Manual Técnico

Proyecto 1
Arquitectura de Computadores y Ensambladores 1

Grupo 10

Integrantes

Carnet	Nombre
202003381	Luisa María Ortíz Romero
202000560	Marjorie Gissell Reyes Franco
202001534	Erwin Fernando Vásquez Peñate
201612332	Carlos Emilio Campos Morán
202004812	Fredy Samuel Quijada Ceballos
201504464	Herberth Abisai Avila Ruiz
202010751	Paulo Vladimir Argueta Ortega

Tabla de Contenido

Arduino Sensory Parking

Realizar una aplicación en Android con la cual se pueda observar los espacios disponibles y los espacios ocupados de un parqueo, la lectura de la información de la disponibilidad/ocupación de los parqueos se realizará por medio del uso de Arduino simulado el circuito eléctrico en proteus.

La detección para saber si un espacio esta libre u ocupado se realizara por medio del uso de un sensor dentro de su simulación, a estos sensores los acompañara una luz led que servirá de ayuda visual a los conductores que estarán viendo de forma física si el espacio ya está ocupado/disponible/reservado. El estacionamiento que diseñaran con su circuito simulado deberá ser como mínimo de 2 niveles y contar con un mínimo de 16 espacios para parquear vehículos.
En el circuito deberán mostrar en una pantalla LCD el total de parqueos con los que se cuentan, dando el detalle de la cantidad de espacios disponibles, espacios ocupados y reservados.
Como parte de una simulación de barrera de acceso se deberán usar motores stepper, el cual simule que la barrera se levanta para dar acceso a los vehículos y vuelve a bajar después de que el vehículo termino de pasar por completo, se contará con dos barreras, una para la entrada al parqueo y otra para la salida, tome en cuenta que estas barreras a partir del estado de cierre, deberán moverse únicamente 90 grados en contra de las manecillas del reloj y regresar a 0 grados cuando están cerradas nuevamente. La apertura de la barrera tanto para entrar y salir la realizaran los usuarios desde la aplicación del telefono, para esta función de apertura de la barrera deberá tomar en cuenta que los usuarios la pueden utilizar únicamente por medio de comunicación Bluetooth, por lo cual deberán ser capaces de simular un bluetooth y comunicar su aplicación con él.

Requerimientos

- Proteus 8.10 SP0 (Build 29203) (Para Simulacion)
- Arduino IDE
  - Versión: 1.8.15.0
- AppInventor2
- Visual Studio Code
  - Versión: 1.74.0
- Flask
- Microsoft Windows 10

Componentes

- 3 Arduino Mega 2560
- 3 COMPIM (COM Port Physical Interface Model)
- 1 Bluetooth HC-05
- 1 Pantalla LCD 16x2
- 2 Motores Bistepper
- 1 Speaker
- 32 TORCH_LDR (Torch and Light Dependent Resistor)
- 96 Luces LED
- 128 Resistencias de 10k
- 64 Compuertas NOT
- 64 Compuertas AND

Librerias

Es necesario el uso de las librerias LedControl.h (Luces led), panamahitek.Arduino (Comunicación).

```
// ARDUINO
#include <LiquidCrystal.h>
```

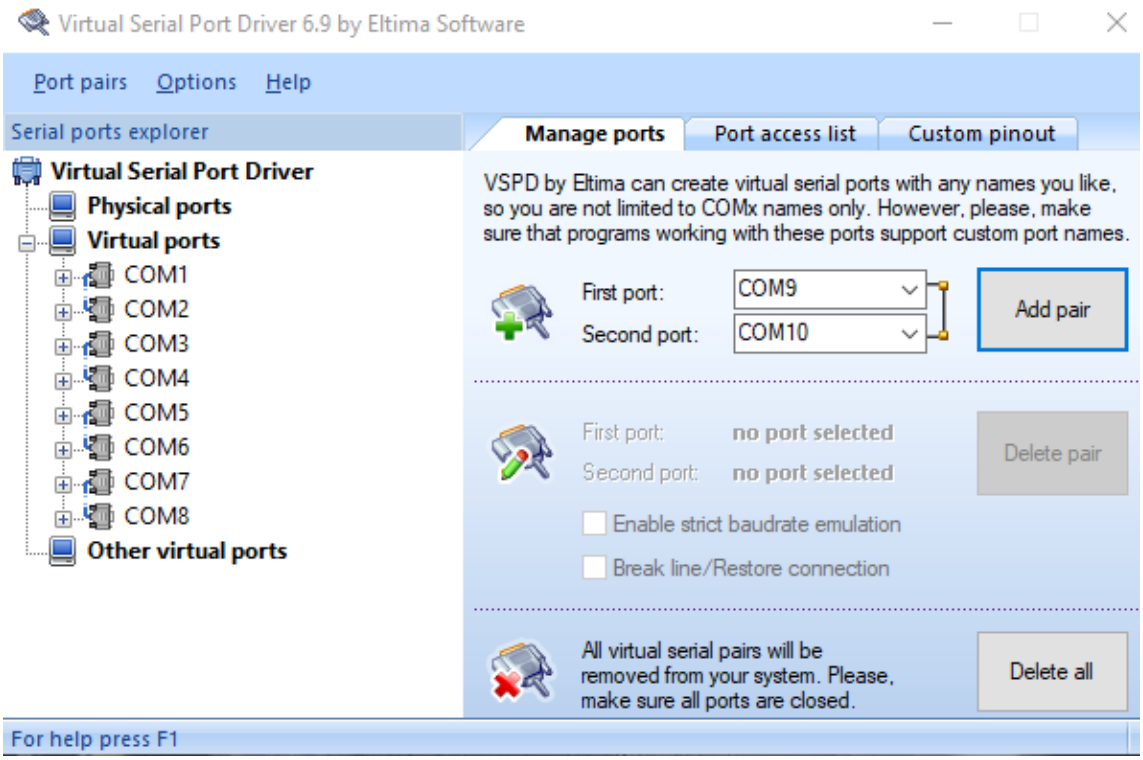
```
// PYTHON APLICACION
from flask import Flask
from flask_cors import CORS
from flask.globals import request
from instances import position_handler

// PYTHON SERVICIO - COMUNICACION ARDUINO
import serial
import time
import requests
import json
import threading
```

