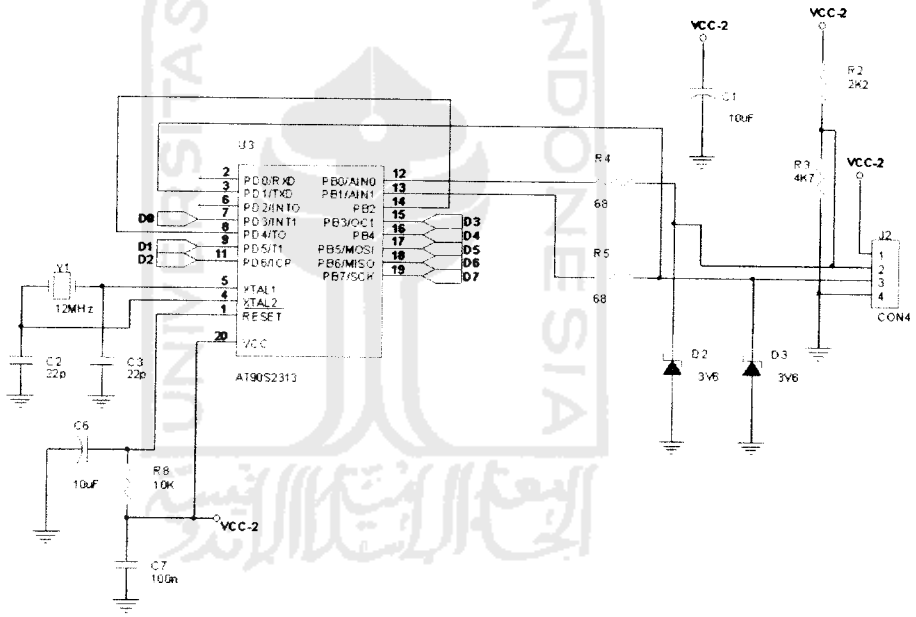
BAB IV

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.

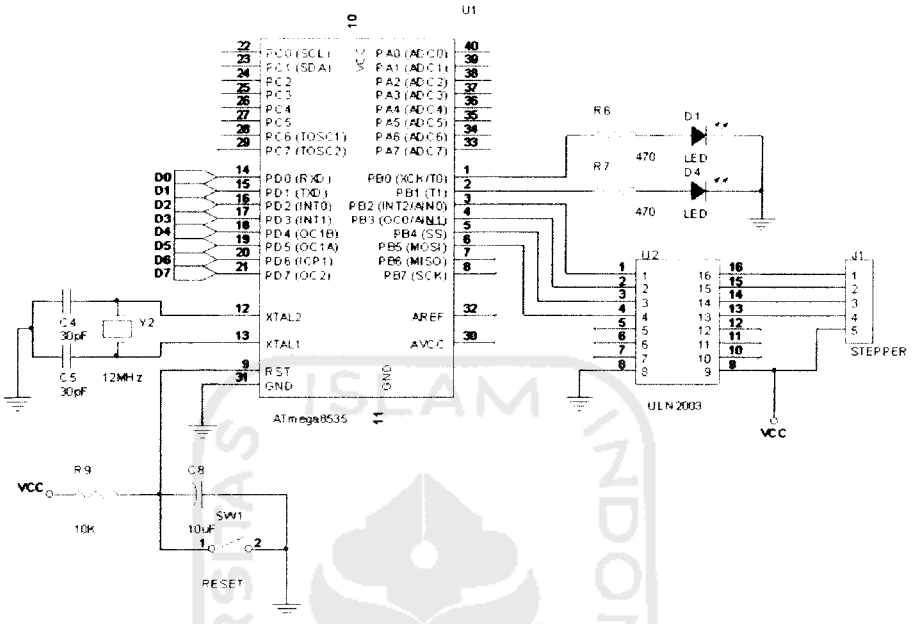
4.1.1 Analisa Rangkaian Mikrokontroler AT90S2313



