

Proyecto Final.

Fundamentos de Sistemas Embebidos

Autor: Ramirez Fierro Salma Arely.

Entrega: Lunes 16 de Agosto, 2021

Objetivo.

El alumno implementará un concentrador que permita monitorear y administrar remotamente vía navegador los dispositivos inteligentes domésticos vinculados.

Introducción.

Cuando hablamos de un concentrador Smart Home nos referimos a un comando central, que permite monitorear y administrar remotamente vía navegador los dispositivos inteligentes domésticos vinculados.

El presente sistema embebido busca realizar las siguientes funciones:

- Desplegado de cámaras de vigilancia conectadas 1
- Encendido y apagado de luces (al menos una)
- Atenuado de luces (al menos una)
- Detección de timbre de puerta (al menos una)
- Apertura remota de la puerta de la cochera
- Programado de encendido y apagado de luces e interruptores
- El sistema embebido incorporará un servidor web el control remoto (local vía IP) de las funciones descritas anteriormente.

Antecedentes Teóricos.

Entre los conceptos básicos que tenemos que tener contemplados son en primer lugar lo que es un sistema embebido, hablamos de un “sistema de computación” diseñado para realizar funciones en específicas.



Por otro lado, una parte fundamental para la realización del proyecto es un *Servidor web*, este es aquel que le permite a un usuario visualizar una página web en su navegador. Para el funcionamiento correcto de un servidor web necesitamos un cliente web que realice una petición http o https a través de un navegador como Chrome, Firefox o Safari y un servidor donde esté almacenada la información.

Imagen 1. Servidor web.

Máquina Virtual.

Las máquinas virtuales son ordenadores de software que proporcionan la misma funcionalidad que los ordenadores físicos. Ejecutan aplicaciones y un sistema operativo. Los archivos más importantes que componen una máquina virtual son un archivo de registro, un archivo de configuración de NVRAM, un archivo de disco virtual y un archivo de configuración.

Simulador de Raspberry.

El término correcto sería emulador, por ejemplo, VirtualBox es un reconocido programa de virtualización (y emulación) de hardware, y es uno de los mejores emuladores de Raspberry Pi para Windows (es el que utilizamos a lo largo del proyecto) ofrece la oportunidad de ejecutar el sistema operativo Raspberry Pi en su PC.

Lista de Materiales o Componentes Electrónicos.

- Plataforma: Máquina virtual con sistema operativo Raspbian.
- Intérprete de Python 3.5 instalado.
- Código del Programa 2.

Descripción del funcionamiento de la tarjeta controladora. Así como de los componentes relevantes de los mismos

Raspberry Pi

La Raspberry Pi es una computadora de bajo costo y con un tamaño compacto. Es un pequeño computador que corre un sistema operativo Linux capaz de permitir explorar la computación y aprender a programar lenguajes como Scratch y Python. La placa más popular es la Raspberry Pi 3 B+, el funcionamiento de esta placa es el siguiente:

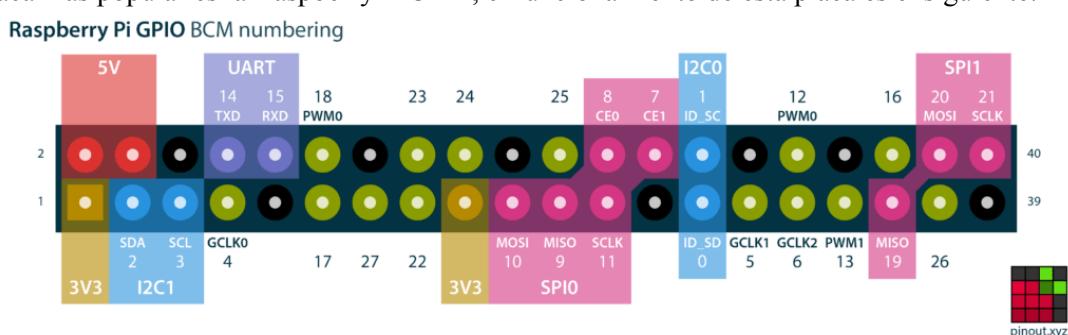


Imagen 2. Pinout GPIO Raspberry Pi – Cortesía de pinout.xyz

La Raspberry Pi 3 B+ cuenta con un GPIO de 40 pines, este último es el que permite el contacto de ella con el mundo exterior, raspberry trabaja con el nivel de 3.3V y si se requiere conectar sensores de 5V se necesita un conversor de niveles lógicos como un MC100582. Además cuenta con puertos de comunicación I2C, SPI y UART.