Estructura del funcionamiento

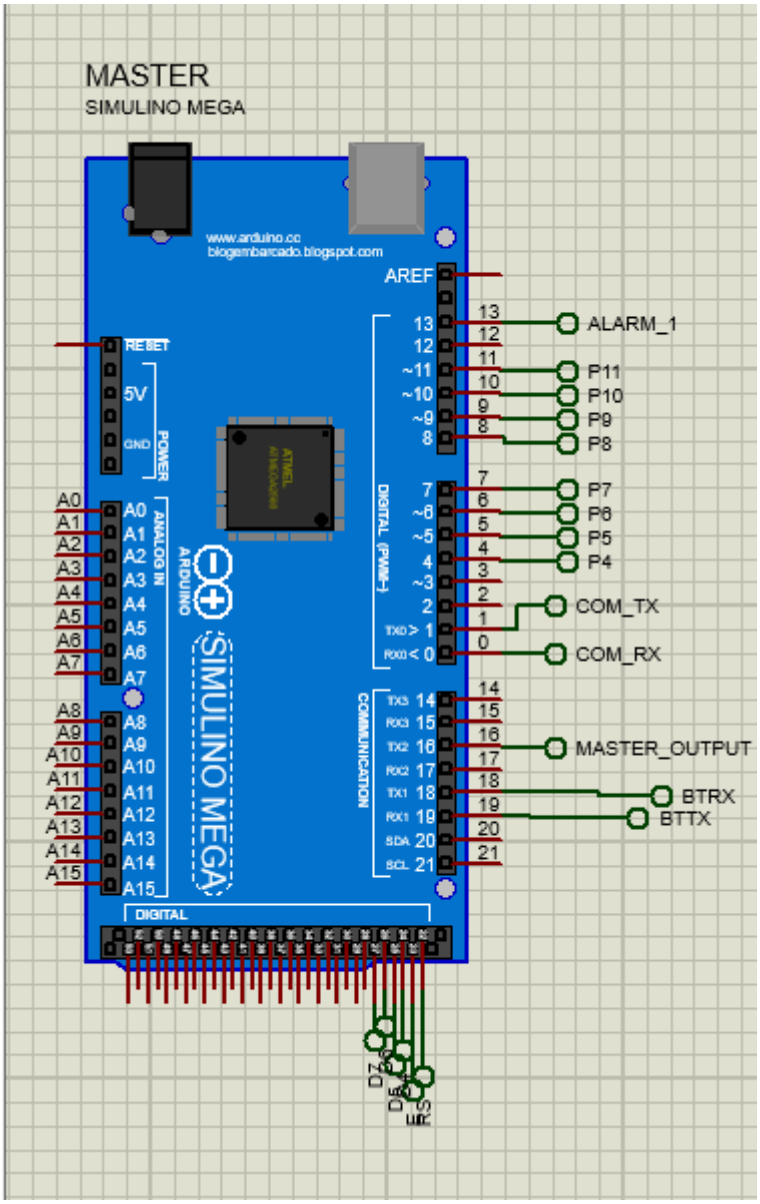
Virtual Serial Port Driver

Puertos virtuales para la comunicacion entre los servicios en Python y el Arduino en Proteus. Tambien entre Arduinos.
Se crean ocho puertos COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8

