Práctica 1:
Instalación de Raspbian en la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a instalar un sistema operativo basado en Linux, como sistema operativo embebido, en una tarjeta microcontroladora.

2. Introducción

Raspbian es el sistema operativo más popular para Raspberry Pi, además de ser el único con soporte oficial. Raspbian es una distribución de Linux basada en Debian, optimizado para la Raspberry Pi y que permite a esta operar como una PC. La distro incorpora terminal y navegador web entre otros programas.

3. Instrucciones

Instalar Raspbian en la Raspberry Pi es sencillo. Basta con descargar Raspbian y grabar la imagen de disco en una tarjeta de memoria microSD, desde la cual arrancará el sistema operativo.

Para esta práctica se necesitará:

- Una tarjeta de memoria microSD de al menos 4 GB (se recomiendan 8GB)
- Una computadora capaz de leer y escribir tarjetas microSD (o bien un adaptador para la misma) y conexión a internet para descargar la imagen de Raspbian.
- Una Raspberry Pi 2B o posterior
- Un monitor con soporte para HDMI
- Un teclado USB
- Un mouse USB
- Una fuente de alimentación de 5V@1A con adaptador microUSB

Importante: Si no cuenta con monitor, teclado y mouse, aún es posible instalar Raspbian en la Raspberry Pi. Consulte el **??**.

3.1. Paso 1: Descargar Raspbian

Ingrese a `https://www.raspberrypi.org/downloads/raspbian/` y descargue alguna de las imágenes de Raspbian disponibles (véase Figura 1). Si se descargó un archivo Zip, habrá que descomprimirlo para extraer la imagen.

La versión a descargar dependerá de la capacidad de la memoria microSD y la cantidad de recursos de la tarjeta Raspbian. Para tarjetas de 4GB se aconseja el sabor *Raspbian Buster with desktop*, mientras que usuarios con tarjetas de sólo 512MB de RAM querrán instalar *Raspbian Buster Lite*.

Ligas de acceso rápido se proporcionan a continuación por conveniencia:

- Raspbian Buster with desktop and recommended software
- Raspbian Buster with desktop
- Raspbian Buster Lite

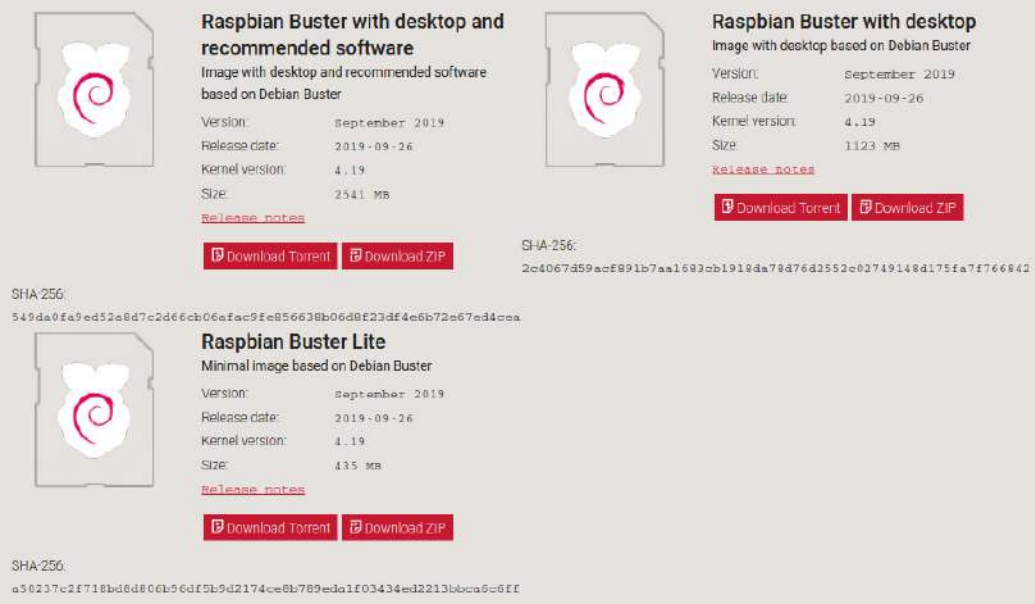


Figura 1: Versiones disponibles (o sabores) de Raspbian

3.2. Paso 2: Escribir imagen en la microSD

Si no lo ha hecho, introduzca la memoria microSD en la computadora. La memoria tendrá que formatearse, por lo que se aconseja respaldar la información.

Escribir la imagen de Raspbian en la microSD requiere de un programa externo según el sistema operativo.

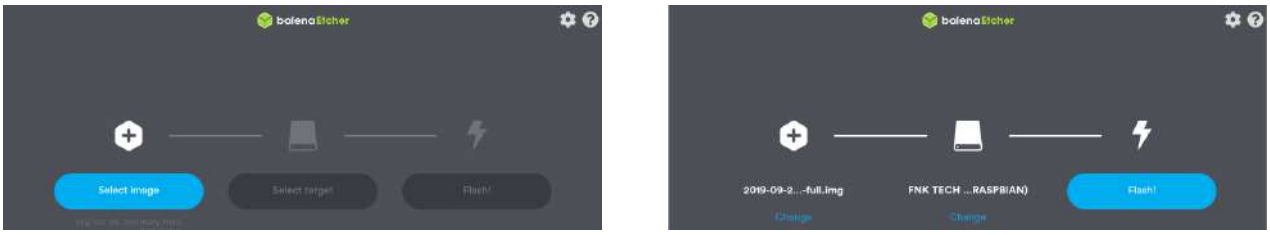
3.2.1. Escribir imagen usando Linux

Descargue Etcher en `~/Downloads`, descomprima y ejecute; por ejemplo

```
$ cd ~/Downloads
$ unzip balena-etcher-electron-1.5.75-linux-x64.zip
$ ./balenaEtcher-1.5.75-x64.AppImage
```

A continuación, siga los pasos de Etcher para grabar la imagen (véase Figura 2)

1. Seleccione la imagen IMG de Raspbian
2. Seleccione el medio en el cual se grabará la imagen de Raspbian (punto de montaje de la microSD)
3. De click en *Flash!* para empezar el proceso de grabado



(a) Selección de imagen a grabar

(b) Listo para grabar la imagen

Figura 2: Escritura de la imagen IMG de Raspbian en la microSD con Etcher

Cuando la escritura de datos en la microSD termine, notará que ésta ha sido reparticionada en `boot` (partición de arranque tipo *FAT32*) y `rootfs` (partición raíz tipo *EXT4*).

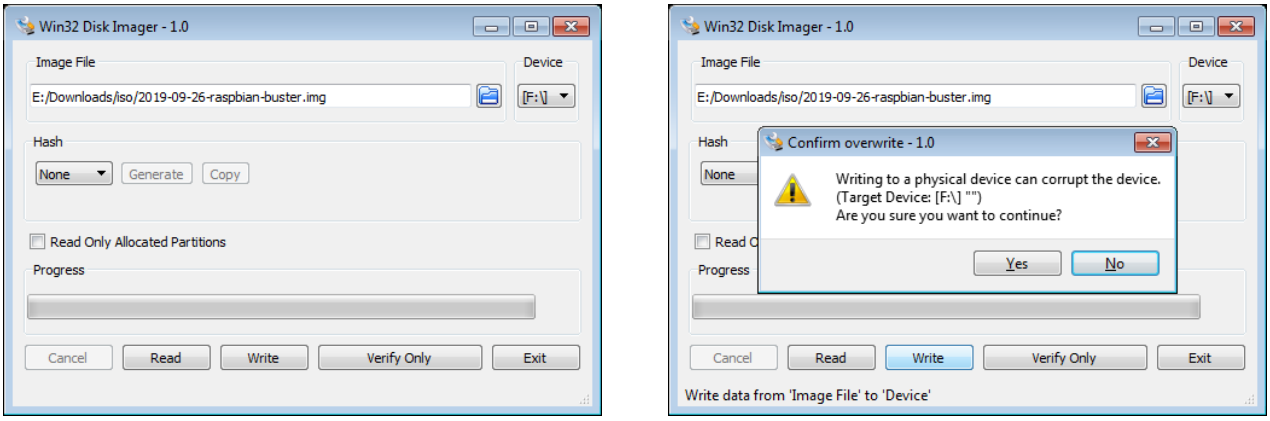
Si no cuenta con monitor, teclado o mouse, consulte el Apéndice A. De otro modo, desmonte la tarjeta microSD e insértela en la Raspberry Pi.

3.2.2. Escribir imagen usando Windows

Descargue e instale Win32 Disk Imager. Si no ha descmprimido la imagen de Raspbian, proceda a hacerlo con un programa que descomprima archivos Zip como 7zip (Windows 7 y posterior deberí soportar descompresión zip).

A continuación, siga los pasos de Win32 Disk Imager para grabar la imagen (véase Figura 3)

1. Seleccione la imagen IMG de Raspbian en *Image File*
2. Seleccione la unidad en la cual se grabará la imagen de Raspbian (letra asignada a la microSD) en *Device*
3. De click en *Write* para iniciar el proceso de escritura.
4. Win32 Disk Imager mostrará una advertencia, de click en *Yes* para continuar.



(a) Selección de imagen a grabar

(b) Listo para grabar la imagen

Figura 3: Escritura de la imagen IMG de Raspbian en la microSD con Win32 Disk Imager

Cuando la escritura de datos en la microSD termine, Windows asignará una letra al nuevo volúmen de datos de la microSD, llamada `boot`. Desmonte la tarjeta microSD e insértela en la Raspberry Pi.

3.3. Paso 3: Configurar Raspbian

Para configuar Raspbian, la Raspberry Pi deberá tener una tarjeta de memoria microSD con una imagen de Raspbian precargada y estar conectada a un monitor vía el puerto HDMI incorporado. Además, se precisa de un teclado y apuntador (mouse) USB para realizar el proceso de configuración. A continuación, conecte la Raspberry Pi y espere entre 1 y 3 minutos a que el sistema operativo cargue.

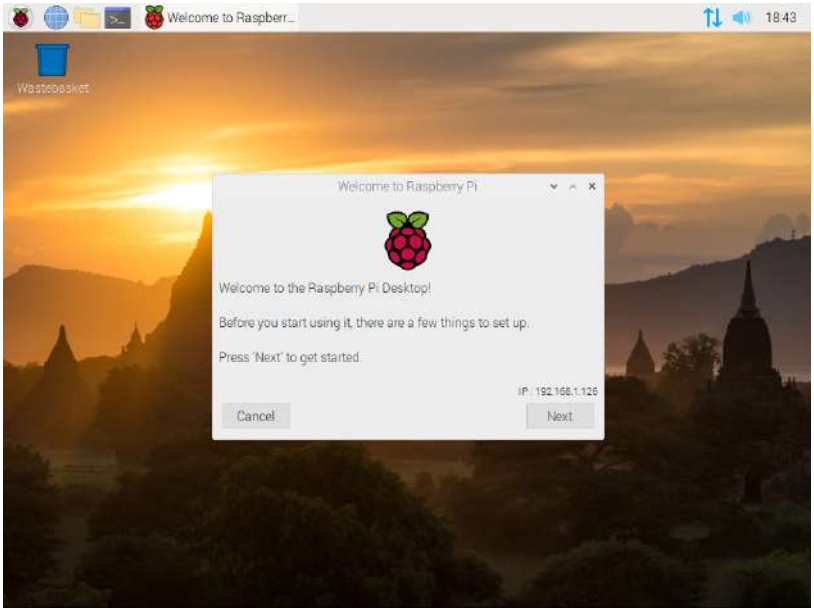


Figura 4: Escritorio de Raspbian

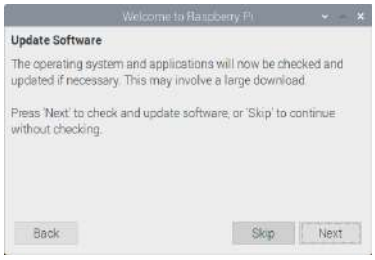
Una vez terminado el proceso de arranque, siga el asistente para configurar Raspbian, tal como se muestra en la Figura 5.

(a) Inicio del Asistente de Conficuración

(b) Selección de idioma y datos de localización

(c) Cambio de contraseña del usuario principal *pi*

(d) Calibración de pantalla



(e) Actualización de Software Requiere conexión a internet

(f) Fin del asistente y reinicio

Figura 5: Asistente de configuración de Raspbian

Finalmente, se aconseja expandir la partición de Raspbian a su máxima capacidad. Para ello, ejecute en una terminal la herramienta de configuración `sudo raspi-config` (véase Figura 7), seleccione la opción 7, *Opciones avanzadas* (Avanced Options), y luego la opción A1, *Extender el sistema de archivos* (Expand Filesystem).

A. Instalación de Raspbian via SSH

En caso de que no se cuente con un monitor, es posible instalar Raspbian vía SSH. Para ello es necesario habilitar conexiones vía SSH ya que estas se encuentran deshabilitadas por defecto.

A.1. Habiliar conexiones SSH

Para habilitar SSH, se necesita crear un archivo vacío llamado `SSH` en la raíz de la partición `rootfs`. Supóngase que la microSD está asociada a `/dev/sdc`, las paticiones `boot` y `rootfs` serán entonces `/dev/sdc1` y `/dev/sdc2` respectivamente. El proceso de creación del archivo `SSH` en la raíz de la partición `rootfs` es como sigue:

```
$ mkdir /media/user/rootfs
# mount /dev/sdc2 /media/user/rootfs
# touch /media/user/rootfs/SSH
```

Nótese que los comandos marcados con # deben ejecutarse con permisos de super usuario (i.e. *root*, o mediante `sudo` en Ubuntu).

Una vez completado este proceso, desmonte la memoria microSD e insértela en la Raspberry Pi.

A.2. Configurar Raspbian vía SSH

Para configuar Raspbian via SSH, la Raspberry Pi deberá estar conectada a la red local vía un cable Ethernet y tener una tarjeta de memoria microSD con una imagen de Raspbian precargada.

A continuación, conecte la Raspberry Pi y espere entre 1 y 3 minutos a que el sistema operativo cargue. Utilice un escaner de IP o consulte su enrutador para conocer la IP asignada a la Raspberry Pi.

Client Name	Interface	IPv4 Address	MAC Address	Expires Time	
PC	Wireless	192.168.1.112	00:00:00:00:00:00	21:43:42	Delete
raspberrypi	LAN	192.168.1.126	B8:27:EB:FB:F8:93	23:45:26	Delete

Figura 6: Dirección IP de una Raspbery Pi

Una vez conozca la dirección IP de la Raspberry Pi, conéctese a ésta mediante SSH. Secure Shell le advertirá que no puede verificar la autenticidad del certificado, por lo que pedirá que confirme la conexión tecleando `yes`, como se muestra a continuación.

```
$ ssh pi@192.168.1.126

    The authenticity of host '192.168.1.126 (192.168.1.126)' can't be established.
    ECDSA key fingerprint is SHA256:1nrpQeTIb+Gzg4aIJ0WE+V0aLUQgDnQbxOGraWf0Kso.
    Are you sure you want to continue connecting (yes/no)?
```

Teclee `yes` y presione Enter. De inmediato se le solicitará la contraseña

```
    Warning: Permanently added '192.168.1.126' (ECDSA) to the list of known hosts.

    pi@192.168.1.126's password:
```

Teclee `raspberry`, la contraseña por default en Raspbian, y presione Enter. Se concretará la conexión.

```
    Linux raspberrypi 4.19.75-v7+ #1270 SMP Tue Sep 24 18:45:11 BST 2019 armv7l

    The programs included with the Debian GNU/Linux system are free software;
    the exact distribution terms for each program are described in the
    individual files in /usr/share/doc/*/copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb  6 18:28:53 2020

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.
```

Finalmente, para configurar su Raspbian, ejecute `sudo raspi-config` para iniciar la herramienta de configuración (véase Figura 7). Se aconseja definir el idioma, localización, y cambiar la contraseña del usuario por defecto *pi*.

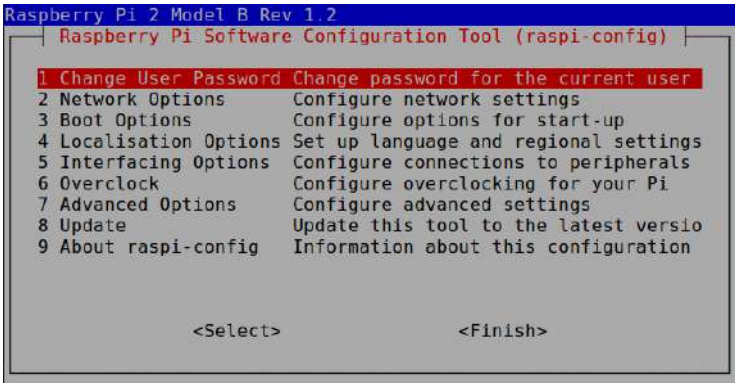


Figura 7: Herramienta de configuración de Raspbian

Práctica 2:
Uso del puerto GPIO de la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a utilizar el puerto GPIO de la Raspberry Pi, configurándo varios pines como salidas digitales para el control de leds y circuitos de lógica TTL.

2. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Además, el alumno necesitará:

- 7 Diodos emisores de luz LEDS
- 8 resistencias de 330Ω
- 1 Condensador de 0.1μF
- 1 Array Darlington ULN2003 (o 7 transistores de potencia)
- 1 Decodificador de 7–segmentos ánodo común
- 1 Display de 7 segmentos ánodo común
- 1 Conector DIL con cable plano tipo listón para el GPIO de la Raspberry Pi (similar al de un Disco Duro PATA, véase Figura 1)
- 1 protoboard o circuito impreso equivalente
- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Cables y conectores varios

Figura 1: Cable plano con conector DIL

3. Instrucciones

1. Alambre el circuito tal y como se detalla en la subsección 3.1
2. Antes de conectar la Raspberry Pi, pruebe el circuito como se explica en la subsección 3.2
3. Realice los programas de las subsecciones 3.3 a 3.5
4. Analice los programas de las subsecciones 3.3 a 3.5, realice los experimentos propuestos en la sección 4 y con los resultados obtenidos responda el cuestionario de la sección 5.

3.1. Paso 1: Alambrado

El proceso de alambrado de esta práctica considera dos circuitos.

El primer circuito permitirá controlar con las salidas digitales del GPIO de la Raspberry Pi el encendido y apagado de siete leds mediante el uso de un encapsulado de varios controladores de potencia tipo Darlington

(*Darlington Array*). De manera simimlar, el segundo circuito se auxiliará de un integrado TTL para desplegar números del 0 al 9 en un display de 7 segmentos (véase Figura 2).