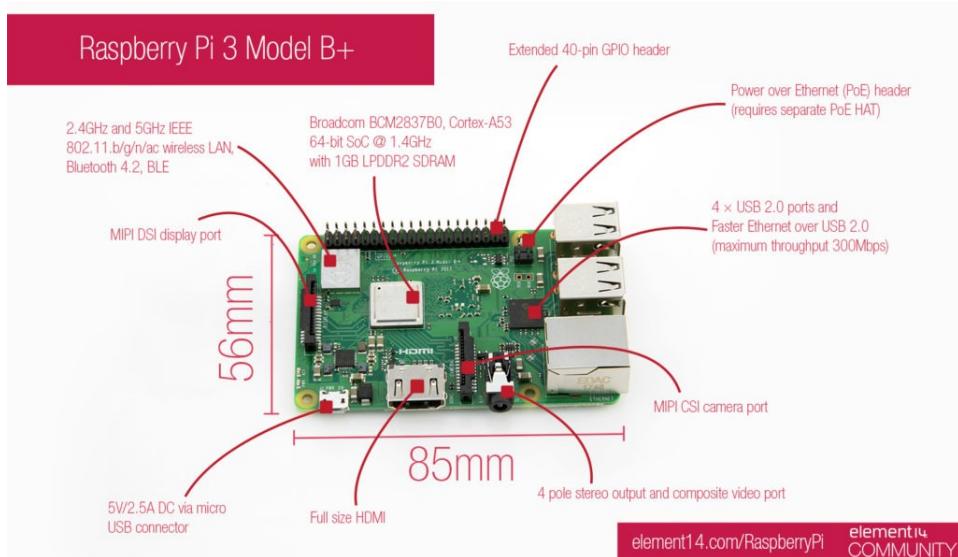


Imagen 3. Elementos de Raspberry Pi 3 B+ – Cortesía de element14.com

La Raspberry Pi 3B+ además cuenta con conexiones tradicionales como son:

- Puertos USB.
- Conector de red ethernet.
- Jack de 3.5mm.
- Puerto HDMI.
- Puerto para memoria microSD

- Conector micro-usb para la alimentación.
- Puerto especial para la cámara y la pantalla.

Software de Raspberry Pi.

Sistema Operativo Raspbian

La versatilidad de software es uno de los factores que más ha impulsado a la Raspberry Pi al éxito, pues puede lograr múltiples usos. Esta gran adaptabilidad viene dado a que la Raspberry Pi trabaja bajo la plataforma de software libre Linux, más específicamente Raspbian, una modificación del sistema operativo Debian.

Descripción del funcionamiento de los componentes electrónicos relevantes (circuitos integrados, encapsulados, controladores, etcétera)

Descargue el simulador de

<https://github.com/SalmaFierro23/Proyecto-Final-Sistemas-Embebidos.git>

Escriba `git clone https://github.com/SalmaFierro23/Proyecto-Final-Sistemas-Embebidos.git`

`cd src`

Si es necesario instale todas las dependencias requeridas.