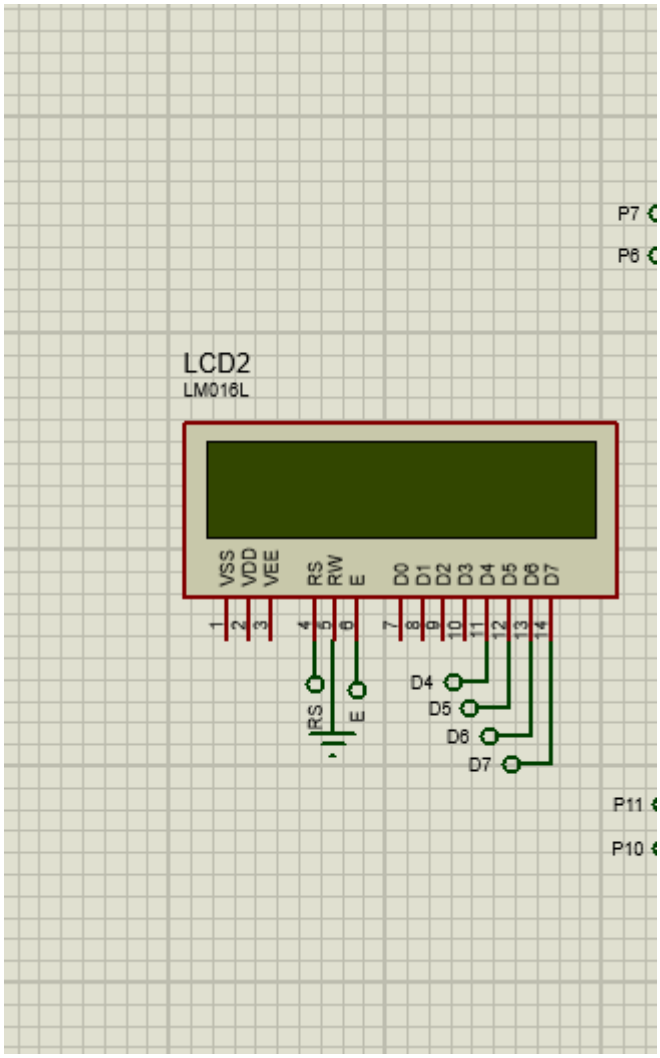


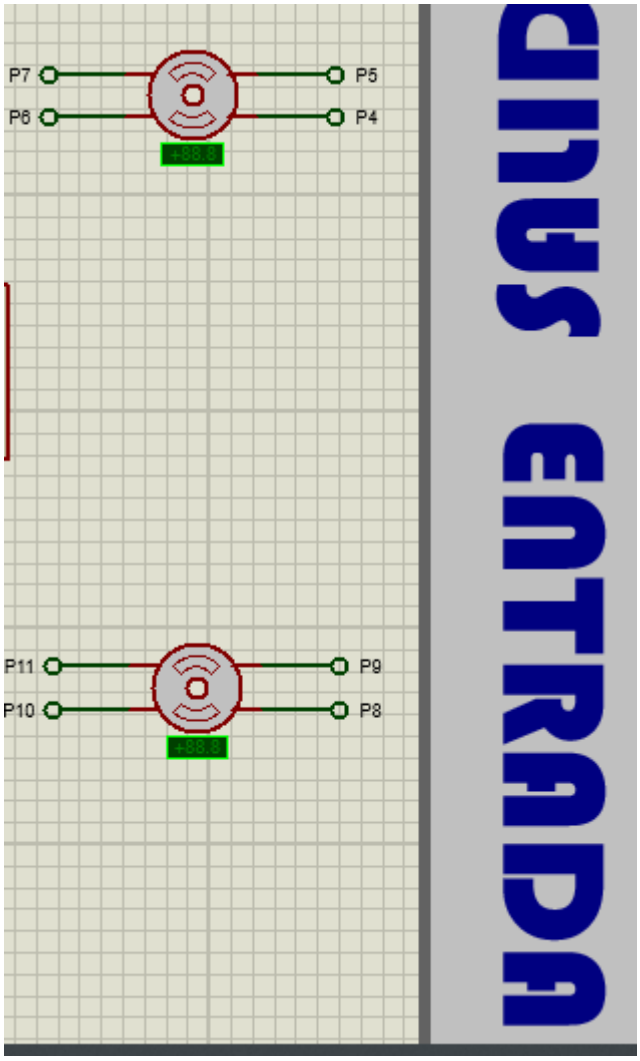
Conexiones Proteus

Arduino Mega MAESTRO

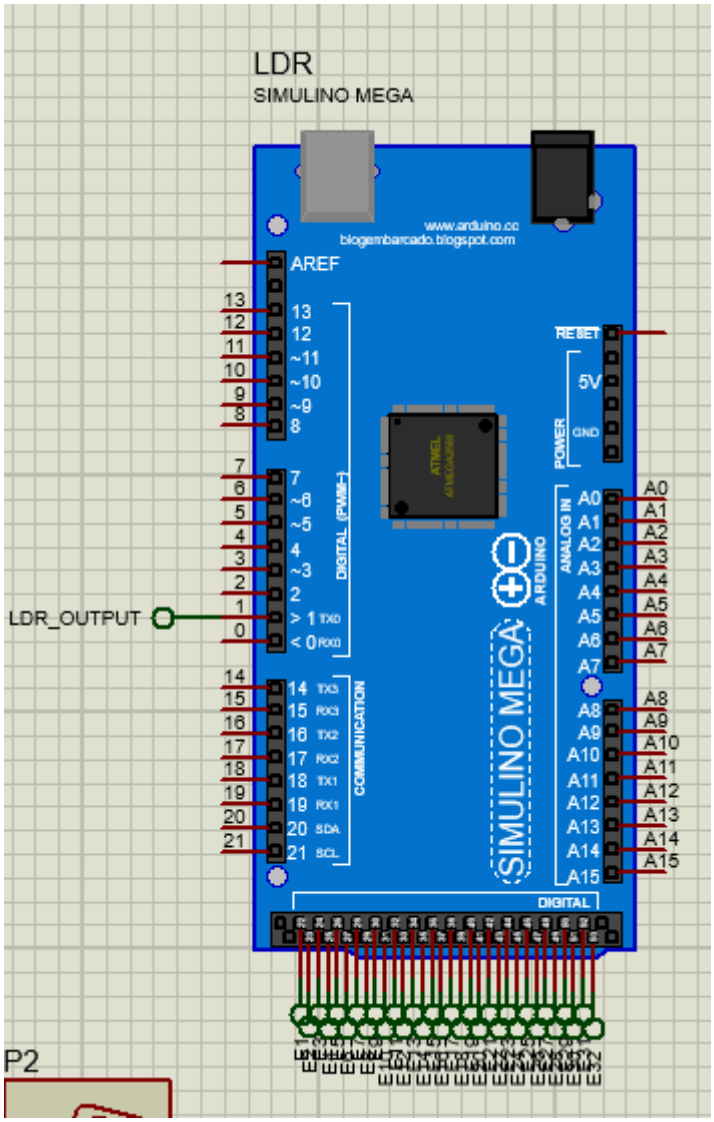


- Los pines digitales 4-11 son utilizados para el control de los motore stepper (salida, entrada) [OUTPUT].
- El pin 13 se utiliza para notificar a la alarma en caso de robo [INPUT].
- Los pines digitales 22-27 son utilizados para el control de la LCD del letrero [OUTPUT].
- Los pines 0,1,16,18,19 son utilizados para la transmision y recepcion de datos.


P7
P6
LCD2
LM016L
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
1
2
3
4
5
6
7
8
9
10
11
12
13
14
RS
E
D4
D5
D6
D7
P11
P10

