PENGENDALI MOTOR STEPPER VIA USB INTERFACE BERBASIS MIKROKONTROLER AVR ATMEGA8535

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana teknik

Program Studi Teknik Elektro Fakultas Teknologi Industri

Universitas Islam Indonesia

Disusun Oleh :

NAMA : Wirawan Aditya

NIM : 01 524 119

JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

YOGYAKARTA

2008

LEMBAR PENGESAHAN PEMBIMBING

TUGAS AKHIR

PENGENDALI MOTOR STEPPER VIA USB INTERFACE

BERBASIS MIKROKONTROLER AVR ATMEGA8535

Disusun Oleh:

NAMA : Wirawan Aditya
NIM : 01 524 119

Disetujui :

Yogyakarta, Januari 2008

Pembimbing I	Pembimbing II
(Wahyudi Budi Pramono, ST)	(Yusuf Aziz Amrullah, ST)

LEMBAR PENGESAHAN PENGUJI

TUGAS AKHIR

PENGENDALI MOTOR STEPPER VIA USB INTERFACE

BERBASIS MIKROKONTROLER AVR ATMEGA8535

Disusun Oleh:

NAMA : Wirawan Aditya

NIM : 01 524 119

Telah Dipertahankan di Depan Sidang Penguji sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Teknik Elektro
Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, Januari 2008

Tim Penguji

Tito Yuwono, ST, M.Sc.
Ketua

Wahyudi Budi Pramono, ST.
Anggota I

Dr. Agung Alfiansyah, ST DEA
Anggota II

Mengetahui,
Ketua Jurusan Teknik Elektro – Fakultas Teknologi Industri
Universitas Islam Indonesia

Tito Yuwono, ST, M.Sc.

ABSTRAK

Port USB dapat digunakan sebagai pengganti *port* serial dan pararel untuk menghubungkan PC/CPU dengan perangkat lain. Salah satu pengembangan kegunaan *port* USB yaitu penggerak motor *stepper* Motor *stepper* dioperasikan dengan menggunakan mikrokontroler AT90S2313 sebagai *pull-up* data ke mikrokontroler ATMega8535 sebagai pelaksana instruksi. Perangkat lunak yang digunakan adalah Delphi 7 yang berisi instruksi berupa sub rutin yang akan dikirimkan melalui *port* USB dan diterima oleh mikrokontroler AT90S2313 untuk menjalankan instruksi pada mikrokontroler ATMega8535 yang akan menggerakkan motor *stepper* sesuai dengan perintah yang inginkan. Untuk menyuplai rangkaian dengan kebutuhan arus yang cukup besar maka dibutuhkan penguat arus yang terdapat pada rangkaian regulator tegangan. Rangkaian tersebut juga digunakan sebagai pembangkit tegangan pada motor *stepper*. Pada alat ini pelindung tegangan berlebih digunakan regulator jenis positif regulator dengan tipe LM7805 untuk penstabil tegangan 5VDC. Setelah rangkaian penggerak motor *stepper* terhubung dengan baik melalui *port* USB maka program *delphi* dapat dijalankan dan ditentukan arah dan derajat yang diinginkan yaitu arah ke kiri dan kanan serta simpangan sudutnya yaitu 10, 30, 60, 90, dan 180 derajat. Dari analisis dapat dilihat bahwa alat dapat berfungsi dengan baik sesuai dengan instruksi yang diberikan. Namun kadangkala terjadi posisi sudut yang tidak tepat, yaitu persentase error deviasi rata-ratanya antara 1,2% – 2 % , hal ini disebabkan faktor motor *stepper* yang tidak normal lagi.

HALAMAN PERSEMBAHAN

Kupersembahkan karya ini :

Ayahanda tercinta, maafkan ananda yang baru ini mewujudkan harapan ayah, ananda mengetahui ayah selalu setia mendoakan ananda..

Ibunda tercinta, senyummu baru ini terwujud dari sekian lama ibunda memberi Nasehat dan tak pernah berhenti berdoa untuk sebuah kata wisuda pada ananda.

Adikku Dwi Piranti dan Mas Bambang yang selalu membantu dan sebagai motivatorku

Yang terindah dalam hidupku, Upit kekasihku, terucap terima kasih untuk sikapmu yang setia menunggu, memberikan doa dan dorongan untukku

KATA PENGANTAR

Bismillaahirrohmaanirroohim.

Dengan mengucap syukur *Alhamdulillah* kehadirat Allah SWT. Yang telah memberikan kekuatan lahir dan batin sehingga kami dapat menyusun Tugas Akhir ini dengan baik. *Sholawat* dan *salam* semoga tetap tercurah kepada Nabi Muhammad SAW. Yang kita nantikan *syafa 'at*nya di hari kiamat.

Penyusunan Tugas Akhir ini merupakan mata kuliah wajib yang dilaksanakan di Jurusan Teknik Elektro Fakultas Teknologi Industri Universitas Islam Indonesia, Yogyakarta sebagai salah satu syarat memperoleh gelar sarjana Strata Satu (S1).

Dalam penyusunan Tugas Akhir ini penulis telah banyak menerima bantuan dan motivasi sereta bimbingan dari berbagai fihak, baik secara moril maupun material, maka pada kesempatan ini penulis ingin menyampaikan rasa terimakasih yang tak terhingga kepada:

1. Bapak Rektor Universitas Islam Indonesia Yogyakarta
2. Bapak Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Tito Yuwono, ST, MSc. selaku Ketua Jurusan Teknik Elektro Fakultas Teknologi Industri.
4. Bapak Yusuf Aziz Amrullah, ST selaku Seketaris Jurusan Teknik Elektro dan sebagai Dosen Pembimbing II.
5. Bapak Wahyudi Budi Pramono, ST selaku pembimbing I
6. Seluruh Dosen Jurusan Teknik Elektro Fakultas Teknologi Industri Universitas Islam Indonesia, terimakasih atas ilmu yang telah Bapak dan Ibu berikan pada saya.
7. Ayahanda tercinta, maafkan ananda yang baru ini mewujudkan harapan ayah, ananda mengetahui ayah selalu setia mendoakan ananda..
8. Ibunda tercinta, senyummu baru ini terwujud dari sekian lama ibunda

memberi semangat dan tak pernah berhenti berdoa untuk sebuah kata wisuda pada ananda.

9. Adikku tersayang Dwi Piranti, Mas Bambang yang selalu memberi semangat, dan Dherrie yang menasehati dan memotivasi saya untuk tetap bersemangat.
10. Yang terindah dalam hidupku, Upit kekasihku tercinta, terucap terima kasih untuk sikapmu yang setia menunggu, memberikan doa dan dorongan untukku.
11. Teman - teman Elektro 01 (Agus, Monti, Andra, Heru, banyak deh) senang ada bersama kalian, semoga sukses yach.
12. Teman - teman Dota community (Handi, Bayu, Pascal, bang Bex, bang Joni, Lia, Kalen the miss of mutung, ect. "Love you Guys"). BL Community (Girie, Bayu, wa' Opix, Mohawk, Yayan, Yongki, Buffon, Anto) serta teman-teman yang tidak bisa saya sebut satu per satu.

Layaknya sebuah teks, bagaimana pun juga adalah sebuah kawasan *multi-interpretable,* yang senantiasa bebas terbuka untuk ditafsirkan, dikritik dan disalahartikan oleh pembaca. Demikian juga dengan Tugas akhir ini, kendatipun penulis telah berusaha seoptimal mungkin, agar penyusunan Tugas akhir ini mendekati sempurna, akan tetapi penulis menyadari pasti masih banyak terdapat kesalahan dan kekurangan didalamnya. Oleh karena itu penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Harapan penulis, semoga Tugas Akhir ini dapat memberikan manfaat bagi penulis kususnya dan bagi pembaca umumnya.

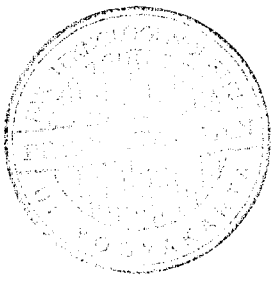
Yogyakarta, Januari 2008

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR



BAB I

PENDAHULUAN

1.1 Latar Belakang

Komputer adalah salah satu terobosan yang sangat mempengaruhi perkembangan teknologi diseluruh dunia. Pada komputer terdapat jenis saluran data dari komputer ke periperal lainnya, koneksi standar yang sudah ada dari dulu adalah koneksi serial (RS-232C), dan koneksi paralel (*Centronics)*. Koneksi serial dipakai untuk mengirimkan data dua-arah (*full duplex*) dari dan ke *mouse*, modem dan beberapa jenis printer model lama. Pada koneksi serial, data dikirimkan satu demi satu, pengiriman data melalui saluran serial memang memakan waktu panjang, kecepatan maksimalnya hanya 115 kilobit per detik (*kilobit per second*, atau kbps). Saluran paralel dipakai untuk *printer*, modem, *CD-ROM drive* eksternal, CD-*Writer*, dan *scanner*. Sesuai namanya, delapan butir data (*bit*) dikirimkan sekaligus melalui delapan saluran paralel yang kecepatannya 2Mbps lebih baik daripada koneksi serial.

Seiring dengan waktu maka ditemukanlah koneksi yang lebih baik yaitu *Universal Serial Bus* (USB). USB merupakan pengembangan dari koneksi serial. USB menjadi populer di industri komputer dan periperalnya mulai bermunculan hingga sekarang. USB memang paling banyak digunakan karena hampir semua *motherboard* telah menyediakan *port* USB. *Universal Serial Bus* (USB) merupakan suatu teknologi yang memungkinkan kita untuk menghubungkan alat (*peripheral*) eksternal ke PC, seperti *scanner*, *printer*, *mouse*, *keyboard*, alat

2. Bahasa pemrograman yang sesuai untuk menggerakkan motor *stepper* tersebut dan mampu memenuhi tujuan pengendalian yaitu bergerak sesuai instruksi yang diinginkan.

1.3 Batasan Masalah

Agar dalam penulisan penelitian ini lebih terarah, maka pembahasan penulisan ini dibatasi pada ruang lingkup pembahasan sebagai berikut :

1. Rangkaian pengoperasian motor *stepper* dapat bekerja melalui koneksi USB *interface*.
2. Pembuatan sistem ini menggunakan perangkat lunak Borland Delphi 7.0.
3. Menggunakan mikrokontroler sebagai komunikasi USB.
4. Motor *stepper* dapat bekerja sesuai dengan instruksi pada program Delphi untuk perputaran ke kiri dan kanan sebesar 10, 30, 60, 90 dan 180 derajat.

1.4 Tujuan Penelitian

Adapun tujuan dari penelitian dapat dirumuskan sebagai berikut :

1. Menggunakan Delphi 7.0 untuk menjalankan motor *stepper*.
2. Membuat rangkaian yang akan menghubungkan antarmuka USB komputer ke motor *stepper* melalui perantara mikrokontroler.
3. Mengaplikasikan penggunaan mikrokontroler ATMega8535.
4. Dapat dikembangkan dan dimanfaatkan oleh masyarakat.

1.5 Sistematika Penulisan

Sistematika penulisan dari laporan akhir penelitian ini yang berupa karya tulis (skripsi) akan dibagi dalam lima bab, dengan isi masing masing bab diuraikan sebagai berikut :

BAB 1 Pendahuluan

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

BAB II Landasan Teori

Menguraikan tentang teori-teori yang menjadi acuan dalam pembuatan tugas akhir, diantaranya teori tentang sistem kerja USB mikrokontroler ATMega8535, mikrokontroler AT90S2313 dan teori pendukung lainnya.

BAB III Perancangan Sistem

Berisi tentang perancangan dan pembuatan sistem pengendalian motor *stepper* mulai dari diagram blok, prinsip kerja, serta pembahasan mengenai *software* sebagai otak sistem pengendalinya.

BAB IV Pengujian, Analisis & Pembahasan

Membahas tentang hasil pengujian dan analisis dari sistem yang dibuat dibandingkan dengan dasar teori sistem atau uraian alasan ilmiah yang lain.

BAB II

DASAR TEORI

2.1 Motor *Stepper*

Motor *stepper* adalah salah satu tipe motor yang sangat populer digunakan sebagai penggerak/pemutar peralatan industri. Prinsip kerja motor *stepper* ini mirip dengan motor DC, yaitu sama-sama dicatu dengan tegangan DC untuk memperoleh medan magnet. Bila DC motor memiliki magnet tetap pada stator, *stepper* motor mempunyai magnet tetap pada rotor. Suatu motor *stepper* biasanya cukup dinyatakan dengan spesifikasi : "berapa phasa", "berapa derajat perstep", "berapa volt tegangan catu untuk tiap lilitan" dan "berapa *ampere*/miliampere arus yang dibutuhkan untuk tiap lilitan".

Pada dasarnya motor *stepper* merupakan motor DC yang tidak memiliki komutator. Secara tipikal, motor *stepper* hanya mempunyai kumparan pada statornya, sedangkan pada bagian rotornya merupakan magnet permanen. Bagian tengah merupakan bagian yang berputar yang disebut rotor, dan bagian yang diam disebut stator. Stator terdiri dari beberapa kutub, makin banyak kutub makin sulit konstruksinya. Pada motor *stepper* dengan empat kutub, setiap kutub memiliki lilitan yang menghasilkan medan magnet yang akan menggerakkan rotor. Bila kumparan mendapatkan tegangan, dengan analogi mendapat logika '1' maka akan dibangkitkan kutub magnet yang berlawanan dengan kutub magnet tetap pada rotor. Sehingga posisi kutub magnet rotor akan ditarik mendekati lilitan yang menghasilkan kutub magnet yang berlawanan tadi. Bila langkah berikutnya, lilitan

pada data yang dikirimkan ke rangkaian *interface* motor *stepper* tersebut.Urutan pemberian arus pada lilitan dan arah putaran yang dikehendaki pada tabel 2.1

Tabel 2.1 Urutan pemberian arus

Step ke	Searah jarum jam				Berlawanan jarum jam			
1	1	0	0	0	0	0	0	1
2	0	1	0	0	0	0	1	0
3	0	0	1	0	0	1	0	0
4	0	0	0	1	0	0	0	0
	Kembali ke step 1				Kembali ke step 1			

Pergerakan motor *stepper* adalah berdasarkan perubahan logika pada input lilitan-lilitannya dan untuk dapat membuat gerakan yang lebih presisi biasanya jumlah batang magnet di rotor diperbanyak dan lilitan dibuat berpasang-pasangan sesuai dengan posisi kutub magnet rotor. Cara lain adalah dengan menggunakan sistem *gear* pada poros rotor tanpa mengubah karakteristik motor *stepper*-nya.

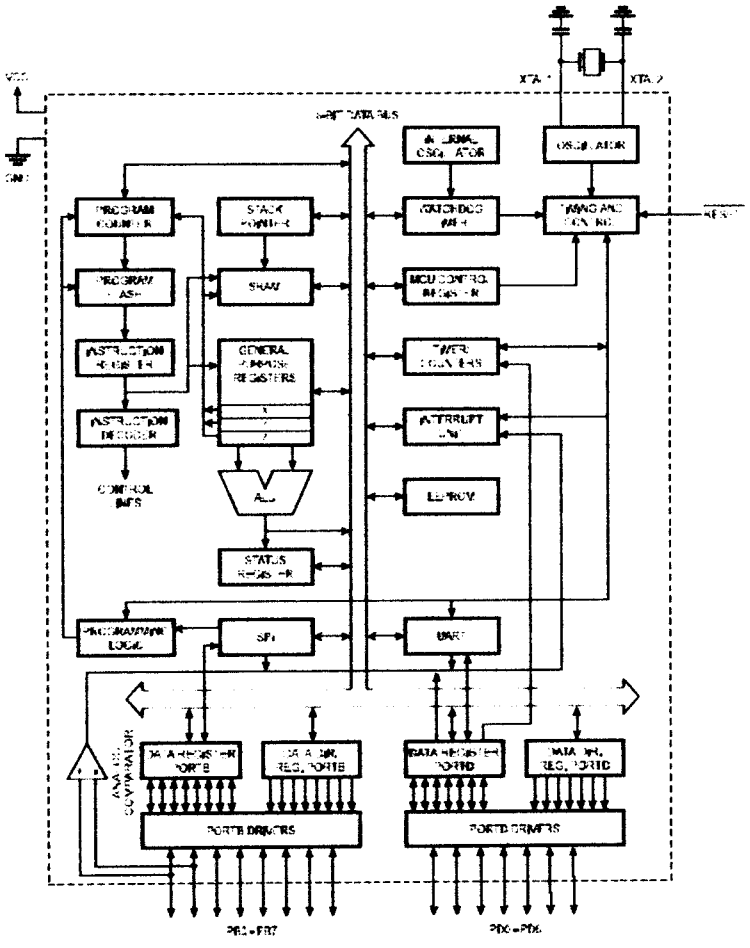
2.2 Mikrokontroler AVR AT90S2313

Mikrokontroler AVR seri AT90S2313 dipilih karena mikrokontroler ini dapat mewakili beberapa seri yang lain, baik dari segi instruksi yang digunakan, maupun *peripheral*-nya yang relatif lengkap.

2.2.1 Arsitektur Mikrokontroler AVR AT90S2313

Mikrokontroler AT90S2313 merupakan mikrokontroler CMOS dengan daya rendah yang memiliki arsitektur AVR RISC 8-bit. Arsitektur ini mendukung kemampuan untuk melaksanakan eksekusi instruksi hanya dalam waktu satu siklus *clock* osilator. Hal ini sangat cocok bila menginginkan suatu desain sistem aplikasi yang cepat dan hemat daya. AVR ini memiliki fitur untuk menghemat

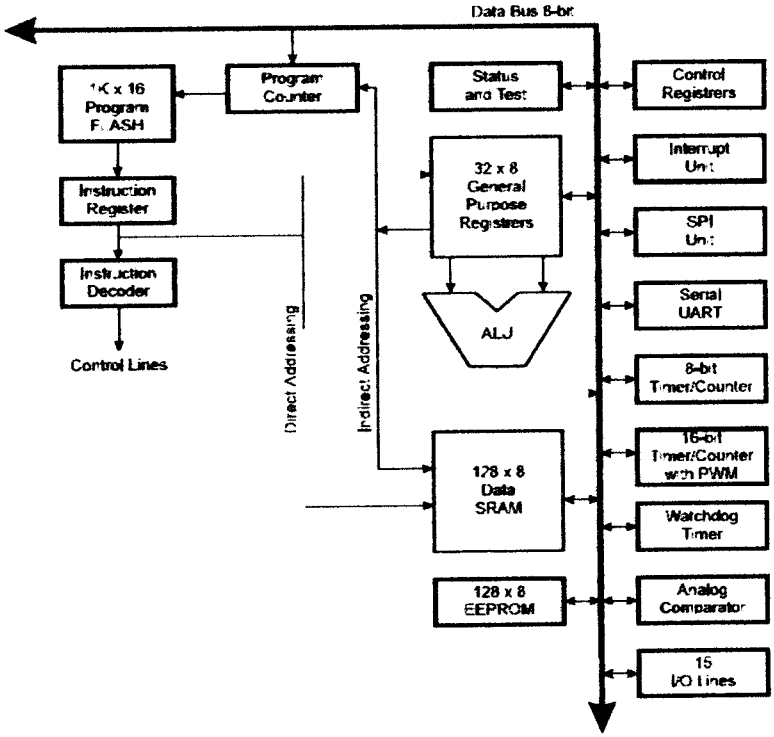
konsumsi daya, yaitu dengan menggunakan mode *sleep*. Mode *sleep* pada mikrokontroler AVR ada dua macam, yaitu mode *idle* dan mode *power-down*.



Gambar 2.2 Blok diagram fungsional AT90S2313

Mikrokontroler AVR memiliki model arsitektur *Harvard*, dimana memori dan *bus* untuk program dan data dipisahkan. Dalam arsitektur AVR, seluruh 32 register umum yang ada terhubung langsung ke ALU prosesor. Hal inilah yang membuat AVR begitu cepat dalam mengeksekusi instruksi. Dalam satu siklus *clock*, terdapat dua register independen yang dapat diakses oleh satu instruksi.

Teknik yang digunakan adalah *fetch during execution* atau memegang sambil mengerjakan. Hal ini berarti, dua operan dibaca dari dua register, dilakukan eksekusi operasi, dan hasilnya disimpan kembali dalam salah satu register, semuanya dilakukan hanya dalam satu siklus *clock*. Arsitektur AVR AT90S2313 ditunjukkan dalam gambar berikut.



Gambar 2.3 Arsitektur AT90S2313

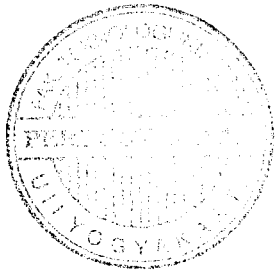
Dari register yang ada, terdapat enam buah register yang dapat digunakan untuk pengalamatan tidak langsung 16-bit sebagai register *pointer*. Register tersebut memiliki nama khusus, yaitu X, Y, dan Z. Masing-masing terdiri dari sepasang register. Register-register khusus tersebut adalah R26:R27 (register X), R28:R29 (register Y), dan R30:R31 (register Z). Selain ketiga pasangan register

tersebut, sebenarnya terdapat satu pasang register lagi yang dapat digunakan bersama untuk pengolahan data 16-bit, yaitu R24:R25. Pasangan register ini tidak memiliki nama khusus sebagaimana ketiga pasangan register yang telah disebutkan sebelumnya.

2.2.2 Fitur AT90S2313

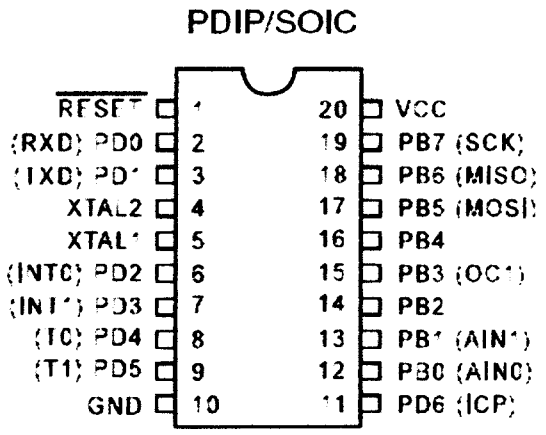
Mikrokontroler AT90S2313 memiliki fitur-fitur utama antara lain sebagai berikut:

1. 118 macam instruksi,
2. 32 x 8 bit *General Purpose Register*,
3. Memori program *Flash* pada ROM 2 K *word* (1K x 16),
4. Memori data SRAM 128 *byte*,
5. Memori EEPROM 128 *byte*,
6. Jalur I/O 15 pin,
7. *Timer/counter* 2 buah,
8. *Output* PWM 1 kanal,
9. Serial I/O menggunakan USART,
10. Komparator analog.



