DOCUMENTACION – PRACTICA 3

09/09/2024

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

Con respecto a la activación del 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 este caso se le asigno un valor de intensidad de 500 de luz de parte del sensor.

```
219  def luces_casa():
220      # LOGICA LUCES DE LA CASA
221      ENCENDIDAS = ENCENDER_LUCES.value
222      light_value = read_adc(LUZ_CHANNEL)  # Leer canal 0 (foto resistencia)
223
224      if light_value > 500:
225          if ENCENDIDAS:
226              for pin in LUCES_CASA:
227                  pin.value = True
228          else:
229              for pin in LUCES_CASA:
230                  pin.value = False
```

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. En este caso se implemento con true y false, con 1 es verdadero y con 0 es falso y dependiendo si la señal del pin que tiene el botón si esta presionado se encenderán en caso contrario seguirá apagado.

```
35   ENCENDER_LUCES = digitalio.DigitalInOut(board.D16)
```

```
231      else:
232          if ENCENDIDAS:
233              for pin in LUCES_CASA:
234                  pin.value = False
235          else:
236              for pin in LUCES_CASA:
237                  pin.value = True
```

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.

Esto se dividio en 3 partes

Se guarda el patron para presionar los botones

```
255  def guardar_botones(password):
256      # LOGICA CONTRA
257      if BOTON_GRUPO.value and BOTON_GRUPO not in password:
258          password.append(BOTON_GRUPO)
259          print("BOTON GRUPO GUARDADO")
260      elif BOTON_SECCION.value and BOTON_SECCION not in password:
261          password.append(BOTON_SECCION)
262          print("BOTON_SECCION GUARDADO")
263      elif BOTON_ASTERISCO.value and BOTON_ASTERISCO not in password:
264          password.append(BOTON_ASTERISCO)
265          print("BOTON_ASTERISCO GUARDADO")
266      return password
```

Se verifica que el patron ingresado sea el correcto al patron que se ingreso.

```
269  def verificar_password(password):
270      aceptada = False
271      counter = 0
272      for item in [BOTON_GRUPO, BOTON_SECCION, BOTON_ASTERISCO]:
273          if password[counter] != item:
274              aceptada = False
275              break
276          else:
277              aceptada = True
278              counter += 1
279
280      if aceptada:
281          print("PASSWORD CORRECTA")
282          return aceptada
283      else:
284          print("PASSWORD INCORRECTA")
285          return aceptada
```

Evaluara la contraseña y verificara, si la contraseña ingresada es correcta permitirá el ingreso a la casa, en caso contrario se contara como un intento, la cantidad de intentos solo se permite de 3 intento.

```
131         # When is in Password mode
132         elif estado_actual_casa == HouseStates.PASSWORD:
133             sensor_fuego()
134             luces_casa()
135             if len(password) < 3:
136                 password = guardar_botones(password)
137             elif len(password) == 3 and BOTON_ENTER.value:
138 
139                 aceptado = verificar_password(password)
140                 password = []
141 
142                 if aceptado:
143                     intento = 0
144 
145                     temp_message_thread = threading.Thread(
146                         target=lcd_temp_message,
147                         args=(new_house_state, "Password", "Correcta"),
148                     )
149                     temp_message_thread.start()
150                     estado_actual_casa = HouseStates.NORMAL
151 
152                 else:
153                     temp_message_thread = threading.Thread(
154                         target=lcd_temp_message,
155                         args=(new_house_state, "Password", "Incorrecta"),
156                     )
157                     intento += 1
```

Si se llega al tercer intento la casa se bloqueara y no se podrá acceder en caso contrario si se podrá acceder a la misma o si el intento no es igual a 3 se podrá seguir intentando ingresar..

```
159                     if intento == 3:
160                         estado_actual_casa = HouseStates.BLOCKED
161                         intento = 0
162                         sleep(0.5)
163                     else:
164                         estado_actual_casa = HouseStates.NORMAL
165                         temp_message_thread.start()
166                         sleep(0.5)
167 
168         sleep(0.1)
```

Y en la LCD aparecerá un contador de 15 segundos en decremento para después volver a ingresar la contraseña correcta.

```
288    def bloquear_casa():
289        for i in range(15, 0, -1):
290            lcd.clear()
291            lcd.move_to(0, 0)
292            lcd.putstr("Fallido Espere")
293            lcd.move_to(7, 1)
294            lcd.putstr(str(i + 1))
295            sleep(1)
```