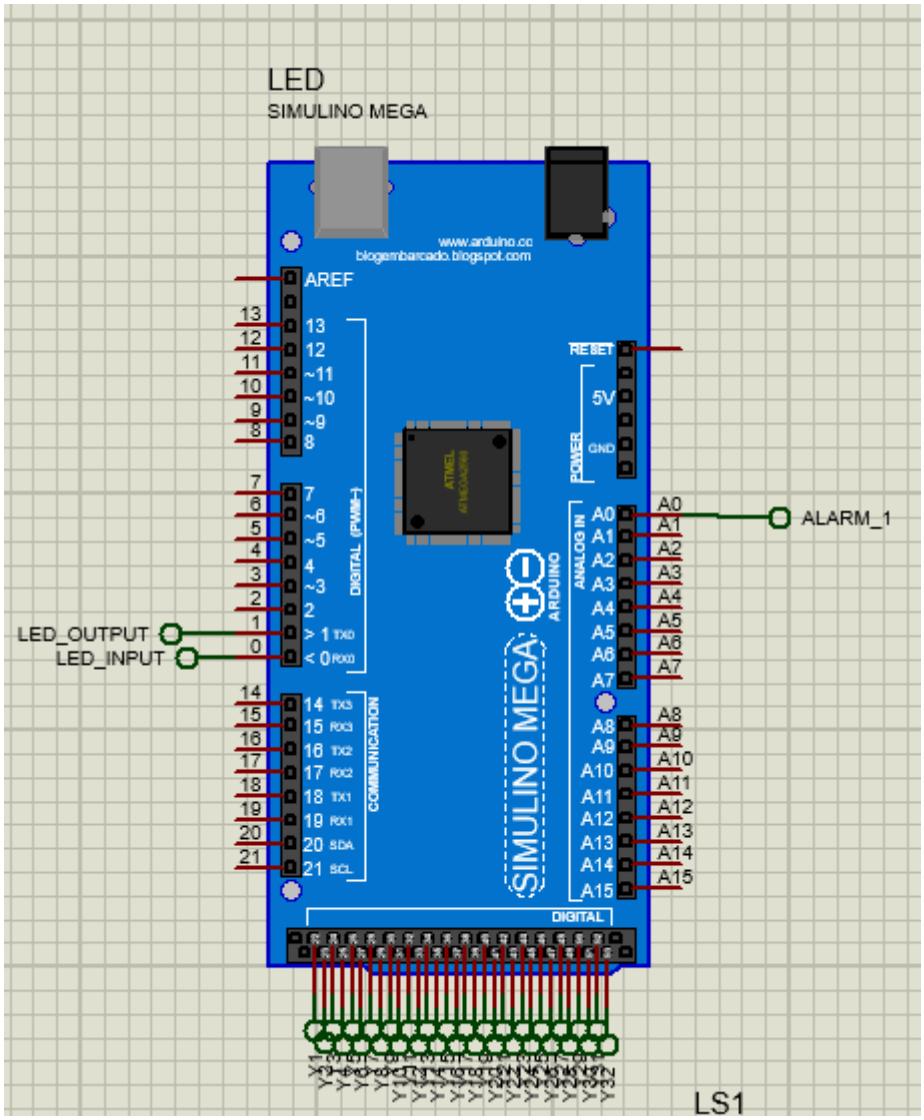


Arduino Mega Esclavo 1 (LDR)



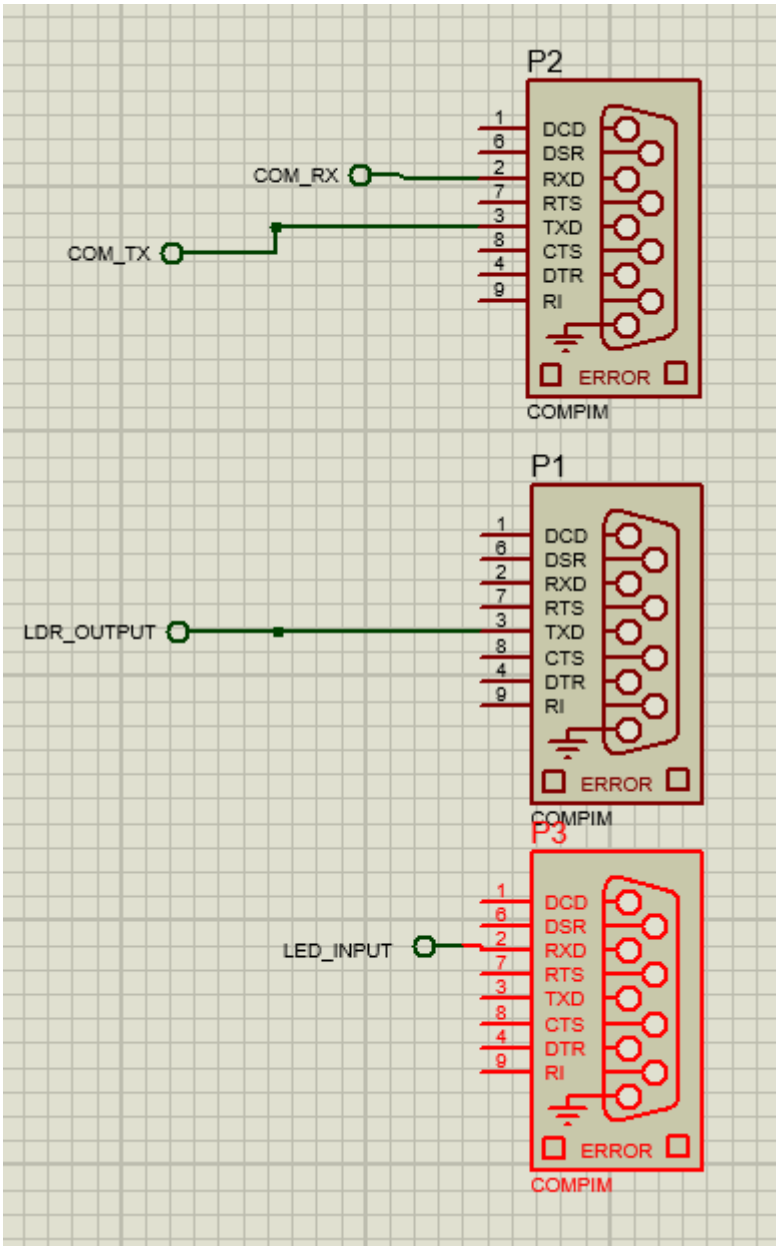
- El pin 1 se utiliza para transmitir datos.
- Los pines digitales 22-53 son utilizados para capturar la señal de que los estacionamientos estan siendo ocupados. Esto con el accionamiento del sensor y el indicador led Rojo.

Arduino Mega Esclavo 2 (LED)



- Los pines 0,1 son utilizados para la transmicion de datos.
- El pin Analogo A0, se utiliza para el accionamiento de la alarma.
- Los pines 22-53 son utilizados para indicar que un estacionamiento ha sido reservado.

