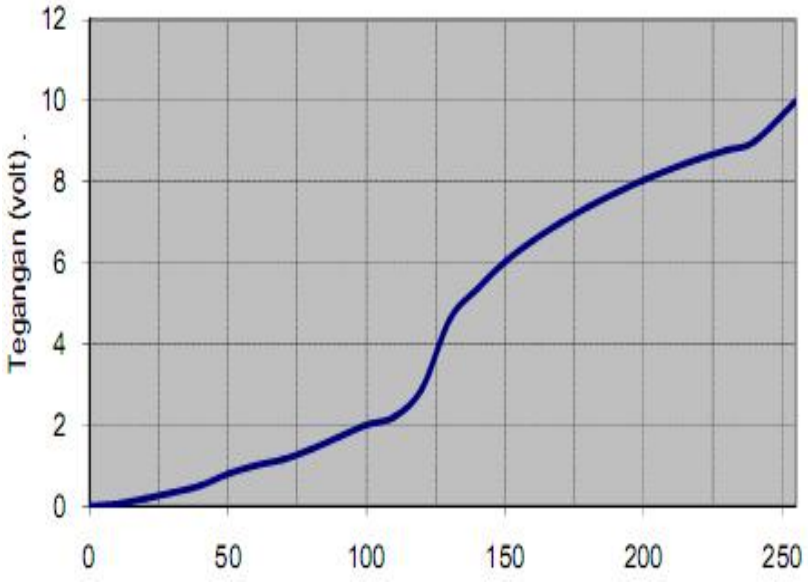
Gambar IV.3. Pengujian pulsa PWM dari driver motor DC

Gambar di atas merupakan bentuk sinyal PWM dengan *duty cycle* 75%, dari gambar terlihat perbandingan antara sinyal high dan sinyal low 3:1.

IV.4. Pengujian tegangan keluaran driver terhadap masukan PWM

Pengukuran tegangan keluaran driver sebagai catu daya motor dengan masukan berupa nilau PWM 8 bit dengan *preskalar* 64 yang dikendalikan oleh mikrokontroller.hasil pengukuran menghasilkan data sebagai berikut :

Hubungan PWM – tegangan keluaran driver



Gambar IV.4. Hubungan PWM dengan tegangan keluaran driver

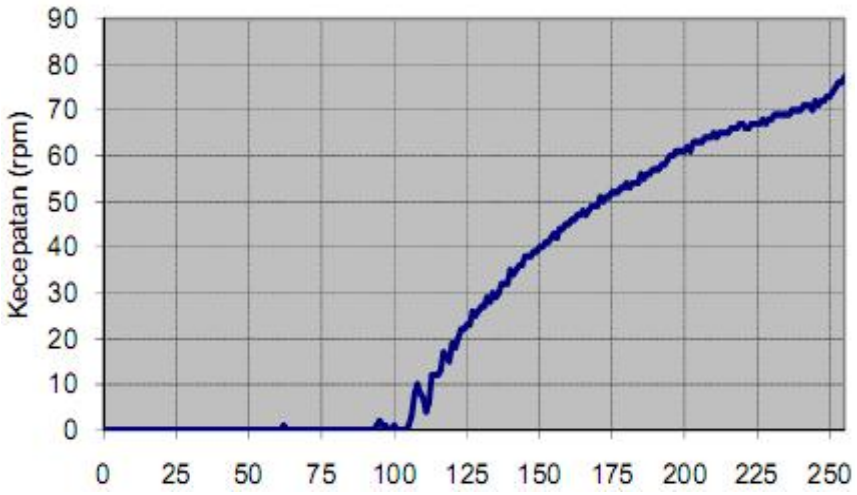
Tampak dari grafik bahwa hubungan nilai PWM dengan tegangan keluaran :

- Tidak linier, karena karakteristik bahan IC L293N
- Terjadi drop tegangan dimana tegangan referensi yang dihasilkan sebesar 12 V, akan tetapi dengan nilai PWM maksimal sebesar 255 hanya menghasilkan tegangan 10 V,hal ini karena karakteristik IC L298N sebagaimana diakibatkan dalam datasheet bahwa akan terjadi drop tegangan sebesar 1,8 – 3,2 Volt.

IV.5. Pengujian kecepatan motor berbeban terhadap masukan PWM

Pengukuran kecepatan motor dengan masukan PWM 8 bit dengan preskalar 64 pada sistem ini dengan waktu pencuplikan 50 ms menghasilkan data sebagai berikut :

Hubungan PWM – Kecepatan motor berbeban



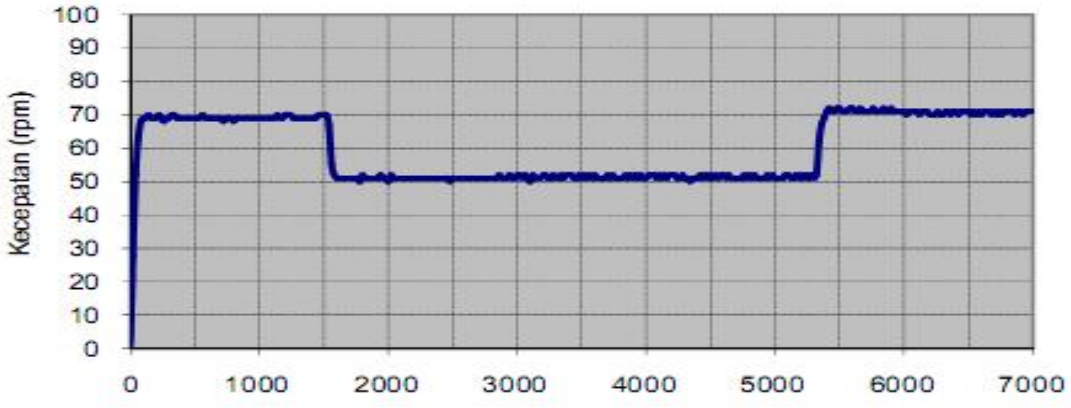
Gambar IV.5. Hubungan PWM dengan kecepatan motor berbeban

Data di atas dan data pengukuran-pengukuran selanjutnya dinyatakan dalam pencuplikan 50 ms dengan 116 pulsa/rotasi sehingga kecepatan dalam rpm adalah jumlah pulsa x 103,45 rpm. Untuk selanjutnya *preskalar* PMW yang digunakan dalam perancangan penelitian sebesar 64.