2.2.3 Konfigurasi Pin AT90S2313

AT90S2313 beredar dalam dua jenis kemasan, yaitu 20 DIP dan 20 SOIC. Dengan kemasannya yang cukup sederhana maka akan memudahkan dalam mempelajari cara-cara pemrograman mikrokontroler AVR tanpa harus dipusingkan oleh instalasi kabel yang melibatkan banyak jalur sebagaimana pada mikrokontroler dengan jumlah pin di atas 40 buah. Berikut adalah pin-pin pada AT90S2313.



Gambar 2.4 Pin AT90S2313

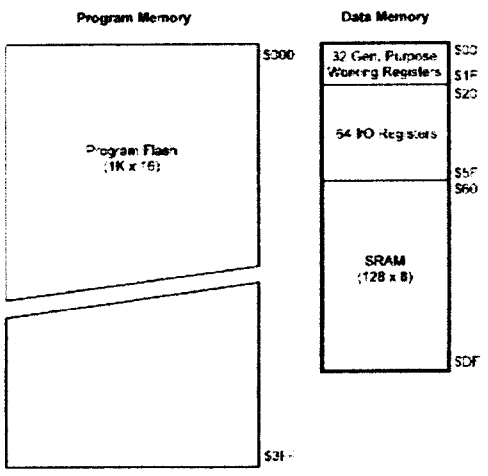
Dari gambar tersebut dapat dijelaskan secara fungsional konfigurasi pin AT90S2313 sebagai berikut:

1. VCC merupakan pin yang berfungsi sebagai pin masukan catu daya.
2. GND merupakan pin *ground.*
3. *Port* B (PB7..PB0) *Port* B merupakan *port* I/O 8-bit *bi-directional.* Pin-pin pada *port* ini dapat diberi resistor *pull-up* internal secara individual. PB0 dan PB1 juga dapat digunakan untuk melayani *input* sebagai komparator analog. *Buffer port* B dapat mencatu arus hingga 20 mA dan dapat secara langsung men-*drive* LED.
4. *Port* D (PD6..PD0) *port* D memiliki tujuh buah pin I/O *bi-directional,* yakni PD6..PD0. Seperti halnya *port* B, pin-pin pada *port* ini juga mampu men-*drive* LED karena dapat mencatu arus hingga 20 mA.
5. RESET , Reset *input.* Kondisi logika rendah "0" lebih dari 50 ns pada pin ini akan membuat mikrokontroler masuk ke dalam kondisi reset.

6. XTAL1, *Input* bagi *inverting oscilator amplifier* dan *input* bagi *clock* internal.
7. XTAL2, *Output inverting oscilator amplifier.*

2.2.4 Organisasi Memori

Dalam organisasi memori AVR, 32 register keperluan umum (GPR) menempati *space* data pada alamat terbawah, yaitu $00 sampai $1F. Sedangkan register-register khusus, untuk penanganan I/O dan kontrol terhadap mikrokontroler, menempati 64 alamat berikutnya, yaitu mulai dari $20 hingga $5F. Register-register ini merupakan register-register khusus digunakan untuk melakukan pengaturan fungsi terhadap berbagai *peripheral* mikrokontroler semacam kontrol register, *timer/counter*, fungsi-fungsi I/O, dan sebagainya. Kelompok register ini dinamakan register I/O. Alamat memori berikutnya digunakan untuk SRAM 128 *byte*, yaitu pada lokasi $60 sampai dengan $DF. Gambaran peta memori untuk AVR AT90S2313 ditunjukkan oleh gambar berikut.



Gambar 2.5 Peta memori AT90S2313

2.2.5 *PortI/O*

Mikrokontroler AVR AT90S2313 mempunyai dua buah port I/O yaitu:

a. *Port* B

Port B merupakan *port* I/O 8-bit *bi-directional* yang masing-masing pinnya dapat dikonfigurasi secara individual. Masing-masing pin dalam *port* ini juga memiliki fasilitas berupa registor *pull-up* internal yang berguna untuk memberikan kondisi yang tentu (tidak ngambang) pada saat dikonfigurasi sebagai *input*, tanpa harus memberikan *pull-up* eksternal. Untuk mendukung penggunaan untuk menangani fungsi *Port* B.

Ada tiga buah alamat memori yang khusus digunakan untuk menangani fungsi *port* B yaitu:

i. *Data register* (*PORTB*) yang berlokasi pada $18 ($38), register ini dapat ditulis maupun dibaca.

ii. *Data Direction register port* B (DDRB) yang berlokasi pada $17 ($37), register ini dapat ditulis atau dibaca.

iii. *Port* B *input* pin (PINB), berlokasi pada $16($36). PINB bukanlah suatu register, namun pin-pin fisik pada *hardware* mikrokontroler. Pin ini hanya dapat dibaca.

Beberapa pin pada *port B* memiliki fungsi alternatif yang dapat digunakan sesuai kebutuhannya. Pin-pin tersebut adalah PB0, PB1, PB3, PB5, PB6 dan PB7. Fungsi-fungsi alternatif tersebut dapat ditunjukan pada Tabel 2.1.

Tabel 2.2 Fungsi alternatif *Port* B

***Port* Pin**	**Fungsi Alternatif**
PB0	AIN0(*Analog Comparator Positive Input*)
PB1	AIN0(*Analog Comparator Negatif Input*)
PB3	OC1(*Timer/Counter1 Output Compare Match Output*)
PB5	MOSI(*Data Input Line for Memory Downloading*)
PB6	MISO(*Data OutputLline for Memory Uploading*)
PB7	SCK (*Serial Clock Input for Memory Programming*)

b. *Port* D

Port D memiliki tiga buah lokasi memori yang berkaitan dengan penggunaanya sebagai *port* I/O. Memori-memori tersebut adalah:

i. *PORT* D *(Data Direction)* berlokasi pada $12.

ii. DDRD (*Data Direction Register port D*) berlokasi pada $11.

iii. PIND (PortD *input pins*) berlokasi pada $10. PIND bukanlah register, PIND hanya dapat dibaca.

Format *Port* D hampir sama dengan *Port* B dalam hal konfigurasi I/O dan dalam penggunaan resistor *pull-up*. Hal yang membedakan dengan *Port* B adalah jumlah pin yang bisa digunakan hanya 7 buah, karena sesuai dengan jumlah pin yang ada pada *Port* D. Pin-pin pada *Port* D juga memiliki fungsi alternatif lain. Fungsi-fungsi alternatif lain tersebut ditunjukan pada Tabel 2.2.

Tabel 2.3 Fungsi alternatif *Port* D.

***Port* Pin**	**Fungsi Alternatif**
PD0	RXD (*Receive Data Input for the UART*)
PD1	TXD (*Transmit Data Output for the UART*)
PD2	INT0 (*External Interrupt 0 Input*)
PD3	INT1 (*External Interrupt 1 Input*)
PD4	T0 (*Timer/Counter External Input*)
PD5	T1 (*timer/Counter External Input*)
PD6	ICP (*Timer/Counter1 Capture Pin*)

2.2.6 Interupsi

Terdapat 10 jenis interupsi yang dapat ditangani oleh AT90S2313. Interupsi tersebut akan mengarahkan aliran program menuju vektor interupsi yang berada pada alamat $01 hingga $0A, telah digunakan untuk vektor reset, yaitu alamat yang akan dituju apabila terjadi salah satu kondisi dari tiga kondisi reset. Agar suatu interupsi dapat difungsikan, maka bit peng-*enable* interupsi yang bersangkutan harus di-*enable* bersama dengan bit-I dalam register SREG yang merupakan bit peng-*enable* interupsi secara global.

2.2.7 *Peripheral* AT90S2313

Peripheral utama pada AT90s2313 meliputi *Timer/Counter*, *WatchDog Timer*, EEPROM, *Analog Comparator*, dan UART. Untuk konfigurasi fungsi

peripheral-peripheral di atas, maka hal yang harus dilakukan adalah mengatur *setting* bit pada register kontrol yang bersangkutan.

AT90S2313 memilik dua modul *Timer/Counter*, yaitu *Timer/Counter0* yang dinamakan TCNT0 dan *Timer/Counter1* yang dinamakan TCNT1L dan TCNT1H. Untuk mengatur kerja *Timer/Counter* perlu dilakukan *setting* terhadap bit-bit dalam register I/O.

Register-register yang berhubungan dengan penggunaan TCNT0 adalah TCCR0 (*Timer/Counter0 Control Register*), TIFR (*Timer/Counter0 Interrupt Flag Register*), dan TIMSK (*Timer/Counter0 Interrupst Mask Register*).

Register-register yang berhubungan dengan pengaturan *Timer/Counter1* adalah TCCR1B, TIFR dan TIMSK. *Timer/Counter1* juga mendukung fungsi *Output Compare* menggunakan register OCR1A

2.3 Mikrokontroler AVR ATMega8535

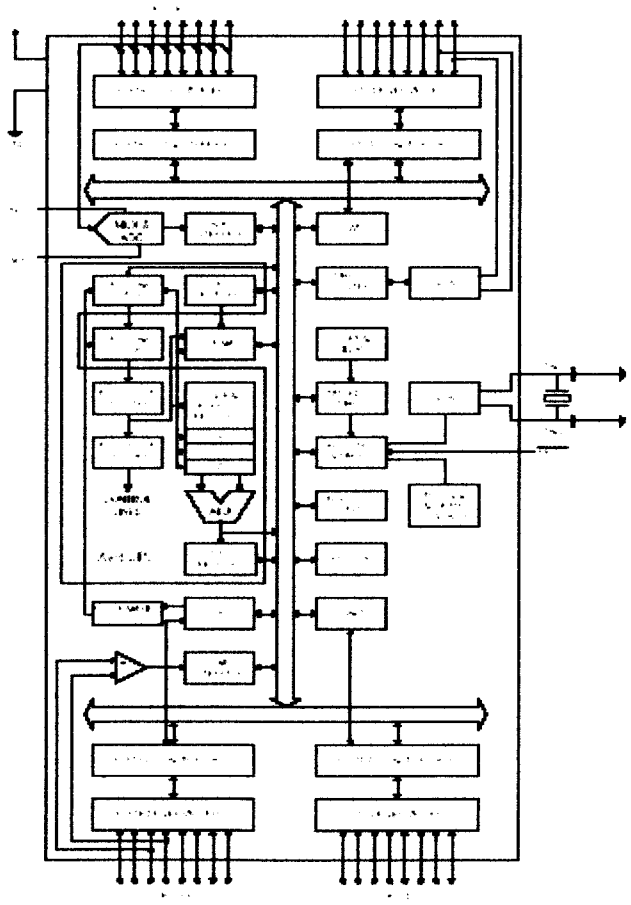
Mikrokontroler adalah suatu terobosan teknologi mikroprosesor dan mikrokomputer, yang mana teknologi ini adalah teknologi semikonduktor dengan kandungan transistor yang lebih banyak, namun hanya membutuhkan ruang yang kecil serta dapat diproduksi secara masal (dalam jumlah banyak) sehingga harganya menjadi lebih murah.

Mikrokontroler ini kemampuan digitalnya menirukan fungsi otak manusia, sehingga meliputi fungsi atau instruksi aritmatika (berhitung), logika (mempertimbangkan suatu kondisi), dan memori. Mikrokontroler ini berbeda halnya dengan mikroprosesor yang hanya pemrosesannya terdiri dari *Central*

Processing Unit (CPU) dan register-register, tanpa memori, tanpa I/O, dan *peripheral* yang dibutuhkan oleh suatu sistem supaya dapat bekerja.

2.3.1 Arsitektur Mikrokontroler AVR ATMega8535

Mikrokontroler ATMega8535 adalah sebuah mikrokontroler yang terdiri dari 8 bit dengan *low power* dan performa tinggi. Termasuk dalam mikrokontroler AVR yang memiliki arsitektur RISC 8 bit, dimana semua instruksi dikemas dalam kode 16-bit dan sebagian besar instruksi dieksekusi dalam satu siklus *clock.*



Gambar 2.6 Blok diagram fungsional ATMega8535

Dari gambar tersebut dapat dilihat bahwa ATMega8535 memiliki bagian sebagai berikut:

1. Saluran I/O sebanyak 32 buah, yaitu *Port* A, *Port* B, *Port* C dan *Port* D.
2. ADC 10 bit sebanyak 8 saluran.
3. Tiga buah *Timer/Counter* dengan kemampuan pembandingan.
4. CPU yang terdiri atas 32 buah register.
5. *Watchdog Timer* dengan isolator internal.
6. SRAM sebesar 512 *byte*.
7. Memori *Flash* sebesar 8 kb dengan kemampuan *Read While Write*.
8. unit interupsi internal dan eksternal.
9. *Port* antarmuka SPI.
10. EEPROM sebesar 512 *byte* yang dapat diprogram saat operasi.
11. Antarmuka komparator analog.
12. *Port* USART untuk komunikasi serial.

2.3.2 Fitur ATMega8535

Kapabilitas detail dari ATMega8535 adalah sebagai berikut:

1. Sistem mikroprosesor 8 bit berbasis *RISC* dengan kecepatan maksimal 16 MHz.
2. Kapabilitas memori *flash* 8 KB, SRAM sebesar 512 *byte*, dan EEPROM (*Electrically Erasable Programmable Read Only Memory*) sebesar 512 *byte*.
3. ADC internal dengan fidelitas 10 bit sebanyak 8 *channel*.

5. *Port* C (PC0..PC7) merupakan pin I/O dua arah dan pin fungsi khusus, yaitu TWI, komparator analog, dan *Timer Oscilator*.
6. *Port* D (PD0..PD7) merupakan pin I/O dua arah dan pin fungsi khusus, yaitu komparator analog, interupsi eksternal dan komunikasi serial.
7. RESET merupakan pin yang digunakan untuk me-reset mikrokontroler.
8. XTAL1 dan XTAL2 merupakan pin masukan *clock* eksternal.
9. AVCC merupakan pin masukan tegangan untuk ADC.
10. AREF merupakan pin masukan tegangan referensi ADC.

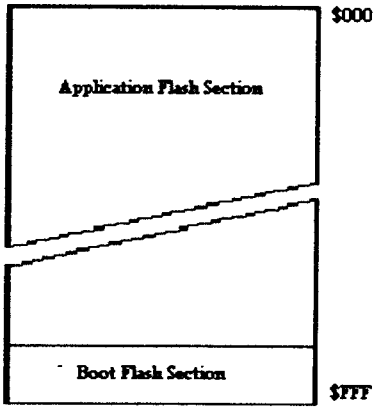
2.3.4 Peta Memori

AVR ATMega8535 memiki ruang pengalamatan memori data dan memori program yang terpisah. Memori data terbagi menjadi 3 bagian, yaitu 32 buah register umum, 64 buah register I/O dan 512 *byte* SRAM *Internal*.

Register keperluan umum menempati *space* data pada alamat terbawah, yaitu $00 sampai $1F. Sementara itu, register khusus untuk menangani I/O dan kontrol terhadap mikrokontroler menempati 64 alamat berikut, yaitu mulai dari $20 hingga $5F. Register tersebut merupakan register khusus digunakan untuk mengatur fungsi terhadap berbagai peripheral mikrokontroler, seperti kontrol register, *timer/counter*, fungsi-fungsi I/O dan sebagainya.

Memori program yang terletak dalam *Flash* PEROM tersusun dalam *word* atau 2 *byte* karena setiap instruksi memiliki lebar 16-bit atau 32-bit. AVR ATMega8535 memiliki 4K *Byte* X16-bit *Flash* PEROM dengan alamat mulai dari

$000 sampai $FFF. AVR tersebut memiliki 12-bit *Program Counter* (PC) sehingga mampu mengalamati isi *Flash.*



Gambar 2.8 Memori ATMega8535

Selain itu, AVR ATMega8535 juga memiliki memori data berupa EEPROM 8-bit sebanyak 512 *byte*. Alamat EEPROM dimulai dari $000 sampai $1FF.

2.4 Perangkat Lunak

Perangkat lunak yang digunakan adalah BORLAND DELPHI 7. Delphi merupakan pemrograman terstruktur yang berbasis pada obyek Pascal dari Borland, bekerja pada ruang lingkup sistem operasi ***Window.*** Struktur bahasanya dengan bahasa objek pascal ini sangat mendukung untuk pemrograman **OOP** (*Object-Oriented Programing*), maksudnya perluasan atas pemrograman terstruktur yang mengutamakan pemakaian-ulang dan enkapsulasi data (kombinasi data dan fungsi-fungsi ke dalam sebuah unit tunggal) berdasarkan fungsinya, Delphi juga mempunyai fungsi untuk memberikan fasilitas pembuatan

aplikasi visual, sehingga meningkatkan produktivitas dalam pembuatan program yang meliputi kualitas pengembangan visual, kecepatan kompilasi, kekuatan bahasa pemrograman, fleksibelitas terhadap arsitektur basis data, dan pola desain dan pemakaian yang diwujudkan oleh *framework*-nya. Tugas akhir ini menggunakan Delphi versi 7.0.

IDE (*Integrated Development Environment*) merupakan lingkungan tempat semua *tools* yang diperlukan untuk desain, menjalankan, dan mencoba aplikasi yang disajikan dan terhubung dengan baik, sehingga memudahkan pengembangan program. IDE ini akan muncul pertama kali saat membuka atau akan menjalankan program Delphi, IDE terdiri atas *main window, component palette, toolbar, object inspector, form desainer, code editor,* dan *code explorer,* sedangkan tampilan IDE seperti berikut.

Main Window (jendela utama) mempunyai tiga bagian yaitu menu utama, *toolbar,* dan *component palette.* Menu utama merupakan bagian yang digunakan untuk mendukung jalannya pembuatan program atau aplikasi, meliputi *file, edit search, view, project, run, component, database, tools, help.* Bagian menu utama tersebut mengandung perintah-perintah yang akan digunakan untuk mendukung pembuatan program atau aplikasi tersebut walaupun di dalamnya ada sebagaian yang sudah ditampilkan dalam *toolbar* atau *speedbar. Toobar* merupakan bagian menu utama yang dibuat dalam bentuk *icon* dengan tujuan untuk pemanfaatan perintah dengan cepat tanpa harus membuka menu utama, sehingga lebih mempermudah dan mempercepat dalam membuat aplikasi atau program. *Component palette* merupakan *toolbar* yang berisi *page control* dengan semua

komponen tersebut yang digunakan untuk membantu dalam penghubung antara program dengan aplikasi yang diinginkan atau sebagai *interface*nya.

Form desainer merupakan jendela yang digunakan untuk membuat aplikasi atau meletakkan komponen yang akan digunakan dalam aplikasi. Penggunaan sebagai berikut dengan cara memilih komponen dari *component pallette* dan meletakkan ke dalam *form,* setelah di dalam *form* maka komponen tersebut dapat diatur posisinya atau ukuran dengan *mouse,* serta dapat mengubah tampilan dan perilaku komponen dengan menggunakan *object inspector* dan *code editor.*

Object inspector terdiri atas dua bagian, yaitu bagian *tab properties* yang berfungsi untuk membuat atau *property* yang dimiliki oleh suatu *item* sesuai dengan yang diinginkan. Bagian *tab events* berisi tentang *event* yang dapat direspon oleh suatu obyek dalam *form.* Jadi setiap obyek yang sudah ada dalam *form* dapat diatur dengan menggunakan *tab properties* serta dapat diperlakukan sesuai dengan keinginan melalui *tab events.*

Code editor merupakan jendela penyunting yang digunakan untuk menuliskan program Delphi. Editor Delphi mempunyai fasilitas *highlight* untuk memudahkan menemukan kesalahan, fasilitas kerangka program sehingga dalam membuat program tidak perlu menuliskan program seluruhnya. Nama *file* yang sedang disunting terdapat dalam *title bar.* Menu lokal dari *code editor* menyediakan berbagai *option* untuk mempermudahkan dalam menuliskan program.

Code explorer digunakan untuk mempermudah melakukan navigasi terhadap *file unit. Code explorer* berisi pohon yang menampilkan semua *type, class, property, method, variable global* dan *rutin global* yang didefinisikan dalam unit serta menampilkan semua unit yang ada di *clausa uses.*

2.5 Universal Serial Bus (USB)

Universal Serial Bus (USB) : Sebuah bus I/O (*input/output*) yang dapat mentransfer data hingga 12 megabit per detik. USB versi 2.0 yang baru saja dikeluarkan mampu memberikan tingkat kinerja dan kecepatan yang sebanding dengan bus kecepatan-tinggi semacam IEEE 1394. Beberapa hal yang perlu diketahui tentang USB:

1. Lebih cepat dibanding *port* paralel atau serial dengan kecepatan transfer hingga 12 mbps.
2. Dapat mengkoneksikan hingga 127 periperal.
3. Diterima secara luas.
4. Tidak cocok untuk periperal dengan *bandwith* tinggi.
5. Membutuhkan Windows 98 ke atas untuk kompatibilitas secara penuh.

Transfer data pada USB dikirim dalam bentuk paket, sementara *port* paralel dan serial mentransfernya dalam bentuk bit individual. Sebagai contoh, bila ingin menyimpan sebuah *file* pada sebuah *drive Zip* USB, pertama-tama PC akan memotong-motong *file* tersebut menjadi potongan-potongan sebesar 64 *byte* setiap potongan menyertakan informasi pengalamatan dan data itu sendiri dan kemudian mengirimkan potongan tersebut ke *port* USB. Kecepatan transfer USB

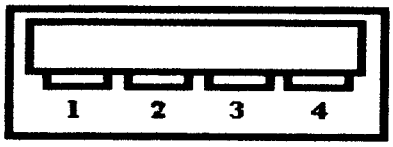
Dalam kasus ini, tidak diperlukan *firmware* yang khusus, dan tidak harus mengetahui cara kerja USB karena *vendor* dari *converter* akan menawarkan *driver* khusus untuk solusi menyeluruh. Kerugiannya adalah harga yang lebih tinggi untuk sebuah sistem.

Oleh karena itu solusi yang dibutuhkan adalah menerapkan USB ke dalam sebuah mikrokontroler yang lebih murah dengan penyamaan protokol USB ke dalam sebuah *firmware* mikrokontroler. Kendala utama dari rancangan ini adalah mendapatkan kecepatan yang cukup. Kecepatan USB berbeda-beda, yaitu: kecepatan rendah (1.5 Mbit/s), kecepatan penuh (12Mbit/s), kecepatan tinggi (480 Mbit/s). Kecepatan maksimal dari mikrokontroler juga terbatas, yakni : AT89C2051 – 2 MIPS = 24 MHz/ (12 siklus/inst), PIC16f84 – 5 MIPS = 20 MHz/ (4 siklus/inst), AT90S23x3 – 10 MIPS = 10 MHz/ (1 siklus/inst). Ada beberapa mikrokontroler dengan kecepatan yang lebih tinggi, namun dengan kecepatan itu cenderung membuat mikrokontroler jarang tersedia dan harganya relatif mahal, ukurannya juga lebih besar. Karena alasan-alasan tersebut, maka AT90S1200/AT90S23x3 mewakili mikrokontroler yang paling murah dan setara dengan USB kecepatan rendah, tetapi mikrokontroler ini tidak sesuai untuk USB dengan kecepatan yang lebih tinggi.