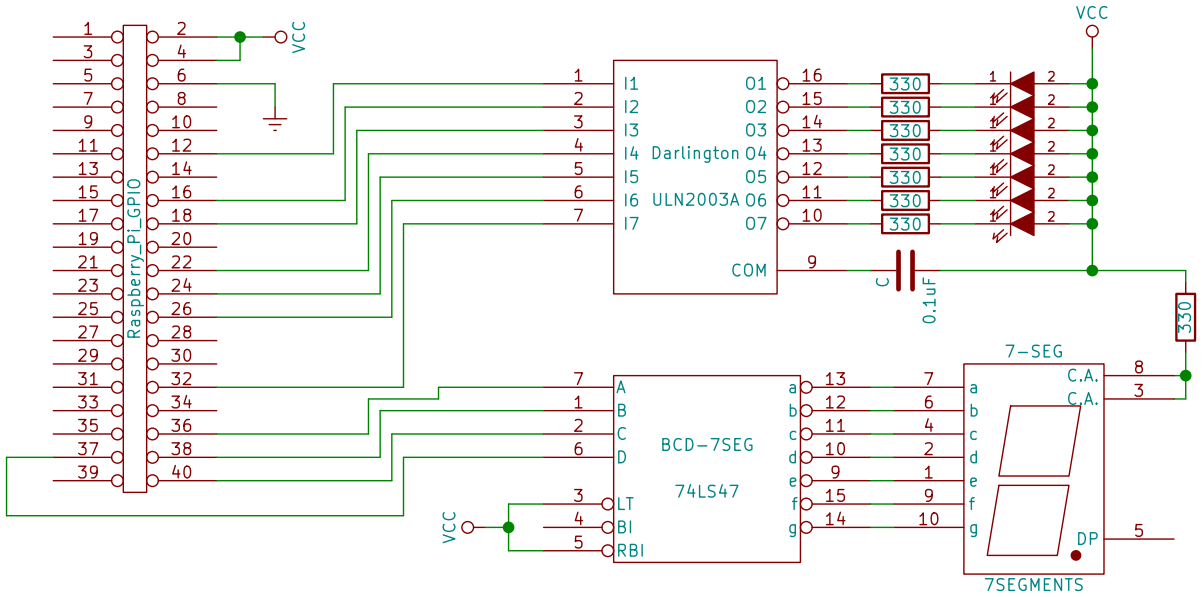


Figura 2: Diagrama de conexiones del circuito a alambrar

3.1.1. Subcircuito 1: leds en línea

. Forme los siete leds en línea, cuidando de que todos tengan la misma orientación. A continuación, conecte el cátodo de cada LED a una resistencia de 300Ω, y ésta a su vez a una salida libre del controlador ULN2003, de tal manera que el primer led de la fila esté conectado a la salida 1, el segundo a la salida 2, y así sucesivamente.

De manera similar, conecte las entradas 1 a 7 del ULN2003 a las salidas GPIO 18, 23, 24, 25, 8, 7, y 12 de la Raspberry Pi mediante el cable tipo listón (pines 12, 16, 18, 22, 24, 26 y 32). La conexión de uno de los leds se conecte al GPIO12/PWM es importante, pues éste pin se utilizará para variar la intensidad del led más adelante.

Complete el alambrado del ULN2003 conectándolo a tierra, conectando el común de éste a VCC mediante el capacitor de 0.1μF. Finalmente, conecte el ánodo de todos los leds a VCC.

3.1.2. Subcircuito 2: Display de 7 segmentos

. Comience conectando las salidas a a g del display del circuito controlador 74LS47 con los pines homónimos del display de siete segmentos. A continuación, conecte el integrado a tierra y las terminales LT y RBI (pines 3 y 5) a VCC, dejando BI (pin 4) sin conectar tal como se muestra en la Figura 2. Como siguiente paso, conecte el ánodo común del display a VCC mediante una resistencia de 330Ω.

A continuación conecte las entradas A, B, C, D a las salidas GPIO 16, 20, 21 y 26 respectivamente (pines 36, 38, 40 y 37 de la Raspberry Pi) mediante el cable tipo listón.

Finalmente, para terminar el alambrado, conecte el 74LS47 a VCC y tierra.

3.2. Paso 2: Configuración y prueba del circuito

Antes de proceder, verifique conexiones con un multímetro en busca de corto circuitos. En particular verifique que exista una impedancia muy alta entre los pines 4 y 6 (VCC y tierra) del cable listón que conectará al GPIO de la Raspberry Pi.

Tras lo anterior, conecte VCC y tierra del circuito alambrado a un eliminador de corriente de 5V y cierre el circuito entre los pines 4 y 32 del cable listón. Si todo está alambrado correctamente, el último del de la fila deberá encender.

Ahora cierre el circuito entre los pines 4 y 37 del cable listón. Si todo está alambrado correctamente, el display de siete segmentos deberá mostrar un 1.

Importante: Ninguno de los circuitos o resistencias debe calentarse. Si alguno de los componentes emitiera calor, verifique las conexiones en busca de corto circuitos o reemplace los componentes dañados.

Verificadas las conexiones, instale los complementos de desarrollo del puerto de propósito general de la Raspberry Pi (deberían venir instalados por defecto).

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

3.3. Paso 3: Led parpadeante

Con todas las conexiones probadas y verificadas, es seguro proceder al control de señales digitales utilizando el GPIO de la Raspberry Pi.

Inicie su Raspberry Pi y, ya sea mediante una terminal remota o directamente en ella, ejecute el siguiente código Python para hacer parpadear uno de los leds del circuito alambrado.

```
# Importa la librería de control del GPIO de la Raspberry Pi
import RPi.GPIO as GPIO
# Importa la función sleep del módulo time
from time import sleep

# Desactivar advertencias (warnings)
GPIO.setwarnings(False)
# Configurar la librería para usar el número de pin.
# Llame GPIO.setmode(GPIO.BCM) para usar el canal SOC definido por Broadcom
GPIO.setmode(GPIO.BOARD)
# Configurar el pin 32 como salida y habilitar en bajo
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)

# El siguiente código hace parpadear el led
while True: # Bucle infinito
  sleep(0.5)                # Espera 500ms
  GPIO.output(32, GPIO.HIGH) # Enciende el led
  sleep(0.5)                # Espera 500ms
  GPIO.output(32, GPIO.LOW)  # Apaga el led
```

3.4. Paso 4: Led parpadeante con PWM

En lugar de utilizar tiempos de espera (mismos que consumen tiempo de procesamiento y energía), es posible hacer parpadear el led de manera mucho más precisa y rápida utilizando uno de los moduladores de ancho de pulso (en inglés *Pulse Width Modulation* o *PWM*) por hardware que incorpora la Raspberry Pi.

Ejecute el siguiente código Python para hacer parpadear uno de los leds del circuito alambrado.

```
# Importa la librería de control del GPIO de la Raspberry Pi
import RPi.GPIO as GPIO
# Importa la función sleep del módulo time
from time import sleep

# Desactivar advertencias (warnings)
GPIO.setwarnings(False)
# Configurar la librería para usar el número de pin.
GPIO.setmode(GPIO.BOARD)
# Configurar el pin 32 como salida y habilitar en bajo
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)
# Inicializar el pin 32 como PWM a una frecuencia de 2Hz
pwm = GPIO.PWM(32, 1)

# El siguiente código hace parpadear el led
pwm.start(50)

```

```
flag = True
while flag:
  try:
    dutyCycle = int(input("Ingrese ciclo de trabajo: "))
    pwm.ChangeDutyCycle(dutyCycle)
  except:
    flag = False
    pwm.ChangeDutyCycle(0)
# Detiene el PWM
pwm.stop()
# Reinicia los puertos GPIO (cambian de salida a entrada)
GPIO.cleanup()
```

3.5. Paso 5: Display de siete segmentos

El último ejemplo consiste en mostrar números en el display de siete segmentos.
Analice y ejecute el código mostrado a continuación.

```
# Importa la librería de control del GPIO de la Raspberry Pi
import RPi.GPIO as GPIO
# Importa la función sleep del módulo time
from time import sleep

# Desactivar advertencias (warnings)
GPIO.setwarnings(False)
# Configurar la librería para usar el número de pin.
GPIO.setmode(GPIO.BOARD)
# Configurar pines 36, 38, 40 y 37 como salida y habilitar en bajo
GPIO.setup(32, 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)

# Mapea bits a los pines de la GPIO
def bcd7(num):
  GPIO.output(32, GPIO.HIGH if num & 0x00000008 else GPIO.LOW )
  GPIO.output(38, GPIO.HIGH if num & 0x00000004 else GPIO.LOW )
  GPIO.output(40, GPIO.HIGH if num & 0x00000002 else GPIO.LOW )
  GPIO.output(37, GPIO.HIGH if num & 0x00000001 else GPIO.LOW )

flag = True
while flag:
  try:
    num = int(input("Ingrese número entero: "))
    bcd(num)
  except:
    flag = False
# Reinicia los puertos GPIO (cambian de salida a entrada)
GPIO.cleanup()
```

4. Experimentos

1. [1pt] Modifique el código de la subsección 3.3 para todos los leds de la fila parpadeen.
2. [1pt] Modifique el código de las subsecciones 3.3 y 3.5 para que los leds de la fila enciendan de manera continua en una marquesina de izquierda a derecha.
3. [2pt] Modifique el código de las subsecciones 3.3 y 3.5 para que los leds de la fila enciendan de manera continua en una marquesina de derecha a izquierda con velocidad variable definida por el usuario.

4. [1pt] Con base en las modificaciones anteriores, genere una marquesina que haga parecer que «la luz rebota» al llegar a las orillas (efecto ping-pong) con velocidad variable definida por el usuario.

5. [3pt] Tomando como base el código de la subsección 3.4, haga que uno de los leds encienda gradualmente a lo largo de un segundo hasta adquirir máxima potencia, permanezca encendido medio segundo, y después se apague gradualmente a lo largo de otro segundo.

5. Cuestionario

1. [0.5pt] Explique por qué usar corrimientos es la manera más eficiente de generar una marquesina.

2. [0.5pt] Explique las ventajas o desventajas que tiene utilizar un modulador de ancho de pulso sobre tiempos de espera programados (*delays*).

3. [0.5pt] ¿Sería posible generar una marquesina circular utilizando el 74LS47 y el display de siete segmentos? Justifique su respuesta.

4. [0.5pt] ¿Es posible configurar cualquier pin de la GPIO como PWM? Justifique su respuesta.

A. Programa Ejemplo: `blink.py`

src/blink.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Importa la librería de control del GPIO de la Raspberry Pi
import RPi.GPIO as GPIO
# Importa la función sleep del módulo time
from time import sleep

# Desactivar advertencias (warnings)
# GPIO.setwarnings(False)
# Configurar la librería para usar el número de pin.
# Llame GPIO.setmode(GPIO.BCM) para usar el canal SOC definido por Broadcom
GPIO.setmode(GPIO.BOARD)

# Configurar el pin 32 como salida y habilitar en bajo
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)

# El siguiente código hace parpadear el led
while True: # Bucle infinito
  sleep(0.5)                 # Espera 500ms
  GPIO.output(32, GPIO.HIGH) # Enciende el led
  sleep(0.5)                 # Espera 500ms
  GPIO.output(32, GPIO.LOW)  # Apaga el led
```

B. Programa Ejemplo: `bcd.py`

src/bcd.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Importa la librería de control del GPIO de la Raspberry Pi
import RPi.GPIO as GPIO
# Importa la función sleep del módulo time
from time import sleep

# Desactivar advertencias (warnings)
# GPIO.setwarnings(False)
# Configurar la librería para usar el número de pin.
GPIO.setmode(GPIO.BOARD)
# Configurar pines 36, 38, 40 y 37 como salida y habilitar en bajo
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)

# Mapea bits a los pines de la GPIO
def bcd7(num):
  GPIO.output(36, GPIO.HIGH if (num & 0x00000001) > 0 else GPIO.LOW )
  GPIO.output(38, GPIO.HIGH if (num & 0x00000002) > 0 else GPIO.LOW )
  GPIO.output(40, GPIO.HIGH if (num & 0x00000004) > 0 else GPIO.LOW )
  GPIO.output(37, GPIO.HIGH if (num & 0x00000008) > 0 else GPIO.LOW )

flag = True
while flag:
  try:
    num = int(input("Ingrese número entero: "))
    bcd7(num)
  except:
    flag = False
  #end try
#end while

# Reinicia los puertos GPIO (cambian de salida a entrada)
GPIO.cleanup()
```

C. Programa Ejemplo: `pwm.py`

src/pwm.py

```
1
2  # Future imports (Python 2.7 compatibility)
3  from __future__ import absolute_import
4  from __future__ import division
5  from __future__ import print_function
6
7  # Importa la librería de control del GPIO de la Raspberry Pi
8  import RPi.GPIO as GPIO
9  # Importa la función sleep del módulo time
10 from time import sleep
11
12 # Desactivar advertencias (warnings)
13 GPIO.setwarnings(False)
14 # Configurar la librería para usar el número de pin.
15 GPIO.setmode(GPIO.BOARD)
16 # Configurar el pin 32 como salida y habilitar en bajo
17 GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)
18 # Inicializar el pin 32 como PWM a una frecuencia de 1Hz
19 pwm = GPIO.PWM(32, 1)
20
21 # El siguiente código hace parpadear el led
22 pwm.start(50)
23 flag = True
24 while flag:
25   try:
26     dutyCycle = int(input("Ingrese ciclo de trabajo: "))
27     pwm.ChangeDutyCycle(dutyCycle)
28   except:
29     flag = False
30     pwm.ChangeDutyCycle(0)
31   #end try
32 #end while
33 # Detiene el PWM
34 pwm.stop()
35 # Reinicia los puertos GPIO (cambian de salida a entrada)
36 GPIO.cleanup()
```

D. Programa Ejemplo: `pwm_fast.py`

src/pwm_fast.py

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)
pwm = GPIO.PWM(32, 1000)