`sudo apt install python3-tk`

`pip install --user -r requirements.txt`

Compruebe que el simulador se está ejecutando con

`./blink.py`

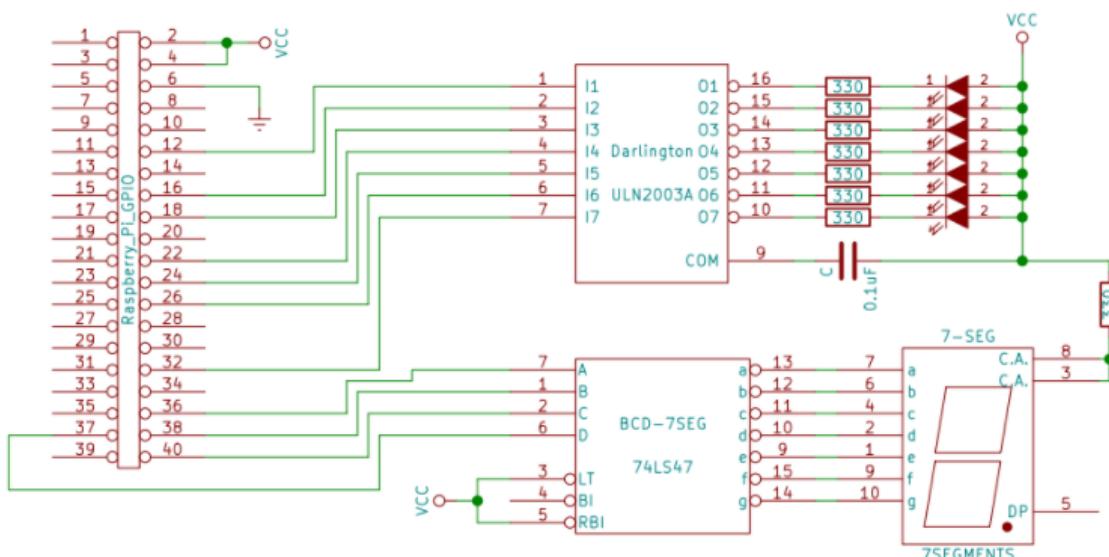


Imagen 4. Simulador implementado en el simulador.

En el archivo webserver.py coloque la ip local del equipo, esto con la finalidad de que el simulador pueda acceder al servidor web y controlar todos los elementos del puerto GPIO.

`ip-4 address grep inet`

```

27 # address = "localhost"
28 address = "192.168.1.83">//modificado
29 # Puerto en el cual el servidor estará atendiendo
30 # El default de un servidor web en producción debería ser 80
31 port = 8080

```

Imagen 5. Código webserver.py

Led parpadeante

Según el Código 1 muestra cómo se haría parpadear un LED mediante tiempos de espera o sleeps utilizando la Raspberry Pi.

```

1 # Import Raspberry Pi's GPIO control library
2 import RPi.GPIO as GPIO
3 # Imports sleep function
4 from time import sleep
5 # Initializes virtual board (comment out for hardware deploy)
6 import virtualboard
7
8 # Set up Rpi.GPIO library to use physical pin numbers
9 GPIO.setmode(GPIO.BOARD)
10 # Set up pin no. 32 as output and default it to low
11 GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)
12
13 # Blink the led
14 while True: # Forever
15     sleep(0.5) # Wait 500ms
16     GPIO.output(32, GPIO.HIGH) # Turn led on
17     sleep(0.5) # Espera 500ms
18     GPIO.output(32, GPIO.LOW) # Turn led off

```

Código 1: blink.py

Display de siete segmentos

Código 2 muestra cómo se operaría un display de siete segmentos mediante una controladora TTL 74LS47 utilizando la Raspberry Pi.

```

1 # Import Raspberry Pi's GPIO control library
2 import RPi.GPIO as GPIO
3 # Imports sleep function
4 from time import sleep
5 # Initializes virtual board (comment for hardware deploy)
6 import virtualboard
7
8 # Set up Rpi.GPIO library to use physical pin numbers
9 GPIO.setmode(GPIO.BOARD)
10 # Set up pins 36, 38, 40 and 37 as output and default them to low
11 GPIO.setup(36, GPIO.OUT, initial=GPIO.LOW)
12 GPIO.setup(38, GPIO.OUT, initial=GPIO.LOW)
13 GPIO.setup(40, GPIO.OUT, initial=GPIO.LOW)
14 GPIO.setup(37, GPIO.OUT, initial=GPIO.LOW)
15
16 def bcd7(num):
17     """Converts num to a BCD representation"""
18     GPIO.output(36, GPIO.HIGH if (num & 0x00000001) > 0 else GPIO.LOW )
19     GPIO.output(38, GPIO.HIGH if (num & 0x00000002) > 0 else GPIO.LOW )

```

```

20 GPIO.output(40, GPIO.HIGH if (num & 0x00000004) > 0 else GPIO.LOW )
21 GPIO.output(37, GPIO.HIGH if (num & 0x00000008) > 0 else GPIO.LOW )
22
23 # Request a number and send it to the display
24 flag = True
25 while flag:
26 try:
27 num = int(input("Enter int between 0 and 15: "))
28 bcd7(num)
29 except:
30 flag = False
31
32 # Reset all ports to its default state (inputs)
33 GPIO.cleanup()

```

Código 2: bcd.py

Código web_server.py

Dirección IP del sistema anfitrión del servidor web y el puerto por el que el servidor web estará recibiendo solicitudes http.

address = "192.168.1.64"

port = 8080

Class WebServer:

Método que sirve cualquier archivo encontrado en el servidor.

```

class WebServer(BaseHTTPRequestHandler):
    def _serve_file(self, rel_path):
        if not os.path.isfile(rel_path):
            self.send_error(404)
            return
        self.send_response(200)
        mime = magic.Magic(mime=True)
        self.send_header("Content-type", mime.from_file(rel_path))
        self.end_headers()
        with open(rel_path, 'rb') as file:
            self.wfile.write(file.read())

```

Imagen 6. Método Serve Files.