PROGRAMACION DE ALARMA

En este caso la alarma se activara solamente si existe un incendio o humo, en este caso si el valor del lector del sensor de temperatura es mayor de 1500, se activara la alarma de en este caso el buzzer se activara, activando la alarma y activando los rociadores.

```
171    def sensor_fuego():
172        # LOGICA SENSOR FUEGO
173        gas_value = read_adc(GAS_CHANNEL)  # Leer canal 1 (sensor de gas)
174        if gas_value > 1500:
175            ROCIADORES.value = True
176            ALARMA.value = True
177        else:
178            ROCIADORES.value = False
179            ALARMA.value = False
```

La temperatura registrada por el sensor será mostradas en la pantalla LCD junto con la fecha

```
182    def mod_temperature_mesg(new_house_state, formatted_date):
183        # print("#" * 15)
184        # print(new_house_state)
185        # print("#" * 15)
186        new_house_state = sensor_temperatura(formatted_date, new_house_state)
```

En esta parte es donde se muestra en la pantalla LCD los valores de temperatura así como la fecha del mismo,

```
92          if estado_actual_casa == HouseStates.NORMAL:
93              # new_house_state = sensor_temperatura(formatted_date)
94
95              if temperature_message_thread != None:
96                  if not temperature_message_thread.is_alive():
97                      temperature_message_thread = threading.Thread(
98                          target=mod_temperature_mesg,
99                          args=(my_state, formatted_date),
100                     )
101                     temperature_message_thread.start()
102             else:
103                 temperature_message_thread = threading.Thread(
104                     target=mod_temperature_mesg, args=(my_state, formatted_date)
105                 )
106                 temperature_message_thread.start()
```

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, en este caso se le coloco un valor de 1500 de sensor de calor .

```
171 def sensor_fuego():
172     # LOGICA SENSOR FUEGO
173     gas_value = read_adc(GAS_CHANNEL)  # Leer canal 1 (sensor de gas)
174     if gas_value > 1500:
175         ROCIADORES.value = True
176         ALARMA.value = True
177     else:
178         ROCIADORES.value = False
179         ALARMA.value = False
180
181
182 def mod_temperature_mesg(new_house_state, formatted_date):
183     # print("#" * 15)
184     # print(new_house_state)
185     # print("#" * 15)
186     new_house_state = sensor_temperatura(formatted_date, new_house_state)
```

```python
def sensor_temperatura(formatted_date, myhs):
    try:
        temperature = DHT11_DEV.temperature
        humidity = DHT11_DEV.humidity

        if humidity is not None and temperature is not None:
            # print("Temp={0:0.1f}*C  Humidity={1:0.1f}%".format(temperature, humidity))
            pass
        else:
            print("Failed to get reading. Try again!")
            temperature = "Err"
            humidity = "Err"

    except RuntimeError as error:
        print(f"Error reading DHT11: {error}")
        temperature = "Err"
        humidity = "Err"

    except OverflowError as error:
        print(f"OverflowError: {error}")
        sleep(2)  # Pausa antes de reintentar la lectura
        temperature = "Err"
        humidity = "Err"

    new_house_state = ["Temp " + str(temperature) + " C", formatted_date]
    myhs.new_house_state = new_house_state
    sleep(2)
    # return new_house_state
```

COMPONENTES Y ELEMENTOS ADICIONALES

En esta parte se resetean los valores de la casa y los valores de la LCD.

```python
def reset_house_state(temp_message_thread, house_state, new_house_state, myhs):
    if temp_message_thread is not None:
        if (
            not temp_message_thread.is_alive()
            and myhs.new_house_state[0] != myhs.house_state[0]
        ):
            myhs.house_state = myhs.new_house_state
            house_state = new_house_state
            lcd_message(myhs.house_state[0], myhs.house_state[1])
    elif temp_message_thread is None and myhs.new_house_state[0] != myhs.house_state[0]:
        myhs.house_state = myhs.new_house_state
        house_state = new_house_state
        lcd_message(myhs.house_state[0], myhs.house_state[1])
```

