DEPARTMENT OF ELECTRICAL & ELECTRONIC ENGINEERING
PREMIER UNIVERSITY, CHITTAGONG

Project Report On
“Smart Multi-storage Car Parking System”

26th February, 2018

Department of Electrical and Electronic Engineering
Premier University, Chittagong
Chittagong-4203, Bangladesh

Premier University

Department of Electrical and Electronic Engineering

Name of Project

Smart Multi-storage Car Parking System

Project Assigned to:

Partha Sarathi Ghosh ID: 12012101200419	Sourav Dhar ID: 12012101200423
……………………… Signature of student	……………………… Signature of student

Supervised by:

Mr. Md. Saifuddin Munna
Lecturer, Dept. of EEE

……………………………..
Signature of the supervisor

Department of Electrical and Electronic Engineering
Premier University, Chittagong
Chittagong-4203, Bangladesh

The thesis titled **“Smart Multi-storage Car Parking System”** submitted by Roll No: **12012101200419** & Roll No: **12012101200423** sessions January-2018 for the partial fulfilment of the requirement for the degree of Bachelor of Science in Electrical and Electronic Engineering.

Dedicated to
Our Beloved Parents,
Our Siblings
&
Our honorable Teachers

Declaration

This is to endorse that the work presented through this project entitled "**Smart Multi-storage Car Parking System**" is the outcome of the investigation carried out by us under the supervision of Mr. Md. Saifuddin Munna, Lecturer, Department of Electrical and Electronic Engineering (EEE), Premier University Chittagong. It is also proclaimed that neither of the particular work nor any part of the work presented here has been submitted elsewhere for the award of any degree or diploma.

Supervised by:
Mr. Md. Saifuddin Munna
Lecturer, Dept. of EEE
Premier University

(Author):

Partha Sarathi Ghosh
ID: 12012101200419

………………………
Signature of student

Sourav Dhar
ID: 12012101200423

………………………
Signature of student

Acknowledgement

At the very beginning, we proclaim that we are very much grateful to the almighty and the most beneficent, for enabling us to complete the project work.

My heartfelt thanks and deepest gratitude to our respected thesis supervisor Mr. Md. Saifuddin Munna, Lecturer, Department of Electrical and Electronics Engineering (EEE), for having faith in us and for engaging us in this research work. I am also thankful for his valuable guidance through this project work. He has always stretched his helping hands whenever we needed. This thesis, what it is today, is the result of his constant guidance, helpful suggestion, and constructive criticisms and also his endless patience. He is the light bearer of this whole research work.

Furthermore, I also thank to our honourable Chairman of our department and also to my faculty members for their constant supports. And last but not the least; I thank my classmates for their endless help and support.

Abstract

Automatic multi-stored car parking system is very good substitute for car parking area. Since in modern world, where space has become a very big problem and in the era of miniaturization it’s become a very crucial necessity to avoid the wastage of space in modern big companies and apartments etc. For example, in a space where more than 100 cars need to be parked, it’s a very difficult task to do and also to reduce the wastage of area, this system can be used. This Automatic Car Parking enables the parking of vehicles-floor after floor and thus reducing the space used. Here any number of cars can be park according to requirement. These make the system modernized and even a space-saving one. This idea is developed using Microcontroller. Here program is written according to this idea using microcontroller. This Automatic Car Parking enables the parking of vehicles-floor after floor and thus reducing the space used. Here any number of cars can be park according to requirement. These make the system modernized and evens a space-saving one. Mathematical modelling is also done to identify the least car parking space available among the difference parking places in a city.

Serial No. Contents Page No.

Chapter 1: Introduction

Chapter 2: Design Description

Chapter 3: System Overview

Chapter 4: Hardware Description

Serial No. Contents Page No.

Chapter 5: Design and Implementation

Chapter 6: Advantages and applications

Chapter 7: Conclusion

Chapter 1

Introduction

Chapter 1: Introduction

1.1 Introduction of Automatic Multi-storage Car Parking System

The population of the world is continuously on the increase and towns and cities have grown up around their public transport system. The increasing population and expanding urban centers has been accomplished by increasing car ownership and increasing demand for movement for various purposes. Regardless of income or social status, the conditions under which people travel have become more and more difficult and sometimes absolutely intolerable. Demand for transport and travel intensity tends to increase sharply with the growing size of a city and town especially when the city center or major centers of activity continues to grow in terms of both size and employment. Parking in public areas can be very tasking with little or no form of security because it is fraught with all sorts of hazards created by either humans or lack of parking structures. In order to reduce the stress of parking and any form of danger or insecurity to cars and owners, adequate parking facilities must be provided to meet up for the demand of parking.

In addition, high population density**,** large number of pavement hawkers, sidewalk encroachments, heterogeneous nature of traffic and commercial area development along all the major roads have compounded the problem of congestion on the main as well as internal roads of these cities.

Over the years engineers and architects have found a way to create more parking spaces within minimum size of land by the design and construction of multi-storey car parks. Multi-storey car park also known as a parking garage or a parking structure is a building designed for car parking with a number of floors or levels on which parking takes place. It is essentially a stacked parking lot that has multiple access and exit system to avoid traffic congestion in and out. Car parking systems have been around almost since the time cars were invented. There are car parking systems in most areas where there is significant amount of traffic.

1.2 Aims and Objectives

This is one of the most unique way of parking lots of cars without the need for a lengthy driveway and huge space. Lifts and pallets are used to park the cars and later retrieve them. It helps in accommodating maximum cars in a limited space. Since it is a multi-storey parking system, very minimal space is required for the construction of this parking system to accommodate lots of cars. The cars are much safer in such parking system as others do not have an access to the car. Moreover, parking the car through narrow drive ways often results to scratches and dents in the car. This can be avoided in multi-storey parking system.

1.3 Methodology

Following are the methods we have used to complete our project as shown below:

1. Searching of different project related to Multi-storage Car Parking System was done by the internet along with block diagram.
2. All the components to be used are selected and searched in the market.
3. The project selection was done by discussing with the supervisor.
4. Analysis and implementation was done then we checked the output with respect to input through Arduino app.

5. According to our objectives, code program of the proposed system is developed using basic language with the help of Arduino software platform.
6. The hex code of the program being created by the Arduino software platform is burnt into the flash code memory of our microcontroller IC.
7. Testing is done at various levels to finalize the appropriate program for the most proper working of the system.

Chapter 2

Design Description

Chapter 2: Design Description

2.1 Requirement and Criteria

In order to understand the Smart Multi-storage Car Parking System, is necessary first to understand the characteristics of it. The parking system have four level which consist of sixteen parking slot. IR module detect the slot which is empty or full by the car. LED is used as the output of the IR module, if the slot is full Red LED becomes bright else Green LED becomes bright. Slot is selected by the keypad according the output LED. The system lift carrying the car and park in the selected parking slot. As if the car is bring out, dial the number of required slot number and the lift bring out the car from the parking slot. Since it is a multi-level parking system, very minimal space is required for the construction of this parking system to accommodate lots of cars. The cars are much safer in such parking system as others do not have an access to the car. Moreover parking the car through narrow drive ways often results to scratches and dents in the car. This can be avoided in multi-storage parking system.

2.2 Stepper Motor

A **stepper motor** is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor for feedback (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed. Switched reluctance motors are very large stepping motors with a reduced pole count, and generally are closed-loop commutated. Brushed DC motors rotate continuously when DC voltage is applied to their terminals. The stepper motor is known by its property to convert a train of input pulses (typically square wave pulses) into a precisely defined increment in the shaft position. Each pulse moves the shaft through a fixed angle. Stepper motors effectively have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. To make the motor shaft turn, first, one electromagnet is given power, which magnetically attracts the gear's teeth. When the gear's teeth are aligned to the first electromagnet, they are slightly offset from the next electromagnet. This means that when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one. From there the process is repeated. Each of those rotations is called a "step", with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle.

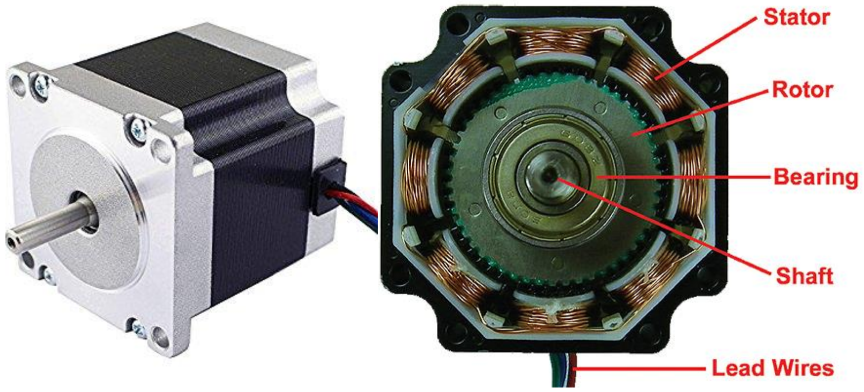

Figure 2.1: Stepper Motor

2.3 Motor Driver

A **motor Driver** is a device or group of devices that serves to govern in some predetermined manner the performance of an electric motor. A motor controller might include a manual or automatic means for starting and stopping the motor, selecting forward or reverse rotation, selecting and regulating the speed, regulating or limiting the torque, and protecting against overloads and faults**.** A stepper, or stepping, motor is a synchronous, brushless, high pole count, poly phase motor. Control is usually, but not exclusively, done open loop, i.e. the rotor position is assumed to follow a controlled rotating field. Because of this, precise positioning with steppers is simpler and cheaper than closed loop controls. Modern stepper controllers drive the motor with much higher voltages than the motor nameplate rated voltage, and limit current through chopping. The usual setup is to have a positioning controller, known as an indexer, sending step and direction pulses to a separate higher voltage drive circuit which is responsible for commutation and current limiting.

2.4 Circuit Design and Simulation

The circuit was simulated in the software Proteus v8.0 before hardware implementation. In this software, all the components of the circuit required were selected from the device library and connections were done by wire. The program was compiled using ARDUINO. The hex file of the program was loaded into the Proteus. Finally, the simulation was done. The circuit used for simulation of stepper motor, IR sensor, servo, LCD and 4x4 matrix keypad circuit. The connection to the microcontroller is shown in the circuit diagram.