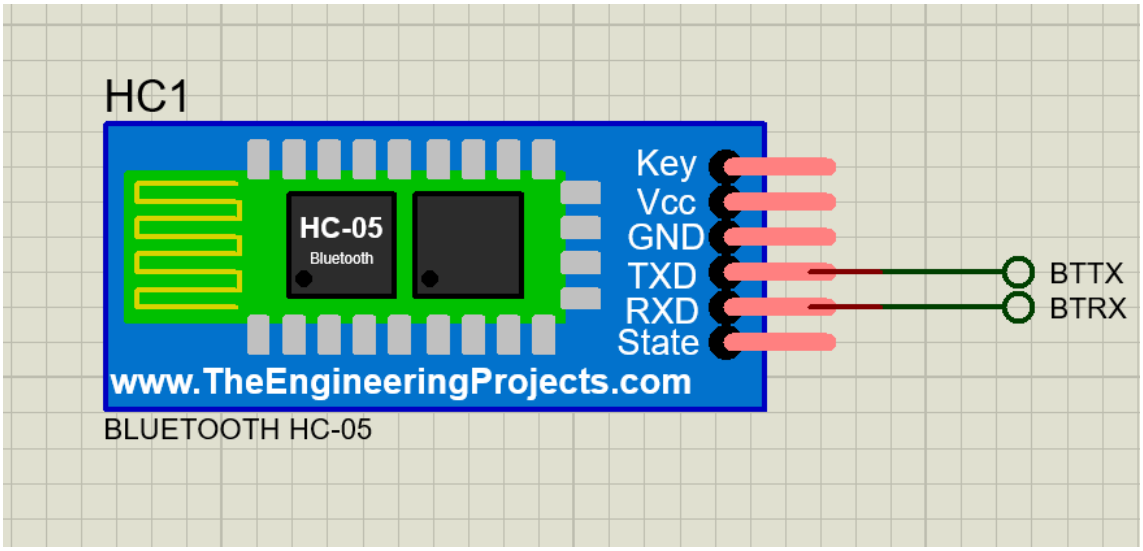
COMPIM (COM Port Physical Interface Model)



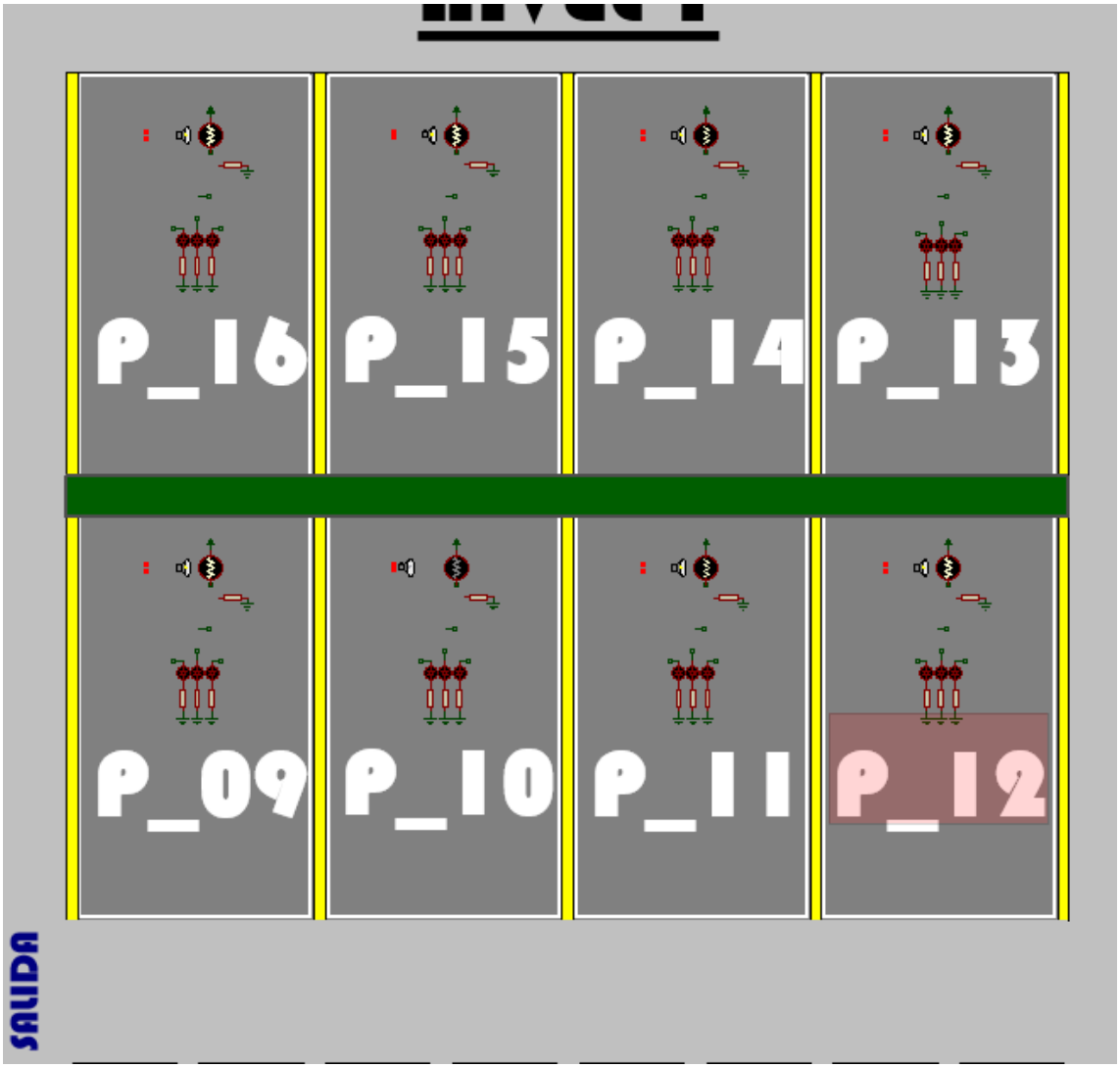
Configuración de los COMPIM, establecidos en los puerto COM1, COM5 y COM7 para la comunciación entre los arduinos Mega.

Bluetooth

Utilizado para el accionamiento de los motores.



Simulacion de los parqueos



Tablero que simula la posicion de cada uno de los parqueos, cada uno con su sensor y luces led.