pwm.start(50)
flag = True
while flag:
  try:
    dutyCycle = int(input("Duty cycle: "))
    pwm.ChangeDutyCycle(dutyCycle)
  except:
    flag = False
    pwm.ChangeDutyCycle(0)
  #end try
#end while
pwm.stop()
GPIO.cleanup()
```

Práctica 3:
Instalación de Windows 10 IoT Core en la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a instalar un sistema operativo basado en Windows, como sistema operativo embebido, en una tarjeta microcontroladora.

2. Instrucciones

Para Windows 10 en la Raspberry se necesita lo siguiente:

- Una computadora cono Microsoft Windows 10 capaz de leer y escribir tarjetas microSD (o bien un adaptador para la misma) y conexión a internet para descargar la imagen de Windows 10 IoT Core.
- Una tarjeta de memoria microSD de al menos 16 GB (se recomiendan 32GB)
- Una Raspberry Pi 2 o posterior (se requieren versiones con 1GB de ram o más)
- Un monitor con soporte para HDMI
- Un teclado USB
- Un mouse USB
- Una fuente de alimentación de 5V@1A con adaptador microUSB

IMPORTANTE: Se necesita Windows 10 para poder instalar Windows 10 IoT Core en una Raspberry Pi. Si no cuenta con una máquina que tenga Windows 10 precargado, no podrá realizarse esta práctica. Debido a que Windows 10 es software propietario, la realización de esta práctica no es obligatoria.

2.1. Paso 1: Descargar Windows 10 IoT Core

1. Vaya a centro de desarrollo de Windows 10 (Windows 10 developer center).
2. De clic en *Get Windows 10 IoT Core Dashboard* para descargar la aplicación requerida.

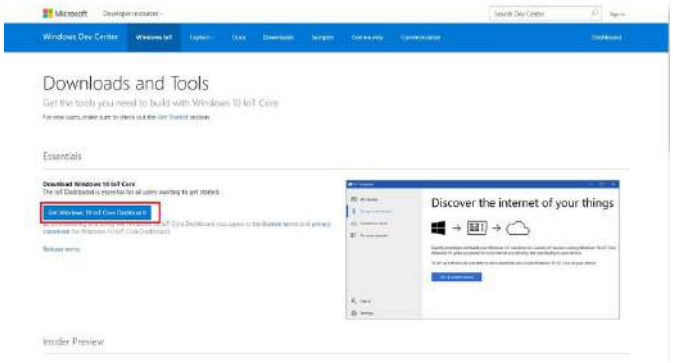


Figura 1: Descarga de Windows 10 IoT Core Dashboard

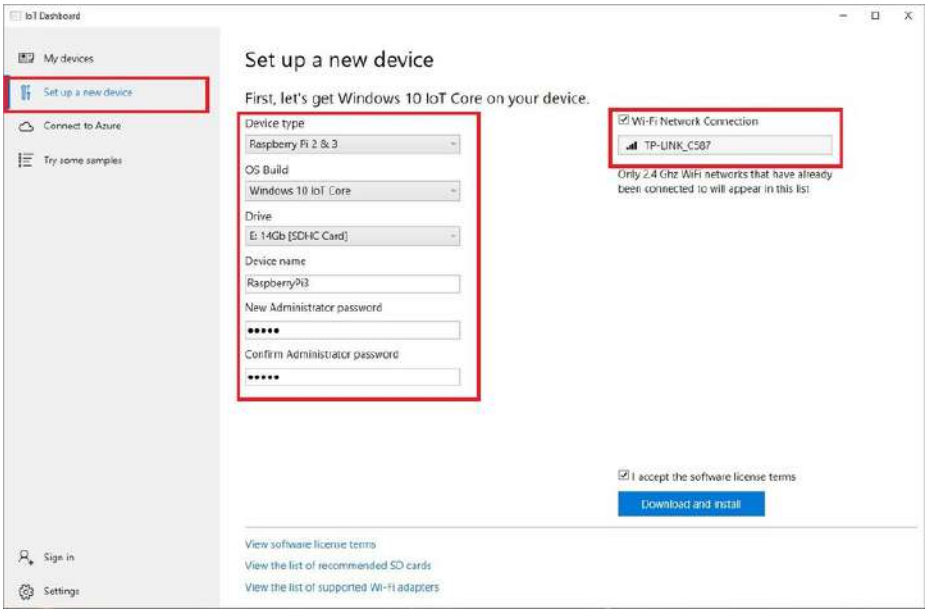


Figura 2: Configuración de la descarga de Windows 10 IoT Core

También puede utilizarse ésta liga: https://docs.microsoft.com/en-us/windows/iot-core/downloads

3. Instale la aplicación y ejecútela.

4. Seleccione la opción *configurar un nuevo dispositivo* (Set up a new device) en la barra lateral y seleccione configurar Windows 10 IoT Core en una Raspberry Pi tal como se muestra en la Figura 2 Tenga cuidado de seleccionar la letra de unidad correcta de la memoria microSD, una conexión a internet, y de anorar un nombre de dispositivo y contraseña.

5. De clic en el botón *Descargar e instalar* (Download and install).

La aplicación descargará los archivos requeridos de los servidores Microsoft y los exscribirá en la memoria microSD. Tenga en cuenta que este proceso puede llevar varias horas, especialmente con conexiones lentas.

Windows 10 IoT Core Dashboard le notificará cuando el proceso de descarga y copiado de datos haya concluído (véase Figura 3).

2.2. Paso 2: Configuración de Windows 10 IoT Core en la Raspberry Pi

Una vez terminados la descarga y copiado de datos en la microSD, inserte la tarjeta en la Raspberry Pi. A continuación, conecte la Raspberry Pi a un monitor usando el puerto HDMI, así como a un teclado y ratón. Finalmente, conecte la Raspberry Pi a la alimentación.

La raspberry Pi tardará aproximadamente dos minutos en arrancar y mostrará una pantalla de carga durante el proceso de arranque (véase Figura 4).

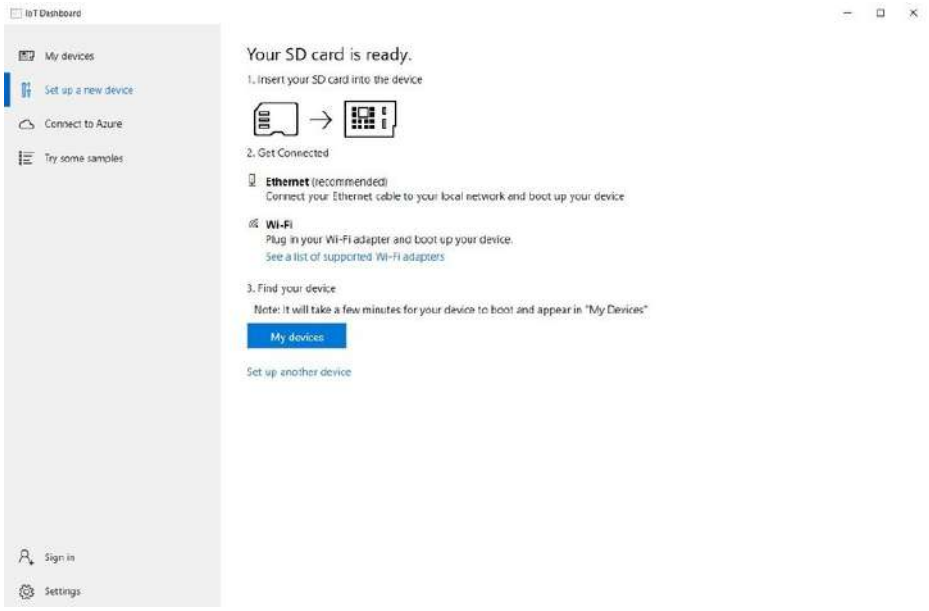


Figura 3: Imagen de Windows 10 IoT Core instalada en la memoria microSD

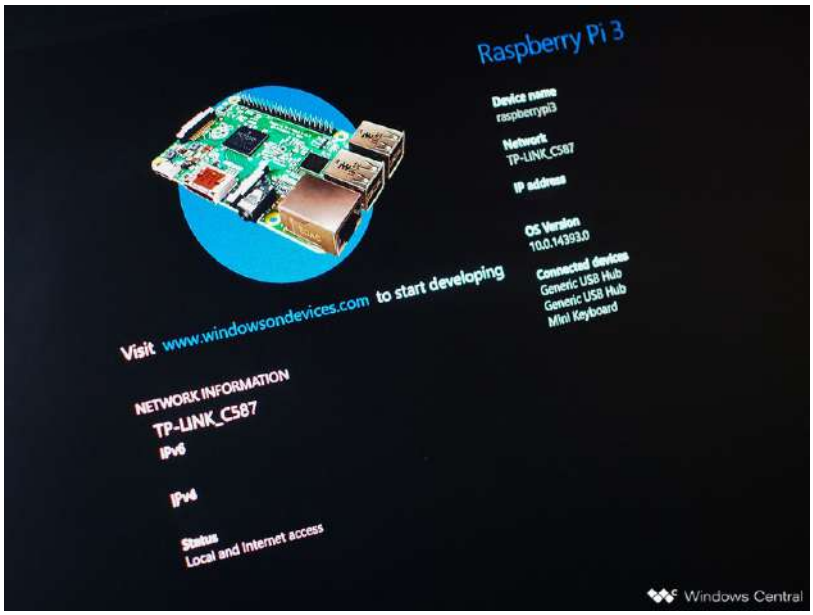


Figura 4: Arranque de Windows 10 IoT Core en la Raspberry Pi

Notará que a diferencia de otros sistemas Windows, hay muy poco que configurar a la Raspberry Pi. Se le solicitará que elija el idioma y que entre la contraseña de la red inalámbrica con la que desea conectarse a la Internet (si su Raspberry Pi está equipada con WiFi).

Notará que el ambiente es bastante austero y carece de aplicaciones. Esto es debido a que una vez que se instale una aplicación en la Raspberry Pi, Windows desaparecerá y sólo le permitirá interactuar con la App.

Si la Raspberry Pi está conectada a la red local, podrá ver que ésta aparece en la aplicación de desarrollo (Figura 5).

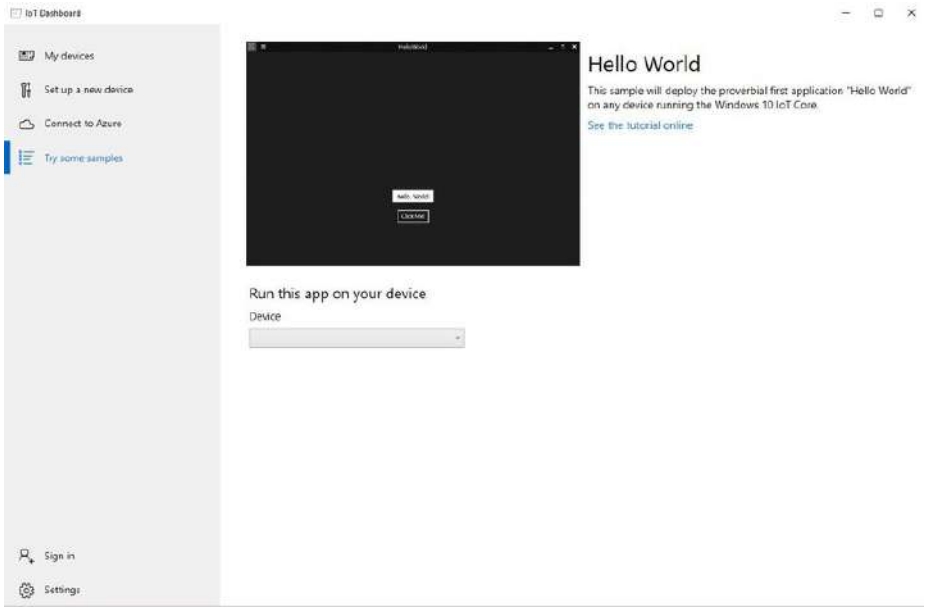


Figura 5: Raspberry Pi en la Windows 10 IoT Core Dashboard

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
FUNDAMENTOS DE SISTEMAS EMBEBIDOS

PRÁCTICA 4 "MANEJO REMOTO DE RASPBERRY PI MEDIANTE BOTS DE TELEGRAM"

Objetivo: Manejo y monitoreo remoto de Raspberry Pi y sus puertos GPIO mediante la programación y aplicación de Bots de Telegram

Introducción.

Una de las características más importantes de los sistemas embebidos en tiempo real es que brinden la posibilidad al usuario de obtener información del medio en el que se encuentra cuando éste lo requiera; el manejo remoto de los sistemas embebidos cobra gran relevancia cuando este se encuentra en un ambiente hostil para el ser humano o alejado a gran distancia de los usuarios para los que fue diseñado. Por lo anteriormente señalado, en esta práctica se pretende que la creación de Bots a través de la API de Telegram, permita a los usuarios y diseñadores de Sistemas embebidos ejecutar instrucciones en su tarjeta de desarrollo (para esta asignatura Raspberry Pi) mediante algún dispositivo inteligente con conexión a internet.

En un contexto general, podemos definir a un Bot como un programa autónomo e interactivo que son creados para realizar un propósito específico, el cual puede ser controlado por secuencias de texto o inteligencia artificial. En el caso particular de la API para desarrollar Bots de Telegram, es compatible para la mayoría de los lenguajes de programación de alto nivel, por lo que puede ser fácilmente programado en el lenguaje de programación predilecto; pero para la Raspberry Pi se utiliza con mayor frecuencia Python.

Cada Bot, es identificado mediante un token único que nos va a permitir comunicarnos con nuestra tarjeta de desarrollo mediante Telegram, por lo que, el desarrollo de esta práctica nos permitirá desarrollar sistemas embebidos que realicen diversas acciones mediante la utilización de hardware diverso y sensores de forma remota y la posibilidad de realizarlas a petición del usuario o de forma automática.

Duración.

1 semana

Material.

- Raspberry Pi
- Dispositivo inteligente (celular, tablet, etc.)
- 1 LED
- 1 resistencia 330 Ω
- 2 cables jumpers macho-hembra
- 1 sensor ultrasónico (puede ser reemplazado por fotoresistencia, sensor PIR o inclusive push button)

Desarrollo.

Antes de iniciar con el desarrollo y la programación de Bots en la API de Telegram, es necesario realizar algunos pasos de instalación e importación de bibliotecas, tanto en la Raspberry Pi como en el dispositivo inteligente con el que se comunicará la Tarjeta de desarrollo y Telegram. Por lo que se deben realizar los pasos que se enlistan a continuación.

Instalación en Raspberry

- sudo apt-get update && sudo apt-get upgrade -y
- sudo apt-get install python-setuptools
- sudo apt-get install python-pip
- sudo pip install pyTelegramBotAPI
- sudo apt-get install sysstat
- sudo chown -R pi:pi /home/pi/pyTelegramApi
- cd /home/pi/pyTelegramBotApi
- sudo python setup.py install

Instalación en Dispositivo Inteligente

- Instalar Telegram
- Buscar @BotFather.
- Escribir el comando /newbot.
- Ingresar el alias o nombre mostrado que será asignado al bot.
- Ingresar un nombre de usuario para el bot.
- Guardar el token generado.
- Buscar el bot creado por el nombre de usuario.

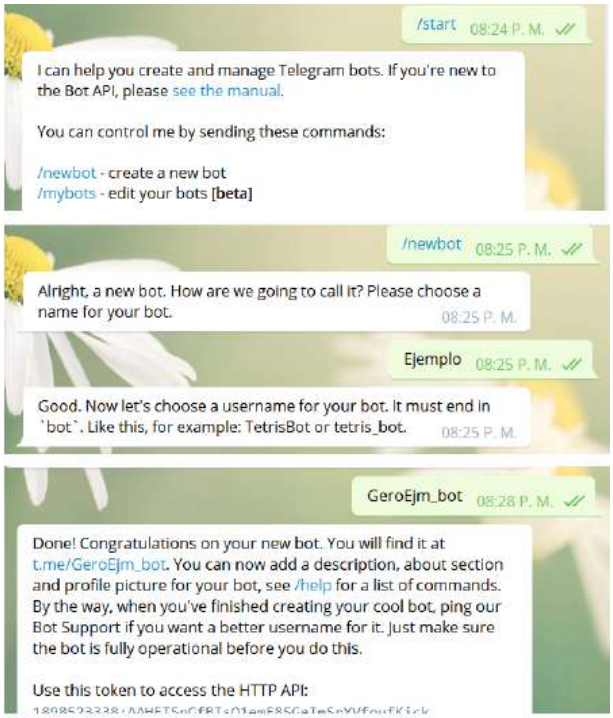


Figura 1. Captura de pantalla de la creación de un nuevo Bot en Telegram

Adicionalmente a los pasos anteriores, es necesario realizar una configuración utilizando una terminal en la Raspberry Pi, para poder introducir el token único creado por nuestro @BotFather en Telegram y que fue asignado al Bot creado para esta práctica. Cabe recordar que, sin este paso, nuestra Raspberry y nuestro Dispositivo inteligente no podrán comunicarse, por lo que se deben seguir las instrucciones que se indican a continuación:

Configuración en Raspberry

- Escribir el comando cd /home/pi/pyTelegramBotApi/examples para ingresar al directorio que contiene ejemplos de la biblioteca descargada
- Abrir el archivo echo_bot.py.
- Sustituir el valor de la variable API_TOKEN con el Token generado al crear el bot en el chat de BotFather dentro de la aplicación de Telegram.

Una vez realizado lo anterior, contamos con lo necesario para iniciar la programación de nuestros Bots mediante la API de Telegram. Para verificar que se ha establecido correctamente la conexión entre Telegram y nuestra Raspberry Pi, realizaremos la siguiente actividad:

Comprobación de Conectividad

- Abrir el archivo echo_bot.py.
- Modificar los valores de retorno de los comandos /help y /start que se encuentran dentro del archivo
- Ejecutar el programa con el comando python echo_bot.py .
- Escribir los comandos /help y /start en la aplicación de Telegram, dentro del chat con el bot, para que este devuelva el mensaje escrito en el punto anterior.

Nota: Si concluyó satisfactoriamente la actividad anterior, verifica con tu profesor el correcto funcionamiento.

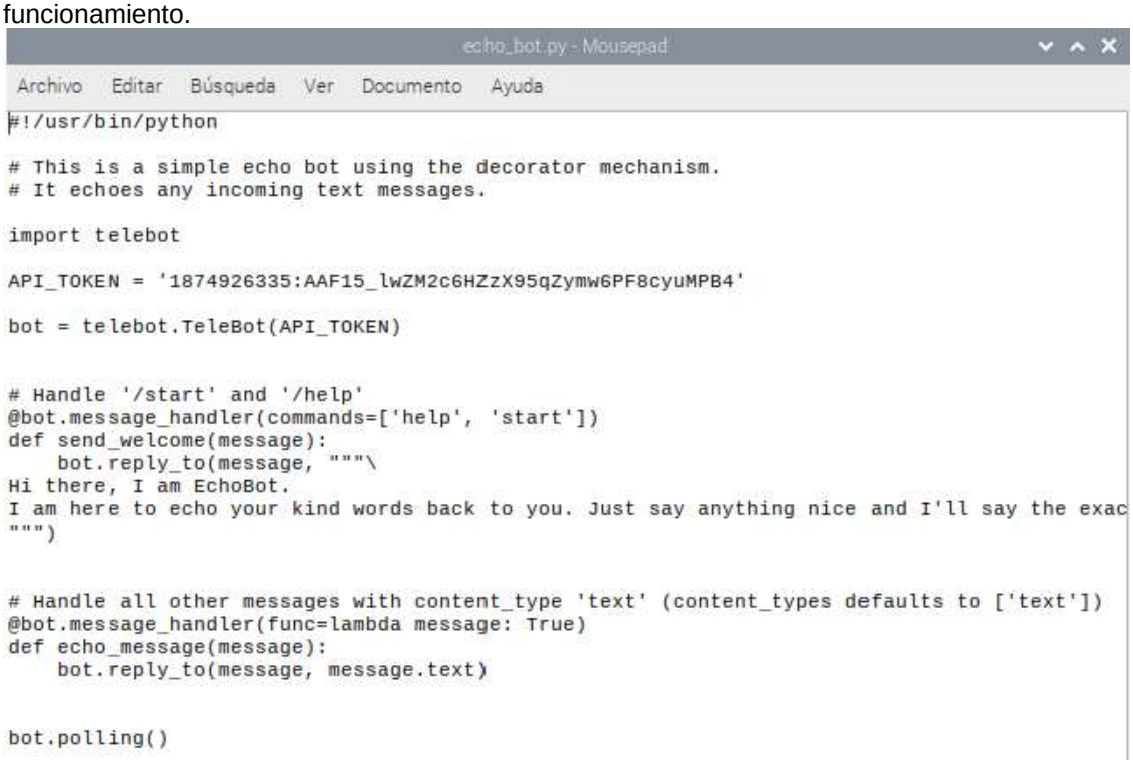


Figura 2. Ejemplo de archivo echo_bot.py modificado con API_TOKEN generado y mensaje de retorno comando /help y /start

Una vez verificada la conectividad entre nuestra tarjeta de desarrollo y Telegram, podemos realizar algunos ejemplos de Bots para ejecutar acciones básicas de forma remota en nuestra Raspberry Pi. Para ello te sugerimos realizar el siguiente ejemplo utilizando tu archivo echo_bot.py.

Cabe destacar que para poder encender y apagar un LED de forma directa en la Raspberry Pi, se utilizaría un código en Python como el siguiente:

```
from gpiozero import LED
from time import sleep
led = LED(17)
while True:
led.on()
sleep(1)
led.off()
sleep(1)
```

Partiendo de lo anterior, podemos destacar algunas líneas importantes en el código siguiente, mismo que deberás colocar en tu archivo echo_bot.py y posteriormente ejecutarlo.

Encender y Apagar un LED usando Telegram

```
import os
from gpiozero import LED
import telebot

led= LED(17)

API_TOKEN= 'debe colocar el token que se generó para su bot'

bot=telebot.TeleBot(API_TOKEN)

@bot.message_handler(commands=['apaga'])
def turn_off(message):
    led.off()
    bot.reply_to(message,"""\
    se apago\
    """)