Aquí se definió los mensajes de temperatura del sensor que se verán en la LCD

```python
def lcd_temp_message(state: list, line1="", line2=""):
    lcd.clear()
    lcd.move_to(0, 0)
    lcd.putstr(line1)
    lcd.move_to(0, 1)
    lcd.putstr(line2)

    sleep(5)

    lcd.clear()
    lcd.move_to(0, 0)
    lcd.putstr(state[0])
    lcd.move_to(0, 1)
    lcd.putstr(state[1])
    sleep(0.5)
```

LIBRERÍAS UTILIZADAS

Estas son las librerías que se utilizaron en el desarrollo del programa, en donde se utilizo la librería de lcd_api para la pantalla LCD, una librería para comunicar la LCD con I2C, la librería spidev para la comunicación SPI, board para obtener los pines físicos de la rapsberry pi, una librería para los hilos, una librería para la fecha y hora. Y la librería time que permitirá las pausas entre ejecuciones.

```python
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
from states import HouseStates

import spidev
import adafruit_dht
import board
import digitalio
from adafruit_servokit import ServoKit

import threading
from datetime import datetime
from time import sleep
```

DEFINICION DE PINES

Aquí se definió los pines utilizados en todo el funcionamiento, los pines de los sensores como de la casa, los botones para encender las luces, los botones de la contraseña, la alarma, los rociadores, las luces, así como para el sensor de luz y temperatura y el botón del aire acondicionado.

```
30  # Sensor de temp y humedad
31  DHT11_DEV = adafruit_dht.DHT11(board.D21)
32
33  # Configuraciones de pines usando digitalio
34  LUCES_CASA = [digitalio.DigitalInOut(board.D13)]
35  SENSOR_LUZ = digitalio.DigitalInOut(board.D6)
36  ENCENDER_LUCES = digitalio.DigitalInOut(board.D16)
37  BOTON_AIRE = digitalio.DigitalInOut(board.D17)
38  ALARMA = digitalio.DigitalInOut(board.D12)
39  ROCIADORES = digitalio.DigitalInOut(board.D21)
40  BOTON_RIEGO = digitalio.DigitalInOut(board.D19)
41  REGADORES = digitalio.DigitalInOut(board.D20)
42  BOTON_SECCION = digitalio.DigitalInOut(board.D27)
43  BOTON_ASTERISCO = digitalio.DigitalInOut(board.D24)
44  BOTON_GRUPO = digitalio.DigitalInOut(board.D4)
45  BOTON_ENTER = digitalio.DigitalInOut(board.D5)
```

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.

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

LIBRERÍA THREADING

es una parte del módulo estándar de Python y permite la ejecución simultánea de código a través de hilos. Esta capacidad es útil para realizar múltiples tareas al mismo tiempo en un programa, como leer datos de un sensor mientras se actualiza una pantalla o se ejecuta otro código en segundo plano.

Link: https://docs.python.org/3/library/threading.html

LIBRERÍA TIME

módulo estándar de Python y ofrece varias funciones relacionadas con el manejo de tiempo. Una de las funciones más comunes es sleep(), que permite pausar la ejecución del programa por un número específico de segundos. También incluye funciones para medir el tiempo transcurrido, obtener la hora actual, y realizar conversiones entre diferentes formatos de tiempo.