IV.6. Pengujian ketika terjadi gangguan pada sistem kendali PWM

Untuk melihat efek kendali PWM ketika terjadi perubahan beban, terlebih dahulu diuji coba perubahan beban pada sistem *loop* terbuka. Perubahan beban diberikan dengan menghubungkan generator ke LED. Hasil pengujian pada sistem ketika terjadi gangguan menghasilkan data sebagai berikut :

Pengujian ketika terjadi gangguan pada sistem



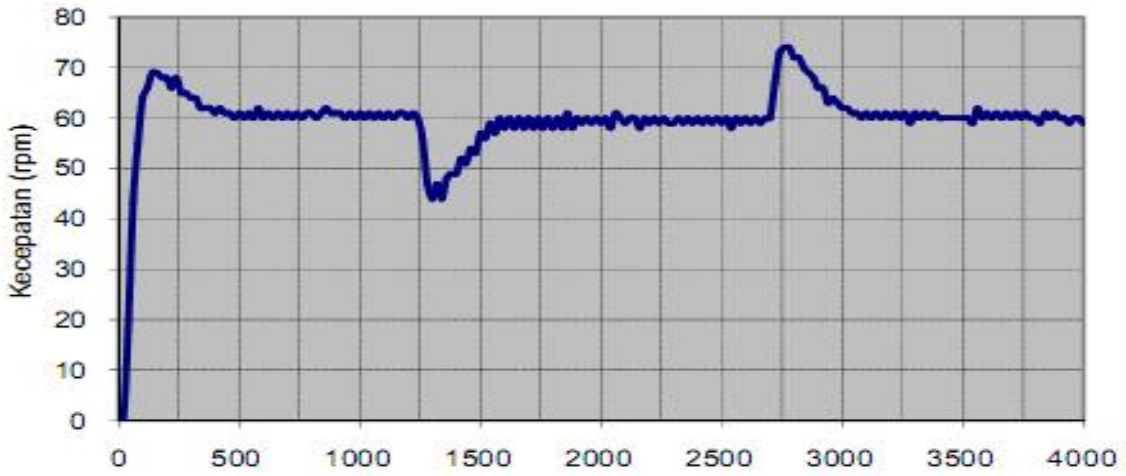
Waktu (ms)

Gambar IV.6. Pengujian gangguan pada *set point* 70

Dalam sistem ini, ketika terjadi gangguan kecepatan menurun seperti terlihat dalam grafik di atas. Ketika beban dikembalikan ke kondisi semula maka kecepatan kembali seperti pada saat sebelum terjadi penambahan beban.

Adapun setelah sistem diimplementasikan, maka hasil pengamatan terhadap penambahan beban dengan catu daya 12 Volt di tunjukkan dalam grafik berikut :

Pengujian perubahan beban pada SP 60

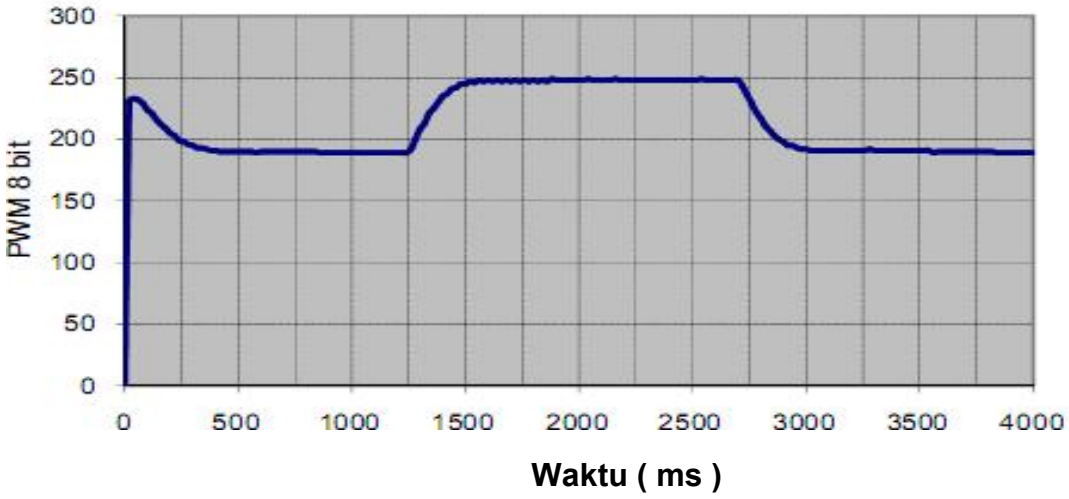


Waktu (ms)

Gambar .IV.7. Pengujian perubahan beban pada *set point* 60

Ketika terjadi perubahan beban, maka sistem memberikan kompensasi untuk menjaga kecepatan agar tetap pada nilai *set point,* perubahan *duty cycle* ditunjukkan dalam gambar berikut ini.

Perubahan *duty cycle* ketika terjadi perubahan beban pada *SP* 60

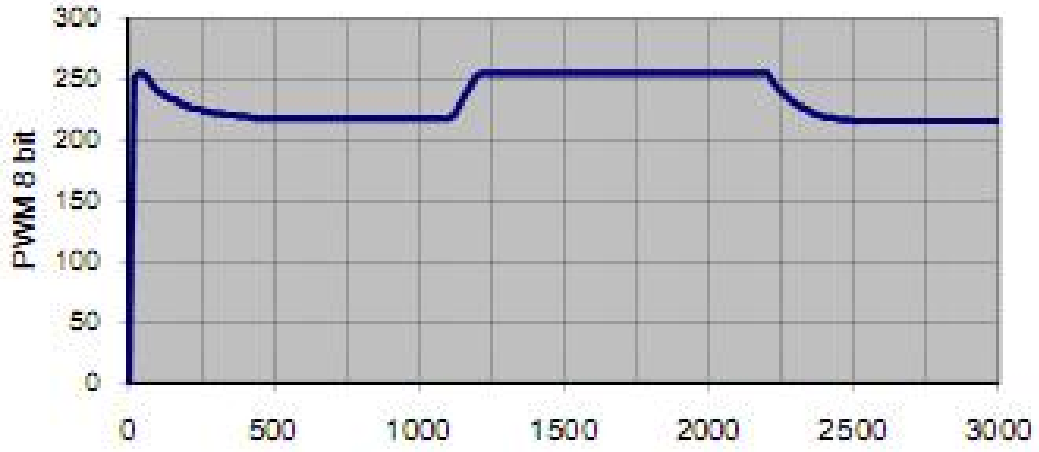


Gambar IV.8. Perubahan *duty cycle* ketika perubahan beban pada *set point* 60

Dari grafik di atas dapat disimpulkan bahwa sistem yang telah di bangun mampu menjaga agar kecepatan berada pada nilai *set point* walaupun terjadi perubahan beban. Pengujian juga dilakukan dengan variasi nilai *set point,* pengujian perubahan beban pada *set point* 70 dengan catu daya 12 Volt menghasilkan data sebagai berikut :

Berikut sinyal kompensasi ketika terjadi perubahan beban, nilai *duty cycle* bertambah ketika terjadi perubahan beban.

Perubahan *duty cycle* ketika terjadi perubahan beban pada *SP* 70



Gambar IV.9. Perubahan *duty cycle* ketika terjadi perubahan beban pada set point 70

Pada *set point* 70 terlihat bahwa sistem sudah tidak mampu menjaga kecepatan sesuai set point, hal ini terjadi karena catu daya sudah tidak mampu mengkompensasi perubahan beban, walaupun sudah diberikan catu daya maksimal berupa *duty cycle* 100% tetapi karena bebannya terlalu berat, maka sistem sudah tidak mampu menjaga nilai sesuai *set point* . Jadi sistem mampu menjaga nilai sesuai *set point* ketika terjadi perubahan beban selama kompensasi mampu mengimbangi perubahan beban . Hal ini berkaitan erat dengan kemampuan catu daya yang diberikan pada tegangan referensi *dreiver* motor.

BAB V
KESIMPULAN DAN SARAN

Setelah semua proses perancangan dan pengujian dilakukan, didapatkan beberapa kesimpulan dan saran-saran bagi yang akan memanfaatkan dan mengembangkan penelitian ini lebih lanjut.

