DOCUMENTACION – PRACTICA 2

19/08/2024

Universidad de San Carlos de Guatemala

Curso: Arquitectura de computadores y ensambladores 1

Nombre	Carné
Luis Manuel Pichiyá Choc	202201312
Edgar Josías Cán Ajquejay	202112012
Sebastián Alejandro Vásquez Cartagena	202109114
Geovanni Eduardo Nufio Illescas	201901444
Cristhian Raúl Guamuch Cumes	202208930

CONTENIDO

PROGRAMACION USADA EN EL PROTOTIPO

PROGRAMACION DE LUCES CON SENSOR

En este caso se utiliza el sensor TORCH-LDR como sensor de luz en donde entre mas acercamiento tiene con la luz natural, los LEDS se mantendrán apagados, sin embargo cuando el mismo se va alejando cada vez mas hasta que no exista luz natural en la casa, todas las luces se procederán a encender de forma automática. En la primera imagen se observa los valores del arreglo de luces que tiene la casa, y en la segunda imagen la lógica para el sensor como se explico anteriormente.

```
17  LUCES_CASA = [33, 32, 8, 10]
```

```
48      # ENCENDER Y APAGAR LUZ CON SENSOR
49      for pin in LUCES_CASA:
50          GPIO.setup(pin, GPIO.OUT)
51
52      GPIO.setup(SENSOR_LUZ, GPIO.IN)
53      GPIO.setup(ENCENDER_LUCES, GPIO.IN)
```

PROGRAMACION LUCES DE FORMA MANUAL

En este caso se utiliza un botón que esta conectado a la rapsberry en donde al momento de indicar de que si el sensor de luz es 0 es decir que esta apagado y que el botón para encender la luz este activado es decir mande una señal de 1, entonces todas las luces se encenderán mientras el botón este encendido, en la primera imagen se observa los valores que tiene los sensores de luz y de encender luces en los pines de la rapsberry donde se asigno y la siguiente imagen se muestra la lógica del mismo.

```
18  SENSOR_LUZ = 31
19
20  ENCENDER_LUCES = 36
```

```
60      #BOTON DE LUZ APAGADO
61      while True:
62          # Send some test
63
64          if GPIO.input(SENSOR_LUZ) == 0 or GPIO.input(ENCENDER_LUCES) == 1:
65              for pin in LUCES_CASA:
66                  GPIO.output(pin, GPIO.LOW)
67          else:
68              for pin in LUCES_CASA:
69                  GPIO.output(pin, GPIO.HIGH)
70
71
72          sleep(1)
```

PROGRAMACION DE CONTRASEÑA PARA INGRESAR

Para la contraseña se definieron los pines para cada botón de la contraseña y se configuro la misma en el GPIO de la raspberry, posteriormente se evaluara el patron de presionar cada botón en el orden que se le especifico, conforme ingrese la contraseña se guardara el botón presionado y evaluara el patron si el patron es correcto, mostrara que la contraseña es correcta, en caso contrario mostrara que es incorrecta, esto se mostrara en consola como en la pantalla LCD. Así mismo cuando se presione enter de forma automática sea correcta o no se borrara el patron de la memoria.

```
23  ## Contrasena
24  BOTON_SECCION = 3
25  BOTON_ASTERISCO = 5
26  BOTON_GRUPO = 7
27  BOTON_ENTER = 29
```

```
103     GPIO.setup(BOTON_GRUPO, GPIO.IN)  # primero
104     GPIO.setup(BOTON_SECCION, GPIO.IN)  # segundo
105     GPIO.setup(BOTON_ASTERISCO, GPIO.IN)  # tercero
106     GPIO.setup(BOTON_ENTER, GPIO.IN)
```

```
162         # LOGICA CONTRA
163         if GPIO.input(BOTON_GRUPO) == 1 and not BOTON_GRUPO in password and enter_reiniciado:
164             password.append(BOTON_GRUPO)
165             print("BOTON GRUPO GUARDADO")
166         elif GPIO.input(BOTON_SECCION) == 1 and not BOTON_SECCION in password and enter_reiniciado:
167             password.append(BOTON_SECCION)
168             print("BOTON_SECCION GUARDADO")
169         elif GPIO.input(BOTON_ASTERISCO) == 1 and not BOTON_ASTERISCO in password and enter_reiniciado:
170             password.append(BOTON_ASTERISCO)
171             print("BOTON_ASTERISCO GUARDADO")
```

```
175         if GPIO.input(BOTON_ENTER) == 1 and len(password) == 3 and enter_reiniciado:
176             # enviar informacion al display
177             counter = 0
178             valid = False
179             for item in [BOTON_GRUPO, BOTON_SECCION, BOTON_ASTERISCO]:
180                 if password[counter] != item:
181                     valid = False
182                     break
183                 else:
184                     valid = True
185                     counter += 1
186 
187             if valid:
188                 message_thread = threading.Thread(target=lcd_temp_message, args=(house_state, "Password", "Correcta"))
189                 message_thread.start()
190                 print("PASSWORD CORRECTA")
191             else:
192                 message_thread = threading.Thread(target=lcd_temp_message, args=(house_state, "Password", "Incorrecta"))
193                 message_thread.start()
194                 print("PASSWORD INCORRECTA")
195 
196             password = []
197             enter_reiniciado = False
198         if GPIO.input(BOTON_ENTER) == 0:
199             enter_reiniciado = True
```

