VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE2006 - Microprocessor and Interfacing (J component)

Project Title: Vehicle Parking System using RFID and Arduino

Faculty: Naresh K

Slot : D1

Team Members

1. Arush Saxena - 20BCE2106

2. Hetarth Raval - 20BCE0826

3. Katewa Chintusingh Ranjeet - 20BCE0169

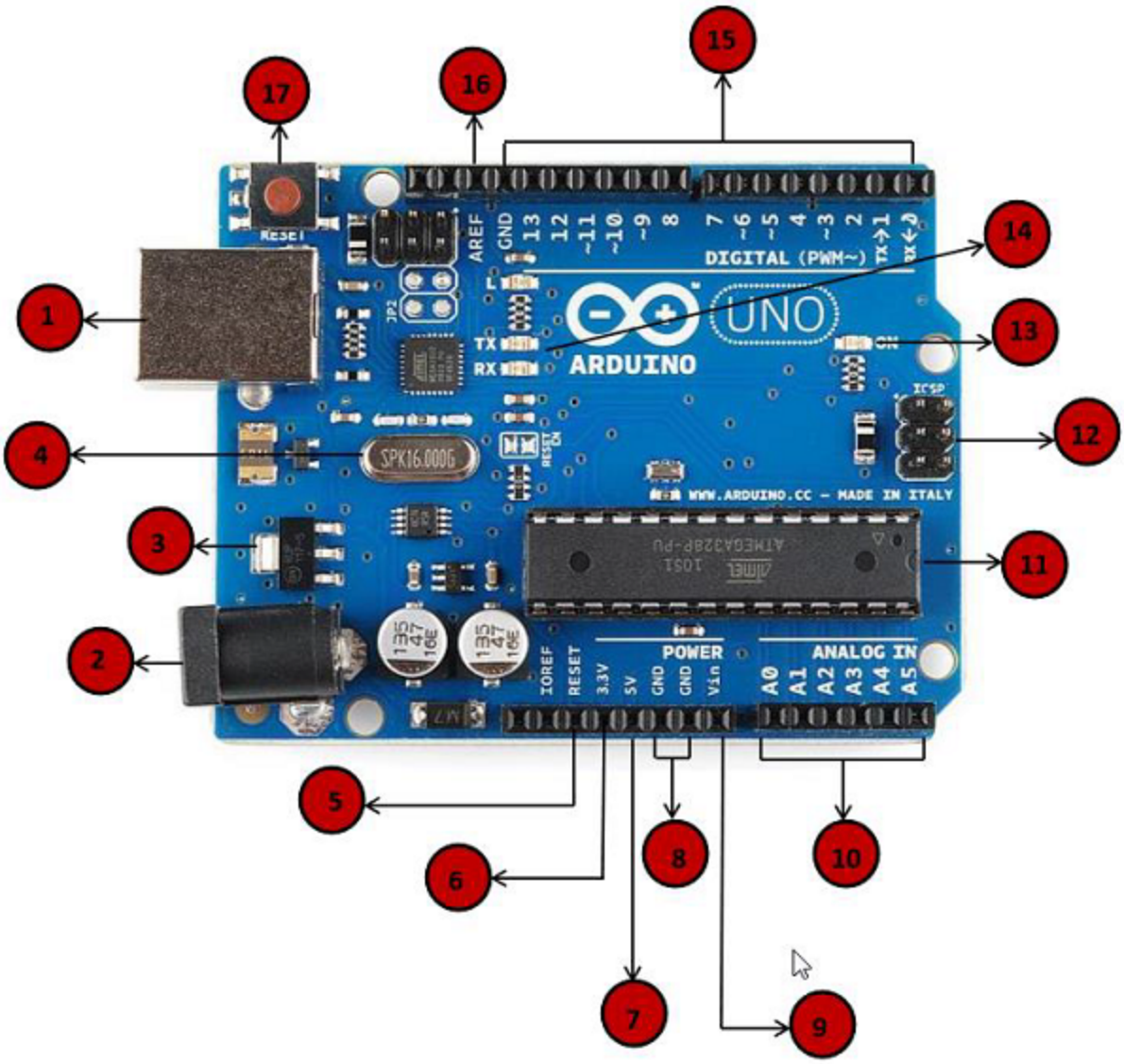
4. Sanskar Dagar - 20BCE0659

Arduino – Basics and Design

An Arduino board is a one type of microcontroller based kit. The first Arduino technology was developed in the year 2005 by David Cuartielles and Massimo Banzi. The designers thought to provide an easy and low cost board for students, hobbyists and professionals to build devices. Arduino boards can be purchased from the seller or directly we can make them at home using various basic components. The best examples of Arduino for beginners and hobbyists include motor detectors and thermostats, and simple robots. In the year 2011, Adafruit industries expected that over 3 lakhs Arduino boards had been produced. But, 7 lakhs boards were in user's hands in the year 2013. Arduino technology is used in many operating devices like communication or controlling.

Now that you know the origin of Arduino, it is essential to get yourself acquainted with the hardware that Arduino as a company offers. One of the main reasons for Arduino being so accessible and affordable across the globe is because all of the Arduino hardware is open source. Being open-source has a plethora of advantages- anyone can access the design and build of the device and make improvements; anyone can use the same hardware design to create their product lineup. Since Arduino is open-source, it has its own devoted community that strives to help the core company develop and improve its hardware products. Another significant advantage of being open source, especially in the case of hardware, is that local companies can create replicas of the products, making it more accessible and affordable to the local consumers as it avoids hefty customs and shipping charges. All of these advantages contribute to Arduino being so widespread, affordable and ever-improving.

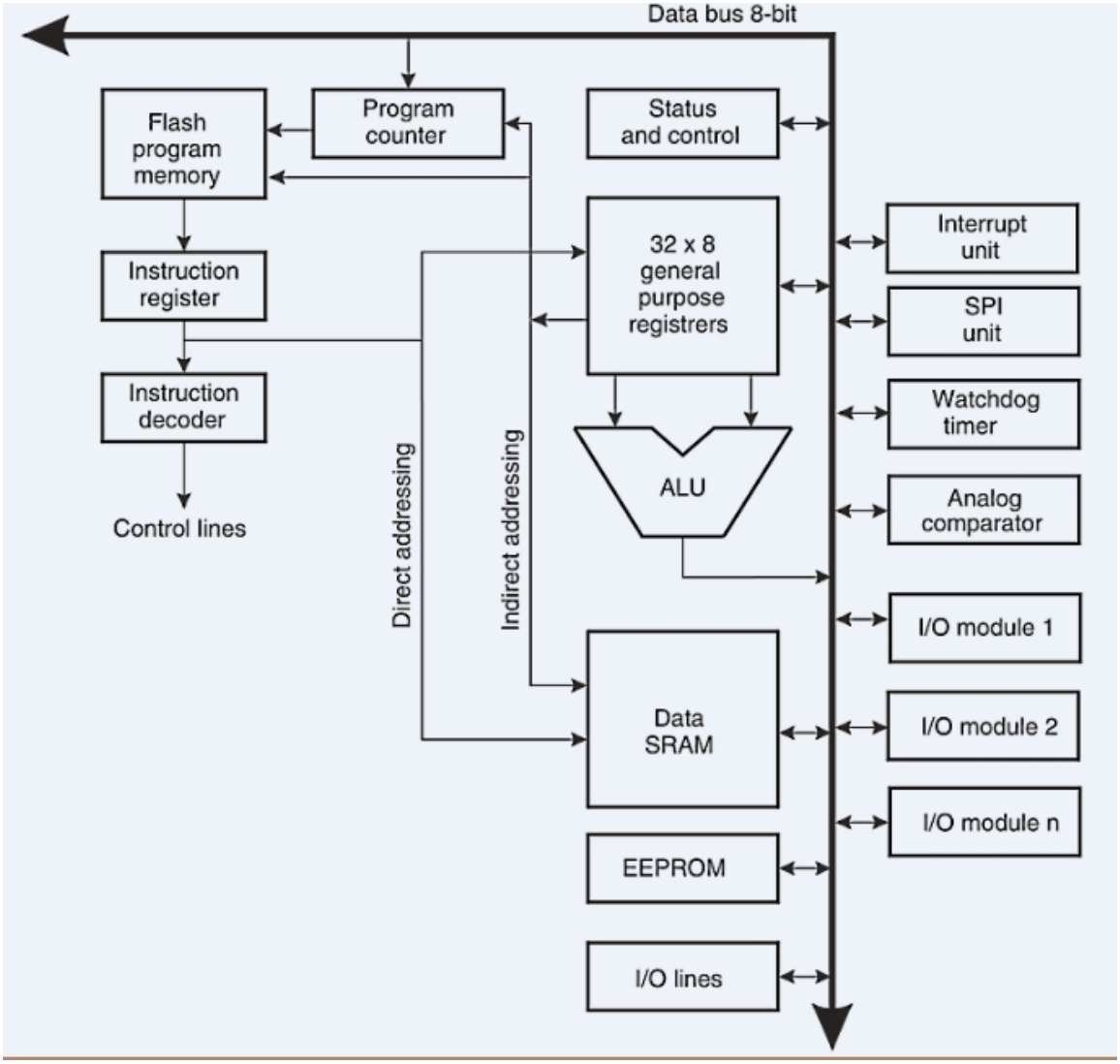
It is necessary to know that Arduino doesn't necessarily offer just one piece of hardware, it provides a range of boards, each of which caters to a different level of expertise and have different use-cases altogether. Arduino Uno is one of the most basic and popular boards that Arduino offers. This is because it features an ATMega328 microcontroller that is both cheap and powerful enough for most basic beginner-level projects. Once you're familiar with Arduino IDE, you can move up to boards with more powerful and sophisticated chipsets like the MKR range which is concerned with IoT applications and inter compatibility, or the Nano range which as the name suggests is designed to keep the form factor as small as possible while packing most of the features and power of the full-sized boards.