2.5 Block Diagram

2.5.1 Block Diagram for Parking Cells

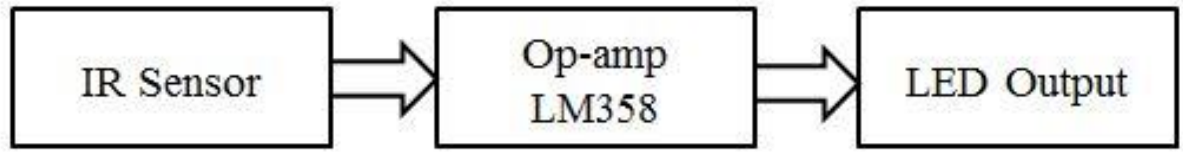

Figure 2.2: Block Diagram of Parking Cells

2.5.2 Block Diagram for Elevator

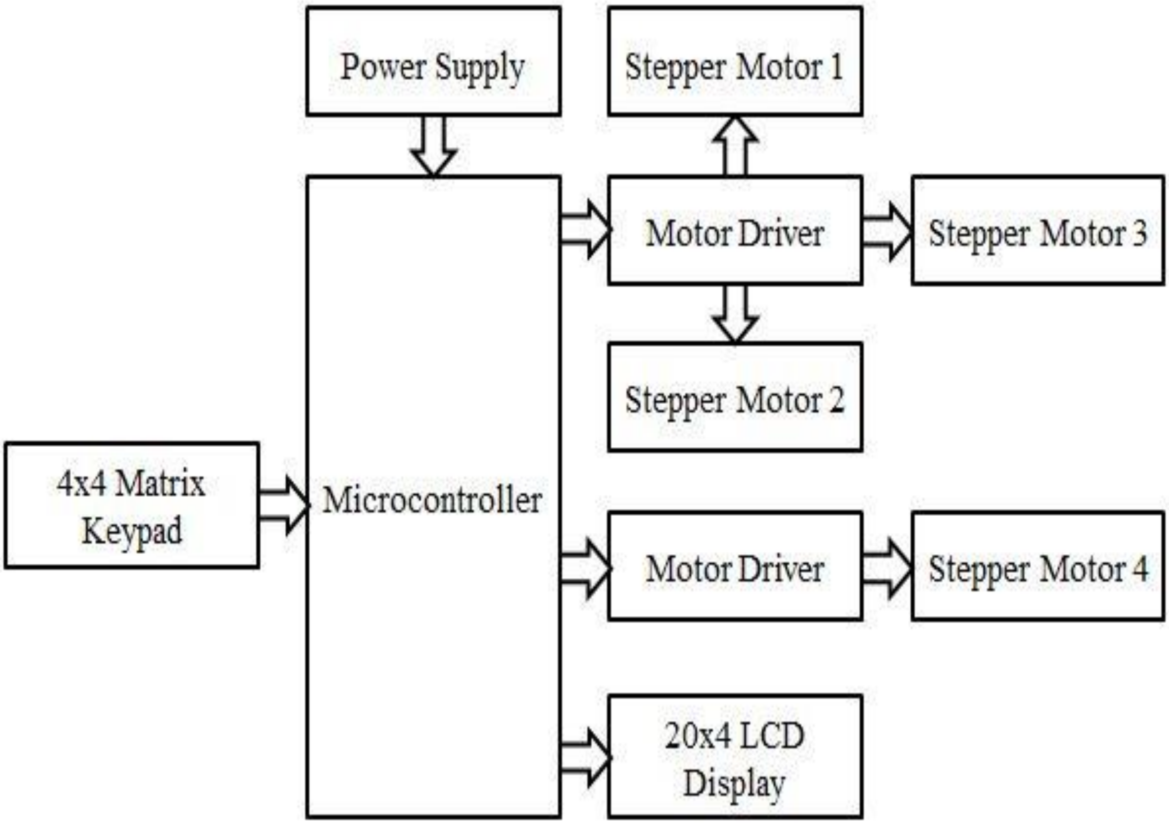

Figure 2.3: Block Diagram of Elevator

Chapter 3

System Overview

Chapter 3: System Overview

3.1 Electrical Architecture

The developed electronic hardware for the system is composed by, the microcontroller boards, the driver ICs and IR module.

3.2 Microcontroller Boards

The proposed system uses an ATMEGA microcontroller board which is programmed by open source IDE. The microcontroller is the central brain portion of this project. This component has the following various functions:

- It contains the code written using Arduino Coding Language stored in its memory.
- It accepts data from the computer or Bluetooth Module - It sends commands to the driver.
- It decodes the data sent from the Bluetooth Module and decides what action to take.
- It executes the logic to suit the requirements of the user.

3.3 Motor Driver – TB6560AHQ

The driver used 6N137 high-speed Opto-Coupler, to ensure a high speed without step out. Toshiba TB6560AHQ chip, low voltage shutdown of overheating parking and over current protection circuit to ensure optimal performance. Rated output: ± 3A, peak 3.5A. For 42, 57, 86 Stepper within 3A 2/4 phase/4 wire/6 wire stepper motor, not fit for more than 3A stepper motor. Excitation Mode: synchronizing, half step, 1/8 step, 1/16 step, a maximum of 16 segments. Volume: 50x75x35 (MM). Characteristics: the current level adjustable to meet various application needs, the Automatic half-decay adjustable, 6N137 high-speed optical coupling, to ensure a high speed not step out, thick tooth radiators, good heat dissipation.

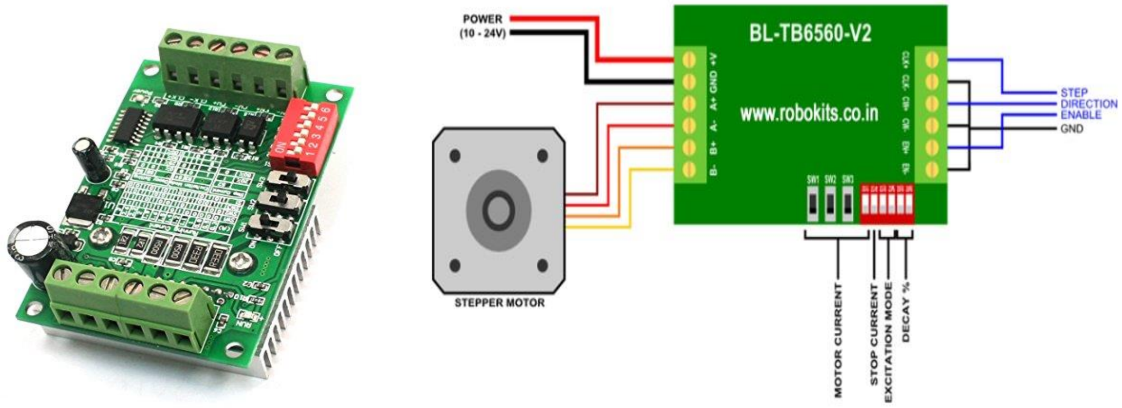

Figure 3.1: Motor Driver–TB6560AHQ

3.4 IR Sensor

An infrared sensor is an electronic device, that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measures only infrared radiation, rather than emitting it that is called as a passive IR sensor. Usually in the infrared spectrum, all the objects radiate some form of thermal radiations. These types of radiations are invisible to our eyes, that can be detected by an infrared sensor. The emitter is simply an IR LED and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, the resistances and these output voltages, change in proportion to the magnitude of the IR light received.

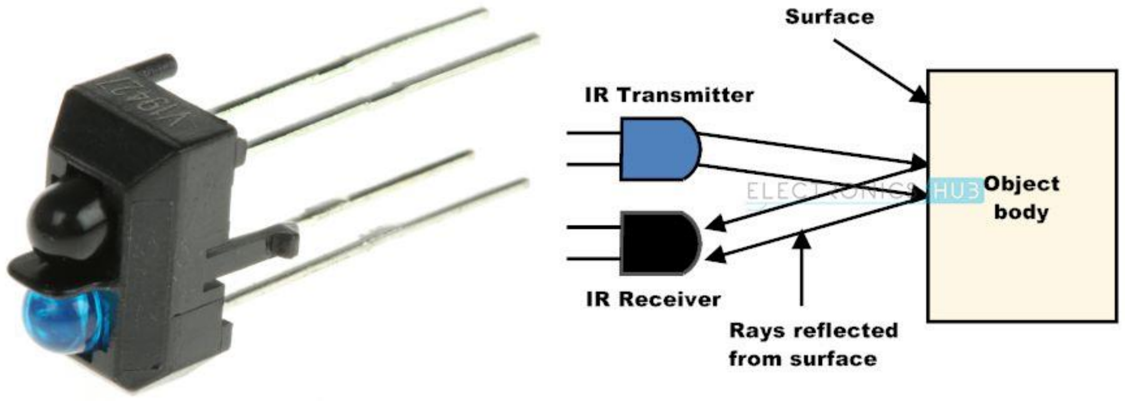

Figure 3.2: IR Sensor

Chapter 4

Hardware Description

Chapter 4: Hardware Description

4.1 Microcontroller

A microcontroller (sometimes abbreviated µC or MCU) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications in motor vehicles, robots, office machines, complex medical devices, mobile radio transceivers, vending machines, home appliances, and various other devices. They are available in different version starting from 6 pin to as high as 80 to 100 pins or even higher depending on the features. Whereas the microcontrollers operate from a few MHz to 30 to 50 MHz.

In contrast to the microprocessors used in personal computers or other general purpose applications where tasks are unspecific like developing software, games, websites, photo editing, creating documents etc. It is an integrated part in a real-time control or communication system. It has the specified computational capabilities and the enhanced I/O operation capabilities. Today's microprocessor operate above 1GHz as they perform complex tasks.

4.2 Why Microcontroller?

Microcontrollers have become common and very popular in many areas, and can be found hidden in many categories of products like home appliances, computer equipment, automobiles, etc. From a microwave oven to Refrigerator to high tech robots contains a microcontroller. Microcontrollers are used in running the traffic light at an intersection, in automobiles, the engine, the anti-lock brakes, the cruise control, industrial automation and so on. From the simplest gadgets, like digital cameras, cell phones, printing machines etc, to highly sophisticated satellite, robots and nuclear missile systems, the microcontrollers have become ubiquitous.

4.3 What is ARDUINO?

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures kits for building digital devices and interactive objects that can sense and control the physical world. Arduino boards may be purchased preassembled, or as do-it-yourself kits; at the same time, the hardware design information is available for those who would like to assemble an Arduino from scratch.

The project is based on a family of microcontroller board designs manufactured primarily by Smart Projects in Italy, and also by several other vendors, using various 8-bit Atmel AVR microcontrollers or 32-bit Atmel ARM processors. These systems provide sets of digital and analog I/O pins that can be interfaced to various extension boards and other circuits. The boards feature serial communications interfaces, including USB on some models, for loading programs from personal computers. For programming the microcontrollers, the Arduino platform provides an integrated development environment (IDE) based on the Processing project, which includes support for C and C++ programming languages. The first Arduino was introduced in 2005. The project leaders sought to provide an inexpensive and easy way for hobbyists, students, and professionals to create devices that interact with their environment using sensors and actuators. Common examples for beginner hobbyists include simple robots, thermostats and motion detectors. Adafruit Industries estimated in mid-2011 that over 300,000 official Arduinos had been commercially produced, and in 2013 that 700,000 official boards were in users' hands.

4.4 History of Arduino

Arduino started in 2005 as a project for students at the Interaction Design Institute Ivrea in Ivrea, Italy. At that time program students used a "BASIC Stamp" at a cost of 8000tk, considered expensive for students. Massimo Banzi, one of the founders, taught at Ivrea. The name "Arduino" comes from a bar in Ivrea, where some of the founders of the project used to meet. The bar itself was named after Arduino, Margrave of Ivrea and King of Italy.

4.5 What is Atmel?

Atmel Corporation is an American-based designer and manufacturer of semiconductors, founded in 1984. The company focuses on embedded systems built around microcontrollers. Its products include microcontrollers (8-bit AVR, 32-bit AVR, ARM-based, automotive grade, and Intel 8051 derivatives) radio frequency (RF) devices including Wi-Fi, EEPROM, and flash memory devices, symmetric and asymmetric security chips, touch sensors and controllers, and application-specific products. Atmel supplies its devices as standard products, application-specific integrated circuits (ASICs), or application-specific standard product (ASSPs) depending on the requirements of its customers.