2.5.2 Prinsip Kerja USB

Secara fisik USB *interface* mempunyai 4 kabel; 2 kabel untuk *power* dari *hardware* / alat lain (Vcc dan GND) dan 2 kabel lainnya untuk sinyal (DATA+ dan DATA-). Kabel *power* memberikan tegangan sekitar 5 volt dan maksimal 500mA. Alat mendapat suplai tegangan dari Vcc dan GND. Kabel sinyal diberi

name DATA+ dan DATA- berperan untuk mengatur komunikasi antara komputer (*host*) dengan alat yang terhubung (*device*). Sinyal dari kabel-kabel ini bersifat dua arah. Level tegangan berbeda ketika DATA+ pada level tinggi, sedangkan DATA- pada level yang rendah, tetapi ada beberapa saatnya dimana DATA+ dan DATA- berada pada level yang sama (EOP-*status idle*). Bentuk diagram dari USB adalah sebagai berikut:



Gambar 2.9 Diagram konektor USB

Semua jenis konektor USB dihubungkan dengan 4 kabel berikut:

Tabel 2.4 Pengkabelan USB

Nomor Pin	Warna Kabel	Fungsi
1	Merah	V_{BUS} (5 volt)
2	Putih	D-
3	Hijau	D+
4	Hitam	*ground*

2.5.3 Karakteristik Elektris USB

Rentang tegangan sinyal USB adalah 0,3 volt hingga 3,6 volt (pada beban 1,5 kΩ). Logika tinggi didapat jika tegangan sudah melebihi 2,8 volt terhadap

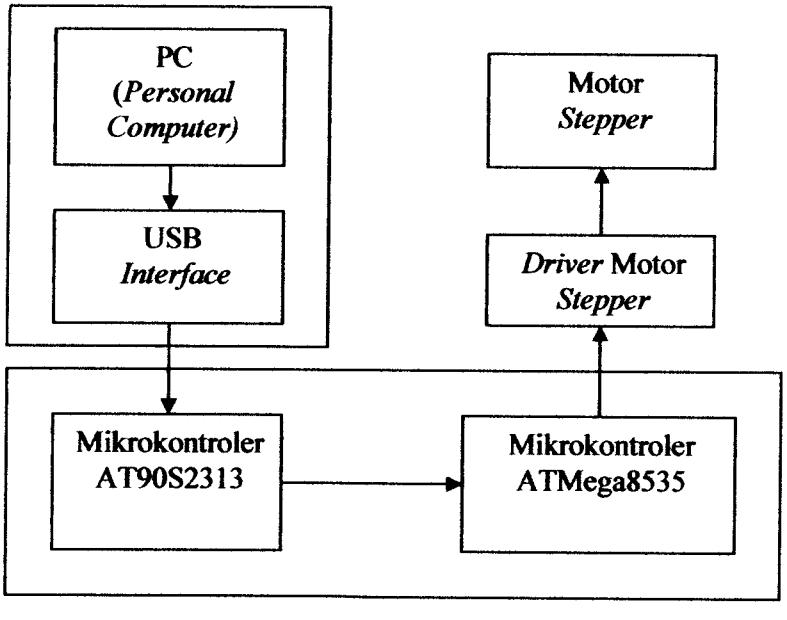
GND pada beban diferensial "1" dikirim dengan menarik D+ hingga lebih besar dari 2,8 volt dengan sebuah resistor 15kΩ terhubung ke GND dan sekaligus menarik D- hingga dibawah 0,3 volt dengan sebuah resistor 1,5 kΩ terhubung ke 3,6 volt. Hal yang sama diferensial "0" adalah D- lebih besar dari 2,8 volt dan D+ lebih rendah dari 0,3 volt dengan resistor *pull-up* dan *pull-down* yang sama. Di bagian penerima, diferensial "1" didefinisikan sebagai D+ lebih besar 200mV dari D-, dan diferensial "0" berarti D+ lebih kecil dari 2000mV dibanding D-. PAda USB kecepatan tinggi (480 MBits/s) digunakan sember arus tetap 17,78 mA untuk mengurangi *noise*.

BAB III

PERANCANGAN SISTEM

Dalam bab III ini akan dibahas mengenai perancangan sistem yang di dalamnya terdapat perancangan rangkaian elektronika dan sistem pengendalian motor *stepper* dengan komunikasi USB berbasis Delphi berdasar pada teori-teori yang telah di bahas sebelumnya. Perancangan sistem tersebut dapat digambarkan melalui diagram blok seperti terlihat pada gambar 3.1

3.1 Diagram Blok Sistem



Gambar 3.1 Diagram blok sistem

Adapun penjelasan dari diagram blok tersebut adalah sebagai berikut:

PC *(Personal Computer)*

Blok ini berfungsi sebagai *interface* (antarmuka) antara user dengan komputer/PC sebagai pengendali. Pengendali yang dimaksud adalah dengan menggunakan bahasa Delphi, data-data yang diperlukan diproses, kemudian komputer/PC akan mengirimkan data yang diperoleh ke AT90S2313 yang berfungsi sebagai *pull-up* data pada mikrokontroler ATMega8535. Pengiriman data ini ditransfer melalui komunikasi USB, yang sintaksnya telah disediakan pada Delphi.

USB *(Universal Serial Bus)*

Sebuah bus I/O (*input/output*) yang dapat mentransfer data hingga 12 megabit per detik. Digunakan sebagai penghubung antara PC dengan Mikrokontroler AT90S2313 yang akan meneruskan data-data yang telah diproses ke mikrokontroler ATMega8535.

Mikrokontroler AT90S2313

Mikrokontroler AT90S2313 memilik instruksi yang dihimpun dalam kode 16-bit dan sebagian besar instruksinya dieksekusi dalam 1 (satu) siklus *clock.* berfungsi sebagai *pull-up* data-data instruksi pada ATMega8535.

Mikrokontroler ATMega8535

Pada blok ini mengendalikan arah gerakan motor *stepper* ke kiri dan ke kanan dan menentukan besarnya derajat arah yang diinginkan. ATMega8535 sendiri berfungsi untuk mengubah data-data dari komputer (PC) agar dapat dibaca oleh motor *stepper.*

Perancangan Perangkat Lunak

Perangkat lunak sisi PC yaitu program Delphi 7. *Source* program dimodifikasi untuk 3 perintah dasar yaitu memutar ke kiri, memutar ke kanan, dan motor berhenti. Serta 5 perintah derajat sudut yaitu 10, 30, 60, 90, 180 derajat.

Driver Motor *Stepper*

Pengarah motor *Stepper* menerima isyarat *low-level* dari pengindeks atau sistem pengendalinya dan mengkonversi ke dalam energi listrik untuk menjalankan motor *stepper* tersebut.

Motor *Stepper*

Motor *stepper* ini adalah sebagai objek, yang mana arah putaran motor atau besarnya derajat arah yang diinginkan diatur. Kecepatan motor *stepper* dapat diatur dengan menentukan *step* yang diinginkan.

3.2 Perancangan Perangkat Keras

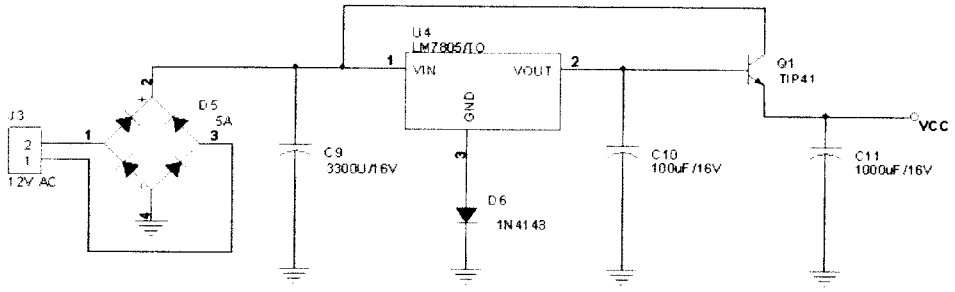
Pada penelitian ini perangkat keras yang digunakan berupa motor *stepper*, yang didukung oleh perlengkapan-perlengkapan yang lain seperti : transformator atau catu daya, sistem mikrokontroler AT90S2313, mikrokontroler ATMega8535, dan rangkaian regulator yang berfungsi sebagai pembangkit tegangan pada motor *stepper*.

3.2.1 Catu Daya

Catu daya merupakan bagian yang sangat penting pada rangkaian karena tanpa catu daya alat ini tidak dapat bekerja. Motor *stepper* memerlukan catu daya yang dapat memberikan tegangan sebesar 5 V.

Mempertahankan suatu level tegangan yang konstan sangat diperlukan dalam rangkaian catu daya ini, dengan demikian rangkaian catu daya pada tugas akhir ini menggunakan rangkaian regulator tegangan (*voltage regulator)* yang mengandung sejumlah rangkaian untuk tegangan referensi, alat pengontrol, penguat komparator, dan pelindung tegangan berlebih (*overload protection*).

Pada alat ini pelindung tegangan berlebih digunakan regulator jenis positif regulator dengan tipe LM7805 untuk penstabil tegangan 5VDC



Gambar 3.2 Rangkaian regulator tegangan

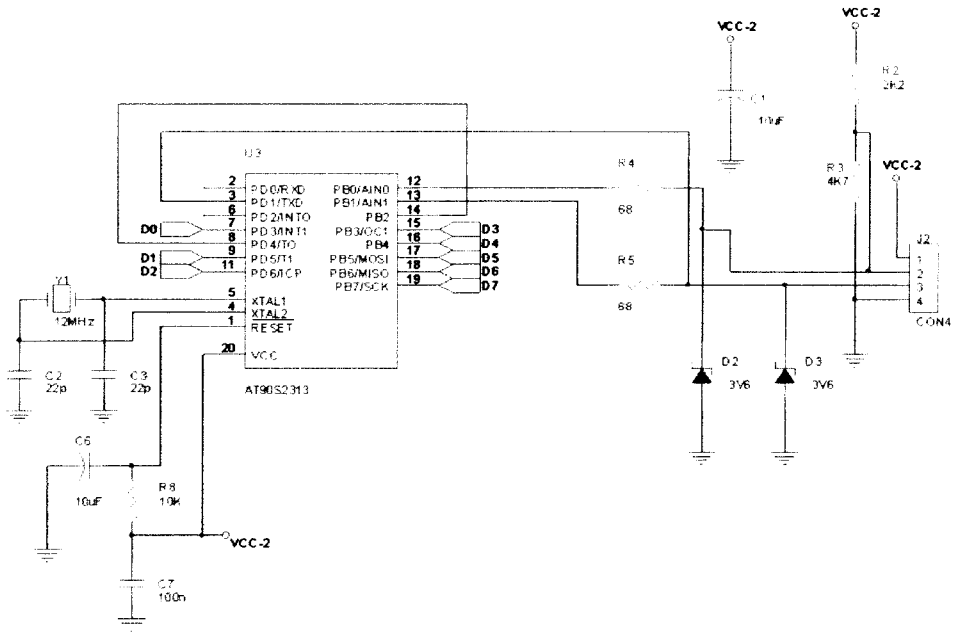
Dalam perancangan ini dipergunakan beberapa komponen seperti pada Gambar 3.2 dengan penjelasan sebagai berikut :

1. Dioda *Bridge*, merupakan penyearah yang mengubah arus AC dari transformator menjadi arus DC, namun sinyal keluarannya masih mengandung riak atau *ripple*.
2. Kapasitor elektrolit C9 dengan nilai 3300uF berfungsi untuk memperhalus dan menghilangkan riak keluaran dari penyearah dioda *bridge*.

3. IC LM7805 merupakan regulator tegangan yang berfungsi untuk menurunkan tegangan dari 12V menjadi 5V, serta menyetabilkan keluarannya.
4. Dioda 1N4148 berfungsi untuk menjaga keluaran LM7805 agar berada di 5V, karena tanpa dioda ini keluaran LM7805 hanyalah sebesar 4,8V.
5. Kapasitor elektrolit 100uF berfungsi untuk melakukan filtrasi keluaran dari LM7805 agar lebih halus sebelum diumpankan ke penguat arus TIP 41.
6. TIP 41 merupakan transistor penguat arus untuk memperbesar arus keluaran dari catu daya, hal ini diperlukan untuk menyuplai mikrokontroler serta motor *stepper* yang membutuhkan arus yang cukup besar.

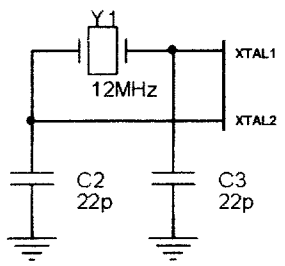
3.2.2 Mikrokontroler AT90S2313

Mikrokontroler AT90S2313 digunakan sebagai *interface* antara PC/komputer dengan mikrokontroler ATMega8535 sebelum diproses ke motor *stepper*. Perangkaian dan konfigurasi pin-pin dari mikrokontroler ini telah dijelaskan pada bab sebelumnya. Sistem minimum mikrokontroler AT90S2313 dapat dilihat pada Gambar 3.3 berikut.



Gambar 3.3 Sistem minimum mikrokontroler AT90S2313

Rangkaian sistem minimum dari mikrokontroler AT90S2313 terdiri dari rangkaian osilator dan *power on reset*. Rangkaian osilator ini digunakan untuk membangkitkan *clock*/detak. Kristal yang dipakai adalah 12 MHz, untuk menghasilkan kecepatan transmisi USB 1.0 sebesar 12 MHz / 8 = 1,5 MHz.



Gambar 3.4 Rangkaian osilator eksternal

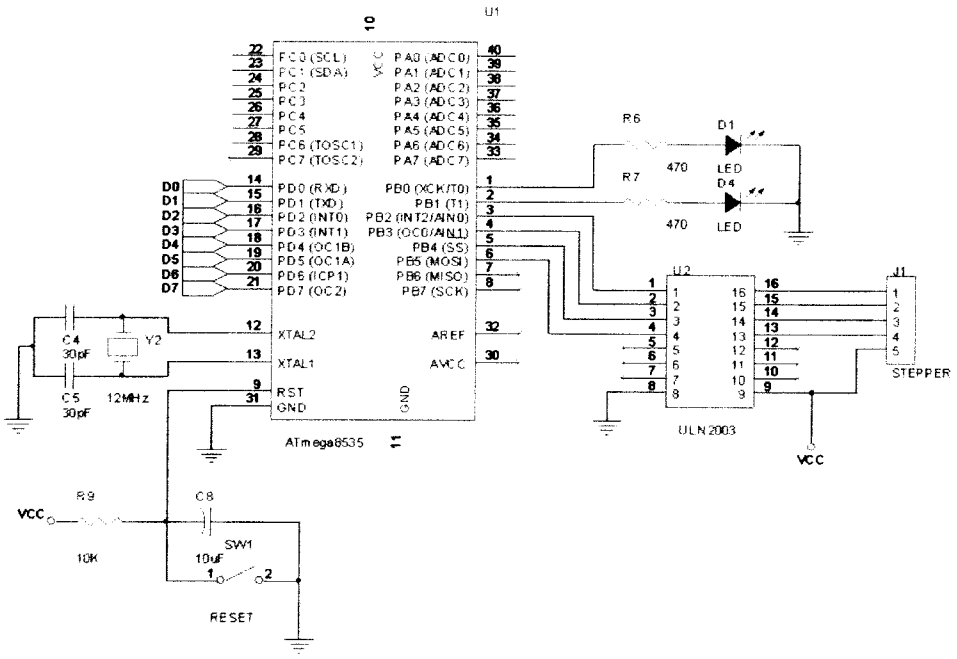
arus hingga 20 mA.dan men-*drive* LED secara langsung. *Port* PB5 dan PB6 sebagai jalur *input* dan *output* data untuk *download* memori.

***Port* D Mikrokontroler AT90S2313**

Pada *Port* D0 berfungsi untuk menerima data *input* dari USB komputer, *port* D1 berfungsi mengirim data *output* kembali ke komputer melalui USB, *port* PD4 dan PD5 berfungsi sebagai sumber *clock* yang digunakan.

3.2.3 Mikrokontroler ATMega8535

Sistem minimum mikrokontroler ATMega8535 digunakan sebagai *interface* antara PC dengan motor *stepper* yang data-data sebelumnya telah di *pull-up* oleh mikrokontroler AT90S2313. Perangkaian dan konfigurasi pin-pin dari mikrokontroler ini telah dijelaskan pada bab sebelumnya.



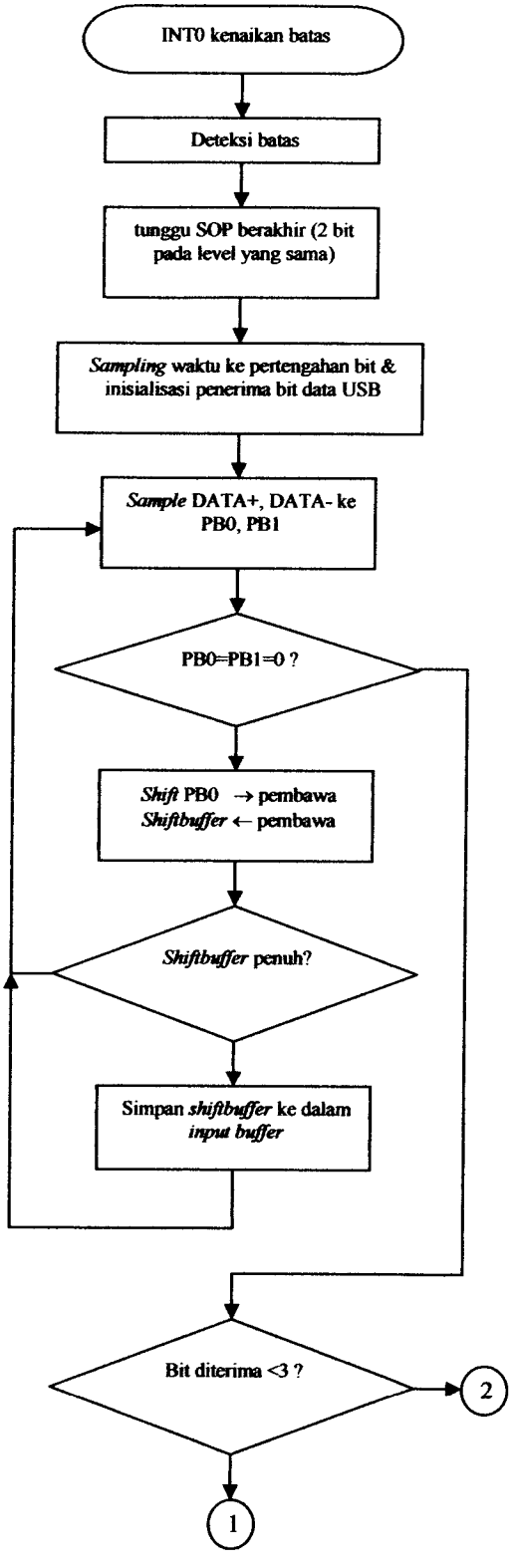
Gambar 3.6 Sistem minimum mikrokontroler ATMega8535

3.3 Perancangan Perangkat Lunak

Dalam perancangan perangkat lunak besarnya derajat serta arah putaran motor *stepper* ditentukan dalam program yang dapat dipilih oleh *user*. Untuk alat ini telah diberi ketentuan kecepatan motor *stepper* 10 derajat per step.

Penulisan program yang digunakan dalam rangkaian ini adalah dengan bahasa Delphi menggunakan bahasa *assembler* yang diperuntukkan bagi mikrokontroler ATMega8535.

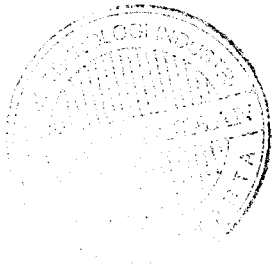
Untuk dapat mengetahui lebih jelas dari proses pemrograman pada mikrokontroler, dari gambar diagram alir, jalannya program pada mikrokontroler akan akan terimplementasi pada motor *stepper*. maka berikut akan ditampilkan *flowchart* atau diagram alirnya.


INT0 kenaikan batas
Deteksi batas
tunggu SOP berakhir (2 bit pada level yang sama)
Sampling waktu ke pertengahan bit & inisialisasi penerima bit data USB
Sample DATA+, DATA- ke PB0, PB1
PB0=PB1=0 ?
Shift PB0 → pembawa
Shiftbuffer ← pembawa
Shiftbuffer penuh?
Simpan shiftbuffer ke dalam input buffer
Bit diterima <3 ?
2
1

memulai penerimaan serial data USB (bisa juga disebut "Resepsi USB"). Interupsi eksternal terjadi pada batas peningkatan di pin INT0 (batas peningkatan menunjukkan awal dari pola *sync* dari sebuah paket USB). Hal ini mengaktifkan rutin penerimaan USB.

Pertama-tama *sampling* data harus disinkronkan ke dalam pertengahan lebar bit, hal ini dikerjakan sesuai pola *sync* (berupa gelombang kotak). Karena durasi bit hanya 8 siklus dari *clock* XTAL dan interupsi yang terjadi dapat ditunda (+/- 4 siklus), batas sinkronisasi dalam pola *sync* harus dijalankan dengan hati-hati. Akhir dari pola *sync* dan awal dari bit data dideteksi menurut dua bit terakhir dengan level rendah dalam paket *sync*.

Setelah ini, *sampling* data dimulai. *Sampling* dijalankan di pertengahan bit. Karena kecepatan data 1.5Mbit/detik (1.5MHz) dan kecepatan mikrokontroler 12MHz, maka kita hanya mempunyai 8 siklus untuk sampling bit data, menyimpannya ke dalam *buffer bit*, menggilir *buffer* bit , memeriksa jika seluruh *byte* telah diterima, menyimpan bit ke dalam SRAM, dan memeriksa untuk EOP. Proses ini mungkin merupakan bagian paling penting dari *firmware*, segala hal harus dikerjakan secara sinkron pada saat yang tepat. Ketika seluruh paket USB telah diterima, penguraian kode paket harus dijalankan. Pertama-tama, alat harus menentukan dengan cepat tipe paket (*SETUP, IN, OUT, DATA*) dan menerima alamat USB. Penguraian kode yang cepat ini harus dijalankan di dalam rutin interupsi karena sebuah jawaban dibutuhkan sangat cepat setelah penerimaan paket USB (alat harus menjawab dengan ACK ketika paket dengan alamat alat



telah diterima, dan dengan NAK ketika paket adalah untuk alat tetapi tidak ada jawaban yang sudah siap).

Pada akhir rutin penerimaan (setelah paket ACK/NAK dikirimkan) sampel data dari *buffer* harus dikopikan ke dalam *buffer* yang lain dimana penguraian kode akan dijalankan. Ini dilakukan agar *buffer* penerima bisa menerima paket baru.