Using the above image as a reference, the labeled components of the board respectively are:

1) **USB:** can be used for both power and communication with the IDE
2) **Barrel Jack:** used for power supply
3) **Voltage Regulator:** regulates and stabilizes the input and output voltages
4) **Crystal Oscillator:** keeps track of time and regulates processor frequency
5) **Reset Pin:** can be used to reset the Arduino Uno
6) **3.3V pin:** can be used as a 3.3V output
7) **5V pin:** can be used as a 5V output
8) **GND pin:** can be used to ground the circuit
9) **Vin pin:** can be used to supply power to the board

10) **Analog pins(A0-A5):** can be used to read analog signals to the board
11) **Microcontroller(ATMega328):** the processing and logical unit of the board
12) **ICSP pin:** a programming header on the board also called SPI
13) **Power indicator LED:** indicates the power status of the board
14) **RX and TX LEDs:** receive(RX) and transmit(TX) LEDs, blink when sending or receiving serial data respectively
15) **Digital I/O pins:** 14 pins capable of reading and outputting digital signals; 6 of these pins are also capable of PWM
16) **AREF pins:** can be used to set an external reference voltage as the upper limit for the analog pins
17) **Reset button:** can be used to reset the board



Arduino Architecture:

Arduino's processor basically uses the Harvard architecture where the program code and program data have separate memory. It consists of two memories- Program memory and the data memory.The code is stored in the flash program memory, whereas the data is stored in the data memory. The Atmega328 has 32 KB of flash memory for storing code (of which 0.5 KB is used for the bootloader), 2 KB of SRAM and 1 KB of EEPROM and operates with a clock speed of 16MHz.

How to program an Arduino?

The most important advantage with Arduino is the programs can be directly loaded to the device without requiring any hardware programmer to burn the program. This is done because of the presence of the 0.5KB of Bootloader which allows the program to be burned into the circuit. All we have to do is to download the Arduino software and write the code.The Arduino tool window consists of the toolbar with the buttons like verify, upload, new, open, save, serial monitor. It also consists of a text editor to write the code, a message area which displays the feedback like showing the errors, the text console which displays the output and a series of menus like the File, Edit, Tools menu.

5 Steps to program an Arduino:

- Programs written in Arduino are known as sketches. A basic sketch consists of 3 parts
    1. Declaration of Variables
    2. Initialization: It is written in the setup () function.
    3. Control code: It is written in the loop () function.

- The sketch is saved with the .ino extension. Any operations like verifying, opening a sketch, saving a sketch can be done using the buttons on the toolbar or using the tool menu.
- The sketch should be stored in the sketchbook directory.
- Choose the proper board from the tools menu and the serial port numbers.

- Click on the upload button or choose upload from the tools menu. Thus the code is uploaded by the bootloader onto the microcontroller.

Few of basic Arduino functions are:

- **digitalRead**(pin): Reads the digital value at the given pin.
- **digitalWrite**(pin, value): Writes the digital value to the given pin.
- **pinMode**(pin, mode): Sets the pin to input or output mode.
- **analogRead**(pin): Reads and returns the value.
- **analogWrite**(pin, value): Writes the value to that pin.
- **serial.begin**(baud rate): Sets the beginning of serial communication by setting the bit rate.

How to Design your own Arduino?

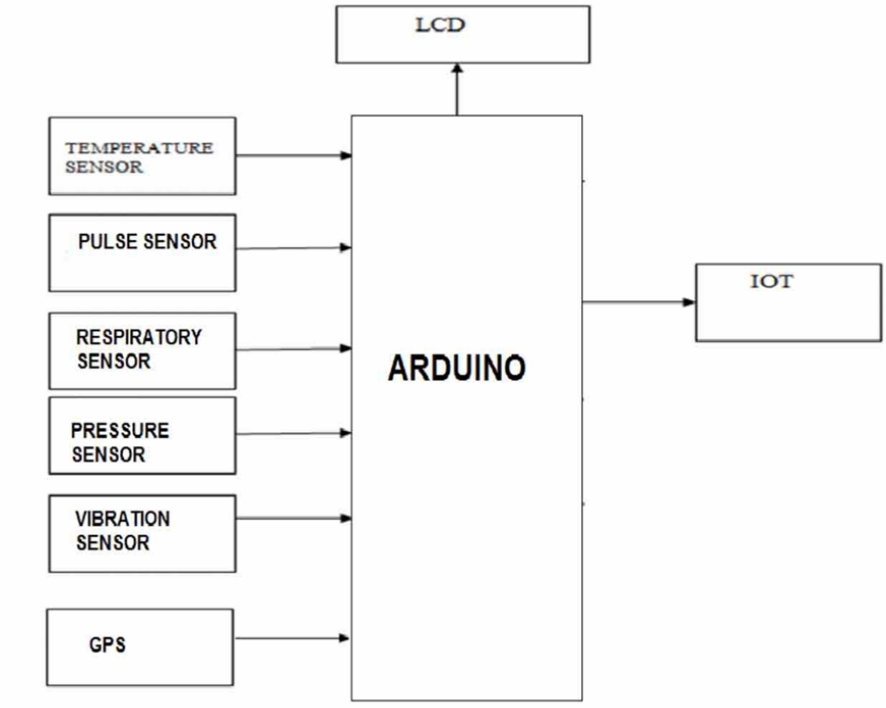
We can also design our own Arduino by following the schematic given by the Arduino vendor and also available at the websites. All we need are the following components- A breadboard, a led, a power jack, an IC socket, a microcontroller, a few resistors, 2 regulators, 2 capacitors.

- The IC socket and the power jack are mounted on the board.
- Add the 5v and 3.3v regulator circuits using the combinations of regulators and capacitors.
- Add proper power connections to the microcontroller pins.
- Connect the reset pin of the IC socket to a 10K resistor. • Connect the crystal oscillators to pins 9 and 10
- Connect the led to the appropriate pin.
- Mount the female headers onto the board and connect them to the respective pins on the chip.
- Mount the row of 6 male headers, which can be used as an alternative to upload programs.
- Upload the program on the Microcontroller of the readymade Adruino and then pry it off and place it back on the user kit.

7 Reasons why Arduino is being preferred these days

1. It is inexpensive
2. It comes with an open source hardware feature which enables users to develop their own kit using the already available one as a reference source.
3. The Arduino software is compatible with all types of operating systems like Windows, Linux, and Macintosh etc.
4. It also comes with an open source software feature which enables experienced software developers to use the Arduino code to merge with the existing programming language libraries and can be extended and modified.
5. It is easy to use for beginners.
6. We can develop an Arduino based project which can be completely stand alone or projects which involve direct communication with the software loaded in the computer.
7. It comes with an easy provision of connecting with the CPU of the computer using serial communication over USB as it contains built-in power and reset circuitry.

So this is the basic idea regarding an Arduino. You can use it for many types of applications. For instance in applications involving controlling some actuators like motors, generators, based on the input from sensors.



