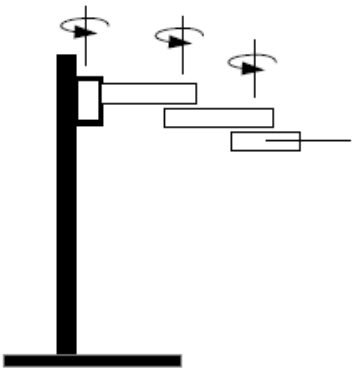
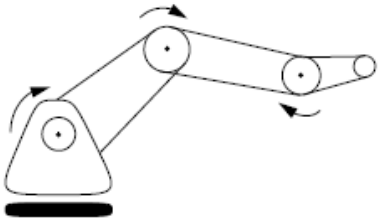
I | Inspiration in our Design and Mechanism

When we thought of building our Robot Arm, the first thing that we needed to decide upon was the design. We had scoured the internet to get an idea of how to build Robot Arms and their contemporary designs.

The basic two types that we found were **I. Anthropomorphic** and **II. SCARA**.



SCARA



ANTHRPOMORPHIC

Amongst these, SCARA was a design that uses the motion of the motors such that the rotational axis is always perpendicular to the ground. We found this design, attractive, yet limited in its approach. This was simple; however the deployable areas were fairly limited. Gravity balancing was not an issue in here and challenges involved were minimum. We chose to discard this design and proceed with the latter.

The Anthropomorphic design represented fairly the human arm. This design was a juxtaposition of vertical and horizontal axes, thus providing a high mobility and flexibility. Gravity balancing was a challenge involved since the Arm's weight had to be balanced entirely by the power of the Motors.

Thus we proceeded with the Anthropomorphic design. We also discarded ideas on building our robot from metal casts. Our aim was to keep our design flexible and casting the structure out of a die would only mean a frozen and rigid design that would inhibit any type of future modifications.

We proceeded with a frame type Anthropomorphic design, that not only allowed rapid prototyping, but also the flexibility that we required.

We froze the number axes to four. This was a well-thought-off decision so that the tradeoff between cost, complexity and ease of fabrication may be easily dealt with.

We had consulted the design of the robot arm Lynx-6 by Lynxmotion before we built our robot. This robot was the nearest to what we had visualized and gave us a fair enough idea about how the fabrication could be made possible.

II| Robot Arm Controller Architecture Concept

Once the design has been visualized it was our turn to conceptualize the Architecture of the control mechanism of the robot. Our aims were to produce the following results:

- Implement individual axis control of the arm
- Implement automatic control of the Robot Arm through a learning technique
- Implement Forward Kinematics
- Implement Inverse Kinematics

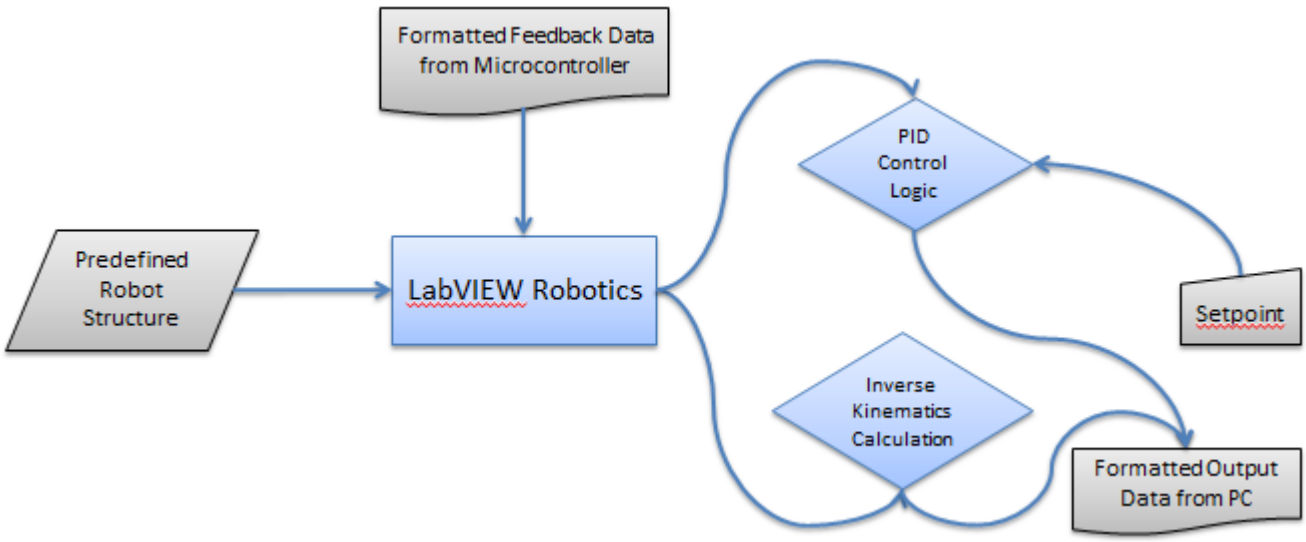
The first mechanism was simple. However the latter three mechanisms use a heavy computation that involves floating point arithmetic due to the inherent nature of calculations that involve matrices and their logical transforms.

This kind of facility is not available with a simple 8-bit Micro-controller. Only advanced 32-bit controllers have the ability to compute floating point arithmetic and often come with a FPU coprocessor.

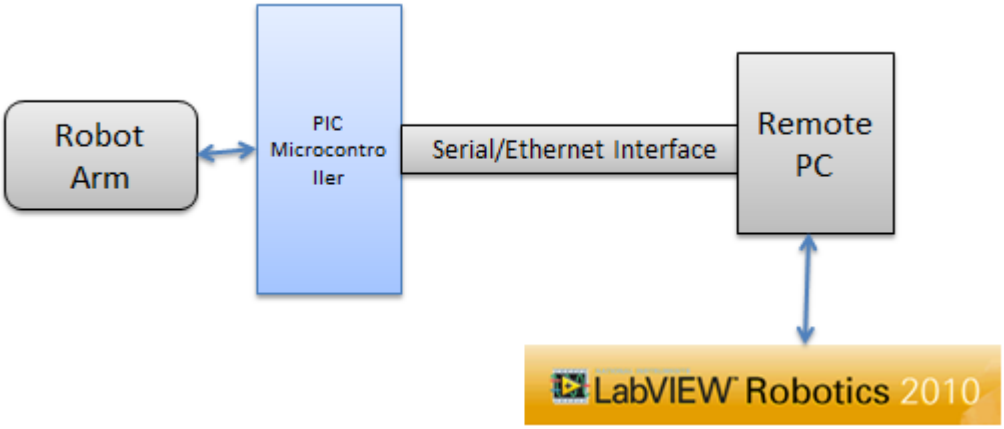
We decided to go ahead to perform the computations on a laptop that will aid the function and control of the Robot.

A 8-bit Micro-controller was used to act as an interface between the servo motors and the PC. We chose the PIC16F877a due to its robustness, easy availability and reliability in industrial environments.

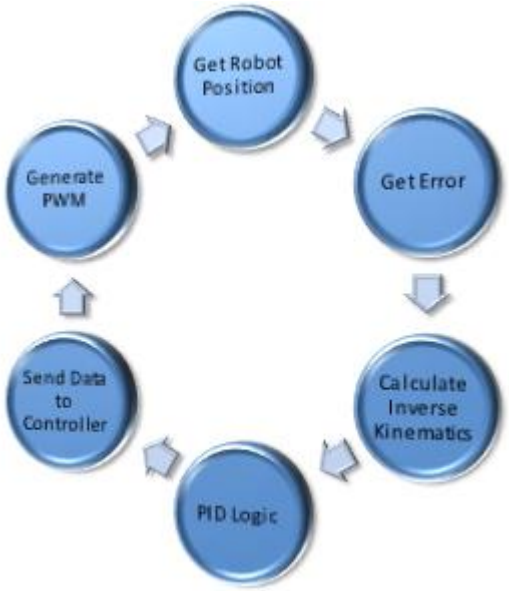
The architecture is explained diagrammatically below:



Basic Architecture



Overview of the Hardware Architecture used

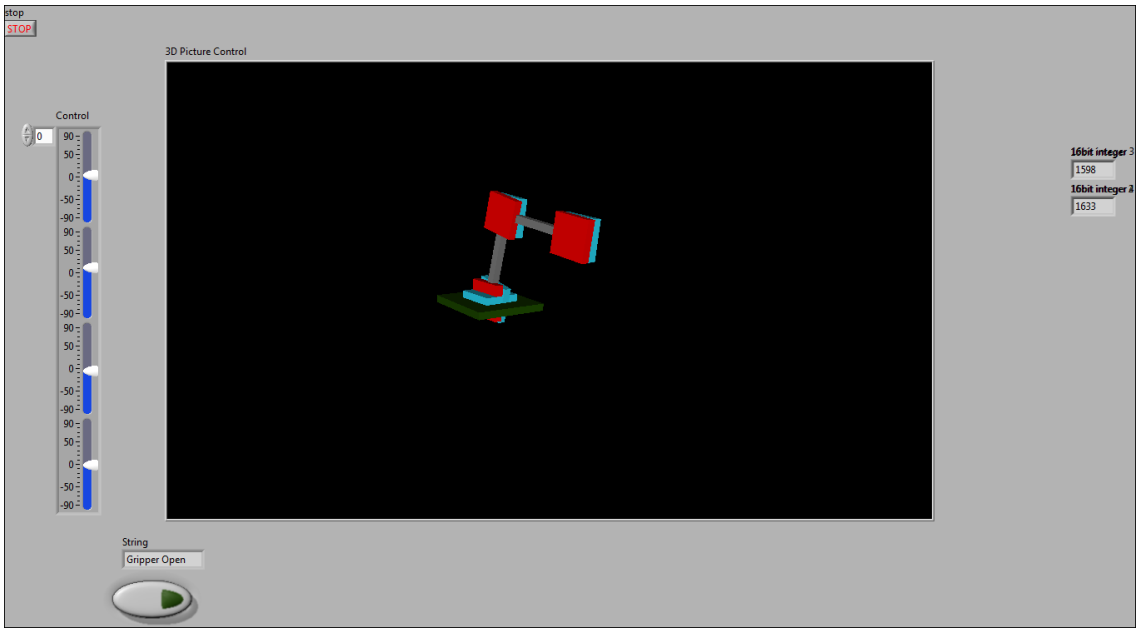


Estimated Operation flow of a single cycle

III | Simulations of Concepts and Improvisations

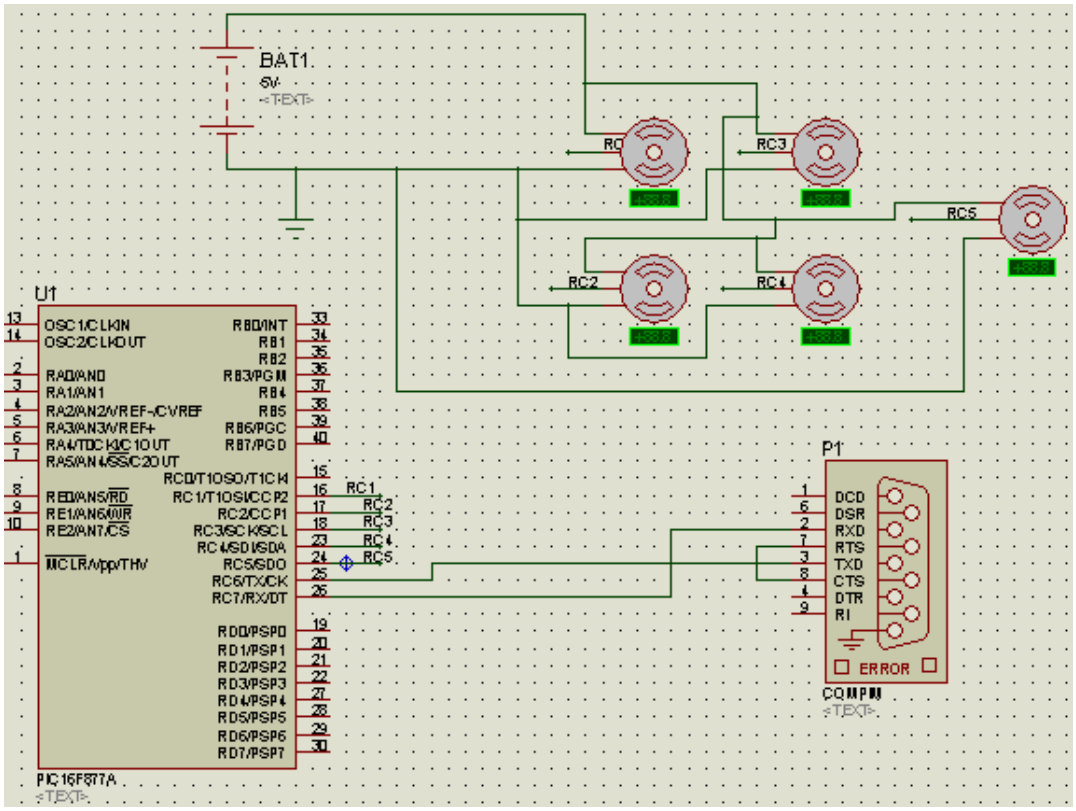
Our ideologies could never have been correct without simulating our designs and codes. The simulations, ideas, intuitions and improvisations put together gave birth to the project as it is in its current form.

- **Task 1: Test the viability of our mechanical design**



  - We were lucky enough to have the DH parameters of the Lynx-6 Arm at our hand. We simulated our design with the DH parameters of the Lynx-6 Arm which was similar to our Arm. The simulation platform initially was the Robotics Toolbox for Matlab. Then we learnt that an advanced form of the toolbox was available in LabVIEW Robotics 2010. The final simulations of the motions were being carried out in LabVIEW.
  - We also found out techniques to integrate a Solidworks Design with LabVIEW and simulate the Solidworks Design from LabVIEW itself. We only performed a basic simulation using this technique due to lack of proper knowledge in designing in Solidworks.

Task 2: Test the Servo Operations in Proteus 7.5



- Proteus 7.5 provided us with the platform to carry out electronic simulations to test our circuit in a block wise fashion
- We designed basic interfaces between various components used like EEPROM, Servo Motors, LCD, UART and tested them individually.
- The trickiest part was to get the servos ready for operation. We've never used a servo before and this provided us a very good learning experience.
- Once all components were simulated individually with success, the final simulation had to be the integration of all the former results.
- This part was highly critical as use of interrupts came in while operating the Serial transceiver. Also operating all the servos together needed a great deal of precision in timing the controller signals as all the servos expected a response from the controller in a period of 20ms.
- Thus, to serve the servos in a proper protocol, we had to avoid the collision of the the timings. The machine cycle taken by each instruction had to be taken into consideration to execute this step successfully.
- For all the above procedures the coding platform used was MikroC.
- However in the later stages this had to be altered due to the fact that the bootloader installed in our PIC16F877a supported only the Hitec PIC-C Compiler.
- Thus we needed to port our code to the PIC-C Compiler to actually execute the code on our device. The task was not critical was time consuming as the methods and API's were fairly different.

- **Task 3: Simulate the Control of Servo operations by generating commands from the LabVIEW interface to control motors in LabVIEW**
  - To accomplish this task we needed to use two components to act as a virtual serial bridge between two applications: VISA and the Virtual Serial Port.
  - VISA was used by LabVIEW as a library to generate serial instructions.
  - The virtual serial port driver enable us to create a pair of serial ports on our Windows PC to act as a channel between LabVIEW and Proteus. Herein Proteus was a virtual replacement for our actual hardware.
  - The control system was designed in LabVIEW so as to generate instructions that are legible to the PIC16F877a controller. The legibility was provided by suitable coding the PIC16F877a controller to understand the specific instructions to carry out certain tasks.
  - The success of this task indicated that our serial control mechanism was good to go.

Once the simulations were ready, we knew it was time to develop the PCB. Also, we kept in mind that simulation results could never be perfect and it’s a different game when it came to running code on the actual device.

However the simulations helped to build our logic block by block and helped correct our ideas and made it possible to provide us a basic level to start off with a precise and accurate knowledge of what's in store for us next.

IV | Mechanical Hardware Design

We followed a bottom-up approach in designing the mechanical structure of the Arm. We started off from designing the base, and then figured out that we needed a bearing housing, manufactured the bearing housing, the shoulder and then then the rest of the links.

Design Considerations: Before designing the arm, the parameters that we expected from the arm were frozen. The parameters required were the following:

- Total Arm Reach: 45cms Horizontal
- Arm Speed: 30degrees/sec/axis
- Accuracy: +-0.5cm
- Grip width: 60mm
- Grip power: 5kg/cm
- Payload: 200gm maximum
- Best suitable material: Carbon Fiber/ Aluminum

The design of the mechanical structure was indeed a critical task since any error or misalignment would trigger a malfunction since any error may not be detected automatically by the underlying code. The robot will not follow the instructions provided by the PC in the intended fashion and it would have been a really hectic task to compensate the errors.

Tools Used: All the mechanical components were being fabricated by hand by using the following machines:

- Lathe Machine
- Oxy-Acetylene Welding
- Press
- Drill
- Hacksaws, Cutters, Chisels, Files, Mallet, Hammer, Pliers, Screw Drivers etc.

Oxy-Acetylene Welding used to weld Aluminum

Facing the Bearing Housing in Lathe

Metal Cutter to shear the Aluminum plates

Press Machine to bend Steel at right angles

Drilling our jobs

Mechanical Components: The machines will be discussed later along with the mechanical components produced from them. This will provide a free flow story telling of the process that we followed while manufacturing the arm.

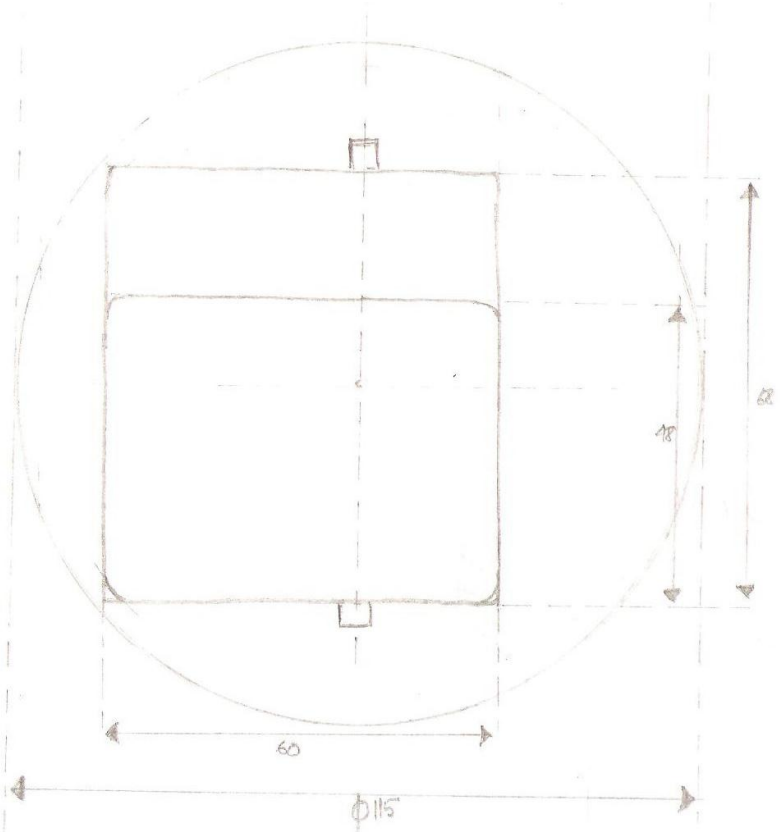
All the mechanical components involved are described and discussed below along with their mode of fabrication:

- Base
- Bearing Housing
- Shoulder
- Elbow
- Fingers
- Accessories
    - Bearing Shaft
    - Dummy Shaft
    - Locking Mechanism for unpowered Links

According to the requirements that are decided upon we started the design of the Robot Arm. The drawing were entirely done on ED Sheets and sometimes on A4 paper since there was no proper knowhow of the CAD tools like AutoCAD, Solid Works etc.