Selama proses penerimaan tipe paket diuraikan kodenya dan *Flag* koresponden di set. *Flag* ini di uji dalam *loop* program utama dan menurut nilainya, tindakan yang sesuai akan diambil dan jawaban koresponden akan disiapkan dengan tanpa memperhatikan ketentuan kecepatan mikrokontroler.

INT0 harus diaktifkan untuk menjaga waktu invokasi yang sangat cepat pada semua rutin *firmware*, sehingga tidak ada interupsi yang menyebabkan eksekusi interupsi yang lain tidak berjalan (sebagai contoh serial *line* interupsi yang diterima). INT0 harus bisa melakukan proses ini, penerimaan yang cepat di dalam INT0 dalam rutin interupsi sangat penting, dan penting juga untuk mengoptimalkan *firmware* untuk kecepatan dan waktu yang tepat. Yang terpenting adalah mengoptimalkan register *backup* dalam rutin interupsi.

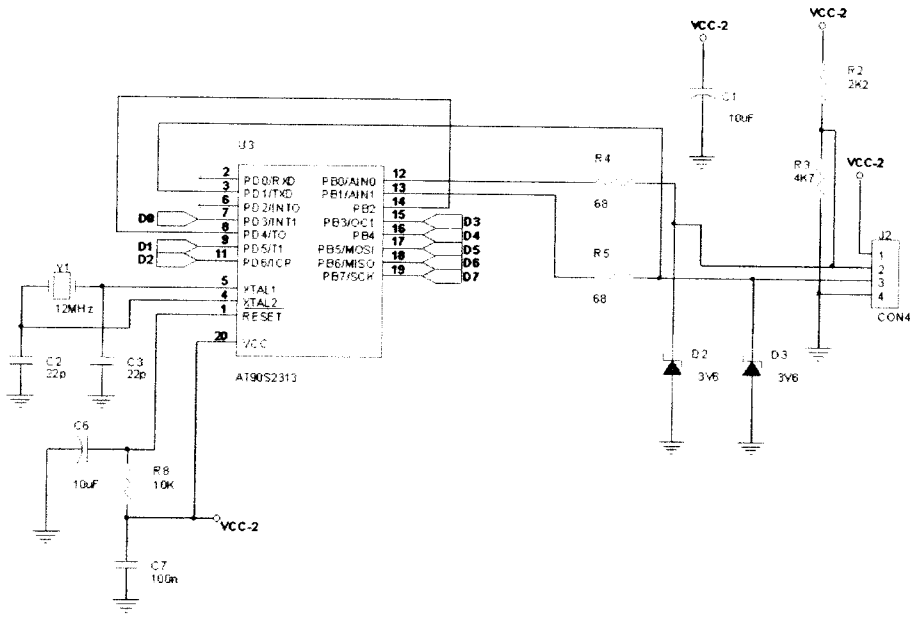
BAB IV

ANALISA DAN PEMBAHASAN

4.1 Analisa *Hardware* (perangkat keras)

Rangkaian *hardware* terbagi menjadi tiga bagian, yaitu rangkaian mikrokontroler AT90S2313, mikrokontroler ATMega8535, dan rangkaian regulator (catu daya).Berikut analisa rangkaian tersebut.

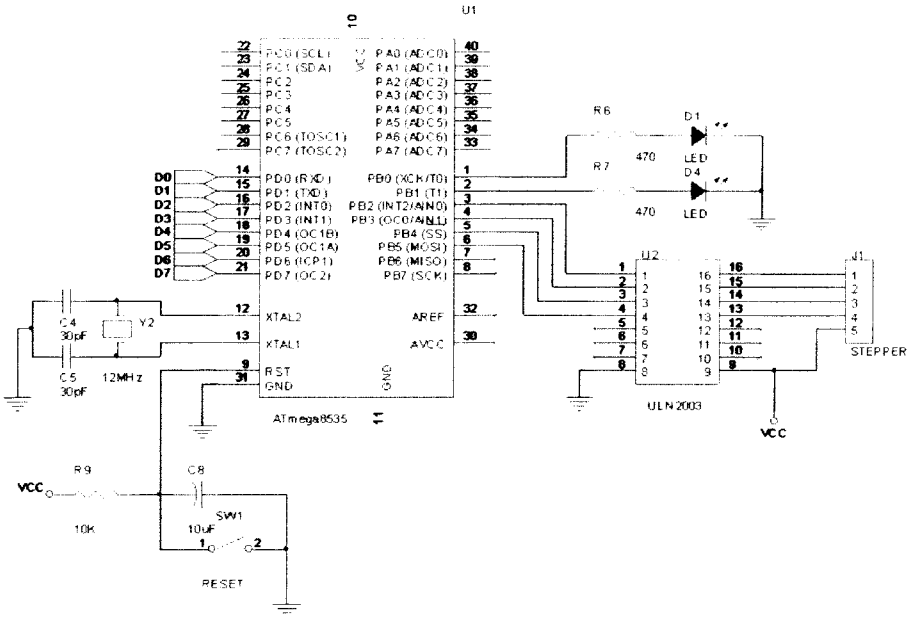
4.1.1 Analisa Rangkaian Mikrokontroler AT90S2313



Gambar 4.1 Rangkaian mikrokontroler AT90S2313

Rangkaian sistem minimum dari mikrokontroler AT90S2313 terdiri dari rangkaian osilator dan *power on reset*. Rangkaian osilator ini digunakan untuk membangkitkan *clock*/detak. Kristal yang dipakai adalah 12 MHz, untuk menghasilkan kecepatan transmisi USB 1.0 sebesar 12 MHz / 8 = 1,5MHz.

4.1.2 Analisa Rangkaian Mikrokontroler ATMega8535



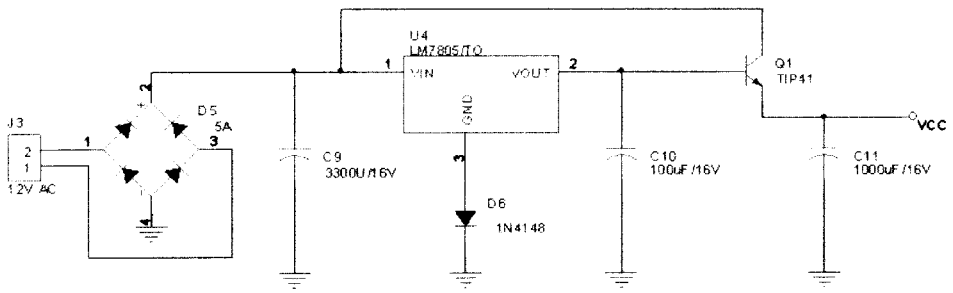
Gambar 4.2 Rangkaian mikrokontroler ATMega8535

Rangkaian sistem minimum dari mikrokontroler ATMega8535 terdiri dari rangkaian osilator dan *power on reset*. Rangkaian osilator ini digunakan untuk membangkitkan *clock*/detak..

Pada saat sumber tegangan diaktifkan kapasitor terhubung singkat sehingga arus mengalir dari VCC langsung ke kaki RST sehingga reset berlogika 1, kemudian kapasitor terisi hingga tegangan pada kapasitor sama dengan VCC. Pada saat itu kapasitor terisi penuh. Dengan demikian tegangan reset akan turun menjadi 0 sehingga kaki RST berlogika 0.

4.1.3 Analisa Rangkaian Regulator Tegangan

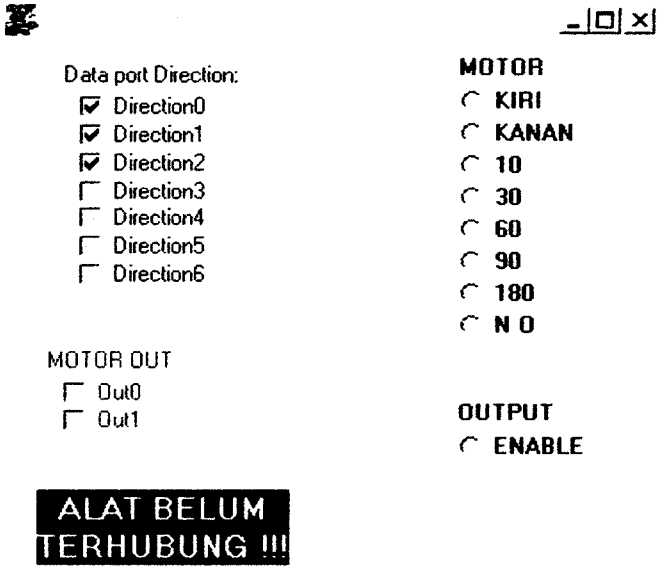
Pada rangkaian regulator ini tegangan bolak-balik(AC) dari transformator sebesar 12V akan disearahkan oleh dioda *bridge*. Keluaran dari penyearah ini akan diturunkan menjadi 5V dan distabilkan oleh IC LM7805. Untuk menyuplai rangkaian dengan kebutuhan arus yang cukup besar maka dibutuhkan penguat arus yang pada rangkaian ini mempergunakan transistor TIP 41 yang mempunyai karakteristik sebagai penguat arus.



Gambar 4.3 Rangkaian *power* suplai dengan regulator

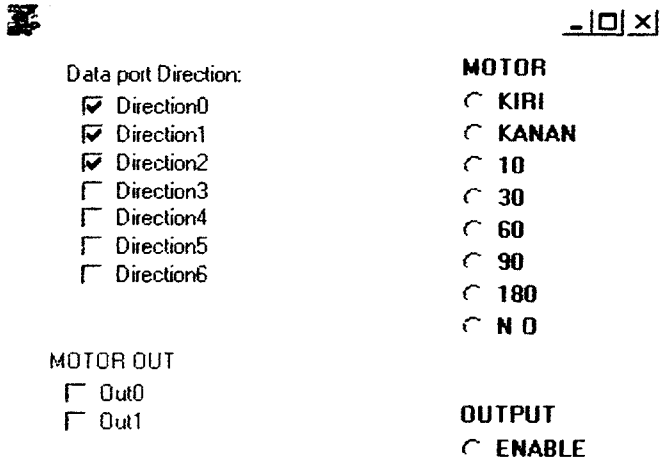
4.2 Analisa Software (perangkat lunak)

Berikut tampilan perangkat lunak uji coba penggerak Motor *stepper* dengan komunikasi via *port* USB menggunakan Delphi sebagai *development tool.*



Gambar 4.4 Tampilan delphi awal

Alat (*device*) dihubungkan dengan *port* USB pada PC, jika tulisan "alat belum terhubung" hilang maka keadaan ini menunjukkan alat telah berhasil terhubung secara baik dengan PC melalui USB *interface*. *Data port direction* pada *form* harus diaktifkan setiap uji coba akan dijalankan. Piranti ini berfungsi sebagai *pull-up* data pada mikrokontroler AT90S2313 ke mikrokontroler ATMega8535. *Pull-up* data berfungsi untuk menjaga agar sinyal digital tidak mengambang antara logika 0 dan logika 1. Tampilan pada program Delphi akan seperti berikut.



Gambar 4.5 Tampilan delphi akhir

Dengan demikian alat (*device*) tersebut dapat diuji apakah hasil dari pengujian arah perputaran dan derajat sudut yang diinginkan sesuai dengan masukan yang diberikan pada program Delphi. Jika alat telah terhubung seperti keadaan diatas maka langkah selanjutnya yaitu memilih salah satu *data port direction*, cukup salah satu yang dipilih maka seluruh *data port direction* akan tertandai. Kemudian pilih arah dan sudut perputaran yang diinginkan.

Sebagai contoh pilih arah kanan lalu pilih sudut 90^0 maka alat akan bergerak ke kanan sebesar 90^0, setelah itu jarum penunjuk arah diposisikan pada 0^0 dengan menggesernya atau dengan memberikan masukan ke arah yang berlawanan yaitu memilih arah kiri dan sudut 90^0 maka jarum indikator akan kembali ke posisi awal. Begitu pula dengan sudut perputaran lainnya, metode pengoperasiannya juga sama seperti yang sudah dijelaskan.

4.2.1 *Software Assembly* Mikrokontroler AT90S2313

Deskripsi singkat dari subrutin *firmware* yang digunakan pada mikrokontroler AT90S2313 akan dijelaskan secara singkat sebagai berikut.

Reset:

Inisialisasi dari sumber-sumber AVR mikrokontroler yaitu *Stack*, serial *line*, *buffer* USB, interupsi-interupsi, dan lain-lain.

Main:

Loop program utama, mengawasi nilai *flag*, pengesetan *flag*, menjalankan perintah yang diminta. Sebagai tambahan, rutin ini mengawasi USB *reset* pada *line* data dan menginisialisasi ulang *Interface* USB dan mikrokontroler.

Int0Handler:

Merupakan servis rutin interupsi untuk interupsi eksternal INT0. Mesin penerimaan/pengiriman. Emulasi dari *line* data USB, pengendalian data ke dalam *buffer*, memutuskan alamat USB, pengenalan paket, pengiriman jawaban ke komputer, secara mendasar merupakan jantung dari mesin USB.

MyNewUSBAddress:

Didapatkan dari rutin penerimaan INT0 jika ada permintaan hadir untuk mengubah alamat USB. Alamat diubah dan dikodekan ke NRZI yang setara dengan penguraian kode alamat tercepat selama penerimaan paket USB disiapkan.

InitNAKbuffer:

Memulai *buffer* dalam RAM dengan data NAK. *Buffer* ini secara periodik mengirim jawaban sehingga menjaga alat dalam memori.

SendNAK:

Mengirim paket NAK ke *line* USB.

ComposeSTALL:

Memulai *buffer* dalam RAM dengan data STALL. *Buffer* ini secara periodik mengirimkan jawaban sehingga menjaga alat dalam memori.

DecodeNRZI:

Menjalankan penguraian kode NRZI. Data dari *line* USB ke dalam *buffer* adalah kode NRZI. Rutin ini menghilangkan kode NRZI dari data.

BitStuff:

Menghilangkan atau menambahkan *bitstuffing* pada data yang diterima USB. *Bitstuffing* ditambahkan oleh *hardware* host (komputer) menurut spesifikasi USB untuk memastikan sinkronisasi dalam *sampling* data. Rutin ini menghasilkan data tanpa *bitstuffing* atau data untuk pengiriman dengan *bitstuffing*.

ShiftInsertBuffer:

Rutin *auxiliary* digunakan ketika menjalankan penambahan *bitstuffing*. Menambahkan satu bit pada data *output buffer* sehingga meningkatkan panjang *buffer*. *Remainder* yang tertinggal dalam *buffer* dikeluarkan.

ShiftDeleteBuffer:

Rutin *auxiliary* digunakan ketika menjalankan penghilangan *bitstuffing*. Menghilangkan satu bit dari *output buffer* sehingga menurunkan panjang *buffer*. *Remainder* yang tertinggal di dalam *buffer* akan dimasukkan.

MirrorInBufferBytes:

Penukaran *order* pada *byte* karena data diterima dari *line* USB ke *buffer* dalam susunan yang terbalik (LSB/MSB).

CheckCRCIn:

Menjalankan CRC (*Check Redundancy* secara siklik) pada paket data yang diterima. CRC ditambahkan ke paket USB untuk mendeteksi korupsi data.

AddCRCOut:

Menambahkan CRC *field* ke dalam paket data *output*. CRC dihitung sesuai dengan spesifikasi USB dari *field* USB yang diberikan.

CheckCRC:

Rutin *auxiliary* digunakan untuk memeriksa CRC dan penambahan.

LoadDescriptorFromROM:

Menyimpan data dari ROM ke *output buffer* USB (sebagai jawaban USB)

LoadDescriptorFromZeroInsert:

Menyimpan data dari ROM ke *output buffer* USB (sebagai jawaban USB) tapi setiap *byte* ditambahkan sebagai *zero*. Ini digunakan ketika deskriptor *string* dalam format UNICODE diminta (penyimpanan ROM).

LoadDescriptorFromSRAM:

Menyimpan data dari RAM ke *output buffer* USB (sebagai jawaban USB).

LoadDescriptorFromEEPROM:

Menyimpan data dari data EEPROM ke *output buffer* USB (sebagai jawaban USB).

LoadxxxDescriptor:

Menjalankan seleksi untuk lokasi sumber jawaban : ROM, RAM atau EEPROM.

PrepareUSBOutAnswer:

Menyiapkan jawaban USB ke *output buffer* sesuai permintaan oleh *host* USB, dan menjalankan aksi yang diminta. Menambahkan *bitstuffing* untuk menjawab.

PrepareUSBAnswer:

Rutin yang utama adalah menjalankan perintah yang diminta dan menyiapkan jawaban koresponden. Pertama-tama rutin akan menentukan aksi mana yang akan dijalankan, menemukan angka fungsi dari paket data *input* dan kemudian menjalankan fungsi yang diminta. Parameter fungsi ditempatkan dalam paket data *input*. Rutin dibagi menjadi 2 bagian: permintaan standar dan permintaan spesifik *vendor*. Permintaan standar adalah penting dan digambarkan dalam spesifikasi USB (*SET_ADDRESS, GET_DESCRIPTOR,...*). Permintaan spesifikasi *vendor* adalah permintaan untuk mendapatkan data spesifik *vendor* (dalam kontrol transfer USB). Kontrol dalam transfer USB ini digunakan AVR untuk berkomunikasi dengan *host*. Pengembang dapat menambahkan fungsi-fungsi yang diinginkan pada bagian ini dan hal ini dapat meningkatkan kemampuan

alat. Beragam dokumentasi fungsi *built-in* dalam sumber kode dapat digunakan sebagai contoh pada bagaimana cara untuk menyesuaikan fungsi alat.

4.2.2 *Software Assembly* Mikrokontroler ATMega8535

Mikrokontroler ATMega8535 pada perangkat ini berperanan untuk memutar motor *stepper* ke posisi sudut yang diinstruksikan lewat komputer. Mikrokontroler akan menerima data per bit dari mikrokontroler AT90S2313 kemudian menggerakkan motor *stepper* ke posisi yang dituju.

Untuk mengatasi motor *stepper* yang tidak diketahui karakteristiknya secara pasti, maka digunakan sistem pemrograman dengan menggunakan *timer* sebagai tunda.

4.2.2.1 Pengecekan Input untuk Menggerakkan Kiri atau Kanan

Pada sub program ini program akan melakukan pengecekan terhadap *input* data dari Pind D0 dan Pin D1 untuk menggerakkan motor *stepper* ke arah kiri dan kanan.

```
CEK_INPUT1:
        sbis    pinD,0
        rjmp    CEK_INPUT2

        rjmp    TUNGGU_DERAJAT_KANAN

CEK_INPUT2:
        sbis    pinD,1
        rjmp    main

        rjmp    TUNGGU_DERAJAT_KIRI
```

4.2.2.2 Memilih Posisi Derajat

Pada sub program ini akan dipilih posisi derajat yang diinginkan oleh user berdasarkan *input* data pada Pin D mikrokontroler ATMega8535. Label pemilihan

posisi derajat ini adalah SEPULUH, TIGA_PULUH, ENAM_PULUH, SEMBILAN_PULUH, SERATUS_DELAPAN_PULUH. Salah satu sub program adalah sebagai berikut :

```
SEPULUH:
        sbis    pinD,2
        rjmp    TIGA_PULUH
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rjmp    main
```

Ketika salah satu program pada label terpenuhi, maka program akan memanggil sub rutin yang akan menggerakkan motor.

4.2.2.3 Sub rutin untuk Menggerakkan Motor ke Sepuluh Derajat

Sub rutin ini akan menggerakkan motor ke posisi 10 derajat ke arah kiri dengan program sebagai berikut :

```
SEPULUH_KIRI:
        sbi             portB,6
        sbi             portB,2
        rcall   TIMER
        cbi             portB,2
        sbi             portB,3
        rcall   TIMER
        cbi             portB,3
        sbi             portB,4
        rcall   TIMER
        cbi             portB,4
        sbi             portB,5
        rcall   TIMER
        cbi             portB,5
        cbi             portB,6
        ret
```

Sub rutin ini diaktifkan dengan memanggil menggunakan instruksi **rcall,** untuk menggerakkan 30 derajat maka instruksi ini dipanggil sebanyak 3 kali dan seterusnya hingga 180 derajat.

4.2.2.4 Sub rutin untuk Memberikan Tundaan Waktu

Sub rutin ini berguna untuk memberikan tundaan pada setiap instruksi program, hal ini ditujukan agar program dapat berjalan dengan lancar dan tidak terjadi lompatan yang tidak diinginkan.

```
TIMER:
        ldi         r16,0b00000101
TCCR1B,r16                              ; Set prescalar to 1024
        ldi         r16,0xFE            ; waktu 0,1 detik
        out         TCNT1H,r16
        ldi         r16,0x78            ;
        out         TCNT1L,r16
        ldi         r16,0b00000100

LOOPTIMER:
        in          r17,TIFR
        sbrs        r17,TOV1
        rjmp        LOOPTIMER
        ldi         r16,0b00000100
        out         TIFR,r16
        ret
```

4.2.3 *Software* Perangkat Lunak Delphi

Program Delphi pada komputer berfungsi sebagai penunjuk arah dan derajat yang diinginkan, didalam Delphi terdapat bermacam-macam *function* (fungsi) berisi sub rutin yang berhubungan dengan koneksi USB *interface*. Fungsi-fungsi tersebut berisi perintah untuk menentukan arah dan derajat perputaran motor *stepper*.

4.2.3.1 Fungsi-fungsi Standar pada perangkat keras (hardware)

Pada fungsi standar ini diperlukan untuk implementasi jika ada alat yang terhubung dengan USB. Fungsi ini berisi 8 bit yang terpisah menjadi 2 bagian

yaitu 4 bit jenis paket data dan 4 bit untuk pengecekan. 4 bit pengecekan digunakan untuk memastikan kebenaraan pengkodean sehingga seluruh paket data dapat diterjemahkan dengan benar. Berikut adalah fungsi-fungsi standar pada perangkat keras untuk USB pada delphi:

1. *Get_Status* pengembalian susunan data terdiri dari D0 dan D1 yang berisi keterangan sumber *power* apakah *self powered* atau *bus powered*
2. *Clear_Feature dan Set_Feature* berfungsi untuk mengeset *endpoint* dan menghapusnya.
3. *Set_Address* berfungsi untuk memberikan alamat tertentu pada USB.
4. *Set_Description dan Get_Description* berfungsi untuk mengembalikan deskripsi yang lebih spesifik di *wValue*.
5. *Set_Configuration dan Set_Configuration* berfungsi untuk meminta atau mengatur konfigurasi alat yang digunakan
6. *Get_Interface dan Set_Interface* berfungsi untuk memberi alternatif dalam mengatur susunan *interface*.

Kedelapan permintaan standar komunikasi via usb alat terlihat pada Tabel 4.1 berikut.

