"Saya/kami akui bahawa saya telah membaca karya ini pada pandangan saya/kami karya ini adalah memadai dari skop dan kualiti untuk tujuan penganugerahan Ijazah Sarjana Muda Kejuruteraan Elektronik (Kejuruteraan Komputer)."

Tandatangan : ....
Nama Penyelia : PN NIZA BTE MOHD IDRIS
Tarikh : 05/05/06

A PC CONTROL SYSTEM FOR CONTROLLING BLINDS ON WINDOWS

MOHAMAD AZHRI BIN ZULKIFLI

Tesis Ini Dihantar Sebagai Memenuhi Sebahagian Daripada Syarat Penganugerahan
Ijazah Sarjana Muda Kejuruteraan Elektronik (Kejuruteraan Komputer)

Fakulti Kejuruteraan Elektronik dan Kejuruteraan Komputer
Kolej Universiti Teknikal Kebangsaan Malaysia

March 2006

Saya akui laporan ini adalah hasil kerja saya sendiri kecuali ringkasan dan petikan yang tiap-tiap satunya telah saya jelaskan sumbernya."

Tandatangan : ....

Nama Penulis : MOHAMAD AZHRI ZULKIFLI

Tarikh : 05 / 05 / 06

DEDIKASI

Untuk ayah dan ibu tersayang

ZULKIFLI BIN ABDULLAH
WAN SAADIAH BINTI TAIB

Abang dan adik

MOHAMAD AZHLAN
MOHAMAD RIDZUAN
MOHAMAD HAFIZ

Dan tidak lupa kepada semua rakan-rakan

PENGHARGAAN

Assalamualaikum….

Setinggi-tinggi ucapan kesyukuran ke atas Allah s.w.t kerana dengan izin-nya dapat juga saya menyiapkan projek sarjana muda ini.Setinggi-tinggi penghargaan terima kasih diucapkan juga kepada penyelia saya iaitu Pn Niza Binti Idris di atas bimbingan dan tunjuk ajar beliau berikan di sepanjang melaksanakan projek sarjana muda ini.

Tidak lupa juga, penghargaan ditujukan kepada Penyelaras Projek Sarjana Muda iaitu Mr Soo Yew Guan yang mana telah memberi sedikit sebanyak petunjuk mengenai laporan dan juga susun atur perjalanan projek, pensyarah-pensyarah, juruteknik-juruteknik yang banyak memberi kerjasama dalam menggunakan makmal-makmal yang berkaitan di sepanjang perjalanan projek ini dijalankan.

Akhir sekali, ucapan ribuan terima kasih ini juga ditujukan kepada kedua ibu bapa saya, ahli keluarga, kawan-kawan dan juga mana-mana pihak yang terlibat secara langsung ataupun tidak dalam menyiapkan dan menyempurnakan projek ini.

ABSTRAK

Projek ini adalah bertujuan untuk membina sebuah model tirai tingkap yang dikawal oleh komputer. Model melibatkan penggunaan perisian Visual Basic, *PIC (peripheral interface controller) microcontroller* dan Motor Servo. Konsep untuk model ini adalah *PIC microcontroller* akan menerima arahan atau data yang dimasukkan oleh pengguna melalui pengantaramuka grafik pengguna pada komputer, kemudian menggerakkan motor servo berdasarkan arahan tersebut. Pergerakkan motor servo akan menggerakkan kedudukan tirai pada tingkap. Projek ini dicadangkan penggunaanya untuk kegunaan di pejabat, rumah dan kilang. Sebagai contoh, ianya boleh digunakan oleh institusi pertanian yang mahu mengawal kadar pengcahayaan kawasan semaian anak benih, pejabat yang mempunyai bilangan tingkap yang banyak, menyelesaikan masalah mengubah kedudukan tirai tingkap bagi bingkai tingkap yang sukar dicapai dan banyak lagi yang boleh diaplikasikan oleh projek ini. Masalah yang dihadapi sebelum projek ini diperkenalkan ialah pengguna perlu ke setiap tingkap untuk mengubah kedudukan tirai secara manual atau menggunakan alat kawalan jarak jauh. Ini sudah pasti tidak praktikal untuk bangunan yang banyak tingkap. Kebaikkan yang diperolehi daripada projek yang direka ini ialah ianya dapat menyelesaikan masalah yang dihadapi dan menjimatkan masa pengguna.

ABSTRACT

The project when completion will allow user to control windows curtain using a computer. Visual Basic is use as it platform in designing the grafic user interface. As a controller, the PIC (peripheral interface controller) and the servo motor will be use as an output device to make the system working. User will gain an easy control of window curtain by only intercept with the grafic user interface that has been design. This will result the opening and closing of the window curtain with the designed interface. The system can be implementing as a light control mechanism to the agriculture industries where controlling light to plants are critical, office or residence with a large space area and in solving the problem in controlling curtain for a huge window type where user will no have to done it manually like what they have done normally. An easy problem solving system can be implement to the daily life problem such the curtain controlling during completion if this system.

ISI KANDUNGAN

SENARAI JADUAL

SENARAI RAJAH

SENARAI SINGKATAN

CISC	Complex Intruction System Computer
CMOS	Complementary Metal Oxide Semiconductor
CTS	Clear To Send
DCE	Data Communication Equipment
DTE	Data Terminal Equipment
EEPROM	Electrical Erasable Programmable Random Access Memory
GUI	Graphical User Interface
LED	Light Emmiting Diode
PDA	Personal Digital Assistant
PIC	Peripheral Interface Controller
PLC	Programmable Logic Controller
RAM	Random Access Memory
RISC	Reduce Instruction System Computer
ROM	Read Only Memory
RTS	Ready To Send

BAB 1

PENDAHULUAN

1.1 PENGENALAN

Sistem kawalan pada masakini telah berubah dan lebih tertumpu kepada sistem kawalan berkomputer. Secara amnya sistem kawalan pada masa lepas lebih tertumpu kepada sistem kawalan analog. Sistem kawalan analog ialah sistem kawalan yang menggunakan komponen seperti butang kawalan, suis, perintang boleh laras dan sebagainya. Manakala pada masakini kawalan terhadap sesuatu sistem lebih kepada penggunaan komputer.

Terdapat banyak perisian yang boleh diguna untuk membangunkan sesebuah sistem kawalan digital. Salah satu yang paling digemari dan mudah digunakan adalah perisian *Visual Basic 6.0.* Dengan menggunakan perisian ini, sesebuah sistem kawalan boleh dibangunkan dengan mempunyai fungsi yang mesra pengguna. Sistem yang telah dibangunkan untuk projek ini adalah sebuah sistem kawalan motor servo di mana sistem dapat mengawal pergerakkan samaada ke kiri atau ke kanan.

Aplikasi untuk projek ini adalah cuba menggerakkan kedudukan tirai menggunakan pengantaramuka grafik pengguna (*Graphical User Interface*) pada komputer. Konsep untuk model ini adalah *Programmable Interface Controller (PIC) Microcontroller* akan menerima arahan atau data yang dimasukkan oleh pengguna melalui pengantaramuka grafik pengguna pada komputer, kemudian menggerakkan motor servo berdasarkan arahan tersebut. Pergerakkan motor servo akan menggerakkan kedudukan tirai pada tingkap.

Penggunaan *PIC microcontroller* adalah tepat kerana ia merupakan litar bersepadu yang mudah diprogram, murah, dan mempunyai keupayaan operasi yang tinggi. Ia boleh dikatakan sebagai sebuah komputer yang lengkap dalam litar bersepadu yang kecil. Dalam projek ini *PIC microcontroller* yang digunakan ialah PIC16F84A. Manakala motor yang digunakan untuk projek ini ialah motor servo. Motor servo dipilih kerana ia mempunyai kuasa kilasan yang tinggi, bersaiz kecil, mudah dikawal, lengkap dengan litar dalaman dan lengkap dengan kotak gear dalaman.

1.2 OBJEKTIF PROJEK

Objektif projek rekabentuk sistem pengawalan tirai tingkap berkomputer ialah:

1. Merekabentuk sebuah model prototaip tirai tingkap yang dikawal sepenuhnya menggunakan komputer.
2. Mengkaji aturcara menggunakan perisian *Visual Basic* yang boleh membenarkan pengguna berkomunikasi melalui pengantaramuka grafik pengguna bagi mengawal sistem.

3. Membangunkan aturcara *PIC microcontroller* yang boleh menerima arahan melalui kabel sesiri dari komputer dan seterusnya mengeluarkan keluaran seperti yang dikehendaki.
4. Memastikan pergerakkan motor servo berdasarkan arahan yang diberi melalui pengantaramuka grafik pengguna pada komputer.
5. Menjimatkan masa pengguna untuk mengubah kedudukan tirai tingkap mereka.
6. Mengaplikasikan segala pengetahuan yang diperolehi dalam bidang elektronik dan komputer seperti mengkaji dan membangunkan litar elektronik, aturcara *PIC microcontroller*, aturcara perisian *Visual Basic* dan pengantaramuka grafik pengguna yang mesra pengguna di dalam mencipta sesuatu yang memudahkan kehidupan seharian.
7. Merekabentuk dan merealisasikan satu sistem yang menggabungkan antara mekanikal dan elektronik yang mana bersesuaian dengan kursus yang diceburi.

1.3 KEPENTINGAN PROJEK

Sekiranya model prototaip berjaya berfungsi sepenuhnya, maka ia boleh dimajukan untuk dijadikan salah satu elemen penghasilan rumah pintar. Seperti yang diketahui rumah pintar merupakan sebuah rumah yang menggunakan komputer sepenuhnya dalam pengawalan seperti mengawal buka dan tutup pintu pagar rumah. Selain itu, pengawalan menggunakan komputer terhadap pergerakkan motor servo dapat diadaptasikan ke atas aplikasi lain seperti projek sistem pengawal tangan robotik (*robotic arm controller*). Selain itu projek ini amat berguna terhadap projek yang berasaskan kepada aplikasi robotik. Buat masa ini, pengawalan tirai tingkap menggunakan komputer masih tidak terdapat di sekitar Negeri Melaka. Teknologi yang digunakan untuk mengawal tirai tingkap yang sedia ada pada masakini menggunakan alat kawalan jauh. Maka, diharap projek prototaip ini dapat dijadikan inovasi kepada kaedah pengawalan tirai tingkap yang lebih efisien.

1.4 PENYATAAN MASALAH

Aplikasi pengawalan turun dan naik tirai dipilih untuk projek ini disebabkan beberapa faktor masalah yang wujud dalam persekitaran kita. Antara masalah yang dapat diselesaikan ialah:

1. Projek ini boleh digunakan oleh kilang yang mahu mengawal kadar pencahayaan untuk penghasilan produk. Kadar pencahayaan yang masuk ke dalam bangunan dapat dikawal melalui komputer dimana komputer akan mengawal kedudukan tirai pada tingkap bangunan berdasarkan kemahuan pengguna.
2. Bagi pejabat yang mempunyai banyak tingkap, ia akan memudahkan pengawalan kedudukan tirai dengan hanya menggunakan sebuah komputer. Pengguna tidak perlu ke setiap tingkap untuk membuka atau menutup tirai.
3. Selain itu, ia juga dapat menyelesaikan masalah mengubah kedudukan tirai tingkap bagi bingkai tingkap yang sukar dicapai.
4. Menyelesaikan masalah penggunaan alat kawalan jauh sekiranya sesebuah rumah itu mempunyai bilangan bilik yang banyak. Pengguna tidak perlu ke setiap bilik untuk mengubah kedudukan tirai.
5. Dapat menjimatkan masa pengguna.

1.5 METADOLOGI PENYELIDIKAN

Untuk memudahkan perlaksanaan projek dijalankan ia dibahagi kepada empat bahagian yang penting. Bahagian yang pertama ialah kajian literatur yang bertujuan untuk mendapatkan pemahaman keseluruhan projek yang akan dibina. Bahagian yang kedua ialah pembinaan litar dan pembangunan aturcara. Bahagian yang ketiga ialah pengujian litar dan bahagian yang terakhir ialah bahagian pembinaan litar menggunakan papan litar bercetak dan melengkapkan prototaip.

Bahagian pertama ialah kajian literatur penting untuk memahami teori dan membuat pemilihan terhadap peranti yang ingin diguna. Ia dapat dilakukan dengan membuat rujukan buku-buku, jurnal, artikel, serta halaman internet yang bersesuaian untuk menjayakan projek ini.

Bahagian kedua ialah pembinaan litar dan pembangunan aturcara. Bahagian pembinaan litar terbahagi kepada dua iaitu litar satu dan litar dua. Dua litar telah dibina untuk dijadikan ujikaji. Pengubahsuaian telah dilakukan untuk membaiki litar yang sedia ada. Litar yang terbaik dan sesuai akan dibangun di atas papan litar bercetak. Manakala pembangunan aturcara terbahagi kepada dua iaitu pembangunan aturcara *PIC microcontroller* dan pembangunan aturcara *visual basic.*

Bahagian ketiga ialah bahagian pengujian litar. Setelah aturcara dan litar-litar ujikaji siap dibangunkan, aturcara akan dimuat turun ke dalam *PIC microcontroller* untuk pengujian. Dalam proses ini masalah yang timbul akan dikenalpasti dan cuba diselesaikan.

Bahagian keempat iaitu bahagian terakhir yang mana ia penting untuk menunjukkan hasil ataupun keluaran setelah ketiga-tiga bahagian disempurnakan. Bahagian ini ialah bahagian pembinaan litar menggunakan papan litar bercetak dan melengkapkan prototaip. Setelah ujikaji terhadap kedua-dua litar dijalankan, litar yang dijangkakan sesuai dan berpotensi akan dipilih untuk diadaptasikan di atas papan litar bercetak.

1.6 KANDUNGAN TESIS

Tesis ini dibahagikan kepada lima bab utama. Bab 1 mengandungi pengenalan kepada projek, objektif, kepentingan, penyataan masalah dan metodologi penyelidikan. Bab 2 pula mengandungi kajian literatur seperti kajian terhadap litar, komponen berkaitan, jenis motor yang sesuai digunakan dan cara penggunaan beberapa perisian utama seperti MPLAB IDE v7.20 dan IC-Prog 1.05A. Bab 3 pula menjelaskan metadologi pembangunan projek dari mula projek dibangunkan sehingga projek siap sepenuhnya. Manakala bab 4 ialah bab keputusan projek. Di dalam bab ini akan diterangkan mengenai litar yang dibangunkan, pengubahsuaian terhadap litar dan pembangunan perisian. Bab 5 merupakan bab yang terakhir yang mengandungi kesimpulan dan cadangan terhadap projek yang dibangunkan.

BAB 2

KAJIAN LITERATUR

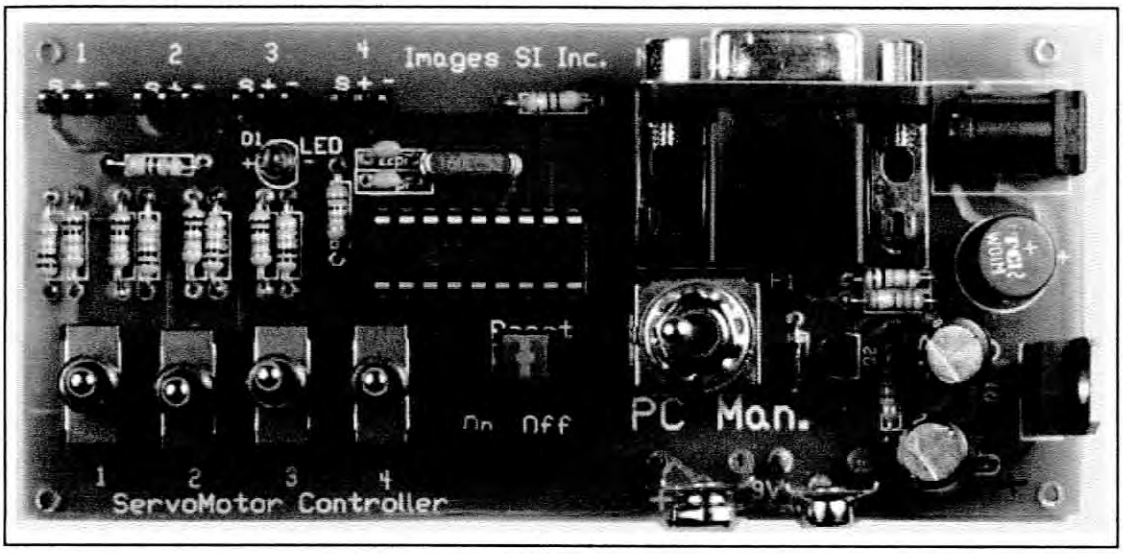
Untuk memudahkan pelaksanaan projek, kajian literatur telah dilakukan terhadap perkara-perkara yang ada kaitan dengan projek ini. Kajian yang dilakukan adalah bertujuan untuk menyelesaikan masalah dan limitasi projek yang sedia ada. Disamping itu ia akan meningkatkan pemahaman berkaitan kaedah yang digunakan dan cara pengoperasiannya. Kajian latar belakang yang dilakukan ini terbahagi kepada 5 bahagian iaitu:

1. Kajian terhadap litar-litar yang digunakan untuk mengawal motor servo menggunakan *PIC microcontroller*. Tumpuan diberikan kepada cara pengoperasiannya.
2. Kajian terhadap *controller* seperti *programmable logic controller (PLC), microprocessor* dan *microcontroller* kerana ia adalah komponen utama dalam projek ini. Kajian terhadap komponen dilakukan bertujuan untuk memilih *controller* yang paling sesuai untuk pembangunan projek.
3. Kajian terhadap motor servo. Tumpuan diberikan pada spesifikasi motor servo tersebut dan konsep kawalan motor servo menggunakan *PIC microcontroller*.

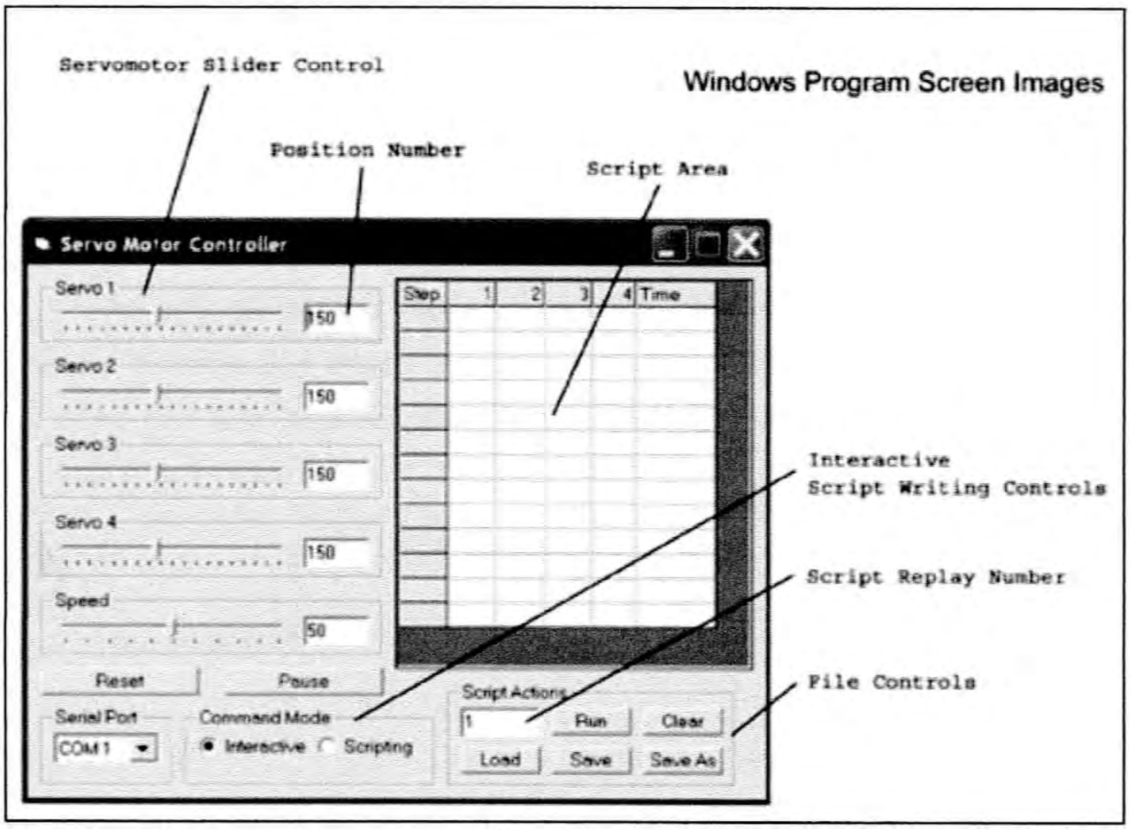
4. Kajian terhadap port kabel sesiri. Kabel sesiri digunakan sebagai medium penghantaran data atau arahan dari komputer ke litar *PIC microcontroller*.
5. Penerangan perisian berkaitan seperti *MPLAB IDE* dan *IC-Prog 1.05A*. Kaedah penggunaan kedua-dua perisian ini akan diterangkan secara ringkas.

2.1 LITAR KAWALAN MOTOR SERVO

Tumpuan diberikan kepada pembangunan litar kawalan motor servo menggunakan *PIC microcontroller*. Setelah kajian untuk pemilihan motor dan *controller* dilakukan, motor servo dan *PIC microcontroller* dikenal pasti sebagai komponen yang paling sesuai untuk pembangunan litar projek. Oleh kerana itu, kajian literatur secara lebih mendalam terhadap pembangunan litar kawalan motor servo dijalankan berbanding litar kawalan motor stepper dan litar kawalan motor arus terus. Litar kawalan motor servo terdapat dijual di internet dengan harga mencecah USD 99.95. Rajah 2.1 merupakan salah satu litar kawalan motor servo yang dijual di internet. Litar tersebut menggunakan suis dan pengantaramuka grafik pengguna seperti dalam Rajah 2.2 sebagai unit kawalan motor.



Rajah 2.1: Litar Kawalan Motor Servo



Servo 1 150

Servo 2 150

Servo 3 150

Servo 4 150

Speed 50

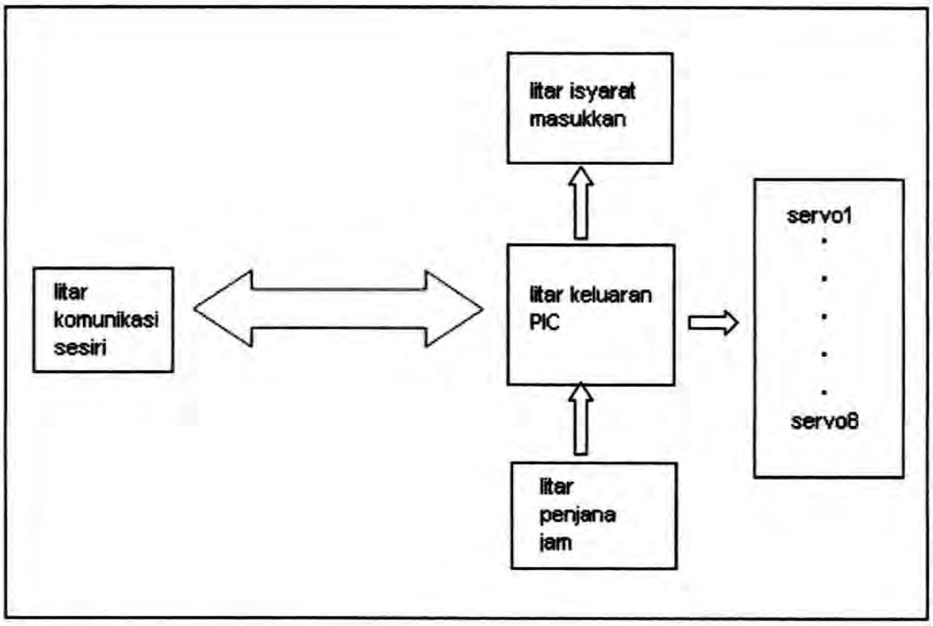
Serial Port COM 1

Command Mode ● Interactive ○ Scripting

Script Actions 1

Rajah 2.2: Pengantaramuka Grafik Pengguna

2.1.1 Gambarajah Blok Litar Kawalan Motor Servo



Rajah 2.3: Gambarajah Blok Litar Kawalan Motor Servo

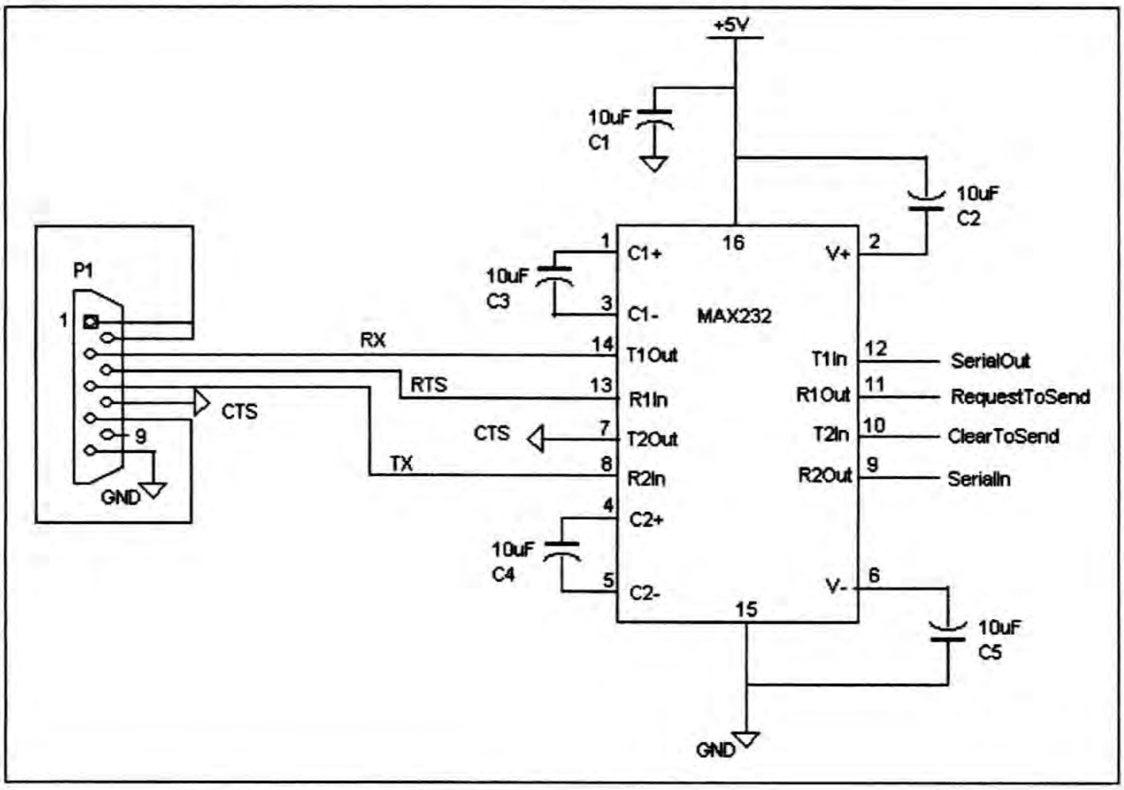
Daripada gambarajah blok di atas, litar kawalan motor servo merupakan gabungan beberapa litar seperti litar komunikasi sesiri, litar penjana jam, litar isyarat masukkan dan litar keluaran PIC16F84A. Litar ini mampu menampung keluaran sebanyak 8 buah motor servo.