PROGRAMACION DE ALARMA

En este caso la alarma se activara solamente si existe un incendio o humo, si es 1 se activaran los rociadores y a su vez se activara la alarma, en caso contrario los rociadores estarán apagado, es decir cuando sea 0 y por lo tanto la alarma dejara de sonar.

```
146         # LOGICA SENSOR FUEGO
147         if GPIO.input(SENSOR_FUEGO) == 1:
148             GPIO.output(ROCIADORES, GPIO.HIGH)
149             GPIO.output(ALARMA, GPIO.HIGH)
150         else:
151             GPIO.output(ROCIADORES, GPIO.LOW)
152             GPIO.output(ALARMA, GPIO.HIGH)
```

PROGRAMACION SENSOR DE TEMPERATURA CON AIRE ACONDICIONADO

En este caso todos los datos serán presentados en la pantalla LCD en donde se mostrara la temperatura en grados centígrados. Si la temperatura de la casa es mayor a 27 grados y el botón del aire esta desactivado, procederá a encender de forma automática el aire acondicionado porque la temperatura es alta, ahora si la temperatura de la casa es mayor a 27 grados pero el botón de aire esta activado se apaga el aire acondicionado automáticamente. Ahora si la temperatura es menor a 27 grados entonces procederá a apagar los pines del aire acondicionado por lo que no funcionara ya que no es necesario porque la temperatura es menor a la impuesta.

```
197         # LOGICA DE TEMPERATURA
198         sleep(1)
199         # Leer temperatura
200         temp_level = ReadChannel(channel_temp)
201         temp = ConvertTemp(temp_level, 2)
202         new_house_state = ["Temp " + str(int(temp)) + " c", formatted_date]
203         if temp > 27 and GPIO.input(BOTON_AIRE) == 0:
204             GPIO.output(15, GPIO.HIGH)
205             GPIO.output(16, GPIO.LOW)
206             sleep(0.5)
207             GPIO.output(15, GPIO.LOW)
208             GPIO.output(16, GPIO.HIGH)
209         elif temp > 27 and GPIO.input(BOTON_AIRE) == 1:
210             GPIO.output(15, GPIO.LOW)
211             GPIO.output(16, GPIO.LOW)
212         else:
213             GPIO.output(15, GPIO.LOW)
214             GPIO.output(16, GPIO.LOW)
```