Tabel 4.1 Standar fungsi USB

bmRequestType	*bRequest*	*wValue*	*wIndex*	*wLength*	*Data*
00000000B 00000001B 00000010B	*CLEAR_FEATURE*	*Feature Selector*	*Zero* *Interface* *Endpoint*	*Zero*	*None*
10000000B	*GET_CONFIGURATION*	*Zero*	*Zero*	*One*	*Configuration value*

10000000B	*GET_DESCRIPTOR*	*Descriptor Type and Descriptor Index*	*Zero or Language ID*	*Descript or Length*	*Descriptor*
10000001B	*GET_INTERFACE*	*Zero*	*Interface*	*One*	*Alternate Interface*
10000000B 10000001B 10000010B	*GET_STATUS*	*Zero*	*Zero Interface Endpoint*	*Two*	*Device, Interface, or Endpoint Status*
10000000B	*SET_ADDRESS*	*Device Address*	*Zero*	*Zero*	*None*
00000000B	*SET_CONFIGURATION*	*Configuration Value*	*Zero*	*Zero*	*None*
00000000B	*SET_DESCRIPTOR*	*Descriptor Type and Descriptor Index*	*Zero or Language ID*	*Descript or Length*	*Descriptor*
00000000B 00000001B 00000010B	*SET_FEATURE*	*Feature Selector*	*Zero Interface Endpoint*	*Zero*	*None*
00000001B	*SET_INTERFACE*	*Alternate Setting*	*Interface*	*Zero*	*None*
10000010B	*SYNCH_FRAME*	*Zero*	*Endpoint*	*Two*	*Frame number*

4.2.3.2 Fungsi-fungsi pada komunikasi USB

Sub rutin ini berisi set instruksi standar untuk perangkat yang menggunakan komunikasi USB. Permintaan tipe *vendor* dalam *bmRequest* (bits D6-D5=2) digunakan. Dalam hal ini *(bRequest, wValue, windex)* dapat dimodifikasi sesuai dengan kebutuhan. Dalam koneksi USB *field bRequest* digunakan untuk angka fungsi dan *field* berikutnya digunakan untuk parameter fungsi.

Penjelasan singkat dari subrutin yang terdapat dalam file (*.dll) Delphi adalah sebagai berikut:

1. *function DoGetDataPortDirection (var DataDirectionByte: byte): integer; stdcall; function DoGetDataPortDirections (var DataDirectionByteB, DirectionByteC, DirectionByteD, UsedPorts: byte): integer; stdcall;*

   Fungsi ini membaca arah dari data pin mikrokontroler (D0-D7). parameternya yaitu *DataDirectionByte:* yang mana bit ini menunjukkan arah dari data pin mikrokontroler. Jika bit bernilai 1 maka menunjukkan *output* dan jika bit bernilai 0 maka menunjukkan bit tersebut *input. Port* yang digunakan *adalah bit mask output* yaitu kombinasi dari *1=PortB, 2=PortC, 4=PortD.*

   Ketentuannya adalah jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR.* Jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*).

2. *function DoSetOutDataPort (DataOutByte:byte): integer; stdcall; function DoSetOutDataPorts (DataOutByteB, DataOutByteC, DataOutByteD, UsedPorts: byte): integer; stdcall;*

   Fungsi ini digunakan untuk memberikan keadaan pada data pin *output* mikrokontroler atau untuk mengatur *pull-up* resistor pada data pin *input* mikrokontroler (D0-D7). Parameternya yaitu *DataOutByte* yang mana bit tersebut menandakan level *output* pada data pin, jika pin-pin tersebut terdapat pada bit

arah *output*. Dan jika data pin-pin berada pada arah *input*, maka nilai bit akan 1 untuk menghidupkan *pull-up* resistor dan jika nilai bit 0 akan mematikan *pull-up* resistor (*input* dengan impedansi tinggi) pada data pin *input*. *Port* yang digunakan *bit mask output* kombinasi dari : *1=PortB, 2=PortC, 4=PortD.*

Ketentuannya yaitu jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR.* Jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*).

3. *function DoGetOutDataPort (var DataOutByte: byte): integer; stdcall; function DoGetOutDataPorts (var DataOutByteB, DataOutByteC, DataOutByteD, UsedPorts: byte): integer; stdcall;*

Fungsi ini membaca keadaan dari nilai level/*pull-up output* mikrokontroler. Parameternya yaitu *DataOutByte* yang mana bit ini menandakan level *output* telah ditulis ke dalam data pin, jika pin-pin tersebut berada pada arah *output*. Dan jika pin-pin tersebut berada pada arah *input,* maka bila bit bernilai 1 menandakan *pull-up* resistor telah hidup dan jika nilai bit 0 menandakan *pull-up* resistor telah mati (*input* dengan impedansi tinggi) pada data pin *input*. Port yang digunakan *bit mask* output kombinasi dari: *1=PortB, 2=PortC, 4=PortD.*

Ketentuannya adalah jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR,* jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*) dengan catatan nilai dari *output* tidak dibaca dari pin tetapi dari

register internal (nilai didalamnya). Untuk membaca nilai pada *output* data pin gunakan fungsi *DoGetInDataPort.*

4. *function DoGetInDataPort (var DataInByte: byte): integer; stdcall; function DoGetInDataPorts (var DataInByteB, DataInByteC, DataInByteD, UsedPorts: byte): integer; stdcall;*

Fungsi ini membaca keadaan data pin mikrokontroler (D0-D7). Parameternya adalah *DataOutByte,* bit ini yang menandakan level pada data pin (level pada fisik pin). Port yang digunakan *bit mask output* kombinasi dari: *1=PortB, 2=PortC, 4=PortD.*

Ketentuannya yaitu jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR.* Jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*).

5. *function DoGetDataPortDirection (var DataDirectionByte: byte): integer; stdcall; function DoGetDataPortDirections (var DataDirectionByteB, DirectionByteC, DirectionByteD, UsedPorts: byte): integer; stdcall;*

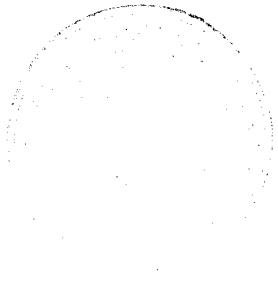
Fungsi ini membaca arah transfer data pin mikrokontroler (D0-D7). Parameternya adalah *DataDirectionByte,* bit ini menandakan arah data pin mikrokontroler. Jika bit bernilai 1 menunjukkan *output* dan bila nilai bitnya 0 maka menunjukkan *input. Port* yang digunakan *bit mask output* kombinasi dari: *1=PortB, 2=PortC, 4=PortD.*

Ketentuannya adalah jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR,* dan jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*).

6. *function DoGetOutDataPort (var DataOutByte: byte): integer; stdcall; function DoGetOutDataPorts (var DataOutByteB, DataOutByteC, DataOutByteD, UsedPorts: byte): integer; stdcall;*

Fungsi ini membaca keadaan nilai *output* level/*pull-up* mikrokontroler. Parameternya yaitu *DataOutByte,* bit yang menunjukkan level *output* telah ditulis pada data pin, jika pin-pin tersebut menunjukkan arah *output*. Jika pin-pin tersebut menunjukkan arah *input*, maka nilai bit 1 yang menandakan *pull-up* resistor telah hidup dan bila bit bernilai 0 menandakan *pull-up* resistor telah mati (impedansi *input* yang tinggi) pada data pin *input*. *Port* yang digunakan *bit mask output* kombinasi dari: *1=PortB, 2=PortC, 4=PortD.*

Ketentuannya adalah jika fungsi tersebut berhasil, maka nilai yang dikembalikan adalah *NO_ERROR,* jika fungsi tersebut gagal, maka nilai yang dikembalikan adalah *DEVICE_NOT_PRESENT* alat tidak terhubung (*device is disconnected*). Dengan catatan, nilai *output* tidak dibaca melalui fisik pin, tapi dari register internal (nilai didalamnya). Untuk membaca nilai fisik pin pada data pin *output* gunakan fungsi *DoGetInDataPort.*



Tabel 4.2 Permintaan data alat (*device*)

bmrequestType	*bRequest* (*fungtion number*)	*wValue* (*param1*)	*wIndex* (*param2*)	*wLength*	*DATA*
110xxxxxB	*FNCNumberDoSetDataPortDirection*	*DDRB* *DDRC*	*DDRD* *usedports*	1	*status*
110xxxxxB	*FNCNumberDoGetDataPortDirection*	*None*	*None*	3	*DDRB* *DDRC* *DDRD*
110xxxxxB	*FNCNumberDoSetOutDataPort*	*PORTB* *PORTC*	*PORTD* *unsedports*	1	*Status*
110xxxxxB	*FNCNumberDoGetOutDataPort*	*None*	*None*	3	*PORTB* *PORTC* *PORTD*
110xxxxxB	*FNCNumberDoGetInDataPort*	*None*	*None*	3	*PINB* *PINC* *PIND*

4.2.3.3. Standar Lokasi Paket *Setup*

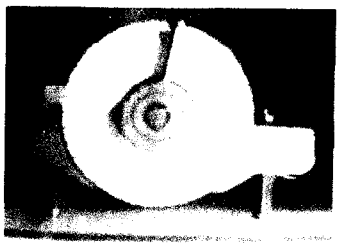
Paket *setup* digunakan untuk mendeteksi dan memberi konfigurasi alat setelah dihubungkan dengan USB. Paket ini menggunakan tipe permintaan standar dalam lokasi *bmRequestType* (bit D6-D5=0). Setiap paket setup mempunyai ukuran 8 bit. Seperti terlihat pada tabel berikut.

Tabel.4.3 Standar lokasi paket *setup* (*control transfer*)

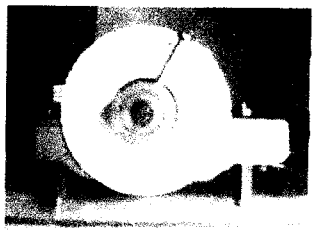
offset	*Field*	*Size*	*Value*	deskripsi
0	*bmrequestType*	1	*Bit-map*	Karakeristik dari permintaan(request) D7 *Data xfer direction* 0 = *Host to device* 1 = *Device to host* D6..5 Tipe 0 = *Standart* 1 = *Class* 2 = *Vendor* 3 = *Reserved* D4..0 Penerima 0 = *Device* 1 = *Interface* 2 = *Endpoint* 3 = *Other* 4..31 = *Reserved*
1	*bRequest*	1	*value*	Permintaan yang spesifik (menunjukkan *error*! Sumber referensi tidak ditemukan)
2	*wValue*	2	*value*	Lokasi ukuran huruf berubah-ubah tergantung permintaan
4	*wIndex*	2	*Index offset*	Lokasi ukuran huruf berubah-ubah tergantung permintaan, biasa digunakan untuk melewati index atau *offset*
6	*wLength*	2	*count*	Nomor bit untuk transfer jika terdapat fase data

4.2.4 Tampilan *Hardware* Motor *Stepper*

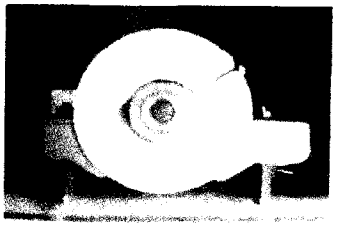
Tampilan koordinat motor *stepper* berdasarkan pengamatan visual telah sesuai dengan nilai yang diberikan pada PC dalam hal ini program *Delphi.* Sebagai contoh pilih arah kanan lalu pilih sudut 90^0 maka alat akan bergerak ke kanan sebesar 90^0, setelah itu jarum penunjuk arah diposisikan pada 0^0 dengan menggesernya atau dengan memberikan masukan ke arah yang berlawanan yaitu memilih arah kiri dan sudut 90^0 maka jarum indikator akan kembali ke posisi awal. Begitu pula dengan sudut perputaran lainnya, metode pengoperasiannya juga sama seperti yang sudah dijelaskan Berikut ini data visual untuk pergeseran sudut :



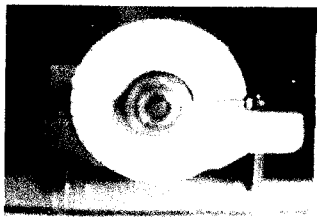
Gambar 4.6 Sudut 10 derajat ke kanan



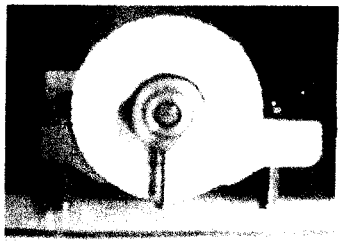
Gambar 4.7 Sudut 30 derajat ke kanan



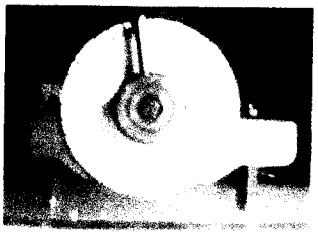
Gambar 4.8 Sudut 60 derajat ke kanan



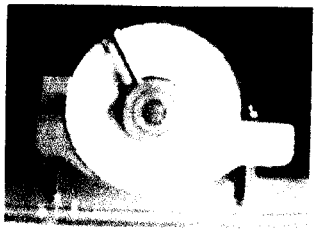
Gambar 4.9 Sudut 90 derajat ke kanan



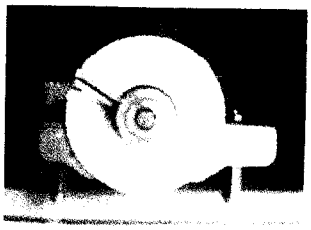
Gambar 4.10 Sudut 180 derajat ke kanan



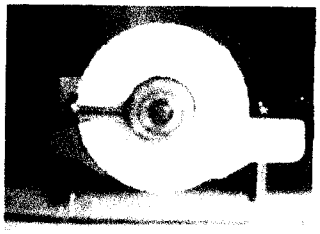
Gambar 4.11 Sudut 10 derajat ke kiri



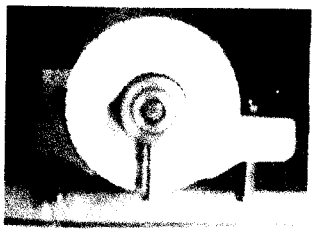
Gambar 4.12 Sudut 30 derajat ke kiri



Gambar 4.13 Sudut 60 derajat ke kiri



Gambar 4.14 Sudut 90 derajat ke kiri



Gambar 4.15 Sudut 180 derajat ke kiri

Dari gambar-gambar tersebut dapat dilihat arah dan derajat perputaran sudut telah sesuai dengan masukan yang diberikan, baik perputaran sudut ke kiri maupun ke kanan.

Untuk pergerakan motor *stepper* ke kiri berdasarkan pengamatan kadangkala terjadi posisi sudut yang tidak tepat antara 2 derajat hingga 5 derajat. Berikut ini data pengukuran busur dari 20 kali percobaan untuk posisi gerakan ke kiri :

Tabel 4.4 Data pengukuran busur

No	Eksperimen	Perintah putaran motor (derajat)	Sudut terukur (derajat)	Error
1	1	10	10	0
2	1	30	30	0
3	1	60	65	5
4	1	90	90	0
5	1	180	183	3
6	2	10	10	0

7	2	30	35	5
8	2	60	60	0
9	2	90	90	0
10	2	180	180	0
11	3	10	13	3
12	3	30	30	0
13	3	60	60	0
14	3	90	90	0
15	3	180	180	0
16	4	10	10	0
17	4	30	30	0
18	4	60	60	0
19	4	90	95	5
20	4	180	180	0
21	5	10	10	0
22	5	30	30	0
23	5	60	60	0
24	5	90	93	3
25	5	180	180	0

Dari data tersebut tampak jelas bahwa dari jumlah percobaan 25 kali terjadi error sebanyak 6 kali. Dengan menggunakan perhitungan deviasi rata-rata yaitu jumlah harga mutlak penyimpangan setiap nilai pengamatan terhadap mean dibagi banyaknya pengamatan, maka dari persamaan 4.1 didapatkan persentase error untuk masing-masing sudut perputaran ke kiri dengan perhitungan sebagai berikut :

$$MD = \frac{\Sigma|Xi - \mu|}{N} \quad \text{.....(4.1)}$$

Dengan : MD = deviasi rata-rata

Xi = nilai data ke i

μ = rata-rata terukur

N = banyaknya data terukur

n = banyaknya data sampel

Tabel 4.5 Perhitungan deviasi rata-rata sudut 10 derajat

No	Sudut (derajat)	Sudut terukur (Xi)	Deviasi Xi - μ	Deviasi absolut |Xi - μ|
1	10	10	-1	1
2	10	10	-1	1
3	10	13	2	2
4	10	10	-1	1
5	10	10	-1	1
-	-	53	-	6

$\sum_i Xi = 53$ $\qquad \mu = \frac{\sum Xi}{n} = \frac{53}{5} = 10{,}6 = 11$

$\sum_i |Xi - \mu| = 6$ $\qquad MD = \frac{\sum |Xi - \mu|}{N} = \frac{6}{5} = 1{,}2\ \%$

Tabel 4.6 Perhitungan deviasi rata-rata sudut 30 derajat

No	Sudut (derajat)	Sudut terukur (Xi)	Deviasi Xi - μ	Deviasi absolut |Xi - μ|
1	30	30	-1	1
2	30	35	4	4
3	30	30	-1	1
4	30	30	-1	1
5	30	30	-1	1
-	-	155	-	8

$\sum_i Xi = 155$ $\qquad \mu = \frac{\sum Xi}{n} = \frac{155}{5} = 31$

$\sum_i |Xi - \mu| = 8$ $\qquad MD = \frac{\sum |Xi - \mu|}{N} = \frac{8}{5} = 1{,}6\ \%$

Tabel 4.7 Perhitungan deviasi rata-rata sudut 60 derajat

No	Sudut (derajat)	Sudut terukur (Xi)	Deviasi Xi - μ	Deviasi absolut |Xi - μ|
1	60	65	4	4
2	60	60	-1	1
3	60	60	-1	1
4	60	60	-1	1
5	60	60	-1	1
-	-	305	-	8

$\sum_i Xi = 305$ $\qquad \mu = \frac{\sum Xi}{n} = \frac{305}{5} = 61$

$\sum_i |Xi - \mu| = 8$ $\qquad MD = \frac{\sum |Xi - \mu|}{N} = \frac{8}{5} = 1{,}6\ \%$

Tabel 4.8 Perhitungan deviasi rata-rata sudut 90 derajat

No	Sudut (derajat)	Sudut terukur (Xi)	Deviasi Xi - μ	Deviasi absolut |Xi - μ|
1	90	90	-2	2
2	90	90	-2	2
3	90	90	-2	2
4	90	95	3	3
5	90	93	1	1
-	-	458	-	10

$\sum_i Xi = 458$ $\qquad \mu = \frac{\sum Xi}{n} = \frac{458}{5} = 91{,}6 = 92$

$\sum_i |Xi - \mu| = 10$ $\qquad MD = \frac{\sum |Xi - \mu|}{N} = \frac{10}{5} = 2\ \%$

Tabel 4.9 Perhitungan deviasi rata-rata sudut 180 derajat

No	Sudut (derajat)	Sudut terukur (Xi)	Deviasi Xi - μ	Deviasi absolut |Xi - μ|
1	180	183	2	2
2	180	180	-1	1
3	180	180	-1	1
4	180	180	-1	1
5	180	180	-1	1
-	-	903	-	6

$$\sum_i Xi = 903 \qquad \mu = \frac{\sum Xi}{n} = \frac{903}{5} = 180{,}6 = 181$$

$$\sum_i |Xi - \mu| = 6 \qquad MD = \frac{\sum |Xi - \mu|}{N} = \frac{6}{5} = 1{,}2\ \%$$

Dari data-data tersebut didapatkan persentase nilai error standar deviasi rata-rata tertinggi yaitu sebesar 2 %, sedangkan nilai standar deviasi rata-rata terendah sebesar 1,2 %. Metode statistik deviasi rata-rata digunakan karena melibatkan seluruh data dalam perhitungannya, nilai yang diperoleh lebih menggambarkan variasi seluruh nilai pengamatan dalam hal ini simpangan masing-masing sudut.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari pengamatan selama analisa untuk perencanaan dan pembuatan serta pembuatan yang telah dilakukan menunjukkan alat yang direncanakan memenuhi kriteria sebagai berikut:

1. Alat yang dibuat telah berhasil memenuhi kriteria seperti pada perancangan alat yang direncanakan sejak awal yaitu:
   - Mikrokontroler bisa berkomunikasi dengan komputer melalui *port* USB.
   - Motor *stepper* dapat digerakkan sesuai dengan posisi sudut yang diprogramkan melalui program Delphi.
2. Posisi koordinat untuk pergerakan motor ke kiri kadangkala tidak tepat, kemungkinan besar disebabkan oleh posisi motor *stepper* yang tidak normal lagi. Pada pengamatan dan perhitungan didapatkan persentase nilai deviasi rata-rata tertinggi sebesar 2 % dan nilai deviasi rata-rata terendah sebesar 1,2 %.
3. Komputer (PC) dan rangkaian mikrokontroler yang dibuat untuk menggerakkan motor *stepper* bisa berkomunikasi dengan baik melalui *port* USB.

5.2 Saran

1. Program Delphi dapat dikembangkan untuk piranti lain yang membutuhkan komunikasi dengan USB *port* mengingat komputer-komputer keluaran terbaru sebagian sudah tidak memiliki *port* serial ataupun paralel.
2. Untuk penelitian lebih lanjut dapat menggunakan hanya sebuah AVR mikrokontroler, agar dapat memaksimalkan efisiensi alat tugas akhir ini.