Aplicacion con Python

API para la comunicacion con la aplicacion

```
for position in position_handler.positions:
    print(position.id)
@app.route("/")
def index():
    return "<h1>Ruta Principal 14k</h1>"


@app.route('/set-reserved/<id>',methods=['POST'])
def reserved(id):
    response={}
    user=request.json['user']
```

```
    position_handler.set_parking_reserved(int(id.split("=")[1]),int(user))
    return response

@app.route('/set-occupied/<id>',methods=['POST'])
def occupied(id):
    response={}
    user=request.json['user']
    position_handler.set_parking_occupied(int(id.split("=")[1]),int(user))
    return response

@app.route('/decline-reserved/<id>',methods=['POST'])
def decline(id):
    response={}
    user=request.json['user']
    position_handler.set_parking_free_by_reserved(id,int(user))
    return response

@app.route('/accept-reserved/<id>',methods=['POST'])
def accept(id):
    response={}
    user=request.json['user']
    position_handler.set_parking_occupied_by_reserved(id,int(user))
    return response


@app.route('/get-level-01',methods=['GET'])
def level_one():
    return "{"+position_handler.return_level_one().replace("["," 
{").replace("]","}").replace("{","").replace("}","")+"}"

@app.route('/get-level-02',methods=['GET'])
def level_two():
    return "{"+position_handler.return_level_two().replace("["," 
{").replace("]","}").replace("{","").replace("}","")+"}"

@app.route('/analytics',methods=['GET'])
def analytics():
    return position_handler.return_parking_stats()

@app.route('/refresh',methods=['GET'])
def refresh():
    return position_handler.return_parking_stats()

@app.route('/alarm-stats/<user>',methods=['GET'])
def alarm(user):
    return position_handler.return_alarm_analytics(int(user))

@app.route('/alarm-off',methods=['POST'])
def off():
    response={}
    user=request.json['user']
    position_handler.turn_off_the_alarm(int(user))
```

```
    return response

@app.route('/alarm-on',methods=['POST'])
def on():
    response={}
    user=request.json['user']
    position_handler.turn_on_the_alarm(int(user))
    return response
```

Servicios API con Python y comunicacion entre Arduinos

Comunicacion serial con Python

```
class Service:
    def __init__(self):
        self.serMASTER = serial.Serial('COM2', 9600, timeout=1)
        self.serLDR = serial.Serial('COM8', 9600, timeout=1)
        self.serLED = serial.Serial('COM6', 9600, timeout=1)

        self.serMASTER.flush()
        self.serLDR.flush()
        self.serLED.flush()


    def writeMASTER(self, msg):
        self.serMASTER.write(msg.encode('utf-8')+b"\r\n")
        time.sleep(0.1)

    def writeLDR(self, msg):
        self.serLDR.write(msg.encode('utf-8')+b"\r\n")
        time.sleep(0.1)

    def writeLED(self, msg):
        self.serLED.write(msg.encode('utf-8')+b"\r\n")
        time.sleep(0.1)

    def readMASTER(self):
        self.serMASTER.flush()
        return self.serMASTER.readline().decode('utf-8').rstrip()

    def readLDR(self):
        self.serLDR.flush()
        return self.serLDR.readline().decode('utf-8').rstrip()

    def readLED(self):
        self.serLED.flush()
        return self.serLED.readline().decode('utf-8').rstrip()

    def closeMaster(self):
        self.serMASTER.close()

    def closeLDR(self):
```

```
        self.serLDR.close()

    def closeLED(self):
        self.serLED.close()

    def time(self):
        return time.strftime("%H:%M:%S", time.localtime())

service = Service()
```

Servicios API con Python para los Arduinos

```
class API:
    def __init__(self):
        self.data = ""
        self.base_url = "http://3.144.214.162:5000/"
        self.parqueos_reservados = 0

    def get_all_reserved(self):
        url ="get-all-reserved"
        data= ""
        converted_data = ""
        try:
            response_API = requests.get(self.base_url+url)
            data = response_API.text
            parse = json.loads(data)
            self.parqueos_reservados = len(parse)


            lista_reservados  =
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
            for p in parse:
                lista_reservados[int(p-1)] = 1;

            for i in lista_reservados:
                converted_data += str(i)

            print("API DATA: "+converted_data)

            return converted_data
        except Exception as e:
            print("Error: "+str(e))
            return "00000000000000000000000000000000"


    def post_parking_change(self, parqueos):
        url = "change-parking-proteus"
        response = ""
        for parqueo in parqueos:
            body = {
                "id": int(parqueo)
            }
```

```
            try:
                response_API = requests.post(self.base_url+url, json=body)
                print(response_API.text)
                response += response_API.text
            except:
                pass
        return response

    def get_parking_alarm(self):
        url = "get-alarm-status"
        data = ""
        converted_data = ""
        try:
            response_API = requests.get(self.base_url+url)
            data = response_API.text
            parse = json.loads(data)

            index =-1;
            converted_data = str(parse[0]) + ";" +str(parse[1])


            print("ALARM DATA: "+converted_data)


            return converted_data
        except Exception as e:
            print("Error: "+str(e))
            return "0;0"
```