- **The Base of the Robot**
    - **Requirement:** The Robot base was designed such that it had the ability to hold the base servo. Also the design had to be such that it had the ability to support the entire structure to prevent a toppling of the Robot under its maximum extended conditions.

    - **Design:** The base of the Robot was thus best suited to be a cylindrical structure. The diameter was kept such that it would just prevent a toppling. Also we decided to bolt the base to the ground if any toppling would occur. The diameter of the base was kept at 150mm. And a height of 60mm. This was just enough for it to maintain all the required parameters.



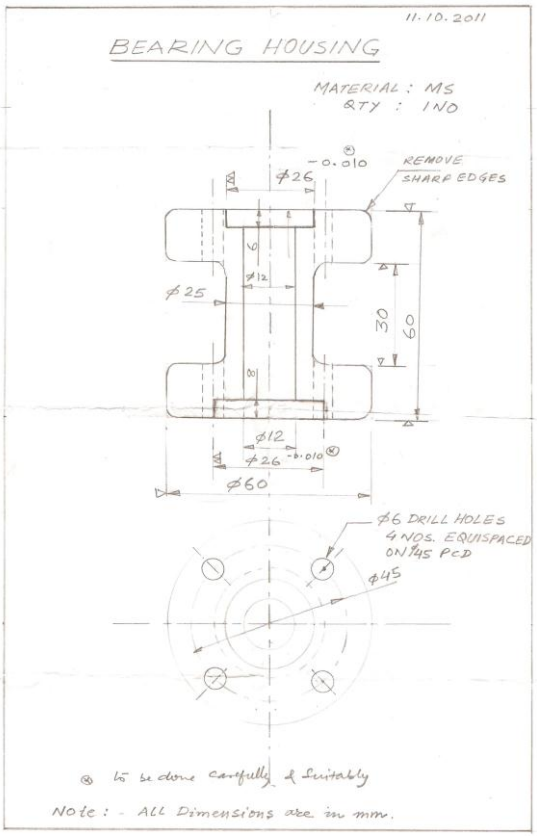
Topview of the Base

- **Manufacturing:** Initially a prototype base was being built by using aluminum sheet of width 2mm. We cut out two pieces from the sheet, one circular and the other rectangular. The rectangular sheet was folded into a cylinder and the circular sheet was fixed atop the cylinder by means of Oxy-Acetylene welding. This was the only way of welding aluminum sheets known to us and the one for which we had the resource. Then however we changed our base to a more firm structure, because of two primary reasons: We did not like the ergonomics and he finish obtained from the Oxy-Acetylene Welding process; also the base seemed to be a little too small and tended to get toppled too easily. The new base was built from a container out of an old Hawkin's Pressure Cooker. This structure seemed more stable and ergonomical at the same time. The base needed to give support to the bearing housing and the base motor, so appropriate arrangements had to be made so that everything made be fixed easily and in a reproducible fashion. Also simplicity and symmetry was kept in mind so that the robot may be assembled and de-assembled by a layman just by using simple instructions and guidelines. This enabled our motive of building a robot for educational as well as universal usage. The simple yet exhaustive design of the base only implied that it will function as it should with compromising the flexibility and all without restricting the possibilities of modifications that may be made to the base in future.

- **The Bearing Housing**
  - **Requirement:** We figured out that we needed a bearing housing because there existed two different kinds of forces on the base. There existed a torsional force and a uniform lateral force as well as the upper structure of the Robot Arm moved on top of the Base. We also needed two different types of bearings to compensate these two different types of forces. To compensate the torsional force we needed a Thrust Bearing. And to balance out the Lateral Force, we needed a Ball Bearing. The two

different types of bearings were being chosen from the Mechanical Data Book. The bearing size was being selected most suitably and the closest to what was possible to our pre-notions and imaginations.

- **Design:** The bearing housing was designed as a toroidal structure. The toroid shape was being chosen over a cylindrical shape to reduce weight of the structure while maintaining the same characteristics and usage. The height that we required was fixed and the other dimensions were being chosen to match the most suitable parameters. The toroidal structure carried a Shaft inside it that rotated freely amidst the two bearings either of which resided on either ends of the bearing housing. The shaft connected the motor on one end and the rest of the Robotic System on the other end.

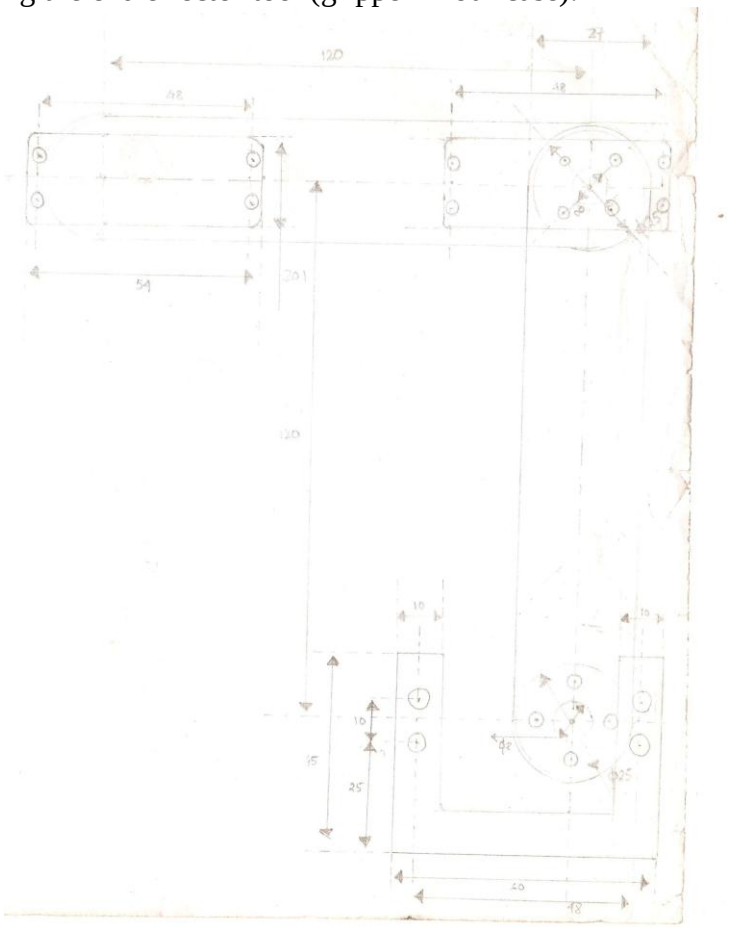


The Bearing Housing

- **Manufacturing:** The only option to build such an accurate bearing housing was to fabricate the entire thing ourselves. We chose the metal as Aluminum so that the weight/strength ratio is very small. We purchased a solid Aluminum cylinder from the market in its crude form. Turned it in a lathe machine to achieve perfect roundness. Then the hole for the shaft was created by drilling using the Lathe machine itself at a low speed. The groove to create the toroid shape was created later so as to reduce the weight of the system even more, and provide a unique ergonomics to the design. Also the bearing seats were created with heavy caution so that the bearings on either side fitted tightly onto the structure. Also the depths of the seats were to be made with high caution so that the ball bearing was just inserted into the bearing housing. In case of the Thrust bearing we intended to keep the depth such that one plate of the bearing lied just outside the bearing housing. This was done to keep the external robot structure rest on one plate of the Thrust bearing. While working on the Lathe machine we received proper guidelines and instructions from

the authority on how to operate and machinery. We achieved the perfection that was required through practice and time. Also later, drilling had to be done on the bottom face of the Bearing Housing so that it may be bolted tightly to the base. This was being done using a 5mm drill at medium speed.

- **Shoulder**
  - **Requirement:** We required the shoulder to transmit power from the shoulder motor to the forward linkages. This linkage was required to lift up the forward linkages including the end effector tool (gripper in our case).



Shoulder and Elbow links

  - **Design:** The design in this phase was relatively simple. We only had to align two aluminum strips in parallel. However the constraint was that the center of rotation of both the plates had to lie on the same horizontal line. This horizontal line also had to be exactly parallel to the ground. Also to hold together the two Aluminum plates to form a link, one aluminum shaft was used. This shaft had a diameter of 11mm, had a height of 60mm and had threading on either side so that bolts may be attached to fix the two plates parallel to each other. To connect this link with the base we attached the servo horn with plate. And to the just opposite of the center of the metal horn, to the other link a hole of 8mm was made. So that a dummy shaft could be inserted about which the free plate could rotate without a problem. Also metal circlips were fixed on the shaft to prevent bubbling of the shoulder links.

  - **Manufacture:** The material chosen for the links are 3mm thick and 60mm wide aluminum plates. The most optimal material for the job would have been carbon fiber but however due to its extreme price and lack of availability, we chose to stick to the aluminum plates. The aluminum plates were drilled in the proper locations. We used a 5mm Drill, an 8mm Drill and an 8mm Thread Cutter for the manufacture

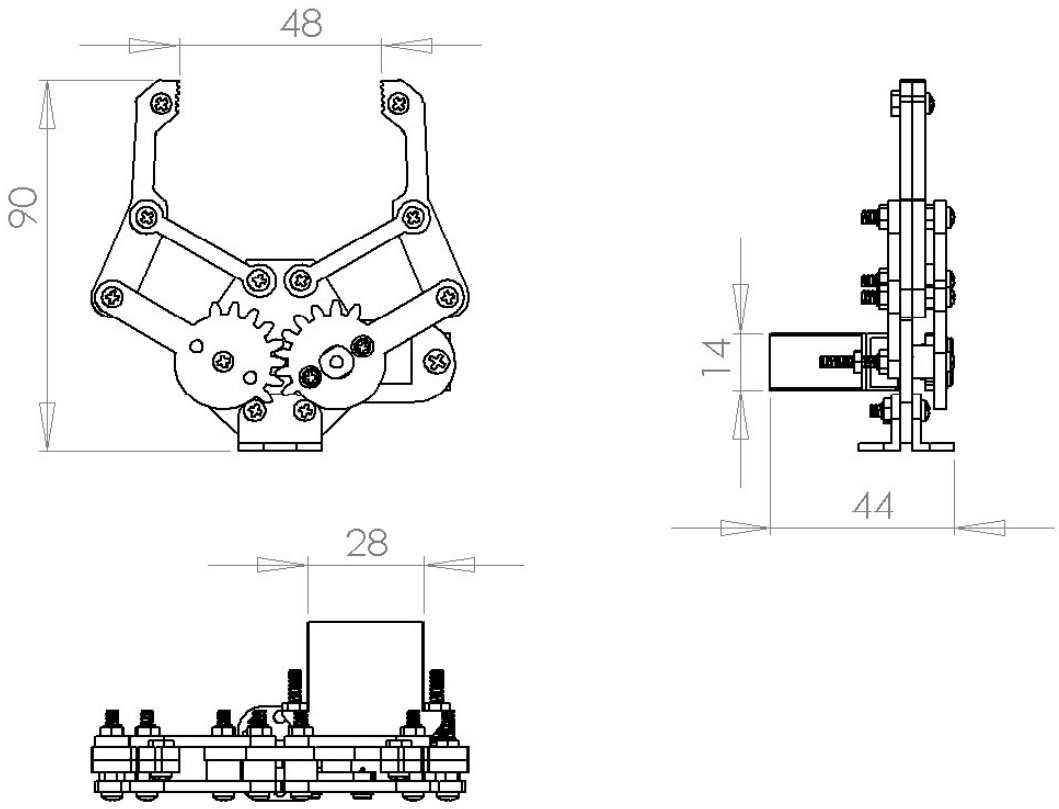
process.

- **Elbow**
  - **Requirement:** The Elbow had to carry two servo motors on each end. The rest of the design should be similar to the Shoulder Link. The first servo motor is used to move the wrist up and down. The second motor is used to move the end effector tool. Also the elbow had to be very light weight since a small increase of weight at this distance from the base of the robot would exert a high moment on the shoulder motor.
  - **Design:** The shoulder motor had an H shaped design to maintain uniformity and also rigidity of the linkages. The H shaped material had a rectangular window so that the servos may be attached easily and the wings maybe be fixed using bolts on the aluminum plate. Also the center of rotation of the Elbow lied exactly on the center of the Elbow plate and the perpendicular to the ground.

  - **Manufacture:** We were required to cut V-Grooves on the metal plates to create the H-Shape of the plates. This was achieved by using a hacksaw and a file to smoothen the edges and required repeated cutting of the V- Groove to achieve the desired cut. The cuts for the motors in between the Arm was done using multiple drilling along the edges of the cut section that was required and then cut using a fine chisel and hammer. The edges were filed later to achieve the desire finish. Also drillings were done inappropriate positions so that the motors may be attached firmly and parallel to the linkages. Also care was taken at each stage so that the alignment was maintained and the Arm was still in good shape and perfect.

- **Gripper**
  - **Requirement:** The gripper was required to be a firm and rigid structure that performed the desired operation smoothly and quickly. Also it should be able to sustain the force exerted by the gripper motor (5kg/cm) without any bending/distortion in shapes. It should be built for repeated usage. A prolonged use must not wear out the device.

  - **Design:** The design was copied from the Gripper design that was available with Sparkfun Electronics®.



- **Manufacture:** The material used for the gripper was Duralmin. We used Duralmin due to the extremely low weight, soft type that enabled us to cut out the metal in any shape we wanted to. The Gripper was manufactured by Sparkfun Electronics®, USA by using a press device that was used to cut out the shapes designed when a heavy cutter fell on a block of aluminum. The design accuracy achieved was really incredible. This was required badly as any inconsistency with the design and the actual product was intolerable and would render our work useless.

- **Accessories**
  - **Bearing Shaft**
    - **Requirement:** The bearing shaft was required to have a diameter 12mm<d<11mm. And it should have a height that would directly enable us to connect the base motor and the housing of the shoulder motor.
    - **Design:** The design of the shaft was easy. However the methods of attachments on the either ends was tricky and critical. We finally fixed the diameter of the shaft to 11mm. This shaft also had threading on both ends to facilitate the attachments on either side. Also it is locked by a circlip on the bottom side to lock it into its position.
    - **Manufacture:** The shaft was manufactured with great fidelity by turning in a lathe machine. Also the ends were faced to achieve a superior finish. The edges were threaded using the Lathe Machine.

  - **Dummy Shaft**
    - **Requirement:** The dummy shaft was needed to hold the unpowered side of the links. We needed three such dummy shafts. Also the dummy shafts had to have some mechanism to hold the unpowered side of the links in a fixed place to prevent bubbling.
    - **Design:** The dummy shaft should have a length of 40mm. The locking mechanism is implemented by putting two grooves of depth 1mm, put apart

at a distance of 3mm so that two circlips could be inserted on either sides of the unpowered plate of the link to lock it into position effectively.

- **Manufacture:** The dummy shaft is constructed from the same aluminum rod we used earlier for the bearing shaft. It was manufactured by turning the rod in a length including the grooves. Also a thread of 40mm was put on one end so that a bolt may be attached from the inner side.

- **Other attachments:** Other attachments that were necessary were primary things like nuts, bolts, screws, circlips of various sizes.

Final Photo of the Robot (base unattached)

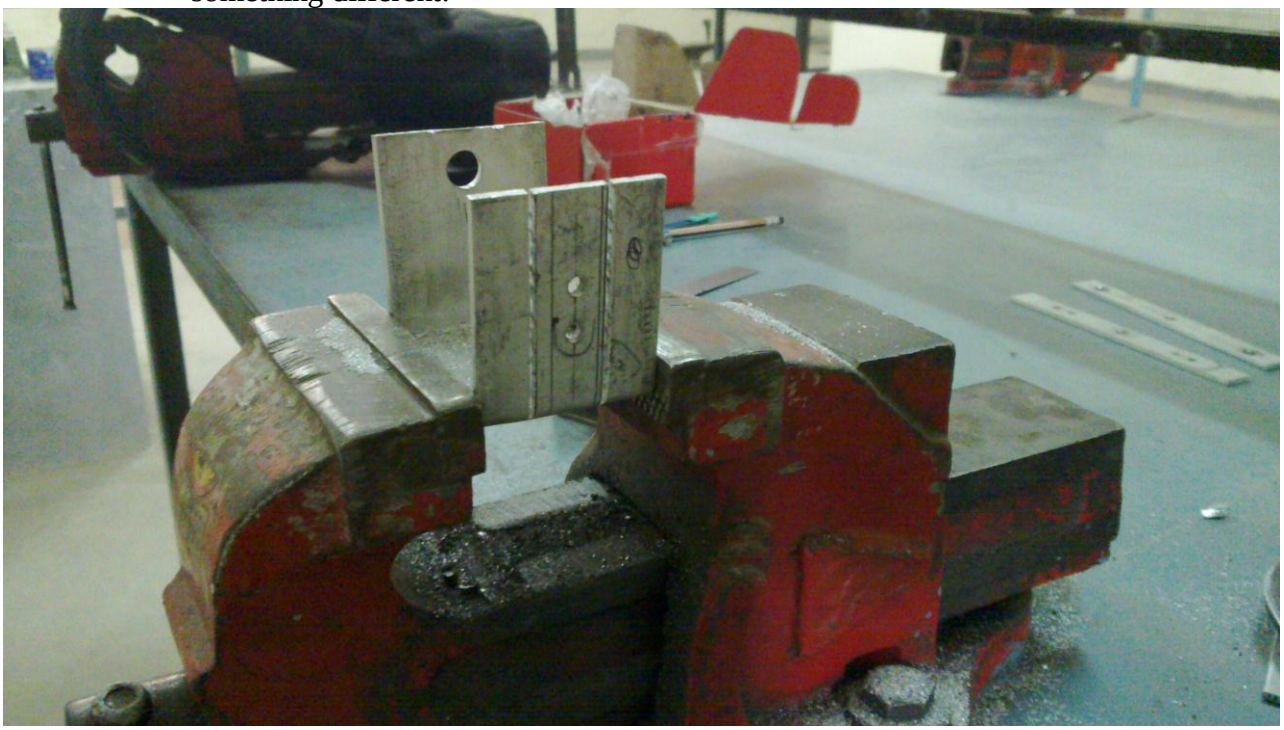
Problem with the structure we built

We came across a pretty interesting problem with our structure once we thought it was ready and good to go. The Shoulder motor was unable to function normally. We narrowed down the reason with an increased weight of the forward links that were tending to exert a higher moment on the Shoulder servo.