4.6 What is AVR?

The AVR is a modified Harvard architecture8-bitRISC single-chip microcontroller, which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time. AVR chips became popular after they were designed into the 8-bit Arduino platform.

4.7 Why Arduino?

There are many other microcontrollers and microcontroller platforms available for physical computing. PIC board Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handy board, 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 850 tk.

- Cross-platform - The Arduino software runs on Windows, and Linux operating systems. Most microcontroller systems are limited to Windows.

- Simple, clear programming environment - The Arduino programming environment 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 the look and feel of Arduino.

- 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 Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules 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.

4.8 Getting Started with Arduino

Introduction: What Arduino is and why you'd want to use it.

Installation: Step-by-step instructions for setting up the Arduino software and connecting it to an Arduino Uno, Mega2560, Duemilanove, Mega, or Diecimila.

- Windows
- Mac OS X
- Linux

Environment: Description of the Arduino development environment and how to change the default language.

Libraries: Using and installing Arduino libraries.

Troubleshooting: Advice on what to do if things don't work.

4.9 Kinds of Arduino

- **Arduino UNO**

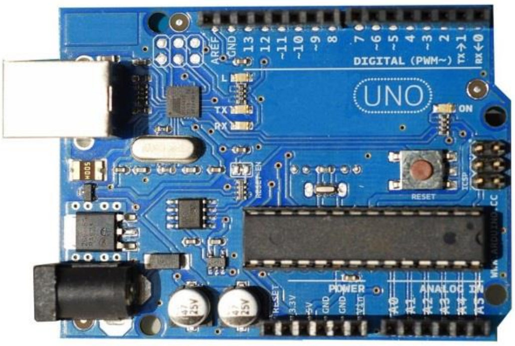

Figure 4.1: Arduino UNO

- **Arduino NANO**

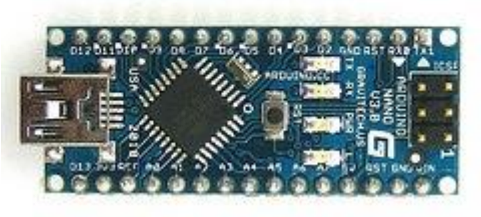

Figure 4.2: Arduino NANO

- **Arduino Lily PAD**

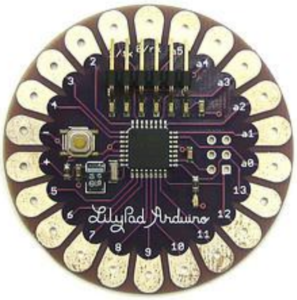

Figure 4.3: Arduino Lily PAD

- **Arduino MEGA 2560**

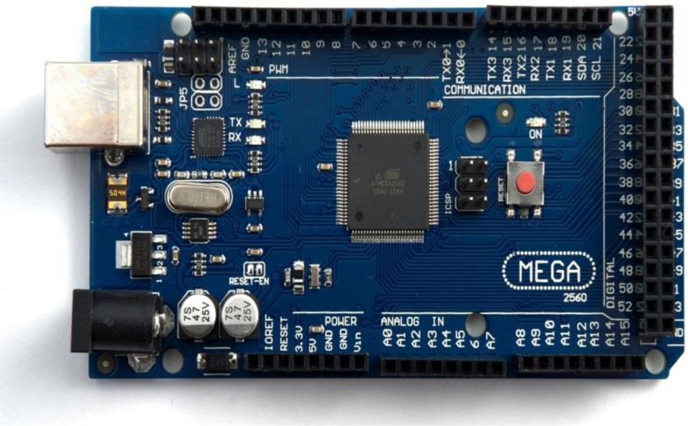

Figure 4.4: Arduino MEGA

- **Arduino NETDUINO**

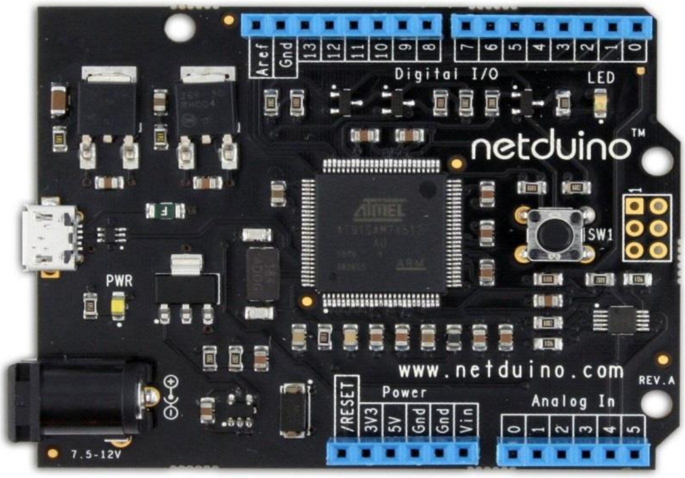

Figure 4.5: Arduino NETDUINO

- **Arduino Ethernet, etc.**

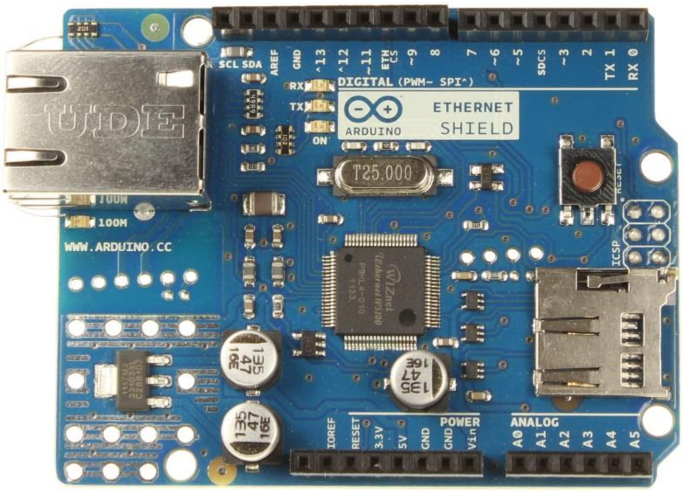

Figure 4.6: Arduino Ethernet

4.10 What is Arduino UNO?

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDIUSB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform.

4.11 Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader

SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

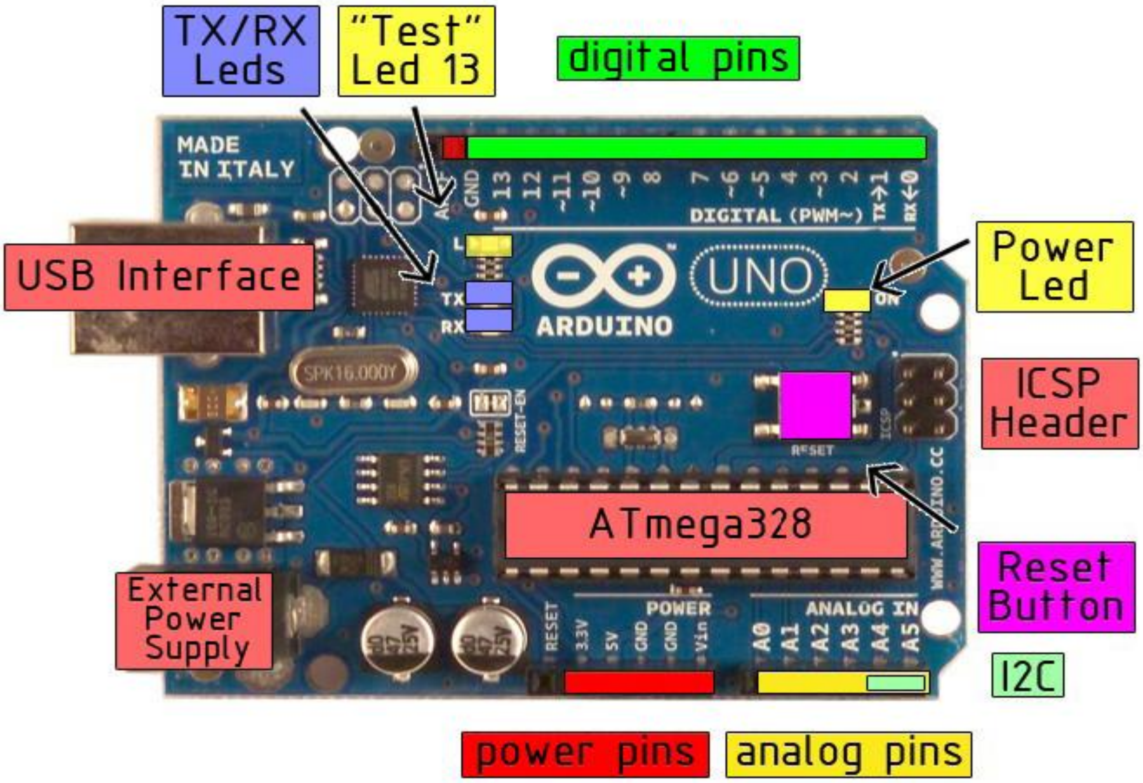

Figure 4.7: Arduino UNO Layout

4.12 Pin Diagram of Atmega328p microcontroller

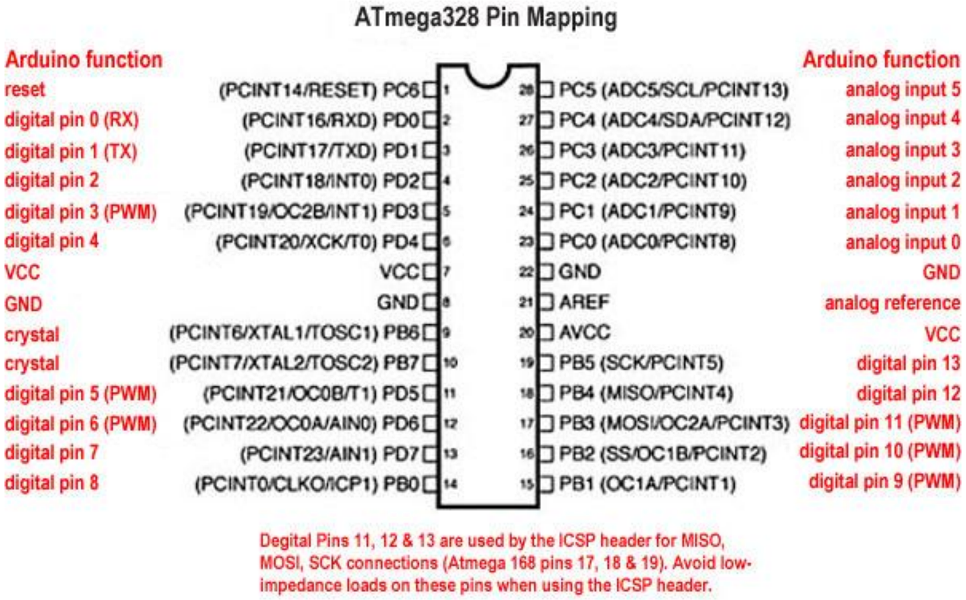

Figure 4.8: Pin Diagram of Atmega328p

4.13 Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

- **GND.** Ground pins.