@bot.message_handler(commands=['enciende'])
def turn_on(message):
    led.on()
    bot.reply_to(message,"""\
    se encendio\
    """)
```

Con el ejemplo anterior, se pretende que logre identificar el código necesario para la creación de un Bot que ejecute comandos básicos de la librería gpiozero para Raspberry Pi.

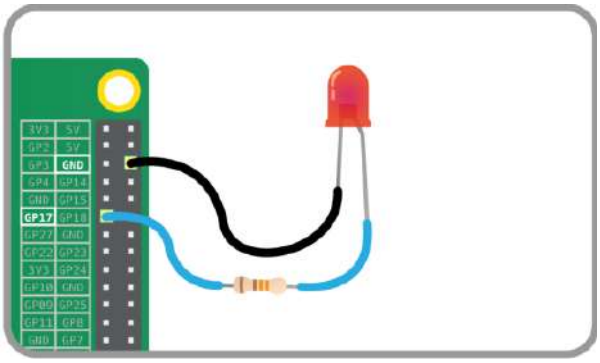


Figura 3. Ejemplo de conexión de LED para ejercicio de encendido y pagado del mismo

Ejercicio Propuesto.

- Diseñar una alarma de proximidad o movimiento utilizando un sensor ultrasónico, sensor PIR, fotoresistencia o el dispositivo que tenga disponible, que sea capaz de enviar una alerta al usuario mediante chat Bot de Telegram.

Sugerencias.

- *Recuerde importar las librerías necesarias; para el caso de este ejercicio, se requiere importar la librería DistanceSensor y LED de gpiozero si utiliza un sensor ultrasónico; si utiliza la fotoresistencia requiere de importar la librería LightSensor.*

Evaluación

Se comprobará que el alumno adquirió los conocimientos esperados mediante la siguiente lista de verificación.

- ✓ El alumno logró enviar un mensaje al usuario mediante la modificación de la función /help y /start del archivo echo_bot.py
- ✓ El alumno logró encender y apagar un LED mediante instrucciones en chat Bot de Telegram.
- ✓ El alumno logró implementar la alarma de proximidad, ya que, al activarse el sensor ultrasónico, se envía un mensaje de alerta al usuario mediante chat Bot de Telegram.
- ✓ El alumno comenta las líneas de código relevantes para cada ejercicio.

Bibliografía

García, Manuel J., Bot de Telegram que ejecute aventuras conversacionales, Universidad de Jaén, Recuperado el 24 de agosto de 2021 de.http://tauja.ujaen.es/bitstream/10953.1/5944/1/TFG_Garcia-Quesada_Manuel-Jesus.pdf

¿Qué es un robot? Definición de robot. Cloudflare. Recuperado el 28 de agosto de 2021 de: https://www.cloudflare.com/es-la/learning/bots/what-is-a-bot/

Lorenzo, P.(2016). Controla tu Raspberry Pi mediante Telegram. fwhibit. Recuperado el 28 de agosto de 2021 de: https://fwhibbit.es/controla-tu-raspberry-pi-mediante-telegram

Nutall, B. (2017). Gpiozero Documentation. Raspberry Pi Foundation, Dave Jones. Recuperado el 29 de agosto del 2021 de https://gpiozero.readthedocs.io/en/stable/

Práctica 5:
Modulación de potencia de una lámpara incandecente usando Arduino y la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a modular la potencia de una carga resistiva de alta potencia opto-acoplada a un microcontrolador por medio de un detector de cruce por cero y un TRIAC.

2. Introducción

La presente práctica resume los pasos a seguir para modular la cantidad de corriente que pasa por un foco incandescente con un microcontrolador. En particular, se interesa en el uso de un circuito de detección de cruce por cero para conmutar un triac acoplado a un Arduino UNO/Mega. Los datos registrados serán posteriormente enviados vía I^2C a una Raspberry Pi para controlar la intencidad del foco.

2.1. El puente rectificador

Un puente rectificador o rectificador de onda completa es un arreglo de 4 diodos que permiten el paso de la corriente en un sólo sentido, invirtiendo así la parte negativa de una señal de AC respecto a su voltaje de referencia. Los puentes rectificadores son un componente fundamental en los transformadores de corriente de AC a DC.

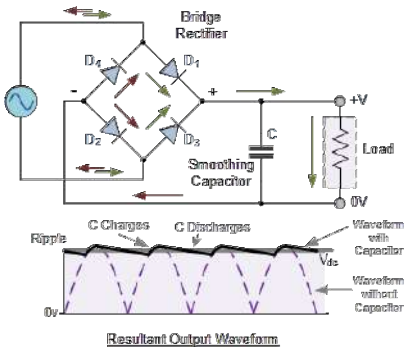


Figura 1: Rectificador de onda completa[1]

El circuito funciona de la siguiente manera. En la primera parte del ciclo, cuando el voltaje comienza a aumentar, la corriente fluye de la parte norte del puente (arriba) a través de D_1 hacia la carga, y luego de regreso por D_2 hacia el neutro (véase Figura 1). En esta primera etapa, D_3 y D_4 actuan como barreras evitando que la corriente fluya directamente de la fase al neutro (corto circuito). Tras pasar por la carga, el voltaje en en el ánodo de D_4 ha caído y es menor que en el cátodo, por lo que no habrá flujo de corriente en esta dirección, pero aún es positivo respecto al neutro ($V_N = 0$), por lo que la corriente tendrpa que pasar por D_1. De forma análoga, el voltaje en el cátodo de D_3 es menor que en el ánodo, por lo que tampoco habrá flujo en esta dirección.

En la segunda parte del ciclo, el voltaje de fase disminuye respecto al neutro, por lo que la corriente fluye de la parte sur del puente (abajo) a través de D_3 hacia la carga, y luego de regreso por D_4 hacia la fase (véase Figura 1). En esta segunda etapa, D_1 y D_2 actuan como barreras evitando que la corriente fluya directamente del neutro a la fase (corto circuito). Tras pasar por la carga, el voltaje en en el ánodo de D_2 ha caído y es menor que en el cátodo, por lo que no habrá flujo de corriente en esta dirección, pero aún es positivo respecto a la fase ($V_N = 0$), por lo que la corriente tendrpa que pasar por D_4. De forma análoga, el voltaje en el cátodo de D_1 es menor que en el ánodo, por lo que tampoco habrá flujo en esta dirección.

[1] Fuente de imagen: https://lasopaeden528.weebly.com/bridge-rectifier-calculator.html

2.2. Optoacopladores

Un optoacoplador o optoaislador es un circuito integrado que permite aislar mecánicamente dos circuitos, por lo que se usa comunmente para separar la lógica de control de los circuitos de potencia. El principio básico de un optoacoplador, como su nombre lo indica, es utilizar transductores ópticos para la transmisión de señales eléctricas. Así, en un lado del optoacoplador se tendrá siempre un diodo LED y en el otro extremo un fotoreceptor, que puede ser un foto SRC, un foto dárlington, un foto TRIAC o un fototransistor, siendo este último el más común.

En un optoacoplador, el diodo LED emite una cantidad de luz directamente proporcional a la corriente que circula por éste y, al incidir ésta en el fotoreceptor, el estímulo luminoso activa el paso de corriente a través de éste. Por ejemplo, en el caso de un fototransistor, al incidir la luz en la juntura de la base ésta se ioniza, generando un puente de iones que permite el flujo entre los extremos del transistor. Además, la mayoría de los optoacopladores tienen un pin conectado directamente a la base que sirve para ajustar la sensibilidad de la misma mediante la inyección de un voltaje pequeño.

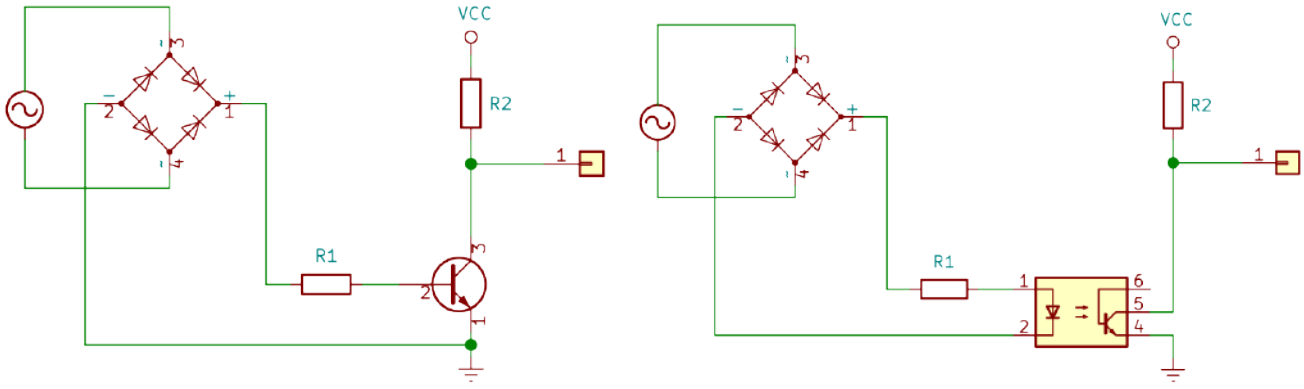
Los fototransistores foto-dárlingtons se utilizan principalmente en circuitos DC, mientras que los foto SCR y los foto TRIACs permiten controlar los circuitos de AC. Existen muchos otros tipos de combinaciones de fuente-sensor tales como LED-fotodiodo, LED-LÁSER, pares de lámpara-fotorresistencia, optoacopladores reflectantes y ranurados.

Por ejemplo, el integrado 4N25 puede usarse para monitorear el voltaje de la línea de tensión doméstica con un integrado. Según su hoja de especificaciones [1], la entrada del 4N25 acepta hasta 60mA, permite un flujo por el colector de hasta 50mA, y aisla hasta $5000V_{\text{RMS}}$. Si se toma como entrada un voltaje de línea rectificado de $127V_{\text{RMS}}$ y se limita la corriente a la corriente de prueba de 50mA indicada en la hoja de especificaciones [1], el 4N25 tendrá que acoplarse con una resistencia de al menos 3K9Ω aproximada mediante la fórmula:

$$R = \frac{V}{I} = \frac{127V_{\text{RMS}} \times \sqrt{2}}{0.050A} \approx \frac{179.61V}{0.050A} \approx 3692.1\Omega$$

2.3. Detector de cruce por cero

Un circuito detector de cruce por cero es un circuito electrónico diseñado para detectar cuando una señal senoidal pasa por cero. Estos circuitos se usan comunmente en electrónica de potencia tanto para detectar la frecuencia de la línea y hacer cálculo de fases. Además, al reducirse la diferencia de potencial a cero entre la fase y el neutro, la corriente instantánea también es cero, haciendo de éste el momento ideal para cortar la alimentación sin dañar las cargas.



(a) Detector de cruce por cero simple

(b) Detector de cruce por cero optoacoplado

Figura 2: Detectores de cruce por cero

La forma más sencilla de alambrar un circuito detector de cruce por cero es mediante un puente rectificador, dos resistencias y un transistor tipo NPN en modo interruptor (véase Figura 2a). El puente rectificador se encarga de invertir la parte negativa de la señal de AC, evitando así corrientes inversas que el transistor es incapaz de manejar. Cuando hay voltaje en la línea, éste habilita la base cerrando el circuito del transistor y conectando el pin de sensado a tierra (la resistencia de base evita el corto circuito y ajusta el umbral de sensibilidad). Tan pronto

como la diferencia de potencial entre la línea y el neutro cae a cero (o suficientemente bajo como la resistencia de base permita) el circuito se abre y en el pin de sensado se registra Vcc.

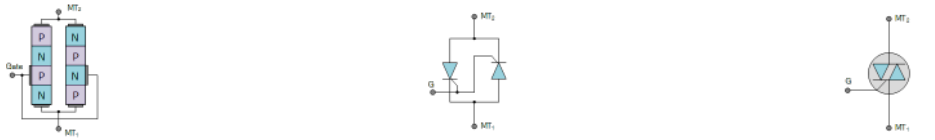
Para calcular R_1 es necesario tomar en cuenta las características eléctricas del transistor y los voltajes de pico de la línea. Se sabe que $V_{\text{pico}} = V_{\text{RMS}}\sqrt{2}$, por lo que usando la ley de ohm se tiene:

$$R_1 = \frac{V_{\text{RMS}}\sqrt{2}}{i_{\text{transistor}}} \approx \frac{1.4142 V_{\text{RMS}}}{i_{\text{transistor}}} \tag{1}$$

Sin embargo, el voltaje de la línea rara vez viene rectificado y un transitorio de corriente derivado de una descarga inductiva (arranque de refrigerador o microondas) puede incluso duplicar el voltaje de la línea, quemando no sólo el transistor sino el microprocesador. Es por esto que es muy aconsejable utilizar un optoacoplador en lugar de un simple transistor, tal como muestra la figura Figura 2b. Los principios de operación son los mismos.

2.4. TRIACs

Un TRIAC o triodo interruptor para corriente alterna (*Triode AC Switch*) es un integrado de estado sólido compuesto por dos tristores conectados en paralelo inverso (véase Figura 3b) que permite conmutar la corriente que pasa por un circuito de AC a alta frecuencia de manera similar a como operan los transistores bipolares y FETs en DC. Es decir, un TRIAC es un interruptor de estado sólido que puede operar a gran velocidad que, a diferencia de los relés, no existe la posibilidad de que un arco eléctrico funda los metales y el dispositivo se quede en encendido permanente, sino que al quemarse un TRIAC siempre abre el circuito. Por otro lado, basta una corriente muy pequeña entre el gate (G) y cualquiera de las terminales (MT_1 y MT_2) para encender al TRIAC, lo que lo convierte en el aliado ideal para controlar dispositivos de alta potencia con un microcontrolador.



(a) Construcción física (b) Equivalente en tristores (c) Símbolo

Figura 3: El Triac[2]

Como siempre, al utilizar un TRIAC es deseable aislar la parte de corriente directa del circuito de la parte de corriente alterna, es decir, el TRIAC deberá estar aislado pero acoplado al circuito DC. Esto normalmente se realiza mediante el uso de optoacopladores tipo MOC, tal como se ilustra en la Figura 4.

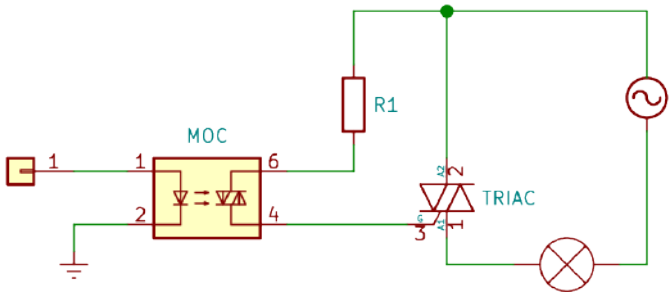


Figura 4: Triac optoacoplado

2.5. Modulación de potencia de carga resistiva en AC

A diferencia de un circuito de DC, la modulación de la potencia de una carga en un circuito de AC de una fase incolucra cuatro parámetros: i) el voltaje en la carga, ii) la corriente que circula por la carga, iii) la impedancia de la carga y iv) el ángulo de fase. O, matemáticamente hablando:

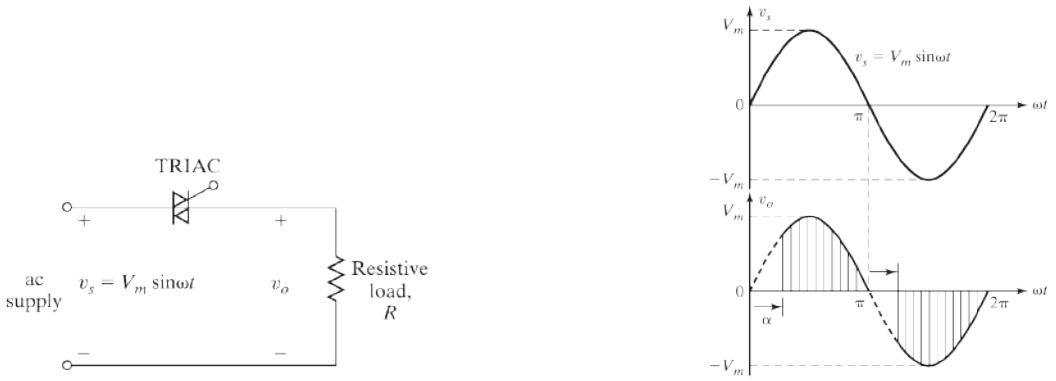
[2]Fuente de imagen: https://www.electronics-tutorials.ws/power/triac.html

$$p(t) = vi \tag{2}$$
$$= V_m \sin(\omega t + \theta_v)\, I_m \sin(\omega t + \theta_i) \tag{3}$$

donde:

- ω la velocidad angular tal que $\omega = 2\pi f$
- θ el ángulo de fase
- θ el ángulo de fase
- V_m el voltaje máximo
- I_m la corriente máxima

De estos cuatro parámetros, en un circuito puramente resistivo la impedancia R es constante, la corriente $i(\omega t + \theta_i)$ es proporcional al voltaje a la entrada de la carga y a la resistencia de la misma, y el voltaje $v(\omega t + \theta_v)$ oscila en el rango $[-V_m, V_m]$, con V_m el voltaje de pico. No obstante, se puede modificar voltaje promedio en la carga al variar el ángulo de fase y así controlar la potencia. Esta técnica es de hecho el principio fundamental de los circuitos convertidores de AC-AC a base de TRIAC como el que se muestra en la Figura 5.



(a) Circuito

(b) Señal voltaica

Figura 5: Convertidor AC-AC de una fase.[3]

Nótese que los ángulos de defasamiento de voltaje θ_v y corriente θ_i han desaparecido. Esto se debe a que en un circuito de AC de una sola fase $\theta_v = 0$ por ser la única fase y por ende la referencia. Además, en un circuito puramente resistivo las señales de voltaje y corriente están acopladas, por lo que $\theta_i = \theta_v$ tal como se muestra en la Figura 6.

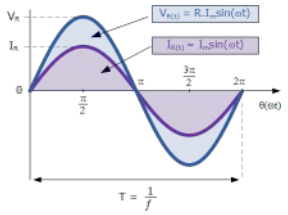


Figura 6: Voltaje y corriente en un circuito resistivo monofase.[4]

En la mayoría de los casos lo que interesa no es el cálculo de la potencia neta, sino controlar el factor de potencia, el decir, el porcentaje de la potencia máxima que la carga está entregando. Al estar sincronizadas la corriente y el voltaje por ser un circuito resistivo y no existir más que una fase, el cálculo de la potencia se simplifica y se convierte en el cociente del voltaje promedio aplicado a la carga respecto al voltaje RMS de la línea. Es decir

[3]Fuente de la imagen: Rashid [2, pp. 33].

$$P_f = \frac{V_o}{V_{RMS}} \tag{4}$$

y dado que V_{RMS} es fijo, lo que interesa calcular es el voltaje aplicado a la carga V_o que se calcula como [2, 579–583]:[5]

$$\begin{aligned} V_o^2 =& \frac{2}{2\pi}\int_{\alpha}^{\pi} 2V_{RMS}^2 \sin^2(\omega t)\, d(\omega t) && (5)\\ =& \frac{4V_{RMS}^2}{4\pi}\int_{\alpha}^{\pi} \left(1-\cos(2\omega t)\right) d(\omega t) && (6)\\ =& \frac{V_{RMS}^2}{\pi}\left(\pi - \alpha + \frac{\sin(2\alpha)}{2}\right) && (7) \end{aligned}$$

Por lo tanto

$$\begin{aligned} V_o =& \left[\frac{V_{RMS}^2}{\pi}\left(\pi - \alpha + \frac{\sin(2\alpha)}{2}\right)\right]^{\frac{1}{2}} && (8)\\ =& V_{RMS}\sqrt{\frac{1}{\pi}\left(\pi - \alpha + \frac{\sin(2\alpha)}{2}\right)} && (9) \end{aligned}$$

Ahora, supóngase que se desea modular la potencia de un foco incandescente de $60W$ conectado a una línea estándar de $V_{RMS} = 120V$, 60Hz. Como $P = \frac{V^2}{R}$ se puede calcular tanto la resistencia del foco como la corriente a fin de elegir el TRIAC adecuado:

$$\begin{aligned} R =& \frac{V^2}{P}\\ =& \frac{(120V)^2}{60W}\\ =& \frac{14400V^2}{60W}\\ =& 240\Omega \end{aligned}$$

y como $I = \frac{V}{R}$

$$\begin{aligned} I_{RMS} &= \frac{120V}{240\Omega} = 0.5A\\ I_m &= \sqrt{2}\times I_{RMS} = \sqrt{2}\times 0.5A\\ &= 0.7A \end{aligned}$$

[5] El valor medio o efectivo de cualquier función $f(\omega t)$ con periodo de $2\pi rad$ está determinado por $F = \sqrt{\frac{1}{2\pi}\int_0^{2\pi} f^2(\omega t)\, d\omega t}$

Ahora bien, si se usa un ángulo de disparo en el TRIAC de $\alpha = \frac{\pi}{2}$ la potencia de el foco con base en las Ecuaciones (4) y (9) será:

$$\begin{aligned}
P_f &= \frac{V_o}{V_{RMS}} \\
&= \frac{\cancel{V_{RMS}} \left[\frac{1}{\pi}\left(\pi - \alpha + \frac{\sin(2\alpha)}{2}\right)\right]^{\frac{1}{2}}}{\cancel{V_{RMS}}} \\
&= \left[\frac{1}{\pi}\left(\pi - \frac{\pi}{2} + \frac{\sin\left(\frac{2\pi}{2}\right)}{2}\right)\right]^{\frac{1}{2}} \\
&= \left[\frac{1}{\pi}\left(\frac{\pi}{2} + \frac{\sin(\pi)}{2}^{0}\right)\right]^{\frac{1}{2}} \\
&= \left(\frac{1}{\cancel{\pi}} \cdot \frac{\cancel{\pi}}{2}\right)^{\frac{1}{2}} \\
&= \sqrt{\frac{1}{2}} \\
&= 0.707 \\
&= 70.7\,\%
\end{aligned}$$

De este desarrollo se concluye, además, que el factor de potencia no depende de los valores de voltaje, sino sólo del ángulo de disparo del TRIAC α.

Como $\alpha = \omega t$, para conocer el tiempo de disparo τ basta con sustituir $t = \tau$ y despejar. Así:

$$\begin{aligned}
\tau =& \frac{\alpha}{\omega} \\
=& \frac{\alpha}{2\pi f} \\
=& \left.\frac{\frac{\pi}{2}}{2\pi f}\right|_{f=60\text{Hz}} \\
=& \frac{1}{4 \times 60\text{Hz}} = \frac{1}{240\text{Hz}} \\
=& 0.004166\bar{6}\text{s} \\
\approx& 4.2\text{ms}
\end{aligned}$$

Un problema importante a considerar es que la ecuación $P_f = \left[\frac{1}{\pi}\left(\pi - \alpha + \frac{\sin(2\alpha)}{2}\right)\right]^{\frac{1}{2}}$ no es biyectiva (tiene una componente periódica senoidal) y por lo tanto no es posible calcular su inversa de forma analítica. En otras palabras, no es posible obtener una expresión algebraica para calcular $\alpha\,(P_f)$, y por tampoco es posible inferir $\tau\,(P_f)$.

Existen varias soluciones alterna a este problema. Por ejemplo, pueden utilizarse varios métodos de aproximación numérica para cada uno de los segmentos de la curva y utilizar el más adecuado para en cada uno de los rangos de interés. Otros métodos que requieren un número mucho menor de cálculos hacen uso de una tabla de valores discretos (por ejemplo incrementos de 1 % en el factor de potencia) e interpolan linealmente entre estos, dejando el error como una perturbación a corregir por el controlador (véase Tabla 1).

Tabla 1: Relación entre factor de potencia y tiempo de disparo de TRIAC
Tabla para interpolación con incrementos de 5 % y alimentación de AC a 60Hz

Factor de potencia [%]	Tiempo de disparo [ms]	Factor de potencia [%]	Tiempo de disparo [ms]	Factor de potencia [%]	Tiempo de disparo [ms]
100	0.000	65	4.464	30	6.232
95	2.060	60	4.734	25	6.487
90	2.696	55	4.993	20	6.750
85	3.157	50	5.245	15	7.030
80	3.538	45	5.493	10	7.334
75	3.874	40	5.738	5	7.688
70	4.179	35	5.984	0	8.203

2.6. Bus I²C

I²C es un protocolo serial inventado por Phillips y diseñado para conectar dispositivos de baja velocidad mediante interfaces de dos hilos (Figura 7). El protocolo permite un número virtualmente ilimitado de dispositivos interconectados donde más de uno puede ser un dispositivo maestro. El bus I2C es popular debido a su facilidad de uso y fácil configuración. Sólo es necesario definir la velocidad máxima del bus, que está conformado por dos cables con resistencias pull-up [3].

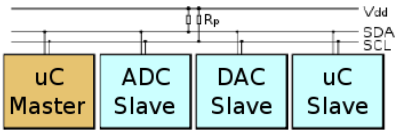


Figura 7: Bus I²C

I²C utiliza solamente dos cables: SCL (reloj) y SDA (datos). La transferencia de datos es serial y transmite paquetes de 8 bits con velocidades de hasta 5MHz. Además, es requisito que cada dispositivo esclavo tenga una dirección de 7 bits que (el bit más significativo se utiliza para indicar si el paquete es una lectura o una escritura) debe ser única en el bus. Los dispositivos maestros no necesitan dirección ya que estos generan la señal de reloj y coordinan a los dispositivos esclavos [3].

3. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

- 1 Arduino UNO o Arduino Mega
- 1 TRIAC BT138 o BT139
- 4 diodos 1N4007 o puente rectificador equivalente
- 1 optoacoplador MOC 3021
- 1 optoacoplador 4N25
- 1 foco incandescente (NO AHORRADOR NI LED)
- 1 resistencia de 15kΩ
- 1 resistencia de 10kΩ
- 2 resistencia de 4k7Ω
- 1 resistencia de 1kΩ
- 2 resistencia de 470Ω
- 2 resistencia de 330Ω
- 1 protoboard o circuito impreso equivalente
- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Cables y conectores varios

4. Instrucciones

1. Alambre el circuito mostrado en las Figuras 8 y 9.
2. Realice los programas de la Subsección 4.3
3. Analice los programas de la subsección 4.3 y realice los experimentos propuestos en la sección 5.

4.1. Paso 1: Alambrado

El proceso de alambrado de esta práctica considera dos circuitos. El primer circuito (véase Figura 8) opera con corriente alterna e integra un detector de cruce por cero y un convertidor AC-AC con base en un TRIAC. Ambos subcircuitos cuentan con optoacopladores que servirán como interfaz para una conexión segura al circuito de DC.

El segundo circuito (véase Figura 9) está encargado de detectar el cruce por cero y enviar la señal de activación al TRIAC en el momento oportuno de acuerdo con la potencia requerida por el usuario (el brillo del foco) mediante una interfaz gráfica.

Para este fin, se alambra la señal del subcircuito detector de cruce por cero a un pin te interrupción del Arduino que iniciará un *timer* en hardware y enviará la señal de activación al TRIAC una vez que transcurra el tiempo de activación para obtener así la potencia deseada. Asimismo, el Arduino recibirá via I^2C de la Raspberry Pi la potencia solicitada por el usuario mediante una interfaz web.

Cabe mencionar que, al ser un circuito completamente digital, al GPIO de la Raspberry Pi podría configurársele un pin en modo interrupción para recibir la señal del detector de cruce por cero y otro para el envío de la señal de activación del TRIAC. Sin embargo, el paquete `RPi.GPIO` para el control de la GPIO con Python no soporta el uso de interrupciones de timer en hardware, por lo que no es posible garantizar que la señal de activación del TRIAC será enviada sin retrasos. Es por este motivo que se utiliza un Arduino como auxiliar.

4.1.1. Circuito de potencia en AC

Alambre primero el circuito de corriente alterna de la Figura 8 tras verificar los valores de las resistencias de los optoacopladores. Considere que si la resistencia de gatillo es muy grande, el optoacoplador no recibirá suficente corriente y no encenderá lo suficiente como para disparar el fotosensor. Por otro lado, si la resistencia es demasiado pequeña el optoacoplador se quemará irremediablemente.

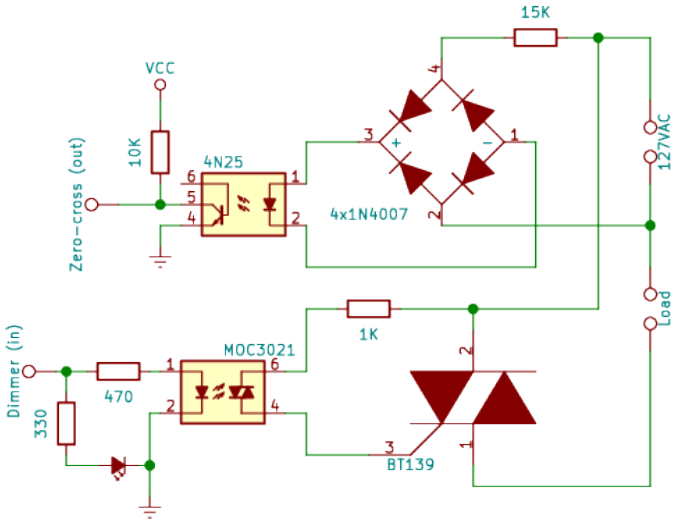


Figura 8: Circuito de potencia en AC

Tras alambrar el circuito, es una buena idea probar el detector de cruce por cero con un osciloscopio, o al menos con un led, que deberá encender tenuemente. De igual manera, conviene probar el encendido del foco inyectando 5V al MOC que acopla al TRIAC.

Tabla 2: Conexiones I^{2}C entre Raspberry Pi y un Arduino

Pin Raspberry		Conexión			Pin Arduino UNO	Pin Arduino Mega
3	(GPIO2)	Raspberry Pi SDA	→	Arduino SDA	A4	SDA (PIN 20)
5	(GPIO3)	Raspberry Pi SCL	→	Arduino SCL	A5	SCL (PIN 21)
6	(GND)	Raspberry Pi GND	→	Arduino GND	GND	GND

Importante

Asegúrese de verificar con un multímetro que el circuito de AC está debidamente aislado y que no se tienen valores mayores a 5V en el segmento de DC. De otro modo podría quemar su Arduino y su Raspberry Pi.

Continúe el alambrado del circuito.

4.1.2. Circuito de control en DC

Alambre el circuito de corriente directa de la Figura 9 tras verificar la tensión de las señales optoacopladas conectando el bus I^{2}C entre la Raspberry Pi y el Arduino como ilustran la Tabla 2 y la Figura 9. Hay tutoriales que sugieren utilizar un convertidor de niveles de voltaje cuando se conecta una Raspberry Pi a un arduino mediante I^{2}C, especialmente cuando la Raspberry Pi opera a 3.3V. Esto **NO** es necesario si la Raspberry Pi está configurada como dispositivo maestro o *master* y el Arduino como dispositivo esclavo o *slave*.

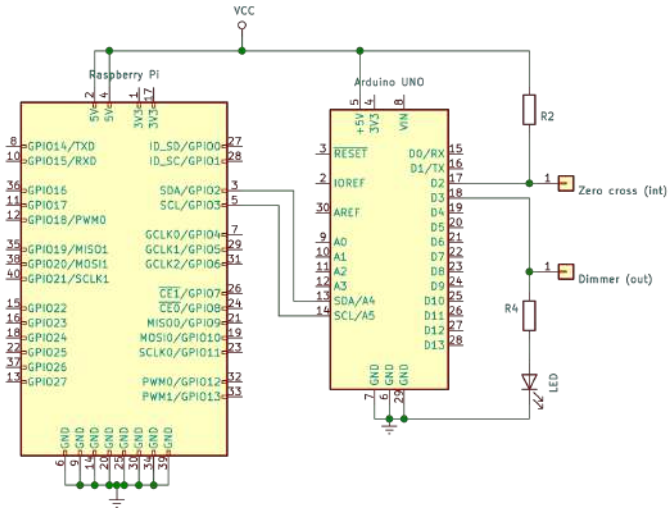


Figura 9: Circuito de control en DC

Esto es posible debido a que el Arduino no tiene resistencias de acoplamiento a positivo o *pull-up* integradas, mientras que los pines I^{2}C de la Raspberry Pi están conectados internamente a la línea de 3.3V mediante resistencias de 1.8kΩ. Por este motivo, tendrán que quitarse las resistencias de *pull-up* a cualquier otro dispositivo esclavo que se conecte al bus I^{2}C de la Raspberry Pi.[6]

A continuación pruebe el alambrado del circuito de DC con el programa de prueba del Apéndice A. El programa es muy simple, pues sólo configura interrupciones y cambia el estado de los pines cuando estas se producen.

Código ejemplo 1: `arduino-code-i2c.cpp:14` — Dirección asignada al dispositivo esclavo