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



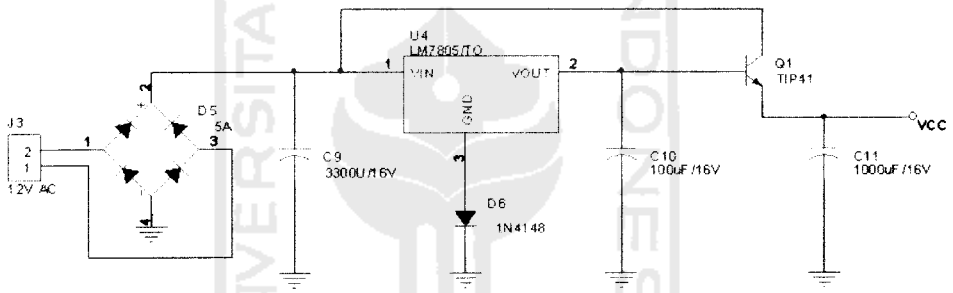
Gambar 4.2 Rangkaian mikrokontroler ATMega8535

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

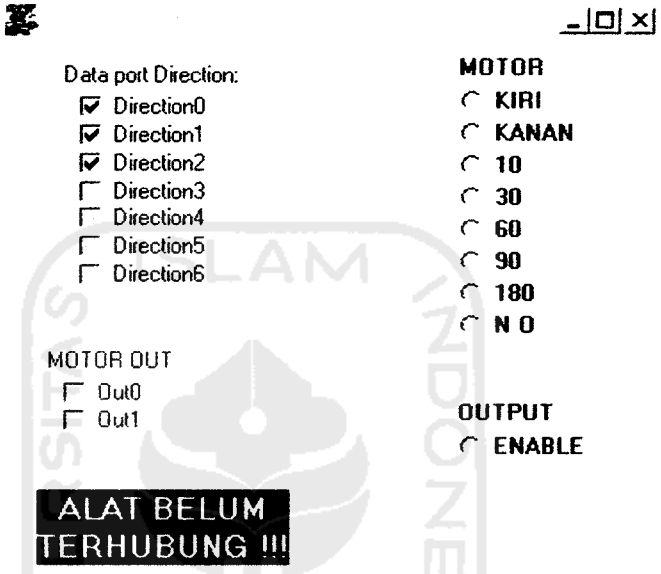
Pada rangkaian regulator ini tegangan bolak-balik(AC) dari transformator sebesar 12V akan disearahkan oleh dioda *bridge*. Keluaran dari penyearah ini akan diturunkan menjadi 5V dan distabilkan oleh IC LM7805. Untuk menyuplai rangkaian dengan kebutuhan arus yang cukup besar maka dibutuhkan penguat arus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



Gambar 4.3 Rangkaian *power* suplai dengan regulator

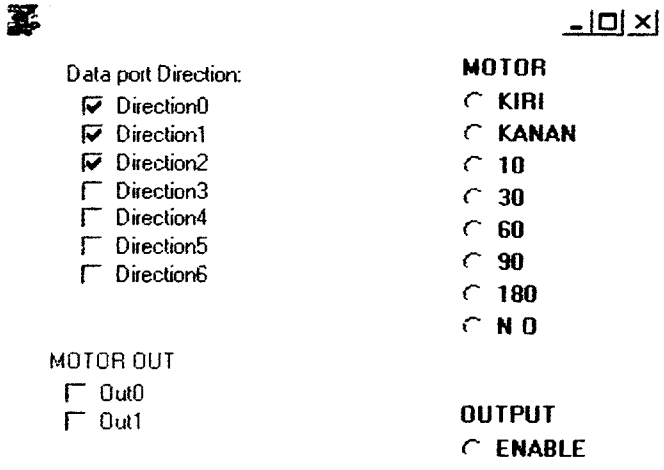
4.2 Analisa Software (perangkat lunak)

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.

FinishReceiving:

Merupakan pengkopian baris data dari paket penerimaan USB untuk penguraian kode paket (untuk penguraian kode NRZI dan *bitstuffing*).