PROGRAMACION SISTEMA DE RIEGO DE INVERNADERO

Inicializando los pines a usar. para esta parte se definió el sistema de riego que funcionara a través de un botón en donde si este esta presionado es decir si enviar una señal de 1, se activaran los regadores, en caso contrario estará apagado, el que este activado o desactivado se mostrara en la pantalla LCD indicando que se esta regando caso contrario no mostrara nada porque no esta presionado el boton.

```
114     GPIO.setup(BOTON_RIEGO, GPIO.IN)
115     GPIO.setup(REGADORES, GPIO.OUT)
```

```
137     # BOTON DE LUZ APAGADO
138     while True:
139         # LOGICA BOTON DE RIEGO
140         if GPIO.input(BOTON_RIEGO) == 1:
141             GPIO.output(REGADORES, GPIO.HIGH)
142             lcd_string("Regando", LCD_LINE_1)
143             lcd_string("Invernadero", LCD_LINE_2)
144             sleep(2)
145         else:
146             GPIO.output(REGADORES, GPIO.LOW)
147             new_house_state[0] = house_state[0] + " "
```

COMPONENTES Y ELEMENTOS ADICIONALES

Define un mapeo que tiene la pantalla LCD para cada pin del LCD.

```
12  # Define GPIO to LCD mapping (using BOARD numbering)
13  LCD_RS = 26  # BOARD pin 37
14  LCD_E = 15  # BOARD pin 18
15  LCD_D4 = 13  # BOARD pin 15
16  LCD_D5 = 37  # BOARD pin 12
17  LCD_D6 = 22  # BOARD pin 36
18  LCD_D7 = 18 # BOARD pin 32
```

LIBRERÍAS UTILIZADAS

Estas son las librerías que se utilizaron, que controlan los pines de la raspberry, una librería de hilos manejando varias tareas en un cierto tiempo, una librería para mostrar la fecha y hora y una librería para comunicación SPI.

```
1  import RPi.GPIO as GPIO
2  import threading
3  from datetime import datetime
4  from time import sleep
5  import spidev
```

DEFINICION DE PINES

Aquí se definió los pines utilizados en todo el funcionamiento, tanto pines de los sensores, los pines que se usaron para la pantalla LCD, los valores predeterminados que tiene la pantalla LCD, así como la variables para los hilos de tiempo en algunos componentes. También se colocaron valores para el sensor de luz así como para la temperatura.

```python
# Definir los pines
LDR_PIN = 29
LED_PIN = 31

# Define GPIO to LCD mapping (using BOARD numbering)
LCD_RS = 26  # BOARD pin 37
LCD_E = 12  # BOARD pin 18
LCD_D4 = 13  # BOARD pin 15
LCD_D5 = 37  # BOARD pin 12
LCD_D6 = 22  # BOARD pin 36
LCD_D7 = 18 # BOARD pin 32

# Define some device constants
LCD_WIDTH = 16  # Maximum characters per line
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80  # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0  # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
```

```python
####################
LUCES_CASA = [33]
SENSOR_LUZ = 31

ENCENDER_LUCES = 36

BOTON_AIRE = 11

SENSOR_FUEGO = 32
ROCIADORES = 40
ALARMA = 32

BOTON_RIEGO = 35
REGADORES = 38


## Contrasena
BOTON_SECCION = 3
BOTON_ASTERISCO = 5
BOTON_GRUPO = 7
BOTON_ENTER = 29
```

```python
# Variables para el sensor de luz
spi = spidev.SpiDev()
spi.open(0, 0)

# Puerto de la temperatura
channel_temp = 0
```

MOSTRAR DATOS TEMPERATURA

lee el valor del canal del sensor de temperatura a través de SPI y convierte el valor leído en una temperatura en grados Celsius.

```python
# Leer puerto de la temperatura
def ReadChannel(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    data = ((adc[1] & 3) << 8) + adc[2]
    return data

# Voltaje recibido a la temperatura
def ConvertTemp(data, places):
    temp = ((data * 330) / float(1023))
    temp = round(temp, places)
    return temp
```

LIBRERIAS USADAS

LIBRERÍA ADAFRUIT

La librería Adafruit proporciona una serie de módulos de software diseñados para interactuar con hardware desarrollado por Adafruit, como sensores, pantallas, y otros dispositivos electrónicos. Estas bibliotecas están destinadas a facilitar la integración y control de dispositivos en proyectos de electrónica y programación, particularmente en plataformas como Raspberry Pi, Arduino, y otros microcontroladores.