Link: https://docs.python.org/3/library/time.html

LIBRERÍA BOARD

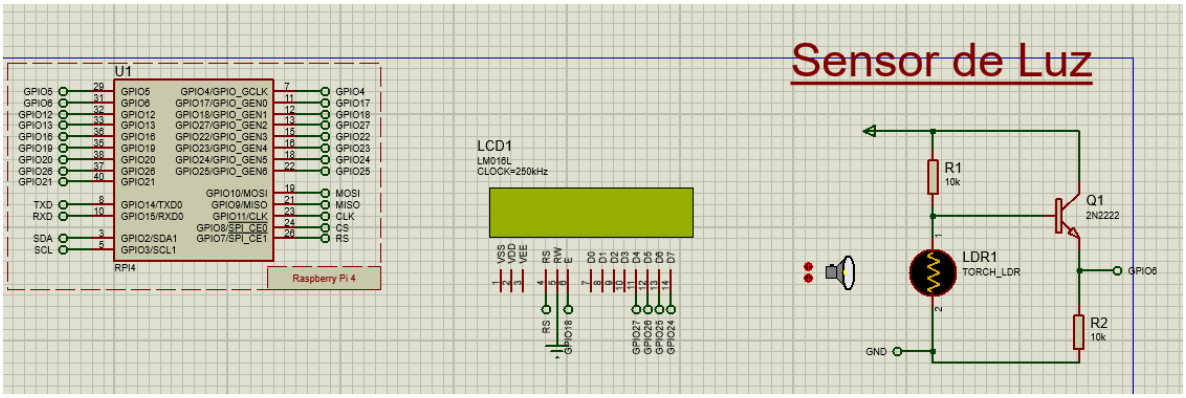
es parte del ecosistema de Adafruit y proporciona una abstracción sobre los pines físicos de plataformas como la Raspberry Pi. Esta librería facilita la identificación y uso de los pines GPIO y otros pines de comunicación (como I2C, SPI, etc.) al proporcionar nombres predefinidos y una interfaz sencilla para acceder a ellos. Es comúnmente utilizada junto con otras librerías de Adafruit para simplificar la programación de proyectos de hardware.

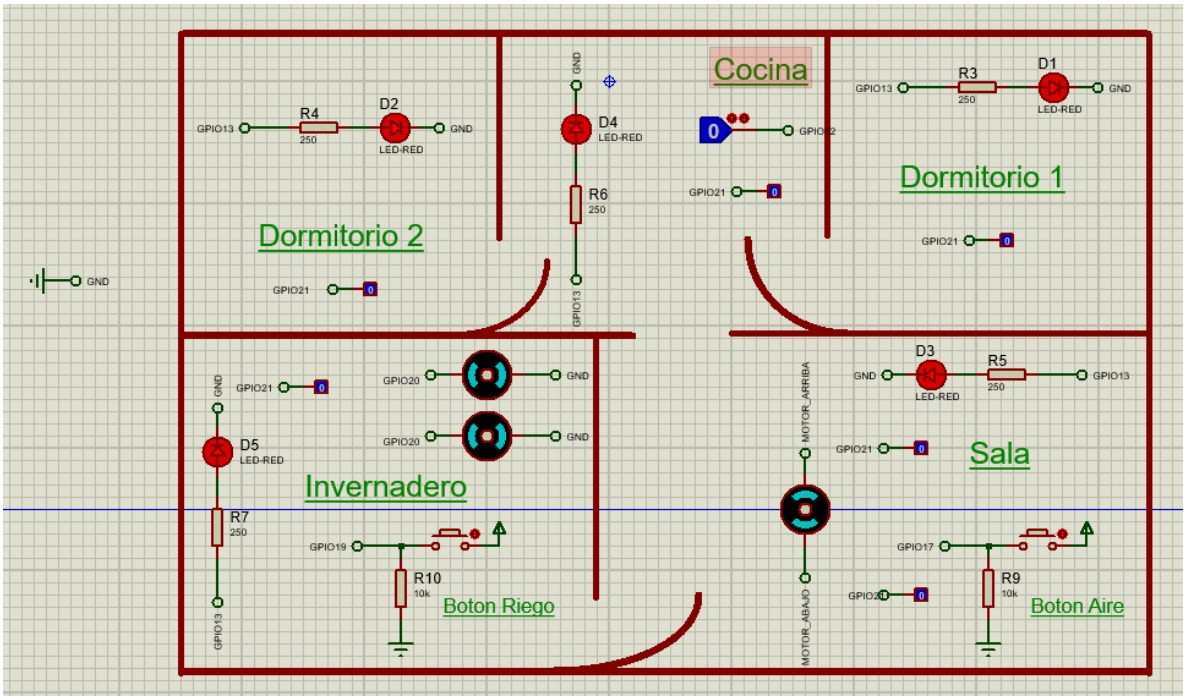
Link: https://docs.circuitpython.org/en/latest/shared-bindings/board/