Método que sirve el archivo de interfaz a un usuario. Es decir el que deja visualizar la pagina donde tenemos implementada la simulación de la casa.

```

def _serve_ui_file(self):
    if not os.path.isfile("home.html"):
        err = "home.html not found."
        self.wfile.write(bytes(err, "utf-8"))
        print(err)
        return
    try:
        with open("home.html", "r") as f:
            content = "\n".join(f.readlines())
    except:
        content = "Error reading home.html"
        self.wfile.write(bytes(content, "utf-8"))

def _parse_post(self, json_obj):
    if not 'action' in json_obj or not 'value' in json_obj:
        return
    switcher = {
        'casa' : casa

    }
    func = switcher.get(json_obj['action'], None)
    if func:
        print('\tCall{}({})'.format(func, json_obj['value']))
        func(json_obj['value'])

```

Imagen 7. Método Serve Ui Files y Parse Post

Funciones relevantes.

Con do_Get controlamos las solicitudes recibidas, las páginas.

```

def do_GET(self):
    # Revisamos si se accede a la raiz.
    # En ese caso se responde con la interfaz por defecto
    if self.path == '/':
        # 200 es el código de respuesta satisfactorio (OK)
        # de una solicitud
        self.send_response(200)
        # La cabecera HTTP siempre debe contener el tipo de datos mime
        # del contenido con el que responde el servidor
        self.send_header("Content-type", "text/html")
        # Fin de cabecera
        self.end_headers()
        # Por simplicidad, se devuelve como respuesta el contenido del
        # archivo html con el código de la página de interfaz de usuario
        self._serve_ui_file()
    # En caso contrario, se verifica que el archivo exista y se sirve
    else:
        self._serve_file(self.path[1:])

```

Imagen 7. Función do_Get.

Con do_Post se gestionan los comandos para Raspberry Pi.

```
def do_POST(self):
    # Primero se obtiene la longitud de la cadena de datos recibida
    content_length = int(self.headers.get('Content-Length'))
    if content_length < 1:
        return
    # Despues se lee toda la cadena de datos
    post_data = self.rfile.read(content_length)
    # Finalmente, se decodifica el objeto JSON y se procesan los datos.
    # Se descartan cadenas de datos mal formados
    try:
        jobj = json.loads(post_data.decode("utf-8"))
        self._parse_post(jobj)
    except:
        print(sys.exc_info())
        print("Datos POST no reconocidos")
```

Imagen 8. Función post_data.

En la función principal **main** se inicializa una nueva instancia de HTTP Server y se mantiene al servidor web ejecutándose en segundo plano.

Código led_manager.py

Configuramos pines de salida y de bajada.

```
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(26, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(24, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(22, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(18, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(16, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(12, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(10, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(36, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(38, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(40, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(37, GPIO.OUT, initial=GPIO.LOW)
```

Imagen 9. Inicialización de pines.

Declaramos PWM en los pines que utilizaremos.

```
pwm8 = GPIO.PWM(32, 1)
pwm7 = GPIO.PWM(26, 1)
pwm6 = GPIO.PWM(24, 1)
pwm5 = GPIO.PWM(22, 1)
pwm4 = GPIO.PWM(18, 1)
pwm3 = GPIO.PWM(16, 1)
pwm2 = GPIO.PWM(12, 1)
pwm1 = GPIO.PWM(10, 1)
```

Imagen 10. Asignamos pines a los PWM.

Función que pone fin a la operación de los pwm.

```
def pwm_off():
    pwm1.stop()
    pwm2.stop()
    pwm3.stop()
    pwm4.stop()
    pwm5.stop()
    pwm6.stop()
    pwm7.stop()
    pwm8.stop()
pass
```

Imagen 11.

Función que controla los focos de la casa.

```
def ledsoff():
    GPIO.output(10, GPIO.LOW)
    GPIO.output(12, GPIO.LOW)
    GPIO.output(16, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(22, GPIO.LOW)
    GPIO.output(24, GPIO.LOW)
    GPIO.output(26, GPIO.LOW)
    GPIO.output(32, GPIO.LOW)
    GPIO.output(36, GPIO.LOW)
    GPIO.output(38, GPIO.LOW)
    GPIO.output(40, GPIO.LOW)
    GPIO.output(37, GPIO.LOW)
pass
```

Imagen 12. Función que detiene la operación de los focos.