Link: https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi

LIBRERÍA SPIDEV

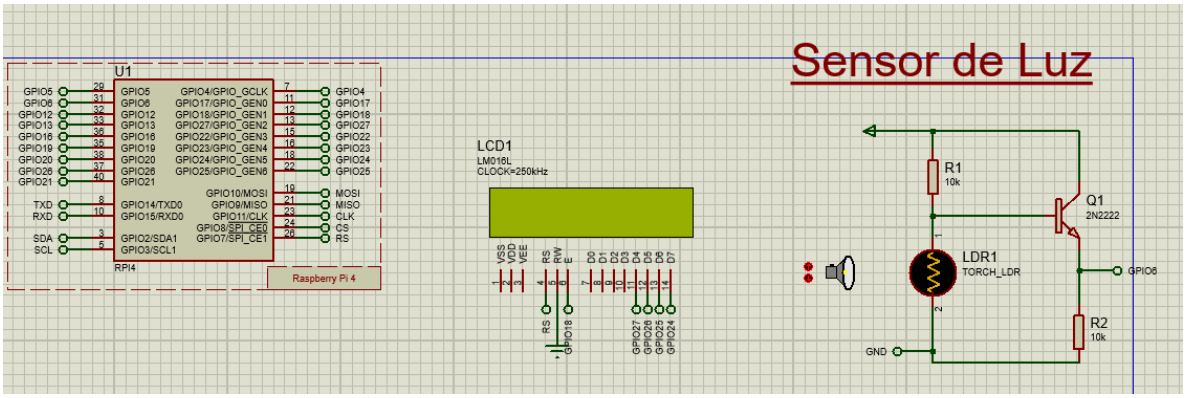
es una librería de Python que proporciona una interfaz para el bus SPI (Serial Peripheral Interface) en sistemas basados en Linux, como la Raspberry Pi. Esta librería permite comunicarse con dispositivos conectados a través del bus SPI, como sensores, controladores de pantalla, memorias flash, entre otros