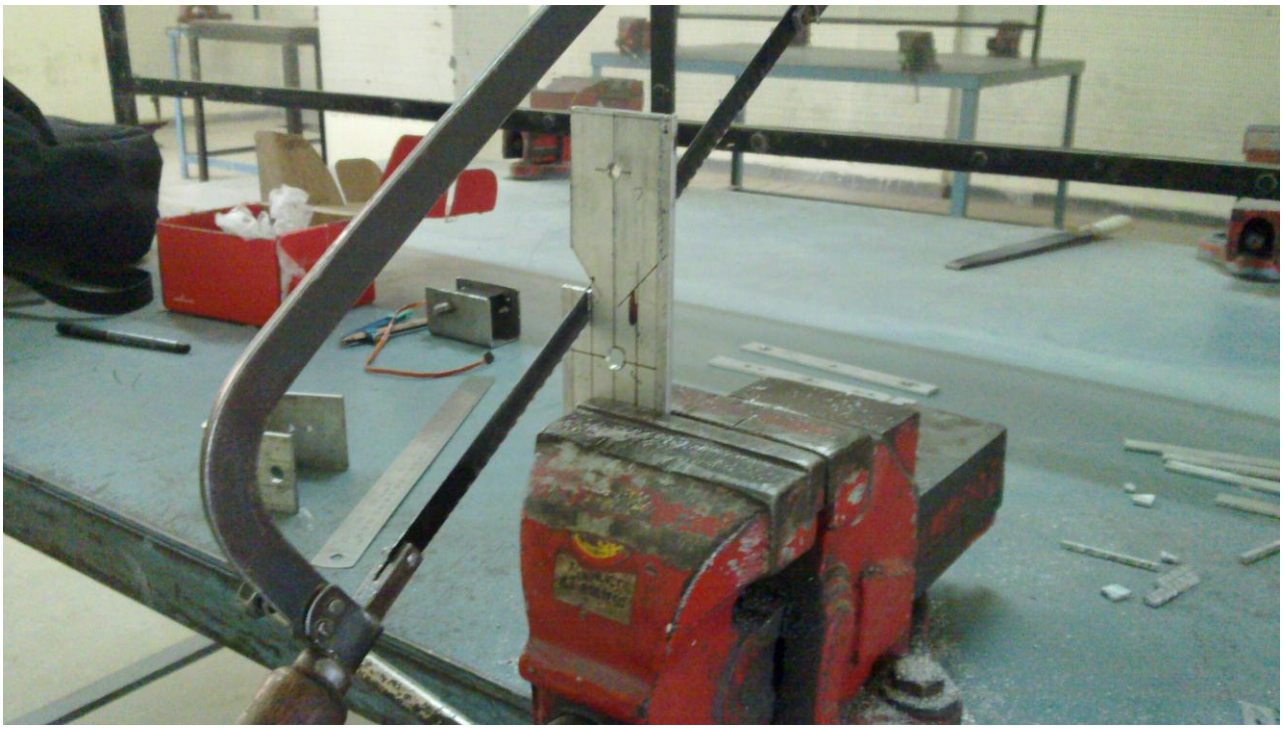
- **Attempts to fix the problem:** We brainstormed a lot to find mechanisms to fix the problems with the structure we had built. These were the following options:
  - **Find a motor with a higher capacity:** This was apparently the simplest decision. However we could not find a motor that provided a torque higher than 13kg/cm in

the local stores and the online stores based in India. This option was proved much difficult than it appeared to and we discarded this option.

- **Reduce weight of the structure:** We cut off irrelevant sections of the links in an attempt to reduce the weight of the system. But the material was aluminum. Laying off every cubic cm of the metal provided a weight drop of only 20gms. We also thought of perforating the entire structure by using a drill of 25mm diameter. However our calculations indicated that only some 250gms may be reduced by this process. This was simply nowhere near to what we needed. We knew we needed something different.



Sawing off all unwanted portions of the chassis



Cutting V-Grooves in the Elbow

Cutting off portions from the shoulder

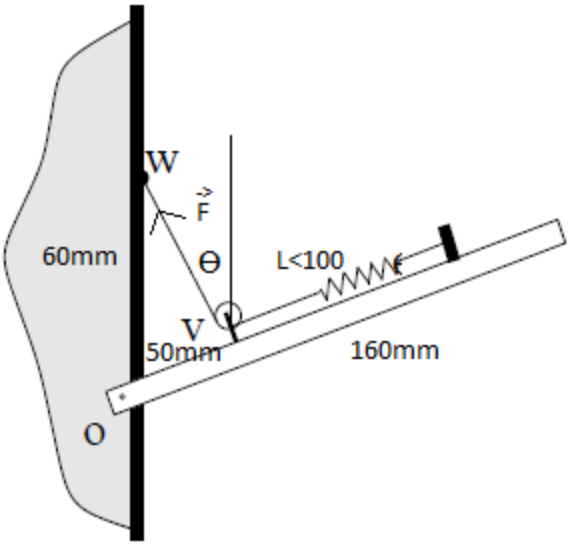
- **Cancel out the effects of gravity:** We came to know this by studying two different things. Automatically sustained Desk Lamps that tend to maintain their state once their state was altered by hand. Also some industrial robotic arms used an antigravity mechanism to lift heavy loads my using normal motors. This seemed just the option we needed to balance our Robot!

Statically balanced desk lamps

- **The Antigravity Spring Mechanism:** Our attempt was to build a mechanism that will store the loss/gain of potential energy of the arms as they changed positions in a spring. The spring will provide the energy exactly when required. The effects of gravity would be cancelled out entirely. And the shoulder motor now should function entirely to move the links. It won't have any contribution to support the weight of the linkages. It thus provided an antigravity effect, and was aptly named as "The Antigravity Spring Mechanism".
  - **Calculations involved:** We needed to calculate the exact properties of the spring that we required for our purpose. This was to specify the exact parameters to the spring manufacturer to get it built. Some of the factors were constrained. They were needed to be taken into consideration to provide the mechanism for the spring. The strained factors are being given as follows:
    - **Spring Length:** This should be less than 100 in its stretched condition.
    - **Height of the perpendicular structure:** This could not be more than 60mm from the center of the axis of rotation of the shoulder motor. Being greater than 60mm would make the structure collide with the shaft that binds together the two plates of the shoulder link.

The calculations to find the parameters of the spring

The most initial parameter that we were needed to find was the spring constant K. By using the K, we needed to find the Outer diameter of the Spring, the wire diameter that the Spring was to be built from and also the number of coils that were required to form the Spring.



Total weight of the Arm (including payload, motors and all accessories = 650gms
Distance of estimated center of gravity of the arm from O = 250mm

Finding Θ from the above figure:

Θ = tan^-1(50/60)
Or, Θ = 40 degrees

Now, to find out the force we require at point V:

Balancing all forces along the Z-axis, we get:

50 * F * cos40 = 650gms * 250 mm
F = ((650 * 250)/cos40)
Or, F = 21212.8gm
Or, F = 4.5Kg

Now to calculate the spring constant that we need:

Taking stretched spring length = 95mm
We know a spring can expand to a maximum of 170% its original length

Therefore, the un-stretched length(x) of the spring should be:

x * 1.7 = 95
Or, x = 95/1.7 = 56mm

Thus, Change in length, Δmax = 95 – 56 = 39mm

Thus this 39mm has to provide a force of 4.5kg (max)

5kg = 39 * K
or, K = 0.15kg/mm

By using the Spring properties given in the Mechanical Data book, we found out the best dimensions that will cater our needs. The total spring specification put together are the following:

- Type: Tension
- D outer: 6mm
- d wire: 1mm
- Number of turns: 50
- Material: Standard Spring Steel
- Normal length = 50mm<L<56mm
- Expanded length < 100mm

V | Electronic Hardware Design

As shown in our architecture, we were required to build an Electronic hardware that would be able to act as an interface between the Servo Motors of the Robot Arm and the PC. The PC would send instructions to control the servo motors, these instructions would actually be decoded by the Electronic Hardware which will in turn store the instructions and perform the operation in the required time constraint and in the exact protocol as required to make the Arm work in a proper fashion.

Requirements from the Electronic Hardware

- Communication- Communication is an integral part of every application. Similarly our project has serial communication which has been implemented with the help of MAX-232.
- Motor Control- As per the requirements the electronic hardware should be able to control at least five servo motors.
- Debugging- The electronic hardware should provide facility for easy debugging which has been achieved by interfacing a LCD (2*16) with our controller.
- Storage- The electronic hardware must provide some means to store the path data that the robot should be able to follow when switched to playback mode.

Components and Performance Required

PIC 16F877A

Due to the robustness of the design which has been proved for long in the industry, we chose to use PIC 16F877A microcontroller. Moreover we were familiar with architecture. It provides us with every feature that we require to build the interface for our robot.

PIC Microcontroller

Peripheral Interface Controllers (PIC) is a family of microcontrollers by Microchip Technology. PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

The PIC architecture is characterized by its multiple attributes:

- Separate code and data spaces (Harvard architecture)
- A small number of fixed length instructions
- Most instructions are single cycle execution (2 clock cycles, or 4 clock cycles in 8bit models), with one delay cycle on branches and skips
- One accumulator (W0), the use of which (as source operand) is implied (i.e. is not encoded in the opcode)
- All RAM locations function as registers as both source and/or destination of math and other functions.
- A hardware stack for storing return addresses
- A fairly small amount of addressable data space (typically 256 bytes), extended through banking
- Data space mapped CPU, port, and peripheral registers
- The program counter is also mapped into the data space and writable (this is used to implement indirect jumps).

There is no distinction between memory space and register space because the RAM serves the job of both memory and registers, and the RAM is usually just referred to as the register file or simply as the registers.

Data space (RAM)

PICs have a set of registers that function as general purpose RAM. Special purpose control registers for on-chip hardware resources are also mapped into the data space. The addressability of memory varies depending on device series, and all PIC devices have some banking mechanism to extend addressing to additional memory. Later series of devices feature move instructions which can cover the whole addressable space, independent of the selected bank. In earlier devices, any register move had to be achieved via the accumulator.

To implement indirect addressing, a "file select register" (FSR) and "indirect register" (INDF) are used. A register number is written to the FSR, after which reads from or writes to INDF will actually be to or from the register pointed to by FSR. Later devices extended this concept with post- and pre- increment/decrement for greater efficiency in accessing sequentially stored data. This also allows FSR to be treated almost like a stack pointer (SP).

Code space

The code space is generally implemented as ROM, EPROM or flash ROM. In general, external code memory is not directly addressable due to the lack of an external memory interface.

Word size

All PICs handle (and address) data in 8-bit chunks. However, the unit of addressability of the code space is not generally the same as the data space. For example, PICs in the baseline and mid-range families have program memory addressable in the same word size as the instruction width, i.e. 12 or 14 bits respectively. In contrast, in the PIC18 series, the program memory is addressed in 8-bit increments (bytes), which differ from the instruction width of 16 bits.

In order to be clear, the program memory capacity is usually stated in number of (single word) instructions, rather than in bytes.

Stacks

PICs have a hardware call stack, which is used to save return addresses. The hardware stack is not software accessible on earlier devices, but this changed with the 18 series devices.

Hardware support for a general purpose parameter stack was lacking in early series, but this greatly improved in the 18 series, making the 18 series architecture friendlier to high level language compilers.

Instruction set

A PIC's instructions vary from about 35 instructions for the low-end PICs to over 80 instructions for the high-end PICs. The instruction set includes instructions to perform a variety of operations on registers directly, the accumulator and a literal constant or the accumulator and a register, as well as for conditional execution, and program branching.

Some operations, such as bit setting and testing, can be performed on any numbered register, but bi-operand arithmetic operations always involve W (the accumulator), writing the result back to either W or the other operand register.

PIC cores have skip instructions which are used for conditional execution and branching. The skip instructions are 'skip if bit set' and 'skip if bit not set'. Because cores before PIC18 had only unconditional branch instructions, conditional jumps are implemented by a conditional skip (with the opposite condition) followed by an unconditional branch. Skips are also of utility for conditional execution of any immediate single following instruction.

In general, PIC instructions fall into 5 classes:

1. Operation on working register (WREG) with 8-bit immediate ("literal") operand. E.g. movlw (move literal to WREG), andlw (AND literal with WREG). One instruction peculiar to the PIC isretlw, load immediate into WREG and return, which is used with computed branches to produce lookup tables.
2. Operation with WREG and indexed register. The result can be written to either the Working register (e.g. addwf *reg*,w). or the selected register (e.g. addwf *reg*,f).
3. Bit operations. These take a register number and a bit number, and perform one of 4 actions: set or clear a bit, and test and skip on set/clear. The latter are used to perform conditional branches. The usual ALU status flags are available in a numbered register so operations such as "branch on carry clear" are possible.
4. Control transfers. Other than the skip instructions previously mentioned, there are only two: goto and call.
5. A few miscellaneous zero-operand instructions, such as return from subroutine, and sleep to enter low-power mode.

Performance

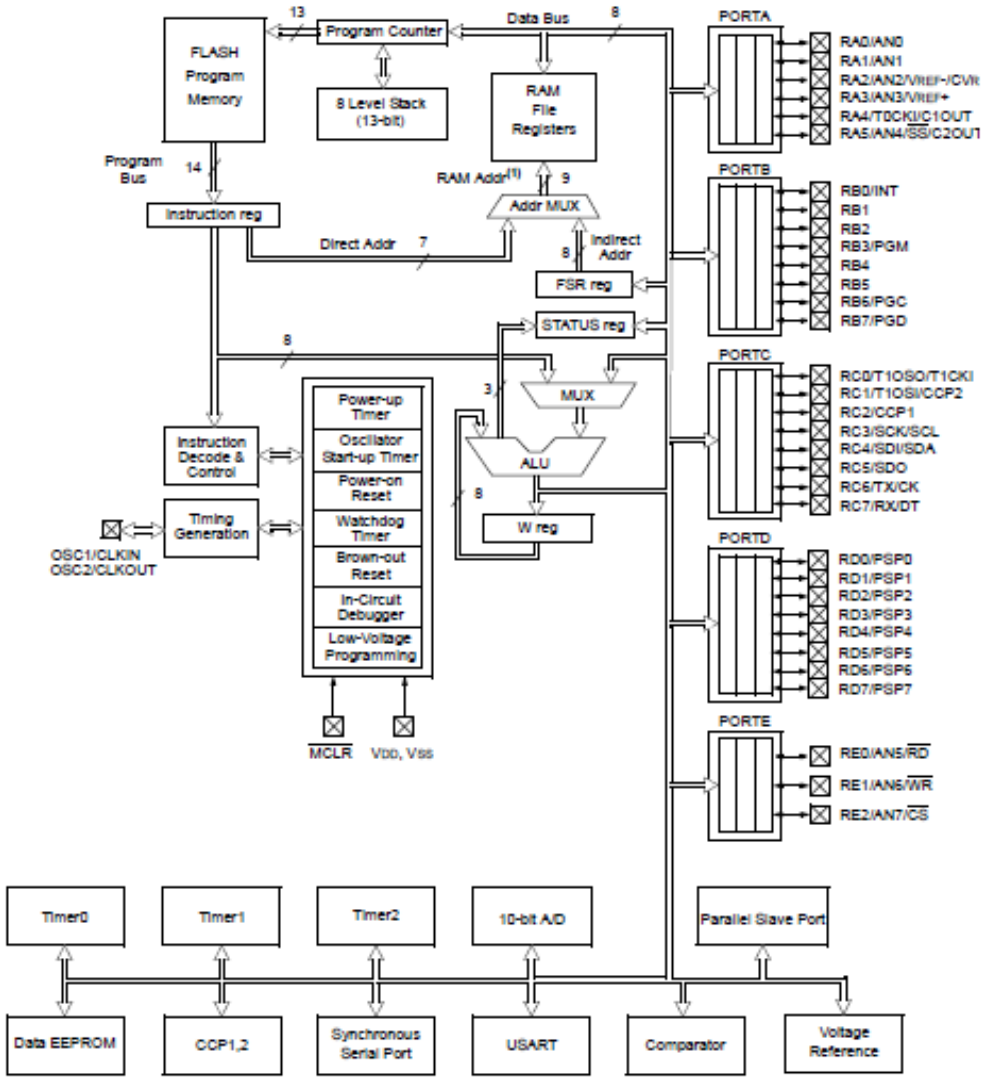
The architectural decisions are directed at the maximization of speed-to-cost ratio. The PIC architecture was among the first scalar CPU designs, and is still among the simplest and cheapest. The Harvard architecture—in which instructions and data come from separate sources - simplify timing and microcircuit design greatly, and this benefits clock speed, price, and power consumption.

The PIC instruction set is suited to implementation of fast lookup tables in the program space. Such lookups take one instruction and two instruction cycles. Many functions can be modelled in this way. Optimization is facilitated by the relatively large program space of the PIC and by the design of the instruction set, which allows for embedded constants.

Execution time can be accurately estimated by multiplying the number of instructions by two cycles; this simplifies design of real-time code. Similarly, interrupt latency is constant at three instruction cycles. External interrupts have to be synchronized with the four clock instruction cycle; otherwise there can be a one instruction cycle jitter. Internal interrupts are already synchronized. The constant interrupt latency allows PICs to achieve interrupt driven low jitter timing sequences. An example of this is a video sync pulse generator. This is no longer true in the newest PIC models, because they have a synchronous interrupt latency of three or four cycles.