- **IOREF.** This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

4.14 Memory

The ATmega328 chip found on the Uno has the following amounts of memory:

Flash 32k bytes (of which .5k is used for the bootloader)
SRAM   2k bytes
EEPROM 1k byte

4.15 Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digital Write(), and digital Read() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attach Interrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analog Write() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

4.16 Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A Software Serial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

4.17 Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using Arduino ISP or similar; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

4.18 Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

4.19 USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

4.20 Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

4.21 Block Diagram Arduino UNO Microcontroller

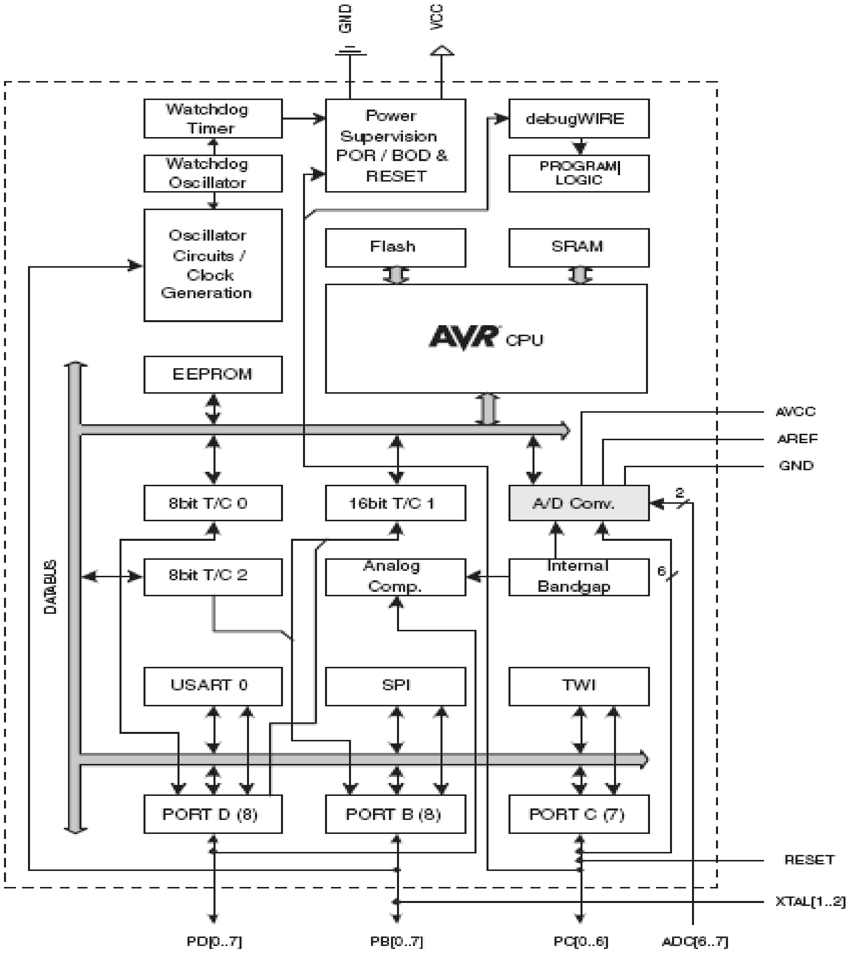

Figure 4.9: Block diagram of Atmega328p

4.22 Pin Description of Atmega 328P:

4.22.1 Pin Descriptions

4.22.2 VCC

Digital supply voltage.

4.22.3 GND

Ground.

4.22.4 Port B (PB7:0) 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, PB7..6 is used as TOSC2.1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

4.22.5 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..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.

4.22.6 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 un-programmed, 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.

4.22.7 Port D (PD7:0)

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.

4.22.8 AVCC

AVCC is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. 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 PC6..4 use digital supply voltage, VCC.

4.22.9 AREF

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

4.22.10 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter.

These pins are powered from the analog supply and serve as 10-bit ADC channels.

4.23 How to use

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. http://arduino.cc/en/Guide/HomePage.

Once you have downloaded/unzipped the arduino IDE, you can plug the Arduino to your PC via USB cable.

Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well-known programming "hello world", select **File>Sketchbook> Arduino-0017>Examples>Digital>Blink**

Once you have your sketch you'll see something very close to the screenshot on the right. In **Tools>Board** select Now you have to go to **Tools>SerialPort**and select the right serial port, the one arduino is attached to.

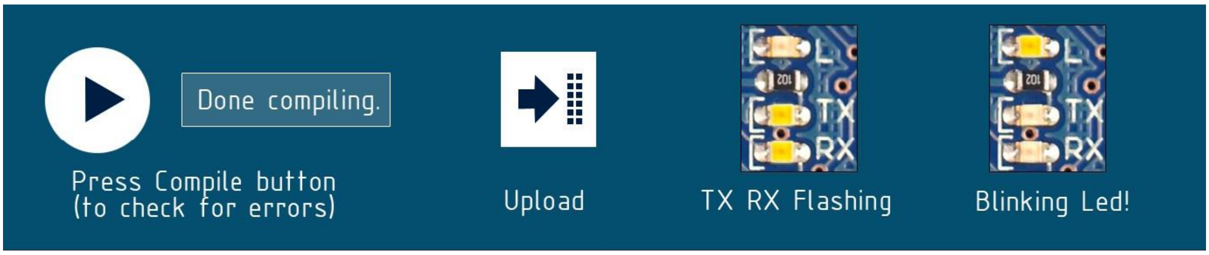

Figure 4.10: How to use

4.24 Dimensioned Drawing

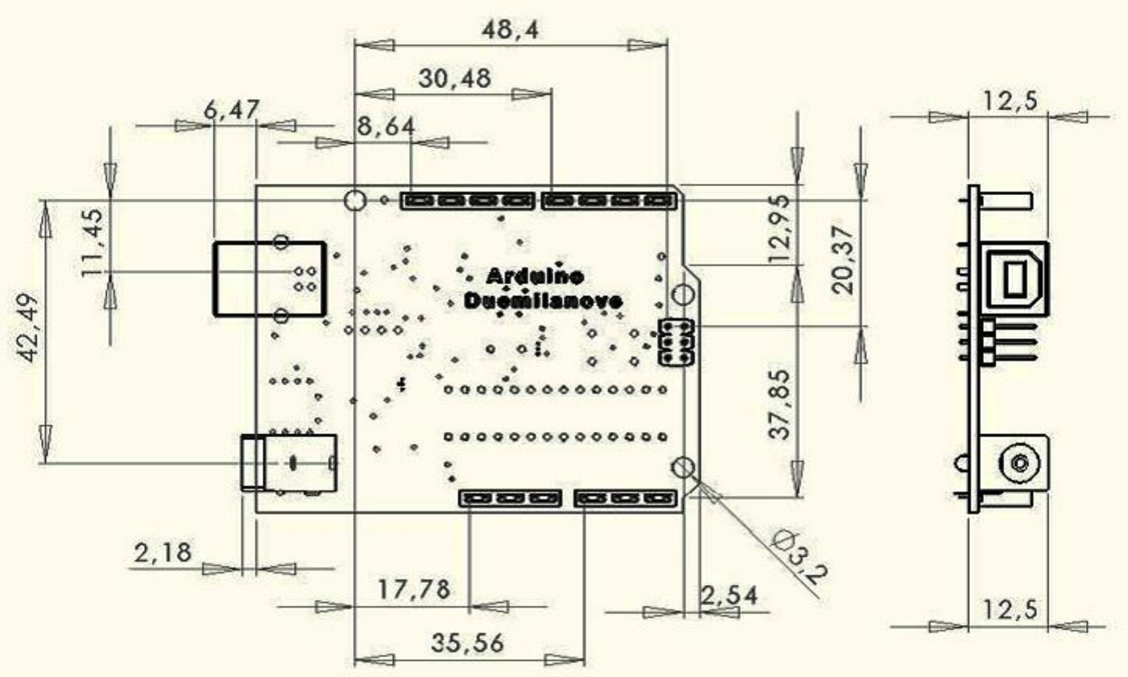

Figure 4.11: Arduino UNO Drawing

4.25 Schematic, Reference Design & Pin Mapping

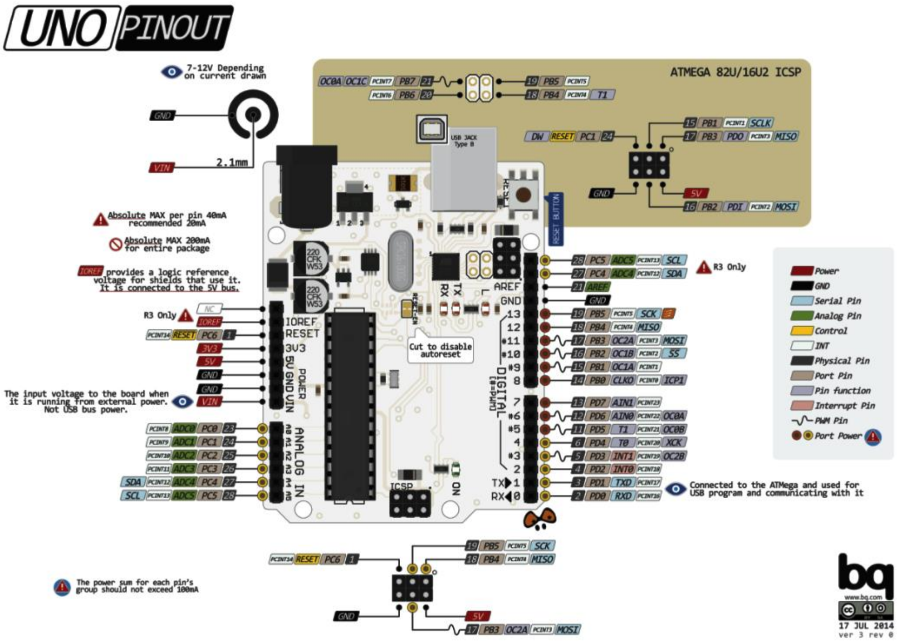

Figure 4.12: Arduino UNO Pin Mapping

4.26 Overview

Arduino/Genuino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

4.27 Features of Arduino UNO Microcontroller

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P)
	of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)

Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

4.28 Stepper Motor – EM242

A stepper motor is an electromechanical device it converts electrical power into mechanical power. Also it is a brushless, synchronous electric motor that can divide a full rotation into an expansive number of steps. The motor’s position can be controlled accurately without any feedback mechanism, as long as the motor is carefully sized to the application. Stepper motors are similar to switched reluctance motors.

The stepper motor uses the theory of operation for magnets to make the motor shaft turn a precise distance when a pulse of electricity is provided. The stator has eight poles, and the rotor has six poles. The rotor will require 24 pulses of electricity to move the 24 steps to make one complete revolution. Another way to say this is that the rotor will move precisely 15° for each pulse of electricity that the motor receives.