2.1.2 Litar Komunikasi Sesiri

Dengan merujuk Rajah 2.4, litar komunikasi sesiri ini dihubungkan dengan komputer utama bermula di P1, penghubung BD9 perempuan (*female DB9 connector)*. Pin 1,4 dan 6 disambungkan bersama untuk memastikan komputer utama menerima data sesiri dari *PIC microcontroller* pada bila-bila masa. Pin 5 disambungkan ke litar bumi manakala pin yang selebihnya disambungkan ke MAX232.

Chip MAX232 digunakan untuk menukar isyarat voltan dari RS232 kepada isyarat logik CMOS yang boleh digunakan oleh *PIC microcontroller*. Ia mempunyai dua penerima yang menukar isyarat RS232 kepada isyarat logik CMOS dan dua pemancar yang menukar isyarat logik CMOS ke isyarat voltan RS232. Kapasitor C1, C2, C3, C4, dan C5 digunakan untuk menjana voltan +10 volt dan – 10 volt.

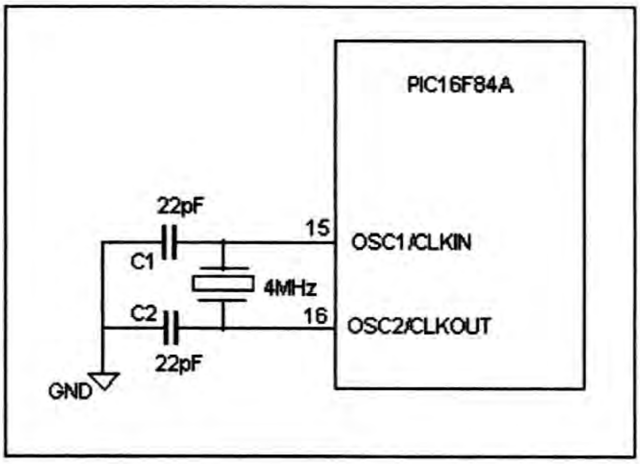
Garis RX dari port sesiri disambungkan kepada keluaran pemancar pertama (T1Out) di mana ia dikawal oleh isyarat *Serial Out* dari *PIC microcontroller*. Garis TX disambungkan kepada masukkan penerima kedua (R2In) di mana ia mengawal isyarat *Serial In* yang disambungkan ke *PIC microcontroller*. Garis CTS dari port sesiri disambungkan kepada keluaran pemancar kedua (T2Out) di mana ia dikawal oleh isyarat *Clear To Send* dari *PIC microcontroller*. Manakala garis RTS disambungkan kepada masukkan penerima pertama (R1In) di mana ia mengawal isyarat *Request To Send* yang disambungkan ke *PIC microcontroller*.



Rajah 2.4: Litar Komunikasi Sesiri

2.1.3 Litar Penjana Jam

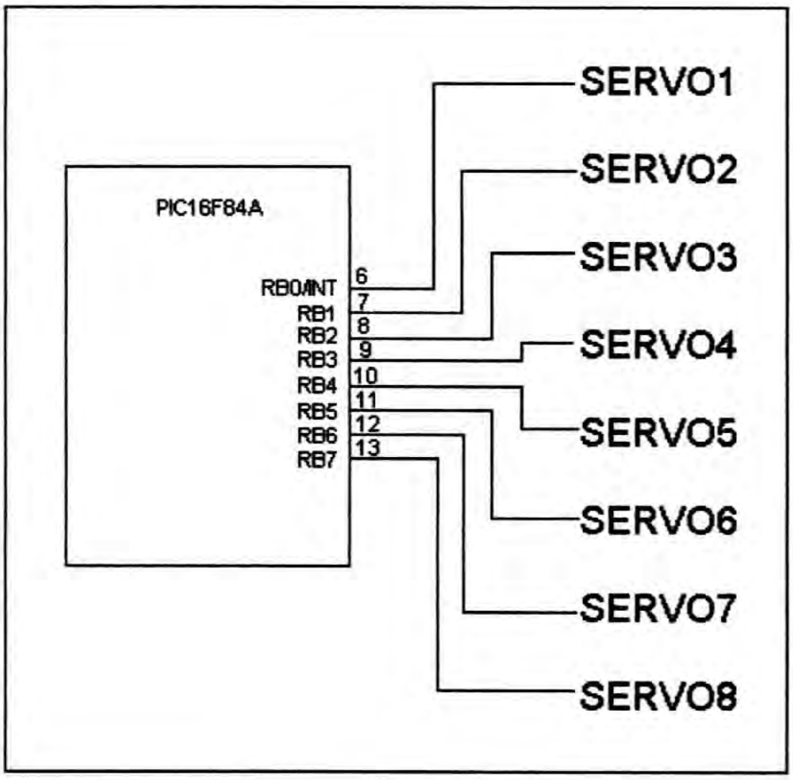
Dengan merujuk Rajah 2.5, PIC16F84A ini menggunakan pengayun kristal $4x10^6$ Hertz. Ia juga menggunakan dua kapasitor seramik bernilai $22x10^{-12}$ F (C1 dan C2). Isyarat jam ini dibahagikan kepada 4 bahagian untuk mengawal jam arahan *PIC microcontroller* bagi membolehkan *PIC microcontroller* melaksanakan arahan dalam masa $1x10^{-6}$ saat untuk setiap arahan.



Rajah 2.5: Litar Penjana Jam

2.1.4 Litar Keluaran PIC16F84A

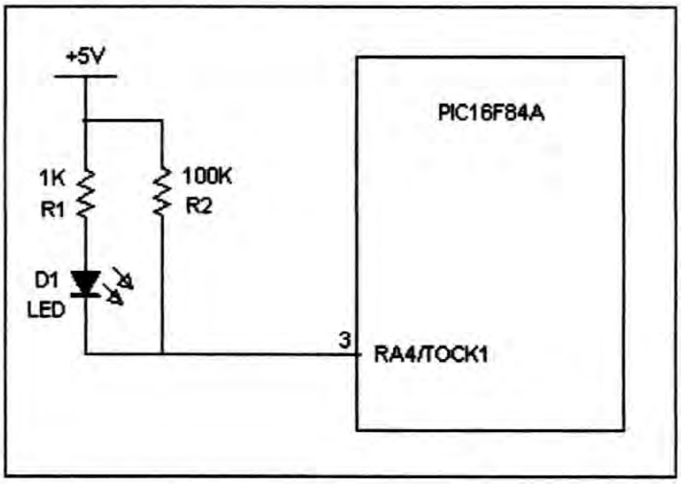
Dengan merujuk Rajah 2.6, semua pin RB bermula dari RB0 ke RB7 digunakan sebagai keluaran *PIC microcontroller* untuk mengawal motor servo. Kesemua pin ini akan disambungkan ke pin isyarat masukkan pada motor servo.



Rajah 2.6: Litar Keluaran PIC16F84A

2.1.5 Litar Isyarat Masukkan

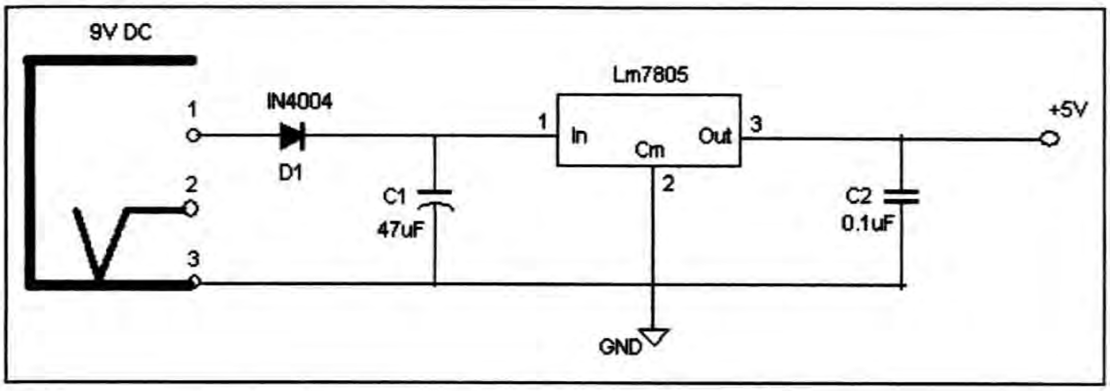
Dengan merujuk Rajah 2.7, LED D1 disambungkan pada pin RA4 dan pada bekalan kuasa 5 volt melalui perintang R1. Perintang R1 bertindak sebagai perintang penghad arus. Aturcara PIC akan menyalakan LED setiap kali bit data diterima dari komputer utama. Ia akan menyala semasa RA4 dalam keadaan aktif rendah.



Rajah 2.7: Litar Isyarat Masukkan

2.2 LITAR BEKALAN KUASA

Dengan merujuk Rajah 2.8, litar bersepadu alat atur (*IC Regulator*) digunakan untuk mendapatkan keluaran 5 volt yang tepat. Litar bersepadu yang digunakan ialah Lm7805. Diod D1 menawarkan kekutuban terbalik untuk keselamatan litar. Kapasitor C2 digunakan menstabilkan pengayun manakala kapasitor C1 digunakan untuk menapis kuasa dari bekalan 9 volt.



Rajah 2.8: Litar Bekalan Kuasa

2.3 *CONTROLLER*

Terdapat pelbagai jenis *controller* yang boleh digunakan untuk mengawal sesuatu sistem. Tiga *controller* utama yang banyak digunakan pada masakini ialah *programmable logic controller (PLC), microprocessor* dan *microcontroller*. PLC banyak digunakan dalam aplikasi sistem kawalan peralatan elektrik. PLC selalunya digunakan di kilang-kilang sebagai sistem kawalan produksi produk. Pada umumnya ia jarang digunakan untuk sistem kawalan elektronik. Untuk menggunakan jenis *controller* ini di dalam projek agak tidak sesuai memandangkan harganya yang terlalu mahal. Harga untuk *PLC controller* selalunya melebihi RM500. Harga ini agak mahal jika dibandingkan dengan *microprocessor* atau *Microcontroller*.

Microprocessor selalunya digunakan dalam sistem kawalan elektronik. Ia banyak digunakan di dalam telepon mudah alih, pembantu digital peribadi (PDA) dan sistem kawalan robotik. Litar bersepadu microprocessor tidak boleh bekerja sendiri. Ia memerlukan litar bersepadu ingatan, litar bersepadu masukkan dan keluaran, litar bersepadu penjana jam dan keluaran (*peripheral device*). Ini memberi kelebihan kepada *microprocessor* di mana ia mudah di tingkat upaya. Contoh *microprocessor* ialah Intel 8085, Atmel dan Motorola 68000.

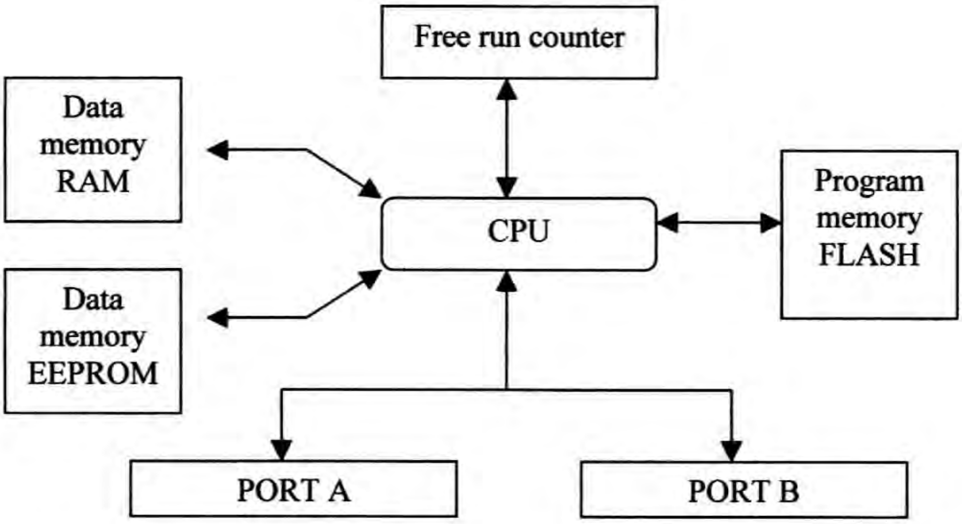
Manakala *microcontroller* merupakan sebuah komputer yang lengkap di dalam litar bersepadu yang kecil. *Microcontroller* mempunyai unit pemproses pusat (CPU), ingatan terbina dalaman, daftar, litar masukkan dan keluaran. Ia merupakan litar bersepadu yang boleh beroperasi tanpa memerlukan peralatan atau litar tambahan. Ini membezakannya dengan *microprocessor*. Kelebihan *microcontroller* berbanding *controller* yang lain ialah ia murah dan senang digunakan. Keburukannya pula ialah ia sukar untuk di tingkat upaya. *Controller* jenis ini banyak digunakan dalam kamera digital, television, robot, peralatan elektronik dan sistem kawalan kenderaan.

PIC16F84A microcontroller telah dipilih untuk digunakan sebagai *controller* dalam projek ini. Pemilihan disebabkan oleh harganya yang murah dan senang diprogram.

2.3.1 Kajian Terhadap PIC16F84A

PIC (*Peripheral Interface Controller*) seperti PIC 16F84A merupakan sebuah komputer yang lengkap di dalam litar bersepadu yang kecil. Ia mengandungi:

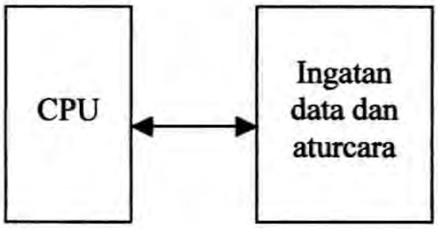
1. Unit Pemproses Pusat (CPU)
2. Daftar (*Registers*)
3. Masukkan/Keluaran (*Input/Output*)
4. Ingatan (*Memory*)

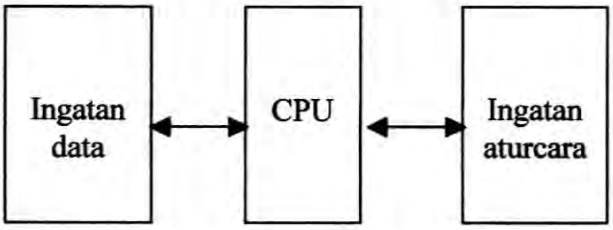


Rajah2.9: Gambarajah blok PIC16F84A

2.3.1.1 Senibina *PIC microcontroller*

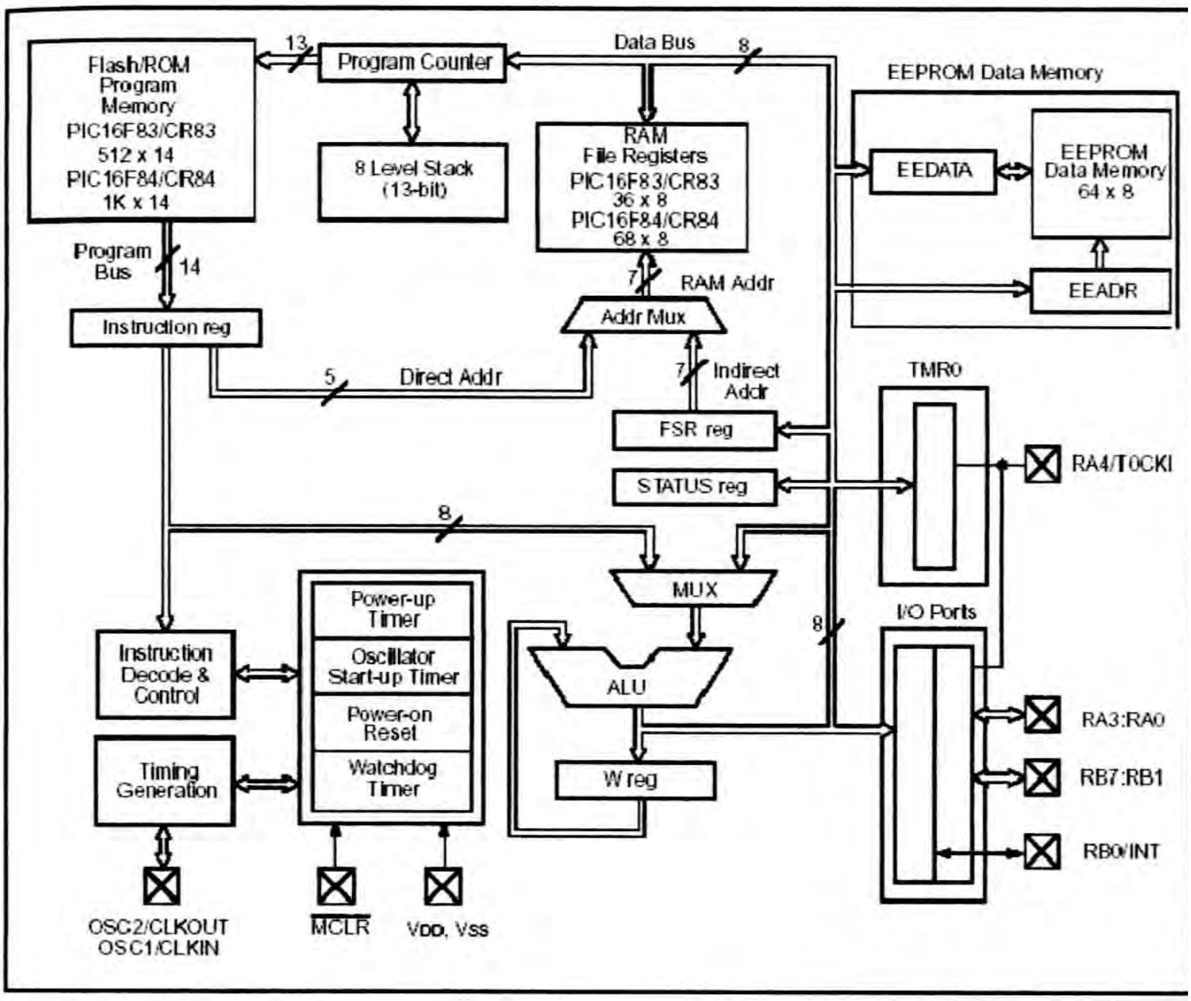
Microcontroller terdahulu menggunakan senibina Von-Nuemann. Senibina ini membenarkan data dan program disimpan pada ingatan yang sama. Tetapi semua jenis *PIC microcontroller* menggunakan senibina Harvard. Senibina ini memisahkan ingatan data dan ingatan program. Rajah 2.7 dan Rajah 2.8 di bawah menunjukkan senibina Von-Nuemann dan Harvard.





Rajah 2.10: Senibina Von-Nuemann

Rajah 2.11: Senibina Harvard



Rajah 2.12: Gambarajah Blok PIC16F84A yang lengkap

2.3.1.2 Teknologi EEPROM

PIC19F84A menggunakan teknologi *electrical erasable programmable random access memory*. EEPROM adalah dari jenis PROM yang mana data hanya dapat dipadamkan dengan mendedahkannya kepada cas elektrik. Data yang disimpan dalam ingatan EEPROM tidak akan hilang walaupun bekalan kuasa diputuskan. Seperti juga jenis ROM yang lain, EEPROM tidak sepantas ingatan RAM. EEPROM adalah lebih menyerupai ingatan *flash* atau *flash EEPROM.* Perbezaan prinsip untuk jenis-jenis ingatan ini ialah EEPROM perlu ditulis atau dipadam data dalam satu bait pada satu masa, sedangkan ingatan *flash* membenarkan data ditulis atau dipadam dalam bentuk block. Ini menjadikan ingatan *flash* laju. Teknologi ini membenarkan 10,000,000 kitaran tulis atau padam dalam ingatan data.

2.3.1.3 Ingatan *Flash* untuk Ingatan Program.

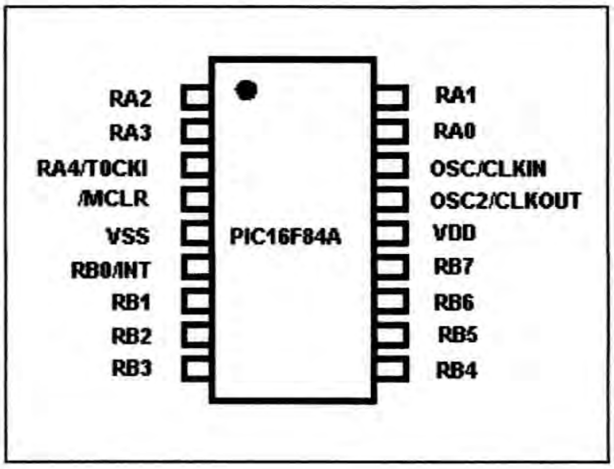
Ingatan *flash* ialah untuk menyimpan aturcara atau program yang ditulis. Ingatan *flash* membenarkan tulis atau padam pada lokasi ingatan yang banyak dalam satu operasi aturcara. Ingatan yang dibangunkan daripada teknologi *flash* boleh diprogram dan dibersihkan programnya lebih dari sekali. Ini menjadikan microcontroller ini sesuai untuk pembangunan perkakasan.

2.3.1.4 Masa Lari Bebas (*free-run-timer*)

Daftar yang berkapasiti 8 bit dan terbina di dalam microcontroller. Ia akan menambah nilai setiap 4 kitaran frekuensi jam sehingga mencapai nilai tertinggi iaitu 255.

2.3.1.5 Pin PIC16F84A

PIC16F84A mempunyai 18 pin. PIC ini mempunyai 13 port masukkan atau keluaran Rajah 2.13 di bawah menunjukkan pin layout.



Rajah 2.13: Gambarajah Pin PIC16F84A

2.3.1.6 Set Arahan PIC16F84A

Ada dua jenis set arahan untuk microcontroller. Set yang pertama ialah *complex instruction set computer (CISC)* dan set yang kedua ialah *reduce instruction set computer (RISC)*. Semua *PIC microcontroller* menggunakan jenis set arahan RISC. Untuk PIC16F84A hanya mempunyai 35 arahan sahaja. Setiap arahan dilaksanakan dalam satu kitaran kecuali *CALL, GOTO, BTFSC, BTFSS, RETURN, RETLW, RETFIE, INCFSZ dan DECFSZ* yang memerlukan dua kitaran.