V.1 Kesimpulan

1. Sistem kendali kecepatan motor DC berbasis PWM telah berhasil dibuat sehingga mampu mengkompensasi perubahan beban.
2. Dengan pencatudayaan PWM akan lebih hemat karena duty cycle otomatis menyesuaikan tanpa ada daya yang dibuang ke transistor seperrti pada catu daya analog.
3. Besar *rise time* dan *setting time* pada kendali PWM tergantung nilai *set pointnya* (jumlah pulsa) yang diberikan, semakin tinggi semakin cepat mencapai *rise time.* Kendali PWM kecepatan motor dc ini mampu meningkatkan *rise time* akan tetapi mengalami penurunan pada *setting.*
4. Besar *rise time* dan *settling* time pada sistem kendali tergantung nilai *set point*nya, semakin tinggi set pointnya semakin cepat mencapai *rise time .*
5. Kendali kecepatan motor DC ini mampu meningkatkan *rise time* akan tetapi mengalami penurunan pada *settling time.*
6. Ketidaklinieran *plant* tidak memberikan pengaruh yang signifikan pada sistem kendali kecepatan motor dc.

Kemudian jika ingin mendapatkan nilai yang paling baik dengan

cara mencoba-coba sampai didapatkan nilai yang dianggap telah mencukupi kebutuhan sistem yang dirancang.

V.2 Saran

Sistem yang dibangun masih berupa prototype dan belum dapat diaplikasikan secara langsung pada objek tertentu dilapangan , maka kami sarankan untuk pengembangan sistem ini agar menjadi lebih aplikatif dengan melakukan penambahan atau penggunaan komponen yang kapasitasnya lebih besar dan lebih tahan terhadap panas saat dioperasikan .

DAFTAR PUSTAKA

[1] Mengontrol kecepatan motor dc jarak jauh www.scribd.com/doc/400/4036 (diakses :Januari 2012)

[2] Peter Christionto, Pengaturan Kecepatan Motor DC dengan Adaptif Fuzzy Logic Control. WWW.Citenlike.org /tag/fuzzy – logic (diakses : januari 2012)

[3] Fuji Hartono, Analisis Pengendali Kecepatan Motor DC menggunakan Metode Logika Fuzzy WWW. te .ITB.ac. id / Hartono. (diakses : Maret 2011)

[4] lammert, USB to RS 232 Converter WWW. Lammertbies.nl/comm./info/RS-232-usb html (diakses : Januari 2012)

[5] Wikipedia,USART en.wikipedia.org/wiki/universal_asynchronous_receiver / transmitter (diakses : januari 2012)

[6] Lingga Wardhana. 2006. Belajar Sendiri Mikrokontroler AVR Seri ATMega 8535 Simulasi, Hardware dan Aplikasi. Andi,Yogyakarta.

[7] Wikipedia, PWM (Pulse Width Modulation) en.wikipedia.org/wiki/pulse – width – modulation (diakses : januari 2012)

[8] Insan, *H – Bridge* control arah gerak motor, http://Insansainsproject.wordpress.com/2008/06/05/h-bridge- kontrol-arah-motor (diakses : maret 20011)

[9] Driver, Book Spesification L293D,SGS – THOMSON Microelectronics.

[10] Spesifikasi motor dc www.pricetransformer.com/control/448873/1/NSK - Ball - Screm – Ballscrem diakses : januari 2012

[11] Spesifikasi generator dc http: // www.gws.com.tw/english/product/servo/standart .htm (diakses : januari 2012)

[12] Jeff Duntemann. 1992. Assembly Language step by step. John Willey and Sons Inc.

[13] Muhammad Ali Mazidi. The 8051 Microcontroller & Embedded Systems Using assembly and C. Prentice-Hall Inc.

[14] Usman. 2008. Teknik Antar Muka + Pemrograman Mikrokontroler AT89S52. Andi, Yogyakarta..

[15] Romy Budi Widodo. 2009. Embedded System, Menggunakan Mikrokontroler dan Pemrograman C. Andi, Yogyakarta.

[16] Wahana Komputer. 2009. Aplikasi Cerdas Menggunakan Delphi. Andi, Yogyakarta.

[17] Widodo Budiharto 2005, Perancangan Sistem dan Aplikasi Mikrokontroller,PT. Elex Media Komputindo, Jakarta.
[18] AVR InstructionSet, http://www.atmel.com, diakses januari 2012.

