

DESIGN AND FABRICATION OF KITCHEN VESSEL CLEANING ROBOT

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in
Mechanical Department.

By

G.VISHAL (37150218)

DITTO Y (37150056)



**DEPARTMENT OF MECHANICAL ENGINEERING
SCHOOL OF MECHANICAL ENGINEERING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

MARCH -2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC | 12B Status by UGC | Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF MECHANICAL ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **G.VISHAL (37150218)**, **DITTO .Y (37150056)** who have done the project work as a team who carried out the project entitled "**DESIGN AND FABRICATION OF KITCHEN VESSEL CLEANING ROBOT**" under our supervision from **October 2020** to **March 2021**.

MR.J.Jeya Jeevehan

Internal Guide

Head of the Department

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

DECLARATION

We, **G.VISHAL (37150218)** and **DITTO .Y (37150056)** hereby declare that the Project Report entitled “**DESIGN AND FABRICATION OF KITCHEN VESSEL CLEANING ROBOT**” done by us under the guidance of **MR.J.Jeya Jeevehan** Department of Mechanical Engineering is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Mechanical Engineering.

DATE:

PLACE:

1.

2.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. S. Prakash**, Dean, School of Mechanical Engineering and **Dr. G. Arunkumar**, Head of the Department, Dept. of Mechanical Engineering for providing the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my guide **MR.J.JEYA JEEVAHAN** for his valuable guidance, suggestion and constant encouragement paved way for the successful completion of my project work.

I would like to express my thanks all Teaching and Non-Teaching staff members of the Department of Mechanical Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

Nowadays, Machines are widely controlled by control system. To meet the need of exploding population economic and effective control of robotics is necessary. The aim is to design and develop a automatic vessel cleaning system based an robotic arm system using Ultra sonic Sensor and the control unit. This work introduces a project that aims in designing a automatic vessel washer and dryer using ultrasonic sensor, camera and a arduino microcontroller. The ultrasonic sensor in the system senses the walls of the vessel . Arduino is used as the microcontroller to evaluate the process and to run the system which is also helped by servo motor driver. Robotic arms helps to pick , hold and clean. This project is mainly a model of two robotic arms where one fitted with brushes ,water and detergent outlet and another arm to hold and rotate the vessel or the material which is need to be cleaned. Vessel washing is lot time consuming work to overcome this an automatic machine is used which washes the vessel one by one and dries it and also takes less time frame This machine reuses water in the water tank so, the water used is less compared to dish washers. Unlike dishwashers , this machine cleans all types of apron and all types of vessels irrespective of the shape. Unlike dishwashers , the material used in this machines are less in cost as a result the whole product will be of less cost so that middle class people can afford it.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	I
	LIST OF FIGURES	II
1	INTRODUCTION	1
	1.1 INTRODUCTION TO AUTOMATION AND ROBOTICS	2
	1.2 SMART MANUFACTURING AND MODERN ROBOTICS	4
	1.3 USE OF HUMANOID ROBOT AS A HOME APPLIANCE	4
2	LITERATURE SURVEY	5
3	AIM AND SCOPE OF THE PROJECT	9
	3.1 AIM	9
	3.2 SCOPE OF PRESENT INVESTIGATION	10
	3.3 PRESENT INVESTIGATION	11

CHAPTER NO.	TITLE	PAGE NO.
4	PROJECT DESCRIPTION AND METHODOLOGY	12
	4.1 BLOCK DIAGRAM	14
	4.2 SYSTEM REQUIREMENTS	16
	4.2.1 HARDWARE REQUIREMENTS	19
	4.2.2 SOFTWARE REQUIREMENTS	22
	4.3 MANUFACTURING PROCESS	26
	4.4 INSPECTION	32
	4.5 ASSEMBLY	35
	4.6 NEED FOR AUTOMATION	38
	4.7 HARDWARE REQUIREMENT IN DETAIL	39
	4.7.1 POWER SUPPLY UNIT	39
	4.7.1.1 INTRODUCTION	40
	4.7.1.2 PIN DIAGRAM	42
	4.7.1.3 BLOCK DIAGRAM	44
	4.7.1.4 PIN DESCRIPTION	45
	4.7.1.5 ARCHITECTURE DIAGRAM	46
	4.7.1.6 ATMEGA 328 AND ARDUINO	48
	4.7.2 LIQUID CRYSTAL DISPLAY	52
	4.7.2.1 PIN DESCRIPTION	55
	4.7.3 RELAY DRIVER	57
	4.7.4 ULTRASONIC SENSOR MODULE	59
	4.8 SOFTWARE REQUIREMENT	60
	4.8.1 ARDUINO IDE 1.8.5	61
	4.8.2 WHY ARDUINO?	63
	4.9 METHODOLOGY	64
	4.9.1 FLOWCHART AND ALGORITHM USED	64
	4.9.2 WORKING OF THE PROJECT	75

CHAPTER	TITLE	PAGE
5	RESULT AND DISCUSSION	78
	5.1 RESULT	78
	5.2 DISCUSSION	78
6	SUMMARY AND CONCLUSION	80
	6.1 SUMMARY	80
	6.2 CONCLUSION	81
7	REFERENCE	83
8	IMAGES OF THE PROJECT	87

CHAPTER 1

INTRODUCTION

A qualitative comparison of the cost effectiveness of manual labour, hand automation and soft automation as a function of the production volume. The robot can convey many different meanings in the mind of the reader, depending on the context. Here, a robot will be taken to mean an industrial robot, also called a robotic arm. Robots have replaced humans in the assistance of performing those repetitive and dangerous tasks which humans prefer not to do, or are unable to do due to size limitations, or even those such as in outer space or at the bottom of the sea where humans could not survive the extreme environments. So the development of robotic field means development a country. For automaton the industry, robot plays a vital role. In this sector a robot can do the work of hundred workers at a time. A robotic arm is a type of mechanical devices which is an industrial robot, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robot. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement. The links of the manipulator can be considered to form a kinematic chain. The terminus of the kinematic chain of the manipulator is called the end effector and it is analogous to the human hand. In many industry human can be replace by robotic arm because of its low cost and low time consuming for loading and unloading metallic goods. There are concerns about the increasing use of robots and their role in society. Robots are blamed for rising unemployment as they replace workers in some functions. The use of robots in military combat raises ethical concerns. The possibility of robot autonomy and potential repercussions has been addressed in fiction and may be a realistic concern in the future.

Automatic systems and robots Robots can be considered as the most advanced automatic systems and robotics, as a technique and scientific discipline, can be considered as the evolution of automation with interdisciplinary integration with other technological fields. An automatic system can be defined as a system which is able to repeat specific operations generally with a low degree of intellectual and manipulative levels, but that can be easily programmed in agreement with demands of productivity. It is worthy of note that an automatic system is generally able to perform one operation for which its mechanical structure has been designed. Aspects of flexibility depend on the possibility of reprogramming the control unit, which is generally able only to modify the time sequence of the designed operations. Therefore, an automatic system of industrial type can be thought of as composed of two parts: - hardware with mechanical, electrical, pneumatic, and hydraulic components that provide the mechanical capability to perform an a priori-determined operation of movement and/or manipulation; - control and operation counterpart with electronic components and software that provide the capability of autonomy and flexibility to the working of the system. The two parts are essential in an automatic system and are integrated in the sense that their design and operation must be considered as a unique goal in order to obtain and operate an automatic system with the best performances. Therefore, an automatic system in which the hardware part is preponderant cannot have a suitable flexibility for a flexible production in agreement with the demands of productivity and market. In some cases these limited solutions are still required from productivity, when the product of a massive production is absorbed from the market with certain regularity during a period of time longer than the amortization time of the automatic system. Those systems with low flexibility are generally denominated as 'rigid automatic systems'. The variety of production systems is represented as a function of the productivity level. Automatic systems are often designed with the aim of having the possibility of an easy updating of their structure and operation with the purpose of being adjusted more quickly to the mutable demands of production and design of products. These very flexible systems are denominated as 'flexible automatic systems'. In addition, it is worthy of note that a system with only the mechanical counterpart, although versatile, cannot be considered an automatic system since the updating of its operation is not obtained by means of control units. Such a system can be properly reprogrammed, but it requires the manual action of one or more human operators to change some components of the machinery or their running. Therefore, these systems can

be properly named as machinery or mechanical systems. M. Ceccarelli, Fundamentals of Mechanics of Robotic Manipulation , Springer Science+Business Media Dordrecht 2004

Introduction to Automation and Robotics Variety of automatic and robotized systems as functions of productivity level and product demand. Similarly, control systems cannot be considered as automatic systems, since they are not able to perform mechanical tasks, although they can be provided with a high level of flexibility in terms of re-programmability and memory capability. Therefore these systems can be properly named as control units or electrical–electronic–informatics systems. However, an automatic system is designed and built by using a suitable combination and integration of mechanical systems and control units. Indeed, the success and engineering application of automatic systems strongly depends on the practical integration of the above-mentioned counterparts. A robot can be defined as a system which is able to perform several manipulative tasks with objects, tools, and even its extremity (end-effector) with the capability of being reprogrammed for several types of operations. There is an integration of mechanical and control counterparts, but it even includes additional equipment and components, concerned with sensorial capabilities and artificial intelligence. Therefore, the simultaneous operation and design integration of all the above-mentioned systems will provide a robotic system. In fact, more than in automatic systems, robots can be characterized as having simultaneously mechanical and re-programming capabilities. The mechanical capability is concerned with versatile characteristics in manipulative tasks due to the mechanical counterparts, and re-programming capabilities concerned with flexible characteristics in control abilities due to the electric-electronics-informatics counterparts. Therefore, a robot can be considered as a complex system that is composed of several systems and devices to give:

- mechanical capabilities (motion and force),
- sensorial capabilities (similar to human beings and/or specific others),
- intellectual capabilities (for control, decision, and memory).

Smart manufacturing and modern robotics

A recent trend in manufacturing is to implement increased automation and data exchange, thus leading to a smarter manufacturing approach. Such efforts are combined with a gradual progress toward realizing the benefits of the fourth industrial revolution and its interconnected cyber physical systems essentially operating without human intervention. Modern robotics combined with other smart industry enablers such as artificial intelligence (AI), data analytics, mobile devices, scale-out, systems integration, and digitalization will help biomanufacturers overcome multiple roadblocks. Progress in companion technologies (sensors, end-effectors, adaptive and reconfigurable systems, intuitive programming, systems interoperability, and cooperative and collaborative approaches) and increasing affordability also will trail-blaze the path for robots into small and mid-sized biomanufacturing facilities.

>Although the scope of this discussion was limited to traditional recombinant protein manufacturing, smaller, more modular-sized facilities for cell and gene therapies and individual product manufacturing such as chimeric antigen receptor and autologous cell therapies also will benefit by the implementation of robotics.

The experience from other industries that have implemented robotics is positive. In the automotive industry, the International Federation of Robotics has concluded that the introduction of robots increases productivity and competitiveness leading to increased demand, without drastic negative effects on employment levels. Thus, the time is ripe for the biomanufacturing industry to strongly consider robotics in its daily operations.

Use of humanoid robot as a home appliance

Humanoid robots can be very helpful as a cleaning assistant in various fields like in industries, stores, as a household etc. As we all know the drastic changes the humanoid robots made in different sectors of industries. In this contemporary world humanoid robots take over home appliances too. Many home appliances are replaced with humanoid robots as they automatically sense their mode of work like when, how and where should they process their operation. Due to this replacement lot of time, effort and even stress can be reduced as these automatic humanoid robots

automatically does all the jobs that the master want them to do. These humanoid robots as a home appliance or as a helping assistant will be very usefull as their work will be in perfect condition and errors they conduct will also be very less. Humanoid robots can be used as a cleaning assistant in home as we all know cleaning vessels ,floors etc is a lot effort and time consuming work . This can be completely omitted due to the introduction of these robots in home appliance sector.

CHAPTER -2

LITERATURE SURVEY

[1] A. Elfasakhany, E. Yanez, K. Baylon, and R. Salgado, "Design and development of a competitive low-cost robot arm with four degrees of freedom," *Modern Mechanical Engineering*, vol. 1, p. 47, 2011.

The main focus of this work was to design, develop and implementation of competitively robot arm with en-hanced control and stumpy cost. The robot arm was designed with four degrees of freedom and talented to accomplish accurately simple tasks, such as light material handling, which will be integrated into a mobile platform that serves as an assistant for industrial workforce. The robot arm is equipped with several servo motors which do links between arms and perform arm movements. The servo motors include encoder so that no controller was implemented. To control the robot we used Labview, which performs inverse kinematic calculations and communicates the proper angles serially to a microcontroller that drives the servo motors with the capability of modifying position, speed and acceleration. Testing and validation of the robot arm was carried out and results shows that it work properly.

[2] C. F. Olson, "Probabilistic self-localization for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 16, pp. 55-66, 2000.

We describe probabilistic self-localization techniques for mobile robots that are based on the principle of maximum-likelihood estimation. The basic method is to compare a map generated at the current robot position with a previously generated map of the environment in order to probabilistically maximize the agreement between the maps. This method is able to operate in both indoor and outdoor environments using either discrete features or an occupancy grid to represent the world map. The map may be generated using any method to detect features in the robot's surroundings, including vision, sonar, and laser range-finder. We perform an efficient global search of the pose space that guarantees that the best position is found according to the probabilistic map agreement measure in a discretized pose space. In addition, subpixel localization and uncertainty estimation are performed by fitting the likelihood function with a parameterized surface. We describe the application of these techniques in several experiments.

[3] C. Bartneck and J. Forlizzi, "A design-centred framework for social human-robot interaction," in Proceedings of the 13th IEEE International Workshop on Robot and Human Interactive Communication, 2004, pp. 31-33.

Robots currently integrate into our everyday lives, but little is known about how they can act socially. In this paper, we propose a definition of social robots and describe a framework that classifies properties of social robots. The properties consist of form, modality, social norms, autonomy, and interactivity. Finally, we provide broad guidelines for the design of social robots. Robotics is a broad discipline. The United Nations (UN), in a recent robotics survey, groups robotics into three major categories: industrial robotics, professional service robotics, and personal service robotics (UN, 2002). Industrial robots have historically represented the vast majority of robotic development, with many deployed in the automotive industry, beginning with the entire automation of a Nissan plant in the 1990s (MetalForming, 1997). Professional service robots are the subject of a much less practiced field, but it is quickly growing. These robots work in domains inaccessible to people, such as navigating abandoned mines and cleaning up nuclear waste. Personal service robots have the highest expected growth rate. They are estimated to grow from 176,500 in 2001 to 2,021,000 in 2005 — an amazing growth rate (UN, 2002)! Personal service robots assist people directly in domestic and institutional settings. Many robots interact with people who have no special skills or training to operate the robot. The design and interaction of these robots will be critical, and raises a number of research and design challenges.

[4] J. L. Jones, "Robots at the tipping point: the road to iRobot Roomba," Robotics & Automation Magazine, IEEE, vol. 13, pp. 76-78, 2006.

Consumer robotics has achieved its first significant commercial success with the arrival of the iRobot Roomba Robotic Floorvac. Roomba has succeeded in the marketplace where other robots failed or feared to tread. Like many good products, Roomba has been rewarded for putting customer's interests first. Roomba accomplishes a task customers care about and does so at a price competitive with other methods. No breakthrough technologies were required, just familiar components used in new ways. A simple set of principles helped designers provide value to our customers by keeping the focus on the application, robot cost and functionality, and system complexity.

[5] R.-J. Wang, J.-W. Zhang, J.-M. Xu, and H.-Y. Liu, "The multiple-function intelligent robotic arms," in Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on, 2009, pp. 1995-2000.

This paper details the design, production, and programming methodology of a multiple-function robotic arm. All of the hardware and software of this robotic arm were designed and produced by the authors. This robotic arm won the championship of the first competition of HIWIN intelligent robotic arms on Aug. 22, 2008 in Taiwan. The main design goal of this robotic arm was to present the following functions: fancy dancing, weight lifting, Chinese calligraphy, and color classification. Another design goal was to minimize cost and maximize performance. On the other hand, a set of the robotic arms are also applied to show martial arts and play the rock-scissors-paper game with modifying and mimetic hands. The characteristics of the robotic arms include: 1. The shoulder of robotic arm includes a pair of motor structures to enhance the ability to lift weight. 2. The stability and accurateness of the robotic arm are optimized for the requirement of high performance throughout the whole structural design. 3. In order to increase the moving ability of the robot, the robotic arm was designed with a four-wheeled transmission structure and track. 4. Two mimetic robotic arms work in concert to present the fancy shows. 5. Five kinds of machine hands were designed to meet the requirements of the six appointed functions.

[6] L. B. Duc, M. Syaifuddin, T. T. Toai, N. H. Tan, M. N. Saad, and L. C. Wai, "Designing 8 degrees of freedom humanoid robotic arm," in Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on, 2007, pp. 1069-1074.

This study presents the processes undertaken in the design and development of an intelligent 8 degrees of freedom (DOF) humanoid robotic arm using specified-made parts. The robotic arm imitates a human arm and able to grasp many objects of different shapes and sizes ranging from a small pen to a large ball. The robotic arm consists of a shoulder, an elbow, a wrist and five fingers and each finger is designed with three moveable sections except only two for the thumb. The sections in each finger will either be pushed or pulled synchronously using internal linkages and controlled by a 18F4431 microcontroller. Basic movement is pre-programmed to ensure that all the movement is within the allowable constraints, and can be controlled by human operator or automatically controlled by a computer. The 8 DOFs humanoid robotic arm provides wide trajectory coverage over the three-dimensional space around the base. The

specified trajectory of each finger, wrist and arm are bio-inspired and carefully designed to resemble the human arm movement.

[7] M. A. K. Yusoff, R. E. Samin, and B. S. K. Ibrahim, "Wireless mobile robotic arm," *Procedia Engineering*, vol. 41, pp. 1072-1078, 2012.

In recent year, with the increase usage of wireless application, the demand for a system that could easily connect devices for transfer of data over a long distance - without cables, grew stronger. This paper presents the development of a wireless mobile robot arm. A mobile robot that functional to do pick and place operation and be controlled by using wireless PS2 controller. It can move forward, reverse, turn right and left for a specific distance according to the controller specification. The development of this robot is based on Arduino Mega platform that will be interfaced with the wireless controller to the mobile robotic arm. Analysis such as speed, distance, load that can be lifted of the robot has been done in order to know its performance. Finally, this prototype of the robot is expected to overcome the problem such as placing or picking object that far away from the user, pick and place hazardous object in the fastest and easiest way.

CHAPTER - 3

AIM AND SCOPE

3.1 AIM

The primary reason of this project is to reduce the effort of work in kitchen especially the vessel cleaning process. Nowadays a lot people are so busy as they have no time to spend in cleaning process in kitchen. The vessel cleaning is one of main hectic work to be done and is also more effortfull work to be done. In vessel cleaning it should not only be cleaned but also should be perfect regarding the hygiene issue .We conducted a survey asking some people who often does the vessel cleaning process ,two of them were housewife and the other one was a restaurant employee . Nowadays almost all the manufacturing process is being atomized in order to deliver the products at a faster rate. The manufacturing operation is being atomized for the following reasons.