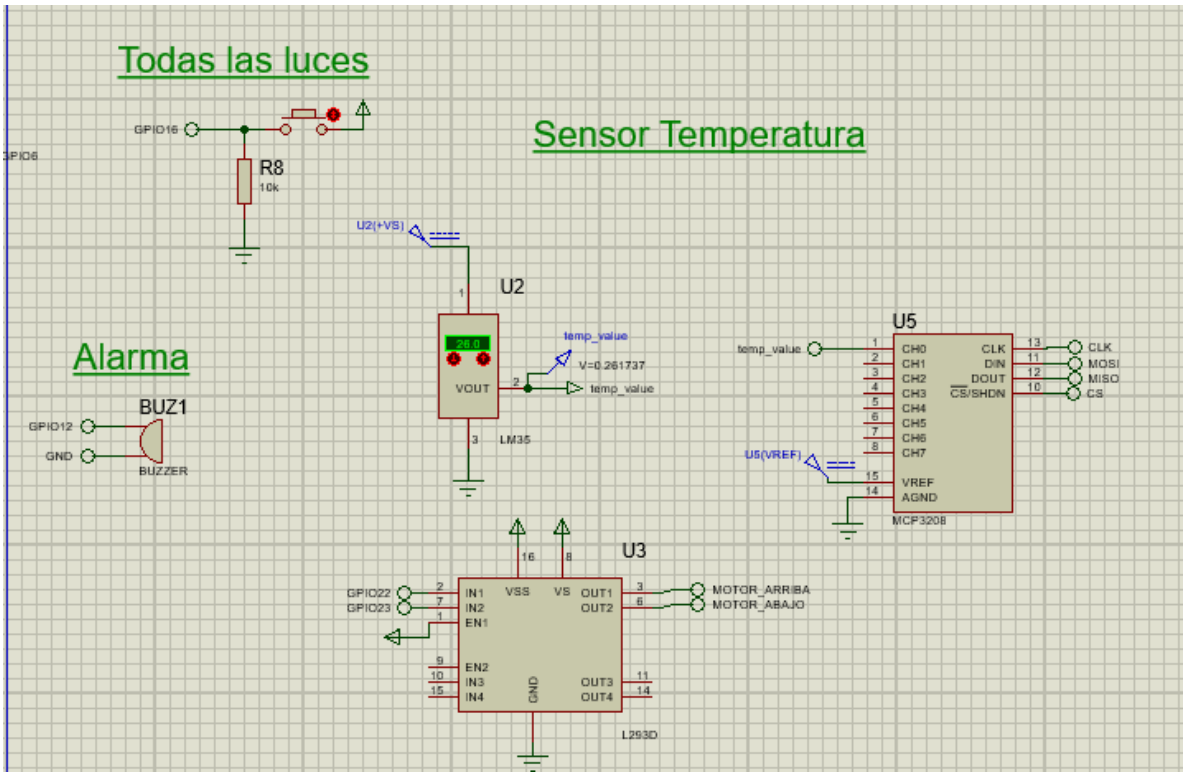
COSTO DEL PROTOTIPO

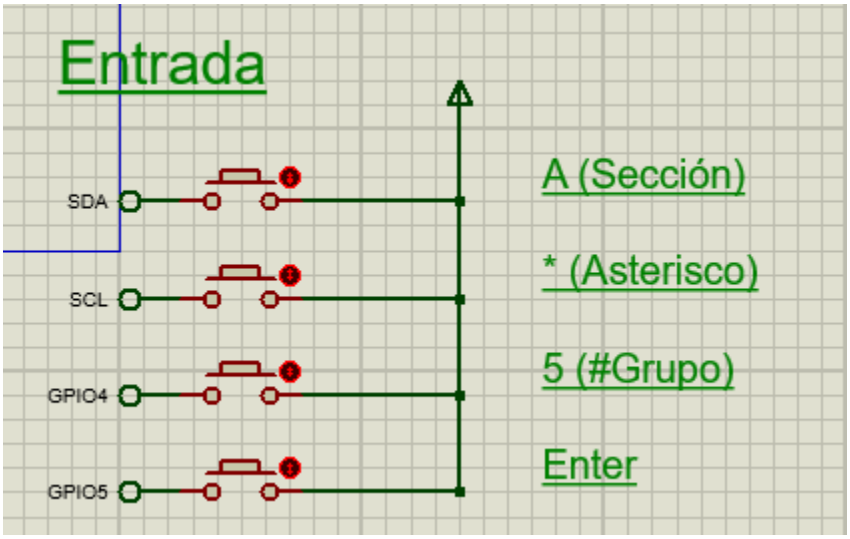
COMPONENTE	COSTO
MAQUETA	**Q250.00**
RAPSBERRY PI 4	**Q550.00**
SENSOR DE LLAMA	**Q60.00**
SENSOR DE LUZ	**Q46.00**
VENTILADORES	**Q60.00**
PANTALLA LCD 16X2	**Q40.00**
LEDS DE COLORES	**Q40.00**
PUSH BUTTONS	**Q15.00**
RESISTENCIAS(VARIAS)	**Q25.00**
CONVERSOR LOGICO	**Q29.00**
BUZZER	**Q10.00**
PROTOBOARDS	**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