Monitoreo de informacion del parqueo para actualizar la APP

```
class Monitor:
    def __init__(self, service):
        self.service = service
        self.thread = threading.Thread(target=self.run)
        self.thread.start()
        self.last_ldr = "LDR;00000000000000000000000000000000"
        self.last_led = "LED;00000000000000000000000000000000"
        self.parqueos_reservados= 0
        self.parqueos_ocupados = 0
        self.parqueos_disponibles = 0 # 32 - (reservados + ocupados)
        self.api = API()

    def run(self):
        while True:
            entrada_master = self.service.readMASTER()
            entrada_ldr = self.service.readLDR()
            entrada_led = self.service.readLED()
            print(self.service.time(),"@ MASTER >" ,entrada_master)
            print(self.service.time(),"@ LDR    >" ,entrada_ldr)
            print(self.service.time(),"@ LED    >" ,entrada_led)
```

```
            if entrada_ldr.startswith("LDR;"):
                print("LDR DATA DETECTED")
                self.compareLDR(entrada_ldr)


            if entrada_led.startswith("LED;"):
                print("LED DATA DETECTED")
                comandos = []
                comandos = entrada_led.split(";")
                #comandos[1] = parqueo
                #comandos[2] = estado
                if(comandos[2] == "1"): #1 = reservar #0 = liberar
                    self.agregarReserva(comandos[1])
                else:
                    self.liberarReserva(comandos[1])


            print("API CHECKs")
            self.compareLED("LED;"+self.api.get_all_reserved())
            self.compareAlarm(self.api.get_parking_alarm())
            self.parqueos_reservados = self.api.parqueos_reservados
            self.parqueos_disponibles = 32 - (self.parqueos_reservados +
self.parqueos_ocupados)

            data_parqueos =
str(self.parqueos_disponibles)+";"+str(self.parqueos_reservados)+";"+str(self.parqueos_c

            print("PARQUEOS: "+data_parqueos)
            self.service.writeMASTER("P;"+data_parqueos)
            time.sleep(1)

    def compareLDR(self, entrada_ldr):
        self.parqueos_ocupados = entrada_ldr.count("1")

        print("\t-Comparando LDR")
        data_to_send = "";
        parqueos_diferentes = []
        # comparar entrada_ldr con el ultimo valor de entrada_ldr
        if entrada_ldr != self.last_ldr:
            print("\t\t- LDR cambiado")
            #buscar que indice es distinto
            for i in range(4, len(entrada_ldr)):
                if entrada_ldr[i] != self.last_ldr[i]:
                    data_to_send+= "1"

                else:
                    data_to_send+= "0"


            for i in range(len(data_to_send)):
                if data_to_send[i] == "1":
```

```
                    parqueos_diferentes.append(i+1)
            print("\t\t- Data to send: ", data_to_send, " Parqueos diferentes: ",
parqueos_diferentes)

            # iterar parqueos diferentes [2,4,6]

            self.api.post_parking_change(parqueos_diferentes)

            self.last_ldr = entrada_ldr

        else:
            print("\t\t- LDR sin cambios")

    def compareLED(self, entrada_led):
        print("\t-Comparando LED")
        data_to_send = "";
        parqueos_diferentes = []
        # comparar entrada_led con el ultimo valor de entrada_led
        if entrada_led != self.last_led:
            print("\t\t- LED cambiado")
            #buscar que indice es distinto
            for i in range(4, len(entrada_led)):
                if entrada_led[i] != self.last_led[i]:
                    data_to_send+= "1"

                else:
                    data_to_send+= "0"


            for i in range(len(data_to_send)):
                if data_to_send[i] == "1":
                    parqueos_diferentes.append(i+1)
            print("\t\t- Data to send: ", data_to_send, " Parqueos reservados: ",
parqueos_diferentes)

            self.service.writeLED(entrada_led)
            self.last_led = entrada_led
        else:
            print("\t\t- LED sin cambios")


    def compareAlarm(self, status):
        # if(status == "1;0"):

        # else{

        # }
        self.service.writeLED(str(status))

# main
```

```
monitor = Monitor(service)
monitor.thread.join()
```

Código Arduino

Bloque de declaraciones

Este bloque contiene todas las variables, definiciones y constantes requeridas para el control de las matrices led y los pines definidos para el arduino.
Los arreglos static byte contiene los bytes que conforman cada uno de los caracteres a leer.

MAESTRO

```
bool ldr_data_ready = false;
String ldr_data = "";

bool led_test = false;

char estado = '0';

// BARRERAS

//PINES PARA LA BARRERA DE ENTRADA
#define BARR1_1 8
#define BARR1_2 9
#define BARR1_3 10
#define BARR1_4 11
//PINES PARA LA BARRERA DE SALIDA
#define BARR2_1 4
#define BARR2_2 5
#define BARR2_3 6
#define BARR2_4 7

// PANTALLA
#include <LiquidCrystal.h>
LiquidCrystal lcd(22, 23, 24, 25, 26, 27);

int x1 = 16;
int x2 = 17;
int x3 = 18;
int x4 = 19;
int x5 = 21;

int parqueos_disponibles = 0;
int parqueos_ocupados = 0;
int parqueos_reservados = 0;
```

ESCLAVO1 LDR

```
int LDR_values[32] = {0};
bool ldr_readed = false;
bool ldr_printed = false;
bool data_requested = false;
int last_millis = -1;

#define LDR_CHECK 13
```

ESCLAVO2 LED

```
int LED_values[32] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int last_millis = -1;

bool led_data_ready = false;
String led_data = "";

#define LED_CHECK  13
#define BUZZER A0
```

Bloque SETUP

Este bloque se encarga de definir el modo de los pines e inicializacion de objetos.

MAESTRO

```
void setup() {

  // empezar puerto serial 0 => PC
  Serial.begin(9600);
  Serial.println("MASTER READY");

  // empezar puerto serial 1 => ARDUINO LDR
  // empezar puerto serial 2 => ARDUINO MASTER

  Serial1.begin(9600);
  Serial2.begin(9600);
  Serial2.println("MASTER READY");

  pinMode(ALARM_CHECK, INPUT);

  pinMode(BARR1_1, OUTPUT);
  pinMode(BARR1_2, OUTPUT);
  pinMode(BARR1_3, OUTPUT);
  pinMode(BARR1_4, OUTPUT);

  pinMode(BARR2_1, OUTPUT);
  pinMode(BARR2_2, OUTPUT);
  pinMode(BARR2_3, OUTPUT);
  pinMode(BARR2_4, OUTPUT);
```

```
  //AL INICIO COLOCAR LOS MOTORES EN SU ESTADO INICIAL
  estadoInicial_1();
  estadoInicial_2();

    //PANTALLA
    lcd.begin(16, 2);

}
```

ESCLAVO1 LDR

```
void setup(){
    // es el serial 0 por que se comunicarara por el puerto Serial0 al puerto Serial1
de la placa de arduino
    Serial.begin(9600);
    // Serial.println("LDR READY");
    for(int i = 0; i< 32; i++){
        pinMode(i+22, INPUT);
    }

    pinMode(LDR_CHECK, INPUT);

    Serial.println("LDR READY");

}
```

ESCLAVO2 LED

```
void setup(){
    // SOLO RECIBIRA INFORMACION
    Serial.begin(9600);
    for(int i = 0; i<32; i++){
        pinMode(i+22, OUTPUT);
        digitalWrite(i+22, LOW);
    }

    pinMode(LED_CHECK, INPUT);

}
```

Bloque LOOP

Este bloque contiene lo que se ejecuta ciclicamente.