- ✖ To reduce man power
- ✖ To increase the efficiency of the plant
- ✖ To reduce the work load
- ✖ To reduce the production cost
- ✖ To reduce the time consumption
- ✖ To reduce the material handling
- ✖ To reduce the fatigue of workers
- ✖ To achieve good quality work
- ✖ Less Maintenance

3.2 SCOPE OF THE PRESENT INVESTIGATION

- This project fabricates the prototype of kitchen vessel cleaning robot . The mechanism parts are not in original sizes. Otherwise, the shape and design of each part is similar with the original design. Since we only fabricate the prototypes, during the test section, the maximum load is lower than what we expect for the exact design.
- This new design of sewage cleaning machine mechanism is for heavy duty work. The design and selection of material element and parameters depends on this scope which can undergo heavy duty work. The cost and processing time can be due to this scope

3.3PRESENT INVESTIGATION

- In this world there are automatic machines ,humanoid robots are used to fullfill various needs but there is no helping assistant or a machine to act as a helping assistant in kitchen mainly in the process of vessel cleaning. Vessel cleaning is a lot time consuming , stressfull ,work to be done. This work affect us physically and also mentally.
- There are machines like dishwasher which is used to wash dishes and spoons but in that the dishes and spoons should be oilfree and also should be placed at certain places to make it clean . This dishwasher does not clean perfectly and is also expensive too.
- We conducted a survey different asking people what are the main concern you face while cleaning kitchen vessels. They were two housewives and one from a leading restaurant who work as a vessel cleaner in that restaurant.

Person 1 :

Name : Hemalatha (Housewife)

Place : Kerala

Age : 50

- ✖ Need to wash vessels 3-4 times a day
- ✖ It will be higher if there relatives ,friend or anybody else.
- ✖ If the cleaning process takes so long legs ,back bone ,hand will start to pain.
- ✖ Some bases are allergic too which makes hands so rough.
- ✖ There is a important need of a vessel washer assistant so that lot of work ,time , stressfull all can be reduced.

Person 2 :

Name : Savithri

Place : Kanyakumari

Age : 45

- ✖ This is a daily task which cannot be refused
- ✖ Cleaning kitchen vessels consumes more time so that the time spent with will also be less.
- ✖ Therefore it also affects me mentally.
- ✖ The process should also be clean as it's a matter of hygiene

Person 3 :

Name : Venkatesh

Place : Chennai

Age : 40

- ✖ There are more number of vessels which comes every hour and there will be restless job to do.
- ✖ As its my daily job the pain the experienced at initial times is now not a matter of fact.

- ✖ It will be so helpful if there is a machine or robot which can help in this vessel cleaning process.
- ✖ There will large vessels as you know it's a restaurant ,it will be so tough to wash large vessels and also it's a lot of time consuming process.

CHAPTER 4

PROJECT DESCRIPTION AND METHODOLOGY

4.1 BLOCK DIAGRAM:

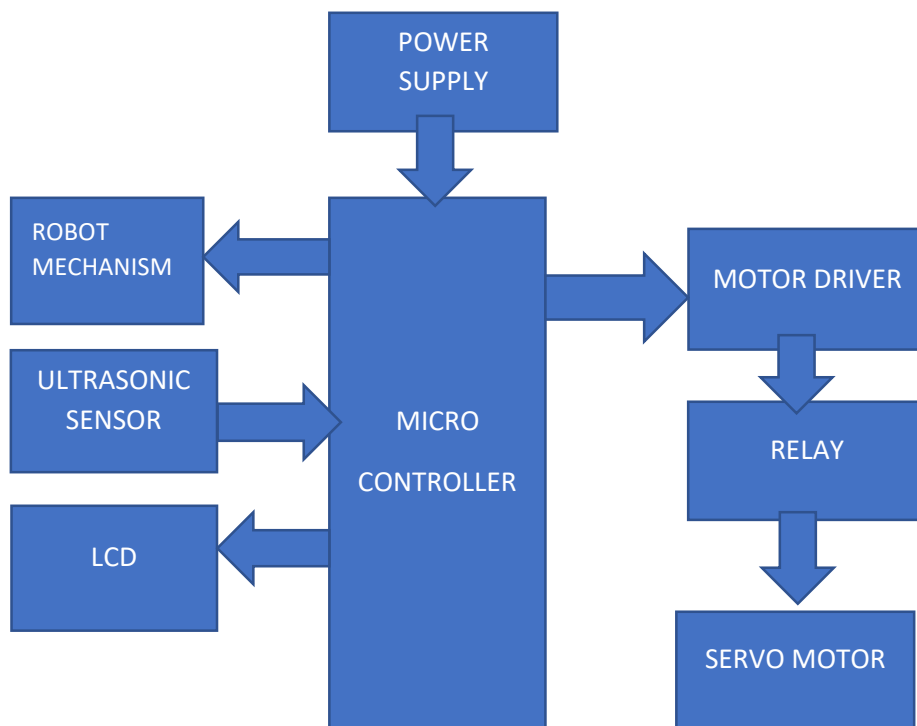


FIG 4.1

4.2 SYSTEM REQUIREMENTS

4.2.1 HARDWARE REQUIREMENTS:

- ARDUINO MICROCONTROLLER
- ULTRASONIC SENSOR

- HUMANOID ROBOTIC ARMS OF LENTH 30CM
- JUMPER WIRES
- BATTERY
- MOTOR DRIVER
- SERVOMOTORS
- COPPER COIL ROLL
- GRIPPER
- BRUSHES
- WATER PUMP

4.2.2 SOFTWARE REQUIREMENTS:

- ARDUINO IDE
- EMBEDDED C
- PROTEUS

4.3 MANUFACTURING PROCESS

Manufacturing processes are the steps through which raw materials are transformed into a final product. The manufacturing process begins with the creation of the materials from which the design is made. These materials are then modified through manufacturing processes to become the required part. Manufacturing processes can include treating (such as heat treating or coating), machining, or reshaping the material. The manufacturing process also includes tests and checks for quality assurance during or after the manufacturing, and planning the production process prior to manufacturing.

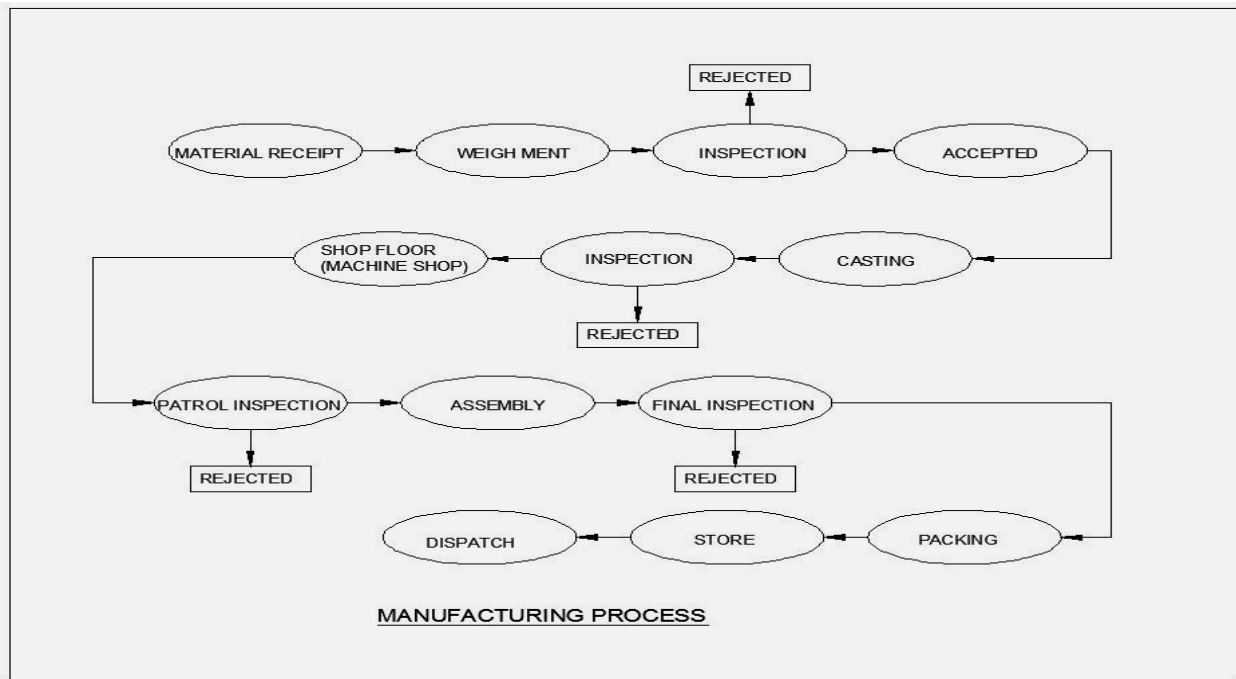


FIG 4.2

Welding is a process for joining similar metals. Welding joins metals by melting and fusing 1, the base metals being joined and 2, the filler metal applied. Welding employs pinpointed, localized heat input. Most welding involves ferrous-based metals such as steel and stainless steel. Weld joints are usually stronger than or as strong as the base metals being joined.

Welding is used for making permanent joints. It is used in the manufacture of automobile bodies, aircraft frames, railway wagons, machine frames, structural works, tanks, furniture, boilers, general repair work and ship building.

Several welding processes are based on heating with an electric arc, only a few are considered here, starting with the oldest, simple arc welding, also known as shielded metal arc welding (SMAW) or stick welding. In this process an electrical machine (which may be DC or AC, but nowadays is usually AC) supplies current to an electrode holder which carries an electrode which is normally coated with a mixture of chemicals or flux. An earth cable connects the work piece to the welding machine to provide a return path for the current. The weld is initiated by tapping ('striking') the tip of the electrode against the work piece which initiates an electric arc. The high temperature generated (about

6000°C) almost instantly produces a molten pool and the end of the electrode continuously melts into this pool and forms the joint.

4.4 INSPECTION:

Critical appraisal involving examination, measurement, testing, gauging, and comparison of materials or items. An inspection determines if the material or item is in proper quantity and condition, and if it conforms to the applicable or specified requirements. Inspection is generally divided into three categories: (1) Receiving inspection, (2) In-process inspection, and (3) Final inspection. In quality control (which is guided by the principle that "Quality cannot be inspected into a product") the role of inspection is to verify and validate the variance data; it does not involve separating the good from the bad.

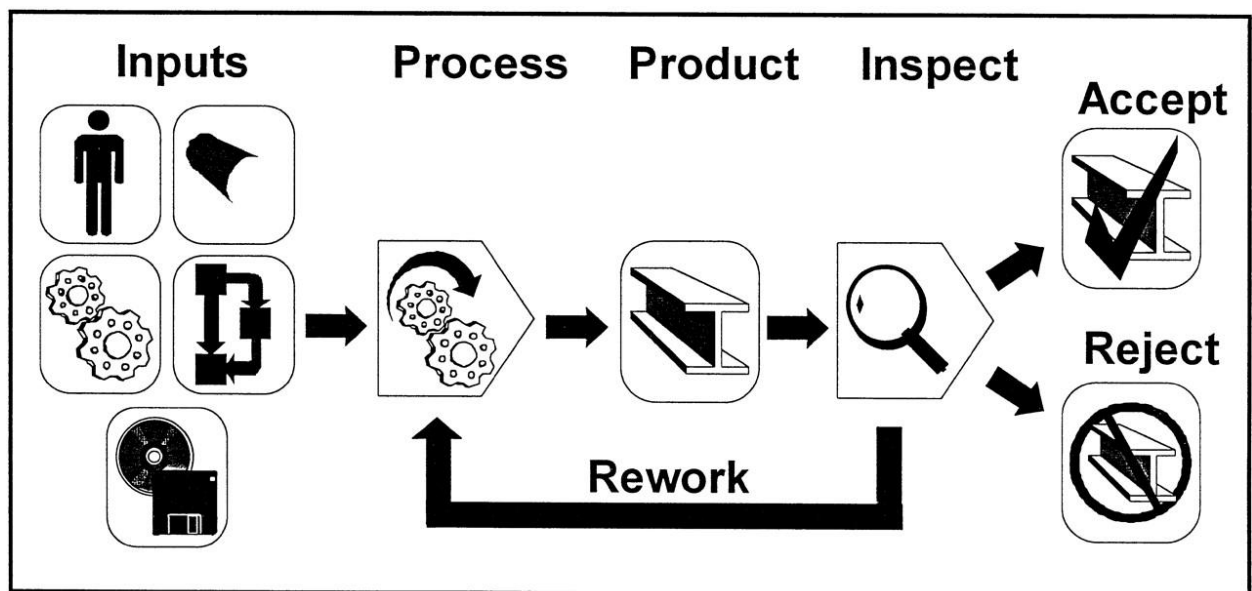


FIG 4.3

4.5 ASSEMBLY:

An assembly line is a manufacturing process (most of the time called a progressive assembly) in which parts (usually interchangeable parts) are added as the semi-finished assembly moves from work station to work station where the parts are added in sequence until the final assembly is produced. By mechanically moving the parts to the assembly work and moving the semi-finished assembly from work station to work station, a finished product can be assembled much faster and with much less labor than by having workers carry parts to a stationary piece for assembly.

4.6 NEED FOR AUTOMATION

Nowadays almost all the manufacturing process is being automated in order to deliver the products at a faster rate. The manufacturing operation is being automated for the following reasons.

- ✧ To achieve mass production
- ✧ To reduce man power
- ✧ To increase the efficiency of the plant
- ✧ To reduce the work load
- ✧ To reduce the production cost
- ✧ To reduce the production time
- ✧ To reduce the material handling
- ✧ To reduce the fatigue of workers
- ✧ To achieve good product quality
- ✧ Less Maintenance

4.7 HARDWARE REQUIREMENTS IN DETAIL:

4.7.1 POWER SUPPLY UNIT:

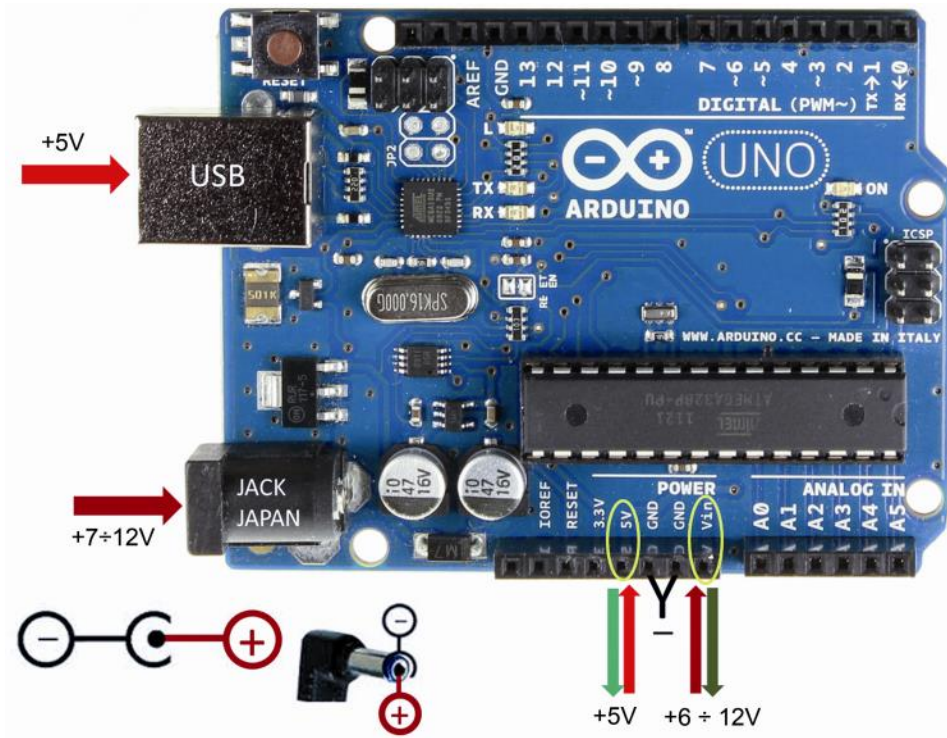


FIG 4.4

- Must be a DC adapter (i.e. it has to put out DC, not AC);
- should be between 9V and 12V DC (see note below);
- must be rated for a minimum of 250mA current output, although you will likely want something more like 500mA or 1A output, as it gives you the current necessary to power a servo or twenty LEDs if you want to.
- Must have a 2.1mm power plug on the Arduino end, and
- The plug must be "centre positive", that is, the middle pin of the plug has to be the + connection.

4.7.1.1 INTRODUCTION

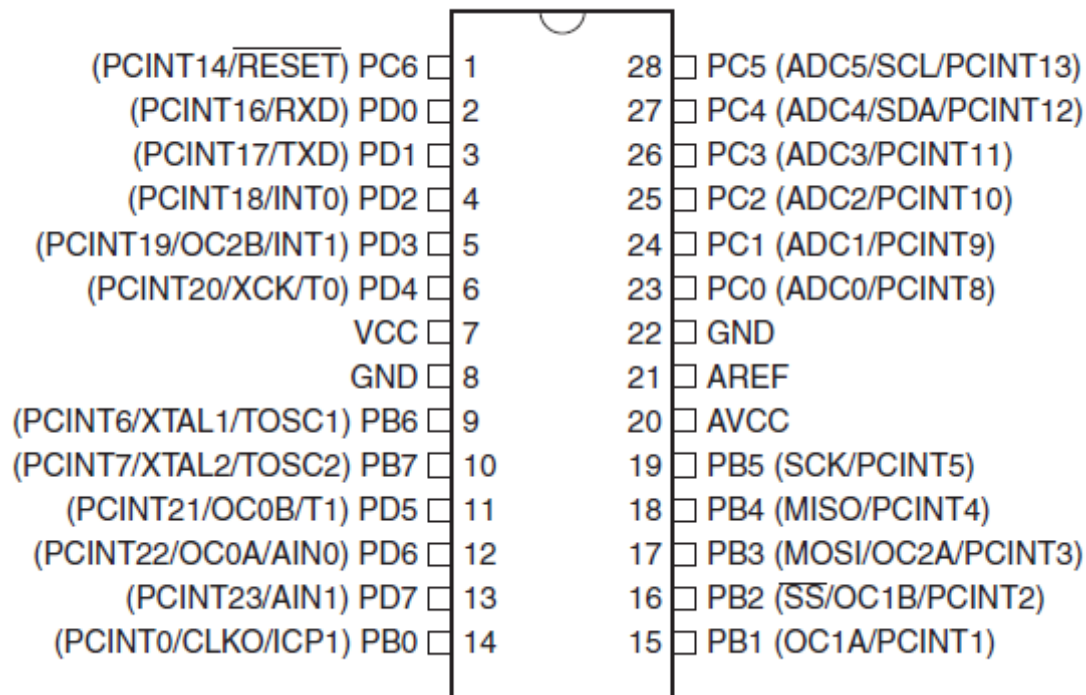
ATmega-328 is basically an Advanced Virtual RISC (AVR) micro-controller. It supports the data up to eight (8) bits. ATmega-328 has 32KB internal builtin memory. This micro-controller has a lot of other characteristics.

ATmega 328 has 1KB Electrically Erasable Programmable Read Only Memory (EEPROM). This property shows if the electric supply supplied to the micro-controller is removed, even then it can store the data and can provide results after providing it with the electric supply. Moreover, ATmega-328 has 2KB Static Random Access Memory (SRAM). Other characteristics will be explained later. ATmega 328 has several different features which make it the most popular device in today's market. These features consist of advanced RISC architecture, good performance, low power consumption, real timer counter having separate oscillator, 6 PWM pins, programmable serial USART, programming lock for software security, throughput up to 20 MIPS etc. ATmega-328 is mostly used in Arduino.