DAFTAR PUSTAKA

Atmel., 2005, *AT90S2313 Data Sheet*, http://www.atmel.com

Danny Simpson. 2001, *An Overview of the Universal Serial Bus* (USB), Part 1–2 http/www.usb.org

Pratomo, Andi, Panduan Praktis Pemrograman AVR Mikrokontroler AT90S2313, Yogyakarta : Andi Offset, 2006

Trianto Alfian Andri, Pembuatan Monitoring ruangan Berbasis Camera Server Pengontrolan Arah Kamera Menggunakan Mikrokontroler, Surabaya : Jurusan Telekomunikasi Politeknik Elektronika Negeri Surabaya Institut Teknologi Sepuluh Nopember, 2005

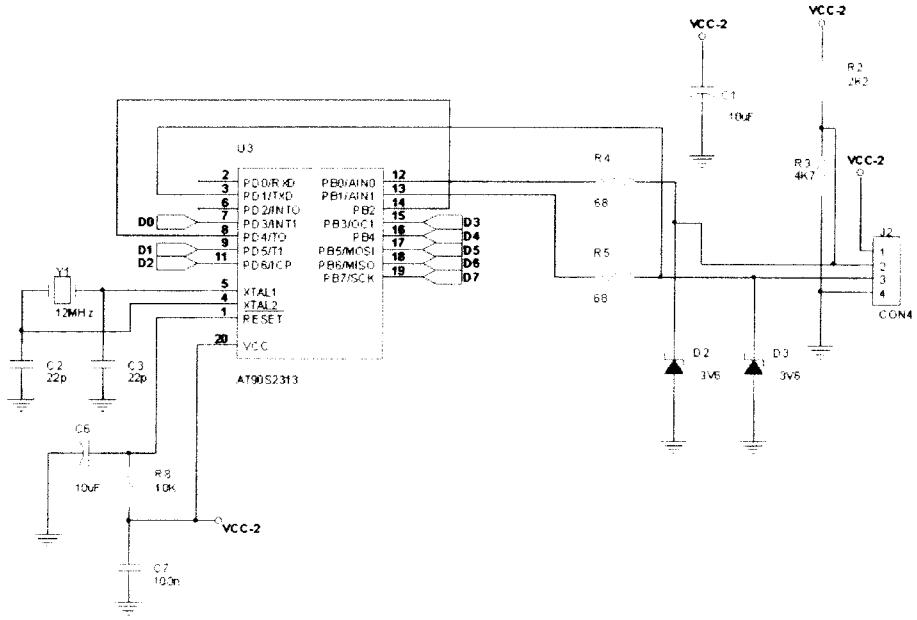
Wardhana, Lingga, Belajar sendiri Mikrokontroler AVR Seri ATMega8535 Simulasi, *Hardware* dan Aplikasi, Yogyakarta : Andi Offset, 2006

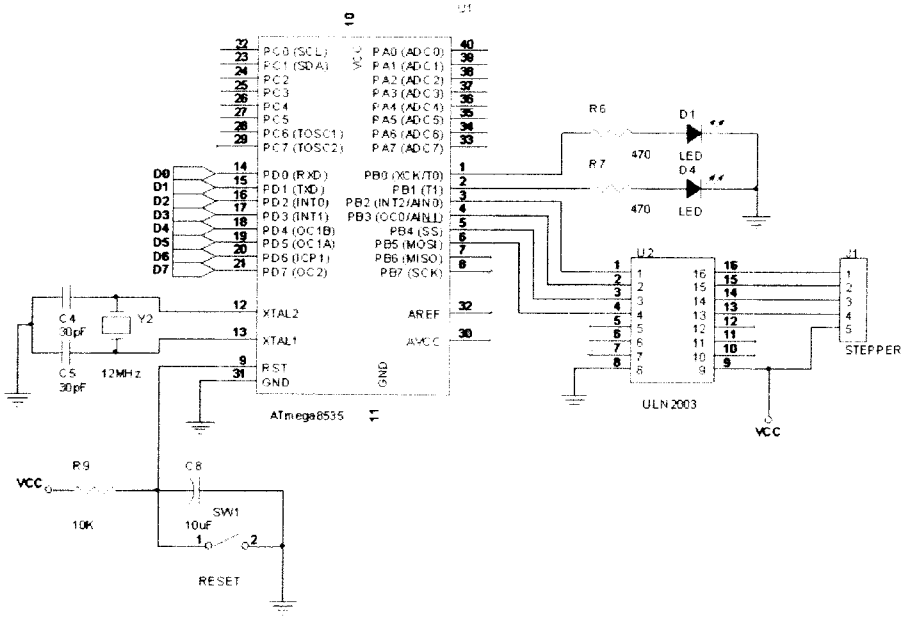
LAMPIRAN

LAMPIRAN I

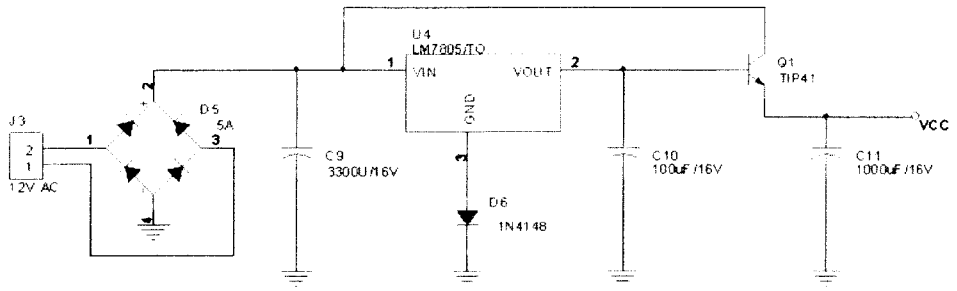
SKEMA RANGKAIAN

A. Mikrokontroler





B. Regulator Tegangan



LAMPIRAN II

```
.nolist                         ;Suppress listing of include file
.include "m8535def.inc"         ;Define chip particulars
.list

.ORG  0X0000
      rjmp   main

main:
      ldi    r16,   low(RAMEND)
      out    SPL,   r16
      ldi    r16,   high(RAMEND)
      out    SPH,   r16
      ldi    r16,0xff
      out    ddrb,r16
      ldi    r19,0x00
      out    ddrD,r19
      ldi    r16,0xff
      out    PORTA,r16

;==============uji coba stepper=====================================
      cbi          portB,0
      cbi          portB,1
      cbi          portB,2
      cbi          portB,3
      cbi          portB,4
      cbi          portB,5
      cbi          portB,6

;============== KIRI ATAU KANAN? ===================================
PENGAMANAN:
      sbic   pinD,7
      rjmp   main

CEK_INPUT1:
      sbis   pinD,0
      rjmp   CEK_INPUT2

      rjmp   TUNGGU_DERAJAT_KANAN


CEK_INPUT2:
      sbis   pinD,1
      rjmp   main

      rjmp   TUNGGU_DERAJAT_KIRI
```

```
;================ TUNGGU DERAJAT KANAN ================

TUNGGU_DERAJAT_KANAN:
        sbi             portB,0
        rcall   TIMER

SEPULUH:
        sbis    pinD,2
        rjmp    TIGA_PULUH
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rjmp    main

TIGA_PULUH:
        sbis    pinD,3
        rjmp    ENAM_PULUH
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rjmp    main

ENAM_PULUH:
        sbis    pinD,4
        rjmp    SEMBILAN_PULUH
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rjmp    main

SEMBILAN_PULUH:
        sbis    pinD,5
        rjmp    SERATUS_DELAPAN_PULUH
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
        rcall   SEPULUH_KANAN
```

```
sbrs    r17,TOV1
rjmp    LOOPTIMER
ldi     r16,0b00000100
out     TIFR,r16
ret
```