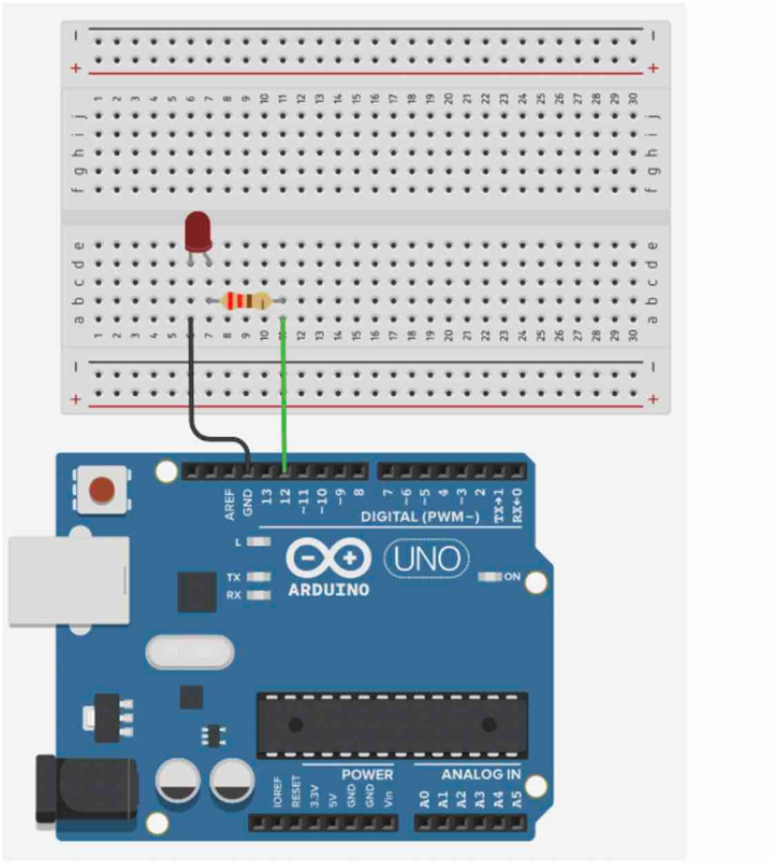
Arduino LED

First, you will setup your circuit with an Arduino board and an LED, and then discover different ways to control the LED.I will use Arduino Uno for the examples but the instructions here apply to any Arduino board.If you want to just try the LED code on your Arduino, without doing the circuit, well, good news! You can just use the internal LED on pin 13 if you want.

Creating the Arduino LED circuit

- Arduino board.
- LED (any color, I will use red).
- Breadboard.
- 220 Ohm resistor (more info on the value later on).
- Some male to female wires.

Building the circuit



How to build the circuit?

- First make sure that the Arduino is powered off (no USB cable plugged to anything).
- Check the LED, you will see that one of the leg is shorter than the other one.
- Plug the shorter leg of the LED to a hole on the breadboard. Connect that leg to a GND pin of the Arduino, using a black cable if possible (convention for GND).
- Plug the longer leg of the LED to a different hole, on a different and independent line of the breadboard.
- Add a 220 Ohm resistor between this longer leg and a digital pin of the Arduino, using an additional colored wire (no red, no black) for convenience.
  Here we choose digital pin 12 on the Arduino Uno.

Arduino code to power on an LED

Power on the LED with digitalWrite()

Let's start with the most simple thing. Let's say you just want to power on the LED when the Arduino program starts. This code will help us understand the setup and how digital pins work.

```
1.  #define LED_PIN 12
2.
3.  void setup()
4.  {
5.    pinMode(LED_PIN, OUTPUT);
6.    digitalWrite(LED_PIN, HIGH);
7.  }
8.
9.  void loop() {}
```

This code is very short and will just power on the LED on pin 12. Let's analyze it.

#define LED_PIN 12

First, and this is a best practice, we create a definition for the pin number. This way, anytime we need to use this digital pin, we can write LED_PIN instead of the hard-coded number. In the future, if you ever need to use a different digital pin (for example 11), then you just need to change the number here and it will update it everywhere in your program.

```
3.  void setup()
4.  {
5.    pinMode(LED_PIN, OUTPUT);
```

We enter the void setup() function, which will be executed once at the beginning of the program.

Before we can actually use a digital pin, we need to set a mode. Basically you have 2 modes: output (if you want to control a component), or input (if you want to read some information from a component). Here, we want to control the LED, so we choose output. For that, we use the pinMode() function with 2 parameters:

- Pin number: here we use the LED_PIN that we previously defined.
- Mode: for output mode, we have to use OUTPUT (all uppercase).

After the execution of this line, the digital pin 12 will be set as output, and we can control the LED.

```
6.    digitalWrite(LED_PIN, HIGH);
7.  }
```

Now, to control the LED, it's very simple. We need to use the digitalWrite() function with 2 parameters:

- Pin number: again, we use the defined LED_PIN.
- State: we have 2 choices here. HIGH to power on the LED, and LOW to power it off.

So, after this line, the LED will be powered on.

Note: it's important to call pinMode() before digitalWrite(), otherwise the LED won't be powered on because the mode is not set.

void loop() {}

After the void setup() function is finished, we enter the void loop() and this function will be executed again and again, until you power off the Arduino.

Here we didn't write anything in the void loop() because we just want to power on the LED and that's it.

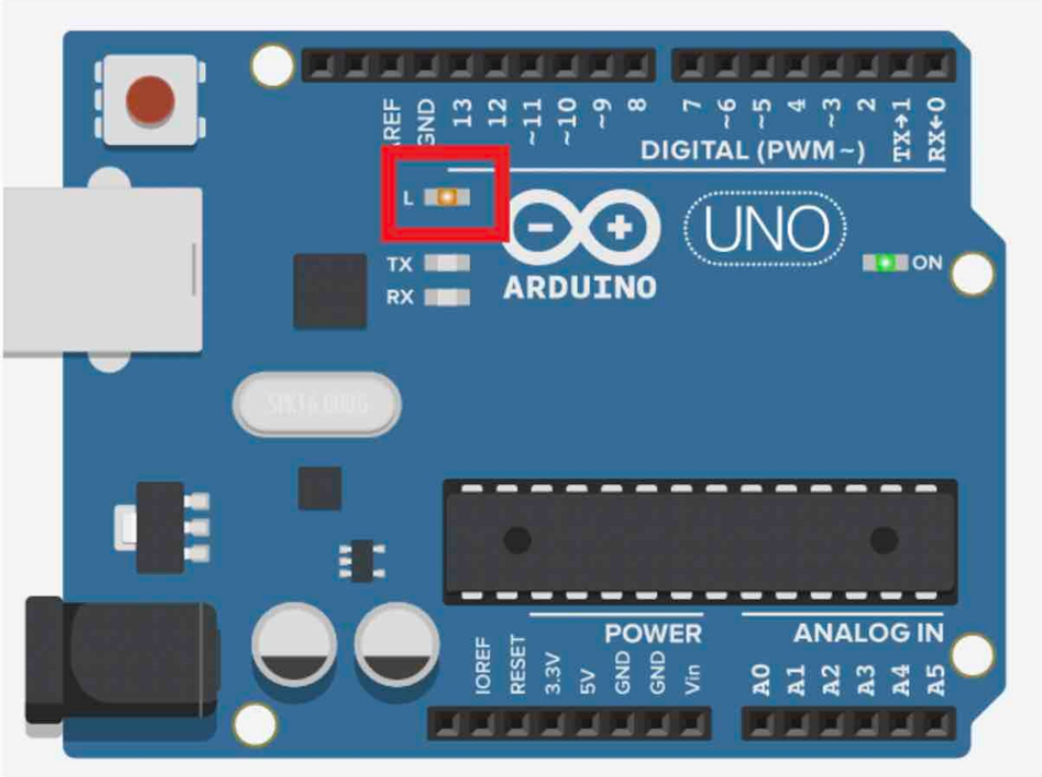
Test using only the built-in Arduino LED

If you don't want to build the circuit, or want to test with something even simpler, you can use the built-in LED on the Arduino, which is soldered on digital pin 13.

You can use LED_BUILTIN which is already predefined for this LED.

```
1.  void setup()
2.  {
3.    pinMode(LED_BUILTIN, OUTPUT);
4.    digitalWrite(LED_BUILTIN, HIGH);
5.  }
6.
7.  void loop() {}
```

And you'll see the built-in LED powered on. Note: the location of the LED can vary depending on the type of your Arduino board.



Make the blink LED example

Now that you have the circuit and the code to set up the LED, let's do something a bit more interesting.