ATmega328 is an eight (8) bit micro-controller. It can handle the data sized of up to eight (8) bits. It is an AVR based micro-controller. Its built in internal memory is around 32KB. It operates ranging from 3.3V to 5V. It has an ability to store the data even when the electrical supply is removed from its biasing terminals. Its excellent features include the cost efficiency, low power dissipation, programming lock for security purposes, real timer counter with separate oscillator.

4.7.1.2 PIN DIAGRAM

Atmega328



ATmega328 Pinout

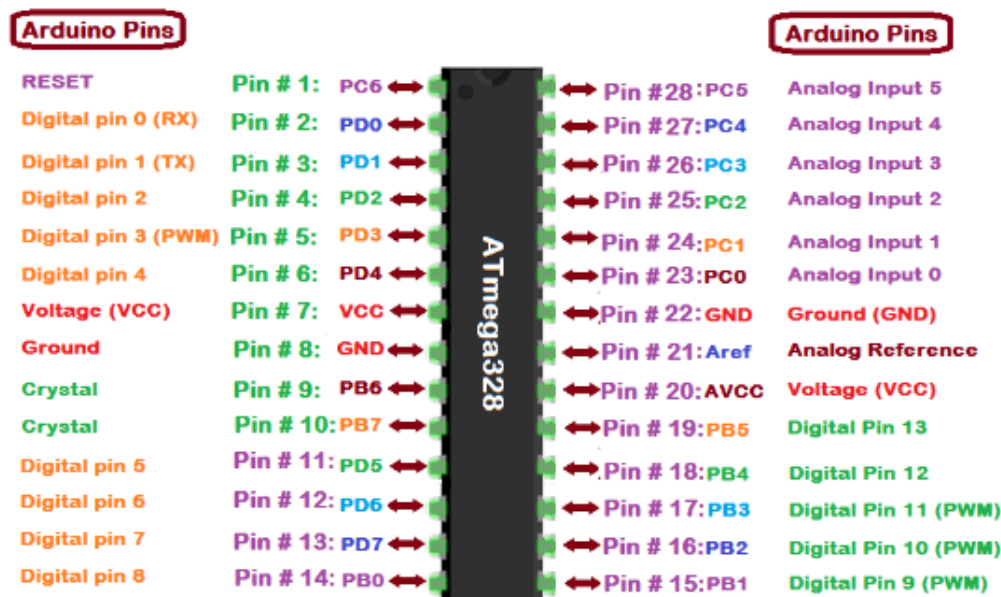


FIG 4.5

4.7.1.3 BLOCK DIAGRAM

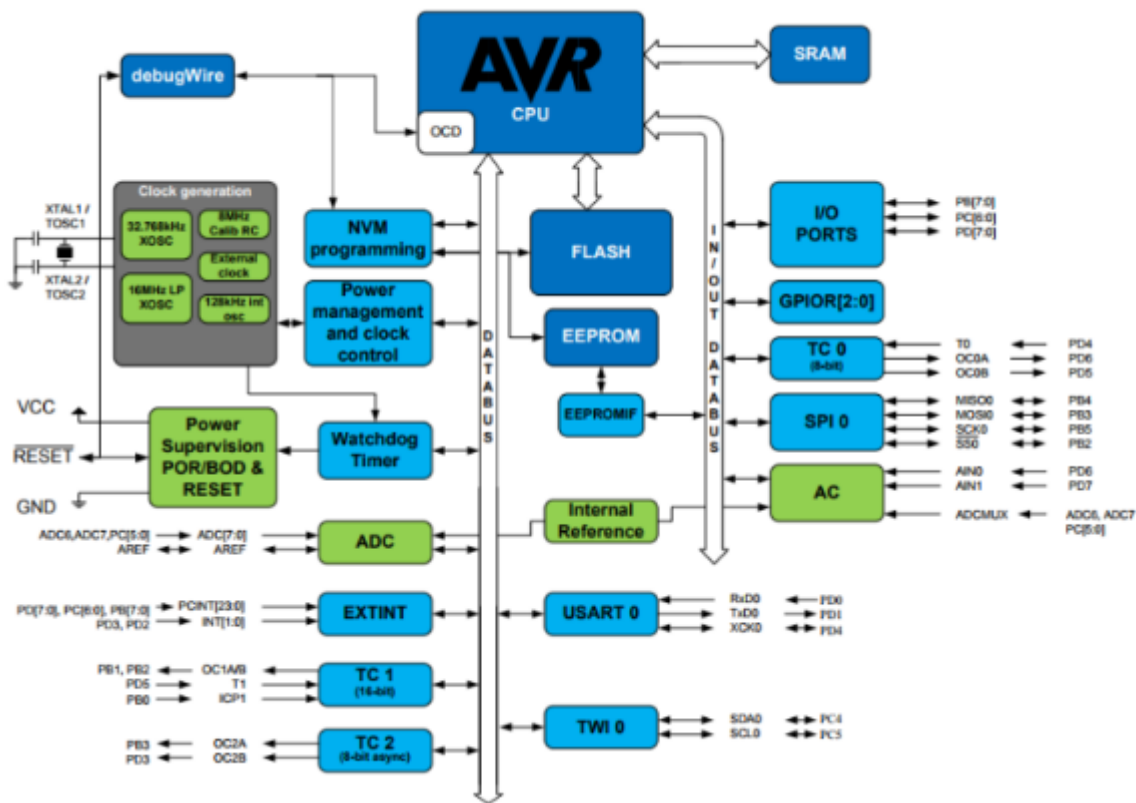


FIG 4.6

4.7.1.4 PIN DESCRIPTION

VCC Digital supply voltage. 5.2.2.

GND Ground. 5.2.3.

Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2 Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

Port C (PC[5:0]) Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source

current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PC6/RESET If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset. The various special features of Port C are elaborated in the Alternate Functions of Port C section.

Port D (PD[7:0]) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

AVCC AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC[6:4] use digital supply voltage, VCC.

AREF AREF is the analog reference pin for the A/D Converter.

ADC[7:6] (TQFP and VFQFN Package Only) In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

4.7.1.5 ARCHITECHURE DIAGRAM

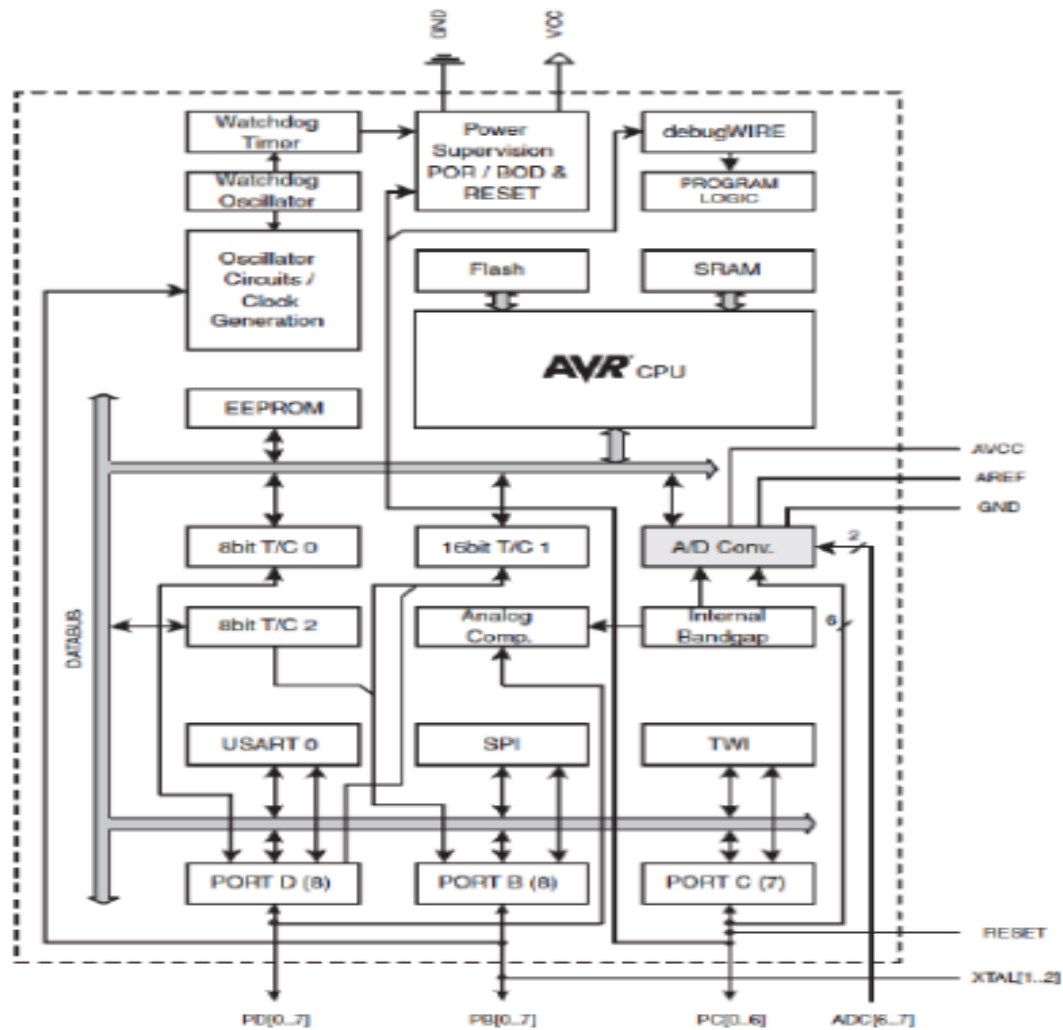


FIG 4.7

AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory. The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations.

One of these address pointers can also be used as an address pointer for look up tables in Flash program memory.

These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section. The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation. Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space.

Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction. Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture. The memory spaces in the AVR architecture are all linear and regular memory maps. A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority. The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, this device has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

ALU – Arithmetic Logic Unit The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are

executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See Instruction Set Summary section for a detailed description.

Status Register The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

Status Register When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F. Name: SREG Offset: 0x5F Reset: 0x00 Property: When addressing as I/O Register: address offset is 0x3F

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Global Interrupt Enable The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The Ibit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Copy Storage The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

Half Carry Flag The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Flag is useful in BCD arithmetic. See the Instruction Set Description for detailed information.

Sign Flag, $S = N \oplus V$ The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the Instruction Set Description for detailed information.

Two's Complement Overflow Flag The Two's Complement Overflow Flag V supports two's complement arithmetic. See the Instruction Set Description for detailed information.

Negative Flag The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the Instruction Set Description for detailed information.

Zero Flag The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the Instruction Set Description for detailed information.

Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. See the table for Stack Pointer details.

Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. They

have determined priority levels: The lower the address the higher is the priority level. RESET has the highest priority, and next is INTO – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOTRST Fuse. When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed. There are basically two types of interrupts: The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority. The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered. When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served. The Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software. When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction.

MEMORY DESCRIPTION

ATmega 328 has three types of memories e.g. EEPROM, SRAM etc.

Flash Memory has 32KB capacity. It has an address of 15 bits. It is a Programmable Read Only Memory (ROM). It is non volatile memory. The ATmega328/P contains 32Kbytes On-

chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16. For software security, the Flash Program memory space is divided into two sections - Boot Loader Section and Application Program Section in the device . The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega328/P Program Counter (PC) is 14 bits wide, thus addressing the 16K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in Boot Loader Support – Read-While-Write Self-Programming. Refer to Memory Programming for the description on Flash data serial downloading using the SPI pins. Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction. Timing diagrams for instruction fetch and execution are presented in Instruction Execution Timing

SRAM stands for Static Random Access Memory. It is a volatile memory i.e. data will be removed after removing the power supply. Device SRAM Memory is organized. The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used. The lower 2303 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 2K locations address the internal data SRAM.

The five different addressing modes for the data memory cover:

- Direct – The direct addressing reaches the entire data space.
- Indirect with Displacement – The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

- Indirect – In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

- Indirect with Pre-decrement – The address registers X, Y, and Z are decremented.

- Indirect with Post-increment – The address registers X, Y, and Z are incremented. The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 2K bytes of internal data SRAM in the device are all accessible through all these addressing modes.

EEPROM stands for Electrically Erasable Programmable Read Only Memory. It has a long term data. The ATmega328/P contains 1K bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

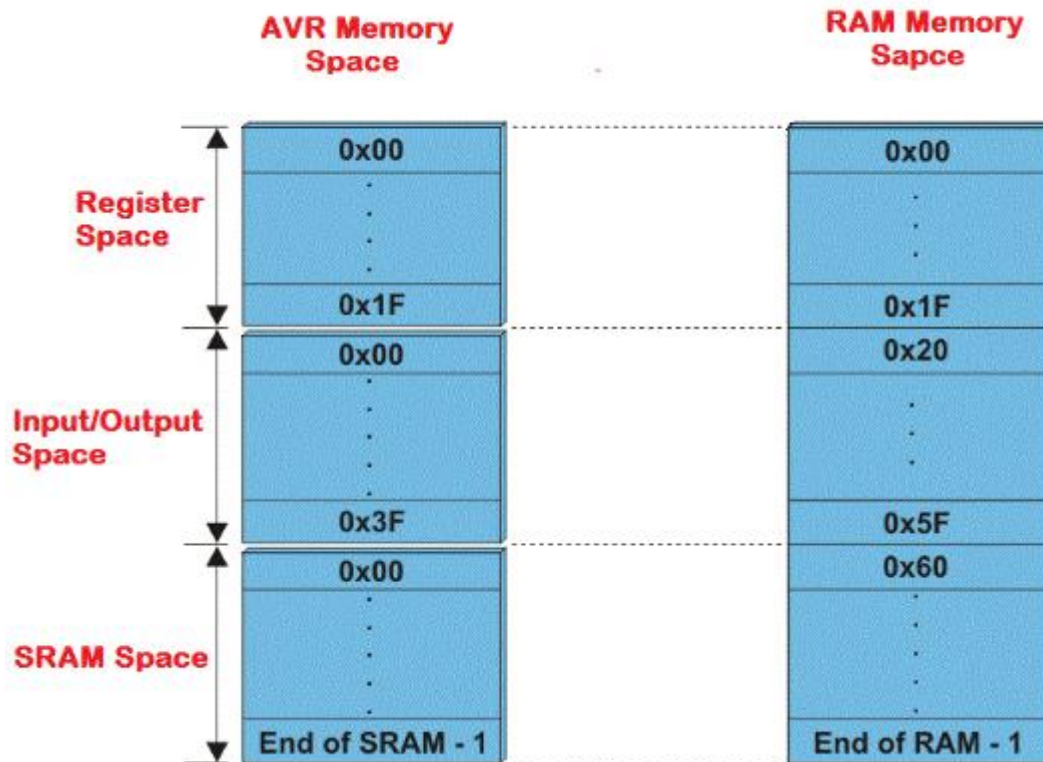


FIG 4.8

REGISTERS

- ATmega-328 has thirty two (32) General Purpose (GP) registers.
- These all of the registers are the part of Static Random Access Memory (SRAM).

7	0	Addr.
R0		0x00
R1		0x01
R2		0x02
...		
R13		0x0D
R14		0x0E
R15		0x0F
R16		0x10
R17		0x11
...		
R26		0x1A
R27		0x1B
R28		0x1C
R29		0x1D
R30		0x1E
R31		0x1F

FIG 4.9

4.7.1.6 ATmega328 and Arduino

- ATmega-328 is the most micro-controller that is used while designing.
- ATmega-328 is the most important part of Arduino.
- The program is uploaded on the AVR micro-controller attached on Arduino.



FIG 4.10

4.7.1.7 UART

The first communication protocol we'll cover is Universal Asynchronous Receiver/Transmitter (UART). UART is a form of serial communication because data is transmitted as sequential bits (we'll get to this in a bit). The wiring involved with setting up UART communication is very simple: one line for transmitting data (TX) and one line for receiving data (RX). As you may expect, the TX line is used to for the data to send device, and the RX line is used to receive data. Together the TX and RX lines of a device using serial communication form a serial port through which communication can occur.

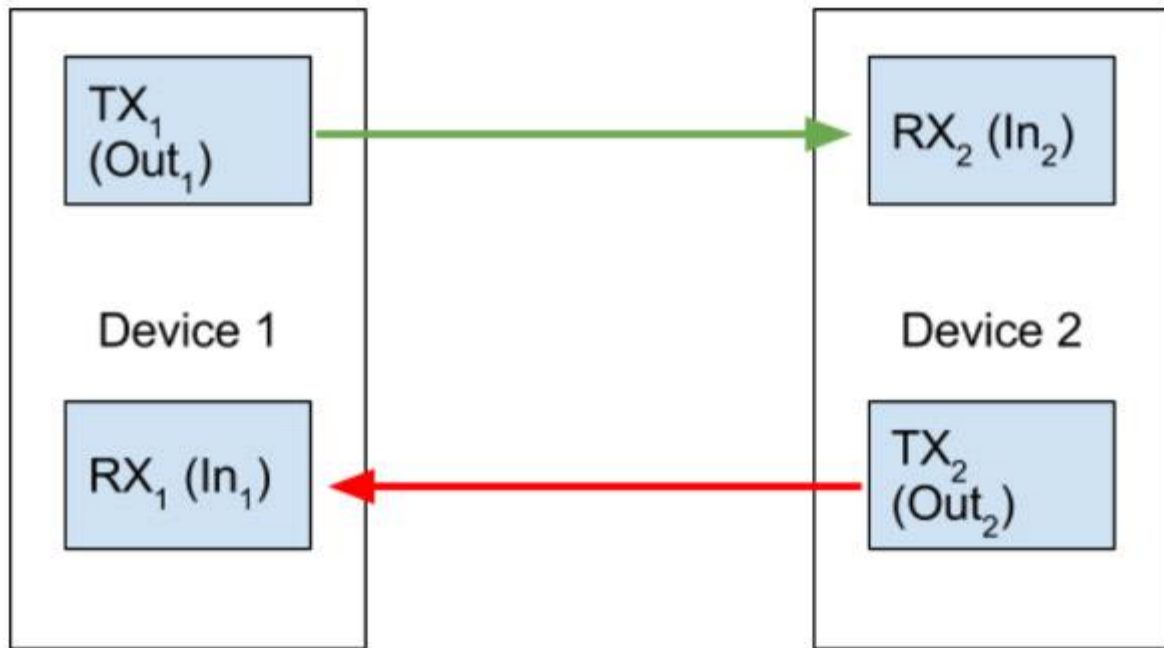


FIG : 4.11

Figure: Hardware connection diagram for UART

The term UART actually refers to the onboard hardware that manages the packaging and translation of serial data. For a device to be able to communicate via the UART protocol, it must have this hardware! On the Arduino Uno, there is one serial port dedicated for communication with the computer the Arduino is connected to. That's right! USB, which stands for Universal Serial Bus, is a serial port! On the Arduino Uno, this USB connection is broken out through onboard hardware into two digital pins, GPIO 0 and GPIO 1, which can be used in projects that involve serial communication with electronics other than the computer.