Método que comienza el modo de Casa Inteligente realizando quince funciones diferentes.

```
def casa(type='inteligente'):
    switcher = {
        'luces'      : _casa_luces,
        'apagar_luces' : _casa_apagar_luces,
        'camaras'   : _casa_camaras,
        'camaras_apagadas' : _casa_camaras_apagadas,
        'timbre'     : _casa_timbre,
        'atenuar_75'  : _casa_atenuar_75,
        'atenuar_50'  : _casa_atenuar_50,
        'atenuar_25'  : _casa_atenuar_25,
        'atenuar_10'  : _casa_atenuar_10,
        'abrir_cochera' : _casa_abrir_cochera,
        'cerrar_cochera' : _casa_cerrar_cochera,
        'programar_encendido_luces6' :
        'programar_encendido_luces12' :
        'programar_apagado_luces6' :
        'programar_apagado_luces12' :

    }
    func = switcher.get(type, None)
```

Imagen 13. Type toma 15 valores de la casa inteligente.

Función para apagar o encender las luces de la casa.

```
def _casa_luces():
    ledsoff()
    pwm_off()

    #while True: # Forever
    #    ... # Wait 500ms
    GPIO.output(32, GPIO.HIGH) # Turn led on
    GPIO.output(26, GPIO.HIGH)
    GPIO.output(24, GPIO.HIGH)
    GPIO.output(22, GPIO.HIGH)
    GPIO.output(18, GPIO.HIGH)
    GPIO.output(16, GPIO.HIGH)
    GPIO.output(12, GPIO.HIGH)
    GPIO.output(10, GPIO.HIGH)

    print("Luces encendidas")

    pass

def _casa_apagar_luces():
    ledsoff()
    pwm_off()

    GPIO.output(32, GPIO.LOW) # Turn led off
    GPIO.output(26, GPIO.LOW)
    GPIO.output(24, GPIO.LOW)
    GPIO.output(22, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(16, GPIO.LOW)
    GPIO.output(12, GPIO.LOW)
    GPIO.output(10, GPIO.LOW)

    print("Luces Apagadas")
```

Imagen 14. def_casa_luces y def_casa_apagar_luces

Función para apagar o encender las cámaras de vigilancia de la casa inteligente.

```
def _casa_camaras():
    ledsoff()

    pwm6.start(50) #Se activa sexta camara
    pwm5.start(50) #Se activa quinta camara
    pwm4.start(50) #Se activa cuarta camara
    pwm3.start(50) #Se activa tercera camara
    print("Desplegado de camaras de vigilancia")

    pass
```

Imagen 15. Función para activar cámaras de vigilancia.

```
def _casa_camaras_apagadas():
    ledsoff()
    pwm_off()

    print("Apagando camaras de vigilancia")

    pass
```

Imagen 16. Función para desactivar cámaras de vigilancia.

Función para activar el timbre de la casa inteligente.

```
def _casa_timbre():
    ledsoff()
    pwm_off()
    GPIO.output(10, GPIO.HIGH)
    print("Timbre ON")
    sleep(2)

    GPIO.output(10, GPIO.LOW)
    print("Timbre OFF")
    sleep(1)
```

Imagen 16. Función para activar el timbre.

Logramos conseguir que los focos tengan una atenuación que va desde un 75% a un 10%.

```
def _casa_atenuar_75():
    pwm_off()
    ledsoff()
    pwm8.start(75)
    pwm7.start(75)
    pwm6.start(75)
    pwm5.start(75)
    pwm4.start(75)
    pwm3.start(75)
    pwm2.start(75)
    pwm1.start(75)
    print("Atenuando focos al 75%")

    pass
```

Imagen 16. Ejemplo de atenuación de focos a un 75% de la casa inteligente.