```
1 #define I2C_SLAVE_ADDR 0x0A
```

[6]Para más información sobre el papel de las resistencias de acoplamiento a positivo o *pull-up* en un bus I^{2}C se puede consultar http://dsscircuits.com/articles/effects-of-varying-i2c-pull-up-resistors

Al terminar el alambrado debería tener completo el citcuito de la Figura 10

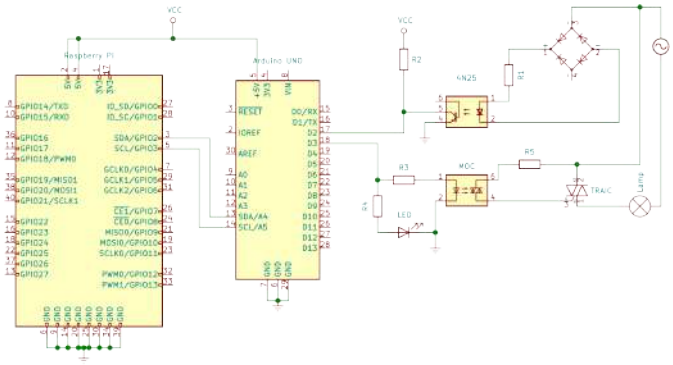


Figura 10: Diagrama de circuito alambrado completo

4.2. Paso 2: Configuración de comunicaciones I²C

Primero ha de configurarse la Raspberry Pi para funcionar como dispositivo maestro o *master* en el bus I²C. Para esto, inicie la utilidad de configuración de la Raspberry Pi con el comando

```
# raspi-config
```

y seleccione la opción 5: Opciones de Interfaz (*Interfacing Options*) y active la opción `P5` para habilitar el I²C.

A continuación, verifique que el puerto I²C no se encuentre en la lista negra. Edite el archivo `/etc/modprobe.d/raspi-blacklist.conf` y revise que la línea `blacklist spi-bcm2708` esté comentada con #.

Código ejemplo 2: /etc/modprobe.d/raspi-blacklist.conf

```
# blacklist spi and i2c by default (many users don't need them)
# blacklist i2c-bcm2708
```

Como paso siguiente, se habilita la carga del driver I²C. Esto se logra agregando la línea `i2c-dev` al final del archivo `/etc/modules` si esta no se encuentra ya allí.

Por último, se instalan los paquetes que permiten la comunicación mediante el bus I²C y se habilita al usuario predeterminado *pi* (o cualquier otro que se esté usando) para acceder al recurso.

```
# apt-get install i2c-tools python-smbus
# adduser pi i2c
```

Reinicie la Raspberry Pi y pruebe la configuración ejecutando `i2cdetect -y 1` para buscar dispositivos conectados al bus I²C. Debería ver una salida como la siguiente:

```
\$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

4.3. Paso 3: Control en lazo abierto de la potencia de una carga resistiva

Antes de proceder, verifique conexiones con un multímetro en busca de corto circuitos. En particular verifique que los circuitos de AC y DC funcionan de manera independiente y que existe una impedancia infinita entre pines optoacoplados.

Con la Raspberry Pi configurada, basta con generar los dos programas para transferir la potencia de salida deseada de la Raspberry Pi al Arduino quien se encargará cortar el flujo de corriente en el instante correcto para obtener la potencia deseada.

Primero, es necesario configurar al Arduino como dispositivo esclavo e inicializar el bus I^2C, tal como se muestra en los Códigos de Ejemplo 3 y 4. Las comunicaciones via I^2C son asíncronas, por lo que se requerirá almacenar la potencia deseada en una variable global que será leída por la función que atenderá las peticiones de datos del dispositivo maestro (la Raspberry Pi).

Código ejemplo 3: `arduino-code-i2c.cpp:14` — Dirección asignada al dispositivo esclavo

```
#define I2C_SLAVE_ADDR 0x0A
```

Código ejemplo 4: `arduino-code-i2c.cpp:28-33` — Configuración del bus I^2C y funciones de control

```
  // Configure I2C to run in slave mode with the defined address
  Wire.begin(I2C_SLAVE_ADDR);
  // Configure the handler for received I2C data
  Wire.onReceive(i2c_received_handler);
  // Configure the handler for request of data via I2C
  Wire.onRequest(i2c_request_handler);
```

Tanto el envío como la recepción de datos se realizan byte por byte, por lo que es necesario convertir la potencia (*float*) en un arreglo de bytes que pueda ser transmitido. Esto se hace en la funciones `i2c_request_handler` e `i2c_received_handler` tal como se ilustra en los Códigos de Ejemplo 5 y 6.

Código ejemplo 5: `arduino-code-i2c.cpp:46-48` — Escritura de flotantes en el bus I^2C

```
void i2c_request_handler(){
  Wire.write((byte*) &power, sizeof(float));
}
```

Código ejemplo 6: `arduino-code-i2c.cpp:54-62` — Lectura de flotantes del bus I^2C

```
void i2c_received_handler(int count){
  float f;
  byte *fp;
  if(count != 4) return;
  fp = &f;
  for(byte i = 0; i < count; ++i)
    fp[i] = (byte)Wire.read();
  power = f;
}
```

Del lado de la Raspberry Pi, primero ha inicializarse el bus I^2C y posteriormente se realizarán las lecturas en un poleo o bucle infinito, cada una de las cuales se irá almacenando en un archivo bitácora. La inicialización del bus requiere de una simple línea (véase Código de Ejemplo 7).

Código ejemplo 7: `raspberry-code-i2c.py:26` — Configuración del bus I^2C

```
  try:
```

La conversión de un arreglo de bytes a punto flotante en Python no es inmediata. Para esta opreación se utilizará la librería `struct` que empaquetará y desempaquetará flotantes (*float*) en listas de 4 bytes que pueden ser enviados o recibidos del arduino via I^2C tal como se muestra en los Códigos de Ejemplo 8 y 9

Código ejemplo 8: `raspberry-code-i2c.py:37-44` — Escritura de flotantes en el bus I^2C

```
def writePower(pwr):
  try:
    data = struct.pack('<f', pwr) # Packs number as float
    # Creates a message object to write 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
    i2c.i2c_rdwr(msg)  # Performs write
  except:
    pass
```

Código ejemplo 9: `raspberry-code-i2c.py:25-35` — Lectura de flotantes del bus I^2C

```
def readPower():
  try:
    # Creates a message object to read 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write
    data = list(msg)   # Converts stream to list
    pwr = struct.unpack('<f', ''.join([chr(c) for c in data]))
    # print('Received temp: {} = {}'.format(data, pwr))
    return pwr
  except:
    return None
```

El resto del programa es trivial, pues consiste sólo en solicitar al usuario un valor de potencia y enviarlo al Arduino.

Por conveniencia, los códigos completos de los programas de ejemplo se encuentran en los Apéndices A a C.

5. Experimentos

1. [6pt] Alambre el circuito completo y combine el código de los Apéndices A a C para poder controlar la intensidad del brillo del foco incandecente con la Raspberry Pi usando los valores tecleados en la consola (porentaje de 0–100 % de la potencia total).

2. [4pt] Modifique el código anterior para que la consola presente la potencia real modulada por el Arduino (equivalente en potencia del tiempo de encendido en milisegundos).

3. [+3pt] Modifique el código del punto 2 para que el arduino pueda modular la potencia del foco incandecente entre 0 % y 100 % con una resolución máxima de 1 %. Imprima la potencia reportada por el arduino con un dígito decimal.

4. [+2pt] Con base en lo aprendido, modifique el código delpunto 3 para que la Raspberry Pi sirva una página web donde se pueda modificar con un control gráfico la potencia de encendido del foco.

6. Referencias

Referencias

[1] *4N25 Optocoupler, Phototransistor Output, with Base Connection.* Vishay Semiconductors, 8 2010. Revised: January, 2010.

[2] Muhammad H Rashid. Power electronic, devices, circuits, and applications. *Handbook, Second Edition, Burlington*, 2006.

[3] I2C Info: A Two-wire Serial Protocol. I2c info – i2c bus, interface and protocol, 2020. https://i2c.info/, Last accessed on 2020-03-01.

[4] *MOC3021 Random-Phase Optoisolator TRIAC Driver Output.* Fairchild Semiconductors, 8 2010. Revised: January, 2010.

[5] The Arduino Project. Introduction to the arduino board, 2020. https://www.arduino.cc/en/reference/board, Last accessed on 2020-03-01.