Let's make the LED blink, which means that we are going to:

1. Power on the LED,
2. wait,
3. Power off the LED,
4. wait,
5. Go back to 1.

Here's the code to do that (using our LED connected to digital pin 12).

```
#define LED_PIN 12

void setup()
{
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_PIN, LOW);
  delay(500);
}
```

And in the void loop(), we handle the blink mechanism. First we power on the LED with digitalWrite() and HIGH, then we wait using delay(). The delay() function will block the program for a given amount of time (in milliseconds). So, here, we block the program for 500 milliseconds, or 0.5 second.After that, we power off the LED with digitalWrite() and LOW, and wait again for 0.5 seconds using delay(). This way, we have a complete cycle of "LED powered on" + "LED powered off".

Once we exit the void loop(), it is called again, and thus the LED is powered on again, etc etc.This blink example is maybe one of the most common examples, but also super useful when you get started with Arduino. If you correctly understand the circuit and the code, you've already made big progress.

Make the LED fade in/fade out

Let's try something more interesting with PWM. Here we want to make the LED fade in (which means the brightness will slowly increase until the max), and then fade out (brightness will slowly decrease), so we can create a nice effect with the LED.

```
1.  #define LED_PIN 11
2.
3.  void setup()
4.  {
5.    pinMode(LED_PIN, OUTPUT);
6.  }
7.
8.  void loop()
9.  {
10.   for (int i = 0; i <= 255; i++) {
11.     analogWrite(LED_PIN, i);
12.     delay(10);
13.   }
14.
15.   for (int i = 255; i >= 0; i--) {
16.     analogWrite(LED_PIN, i);
17.     delay(10);
18.   }
19. }
```

PWM functionality with Arduino LED

As for now, you've seen that you can control an LED with 2 states: HIGH (fully on with 5V), or LOW (fully off with 0V). What if you want to control the brightness of the LED, for example to 40% of the max brightnessThat's where the PWM functionality is useful.

Very basically put, a PWM will allow you to output a voltage which is a percentage of the max voltage. So, if you want 40% brightness, you'd have to provide 40% of the voltage, which is 40% * 5V = 2V. Even if that sounds complicated, you will see that it's super easy to do in the code.First, we update the defined line with the pin number 11 instead of 12. As you can see this is very practical, as we don't have to change the number anywhere else in the code.

Then, instead of using digitalWrite(), we use analogWrite() with 2 parameters:

- Digital pin, here LED_PIN as previously defined.
- Percentage of the voltage, scaled to 0-255 (one byte of data). If we want 40% we need to do 255 * 0.4 = 102. This will correspond to an output of 5V * 0.4 = 2V.

Let's write the code to power on the LED, but with just 40% of the max voltage.

```
1.  #define LED_PIN 11
2.
3.  void setup()
4.  {
5.    pinMode(LED_PIN, OUTPUT);
6.    analogWrite(LED_PIN, 102);
7.  }
8.
9.  void loop() {}
```

Arduino code for 3 LEDs

Let's make a very simple application: we want first the red LED to be powered on, then the yellow one, then the green one, and back to the red one.

```
#define LED_PIN_1 11
#define LED_PIN_2 10
#define LED_PIN_3 9

void setup()
{
  pinMode(LED_PIN_1, OUTPUT);
  pinMode(LED_PIN_2, OUTPUT);
  pinMode(LED_PIN_3, OUTPUT);
}

void loop()
{
  digitalWrite(LED_PIN_1, HIGH);
  digitalWrite(LED_PIN_2, LOW);
  digitalWrite(LED_PIN_3, LOW);
  delay(1000);
  digitalWrite(LED_PIN_1, LOW);
  digitalWrite(LED_PIN_2, HIGH);
  digitalWrite(LED_PIN_3, LOW);
  delay(1000);
  digitalWrite(LED_PIN_1, LOW);
  digitalWrite(LED_PIN_2, LOW);
  digitalWrite(LED_PIN_3, HIGH);
  delay(1000);
}
```

Nothing really complicated: for the setup we just duplicate the code for each new LED. We add a define and we set up the mode with pinMode().

Then, well you can use digitalWrite() on each LED – and analogWrite() if the pin is PWM compatible – whenever you want. In this case we just power on each LED, once at a time, with a 1 second delay.

Vehicle Parking System using RFID and Arduino

Abstract

The Vehicle Parking System is designed to automate the process of allocating parking slots to cars. Our proposed system allots parking to vehicles entering the parking lot. It aims to provide easy and quick parking. The driver can accurately see the available space in the parking lot. If there is available space, the barrier gets lifted and the vehicle may pass through. This is achieved by using RFID, Arduino UNO and Servo Motor. When the vehicle leaves the parking lot, the allotted space is deallocated, and an empty space is created for a new vehicle to enter.

Introduction

Parking allocation can be very difficult especially in huge parking lots with a lot of vehicles continuously requesting parking. It's difficult for the parking staff to communicate with each other on different sublevels of parking lots. They might have to shout and run to the parking entrance to inform others about the availability or unavailability of space in the parking lot.

Our proposed Automatic Parking System aims to make it easier for the staff to allot and manage space in the parking lot. The Vehicle Parking System is an automatic, adaptable, user-friendly and technologically advanced method for reserving parking spaces for the car being driven. Additionally, the operator can easily maintain vehicle tracking utilizing the parking spot and leaving it. The system reads the unique id number from the vehicle and allots a spot to that vehicle. The spot can be used by the vehicle until it is checked out by scanning again and finally exiting the parking lot.

Background

Allotting parking space can be a very laborious job especially when there's many cars waiting at the entrance to enter the car park. This project aims to help the people responsible for allotting and managing parking space by providing them

with automatic allocation, deallocation of parking slots and displaying the number of available slots to make the work faster.

Motivation

Nearly everyone in the modern world today owns a personal automobile, which has evolved into a basic requirement for the populace. Thus, it has been established that annual growth in automobile utilization is rapid. Parking spaces are increasingly hard to come by in cities because of the increase, especially during rush hour. The Parking System's primary goal is to make it easier and faster for drivers to locate locations with parking spaces.

Implementation

In our project we have implemented an Automatic Vehicle Parking System. The system automatically detects whether the parking slot is empty or not. If the slot is empty in the automated car parking the new vehicles are allowed to enter and the yellow LED glows else the entrance is blocked by the barrier operated by servo motors in case the parking is full. In case of no available slots or unauthorized access the red LED glows and buzzer makes a sound.

The system consists of Arduino UNO, Servo Motor and RFID. When a driver wishes to enter the parking area, the card with an id unique to each vehicle should be scanned by the RFID.

The parking area has a fixed number of parking slots. That means vehicles can register themselves through the RFID and the Servo Motor will get activated and will lift the barrier. The vehicle can then enter the parking area until there are free parking spaces available.

When all the slots are full, no more vehicles will be allowed to enter the area and the system displays a count of zero. If a driver wishes to leave the parking lot, the vehicle must be scanned again. This time, the system will detect that the vehicle was already allotted a space and will deallocate the space and will let the vehicle pass through the exit.

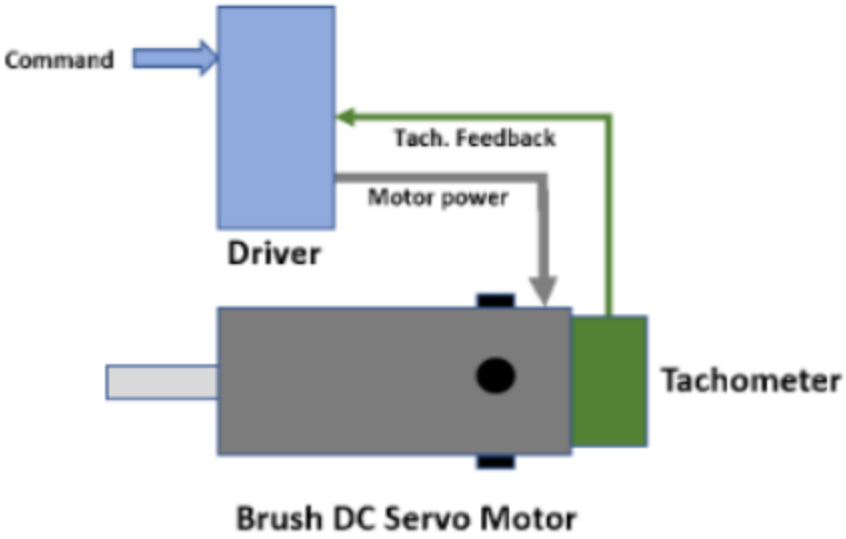
RFID Reader Working:

An antenna on the reader transmits a signal to the tag in the RFID system. Along with the data stored in its memory, the tag gets this information and delivers it again. This signal is received by the reader and sent to the processor for additional processing. A 125 KHz carrier signal is sent to the tag coil as the RFID card (tag) is swiped against the RFID reader; the tag coil receives this signal and modulates it. The reader, which is connected to the microcontroller, receives this modulated signal. When this data is received, the microcontroller is programmed to compare it to the information in the current database. If the data matches, the microcontroller-interfaced LCD will display the pertinent information about that specific vehicle.