LAMPIRAN - LAMPIRAN

LAMPIRAN 1

Listing Program Bahasa assambly untuk Mikrokontroller
AVR Atmega 8535

Listing program Code Vision AVR Atmega 8535

```
;CodeVisionAVR C Compiler V2.03.9 Standard
;(C) Copyright 1998-2008 Pavel Haiduc, HP InfoTech s.r.l.
;http://www.hpinfotech.com

;Chip type           : ATmega8535
;Program type        : Application
;Clock frequency     : 4,000000 MHz
;Memory model        : Small
;Optimize for        : Size
;(s)printf features  : int, width
;(s)scanf features   : int, width
;External RAM size   : 0
;Data Stack size     : 128 byte(s)
;Heap size           : 0 byte(s)
;Promote char to int : No
;char is unsigned    : Yes
;global const stored in FLASH  : Yes
;8 bit enums         : Yes
;Enhanced core instructions    : On
;Smart register allocation : On
;Automatic register allocation : On
```

```
#pragma AVRPART ADMIN PART_NAME ATmega8535
#pragma AVRPART MEMORY PROG_FLASH 8192
#pragma AVRPART MEMORY EEPROM 512
#pragma AVRPART MEMORY INT_SRAM SIZE 512
#pragma AVRPART MEMORY INT_SRAM START_ADDR 0x60

.LISTMAC
.EQU UDRE=0x5
.EQU RXC=0x7
.EQU USR=0xB
.EQU UDR=0xC
.EQU SPSR=0xE
.EQU SPDR=0xF
.EQU EERE=0x0
.EQU EEWE=0x1
.EQU EEMWE=0x2
.EQU EECR=0x1C
.EQU EEDR=0x1D
.EQU EEARL=0x1E
.EQU EEARH=0x1F
.EQU WDTCR=0x21
.EQU MCUCR=0x35
.EQU GICR=0x3B
```

```
.EQU SPL=0x3D
.EQU SPH=0x3E
.EQU SREG=0x3F

.DEF R0X0=R0
.DEF R0X1=R1
.DEF R0X2=R2
.DEF R0X3=R3
.DEF R0X4=R4
.DEF R0X5=R5
.DEF R0X6=R6
.DEF R0X7=R7
.DEF R0X8=R8
.DEF R0X9=R9
.DEF R0XA=R10
.DEF R0XB=R11
.DEF R0XC=R12
.DEF R0XD=R13
.DEF R0XE=R14
.DEF R0XF=R15
.DEF R0X10=R16
.DEF R0X11=R17
.DEF R0X12=R18
.DEF R0X13=R19
.DEF R0X14=R20
```

```
.DEF R0X15=R21
.DEF R0X16=R22
.DEF R0X17=R23
.DEF R0X18=R24
.DEF R0X19=R25
.DEF R0X1A=R26
.DEF R0X1B=R27
.DEF R0X1C=R28
.DEF R0X1D=R29
.DEF R0X1E=R30
.DEF R0X1F=R31

.MACRO __CPD1N
CPI  R30,LOW(@0)
LDI  R26,HIGH(@0)
CPC  R31,R26
LDI  R26,BYTE3(@0)
CPC  R22,R26
LDI  R26,BYTE4(@0)
CPC  R23,R26
.ENDM

.MACRO __CPD2N
CPI  R26,LOW(@0)
LDI  R30,HIGH(@0)
```

```
CPC  R27,R30
LDI  R30,BYTE3(@0)
CPC  R24,R30
LDI  R30,BYTE4(@0)
CPC  R25,R30
.ENDM

.MACRO __CPWRR
CP   R@0,R@2
CPC  R@1,R@3
.ENDM

.MACRO __CPWRN
CPI  R@0,LOW(@2)
LDI  R30,HIGH(@2)
CPC  R@1,R30
.ENDM

.MACRO __ADDB1MN
SUBI R30,LOW(-@0-(@1))
.ENDM

.MACRO __ADDB2MN
SUBI R26,LOW(-@0-(@1))
.ENDM
```

```
.MACRO __ADDW1MN
SUBI R30,LOW(-@0-(@1))
SBCI R31,HIGH(-@0-(@1))
.ENDM

.MACRO __ADDW2MN
SUBI R26,LOW(-@0-(@1))
SBCI R27,HIGH(-@0-(@1))
.ENDM

.MACRO __ADDW1FN
SUBI R30,LOW(-2*@0-(@1))
SBCI R31,HIGH(-2*@0-(@1))
.ENDM

.MACRO __ADDD1FN
SUBI R30,LOW(-2*@0-(@1))
SBCI R31,HIGH(-2*@0-(@1))
SBCI R22,BYTE3(-2*@0-(@1))
.ENDM

.MACRO __ADDD1N
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
```

```
SBCI R22,BYTE3(-@0)
SBCI R23,BYTE4(-@0)
.ENDM

.MACRO __ADDD2N
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
SBCI R24,BYTE3(-@0)
SBCI R25,BYTE4(-@0)
.ENDM

.MACRO __SUBD1N
SUBI R30,LOW(@0)
SBCI R31,HIGH(@0)
SBCI R22,BYTE3(@0)
SBCI R23,BYTE4(@0)
.ENDM

.MACRO __SUBD2N
SUBI R26,LOW(@0)
SBCI R27,HIGH(@0)
SBCI R24,BYTE3(@0)
SBCI R25,BYTE4(@0)
.ENDM
```

```
.MACRO __ANDBMNN
LDS  R30,@0+@1
ANDI R30,LOW(@2)
STS  @0+@1,R30
.ENDM

.MACRO __ANDWMNN
LDS  R30,@0+@1
ANDI R30,LOW(@2)
STS  @0+@1,R30
LDS  R30,@0+@1+1
ANDI R30,HIGH(@2)
STS  @0+@1+1,R30
.ENDM

.MACRO __ANDD1N
ANDI R30,LOW(@0)
ANDI R31,HIGH(@0)
ANDI R22,BYTE3(@0)
ANDI R23,BYTE4(@0)
.ENDM

.MACRO __ANDD2N
ANDI R26,LOW(@0)
ANDI R27,HIGH(@0)
```

```
ANDI R24,BYTE3(@0)
ANDI R25,BYTE4(@0)
.ENDM

.MACRO __ORBMNN
LDS  R30,@0+@1
ORI  R30,LOW(@2)
STS  @0+@1,R30
.ENDM

.MACRO __ORWMNN
LDS  R30,@0+@1
ORI  R30,LOW(@2)
STS  @0+@1,R30
LDS  R30,@0+@1+1
ORI  R30,HIGH(@2)
STS  @0+@1+1,R30
.ENDM

.MACRO __ORD1N
ORI  R30,LOW(@0)
ORI  R31,HIGH(@0)
ORI  R22,BYTE3(@0)
ORI  R23,BYTE4(@0)
.ENDM
```

```
    .MACRO __ORD2N
    ORI  R26,LOW(@0)
    ORI  R27,HIGH(@0)
    ORI  R24,BYTE3(@0)
    ORI  R25,BYTE4(@0)
    .ENDM

    .MACRO __DELAY_USB
    LDI  R24,LOW(@0)
__DELAY_USB_LOOP:
    DEC  R24
    BRNE __DELAY_USB_LOOP
    .ENDM

    .MACRO __DELAY_USW
    LDI  R24,LOW(@0)
    LDI  R25,HIGH(@0)
__DELAY_USW_LOOP:
    SBIW R24,1
    BRNE __DELAY_USW_LOOP
    .ENDM

    .MACRO __GETD1S
    LDD  R30,Y+@0
```

```
LDD  R31,Y+@0+1
LDD  R22,Y+@0+2
LDD  R23,Y+@0+3
.ENDM

.MACRO __PUTD1S
STD  Y+@0,R30
STD  Y+@0+1,R31
STD  Y+@0+2,R22
STD  Y+@0+3,R23
.ENDM

.MACRO __PUTD2S
STD  Y+@0,R26
STD  Y+@0+1,R27
STD  Y+@0+2,R24
STD  Y+@0+3,R25
.ENDM

.MACRO __POINTB1MN
LDI  R30,LOW(@0+@1)
.ENDM

.MACRO __POINTW1MN
LDI  R30,LOW(@0+@1)
```

```
LDI  R31,HIGH(@0+@1)
.ENDM

.MACRO __POINTD1M
LDI  R30,LOW(@0)
LDI  R31,HIGH(@0)
LDI  R22,BYTE3(@0)
LDI  R23,BYTE4(@0)
.ENDM

.MACRO __POINTW1FN
LDI  R30,LOW(2*@0+@1)
LDI  R31,HIGH(2*@0+@1)
.ENDM

.MACRO __POINTD1FN
LDI  R30,LOW(2*@0+@1)
LDI  R31,HIGH(2*@0+@1)
LDI  R22,BYTE3(2*@0+@1)
LDI  R23,BYTE4(2*@0+@1)
.ENDM

.MACRO __POINTB2MN
LDI  R26,LOW(@0+@1)
.ENDM
```

```
.MACRO __POINTW2MN
LDI  R26,LOW(@0+@1)
LDI  R27,HIGH(@0+@1)
.ENDM

.MACRO __POINTBRM
LDI  R@0,LOW(@1)
.ENDM

.MACRO __POINTWRM
LDI  R@0,LOW(@2)
LDI  R@1,HIGH(@2)
.ENDM

.MACRO __POINTBRMN
LDI  R@0,LOW(@1+@2)
.ENDM

.MACRO __POINTWRMN
LDI  R@0,LOW(@2+@3)
LDI  R@1,HIGH(@2+@3)
.ENDM

.MACRO __POINTWRFN
```

```
LDI  R@0,LOW(@2*2+@3)
LDI  R@1,HIGH(@2*2+@3)
.ENDM

.MACRO __GETD1N
LDI  R30,LOW(@0)
LDI  R31,HIGH(@0)
LDI  R22,BYTE3(@0)
LDI  R23,BYTE4(@0)
.ENDM

.MACRO __GETD2N
LDI  R26,LOW(@0)
LDI  R27,HIGH(@0)
LDI  R24,BYTE3(@0)
LDI  R25,BYTE4(@0)
.ENDM

.MACRO __GETD2S
LDD  R26,Y+@0
LDD  R27,Y+@0+1
LDD  R24,Y+@0+2
LDD  R25,Y+@0+3
.ENDM
```

```
.MACRO __GETB1MN
LDS  R30,@0+@1
.ENDM

.MACRO __GETB1HMN
LDS  R31,@0+@1
.ENDM

.MACRO __GETW1MN
LDS  R30,@0+@1
LDS  R31,@0+@1+1
.ENDM

.MACRO __GETD1MN
LDS  R30,@0+@1
LDS  R31,@0+@1+1
LDS  R22,@0+@1+2
LDS  R23,@0+@1+3
.ENDM

.MACRO __GETBRMN
LDS  R@0,@1+@2
.ENDM

.MACRO __GETWRMN
```

```
LDS  R@0,@2+@3
LDS  R@1,@2+@3+1
.ENDM

.MACRO __GETWRZ
LDD  R@0,Z+@2
LDD  R@1,Z+@2+1
.ENDM

.MACRO __GETD2Z
LDD  R26,Z+@0
LDD  R27,Z+@0+1
LDD  R24,Z+@0+2
LDD  R25,Z+@0+3
.ENDM

.MACRO __GETB2MN
LDS  R26,@0+@1
.ENDM

.MACRO __GETW2MN
LDS  R26,@0+@1
LDS  R27,@0+@1+1
.ENDM
```

```
.MACRO __GETD2MN
LDS  R26,@0+@1
LDS  R27,@0+@1+1
LDS  R24,@0+@1+2
LDS  R25,@0+@1+3
.ENDM

.MACRO __PUTB1MN
STS  @0+@1,R30
.ENDM

.MACRO __PUTW1MN
STS  @0+@1,R30
STS  @0+@1+1,R31
.ENDM

.MACRO __PUTD1MN
STS  @0+@1,R30
STS  @0+@1+1,R31
STS  @0+@1+2,R22
STS  @0+@1+3,R23
.ENDM

.MACRO __PUTB1EN
LDI  R26,LOW(@0+@1)
```

```
LDI  R27,HIGH(@0+@1)
RCALL __EEPROMWRB
.ENDM

.MACRO __PUTW1EN
LDI  R26,LOW(@0+@1)
LDI  R27,HIGH(@0+@1)
RCALL __EEPROMWRW
.ENDM

.MACRO __PUTD1EN
LDI  R26,LOW(@0+@1)
LDI  R27,HIGH(@0+@1)
RCALL __EEPROMWRD
.ENDM

.MACRO __PUTBR0MN
STS  @0+@1,R0
.ENDM

.MACRO __PUTDZ2
STD  Z+@0,R26
STD  Z+@0+1,R27
STD  Z+@0+2,R24
STD  Z+@0+3,R25
```

```
.ENDM

.MACRO __PUTBMRN
STS  @0+@1,R@2
.ENDM

.MACRO __PUTWMRN
STS  @0+@1,R@2
STS  @0+@1+1,R@3
.ENDM

.MACRO __PUTBZR
STD  Z+@1,R@0
.ENDM

.MACRO __PUTWZR
STD  Z+@2,R@0
STD  Z+@2+1,R@1
.ENDM

.MACRO __GETW1R
MOV  R30,R@0