A. Programa Ejemplo: `arduino-test-dc.cpp`

src/arduino–test–dc.cpp

```
1  // Digital 2 is Pin 2 in UNO
2  #define ZXPIN 2
3  // Digital 3 is Pin 3 in UNO
4  #define TRIAC 3
5
6  // Globals
7  volatile bool flag = false;
8  int step = 0;
9  int pfactor = 0;
10
11 // Prototypes
12 float read_temp(void);
13 float read_avg_temp(int count);
14
15 /**
16 * Setup the Arduino
17 */
18 void setup(void){
19   // Setup interupt pin (input)
20   pinMode(ZXPIN, INPUT);
21   attachInterrupt(digitalPinToInterrupt(ZXPIN), zxhandle, RISING);
22   // Setup output (triac) pin
23   pinMode(TRIAC, OUTPUT);
24   // Blink led on interrupt
25   pinMode(13, OUTPUT);
26
27   // Setup the serial port to operate at 9600bps
28   Serial.begin(9600);
29 }
30
31 void loop(){
32   if(!flag){
33     // Slack until next interrupt
34     delay(1);
35     return
36   }
37   // Reset flag & blink led
38   flag = !flag;
39   digitalWrite(13, LOW);
40   // Enable power for STEP milliseconds
41   if(pfactor < 8)
42     digitalWrite(3, HIGH);
43   delay(pfactor);
44   // Disable power
45   digitalWrite(3, LOW);
46   // Increment/reset power factor every 20 cycles
47   if(++step >= 19){
48     step = 0;
49     if(++pfactor > 8) pfactor = 0;
50   }
51 }
52
53 void zxhandle(){
54   flag = true;
55   digitalWrite(13, HIGH);
56 }
```

B. Programa Ejemplo: `arduino-code-i2c.cpp`

src/arduino–code–i2c.cpp

```cpp
#include <Wire.h>

// Constants
#define I2C_SLAVE_ADDR 0x0A
#define BOARD_LED 13

// Global variables
float power = 0;

// Prototypes
void i2c_received_handler(int count);
void i2c_request_handler(int count);

/**
* Setup the Arduino
*/
void setup(void){
  // Configure I2C to run in slave mode with the defined address
  Wire.begin(I2C_SLAVE_ADDR);
  // Configure the handler for received I2C data
  Wire.onReceive(i2c_received_handler);
  // Configure the handler for request of data via I2C
  Wire.onRequest(i2c_request_handler);

  // Setup the serial port to operate at 56.6kbps
  Serial.begin(56600);

  // Setup board led
  pinMode(BOARD_LED, OUTPUT);
}

/**
* Handles data requests received via the I2C bus
* It will immediately reply with the power stored
*/
void i2c_request_handler(){
  Wire.write((byte*) &power, sizeof(float));
}

/**
* Handles received data via the I2C bus.
* Data is stored in local variable power.
*/
void i2c_received_handler(int count){
  float f;
  byte *fp;
  if(count != 4) return;
  fp = &f;
  for(byte i = 0; i < count; ++i)
    fp[i] = (byte)Wire.read();
  power = f;
}

void loop(){
  char buffer[20];
  sprintf(buffer, "Power = %.2f\n", power);
  Serial.write(buffer)
  delay(1000);
}
```

C. Programa Ejemplo: `raspberry-code-i2c.py`

src/raspberry–code–i2c.py

```python
import smbus
import struct
import time

# Arduino's I2C device address
SLAVE_ADDR = 0x0A # I2C Address of Arduino 1

# Initialize the I2C bus;
# RPI version 1 requires smbus.SMBus(0)
i2c = smbus.SMBus(1)

def readPower():
  try:
    # Creates a message object to read 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write
    data = list(msg)   # Converts stream to list
    pwr = struct.unpack('<f', ''.join([chr(c) for c in data]))
    # print('Received temp: {} = {}'.format(data, pwr))
    return pwr
  except:
    return None

def writePower(pwr):
  try:
    data = struct.pack('<f', pwr) # Packs number as float
    # Creates a message object to write 4 bytes from SLAVE_ADDR
    msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
    i2c.i2c_rdwr(msg)  # Performs write
  except:
    pass

def main():
  while True:
    try:
      power = input("Power? ")
      power = float(power)
      if power >= 0 and power <= 100:
        writePower(power)
        print("\tPower set to {}".format(readPower()))
      else:
        print("\tInvalid!")
    except:
      print("\tInvalid!")

if __name__ == '__main__':
  main()
```

Práctica 1:
Punto de acceso con enrutador en puente
Sistemas Embebidos Avanzados

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a configurar la Raspberry Pi como punto de acceso inalámbrico con conexión compartida a Internet via un adaptador tipo puente.

2. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Si se cuenta con una Raspberry Pi sin WiFi integrado (e.j Raspberry Pi2), se precisará de un adaptador WiFi USB compatible para la misma.

3. Instrucciones

1. Configure la Raspberry Pi como punto de acceso inalámbrico.
2. Configure la Raspberry Pi como enrutador.

3.1. Paso 1: Configuración de la Raspberry Pi como punto de acceso inalámbrico

Para operar como punto de acceso la Raspberry Pi necesita tener instalado el software apropiado, incluyendo un servidor DHCP para proporcionar a los dispositivos que se conecten una dirección IP.

Se comienza por instalar los paquetes `DNSMasq` y `HostAPD`:

```
# apt-get install dnsmasq hostapd bridge-utils
```

Si están ejecutándose los servicios, deténgalos a fin de poder reconfigurarlos

```
# systemctl stop dnsmasq
# systemctl stop hostapd
```

3.1.1. Configuración del adaptador y el cliente DHCP

Para configurar una red independiente con servidor DHCP la Raspberry Pi debe tener asignada una dirección IP estática en el adaptador inalámbrico que proveerá la conexión. Debido a que la Raspberry Pi tiene un procesador pequeño, se configurará para servir en una red privada clase C, es decir con direcciones IP del tipo 198.168.x.x. Así mismo, se supondrá que el dispositivo inalámbrico utilizado es `wlan0`.

Para configurar la dirección IP estática edite el archivo de configuración `/etc/dhcpcd.conf` como superusuario:

```
interface wlan0
    static ip_address=192.168.100.254/24
    nohook wpa_supplicant
```

A continuación, reinicie el cliente DHCP

```
# service dhcpcd restart
```

3.1.2. Configuración del servidor DHCP

El siguiente paso consiste en configurar el servidor DHCP, provisto por el servicio `dnsmasq`.

De manera predeterminada el archivo de configuración `/etc/dnsmasq.conf` contiene mucha información que no es necesaria, por lo que es más fácil comenzar desde cero. Respáldelo y cree uno nuevo con el siguiente texto:

Archivo 1: `/etc/dnsmasq.conf`

```
# Use the required wireless interface - usually wlan0
interface=wlan0
# Reserve 20 IP addresses, set the subnet mask, and lease time
dhcp-range=192.168.100.200,192.168.100.220,255.255.255.0,24h
```

Esta configuración proporcionará 20 direcciones IP entre 192.168.100.200 y 192.168.100.220, válidas durante 24 horas.

Ahora debe iniciarse el servidor DHCP

```
# systemctl start dnsmasq
```

3.1.3. Configuración del punto de acceso

Para configurar el punto de acceso se debe editar el archivo de configuración `/etc/hostapd/hostapd.conf` con los parámetros adecuados.

Respáldelo y cree uno nuevo con el siguiente texto:

Archivo 2: `/etc/hostapd/hostapd.conf`

```
# Wireless interface
interface=wlan0
# To be used later
# bridge=br0
# Specification: IEEE802.11. Will be commented out later
driver=nl80211
# The SSID or name of the network
ssid=Raspbberry
# Password of the network
wpa_passphrase=12345678
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
# Mode and frequency of operation
hw_mode=g
# Broadcast channel
channel=5
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
rsn_pairwise=CCMP
```

La configuración ingresada configura la Raspberry Pi para crear una red inalámbrica tipo 802.11g en el canal 5 de nombre *Raspbberry* y contraseña `12345678` con seguridad WPA2.

Los modos de operación posibles son:

- a = IEEE 802.11a (5 GHz)
- b = IEEE 802.11b (2.4 GHz)
- g = IEEE 802.11g (2.4 GHz)

Importante: Tanto el nombre de la red o SSID y la contraseña no deben entrecomillarse. La contraseña debe tener entre 8 y 64 caracteres. **Cambie el SSID a `Raspberry_Apellido` para evitar conflictos.**

Ahora edite el archivo `/etc/default/hostapd` y reemplace la línea que comienza con `#DAEMON_CONF` con:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

3.1.4. Configuración e inicio del punto de acceso

Finalmente, habilite los servicios para iniciar el punto de acceso:

```
# systemctl unmask hostapd
# systemctl enable hostapd
# systemctl start hostapd
```

Verifique que los servicios se están ejecutando

```
# systemctl status hostapd
# systemctl status dnsmasq
```

Nota: El servicio `hostapd` requiere acceso exclusivo a la tarjeta de red inalámbrica que podría estar ocupada por el proceso `wpa_supplicant`. Si `hostapd` se reusara a iniciar indicando un error tal como *Could not configure driver mode nl80211 driver initialization failed*, termine los procesos que puedan estar utilizando la tarjeta de red inalámbrica, por ejemplo ejecutando `killall wpa_supplicant`.

3.2. Paso 2: Habilitación de enrutado de paquetes

El siguiente paso consiste en habilitar el reenvío de paquetes IP. Para este fin, edite el archivo `/etc/sysctl.conf` y habilite la siguiente línea:

```
net.ipv4.ip_forward=1
```

Agregue una mascarada para el tráfico de salida proveniente del adaptador de red ethernet `eth0`:

```
# iptables -t nat -A  POSTROUTING -o eth0 -j MASQUERADE
```

Agregue la regla de guardado de tablas IP.

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Edite el archivo `/etc/rc.local` y agregue la siguiente línea justo antes de `exit 0` para que la configuración se active de manera automática al iniciar la Raspberry Pi.

```
iptables-restore < /etc/iptables.ipv4.nat
```

Ahora reinicie la Raspberry Pi y verifique que las configuraciones sigan vigentes.

Utilizando un dispositivo inalámbrico, busque las redes disponibles. Si la configuración se realizó correctamente, podrá ver en la lista el SSID de la lista que se espcificó en el *hostapd*, y debería poder conectarse a esta con la contraseña almacenada.

Si se habilitaron las conexiones entrates vía SSH, debería poder iniciar sesión también al conectarse de manera inalámbrica con el siguiente comando:

```
$ ssh pi@192.168.100.254
```

La Raspberry pi es ahora un punto de acceso permanente y otros dispositivos pueden coectarse a ella vía su dirección IP.

3.3. Paso 3: Conexión a internet compartida

Uno de los usos más comunes que se le da a la Raspberry Pi es el de punto de acceso para compartir una conexión inalámbrica vía el puerto Ethernet. Así, cualquiera que se conecte al punto de acceso de la Raspberry Pi tendrá también acceso a internet. Es decir, la Raspberry Pi funcionará como una especie de mini-router.

Para lograr esto, es necesario definir un *puente* entre el dispositivo inalámbrico el Ethernet. Esta red virtual tipo puente permitirá la transmisión del tráfico entre las dos interfaces, pero para que el puenteo funcione es necesario que el servidor DHCP no intente asignar direcciones IP a ninguna de las interfaces involucradas en el puente.

Para lograr esto, edite el archivo `/etc/dhcpcd.conf` y agregue las siguientes líneas al final:

```
denyinterfaces eth0 wlan0
```

A continuación se crea la conexión virtual tipo puente, a la cual llamaremos `br0`

```
# brctl addbr br0
```

El siguiente paso consiste en enlazar los puertos de red, en particular `eth0` y `br0`

```
# brctl addif br0 eth0
```

Ahora, se necesitan crear los archivos de configuración que permiten a los servicios operar correctamente la conexión de puente.

Estos archivos son los siguientes:

Archivo 3: `/etc/systemd/network/bridge-br0.netdev`

```
[NetDev]
Name=br0
Kind=bridge
```

Archivo 4: `/etc/systemd/network/bridge-br0-slave.network`

```
[Match]
Name=eth0

[Network]
Bridge=br0
```

Archivo 5: `/etc/systemd/network/bridge-br0.network`

```
[Match]
Name=br0

[Network]
Address=192.168.10.100/24
Gateway=192.168.10.1
DNS=8.8.8.8
```

Finalmente, reinicie el servicio `systemd-networkd` y verifique que el puente ha sido creado exitosamente:

```
# systemctl restart systemd-networkd
# brctl show br0
```

Ahora es necesario reconfigurar el punto de acceso para utilizar el puente como parte de la conexión. Edite el archivo `/etc/hostapd/hostapd.conf` agregando (o quitando la marca de comentario) la línea `bridge=br0` justo debajo de `interface=wlan0`, y elimine o comente la línea del driver, por ejemplo:

Archivo 6: `/etc/hostapd/hostapd.conf` líneas 1 a 6

```
# Wireless interface
interface=wlan0
# Use the bridge connection
bridge=br0
# Not needed anymore
# driver=nl80211
```

Para terminar de configurar el punto de acceso con enrutador en puente reinicie la Raspberry Pi y habilite punto de acceso inalámbrico

```
# systemctl unmask hostapd
# systemctl enable hostapd
# systemctl start hostapd
```

Ahora debería ser capaz de conectarse a internet mediante la Raspberry Pi. Notará que la Raspberry Pi se ha vuelto invisible y cualquier dispositivo que conecte a esta se comportará como si se hubiera conectado al enrutador principal via cable Ethernet. El puente se encarga de redireccionar los paquetes, y esto incluye configuraciones DHCP, por lo que el punto de acceso de la Raspberry Pi no asigna nuevas direcciones IP a los dispositivos conectados.

Si tiene acceso, se pueden verificar las conexiones con el comando

```
ip addr
```

Observará que tanto `wlan0` como `eth0` carecen de direcciones IP, las cuales ahora son controladas por el puente. Es possible asignar una dirección IP estática al puente para acceder al dispositivo si así se requiere, pero en general esto no suele ser necesario.

4. Experimentos

1. [5 pts] Conecte su teléfono celular o algún otro dispositivo móvil a la red cableada del laboratorio utilizando la Raspberry Pi.

5. Cuestionario

1. [1.0pt] Explique qué cambios deben hacerse a la configuración cuando la Raspberry Pi en modo puente se conecta a una red sin servidor DHCP usando IP estática.
2. [2.0pt] Explique qué cambios deben realizarse a la configuración para que la Raspberry Pi reciba una dirección IP para administración remota cuando se conecta en modo puente a una red con servidor DHCP.
3. [3.0pt] Explique qué cambios deben realizarse a la configuración para que la Raspberry Pi inicie (arranque) en modo puente sin necesidad de comandos adicionales.

Práctica 2:
Control remoto de la Raspberry Pi via WiFi y servidor web

Sistemas Embebidos Avanzados

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a configurar la Raspberry Pi como punto de acceso inalámbrico que permita acceder a un servidor web simple que controle el puerto GPIO de la misma.

2. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

Si se cuenta con una Raspberry Pi sin WiFi integrado (e.j Raspberry Pi2), se precisará de un adaptador WiFi USB compatible para la misma.

Además, el alumno necesitará el alambrado de la Práctica 3.

3. Instrucciones

1. Alambre el circuito tal y como se detalla en la Práctica 3.
2. Realice los ejercicios y experimentos de la Práctica 3.

3.1. Paso 1: Configuración de la Raspberry Pi como punto de acceso inalámbrico

Para operar como punto de acceso la Raspberry Pi necesita tener instalado el software apropiado, incluyendo un servidor DHCP para proporcionar a los dispositivos que se conecten una dirección IP.

Se comienza por instalar los paquetes `DNSMasq` y `HostAPD`:

```
# apt-get install dnsmasq hostapd
# pip3 install -U python-magic
```

Si están ejecutándose los servicios, deténgalos a fin de poder reconfigurarlos

```
# systemctl stop dnsmasq
# systemctl stop hostapd
```

3.1.1. Configuración del adaptador y el cliente DHCP

Para configurar una red independiente con servidor DHCP la Raspberry Pi debe tener asignada una dirección IP estática en el adaptador inalámbrico que proveerá la conexión. Debido a que la Raspberry Pi tiene un procesador pequeño, se configurará para servir en una red privada clase C, es decir con direcciones IP del tipo 198.168.x.x. Así mismo, se supondrá que el dispositivo inalámbrico utilizado es `wlan0`.

Para configurar la dirección IP estática edite el archivo de configuración `/etc/dhcpcd.conf` como superusuario:

```
interface wlan0
    static ip_address=192.168.1.254/24
    nohook wpa_supplicant
```

A continuación, reinicie el cliente DHCP

```
# service dhcpcd restart
```

3.1.2. Configuración del servidor DHCP

El siguiente paso consiste en configurar el servidor DHCP, provisto por el servicio `dnsmasq`.

De manera predeterminada el archivo de configuración `/etc/dnsmasq.conf` contiene mucha información que no es necesaria, por lo que es más fácil comenzar desde cero. Respáldelo y cree uno nuevo con el siguiente texto:

```
# Use the require wireless interface - usually wlan0
interface=wlan0
# Reserve 20 IP addresses, set the subnet mask, and lease time
dhcp-range=192.168.1.200,192.168.1.220,255.255.255.0,24h
```

Esta configuración proporcionará 20 direcciones IP entre 192.168.1.200 y 192.168.1.220, válidas durante 24 horas.

Ahora debe iniciarse el servidor DHCP

```
systemctl start dnsmasq
```

3.1.3. Configuración del punto de acceso

Para configurar el punto de acceso se debe editar el archivo de configuración `/etc/hostapd/hostapd.conf` con los parámetros adecuados.

Respáldelo y cree uno nuevo con el siguiente texto:

```
# Wireless interface
interface=wlan0
# Specification: IEEE802.11
driver=nl80211
# The SSID or name of the network
ssid=Raspbberry
# Password of the network
wpa_passphrase=12345678
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
# Mode and frequency of operation
hw_mode=g
# Broadcast channel
channel=5
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
rsn_pairwise=CCMP
```

La configuración ingresada configura la Raspberry Pi para crear una red inalámbrica tipo 802.11g en el canal 5 de nombre *Raspbberry* y contraseña `12345678` con seguridad WPA2.

Los modos de operación posibles son:

- a = IEEE 802.11a (5 GHz)
- b = IEEE 802.11b (2.4 GHz)
- g = IEEE 802.11g (2.4 GHz)

Importante: Tanto el nombre de la red o SSID y la contraseña no deben entrecomillarse. La contraseña debe tener entre 8 y 64 caracteres. **Cambie el SSID a `Raspberry_Apellido` para evitar conflictos.**

Ahora edite el archivo `/etc/default/hostapd` y reemplace la línea que comienza con `#DAEMON_CONF` con:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

3.1.4. Configuración e inicio del punto de acceso

Finalmente, habilite los servicios para iniciar el punto de acceso:

```
# systemctl unmask hostapd
# systemctl enable hostapd
# systemctl start hostapd
```

Verifique que los servicios se están ejecutando

```
# systemctl status hostapd
# systemctl status dnsmasq
```

Nota: El servicio `hostapd` requiere acceso exclusivo a la tarjeta de red inalámbrica que podría estar ocupada por el proceso `wpa_supplicant`. Si `hostapd` se reusara a iniciar indicando un error tal como *Could not configure driver mode nl80211 driver initialization failed*, termine los procesos que puedan estar utilizando la tarjeta de red inalámbrica, por ejemplo ejecutando `killall wpa_supplicant`.

3.2. Paso 2: Configuración de la Raspberry Pi como servidor Web

Raspbian es una variante d Debian, por lo que se le dará bien servir páginas web de forma segura, especialmente cuando se utiliza Apache. Sin embargo, configurar Apache para enlazarse con Python y operar la GPIO no es una tarea trivial, por lo que en esta práctica se utilizará un servidor web simple basado en el `BaseHTTPRequestHandler` que incorpora el paquete `http.server` de Python.

Para habilitar un servidor web en Python, basta con heredar de la clase `BaseHTTPRequestHandler` e implementar el método `do_GET` para que imprima el código HTML al socket vía el método `self.wfile.write` tal como se muestra en el Código de Ejemplo 1.

Código ejemplo 1: Archivo `simple-webserver.py`

```
1  from http.server import BaseHTTPRequestHandler, HTTPServer
2
3  class WebServer(BaseHTTPRequestHandler):
4    def do_GET(self):
5      self.send_response(200)
6      self.send_header("Content-type", "text/html")
7      self.end_headers()
8      self.wfile.write(bytes("<html><body>Hola Mundo!!!</body></html>", "utf-8"))
9
10 def main():
11   webServer = HTTPServer(("192.168.1.254", 80), WebServer)
12   print("Servidor iniciado")
13   print ("\tAtendiendo solicitudes entrantes")
14   try:
15     webServer.serve_forever()
16   except KeyboardInterrupt:
17     pass
18   webServer.server_close()
19   print("Server stopped.")
```

Script de Python presentado inicia un servidor web que atiende todas las peticiones entrantes vía la interfaz con la IP 192.168.1.254 (el punto de acceso) en el puerto 80 (HTTP predeterminado). A cada petición se le devolverá una señal de estado HTTP200 u *OK*, seguido por código HTML. Es importante aclarar que para cada archivo servido se debe especificar el tipo de archivo en la cabecera.

Importante: El puerto 80 (y en general todos los puertos por debajo del 2014) están reservados para servicios de sistema, por lo que Python fallará al intentar levantar el servidor web en este puerto. Existen dos opciones: puede ejecutar el proceso como superusuario con `sudo` o bien usar otro puerto como el 8080.

Genere el archivo `simple-webserver.py` y ejecútelo. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: `http://192.168.1.254`.