PIC16F877a Features:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: 20 MHz clock input
- Up to 8K x 14 words of Flash Program Memory,
- Up to 368 x 8 bytes of Data Memory (RAM),
- Up to 256 x 8 bytes of EEPROM Data Memory
- 5 Ports A,B,C,D and E
- Pin out compatible to other 28-pin or 40/44-pin microcontrollers

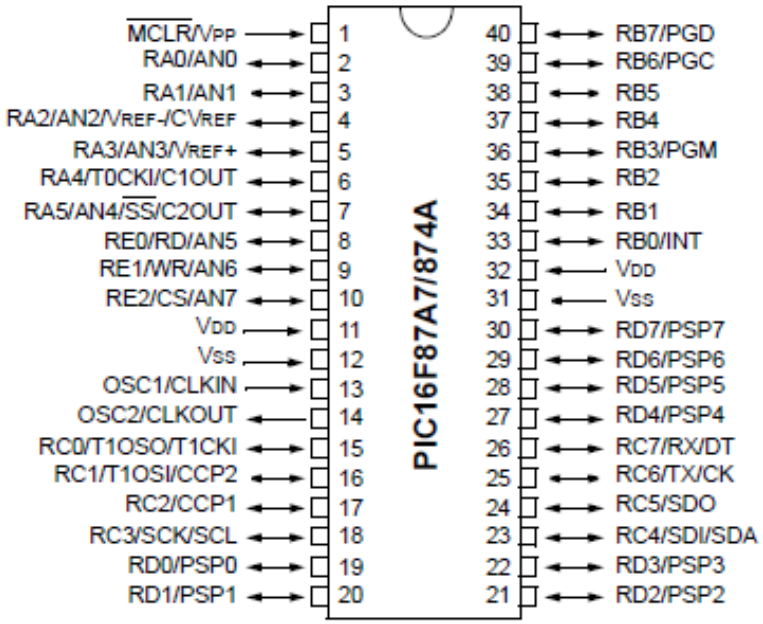


PIC16F877a Controller Architecture

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescalar
- Timer1: 16-bit timer/counter with prescalar,
- Timer2: 8-bit timer/counter with 8-bit period register, prescalar and post scalar
- Two Capture, Compare, PWM modules
- Synchronous Serial Port with :SPI-Serial Peripheral Interface
- Universal Synchronous Asynchronous Receiver Transmitter (USART)
- Parallel Slave Port (PSP)

Pin Diagram of PIC16F877a



Pin Description of 16F877A

Pin Name	DIP PIN #	I/O TYPE	Buffer Type	Description
OSC1/CLKI OSC1 CLKI	13	I	ST/CMOS	Oscillator crystal or external clock input. Oscillator crystal input or external clock source input. ST buffer when configured in RC mode. Otherwise CMOS. External clock source input. Always associated with pin function OSC1
OSC2/CLKOUT OSC2	14	O	-	Oscillator crystal or clock output. Oscillator crystal output. Connects to crystal or resonator in

CLKO				Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
MCLR/VPP MCLR VPP	1	I/P	ST	Master Clear (input) or programming voltage (output). Master Clear (Reset) input. This pin is an active low RESET to the device. Programming voltage input.
				PORTA is a bi-directional I/O port.
RA0/AN0 RA0 AN0	2		TTL	Digital I/O. Analog input 0.
RA1/AN1 RA1 AN1	3		TTL	Digital I/O. Analog input 1.
RA2/AN2/VREF-/CVREF RA2 AN2 VREFCVREF	4		TTL	Digital I/O. Analog input 2. A/D reference voltage (Low) input. Comparator VREF output.
RA3/AN3/VREF+ RA3 AN3 VREF+	5		TTL	Digital I/O. Analog input 3. A/D reference voltage (High) input.
RA4/T0CKI/C1OUT RA4 T0CKI C1OUT	6		ST	Digital I/O – Open drain when configured as output. Timer0 external clock input. Comparator 1 output.
RA5/SS/AN4/C2OUT RA5 SS AN4 C2OUT	7		TTL	Digital I/O. SPI slave select input. Analog input 4. Comparator 2 output.
				PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT RB0 INT	33	I/O I	TTL/ST	Digital I/O. External interrupt.
RB1	34	I/O	TTL	Digital I/O.
RB2	35	I/O	TTL	
RB3/PGM	36	I/O	TTL	

RB3 PGM				Digital I/O. Low voltage ICSP programming enable pin.
RB4	37	I/O	TTL	Digital I/O
RB5	38	I/O	TTL	Digital I/O
RB6/PGC RB6 PGC	39	I/O	TTL/ST	 Digital I/O. In-Circuit Debugger and ICSP programming clock.
RB7/PGD RB7 PGD	40	I/O	TTL/ST	 Digital I/O. In-Circuit Debugger and ICSP programming data.
				PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI RC0 T1OSO T1CKI	15	I/O O I	ST	Digital I/O. Timer1 oscillator output. Timer1 external clock input
RC1/T1OSI/CCP2 RC1 T1OSI CCP2	16	 I/O I I/O	ST	Digital I/O. Timer1 oscillator input. Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1 RC2 CCP1	17	I/O I/O	ST	Digital I/O. Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL RC3 SCK SCL	18	 I/O I/O I/O	ST	Digital I/O. Synchronous serial clock input/output for SPI mode. Synchronous serial clock input/output for I2C mode
RC4/SDI/SDA RC4 SDI SDA	23	 I/O I I/O	ST	Digital I/O. SPI data in. I2C data I/O.
RC5/SDO RC5 SDO	24	 I/O O	ST	Digital I/O. SPI data out
RC6/TX/CK RC6 TX CK	25	 I/O O I/O	ST	Digital I/O. USART asynchronous transmit. USART 1 synchronous clock.
RC7/RX/DT RC7 RX DT	26	 I/O I I/O	ST	Digital I/O. USART asynchronous receive. USART synchronous data.
				PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor

				bus.
RD0/PSP0	19		ST/TTL	
RD0		I/O		
PSP0		I/O		Digital I/O.
				Parallel Slave Port data.
RD1/PSP1	20		ST/TTL	
RD1		I/O		
PSP1		I/O		
				Digital I/O.
RD2/PSP2	21		ST/TTL	Parallel Slave Port data.
RD2		I/O		
PSP2		I/O		Digital I/O.
				Parallel Slave Port data.
RD3/PSP3	22		ST/TTL	
RD3		I/O		
PSP3		I/O		Digital I/O.
RD4/PSP4	27		ST/TTL	Parallel Slave Port data.
RD4		I/O		
PSP4		I/O		Digital I/O.
				Parallel Slave Port data.
RD5/PSP5	28		ST/TTL	
RD5		I/O		
PSP5		I/O		Digital I/O.
				Parallel Slave Port data.
RD6/PSP6	29		ST/TTL	
RD6		I/O		
PSP6		I/O		Digital I/O.
				Parallel Slave Port data.
RD7/PSP7	30		ST/TTL	
RD7		I/O		
PSP7		I/O		Digital I/O.
				Parallel Slave Port data.
				PORTE is a bi-directional I/O port.
RE0/RD/AN5	8		ST/TTL	Digital I/O.
RE0		I/O		Read control for parallel slave port.
RD		I		Analog input 5.
AN5		I		
				Digital I/O.
RE1/WR/AN6	9		ST/TTL	Write control for parallel slave port.
RE1				Analog input 6.
WR		I/O		S
AN6		I		Digital I/O.
		I		Chip select control for parallel slave
RE2/CS/AN7	10		ST/TTL	port.
RE2		I/O		Analog input 7.
CS		I		
AN7		I		

Pins of PIC16F877a used in our project

We used the following pins for the given purposes in our project.

- Port A (Pins 2-7) are connected to the servo motors.
- Two pins of port E (pins 8&9) are connected to the servo motors.
- Pins 13 and 14 are connected to the crystal oscillator.
- Pins 12, 18 and 23 are connected to the EEPROM.
- Pins 25 and 26 are connected to the serial communicator MAX232.
- Six pins of port B (33-38) are connected to the LCD.

EEPROM 24C64

The Microchip Technology Inc. 24LC64 (24XX64*) is a 64 Kbit Electrically Erasable PROM that we have interfaced with the microcontroller. The device is organized as eight blocks of 1K x 8-bit memory with a 2-wire serial interface. Low voltage design permits operation down to 1.8V with standby and active currents of only 1 μA and 1 mA respectively.

It has been developed for advanced, low power applications such as personal communications or data acquisition. The 24XX64 also has a page-write capability for up to 32 bytes of data. Functional address lines allow up to eight devices on the same bus, for up to 512 Kbits address space. It has been use since:

- It is available at low prices.
- It is easily available.
- It makes use of the SPI, i.e. Serial Peripheral Interface. Rather than using any hardware modules for communications, this system implements the SPI bus through software so that any I/O port can be used for communicating with EEPROMs.

General Description of an EEPROM

EEPROM stands for **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory and is a type of non-volatile memory used in computers and other electronic devices to store small amounts of data that must be saved when power is removed, e.g., calibration tables or device configuration.

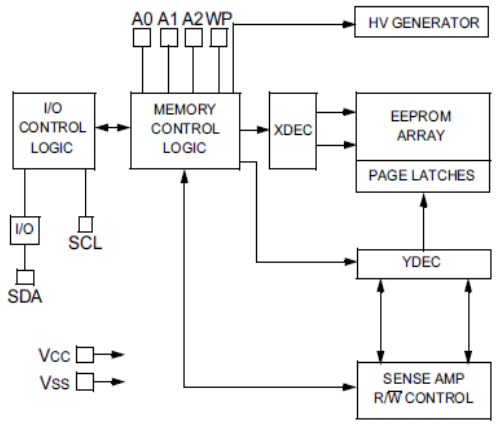
EEPROM is user-modifiable read-only memory (ROM) that can be erased and reprogrammed (written to) repeatedly through the application of higher than normal electrical voltage generated externally or internally in the case of modern EEPROMs. EPROM usually must be removed from the device for erasing and programming, whereas EEPROMs can be programmed and erased in circuit. Originally, EEPROMs were limited to single byte operations which made them slower, but modern EEPROMs allow multi-byte page operations. It also has a limited life - that is, the number of times it could be reprogrammed was limited to tens or hundreds of thousands of times. That limitation has been extended to a million write operations in modern EEPROMs. In an EEPROM that is frequently reprogrammed while the computer is in use, the life of the EEPROM can be an important design consideration. It is for this reason that EEPROMs were used for configuration information, rather than random access memory.

Features of 24C64 EEPROM

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

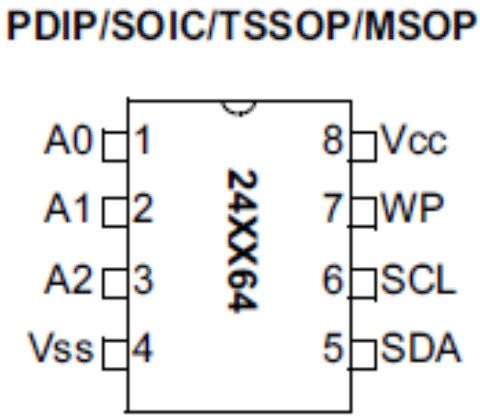
BLOCK DIAGRAM OF 24C64 EEPROM

The functional block diagram of the EEPROM is shown below:



Pin Diagram 24C64 EEPROM

The 24XX64 is available in the standard 8-pin PDIP, surface mount SOIC, TSSOP and MSOP packages.



Interfacing of the 24C64EEPROM with PIC 16F877A

- SCL (Pin 6) of EEPROM array receives the clock from SCK (Pin 18) of the microcontroller.
- SDA (Pin 5) of EEPROM gets data from SDI (Pin 23) of the microcontroller.

MAX 232

The MAX 232 is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

The drivers provide RS-232 voltage level outputs (approx. ± 7.5 V) from a single + 5 V supply via on-chip charge pumps and external capacitors. This makes it useful for implementing RS-232 in devices that otherwise do not need any voltages outside the 0 V to + 5 V range, as power supply design does not need to be made more complicated just for driving the RS-232 in this case.

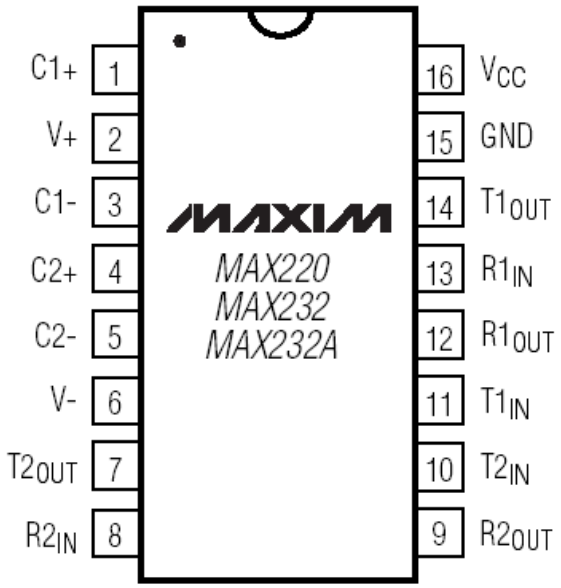
The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to standard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.

The later MAX232A is backwards compatible with the original MAX232 but may operate at higher baud rates and can use smaller external capacitors – 0.1 μF in place of the 1.0 μF capacitors used with the original device.

The newer MAX3232 is also backwards compatible, but operates at a broader voltage range, from 3 to 5.5 V.

Pin Diagram of MAX 232

The Pin diagram of the MAX232 logic level shifter is shown below. This has been obtained from the datasheet.



MAX232(A) DIP Package Pin Layout Description

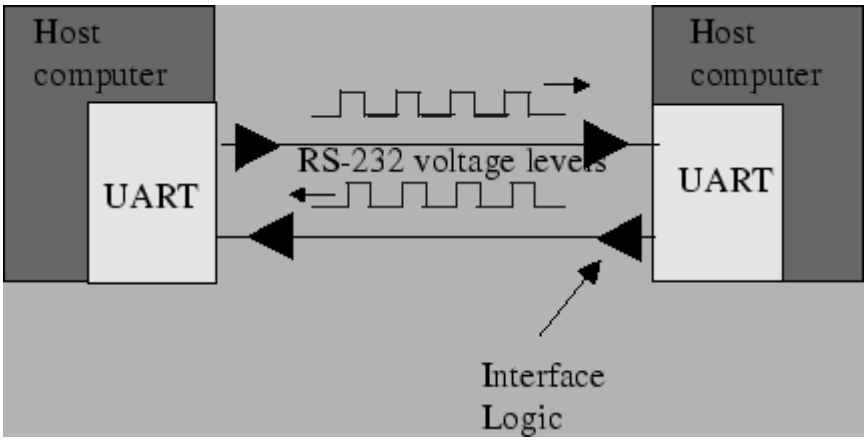
Nbr	Name	Purpose	Signal Voltage	Capacitor Value MAX232	Capacitor Value MAX232A
1	C1+	+ connector for capacitor C1	capacitor should stand at least 16V	1µF	100nF
2	V+	output of voltage pump	+10V, capacitor should stand at least 16V	1µF to VCC	100nF to VCC
3	C1-	- connector for capacitor C1	capacitor should stand at least 16V	1µF	100nF
4	C2+	+ connector for capacitor C2	capacitor should stand at least 16V	1µF	100nF
5	C2-	- connector for capacitor C2	capacitor should stand at least 16V	1µF	100nF
6	V-	output of voltage pump / inverter	-10V, capacitor should stand at least 16V	1µF to GND	100nF to GND
7	T2out	Driver 2 output	RS-232		
8	R2in	Receiver 2 input	RS-232		
9	R2out	Receiver 2 output	TTL		
10	T2in	Driver 2 input	TTL		
11	T1in	Driver 1 input	TTL		
12	R1out	Receiver 1 output	TTL		
13	R1in	Receiver 1 input	RS-232		
14	T1out	Driver 1 output	RS-232		
15	GND	Ground	0V	1µF to VCC	100nF to VCC
16	VCC	Power supply	+5V	see above	see above

V+(2) is also connected to VCC via a capacitor (C3). V-(6) is connected to GND via a capacitor (C4). And GND(15) and VCC(16) are also connected by a capacitor (C5), as close as possible to the pins.

The RS-232 Serial Protocol

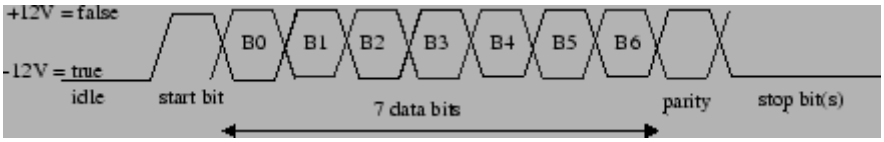
The RS-232 serial communication protocol is a standard protocol used in asynchronous serial communication. It is the primary protocol used over modem lines. It is the protocol used by the MicroStamp11 when it communicates with a host PC.

Figure shows the relationship between the various components in a serial ink. These components are the UART, the serial channel, and the interface logic. An interface chip known as the **universal asynchronous receiver/transmitter** or **UART** is used to implement serial data transmission. The UART sits between the host computer and the serial channel. The serial channel is the collection of wires over which the bits are transmitted. The output from the UART is a standard TTL/CMOS logic level of 0 or 5 volts. In order to improve bandwidth, remove noise, and increase range, this TTL logical level is converted to an RS-232 logic level of $\lfloor -12$ or $\lfloor +12$ volts before being sent out on the serial channel. This conversion is done by the interface logic shown in figure.



Asynchronous (RS-232) serial link

A frame is a complete and nondivisible packet of bits. A frame includes both information (e.g., data and characters) and overhead (e.g., start bit, error checking and stop bits). In asynchronous serial protocols such as RS-232, the frame consists of one start bit, seven or eight data bits, parity bits, and stop bits. A timing diagram for an RS-232 frame consisting of one start bit, 7 data bits, one parity bits and two stop bits is shown below in figure . Note that the exact structure of the frame must be agreed upon by both transmitter and receiver before the comm-link must be opened.



RS-232 Frame (1 start bit, 7 data bits, 1 parity bits, and 2 stop bits)

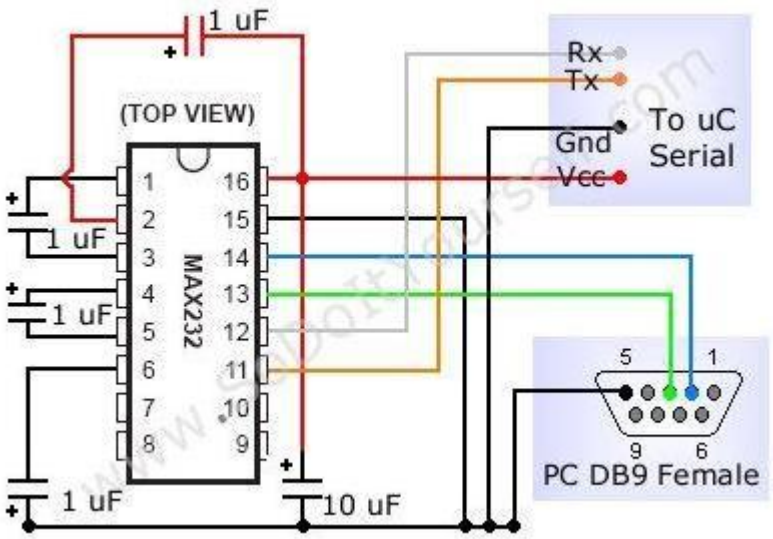
Most of the bits in a frame are self-explanatory. The start bit is used to signal the beginning of a frame and the stop bit is used to signal the end of a frame. The only bit that probably needs a bit of explanation is the parity bit. Parity is used to detect transmission errors. For even parity checking, the number of 1's in the data plus the parity bit must equal an even number. For odd parity, this sum must be an odd number. Parity bits are used to detect errors in transmitted data. Before sending out a frame, the transmitter sets the parity bit so that the frame has either even or odd parity. The receiver and transmitter have already agreed upon which type of parity check (even or odd) is being used. When the frame is received, then the receiver checks the parity of the received frame. If the

parity is wrong, then the receiver knows an error occurred in transmission and the receiver can request that the transmitter re-send the frame.

In cases where the probability of error is extremely small, then it is customary to ignore the parity bit. For communication between the MicroStamp11 and the host computer, this is usually the case and so we ignore the parity bit.

The bit time is the basic unit of time used in serial communication. It is the time between each bit. The transmitter outputs a bit, waits one bit time and then outputs the next bit. The start bit is used to synchronize the transmitter and receiver. After the receiver senses the true-false transition in the start bit, it waits one half bit time and then starts reading the serial line once every bit time after that. The baud rate is the total number of bits (information, overhead, and idle) per time that is transmitted over the serial link. So we can compute the baud rate as the reciprocal of the bit time.

Connections of the MAX232 with the DB9 Female Connector



Servo Motors

Servo motors are electro-mechanical devices which provide good torque. If you open a servo motor you will see a simple DC motor connected to a potentiometer and as motor rotates output value of potentiometer also changes. A servo motor has three wires first one for voltage, second for data and third for ground. Generally value for voltage is 5V and the input signal varies from 4 to 6V. A servo motor expects a pulse every 20ms and width of the pulse varies from .8ms to 2.5ms. 0.8ms for maximum rotation to -90 and 2.5ms for +90, these values are specific for the motors we have used.

I\O Port for Motor Control

For controlling a servo no motor driving IC's are not required. It can be directly interfaced with the controller's any I\O port. We have connected motor with PORT A of controller.