MOV  R31,R@1
.ENDM
```

```
.MACRO __GETW2R
MOV  R26,R@0
MOV  R27,R@1
.ENDM

.MACRO __GETWRN
LDI  R@0,LOW(@2)
LDI  R@1,HIGH(@2)
.ENDM

.MACRO __PUTW1R
MOV  R@0,R30
MOV  R@1,R31
.ENDM

.MACRO __PUTW2R
MOV  R@0,R26
MOV  R@1,R27
.ENDM

.MACRO __ADDWRN
SUBI R@0,LOW(-@2)
SBCI R@1,HIGH(-@2)
.ENDM
```

```
.MACRO __ADDWRR
ADD  R@0,R@2
ADC  R@1,R@3
.ENDM

.MACRO __SUBWRN
SUBI R@0,LOW(@2)
SBCI R@1,HIGH(@2)
.ENDM

.MACRO __SUBWRR
SUB  R@0,R@2
SBC  R@1,R@3
.ENDM

.MACRO __ANDWRN
ANDI R@0,LOW(@2)
ANDI R@1,HIGH(@2)
.ENDM

.MACRO __ANDWRR
AND  R@0,R@2
AND  R@1,R@3
.ENDM
```

```
.MACRO __ORWRN
ORI  R@0,LOW(@2)
ORI  R@1,HIGH(@2)
.ENDM

.MACRO __ORWRR
OR   R@0,R@2
OR   R@1,R@3
.ENDM

.MACRO __EORWRR
EOR  R@0,R@2
EOR  R@1,R@3
.ENDM

.MACRO __GETWRS
LDD  R@0,Y+@2
LDD  R@1,Y+@2+1
.ENDM

.MACRO __PUTWSR
STD  Y+@2,R@0
STD  Y+@2+1,R@1
.ENDM
```

```
.MACRO __MOVEWRR
MOV  R@0,R@2
MOV  R@1,R@3
.ENDM

.MACRO __INWR
IN   R@0,@2
IN   R@1,@2+1
.ENDM

.MACRO __OUTWR
OUT  @2+1,R@1
OUT  @2,R@0
.ENDM

.MACRO __CALL1MN
LDS  R30,@0+@1
LDS  R31,@0+@1+1
ICALL
.ENDM

.MACRO __CALL1FN
LDI  R30,LOW(2*@0+@1)
LDI  R31,HIGH(2*@0+@1)
RCALL __GETW1PF
```

```
ICALL
.ENDM

.MACRO __CALL2EN
LDI  R26,LOW(@0+@1)
LDI  R27,HIGH(@0+@1)
RCALL __EEPROMRDW
ICALL
.ENDM

.MACRO __GETW1STACK
IN   R26,SPL
IN   R27,SPH
ADIW R26,@0+1
LD   R30,X+
LD   R31,X
.ENDM

.MACRO __NBST
BST  R@0,@1
IN   R30,SREG
LDI  R31,0x40
EOR  R30,R31
OUT  SREG,R30
.ENDM
```

```
.MACRO __PUTB1SN
LDD  R26,Y+@0
LDD  R27,Y+@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X,R30
.ENDM

.MACRO __PUTW1SN
LDD  R26,Y+@0
LDD  R27,Y+@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1SN
LDD  R26,Y+@0
LDD  R27,Y+@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
RCALL __PUTDP1
```

```
.ENDM

.MACRO __PUTB1SNS
LDD  R26,Y+@0
LDD  R27,Y+@0+1
ADIW R26,@1
ST   X,R30
.ENDM

.MACRO __PUTW1SNS
LDD  R26,Y+@0
LDD  R27,Y+@0+1
ADIW R26,@1
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1SNS
LDD  R26,Y+@0
LDD  R27,Y+@0+1
ADIW R26,@1
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1PMN
```

```
LDS  R26,@0
LDS  R27,@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X,R30
.ENDM

.MACRO __PUTW1PMN
LDS  R26,@0
LDS  R27,@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1PMN
LDS  R26,@0
LDS  R27,@0+1
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1PMNS
```

```
LDS  R26,@0
LDS  R27,@0+1
ADIW R26,@1
ST   X,R30
.ENDM

.MACRO __PUTW1PMNS
LDS  R26,@0
LDS  R27,@0+1
ADIW R26,@1
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1PMNS
LDS  R26,@0
LDS  R27,@0+1
ADIW R26,@1
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1RN
MOVW R26,R@0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
```

```
ST   X,R30
.ENDM

.MACRO __PUTW1RN
MOVW R26,R@0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1RN
MOVW R26,R@0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1RNS
MOVW R26,R@0
ADIW R26,@1
ST   X,R30
.ENDM

.MACRO __PUTW1RNS
```

```
MOVW R26,R@0
ADIW R26,@1
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1RNS
MOVW R26,R@0
ADIW R26,@1
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1RON
MOV  R26,R@0
MOV  R27,R@1
SUBI R26,LOW(-@2)
SBCI R27,HIGH(-@2)
ST   X,R30
.ENDM

.MACRO __PUTW1RON
MOV  R26,R@0
MOV  R27,R@1
SUBI R26,LOW(-@2)
SBCI R27,HIGH(-@2)
```

```
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1RON
MOV  R26,R@0
MOV  R27,R@1
SUBI R26,LOW(-@2)
SBCI R27,HIGH(-@2)
RCALL __PUTDP1
.ENDM

.MACRO __PUTB1RONS
MOV  R26,R@0
MOV  R27,R@1
ADIW R26,@2
ST   X,R30
.ENDM

.MACRO __PUTW1RONS
MOV  R26,R@0
MOV  R27,R@1
ADIW R26,@2
ST   X+,R30
ST   X,R31
```

```
.ENDM

.MACRO __PUTD1RONS
MOV  R26,R@0
MOV  R27,R@1
ADIW R26,@2
RCALL __PUTDP1
.ENDM


.MACRO __GETB1SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
LD   R30,Z
.ENDM

.MACRO __GETB1HSX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
LD   R31,Z
.ENDM

.MACRO __GETW1SX
```

```
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
LD   R0,Z+
LD   R31,Z
MOV  R30,R0
.ENDM

.MACRO __GETD1SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
LD   R0,Z+
LD   R1,Z+
LD   R22,Z+
LD   R23,Z
MOVW R30,R0
.ENDM

.MACRO __GETB2SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R26,X
.ENDM
```

```
.MACRO __GETW2SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R0,X+
LD   R27,X
MOV  R26,R0
.ENDM

.MACRO __GETD2SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R0,X+
LD   R1,X+
LD   R24,X+
LD   R25,X
MOVW R26,R0
.ENDM

.MACRO __GETBRSX
MOVW R30,R28
SUBI R30,LOW(-@1)
SBCI R31,HIGH(-@1)
```

```
LD   R@0,Z
.ENDM

.MACRO __GETWRSX
MOVW R30,R28
SUBI R30,LOW(-@2)
SBCI R31,HIGH(-@2)
LD   R@0,Z+
LD   R@1,Z
.ENDM

.MACRO __LSLW8SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
LD   R31,Z
CLR  R30
.ENDM

.MACRO __PUTB1SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
ST   X,R30
.ENDM
```

```
.MACRO __PUTW1SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
ST   X+,R30
ST   X,R31
.ENDM

.MACRO __PUTD1SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
ST   X+,R30
ST   X+,R31
ST   X+,R22
ST   X,R23
.ENDM

.MACRO __CLRW1SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
ST   X+,R30
ST   X,R30
```

```
.ENDM

.MACRO __CLRD1SX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
ST   X+,R30
ST   X+,R30
ST   X+,R30
ST   X,R30
.ENDM

.MACRO __PUTB2SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
ST   Z,R26
.ENDM

.MACRO __PUTW2SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
ST   Z+,R26
ST   Z,R27
```

```
.ENDM

.MACRO __PUTD2SX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
ST   Z+,R26
ST   Z+,R27
ST   Z+,R24
ST   Z,R25
.ENDM

.MACRO __PUTBSRX
MOVW R30,R28
SUBI R30,LOW(-@0)
SBCI R31,HIGH(-@0)
ST   Z,R@1
.ENDM

.MACRO __PUTWSRX
MOVW R30,R28
SUBI R30,LOW(-@2)
SBCI R31,HIGH(-@2)
ST   Z+,R@0
ST   Z,R@1
```

```
.ENDM

.MACRO __PUTB1SNX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R0,X+
LD   R27,X
MOV  R26,R0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X,R30
.ENDM

.MACRO __PUTW1SNX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R0,X+
LD   R27,X
MOV  R26,R0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X+,R30
ST   X,R31
```

```
.ENDM

.MACRO __PUTD1SNX
MOVW R26,R28
SUBI R26,LOW(-@0)
SBCI R27,HIGH(-@0)
LD   R0,X+
LD   R27,X
MOV  R26,R0
SUBI R26,LOW(-@1)
SBCI R27,HIGH(-@1)
ST   X+,R30
ST   X+,R31
ST   X+,R22
ST   X,R23
.ENDM

.MACRO __MULBRR
MULS R@0,R@1
MOVW R30,R0
.ENDM

.MACRO __MULBRRU
MUL  R@0,R@1
MOVW R30,R0
```

```
                .ENDM

                .MACRO __MULBRR0
                MULS R@0,R@1
                .ENDM

                .MACRO __MULBRRU0
                MUL  R@0,R@1
                .ENDM

                .MACRO __MULBNWRU
                LDI  R26,@2
                MUL  R26,R@0
                MOVW R30,R0
                MUL  R26,R@1
                ADD  R31,R0
                .ENDM

                .CSEG
                .ORG 0x00

            ;INTERRUPT VECTORS
000000 c026         RJMP __RESET
000001 cffe         RJMP 0x00
000002 cffd         RJMP 0x00
```