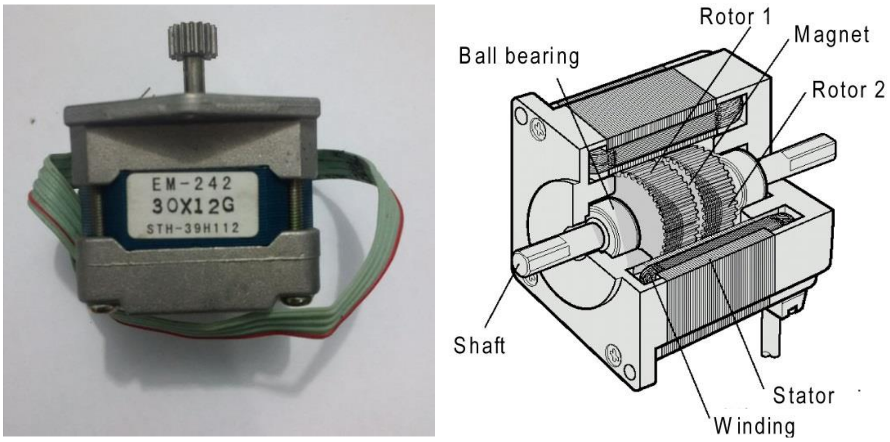

Figure 4.13: Stepper Motor-EM242

4.28.1 Working Principle of Stepper Motor

A stepper motor is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor for feedback (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed. Switched reluctance motors are very large stepping motors with a reduced pole count, and generally are closed-loop commutated. Brushed DC motors rotate continuously when DC voltage is applied to their terminals. Also discussed was the many uses at which stepper motors can be used. There are two basic winding arrangements for the electromagnetic coils in a two phase stepper motor, one being bipolar and the other is unipolar stepper motor. The unipolar stepper motor operates with one winding with a centre tap per phase.

The stepper motor is known by its property to convert a train of input pulses (typically square wave pulses) into a precisely defined increment in the shaft position. Each pulse moves the shaft through a fixed angle. Stepper motors effectively have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. To make the motor shaft turn, first, one electromagnet is given power, which magnetically attracts the gear's teeth. When the gear's teeth are aligned to the first electromagnet, they are slightly offset from the next electromagnet. This means that when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one. From there the process is repeated. Each of those rotations is called a "step", with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle.

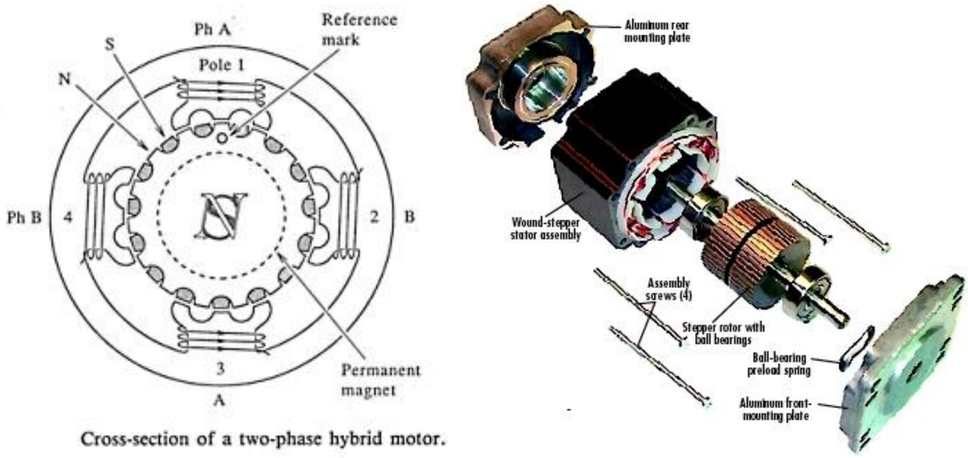
Ph A
Reference mark
S
N
Pole 1
Ph B
4
2
B
3
A
Permanent magnet
Cross-section of a two-phase hybrid motor.
Aluminum rear mounting plate
Wound-stepper stator assembly
Assembly screws (4)
Stepper rotor with ball bearings
Ball-bearing preload spring
Aluminum front-mounting plate

Figure 4.14: Parts of Stepper Motor

Stepper motors are either bipolar, requiring two power sources or a switchable polarity power source, or unipolar, requiring only one power source. They are powered by dc current sources and require digital circuitry to produce the coil energizing sequences for rotation of the motor.

4.29 Motor Driver – TB6560AHQ

The driver used 6N137 high-speed Opto-Coupler, to ensure a high speed without step out. Toshiba TB6560AHQ chip, low voltage shutdown of overheating parking and over current protection circuit to ensure optimal performance.

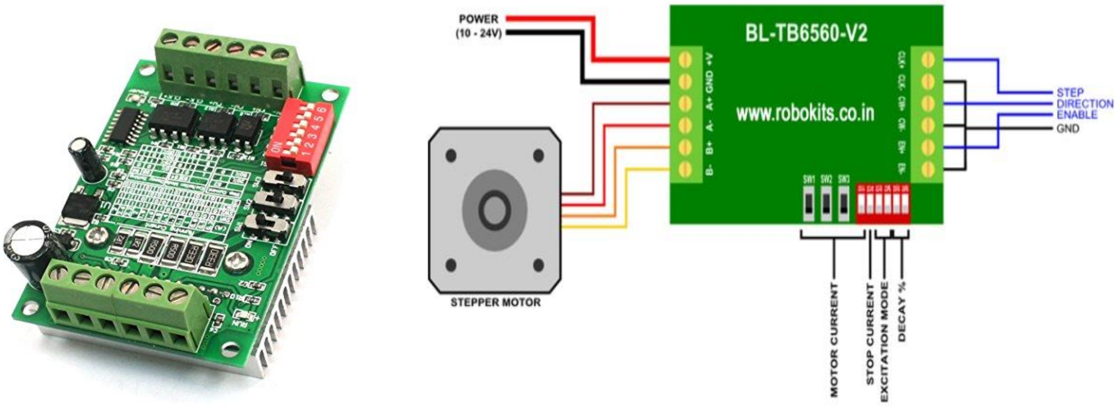

Figure 4.15: Motor Driver–TB6560AHQ

Rated output: ± 3A, peak 3.5A. For 42, 57, 86 Stepper within 3A 2/4 phase/4 wire/6 wire stepper motor, not fit for more than 3A stepper motor. Excitation Mode: synchronizing, half step, 1/8 step, 1/16 step, a maximum of 16 segments. Volume: 50x75x35 (MM). Characteristics: the current level adjustable to meet various application needs, the Automatic half-decay adjustable, 6N137 high-speed optical coupling, to ensure a high speed not step out, thick tooth radiators, good heat dissipation.

4.30 Power Supply

The electrical power is almost exclusively generated, transmitted and distributed in the form of ac because of economical consideration but for operation of most of the electronic devices and circuits, dc supply is required. Dry cells and batteries can be used for this purpose. No doubt, they have the advantages of being portable and ripple free but their voltages are low, they need frequent replacement and are expensive in comparison to conventional dc power supplies.

Figure 4.16: DC Power Supply

Now-a-days, almost all electronic equipment include a circuit that converts ac supply into dc supply. The part of equipment that converts ac into dc is called DC power supply. In general at the input of the power supply there is a power transformer. It is followed by a rectifier (a diode circuit)a smoothing filter and then by a voltage regulator circuit.

From the block diagram, the basic power supply is constituted by four elements viz a transformer, a rectifier, a filter, and a regulator put together. The output of the dc power supply is used to provide a constant dc voltage across the load. Let us briefly outline the function of each of the elements of the dc power supply.

Transformer is used to step-up or step-down (usually to step-down) the-supply voltage as per need of the solid-state electronic devices and circuits to be supplied by the dc power supply. It can provide isolation from the supply line-an important safety consideration. It may also include internal shielding to prevent unwanted electrical noise signal on the power line from getting into the power supply and possibly disturbing the load.

4.30.1 Rectifier

Rectifier is a device which converts the sinusoidal ac voltage into either positive or negative pulsating dc. P-N junction diode, which conducts when forward biased and practically does not conduct when reverse biased, can be used for rectification i.e. for conversion of ac into dc. The rectifier typically needs one, two or four diodes. Rectifiers may be either half-wave rectifiers or full-wave rectifiers (center-tap or bridge) type. The output voltage from a rectifier circuit has a pulsating character i.e., it contains unwanted ac components

(components of supply frequency f and its harmonics) along with dc component. For most supply purposes, constant direct voltage is required than that furnished by a rectifier. To reduce ac components from the rectifier output voltage a filter circuit is required.

Thus filter is a device which passes dc component to the load and blocks I ac components of the rectifier output. Filter is typically constructed from reactive circuit I elements such as capacitors and/or inductors and resistors. The magnitude of output dc voltage may vary with the variation of either the input ac voltage or the magnitude of load current. So at the output of a rectifier filter combination a voltage regulator is required, to provide an almost constant dc voltage at the output of the regulator. The voltage regulator may be constructed from a Zener diode, and or discrete transistors, and/or integrated circuits (ICs). Its main function is to maintain a constant dc output voltage. However, it also rejects any ac ripple voltage that is not removed by the filter. The regulator may also include protective devices such as short-circuit protection, current limiting, thermal shutdown, or over-voltage protection.

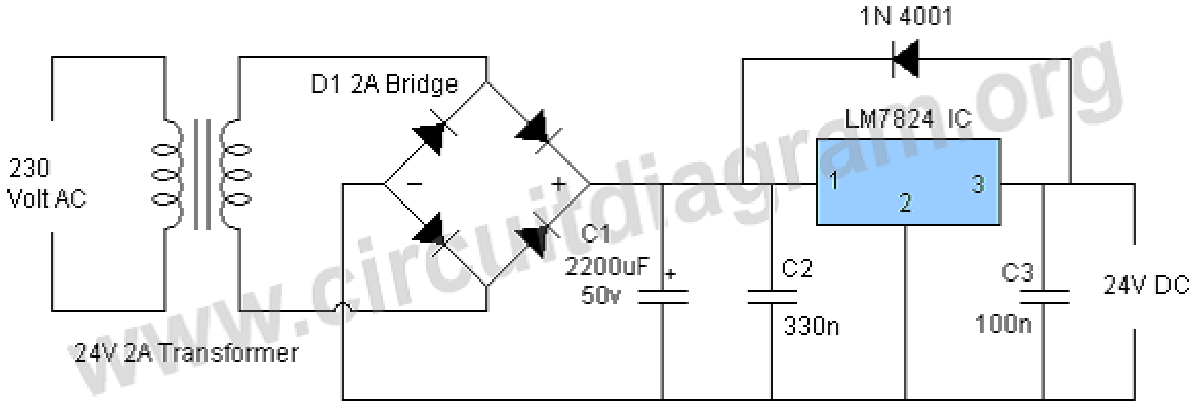

Figure 4.17: Circuit diagram of 24V DC power supply

4.31 IR Sensor

An infrared sensor is an electronic device that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measures only infrared radiation, rather than emitting it that is called as a passive IR sensor. Usually in the infrared spectrum, all the objects radiate some form of thermal radiations.

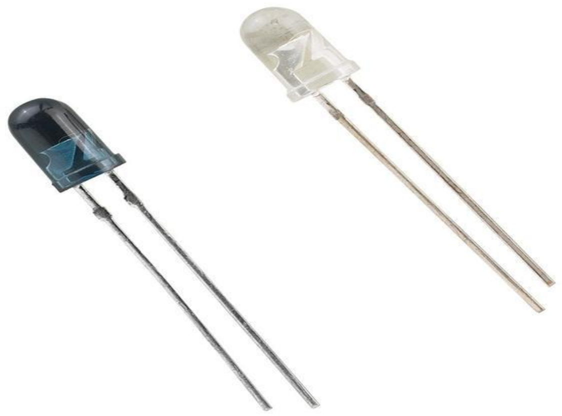

Figure 4.18: IR Sensor

These types of radiations are invisible to our eyes that can be detected by an infrared sensor. The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, the resistances and these output voltages, change in proportion to the magnitude of the IR light received.