Función para abrir y cerrar la cochera.

```
def _casa_abrir_cochera():
    pwm_off()
    ledsoff()

    sleep(0.5)                      # Wait 500ms
    GPIO.output(10, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(10, GPIO.LOW)         # Turn led off
    GPIO.output(12, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(12, GPIO.LOW)         # Turn led off
    GPIO.output(16, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(16, GPIO.LOW)         # Turn led off
    GPIO.output(18, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(18, GPIO.LOW)         # Turn led off
    GPIO.output(22, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(22, GPIO.LOW)         # Turn led off
    GPIO.output(24, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(24, GPIO.LOW)         # Turn led off
    GPIO.output(26, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(26, GPIO.LOW)         # Turn led off
    GPIO.output(32, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(32, GPIO.LOW)         # Turn led off
    print("Abriendo cochera%")

    pass
```

Imagen 17.Funció n para abrir cochera.

```
def _casa_cerrar_cochera():

    sleep(0.5)                      # Wait 500ms
    GPIO.output(32, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(32, GPIO.LOW)         # Turn led off
    GPIO.output(26, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(26, GPIO.LOW)         # Turn led off
    GPIO.output(24, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(24, GPIO.LOW)         # Turn led off
    GPIO.output(22, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(22, GPIO.LOW)         # Turn led off
    GPIO.output(18, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(18, GPIO.LOW)         # Turn led off
    GPIO.output(16, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(16, GPIO.LOW)         # Turn led off
    GPIO.output(12, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(12, GPIO.LOW)         # Turn led off
    GPIO.output(10, GPIO.HIGH)        # Turn led on
    sleep(0.5)
    GPIO.output(10, GPIO.LOW)         # Turn led off
    print("Cerrando cochera%")

    pass
```

Imagen 18.Funció n para cerrar la cochera.

Función que simula la programación del encendido y apagado de las luces de la casa inteligente. En el proyecto programamos 6 y 12 segundos que solo son representaciones de los valores reales en nuestra casa 6 horas y 12 horas.

```
def _casa_programar_apagado_luces6():

    pwm_off()

    sleep(6)                      # Wait 500ms
    GPIO.output(32, GPIO.LOW)        # Turn led off
    GPIO.output(26, GPIO.LOW)
    GPIO.output(24, GPIO.LOW)
    GPIO.output(22, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(16, GPIO.LOW)
    GPIO.output(12, GPIO.LOW)
    GPIO.output(10, GPIO.LOW)
    print("Apagando luces en 6 segundos%")
    pass
```

Imagen 17.Funció n para apagar luces en 6 horas y para prenderlas en 6 horas.

```
def _casa_programar_encendido_luces6():

    ledsoff()
    pwm_off()

    sleep(6)                      # Wait 500ms
    GPIO.output(32, GPIO.HIGH)        # Turn led off
    GPIO.output(26, GPIO.HIGH)
    GPIO.output(24, GPIO.HIGH)
    GPIO.output(22, GPIO.HIGH)
    GPIO.output(18, GPIO.HIGH)
    GPIO.output(16, GPIO.HIGH)
    GPIO.output(12, GPIO.HIGH)
    GPIO.output(10, GPIO.HIGH)
    print("Encendiendo luces en 6 segundos%")
    pass
```

Código virtualboard.py

Imagen 18. Apagando la interfaz de usuario.

```
def exit_handler():
    print('Shutting down board GUI')
    if _async_board_thread:
        _async_board_thread.join()

def _async_board_worker():
    global _async_board_thread
    board = Board()
    board.connect(GPIO._io_pins)
    try:
        mainloop()
    except:
        pass
    _async_board_thread = None

_register(exit_handler)
_async_board_thread.start()
```

Integración de los componentes de software o módulos en una solución de software

Función para apagar o encender las luces de la casa.

Resultados.

Luces , cámaras de Vigilancia y atenuación de luces.