Con ligeras modificaciones es posible servir cualquier tipo de archivo. Todas las peticiones ingresadas en la barra de direcciones del navegador llegarán por método *GET*, por lo que deberán ser procesadas en el método `do_GET`,

accediendo al atributo de clase `self.path`, relativo al directorio de trabajo. En caso de que no se proporcione un archivo, `do_GET` tendrá que proporcionar la página por defecto, típicamente nombrada `index.html`, pero que en este caso por motivos didácticos se ha nombrado `user_interface.html` (véase Código de Ejemplo 2).

Código ejemplo 2: Método `do_GET` del archivo webserver.py

```
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:])
```

Para servir un archivo se tiene que verificar que el éste exista, proporcionar su tipo mime en la cabecera y devolver los datos como una cadena binaria. Esto se realiza en el método interno `_serve_file`. Si el archivo no se encontrare, se devuelve un error *HTTP404* como se muestra en el Código de Ejemplo 3:

Código ejemplo 3: Método `_serve_file` del archivo webserver.py

```
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())
```

La interacción cliente servidor se lleva a cabo de manera similar. Dependerá de si los datos se envían por método *GET* o *POST*, de los cuales se prefiere el segundo pues hace más difícil inyectar datos. De manera análoga se utiliza el método `do_POST` que recibe y procesa los datos. En esta práctica, se utilizan dados codificados mediante JSON para hacer llamadas asíncronas del cliente y sin respuesta por parte del servidor (véase Código de Ejemplo 4).

Código ejemplo 4: Método `do_POST` del archivo webserver.py

```
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
  # Después 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 recnocidos")
```

El método `do_POST` preentado en el Código de Ejemplo 4 interpreta los datos recibidos como cadenas de texto unicode de 8 bits (*utf-8*) que contienen objetos en JSON que son decodificados a un diccionario de Python. El

diccionario es después enviado al método interno `_parse_post` mostrado en el Código de Ejemplo 5 que analiza los datos y realiza las acciones pertinentes.

Código ejemplo 5: Método `_parse_post` del archivo webserver.py

```
def _parse_post(self, json_obj):
  if not 'action' in json_obj or not 'value' in json_obj:
    return
  switcher = {
    'led'     : leds,
    'marquee' : marquee,
    'numpad'  : bcd
  }
  func = switcher.get(json_obj['action'], None)
  if func:
    print('\tCall{}({})'.format(func, json_obj['value']))
    func(json_obj['value'])
```

Genere los archivos `webserver.py` y `user_interface.html` (véase Apéndices A y B), luego ejecute el scrypt de Python. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: `http://192.168.1.254`. Debería ver una pantalla similar a la siguiente.

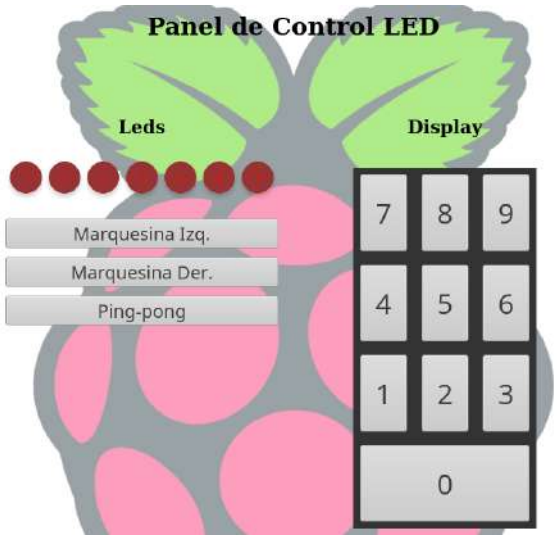


Figura 1: Caption: Intefaz de usuario del controlador de Leds en la Raspberry Pi.

4. Experimentos

Integre el código de la Práctica 3 en un archivo python llamado `led_manager.py` y que ofrezca las siguientes funciones:

1. [2 pts] Encendido del del 1–7 al presionar el boton adecuado
2. [2 pts] Desplegado de la marquesina izquierda al presionar el boton adecuado
3. [2 pts] Desplegado de la marquesina derecha al presionar el boton adecuado
4. [2 pts] Desplegado de la marquesina tipo ping-pong al presionar el boton adecuado
5. [2 pts] Desplegado del dígito correcto en el display de 7 segmentos al presionar el boton correspondiente

A. El archivo **webserver.py**

Código ejemplo 6: Archivo webserver.py

```
import os
import sys
import json
import magic
from led_manager import leds, bcd, marquee
from http.server import BaseHTTPRequestHandler, HTTPServer
# import time
# import time

# Nombre o dirección IP del sistema anfitrión del servidor web
# address = "localhost"
address = "192.168.1.254"
# Puerto en el cual el servidor estará atendiendo solicitudes HTTP
# El default de un servidor web en produción debe ser 80
port = 8080


class WebServer(BaseHTTPRequestHandler):
  """Sirve cualquier archivo encontrado en el servidor """
  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())


  """Sirve el archivo de interfaz de usuario"""
  def _serve_ui_file(self):
    if not os.path.isfile("user_interface.html"):
      err = "user_interface.html not found."
      self.wfile.write(bytes(err, "utf-8"))
      print(err)
      return
    try:
      with open("user_interface.html", "r") as f:
        content = "\n".join(f.readlines())
    except:
      content = "Error reading user_interface.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 = {
      'led'     : leds,
      'marquee' : marquee,
      'numpad'  : bcd
    }
    func = switcher.get(json_obj['action'], None)
    if func:
      print('\tCall{}({})'.format(func, json_obj['value']))
      func(json_obj['value'])


  """do_GET controla todas las solicitudes recibidas vía GET, es
  decir, páginas. Por seguridad, no se analizan variables que lleguen
  por esta vía"""
  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:])



  """do_POST controla todas las solicitudes recibidas vía POST, es
  decir, envíos de formulario. Aquí se gestionan los comandos para
  la 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
    # Después 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 recnocidos")

def main():
  # Inicializa una nueva instancia de HTTPServer con el
  # HTTPRequestHandler definido en este archivo
  webServer = HTTPServer((address, port), WebServer)
  print("Servidor iniciado")
  print ("\tAtendiendo solicitudes en http://{}:{}".format(
    address, port))

  try:
    # Mantiene al servidor web ejecutándose en segundo plano
    webServer.serve_forever()
  except KeyboardInterrupt:
    # Maneja la interrupción de cierre CTRL+C
    pass
  except:
    print(sys.exc_info())
  # Detiene el servidor web cerrando todas las conexiones
  webServer.server_close()
  # Reporta parada del servidor web en consola
  print("Server stopped.")


# Punto de anclaje de la función main
if __name__ == "__main__":
  main()
```

B. El archivo `user_interface.html`

Código ejemplo 7: Archivo user_interface.html

```
<!DOCTYPE html>
<html>
<head>
<title>Panel de Control LED - Raspberry Pi</title>
<meta charset="ISO-8859-1">
<style type="text/css">
  html{
    width:                  100vw;
    height:                 100vh;
    min-width:              100vw;
    min-height:             100vh;
    margin:                 0;
    padding:                0;
    box-sizing:             border-box;
    overflow:               hidden;
  }

  body{
    width:                  800px;
    height:                 100vh;
    max-width:              800px;
    min-height:             100vh;
    padding:                0;
    margin:                 0 auto;
    box-sizing:             border-box;
  }

  body::after{
    content:                "";
    position:               absolute;
    top:                    0;
    left:                   0;
    bottom:                 0;
    right:                  0;
    z-index:                -1;
    opacity:                0.5;
    background-image:       url('img/raspberry.png');
    background-repeat:      no-repeat;
    background-attachment:  fixed;
    background-position:    center;
    background-size:        contain;
  }

  header{
    width:                  100%;
    padding:                0;
    margin:                 0;
    box-sizing:             border-box;
    text-align:             center;
  }

  h1{
    height:                 2em;
    padding:                1em 0;
  }

  .container{
    width:                  100%;
    padding:                0;
    margin:                 0;
    box-sizing:             border-box;
    display:                flex;
    flex-direction:         row;
  }

  .column{
    flex:                   1 0 0;
    display:                flex;
    flex-direction:         column;
  }

  .numpad{
    width:                  6.5em;
    height:                 12.75em;
    background-color:       #333333;
    margin:                 0.5em auto;
    padding:                0;
    font-size:              28pt;
    flex-wrap:              wrap;
  }

  .numbutton{
    font-size:              inherit;
    flex:                   1 0 0;
    margin:                 0.25em;
  }

  .ledstrip{
    justify-content:        space-evenly;
    width:                  90%;
    height:                 4em;
    padding:                0;
    margin:                 0.5em auto;
    /*background-color:       #333333;*/

  }

  .ledbutton{
    width:                  3.5em;
    height:                 3.5em;
    color:                  #F00;
    background-color:       #933;
    border-radius:          50%;
    margin:                 0.25em;
    padding:                0;
    border:                 none;
    box-shadow:             0px 4px 5px rgba(0, 0, 0,
     0.2);
  }

  .ledbutton:hover{
    background-color:       #F33;
  }

  .widebutton{
    font-size:              18pt;
    width:                  90%;
    margin:                 0.25em auto;
  }

  .on{
    color:                  #0F0;
    background-color:       #3F3;
  }
</style>
</head>
<body>
  <header><h1>Panel de Control LED</h1></header>
  <section class="container">
    <article class="column">
      <header><h2>Leds</h2></header>
      <section class="container ledstrip">
        <button class="ledbutton" onclick="handle(this,
     'led', 1)">1</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 2)">2</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 3)">3</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 4)">4</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 5)">5</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 6)">6</button>
        <button class="ledbutton" onclick="handle(this,
     'led', 7)">7</button>
      </section>
      <button class="widebutton" onclick="handle(this, '
    marquee', 'left')">Marquesina Izq.</button>
      <button class="widebutton" onclick="handle(this, '
    marquee', 'right')">Marquesina Der.</button>
```

```
      <button class="widebutton" onclick="handle(this, 'marquee', 'pingpong')">Ping-pong</button>
    </article>
    <article class="column">
      <header><h2>Display</h2></header>
      <section class="container numpad">
        <button class="numbutton" onclick="handle(this, 'numpad', 7)">7</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 8)">8</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 9)">9</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 4)">4</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 5)">5</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 6)">6</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 1)">1</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 2)">2</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 3)">3</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 0)">0</button>
      </section>
    </article>
  </section>
</body>
</html>
<script language="javascript">
<!--
function deactivateAll(){
  var buttons = document.getElementsByTagName('button');
  for(button in buttons)
    button.classList.remove("on")
}

function activate(sender){
  if(sender == null)
    return;
  sender.classList.add("on");
}

function handle(sender, action, value){
  // deactivateAll();
  // activate(sender);
  submit(action, value);
}

function submit(action, value){
  var xhr = new XMLHttpRequest();
  xhr.open("POST", window.location.href, true);
  xhr.setRequestHeader('Content-Type', 'application/json');
  xhr.send(JSON.stringify({
    'action' : action,
    'value' :  value,
  }));
}
//-->
</script>
```

Práctica 3:
Lectura de datos analógicos usando Arduino y la Raspberry Pi

Sistemas Embebidos Avanzados

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno aprenderá a leer e interpretar señales analógicas con un microcontrolador.

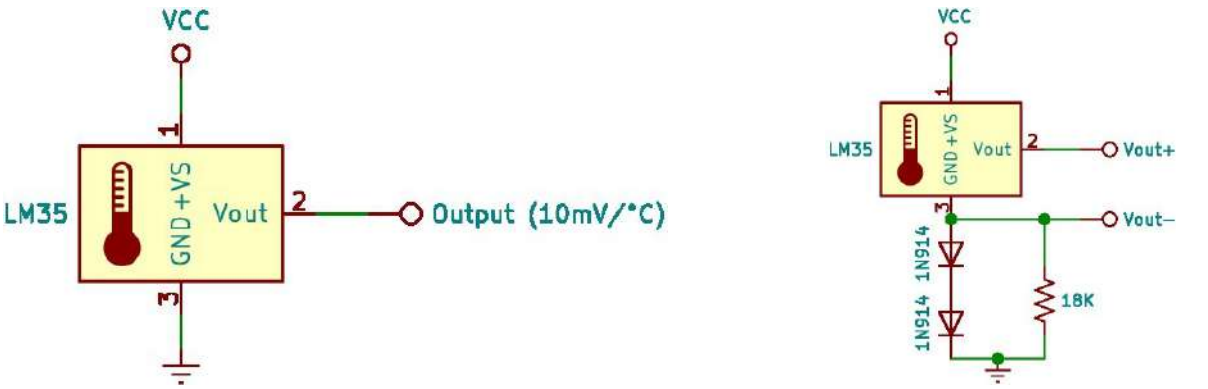
2. Introducción

La presente práctica resume los pasos a seguir para leer una señal analógica con un microcontrolador. En particular, se interesa en la lectura de la temperatura registrada por un sensor LM35 mediante un Arduino UNO/Mega. Los datos registrados serán posteriormente enviados vía I^2C a una Raspberry Pi para llevar una bitácora de temperatura que podrá ser desplegada en un navegador web.

2.1. El sensor LM35

El circuito integrado LM35 es un sensor de temperatura cuya salida de voltaje o respuesta es linealmente proporcional a la temperatura registrada en escala centígrada. Una de las principales ventajas del LM35 sobre otros sensores lineales calibrados en Kelvin, es que no se requiere restar constantes grandes para obtener la temperatura en grados centígrados. El rango de este sensor va de -55°C a 150°C con una precisión que varía entre 0.5°C y 1.0°C dependiendo la temperatura medida [1].

Las configuraciones más comunes para este integrado se muestran en la Figura 1. La configuración (Figura 1a) básica, la más simple posible pues sólo requiere conectar al integrado LM35 entre Vcc y Gnd, permite medir temperaturas entre 2°C a 150°C. Por otro lado, la configuración (Figura 1a) clásica permite medir en todo el rango completo del sensor, es decir entre -55°C y 150°C, pero requiere de un par de diodos 1N914 y una resistencia de 18KΩ para proporcionar los voltajes de referencia. En ambos casos, el LM35 ofrece una diferencial de $10mV/^oC$, por lo que los voltajes medidos rara vez excederán de 2V respecto a tierra.



(a) Configuración básica: 2°C a 150°C

(b) Configuración clásica: -55°C a 150°C

Figura 1: Configuraciones típicas del LM35

Cuando opera en rango completo y las temperaturas registradas son inferiores a cero, se permite un flujo de corriente inverso entre los pines GND y V_{out} del LM35, es decir, una salida de voltaje negativo respecto a la referencia. Debido a que el LM35 no puede generar voltajes inferiores respecto a la referencia del circuito (tierra) se utilizan dos diodos 1N914 en serie colocados en el pin de referencia o tierra del LM35 (véase Figura 1b) para elevar el voltaje del subcircuito del LM35 aproximadamente 1.2V por encima del voltaje de referencia o tierra general. Así, cuando el LM35 entre en contacto con temperaturas negativas, el voltaje de diodo o V_{DD} referenciable mediante la resistencia de 18K hará posible que el voltaje de V_{out+} sea inferior al de V_{out-} y pueda calcularse la diferencia, tal como se muestra en la Tabla 1.

2.2. Convertidor Analógico—Digital

Para leer la señal del LM35 se requiere de un Convertidor Analógico Digital o ADC (por sus siglas en inglés: *Digital-Analog Converter*). Un ADC se elige con base en dos factores clave: su precisión y su tiempo de muestreo. Debido a que la aplicación del ADC será convertir mediciones de temperatura y los cambios de temperatura son muy lentos,[1] puede obviarse el tiempo de muestreo. En cuanto a la precisión, los convertidores A/D más comunes son de 8 y 10 bits, de los cuales ha de elegirse uno.

Tabla 1: Salida de un LM35 en rango completo

Temp [°C]	V_{out+} [V]
-55	0.65
0	1.20
50	1.70
100	2.20
150	2.70

La precisión del ADC se calcula tomando en cuenta el rango de operación y la precisión del componente analógico a discretizar. El LM35 tiene un rango de 205°C, una diferencial de voltaje $\Delta V = 10mV/^oC$ y una precisión máxima de 0.5°C, por lo que el sensor entregará un máximo de 2.5V respecto al voltaje de referencia del mismo, con incrementos de 5mV. Debido a que 256 valores para un rango de 205°C en incrementos de 0.5°C (es decir 410 valores) es claramente insuficiente para este sensor, por lo que será conveniente utilizar un convertidor A/D de 10 bits.

Un ADC típico de 10 bits convertirá las señales analógicas entre voltajes de referencia V_{Ref-} y V_{Ref+} como un entero con valores entre 0 y 1023, interpretando los valores V_{Ref-} como 0 lógico y V_{Ref+} como 1023 de manera aproximadamente lineal. El decir, la lectura obtenida es directamente proporcional al voltaje dentro del rango, estimable mediante la fórmula:

$$V_{out} = value \times \frac{V_{Ref+} - V_{Ref-}}{1024} \tag{1}$$

En una configuración simple, V_{Ref-} y V_{Ref+} se conectan internamente dentro del Arduino a tierra y V_{CC} respectivamente. Esto simplifica la fórmula como:

$$V_{out} = value \times \frac{5V}{1024} = value \times 0.00488V \tag{2}$$

En lo concerniente al Arduino, éste incorpora un convertidor analógico-digital de 10 bits con soporte para voltaje de referencia V_{Ref+}, denominado *AREF* según las especificaciones del mismo [2]. Considerando que el LM35 en rango completo entrega hasta 2.05V ($10mV \times (150 - -55) = 2.05V$) la mayor parte de los 1024 valores jamás serán ocupados. Por este motivo, conviene sacar partido del pin de voltaje de referencia *AREF* del Arduino mediante un divisor de voltaje (véase Figura 2). En consecuencia, el pin *AREF* requerirá de un divisor de voltaje con salida de 2.73V tal como se muestra en la Figura 2 para dar mayor precisión al convertidor A/D.

Con esta nueva configuración, se puede calcular de nueva cuenta la precisión del sensor digital una vez decodificado el valor analógico leído del LM35 dividiendo los 2.73V de referencia entre los 1024 valores posibles que entrega el ADC como sigue:

$$\Delta V = \frac{2.73V}{1024} = 0.00267V \tag{3}$$

Debido a que la resolución máxima del sensor LM35 determinada por su factor de incertidumbre es de 0.5°C equivalentes a 0.005V, ambas configuraciones (con y sin el divisor de voltaje) serán adecuadas para operar al sensor.

2.3. Bus I^2C

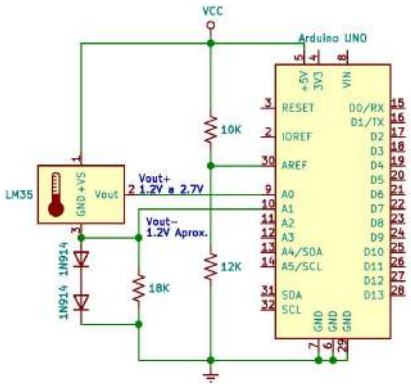


Figura 2: Circuito medidor de temperatura LM35 con Arduino

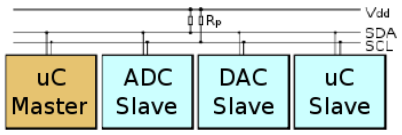


Figura 3: Bus I^{2}C

I^{2}C es un protocolo serial inventado por Phillips y diseñado para conectar dispositivos de baja velocidad mediante interfaces de dos hilos (Figura 3). El protocolo permite un número virtualmente ilimitado de dispositivos interconectados donde más de uno puede ser un dispositivo maestro. El bus I2C es popular debido a su facilidad de uso y fácil configuración. Sólo es necesario definir la velocidad máxima del bus, que está conformado por dos cables con resistencias pull-up [3].