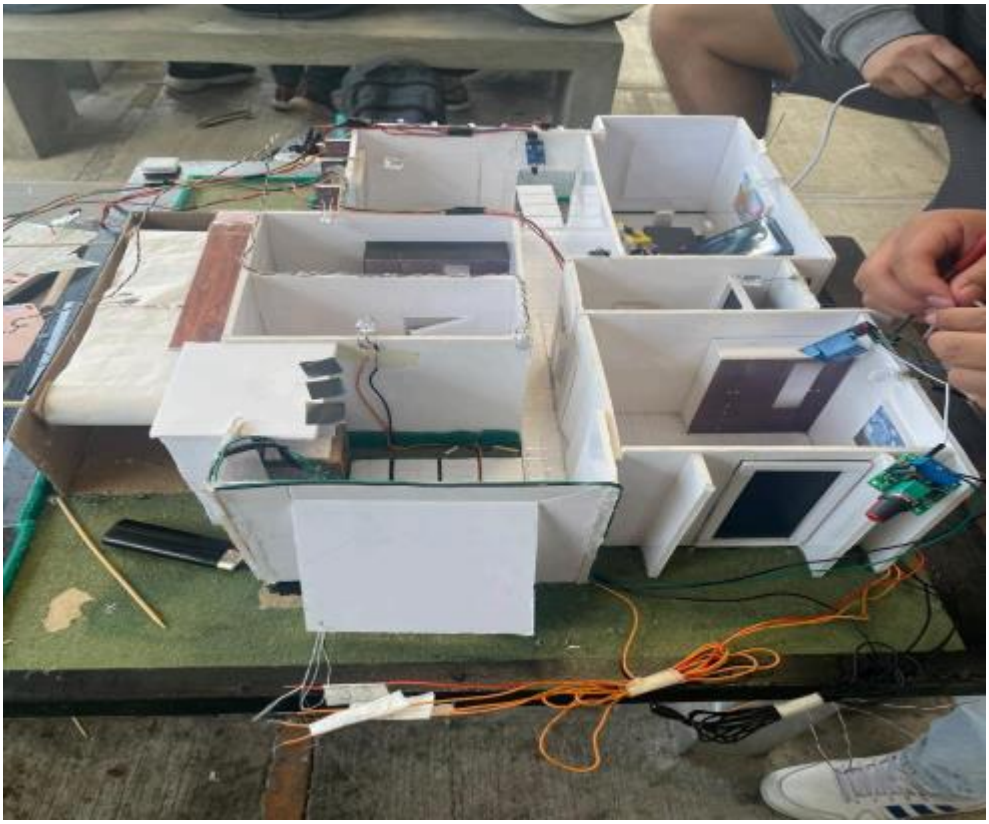





Todas las luces
GPIO16
R8
10k
Sensor Temperatura
U2(+VS)
U2
temp_value
V=0.261737
temp_value
VOUT
LM35
Alarma
BUZ1
GPIO12
GND
BUZZER
U5
temp_value
CH0
CH1
CH2
CH3
CH4
CH5
CH6
CH7
VREF
AGND
CLK
DIN
DOUT
CS/SHDN
U5(VREF)
MCP3208
CLK
MOSI
MISO
CS
U3
GPIO22
GPIO23
IN1
IN2
EN1
EN2
IN3
IN4
VSS
VS
GND
OUT1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


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

DISEÑO DE LA MAQUETA EN FISICO



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.

Seguridad: Implementar medidas de seguridad en el software para evitar comportamientos inesperados o peligrosos en caso de fallas en los sensores o la alimentación.

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.

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.

Protección contra la humedad: Si el prototipo se instalará en un ambiente húmedo, proteger los componentes electrónicos con recubrimientos o carcasas impermeables.