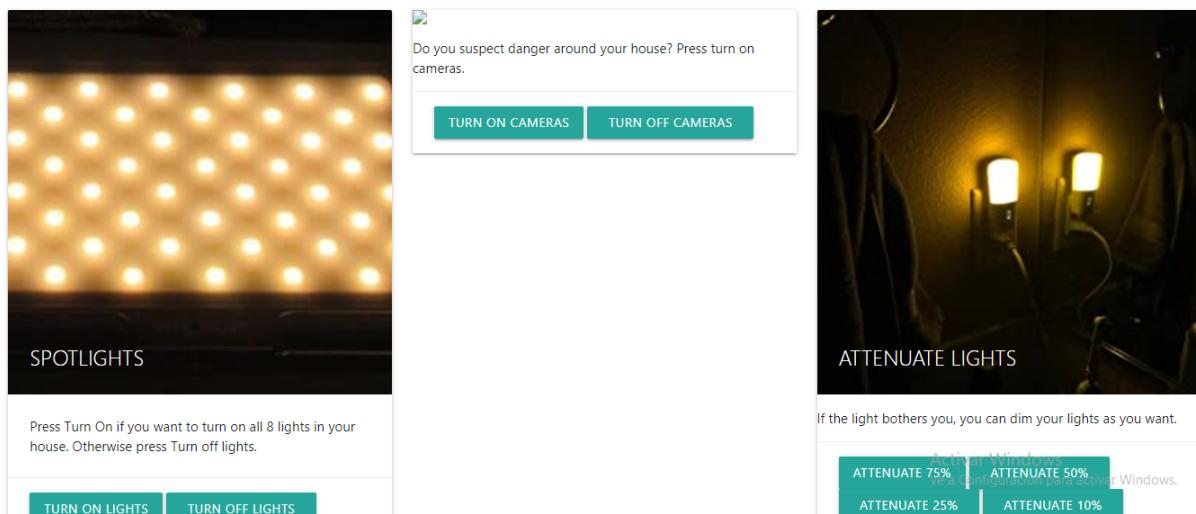


Imagen 19. Interfaz de usuario.

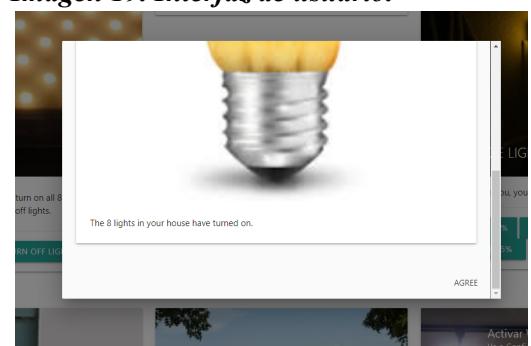


Imagen 20. Luz encendida.

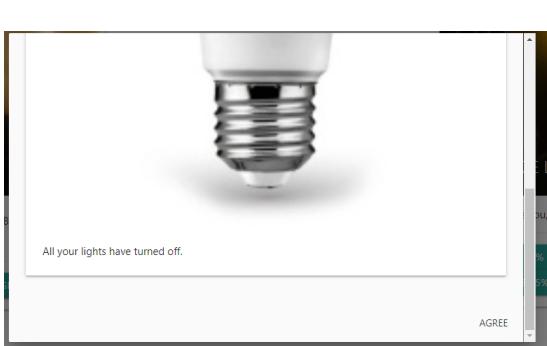


Imagen 21. Luz apagada.

Atenuación que va desde un 75% a un 10%.



If the light bothers you, you can dim your lights as you want.

ATTENUATE 75%

ATTENUATE 50%

ATTENUATE 25%

ATTENUATE 10%

Imagen 22. Ejemplo de atenuación de focos a un 75% de la casa inteligente.

Timbre, cochera y función que mide el tiempo para prender o apagar luces



If you want to enter the house, ring the bell.

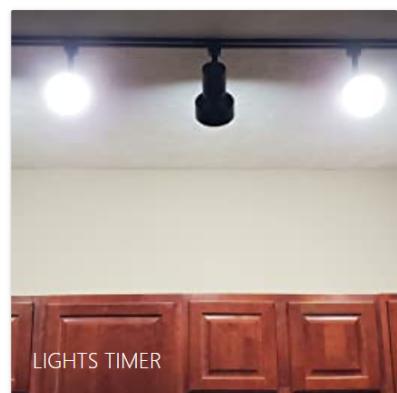
RING THE DOORBELL



Press open garage in case you want to put your car in or out.

OPEN GARAGE

CLOSE GARAGE



Set the time you want to turn on or off your lights.

TURN ON LIGHTS IN 60 MIN

Activar Windows

TURN ON LIGHTS IN 120 MIN para activar Windows.

TURN OFF LIGHTS IN 60 MIN

Imagen 23. Simulación de Casa Inteligente.

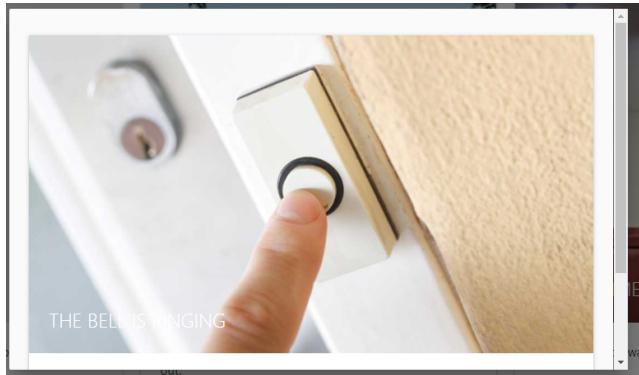


Imagen 24. Timbre activado .