- Base Servo: PIN A0
- Shoulder Servo: PIN A1
- Elbow Servo: PIN A2
- Wrist Servo: PIN A3
- Gripper Servo: PIN A5

Power Supply

Two power supplies have been used:

1. 12 v C gate hardware adaptor for servos.
2. 12 v dc battery for other components.

We figured out that the current drawn by the servo motors combined together was quite high (almost 3 amperes under full load) however the magnitude of the current drawn varied heavily as we switched the operation from one motor to the other. this could have triggered a controller hang due to inherent security measures that are built into the controller like BOR etc. this would have hampered the normal operation of a robot heavily and at times it might even have been unusable. So by using a different power circuitry for the motors and making sure that there are no common Vcc or GND between the motors and rest of the circuit we totally eradicated any possibility of the problem.

Voltage Regulator 7805

Voltage Regulator IC-7805

A **linear regulator** is a voltage regulator based on an active device (such as a bipolar junction transistor, field effect transistor or vacuum tube) operating in its "linear region" (in contrast, a switching regulator is based on a transistor forced to act as an on/off switch) or passive devices like zener diodes operated in their breakdown region. The regulating device is made to act like a variable resistor, continuously adjusting a voltage divider network to maintain a constant output voltage. It is very inefficient compared to a switched-mode power supply, since it sheds the difference voltage by dissipating heat.

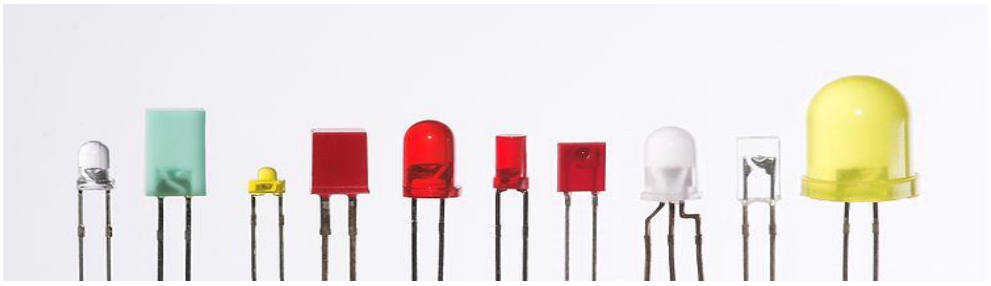
All linear regulators require an input voltage at least some minimum amount higher than the desired output voltage. That minimum amount is called the dropout voltage. For example, a common regulator such as the 7805 has an output voltage of 5V, but can only maintain this if the input voltage remains above about 7V, before the output voltage begins sagging below the rated output. Its dropout voltage is therefore 7V - 5V = 2V. Load that it can take maximum is 1 ampere.

7805 is connected at one end to the dc input jack and at the output to two rectifier capacitors and one protection diode.

There are two surface of the DC jack; Outer surface and the inner surface. Conventionally, the inner surface of the DC jack is Vcc and outer is GND and the connections in the PCB are made accordingly. However if accidently the polarities of the dc jack are inverted, it is the protection diode that ensures no power is fed to the circuit by a reverse bias condition.

LED (Light Emitting Diode)

A light-emitting diode is a semiconductor light source. LEDs are used as indicator lamps in many devices and are increasingly used for other lighting.

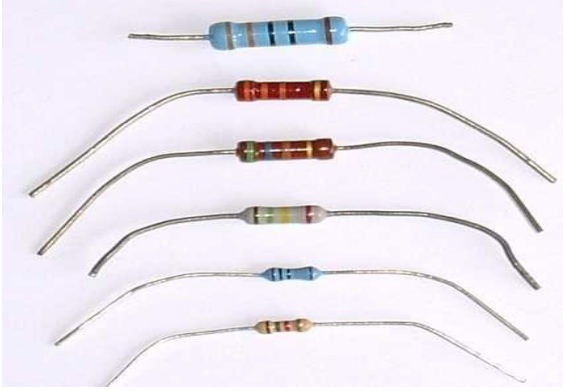


LED

When a light-emitting diode is forward biased (switched on), electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence and the colour of the light (corresponding to the energy of the photon) is determined by the energy gap of the semiconductor.

Resistors

A resistor is a two-terminal passive electronic component which implements electrical resistance as a circuit element. When a voltage V is applied across the terminals of a resistor, a current I will flow through the resistor in direct proportion to that voltage. This constant of proportionality is called conductance, G. The reciprocal of the conductance is known as the resistance R, since, with a given voltage V, a larger value of R further "resists" the flow of current I as given by Ohm's law: V=IR



Practical resistors can be made of various compounds and films, as well as resistance wire (wire made of a high-resistivity alloy, such as nickel-chrome). Resistors are also implemented within integrated circuits, particularly analog devices, and can also be integrated into hybrid and printed circuits.

The resistors we have used are of 1K,8K and 10K.

Capacitors

A capacitor is a passive electronic component consisting of a pair of conductors separated by a dielectric (insulator). When there is a potential difference (voltage) across the conductors, a static electric field develops across the dielectric, causing positive charge to collect on one plate and negative charge on the other plate. Energy is stored in the electrostatic field. An ideal capacitor is characterized by a single constant value, capacitance, measured in farads. This is the ratio of the electric charge on each conductor to the potential difference between them.

Capacitors are widely used in electronic circuits for blocking direct current while allowing alternating current to pass, in filter networks, for smoothing the output of power supplies, in the resonant circuits that tune radios to particular frequencies and for many other purposes.

The capacitance is greatest when there is a narrow separation between large areas of conductor, hence capacitor conductors are often called "plates", referring to an early means of construction. In practice the dielectric between the plates passes a small amount of leakage current and also has an electric field strength limit, resulting in a breakdown voltage, while the conductors and leads introduce an undesired inductance and resistance. The capacitor being used in our project is ceramic

capacitor. In electronics, a ceramic capacitor is a ***capacitor*** constructed of alternating layers of ***metal*** and ***ceramic***, with the ceramic material acting as the ***dielectric***. The ***temperature coefficient*** depends on whether the dielectric is ***Class 1*** or ***Class 2***. A ceramic capacitor (especially the class 2) often has high ***dissipation.***

A ceramic capacitor is a two-terminal, non-polar device. The classical ceramic capacitor is the "disc capacitor". This device pre-dates the transistor and was used extensively in vacuum-tube equipment (e.g., radio receivers) from about 1930 through the 1950s, and in discrete transistor equipment from the 1950s through the 1980s. As of 2007, ceramic disc capacitors are in widespread use in electronic equipment, providing high capacity and small size at low price compared to other low value capacitor types.

Ceramic capacitors come in various shapes and styles, including:

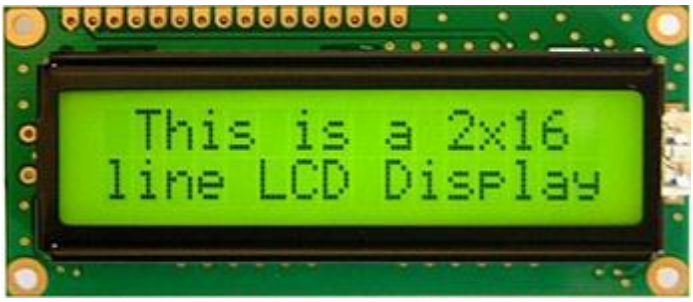
- Disc, resin coated, with through-holes leads
- Multilayer rectangular block, surface mount.

The capacitors that we have used are:

1. Electrolytic capacitors(4)-100 µF
2. Ceramic capacitors(8)-22 Pf

LCD (Liquid Crystal Display)

An LCD is a small low cost display. It is easy to interface with a micro-controller because of an embedded controller (the black blob on the back of the board).This controller is standard across many displays (HD 44780) which means many micro-controllers have libraries that make displaying messages as easy as a single line of code.



A **liquid crystal display** (**LCD**) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). LCDs do not emit light directly.

They are used in a wide range of applications, including computer monitors, television, instrument panels, aircraft cockpit displays, signage, etc. They are common in consumer devices such as video players, gaming devices, clocks, watches, calculators, and telephones. LCDs have displaced cathode ray tube (CRT) displays in most applications. They are usually more compact, lightweight, portable, less expensive, more reliable, and easier on the eyes. They are available in a wider range of screen sizes than CRT and plasma displays, and since they do not use phosphors, they cannot suffer image burn-in.

LCDs are more energy efficient and offer safer disposal than CRTs. Its low electrical power consumption enables it to be used in battery-powered electronic equipment. It is an electronically-modulated optical device made up of any number of pixels filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in colour or monochrome.

The Design Process

We knew we were good to build our PCB design once our simulations were run successfully. We chose EAGLE™ to be our design platform since it is very much flexible, comes with a huge library of predefined components. And in cases where a predefined library is not available, in most of the cases, the device manufacturer provides a downloadable library of its own devices. Very few other CAD tools come with such an exhaustive library.

This becomes a primary reason why this tool is used by students, hobbyists and professional alike. The tool is easy to learn, intuitive and it becomes really easy to implement circuits. This leads to a higher productivity and also a flexible approach to learn and implement CAD. The created files may be exported to be viewed on any Gerber viewer. Also the Drill data, Gerber and other information that are generated can directly be fed to a PCB printing device to print our PCB.

EAGLE is not a freeware. However there exists a trial version that has some limitations on the size of the board and restricts commercial application. This however suited us best, since our complexity involved was fairly small and our design fitted well within the boundaries of the trial version.

The tool may be obtained from the website www.cadsoftusa.com.

The tool offers two modes of the design process, viz. **I. Schematic** and **II. Layout**

- **Schematic:** The schematic is the logical drawing of the circuit as we draw them on paper. This type of drawing actually represents the logical connections between various connections. Although this has got nothing to do with the actual PCB design, this a critical task since the schematic is used to generate rat nests which are used to actually draw the tracks on the layout. This helps avoid any sort of improper connections in the PCB design which may be unwanted or accidental.

- **Layout:** The layout is the drawing of the PCB in exactly the way we see it printed on the Board. The layout takes into consideration the exact footprints of all the components, i.e. how much area on the surface of the board they will take up when placed actually. This is essential because it is not possible to place two components on the same region of space. This is because it is physically impossible to overlap to physical components. The footprint of the components, i.e. the area they cover up and also the exact places we need to drill into to place the component are found in the vast and exhaustive library of EAGLE itself. Thus it may be seen that the vast and exhaustive library of EAGLE saves us the hard work of creating libraries and components on our own. This approach enables us to achieve zero-hassle in building the PCB's.

Our schematic uses the following components from the Library in the design:

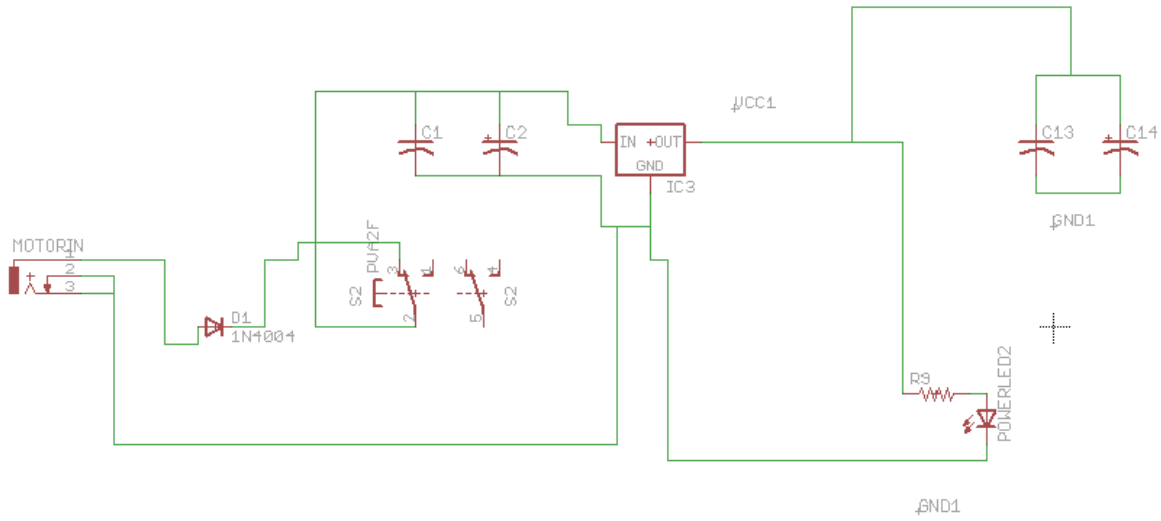
- Microcontroller - PIC16F877P
- Resistances - R-US_0207/10 (R-US_), Library: RCL
- Reset Button - PVA1R, Library: switch_misc
- Oscillator - CRYSTALHC18U-V (CRYSTAL)
- Motor/LCD Connectors - MA08-1W (Con-lstb)
- LEDs - LED3MM(LED)
- Electrolytic Capacitors - CPOL-USE5-6 (CPOL-US) (rcl)
- Max232- MAX232- DIL40 (maxim)
- Potentiometer - TRIM_EU-B25P (TRIM_EU-) (pot)
- Power jack - DCJ0303 (con-jack)

- Power switch - PVA2F (PVA2) (switch-misc)
- DB9 Connector - F09VPS (F09?S) (con-subd)
- Voltage Regulator - 78MXXL (v-reg)
- Diode - 1N4004 (diode)
- EEPROM - 24C64P (24*) (DIL8) (Microchip)

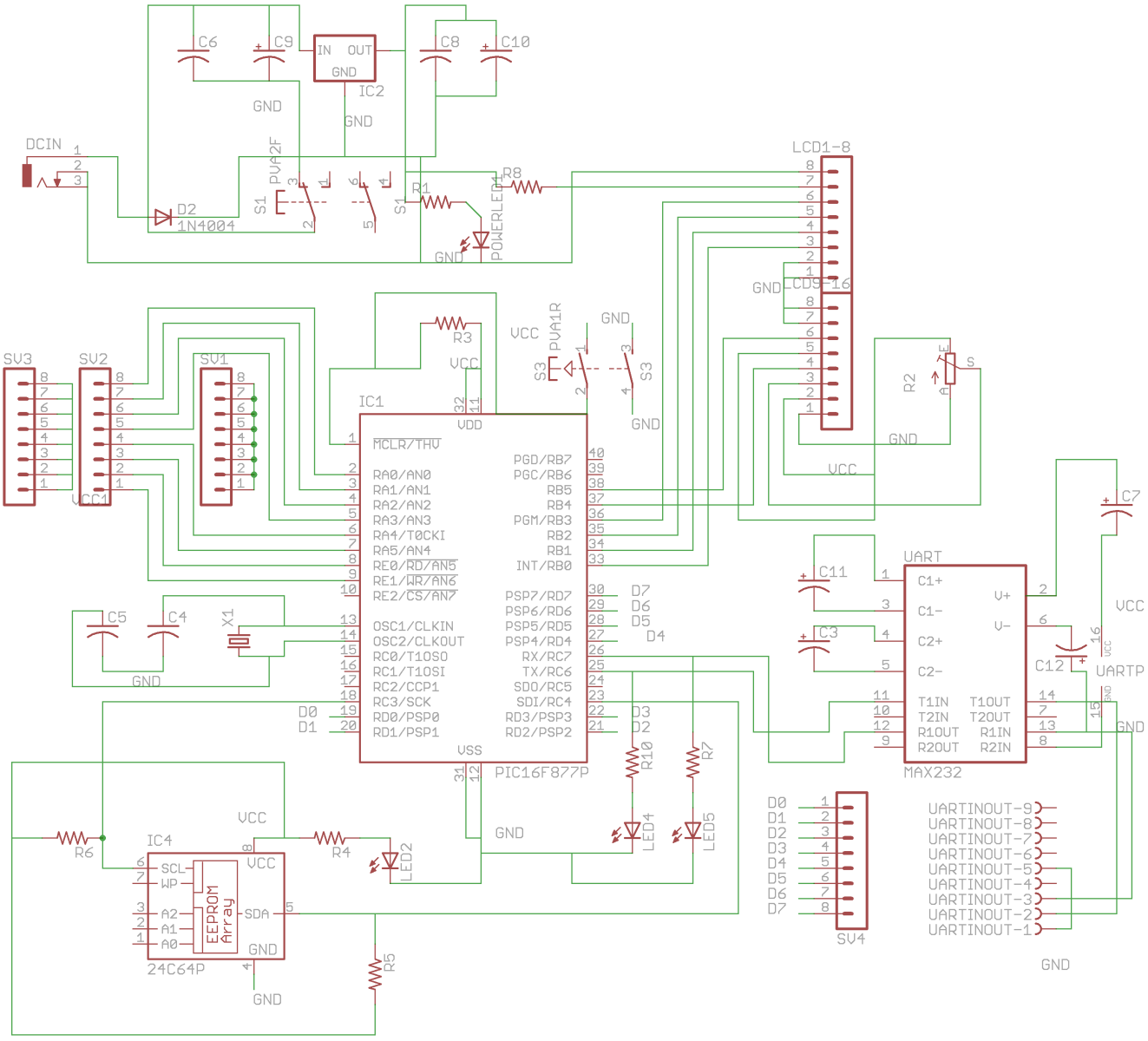
The Design Process

All the above libraries were provided either by CadSoft itself or by the Device Manufacturer or in a very few cases by some other 3rd party vendor. They were used as much as required without a limitation. The libraries are royalty free and thus bear no issues with personal usage of any kind. Amongst the components that are used above, some of them were not even used exactly in the main board itself, but however they were used because they bear a similar footprint to what we actually require in our case.

We turned the DRC off while building our schematic since sometimes it becomes too obvious that the small errors may be easily neglected. Usually the errors comes in forms that $NETX is not connected to $NETY. This primarily comes due a grid size bug in EAGLE, and was handled with extreme caution. Usually this leads to no effects on the actual design of the schematic.