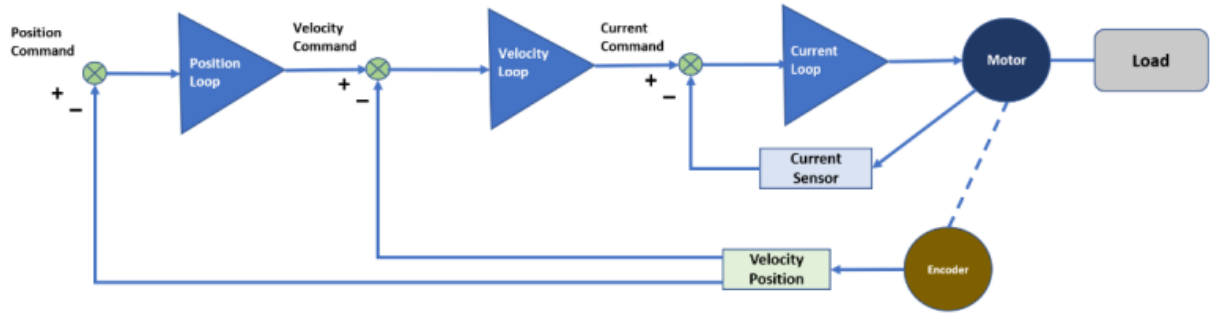
Servo Motor Working:

A servo motor is an electromechanical device that produces torque and velocity based on the supplied current and voltage. A servo motor works as part of a closed loop system providing torque and velocity as commanded from a servo controller utilizing a feedback device to close the loop. The feedback device supplies information such as current, velocity, or position to the servo controller, which adjusts the motor action depending on the commanded parameters.

A simple industrial servo motor consists of a permanent magnet DC motor with an integral tachometer that provides an output voltage proportional to speed. The drive electronics deliver the necessary voltage and current to the motor based on the voltage fed back from the tachometer. In this example, a commanded speed (represented as a command reference voltage) is set in the driver, then the circuitry in the driver compares the tachometer feedback voltage and determines if the desired speed has been accomplished - known as a closed velocity loop. The velocity loop is monitoring the commanded velocity and tachometer feedback, while the driver adjusts the power to the motor to maintain the desired commanded velocity.



Multiple embedded loops are tuned for optimal performance to provide precision motion control. The system consists of current, velocity, and position loops that utilize precision feedback elements. Each loop signals the subsequent loop and monitors the appropriate feedback elements to make real time corrections to match the commanded parameters.



The base loop is the current or torque loop. Current is proportional to torque in a rotary motor (or force in a linear motor), which provides acceleration or thrust. A current sensor is the device that provides feedback related to the current flowing through the motor. The sensor sends a signal back to the control electronics - typically an analog or digital signal proportional to the motor current. This signal is subtracted from the commanded signal. When the servo motor is at the commanded current, the loop will be satisfied until the current drops below the commanded current. The loop will then increase current until the commanded current is reached, with the cycle continuing at sub second update rates.

The velocity loop works in the same fashion with voltage proportional to velocity. The velocity loop sends the current loop a command to increase current (thus increasing voltage) when the velocity falls below the commanded velocity.

The position loop accepts a command for a PLC or motion controller, which in turn provides a velocity command that is fed to the velocity loop, which in turn commands the required current to accelerate, maintain, and decelerate the motor to move to the commanded position. All three loops work in optimized synchrony to provide smooth and precise control of the servo mechanism.

Buzzer Working:
An arduino buzzer, also called a piezo buzzer, is basically a tiny speaker that can be connected directly to an Arduino. It can be made to sound a tone at the set frequency. The buzzer produces sound based on the reverse of the piezoelectric effect.

Piezoelectric effect - Piezoelectricity is an effect where certain crystals change shape when electricity is applied to them. By applying an electric signal at the right frequency, the crystal can make sound.

The buzzer produces the same noisy sound irrespective of the voltage variation applied to it. It consists of piezo crystals between two conductors. When a potential is applied across these crystals, they push on one conductor and pull on the other. This, push and pull action, results in a sound wave. Most buzzers produce sound in the range of 2 to 4 kHz.

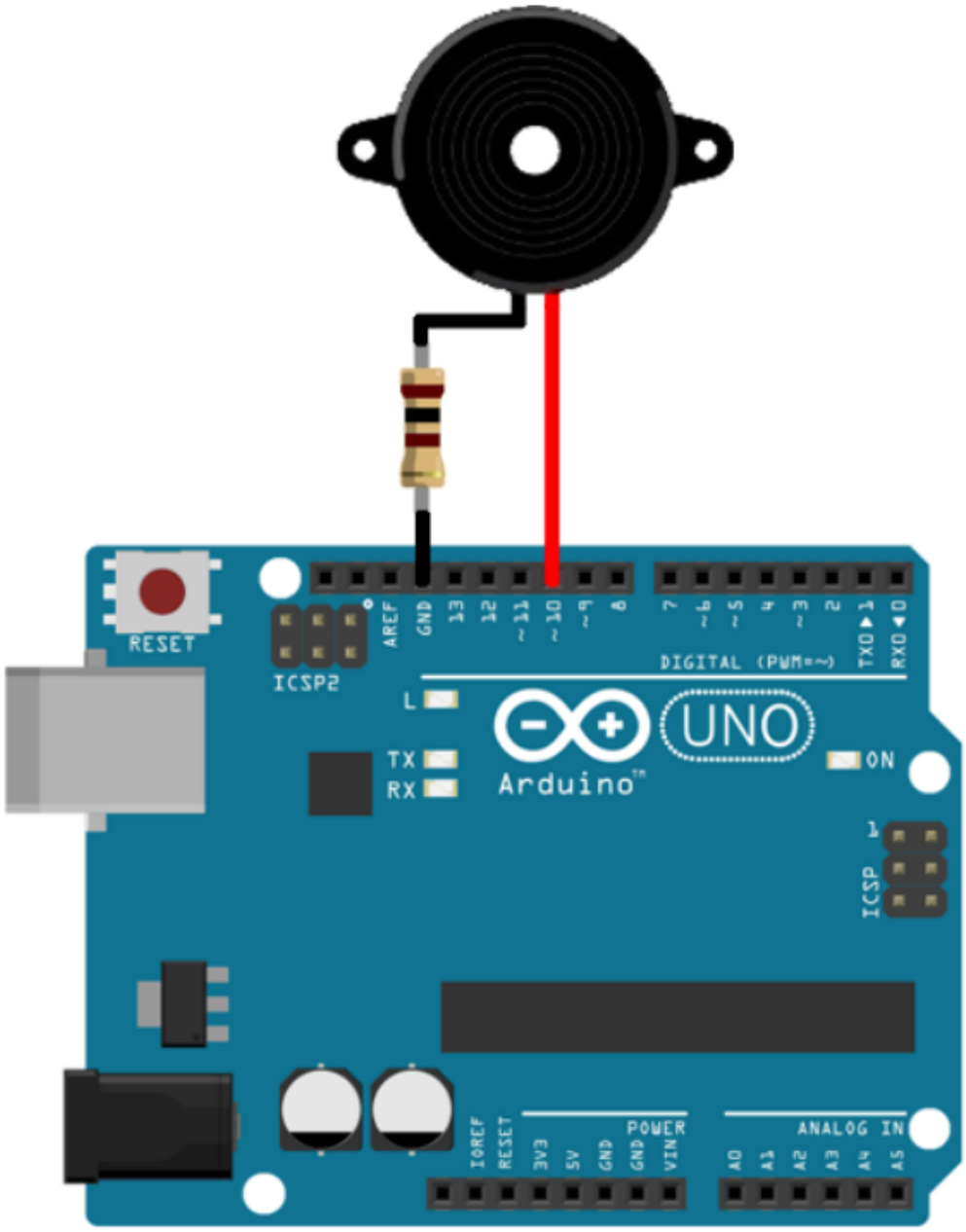
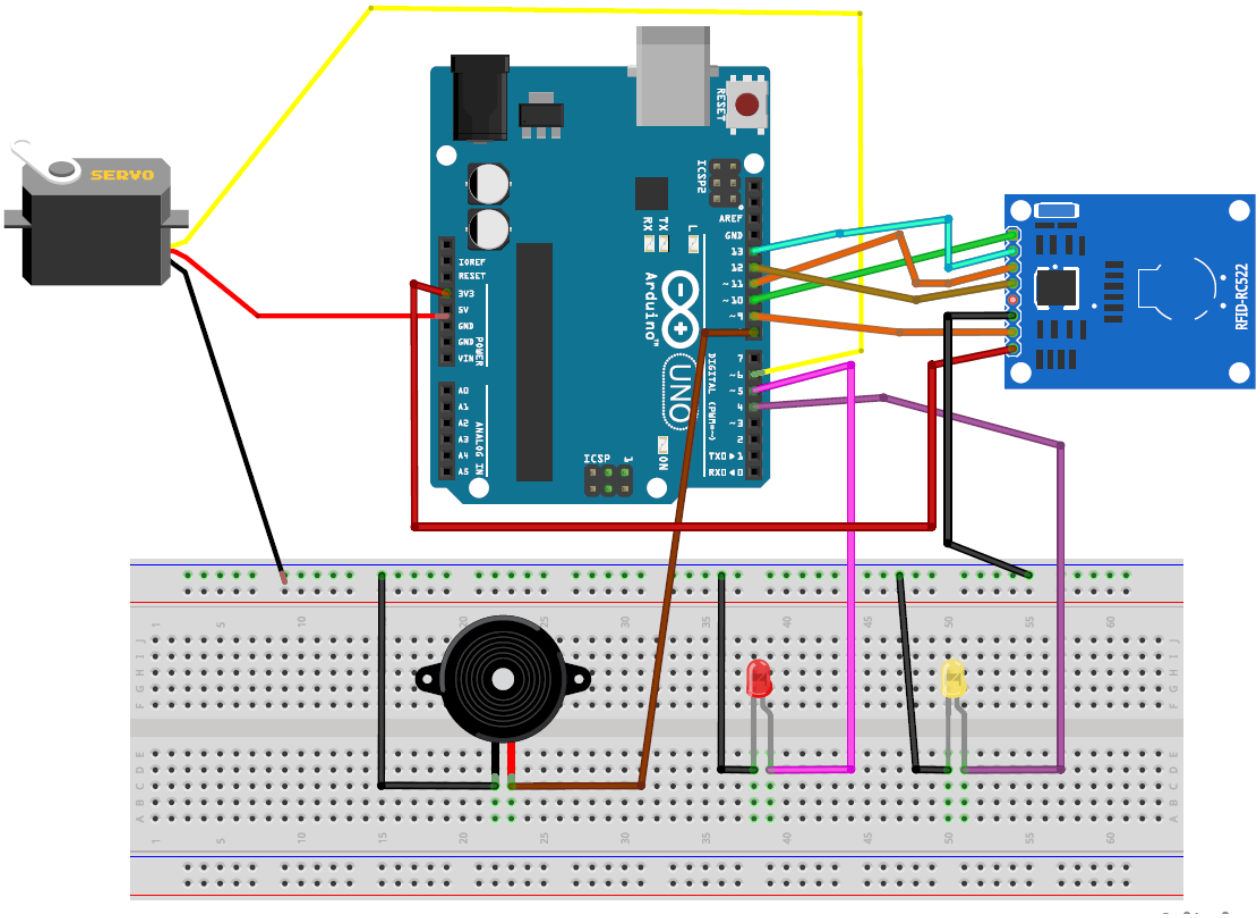