Link: https://github.com/doceme/py-spidev

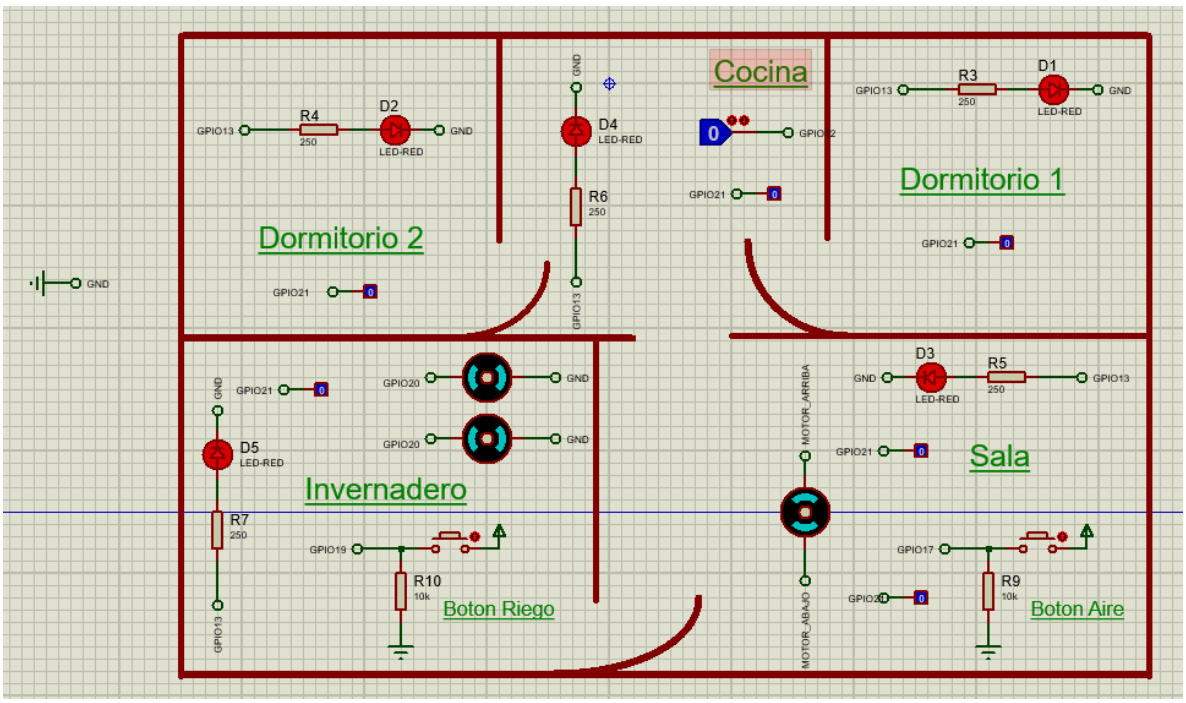
COSTO DEL PROTOTIPO

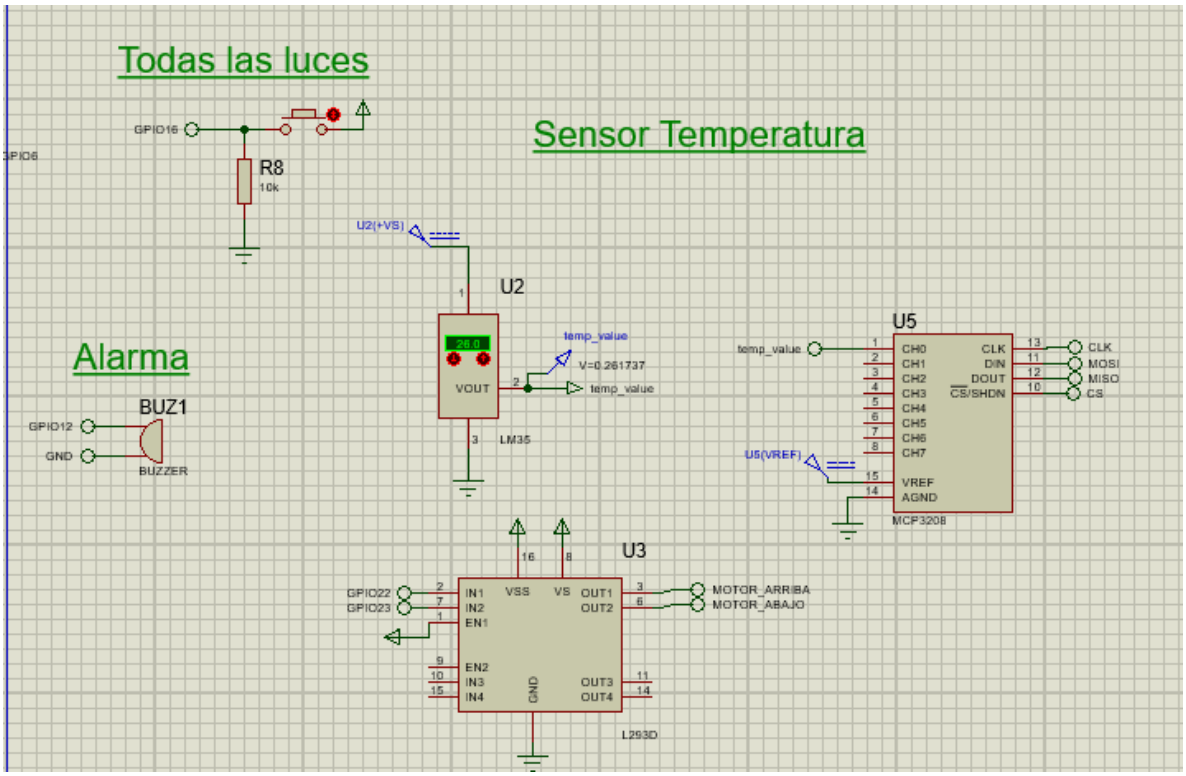
COMPONENTE	COSTO
MAQUETA	**Q250.00**
RAPSBERRY PI 4	**Q550.00**
SENSOR DE LLAMA	**Q60.00**
SENSOR DE LUZ	**Q46.00**
VENTILADORES	**Q60.00**
PANTALLA LCD 16X2	**Q40.00**
LEDS DE COLORES	**Q40.00**
PUSH BUTTONS	**Q15.00**
RESISTENCIAS(VARIAS)	**Q25.00**
CONVERSOR LOGICO	**Q29.00**
BUZZER	**Q10.00**
PROTOBOARDS	**Q60.00**
JUMPERS(MACHO Y HEMBRA)	**Q24.00**
SENSOR DE HUMEDAD	**Q25.00**
SENSOR DE TEMPERATURA	**Q30.00**
ESTAÑO	**Q15.00**
CABLE PARA PROTOBOARD	**Q25.00**
CAUTIN	**Q70.00**
CORTACABLES	**Q75.00**
TOTAL	**Q1100.00**