Motor Power Supply Circuit



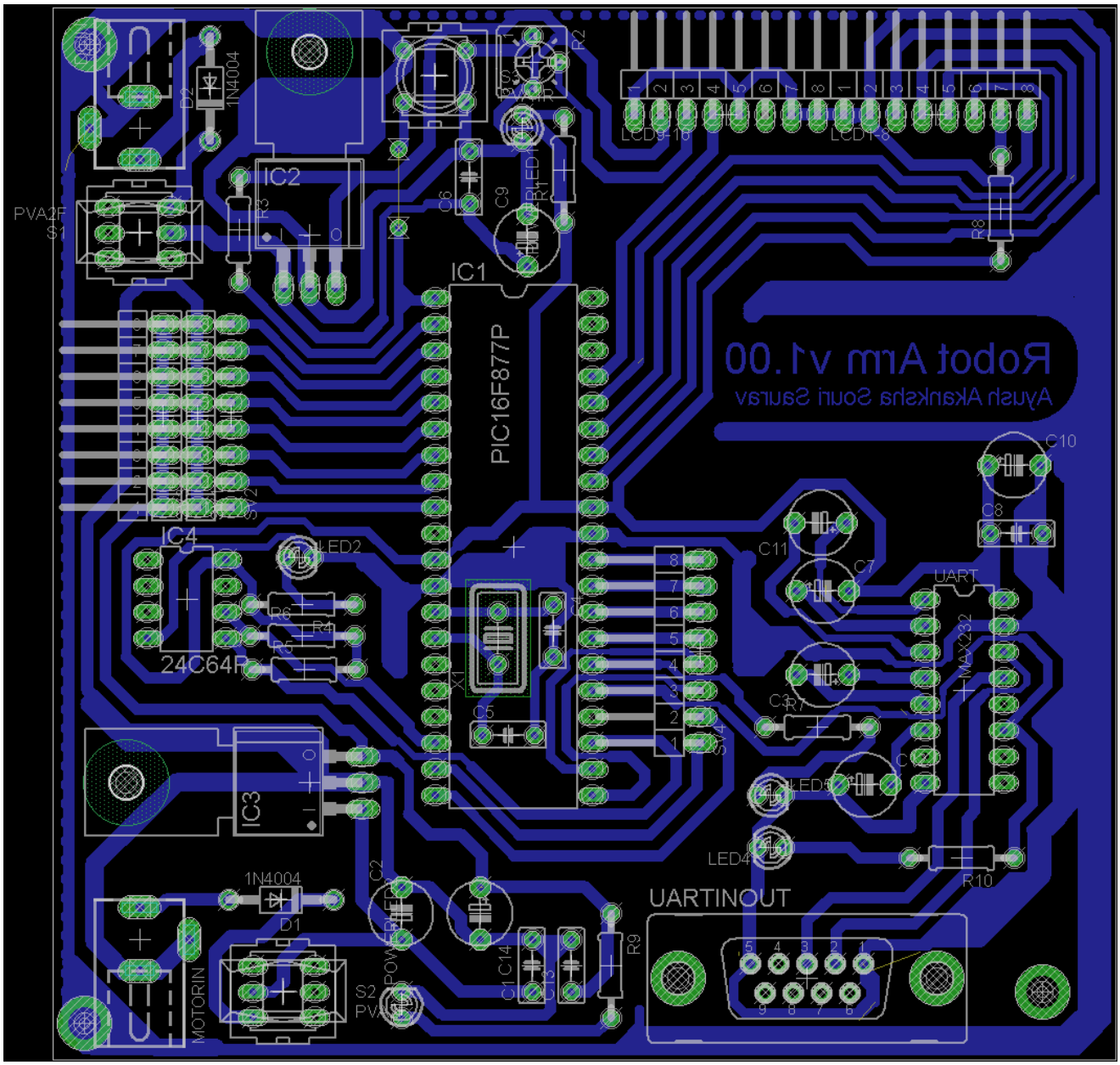
The Rest of the Schematic

Once the schematic was connected entirely, it was reviewed by each of our team members. We were pretty sure that the board will work just fine since we have been pretty successful in the simulations part, and also we had a long term background in working with this specific microcontroller. We were somewhat aware of all the problems that might be possible and took the necessary precautions at all stages to overcome them.

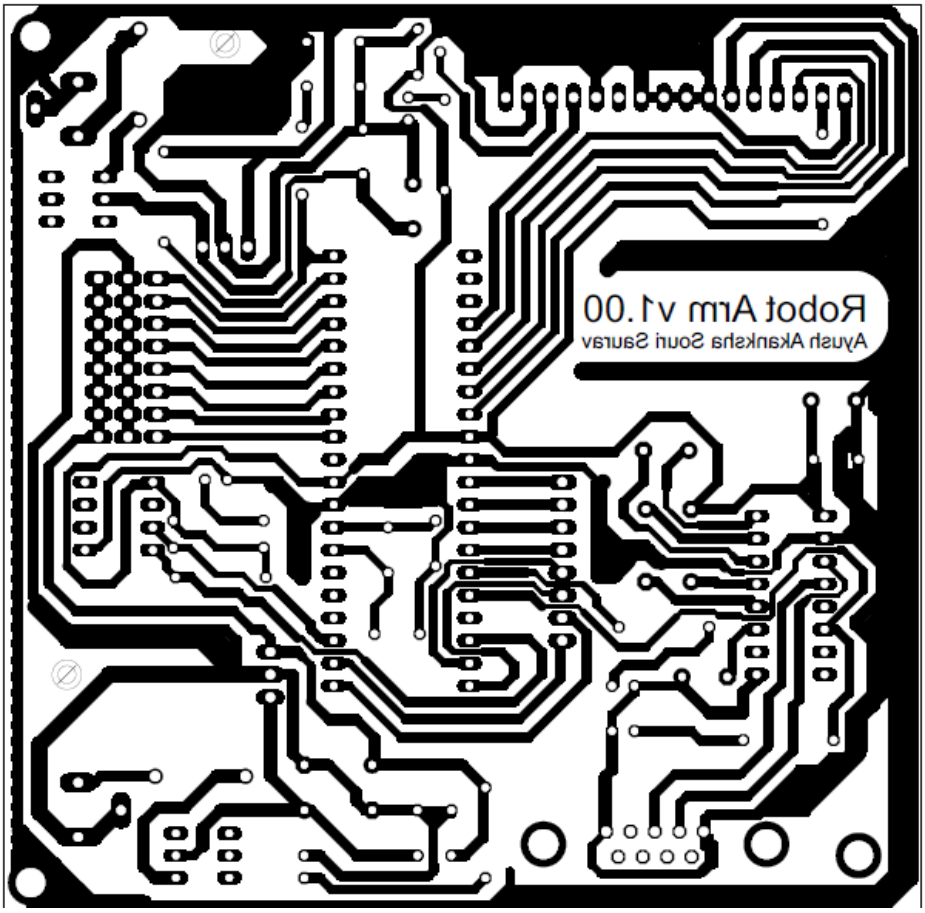
The layout of the PCB was being designed finally. This was a time taking process that had to be handled very carefully. Our intention was to create a home-grown PCB that was single sided, with as little jumpers as possible.

We kept the track widths all between 0.016 to 0.066 mils. The track width was kept small in regions where track congestion was very high. Otherwise we maintained a higher value of the track width. This was to ensure the Robustness of the PCB and avoid damage due to wear and tear or prolonged usage. It is a fact that over a prolonged period of usage, the copper tends to come apart from the epoxy surface. This had to be a factor that depends on the quality of the PCB. Now although our PCB had a proper Epoxy type material, we maintained a higher track width to avoid any possible problem that we may encounter at a later stage after prolonged usage.

The Layout design process took roughly about 9hours. The result was impressive with only 2 jumpers on our entire circuit.



The Final PCB Layout



The Butter Paper Outline for the Screen Print process

We also did take care to create proper Ground Planes wherever there was an opportunity to do so. There is much more to discuss than can be covered here, but we'll highlight some of the key features and encourage the reader to pursue the subject in greater detail.

A ground plane acts as a common reference voltage, provides shielding, enables heat dissipation, and reduces stray inductance (but it also increases parasitic capacitance). While there are many advantages to using a ground plane, care must be taken when implementing it, because there are limitations to what it can and cannot do.

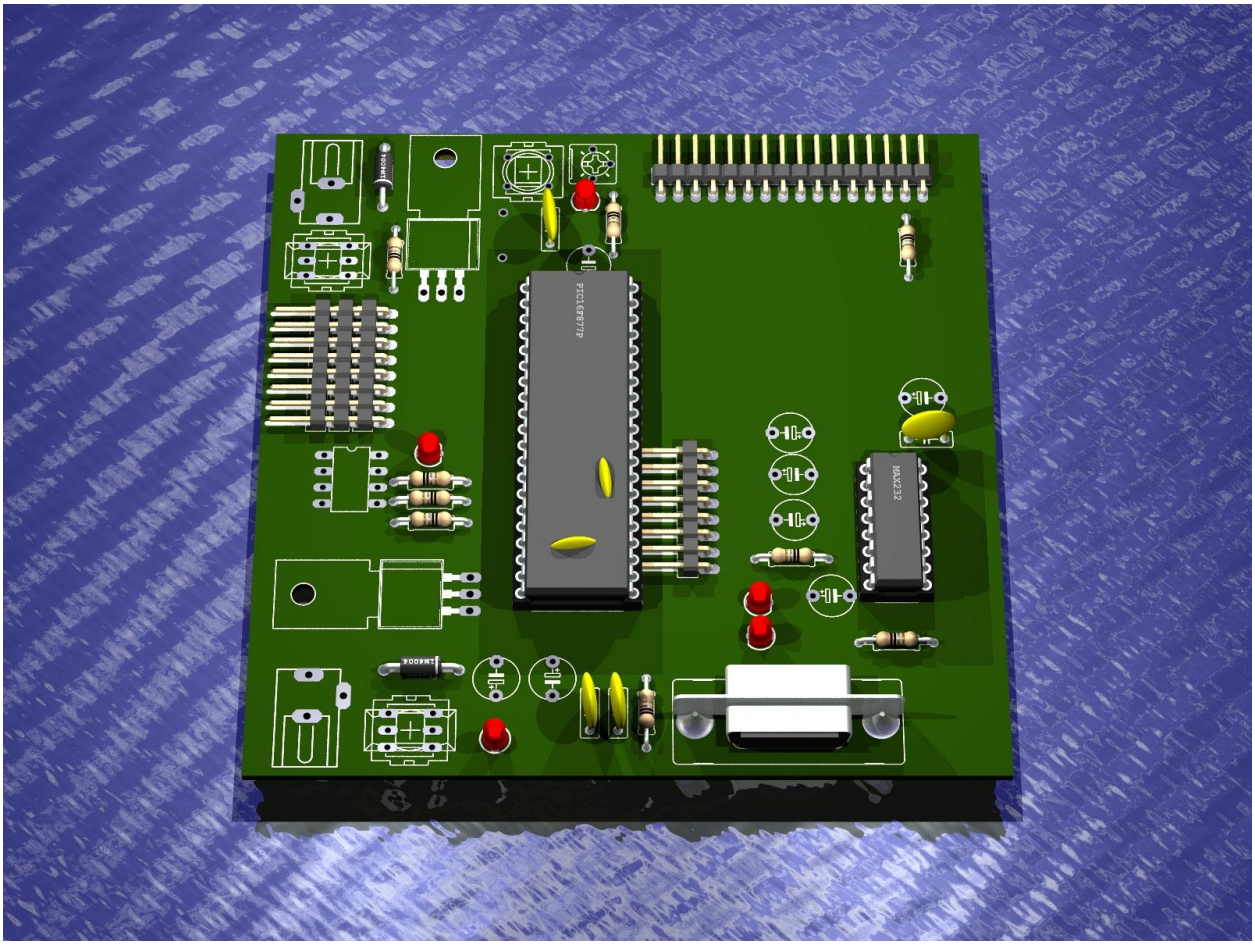
Ideally, one layer of the PCB should be dedicated to serve as the ground plane. Best results will occur when the entire plane is unbroken. Resist the temptation to remove areas of the ground plane for routing other signals on this dedicated layer. The ground plane reduces trace inductance by magnetic-field cancellation between the conductor and the ground plane. When areas of the ground plane are removed, unexpected parasitic inductance can be introduced into the traces above or below the ground plane.

Because ground planes typically have large surface and cross-sectional areas, the resistance in the ground plane is kept to a minimum. At low frequencies, current will take the path of least resistance, but at high frequencies current follows the path of least impedance.

Analog and digital circuitry, including grounds and ground planes, should be kept separate when possible. Fast-rising edges create current spikes flowing in the ground plane. These fast current spikes create noise that can corrupt analog performance. Analog and digital grounds (and supplies) should be tied at one common ground point to minimize circulating digital and analog ground currents and noise.

Using 3d rendering to find misplaced components in the board

Just after the layout was made, it was checked against any misplaced component that may obstruct the proper working/ interaction with another component. We used Eagle 3d to generate the POV data and used the POV-Ray application to render the 3-d image of our board before actually building it.



EAGLE-3D and POV Ray Generated 3d Image of our PCB

The Fabrication process

Once the Design has been done completely, it had to be made ready so that it was ready for the silkscreen printing process on the PCB.

All unwanted layers in the design e.g. Component Layer, Masking Layer, Unconnected Connections layer, Ratsnests etc. were turned off leaving behind only the tracks and drills.

This was exported to a Monochrome design on an A4 .pdf document. The size of the paper at all stages was crucial because it had to be exactly the same as in the software. Any automatic/manual scaling would make our design inconsistent to what we actually require. All footprint sizes, track widths will change leaving behind the design unusable.

This is the reason why we always have to set scaling to "Off" while printing the circuit. The circuit had to be finally printed in a butter paper.

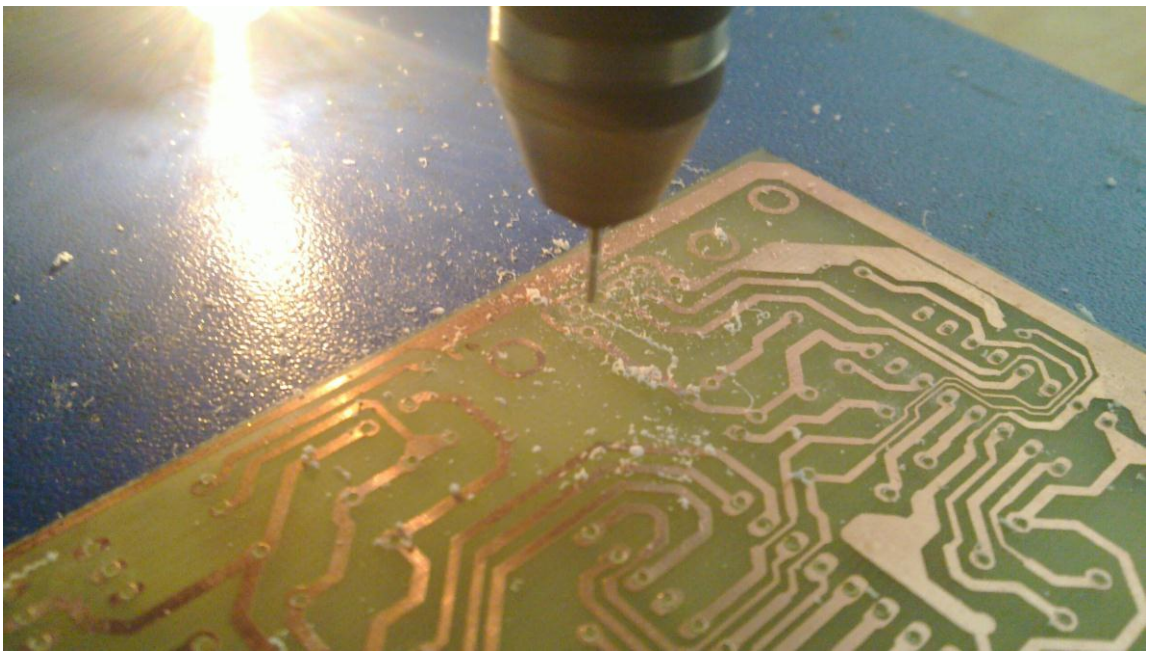
Cutting the Copper Clad Board before the silk screen print

The butter paper design had to be given to a Silk Screen printer to get the artwork printed on the copper layer of the Copper Board. Once the design was being transferred from the butter paper to the Copper Layer of the PCB, the ink covered the areas of the PCB where we intended to get our tracks and drills.

- **Etching:** The ink deposited on top of the copper layer of the PCB actually helps protect the underlying copper from reacting with the Etching solution (FeCl3). The PCB is immersed into a FeCl3 solution that is ideally lukewarm and has got some dilute HCl mixed into the solution. The entire thing is shaken slowly for about half an hour until all the excess copper was totally washed away. The ink deposits were then washed away by using a detergent bar, very carefully, not to disturb any pads or tracks. Finally the PCB was etched and ready for drilling and soldering.

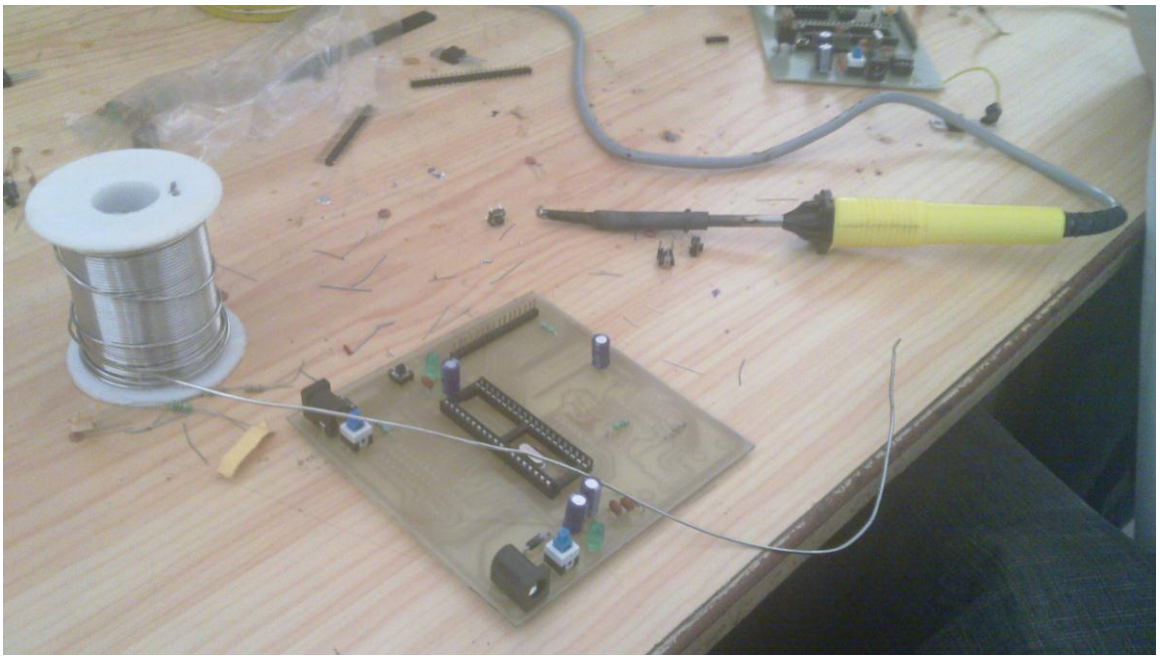
Etching the PCB

- **Drilling:** We used a 0.8mm drill to perforate the parts of the PCB where components were required to be attached. Also a 1mm drill had to be used in cases where Diodes had to be attached. For attaching the DB9 connector, two drills of 4mm wide had to be done to make sure the Connector fits snugly onto the board and does not go out of position under continuous usage. We refrained from tinning our board, since it was only a prototype and we were not expecting a hug life expectancy out of our Board.