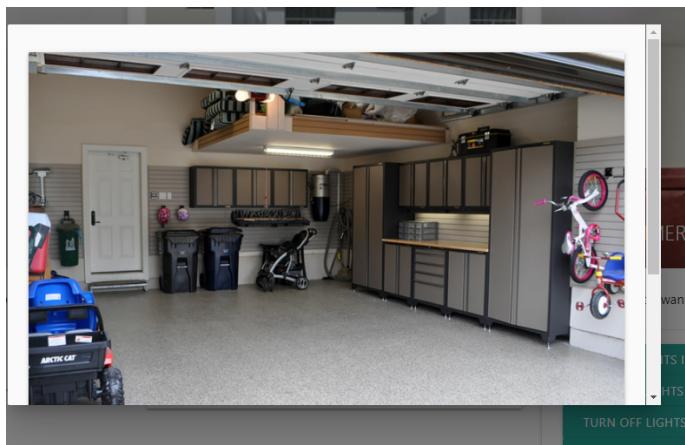


Imagen 25. Cochera abierta .

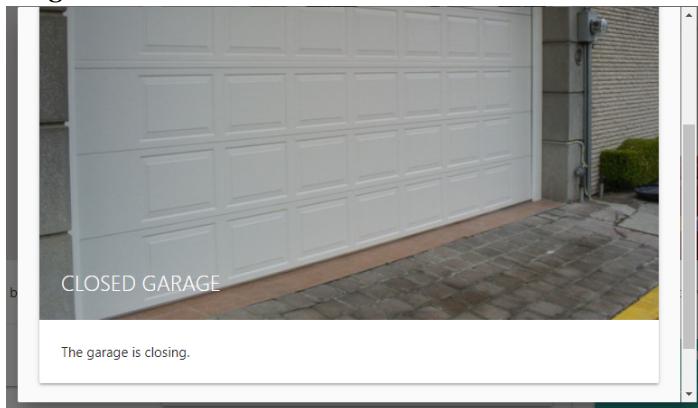


Imagen 26. Cochera Cerrada.



Imagen 27. Encendido y apagado de luces.

Link de Video :<https://www.youtube.com/watch?v=5QWoXep6CeU>

Link de Github:<https://github.com/SalmaFierro23/Proyecto-Final-Sistemas-Embebidos.git>

Conclusiones

El objetivo que nosotros nos planteamos con este proyecto fue implementar un concentrador que permitiera monitorear y administrar remotamente vía navegador los dispositivos inteligentes domésticos vinculados. Al no contar con las condiciones óptimas para desarrollar el proyecto con el hardware, fue necesario que recurriremos a la simulación de una raspberry pi, con una máquina virtual con sistema operativo Linux, fue de vital importancia contar con el programa dos propuesto por el profesor de la materia José Mauricio Matamoros de María y Campos para la resolución de nuestro proyecto pues con él comenzamos a desarrollar, logramos modificarlo de tal manera que se convirtió en el sistema embebido que necesitábamos es decir nuestra casa inteligente. En cada uno de los elementos que conforman la casa se necesito un análisis de cómo es que se podría representar.

Cuestionario

Bibliografía de consulta

¿Qué es un servidor web y para qué sirve? - Webempresa. (s/f). Webempresa.mx. Recuperado el 14 de agosto de 2021, de <https://www.webempresa.mx/hosting/que-es-servidor-web.html>

(S/f). Org.mx. Recuperado el 14 de agosto de 2021, de

<https://profesionistas.org.mx/wp-content/uploads/2018/06/Comocrearunapaginaweb.png>

¿Qué es una máquina virtual? (2021, agosto 3). Vmware.com.

<https://www.vmware.com/latam/topics/glossary/content/virtual-machine.html>
paguayo. (2019, febrero 14).

¿Qué es Raspberry Pi? Raspberrypi.cl. <https://raspberrypi.cl/que-es-raspberry/>
paguayo. (2019b, febrero 14).

Software de Raspberry Pi. Raspberrypi.cl. <https://raspberrypi.cl/software-de-raspberry/>
Programa 2 -Control remoto de una simulación de puerto GPIO de la Raspberry Pi via WiFi y servidor web, Autor: José Mauricio Matamoros de María y Campos