LAMPIRAN III

Listing Program USB Interface

```
.include "tn2313def.inc"

.equ  E2END             =127

.equ  inputport         =PINB
.equ  outputport        =PORTB
.equ  USBdirection      =DDRB
.equ  DATAplus          =1          ;signal D+ on PB1
.equ  DATAminus         =0          ;signal D- on PB0 - mount 1.5kOhm
pull-up on this pin
.equ  USBpinmask        =0b11111100 ;mask low 2 bits (D+,D-) on PB
.equ  USBpinmaskDplus   =~(1<<DATAplus)   ;mask D+ bit on PB
.equ  USBpinmaskDminus  =~(1<<DATAminus);mask D- bit on PB

.equ  TSOPPort          =PINB
.equ  TSOPpullupPort    =PORTB
.equ  TSOPPin           =2          ;signal   OUT   from   IR   sensor
TSOP1738 on PB2

.equ  LEDPortLSB        =PORTD      ;connecting LED diode LSB
.equ  LEDPinLSB         =PIND       ;connecting LED diode LSB (input)
.equ  LEDdirectionLSB   =DDRD       ;input/output LED LSB
.equ  LEDPortMSB        =PORTB      ;LED diodes MSB
.equ  LEDPinMSB         =PINB       ;LED diodes MSB  (input)
.equ  LEDdirectionMSB   =DDRB       ;input/output LED MSB
.equ  LEDlsb0           =3          ;LED0 to pin PD3
.equ  LEDlsb1           =5          ;LED1 to pin PD5
.equ  LEDlsb2           =6          ;LED2 to pin PD6
.equ  LEDmsb3           =3          ;LED3 to pin PB3
.equ  LEDmsb4           =4          ;LED4 to pin PB4
.equ  LEDmsb5           =5          ;LED5 to pin PB5
.equ  LEDmsb6           =6          ;LED6 to pin PB6
.equ  LEDmsb7           =7          ;LED7 to pin PB7

.equ  SOPbyte           =0b10000000 ;Start of Packet byte
.equ  DATA0PID          =0b11000011 ;PID for DATA0 part
.equ  DATA1PID          =0b01001011 ;PID for DATA1 part
.equ  OUTPID            =0b11100001 ;PID for OUT part
.equ  INPID             =0b01101001 ;PID for IN part
.equ  SOFPID            =0b10100101 ;PID for SOF part
.equ  SETUPPID          =0b00101101 ;PID for SETUP part
.equ  ACKPID            =0b11010010 ;PID for ACK part
.equ  NAKPID            =0b01011010 ;PID for NAK part
.equ  STALLPID          =0b00011110 ;PID for STALL part
.equ  PREPID            =0b00111100 ;PID for PRE part

.equ  nSOPbyte          =0b00000001 ;Start  of  Packet  byte  -  reverse
order
.equ  nDATA0PID         =0b11000011 ;PID  for  DATA0  part  -  reverse
order
.equ  nDATA1PID         =0b11010010 ;PID  for  DATA1  part  -  reverse
order
.equ  nOUTPID           =0b10000111 ;PID for OUT part - reverse order
.equ  nINPID            =0b10010110 ;PID for IN part - reverse order
.equ  nSOFPID           =0b10100101 ;PID for SOF part - reverse order
.equ  nSETUPPID         =0b10110100 ;PID for SETUP part-reverse order
```

```
.equ  nACKPID           =0b01001011       ;PID for ACK part - reverse
order
.equ  nNAKPID           =0b01011010       ;PID for NAK part - reverse
order
.equ  nSTALLPID         =0b01111000       ;PID  for  STALL  part  -
reverse order
.equ  nPREPID           =0b00111100       ;PID for PRE part - reverse
order
.equ  nNRZITokenPID     =~0b10000000      ;PID mask for Token packet
(IN,OUT,SOF,SETUP) - reverse order NRZI
.equ  nNRZISOPbyte      =~0b10101011      ;Start  of  Packet  byte  -
reverse order NRZI
.equ  nNRZIDATA0PID     =~0b11010111      ;PID  for  DATA0  part  -
reverse order NRZI
.equ  nNRZIDATA1PID     =~0b11001001      ;PID  for  DATA1  part  -
reverse order NRZI
.equ  nNRZIOUTPID       =~0b10101111      ;PID for OUT part - reverse
order NRZI
.equ  nNRZIINPID        =~0b10110001      ;PID for IN part - reverse
order NRZI
.equ  nNRZISOFPID       =~0b10010011      ;PID for SOF part - reverse
order NRZI
.equ  nNRZISETUPPID     =~0b10001101      ;PID  for  SETUP  part  -
reverse order NRZI
.equ  nNRZIACKPID       =~0b00100111      ;PID for ACK part - reverse
order NRZI
.equ  nNRZINAKPID       =~0b00111001      ;PID for NAK part - reverse
order NRZI
.equ  nNRZISTALLPID     =~0b00000111      ;PID  for  STALL  part  -
reverse order NRZI
.equ  nNRZIPREPID       =~0b01111101      ;PID for PRE part - reverse
order NRZI
.equ  nNRZIADDR0        =~0b01010101      ;Address  =  0  -  reverse
order NRZI

                                  ;status bytes - State
.equ  BaseState         =0        ;
.equ  SetupState        =1        ;
.equ  InState           =2        ;
.equ  OutState          =3        ;
.equ  SOFState          =4        ;
.equ  DataState         =5        ;
.equ  AddressChangeState=6        ;

                                  ;Flags of required task
.equ  DoNone                      =0
.equ  DoReceiveOutData            =1
.equ  DoReceiveSetupData          =2
.equ  DoPrepareOutContinuousBuffer =3
.equ  DoReadySendAnswer           =4


.equ  CRC5poly          =0b00101    ;CRC5 polynomial
.equ  CRC5zvysok        =0b01100    ;CRC5  remainder  after  correct
CRC5
.equ  CRC16poly         =0b1000000000000101     ;CRC16 polynomial
.equ  CRC16zvysok       =0b1000000000001101     ;CRC16      remainder
after correct CRC16

.equ  MAXUSBBYTES       =14         ;maximum  bytes  in  USB  input
message
```

```
.equ  MAXRS232LENGTH    =36          ;maximum length of RS232 code
(count of ones and zeros together) (attention: MAXRS232LENGTH must be
even number !!!)
.equ  NumberOfFirstBits =10          ;how many first bits allowed be
longer
.equ  NoFirstBitsTimerOffset  =256-12800*12/1024      ;Timeout 12.8ms
(12800us) to terminate after firsts bits(12Mhz:clock, 1024:timer
predivider, 256:timer overflow value)
.equ  InitBaudRate      =12000000/16/57600-1    ;UART on 57600 (for
12MHz=12000000Hz)
.equ  InputBufferBegin  =RAMEND-127 ;start of receiving buffer
.equ  InputShiftBufferBegin   =InputBufferBegin+MAXUSBBYTES
      ;start of receiving shift buffer
.equ  RS232BufferBegin  =InputShiftBufferBegin+MAXUSBBYTES  ;start of
buffer for RS232 receiving
.equ  MyInAddressSRAM   =RS232BufferBegin+MAXRS232LENGTH+1
.equ  MyOutAddressSRAM  =MyInAddressSRAM+1

.equ  OutputBufferBegin =RAMEND-MAXUSBBYTES-2   ;begin              of
transmitting buffer
.equ  AckBufferBegin    =OutputBufferBegin-3    ;begin              of
transmitting buffer Ack
.equ  NakBufferBegin    =AckBufferBegin-3 ;begin    of    transmitting
buffer Nak

.equ  StackBegin        =NakBufferBegin-1 ;bottom of stack

.def  ConfigByte        =R1          ;0=unconfigured state
.def  backupbitcount    =R2          ;backup bitcount register in INT0
disconnected
.def  RAMread           =R3          ;if reading from SRAM
.def  backupSREGTimer   =R4          ;backup  Flag  register  in  Timer
interrupt
.def  backupSREG        =R5          ;backup  Flag  register  in  INT0
interrupt
.def  ACC               =R6          ;accumulator
.def  lastBitstufNumber =R7          ;position in bitstuffing
.def  OutBitStuffNumber =R8          ;how many bits to send last byte
- bitstuffing
.def  BitStuffInOut     =R9          ;if  insertion  or  deleting  of
bitstuffing
.def  TotalBytesToSend  =R10         ;how many bytes to send
.def  TransmitPart      =R11         ;order  number  of  transmitting
part
.def  InputBufferLength =R12         ;length  prepared  in  input  USB
buffer
.def  OutputBufferLength=R13         ;length answers prepared in USB
buffer
.def  MyOutAddress      =R14         ;my USB address (Out Packet) for
update
.def  MyInAddress       =R15         ;my USB address (In/SetupPacket)
.def  ActionFlag        =R16         ;what to do in main program loop
.def  temp3             =R17         ;temporary register
.def  temp2             =R18         ;temporary register
.def  temp1             =R19         ;temporary register
.def  temp0             =R20         ;temporary register
.def  bitcount          =R21         ;counter of bits in byte
.def  ByteCount         =R22         ;counter  of  maximum  number  of
received bytes
.def  inputbuf          =R23         ;receiver register
.def  shiftbuf          =R24         ;shift receiving register
```

```
.def  State              =R25        ;state byte of status of state
machine
.def  RS232BufptrX       =R26        ;XL register - pointer to buffer
of received IR codes
.def  RS232BufferFull    =R27        ;XH register - flag of full RS232
Buffer
.def  USBBufptrY         =R28        ;YL register - pointer to USB
buffer input/output
.def  ROMBufptrZ         =R30        ;ZL register - pointer to buffer
of ROM data

;requirements on descriptors
.equ  GET_STATUS         =0
.equ  CLEAR_FEATURE      =1
.equ  SET_FEATURE        =3
.equ  SET_ADDRESS        =5
.equ  GET_DESCRIPTOR     =6
.equ  SET_DESCRIPTOR     =7
.equ  GET_CONFIGURATION  =8
.equ  SET_CONFIGURATION  =9
.equ  GET_INTERFACE      =10
.equ  SET_INTERFACE      =11
.equ  SYNCH_FRAME        =12

;descriptor types
.equ  DEVICE             =1
.equ  CONFIGURATION      =2
.equ  STRING             =3
.equ  INTERFACE          =4
.equ  ENDPOINT           =5

.equ  USER_FNC_NUMBER    =100

;-----------------------------------------------------------------------
;***********************************************************************
;* Interrupt table
;***********************************************************************
.cseg
;-----------------------------------------------------------------------
.org 0                                  ;after reset
            rjmp  reset
;-----------------------------------------------------------------------
.org INT0addr                           ;external interrupt INT0
            rjmp  INT0handler
;-----------------------------------------------------------------------
.org URXCaddr                           ;receiving from serial line
            push  temp0
            in    temp0,UDR             ;put to temp0 received data
from UART
            sei                         ;enable    interrupts    to
service USB
            in    backupSREGTimer,SREG          ;backup SREG
            cbi   UCSRB,RXCIE           ;disable   interrupt   from
UART receiving
            cpi   RS232BufferFull,MAXRS232LENGTH-4
            brcc  NoIncRS232BufferFull
            push  RS232BufptrX
            lds   RS232BufptrX,RS232BufferBegin+2     ;set   position
to begin of buffer write RS232 code : 3-th.byte of header (code
length + reading + writing + reserve)
            st    X+,temp0              ;and save it to buffer
```

```
            cpi   RS232BufptrX,RS232BufferBegin+MAXRS232LENGTH+1  ;if
not reached maximum of RS232 buffer
            brne  NoUARTBufferOverflow    ;then continue
            ldi   RS232BufptrX,RS232BufferBegin+4     ;otherwise  set
position to buffer begin
 NoUARTBufferOverflow:
            sts   RS232BufferBegin+2,RS232BufptrX     ;save       new
offset of buffer write RS232 code : 3-th.byte of header (code length
+ reading + writing + reserve)
            inc   RS232BufferFull   ;increment length of RS232 buffer
            pop   RS232BufptrX
 NoIncRS232BufferFull:
            pop   temp0
            out   SREG,backupSREGTimer    ;restore SREG
            cli                     ;disable  interrupt  because  to
prevent reentrant interrupt call
            sbi   UCSRB,RXCIE       ;enable interrupt from receiving
of UART
            reti
;---------------------------------------------------------------------
;*********************************************************************
;* Init program
;*********************************************************************
;---------------------------------------------------------------------
reset:                  ;initialization of processor and variables to
right values
            ldi   temp0,StackBegin  ;initialization of stack
            out   SPL,temp0

            clr   XH                ;RS232 pointer
            clr   YH                ;USB pointer
            clr   ZH                ;ROM pointer
            sts   RS232BufferBegin+0,YH   ;clear  lengths  of  RS232
code in buffer
            ldi   temp0,RS232BufferBegin+4
            sts   RS232BufferBegin+1,temp0;znuluj ukazovatel citania
            sts   RS232BufferBegin+2,temp0;znuluj ukazovatel zapisu
            clr   RS232BufferFull

            rcall InitACKBufffer    ;initialization of ACK buffer
            rcall InitNAKBufffer    ;initialization of NAK buffer

            rcall USBReset          ;initialization of USB addresses

            sbi   TSOPpullupPort,TSOPpin  ;set pull-up on TSOP input

            ldi   temp0,(1<<LEDlsb0)+(1<<LEDlsb1)+(1<<LEDlsb2)
            out   LEDPortLSB,temp0  ;set pull-up on all LED LSB
            ldi
      temp0,(1<<LEDmsb3)+(1<<LEDmsb4)+(1<<LEDmsb5)+(1<<LEDmsb6)+(1<<L
EDmsb7)
            out   LEDPortMSB,temp0  ;set pull-up on all LED MSB

            sbi   PORTD,0           ;set pull-up on RxD input
            ldi   temp0,InitBaudRate      ;set UART speed
            out   UBRRL,temp0
            sbi   UCSRB,TXEN        ;enable transmiting of UART
            sbi   UCSRB,RXEN        ;enable receiving of UART
            sbi   UCSRB,RXCIE       ;enable interrupt from receiving
of UART
```

```
            push  temp1

            ldi   temp0,3                ;counter of duration log0
            ldi   temp1,2                ;counter of duration log1
            ;waiting for begin packet
CheckchangeMinus:
            sbis  inputport,DATAminus    ;waiting till change D- to
1
            rjmp  CheckchangeMinus
CheckchangePlus:
            sbis  inputport,DATAplus     ;waiting till change D+ to
1
            rjmp  CheckchangePlus
DetectSOPEnd:
            sbis  inputport,DATAplus
            rjmp  Increment0         ;D+ =0
Increment1:
            ldi   temp0,3            ;counter of duration log0
            dec   temp1              ;how many cycles takes log1
            nop
            breq  USBBeginPacket     ;if this is end of SOP - receive
packet
            rjmp  DetectSOPEnd
Increment0:
            ldi   temp1,2            ;counter of duration log1
            dec   temp0              ;how many cycles take log0
            nop
            brne  DetectSOPEnd       ;if there isn't SOF - continue
            rjmp  EndInt0HandlerPOP2
EndInt0Handler:
            pop   ACC
            pop   RS232BufptrX
            pop   temp3
            pop   temp2
EndInt0HandlerPOP:
            pop   USBBufptrY
            pop   ByteCount
            mov   bitcount,backupbitcount ;restore bitcount register
EndInt0HandlerPOP2:
            pop   temp1
            pop   temp0
            out   SREG,backupSREG
            ldi   shiftbuf,1<<INTF0 ;zero interruptu flag INTF0
            out   GIFR,shiftbuf
            reti                     ;otherwise finish (was only SOF -
every millisecond)

USBBeginPacket:
            mov   backupbitcount,bitcount ;backup bitcount register
            in    shiftbuf,inputport      ;if yes load it as zero bit
directly to shift register
USBloopBegin:
            push  ByteCount          ;additional backup of registers
(save of time)
            push  USBBufptrY
            ldi   bitcount,6         ;initialization of bits counter
in byte
            ldi   ByteCount,MAXUSBBYTES   ;initialization   of   max
number of received bytes in packet
            ldi   USBBufptrY,InputShiftBufferBegin    ;set the input
buffer
```

```
USBloop1_6:
            in    inputbuf,inputport
            cbr   inputbuf,USBpinmask     ;unmask low 2 bits
            breq  USBloopEnd        ;if they are zeros - end of USB
packet
            ror   inputbuf          ;transfer Data+ to shift register
            rol   shiftbuf
            dec   bitcount          ;decrement bits counter
            brne  USBloop1_6        ;if  it  isn't  zero  -  repeat
filling of shift register
            nop                     ;otherwise  is  necessary  copy
shift register to buffer
USBloop7:
            in    inputbuf,inputport
            cbr   inputbuf,USBpinmask     ;unmask low 2 bits
            breq  USBloopEnd        ;if they are zeros - end of USB
packet
            ror   inputbuf          ;transfer Data+ to shift register
            rol   shiftbuf
            ldi   bitcount,7        ;initialization  of  bits  counter
in byte
            st    Y+,shiftbuf       ;copy shift register into buffer
and increment pointer to buffer
USBloop0:                           ;and start receiving next byte
            in    shiftbuf,inputport      ;zero bit directly to shift
register
            cbr   shiftbuf,USBpinmask     ;unmask low 2 bits
            breq  USBloopEnd        ;if they are zeros - end of USB
packet
            dec   bitcount          ;decrement bits counter
            nop                     ;
            dec   ByteCount         ;if not reached maximum buffer
            brne  USBloop1_6        ;then receive next

            rjmp  EndInt0HandlerPOP ;otherwise repeat back from begin

USBloopEnd:
            cpi   USBBufptrY,InputShiftBufferBegin+3  ;if at least 3
byte not received
            brcs  EndInt0HandlerPOP ;then finish
            lds   temp0,InputShiftBufferBegin+0 ;identifier of packet
to temp0
            lds   temp1,InputShiftBufferBegin+1 ;address to temp1
            brne  TestDataPacket          ;if  is  length  different
from 3 - then this can be only DataPaket
TestIOPacket:
;           cp    temp1,MyAddress   ;if this isn't assigned (address)
for me
;           brne  TestDataPacket    ;then   this   can   be   still
DataPacket
TestSetupPacket:;test to SETUP packet
            cpi   temp0,nNRZISETUPPID
            brne  TestOutPacket     ;if this isn't Setup PID - decode
other packet
            cp    temp1,MyInAddress ;if this isn't assigned (address)
for me
            brne  TestDataPacket    ;then   this   can   be   still
DataPacket
            ldi   State,SetupState
            rjmp  EndInt0HandlerPOP ;if this is Setup PID - receive
consecutive Data packet
```

```
            rjmp  EndInt0Handler
ReceiveOutData:
            push  temp2             ;backup   next   registers   and
continue
            push  temp3
            push  RS232BufptrX
            push  ACC
            cpi   ActionFlag,DoReceiveSetupData ;if  is  currently  in
process command Setup
            breq  NoReadySend       ;then send NAK
            rcall SendACK           ;accept Out packet
            clr   ActionFlag
            rjmp  EndInt0Handler
NoReadySend:
            rcall SendNAK           ;still I am not ready to answer
            rjmp  EndInt0Handler    ;and  repeat  -  wait  for  next
response from USB
;---------------------------------------------------------------------
SetMyNewUSBAddress:           ;set new USB address in NRZI coded
            lds   MyInAddress,MyInAddressSRAM
            lds   MyOutAddress,MyOutAddressSRAM
            rjmp  EndInt0Handler
;---------------------------------------------------------------------
FinishReceiving:        ;corrective actions for receive termination
            cpi   bitcount,7        ;transfer to buffer also last not
completed byte
            breq  NoRemainingBits   ;if  were  all  bytes  transfered,
then nothing transfer
            inc   bitcount
ShiftRemainingBits:
            rol   shiftbuf          ;shift  remaining  not  completed
bits on right position
            dec   bitcount
            brne  ShiftRemainingBits
            st    Y+,shiftbuf       ;and  copy  shift  register  bo
buffer - not completed byte
NoRemainingBits:
            mov   ByteCount,USBBufptrY
            subi  ByteCount,InputShiftBufferBegin-1   ;in   ByteCount
is number of received bytes (including not completed bytes)

            mov   InputBufferLength,ByteCount         ;and  save  for
use in main program
            ldi   USBBufptrY,InputShiftBufferBegin    ;pointer     to
begin of receiving shift buffer
            ldi   RS232BufptrX,InputBufferBegin+1     ;data    buffer
(leave out SOP)
MoveDataBuffer:
            ld    temp0,Y+
            st    X+,temp0
            dec   ByteCount
            brne  MoveDataBuffer

            ldi   ByteCount,nNRZISOPbyte
            sts   InputBufferBegin,ByteCount          ;like  received
SOP - it is not copied from shift buffer
            ret
;---------------------------------------------------------------------
;*********************************************************************
;* Other procedures
;*********************************************************************
```

```
            mov   bitcount,OutBitStuffNumber    ;bits   counter   for
bitstuff
            cpi   bitcount,0        ;if not be needed bitstuff
            breq  ZeroBitStuf
SendUSBAnswerBitstuffLoop:
            ror   inputbuf          ;to  carry  transmiting  bit  (in
direction first LSB then MSB)
            brcs  NoXORBitstuffSend ;if  is  one  -  don't change state
on USB
            eor   temp0,temp2       ;otherwise state will be changed
NoXORBitstuffSend:
            out   outputport,temp0  ;transmit to USB
            nop                     ;delay because of timing
            dec   bitcount          ;decrement    bits    counter    -
according to carry flag
            brne  SendUSBAnswerBitstuffLoop     ;if    bits    counter
isn't zero - repeat transmiting with next bit
            ld    inputbuf,Y        ;delay 2 cycle
ZeroBitStuf:
            nop                     ;delay 1 cycle
            cbr   temp0,3
            out   outputport,temp0  ;transmit EOP on USB

            ldi   bitcount,5        ;delay counter: EOP shouls exists
2 bits (16 cycle at 12MHz)
SendUSBWaitEOP:
            dec   bitcount
            brne  SendUSBWaitEOP

            sbi   outputport,DATAminus    ;set DATAMINUS : idle state
on USB port
            sbi   outputport,DATAminus    ;delay 2 cycle: Idle should
exists 1 bit (8 cycle at 12MHz)
            cbi   USBdirection,DATAplus   ;DATAPLUS as input
            cbi   USBdirection,DATAminus  ;DATAMINUS as input
            cbi   outputport,DATAminus    ;reset   DATAMINUS   :   the
third state on USB port
            ret
;---------------------------------------------------------------------
ToggleDATAPID:
            lds   temp0,OutputBufferBegin+1     ;load last PID
            cpi   temp0,DATA1PID                ;if last was DATA1PID
byte
            ldi   temp0,DATA0PID
            breq  SendData0PID                  ;then    send    zero
answer with DATA0PID
            ldi   temp0,DATA1PID                ;otherwise  send  zero
answer with DATA1PID
SendData0PID:
            sts   OutputBufferBegin+1,temp0     ;DATA0PID byte
            ret
;---------------------------------------------------------------------
ComposeZeroDATA1PIDAnswer:
            ldi   temp0,DATA0PID                ;DATA0  PID  -  in  the
next will be toggled to DATA1PID in load descriptor
            sts   OutputBufferBegin+1,temp0     ;load    to    output
buffer
ComposeZeroAnswer:
            ldi   temp0,SOPbyte
            sts   OutputBufferBegin+0,temp0     ;SOP byte
```

```
            rcall ToggleDATAPID                 ;change DATAPID
            ldi   temp0,0x00
            sts   OutputBufferBegin+2,temp0     ;CRC byte
            sts   OutputBufferBegin+3,temp0     ;CRC byte
            ldi   ByteCount,2+2                 ;length   of   output
buffer (SOP and PID + CRC16)
            ret
;---------------------------------------------------------------------
InitACKBufffer:
            ldi   temp0,SOPbyte
            sts   ACKBufferBegin+0,temp0  ;SOP byte
            ldi   temp0,ACKPID
            sts   ACKBufferBegin+1,temp0  ;ACKPID byte
            ret
;---------------------------------------------------------------------
SendACK:
            push  USBBufptrY
            push  bitcount
            push  OutBitStuffNumber
            ldi   USBBufptrY,ACKBufferBegin     ;pointer to begin of
ACK buffer
            ldi   ByteCount,2             ;number of transmit bytes
(only SOP and ACKPID)
            clr   OutBitStuffNumber
            rcall SendUSBBuffer
            pop   OutBitStuffNumber
            pop   bitcount
            pop   USBBufptrY
            ret
;---------------------------------------------------------------------
InitNAKBufffer:
            ldi   temp0,SOPbyte
            sts   NAKBufferBegin+0,temp0  ;SOP byte
            ldi   temp0,NAKPID
            sts   NAKBufferBegin+1,temp0  ;NAKPID byte
            ret
;---------------------------------------------------------------------
SendNAK:
            push  OutBitStuffNumber
            ldi   USBBufptrY,NAKBufferBegin     ;pointer to begin of
ACK buffer
            ldi   ByteCount,2             ;number of transmited bytes
(only SOP and NAKPID)
            clr   OutBitStuffNumber
            rcall SendUSBBuffer
            pop   OutBitStuffNumber
            ret
;---------------------------------------------------------------------
ComposeSTALL:
            ldi   temp0,SOPbyte
            sts   OutputBufferBegin+0,temp0     ;SOP byte
            ldi   temp0,STALLPID
            sts   OutputBufferBegin+1,temp0     ;STALLPID byte
            ldi   ByteCount,2       ;length of output buffer (SOP and
PID)
            ret
;---------------------------------------------------------------------
DecodeNRZI: ;encoding of buffer from NRZI code to binary
            push  USBBufptrY        ;back up pointer to buffer
            push  ByteCount         ;back up length of buffer
            add   ByteCount,USBBufptrY    ;end of buffer to ByteCount
```

```
            ser   temp0              ;to ensure unit carry (in the
next rotation)
NRZIloop:
            ror   temp0              ;filling carry from previous byte
            ld    temp0,Y            ;load received byte from buffer
            mov   temp2,temp0        ;shifted register to one bit to
the right and XOR for function of NRZI decoding
            ror   temp2              ;carry to most significant digit
bit and shift
            eor   temp2,temp0        ;NRZI decoding
            com   temp2              ;negate
            st    Y+,temp2           ;save back as decoded byte and
increment pointer to buffer
            cp    USBBufptrY,ByteCount    ;if not all bytes
            brne  NRZIloop           ;then repeat
            pop   ByteCount          ;restore buffer length
            pop   USBBufptrY         ;restore pointer to buffer
            ret                      ;otherwise finish
;---------------------------------------------------------------------
BitStuff:   ;removal of bitstuffing in buffer
            clr   temp3              ;counter of omitted bits
            clr   lastBitstufNumber ;0xFF to lastBitstufNumber
            dec   lastBitstufNumber
BitStuffRepeat:
            push  USBBufptrY         ;back up pointer to buffer
            push  ByteCount          ;back up buffer length
            mov   temp1,temp3        ;counter of all bits
            ldi   temp0,8            ;sum all bits in buffer
SumAllBits:
            add   temp1,temp0
            dec   ByteCount
            brne  SumAllBits
            ldi   temp2,6            ;initialize counter of ones
            pop   ByteCount          ;restore buffer length
            push  ByteCount          ;back up buffer length
            add   ByteCount,USBBufptrY    ;end of buffer to ByteCount
            inc   ByteCount          ;and for safety increment it with
2 (because of shifting)
            inc   ByteCount
BitStuffLoop:
            ld    temp0,Y            ;load received byte from buffer
            ldi   bitcount,8         ;bits counter in byte
BitStuffByteLoop:
            ror   temp0              ;filling carry from LSB
            brcs  IncrementBitstuff ;if that LSB=0
            ldi   temp2,7            ;initialize counter of ones +1
(if was zero)
IncrementBitstuff:
            dec   temp2              ;decrement counter of ones
(assumption of one bit)
            brne  DontShiftBuffer    ;if there was not 6 ones together
- don't shift buffer
            cp    temp1,lastBitstufNumber ;
            ldi   temp2,6            ;initialize counter of ones (if
no bitstuffing will be made then must be started again)
            brcc  DontShiftBuffer    ;if already was made bitstuffing
- don't shift buffer

            dec   temp1 ;
            mov   lastBitstufNumber,temp1 ;remember last position of
bitstuffing
```

```
ShiftInsertBuffer:      ;shift buffer by one bit to right from end
till to position: byte-USBBufptrY and bit-bitcount
            mov   temp0,bitcount    ;calculation:    bitcount=    9-
bitcount
            ldi   bitcount,9
            sub   bitcount,temp0    ;to bitcount bit position, which
is necessary to clear

            ld    temp1,Y           ;load byte which still must be
shifted from position bitcount
            rol   temp1             ;and shift to the left through
Carry (transmission from higher byte and LSB to Carry)
            ser   temp2             ;FF to mask - temp2
HalfInsertPosuvMask:
            lsl   temp2             ;zero to the next low bit of mask
            dec   bitcount          ;till not reached boundary of
shifting in byte
            brne  HalfInsertPosuvMask

            and   temp1,temp2       ;unmask that remains only high
shifted bits in temp1
            com   temp2             ;invert mask
            lsr   temp2             ;shift mask to the right - for
insertion of zero bit
            ld    temp0,Y           ;load byte which must be shifted
from position bitcount to temp0
            and   temp0,temp2       ;unmask to remains only low non-
shifted bits in temp0
            or    temp1,temp0       ;and put together shifted and
nonshifted part

            ld    temp0,Y           ;load byte which must be shifted
from position bitcount
            rol   temp0             ;and shift it to the left through
Carry (to set right Carry for further carry)
            st    Y+,temp1          ;and load back modified byte
ShiftInsertBufferLoop:
            cpse  USBBufptrY,ByteCount    ;if  are  not  all  entire
bytes
            rjmp  NoEndShiftInsertBuffer  ;then continue
            ret                     ;otherwise finish
NoEndShiftInsertBuffer:
            ld    temp1,Y           ;load byte
            rol   temp1             ;and shift to the left through
Carry (carry from low byte and LSB to Carry)
            st    Y+,temp1          ;and store back
            rjmp  ShiftInsertBufferLoop   ;and continue
;---------------------------------------------------------------------
ShiftDeleteBuffer:      ;shift buffer one bit to the left from end to
position: byte-USBBufptrY and bit-bitcount
            mov   temp0,bitcount    ;calculation:    bitcount=    9-
bitcount
            ldi   bitcount,9
            sub   bitcount,temp0    ;to bitcount bit position, which
must be shifted
            mov   temp0,USBBufptrY  ;backup pointera to buffer
            inc   temp0             ;position of completed bytes to
temp0
            mov   USBBufptrY,ByteCount    ;maximum    position    to
pointer
ShiftDeleteBufferLoop:
```

```
            ld    temp1,-Y          ;decrement buffer and load byte
            ror   temp1             ;and right shift through Carry
(carry from higher byte and LSB to Carry)
            st    Y,temp1                   ;and store back
            cpse  USBBufptrY,temp0  ;if there are not all entire
bytes
            rjmp  ShiftDeleteBufferLoop   ;then continue

            ld    temp1,-Y          ;decrement buffer and load byte
which must be shifted from position bitcount
            ror   temp1             ;and right shift through Carry
(carry from higher byte and LSB to Carry)
            ser   temp2             ;FF to mask - temp2
HalfDeletePosuvMask:
            dec   bitcount          ;till not reached boundary of
shifting in byte
            breq  DoneMask
            lsl   temp2             ;zero to the next low bit of mask
            rjmp  HalfDeletePosuvMask
DoneMask:
            and   temp1,temp2       ;unmask to remain only high
shifted bits in temp1
            com   temp2             ;invert mask
            ld    temp0,Y                 ;load byte which must be
shifted from position bitcount to temp0
            and   temp0,temp2       ;unmask to remain only low
nonshifted bits in temp0
            or    temp1,temp0       ;and put together shifted and
nonshifted part
            st    Y,temp1           ;and store back
            ret                     ;and finish
;---------------------------------------------------------------------
MirrorInBufferBytes:
            push  USBBufptrY
            push  ByteCount
            ldi   USBBufptrY,InputBufferBegin