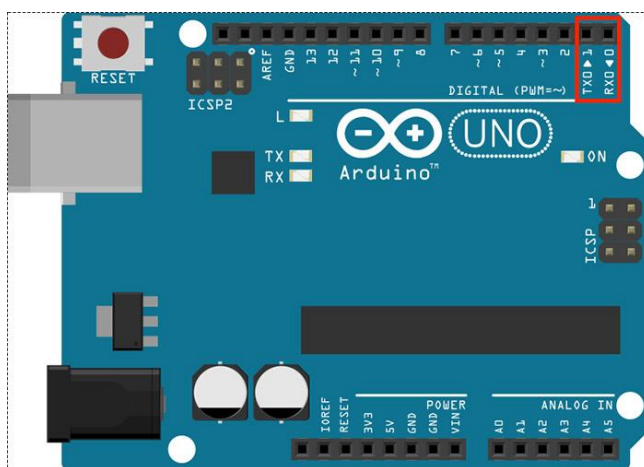


FIG 4.12

Figure: GPIO 0 and GPIO 1 are Serial RX and TX. Any GPIO pin can be used as Serial RX or TX with the Software Serial library

You can also use the Software Serial Arduino library (SoftwareSerial.h) to use other GPIO pins as Serial RX and TX lines.

UART is called asynchronous because the communication does not depend on a synchronized clock signal between the two devices attempting to communicate with each other. Because the communication speed is not defined via this steady signal, the “sender” device cannot be certain that the “receiver” obtains the correct data. Therefore, the devices break data into fixed-size chunks to ensure that the data received is the same as the data that was sent.

The UART data packet looks something like this:

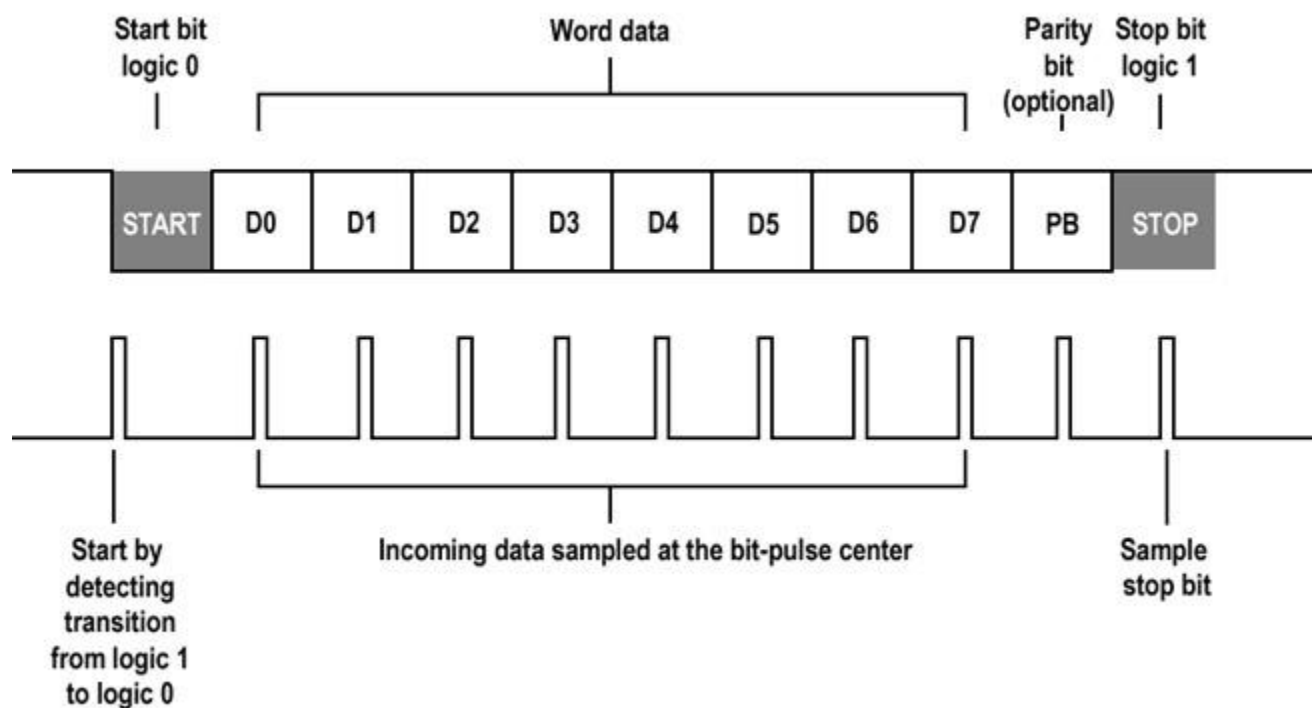


FIG 4.13

Devices that communicate via UART send packets of pre-defined size that contain additional information regarding the start and end of the message and confirmation of whether the message was received correctly. For example, to begin communication, the transmitting device

pulls the transmit line low, indicating the start of a data packet. Long-term, however, this means that UART is slower compared to a synchronized form of communication because only a portion of the data transmitted is for the devices' applications (the rest is for the communication itself!)

When implementing UART serial communication on most embedded platforms such as the Arduino, the user does not have to deal with communication at the bit level. Instead, the platform often provides higher level software libraries that are the only aspect of the communication process the user has to deal with. In the Arduino platform, users can use the *Serial* and *SoftwareSerial* libraries to implement UART communication for their projects.

Here's a brief C++ reference for Arduino Serial and SoftwareSerial initialization and use.

Serial and SoftwareSerial Method	Purpose	Code	Explanation
<i>Constructor (SoftwareSerial only)</i>	Define the GPIO pins that will serve as the UART RX and TX lines	<code>SoftwareSerial comms(2, 3);</code>	Defines a serial connection with RX line on GPIO 2 and TX line on GPIO 3.
<i>Begin</i>	Define the baud rate (transmission speed) for the serial connection in range 4800 to 115200	<code>comms.begin(9600);</code>	Communication on the "comms" serial port will occur at 9600 baud.
<i>Print</i>	Write byte data converted into human-readable characters over serial connection	<code>comms.println("Hello World");</code>	Writes bytes equivalent to <i>Hello World</i> (human-readable characters)
<i>Write</i>	Write raw byte data over the serial connection	<code>comms.write(45);</code>	Writes byte with value 45.
<i>available</i>	Evaluates to true when data is available over the serial connection	<code>if (comms.available())</code>	Enter <i>if</i> statement if there is data available to read over the serial connection
<i>Read</i>	Read data from the serial connection	<code>comms.read();</code>	Reads from serial connection

One important aspect to note about UART communication is that it is designed for communication between only two devices at a time. Because the protocol only sends bits indicating the start of a message, the message content, and the end of a message, there is no method of differentiating multiple transmitting and receiving devices on the same line. If more than one device attempts to transmit data on the same line, bus contention occurs, and the receiving devices will most likely receive garbage unusable data!

Furthermore, UART is half-duplex, which means that even though communication can occur bi-directionally, both devices cannot transmit data to each other at the same time. In a project where two Arduino's are communicating with each other via a serial connection, for example, this would mean that at a given time, only one of the Arduinos can be "talking" to the other. For most applications, however, this fact is relatively unimportant and not disadvantageous in any way.

SPI

The next communication protocol we'll cover is Serial Peripheral Interface (SPI). SPI is different from UART in several key ways:

- Synchronous
- Follows a master-slave model, where there is one master device and multiple slave devices
- More than two lines required for implementation

The hardware connection diagram for SPI is slightly more complicated, and looks something like this:

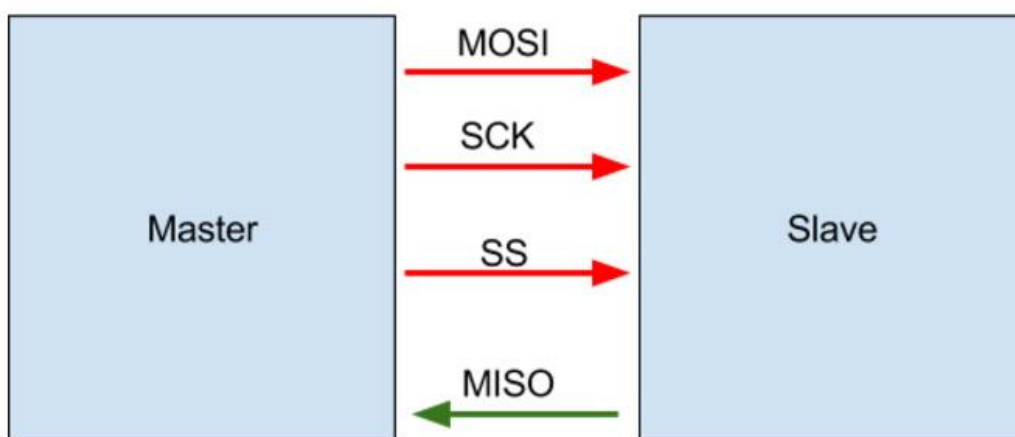


FIG 4.14

- **MOSI (“Master Out Slave In”)**: Data transmission line from master to slave
- **SCK (“Clock”)**: Clock line defining transmission speed and transmission start/end characteristics
- **SS (“Slave Select”)**: Line for master to select a particular slave to communicate with
- **MISO (“Master In Slave Out”)**: Data transmission line from slave to master

The first feature about SPI to note is that it follows a master-slave model. This means that the communication relies on defining one device as a master, and other devices as slaves. This creates a hierarchy between the devices signifying which device is effectively “in control” of the others. We’ll discuss this master-slave model shortly when we describe what a sample communication between a master and slave device might look like.

We noted earlier that multiple slaves can be connected to a single master device. The hardware diagram for such a system might look something like this:

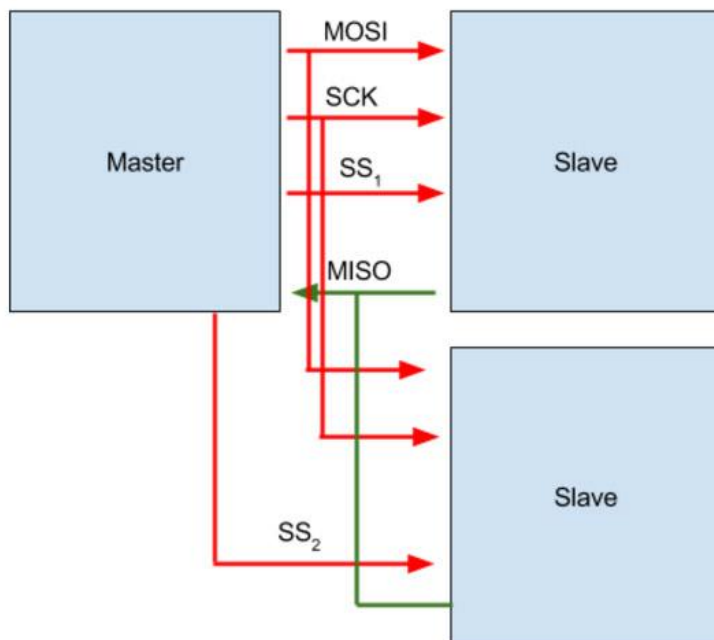


FIG 4.15

SPI does not require a separate transmit and receive line for each slave that is connected to the master. One common receive line (MISO) and one common transmit line (MOSI), along with one common clock (SCK) line are connected between all the slaves and the master. The master

device decides which slave it is communicating with through a separate SS line for each slave. This means that each additional slave that the master communicates with requires another GPIO pin on the master's side.

SPI is synchronous, which means that communication between master and slave is closely tied to a clock signal (a square wave of fixed-frequency) defined by the master device. Here we see one of the direct effects of the master-slave model. The master device drives the communication by specifying the communication speed through the clock signal, and the slaves respond by communicating at that speed. This definition of speed works for any communication speed the master decides on (within the maximum speed tolerable by the slave devices).

In SPI, two specific features of the clock signal determine the start and end of data transmission: the clock polarity (CPOL) and clock phase (CPHA). CPOL refers to the idle state (either low or high) of the clock signal. To conserve power, devices will put the clock line at the idle state when not communicating with any slaves, and the two options available for this idle state are either low or high. CPHA refers to the edge of the clock signal upon which data is captured. A square wave has two edges (the rising edge and the falling edge), and depending on the CPHA setting, data is captured either on the rising edge or the falling edge.

There are four distinct combinations of CPOL and CPHA, that can be summarized in a table.

	CPHA = 0	CPHA = 1
	("first edge" of clock signal)	("second edge" of clock signal)
CPOL = 0 (idle 0)	• "SPI mode 0" • Data capture begins when clock goes from 0 (idle) to 1 for the first time. • Data capture occurs on the rising edge of the clock signal (0→1)	• "SPI mode 1" • Data capture begins when clock goes from 0 (idle) to 1 for the first time. • Data capture occurs on the falling edge of the clock signal (1→0)
CPOL = 1	• "SPI mode 2"	• "SPI mode 3"

- Data capture begins when clock goes from 1 (idle) to 0 for the first time.
 - Data capture occurs on the rising edge of the clock signal (0→1)
- Data capture begins when clock goes from 1 (idle) to 0 for the first time.
 - Data capture occurs on the falling edge of the clock signal (1→0)

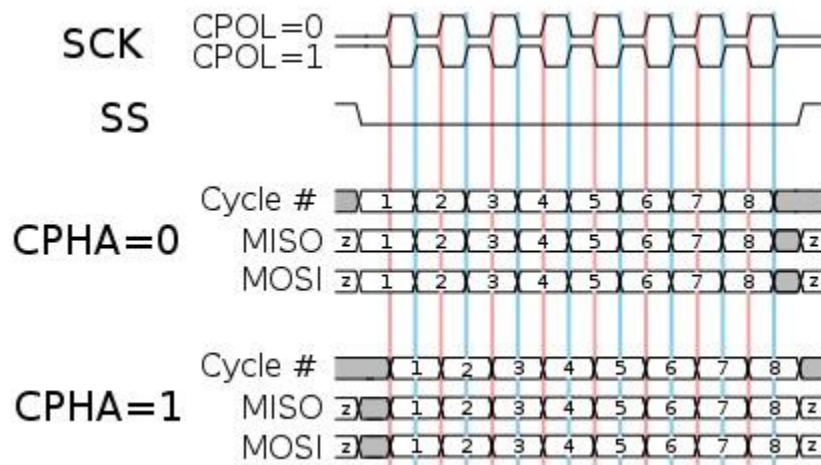


Figure 6. CPOL and CPHA settings visualized

Now we'll focus on SPI implementation on the Arduino using the Arduino as the master device (SPI.h). The SPI digital pin connections for SCK, MOSI, and MISO are predefined on Arduino boards. For the Arduino Uno, the connections are as follows:

- **SCK:** GPIO 13 or ICSP 3
- **MOSI:** GPIO 11 or ICSP 4
- **MISO:** GPIO 12 or ICSP 1
- **SS:** GPIO 10

Any digital pin can be also used for the SS pin. To select the device, this digital pin must be driven low.

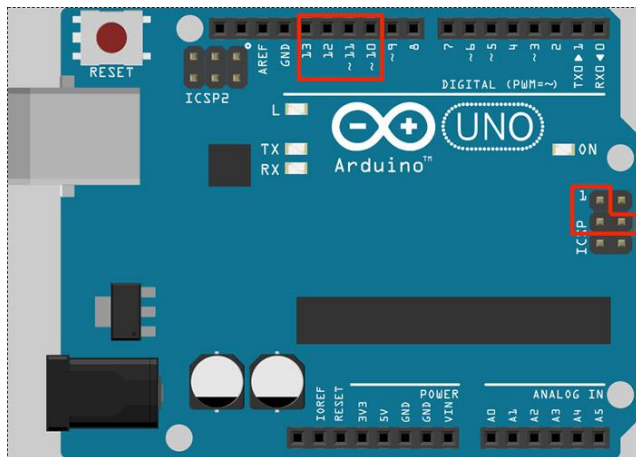


FIG 4.15

Here's a brief C++ reference for Arduino SPI initialization and use.

SPI Method	Purpose	Code	Explanation
Constructor	Define the clock speed, data bit order (most significant bit first or least significant bit first), and SPI mode	<code>SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));</code>	Defines a SPI connection at 14MHz in SPI Mode 0 and data transmission occurring with the most significant bit first
<i>digitalWrite</i>	Select the slave device attached to this GPIO pin	<code>digitalWrite(10, LOW);</code>	Drive the GPIO pin low to select the slave device. Drive the pin high afterwards to de-select the slave.
Transfer	Transfer the byte to the selected slave	<code>SPI.transfer(0x00);</code>	Send a value of 0 as a byte
<i>endTransaction</i>	Ends the SPI transaction (should be called after a <code>digitalWrite(high)</code> is called on the SS line)	<code>SPI.endTransaction();</code>	Ends the SPI transaction

SPI is full-duplex, which means that communication always occurs bidirectionally, even if the application only requires data transfer occur in one direction. SPI full-duplex data transfer is usually implemented with a shift register. (Check out the appendix for a tutorial on shift

registers!) This means that as each bit is read, the bits before it are shifted over by one place. The foremost bit is subsequently popped off and sent back through the SPI connection to the other device!

I²C

Inter-integrated circuit (I²C), pronounced either “i-squared-c” or “i-two-c,” is the final communication protocol we’ll cover in this tutorial. Though its implementation is the most complicated of the three protocols, I²C addresses several drawbacks in the other communication protocols, giving it an advantage over the others in some applications. These include:

- The ability to connect multiple masters to multiple slaves
- Synchronicity (just like SPI), which means higher speed communication
- Simplicity: implementation only requires two wires and some resistors

At the hardware level, I²C is a two-wire interface—the only two wires required for an I²C connection are a data line (called *SDA*) and a clock line (called *SCL*). The data and clock lines are pulled high in their idle states, and when data needs to be sent over the connection, the lines are pulled low through some MOSFET circuits. For the purposes of this tutorial, we will not discuss the MOSFET operation inside I²C circuitry. The important fact to take away is that the system is *open-drain*, which means that the lines can only be driven low by the devices. Thus when using I²C in projects, it is crucial to include *pull-up resistors* (usually something like 4.7kΩ) to ensure that the line is actually pulled high in the idle state.