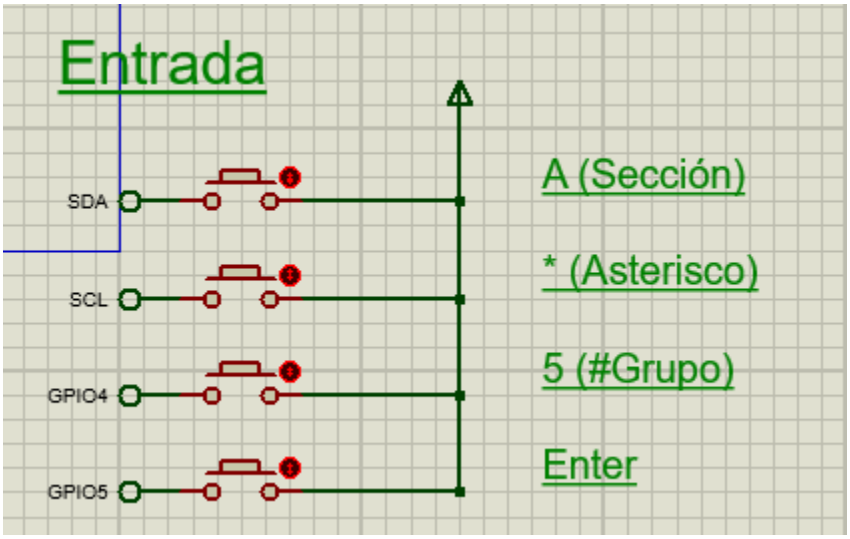
CAPTURAS DEL DIAGRAMA ELECTRONICO EN PROTEUS