4.31.1 Working Principle of IR Sensor

An infrared sensor circuit is one of the basic and popular sensor module in an electronic device. This sensor is analogous to human's visionary senses, which can be used to detect obstacles and it is one of the common applications in real time. In this project, the transmitter section includes an IR sensor, which transmits continuous IR rays to be received by an IR receiver module. An IR output terminal of the receiver varies depending upon its receiving of IR rays. Since this variation cannot be analysed as such, therefore this output can be fed to a comparator circuit. Here an operational amplifier (op-amp) of LM 358 is used as comparator circuit.

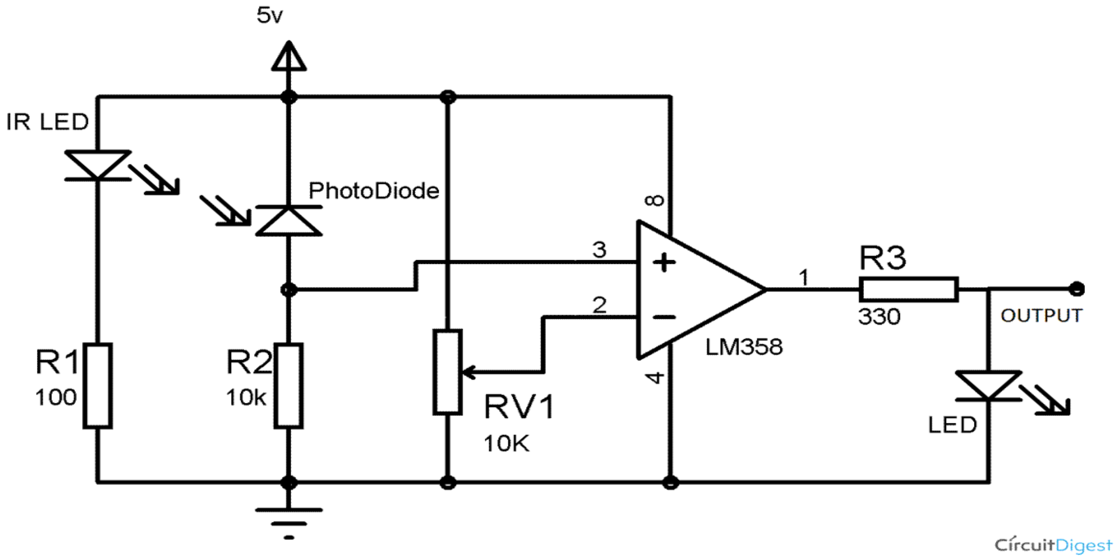

Figure 4.19: Circuit Diagram of IR Sensor

When the IR receiver does not receive a signal, the potential at the inverting input goes higher than that non-inverting input of the comparator IC (LM339). Thus the output of the comparator goes low, but the LED does not glow. When the IR receiver module receives signal to the potential at the inverting input goes low. Thus the output of the comparator (LM 358) goes high and the LED starts glowing. Resistors are used to ensure that minimum 10 mA current passes through the IR LED Devices like Photodiode and normal LEDs respectively. Resistor VR2 is used to adjust the output terminals. Resistor VR1 is used to set the sensitivity of the circuit Diagram.

4.31.2 Operational Amplifier – LM358

The LM358 IC is a great, low power and easy to use dual channel op-amp IC. It is designed and introduced by national semiconductor. It consists of two internally frequency compensated, high gain, independent op-amps. This IC is designed for specially to operate from a single power supply over a wide range of voltages. The LM358 IC is available in a chip sized package and applications of this op amp include conventional op-amp circuits, DC gain blocks and transducer amplifiers. LM358 IC is a good, standard operational amplifier and it is suitable for your needs. It can handle 3-32V DC supply & source up to 20mA per channel. This op-amp is apt, if you want to operate two separate op-amps for a single power supply. It’s available in an 8-pin DIP package. The LM358 series are operational amplifiers which can operate with only a single power supply voltage, have true-differential inputs, and remain in the linear mode with an input common-mode voltage of 0 V DC. These amplifiers operate over a wide range of power supply voltage with little change in performance characteristics. At 25°C amplifier operation is possible down to a minimum supply voltage of 2.3 V DC. Large differential input voltages can be easily accommodated and, as input differential voltage protection diodes are not needed, no large input currents result from large differential input voltages. The differential input voltage may be larger than V+ without damaging the device. Protection should be provided to prevent the input voltages from going negative more than −0.3 V DC (at 25°C). An input clamp diode with a resistor to the IC input terminal can be used.

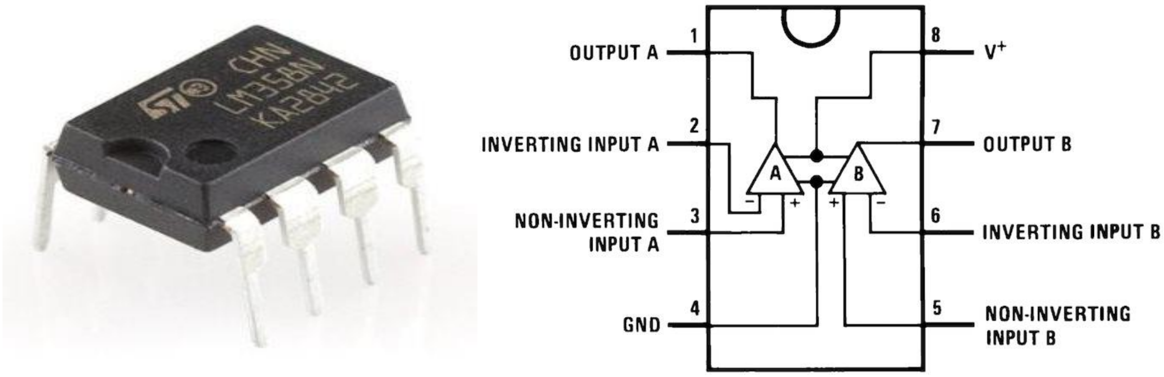

Figure 4.20: Operational Amplifier – LM358

4.32 Liquid Crystal Display (LCD)

Liquid crystal display (LCD)**,** electronic display device that operates by applying a varying electric voltage to a layer of liquid crystal, thereby inducing changes in its optical properties. LCDs are commonly used for portable electronic games, as viewfinders for digital cameras and camcorders, in video projection systems, for electronic billboards, as monitors for computers, and in flat-panel televisions.

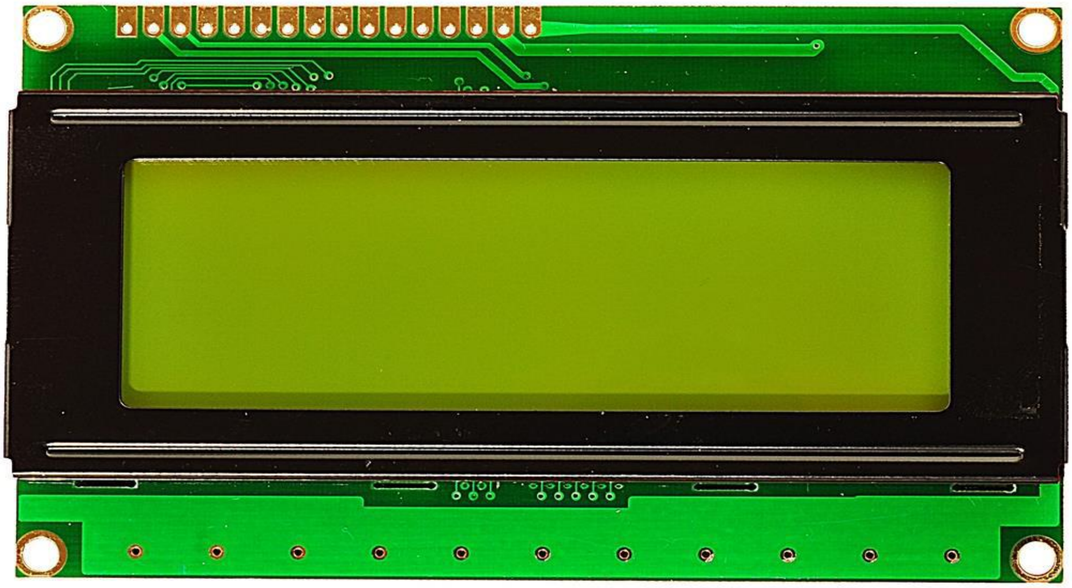

Figure 4.21: 20×4 Liquid Crystal Display (LCD)

4.32.1 Electro optical effects in liquid crystals

Liquid crystals are materials with a structure that is intermediate between that of liquids and crystalline solids. As in liquids, the molecules of a liquid crystal can flow past one another. As in solid crystals, however, they arrange themselves in recognizably ordered patterns. In common with solid crystals, liquid crystals can exhibit polymorphism; i.e., they can take on different structural patterns, each with unique properties. LCDs utilize either nematic or smectic liquid crystals. The molecules of nematic liquid crystals align themselves with their axes in parallel, as shown in the figure. Smectic liquid crystals, on the other hand, arrange themselves in layered sheets; within different smectic phases the molecules may take on different alignments relative to the plane of the sheets.

The optical properties of liquid crystals depend on the direction light travels through a layer of the material. An electric field (induced by a small electric voltage) can change the orientation of molecules in a layer of liquid crystal and thus affect its optical properties. Such a process is termed an electro-optical effect, and it forms the basis for LCDs. For nematic LCDs, the change in optical properties results from orienting the molecular axes either along or perpendicular to the applied electric field, the preferred direction being determined by the details of the molecule’s chemical structure. Liquid crystal materials that align either parallel or perpendicular to an applied field can be selected to suit particular applications.