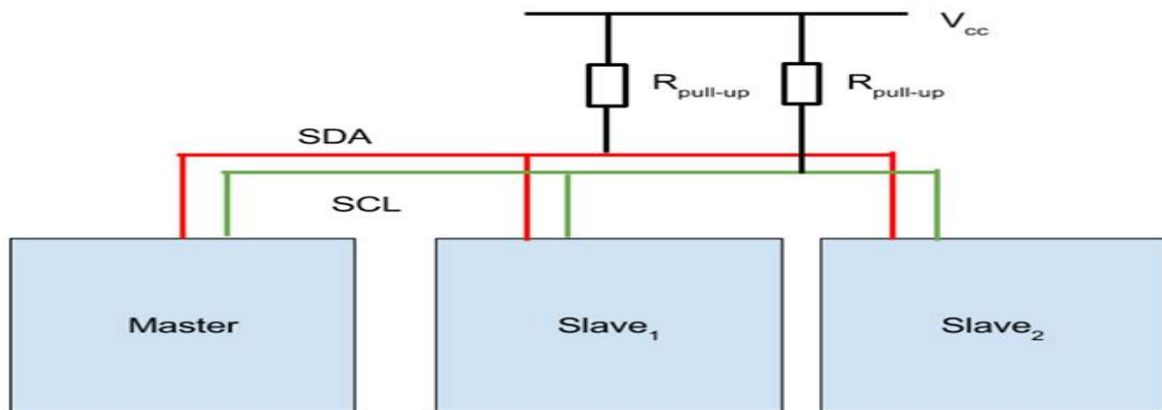
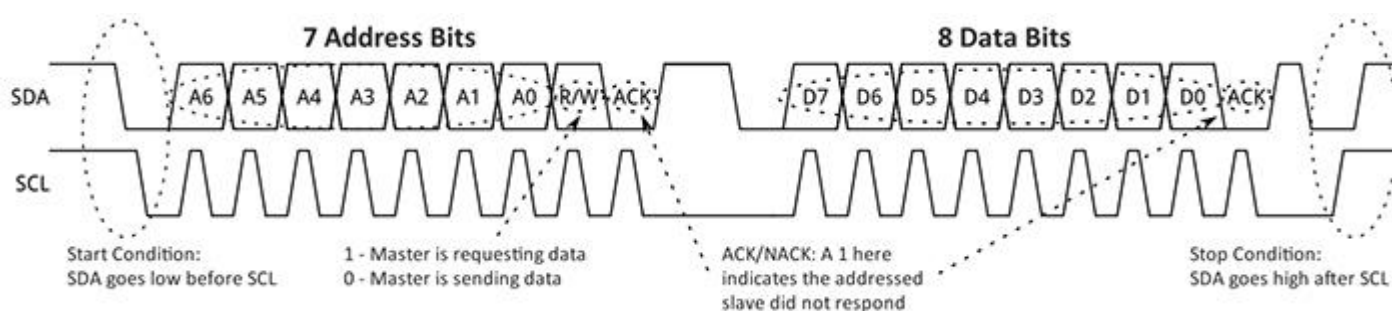


FIG 4.16

I²C is unique because it solves the issue of interfacing with multiple slave devices through addressing. Just like in SPI communication, I²C makes use of a master-slave model to establish the “hierarchy” of communication. However, instead of selecting slaves through separate digital lines, masters select slaves through their unique byte addresses. These addresses might look something like this: 0x1B. This means that connecting more slaves to a master will not require additional digital lines. As long as each slave has a unique address, the application will still be able to differentiate the slaves on these addresses alone. You can think of these addresses like names. To call a slave’s functionality, the master merely calls its name, and only that slave responds.

The address and data portions of the I²C communication line might look something like this.



Notice the ACK and NACK bits on the communication line. These bits denote whether or not the addressed slave responds to the communication—a way of periodically checking whether or not communication is occurring as expected. These bits, of course, are unrelated to the address or data bits being sent, but in the grand scheme of the communication, they add very

little extra time to the communication compared to the many start and end bits and pauses that occur in protocols like UART.

I²C gives slaves freedom to decide what communication requests might look like. Different slaves require different bytes to be written over in different orders over the SDA line to write and request information from them. In some accelerometer modules, for example, bytes are written to denote which hardware register the master wants to read before any read requests are made. For these specifications, the user must refer to the slaves' datasheets for device addresses, register addresses, and device settings.

On the Arduino, I²C implementation occurs through the Wire library (Wire.h). The Arduino can be configured as either an I²C master or slave device. On the Arduino Uno, the connections are as follows:

- **SDA:** Analog Pin 4
- **SCL:** Analog Pin 5

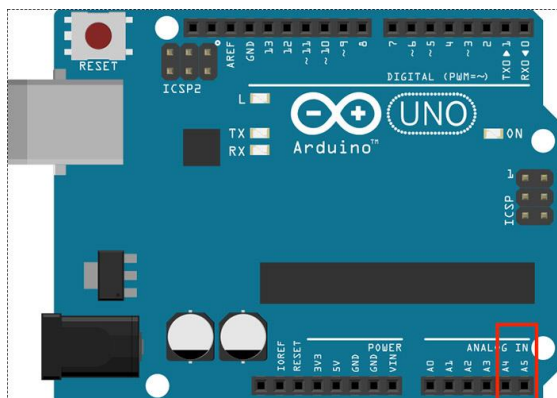


FIG 4.17

Here's a brief C++ reference for Arduino I²C initialization and use.

I ² C Method	(Wire) Purpose	Code	Explanation
Begin	Initiate the library and join the I ² C bus as either a master or slave	<code>Wire.begin();</code>	Joins the I ² C bus as a master. If an address is specified as a parameter to the method, the Arduino joins the bus as a slave with that address.
beginTransmission	For Arduino configured as a I ² C master: initiate transmission with the slave that has the given address	<code>Wire.beginTransmission(0x68);</code>	Begins transmission with the slave that has the hexadecimal address 0x68
Write	Write the byte data over the I ² C bus	<code>Wire.write(0x6B);</code>	Writes byte data of value 0x6B over the I ² C bus
requestFrom	Request a specified number of bytes from the slave with the given address; Has the option of releasing the I ² C line or holding onto it for further communication	<code>Wire.requestFrom(0x68, 6, true);</code>	Requests 6 bytes from the slave that has the address 0x68. Releases the I ² C line after the request is complete. If the last parameter is <i>false</i> , the Arduino will hold onto the

	The requested bytes are put into a buffer to be subsequently read through <code>Wire.read()</code> calls.	I²C line for further communication, not allowing other devices to communicate over it.
Read	Read bytes put into the buffer after transmissions from the slave device. This method is called <i>after</i> a call to <code>Wire.requestFrom</code>	Reads one byte from the buffer (after a call to <code>requestFrom</code>) has been made. To read two bytes from the buffer, this method must be called twice: Example: <pre>Wire.requestFrom(0x68, 2, true);</pre> <pre>Wire.read();</pre> <pre>Wire.read();</pre>
endTransmission	Ends the current transmission; Has the option of releasing the I²C line or holding onto it for further communication	End transmission <code>Wire.endTransmission(true);</code> and releases the I²C line.

Multiple masters can be connected over an I²C bus just by connecting their SDA and SCL lines to the bus's lines. However, only one master can communicate with slaves at a time, because having multiple devices attempting to communicate with each other would lead to bus contention. Similarly, communication cannot occur bidirectionally from master to slave and

from slave to master at the same time because that would also lead to bus contention. This makes I²C half-duplex, just like UART!

APPLICATIONS

- A complete package including ATmega 328 and Arduino can be used in several different real life applications.
- It can be used in embedded systems projects.
- It can also be used in robotics.
- Quad-copter and even small aeroplanes can also be designed through it.
- Power monitoring and management systems can also be prepared using this device.

4.7.2 LIQUID CRYSTAL DISPLAY (LCD)

INTRODUCTION

A liquid crystal display (LCD) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly. They are used in a wide range of applications, including computer monitors television, instrument panels, aircraft cockpit displays, etc. LCDs have displaced cathode ray tube (CRT) displays in most applications. They are usually more compact, lightweight, portable, less expensive, and more reliable.

Each pixel of an LCD typically consists of a layer of molecules aligned between two transparent electrodes and two polarizing filters, the axes of transmission of which are (in most of the cases) perpendicular to each other. With no actual liquid crystal between the polarizing filters, light passing through the first filter would be blocked by the second (crossed) polarizer. In most of the cases the liquid crystal has double refraction.

The surfaces of the electrodes that are in contact with the liquid crystal material are treated so as to align the liquid crystal molecules in a particular direction. Electrodes are made of a transparent conductor called Indium Tin Oxide (ITO).

Before applying an electric field, the orientation of the liquid crystal molecules is determined by the alignment at the surfaces of electrodes. In a twisted nematic device the surface alignment directions at the two electrodes are perpendicular to each other, and so the molecules arrange themselves in a helical structure, or twist. This reduces the rotation of the polarization of the incident light, and the device appears grey. If the applied voltage is large enough, the liquid crystal molecules in the center of the layer are almost completely untwisted and the polarization of the incident light is not rotated as it passes through the liquid crystal layer. This light will then be mainly polarized perpendicular to the second filter, and thus be blocked and the pixel will appear black. By controlling the voltage applied across the liquid crystal layer in each pixel, light can be allowed to pass through in varying amounts thus constituting different levels of gray.

4.7.2.1PIN DESCRIPTION

The most common used LCDs found in the market today are 1 line, 2 line, or 4 line LCDs which have only one controller and support at most of 80 characters, whereas LCDs supporting more than 80 characters make use of 2HD44780 controllers. More LCDs with 1 controller has 14 pins and LCDs with 2 controller has 16 pins.

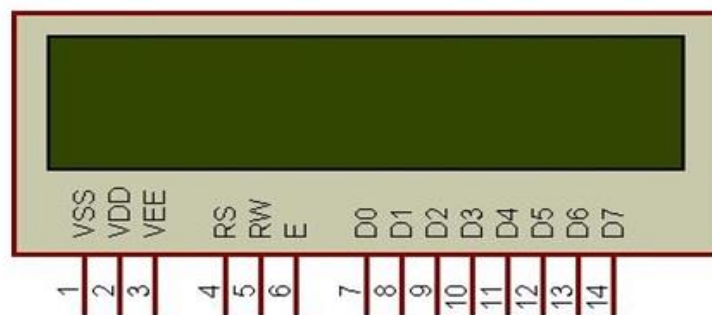


FIG 4.18

FEATURES

- 5 x 8 dots with cursor.
- Built-in controller (KS 0066 or Equivalent)
- + 5V power supply (Also available for + 3V)
- 1/16 duty cycle

- B/L to be driven by pin 1, pin 2 or pin 15, pin 16 or A.K (LED)
- N.V. optional for + 3V power supply.

PIN NO.,	SYMBOL	FUNCTION
1.	Vss	GND
2.	Vdd	+3V or +5V
3.	Vo	Contrast Adjustment
4.	RS	H/L Register Select Signal
5.	R/ $\bar{W}$	H/L Read and Write Signal
6.	E	H→L Enable Signal
7.	DB0	H/L Data Bus Line
8.	DB1	H/L Data Bus Line
9.	DB2	H/L Data Bus Line
8.	DB3	H/L Data Bus Line
11.	DB4	H/L Data Bus Line
12.	DB5	H/L Data Bus Line
13.	DB6	H/L Data Bus Line
14.	DB7	H/L Data Bus Line
15.	A/Vee	+4.2V for LED/Negative Voltage Output
16.	K	Power Supply for B/L (0V)

Pin details of LCD

DISPLAY CHARACTER ADDRESS CODE

Display Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DD RAM Address	00	01														0F
DD RAM Address	40	41														4F

Address code for displaying character

BF- BUSY FLAG

Busy flag is an status indicator flag for LCD. When we send a command or data to the LCD for processing, this flag is set (BF=1) and as soon as the instruction is executed

successfully this flag is cleared (BF=0). This is helpful in producing an exact amount of delay for the LCD processing. To read busy flag, the condition RS=0 and R/W=1 must be met and the MSB of the LCD data bus (D7) act as busy flag. When BF =1 means LCD is busy and will not accept next command or data and BF=0 means LCD is ready for the next command or data to process.

INSTRUCTION AND DATA REGISTER

There are two 8-bit registers in HD44780 controller Instruction and Data register. Instruction register corresponds to the register where you send commands to LCD e.g LCD shift command, LCD clear, LCD address etc. and Data register is used for storing data which is to be displayed on LCD. When the enable signal of the LCD is asserted, the data on the pins is latched in to the data register and data is then moved automatically to the DDRAM and hence is displayed on the LCD. Data register is not only used for sending data to DDRAM but also for CGRAM, the address where you want to send the data, is decided by the instruction you send to LCD. We will discuss more on LCD instruction set further in this chapter.

COMMANDS AND INSTRUCTION SET

Only the instruction register (IR) and the data register (DR) of the LCD can be controlled by the MCU. Before starting the internal operation of the LCD, control information is temporarily stored into these registers to allow interfacing with various MCUs, which operate at different speeds, or various peripheral control devices. The internal operation of the LCD is determined by signals sent from the MCU. These signals, which include register selection signal (RS), read/write signal (R/W), and the data bus (DB0 to DB7), make up the LCD instructions (Table 3).

No.	Instruction	Hex	Decimal
1	Function Set: 8-bit, 1 Line, 5x7 Dots	0x30	48
2	Function Set: 8-bit, 2 Line, 5x7 Dots	0x38	56
3	Function Set: 4-bit, 1 Line, 5x7 Dots	0x20	32
4	Function Set: 4-bit, 2 Line, 5x7 Dots	0x28	40
5	Entry Mode	0x06	6
6	Display off Cursor off (clearing display without clearing DDRAM content)	0x08	8
7	Display on Cursor on	0x0E	14
8	Display on Cursor off	0x0C	12
9	Display on Cursor blinking	0x0F	15
10	Shift entire display left	0x18	24
12	Shift entire display right	0x1C	30
13	Move cursor left by one character	0x10	16
14	Move cursor right by one character	0x14	20
15	Clear Display (also clear DDRAM content)	0x01	1

Commands

There are four categories of instructions that:

- Designate LCD functions, such as display format, data length, etc.
- Set internal RAM addresses
- Perform data transfer with internal RAM
- Perform the miscellaneous functions.

RELAY:



FIG 4.19

FEATURES:

- RW Series Relay covers switching capacity by 10A is spite of miniature size to comply with user's wide selection.

- RWH is approved C-UL & TÜV safety standard.
- The employment of suitable plastic materials is applied under high temperature condition and various chemical solutions.
- Complete protective construction is designed form dust and soldering flux. If required, plastic sealed type is available for washing procedure.
- 12A at 120VAC for RW & 12A at 240VAC for RWH are UL approved.

SPECIFICATIONS

Rated voltage	5 VDC	12 VDC	24 VDC	48 VDC
Rated current	80.0 mA	33.3 mA	16.7 mA	8.96 mA
Coil resistance	62.5 Ω	360 Ω	1,440 Ω	5,358 Ω
Must operate voltage	70% max. of the rated voltage			
Must release voltage	10% min. of the rated voltage			
Max. voltage	180% of rated voltage (at 23°C)			
Power consumption	Approx. 400 mW			Approx. 430 mW

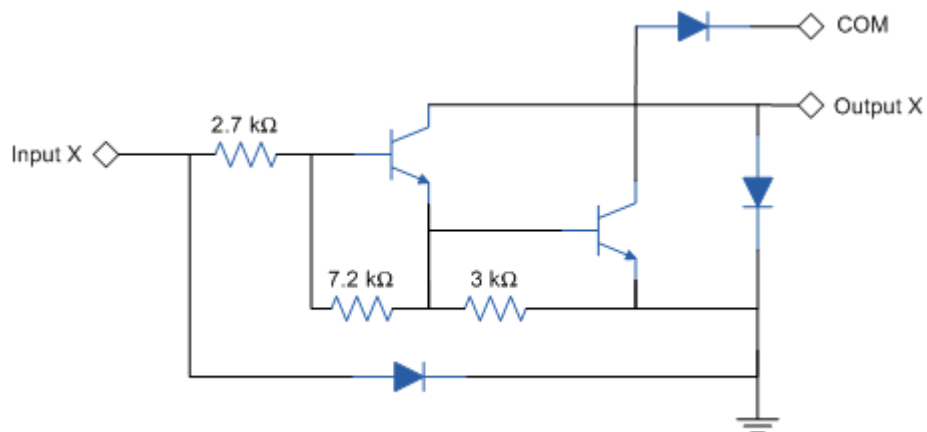
APPLICATIONS:

- Domestic Appliances
- Office Machines
- Audio Equipment
- Coffee-Pots
- Control units

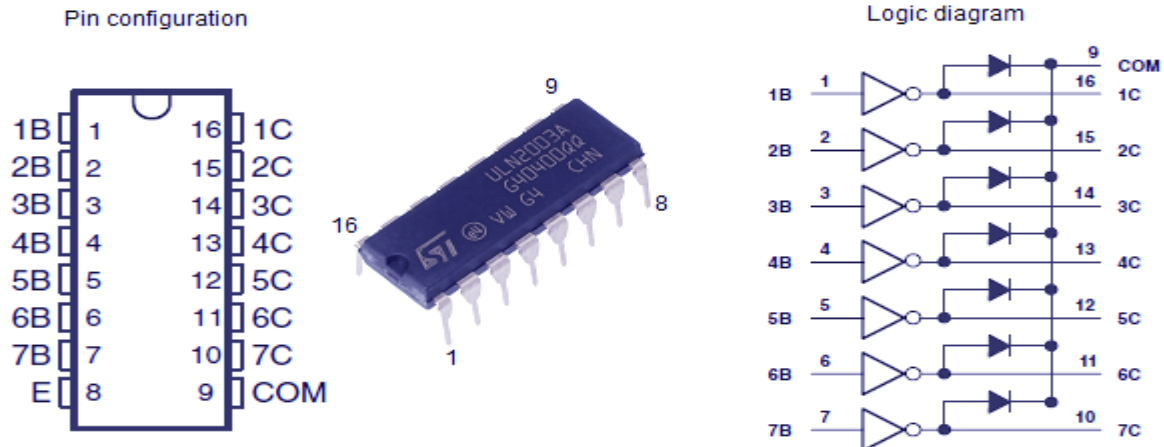
4.7.3 RELAY DRIVER – ULN2003

ULN2003 is a high voltage and high current Darlington array IC. It contains seven open collector darlington pairs with common emitters. A darlington pair is an arrangement of two bipolar transistors. **ULN2003** belongs to the family of ULN200X series of ICs. Different versions of this family interface to different logic families. ULN2003 is for 5V TTL, CMOS logic devices. These ICs are used when driving a wide range of loads and are used as relay drivers, display drivers, line drivers etc. ULN2003 is also commonly used while driving

[Stepper Motors](#). Refer [Stepper Motor interfacing using ULN2003](#). Each channel or darlington pair in **ULN2003** is rated at 500mA and can withstand peak current of 600mA. The inputs and outputs are provided opposite to each other in the pin layout. Each driver also contains a suppression diode to dissipate voltage spikes while driving inductive loads. The schematic for each driver is given below:



PIN DIAGRAM:



PIN DESCRIPTION:

Pin Description:

Pin No	Function	Name
1	Input for 1 st channel	Input 1
2	Input for 2 nd channel	Input 2
3	Input for 3 rd channel	Input 3
4	Input for 4 th channel	Input 4
5	Input for 5 th channel	Input 5
6	Input for 6 th channel	Input 6
7	Input for 7 th channel	Input 7
8	Ground (0V)	Ground
9	Common free wheeling diodes	Common
10	Output for 7 th channel	Output 7
11	Output for 6 th channel	Output 6
12	Output for 5 th channel	Output 5
13	Output for 4 th channel	Output 4
14	Output for 3 rd channel	Output 3
15	Output for 2 nd channel	Output 2
16	Output for 1 st channel	Output 1

DESCRIPTION:

ULN2003 belongs to the family of ULN200X series of ICs. Different versions of this family interface to different logic families. ULN2003 is for 5V TTL, CMOS logic devices. These ICs are used when driving a wide range of loads and are used as relay drivers, display drivers, line drivers etc. Each channel or darlington pair in **ULN2003** is rated at 500mA and can withstand peak current of 600mA. The inputs and outputs are provided opposite to each other in the pin layout. Each driver also contains a suppression diode to dissipate voltage spikes while driving inductive loads.

4.7.4 ULTRASONIC SENSOR MODULE -HC - SR04

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit.