Jadual 2.1: Set Arahan PIC16F84A

Mnemonic, operand	*Description*	*Cycle*	*Status affected*
BYTE-ORIENTED FILE REGISTER OPERATIONS			
ADDWF f,d	*Add W and F*	*1*	*C,DC,Z*
ANDWF f,d	*AND W with f*	*1*	*Z*
CLRF f	*Clear f*	*1*	*Z*
CLRW	*Clear W*	*1*	*Z*
COMF f,d	*Compliment f*	*1*	*Z*
DECF f,d	*Decrement f*	*1*	*Z*
DECFSZ f,d	*Decrement f, skip if 0*	*1(2)*	
INCF f,d	*Increment f*	*1*	*Z*
INCFSZ f,d	*Incremenff, skip if 0*	*1(2)*	
IORWF f,d	*Inclusive OR W with f*	*1*	*Z*
MOVF f,d	*Move f*	*1*	*Z*
MOVWF f	*Move W to f*	*1*	
NOP	*No operation*	*1*	
RLF f,d	*Rotate left f through Carry*	*1*	*C*
RRF f,d	*Rotate right f through Carry*	*1*	*C*
SUBWF f,d	*Subtract W from f*	*1*	*C,DC,Z*
SWAPF f,d	*Swap nibbles in f*	*1*	
XORWF f,d	*Exclusive OR W with f*	*1*	*Z*
BIT-ORIENTED FILE REGISTER OPERATIONS			
BCF f,d	*Bit clear f*	*1*	
BSF f,d	*Bit set f*	*1*	
BTFSC f,d	*Bit test f, skip if clear*	*1(2)*	
BTFSS f,d	*Bit test f, skip if set*	*1(2)*	
LITERAL AND CONTROL OPERATION			
ADDLW k	*Add literal and W*	*1*	*C,DC,Z*
ANDLW k	*AND literal with W*	*1*	*Z*
CALL k	*Call subroutine*	*2*	
CLRWDT	*Clear watchdog timer*	*1*	*/TO,/PD*
GOTO k	*Go to address*	*2*	
IORLW k	*Inclusive OR literal with W*	*1*	*Z*
MOVLW k	*Move literal to W*	*1*	
RETFIE	*Return from interupt*	*2*	
RETLW k	*Return with literal in W*	*2*	
RETURN	*Return from sunroutine*	*2*	
SLEEP	*Go into stanby mode*	*1*	*/TO,/PD*
SUBLW k	*Subtract W from literal*	*1*	*C,DC,Z*
XORLW k	*Exclusive OR literal with W*	*1*	*Z*

Jadual 2.2: Penjelasan Simbol

Symbol	***Description***
f	*Register file address (0x00 to 0x7F)*
w	*Working register*
b	*Bit address within 8 bit file register*
k	*Literal field, constant data or label*
X	*Don't care location*
d	*Destination select:* *0 = store result in W* *1 = store result in register f*
PC	*Program counter*
TO	*Time out bit*
PD	*Power down bit*

2.4 KAJIAN TERHADAP MOTOR SERVO

Antara spesifikasi pemilihan motor servo adalah kerana ia sangat berkuasa, bersaiz kecil, mempunyai daya kilasan yang tinggi, mudah dikawal, lengkap dengan litar dalaman dan lengkap dengan kotak gear dalaman. Motor servo Hitec HS-322HD adalah yang paling sesuai untuk model prototaip ini.

Kleas: Mini Servo

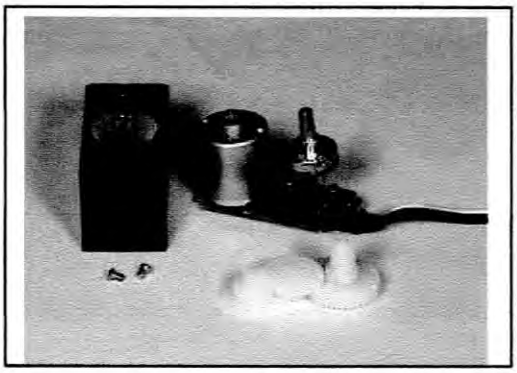
Voltan: 4.8v or 6.0v

Spesifikasi umum:

- Berat: 43 gram
- Gear: Standart
- Kelajuan: 0.19/0.17 saat @ 4.8v/6.0v
- Daya Kilasan: 3.0 /3.7Kg @ 4.8v/6.0v
- 1.57 x 0.78 x 1.43 inci

Rajah 2.14: Motor Servo Hitec HS-322HD

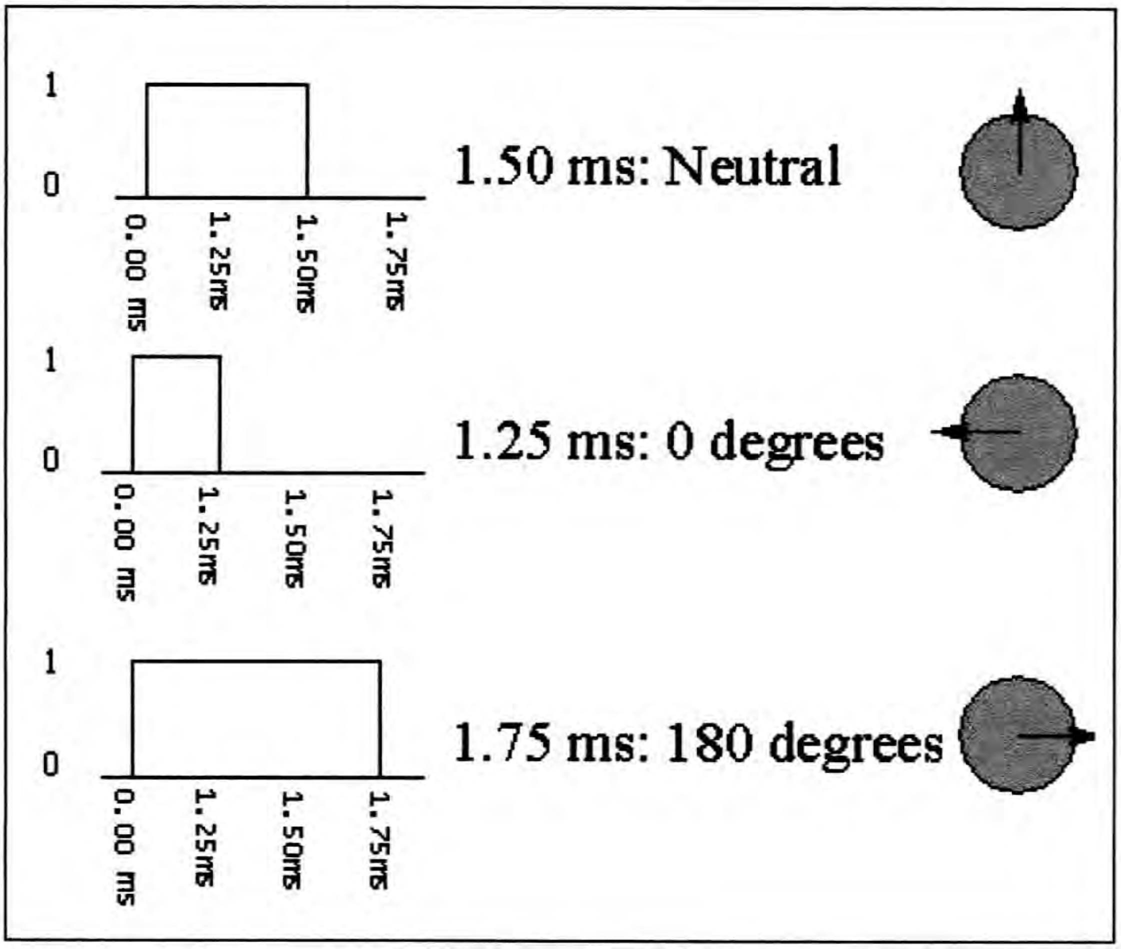
Rajah 2.15: Servo Bersaiz Kecil



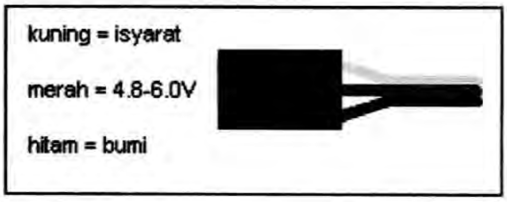
Rajah 2.16: Litar dan Kotak Gear Dalaman

2.4.1 Cara Mengawal Pergerakkan Motor Servo

Motor servo dikawal dengan memanipulasikan *pulse coded modulation.* Servo akan melihat denyut ini setiap 20x10^{-3} saat. Lebar denyut ini akan menentukan kadar putaran motor servo. Sebagai contoh, 1.5x10^{-3} saat denyut akan menyebabkan servo berputar sebanyak 90 darjah. Jika lebar denyut kurang dari 1.5x10^{-3} saat, maka servo akan berputar menghampiri 0 darjah. Dan sekiranya lebar denyut melebihi 1.5x10^{-3} saat, servo akan berputar menghampiri 180 darjah. Konsep ini akan diaplikasikan semasa membuat aturcara *PIC microcontroller.*



Rajah 2.17: Konsep Putaran Motor Servo



Rajah 2.18: Kabel Motor Servo

2.6 KAJIAN PORT KABEL SESIRI

RS232 banyak digunakan dalam penghantaran data dan pengantaramuka bagi port siri. Walaupun port siri agak sukar untuk diprogram berbanding port selari tetapi ia amat sesuai dan efektif dimana penghantaran data tidak memerlukan wayar yang banyak sekaligus menjimatkan kos. RS232 ialah penghubung komunikasi yang membenarkan penghantaran dan penerimaan data dengan hanya memerlukan tiga wayar penghubung. Ketiga-tiga wayar ini ialah wayar hantar, wayar terima dan wayar bumi.

Wayar hantar dan wayar terima akan menghantar dan menerima data diantara komputer dengan peralatan. Data akan dihantar secara siri. Pin untuk wayar terima dan hantar ialah RX dan TX. Selain itu, terdapat beberapa wayar lain di dalam port siri iaitu RTS, CTS,DSR,DTR dan RTS. Penerimaan data di dalam bentuk logik 1 dan 0 dimana ia diwakili oleh tahap voltan 3V sehingga 25V dan -3V sehingga -25V.

Jadual 2.3: Penjelasan Pin DB-9 Lelaki

Nombor Pin	Singkatan	Fungsi (untuk DB-9 Lelaki)
2	RX	Terima Data
3	TX	Hantar Data
7	RTS	Request To Send
8	CTS	Clear To Send
6	DSR	Data Ready State
5	SG	Isyarat Bumi
1	DCD	Data Carrier Detect
4	DTR	Data Terminal Ready.
9	RI	Ring Indicator

Jadual 2.4: Penjelasan Pin DB-9 Perempuan

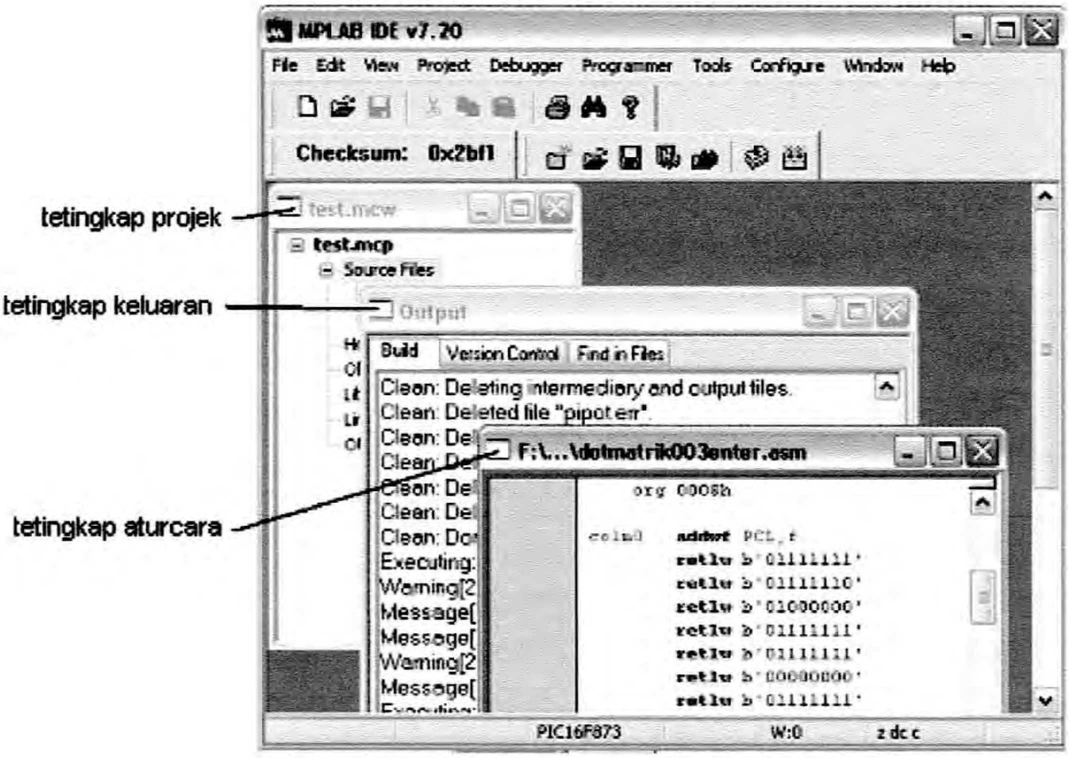
Nombor Pin	Singkatan	Fungsi (untuk DB-9 Perempuan)
3	RX	Terima Data
2	TX	Hantar Data
8	RTS	Request To Send
7	CTS	Clear To Send
6	DSR	Data Ready State
5	GND	Isyarat Bumi
1	DCD	Data Carrier Detect
4	DTR	Data Terminal Ready.
9	RI	Ring Indicator

Peralatan yang menggunakan kabel sesiri untuk komunikasi terbahagi kepada dua kategori iaitu DCE (*Data Communications Equipment*) dan DTE (*Data Terminal Equipment*). DCE ialah peralatan seperti modem manakala DTE ialah komputer utama.

Bahagian DTE		**Bahagian DCE**
3 Hantar Data	——→	**3** Terima Data
2 Terima Data	←——	**2** Hantar Data
7 Request To Send	——→	**8** Clear To Send
8 Clear To Send	←——	**7** Request To Send

2.5 PERISIAN UNTUK PIC16F84A

MPLAB IDE v7.20 digunakan untuk menulis aturcara *PIC microcontroller.* Aturcara ditulis di tetingkap aturcara.

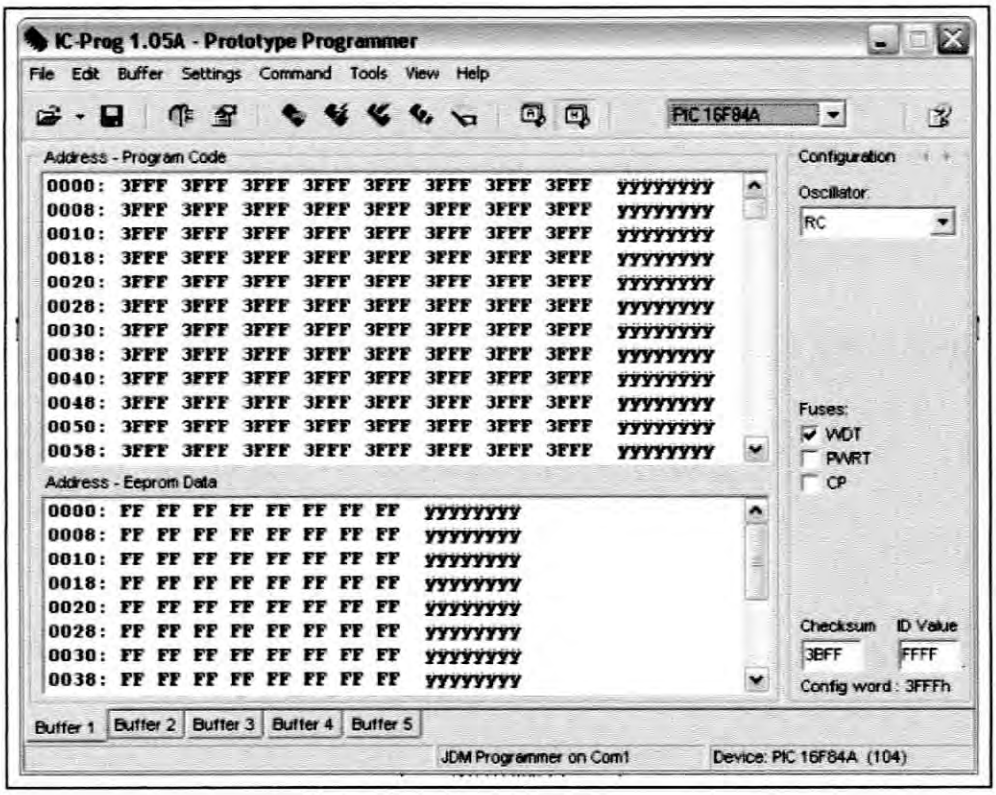


```
Clean: Deleting intermediary and output files.
Clean: Deleted file "pipot.err".
Clean: Del
Clean: De
Clean: Del
Clean: Del
Clean: Do
Executing:
Warning[2
Message[
Message[
Warning[2
Message[
```

```
        org 0008h

colum0  addwf PCL,f
        retlw b'01111111'
        retlw b'01111110'
        retlw b'01000000'
        retlw b'01111111'
        retlw b'01111111'
        retlw b'00000000'
        retlw b'01111111'
```

Rajah 2.19: Pengantaramuka Grafik MPLAB IDE v7.20

Rajah di atas menunjukkan pengantaramuka grafik pengguna bagi perisian MPLAB IDE v7.20. Penulisan aturcara hanya menggunakan sebanyak 35 set arahan. Setelah aturcara ditulis perisian IC-Prog akan digunakan untuk memuat turun aturcara ke dalam *PIC microcontroller.*


IC-Prog 1.05A - Prototype Programmer
File Edit Buffer Settings Command Tools View Help
PIC 16F84A
Address - Program Code
Address - Eeprom Data
Configuration
Buffer 1 Buffer 2 Buffer 3 Buffer 4 Buffer 5

```
0000: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0008: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0010: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0018: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0020: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0028: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0030: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0038: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0040: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0048: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0050: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
0058: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   yyyyyyyy
```

```
0000: FF FF FF FF FF FF FF FF   yyyyyyyy
0008: FF FF FF FF FF FF FF FF   yyyyyyyy
0010: FF FF FF FF FF FF FF FF   yyyyyyyy
0018: FF FF FF FF FF FF FF FF   yyyyyyyy
0020: FF FF FF FF FF FF FF FF   yyyyyyyy
0028: FF FF FF FF FF FF FF FF   yyyyyyyy
0030: FF FF FF FF FF FF FF FF   yyyyyyyy
0038: FF FF FF FF FF FF FF FF   yyyyyyyy
```

Oscillator: RC

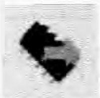
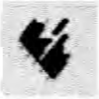
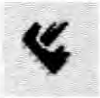
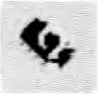
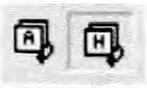
Fuses:
☑ WDT
□ PWRT
□ CP

Checksum: 3BFF
ID Value: FFFF
Config word : 3FFFh

JDM Programmer on Com1
Device: PIC 16F84A (104)

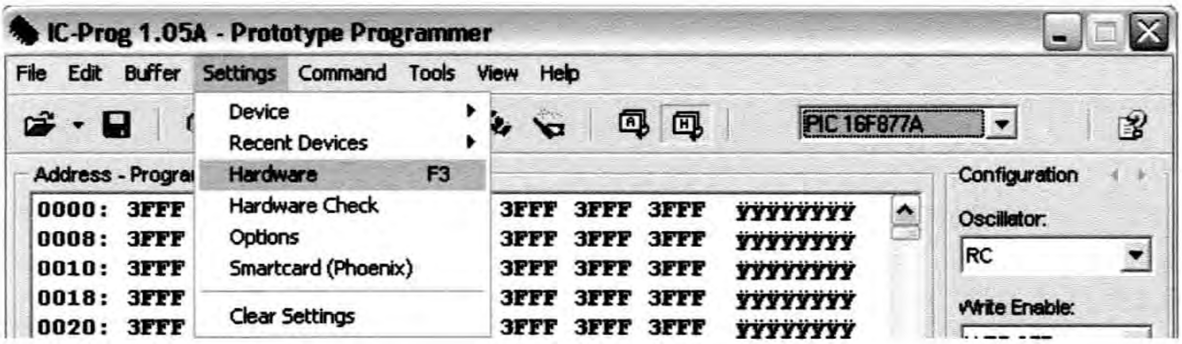
Rajah 2.20: Pengantaramuka Grafik IC-Prog versi 1.05A

Jadual 2.5: Ikon IC-Prog

Ikon	Penjelasan
	Digunakan untuk membaca aturcara yang sediaada dalam PIC
	Digunakan untuk memuat turun aturcara
	Digunakan untuk padam aturcara dalam PIC
	Digunakan untuk kenal pasti PIC
	Pilih untuk papar bahasa himpunan

IC-Prog digunakan untuk memuat turun aturcara yang ditulis menggunakan MPLAB IDE. Terdapat beberapa langkah yang perlu dilakukan untuk proses ini. Langkah-langkah tersebut diterangkan secara ringkas seperti dibawah:

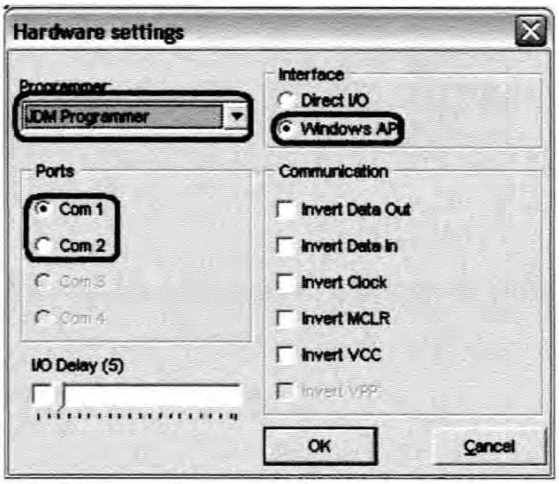
1) Langkah yang pertama. Klik: setting >> hardware.



```
0000: 3FFF ... 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0008: 3FFF ... 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0010: 3FFF ... 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0018: 3FFF ... 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0020: 3FFF ... 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
```

Rajah 2.21: Menu *Setting*

2) Pastikan semua pilihan diset seperti di bawah:


Hardware settings

Programmer: JDM Programmer

Interface: ○ Direct I/O ● Windows API

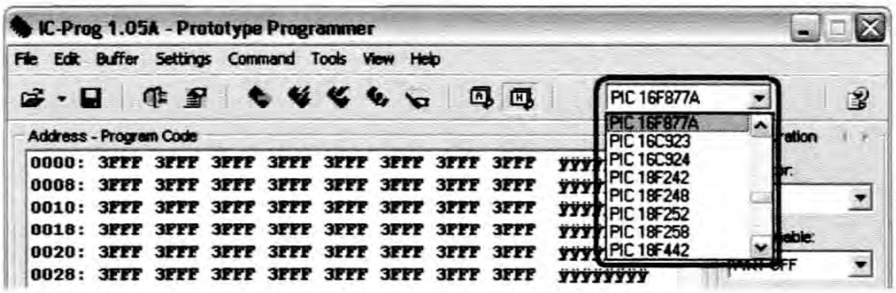
Ports: ● Com 1 ○ Com 2 ○ Com 3 ○ Com 4

Communication: □ Invert Data Out □ Invert Data In □ Invert Clock □ Invert MCLR □ Invert VCC □ Invert VPP

I/O Delay (5)

Rajah 2.22: Menu Set Peranti

3) Pilih jenis PIC yang digunakan.

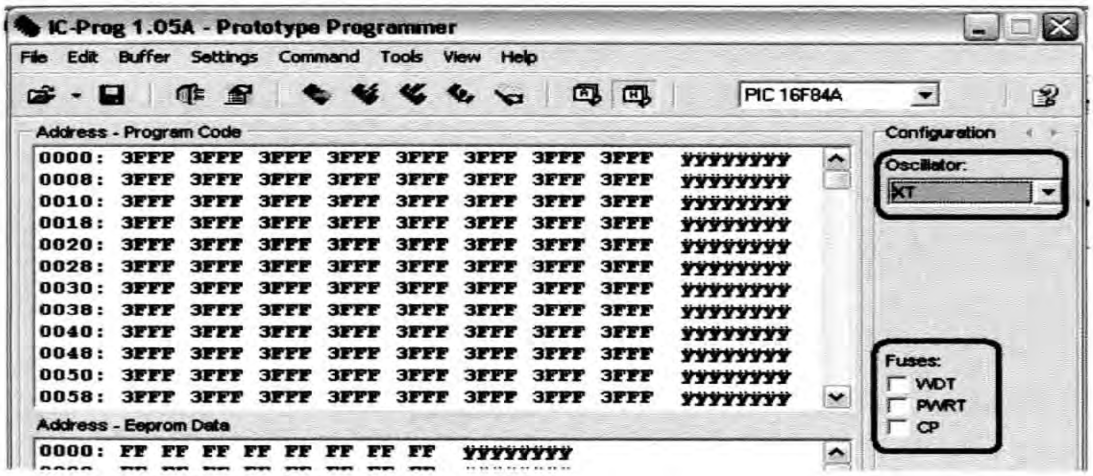


```
0000: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿ
0008: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿ
0010: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿ
0018: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿ
0020: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿ
0028: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
```

Rajah 2.23: Menu Pilih PIC

4) Buka aturcara yang hendak dimuat turun. Aturcara adalah dalam bentuk fail *hex.*

5) Untuk PIC16F84A pilih pengayun kristal dan padam semua fius.


IC-Prog 1.05A - Prototype Programmer
File Edit Buffer Settings Command Tools View Help
PIC 16F84A
Address - Program Code
Configuration
Address - Eeprom Data

```
0000: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0008: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0010: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0018: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0020: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0028: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0030: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0038: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0040: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0048: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0050: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ
0058: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF   ÿÿÿÿÿÿÿÿ

0000: FF FF FF FF FF FF FF FF   ÿÿÿÿÿÿÿÿ
```

Oscillator: XT

Fuses:
□ WDT
□ PWRT
□ CP

Rajah 2.24: Set Opsyen Pengayun dan Fius

6) Sambungkan *PIC burner* pada komputer.