Figure 1 - Arduino buzzer wiring diagram

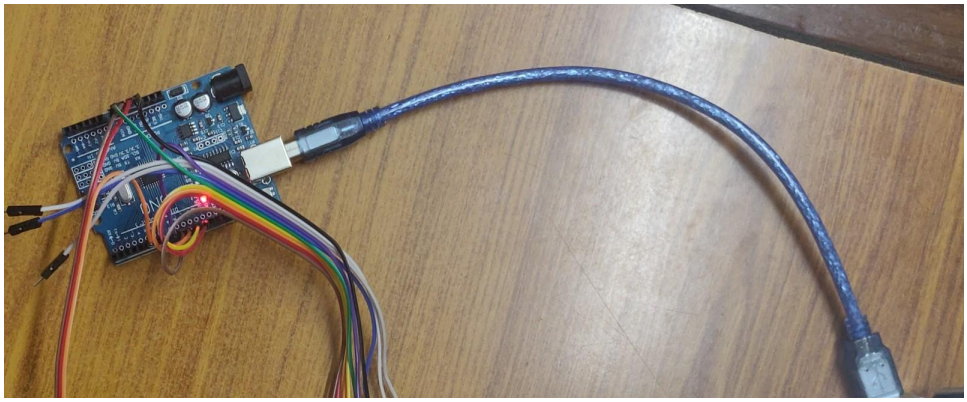
In context of this project, when any unauthorized person tries to access the parking system the access to park car inside will be denied and the buzzer will produce a beep sound or if slots are filled even though any authorized person tries to access it, the buzzer will produce a beep sound and a message will be displayed that slots are filled.