I^{2}C utiliza solamente dos cables: SCL (reloj) y SDA (datos). La transferencia de datos es serial y transmite paquetes de 8 bits con velocidades de hasta 5MHz. Además, es requisito que cada dispositivo esclavo tenga una dirección de 7 bits que (el bit más significativo se utiliza para indicar si el paquete es una lectura o una escritura) debe ser única en el bus. Los dispositivos maestros no necesitan dirección ya que estos generan la señal de reloj y coordinan a los dispositivos esclavos [3].

3. Material

Se asume que el alumno cuenta con un una Raspberry Pi con sistema operativo Raspbian e interprete de Python instalado. Se aconseja encarecidamente el uso de *git* como programa de control de versiones.

- 1 Arduino UNO, Arduino Mega, o Convertidor A/D
- 1 sensor de temperatura LM35 en encapsulado TO-220 o TO-92
- 2 Diodos 1N914
- 2 resistencia de 10kΩ
- 1 resistencia de 12kΩ[2]
- 1 resistencia de 18kΩ
- 1 Condensador de 0.1μF
- 1 protoboard o circuito impreso equivalente
- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Cables y conectores varios

4. Instrucciones

1. Alambre el circuito mostrado en la Figura 2.
2. Realice los programas de las Subsecciones 4.3 y 4.4
3. Analice los programas de las subsecciones 4.3 y 4.4, realice los experimentos propuestos en la sección 5.

[2]La resistencia de 12kΩ puede reemplazarse con resistencias de 13kΩ a 20kΩ dependiendo del voltaje de los diodos.

Tabla 2: Conexiones I^{2}C entre Raspberry Pi y un Arduino

Pin Raspberry		Conexión			Pin Arduino UNO	Pin Arduino Mega
3	(GPIO2)	Raspberry Pi SDA	→	Arduino SDA	A4	SDA (PIN 20)
5	(GPIO3)	Raspberry Pi SCL	→	Arduino SCL	A5	SCL (PIN 21)
6	(GND)	Raspberry Pi GND	→	Arduino GND	GND	GND

4.1. Paso 1: Alambrado

El proceso de alambrado de esta práctica considera dos circuitos. El primer circuito, mostrado en las Figura 2, permite obtener valores discretos del sensor de temperatura LM35. El segundo circuito (Figura 4) consiste en la interfaz de conexión vía I^{2}C entre el microcontrolador que lee el LM35 y la Raspberry Pi que genera los reportes y grafica los resultados.

Alambre primero el subcircuito formado por los dos diodos, el integrado LM35 y la resistencia de 18kΩ. Paso seguido, alimente el subcircuito con 5V y mida la diferencia de potencial existente entre V_{OUT-} y GND. Utilice el valor medido en la fórmula $V_{AREF} = 1.5V + V_{OUT-}$ para calcular los valores de las resistencias que se conectarán al pin AREF del Arduino.

Importante

Asegúrese que $V_{AREF} \leq V_{OUT-}\Big|_{Temp=150^oC}$ para evitar quemar el Arduino.

Continúe el alambrado del circuito. Es conveniente colocar un capacitor de 0.1μF entre VCC y GND para rectificar el voltaje de entrada eliminar cualquier oscilación parásita que pudiere afectar el funcionamiento del LM35. La presencia de este componente es opcional pero altamente recomendada.

Tras alambrar el primer circuito realice el experimento prueba indicado en la Subsección 4.2.

A continuación conecte el bus I^{2}C entre la Raspberry Pi y el Arduino como ilustran la Tabla 2 y la Figura 5. Hay tutoriales que sugieren utilizar un convertidor de niveles de voltaje cuando se conecta una Raspberry Pi a un arduino mediante I^{2}C, especialmente cuando la Raspberry Pi opera a 3.3V. Esto **NO** es necesario si la Raspberry Pi está configurada como dispositivo maestro o *master* y el Arduino como dispositivo esclavo o *slave*.

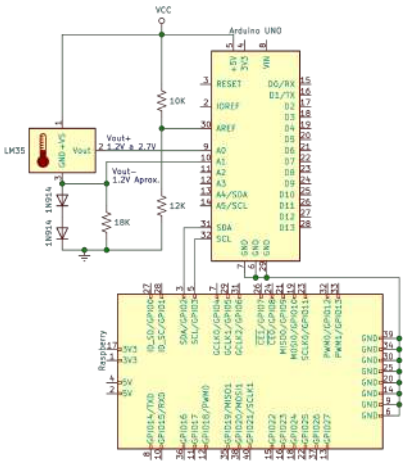


Figura 4: Circuito completo

Esto es posible debido a que el Arduino no tiene resistencias de acoplamiento a positivo o *pull-up* integradas, mientras que los pines I^{2}C de la Raspberry Pi están conectados internamente a la línea de 3.3V mediante resistencias de 1.8kΩ. Por este motivo, tendrán que quitarse las resistencias de *pull-up* a cualquier otro dispositivo esclavo que se conecte al bus I^{2}C de la Raspberry Pi.[3]

4.2. Paso 2: Lectura del sensor LM35

Antes de proceder, verifique conexiones con un multímetro en busca de corto circuitos. En particular verifique que exista una impedancia muy alta entre los pines 5V, GND y AREF del Arduino.

Para leer la temperatura con el Arduino se necesitan convertir los valores discretos leídos por el ADC del microcontrolador en valores de temperatura. Esto se puede realizar mediante un simple análisis debido a la linearidad del LM35. Se tienen dos lecturas en el ADC: V_{OUT+} y V_{OUT-}, de las cuales la segunda es la referencia del LM35 y por lo tanto, la diferencia entre

[3]Para más información sobre el papel de las resistencias de acoplamiento a positivo o *pull-up* en un bus I^{2}C se puede consultar http://dsscircuits.com/articles/effects-of-varying-i2c-pull-up-resistors

[4]Imagen obtenida de https://create.arduino.cc/projecthub/aardweeno/59817b

estos voltajes será proporcional a la temperatura en escala centígrada. Esto expresado matemáticamente es:

$$T[^oC] \propto V_{diff} = V_{OUT+} - V_{OUT-}$$

o bien

$$T[^oC] = k \times V_{diff} = k \times \left(V_{OUT+} - V_{OUT-}\right)$$

lo que implica que en $T = 0^oC; V_{OUT+} = V_{OUT-} \rightarrow V_{diff} = 0$

Es necesario entonces calcular la constante de proporcionalidad k. Sabemos que el ADC entregará lecturas de 0 a 1024 para los voltajes registradoes entre GND y AREF (0V y 2.72V respectivamente), además de que $1^oC = 0.01V$. Luego entonces

$$T[^oC] = V_{diff} \times \frac{2.72[V]}{1024 \times 0.01[\frac{V}{^oC}]}$$

$$T[^oC] = V_{diff} \times \frac{2.72}{10.24}[^oC]$$

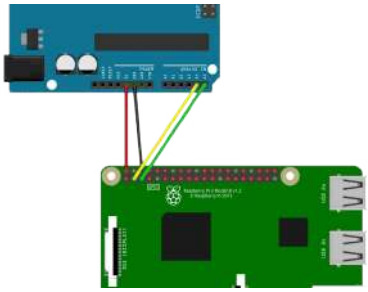


Figura 5: Conexión mediante I^2C de una Raspberry Pi con un Arduino UNO[4]

o bien, generalizando para todo voltaje de referencia:

$$T[^oC] = V_{diff} \times \frac{A_{REF}}{10.24}[^oC]$$

Esta fórmula de conversión de unidades deberá programarse en el microcontrolador que adquiera los valores discretos de temperatura del sensor. El programa de ejemplo para el Arduino se presenta a continuación:

Código ejemplo 1: `arduino-code.cpp`:37–51, `read_temp` function

```
    int vplus  = analogRead(0);
    int vminus = analogRead(1);
   int vdiff = vplus - vminus;

  float temp = vdiff * VAREF / 10.24f;
  return temp;
}
```

En ocasiones los valores pueden fluctuar ligeramente debido a ruido o variaciones de voltaje. Para evitar este tipo de imprecisiones es común utilizar técnicas de filtrado, y uno de los métodos más simples y comúnes es el promedio de varias lecturas consecutivas tal y como se muestra a continuación:

Código ejemplo 2: `arduino-code.cpp`:56–61, `read_avg_temp` function

```
  float avgtemp = 0;
  for(int i = 0; i < count; ++i)
    avgtemp += read_temp();
  return avgtemp / count;
}
```

Otro método mucho más eficaz y seguro es llevar un registro de las últimas N lecturas del sensor en un buffer circular y estimar el siguiente valor probable, descartando aquellas lecturas que estén fuera de rango posible, es decir cuando $\Delta t \geq \epsilon_0$.

4.3. Paso 3: Configuración de comunicaciones I^2C

Primero ha de configurarse la Raspberry Pi para funcionar como dispositivo maestro o *master* en el bus I^2C. Para esto, inicie la utilidad de configuración de la Raspberry Pi con el comando

```
# raspi-config
```

y seleccione la opción 5: Opciones de Interfaz (*Interfacing Options*) y active la opción `P5` para habilitar el I^{2}C.

A continuación, verifique que el puerto I^{2}C no se encuentre en la lista negra. Edite el archivo `/etc/modprobe.d/raspi-blacklist.conf` y revise que la línea `blacklist spi-bcm2708` esté comentada con #.

Código ejemplo 3: /etc/modprobe.d/raspi-blacklist.conf

```
# blacklist spi and i2c by default (many users don't need them)
# blacklist i2c-bcm2708
```

Como paso siguiente, se habilita la carga del driver I^{2}C. Esto se logra agregando la línea `i2c-dev` al final del archivo `/etc/modules` si esta no se encuentra ya allí.

Por último, se instalan los paquetes que permiten la comunicación mediante el bus I^{2}C y se habilita al usuario predeterminado *pi* (o cualquier otro que se esté usando) para acceder al recurso.

```
# apt-get install i2c-tools python-smbus
# adduser pi i2c
```

Reinicie la Raspberry Pi y pruebe la configuración ejecutando `i2cdetect -y 1` para buscar dispositivos conectados al bus I^{2}C. Debería ver una salida como la siguiente:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

4.4. Paso 4: Bitácora de temperatura via I^{2}C

Con la Raspberry Pi configurada, basta con generar los dos programas para transferir las temperaturas registradas en el Arduino a la Raspberry Pi que se encargará de almacenar esta información en un archivo o bitácora.

Primero, es necesario configurar al Arduino como dispositivo esclavo e inicializar el bus I^{2}C, tal como se muestra en los Códigos de Ejemplo 4 y 5. Las comunicaciones via I^{2}C son asíncronas, por lo que se requerirá almacenar la temperatura en una variable global que será leída por la función que atenderá las peticiones de datos del dispositivo maestro (la Raspberry Pi).

Código ejemplo 4: `arduino-code-i2c.cpp:16` — Dirección asignada al dispositivo esclavo

```
1 #define I2C_SLAVE_ADDR 0x0A
```

Código ejemplo 5: `arduino-code-i2c.cpp:37-42` — Configuración del bus I^{2}C y funciones de control

```
1   // Configure I2C to run in slave mode with the defined address
2   Wire.begin(I2C_SLAVE_ADDR);
3   // Configure the handler for received I2C data
4   Wire.onReceive(i2c_received_handler);
5   // Configure the handler for request of data via I2C
6   Wire.onRequest(i2c_request_handler);
```

El envío de datos se realiza byte por byte, por lo que es necesario convertir la medición de temperatura (*float*) en un arreglo de bytes que pueda ser transmitido. Esto se hace en la función `i2c_request_handler` con una llamada a `Wire.write` tal como se ilustra en el Código de Ejemplo 6.

Código ejemplo 6: `arduino-code-i2c.cpp:58-60` — Envío asíncrono de datos

```
1 void i2c_request_handler(){
2   Wire.write((byte*) &temperature, sizeof(float));
3 }
```

Del lado de la Raspberry Pi, primero ha inicializarse el bus I^2C mediante el uso de la librería `smbus`[5]y posteriormente se realizarán las lecturas en un poleo o bucle infinito, cada una de las cuales se irá almacenando en un archivo bitácora. La inicialización del bus requiere de una simple línea (véase Código de Ejemplo 7).

Código ejemplo 7: `raspberry-code-i2c.py:26` — Configuración del bus I^2C

```
1 import smbus
2 import struct
3 # Initialize the I2C bus;
4 # RPI version 1 requires smbus.SMBus(0)
5 i2c = smbus.SMBus(1)
```

La conversión de un arreglo de bytes a punto flotante en Python no es inmediata. Para esta opreación se utilizará la librería `struct` (véase Código de Ejemplo 7) que tomará los cuatro paquetes de 1 byte recibidos vía I^2C del Arduino y los convertira en un *float*, como se muestra en el Código de Ejemplo 8. Nótese el símbolo < (menor qué) a la izquierda del especificador de formato f, el cual se utiliza para definir el endianness de la transmisión de la información.

Código ejemplo 8: `raspberry-code-i2c.py:28-34` — Lectura de flotantes del bus I^2C

```
1 def readTemperature():
2   try:
3     msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
4     i2c.i2c_rdwr(msg)
5     data = list(msg)
6     temp = struct.unpack('<f', ''.join([chr(c) for c in data]))
7     print('Received temp: {} = {}'.format(data, temp))
8     return temp
9   except:
10    return None
```

El resto del programa es trivial, pues consiste sólo en la escritura del *timestamp UNIX* y el valor de temperatura registrado en un archivo de texto y la lectura de datos del arduino cada segundo.

Por conveniencia, los códigos completos de los programas de ejemplo se encuentran en los Apéndices A a C.

5. Experimentos

1. [6pt] Modifique el código de la subsección 4.4 para que la Raspberry Pi imprima en pantalla los valores de temperatura leídos.

2. [4pt] Modifique el código de la subsección 4.4 la Raspberry Pi grafique el histórico de temperaturas registradas, leyendo los valores almacenados e ingresados en la bitácora.

3. [+5pt] Con base en lo aprendido, modifique el código de la subsección 4.4 para que la Raspberry Pi sirva una página web donde se pueda observar la gráfica de temperatura (histórico) desde la bitácora con resolución de hasta 1 minuto.

[5]La implementación de la práctica utiliza `smbus2` que es una reimplementación codificada exclusivamente en Python de la librería `smbus` que es un *wrapper* de la *smbuslib* de C.

6. Referencias

Referencias

[1] *LM35 Precision Centigrade Temperature Sensors*. Texas Instruments, August 1999. Revised: December, 2017.

[2] The Arduino Project. Introduction to the arduino board, 2020. https://www.arduino.cc/en/reference/board, Last accessed on 2020-03-01.

[3] I2C Info: A Two-wire Serial Protocol. I2c info – i2c bus, interface and protocol, 2020. https://i2c.info/, Last accessed on 2020-03-01.

A. Programa Ejemplo: `arduino-code.cpp`

src/arduino–code.cpp

```cpp
#define VAREF 2.7273

// Prototypes
float read_temp(void);
float read_avg_temp(int count);

/**
* Setup the Arduino
*/
void setup(void){
  // Configure ADC to use voltage reference from AREF pin (external)
  analogReference(EXTERNAL);
  // Set ADC resolution to 10 bits
  // analogReadResolution(10);

  // Setup the serial port to operate at 56.6kbps
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

/**
* Reads temperature in C from the ADC
*/
float read_temp(void){
  // The actual temperature
  int vplus  = analogRead(0);
  // The reference temperature value, i.e. 0 C
  int vminus = analogRead(1);
  // Calculate the difference. when V+ is smaller than V- we have negative temp
  int vdiff = vplus - vminus;
  /* Now, we need to convert values to the ADC resolution, AKA 2.72V/1024
  *  We also know that 1C = 0.01V so we can multiply by 2.72V / (0.01V/ C) = 272 C
  *  to get C instead of V. Analogously we can multiply VAREF by 100 but
  *  since we will divide per 1024, it suffice with dividing by 10.24
  */
  float temp = vdiff * VAREF / 10.24f;
  return temp;
}

/**
* Gets the average of N temperature reads
*/
float read_avg_temp(int count){
  float avgtemp = 0;
  for(int i = 0; i < count; ++i)
    avgtemp += read_temp();
  return avgtemp / count;
}

void loop(){
  float temp = read_avg_temp(5);
  Serial.print((int)temp);
  Serial.print(".");
  Serial.println((int)(10 * temp)  % 10);
  digitalWrite(13, HIGH);
  delay(5);
  digitalWrite(13, LOW);
  delay(5);
}
```

B. Programa Ejemplo: **raspberry-code-i2c.py**

src/raspberry–code–i2c.py

```
SLAVE_ADDR = 0x0A # I2C Address of Arduino 1

# Name of the file in which the log is kept
LOG_FILE = './temp.log'

# Initialize the I2C bus;
# RPI version 1 requires smbus.SMBus(0)
i2c = smbus.SMBus(1)

def readTemperature():
  try:
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)
    data = list(msg)
    temp = struct.unpack('<f', ''.join([chr(c) for c in data]))
    print('Received temp: {} = {}'.format(data, temp))
    return temp
  except:
    return None

def log_temp(temperature):
  try:
    with open(LOG_FILE, 'w+') as fp:
      fp.write('{} {} C\n'.format(
        time.time(),
        temperature
      ))
  except:
    return

def main():
  while True:
    try:
      cTemp = readTemperature()
      log_temp(cTemp)
      time.sleep(1)
    except KeyboardInterrupt:
      return

if __name__ == '__main__':
  main()
```

C. Programa Ejemplo: `arduino-code-i2c.cpp`

src/arduino–code–i2c.cpp

```
#include <Wire.h>

// Constants
#define VAREF 2.7273
#define I2C_SLAVE_ADDR 0x0A
#define BOARD_LED 13

// Global variables
float temperature = 0;

// Prototypes
void i2c_received_handler(int count);
void i2c_request_handler(int count);
float read_temp(void);
float read_avg_temp(int count);

/**
* Setup the Arduino
*/
void setup(void){
  // Configure ADC to use voltage reference from AREF pin (external)
  analogReference(EXTERNAL);
  // Set ADC resolution to 10 bits
  // analogReadResolution(10)

  // Configure I2C to run in slave mode with the defined address
  Wire.begin(I2C_SLAVE_ADDR);
  // Configure the handler for received I2C data
  Wire.onReceive(i2c_received_handler);
  // Configure the handler for request of data via I2C
  Wire.onRequest(i2c_request_handler);

  // Setup the serial port to operate at 56.6kbps
  Serial.begin(56600);

  // Setup board led
  pinMode(BOARD_LED, OUTPUT);

}

/**
* Handles data requests received via the I2C bus
* It will immediately send the temperature read as a float value
*/
void i2c_request_handler(){
  Wire.write((byte*) &temperature, sizeof(float));
}

/**
* Handles received data via the I2C bus.
* Data is forwarded to the Serial port and makes the board led blink
*/
void i2c_received_handler(int count){
  char received = 0;
  while (Wire.available()){
    received = (char)Wire.read();
    digitalWrite(BOARD_LED, received ? HIGH : LOW);
    Serial.println(received);
  }

}

/**
* Reads temperature in C from the ADC
```

```
*/
float read_temp(void){
  // The actual temperature
  int vplus  = analogRead(0);
  // The reference temperature value, i.e. 0 C
  int vminus = analogRead(1);
  // Calculate the difference. when V+ is smaller than V- we have negative temp
  int vdiff = vplus - vminus;
  // Temp = vdiff * VAREF / (1024 * 0.01)
  return vdiff * VAREF / 10.24f;
}

void loop(){
  temperature = read_temp();
  delay(100);
}
```