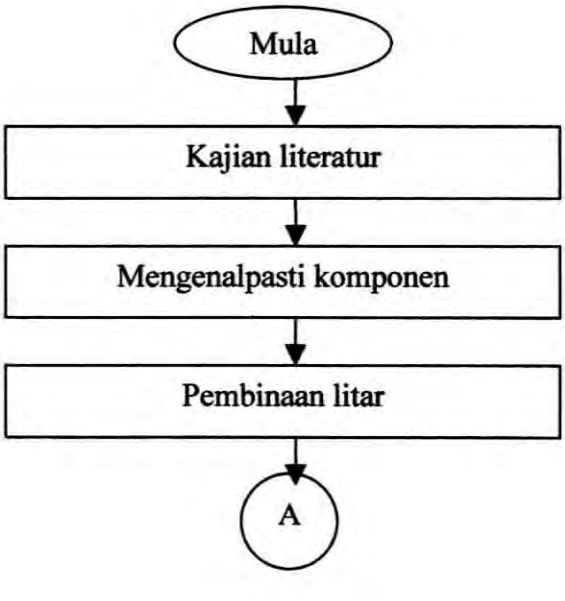
7) Langkah yang terakhir klik ikon program.

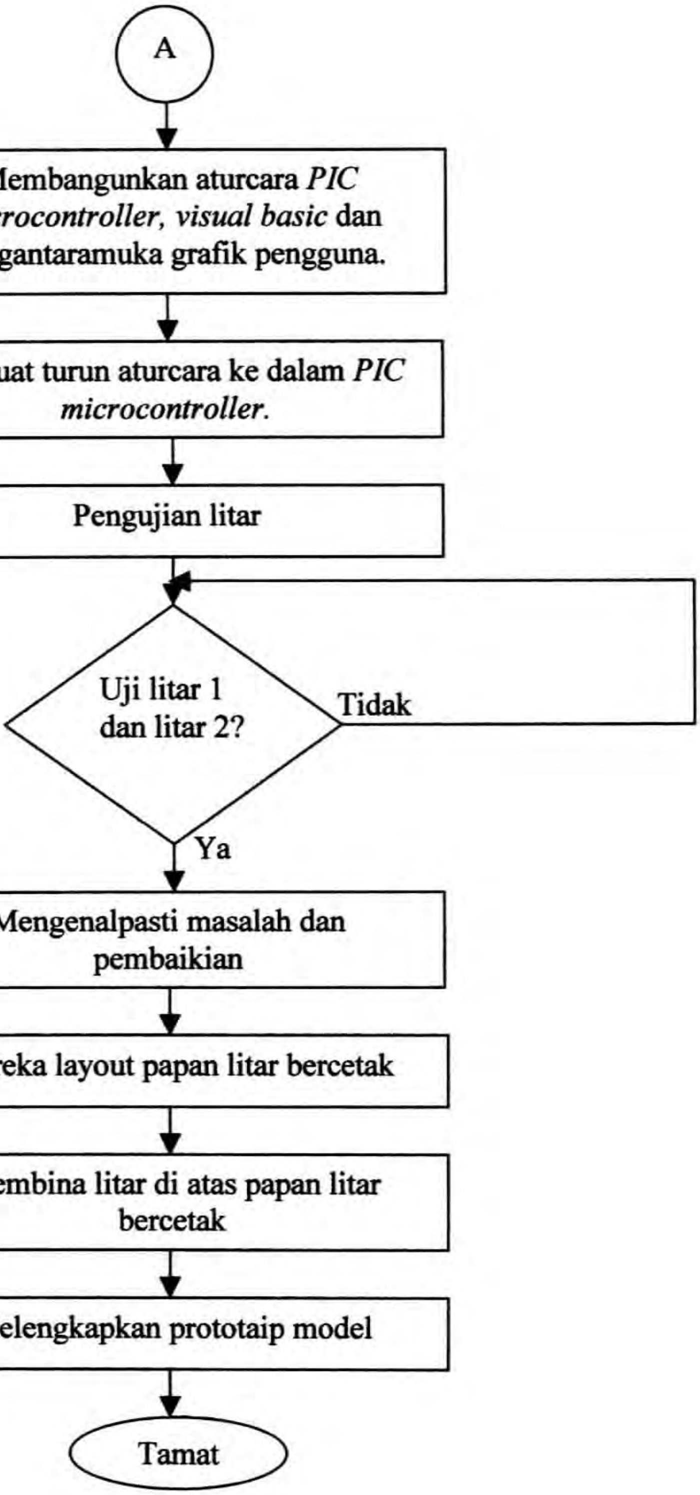
BAB 3

METADOLOGI PROJEK

Merupakan bahagian penerangan prosedur dan kaedah yang akan digunakan untuk melaksanakan projek. Menerangkan langkah demi langkah proses pembinaan model projek sehingga selesai.

3.1 CARTA ALIR METADOLOGI PROJEK




A
Membangunkan aturcara *PIC microcontroller, visual basic* dan pengantaramuka grafik pengguna.
Memuat turun aturcara ke dalam *PIC microcontroller.*
Pengujian litar
Uji litar 1 dan litar 2?
Tidak
Ya
Mengenalpasti masalah dan pembaikian
Mereka layout papan litar bercetak
Membina litar di atas papan litar bercetak
Melengkapkan prototaip model
Tamat

3.2 PENERANGAN METADOLOGI PROJEK

a) Kajian Latar Belakang

1. Mengumpulkan maklumat berkaitan dengan PIC16F84A. Ini termasuklah senibina *PIC microcontroller*, litar *PIC burner* untuk memuat turun aturcara ke dalam *PIC microcontroller*, perisian yang digunakan untuk membangunkan aturcara *PIC microcontroller* seperti *MPLAB IDE* dan perisian yang akan menukar aturcara tersebut ke dalam bentuk fail penomboran enam belas *(hex file)* seperti *IC-Prog 1.05A*.
2. Mengkaji kaedah penggunaan perisian Visual Basic 6.0 untuk membina pengantaramuka grafik pengguna. Ini termasuklah aturcara *servo commander* yang akan menjadikan pengantaramuka berfungsi.
3. Menggumpul maklumat berkaitan motor servo. Ini termasuklah spesifikasi motor servo tersebut, konsep kawalan motor servo menggunakan *PIC microcontroller*, perkakasan tambahan motor servo dan jenis motor servo yang bersesuaian.
4. Menggumpul maklumat berkaitan litar yang menggunakan *PIC microcontroller* untuk mengawal motor servo.
5. Mengkaji kaedah penyambungan kabel serial.

b) Pembinaan Litar

1. Litar-litar untuk penghasilan model projek ini seperti litar *PIC microcontroller 1*, litar *PIC microcontroller 2* dan litar bekalan kuasa dibina di atas *veraboard*.
2. Litar-litar ini akan diuji kefungsiannya setelah ia siap dipasang.
3. Bagi litar bekalan kuasa, rekabentuk dan pengujian dilakukan menggunakan perisian *EWB Multisim*.

c) Membangunkan Aturcara *PIC Microcontroller*, *Visual Basic* dan Pengantaramuka Grafik Pengguna.

1. Terdapat dua aturcara yang akan dibangunkan iaitu aturcara *PIC microcontroller* dan aturcara *visual basic*. Aturcara ini masing-masing menggunakan perisian *MPLAB* dan *Visual Basic 6.0*
2. Pengantaramuka grafik pengguna juga direka semasa proses ini menggunakan perisian *Visual Basic 6.0.*

d) Mengabungkan Litar *PIC Microcontroller* dengan Aturcara.

1. Aturcara yang telah siap dibina akan digabungkan dengan litar dan diuji kefungsiannya.
2. Aturcara *PIC microcontroller* akan dimuat turun menggunakan *PIC burner.*

e) Pengujian Litar

1. Setelah litar *PIC microcontroller 1* dan litar *PIC microcontroller 2* dibina diatas veraboard dan digabungkan dengan aturcaranya, ianya akan pula diuji kefungsian.
2. Kefungsian diuji dengan kaedah memastikan litar *PIC microcontroller 1* dan litar *PIC microcontroller 2* menerima arahan dari pengantaramuka grafik pengguna dan seterusnya menggerakkan motor servo berdasarkan arahan tersebut.
3. Sekiranya terdapat masalah pada mana-mana litar, ia akan terus diperbaiki dan diuji semula.

f) Mereka Layout Papan Litar Bercetak

1. Litar-litar yang telah diuji dan dipastikan berfungsi akan dipilih untuk bina di atas papan litar bercetak.
2. Litar yang telah dikenalpasti untuk dibina diatas papan litar bercetak akan direka menggunakan perisian *proteus ISIS 6 Profesional* dan *proteus ARES 6 Profesional.*
3. Kaedah ini dilakukan dengan teliti untuk mengelakkan kesilapan yang akan menyebabkan kesukaran apabila memasukkan komponen.

g) Membina Litar Papan Litar Bercetak

1. Layout papan litar bercetak yang direka dicetak pada *PCB board*, dan lubang pin komponen ditebuk.
2. Komponen akan dipasangkan ditempat masing-masing dan dipateri kakinya.

h) Melengkapkan Prototaip.

1. Tumpuan diberikan kepada pembangunan bahagian mekanikal seperti merekabentuk model bingkai dan tirai, memasang motor pada bingkai dan memasang litar pada kotak (*casing*).

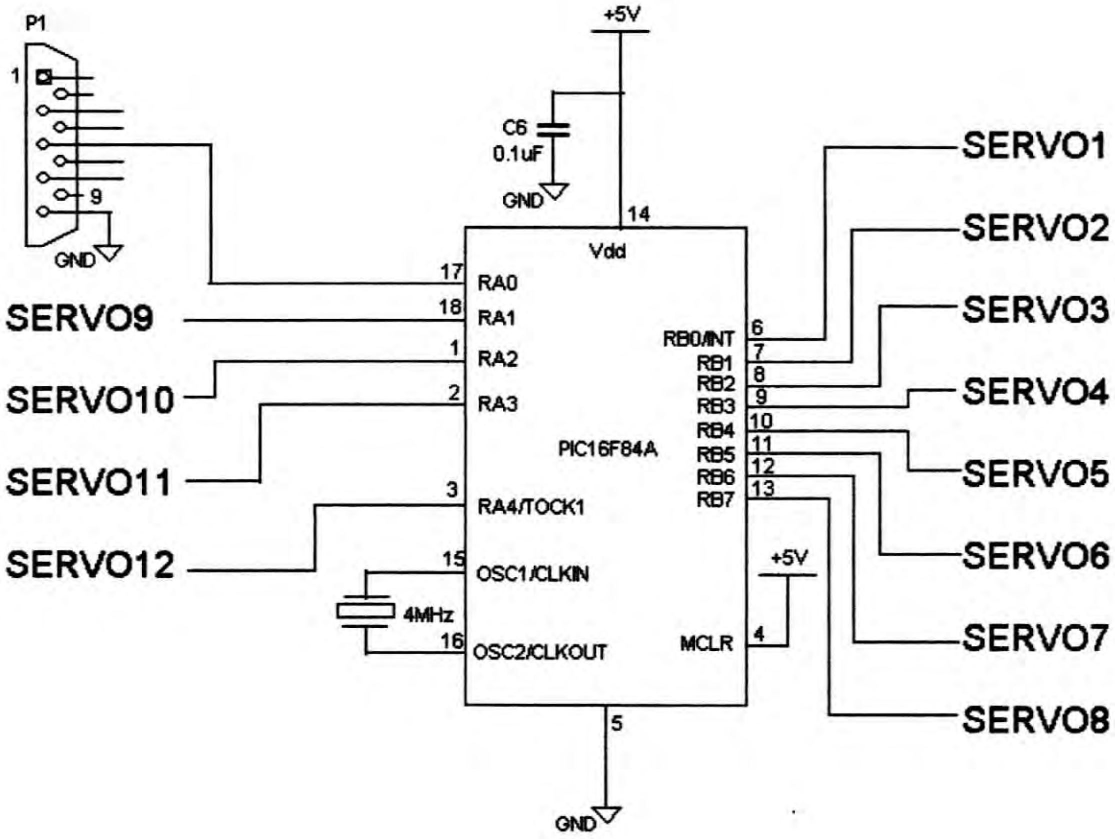
BAB 4

KEPUTUSAN DAN ANALISIS

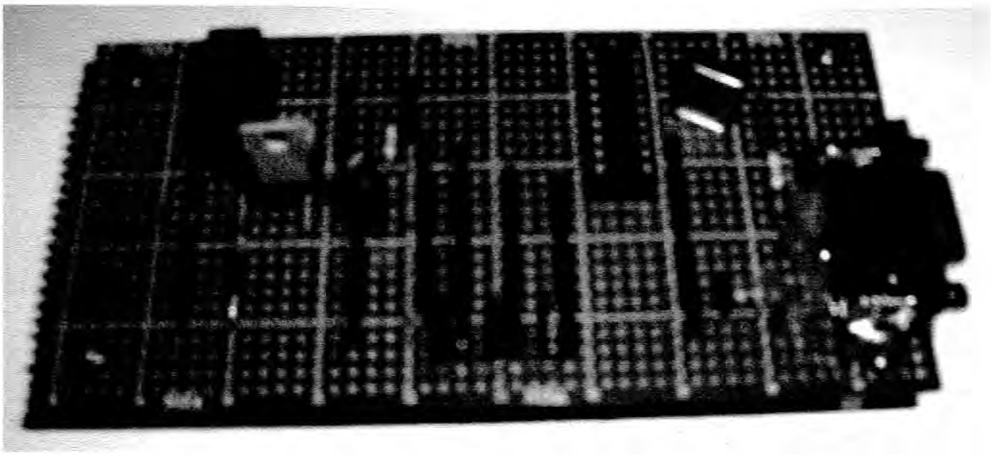
Di dalam bab ini, hasil keputusan dan ujikaji litar dinyata dan dibincangkan. Terdapat dua litar yang telah dibangunkan untuk ujikaji. Litar ujikaji 1 merupakan litar pengawal motor servo yang ringkas manakala litar ujikaji 2 merupakan litar pengawal motor servo yang lebih komplek. Daripada pemerhatian ujikaji, litar 2 telah dipilih untuk dibangunkan di atas papan litar bercetak. Bab ini juga akan membincangkan langkah-langkah pengubahsuaian yang telah dilakukan ke atas motor servo kerana motor servo yang asal hanya dapat berputar sebanyak 60 darjah sahaja. Selain itu, bab ini juga membincangkan aturcara-aturcara yang telah dibangunkan seperti aturcara visual basic dan aturcara *PIC microcontroller*.

4.1 ANALISIS LITAR PENGAWAL MOTOR SERVO – LITAR 1

Litar ujikaji 1 ini adalah litar pengawal motor servo yang ringkas. Ia hanya memerlukan komponen seperti port DB-9 Perempuan, dua buah perintang bernilai $22x10^3$ ohm dan $4.7x10^3$ ohm, sebuah pengayun kristal bernilai $4x10^6$ Hertz dan sebuah kapasitor bernilai $0.1x10^{-9}$ Farad. Pin nombor 5 DB-9 perempuan akan disambungkan pada bumi manakala pin nombor 3 akan disambungkan pada pin RA0 PIC16F84A. Pin no 3 ini berfungsi untuk menghantar data dari komputer utama ke PIC. Pin DB-9 perempuan yang selebihnya tidah disambungkan. Litar ini mempunyai kelemahan dimana komputer utama tidak dapat menerima data dari litar PIC. Untuk litar ini, hanya PIC sahaja dapat menerima data atau arahan pengguna dari komputer. Komputer tidak akan mendapat maklum balas dari litar PIC. Litar ini mampu menampung keluaran sebanyak dua belas buah motor servo.



Rajah 4.1: Litar Skematik Pengawal Motor Servo – Litar 1



Rajah 4.2: Litar Pengawal Motor Servo – Litar 1

4.2 ANALISIS LITAR PENGAWAL MOTOR SERVO – LITAR 2

Litar ujikaji 2 adalah litar pengawal motor servo yang lebih komplek. Ia merupakan gabungan beberapa litar seperti litar komunikasi sesiri, litar penjana jam, litar isyarat masukkan dan litar keluaran PIC16F84A. Litar ini hanya boleh menampung keluaran sebanyak 8 buah motor servo sahaja. Kelebihan litar ini ialah litar ini boleh menghantar maklum balas kepada komputer utama. Litar ini telah dipilih untuk dibangunkan di atas papan litar bercetak.

4.2.1 Kendalian Litar Pengawal Motor Servo – Litar 2

Dengan merujuk rajah 4.3, litar komunikasi sesiri ini dihubungkan dengan komputer utama bermula di P1, penghubung BD9 perempuan (*female DB9 connector).* Pin 1,4 dan 6 disambungkan bersama untuk memastikan komputer utama menerima data sesiri dari *PIC microcontroller* pada bila-bila masa. Pin 5 disambungkan ke litar bumi manakala pin yang selebihnya disambungkan ke MAX232.

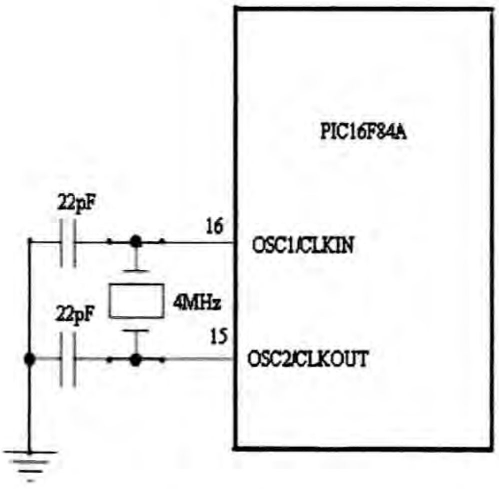
Chip MAX232 digunakan untuk menukar isyarat voltan dari RS232 kepada isyarat logic CMOS yang boleh digunakan oleh *PIC microcontroller*. Ia mempunyai dua penerima yang menukar isyarat RS232 kepada isyarat logik CMOS dan dua pemancar yang menukar isyarat logik CMOS ke isyarat voltan RS232. Kapasitor C1, C2, C3, C4, dan C5 digunakan untuk menjana voltan +10 volt dan – 10 volt.

Garis RX dari port sesiri disambungkan kepada keluaran pemancar pertama (T1Out) dimana ia dikawal oleh isyarat *Serial Out* dari *PIC microcontroller* pin RA0. Garis TX disambungkan kepada masukkan penerima kedua (R2In) dimana ia mengawal isyarat *Serial In* yang disambungkan ke *PIC microcontroller* pin RA3. Garis CTS dari port sesiri disambungkan kepada keluaran pemancar kedua (T2Out) dimana ia dikawal oleh isyarat *Clear To Send* dari *PIC microcontroller* pin RA2. Manakala garis RTS disambungkan kepada masukkan penerima pertama (R1In) dimana ia mengawal isyarat *Request To Send* yang disambungkan ke *PIC microcontroller* pin RA1.

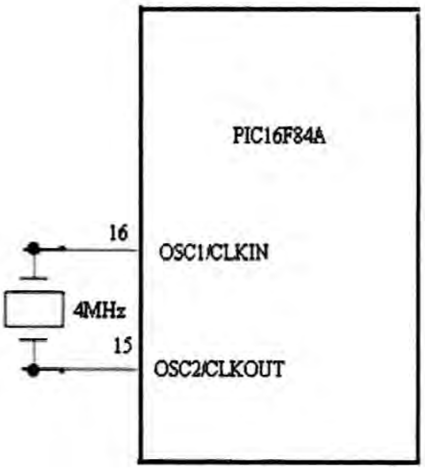
PIC16F84A ini menggunakan pengayun kristal 4X10^6 hertz. Isyarat jam ini dibahagikan kepada 4 bahagian untuk mengawal jam arahan *PIC microcontroller* bagi membolehkan *PIC microcontroller* melaksanakan arahan dalam masa 1x10^{-6} saat untuk setiap arahan. Manakala semua pin RB bermula dari RB0 ke RB7 digunakan sebagai keluaran *PIC microcontroller* untuk mengawal motor servo. Kesemua pin ini akan disambungkan ke pin isyarat masukkan pada motor servo.

LED D1 disambungkan pada pin RA4 dan pada bekalan kuasa 5 volt melalui perintang R1. Perintang R1 bertindak sebagai perintang penghad arus. Aturcara PIC akan menyalakan LED setiap kali bit data diterima dari komputer utama. Ia akan menyala semasa RA4 dalam keadaan aktif rendah.

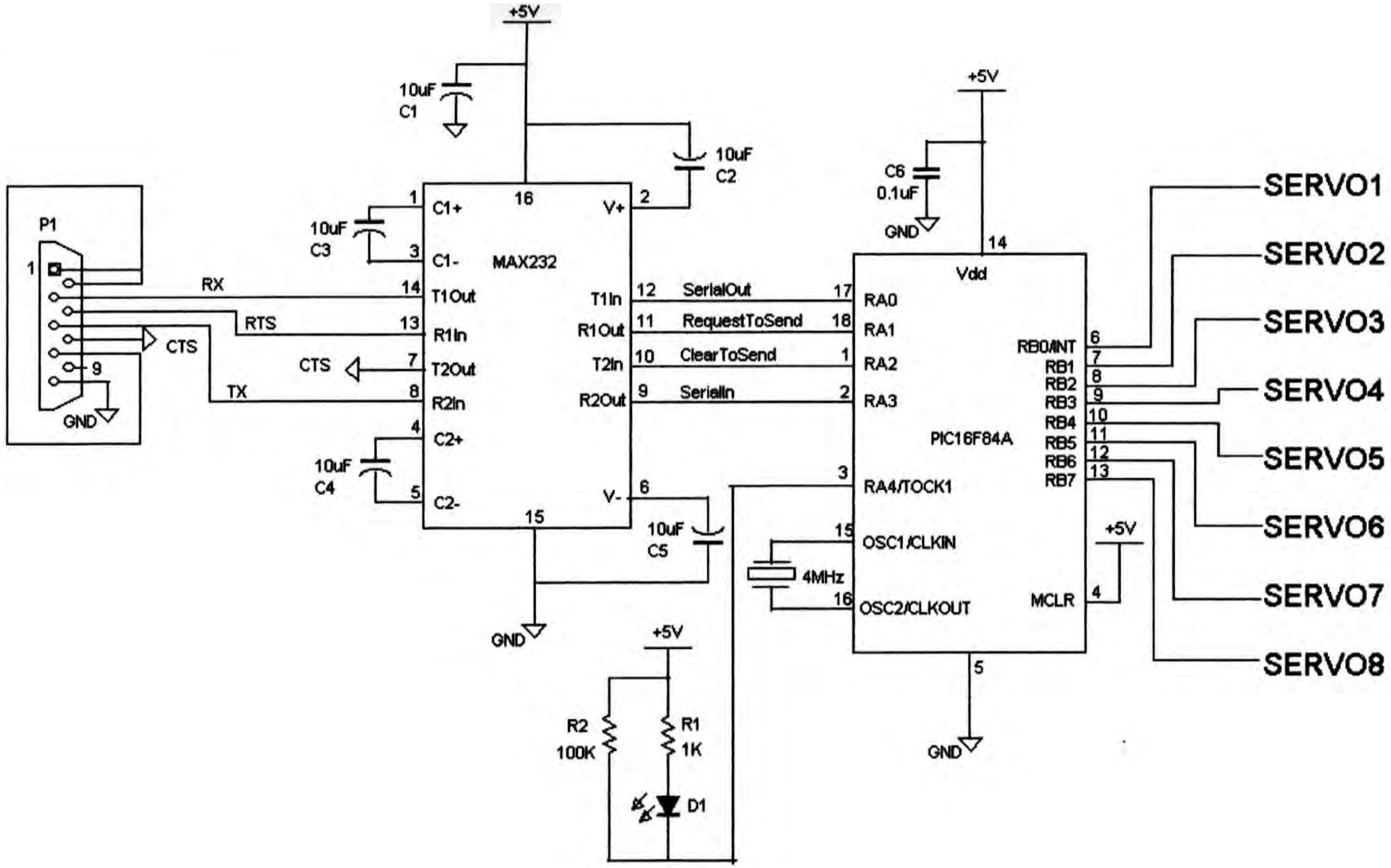
Untuk litar ini terdapat pengubahsuaian dilakukan, iaitu dua kapasitor seramik di tanggalkan dari litar penjana jam yang sebenar. Setelah kapasitor ini ditanggalkan, litar didapati dapat berfungsi mengikut arahan pengantaramuka grafik pengguna.