USB Reset:

Inisialisasi *interface* USB terhadap nilai bawaannya (sejak dinyalakan)

SendPreparedUSBAnswer:

Mengirimkan kandungan *output buffer* yang telah disiapkan ke *line* USB, pengkodean NRZI dan *bitstuffing* dijalankan selama transmisi. Paket dihentikan dengan EOP.

ToggleDATAPID:

Identifikasi paket *Toggle* Data PID antara DATA0 dan DATA1 PID. *Toggling* ini penting selama transmisi untuk setiap spesifikasi USB.

ComposeZeroDATA1PIDAnswer:

Penyusunan jawaban *zero* selama transmisi. Jawaban *zero* tidak ada datanya dan digunakan untuk beberapa kasus sebagai jawaban ketika tidak ada data yang tersedia pada alat.

InitACKBuffer:

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

SendACK:

Mengirimkan paket ACK ke *line* USB.

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.

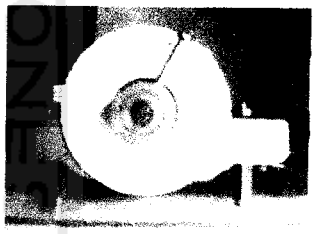
Tabel.4.3 Standar lokasi paket *setup* (*control transfer*)

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

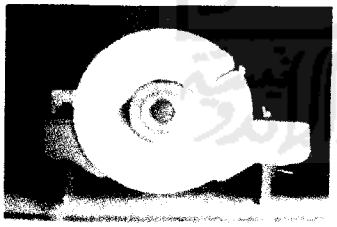
4.2.4 Tampilan *Hardware* Motor *Stepper*

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 :

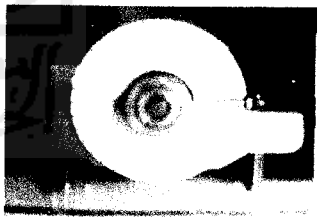
Gambar 4.6 Sudut 10 derajat ke kanan



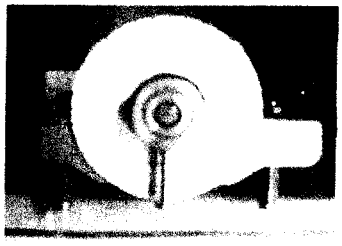
Gambar 4.7 Sudut 30 derajat ke kanan



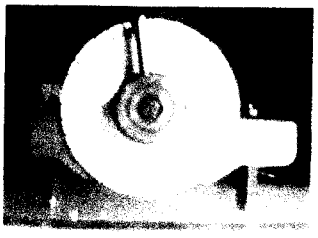
Gambar 4.8 Sudut 60 derajat ke kanan



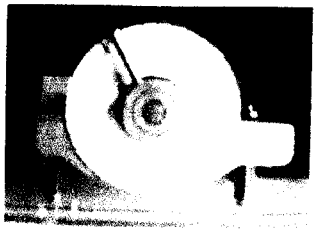
Gambar 4.9 Sudut 90 derajat ke kanan



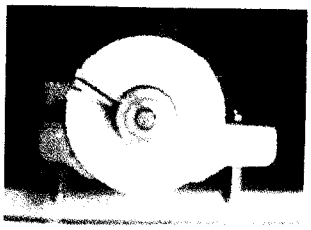
Gambar 4.10 Sudut 180 derajat ke kanan



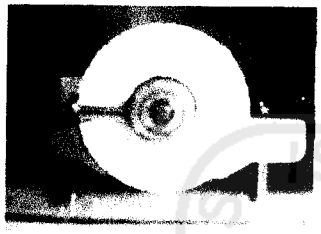
Gambar 4.11 Sudut 10 derajat ke kiri



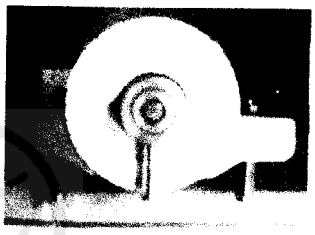
Gambar 4.12 Sudut 30 derajat ke kiri



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.