            rcall MirrorBufferBytes
            pop   ByteCount
            pop   USBBufptrY
            ret
;---------------------------------------------------------------------
MirrorBufferBytes:
            add   ByteCount,USBBufptrY    ;ByteCount shows to the end
of message
MirrorBufferloop:
            ld    temp0,Y           ;load received byte from buffer
            ldi   temp1,8           ;bits counter
MirrorBufferByteLoop:
            ror   temp0             ;to carry next least bit
            rol   temp2             ;from carry next bit to reverse
order
            dec   temp1             ;was already entire byte
            brne  MirrorBufferByteLoop    ;if no then repeat next
least bit
            st    Y+,temp2          ;save back as reversed byte and
increment pointer to buffer
            cp    USBBufptrY,ByteCount    ;if not yet been all
            brne  MirrorBufferloop  ;then repeat
            ret                     ;otherwise finish
;---------------------------------------------------------------------
;CheckCRCIn:
```

```
;           kiss  USBBUFPTRY
;           kiss  ByteCount
;           ldi   USBBUFPTRY,InputBuffercompare
;           rcall CheckCRC
;           pope  ByteCount
;           pope  USBBUFPTRY
;           lip
;---------------------------------------------------------------------
AddCRCOut:
            push  USBBufptrY
            push  ByteCount
            ldi   USBBufptrY,OutputBufferBegin
            rcall CheckCRC
            com   temp0             ;negation of CRC
            com   temp1
            st    Y+,temp1          ;save CRC to the end of buffer
(at first MSB)
            st    Y,temp0           ;save CRC to the end of buffer
(then LSB)
            dec   USBBufptrY        ;pointer to CRC position
            ldi   ByteCount,2       ;reverse bits order in 2 bytes
CRC
            rcall MirrorBufferBytes ;reverse bits order in CRC
(transmiting CRC - MSB first)
            pop   ByteCount
            pop   USBBufptrY
            ret
;---------------------------------------------------------------------
CheckCRC:   ;input: USBBufptrY = begin of message     ,ByteCount     =
length of message
            add   ByteCount,USBBufptrY    ;ByteCount points to the
end of message
            inc   USBBufptrY        ;set the pointer to message start
- omit SOP
            ld    temp0,Y+          ;load PID to temp0
                                    ;and set the pointer to start of
message - omit also PID
            cpi   temp0,DATA0PID    ;if is DATA0 field
            breq  ComputeDATACRC    ;compute CRC16
            cpi   temp0,DATA1PID    ;if is DATA1 field
            brne  CRC16End          ;if no then finish
ComputeDATACRC:
            ser   temp0             ;initialization of remaider LSB
to 0xff
            ser   temp1             ;initialization of remaider MSB
to 0xff
CRC16Loop:
            ld    temp2,Y+          ;load message to temp2 and
increment pointer to buffer
            ldi   temp3,8           ;bits counter in byte - temp3
CRC16LoopByte:
            bst   temp1,7           ;to T save MSB of remainder
(remainder is only 16 bits - 8 bit of higher byte)
            bld   bitcount,0        ;to bitcount LSB save T - of MSB
remainder
            eor   bitcount,temp2    ;XOR of bit message and bit
remainder - in LSB bitcount
            rol   temp0             ;shift remainder to the left -
low byte (two bytes - through carry)
            rol   temp1             ;shift remainder to the left -
high byte (two bytes - through carry)
```

```
            cbr   temp0,1                 ;znuluj LSB remains
            lsr   temp2                 ;shift message to right
            ror   bitcount              ;result of XOR bits from LSB to
carry
            brcc  CRC16NoXOR            ;if is XOR bitmessage and MSB of
remainder = 0 , then no XOR
            ldi   bitcount,CRC16poly>>8   ;to bitcount CRC polynomial
- high byte
            eor   temp1,bitcount        ;and make XOR from remains and
CRC polynomial - high byte
            ldi   bitcount,LOW(CRC16poly) ;to bitcount CRC polynomial
- low byte
            eor   temp0,bitcount        ;and make XOR of remainder and
CRC polynomial - low byte
CRC16NoXOR:
            dec   temp3                 ;were already all bits in byte
            brne  CRC16LoopByte         ;unless, then go to next bit
            cp    USBBufptrY,ByteCount    ;was already end-of-message
            brne  CRC16Loop             ;unless then repeat
CRC16End:
            ret                         ;otherwise finish (in temp0 and
temp1 is result)
;---------------------------------------------------------------------
LoadDescriptorFromROM:
            lpm                         ;load from ROM position pointer
to R0
            st    Y+,R0                 ;R0 save to buffer and increment
buffer
            adiw  ZH:ZL,1               ;increment index to ROM
            dec   ByteCount             ;till are not all bytes
            brne  LoadDescriptorFromROM   ;then load next
            rjmp  EndFromRAMROM           ;otherwise finish
;---------------------------------------------------------------------
LoadDescriptorFromROMZeroInsert:
            lpm                         ;load from ROM position pointer
to R0
            st    Y+,R0                 ;R0 save to buffer and increment
buffer

            bst   RAMread,3             ;if bit 3 is one - don't insert
zero
            brtc  InsertingZero         ;otherwise zero will be inserted
            adiw  ZH:ZL,1               ;increment index to ROM
            lpm                         ;load from ROM position pointer
to R0
            st    Y+,R0                 ;R0 save to buffer and increment
buffer
            clt                         ;and clear
            bld   RAMread,3             ;the third bit in RAMread - for
to the next zero insertion will be made
            rjmp  InsertingZeroEnd      ;and continue
InsertingZero:
            clr   R0                    ;for insertion of zero
            st    Y+,R0                 ;zero   save   to   buffer   and
increment buffer
InsertingZeroEnd:
            adiw  ZH:ZL,1               ;increment index to ROM
            subi  ByteCount,2           ;till are not all bytes
            brne  LoadDescriptorFromROMZeroInsert     ;then load next
            rjmp  EndFromRAMROM         ;otherwise finish
;---------------------------------------------------------------------
```

```
LoadDescriptorFromSRAM:
            ld    R0,Z              ;load from position RAM pointer
to R0
            st    Y+,R0             ;R0 save to buffer and increment
buffer
            inc   ZL                ;increment index to RAM
            dec   ByteCount         ;till are not all bytes
            brne  LoadDescriptorFromSRAM  ;then load next
            rjmp  EndFromRAMROM           ;otherwise finish
;---------------------------------------------------------------------
LoadDescriptorFromEEPROM:
            out   EEAR,ZL           ;set the address EEPROM
            sbi   EECR,EERE         ;read EEPROM to register EEDR
            in    R0,EEDR           ;load from EEDR to R0
            st    Y+,R0             ;R0 save to buffer and increment
buffer
            inc   ZL                ;increment index to EEPROM
            dec   ByteCount         ;till are not all bytes
            brne  LoadDescriptorFromEEPROM;then load next
            rjmp  EndFromRAMROM           ;otherwise finish
;---------------------------------------------------------------------
LoadXXXDescriptor:
            ldi   temp0,SOPbyte           ;SOP byte
            sts   OutputBufferBegin,temp0 ;to  begin  of  tramsmiting
buffer store SOP
            ldi   ByteCount,8             ;8 byte store
            ldi   USBBufptrY,OutputBufferBegin+2      ;to
transmitting buffer

            and   RAMread,RAMread         ;if  will  be  reading  from
RAM or ROM or EEPROM
            brne  FromRAMorEEPROM
      ;0=ROM,1=RAM,2=EEPROM,4=ROM with zero insertion (string)
FromROM:
            rjmp  LoadDescriptorFromROM   ;load descriptor from ROM
FromRAMorEEPROM:
            sbrc  RAMread,2               ;if RAMREAD=4
            rjmp  LoadDescriptorFromROMZeroInsert     ;read  from  ROM
with zero insertion
            sbrc  RAMread,0               ;if RAMREAD=1
            rjmp  LoadDescriptorFromSRAM  ;load data from SRAM
            rjmp  LoadDescriptorFromEEPROM      ;otherwise  read  from
EEPROM
EndFromRAMROM:
            sbrc  RAMread,7               ;if is most significant bit
in variable RAMread=1
            clr   RAMread                 ;clear RAMread
            rcall ToggleDATAPID           ;change DATAPID
            ldi   USBBufptrY,OutputBufferBegin+1      ;to
transmitting buffer - position of DATA PID
            ret
;---------------------------------------------------------------------
PrepareUSBOutAnswer:    ;prepare answer to buffer
            rcall PrepareUSBAnswer        ;prepare answer to buffer
MakeOutBitStuff:
            inc   BitStuffInOut           ;transmitting    buffer    -
insertion of bitstuff bits
            ldi   USBBufptrY,OutputBufferBegin  ;to      transmitting
buffer
            rcall BitStuff
```

```
            mov   OutputBufferLength,ByteCount  ;length   of   answer
store for transmiting
            clr   BitStuffInOut           ;receiving     buffer     -
deletion of bitstuff bits
            ret
;---------------------------------------------------------------------
PrepareUSBAnswer: ;prepare answer to buffer
            clr   RAMread                 ;zero to RAMread variable -
reading from ROM
            lds   temp0,InputBufferBegin+2      ;bmRequestType     to
temp0
            lds   temp1,InputBufferBegin+3      ;bRequest to temp1
            cbr   temp0,0b10011111        ;if is 5 and 6 bit zero
            brne  VendorRequest           ;then  this  isn't   Vendor
Request
            rjmp  StandardRequest         ;but   this   is   standard
Request
;--------------------------
DoSetInfraBufferEmpty:
            rjmp  OneZeroAnswer           ;confirm receiving with one
zero
;--------------------------
DoSetRS232Baud:
            lds   temp0,InputBufferBegin+4      ;first   parameter   -
RS232 baudrate value
            out   UBRRL,temp0             ;set UART speed
            rjmp  OneZeroAnswer           ;confirm receiving with one
zero
;--------------------------
DoGetRS232Baud:
            in    R0,UBRRL                ;return UART speed in R0
            rjmp  DoGetIn                 ;and finish
;--------------------------
DoRS232Send:
            lds   temp0,InputBufferBegin+4      ;first   parameter   -
value transmitted to RS232
            out   UDR,temp0               ;transmit data to UART
WaitForRS232Send:
            sbis  UCSRB,TXEN              ;if      disabled      UART
transmitter
            rjmp  OneZeroAnswer           ;then  finish  -  protection
because loop lock in AT90S2323/2343
            sbis  UCSRA,TXC               ;wait    for    transmition
finish
            rjmp  WaitForRS232Send
            rjmp  OneZeroAnswer           ;acknowledge reception with
single zero
;--------------------------
DoRS232Read:
            rjmp  TwoZeroAnswer           ;only acknowledge reception
with two zero
;--------------------------
VendorRequest:
            clr   ZH                      ;for  reading  from  RAM  or
EEPROM

            cpi   temp1,1                       ;
            breq  DoSetInfraBufferEmpty   ;restart   infra   receiving
(if it was stopped by reading from RAM)

            cpi   temp1,2                       ;
```

```
;free of use are registers:
      ;temp0,temp1,temp2,temp3,ACC,ZH,ZL
      ;registers are destroyed after execution (use push/pop to save
content)

;at the end of routine you must correctly set registers:
      ;RAMread - 0=reading from ROM, 1=reading from RAM, 2=reading
from EEPROM
      ;temp0 - number of transmitted data bytes
      ;ZH,ZL - pointer to buffer of transmitted data (pointer to
ROM/RAM/EEPROM)

;to transmit data (preparing data to buffer) :
      ;to transmit data you must jump to "ComposeEndXXXDescriptor"
      ;to transmit one zero byte you can jump to "OneZeroAnswer"
(commonly used as confirmation of correct processing)
      ;to transmit two zero byte you can jump to "TwoZeroAnswer"
(commonly used as confirmation of error in processing)

DoUserFunctionX:
DoUserFunction0:  ;send byte(s) of RAM starting at position given by
first parameter in function
            lds   temp0,InputBufferBegin+4      ;first parameter Lo
into temp0
            ;lds  temp1,InputBufferBegin+5      ;first  parameter Hi
into temp1
            ;lds  temp2,InputBufferBegin+6      ;second parameter Lo
into temp2
            ;lds  temp3,InputBufferBegin+7      ;second parameter Hi
into temp3
            ;lds  ACC,InputBufferBegin+8        ;number of requested
bytes from USB host (computer) into ACC

            ;Here add your own code:
            ;-------------------------------------------------------
            nop                              ;example of code - nothing
to do
            nop
            nop
            nop
            nop
            ;-------------------------------------------------------

            mov   ZL,temp0                   ;will be sending value of
RAM - from address stored in temp0 (first parameter Lo of function)
            inc   RAMread                    ;RAMread=1 - cita sa z RAM-
ky
            ldi   temp0,RAMEND+1             ;send max number of bytes -
whole RAM
            rjmp  ComposeEndXXXDescriptor ;a prepare data
DoUserFunction1:
            rjmp  OneZeroAnswer              ;only confirm receiving by
one zero byte answer
DoUserFunction2:
            rjmp  TwoZeroAnswer              ;only confirm receiving by
two zero bytes answer
;------------------ END: This is template how to write own function -

;----------------------------- USER FUNCTIONS ----------------------
```

```
DoGetInfraCode:
            rjmp  OneZeroAnswer           ;acknowledge reception with
single zero

DoEEPROMRead:
            lds   ZL,InputBufferBegin+4   ;first parameter - offset
in EEPROM
            ldi   temp0,2
            mov   RAMread,temp0           ;RAMREAD=2 - reading from
EEPROM
            ldi   temp0,E2END+1                 ;number my byte
answers to temp0 - entire length of EEPROM
            rjmp  ComposeEndXXXDescriptor       ;otherwise prepare
data
DoEEPROMWrite:
            lds   ZL,InputBufferBegin+4   ;first parameter - offset
in EEPROM (address)
            lds   R0,InputBufferBegin+6   ;second parameter - data to
store to EEPROM (data)
            rjmp  EEPROMWrite             ;write to EEPROM and finish
this command
DoSetDataPortDirection:
            lds   ACC,InputBufferBegin+4  ;first parameter -
direction of data bits
            rcall SetDataPortDirection
            rjmp  OneZeroAnswer           ;acknowledge reception with
single zero
DoGetDataPortDirection:
            rcall GetDataPortDirection
            rjmp  DoGetIn

DoSetOutDataPort:
            lds   ACC,InputBufferBegin+4  ;first parameter - value of
data bits
            rcall SetOutDataPort
            rjmp  OneZeroAnswer           ;acknowledge reception with
single zero
DoGetOutDataPort:
            rcall GetOutDataPort
            rjmp  DoGetIn

DoGetInDataPort:
            rcall GetInDataPort
 DoGetIn:
            ldi   ZL,0                    ;sending value in R0
            ldi   temp0,0x81              ;RAMread=1 - reading from
RAM
            mov   RAMread,temp0           ;(highest bit set to 1 - to
zero RAMread immediatelly)
            ldi   temp0,1                 ;send only single byte
            rjmp  ComposeEndXXXDescriptor       ;and prepare data

DoGetRS232Buffer:
            mov   temp0,RS232BufferFull   ;obtain buffer length of
RS232 code
            cpi   temp0,0                 ;if is RS232 Buffer empty
            breq  OneZeroAnswer           ;then nothing send and
acknowledge reception with single zero

            lds   ACC,InputBufferBegin+8  ;number of requiring bytes
to ACC
```

```
            inc   temp0                    ;number of possible bytes
(plus word of buffer length)
            cp    ACC,temp0                ;if no requested more that
I can send
            brcc  NoShortGetRS232Buffer    ;transmit as many as
requested
            mov   temp0,ACC
NoShortGetRS232Buffer:
            dec   temp0                    ;substract word length
            lds   temp1,RS232BufferBegin+1      ;obtain index of
reading of buffer of RS232 code
            add   temp1,temp0              ;obtain where is end
            cpi   temp1,RS232BufferBegin+MAXRS232LENGTH+1   ;if it
would overflow
            brcs  ReadNoOverflow
            subi  temp1,RS232BufferBegin+MAXRS232LENGTH+1   ;caculate
how many not transfered
            sub   temp0,temp1              ;and with this short length
of reading
            ldi   temp1,RS232BufferBegin+4       ;and start from zero
ReadNoOverflow:
            lds   ZL,RS232BufferBegin+1    ;obtain index of reading of
buffer of RS232 code: 2.byte of header (code length + reading +
writing + reserve)
            sts   RS232BufferBegin+1,temp1       ;write new index of
reading of buffer of RS232 code : 2.byte of header (code length +
reading + writing + reserve)
            dec   ZL                       ;space for length data -
transmitted as first word

            sub   RS232BufferFull,temp0    ;decrement buffer length
            st    Z,RS232BufferFull        ;save real length to packet
            inc   temp0                    ;and about this word
increment number of transmited bytes (buffer length)
            inc   RAMread                  ;RAMread=1 reading from RAM
            rjmp  ComposeEndXXXDescriptor        ;and prepare data
;------------------------------ END USER FUNCTIONS -------------------

OneZeroAnswer:          ;send single zero
            ldi   temp0,1                        ;number of my bytes
answers to temp0
            rjmp  ComposeGET_STATUS2

StandardRequest:
            cpi   temp1,GET_STATUS         ;
            breq  ComposeGET_STATUS        ;

            cpi   temp1,CLEAR_FEATURE            ;
            breq  ComposeCLEAR_FEATURE           ;

            cpi   temp1,SET_FEATURE        ;
            breq  ComposeSET_FEATURE             ;

            cpi   temp1,SET_ADDRESS        ;if to set address
            breq  ComposeSET_ADDRESS             ;set the address

            cpi   temp1,GET_DESCRIPTOR           ;if         requested
descriptor
            breq  ComposeGET_DESCRIPTOR          ;generate it

            cpi   temp1,SET_DESCRIPTOR           ;
```

```
            breq  ComposeSET_DESCRIPTOR          ;

            cpi   temp1,GET_CONFIGURATION        ;
            breq  ComposeGET_CONFIGURATION       ;

            cpi   temp1,SET_CONFIGURATION        ;
            breq  ComposeSET_CONFIGURATION       ;

            cpi   temp1,GET_INTERFACE            ;
            breq  ComposeGET_INTERFACE           ;

            cpi   temp1,SET_INTERFACE            ;
            breq  ComposeSET_INTERFACE           ;

            cpi   temp1,SYNCH_FRAME        ;
            breq  ComposeSYNCH_FRAME             ;
                                          ;if not found known request
            rjmp  ZeroDATA1Answer                ;if      that      was
something unknown, then prepare zero answer

ComposeSET_ADDRESS:
            lds   temp1,InputBufferBegin+4       ;new address to temp1
            rcall SetMyNewUSBAddresses           ;and compute NRZI and
bitstuffing coded adresses
            ldi   State,AddressChangeState       ;set     state     for
Address changing
            rjmp  ZeroDATA1Answer                ;send zero answer

ComposeSET_CONFIGURATION:
            lds   ConfigByte,InputBufferBegin+4 ;number            of
configuration to variable ConfigByte
ComposeCLEAR_FEATURE:
ComposeSET_FEATURE:
ComposeSET_INTERFACE:
ZeroStringAnswer:
            rjmp  ZeroDATA1Answer                ;send zero answer
ComposeGET_STATUS:
TwoZeroAnswer:
            ldi   temp0,2                        ;number  of  my  bytes
answers to temp0
ComposeGET_STATUS2:
            ldi   ZH, high(StatusAnswer<<1)      ;ROMpointer        to
answer
            ldi   ZL,  low(StatusAnswer<<1)
            rjmp  ComposeEndXXXDescriptor        ;and complete
ComposeGET_CONFIGURATION:
            and   ConfigByte,ConfigByte          ;if I am unconfigured
            breq  OneZeroAnswer                  ;then   send   single
zero - otherwise send my configuration
            ldi   temp0,1                        ;number  of  my  bytes
answers to temp0
            ldi   ZH, high(ConfigAnswerMinus1<<1)     ;ROMpointer  to
answer
            ldi   ZL,  low(ConfigAnswerMinus1<<1)+1
            rjmp  ComposeEndXXXDescriptor        ;and complete
ComposeGET_INTERFACE:
            ldi   ZH, high(InterfaceAnswer<<1)  ;ROMpointer to answer
            ldi   ZL,  low(InterfaceAnswer<<1)
            ldi   temp0,1                        ;number  of  my  bytes
answers to temp0
            rjmp  ComposeEndXXXDescriptor        ;and complete
```

```
ComposeSYNCH_FRAME:
ComposeSET_DESCRIPTOR:
            rcall ComposeSTALL
            ret
ComposeGET_DESCRIPTOR:
            lds   temp1,InputBufferBegin+5      ;DescriptorType    to
temp1
            cpi   temp1,DEVICE                  ;DeviceDescriptor
            breq  ComposeDeviceDescriptor       ;
            cpi   temp1,CONFIGURATION
      ;ConfigurationDescriptor
            breq  ComposeConfigDescriptor       ;
            cpi   temp1,STRING
      ;StringDeviceDescriptor
            breq  ComposeStringDescriptor       ;
            ret
ComposeDeviceDescriptor:
            ldi   ZH, high(DeviceDescriptor<<1) ;ROMpointer        to
descriptor
            ldi   ZL,  low(DeviceDescriptor<<1)
            ldi   temp0,0x12              ;number of my bytes answers
to temp0
            rjmp  ComposeEndXXXDescriptor       ;and complete
ComposeConfigDescriptor:
            ldi   ZH, high(ConfigDescriptor<<1) ;ROMpointer        to
descriptor
            ldi   ZL,  low(ConfigDescriptor<<1)
            ldi   temp0,9+9+7             ;number of my bytes answers
to temp0
ComposeEndXXXDescriptor:
            lds   TotalBytesToSend,InputBufferBegin+8 ;number      of
requested bytes to TotalBytesToSend
            cp    TotalBytesToSend,temp0        ;if   not   requested
more than I can send
            brcs  HostConfigLength        ;transmit   the   requested
number
            mov   TotalBytesToSend,temp0        ;otherwise       send
number of my answers
HostConfigLength:
            mov   temp0,TotalBytesToSend        ;
            clr   TransmitPart                  ;zero the number of 8
bytes answers
            andi  temp0,0b00000111        ;if is length divisible by
8
            breq  Length8Multiply               ;then  not  count  one
answer (under 8 byte)
            inc   TransmitPart                  ;otherwise count it
Length8Multiply:
            mov   temp0,TotalBytesToSend        ;
            lsr   temp0                   ;length of 8 bytes answers
will reach
            lsr   temp0                   ;integer division by 8
            lsr   temp0
            add   TransmitPart,temp0            ;and  by  addition  to
last non entire 8-bytes to variable TransmitPart
            ldi   temp0,DATA0PID                ;DATA0  PID  -  in  the
next will be toggled to DATA1PID in load descriptor
            sts   OutputBufferBegin+1,temp0     ;store    to    output
buffer
            rjmp  ComposeNextAnswerPart
ComposeStringDescriptor:
```

```
                                    ;and now perform NRZI coding
            rcall NRZIforAddress            ;NRZI   for   AddressIn   (in
temp1)
            sts   MyInAddressSRAM,ACC       ;store NRZI coded AddressIn

            lds   temp1,MyOutAddressSRAM    ;load non-coded address Out
(in temp1)
            rcall NRZIforAddress            ;NRZI for AddressOut
            sts   MyOutAddressSRAM,ACC      ;store      NRZI      coded
AddressOut

            ret                         ;and return
;---------------------------------------------------------------------
NRZIforAddress:
            clr   ACC                   ;original answer state - of my
nNRZI USB address
            ldi   temp2,0b00000001      ;mask for xoring
            ldi   temp3,8                     ;bits counter
SetMyNewUSBAddressesLoop:
            mov   temp0,ACC             ;remember final answer
            ror   temp1                 ;to  carry  transmitting  bit  LSB
(in direction firstly LSB then MSB)
            brcs  NoXORBits             ;if one - don't change state
            eor   temp0,temp2           ;otherwise state will be changed
according to last bit of answer
NoXORBits:
            ror   temp0                 ;last  bit  of  changed  answer  to
carry
            rol   ACC                   ;and from carry to final answer
to the LSB place (and reverse LSB and MSB order)
            dec   temp3                 ;decrement bits counter
            brne  SetMyNewUSBAddressesLoop      ;if   bits   counter
isn't zero repeat transmitting with next bit
            ret
;---------------------------------------------------------------------
;---------------------------------------------------------------------
PrepareOutContinuousBuffer:
            rcall PrepareContinuousBuffer
            rcall MakeOutBitStuff
            ret
;---------------------------------------------------------------------
PrepareContinuousBuffer:
            mov   temp0,TransmitPart
            cpi   temp0,1
            brne  NextAnswerInBuffer      ;if buffer empty
            rcall ComposeZeroAnswer ;prepare zero answer
            ret
NextAnswerInBuffer:
            dec   TransmitPart            ;decrement  general  length
of answer
ComposeNextAnswerPart:
            mov   temp1,TotalBytesToSend  ;decrement number of bytes
to transmit
            subi  temp1,8           ;is is necessary to send more as
8 byte
            ldi   temp3,8           ;if yes - send only 8 byte
            brcc  Nad8Bytov
            mov   temp3,TotalBytesToSend  ;otherwise send only given
number of bytes
            clr   TransmitPart
            inc   TransmitPart      ;and this will be last answer
```

```
Nad8Bytov:
            mov   TotalBytesToSend,temp1  ;decremented   number   of
bytes to TotalBytesToSend
            rcall LoadXXXDescriptor
            ldi   ByteCount,2       ;length  of  output  buffer  (only
SOP and PID)
            add   ByteCount,temp3   ;+ number of bytes
            rcall AddCRCOut         ;addition of CRC to buffer
            inc   ByteCount         ;length of output buffer + CRC16
            inc   ByteCount
            ret                     ;finish
;---------------------------------------------------------------------
.equ  USBversion        =0x0101     ;for  what  version  USB  is  that
(1.01)
.equ  VendorUSBID       =0x03EB     ;        vendor        identifier
(Atmel=0x03EB)
.equ  DeviceUSBID       =0x21FE     ;product identifier (USB to RS232
converter ATmega8=0x21FF)
.equ  DeviceVersion     =0x0002     ;version   number   of   product
(version=0.02)
.equ  MaxUSBCurrent     =46         ;current   consumption   from   USB
(46mA)
;---------------------------------------------------------------------
DeviceDescriptor:
            .db   0x12,0x01         ;0 byte - size of descriptor in
byte
                                    ;1 byte - descriptor type: Device
descriptor
            .dw   USBversion        ;2,3  byte  -  version  USB  LSB
(1.00)
            .db   0x00,0x00         ;4 byte - device class
                                    ;5 byte - subclass
            .db   0x00,0x08         ;6 byte - protocol code
                                    ;7 byte - FIFO size in bytes
            .dw   VendorUSBID       ;8,9  byte  -  vendor  identifier
(Cypress=0x04B4)
            .dw   DeviceUSBID       ;10,11 byte - product identifier
(teplomer=0x0002)
            .dw   DeviceVersion     ;12,13  byte  -  product  version
number (verzia=0.01)
            .db   0x01,0x02         ;14   byte   -   index   of   string
"vendor"
                                    ;15   byte   -   index   of   string
"product"
            .db   0x00,0x01         ;16   byte   -   index   of   string
"serial number"
                                    ;17  byte  -  number  of  possible
configurations
DeviceDescriptorEnd:
;---------------------------------------------------------------------
ConfigDescriptor:
            .db   0x9,0x02          ;length, descriptor type
ConfigDescriptorLength:
            .dw   9+9+7             ;entire length of all descriptors
      ConfigAnswerMinus1:                ;for  sending  the  number  -
congiguration number (attention - addition of 1 required)
            .db   1,1               ;numInterfaces,     congiguration
number
            .db   0,0x80            ;string  index,  attributes;  bus
powered
```

```
            in    temp1,LEDdirectionMSB   ;read current MSB state to
temp1 (next bit directions will be not changed)
            rcall MaskPortData
            out   LEDdirectionLSB,temp0   ;and   update   LSB   port
direction
            out   LEDdirectionMSB,temp1   ;and   update   MSB   port
direction
            ret
;---------------------------------------------------------------------
----------------------
SetOutDataPort:
            in    temp0,LEDPortLSB        ;read current LSB state to
temp0 (next data bits will be not changed)
            in    temp1,LEDPortMSB        ;read current MSB state to
temp1 (next data bits will be not changed)
            rcall MaskPortData
            out   LEDPortLSB,temp0        ;and update LSB data port
            out   LEDPortMSB,temp1        ;and update MSB data port
            ret
;---------------------------------------------------------------------
GetInDataPort:
            in    temp0,LEDPinMSB         ;read current MSB state to
temp0
            in    temp1,LEDPinLSB         ;read current LSB state to
temp1
MoveLEDin:
            bst   temp1,LEDlsb0           ;and  move  LSB  bits  to
correct positions (from temp1 to temp0)
            bld   temp0,0                 ;(MSB bits are in correct
place)
            bst   temp1,LEDlsb1
            bld   temp0,1
            bst   temp1,LEDlsb2
            bld   temp0,2
            mov   R0,temp0                ;and store result to R0
            ret
;---------------------------------------------------------------------
GetOutDataPort:
            in    temp0,LEDPortMSB        ;read current MSB state to
temp0
            in    temp1,LEDPortLSB        ;read current LSB state to
temp1
            rjmp  MoveLEDin
;---------------------------------------------------------------------
GetDataPortDirection:
            in    temp0,LEDdirectionMSB   ;read current MSB state to
temp0
            in    temp1,LEDdirectionLSB   ;read current LSB state to
temp1
            rjmp  MoveLEDin
;---------------------------------------------------------------------
EEPROMWrite:
            out   EEAR,ZL                 ;set the address of EEPROM
            out   EEDR,R0                 ;set the data to EEPROM
            cli                           ;disable interrupt
            sbi   EECR,EEMPE              ;set   the   master   write
enable
            sei                           ;enable   interrupt   (next
instruction is performed)
            sbi   EECR,EEPE               ;write
WaitForEEPROMReady:
```

```
            sbic  EECR,EEPE               ;wait to the end of write
            rjmp  WaitForEEPROMReady      ;in  loop  (max  cca  4ms)
(because of possible next reading/writing)
            rjmp  OneZeroAnswer           ;acknowledge reception with
single zero
;---------------------------------------------------------------------
;*********************************************************************
;* End of Program
;*********************************************************************
;*********************************************************************
;* End of file
;*********************************************************************
```