Chapter 5

Design and Implementation

Chapter 5: Design and Implementation

5.1 Circuit Diagram

5.1.1 Circuit Diagram for Elevator

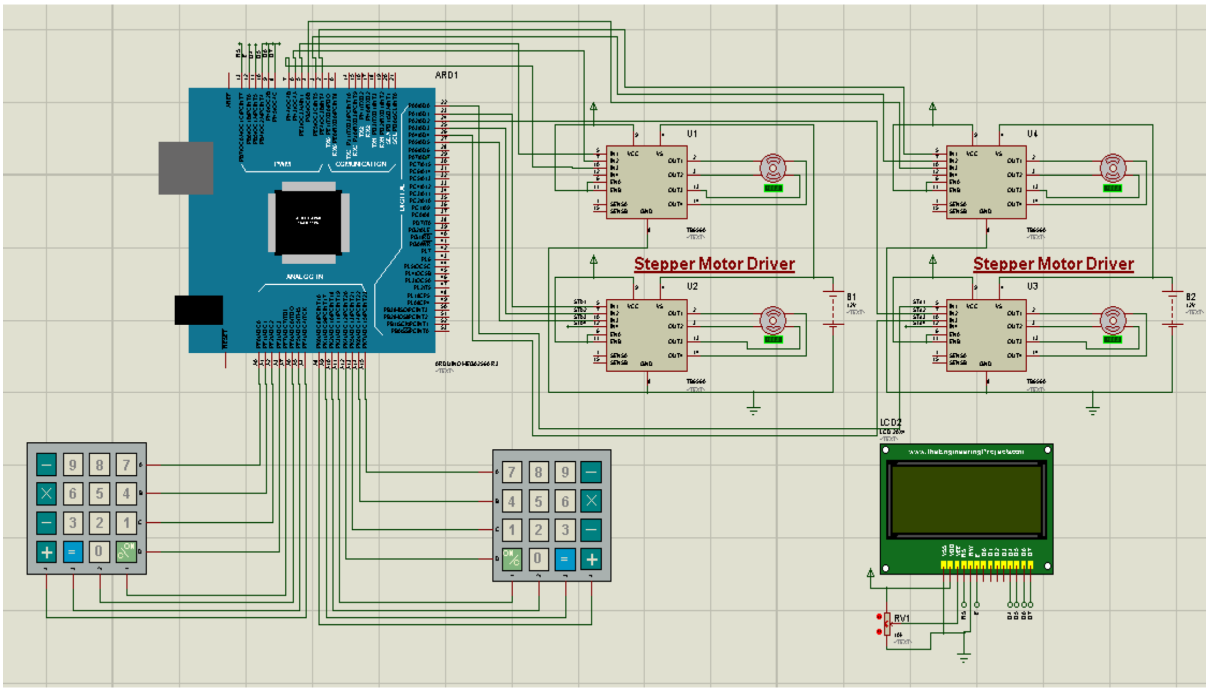

Figure 5.1: Circuit Diagram of Elevator

5.1.2 Circuit Diagram for Parking Cell

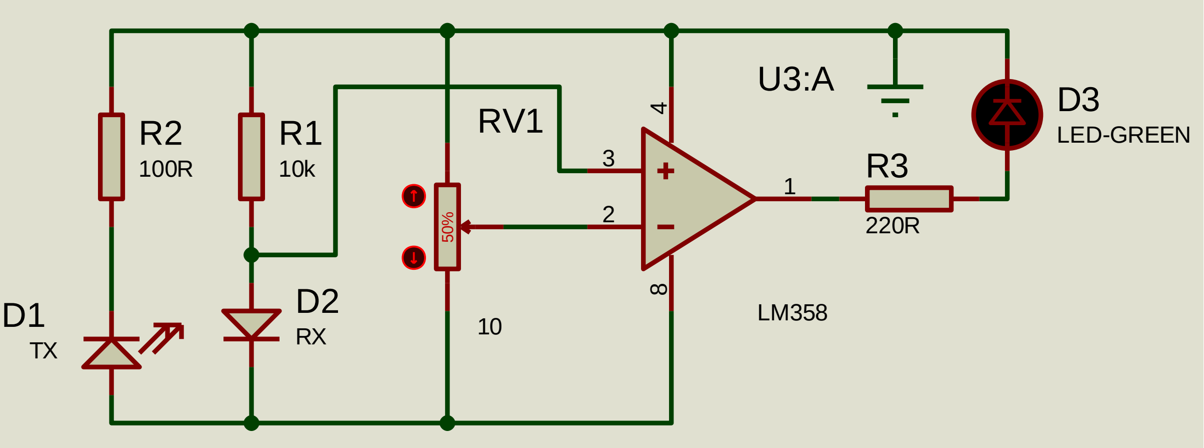

Figure 5.2: Circuit Diagram of Parking Cell

5.2 Source Code of Arduino

```
const int stepPin1 = 5; //SLIDER EVEN
const int dirPin1 = 6;
const int enPin1 = 7;
const int stepPin2 = 23; //SLIDER ODD
const int dirPin2 = 25;
const int enPin2 = 27;
const int stepPin3 = 22; //180 ROTATE STEPPER
const int dirPin3 = 24;
const int enPin3 = 26;
const int stepPin4 = 2; //BIG MOTOR
const int dirPin4 = 3;
const int enPin4 = 4;
#include <Keypad.h>
/*------------------------------KEYPAD--------------------------------------*/
const byte ROWS = 8; // Four rows
const byte COLS = 8; // columns
char keypressed;
// Define the Keymap
char keymap [ROWS][COLS] = {
{'X','2','3','A','E','I','0','Q'},
{'4','5','6','B','F','J','N','R'},
{'7','8','9','C','G','K','O','S'},
{'*','M','#','D','H','l','P','T'},
{'U','Y','c','g','k','o','s','w'},
{'V','Z','d','h','L','p','t','x'},
{'W','a','e','i','m','q','u','y'},
{'1','b','f','j','n','r','v','z'}
};
// Connect keypad ROW0, ROW1, ROW2 and ROW3 to these Arduino pins.
```

```
byte rowPins[ROWS] = { 69, 68, 67, 66, 61, 60, 59, 58 };// Connect keypad COL0, COL1 and COL2 to these Arduino pins.

byte colPins[COLS] = { 65, 64, 63, 62, 57, 56, 55, 54};

// Create the Keypad

Keypad customKeypad = Keypad( makeKeymap(keymap), rowPins, colPins, ROWS, COLS);

void setup() {

 // put your setup code here, to run once:

Serial.begin(9600);

 // Sets the two pins as Outputs

 pinMode(stepPin1,OUTPUT);

 pinMode(dirPin1,OUTPUT);

 pinMode(enPin1,OUTPUT);

 digitalWrite(enPin1,LOW);

 pinMode(stepPin2,OUTPUT);

 pinMode(dirPin2,OUTPUT);

 pinMode(enPin2,OUTPUT);

 digitalWrite(enPin2,LOW);

 pinMode(stepPin3,OUTPUT);

 pinMode(dirPin3,OUTPUT);

 pinMode(enPin3,OUTPUT);

 digitalWrite(enPin3,LOW);

 pinMode(stepPin4,OUTPUT);

 pinMode(dirPin4,OUTPUT);

 pinMode(enPin4,OUTPUT);

 digitalWrite(enPin4,LOW);

}

void loop() {

  keypressed = customKeypad.getKey();   //Read pressed keys

    if ( keypressed == 'z'  )

    {
```

```
  park1();
 }
 if ( keypressed == 'y'  )
 {
  park2();
 }
  if ( keypressed == 'x'  )
 {
  park3();
}
  if ( keypressed == 'w'  )
  {
 park4();
  }
  if ( keypressed == 'v'  )
 {
  park5();
 }
  if ( keypressed == 'u'  )
 {
  park6();
 }
  if ( keypressed == 't'  )
 {
  park7();
 }
  if ( keypressed == 's'  )
 {
  park8();
 }
```

```
      if ( keypressed == 'r'  )
{
  park9();
}
if ( keypressed == 'q'  )
{
  park10();
}
 if ( keypressed == 'p'  )
{
  park11();
}
 if ( keypressed == 'o'  )
 {
park12();
 }
 if ( keypressed == 'n'  )
{
  park13();
}
 if ( keypressed == 'm'  )
{
  park14();
}
 if ( keypressed == 'L'  )
{
  park15();
}
 if ( keypressed == 'k'  )
{
  park16();
}
///////////////////////////////////GET COMMAND/////////////////////////////
if ( keypressed == 'D'  )
{
  get1();
}
if ( keypressed == 'C'  )
{
  get2();
}
 if ( keypressed == 'B'  )
{
  get3();
}
if ( keypressed == 'A'  )
{
  get4();
}
if ( keypressed == '#'  )
{
  get5();
}
if ( keypressed == '9'  )
{
  get6();
}
if ( keypressed == '6'  )
{
  get7();
```

```
}
if ( keypressed == '3'  )
{
  get8();
}
if ( keypressed == 'M'  )
{
  get9();
}
if ( keypressed == '8'  )
{
  get10();
}
 if ( keypressed == '5'  )
{
  get11();
}
if ( keypressed == '2'  )
{
  get12();
}
if ( keypressed == '*'  )
{
  get13();
}
if ( keypressed == '7'  )
{
  get14();
}
if ( keypressed == '4'  )
```

```
      {
        get15();
      }
      if ( keypressed == 'X'  )
      {
        get16();
      }
}
void park1()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
```

```
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
```

```
void park2()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park3()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park4()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park5()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park6()
{
    digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
```

```
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
```

```
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 21000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
```

```
void park7()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park8()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park9()
{
}
void park10()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
 // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 23000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(100);
     digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
  //slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
 digitalWrite(dirPin4,HIGH); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 21000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(100);
     digitalWrite(dirPin4,HIGH); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
}
void park11()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park12()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
```

```
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
```

```
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park13()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park14()
{
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park15()
{
   digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void park16()
{
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
//Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 9000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get1()
{
  //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
```

```
 delay(1000); // One second delay
  //Upore utho
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
 // Makes 400 pulses for making two full cycle rotation
 for(int x = 0; x < 2000; x++) {
   digitalWrite(stepPin4,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin4,LOW);
   delayMicroseconds(500);
 }
 delay(1000);
//Slide out
  digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 700; x++) {
   digitalWrite(stepPin1,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin1,LOW);
   delayMicroseconds(500);
 }
 delay(1000); // One second delay
 //Ekdom nice namo
 digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 11300; x++) {
   digitalWrite(stepPin4,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin4,LOW);
   delayMicroseconds(500);
 }
```

```
  delay(1000); // One second delay
}
void get2()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get3()
{
 //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get4()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
```

```
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get5()
{
    //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
```

```
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get6()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 21000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 23000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get7()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
```

```
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get8()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get9()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(500);
 }
 delay(1000); // One second delay
  //Upore utho
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
 // Makes 400 pulses for making two full cycle rotation
 for(int x = 0; x < 2000; x++) {
  digitalWrite(stepPin4,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin4,LOW);
  delayMicroseconds(500);
 }
 delay(1000);
//Slide out
  digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 700; x++) {
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(500);
 }
 delay(1000); // One second delay
 //Ekdom nice namo
 digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 11300; x++) {
  digitalWrite(stepPin4,HIGH);
```

```
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get10()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 20000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(100);
     digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 12000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
```

```
 for(int x = 0; x < 700; x++) {
   digitalWrite(stepPin2,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin2,LOW);
   delayMicroseconds(500);
 }
 delay(1000); // One second delay
  //Upore utho
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
 // Makes 400 pulses for making two full cycle rotation
 for(int x = 0; x < 2000; x++) {
   digitalWrite(stepPin4,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin4,LOW);
   delayMicroseconds(500);
 }
 delay(1000);
//Slide out
  digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 700; x++) {
   digitalWrite(stepPin2,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin2,LOW);
   delayMicroseconds(500);
 }
 delay(1000); // One second delay
 //Ekdom nice namo
 digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
```

```
  for(int x = 0; x < 23000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
    digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get11()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
```

```
 digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
// Makes 200 pulses for making one full cycle rotation
for(int x = 0; x < 700; x++) {
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(500);
}
delay(1000); // One second delay
 //Upore utho
digitalWrite(dirPin4,LOW); //Changes the rotations direction
// Makes 400 pulses for making two full cycle rotation
for(int x = 0; x < 2000; x++) {
  digitalWrite(stepPin4,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin4,LOW);
  delayMicroseconds(500);
}
delay(1000);
//Slide out
 digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
// Makes 200 pulses for making one full cycle rotation
for(int x = 0; x < 700; x++) {
  digitalWrite(stepPin1,HIGH);
  delayMicroseconds(500);
  digitalWrite(stepPin1,LOW);
  delayMicroseconds(500);
}
delay(1000); // One second delay
//Ekdom nice namo
```

```
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get12()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
```

```
 }
 delay(1000); // One second delay
  //Upore utho
 digitalWrite(dirPin4,LOW); //Changes the rotations direction
 // Makes 400 pulses for making two full cycle rotation
 for(int x = 0; x < 2000; x++) {
   digitalWrite(stepPin4,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin4,LOW);
   delayMicroseconds(500);
 }
 delay(1000);
//Slide out
  digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 700; x++) {
   digitalWrite(stepPin2,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin2,LOW);
   delayMicroseconds(500);
 }
 delay(1000); // One second delay
 //Ekdom nice namo
 digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
 // Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 11300; x++) {
   digitalWrite(stepPin4,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin4,LOW);
   delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get13()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get14()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get15()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin1,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin1,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin1,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin1,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
void get16()
{
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 9300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
//slide in
   digitalWrite(dirPin2,LOW); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
   //Upore utho
  digitalWrite(dirPin4,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 2000; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000);
 //Slide out
   digitalWrite(dirPin2,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 700; x++) {
    digitalWrite(stepPin2,HIGH);
    delayMicroseconds(500);
    digitalWrite(stepPin2,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
  //Ekdom nice namo
  digitalWrite(dirPin4,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 11300; x++) {
    digitalWrite(stepPin4,HIGH);
    delayMicroseconds(500);
```

```
    digitalWrite(stepPin4,LOW);
    delayMicroseconds(500);
  }
  delay(1000); // One second delay
}
```