000003 cffc RJMP 0x00

000004 cffb RJMP 0x00

000005 cffa RJMP 0x00

000006 cff9 RJMP 0x00

000007 cff8 RJMP 0x00

000008 cff7 RJMP 0x00

000009 cff6 RJMP 0x00

00000a cff5 RJMP 0x00

00000b cff4 RJMP 0x00

00000c cff3 RJMP 0x00

00000d cff2 RJMP 0x00

00000e c035 RJMP _adc_isr

00000f cff0 RJMP 0x00

000010 cfef RJMP 0x00

000011 cfee RJMP 0x00

000012 cfed RJMP 0x00

000013 cfec RJMP 0x00

000014 cfeb RJMP 0x00

_tbl10_G100:

000015 2710

000016 03e8

000017 0064

000018 000a .DB 0x10,0x27,0xE8,0x3,0x64,0x0,0xA,0x0

000019 0001 .DB 0x1,0x0

```
                _tbl16_G100:
00001a 1000
00001b 0100
00001c 0010
00001d 0001             .DB  0x0,0x10,0x0,0x1,0x10,0x0,0x1,0x0

                _0x0:
00001e 3025
00001f 5832
000020 3025
000021 5832             .DB  0x25,0x30,0x32,0x58,0x25,0x30,0x32,0x58
000022 3025
000023 5832
000024 3025
000025 5832             .DB  0x25,0x30,0x32,0x58,0x25,0x30,0x32,0x58
000026 0048             .DB  0x48,0x0
                __RESET:
000027 94f8             CLI
000028 27ee             CLR  R30
000029 bbec             OUT  EECR,R30

                ;INTERRUPT VECTORS ARE PLACED
                ;AT THE START OF FLASH
00002a e0f1             LDI  R31,1
00002b bffb             OUT  GICR,R31
```

```
00002c bfeb        OUT  GICR,R30
00002d bfe5        OUT  MCUCR,R30

        ;DISABLE WATCHDOG
00002e e1f8        LDI  R31,0x18
00002f bdf1        OUT  WDTCR,R31
000030 bde1        OUT  WDTCR,R30

        ;CLEAR R2-R14
000031 e08d        LDI  R24,(14-2)+1
000032 e0a2        LDI  R26,2
000033 27bb        CLR  R27
        __CLEAR_REG:
000034 93ed        ST   X+,R30
000035 958a        DEC  R24
000036 f7e9        BRNE __CLEAR_REG

        ;CLEAR SRAM
000037 e080        LDI  R24,LOW(0x200)
000038 e092        LDI  R25,HIGH(0x200)
000039 e6a0        LDI  R26,0x60
        __CLEAR_SRAM:
00003a 93ed        ST   X+,R30
00003b 9701        SBIW R24,1
00003c f7e9        BRNE __CLEAR_SRAM
```

```
                ;STACK POINTER INITIALIZATION
00003d e5ef     LDI  R30,LOW(0x25F)
00003e bfed     OUT  SPL,R30
00003f e0e2     LDI  R30,HIGH(0x25F)
000040 bfee     OUT  SPH,R30

                ;DATA STACK POINTER INITIALIZATION
000041 eec0     LDI  R28,LOW(0xE0)
000042 e0d0     LDI  R29,HIGH(0xE0)

000043 c02e     RJMP _main

                .ESEG
                .ORG 0

                .DSEG
                .ORG 0xE0

                .CSEG
```