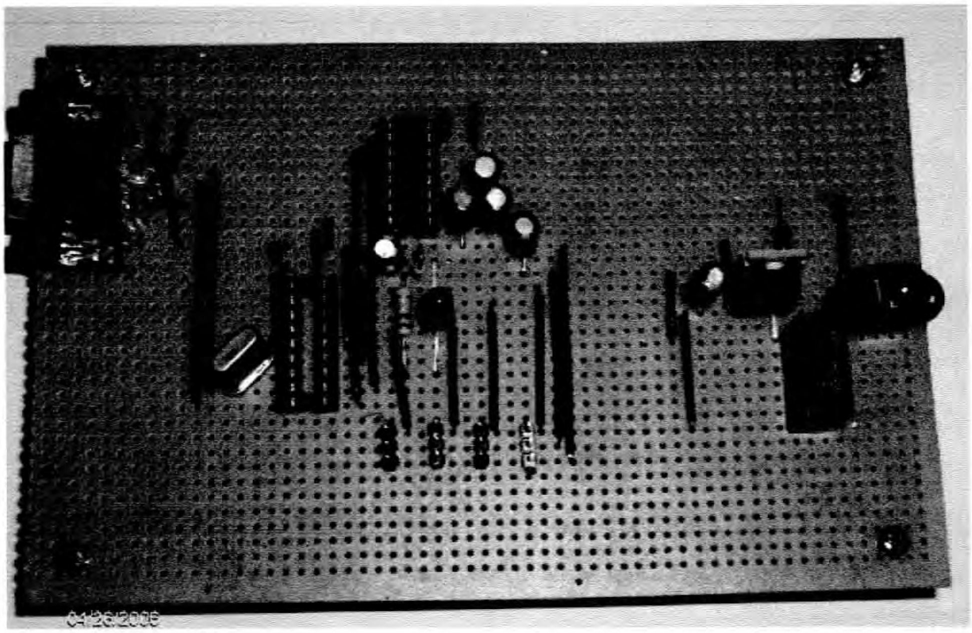
Rajah 4.3: Litar Penjana Jam yang asal



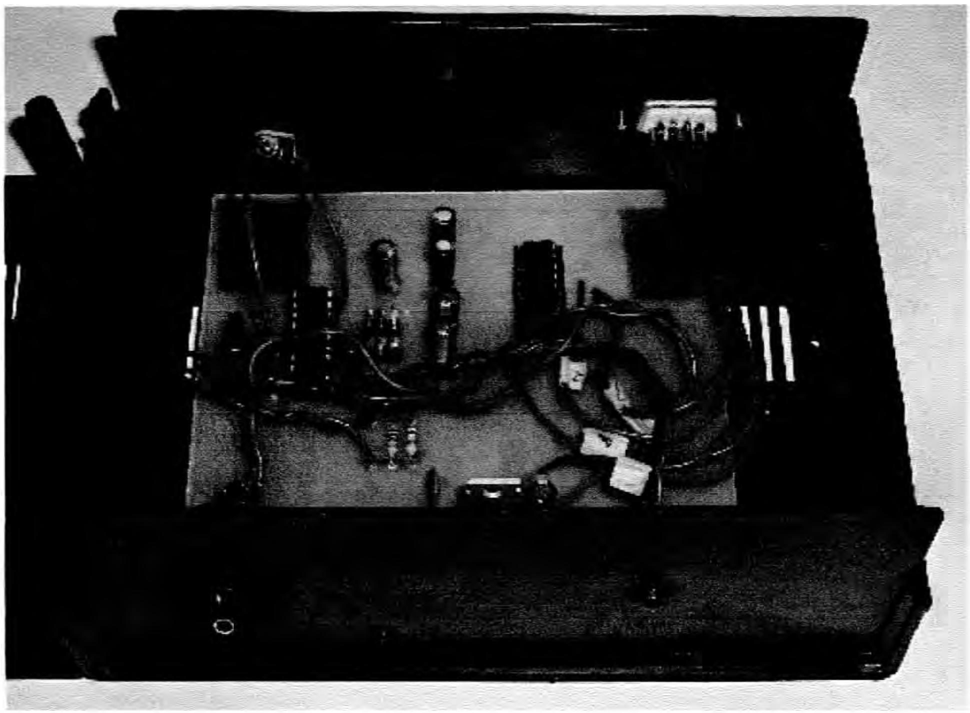
Rajah 4.4: Litar Penjana Jam yang baru



Rajah 4.5: Litar Skematik Pengawal Motor Servo – Litar 2



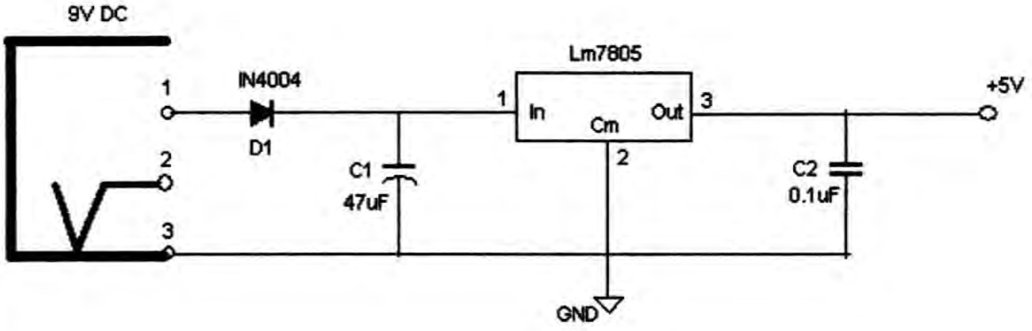
Rajah 4.6: Litar Pengawal Motor Servo – Litar 2



Rajah 4.7:Papan Litar Bercetak – Litar 2

4.3 ANALISIS LITAR BEKALAN KUASA

Litar bersepadu alat atur (*IC Regulator*) digunakan untuk mendapatkan keluaran 5 volt yang tepat. Chip yang digunakan ialah chip Lm7805. Diod D1 menawarkan kekutuban terbalik untuk keselamatan litar. Kapasitor C2 digunakan untuk menstabilkan pengayun manakala kapasitor C1 digunakan untuk menapis kuasa dari bekalan 9 volt



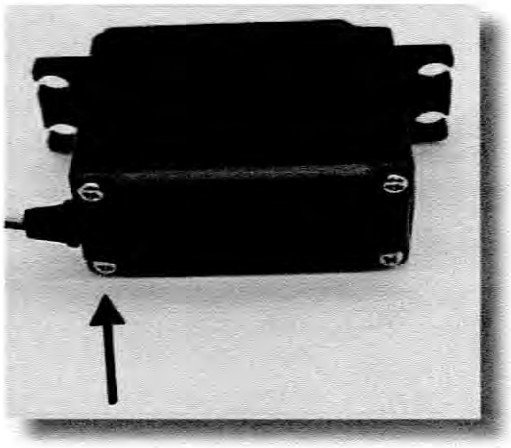
Rajah 4.8: Litar Skematik Bekalan Kuasa

4.4 PENGUBAHSUAIAN MOTOR SERVO

Pengubahsuaian terhadap motor servo perlu dilakukan memandangkan servo yang asal hanya boleh berputar sebanyak 60 darjah ke kanan dan 60 darjah ke kiri. Pengubahsuaian ini perlu dilakukan untuk menjadikan servo boleh berputar sebanyak 360 darjah. Sebelum pengubahsuaian dilakukan, beberapa peralatan diperlukan seperti pemutar skru mata philips, pemutar skru mata rata, playar muncung tirus dan playar pemotong.

Langkah 1

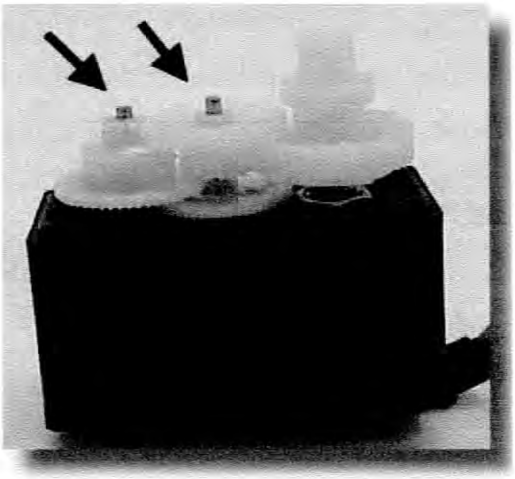
Tanggalkan skru yang terdapat di belakang penutup motor servo menggunakan pemutar skru mata philips.



Rajah 4.9: Skru Belakang Motor Servo

Langkah 2

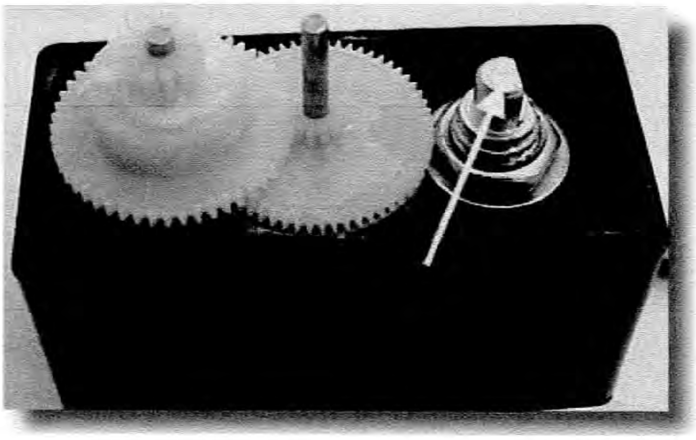
Setelah skru ditanggalkan, buka penutup yang menutup motor servo. Pada masa ini, gear dalam dapat dilihat dengan jelas.



Rajah 4.10: Gear Dalaman Motor Servo

Langkah 3

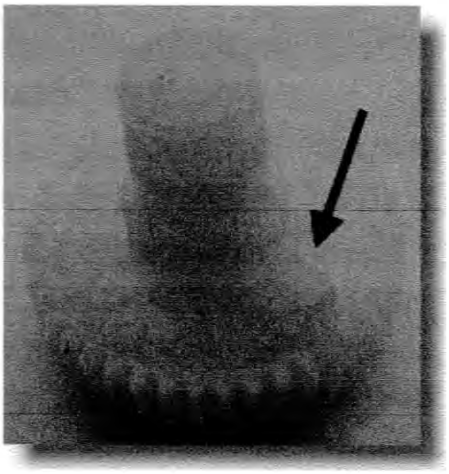
Tombol pada potentiometer seperti di tunjukkan pada rajah dibawah hendaklah dipotong. Pemotongan hendaklah dilakukan secara berhati-hati dan kemas. Ini adalah untuk mengelakkan kerosakkan pada potentiometer. Tombol ini dibuang kerana potentiometer yang melimitkan putaran servo.



Rajah 4.11: Tombol Potentiometer

Langkah 5

Gear seperti rajah di bawah hendaklah di potong pada bahagian yang ditunjukkan oleh anak panah. Bahagian ini juga akan melimitkan putaran servo.

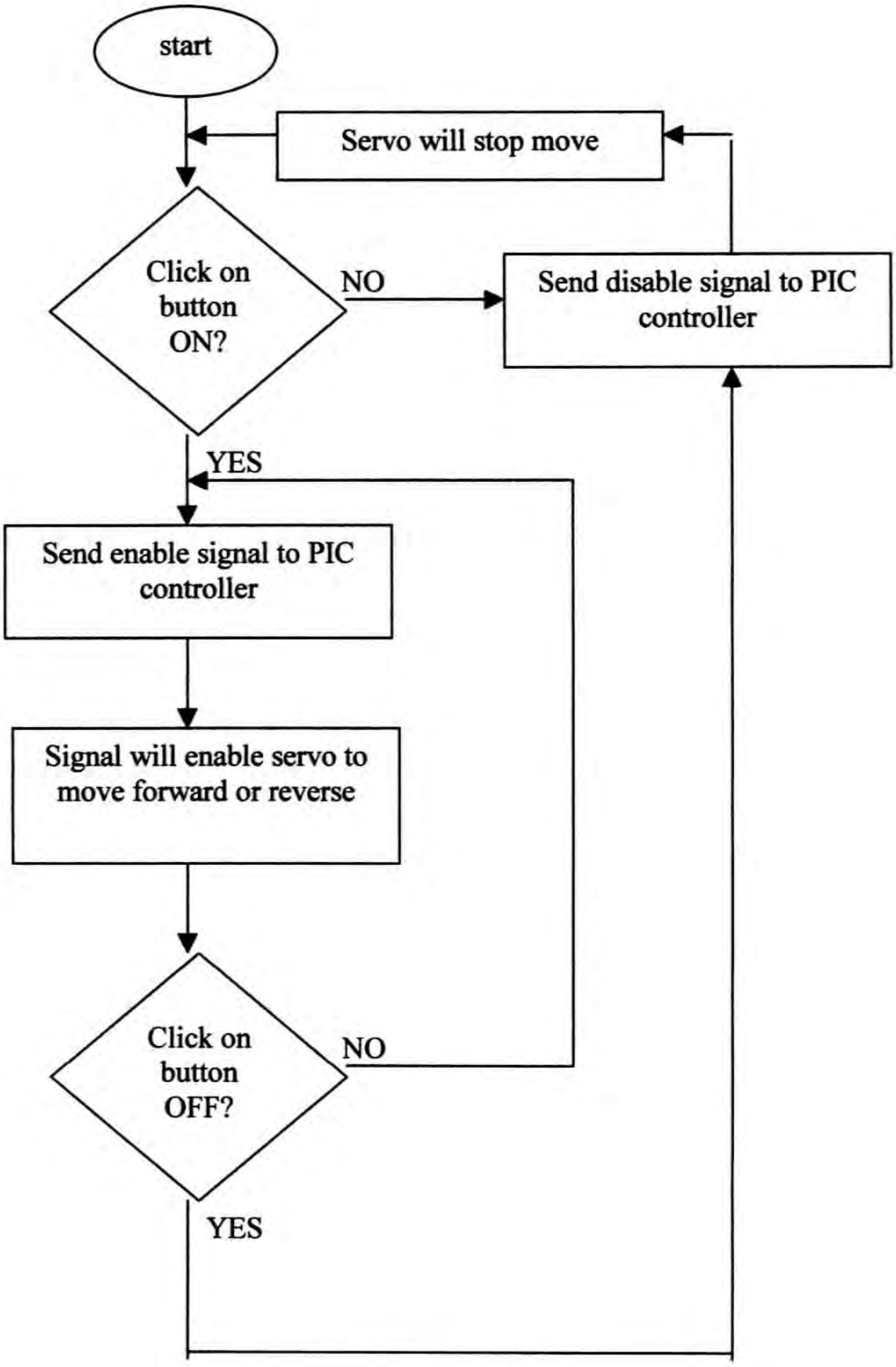


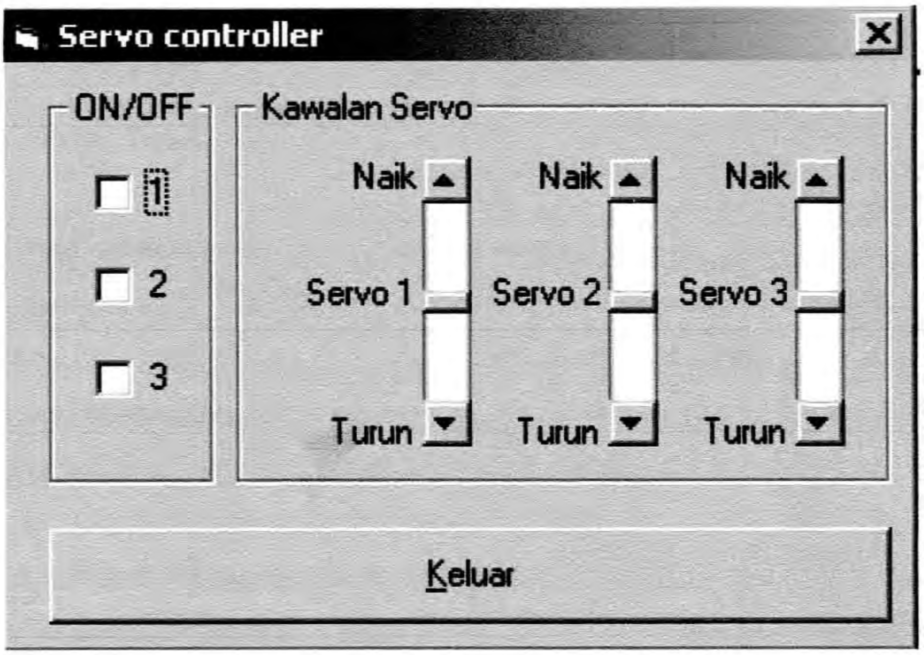
Rajah 4.12: Gear Limitasi Servo

Langkah 6

Setelah semua perkara diatas dilakukan, pasang semula semua komponen motor servo. Ini harus dilakukan secara berhati.

4.5 ANALISIS PEMBANGUNAN PENGANTARAMUKA PENGGUNA DAN ATURCARA VISUAL BASIC


start
Servo will stop move
Click on button ON?
NO
Send disable signal to PIC controller
YES
Send enable signal to PIC controller
Signal will enable servo to move forward or reverse
Click on button OFF?
NO
YES



Rajah 4.13: Pengantaramuka Grafik Pengguna

4.5.1 Aturcara Visual Basic

```
*************************************************************************
Private Sub chkServoAct_Click(Index As Integer)
If chkServoAct(Index).Value = 1 Then          ;button ON activate
comSerialOut.Output = Chr$(48 + Index)        ;enable servo output
End If
If chkServoAct(Index).Value = 0 Then          ;button OFF activate
comSerialOut.Output = Chr$(64 + Index)        ;disable servo output
End If
End Sub

Private Sub Command1_Click()
End                                           ; out from GUI
End Sub
```

```
Private Sub Form_Load()
Let cnt = 0
comSerialOut.CommPort = 1            ;select com 1
comSerialOut.Settings = "2400,N,8,1"  ;set baud rate, 8 bits data, 1 stop bit, no parity
comSerialOut.PortOpen = True         ; com 1 ready to send signal
comSerialOut.Handshaking = 2
End Sub

Private Sub scrServo_Change(Index As Integer)
comSerialOut.Output = Chr$(16 + Index)                ;send position to PIC
comSerialOut.Output = Chr$(scrServo(Index).Value)
End Sub

Private Sub scrServo_Scroll(Index As Integer)
comSerialOut.Output = Chr$(16 + Index)                ;setting output position
                                                      ; value to move forward
                                                      ; or reverse
comSerialOut.Output = Chr$(scrServo(Index).Value)
End Sub
***********************************************************************
```

Jadual 4.1: Penerangan Nilai Decimal

Decimal	*Description*
16	*Set the servo output position value for the corresponding servo*
48	*Enable servo output. Start generating the servo control pulse for the servo.*
64	*Disable servo output. Stop generating the servo control pulse for the servo.*

4.6 METADOLOGI PEMBANGUNAN ATURCARA PIC

Program yang dibangunkan ini menyambungkan 4 buah motor servo pada port b PIC. Motor servo berkerja dengan bertindak balas terhadap gelombang yang dikenakan setiap $20x10^{-3}$s. Lebar gelombang yang dikenakan mempengaruhi arah putaran motor servo.

Secara umumnya lebar gelombang sekitar $1500x10^{-6}$s akan menyebabkan servo berada pada keadaan 90 darjah. Manakala lebar gelombang yang kurang daripada $1500x10^{-6}$s akan menyebabkan servo berputar ke arah 0 darjah. Lebar gelombang yang lebih daripada itu akan menyebabkan servo berputar ke arah 180 darjah.

Jadi masa perlu diatur untuk kawal lebar keluaran gelombang. Untuk melakukan ini, PIC akan dibekalkan dengan pengayun kristal berfrekuensi $4x10^{6}$ hertz. Ini bermakna satu arahan aturcara PIC akan menggunakan $1x10^{-6}$s.

Konfigurasi PORT B:

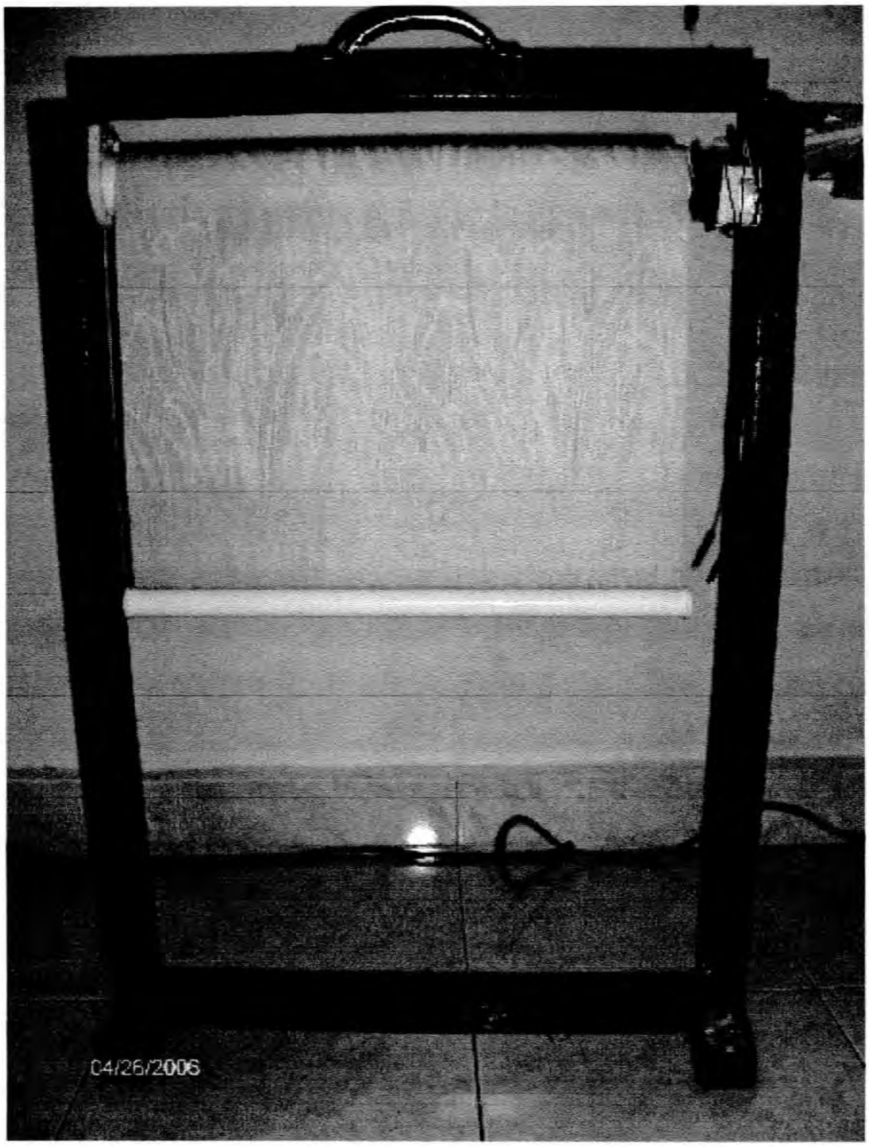
RB0	Servo1
RB1	Servo2
RB2	Servo3
RB3	Servo4

Konfigurasi PORT A:

RA0	*Serial TX*
RA1	*Serial Request To Send*
RA2	*Serial Clear To Send*
RA4	LED (menyala = logik 0, padam = logik 1)

Untuk mengawal motor servo, arahan perlu dihantar ke PIC. Arahan ini dihantar oleh port sesiri pada 2400 baud rate, lapan bit data, satu bit berhenti dan tiada persamaan (*no parity*). Lapan bit ini akan dibahagi kepada 2 bahagian. Setiap satu bahagian ada 4 bit logik. Bahagian pertama akan menentukan pemilihan saluran atau pemilihan servo motor yang hendak diberi arahan dan bahagian kedua mewakili data arahan servo.

4.7 GAMBAR HASIL PROJEK



Rajah 4.14: Gambar Model Tirai Tingkap

BAB 5

KESIMPULAN DAN CADANGAN

5.1 KESIMPULAN

Daripada projek sarjana muda ini, sebuah model tirai tingkap yang dikawal oleh komputer telah berjaya dihasilkan dan berfungsi sepenuhnya. Tirai dapat dinaik dan diturunkan berdasarkan arahan yang diberi oleh pengguna melalui pengantaramuka grafik pengguna pada komputer. Untuk menghasilkan model prototaip ini pelbagai perkara perlu dipelajari. Antaranya ialah mempelajari konsep kawalan motor servo, mempelajari lebih mendalam mengenai *PIC microcontroller*, mempelajari cara menggunakan perisian *Visual Basic*, mempelajari cara menggunakan *perisian IC-Prog 1.05A* dan *MPLab IDE*. Disamping itu, terdapat dua litar pengawal motor servo yang dibangunkan untuk ujikaji tetapi litar 2 lebih diberi penekanan kerana lebih fleksibel dan lebih berdaya maju. Seperti dalam bab 4, litar 2 lebih banyak dibincangkan berbanding litar 1. Manakala dari aspek motor servo, motor servo perlu diubahsuai memandangkan servo yang asal hanya boleh berputar sebanyak 60 darjah sahaja. Pengubahsuaian ini menjadikan motor servo boleh berputar sebanyak 360 darjah iaitu satu bulatan penuh.

5.2 CADANGAN

Cadangan dari aspek aplikasi projek:

1. Digunakan dalam pertanian. Seperti kita ketahui, anak benih pokok memerlukan kawalan pencahayaan untuk hidup. Oleh kerana itu, projek boleh digunakan untuk mengawal kadar pengcahayaan bilik atau kawasan semaian anak benih.
2. Model juga boleh dijadikan salah satu elemen penghasilan rumah pintar.
3. Bagi pejabat yang mempunyai banyak tingkap, ia akan memudahkan pengawalan kedudukan tirai dengan hanya menggunakan sebuah komputer. Pengguna tidak perlu ke setiap tingkap untuk membuka atau menutup tirai.
4. Selain itu, ia juga dapat menyelesaikan masalah mengubah kedudukan tirai tingkap bagi bingkai tingkap yang sukar dicapai.
5. Menyelesaikan masalah penggunaan alat kawalan jauh sekiranya sesebuah rumah itu mempunyai bilik yang banyak. Pengguna tidak perlu kesetiap bilik. untuk mengubah kedudukan tirai.
6. Pergerakkan motor servo tidak semestinya diaplikasikan keatas tirai tingkap tetapi boleh juga diaplikasikan terhadap aplikasi lain seperti pengawal tangan robotik (*robotic arm controller*).