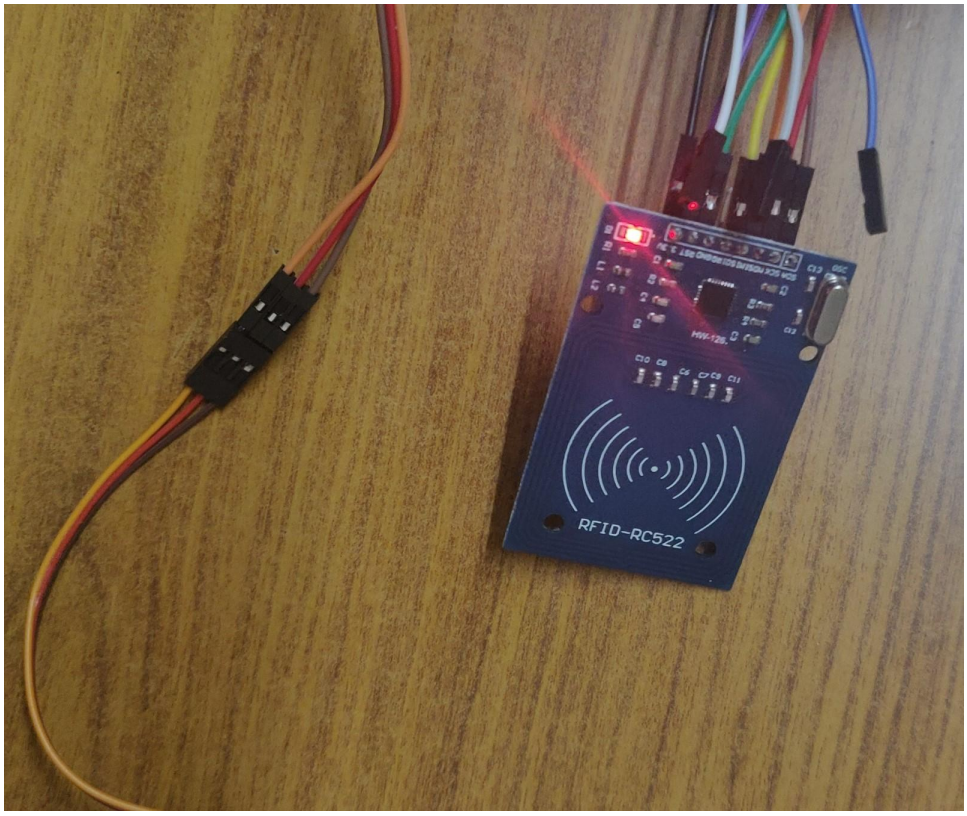
Circuit Diagram

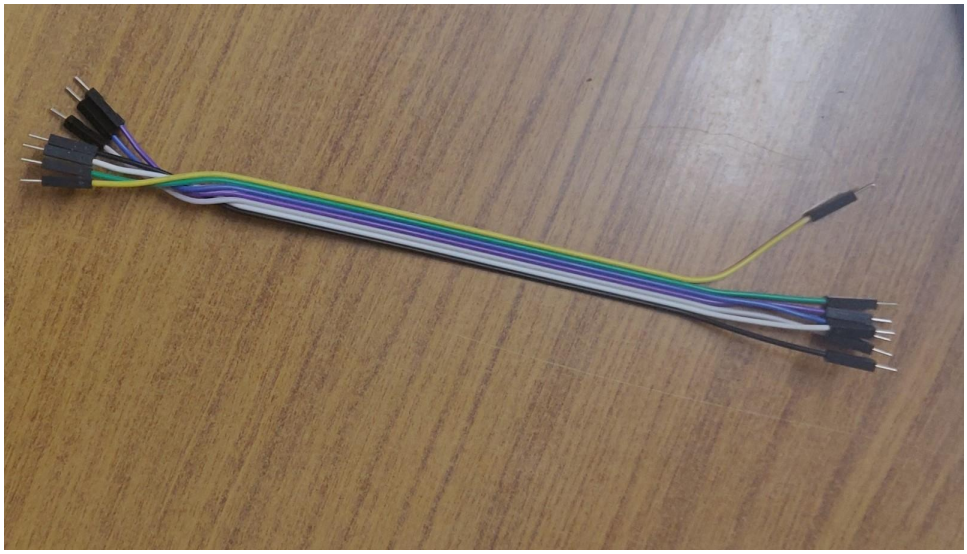


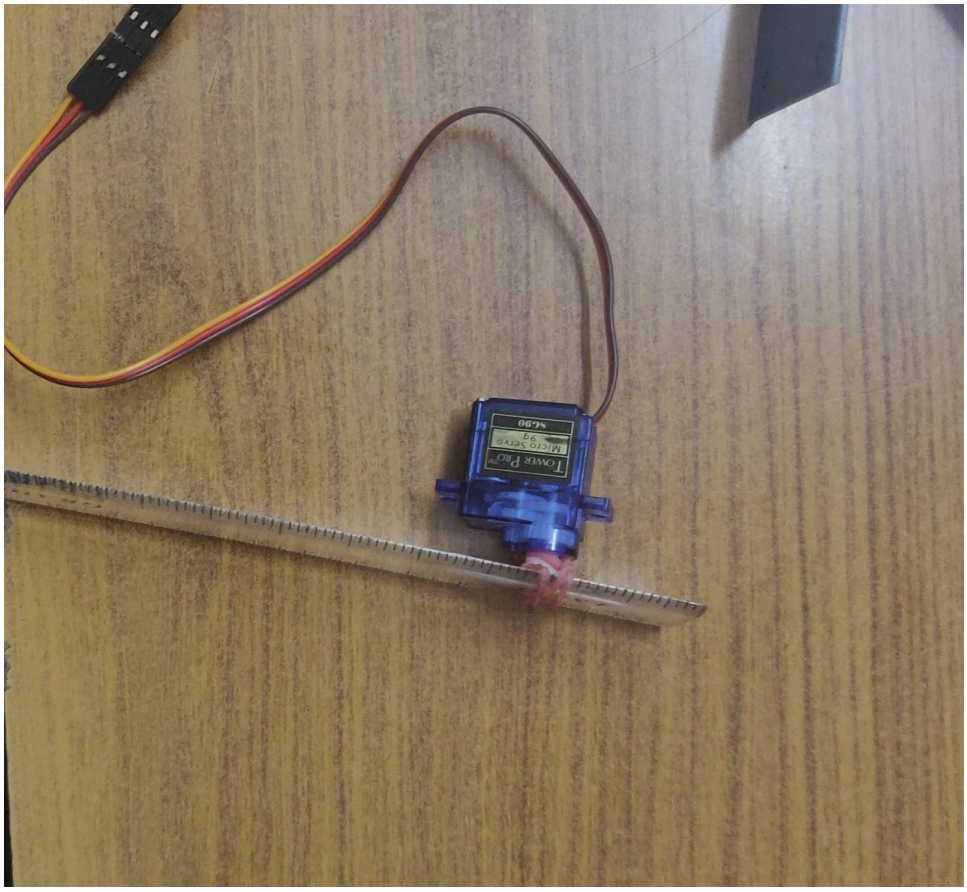
Components used

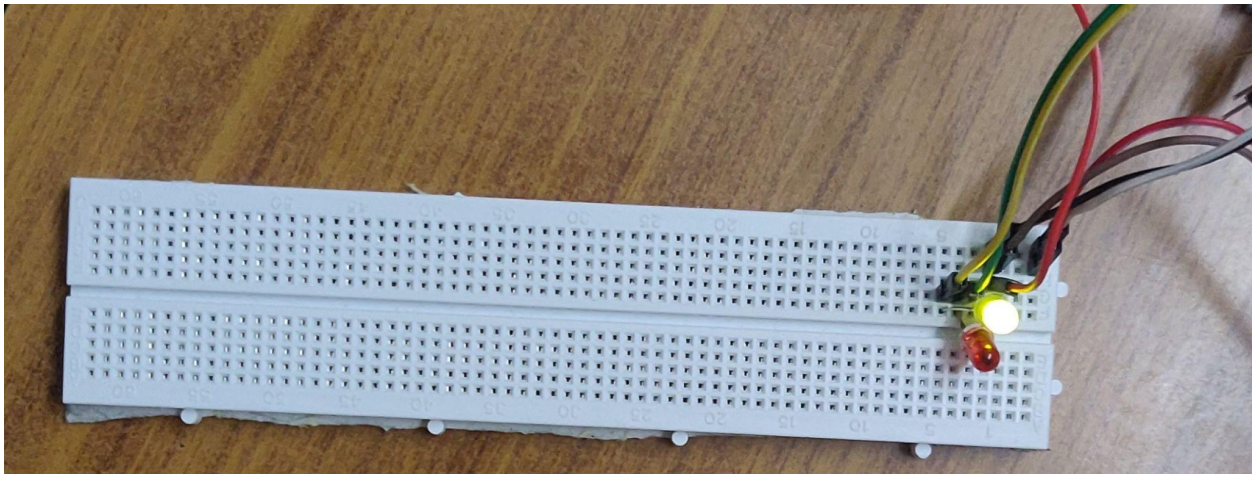
S No	Component Name	Quantity
1.	Arduino Uno	1
2.	RFID	1
3.	Servo Motor	1
4.	Buzzer	1
5.	LED	2
6.	Jumper Wire	1
7.	Breadboard	1