MAESTRO

```
void loop() {
    delay(1000);
    checkForIncomingPetitions();

    if(parqueos_disponibles == 0){
```

```
        nohayparqueos();
        delay(1000);
    }else{
        ocupaciones(parqueos_disponibles, parqueos_ocupados, parqueos_reservados);
        delay(1000);
        parqueosdisponibles(parqueos_disponibles);
        delay(1000);
    }
}
```

ESCLAVO1 LDR

```
void loop(){
    printToSerial();
    delay(2000);
}
```

ESCLAVO2 LED

```
void loop(){
    waitForSignal();
}
```

Funciones y Metodos utilizados por los bloques anteriores

MAESTRO

```
// estado: reservado = 1, no reservado = 0; 2 = alarma desactivada
void enviarReservacion(int parqueo) {
  digitalWrite(LED_CHECK, HIGH);
  Serial1.print("LED;4;1");
  Serial2.print("Reservacion enviada");
  Serial.print("LED;4;1");
  delay(1000);
  digitalWrite(LED_CHECK, LOW);

}

void desactivarAlarma() {
  digitalWrite(LED_CHECK, HIGH);
  Serial1.print("LED;4;2");
  delay(1000);
  digitalWrite(LED_CHECK, LOW);
}

void checkForIncomingPetitions() {
    char estado = '0';
    // Serial1 es bluetooth
    if (Serial1.available() > 0) {
        estado = Serial1.read();
```

```
        // switch de estados

        // estados para bluetooth | A = abrir entrada ; C = abrir salida ; D =
desactivar alarma ; R = reservar parqueo

        if (estado == 'A') {
        // funcion para abrir barrera de entrada
            activarBarrera_1();
        }

        if (estado == 'C' && digitalRead(ALARM_CHECK) == LOW) {
        // funcion para abrir barrera de salida
            activarBarrera_2();
        }
        }
    String mensaje = "";
    while(Serial.available()> 0){
        mensaje += (char)Serial.read();
        delay(10);
    }

    if(mensaje != ""){
        Serial2.println("Recibido");
        Serial2.println(mensaje);
        interpretarMensaje(mensaje);
    }

}

void interpretarMensaje(String msg){
    // parse msg to array
    String mensaje = msg;
    // convertir a arreglo
    char mensaje_array[mensaje.length()];
    mensaje.toCharArray(mensaje_array, mensaje.length());
    int longitud = mensaje.length();

    String disponibles="";
    String reservados="";
    String ocupados="";

    // P;<disponibles>;<reservados>;<ocupados>
    if(mensaje_array[0]  == 'P' && mensaje_array[1] == ';'){
        for(int i=2; i<longitud; i++){
            if(mensaje_array[i] == ';'){
                for(int j=i+1; j<longitud; j++){
                    if(mensaje_array[j] == ';'){
                        for(int k=j+1; k<longitud ; k++){
                            ocupados += mensaje_array[k];
                        }
                        break;
```

```
                    }
                    reservados += mensaje_array[j];
                }
                break;
            }
            disponibles += mensaje_array[i];
        }
    }

    parqueos_disponibles = disponibles.toInt();
    parqueos_reservados = reservados.toInt();
    parqueos_ocupados = ocupados.toInt();

}
```

ESCLAVO1 LDR

```
void waitForSignal(){

    if(digitalRead(LDR_CHECK) == HIGH && (millis() - last_millis) > 1000 ||
last_millis == -1){
        data_requested = true;
    }

    if(ldr_readed == true && data_requested == true){

        ldr_readed = false;
        data_requested = false;
        last_millis = millis();

    }

}

void getLDR(){
    for(int i = 0; i< 32; i++){
        LDR_values[i] = digitalRead(i+22);
    }
}

void printToSerial(){
    getLDR();
    String response = "LDR;";
        //send as string to serial monitor all the values
    for(int i = 0; i< 32; i++){
        response += LDR_values[i];

    }

    Serial.println(response);
}
```

ESCLAVO2 LED

```
void waitForSignal(){
    // leer puerto serial
    String message = "";
    while(Serial.available()> 0){
        message += (char)Serial.read();
        delay(10);
    }
    if(message != ""){
        Serial.println("Recibido: ");
        Serial.println(message);
        interpretarLED(message);
        message = "";
    }

    encenderLED();
}

void askLED(){
    // read Serial info
    String response = "";
    while(Serial.available() > 0){
        response += (char)Serial.read();
    }
    Serial.println("Recibido: ");
    Serial.println(response);

    led_data = "";
    led_data = response;
}

void interpretarLED(String message){
    // convertir mensaje a arreglo de caracteres
    Serial.println("Interpretando");
    Serial.println(message);
    char message_array[message.length()];
    message.toCharArray(message_array, message.length());

    //  los primeros 4 elementos son "LED;" el resto son 32 numeros equivalentes a los
pines que hay que encender en amarillo
    if(message_array[0] == '0' && message_array[1] ==';'){
        String espacio = "";
        for(int i = 2; i<message.length(); i++){
            espacio += message_array[i];
        }

        if(espacio.toInt() != 0){
            sonarAlarma(espacio.toInt());
        }

    }
```

```
    if(message_array[0] == 'L' && message_array[1] == 'E' && message_array[2] == 'D'
&& message_array[3] == ';'){


        for(int i = 4; i<message.length(); i++){
                if(message_array[i] == '1'){
                    LED_values[i-4] = 1;
                }

                if(message_array[i] == '0'){
                    LED_values[i-4] = 0;
                }

        }
    }

}

void sonarAlarma(int espacio){
    for(int i = 600; i<700; i++){
        tone(BUZZER, i);
        digitalWrite(espacio+21, HIGH);
        delay(10);
        digitalWrite(espacio+21, LOW);
        delay(10);

    }
    noTone(BUZZER);

}

void encenderLED(){
    // encender los pines
        for(int i = 0; i<32; i++){

            if(LED_values[i] == 1){
                digitalWrite(i+22, HIGH);
            }else{
                digitalWrite(i+22, LOW);
            }
        }
}
```