Cadangan dari aspek memajukan projek:

1. Menggunakan *PIC microcontroller* yang lebih besar ingatan dan lebih banyak keluaran seperti PIC16F877A.

2. Menggunakan motor servo yang mempunyai kuasa kilasan yang lebih tinggi seperti motor servo HS 255MG dari HITEC.

3. Menghasilkan pengantaramuka grafik pengguna yang lebih mesra.

RUJUKAN

[1] Marc E. Herniter. *Shematic Capture with Electronic Workbench Multisim.* Pearson Prentice Hall; 2004

[2] Charles K. Alexender. *Fundamental of Electronic Circuit 2nd edition.* Prentice Hall; 2002

[3] Joseph Julicher, 14-04-2003, Hardware Techniques for PIC Microcontroller (journal)

[4] Rod Stephens, *prototyping with Visual Basic,* QUE, Indianapolis, Indiana; 2001

[5] John Iovine, *PIC Microcontroller Project Book,* 2nd Edition, McGraw-Hill Companies; 2004

LAMPIRAN A

DATSHEET PIC16F84A

PIC16F8X

18-pin Flash/EEPROM 8-Bit Microcontrollers

Devices Included in this Data Sheet:

- PIC16F83
- PIC16F84
- PIC16CR83
- PIC16CR84
- Extended voltage range devices available (PIC16**LF**8X, PIC16**LCR**8X)

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single cycle except for program branches which are two-cycle
- Operating speed: DC - 10 MHz clock input
  DC - 400 ns instruction cycle

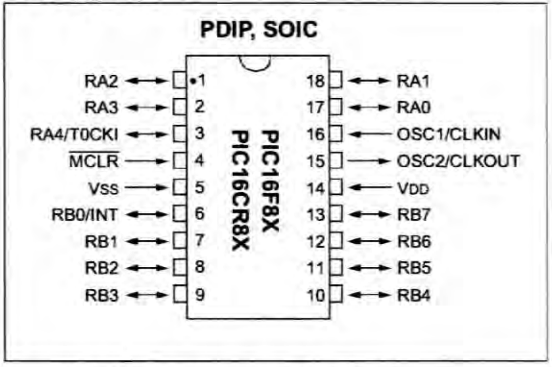
Device	Program Memory (words)	Data RAM (bytes)	Data EEPROM (bytes)	Max. Freq (MHz)
PIC16F83	512 Flash	36	64	10
PIC16F84	1 K Flash	68	64	10
PIC16CR83	512 ROM	36	64	10
PIC16CR84	1 K ROM	68	64	10

- 14-bit wide instructions
- 8-bit wide data path
- 15 special function hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt on change
  - Data EEPROM write complete
- 1000 erase/write cycles Flash program memory
- 10,000,000 erase/write cycles EEPROM data memory
- EEPROM Data Retention > 40 years

Peripheral Features:

- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 20 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Pin Diagrams



Special Microcontroller Features:

- In-Circuit Serial Programming (ICSP™) - via two pins (ROM devices support only Data EEPROM programming)
- Power-on Reset (POR)
- Power-up Timer (PWRT)
- Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Code-protection
- Power saving SLEEP mode
- Selectable oscillator options

CMOS Flash/EEPROM Technology:

- Low-power, high-speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 6.0V
  - Industrial: 2.0V to 6.0V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 μA typical @ 2V, 32 kHz
  - < 1 μA typical standby current @ 2V

1.0 GENERAL DESCRIPTION

The PIC16F8X is a group in the PIC16CXX family of low-cost, high-performance, CMOS, fully-static, 8-bit microcontrollers. This group contains the following devices:

- PIC16F83
- PIC16F84
- PIC16CR83
- PIC16CR84

All PICmicro™ microcontrollers employ an advanced RISC architecture. PIC16F8X devices have enhanced core features, eight-level deep stack, and multiple internal and external interrupt sources. The separate instruction and data buses of the Harvard architecture allow a 14-bit wide instruction word with a separate 8-bit wide data bus. The two stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches (which require two cycles). A total of 35 instructions (reduced instruction set) are available. Additionally, a large register set is used to achieve a very high performance level.

PIC16F8X microcontrollers typically achieve a 2:1 code compression and up to a 4:1 speed improvement (at 20 MHz) over other 8-bit microcontrollers in their class.

The PIC16F8X has up to 68 bytes of RAM, 64 bytes of Data EEPROM memory, and 13 I/O pins. A timer/counter is also available.

The PIC16CXX family has special features to reduce external components, thus reducing cost, enhancing system reliability and reducing power consumption. There are four oscillator options, of which the single pin RC oscillator provides a low-cost solution, the LP oscillator minimizes power consumption, XT is a standard crystal, and the HS is for High Speed crystals. The SLEEP (power-down) mode offers power saving. The user can wake the chip from sleep through several external and internal interrupts and resets.

A highly reliable Watchdog Timer with its own on-chip RC oscillator provides protection against software lock-up.

The devices with Flash program memory allow the same device package to be used for prototyping and production. In-circuit reprogrammability allows the code to be updated without the device being removed from the end application. This is useful in the development of many applications where the device may not be easily accessible, but the prototypes may require code updates. This is also useful for remote applications where the code may need to be updated (such as rate information).

Table 1-1 lists the features of the PIC16F8X. A simplified block diagram of the PIC16F8X is shown in Figure 3-1.

The PIC16F8X fits perfectly in applications ranging from high speed automotive and appliance motor control to low-power remote sensors, electronic locks, security devices and smart cards. The Flash/EEPROM technology makes customization of application programs (transmitter codes, motor speeds, receiver frequencies, security codes, etc.) extremely fast and convenient. The small footprint packages make this microcontroller series perfect for all applications with space limitations. Low-cost, low-power, high performance, ease-of-use and I/O flexibility make the PIC16F8X very versatile even in areas where no microcontroller use has been considered before (e.g., timer functions; serial communication; capture, compare and PWM functions; and co-processor applications).

The serial in-system programming feature (via two pins) offers flexibility of customizing the product after complete assembly and testing. This feature can be used to serialize a product, store calibration data, or program the device with the current firmware before shipping.

1.1 Family and Upward Compatibility

Those users familiar with the PIC16C5X family of microcontrollers will realize that this is an enhanced version of the PIC16C5X architecture. Please refer to Appendix A for a detailed list of enhancements. Code written for PIC16C5X devices can be easily ported to PIC16F8X devices (Appendix B).

1.2 Development Support

The PIC16CXX family is supported by a full-featured macro assembler, a software simulator, an in-circuit emulator, a low-cost development programmer and a full-featured programmer. A "C" compiler and fuzzy logic support tools are also available.

TABLE 3-1 PIC16F8X PINOUT DESCRIPTION

Pin Name	DIP No.	SOIC No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	I	ST/CMOS (3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{\text{MCLR}}$	4	4	I/P	ST	Master clear (reset) input/programming voltage input. This pin is an active low reset to the device.
					PORTA is a bi-directional I/O port.
RA0	17	17	I/O	TTL	
RA1	18	18	I/O	TTL	
RA2	1	1	I/O	TTL	
RA3	2	2	I/O	TTL	
RA4/T0CKI	3	3	I/O	ST	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	I/O	TTL/ST (1)	RB0/INT can also be selected as an external interrupt pin.
RB1	7	7	I/O	TTL	
RB2	8	8	I/O	TTL	
RB3	9	9	I/O	TTL	
RB4	10	10	I/O	TTL	Interrupt on change pin.
RB5	11	11	I/O	TTL	Interrupt on change pin.
RB6	12	12	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming clock.
RB7	13	13	I/O	TTL/ST (2)	Interrupt on change pin. Serial programming data.
V_{SS}	5	5	P	—	Ground reference for logic and I/O pins.
V_{DD}	14	14	P	—	Positive supply for logic and I/O pins.

Legend: I= input O = output I/O = Input/Output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

3.1 Clocking Scheme/Instruction Cycle

The clock input (from OSC1) is internally divided by four to generate four non-overlapping quadrature clocks namely Q1, Q2, Q3 and Q4. Internally, the program counter (PC) is incremented every Q1, the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow is shown in Figure 3-2.

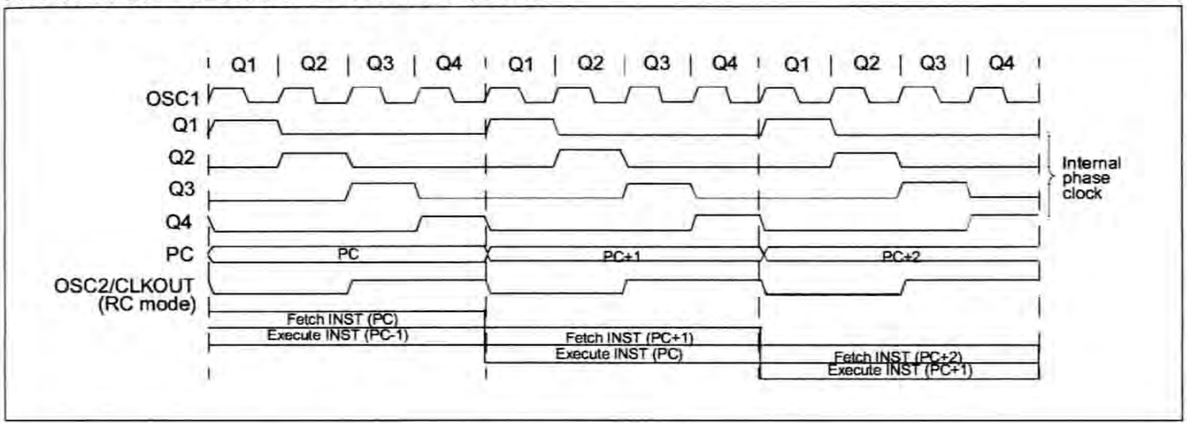
3.2 Instruction Flow/Pipelining

An "Instruction Cycle" consists of four Q cycles (Q1, Q2, Q3 and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g., `GOTO`) then two cycles are required to complete the instruction (Example 3-1).

A fetch cycle begins with the Program Counter (PC) incrementing in Q1.

In the execution cycle, the fetched instruction is latched into the "Instruction Register" in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

FIGURE 3-2: CLOCK/INSTRUCTION CYCLE



EXAMPLE 3-1: INSTRUCTION PIPELINE FLOW

1. MOVLW 55h	Fetch 1	Execute 1				
2. MOVWF PORTB		Fetch 2	Execute 2			
3. CALL SUB_1			Fetch 3	Execute 3		
4. BSF PORTA, BIT3				Fetch 4	Flush	
					Fetch SUB_1	Execute SUB_1

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

4.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F8X. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the "core" are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 7.0.

4.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F83 and PIC16CR83, the first 512 x 14 (0000h-01FFh) are physically implemented (Figure 4-1). For the PIC16F84 and PIC16CR84, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 4-2). Accessing a location above the physically implemented address will cause a wraparound. For example, for the PIC16F84 locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h will be the same instruction.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 4-1: PROGRAM MEMORY MAP AND STACK - PIC16F83/CR83

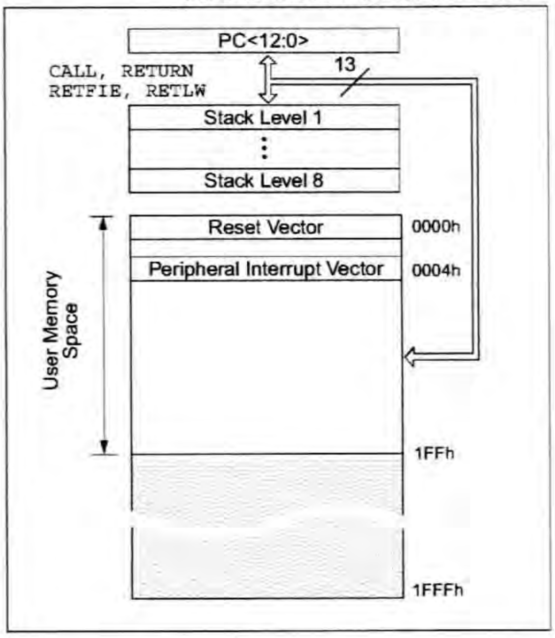
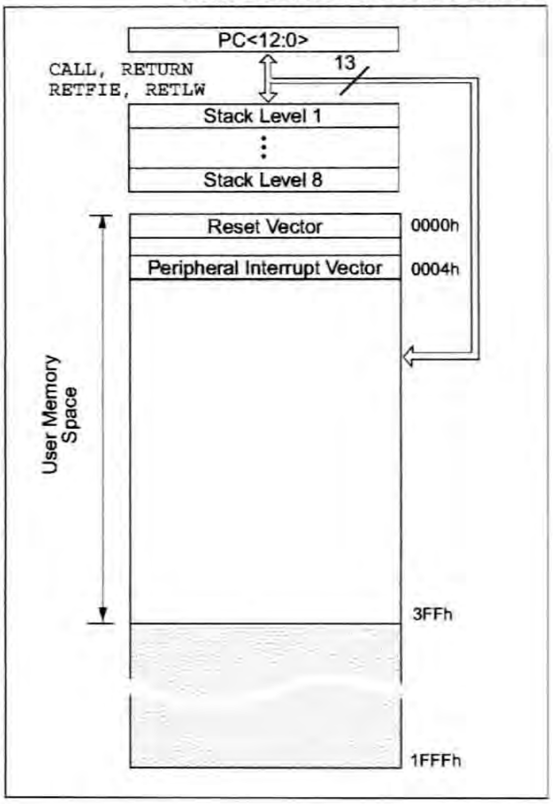


FIGURE 4-2: PROGRAM MEMORY MAP AND STACK - PIC16F84/CR84



4.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 4-1 and Figure 4-2 show the data memory map organization.

Instructions `MOVWF` and `MOVF` can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 4.5). Indirect addressing uses the present value of the RP1:RP0 bits for access into the banked areas of data memory.

Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers implemented as static RAM.

4.2.1 GENERAL PURPOSE REGISTER FILE

All devices have some amount of General Purpose Register (GPR) area. Each GPR is 8 bits wide and is accessed either directly or indirectly through the FSR (Section 4.5).

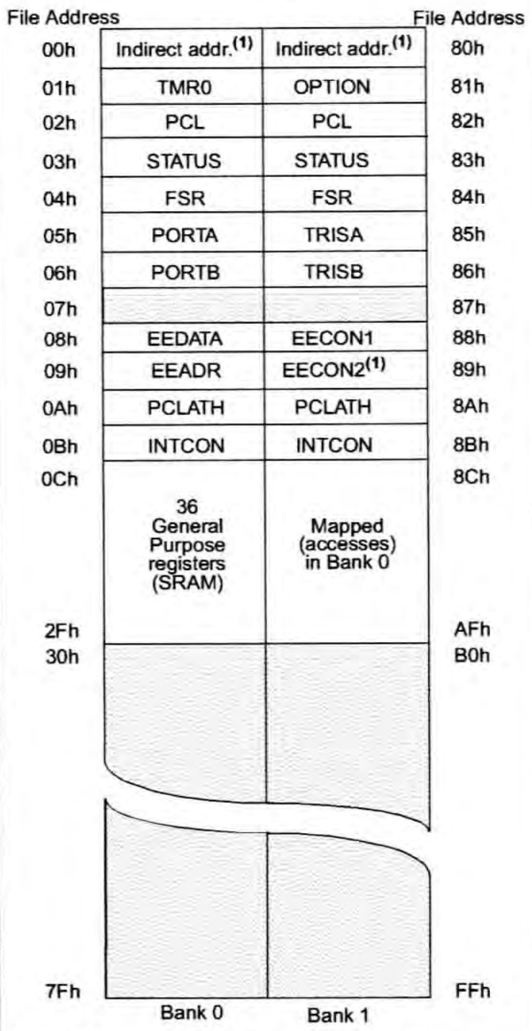
The GPR addresses in bank 1 are mapped to addresses in bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

4.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers (Figure 4-1, Figure 4-2 and Table 4-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

FIGURE 4-1: REGISTER FILE MAP - PIC16F83/CR83

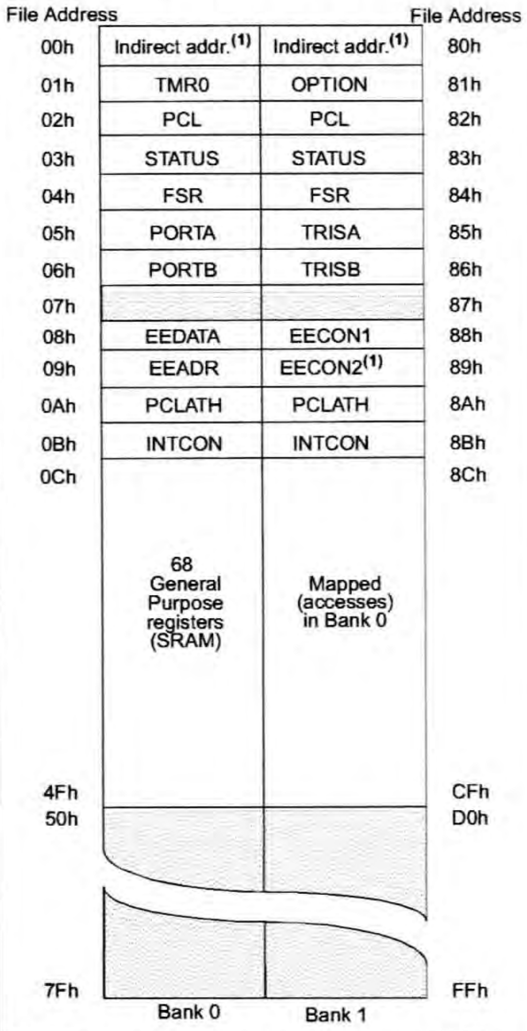


File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.[1]	Indirect addr.[1]	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	▨	▨	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2[1]	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	36 General Purpose registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
2Fh			AFh
30h	▨	▨	B0h
7Fh			FFh

▨ Unimplemented data memory location; read as '0'.

Note 1: Not a physical register.

FIGURE 4-2: REGISTER FILE MAP - PIC16F84/CR84



File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.[1]	Indirect addr.[1]	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	▨	▨	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2[1]	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
4Fh			CFh
50h	▨	▨	D0h
7Fh			FFh

▨ Unimplemented data memory location; read as '0'.

Note 1: Not a physical register.

TABLE 4-1 REGISTER FILE SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note3)
Bank 0											
00h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
01h	TMR0	8-bit real-time clock/counter								xxxx xxxx	uuuu uuuu
02h	PCL	Low order 8 bits of the Program Counter (PC)								0000 0000	0000 0000
03h	STATUS [(2)]	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
04h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07h		Unimplemented location, read as '0'								---- ----	---- ----
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC [(1)]					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
Bank 1											
80h	INDF	Uses contents of FSR to address data memory (not a physical register)								---- ----	---- ----
81h	OPTION_ REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	0000 0000
83h	STATUS [(2)]	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	000q quuu
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	uuuu uuuu
85h	TRISA	—	—	—	PORTA data direction register					---1 1111	---1 1111
86h	TRISB	PORTB data direction register								1111 1111	1111 1111
87h		Unimplemented location, read as '0'								---- ----	---- ----
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2 (not a physical register)								---- ----	---- ----
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC [(1)]					---0 0000	---0 0000
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged. - = unimplemented read as '0', q = value depends on condition.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> is never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ reset.

3: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and the Watchdog Timer Reset.

4.2.2.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{\mathrm{TO}}$ and $\overline{\mathrm{PD}}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper-three bits and set the Z bit. This leaves the STATUS register as `000u uluu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 9-2) because these instructions do not affect any status bit.

> **Note 1:** The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F8X and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.
>
> **Note 2:** The C and DC bits operate as a $\overline{\text{borrow}}$ and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.
>
> **Note 3:** When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

FIGURE 4-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\mathrm{TO}}$	$\overline{\mathrm{PD}}$	Z	DC	C
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IRP**: Register Bank Select bit (used for indirect addressing)
0 = Bank 0, 1 (00h - FFh)
1 = Bank 2, 3 (100h - 1FFh)
The IRP bit is not used by the PIC16F8X. IRP should be maintained clear.

bit 6-5: **RP1:RP0**: Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h - 7Fh)
01 = Bank 1 (80h - FFh)
10 = Bank 2 (100h - 17Fh)
11 = Bank 3 (180h - 1FFh)
Each bank is 128 bytes. Only bit RP0 is used by the PIC16F8X. RP1 should be maintained clear.

bit 4: **$\overline{\mathrm{TO}}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3: **$\overline{\mathrm{PD}}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2: **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1: **DC**: Digit carry/$\overline{\text{borrow}}$ bit (for `ADDWF` and `ADDLW` instructions) (For $\overline{\text{borrow}}$ the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0: **C**: Carry/$\overline{\text{borrow}}$ bit (for `ADDWF` and `ADDLW` instructions)
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred
Note: For $\overline{\text{borrow}}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

4.2.2.2 OPTION_REG REGISTER

The OPTION_REG register is a readable and writable register which contains various control bits to configure the TMR0/WDT prescaler, the external INT interrupt, TMR0, and the weak pull-ups on PORTB.

> **Note:** When the prescaler is assigned to the WDT (PSA = '1'), TMR0 has a 1:1 prescaler assignment.

FIGURE 4-1: OPTION_REG REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled (by individual port latch values)

bit 6: **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5: **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4: **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3: **PSA**: Prescaler Assignment bit
1 = Prescaler assigned to the WDT
0 = Prescaler assigned to TMR0

bit 2-0: **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

4.2.2.3 INTCON REGISTER

The INTCON register is a readable and writable register which contains the various enable bits for all interrupt sources.

Note: Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).

FIGURE 4-1: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

Note: For the operation of the interrupt structure, please refer to Section 8.5.

bit 6: **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit 5: **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: **INTE**: RB0/INT Interrupt Enable bit
1 = Enables the RB0/INT interrupt
0 = Disables the RB0/INT interrupt

bit 3: **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: **T0IF**: TMR0 overflow interrupt flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

bit 1: **INTF**: RB0/INT Interrupt Flag bit
1 = The RB0/INT interrupt occurred
0 = The RB0/INT interrupt did not occur

bit 0: **RBIF**: RB Port Change Interrupt Flag bit
1 = When at least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

LAMPIRAN B

TUTORIAL VISUAL BASIC

"Understanding & Using Visual Basic"

Microcontroller applications with the Basic Stamp, PIC, 8051 and various others can often be enhanced with the use of the PC serial port, and a software interface. Designing your own custom interface software for your next microcontroller application isn't as hard as you may think.

Using the PC serial port to interface to the outside world, and your next microcontroller application, can provide you with some extremely powerful software/hardware solutions.

This series of articles by Jared Hoylman will walk you through a few of the basic concepts, and then move on to the more advanced areas of communicating with your hardware, and having your hardware communicate back to the PC.

Introduction:

- Option Explicit
- DataTypes
- Parsing Strings
- Advanced Parsing
- Sending Data From The PC to a Microcontroller
- Receiving Data From The Microcontroller

Understanding and Using Visual Basic Part 1

By: Jared Hoylman -

Being a VB programmer there are many things that I have picked up over the past few years that greatly improved my programs and programming ability. In this series of articles I am going to cover some of the basics of VB programming and some Tips and Tricks to ease you along your way. This series of articles will start with the basic skills needed and work it's way up to the more advanced topics such as sending and receiving data from a Basic Stamp or Microchip PIC..!

Option Explicit

I am sure many of you have seen the words **Option Explicit** at the top of some VB code before. Why is it there, and what does it do..? Well, the **Option Explicit** statement forces you to declare your variables before you use them. Whoop-t-do, right..? Wrong..! These two simple word can save you hours of headaches debugging your programs..! It can also speed up your program considerably if used right..!

By placing **Option Explicit** at the top of every code module before any procedures you can guarantee that you will not misspell any variables. Lets see an example...

```
Private Sub Command1_Click()
Dim sMississippi As String
sMississipi = "Hello"  '<-- Note the missing "p"
MsgBox sMississippi
End Sub
```

What this code is actually supposed to do is display a MessageBox with the greeting "Hello". Since the variable is misspelled and there is no Option Explicit at the top of the code module, you get a **blank** MessageBox..!

Now go to the very top of the code module and type the words **Option Explicit**. Run the program again. What happened..? You get a "Variable not defined" error. This is a simple fix for what could be a complex problem.

Another reason that **Option Explicit** is so important is because if you do not declare your variables as a specific data type, VB defaults the variable to being type **Variant** *(See data types explained in the next article)*. A **Variant** type variable can hold any kind of data from strings, to integers, to long integers, to dates, to currency, etc. Even though this may sound like the best kind of variable to use, it is not. It is **the slowest** type of variable..! By defining your variables specifically for the kind of values that will be stored in them, will greatly increase your programs performance.

And to make it even easier, how about if I show you how to make VB automatically add **Option Explicit** to every code module..! It's easy.

Click on the **Tools** menu and select **Options...** Now check **Require Variable Declaration** Click **OK**

Now every time you open a new code module the words **Option Explicit** automatically appear at the top..!