Lampiran 2

Register Summary dan Register Instruction
Atmega 8535

RESOURCE USE INFORMATION

Notice:

The register and instruction counts are symbol table hit counts,

and hence implicitly used resources are not counted, eg, the

'lpm' instruction without operands implicitly uses r0 and z,

none of which are counted.

x,y,z are separate entities in the symbol table and are

counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

ATmega8535 register use summary:

r0 : 7 r1 : 0 r2 : 0 r3 : 0 r4 : 0 r5 : 0 r6 : 0 r7 : 0

r8 : 0 r9 : 0 r10: 0 r11: 0 r12: 0 r13: 0 r14: 0 r15: 4

r16: 34 r17: 20 r18: 27 r19: 10 r20: 9 r21: 19 r22: 4 r23: 2

r24: 10 r25: 1 r26: 45 r27: 21 r28: 13 r29: 1 r30: 196 r31: 47

x : 14 y : 106 z : 7

Registers used: 21 out of 35 (60.0%)

ATmega8535 instruction use summary:

adc : 2 add : 3 adiw : 17 and : 0 andi : 5 asr : 0

bclr : 0 bld : 0 brbc : 0 brbs : 0 brcc : 0 brcs : 0

break : 0 breq : 13 brge : 0 brhc : 0 brhs : 0 brid : 0

brie : 0 brlo : 8 brlt : 0 brmi : 0 brne : 20 brpl : 1

brsh : 2 brtc : 0 brts : 0 brvc : 0 brvs : 0 bset : 0

bst : 0 cbi : 0 cbr : 0 clc : 0 clh : 0 cli : 1

cln : 0 clr : 10 cls : 0 clt : 0 clv : 0 clz : 0

com : 0 cp : 3 cpc : 2 cpi : 31 cpse : 0 dec : 2

des : 0 eor : 0 fmul : 0 fmuls : 0 fmulsu: 0 icall : 0

ijmp : 0 in : 8 inc : 0 ld : 18 ldd : 49 ldi : 71

lds : 5 lpm : 8 lsl : 1 lsr : 0 mov : 17 movw : 11

mul : 1 muls : 0 mulsu : 0 neg : 2 nop : 0 or : 0

ori : 7 out : 53 pop : 1 push : 1 rcall : 60 ret : 20

reti : 1 rjmp : 80 rol : 1 ror : 0 sbc : 1 sbci : 1

sbi : 1 sbic : 0 sbis : 2 sbiw : 13 sbr : 0 sbrc : 2

sbrs : 10 sec : 0 seh : 0 sei : 1 sen : 0 ser : 0

ses : 0 set : 0 sev : 0 sez : 0 sleep : 0 spm : 0

st : 34 std : 21 sts : 2 sub : 1 subi : 15 swap : 0

tst : 3 wdr : 0

Instructions used: 49 out of 110 (44.5%)

ATmega8535 memory use summary [bytes]:

Segment Begin End Code Data Used Size Use%

[.cseg] 0x000000 0x000530 1292 36 1328 8192 16.2%

[.dseg] 0x000060 0x0000e7 0 7 7 512 1.4%

[.eseg] 0x000000 0x000000 0 0 0 512 0.0%

Assembly complete, 0 errors, 0 warnings

Lampiran 3

Listing Program Aplikasi Borland Delphi

Untuk Komputer

Listing Program Aplikasi pada Konmputer dengan Borlad Delphi

1. **Form_ Menu dan Unit _ Menu**


```
unit Unit_Menu;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Menus;

type
  TForm_Menu = class(TForm)
    MainMenu1: TMainMenu;
    Se1: TMenuItem;
```

```
    SetRPM1: TMenuItem;
    Keluar1: TMenuItem;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    SetCOMM1: TMenuItem;
    procedure Se1Click(Sender: TObject);
    procedure SetRPM1Click(Sender: TObject);
    procedure Keluar1Click(Sender: TObject);
    procedure SetCOMM1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form_Menu: TForm_Menu;

implementation

uses Unit_SetRPM, Unit_SetPWM, Unit_COMM;
```

```
{$R *.dfm}

procedure TForm_Menu.Se1Click(Sender: TObject);
begin
  form_SetPWM.Show;
end;

procedure TForm_Menu.SetRPM1Click(Sender: TObject);
begin
  form_SetRPM.Show;
end;
procedure TForm_Menu.Keluar1Click(Sender: TObject);
begin
 application.Terminate;
end;
procedure TForm_Menu.SetCOMM1Click(Sender: TObject);
begin
  form_SetCOMM.Show;
end;
end.
```