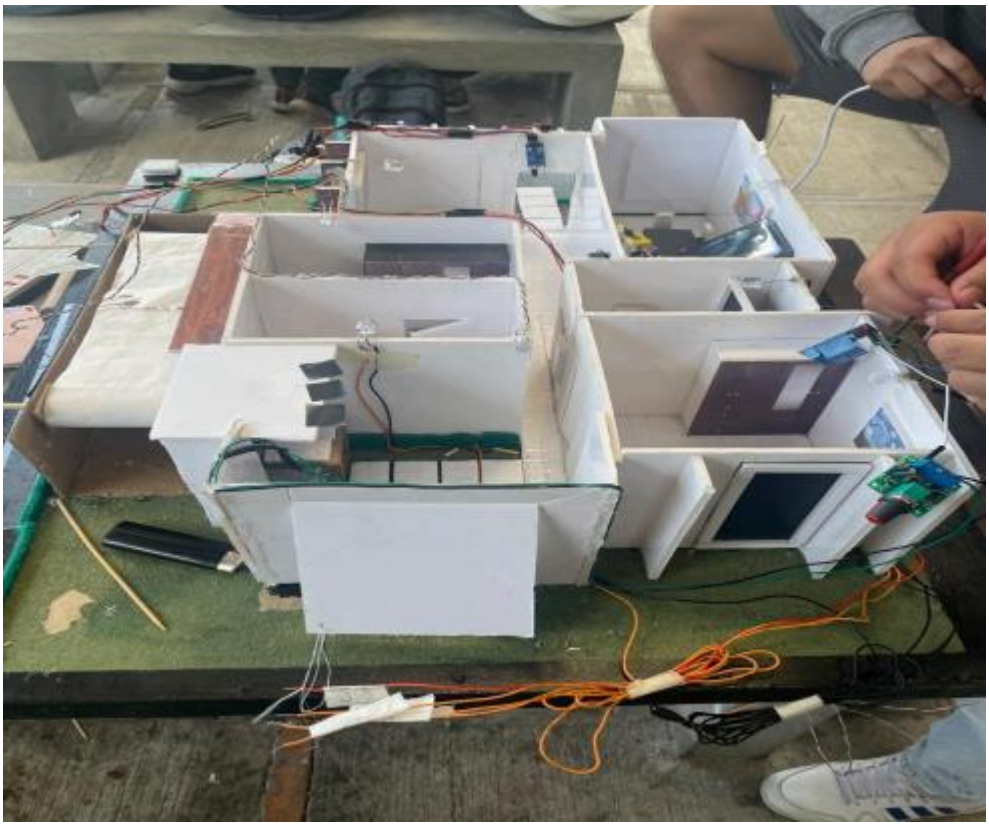


+




Todas las luces
GPIO16
R8
10k
Sensor Temperatura
U2(+VS)
U2
temp_value
V=0.261737
temp_value
VOUT
LM35
Alarma
BUZ1
GPIO12
GND
BUZZER
U5
temp_value
CH0
CH1
CH2
CH3
CH4
CH5
CH6
CH7
VREF
AGND
CLK
DIN
DOUT
CS/SHDN
U5(VREF)
MCP3208
CLK
MOSI
MISO
CS
U3
GPIO22
GPIO23
IN1
IN2
EN1
EN2
IN3
IN4
VSS
VS
GND
OUT1
OUT2
OUT3
OUT4
MOTOR_ARRIBA
MOTOR_ABAJO
L293D


Entrada
SDA
SCL
GPIO4
GPIO5
A (Sección)
* (Asterisco)
5 (#Grupo)
Enter



RESTRICCIONES E INDICACIONES ADICIONALES

Alimentación principal: El sistema requiere una fuente de alimentación estable de 5V DC para la Raspberry Pi 4. Es crucial utilizar un adaptador de corriente de calidad que pueda suministrar al menos 3 amperios para garantizar un funcionamiento correcto, especialmente si se conectan periféricos adicionales.

Componentes externos: Verificar el voltaje de operación de cada componente (sensores, ventiladores, etc.) y asegurarse de que sean compatibles con los 5V o 3.3V proporcionados por la Raspberry Pi. Utilizar un conversor de nivel lógico si es necesario para adaptar los voltajes.

Compatibilidad de sensores: Asegurarse de que los sensores elegidos sean compatibles con la Raspberry Pi 4 y que existan librerías o controladores disponibles para su uso en el lenguaje de programación seleccionado (Python, C++, etc.).

Comunicación: Verificar la forma en que los sensores se comunicarán con la Raspberry Pi (I2C, SPI, UART, etc.) y configurar los pines GPIO correspondientes.

Software: Desarrollar el software necesario para leer los datos de los sensores, procesarlos y activar las salidas (ventiladores, buzzer, pantalla LCD) según la lógica del prototipo.

Alcance de los sensores: Tener en cuenta el alcance de detección de los sensores y ubicarlos estratégicamente para cubrir el área deseada.

Resistencia al calor: Considerar la ubicación del sensor de llama y asegurarse de que los materiales cercanos sean resistentes a altas temperaturas para evitar riesgos de incendio.

Aislamiento eléctrico: Utilizar materiales aislantes adecuados para cubrir cables y conexiones expuestas, previniendo cortocircuitos y descargas eléctricas.