Understanding and Using Visual Basic Part 2

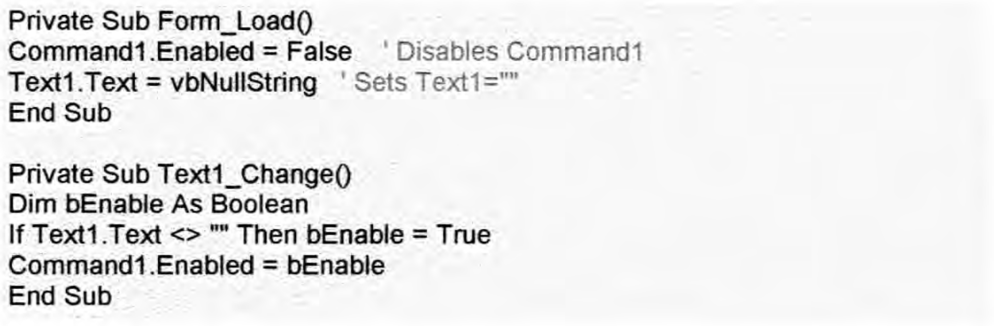
By: Jared Hoylman -

Understanding and Optimizing Data Types

In Visual Basic 6 there are 11 different data types. These are Boolean, Byte, Currency, Date, Double, Integer, Long, Object, Single, String, and Variant. They each have a specific purpose and using them correctly will increase your programs performance. I am going to cover the data types most frequently used.

• Boolean

The Boolean data type has only two states, **True** and **False**. These types of variables are stored as 16-bit (2 Byte) numbers, and are usually used for flags. For example, lets say that you have a textbox (Text1) and a command button (Command1). You only want Command1 to be Enabled when there is text in Text1. You would do something like this...



```
Private Sub Form_Load()
Command1.Enabled = False   ' Disables Command1
Text1.Text = vbNullString   ' Sets Text1=""
End Sub

Private Sub Text1_Change()
Dim bEnable As Boolean
If Text1.Text <> "" Then bEnable = True
Command1.Enabled = bEnable
End Sub
```

Run the program and Command1 will only be enabled when there is text typed into Text1.

• Byte

The Byte data type is an 8-bit variable which can store value from **0 to 255**. This data type is very useful for storing binary data. It can also be very useful when sending/receiving byte values to/from a Basic Stamp or PIC.

• Double

The Double data type is a 64-bit floating point number used when high accuracy is needed. These variables can range from -1.79769313486232e308 to -4.94065645841247e-324 for negative values and from 4.94065645841247e-324 to 1.79769313486232e308 for positive values.

• Integer

The Integer data type is a 16-bit number which can range from **-32768** to **32767**. Integers should be used when you are working with values that can not contain fractional numbers.

• Long

The Long data type is a 32-bit number which can range from **-2,147,483,648** to **2,147,483,647**. Long variables can only contain non-fractional integer values. I myself use Long variables over Integers for increased performance. Most Win32 functions use this data type for this reason.

• Single

The Single data type is a 32-bit number ranging from -3.402823e38 to -1.401298e-45 for negative values and from 1.401298e-45 to 3.402823e38 for positive values. When you need fractional numbers within this range, this is the data type to use.

• String

The String data type is usually used as a variable-length type of variable. A variable-length string can contain up to approximately 2 billion characters. Each character has a value ranging from 0 to 255 based on the ASCII character set. Strings are used when Text is involved.

Putting All Of This Technical Stuff To Use

Just to show you how to use these data types, here is a small example. Lets say that we have a String containing the text, "This VB stuff is pretty dam cool..!", and we want to convert each letter to it's ASCII equivalent. We will then display each letter along with its ASCII equivalent in a MessageBox one at a time.

```
Private Sub Command1_Click()
Dim sText As Strinq
Dim lTextLength As Long
Dim sChar As String
Dim bASCII As Byte
Dim x As Long

sText = "This VB stuff is pretty darn cool..!"
lTextLength = Len(sText) 'Gets # of chars in sText

For x = 1 To lTextLength 'Loop through string one char at a time
  sChar = Mid$(sText, x, 1)'Gets the x'th charcter in sText
  bASCII = Asc(sChar)      'Gets ASCII value of character
  MsgBox "The ASCII value of '" & sChar & "' is " & bASCII 'Display results
Next x

End Sub
```

Now run the code and it will display one character at a time along with it's ASCII value.

Understanding and Using Visual Basic Part 3

By: Jared Hoylman -

Parsing Strings

Strings are one of the most widely used data types, and yet parsing strings is one of the most mis-understood concepts to Visual Basic programmers. So I will show you how it is done.

• The Len Function

The **Len** function simply returns the number of characters within a string. For example...

```
Dim sText As String
Dim lTextLength As Long

sText = "Reynolds Electronics"
lTextLength = Len(sText)
```

After running this code **lTextLength** will equal **20**, which is the number of characters in the string **sText**

• The InStr Function

The **InStr** function will tell you if a string is within a string and where it starts. For example...

```
Dim sText As String
Dim lElectronics As Long

sText = "Reynolds Electronics"
lElectronics = InStr(sText, "Electronics")
```

After running this code **lElectronics** will contain the value **10**. If you count over from the beginning of the string you will notice that the word **Electronics** begins at the tenth letter of **sText**.

You can also use the **Instr** function just to determine whether a string is present within another string.

```
Dim sText As String
Dim lElectronics As Long

sText = "Reynolds Electronics"
If InStr(sText, "Electronics") Then
   MsgBox "Found the word 'Electronics'"
Else
   MsgBox "Did not find the word 'Electronics'"
End If
```

Run this code and it should tell you that it found the word **Electronics**. Now try changing **sText** to something else and run it again to see what happens...

• The Left Function

The **Left** function returns a specified number of characters from the left side of a string. For example run the following code and the results should show up in your Debug window.

```
Dim sText As String
Dim sLeft1 As String
Dim sLeft5 As String
Dim sLeft15 As String

sText = "Reynolds Electronics"
sLeft1 = Left$(sText, 1)
sLeft5 = Left$(sText, 5)
sLeft15 = Left$(sText, 15)

Debug.Print "The first letter is: " & sLeft1
Debug.Print "The first 5 letters are: " & sLeft5
Debug.Print "The first 15 letters are: " & sLeft15
```

• The Right Function

The **Right** function returns a specified number of characters from the right side of a string. For example run the following code and the results should show up in your Debug window.

```
Dim sText As String
Dim sRight1 As String
Dim sRight5 As String
Dim sRight15 As String

sText = "Reynolds Electronics"
sRight1 = Right$(sText, 1)
sRight5 = Right$(sText, 5)
sRight15 = Right$(sText, 15)

Debug.Print "The last letter is: " & sRight1
Debug.Print "The last 5 letters are: " & sRight5
Debug.Print "The last 15 letters are: " & sRight15
```

• The Mid Function

Now the **Mid** function is a little bit trickier so we are going to take this one a little slower. The Mid function needs three values passed to it, the String to search in, a starting position, and a length. What this function actually does is look in a string, starting at the position you tell it to start at, and retrieve the number of characters that you tell it to. So...

```
Dim sText As String
Dim sMidText As String

sText = "Reynolds Electronics"
sMidText = Mid$(sText, 3, 8)
```

After running this code **sMidText** should equal "**ynolds E**", which starts at the 3rd letter and goes through the 11th (3+8=11). See how it works..? It should also be noted that if a length is not stated it will return all characters from the starting position to the end of the string, as you will see below...

Putting It All Together

Now for some real fun..! Lets say that you are receiving data from a Basic Stamp. This data is stored in a buffer called **sBuffer**. Each set of data is separated by an ASCII 13. You want to be able to separate each of these sets of data one by one and display them in the debug window. Here goes...

```
Dim sBuffer As String
Dim lEnd As Long
Dim sData As String

' This is the data in your buffer
sBuffer = "Data1" & Chr$(13) & "Data2" & Chr$(13) & "Data3" & Chr$(13) &
"Data4" & Chr$(13)
lEnd = InStr(sBuffer, Chr$(13)) ' Gets Starting position of ASCII 13

Do ' Starts the loop
If lEnd > 0 Then ' If > 0 then we have an ASCII 13 in the buffer
  sData = Left$(sBuffer, lEnd - 1)
 ' sData will be all characters before lEnd
  sBuffer = Mid$(sBuffer, lEnd + 1)
 ' We want to delete the data that we just got from the buffer including
   the ASCII 13
  Debug.Print sData ' Display the data
  lEnd = InStr(sBuffer, Chr$(13))
 ' Gets Starting position of ASCII 13 in the new buffer
End If
Loop While lEnd > 0 ' Loop while ASCII 13 is still present in the buffer
```

After running this code you should see Data1 through Data4 show up in your Debug Window.

String manipulation is not really that hard. With the use of theses four functions you can parse any string that you want to. It just takes some planning as to how the data will be presented to you, and how you should go about getting the data that you want out of the big string. ;-)

Understanding and Using Visual Basic Part 4

By: Jared Hoylman -

Advanced String Parsing

In the last article we showed the four major string parsing functions along with a few simple examples. In this article we are going to kill two birds with one stone. PBASIC is not very friendly when it comes to conditional statements. Without the **If...ElseIf...Else...EndIf** kind of conditional statements, beginners find it difficult to program a Basic Stamp to do exactly what they want. If the PBASIC language had the **If...ElseIf...Else...EndIf** kind of conditional statements I believe it would be much easier for beginners to learn and program the Basic Stamp. So in this article we are going to use our VB string parsing functions to make a complete application that will manipulate true **If...ElseIf...Else...EndIf** VB style conditionals into ***functioning PBASIC code..!***

Lets take for example a VB type If conditional...

```
If [Condition1=True] Then
  DoCode#1
ElseIf [Condition2=True] Then
  DoCode#2
Else
  DoCode#3
End If
```

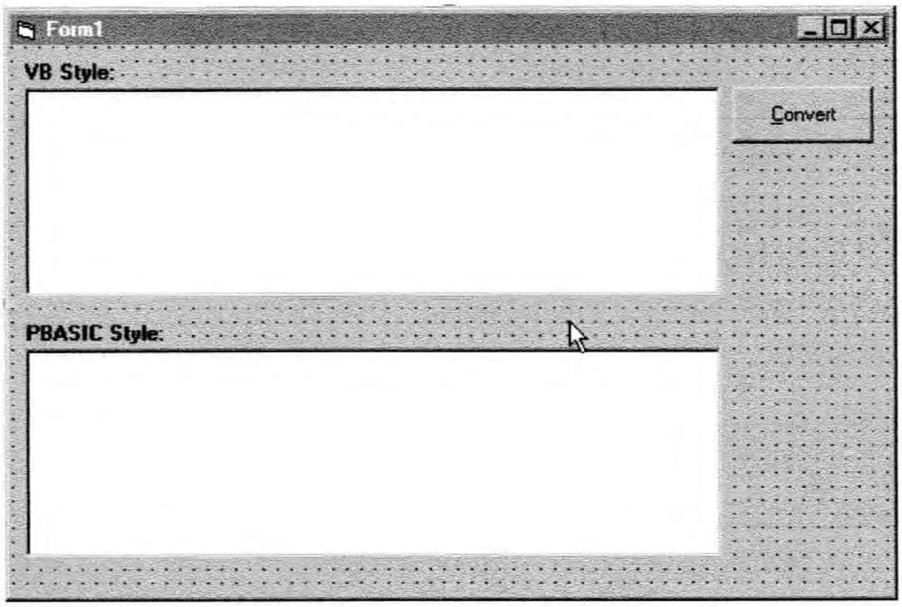
This could be transformed into PBASIC code by doing the following...

```
If [Condition1=True] Then DoCode1
If [Condition2=True] Then DoCode2
DoCode#3
EndIf:

' End of Main Program

' Start Extra Subs
DoCode1:
  DoCode#1
Goto EndIf

DoCode2:
  DoCode#2
Goto EndIf
```

This code would flow the exact same way as the VB typeIf conditional, only it looks a little more complex...

Creating The VB Program

Now you need to open up a New **Standard EXE** project. Add two labels (Label1 and Label2), two Textboxes (txtVB and txtPBASIC), and a command button (cmdConvert) to your form. Your form should look like the one below...



Now we need to set a few properties before we continue...

txtVB and txtPBASIC
Multiline = True
ScrollBars = 2 - Vertical
Text = "" ("" means nothing)

cmdConvert
Caption = "Convert"

And you can change the Label captions to whatever you wish.

How To Go About It

OK. The easiest way that I can think of doing it would be to loop through every line one at a time in the VB type code and convert it to the PBASIC type code. So we need a function to extract a specific line from **txtVB**. I went ahead and wrote one for you. It is a little confusing if you are not experienced in VB programming so I added LOTS of comments to help you out. Add This code to your form.

```
Private Function GetLineText(txtBox As TextBox, _
                             lLine As Long ) As String
Dim x As Long
Dim sText As String      ' Holds Textbox Text
Dim lLineStart As Long  ' Chr That Begins Line
Dim lLineEnd As Long     ' Chr That Ends Line
Dim lLength As Long      ' Length of line
sText = txtBox.Text

' We need to make sure that the text ends in a
' vbCrlf so...

If Right$(sText, 2) <> vbCrLf Then
    sText = sText & vbCrLf
End If

' If you want the first line of the textbox you
' know that the first character of the line
' will be the first character of the TextBox.

If lLine = 1 Then
    lLineStart = 1
Else

' If it isn't line 1 then we must find the first
' character of the line.  We know that each line
' is seperated by a vbCrLf (carriage return and
' line feed). So to find the second line starting
' position we find the 1st vbCrLf.  And to find
' the end of the second line we find the 3rd
' vbCrLf.

' This next little bit of code finds each vbCrlf
' up to (lLine - 1) which is the one that we need.

    lLineStart = 1  ' Initialize Offset
    For x = 1 To lLine - 1
    lLineStart = InStr(lLineStart, sText, vbCrLf)

    ' Compensate for the 2 characters in vbCrLf
    lLineStart = lLineStart + 2
    Next x
End If

' Now we need to find the end of the line.  We
' know that it is the very next vbCrLf after
' lLineStart, so...

lLineEnd = InStr(lLineStart, sText, vbCrLf)

' Get Line Length
lLength = lLineEnd - lLineStart

' Now we have the starting and ending characters
' for the line that we are trying to find.  Do
' you remember the Mid$ statement from the
' previous article..?

GetLineText = Mid$(sText, lLineStart, lLength)
End Function
```

OK. Now with that out of the way we need to discuss how we are going to convert the VB code to PBASIC code. Lets take the example code that I gave above.

```
If [Condition1=True] Then
  DoCode#1
ElseIf [Condition2=True] Then
  DoCode#2
Else
  DoCode#3
End If
```

And the PBASIC equivalent...

```
If [Condition1=True] Then DoCode1
If [Condition2=True] Then DoCode2
DoCode#3
EndIf:

' End of Main Program

' Start Extra Subs
DoCode1:
  DoCode#1
Goto EndIf

DoCode2:
  DoCode#2
Goto EndIf
```

Really it's not that difficult to do. If a line is like **If [condition] Then** or **ElseIf [condition] Then** we simple copy the line and add a Label to the end of the line. All of the code in between these statements gets put into a subroutine at the end of the program. If we find that a line equals **Else** then we simply copy the rest of the lines of code as they are. And finally if a line equals **End If** then we convert it to the Label **EndIf:**.

But first we need to go over another very useful string operator. It is the **Like** operator. The **Like** operator is used to compare two strings, usually using wildcards. So lets see an example...

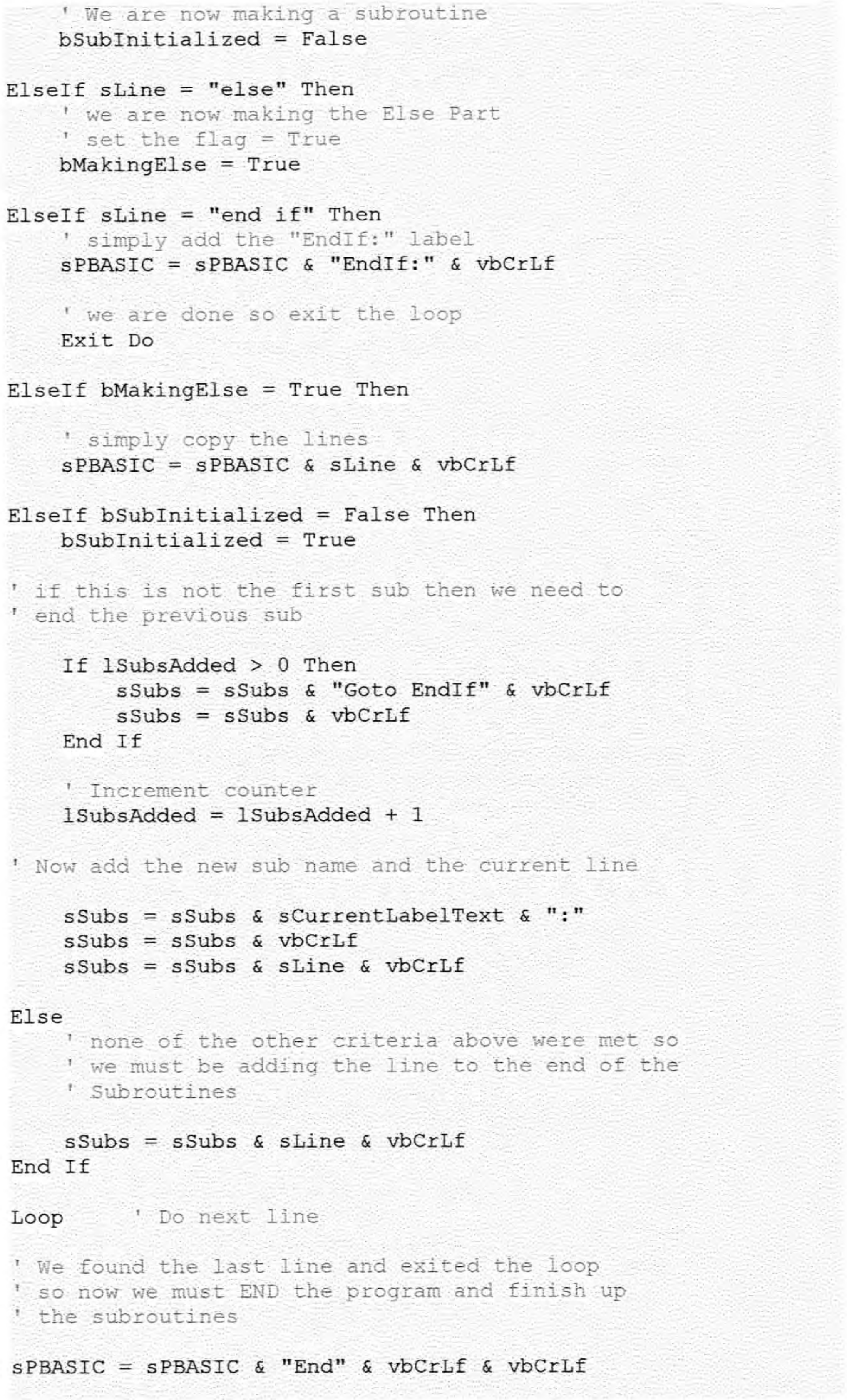
```
Dim bResult As Boolean
bResult = "If X=1 Then" Like "If * Then"
```

In this example **bResult** would equal **True** because the wildcard * can be any number of characters, so the pattern matches.

So with that out of the way here is the core part of the program. Again I added TONS of comments.

```
Private Sub cmdConvert_Click()
Dim sLine As String       ' Holds the line text
Dim sPBASIC As String     ' Hold converted string
Dim sSubs As String       ' Hold subroutines
Dim lLabelCount As Long ' Unique label counter
Dim lCurrentLine As Long      ' Current Line
Dim sCurrentLabelText As String
Dim bSubInitialized As Boolean
Dim bMakingElse As Boolean
Dim lSubsAdded As Long  ' Counts subs added
```

```
' We will simply loop through every line of txtVB
' until we find a line that equals "End If"

lCurrentLine = 0     ' Initialize line counter
lLabelCount = 0      ' Initialize the label count
Do
lCurrentLine = lCurrentLine + 1 ' Get next line
' Get current line text
sLine = GetLineText(txtVB, lCurrentLine)

' Convert line to lowercase for comparison reasons
sLine = LCase(sLine)

' remove all leading and trailing spaces
sLine = Trim(sLine)

If sLine Like "if * then" Then


    ' we have a line like "If [Condition} Then

    ' Increment LabelCount and set label name
    lLabelCount = lLabelCount + 1
    sCurrentLabelText = "Label" & lLabelCount

    ' Add the line
    sPBASIC = sPBASIC & sLine

    ' Then add the label and a vbCrLf
    sPBASIC = sPBASIC & " " & sCurrentLabelText
    sPBASIC = sPBASIC & vbCrLf

    ' We are now making a subroutine
    bSubInitialized = False

ElseIf sLine Like "elseif * then" Then


    ' we have a line like "ElseIf [Condition} Then
    ' we also need to take off the first 4 letters
    ' of the line to be vaid PBASIC

    sLine = Right$(sLine, Len(sLine) - 4)

    ' Increment LabelCount and set label name
    lLabelCount = lLabelCount + 1
    sCurrentLabelText = "Label" & lLabelCount

    ' Add the line
    sPBASIC = sPBASIC & sLine

    ' Then add the label and a vbCrLf
    sPBASIC = sPBASIC & " " & sCurrentLabelText
    sPBASIC = sPBASIC & vbCrLf
```



```
    ' We are now making a subroutine
    bSubInitialized = False

ElseIf sLine = "else" Then
    ' we are now making the Else Part
    ' set the flag = True
    bMakingElse = True

ElseIf sLine = "end if" Then
    ' simply add the "EndIf:" label
    sPBASIC = sPBASIC & "EndIf:" & vbCrLf

    ' we are done so exit the loop
    Exit Do

ElseIf bMakingElse = True Then

    ' simply copy the lines
    sPBASIC = sPBASIC & sLine & vbCrLf

ElseIf bSubInitialized = False Then
    bSubInitialized = True

' if this is not the first sub then we need to
' end the previous sub

    If lSubsAdded > 0 Then
        sSubs = sSubs & "Goto EndIf" & vbCrLf
        sSubs = sSubs & vbCrLf
    End If

    ' Increment counter
    lSubsAdded = lSubsAdded + 1

' Now add the new sub name and the current line

    sSubs = sSubs & sCurrentLabelText & ":"
    sSubs = sSubs & vbCrLf
    sSubs = sSubs & sLine & vbCrLf

Else
    ' none of the other criteria above were met so
    ' we must be adding the line to the end of the
    ' Subroutines

    sSubs = sSubs & sLine & vbCrLf
End If

Loop     ' Do next line

' We found the last line and exited the loop
' so now we must END the program and finish up
' the subroutines

sPBASIC = sPBASIC & "End" & vbCrLf & vbCrLf
```

```
If lSubsAdded > 0 Then
     ' subs were added so we need to end them
     sSubs = sSubs & "Goto EndIf" & vbCrLf
End If

' Now we just have to combine sPBASIC and sSubs
' and show them in txtPBASIC

txtPBASIC = sPBASIC & sSubs

End Sub
```

I know that is really long, but if you take your time and go through the code line by line it is not that hard to figure out the concept that is being used. If you just can't figure it out or just want to download the project already made...then click here.

Be sure to try it out as it might make programming you Basic Stamp just a little bit easier. And if you study this code and fully understand it, you are on your way to becoming a very good VB programmer. ;-)

Limitations

This code does have some limitations..!

- You can not nest If blocks
- There is no error checking. Try typing "End If" as "EndIf" and see what happens.
- There is no syntax checking. You can type anything you want in between the "If" lines and it will copy them anyways.

Anyways I hope that you learn something from this article. There are a lot of neat little tricks in this code that took me years to figure out. Learn from someone that has learned the hard way and make it easy on your self. ;-)

Understanding and Using Visual Basic Part 5

By: Jared Hoylman -

Sending Data To A Microcontroller

In the last article we went a little more in depth into Advanced String Parsing. These skills will be needed when receiving data from a microcontroller. All of the data received will be in a buffer and you will have to separate the data, and use it accordingly.

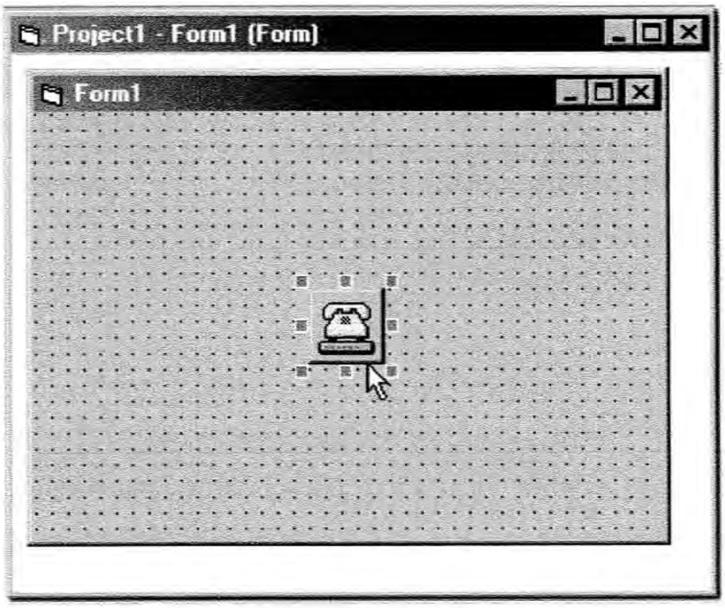
In this article we are going to learn how to send data to a microcontroller and make the microcontroller respond to the data. For simplicity we are just going to send data to turn certain pins on and off.