2. **Form_ COMM dan Unit_ COMM**


```
unit Unit_COMM;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Buttons, Spin;

type
  TForm_SetCOMM = class(TForm)
    Timer1: TTimer;
    SpinEdit1: TSpinEdit;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    BitBtn2: TBitBtn;
```

```
    procedure Kirim_data();
    procedure Terima_data();
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form_SetCOMM: TForm_SetCOMM;
  sAddr, sByte, Data, sData : string;
  hSerialPort : hfile;
  TheDCB : TDCB ;
  BytesWritten, BytesRead : DWord;

implementation

uses Unit_SetPWM, Unit_SetRPM;

{$R *.dfm}
```

```
procedure TForm_SetCOMM.Kirim_data();
begin
  //mengirimkan data pada COMM
  WriteFile(hSerialPort, PChar(sData)^ , Length(sData), BytesWritten, nil);
  //mengosongkan buffer
  PurgeComm(hSerialPort, PURGE_TXABORT or PURGE_RXABORT or
    PURGE_TXCLEAR or PURGE_RXCLEAR);
end;

procedure TForm_SetCOMM.Terima_data();
var
  StartTime , TimedOutTime :  Cardinal;
  sResult : String;
  fTimeOut, FoundENDChar : boolean;
  BytesToRead, BytesRead, dwErrorCode : DWORD;
  statPort : TCOMSTAT;

begin
  TimedOutTime := 1;
  SResult := '';

  StartTime := GetTickCount;
  repeat
```

```
  ClearCommError(hSerialPort, dwErrorCode, @statPort);
  BytesToRead := statPort.cbInQue;

  if BytesToRead > 0 then
  begin
    if BytesToRead >= DWORD(Length(sData)) then
      SetLength(sData, BytesToRead );
    ReadFile(hSerialPort, PChar(sData)^, BytesToRead, BytesRead, nil);
    SResult := SResult + Copy(sData, 0, BytesRead);
  end;
  FoundENDChar := Pos(#72, SResult) <> 0; //#72=H akhir data dari MCU
  fTimeOut := GetTickCount > StartTime + TimedOutTime;
 until FoundENDChar or fTimeOut;
 if FoundENDChar then  Data := SResult else  Data := '';
end;

procedure TForm_SetCOMM.BitBtn1Click(Sender: TObject);
begin
 case spinedit1.Value of
 1:  hSerialPort := CreateFile('COM1', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
 2:  hSerialPort := CreateFile('COM2', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
 3:  hSerialPort := CreateFile('COM3', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
```

```
4:  hSerialPort := CreateFile('COM4', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
5:  hSerialPort := CreateFile('COM5', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
6:  hSerialPort := CreateFile('COM6', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
7:  hSerialPort := CreateFile('COM7', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
8:  hSerialPort := CreateFile('COM8', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));
9:  hSerialPort := CreateFile('COM9', GENERIC_READ or GENERIC_WRITE, 0,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, longint(0));

end;

if hSerialPort <> INVALID_HANDLE_VALUE then
  begin
  // mengubah setting TDCB sbb. :
  GetCommState(hSerialPort, TheDCB);
  TheDCB.BaudRate := CBR_38400;
  TheDCB.ByteSize := 8;
  TheDCB.Parity   := NOPARITY; //MARKPARITY,SPACEPARITY
  TheDCB.StopBits := ONESTOPBIT;
  TheDCB.Flags    := 1;
  SetCommState(hSerialPort, TheDCB);
```

```
   label1.Caption := 'COM'+inttostr(spinedit1.Value)+' Sudah terkoneksi';
   form_SetPWM.timer2.Enabled := false;
   form_SetRPM.timer2.Enabled  := false;
   form_SetPWM.label2.Visible := false;
   form_SetRPM.label2.Visible   := false;
   timer1.Enabled := false;
  end
  else  begin
    timer1.Enabled := true;
    form_SetPWM.timer2.Enabled  := true;
    form_SetRPM.timer2.Enabled := true;
    label1.Caption := 'COM'+inttostr(spinedit1.Value)+' tidak terkoneksi';
  end;

end;

procedure TForm_SetCOMM.BitBtn2Click(Sender: TObject);
begin
  timer1.Enabled := false;
  label1.Visible := true;
  close;
end;

procedure TForm_SetCOMM.Timer1Timer(Sender: TObject);
begin
```

```
  if (label1.Visible = true) then label1.Visible := false
  else label1.Visible := true;
end;
end.
```

3. **Form_ SetRPM dan Unit_SetRPM**


```
unit Unit_SetRPM;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, OleCtrls, Spin, ExtCtrls, Buttons,Math;
```

```
type
  TForm_SetRPM = class(TForm)
    Label1: TLabel;
    Timer1: TTimer;
    BitBtn1: TBitBtn;
    BitBtn3: TBitBtn;
    Label13: TLabel;
    ScrollBar1: TScrollBar;
    Label4: TLabel;
    Label3: TLabel;
    Label2: TLabel;
    Timer2: TTimer;
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure Timer2Timer(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form_SetRPM: TForm_SetRPM;
```

```
  var s_rpm,A0 : string;
      i_rpm,rpm, i, delta : integer;
      pwm,k : byte;


implementation

uses Unit_COMM;

{$R *.dfm}


procedure TForm_SetRPM.Timer1Timer(Sender: TObject);
begin
  form_SetCOMM.Terima_data();
  s_rpm := copy(Data,1,4);
  val('$'+ s_rpm,i_rpm,i);
  rpm := round(i_rpm*1200/116);
  A0 := inttostr(rpm);

  k :=k+1;
  if (k=10) then begin
  Label4.Caption := A0  ; k := 0; end;
  delta := abs(rpm - scrollbar1.Position);
```

```
  if (rpm < scrollbar1.Position) then begin
   if (delta >50) then  pwm := pwm + 5;
   if ((delta <=50) and (delta >=2)) then pwm := pwm +1;
  end;

  if (rpm > scrollbar1.Position) then begin
    if (delta >50) then  pwm := pwm - 5;
    if ((delta <=50) and (delta >=2)) then pwm := pwm - 1;
  end;

  sData := chr(pwm);
  form_SetCOMM.Kirim_data();
  Label3.Caption := inttostr(scrollbar1.Position);
end;

procedure TForm_SetRPM.BitBtn1Click(Sender: TObject);
begin
   timer1.Enabled :=true;
end;

procedure TForm_SetRPM.BitBtn3Click(Sender: TObject);
begin
   sData := chr(0);
   form_SetCOMM.Kirim_data();
   timer1.Enabled :=false;
```

```
  timer2.Enabled :=false;
  label2.Visible := false;
  close;
end;

procedure TForm_SetRPM.Timer2Timer(Sender: TObject);
begin
   if (label2.Visible = true) then  label2.Visible := false
   else label2.Visible := true;
end;

end.
```

4. **Form_ SetPWM dan Unit _ SetPWM**


```
object Form_SetPWM: TForm_SetPWM
```

Left = -151

Top = 79

Width = 473

Height = 267

Caption = 'Set PWM'

Color = clBtnFace

Font.Charset = DEFAULT_CHARSET

Font.Color = clWindowText

Font.Height = -11

Font.Name = 'MS Sans Serif'

Font.Style = []

OldCreateOrder = False

Position = poDesktopCenter

PixelsPerInch = 96

TextHeight = 13

object Label1: TLabel

Left = 38

Top = 83

Width = 92

Height = 26

Caption = 'Set PWM'

Font.Charset = DEFAULT_CHARSET

Font.Color = clWindowText

Font.Height = -23

Font.Name = 'MS Sans Serif'

```
  Font.Style = []
  ParentFont = False
end
object Label13: TLabel
  Left = 38
  Top = 133
  Width = 60
  Height = 26
  Caption = 'RPM :'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -23
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
end
object Label4: TLabel
  Left = 155
  Top = 130
  Width = 134
  Height = 29
  Alignment = taCenter
  AutoSize = False
  Caption = ' label4'
  Color = clSkyBlue
```

```
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clNavy
  Font.Height = -23
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentColor = False
  ParentFont = False
end
object Label3: TLabel
  Left = 368
  Top = 83
  Width = 56
  Height = 24
  Caption = 'Label3'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -19
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
end
object Label2: TLabel
  Left = 96
  Top = 184
  Width = 281
```

```
  Height = 29
  Alignment = taCenter
  AutoSize = False
  Caption = 'COMM Belum Terkoneksi !'
  Color = clSkyBlue
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -23
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentColor = False
  ParentFont = False
  Visible = False
end
object BitBtn1: TBitBtn
  Left = 32
  Top = 24
  Width = 97
  Height = 33
  Caption = 'Mulai'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clGreen
  Font.Height = -16
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
```

```
  ParentFont = False
  TabOrder = 0
  OnClick = BitBtn1Click
end
object BitBtn3: TBitBtn
  Left = 185
  Top = 24
  Width = 89
  Height = 32
  Caption = 'Keluar'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clRed
  Font.Height = -16
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
  TabOrder = 1
  OnClick = BitBtn3Click
end
object ScrollBar1: TScrollBar
  Left = 152
  Top = 83
  Width = 185
  Height = 25
  LargeChange = 10
```

```
    Max = 255
    PageSize = 0
    ParentShowHint = False
    ShowHint = False
    TabOrder = 2
  end
  object Timer1: TTimer
    Enabled = False
    Interval = 50
    OnTimer = Timer1Timer
    Left = 320
    Top = 24
  end
  object Timer2: TTimer
    Interval = 500
    OnTimer = Timer2Timer
    Left = 384
    Top = 24
  end
end
```