5.3 Outlook of the Smart Multi-storage Car Parking System

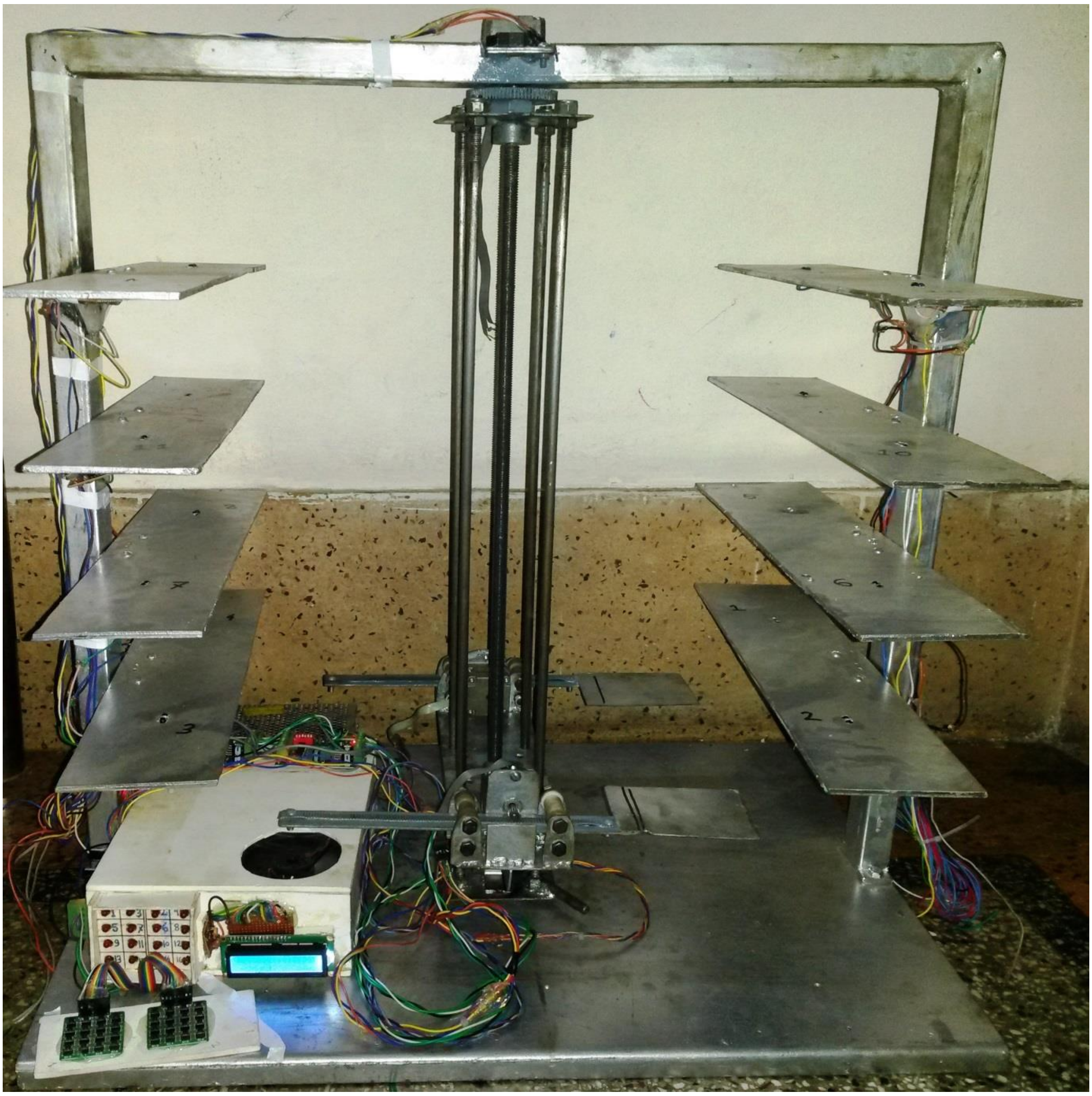

Figure 5.3: Outlook of the Smart Multi-storage Car Parking System

Chapter 6

Advantages and applications

Chapter 6: Advantages and applications

6.1 Advantages

- This is one of the most unique way of parking lots of cars without the need for a lengthy driveway and huge space. Lifts and pallets are used to park the cars and later retrieve them. It helps in accommodating maximum cars in a limited space.

- Since it is a multi-level parking system, very minimal space is required for the construction of this parking system to accommodate lots of cars.

- The cost involved in the construction of multilevel car parking system is low because it is pre-fabricated and is assembled on the site. A lot of hidden costs like security, lighting etc. are reduced.

- The maintenance cost of such parking system is extremely low.

- The cars are much safer in such parking system as others do not have an access to the car. Moreover parking the car through narrow driveways often results to scratches and dents in the car. This can be avoided in multilevel parking system.

- Parking a car in a multi-level parking system is hassle free for a driver as well. Since such parking systems minimize the risks of theft or vandalism, it consequently minimizes the tensions of the user. He/she need not have to worry about the car as long as it parked. Such parking systems are very comfortable to use.

- The product is 100% high quality hot dip galvanized which ensure that humidity, heavy rain and salty weather is no more an enemy of steels and systems. Rusting is no more a problems for end users and systems life is increased to at least 40-50 years with proper maintenance and service.

- This has been a specialty and unique feature of Krishna Park Infracon which delighted many customers. With excellent service and customer satisfaction, Krishna Park Infracon makes the systems user friendly. Even Ladies and Senior citizens can park their cars with immense ease.

6.2 Applications

- Shopping malls.
- Building.
- Restaurants.
- Airports.
- Theaters etc.

Chapter 7

Conclusion

Chapter 7: Conclusion

7.1 Observation

- Program to drive the Motor Driver has been obtained.
- Stepper motor has been implemented.
- Program to drive the Servo has been obtained.
- The program for Keypad and IR Sensor Interface has been run successfully.
- The Keypad Interface has been implemented.
- The program for LCD Display Interface has been run successfully.
- The LCD Display Interface has been implemented.
- The program for LED Indicator Interface has been run successfully.
- The LED Indicator Interface has been implemented.

7.2 Future Scope

- The Smart Multi-storage car Parking System can be adopt so that availability of spaces could be displayed on a smart phone Application or even to satellite navigation device so that drivers will always aware of whether there are free spaces are not.
- And also enhance to send some notifications to users smart phone when vehicle enters to particular shopping places and some streets in a city etc.

References

1. *"How not to create traffic jams, pollution and urban sprawl"*. The Economist. Retrieved 2017-04-14.
2. *"asphaltnation.com"*. Retrieved 24 November 2016.
3. Preservation Online: Storey of the Week Archives: Car Culture.
4. Preservation Chicago.
5. *"The Guide to Parking Lots"*. Discount Park & Ride. 9 January 2015. Retrieved 27 March 2015.
6. Los Angeles Times, 25 June 1963, *"High Rise Developer Defends Loss of View to Convenience"*.
7. Los Angeles Times, 15 March 1964, Tom Cameron, *"$118 Million Going Into Expansion at L.A. State"*.
8. *Queally, James (14 January 2014). "'Where is it?' Panicked 911 calls from wife of lawyer killed in mall carjacking beg for ambulance"*. The Star-Ledger. Retrieved 15 January 2014.
9. *"North Carolina Parking Deck Collapse Kills 1"*. Fox News. 6 December 2007.
10. *"Man dies after Montreal parking garage roof collapses"*. CBC News. 26 November 2008.
11. *Castillo, Gregory (30 October 2012). "Four Dead, Several Injured After Parking Garage Collapses At West Campus"*. Miami-Dade College. Retrieved 26 August 2014.
12. *"Miami Dade College Parking Garage Collapsed Due To Unfinished Column: Lawyer"*. Huffington Post. 29 March 2013. Retrieved 26 August 2014.
13. *"High Precast Parking Systems" (PDF)*. High Concrete Group. Retrieved 3 June 2014.
14. *Beth Broome (June 2010). "1111 Lincoln Road"*. Architectural Record.
15. *Michael Barbaro (24 January 2011). "A Miami Beach Event Space. Parking Space, Too"*. The New York Times. p. A1.
16. *"Wisconsin English varieties"*. Wisconsin Englishes. Center for the Study of Upper Midwestern Cultures; Wisconsin Humanities Council. Retrieved 29 October 2016.
17. *"Buffalo Civic Auto Ramps"*. Buffalo Civic Auto Ramps. Retrieved 2 August 2017.
18. *"Judging the Parking Podiums of DTLA's Newest High-Rises - Los Angeles Magazine"*. Los Angeles Magazine. 2015-10-13. Retrieved 2017-04-14.
19. International Code Council. *International Building Code 2012 (Second Printing).*
20. Sanders McDonald, Shannon.*"Cars, Parking and Sustainability", The Transportation Research Forum http://www.trforum.org/*. Retrieved on 16 October 2012.
21. *Hamelink, Ir. Leon J. (2011), The Mechanical Parking Guide 2011, ISBN 1-466-43786-3.*
22. *"Underground Garage to End Parking Problem"*, February 1931, Popular Mechanics.
23. *"parking sensor indicator/parking space indicator/led parking lot lighting, View car parking led indicator, KEYOP Product Details from Xiamen Keytop Communication & Technology Company Limited on Alibaba.com"*. Retrieved 24 November 2016.
24. "Servant or snoop in the parking garage?" Los Angeles Times.
25. *"House of Cars: Innovation and the Parking Garage"*. Retrieved 24 November 2016.