So lets start off with designing a communications protocol. From VB we will send an ASCII 255, as a synch byte, the pin number (0 to 15), and the on/off state (0 or 1). The microcontroller will wait for three (3) bytes of data. After these three bytes of data are received, it will then either switch the pins to their High or Low state.

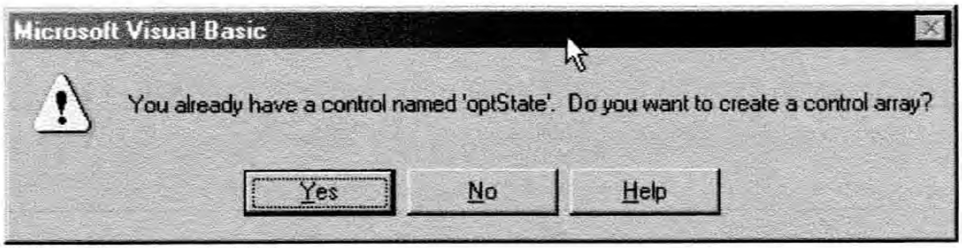
The VB Part

- To get started open Visual Basic.
- Start a new **Standard EXE**.
- Next go to the **Project | Components...** menu
- Check the **MSComm Control**.
- Click **OK**.
- Next **double-click** on the **yellow phone** in the toolbar to add the MSComm control to your form.

Your form should look like this...



Next we are going to add a Combo Box to our form and name it **cboPinNumber**. We are also going to add an Option Button to the form. Add the first one and name it **optState**. Select **optState** and press **Ctrl-C** to copy, and then **Ctrl-V** to paste. You will get a propmt like the one below...

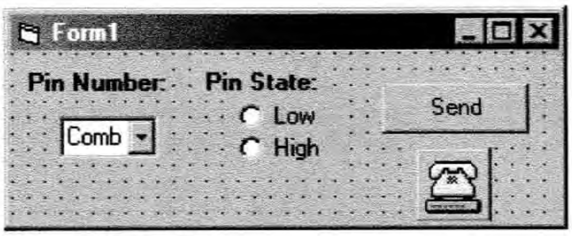


You already have a control named 'optState'. Do you want to create a control array?

Select **Yes**.

Now you should notice that the first option button that you created has a name of **optState(0)** and the second one has a name of **optState(1)**. The number after the control name is the control array index. We will use this to our advantage later. Also add a Command button named **cmdSend** to the form.

Go ahead and pretty up your form and add some labels to make it look better. The only requirement is that you set optState(0)'s Caption property equal to "Low", optState(1)'s Caption property equal to "High", cmdSend's Caption property equal to "Send", and cboPinNumber's Style property equal to (2). Your form should look something like mine, below...



On To The Code

Now that the form is set up and ready to go, we need to start adding our code to the project. The user will select a pin number from **cboPinNumber**, select a pin state from **optState**, and then click **cmdSend** to send the data to the microcontroller. So first of all we need to get the pin numbers into the combo box, set up the MSComm control, and select one of the pin states to start with. So lets add the following code to our form...

```
Private Sub Form_Load()
Dim Pins As Long

'  Add the pin numbers 0 to 15 to cboPinNumber
For Pins = 0 To 15
    cboPinNumber.AddItem CStr(Pins)
Next Pins

' Default to Pin 0 being selected
cboPinNumber.ListIndex = 0

' Default to optState(0) being selected
optState(0).Value = True

'  Use COM1
MSComm1.CommPort = 1

' 2400 baud, no parity, 8 data bits, 1 stop bit
MSComm1.Settings = "2400.N.8.1"
```

```
' Disable DTR
MSComm1.DTREnable = False

' Open the port
MSComm1.PortOpen = True

End Sub
```

Now we just need to add the code to send the data. When the user presses **cmdSend**, we need to do three things.

- Get the pin number...
- Get the pin state...
- Send the data...

Getting the pin number is very easy with the way that we have set up our form. As you may know a combo box lists items starting from a zero index. So this means that if we select pin 7 in the combo box, the combo box's ListIndex property is equal to 7.

To get the selected state we simply write a short **If** statement like so...

```
Dim PinState As Long
If optState(0).Value = True Then
    PinState = 0
Else
    PinState = 1
End If
```

Now lets put it all together and send the data when we press the **cmdSend** button...

```
Private Sub cmdSend_Click()
Dim PinNumber As Long
Dim PinState As Long

'  Get Pin Number
PinNumber = cboPinNumber.ListIndex

'  Get Pin State
If optState(0).Value = True Then
    PinState = 0
Else
    PinState = 1
End If

' Send Out Data
MSComm1.Output = Chr$(255) & Chr$(PinNumber) & Chr$(PinState)

End Sub
```

So we sent out the synch byte (255), followed by the pin number, followed by the pin state.

Finally we need to close the comm port when the VB project unloads so...

```
Private Sub Form_Unload(Cancel As Integer)
MSComm1.PortOpen = False
End Sub
```

We are finished with the VB part...! Not so bad, huh..?

The Microcontroller Part

For simplicity I am going to use a Basic Stamp II to receive the data and act accordingly to the data. You will need to hook up the Basic Stamp II just like you are going to program it and use the programming port (pin 16) for communications. Program the Basic Stamp II with the following code...

```
PinNumber  var   byte
PinState   var   byte

Main:

' Use the programming port to receive
' data at 2400 baud
' Wait for the synch byte (255) and then
' get the PinNumber and PinState
Serin 16,16780,[WAIT(255),PinNumber,PinState]

' If PinState=0 Then go to GoLow
' otherwise go to GoHigh
Branch PinState,[GoLow,GoHigh]
Goto Main


' Set The pin low
GoLow:
LOW PinNumber
Goto Main


' Set the pin high
GoHigh:
HIGH PinNumber
Goto Main
```

Now run the VB project, hook up some LEDs to some of the pins, select the pin number, select either HIGH or LOW, and press send. You should see the LED turn ON when you send a high command, and turn OFF when you send a LOW command.

Conclusion

This project could easily be expanded to send PWM, Serial data, ShiftOut data, and frequencies with just a little work. In the next article we will master the art of receiving data from the stamp and using it in VB..! Until then...have fun..!

Got questions...? Contact the author Jared Hoylman.....

To download a copy of Jared's VB project click here.

Understanding and Using Visual Basic Part 6

By: Jared Hoylman -

Receiving Data From A Microcontroller

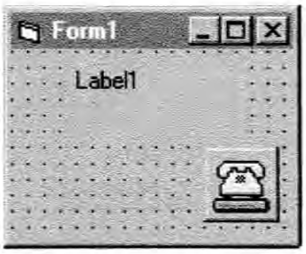
In the last article we sent data to a microcontroller and had it respond to the data that we sent...

In this article we are going to learn how to receive data from a microcontroller and make the PC respond. For simplicity we are going to have a Basic Stamp II get the RCTime value of a simple circuit and send the value to the PC for us to receive using VB.

We are going to get the RCTime value in the Stamp as a Word. This causes a little trouble when sending data to the PC, so I will show you how to split the word into two bytes, a HighByte and LowByte, and then join them back together to preserve the accuracy of the RCTime measurement...

The VB Part

- To get started open Visual Basic.
- Start a new **Standard EXE**.
- Next go to the **Project | Components...** menu
- Check the **MSComm Control**.
- Click **OK**.
- Next **double-click** on the **yellow phone** in the toolbar to add the MSComm control to your form.
- Finally add a label to your form and name it **lblRCTime**.



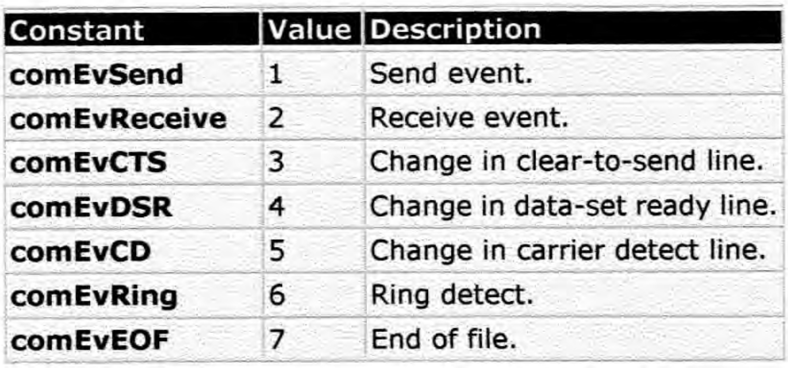
On To The Code
Now that the form is set up and ready, we need to get a quick understanding of how the

MSComm control can receive data from the serial port. There are basically two methods, polling the port and responding to communications events.

Polling the port is done by setting up a timer to check the buffer for data, and if data is there, process it. Polling is better when you have variable length data that starts and ends with header and footer bytes respectively.

The event driven method is designed more for receiving fixed length data. It is also better because you don't waist time polling the buffer for data if none is there. Instead the MSComm control will tell you when data is there by firing the **OnComm()** event. This event fires just like a **Click()** event would fire if you clicked on a Command Button, only it is not the users action that fires this event, something must happen at the serial port.

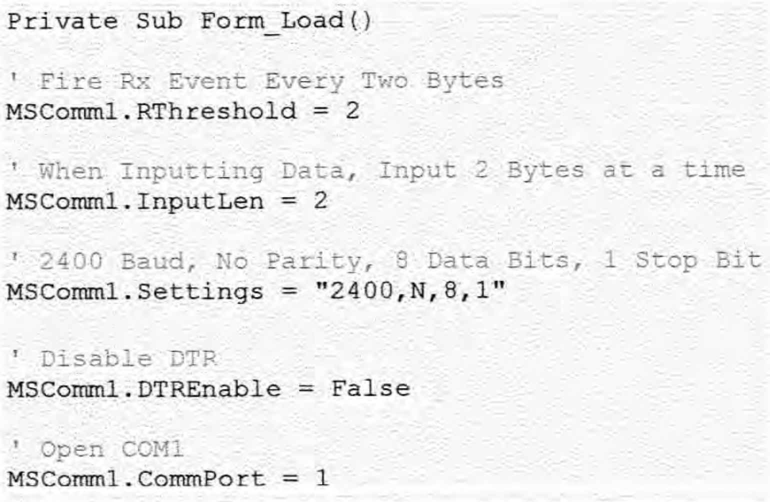
When the **OnComm()** event is fired you must check the value of the **CommEvent** property to see what exactly happened. The **CommEvent** property will contain a different value for each different kind of communication event that occurs. Below is a table that shows the event, the value of the **CommEvent** property, and the corresponding VB Constant...



Constant	Value	Description
comEvSend	1	Send event.
comEvReceive	2	Receive event.
comEvCTS	3	Change in clear-to-send line.
comEvDSR	4	Change in data-set ready line.
comEvCD	5	Change in carrier detect line.
comEvRing	6	Ring detect.
comEvEOF	7	End of file.

In this project we are only concerned with the **comEvReceive** constant which has the value of **2** and is fired when data is available in the buffer.

Now that we have a feel for how the MSComm control will assist us in receiving data, lets first set up the MSComm control. Copy this commented code to your project...



```
Private Sub Form_Load()

' Fire Rx Event Every Two Bytes
MSComm1.RThreshold = 2

' When Inputting Data, Input 2 Bytes at a time
MSComm1.InputLen = 2

' 2400 Baud, No Parity, 8 Data Bits, 1 Stop Bit
MSComm1.Settings = "2400,N,8,1"

' Disable DTR
MSComm1.DTREnable = False

' Open COM1
MSComm1.CommPort = 1
```

```
MSComm1.PortOpen = True

End Sub
```

You may notice that there are two new properties in the above code that have yet to be explained in these articles. The first is **RThreshold**. In this example, setting the **RThreshold** property equal to **2** tells the MSComm control to fire the **comEvReceive** event when there are at least two bytes available in the buffer. The second property, **InputLen**, tells the MSComm control to only give us the first two bytes when we request data from the buffer.

With that out of the way, lets take a look at the code that is executed when the **OnComm()** event is fired. The comments should help you along...

```
Private Sub MSComm1_OnComm()
Dim sData As String ' Holds our incoming data
Dim lHighByte As Long    ' Holds HighByte value
Dim lLowByte As Long     ' Holds LowByte value
Dim lWord As Long        ' Holds the Word result

' If comEvReceive Event then get data and display
If MSComm1.CommEvent = comEvReceive Then

    sData = MSComm1.Input ' Get data (2 bytes)
    lHighByte = Asc(Mid$(sData, 1, 1)) ' get 1st byte
    lLowByte = Asc(Mid$(sData, 2, 1))  ' Get 2nd byte

    ' Combine bytes into a word
    lWord = (lHighByte * &H100) Or lLowByte

    ' Convert value to a string and display
    lblRCTime.Caption = CStr(lWord)

End If
End Sub
```

The above code first checks the **CommEvent** property to see if a **comEvReceive** event has been fired. If so, we input the first two bytes of data, combine the bytes to make a word, and display the value in **lblRCTime**. Is it easier than you thought..?

The only thing left to do is close the COM port when our project is unloaded so...

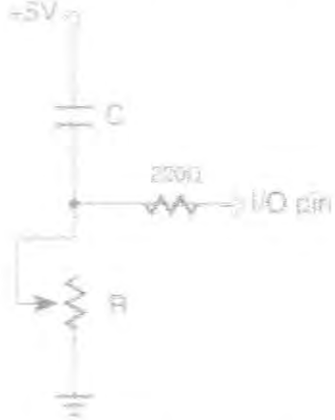
```
Private Sub Form_Unload(Cancel As Integer)
MSComm1.PortOpen = False
End Sub
```

The Microcontroller Part

For simplicity I am going to use a Basic Stamp II to send the data to the PC. You will need to hook up the Basic Stamp II just like you are going to program it and use the programming port (pin 16) for communications. Program the Basic Stamp II with the following code...

```
Result Var Word

Main:
High 0
Pause 1
RCTime 0,1,Result
Serout 16,16780,[Result.HighByte, Result.LowByte]
Pause 100
Goto Main
```

You will need to interface the Basic Stamp with a few components. Below is a schematic...

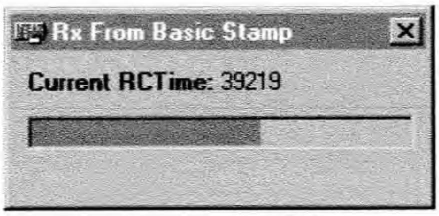


I used a **50k ohm** pot and a **2.2 uF** cap. This gave me results that almost covered the whole range of the RCTime measurement.

After hooking up the above circuit run your VB project, power up your Stamp, adjust the pot and watch the values change. You are now receiving data from the Stamp..!

Can't quite get it..?

If you just can't get it to work, or want to check out my project, I have uploaded something similar for you to download. It is pretty much the same code, but the form is more eye appealing and it shows the RCTime result in an analog sliding scale.



Current RCTime: 39219

Click here to download it...

Conclusion

This project could easily be amended to receive any kind of data. Create you own data logging software, plot data on a graph, even have your microcontroller control your PC. The possibilities are endless..!

I hope you enjoyed and learned from the **Understanding And Using Visual Basic** series of articles. This is the last of this series, but don't be surprised if you see more of my articles at Rentron in the future...

LAMPIRAN C

ATURCARA PIC

Aturcara PIC

```
*********************************************************************
;TITLE: Servo Motor Controller
*********************************************************************
;DEFINATION
*********************************************************************
        list P=PIC16F84A, R=D
        include "P16F84A.INC"
        _CONFIG_XT_OSC&_PWRTE_ON&_WDT_OFF&_CP_OFF
;set the configuration, kristal occilator, no code protection
; only used when writing code for PIC to communicate with serial port.
*********************************************************************
*****REGISTER USAGE*****
temp1  equ     0Ch
CBLOCK temp1
servo0mask              ;mask for servo 0 output
servo0position          ;loop count for servo 0 position
servo1mask              ;mask for servo 1 output
servo1position          ;loop count for servo 1 position
servo2mask              ;mask for servo 2 output
servo2position          ;loop count for servo 2 position
servo3mask              ;mask for servo 3 output
servo3position          ;loop count for servo 3 position
intwsave                ;interrupt w register store
intstatussave           ;interrupt status register store
flags
enables
leddrivecount
currentservomask
databtye
```

```
serialcurrentbyte
serialloopcount
parsercommand
parserdata
pasrsertemp
rtcc19msvalue       equ    107
rtcc1msvalue        equ    247
leddrivetime        equ    10
serialbitdelay      equ    103
serialhalfbitdelay  equ    52
**********************************************************************
;VECTOR
**********************************************************************
        org     0000h
        goto    start
        org     0004h
        goto    interrupt
*********************************************************************
;SUBROUTINES
*********************************************************************
getaddressofservodata       andlw       07h
                            addwf       pcl, f
                            retlw       000Ch
                            retlw       000Fh
                            retlw       0012h
                            movwf       intwsave
                            movf        status, w
                            movwf       intstatussave
                            btfss       flagwaitingforcts
                            goto        intlongdelayfinished
                            bcf         flagwaitingforcts
                            bsf         flagservopulsesafe
                            bcf         intcon, t0if
                            bcf         intcon, t0ie
```

```
                        goto        intfinish


int_longdelayfinished   bsf         serialcts
                        bsf         flagwaitingforcts
                        movlw       rtcc1msvalue
                        movwf       tmr0
                        bcf         intcon, t0if
                        bsf         intcon, t0ie

int_finish              movf        intstatussave, w
                        movwf       status
                        swapf       intwsave, f
                        swapf       intwsave, w
                        retfie

delaywbyfour            addlw       0ffh
                        btfss       status, z
                        goto        delaywbyfour
                        nop

                        return


doservocontrolpulse     call        getaddressofservodata
                        movwf       fsr
                        movf        indf, w
                        incf        fsr, f
                        iorwf       portb, f
                        movf        indf, w
                        addlw       1
                        incf        fsr, f
                        movf        indf, w
                        call        delaywbyfour
                        movf        currentservomask, w
                        xorwf       portb, f
                        return

disableinterrupts       bcf         intcon, gie
                        btfsc       intcon, gie
                        goto        disableinterrupts
                        return

********************************************************************

;MAIN
```

```
**********************************************************************
main                    bcf         status, rp0
                        call        disableinterrupts
                        clrf        porta

                        clrf        portb
                        bsf         status, rp0
                        movlw       0ah
                        movwf       trisa
                        movlw       00h
                        movwf       trisb
                        bcf         status, rp0
                        bsf         leddrive
                        call        initserialport
                        clrf        databyte
pwronmsgloop            movf        databyte, w
                        call        getpwronmsg
                        andlw       0ffh
                        btfsc       status, z
                        goto        endpwronmsgloop
                        call        transmitdatabyte
                        incf        databyte, f
                        goto        pwronmsgloop
                        movlw       01h
                        movwf       servo0mask
                        movlw       02h
                        movwf       servo1mask
                        movlw       04h
                        movwf       servo2mask
                        movlw       08h
                        movwf       servo3mask
                        clrf        flags
                        call        resetstate
                        bsf         status, rp0
                        movlw       86h
                        movwf       option_reg
                        bcf         status, rp0
                        movlw       rtcc19msvalue
                        movwf       tmr0
                        bcf         intcon, t0if
                        clrf        intcon
                        bsf         intcon, t0ie
                        bsf         intcon, gie
                        bcf         serialcts
startbitloop            btfss       serialrx
                        goto        startreceivedata
```

```
                    btfss       flagservopulsesafe
                    goto        startbitloop
                    goto        processservos

startreceivedata    call        disableinterrupts
                    call        receivedatabyte
                    movwf       databyte
                    bsf         intcon, gie
                    btfss       serialreceivevalid
                    goto        startbitloop
                    movlw       leddrivetime
                    movwf       leddrivecount
                    bcf         leddrive
                    movf        databyte, w
                    call        parsecommand
                    goto        startbitloop

processservos       bcf         flagservopulsesafe
                    btfss       flagservo0active
                    goto        doservo1
                    movlw       0

                    call        doservocontrolpulse

doservo1            btfss       flagservo1active
                    goto        doservo2
                    movlw       1

                    call        doservocontrolpulse

doservo2            btfss       flagservo2active
                    goto        doservo3
                    movlw       2

                    call        doservocontrolpulse

doservo3            btfss       flagservo3active
                    goto        doservo4
                    movlw       3

                    call        doservocontrolpulse

doservo4            btfss       flagservo4active
                    goto        doled
                    movlw                 4
                    call        doservocontrolpulse



doled               movf        leddrivecount, f
                    btfsc       status, z
                    goto        resettimer
                    decfsz      leddrivecount, f
                    goto        resettimer
                    bsf         leddrive

resettimer          movlw       rtcc19msvalue
                    movwf       tmr0
                    bcf         intcon, t0if
                    bsf         intcon, t0ie
                    bcf         serialcts
                    goto        startbitloop

resetstate          clrf        enables
                    movlw       128
                    movwf       servo0position
                    movwf       servo1position
                    movwf       servo2position
                    movwf       servo3position
                    clrf        portb
                    call        disableinterrupts
                    clrf        databyte
                    reset       msgloop
                    movf        databyte, w
                    call        getresetmsg
                    andlw       0ffh
                    btfsc       status, z
                    goto        endresetloop
                    call        transmitdatabyte
                    incf        databyte, f
                    goto        resetmsgloop
endresetloop        bsf         intcon, gie
                    return

parsecommand        btfss       parserwaitfordata
                    movwf       parsercommand
                    btfsc       parserwaitfordata
                    movwf       parserdata
                    swapf       parsercommand, w
```

```
                        andlw        0fh

                        movwf        parsertemp
                        movlw        0h

                        subwf        parsertemp, w
                        btfsc        status, z
                        goto         parseresetcmd
                        movlw        1h

                        subwf        parsertemp, w
                        btfsc        status, z
                        goto         parsesetservoposcmd
                        movlw        2h
                        subwf        parsertemp, w
                        btfsc        status, z
                        goto         parsesetservooffsetcmd
                        movlw        3h

                        subwf        parsertemp, w
                        btfsc        status, z
                        goto         parseenableservocmd
                        movlw        4h

                        btfsc        status, z
                        goto         parsedisableservocmd
                        bcf          parserwaitfordata
                        return

parseresetcmd           btfss        parserwaitfordata
                        goto         parserwaitfordatabyte
                        movf         parserdata, f
                        btfss        status, z
                        goto         parseresetinvalid
                        call         resetstate
                        bcf          parserwaitfordata
                        return

parseresetinvalid       bcf          parserwaitfordata
                        movf         parserdata, w
                        goto         parsecommand

parsesetservoposcmd     goto         parserwaitfordatabyte
                        movf         parsercommand, w
                        andlw        07h
                        call         getaddressofservodata
```

```
                            addlw       2
                            movwf       fsr
                            movf        parserdata, w
                            movwf       indf
                            bcf         parserwaitfordata
                            return

parsesetservooffsetcmd      btfss       parserwaitfordata
                            goto        parserwaitfordatabyte
                            movf        parsercommand, w
                            andlw       07h
                            call        getaddressofservodata
                            addlw       1
                            movwf       fsr
                            movf        parserdata, w
                            movwf       indf
                            bcf         parserwaitfordata
                            return

parseenableservocmd         movf        parsercommand, w
                            andlw       07h

                            call        parserchanneltobitmask
                            iorwf       enables, f
                            bcf         parserwaitfordata
                            return

parsedisableservocmd        movf        parsercommand, w
                            andlw       07h

                            call        parserchanneltobitmask

                            xorwf       enables, f
                            bcf         parserwaitfordata
                            return

parserwaitfordatabyte       bsf         parserwaitfordata
                            return

initserialport              bsf         serialtx
                            bsf         serialcts
                            movlw       10
                            movwf       serialloopcount

initserialdelay             movlw       serialbitdelay
                            call        delaywbyfour
```

```
                        decfsz      serialloopcount, f
                        btfss       status, z
                        goto   I    nitserialdelay
                        return

transmitdatabyte        movwf       serialcurrentbyte
                        btfss       serialcts
                        bsf         serialrestorects
                        bsf         serialcts
                        bcf         serialtx
                        movlw       serialbitdelay
                        call        delaywbyfour
                        movlw       8
                        movwf       serialloopcount

txdatabyteloop          bsf         status, c
                        rrf         serialcurrentbyte, f
                        btfss       status, c
                        bcf         serialtx
                        btfsc       status, c
                        bsf         serialtx

                        movlw       serialbitdelay
                        call        delaywbyfour

                        decfsz      serialloopcount, f
                        goto        txdatabyteloop

                        bsf         serialtx
                        movlw       serialbitdelay
                        call        delaywbyfour

                        btfsc       serialrestorects
                        bcf         serialcts
                        bcf         serialrestorects
                        return

receivedatabyte         bcf         serialreceivevalid
                        movlw       0
                        movwf       serialcurrentbyte
                        movlw       serialhalfbitdelay
                        call        delaywbyfour
                        btfsc       serialrx
                        return

                        movlw       8
```

```
                    movwf      serialloopcount

rxdatabyteloop      movlw      serialbitdelay
                    call       delaywbyfour
                    btfss      serialrx
                    bcf        status, c
                    btfsc      serialrx
                    bsf        status, c
                    rrf        serialcurrentbyte, f
                    decfsz     serialloopcount, f
                    goto       rxdatabyteloop
                    movlw      serialbitdelay
                    call       delaywbyfour
                    btfsc      serialrx
                    bsf        serialreceivevalid
                    movf       serialcurrentbyte, w
                    return
                    end
```