WORKING PRINCIPLE

1. Using IO trigger for at least 8us high level signal
2. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
3. The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

FORMULA

Test distance = (high level time × velocity of sound (340m/s) / 2

Product features:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time×velocity of sound
(340M/S) / 2,

Connections as follows:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

ELECTRIC PARAMETERS:

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

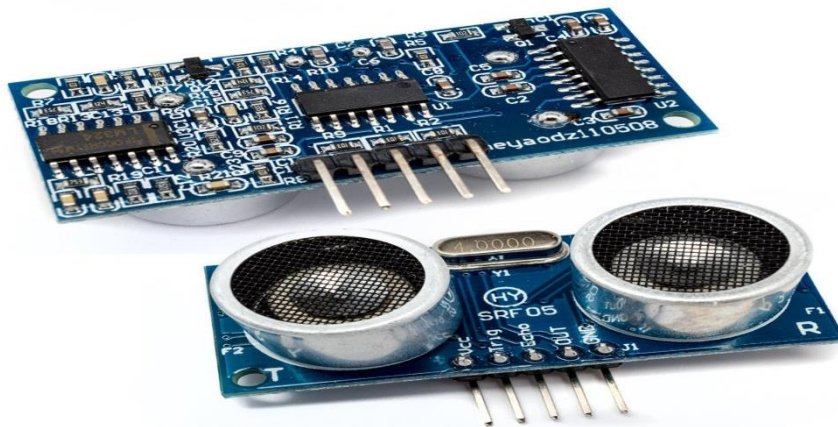
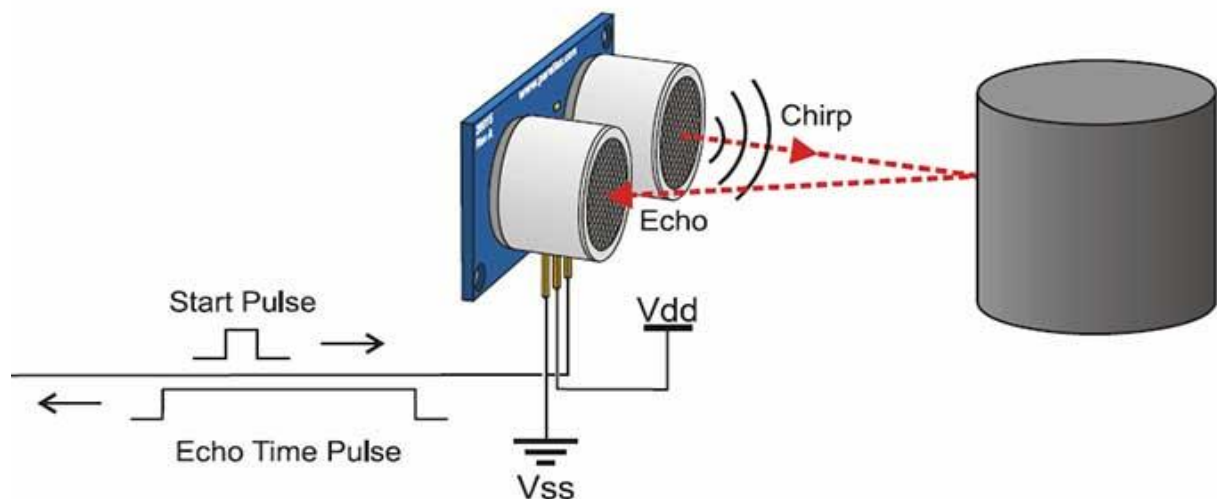


FIG 4.20

WORKING:

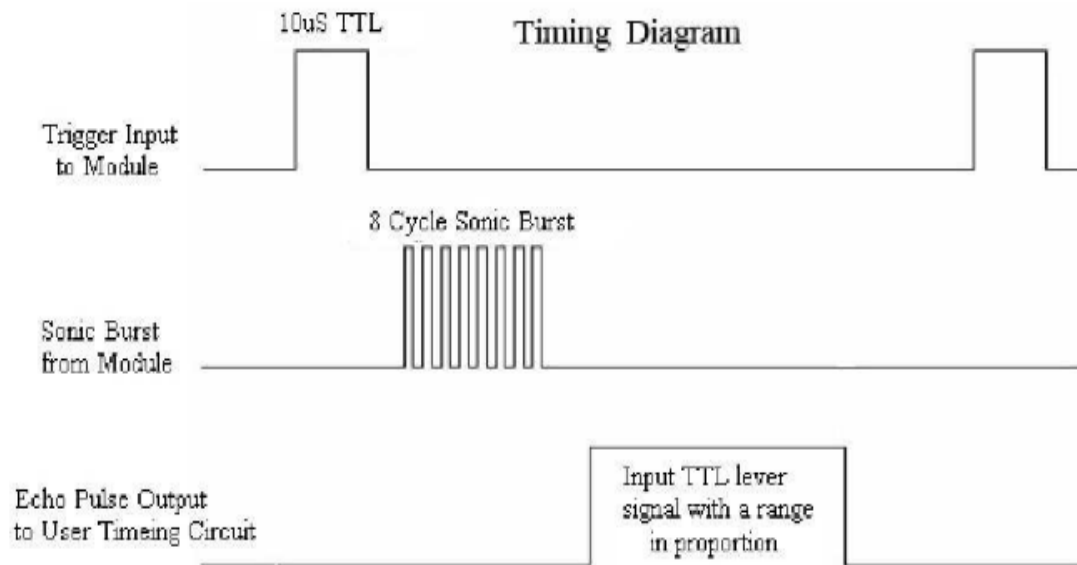
Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. Ultrasonic sensors emit short, high-frequency sound pulses at regular intervals. These propagate in the air at the velocity of sound. If they strike an object, then they are reflected back as echo signals to the sensor, which itself computes the distance to the target based on the time-span between emitting the signal and receiving the echo. As the distance to an object is determined by measuring the time of flight and not by the intensity of the sound, ultrasonic sensors are excellent at suppressing

background interference. Virtually all materials which reflect sound can be detected, regardless of their colour. Even transparent materials or thin foils represent no problem for an ultrasonic sensor. microsonic ultrasonic sensors are suitable for target distances from 20 mm to 10 m and as they measure the time of flight they can ascertain a measurement with pinpoint accuracy. Some of our sensors can even resolve the signal to an accuracy of 0.025 mm. Ultrasonic sensors can see through dust-laden air and ink mists. Even thin deposits on the sensor membrane do not impair its function. Sensors with a blind zone of only 20 mm and an extremely thin beam spread are making entirely new applications possible today: Fill level measurement in wells of microtiter plates and test tubes, as well as the detection of small bottles in the packaging industry, can be implemented with ease. Even thin wires are reliably detected.



TIMING DIAGRAM:

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: $\mu\text{S} / 58 = \text{centimeters}$ or $\mu\text{S} / 148 = \text{inch}$; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



4.8 SOFTWARE REQUIREMENTS:

4.8.1 ARDUINO IDE – 1.8.5

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students,

hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

4.8.2 Why Arduino?

The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50

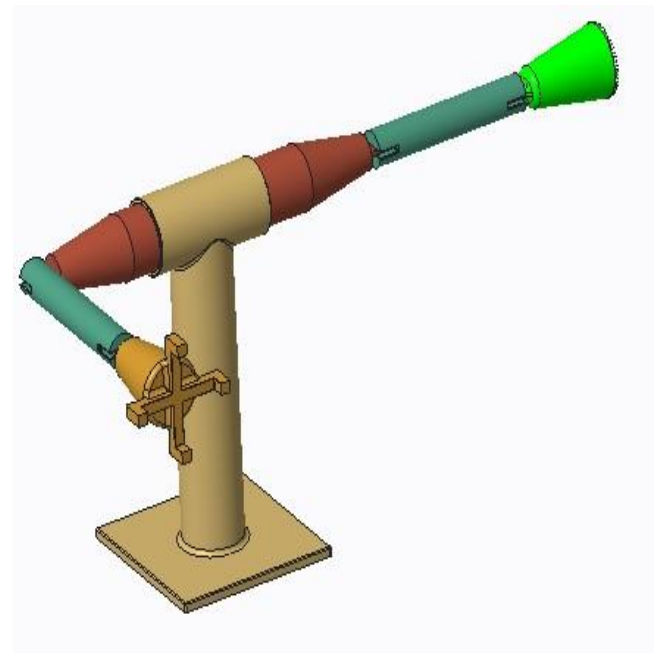
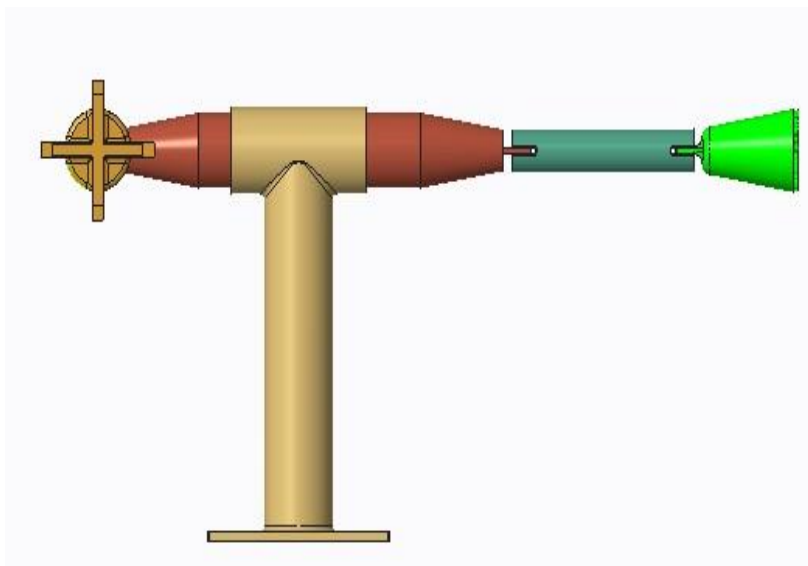
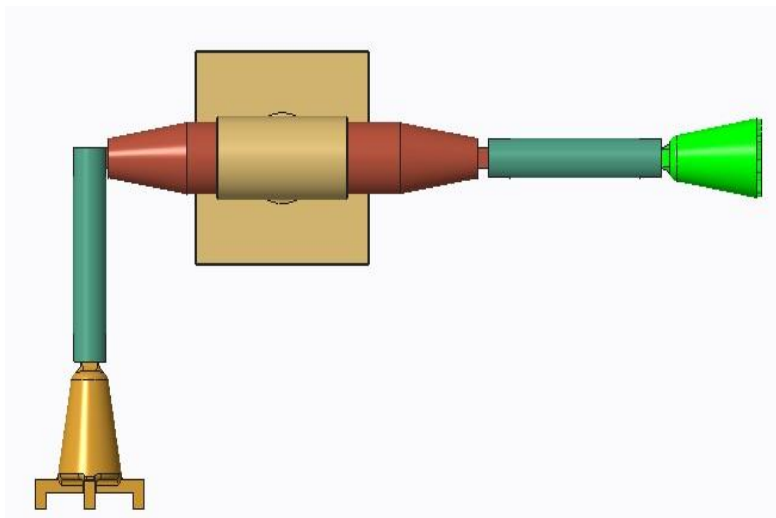
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

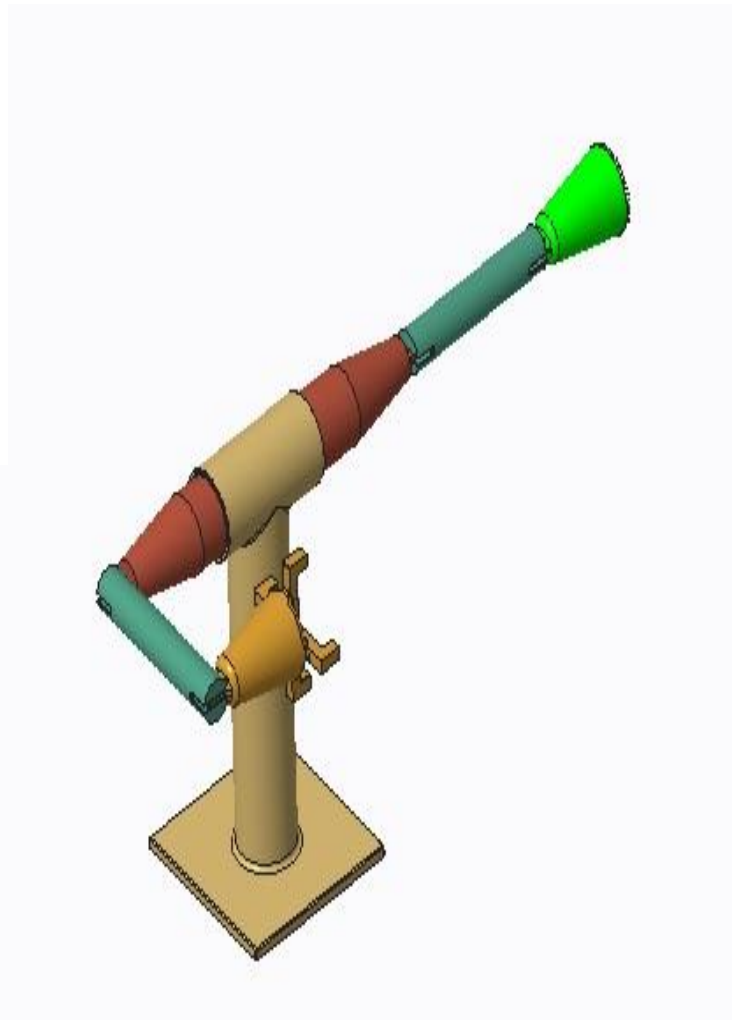
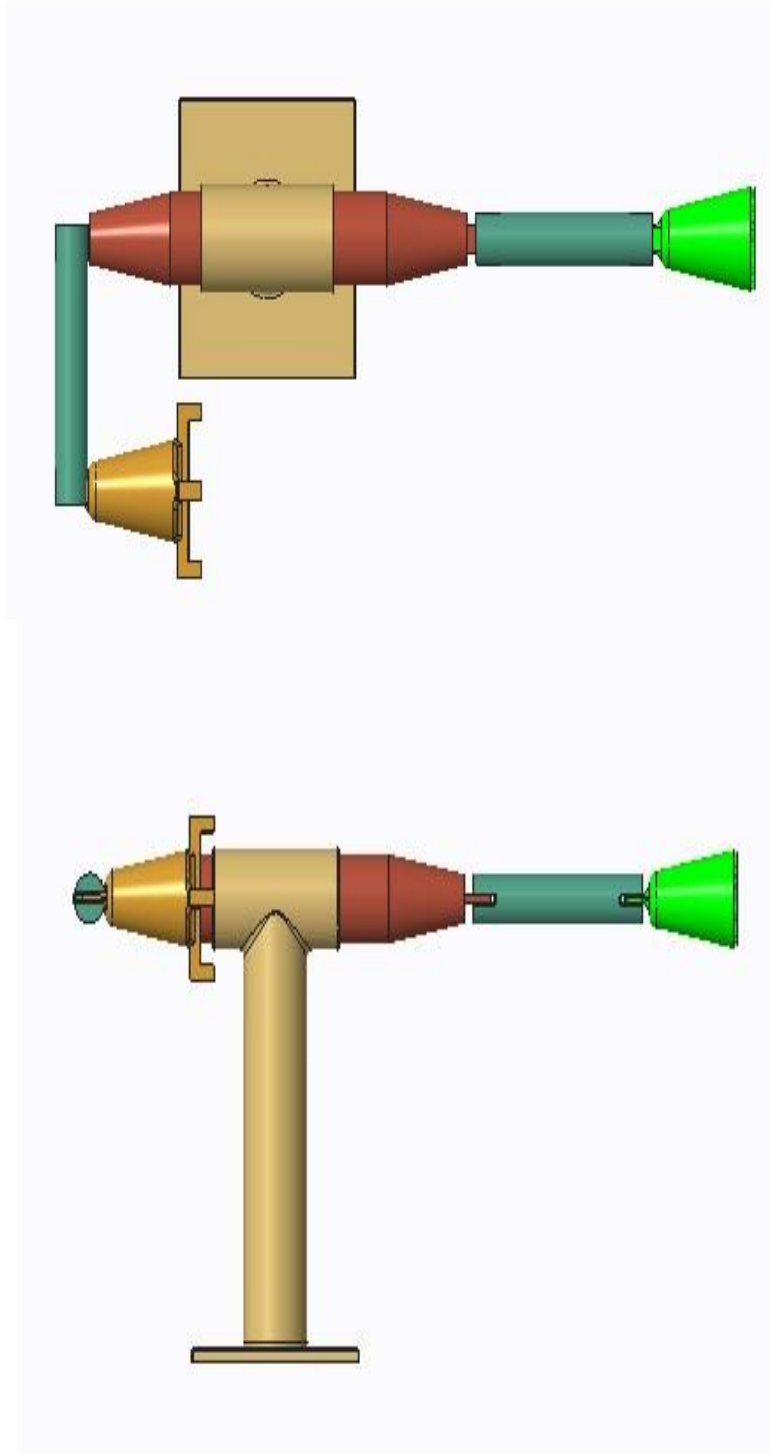
EMBEDDED C:

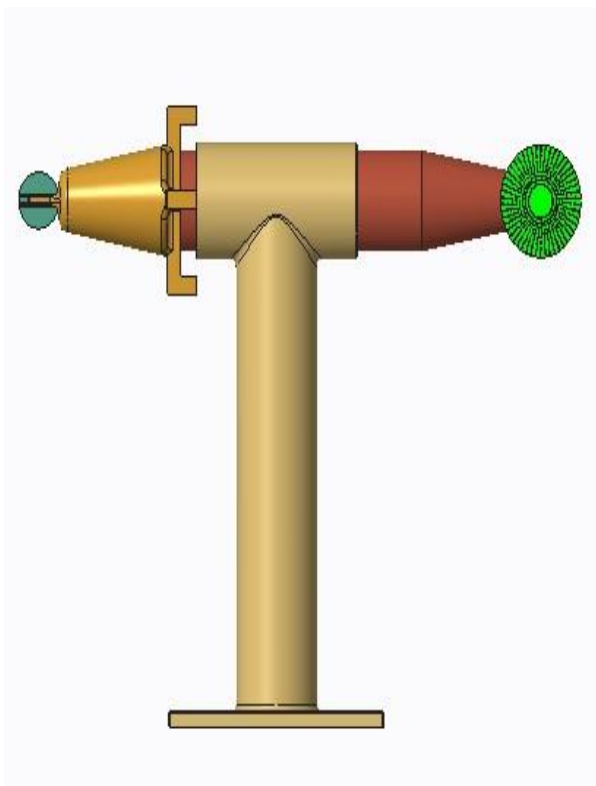
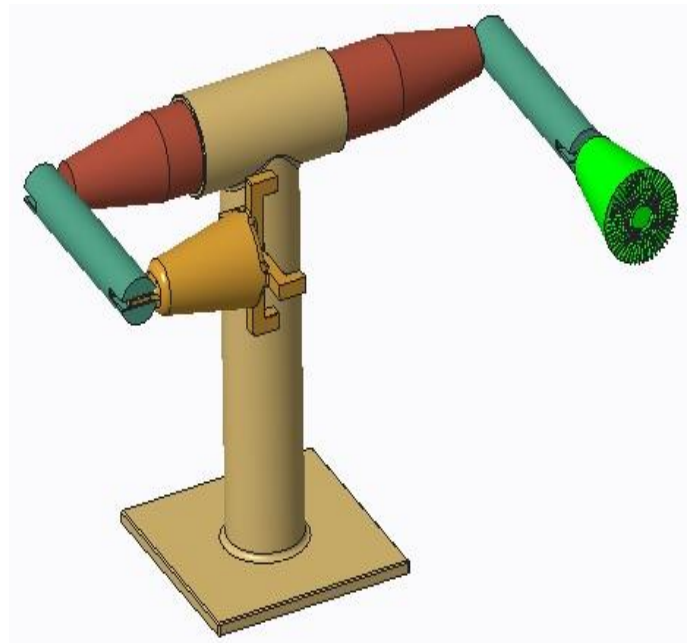
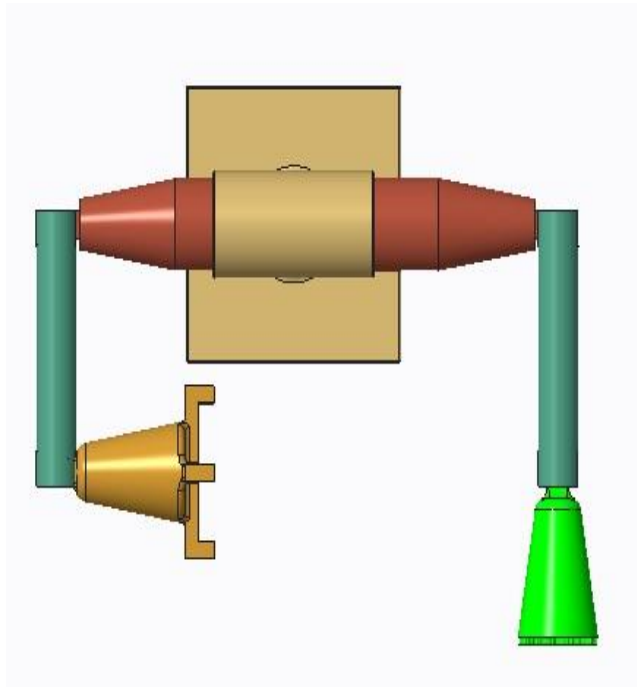
Embedded c is a set of language extension for the C Programming language by the C Standards committee to address commonality issues that exist between C extensions for different embedded systems. Historically embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations. Embedded C uses most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements(if, switch, case),loops(while, for),functions, arrays and strings, structures and union, bit operations, macros, etc. During infancy years of microprocessor based systems, programs were developed using assemblers and fused into the EPROMs. There used to be no mechanism to find what the program was doing. LEDs, switches, etc. were used to check for correct execution of the program .But they were too costly and were not quite reliable as well .As time progressed, use of

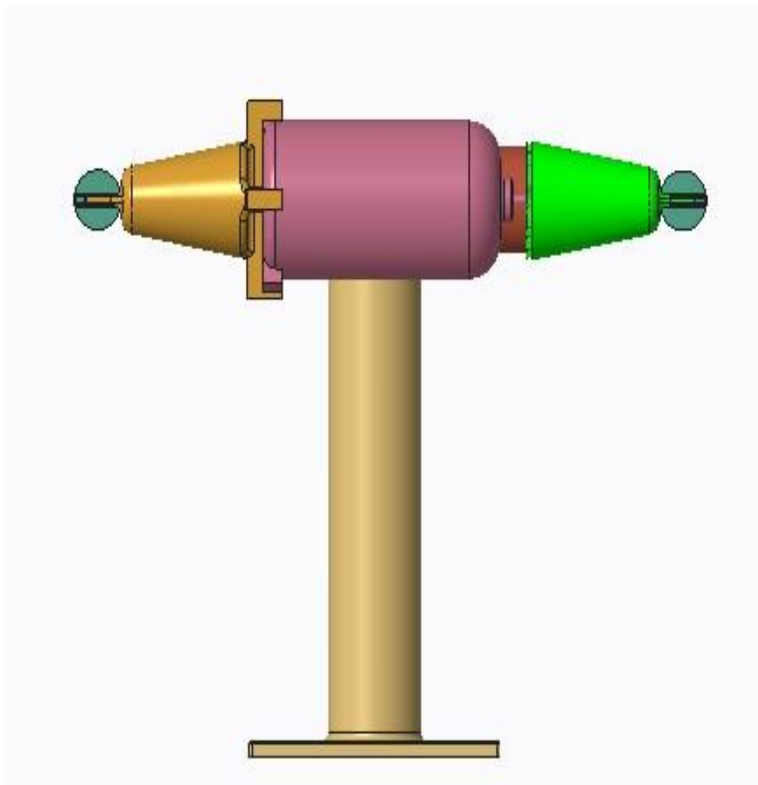
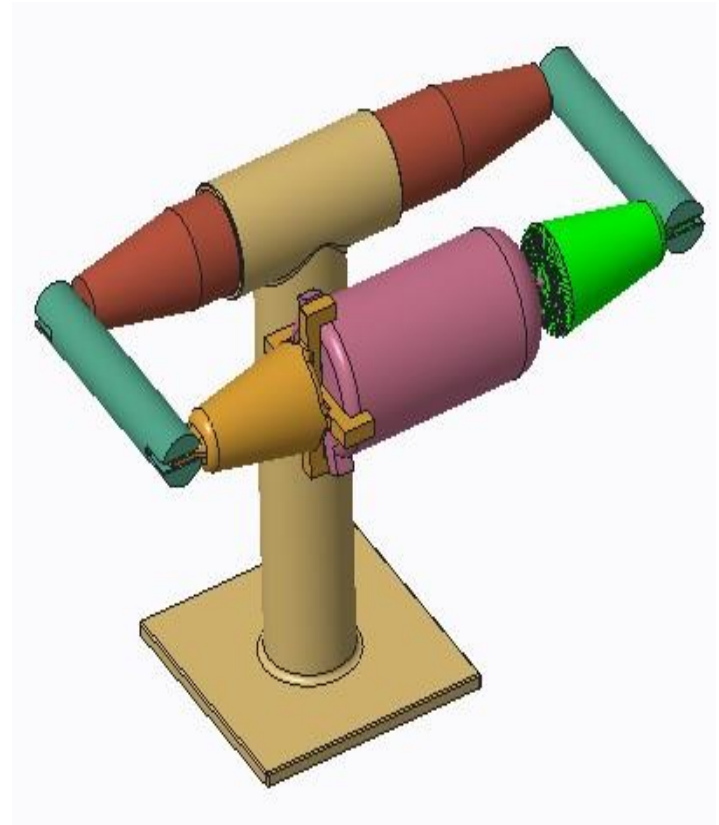
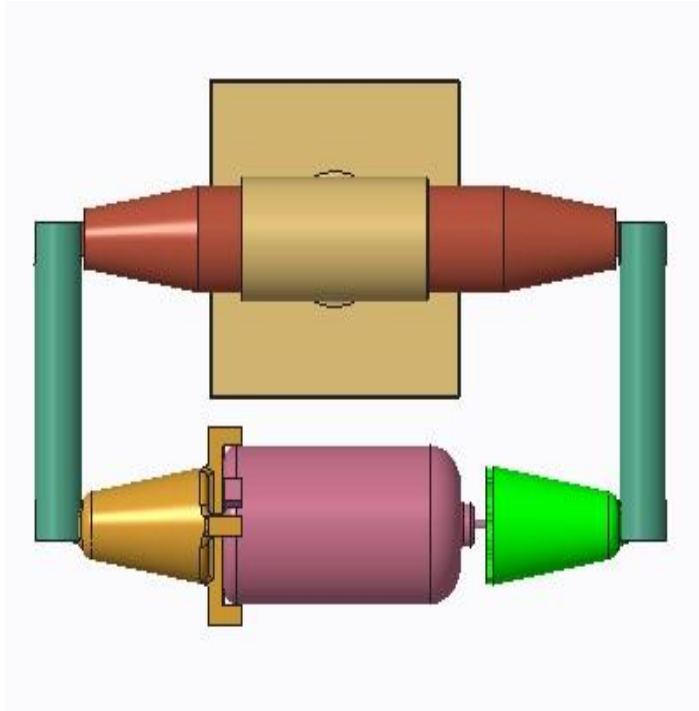
microprocessor-specific assembly-only as the programming language reduced and embedded systems moved onto C as the embedded programming language of choice. C is the most widely used programming language for embedded processors/controllers.

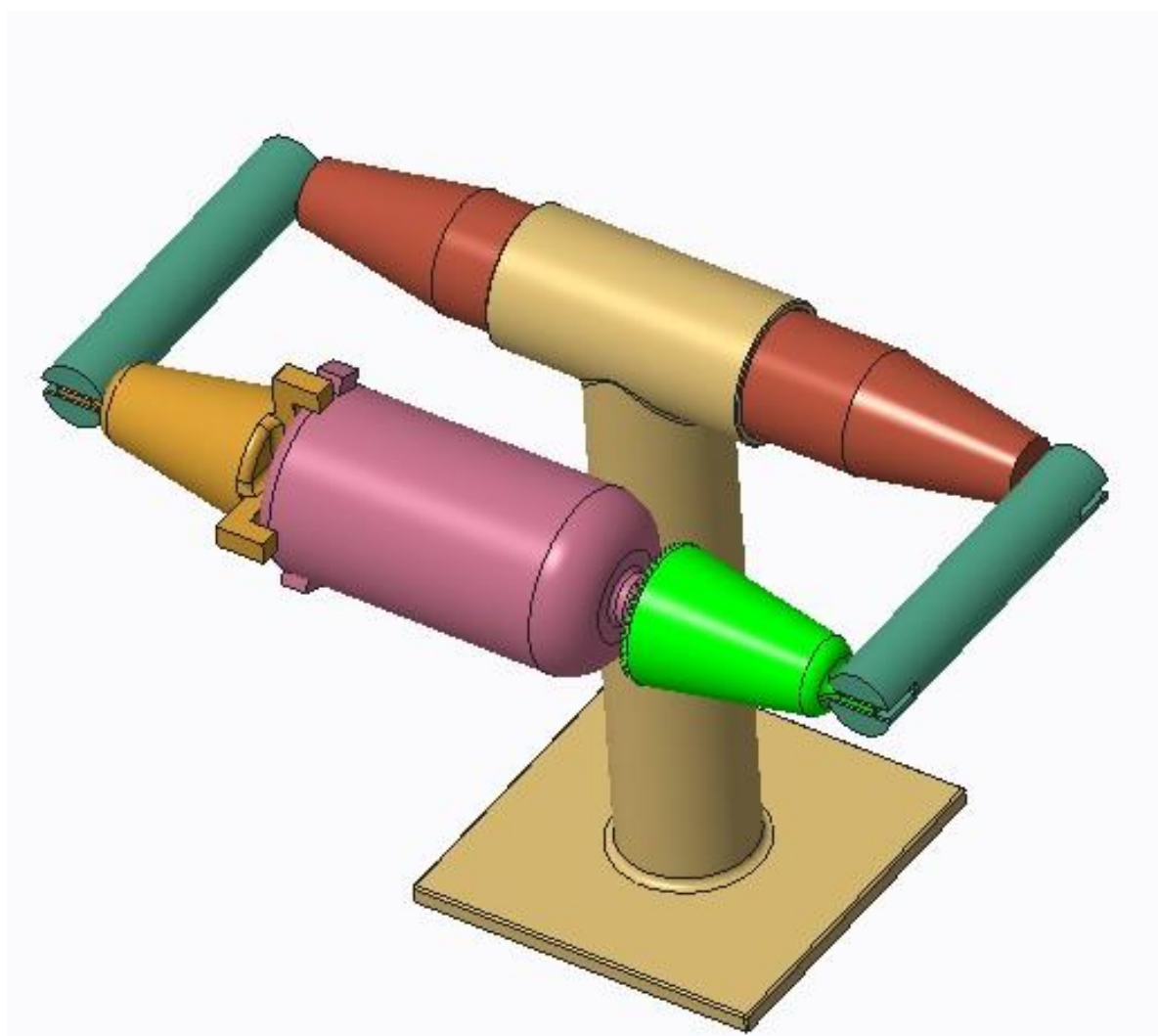
COMPUTERISED WORKING OF THE PROJECT [FIG 4.21]





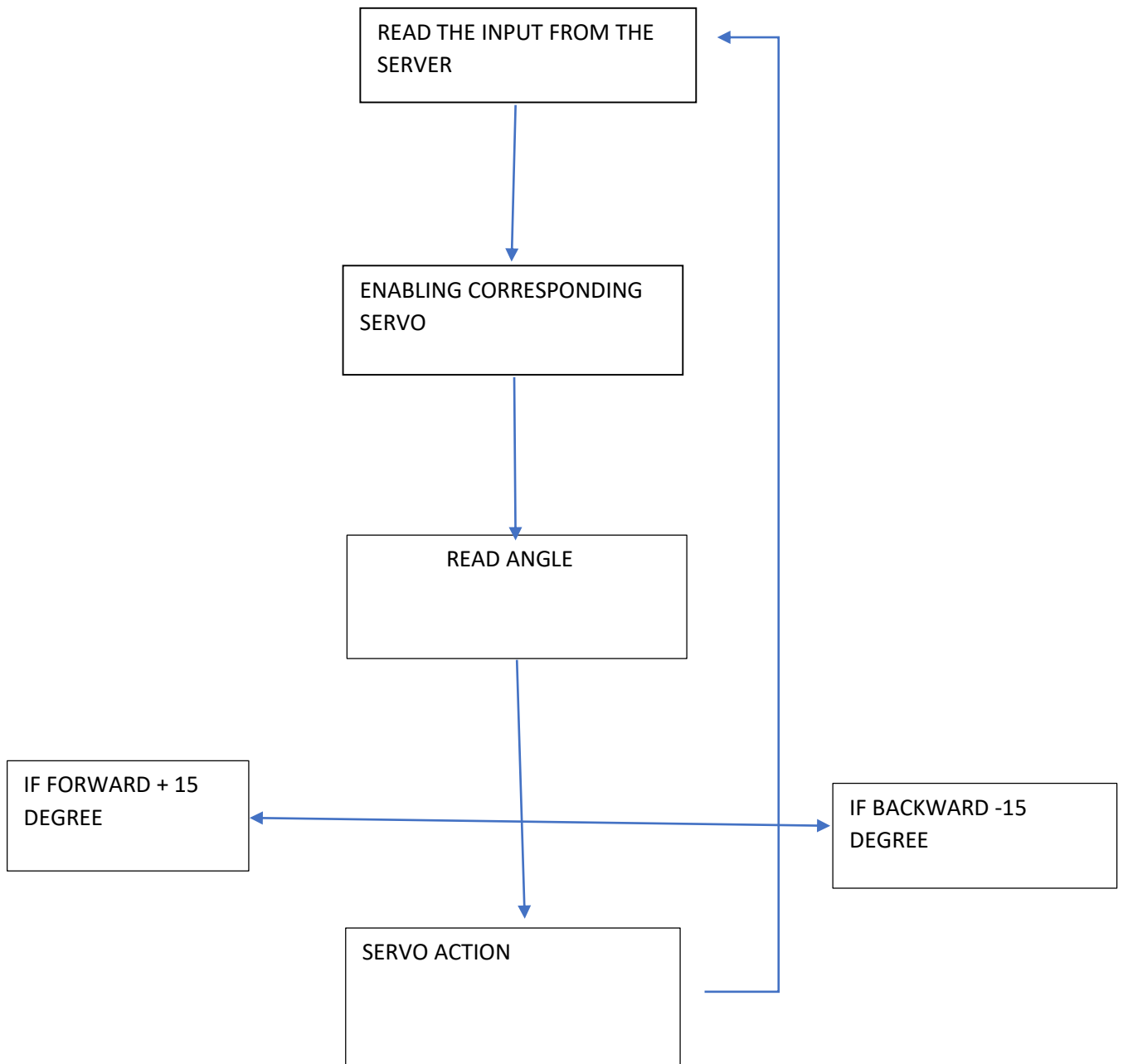






4.9 METHODOLOGY

4.9.1 FLOWCHART AND ALGORITHM USED



ALGORITHM USED

```
#include <Servo.h>
const int sw1 = A0;
```

```
const int sw2 = A1;
const int sw3 = A2;
const int sw4 = A3;
const int sw5 = A4;
const int sw6 = A5;
const int sw7 = 2;
const int sw8 = 3;
const int sw9 = 4;
const int motor = 13;
```

```
Servo myservo1; // create servo object to control a servo
Servo myservo2; // create servo object to control a servo
Servo myservo3; // create servo object to control a servo
Servo myservo4; // create servo object to control a servo
Servo myservo5; // create servo object to control a servo
Servo myservo6; // create servo object to control a servo
Servo myservo7; // create servo object to control a servo
```

```
int swstate = LOW;
char rx_data,rx_value;
unsigned int i=0,check;
int  angled1,backward_angle, angled, forward_angle,angle2;
short basemotorf=0,elmotorf= 0,backwardf=0,forwardf=0;
```

```
int pos = 0;
```

```
void setup()
{
```

```
    pinMode(sw1,INPUT_PULLUP);
    pinMode(sw2,INPUT_PULLUP);
    pinMode(sw3,INPUT_PULLUP);
    pinMode(sw4,INPUT_PULLUP);
    pinMode(sw5,INPUT_PULLUP);
    pinMode(sw6,INPUT_PULLUP);
    pinMode(sw7,INPUT_PULLUP);
    pinMode(sw8,INPUT_PULLUP);
    pinMode(sw9,INPUT_PULLUP);
```

```
    pinMode(motor,OUTPUT);
    digitalWrite(motor,LOW);
```

```
    Serial.begin(9600);
    Serial.println("Cleaning Robot");
    delay(1000);
    myservo1.attach(5); // attaches the servo on pin 9 to the servo object
    myservo1.write(135);
    delay(100);
    myservo1.detach();
```

```

delay(40);
  myservo2.attach(6); // attaches the servo on pin 9 to the servo object
  myservo2.write(180);
delay(100);
  myservo2.detach();
  delay(40);
  myservo3.attach(7); // attaches the servo on pin 9 to the servo object
  myservo3.write(180); // SERVO CONNECTED INVERSE
delay(100);
  myservo3.detach();
  delay(40);
  myservo4.attach(8); // attaches the servo on pin 9 to the servo object
  myservo4.write(45);
delay(100);
  myservo4.detach();
  delay(40);
  myservo5.attach(9); // attaches the servo on pin 9 to the servo object
  myservo5.write(105);
delay(100);
  myservo5.detach();
  delay(40);
  myservo5.attach(10); // attaches the servo on pin 9 to the servo object
  myservo5.write(90);
delay(100);
  myservo5.detach();
  delay(40);

  myservo5.attach(11); // attaches the servo on pin 9 to the servo object
  myservo5.write(15);
delay(100);
  myservo5.detach();
  delay(40);
  myservo5.attach(12); // attaches the servo on pin 9 to the servo object
  myservo5.write(135);
delay(100);
  myservo5.detach();
  delay(40);
  delay(2000);

}

```

```

void loop()
{

  if(digitalRead(sw1)==LOW)
  {
    Serial.println("Pressed");
    myservo1.attach(5);
    angle2 = myservo1.read();