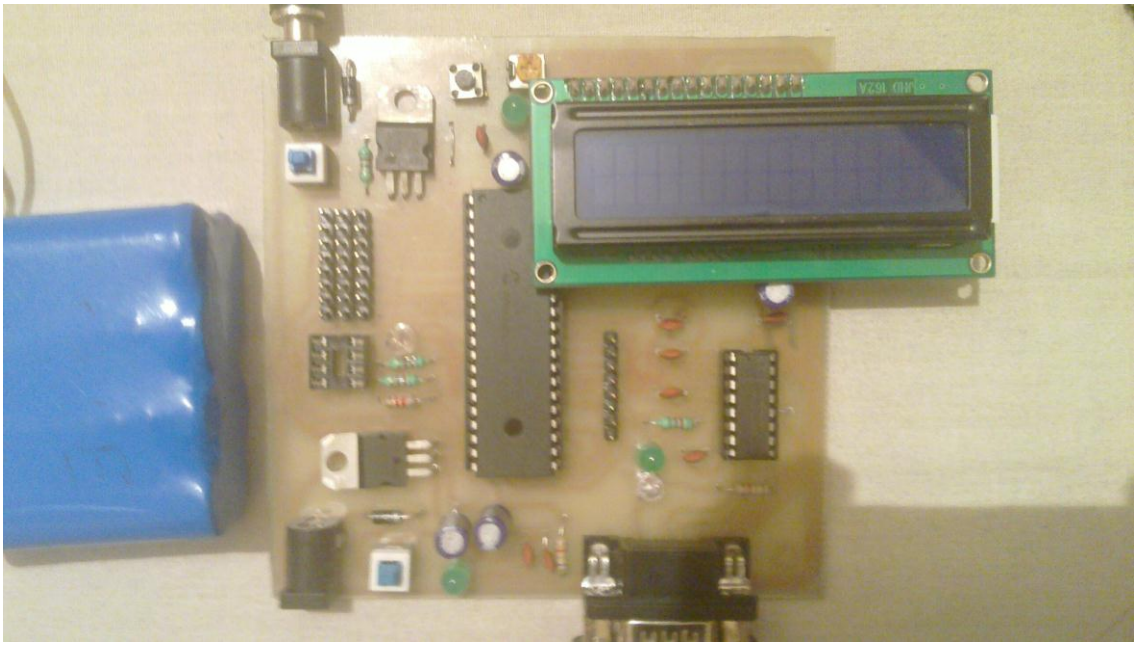
Drilling the PCB (an 0.8mm drill is used in the photo)

- **Soldering:** The soldering was done using a Soldron Iron and standard soldering wire. Ideally lead free solder should have been used, but this was not a necessity in our case, so we stuck to the cheaper and easily available option that we had.



Soldering the PCB

Finally our PCB was ready. It was then tested for any possible errors that might have crept in the way of the build process. The errors may include short circuiting of tracks while soldering the board, and unconnected tracks, unwanted connections etc.



The final PCB

Testing the PCB and Improvisations

The PCB was testing using the following steps:

- All pins in the socket of the controller were tested for unwanted voltages. This was a primary step to do, since it might have damaged the controller in case an error was present. Only after this testing the controller was attached.
- After this, it was now time to verify if the ISP of our board was working properly. We used a USB to Serial adapter for the operation.
- A code was burnt to send a high pulse to all I/O port after a delay of 5secs. This made testing the I/O easy.
- Also the EEPROM was tested if it could store and retrieve data as it should. The debug output was printed on the LCD.
- Finally the motors were connected and tested.

Our PCB was working just fine. So we proceeded with coding the Microcontroller for the project.

Code for PIC16F877a Controller

Compiler used: We used the HITEC PIC-C Compiler to code our PC to Arm interface. This was an industry standard compiler which was tested and trusted upon by a large community of Embedded Developers all around the world since a long time. Another reason for choosing this compiler is that the bootloader installed in controller is compatible with the HEX file generated by PICC.

Code to Control Servo Motor via Serial Instructions from LabVIEW

Code is roughly divided into following parts:

- Machine Cycle Optimizations to avoid Serial/ Motor Control Instruction Collision
- Implementation of interrupts
- Receiving data from LabVIEW via serial communication.
- Integration of labview data with motor.

Machine Cycle Optimizations to avoid Serial/ Motor Control Instruction Collision

A servo motor expects a pulse every 20ms and width of the pulse varies from .8ms to 2.5ms. 0.8ms for maximum rotation to -90 and 2.5ms for +90, these values are specific for the motors we have used.

These pulses are generated by microcontroller. They are many methods for moving servo motors i.e. generating the required pulses and the one we have used provided us the best precision. The PWM signal is formed using “Timer0”. It generates an interrupt every 1.9ms and then send a pulse using delay function. So after very 1.9ms a pulse of width .8ms to 2.5ms is generated for each of the 5 motor. The input value for delay function is in microseconds therefore 1700 different inputs are available for -90 to +90 rotation and thus good precision.

Timer0 is a 8 bit timer therefore it counts 256 steps i.e from 0 to 255. SFR's involved for timer operation are:

TMR0: Timer value is stored in this register.

INTCON: It is used for controlling interrupts.

OPTION_REG: For selecting timer 0 and deciding prescale values. Value to be stored in TMRO is 256 – delay cycles.

Delay cycles = delay required (sec)* crystal frequency/8

If this value is below 255 then its fine and is value is greater than 255 keep on dividing it by 2 till value is less than 255. No. of times you divide it by2 is the prescale. Prescale is nothing a quantity used for decreasing frequency thus increasing the time period. So for are requirement we need delay of 1.9ms and the clock frequency is 20MHz therefore

Delay cycles= ((1.9/1000)*(20×1000000))/8=4750

Dividing 4750 8 times we get value 149 thus we have to count 149 times and therefore TMR0 is loaded with value 106 and OPTION_REG is loaded with 0×85.

OPTION_REG

1	0	0	0	0	1	1	1
RBPU				TOCS	PS2	PS1	PS0

T0CS bit is used for selecting clock source and by clearing it internal clock frequency is selected. In our circuit 5 servo motors are connected to PORTA.

Implementation of interrupts

To generate delay which is required for servo motor pre-defined delay functions is not used but instead Timer 0 is used. We did this because when pre-defined functions are used then the controller is at halt for that particular delay but with timers the controller can perform another task while timer is running at the back.

After completion of count by timer an interrupt will be generated. To respond to that interrupt a ISR(Interrupt service routine) is written. ISR should be as short as possible and in our case it just increments count value and clear the interrupt flag (TMR0IF).And every count value corresponds to motor.

Receiving Data from Labview

To receive data from labview a function named UART is implemented.It is called after pulses are sent to all the servo motor. In UART function each character received is stored in an array (output [6]). When string terminator character ("+") is received then the increment value "output [2]" is converted to integer because the value received is string type. Now the value is in ASCII format so if data sent is 2 then value we have is 50, ASCII for 2 and to get original value 48 is subtracted.

Integration of labview data with motor

The values after conversion are digits and are the converted into a number and are stored in variable sdelay. Now a case structure is used to select different motor (value in output [0]) and in each case value of delay is updated.

C Code

```
#include<16f877a.h>

#fuses XT, NOWDT, NOPROTECT, BROWNOUT, PUT, NOLVP

#use delay(clock=20000000)

#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)  // Jumpers: 8 to 11, 7 to 12

#byte INTCON = 0x0B                // define INTCON's address

#bit RBIF = INTCON.0

#bit INTF = INTCON.1

#bit TMR0IF = INTCON.2

#bit RBIE = INTCON.3

#bit INTE = INTCON.4

#bit TMR0IE = INTCON.5

#bit PEIE = INTCON.6

#bit GIE = INTCON.7

#byte OPTION_REG = 0x181           //define OPTION_REG

#bit RBPU = OPTION_REG.7

#bit INTEDG = OPTION_REG.6

#bit T0CS = OPTION_REG.5

#bit TOSE = OPTION_REG.4

#bit PSA = OPTION_REG.3

#bit PS2 = OPTION_REG.2

#bit PS1 = OPTION_REG.1

#bit PS0 = OPTION_REG.0

#byte TMR0 = 0x01                  //define TMRO

int16 d1,d2,d3,d4,d5;              //delay variables

int count=0;

float countt[4];

int c=0;
```

```
int j=0;
float sdelay;
unsigned char output[6];
#INT_timer0                          // ISR
void isr()
{
if(TMR0IF==1)
{
count+=1;
TMR0IF=0;
}
}
void uart()                          //UART function
{
if(kbhit()==1)                       //polling for serial data
{
output[j]=getc();                    //storing received character in array
j+=1;
if (output[5]==43)                   // checking for string end("+")
{
j=0;
      countt[0]=(unsigned int) output[1];// typecasting the received string to
      int
      countt[1]=(unsigned int) output[2];
      countt[2]=(unsigned int) output[3];
      countt[3]=(unsigned int) output[4];
      countt[0]=countt[0]-48;
      countt[1]=countt[1]-48;
      countt[2]=countt[2]-48;
      countt[3]=countt[3]-48;
sdelay=(countt[0] * 1000)+(countt[1] * 100)+(countt[2] *
10)+(countt[3]);//forming pulse integer from received string
      output[5]=49;
```

```
      switch(output[0])        // output[0] is motor no.
      {
      case 49:                 // case for motor 1,49 ascii for 1
      d1=sdelay;
      break;
      case 50:                 //motor2
      d2=sdelay;
      break;
      case 51:                 //motor 3
d3=sdelay;
break;
case 52:                       //motor 4
d4=sdelay;
break;
case 53:                       //motor 5
d5=sdelay;
break;
}
}
}
}
void home()// initial joint positions
{
d1=d2=d3=d4=d5=800;
}

void main()
{
setup_adc_ports(NO_ANALOGS);
set_tris_a(0x00);                  // making port A output
output_a(0x00);                    //clearing port A
home();
```

```
GIE=1;                          //enabling global interrupt
PEIE=1;                         // enabling peripheral interrupt
OPTION_REG=0x85;                //storing the calculated value in OPTION_REG
TMR0IE=1;                       //enabling timer 0 interrupt
TMR0=106;                       //loading timer value
while(1)
{
switch(count)
{
case 1:                         //motor1
output_high(PIN_A0);
delay_us (d1);
output_LOW(PIN_A0);
c+=1;
count=2;                        // changing count value
break;
case 3:                         //motor2
output_high(PIN_A1);
delay_us (d2);
output_LOW(PIN_A1);
count=4;
break;
case 5:                         //motor3
output_high(PIN_A2);
delay_us (d3);
output_LOW(PIN_A2);
count=6;
break;
case 7:
output_high(PIN_A3);
delay_us (d4);
output_LOW(PIN_A3);
```

```
count=8;
break;
case 9:
output_high(PIN_A5);
delay_us (d5);
output_LOW(PIN_A5);
count=0;                                    //resetting count value
break;
}
if(c>2)
{
uart();                                     //calling UART function
c=0;
}
}
```

VI | Controller Design

Controller Requirements

To control the robot a tool was required which could easily communicate with the electronic hardware and thus control the whole mechanical structure. It should also serve the purpose of GUI so that a person without prior knowledge should be able to control the robot.

Processing Power and performance required to implement the controller

For controlling a robotic structure complex mathematical calculations involving matrices and trigonometry is required. Since microcontroller is only 8 bit it cannot perform either of the above mentioned tasks nor it can do floating point calculations, a controller is required which has both processing power and speed for performing complex calculations and a modern day will be ideal for this.

LabVIEW Robotics

LabVIEW is a graphic programming language tool by National Instruments. It is similar to any text base language tool like Java or c but instead of using text commands, block diagrams are used for implementing a particular algorithm.

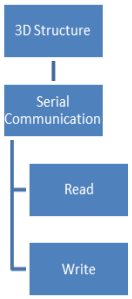
Use of LabVIEW in the project

Labview is used to create a GUI (Graphical User Interface) .Following things have been implemented using labview:

Labview is used to create a GUI (Graphical User Interface) for the project. Following things have been implemented using labview:

- A 3D structure of robot is created by specifying the kinematic parameters.
- Individual control of each joint and the gripper.
- Transfer of motor rotation values to hardware using serial communication.

The above goals are realized using 4 files.



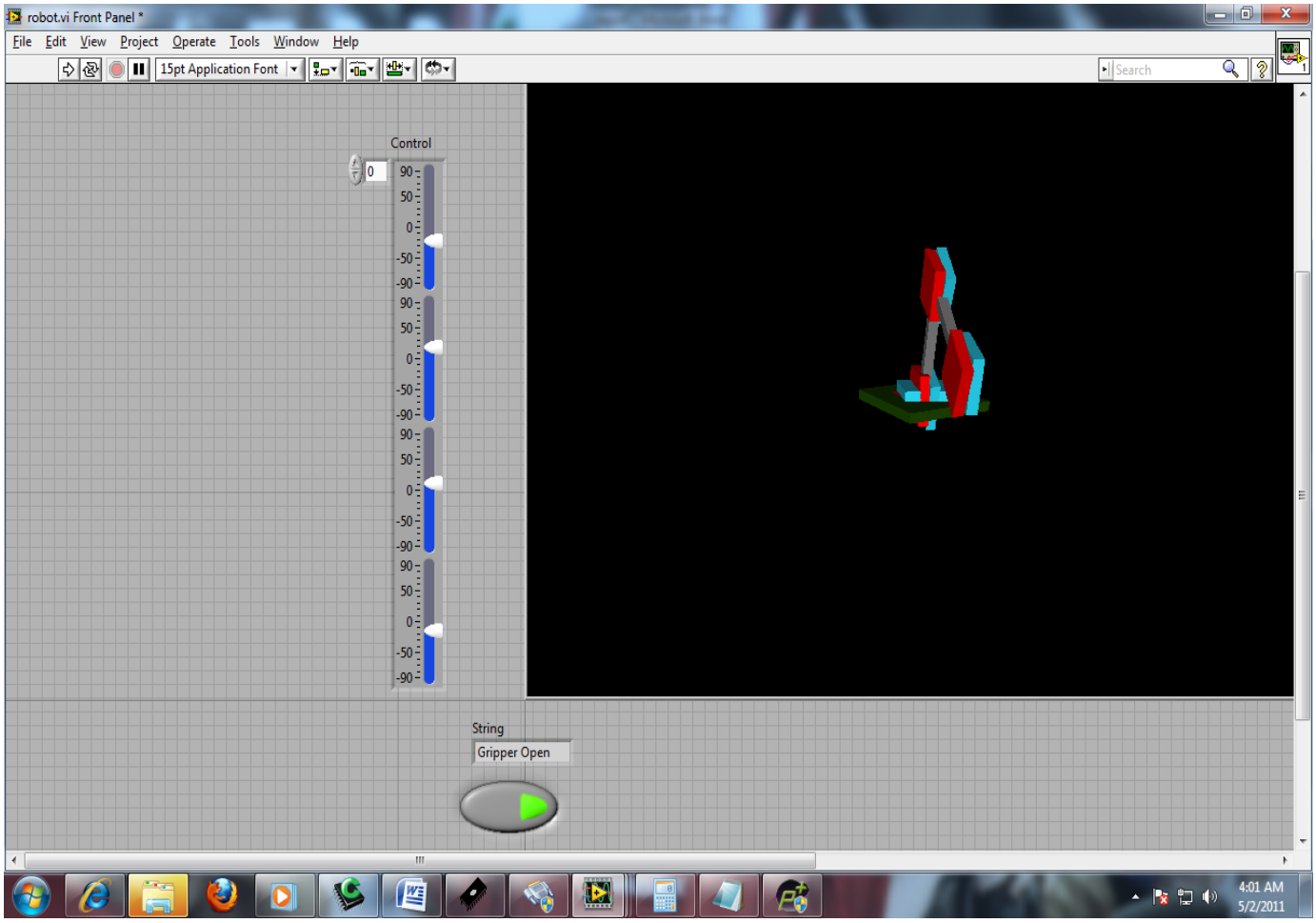
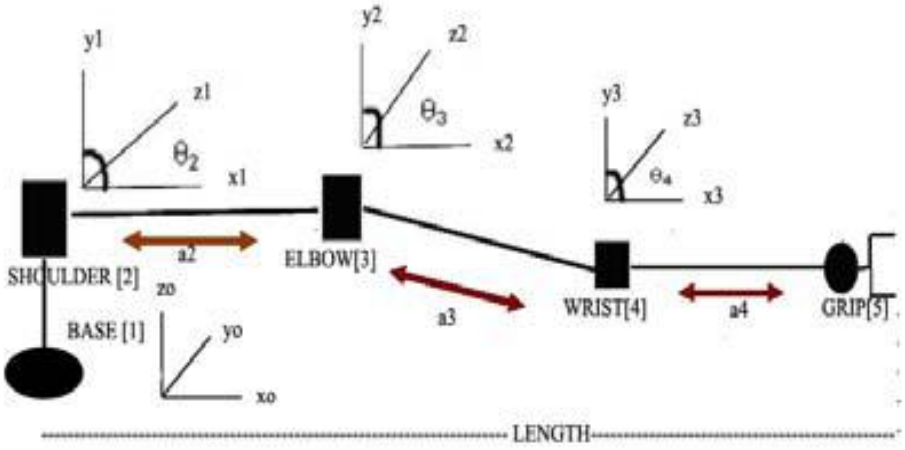


Figure shown above is the GUI. It has 4 sliders for controlling each axis , A button for closing and opening the gripper and a 3D model of the arm.

3D Structure

The 3D structure is created by specifying the D-H parameters(Denavit Hartenberg) of the structure. D-H parameters include 4 quantities which are used to specify the relationship between the links.

- d: offset along previous *z* to the common normal
- θ: angle about previous *z*, from old *x* to new *x*
- a: length of the common normal (aka *a*, but if using this notation, do not confuse with α). Assuming a revolute joint, this is the radius about previous *z*.
- α: angle about common normal, from old *z* axis to new *z* axis.



Above is the coordinate system of the robot. While assigning the coordinates 2 standard assumption are made:

- Z axis is always along the direction of rotation.
- X axis is along common normal between 2 consecutive Z axis

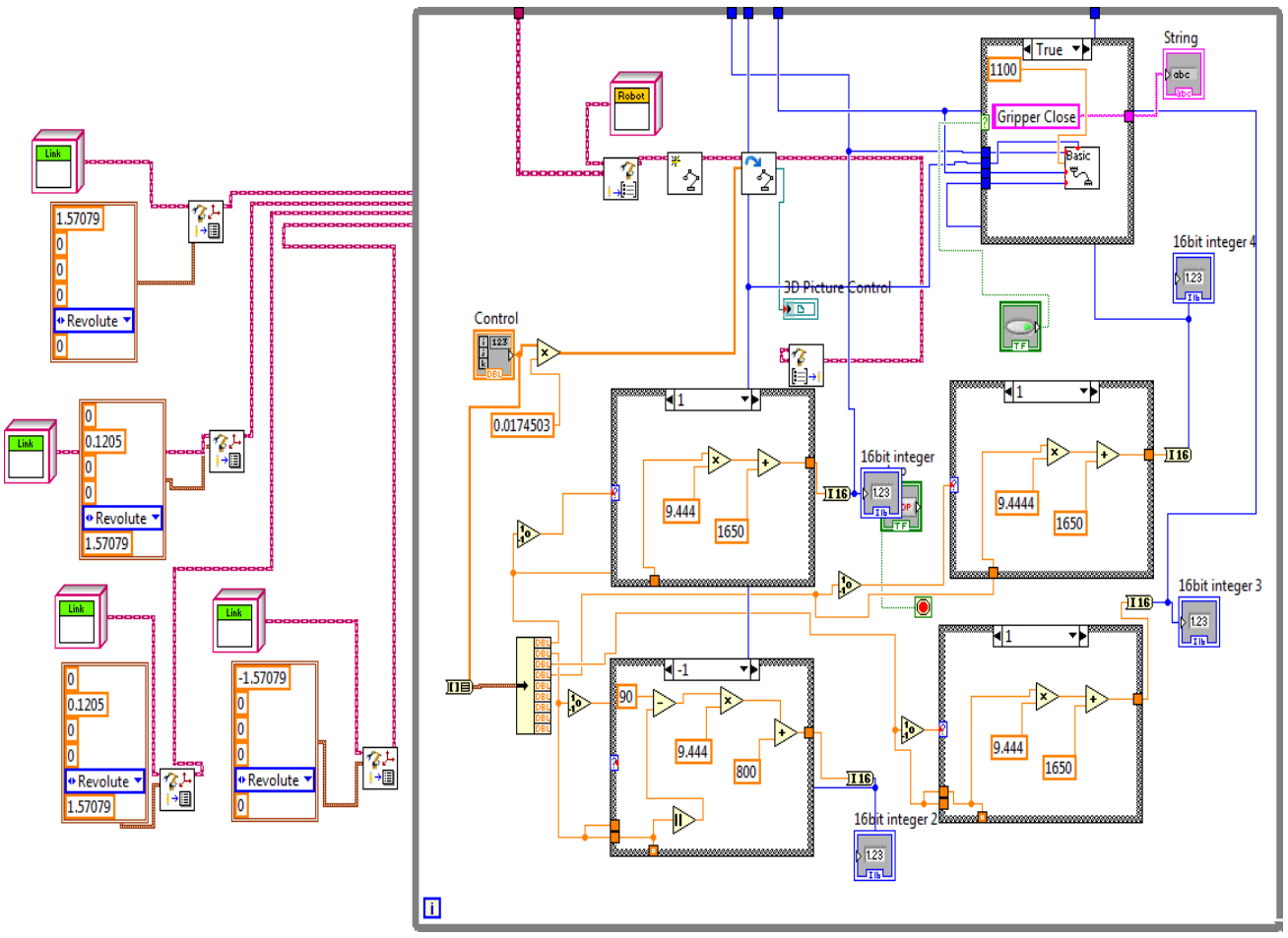
D-H Parameter

.