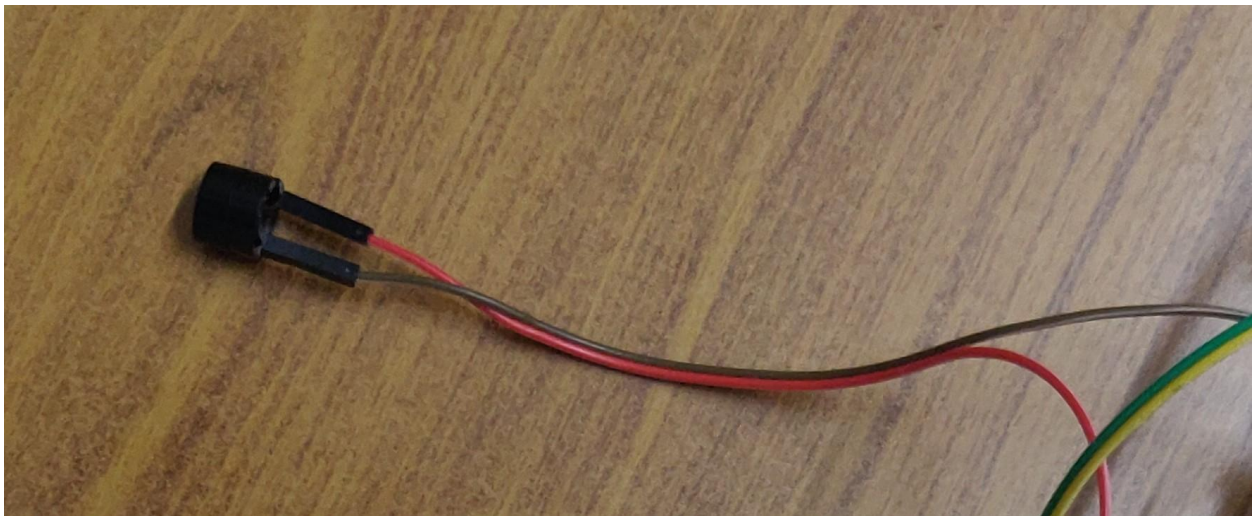



RFID-RC522




TOWER PRO
MICRO SERVO
9G
SG90





Code

Unauthozation code- red highlight
Authorization code- green highlight

```
#include "SPI.h"
#include "MFRC522.h"
#include <Servo.h>

#define SS_PIN 10
#define RST_PIN 9


Servo myservo;

int slot = 1;
String flag;
String check;

int yellow_led = 4;
int red_led = 5;
int mic = 8;

MFRC522 rfid(SS_PIN, RST_PIN);

MFRC522::MIFARE_Key key;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(8, OUTPUT);
  myservo.attach(6);
  myservo.write(0);
```

```
  Serial.println("Scan your card...");
}
void loop() {
  // put your main code here, to run repeatedly:
  if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial())
    return;

  // Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
  // Serial.println(rfid.PICC_GetTypeName(piccType));

  // Check is the PICC of Classic MIFARE type
  if (piccType != MFRC522::PICC_TYPE_MIFARE_MINI &&
      piccType != MFRC522::PICC_TYPE_MIFARE_1K &&
      piccType != MFRC522::PICC_TYPE_MIFARE_4K) {
    Serial.println(F("Your tag is not of type MIFARE Classic."));
    return;
  }
  String strID = "";
  for (byte i = 0; i < 4; i++) {
    strID +=
      (rfid.uid.uidByte[i] < 0x10 ? "0" : "") +
      String(rfid.uid.uidByte[i], HEX) +
      (i != 3 ? ":" : "");
  }
  strID.toUpperCase();
  delay(500);

   if ((strID.indexOf("C3:10:17:0C") >= 0 || strID.indexOf("C1:E7:37:19") >= 0)
&& slot >= 0) {
    if(flag=="C3:10:17:0C" || check=="C1:E7:37:19"){
      if(strID.indexOf("C3:10:17:0C") >= 0 && flag=="C3:10:17:0C"){
      myservo.write(70);
      slot++;
      Serial.println("Bye Vehicle 1,you can go out");
      Serial.print("Slots left: ");
      Serial.println(slot);
      flag="";
```

```
delay(2000);
myservo.write(0);
}
else if(strID.indexOf("C1:E7:37:19") >= 0 && check=="C1:E7:37:19"){

  myservo.write(70);
  slot++;
  Serial.println("Bye Vehicle 2,you can go out");
  Serial.print("Slots left: ");
  Serial.println(slot);
  check="";
  delay(2000);
  myservo.write(0);
}
 else if(slot <=0){
   Serial.println("Slots are over. You can't come in new vehicle");
  myservo.write(0);
  digitalWrite(8, HIGH);
  delay(2000);
  digitalWrite(8, LOW);
}

}
else if(slot <=0){
  Serial.println("Slots are over");
  myservo.write(0);
 digitalWrite(8, HIGH);
 delay(2000);
 digitalWrite(8, LOW);
}
else{
    if(strID.indexOf("C3:10:17:0C") >= 0 && slot > 0){
   Serial.println("Authorised access");
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(8, LOW);
  myservo.write(70); //motor moves 70 degree
  flag = strID;
```

```
      slot--;
      Serial.println("Welcome Vehicle 1, you can come in");
      Serial.print("Slots left: ");
      Serial.println(slot);
      delay(2000);
        }
        else if(strID.indexOf("C1:E7:37:19") >= 0 && slot > 0){
      Serial.println("Authorised access");
      digitalWrite(4, HIGH);
      digitalWrite(5, LOW);
      digitalWrite(8, LOW);
      myservo.write(70); //motor moves 70 degree
      check = strID;
      slot--;
      Serial.println("Welcome Vehicle 1, you can come in");
      Serial.print("Slots left: ");
      Serial.println(slot);
      delay(2000);
       }
      }
}
else
{
  if(strID.indexOf("13:BB:28:13") >= 0){
      Serial.println("Unauthorized access");
      Serial.println("Access denied for Vehicle 2");
      digitalWrite(5, HIGH);
      digitalWrite(4, LOW);
      myservo.write(0);
      digitalWrite(8, HIGH);
      delay(2000);
      digitalWrite(8, LOW);
  }
  else{

      Serial.println("Sorry slots are over");
      Serial.println("You can't come in");
      delay(2000);
```

```
    }

  }

}
```

Code Screenshots

rfid | Arduino 1.8.19

File Edit Sketch Tools Help

rfid

```
}
String strID = "";
for (byte i = 0; i < 4; i++) {
  strID +=
    (rfid.uid.uidByte[i] < 0x10 ? "0" : "") +
    String(rfid.uid.uidByte[i], HEX) +
    (i != 3 ? ":" : "");
}
strID.toUpperCase();
delay(500);

if ((strID.indexOf("C3:10:17:0C") >= 0 || strID.indexOf("C1:E7:37:19") >= 0) && slot >= 0) {
  if(flag=="C3:10:17:0C" || check=="C1:E7:37:19"){
    if(strID.indexOf("C3:10:17:0C") >= 0 && flag=="C3:10:17:0C"){
    myservo.write(70);
    slot++;
    Serial.println("Bye Vehicle 1,you can go out");
    Serial.print("Slots left: ");
    Serial.println(slot);
    flag="";
    delay(2000);
    myservo.write(0);
    }
    else if(strID.indexOf("C1:E7:37:19") >= 0 && check=="C1:E7:37:19"){
```

rfid | Arduino 1.8.19

File Edit Sketch Tools Help

rfid

```
    myservo.write(70);
    slot++;
    Serial.println("Bye Vehicle 2,you can go out");
    Serial.print("Slots left: ");
    Serial.println(slot);
    check="";
    delay(2000);
    myservo.write(0);
  }
  else if(slot <=0){
    Serial.println("Slots are over. You can't come in new vehicle");
    myservo.write(0);
    digitalWrite(8, HIGH);
    delay(2000);
    digitalWrite(8, LOW);
  }

  }
  else if(slot <=0){
    Serial.println("Slots are over");
    myservo.write(0);
   digitalWrite(8, HIGH);
   delay(2000);
   digitalWrite(8, LOW);
  }
```

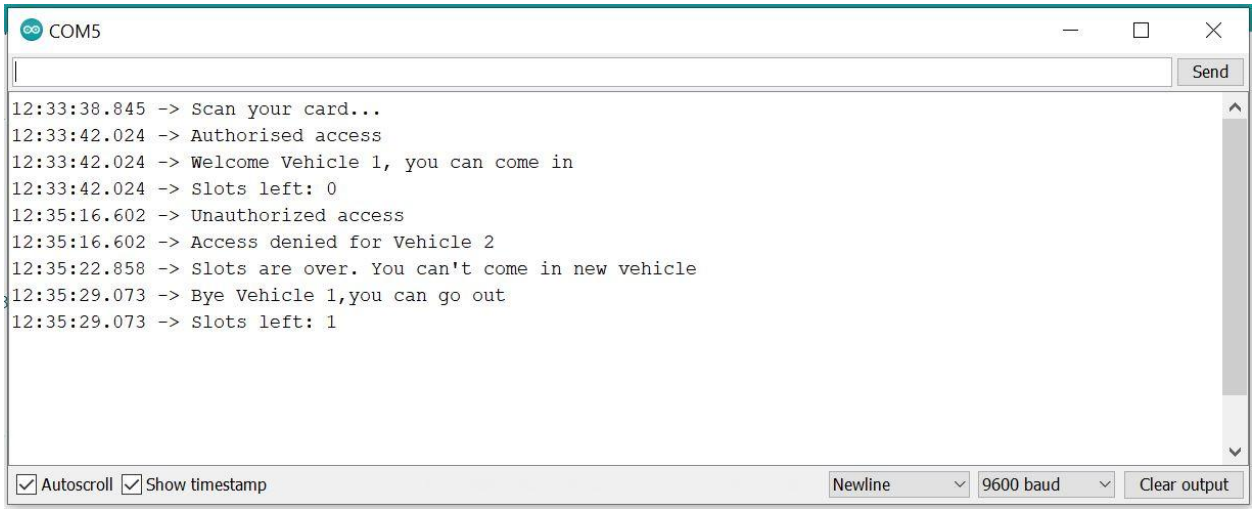
rfid | Arduino 1.8.19

File Edit Sketch Tools Help

rfid

```
  else{
     if(strID.indexOf("C3:10:17:0C") >= 0 && slot > 0){
    Serial.println("Authorised access");
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(8, LOW);
    myservo.write(70); //motor moves 70 degree
    flag = strID;
    slot--;
    Serial.println("Welcome Vehicle 1, you can come in");
    Serial.print("Slots left: ");
    Serial.println(slot);
    delay(2000);
      }
      else if(strID.indexOf("C1:E7:37:19") >= 0 && slot > 0){
    Serial.println("Authorised access");
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(8, LOW);
    myservo.write(70); //motor moves 70 degree
    check = strID;
    slot--;
    Serial.println("Welcome Vehicle 1, you can come in");
    Serial.print("Slots left: ");
    Serial.println(slot);
    delay(2000);
```

Output



Conclusion

The Automatic Vehicle Parking System aims to provide a faster and efficient way to allot and manage the available space in parking lots to make work easier for the staff working in the car park. The entry and exit of the car can be tracked very easily.