```

```

if(angle2<150&&backwardf==0) // going inside forward
{
  if(angle2<=135)
  {
    Serial.println("Forward");
    forward_angle=15;
    forwardf=1;
    up_forward();
  }
  else if(angle2>135)
  {

    backwardf=1;
  }
  if( backwardf==1)
  {
    backward_angle=15;
    Serial.println("backward");
    up_backward();
  }
  if(angle2==0)
  {
    backwardf=0;
  }
}

if(digitalRead(sw2)==LOW)
{
  Serial.println("sw2 Pressed");
  myservo1.attach(6);
  angle2 = myservo1.read();
  Serial.println("Pangle");
  Serial.println(angle2);

  if(angle2<=180&&angle2>=90&&backwardf==0) // going inside forward
  {
    if(angle2>=90)
    {
      Serial.println("Forward");
      forward_angle=15;
      forwardf=1;
      up_forward();
    }
  }
  if(angle2==180)
  {

```

```

    backwardf=1;
}
if( backwardf==1&&angle2>=90)
{
    backward_angle=15;
    Serial.println("backward");
    up_backward();
}
if(angle2==90)
{
    backwardf=0;
}
}

if(digitalRead(sw3)==LOW)
{
    Serial.println("sw3 Pressed");
    myservo1.attach(7);
    angle2 = myservo1.read();

    if(angle2<=180&&angle2>=0&&backwardf==0) // going inside forward
    {

        Serial.println("Forward");
        forward_angle=15;

        up_forward();

    }
    if(angle2==180)
    {

        backwardf=1;
        forwardf=0;

    }
    if( backwardf==1&&angle2>=0)
    {
        backward_angle=15;
        Serial.println("backward");
        up_backward();
    }
    if(angle2==0)
    {
        backwardf=0;
        forwardf=1;
    }
}
//.....righ side

```

```

if(digitalRead(sw4)==LOW)
{
  Serial.println("SW4 Pressed RA");
  myservo1.attach(8);
  angle2 = myservo1.read();

  if(angle2<=180&&backwardf==0) // going inside forward
  {
    if(angle2<=180)
    {
      Serial.println("Forward");
      forward_angle=15;
      forwardf=1;
      up_forward();
    }
  }
  if(angle2==90||angle2==180)
  {

    backwardf=1;
  }
  if( backwardf==1)
  {
    backward_angle=15;
    Serial.println("backward");
    up_backward();
  }
  if(angle2==45)
  {
    backwardf=0;
  }

}

if(digitalRead(sw5)==LOW)
{
  Serial.println("sw5 Pressed RSA");
  myservo1.attach(9);
  angle2 = myservo1.read();
  Serial.println("Pangle");
  Serial.println(angle2);

  if(angle2<=150&&angle2>=105&&backwardf==0) // going inside forward
  {
    if(angle2>=105)
    {
      Serial.println("Forward");
      forward_angle=15;
      forwardf=1;
      up_forward();
    }
  }
}

```

```

    }
    }
    if(angle2==150)
    {

        backwardf=1;
    }
    if( backwardf==1&&angle2>=105)
    {
        backward_angle=15;
        Serial.println("backward");
        up_backward();
    }
    if(angle2==105)
    {
        backwardf=0;
    }

}

if(digitalRead(sw6)==LOW)
{
    Serial.println("sw6 Pressed RS");
    myservo1.attach(10);
    angle2 = myservo1.read();

    if(angle2<=180&&angle2>=0&&backwardf==0) // going inside forward
    {

        Serial.println("Forward");
        forward_angle=15;

        up_forward();

    }
    if(angle2==180)
    {

        backwardf=1;
        forwardf=0;

    }
    if( backwardf==1&&angle2>=0)
    {
        backward_angle=15;
        Serial.println("backward");
        up_backward();
    }
    if(angle2==75)
    {

```

```

backwardf=0;
forwardf=1;
}

}

//.....pick and place .....

if(digitalRead(sw7)==LOW)
{
  Serial.println("sw7 Pressed Rotate");
  myservo1.attach(11);
  angle2 = myservo1.read();

  if(angle2<=180&&angle2>=0&&backwardf==0) // going inside forward
  {

    Serial.println("Forward");
    forward_angle=15;

    up_forward();

  }
  if( backwardf==1&&angle2>=0)
  {
    backward_angle=15;
    Serial.println("backward");
    up_backward();
  }
  if(angle2==180)
  {
    backwardf=1;
    forwardf=0;
  }
  if(angle2==0)
  {
    backwardf=0;
    forwardf=0;
  }
}

if(digitalRead(sw8)==LOW)
{
  Serial.println("sw8 Pressed Pick");
  myservo1.attach(12);
  angle2 = myservo1.read();

  if(angle2<=180&&angle2>=0&&backwardf==0) // going inside forward
  {

```

```

Serial.println("Forward");
forward_angle=15;

up_forward1();

}
if( backwardf==1&&angle2>=0)
{
    backward_angle=15;
    Serial.println("backward");
    up_backward1();
}
if(angle2>=165)//||angle2==180)
{
    backwardf=1;
    forwardf=0;
}
if(angle2==105)
{
    backwardf=0;
    forwardf=0;
}
}

if(digitalRead(sw9)==LOW)
{
    Serial.println("sw9 Pressed for clean");
    Serial.println("DC Motor ON");
    digitalWrite(motor,HIGH);
    delay(1000);

}
if(digitalRead(sw9)==HIGH)
{
    // Serial.println("sw9 Pressed for clean");
    // Serial.println("DC Motor OFF");
    digitalWrite(motor,LOW);
    delay(50);
}
}

```

```

void up_forward() // this motor for elbow
{
  angle2= angle2+forward_angle;
  check = myservo1.read();
  Serial.println(angle2);
  Serial.println(check);
  if(check!=angle2)
  {
    for(i=0;i<10;i++)
    {
      // myservo1.attach(5); // attaches the servo on pin 9 to the servo object
      myservo1.write(angle2);
      delay(100);
      // myservo1.detach();
      delay(15);
    } // waits 15ms for the servo to reach the positio
  }
  myservo1.detach();
}

```

```

void up_backward() // this motor for elbow
{

  angle2= angle2 - backward_angle;
  check = myservo1.read();
  Serial.println(angle2);
  Serial.println(check);

  if(check!=angle2)
  {
    for(i=0;i<10;i++)
    {
      // myservo1.attach(5); // attaches the servo on pin 9 to the servo object
      myservo1.write(angle2);
      delay(100);
      // myservo1.detach();
      delay(15);
    } // waits 15ms for the servo to reach the positio
  } // waits 15ms for the servo to reach the position
  myservo1.detach();
}

```

```

void up_forward1() // this motor for elbow
{
  angle2= angle2+forward_angle;
  check = myservo1.read();
  Serial.println(angle2);
  Serial.println(check);
  if(check!=angle2)

```

```

    {
        for(i=0;i<10;i++)
        {
            // myservo1.attach(5); // attaches the servo on pin 9 to the servo object
            myservo1.write(angle2);
            delay(100);
            // myservo1.detach();
            delay(15);
        } // waits 15ms for the servo to reach the positio
    }
    // myservo1.detach();
}

```

```

void up_backward1() // this motor for elbow
{

```

```

    angle2= angle2 - backward_angle;
    check = myservo1.read();
    Serial.println(angle2);
    Serial.println(check);

```

```

        if(check!=angle2)
        {
            for(i=0;i<10;i++)
            {
                // myservo1.attach(5); // attaches the servo on pin 9 to the servo object
                myservo1.write(angle2);
                delay(100);
            // myservo1.detach();
            delay(15);
        } // waits 15ms for the servo to reach the positio
        } // waits 15ms for the servo to reach the position
    //myservo1.detach();
}

```

```

/* switch(rx_value)
{

```

```

    rx_data='\0';
    break;
    case 0x02:
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("F1 down ");
        delay(1000);
        check = myservo1.read();
        if(check!=0)
        {
            for(i=0;i<10;i++)

```

```

{
  myservo1.attach(2); // attaches the servo on pin 9 to the servo object
  myservo1.write(0);
  delay(100);
  myservo1.detach();
  delay(15);
} // waits 15ms for the servo to reach the positio
}
  rx_data='\0';
  break;

case 0x03:
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("F2 up ");
  delay(1000);
  check = myservo2.read();
  if(check!=180)
  {
    for(i=0;i<10;i++)
    {
      myservo2.attach(3); // attaches the servo on pin 9 to the servo object
      myservo2.write(180);
      delay(100);
      myservo2.detach();
      delay(15);
    } // waits 15ms for the servo to reach the positio
  }
  rx_data='\0';
  break;
case 0x04:
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("F2 down ");
  delay(1000);
  check = myservo2.read();
  if(check!=0)
  {
    for(i=0;i<10;i++)
    {
      myservo2.attach(3); // attaches the servo on pin 9 to the servo object
      myservo2.write(0);
      delay(100);
      myservo2.detach();
      delay(15);
    } // waits 15ms for the servo to reach the positio
  }
  rx_data='\0';
  break;

case 0x05:

```

```

    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("F3 up ");
    delay(1000);
    check = myservo3.read();
    if(check!=0)
    {
        for(i=0;i<10;i++)
        {
            myservo3.attach(4); // attaches the servo on pin 9 to the servo object
            myservo3.write(0);
            delay(100);
            myservo3.detach();
        }
        delay(15);
        // waits 15ms for the servo to reach the positio
    }
    rx_data='\0';
    break;
case 0x06:
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("F3 down ");
    delay(1000);
    check = myservo3.read();
    if(check!=180)
    {
        for(i=0;i<10;i++)
        {
            myservo3.attach(4); // attaches the servo on pin 9 to the servo object
            myservo3.write(180);
            delay(100);
            myservo3.detach();
        }
        delay(15);
        // waits 15ms for the servo to reach the positio
    }
    rx_data='\0';
    break;
case 0x07:
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("F4 up ");
    delay(1000);
    check = myservo4.read();
    if(check!=0)
    {
        // THIS MOTOR IS NOT CONNECTED 180 TO 0
        for(i=0;i<10;i++)
        {
            myservo4.attach(5); // attaches the servo on pin 9 to the servo object
            myservo4.write(0); // NOTE CHANGED INTO 150
            delay(100);
            myservo4.detach();
        }
    }

```

```

delay(15);
} // waits 15ms for the servo to reach the positio
}
break;
case 0x08:
check = myservo4.read();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("F4 down ");
delay(1000);
if(check!=180)
{
    for(i=0;i<10;i++)
    {
        myservo4.attach(5); // attaches the servo on pin 9 to the servo object
        myservo4.write(180);
        delay(100);
        myservo4.detach();
    }
    delay(15);
} // waits 15ms for the servo to reach the positio
}
rx_data='\0';
break;
case 0x09:
lcd.clear();
lcd.setCursor(0,0);
lcd.print("F5 up ");
check = myservo5.read();
if(check!=180)
{
    for(i=0;i<10;i++)
    {
        myservo5.attach(6); // attaches the servo on pin 9 to the servo object
        myservo5.write(180);
        delay(100);
        myservo5.detach();
    }
    delay(15);
} // waits 15ms for the servo to reach the positio
}
rx_data='\0';
break;
case 0x0A:
lcd.clear();
lcd.setCursor(0,0);
lcd.print("F5 down ");
check = myservo5.read();
if(check!=0)
{
    for(i=0;i<10;i++)
    {
        myservo5.attach(6); // attaches the servo on pin 9 to the servo object

```

```

        myservo5.write(0);
        delay(100);
        myservo5.detach();
        delay(15);
    } // waits 15ms for the servo to reach the positio
    }
    rx_data='\0';
    break;

}*/
//      if(emgval>50)
//      {
//          rx_value=0x09;
//          swstate = HIGH;
//          delay(10);
//      }
//
//      if(emgval<50)
//      {
//          rx_value=0x0A;
//          swstate = HIGH;
//          delay(10);
//      }
//
//}
--

```

4.9.2 WORKING OF THE PROJECT

The prototype gets on once its connected to a power supply .The next step would be picking up of the object which has to be cleaned . It picks up the object using a gripper hand which holds the object so tight like human hand used to hold. The gripper hand is fixed to one of the robotic arm. Once the gripper hands picks up the object and the robotic arm comes to a position where the other robotic which is going to clean has a reach to it. Once the object reached to the desired position the other robotic arm which is fixed with a brush in its tip of the arm starts to detect the object. Once it detected the brush starts to rotate and clean the object. Once the cleaning of one side is done ,the gripper hand rotates the other side to make the other sides also clean. Once the whole designated process is done the object is dropped to its initial position.

This whole process takes place with a help of microcontroller , servo motors ,and a remote control to control different operations.The commands are given by the user through remote control to pick up ,clean, drop etc . these commands are sent as signals to the microcontroller here arduino uno is used as microcontroller .These signals are verified by the microcontroller to conduct operation by controlling different servo moters situated in different parts of the prototype.

CHAPTER 5

5 RESULT AND DISCUSSION

5.1 RESULT

Design and fabrication of kitchen vessel cleaning robot is lot more advanced than a dishwasher where the function dishwasher is restricted to a extent . This prototype is controlled by a remote which controls the different operations . Once the start the operation using remote it starts functioning automatically conduction the required operations. The whole process is conducted by different motors suited in different parts of the robot or prototype. Arduino uno microcontroller board is used to conduct the operation where algorithm codes are feeded to it earlier.

5.2 DISCUSSION

The input is given by a remote controller which contains different switches for the commands of the prototype. The commands are given by user using the remote controller for the cleaning process to be conducted. The commands are received by the microcontroller which is the Arduino uno microcontroller which sense the operation to be conducted and conveys the signal to different servo moters and conduct the desired operation.

The kitchen vessel cleaning robot direction and movement possible are given below:

1. Pick up : the motors in the hand of robot helps to pick up the material which have to be cleaned using a gripper hand which is situated in the tip of the robotic arm.
2. Hold position : to hold the material to a desired position
3. Cleaner arm : one arm which holds the material the other arm which is a cleaner arm which is fixed with a brush starts cleaning the material by rotating the brush.
4. Rotatory motion : the arm which holds the material with a gripper arm starts rotating to clean all the sides of the material
5. Drop : once the cleaning process is done the gripper arm places the cleaned material to its initial position.

Chapter 6

SUMMARY AND CONCLUSION

6.1 SUMMARY

This project can be summarised as a system that is used as kitchen assistant in the vessel cleaning process.

This prototype does the job of picking up ,cleaning ,and dropping the material after making it clean to a certain extent.

6.2 Conclusion

The use of humanoid robots as a home appliance especially in kitchen helps a lot in home, restaurants to reduce the pressure of job and also to save time as the process of cleaning vessel is a lot of effort and time consuming operation.

We got to know that lot people are in needfull of a machine which does the job of cleaning vessels as it is considered to be the most stressfull job to be done in home,restaurants etc. We conducted asking people from different sectors to get a knowledge that how a machine should be prepared and how it can usefull to them.

7. REFERENCE

- [1] M. A. K. Yusoff, R. E. Samin, and B. S. K. Ibrahim, "Wireless mobile robotic arm," *Procedia Engineering*, vol. 41, pp. 1072-1078, 2012.
- [2] K. Kadirgama, M. Noor, and M. Rahman, "Optimization of surface roughness in end milling using potential support vector machine," *Arabian Journal for Science and Engineering*, vol. 37, pp. 2269-2275, 2012.
- [3] A. Elfasakhany, E. Yanez, K. Baylon, and R. Salgado, "Design and development of a competitive low-cost robot arm with four degrees of freedom," *Modern Mechanical Engineering*, vol. 1, p. 47, 2011.
- [4] C. F. Olson, "Probabilistic self-localization for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 16, pp. 55-66, 2000.
- [5] I. P. MODEL, "Industry Based Automatic Robotic Arm."
- [6] C. Bartneck and J. Forlizzi, "A design-centred framework for social human-robot interaction," in *Proceedings of the 13th IEEE International Workshop on Robot and Human Interactive Communication*, 2004, pp. 31-33.

- [7] G. Katal, S. Gupta, and S. Kakkar, "DESIGN AND OPERATION OF SYNCHRONIZED ROBOTIC ARM."
- [8] J. L. Jones, "Robots at the tipping point: the road to iRobot Roomba," *Robotics & Automation Magazine*, IEEE, vol. 13, pp. 76-78, 2006.
- [9] A. C. McDonald, "Robot technology: theory, design and applications," Sir Sanford Fleming Coll., Ontario 1986.
- [10] S. Yu, X. Yu, B. Shirinzadeh, and Z. Man, "Continuous finite-time control for robotic manipulators with terminal sliding mode," *Automatica*, vol. 41, pp. 1957-1964, 2005.
- [11] M. Hägele, K. Nilsson, and J. N. Pires, "Industrial robotics," in *Springer handbook of robotics*, ed: Springer, 2008, pp. 963-986. 2nd Integrated Design Project Conference (IDPC) 2015, Faculty of Mechanical Engineering, Universiti Malaysia Pahang, 11 Dec 2015.
- [12] R.-J. Wang, J.-W. Zhang, J.-M. Xu, and H.-Y. Liu, "The multiple-function intelligent robotic arms," in *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on*, 2009, pp. 1995-2000.
- [13] L. B. Duc, M. Syaifuddin, T. T. Toai, N. H. Tan, M. N. Saad, and L. C. Wai, "Designing 8 degrees of freedom humanoid robotic arm," in *Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on*, 2007, pp. 1069-1074.
- [14] C. Carignan, G. Gefke, and B. Roberts, "Intro to Space Mission Design: Space Robotics," in *Seminar of Space Robotics*, University of Maryland, Baltimore, 2002.
- [15] R. Bonitz, J. Slostad, B. Bon, D. Braun, R. Brill, C. Buck, et al., "Mars volatiles and climate surveyor robotic arm," *Journal of Geophysical Research: Planets* (1991–2012), vol. 106, pp. 17623-17634, 2001.
- [16] R. Bonitz, L. Shiraishi, M. Robinson, J. Carsten, R. Volpe, A. Trebi-Ollennu, et al., "The Phoenix Mars lander robotic arm," in *Aerospace conference, 2009 IEEE*, 2009, pp. 1-12.
- [17] R. Boyce and J. Mull, "Complying with the Occupational Safety and Health Administration: Guidelines for the dental office," *Dental Clinics of North America*, vol. 52, pp. 653-668, 2008.

- [18] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, Robotics: modelling, planning and control: Springer Science & Business Media, 2009.
- [19] M. A. Rahman, A. H. Khan, T. Ahmed, and M. M. Sajjad, "Design, Analysis and Implementation of a Robotic Arm-The Animator."
- [20] P. Singh, A. Kumar, and M. Vashisth, "Design of a Robotic Arm with Gripper & End Effector for Spot Welding," Universal Journal of Mechanical Engineering, vol. 1, pp. 92-97, 2013.
- [21] M. W. Spong, Vidyasagar, Robot Dynamics and Control, 1989.
- [22] A. J. Critchlow, Introduction to robotics: Macmillan New York, 1985.
- [23] G. Mikell and N. Roger, "Robótica Industrial. Tecnología, Programación y Aplicaciones. Ed," ed: Mc Graw Hill.
- [24] C. A. Schuler and W. L. McNamee, Industrial electronics and robotics: McGraw-Hill, Inc., 1986.

IMAGES OF THE PROTOTYPE