Joint	αi degree	ai (m)	θi degree	di (m)
1	90	0	θ1	.06
2	0	.14	θ2	0
3	0	.14	θ3	0
4	0	0	θ4	0

To implement this Labview robotics module is used. **Set Kinematic parameters.vi** is used for assigning D-H parameters for every link. After this the whole structure is created using **Setlinks.vi**. Then structure is displayed using **3D picture control vi.**

To control movement of every joint a combination of **initializeplot.vi** and updateplot.vi is used. Input to **updateplot.vi** in an array of 4 slider indicators whose value varies from -90 to +90 thus giving a complete 180 degree movement. Labview only accepts values in radians so these values are converted into radians by multiplying it by .0174503.



Boxes on the left are D-H parameter for every link. The gray box signifies a infinite while loop and the boxes inside it are changing of angle values into motor pulses.

Individual Axes Rotation

For actual motor to rotate +90 a pulse of 2.5 ms is used and for -90 a pulse for .8 ms is used. To generate these pulse values a combination of arithmetic tools are used. For 0 degree, the value to be sent is 1650. If value is negative then we subtract angle value from 90 then multiply it with 9.444 and then add 800. If angle of rotation is greater than 0 then also value is subtracted from 90 and multiplied with 9.444 and then add 1650.

The above mentioned procedure is done to convert value for each of the axis. For gripper control a switch is used, when switched is pressed then the value required to close the gripper is sent along with other 4 axis and when switch is open then the value required to open the gripper is sent along with 4 axis value.These 5 values is sent to another labview vi used for serial communication .Figure below is the block diagram/code of the 3D strucuture vi.

Serial Communication

Our aim is to establish real time communication between robot and labview so that when we move the move the robot in simulation an analogous movement is visible in the actual robot. To achieve the above mentioned goal we used serial communication for transfer motor pulse values discussed above to electronic circuit. For serial communication there is already a built in VI in labview and we will be using that only and also some general string and integer functions.For serial communication VISA driver has to be installed. Then using **VISA configure serial port vi** we set following parameter for serial communication:

- Baud rate :9600
- Port :COM3
- Data bits :8
- No. of stop bits: 1

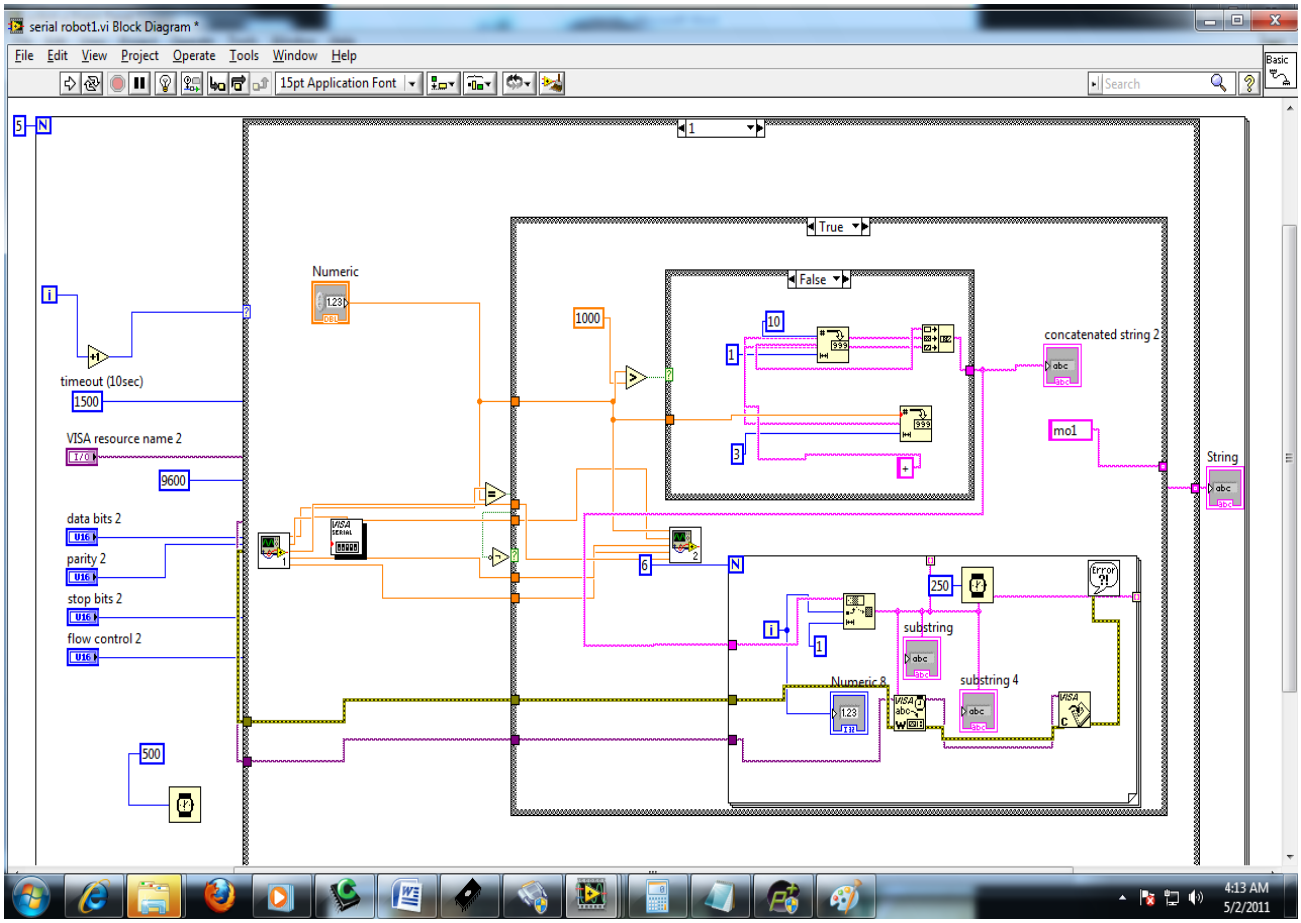
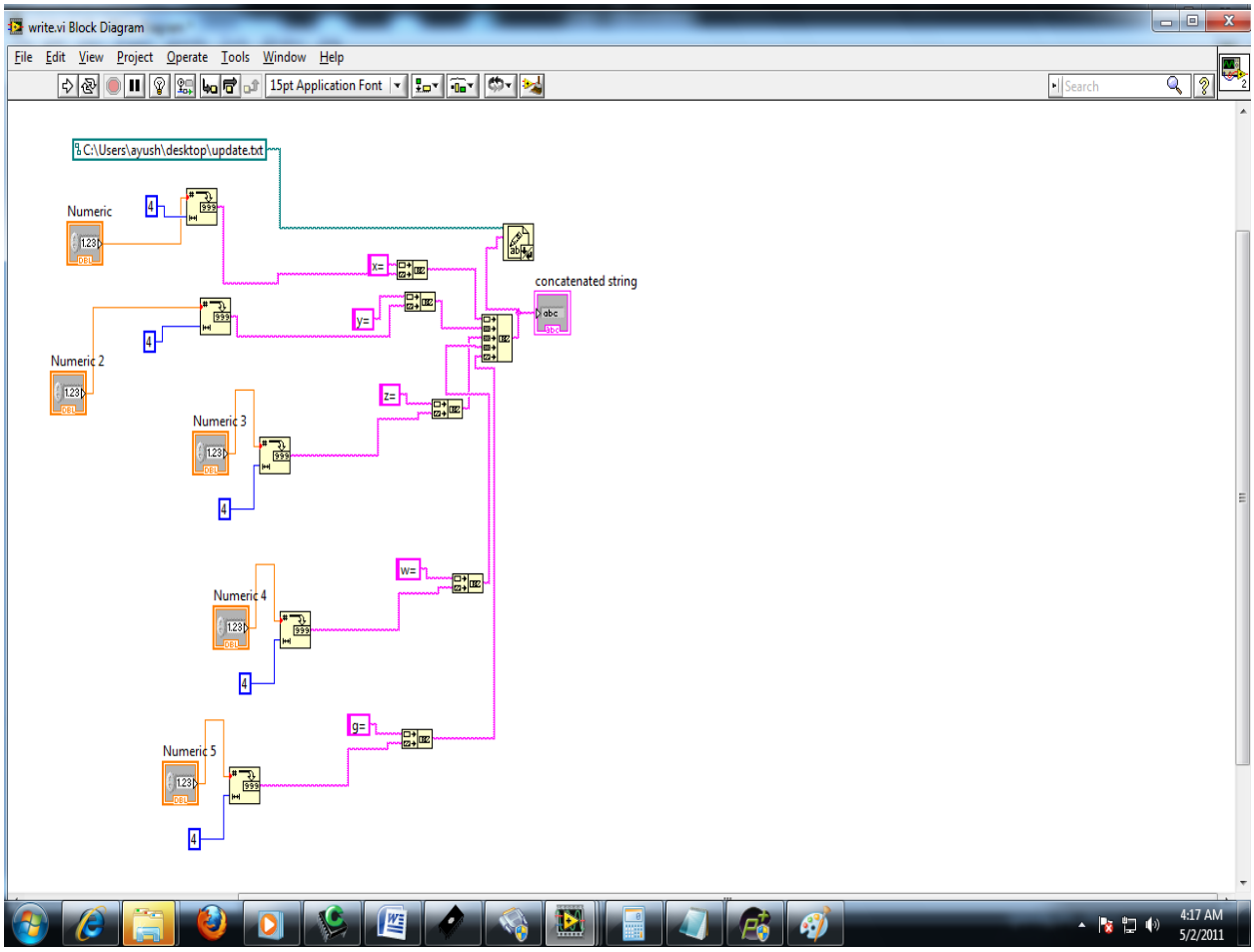


Figure above is the block diagram of Serial communication vi.

The pulse value to be sent is received from the 3D**structure VI.** String is sent is in a predefined order. The first character sent is motor no. , next 4 characters are pulse value and last character is "+" which represents string end. So if we want to rotate 4th motor to+90 then 42500+ is sent to microcontroller. Pulse value is converted into string using **number to decimal string VI**. And then using **string concatenation VI**. Motor no. and "+" is added.

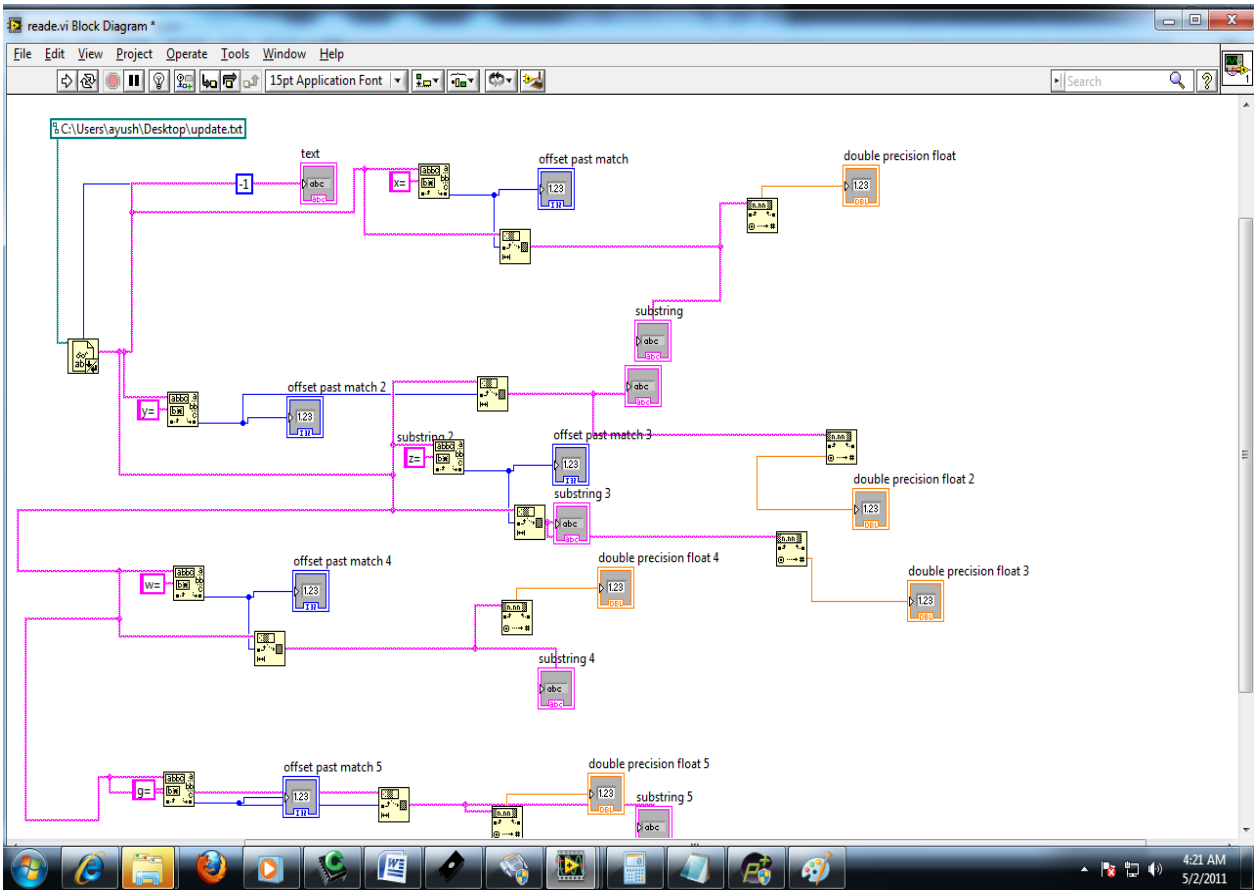
Once the string is generated then it is sent character by character using **VISA writes VI**. Characters to be sent out are separated using **StringSubset.vi**. This whole sequence is used for all the 4 axes and gripper. A **case structure** is used to select the motor and then these pulse values are sent.



Block diagram of write vi

To improve the speed of the communication a dummy feedback system is used . After sending the values to controller they are written to a file (**Write to text file.vi**). So next time values are sent it is compared with previously sent values(**Read from text file.vi**) and if they are same then that motor values are skipped . Thus only those values which have changed are sent only.This whole code is written in one single file . This file has five inputs(4 axis and 1 gripper values). It is used as function in structure file. File reading and writing are implemented in different files and are used as functions in this serial communication file.Both reading and writing is accomplished by using basic string functions like:

- Decimal to string converter
- String Subset
- String concatenate
- String to decimal converter


reade.vi Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
C:\Users\ayush\Desktop\update.txt
text
offset past match
double precision float
substring
offset past match 2
substring 2
offset past match 3
substring 3
double precision float 2
offset past match 4
double precision float 4
double precision float 3
substring 4
offset past match 5
double precision float 5
substring 5
4:21 AM
5/2/2011

VII | Further Improvement Possibilities

This is the first time that we have ever designed a robotic arm. This was more of a prototype than an actual product design. We think that the following modifications can be made to turn our robot into an actual industrial product. These are enlisted as follows:

- **Mechanical** - Ideally our robot should have been constructed out of carbon fiber. However, the high expense of the material and lack of availability compelled us to use aluminum as a cheap replacement of carbon fiber. We plan to reconstruct all the links of our robot using carbon fiber to increase the strength to weight ratio of the material used. Due to simple design it makes it easy to modify. Therefore we plan to make it a six axis robot.
- **Electronic** - Right now we are using a PC to crunch all the data; however we plan to use advanced 32 bit controller so that we can eliminate the use of a pc to control the robot.
- **Control** - We plan to use more advanced and intelligent techniques and algorithms to control our robot. Algorithms like statistical machine learning, inverse kinematics, intelligent object recognition using image processing etc.

VIII | Application Areas

Our robot can be modified to be used in the following areas:

1. Industrial automation
2. To provide students the basic understanding of the building blocks of robotics.
3. As an integral part of Flexible Manufacturing System.
4. As a basic platform for robotics research.
5. As a platform for implementing artificial intelligence algorithm.
6. As a kick start learning utility for enthusiasts of humanoid design.

IX |Concept of the Open Source Project

We plan to make every single intellectual property associated with the project open source and freely available to everyone through the internet.

This decision was meant to be a contribution to the technical community for all the help and knowledge that we have obtained from different people all over the world through various forums, blogs, websites and journals.

We really wish our project to help the community in every possible way that may be required. Also putting the project into the open source domain will ensure its continual growth and recognition. Any individual who may find this project interesting and worthy of use in their own work may modify and reuse our design, code and techniques in their own way to foster the growth of our ideology with which we had initiated the project.

X | References

- *Software Development for the Kinematic Analysis of a Lynx 6 Robot Arm* - Baki Koyuncu, and Mehmet Güzel (World Academy of Science, Engineering and Technology, 2007)

- *Kinematic Modeling and Simulation of a SCARA Robot by Using Solid Dynamics and Verification by MATLAB/Simulink* - Mahdi Salman Alshamasin, Florin Ionescu, Riad Taha Al-Kasasbeh (European Journal of Scientific Research, 2009)

- *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*: Samuel R. Buss, Department of Mathematics, University of California, San Diego, October 7, 2009

- *Jacobian Solutions to the Inverse Kinematics Problem* - Mike Tabaczynski, Tufts University, January 17th, 2006

- *How to Interface a Microchip PIC MCU with a hobby R/C Servo* - Paulo E. Merloti

- *A Robust Inexpensive Multi-Purpose Robotic Arm* - Alana Lafferty, UAP Report (May 20, 2005)

- *Kinematic Analysis for Robotic Arm* - Sırma C. Yavuz, Yildiz Technical University, İstanbul, 2009

- *Application on D.C. Servo Control* Developed by Stephen Bowling, Microchip Technology Inc. Chandler, AZ (AN696)

- LabVIEW Robotics Documentation - *www.ni.com/pdf/manuals/375286a.pdf*

- Robotics Toolbox for MATLAB – *www.petercorke.com/Robotics_Toolbox.html*