Práctica 6:
Lectura de datos analógicos usando Arduino y la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

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

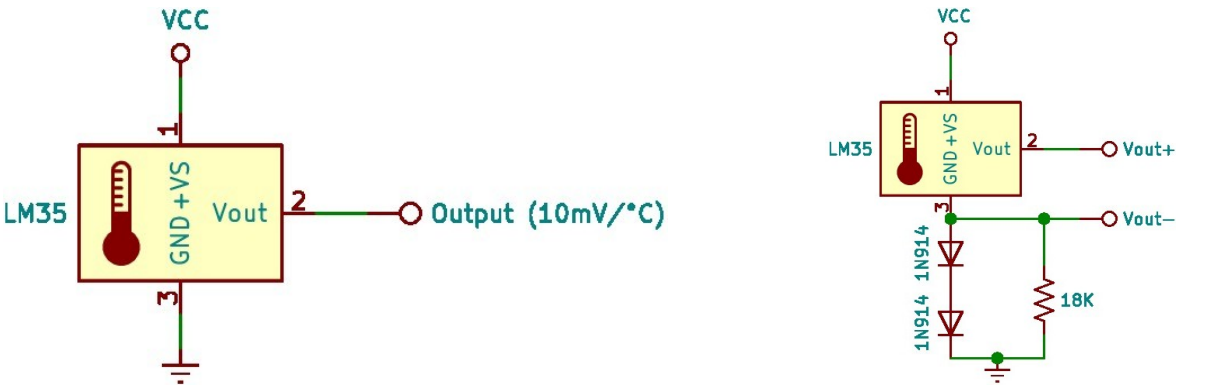
2. Introducción

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

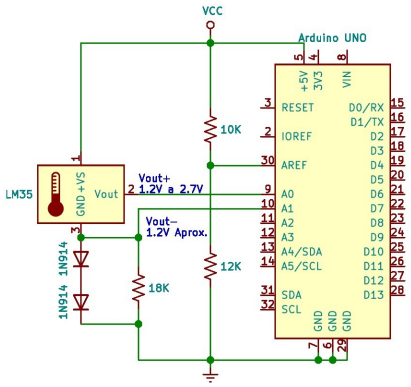


Figura 2: Circuito medidor de temperatura LM35 con Arduino

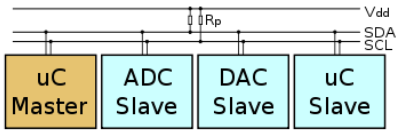


Figura 3: Bus I^{2}C

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 I^{2}C
- 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
- 1 cable USB-B para programar el Arduino
- 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 la 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 de 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.

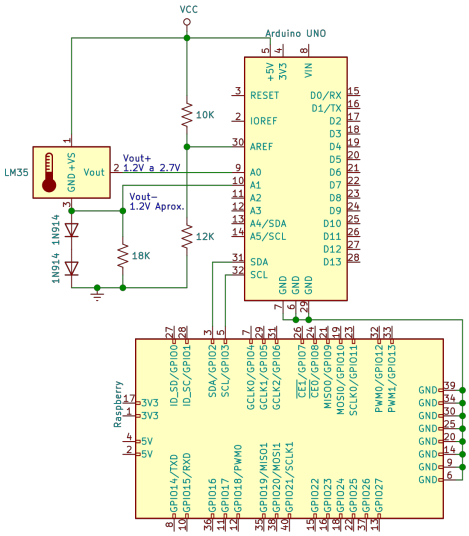


Figura 4: Circuito completo

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*.

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.[4]

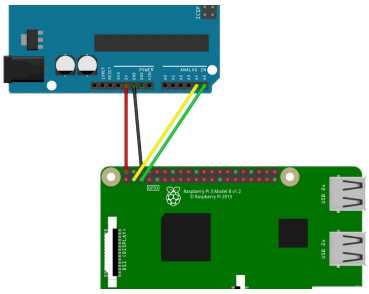


Figura 5: Conexión mediante I^{2}C de una Raspberry Pi con un Arduino UNO[3]

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.

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

[4] 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

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 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

$$T[^oC] = V_{diff} \times \frac{2.72[V]}{1024 \times 0.01[\frac{V}{^oC}]}$$
$$T[^oC] = V_{diff} \times \frac{2.72}{10.24}[^oC]$$

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.

TIP: Calibración

El voltaje de referencia V_{AREF} variará respecto a su valor teórico dependiendo de la tolerancia de las resistencias (típicamente $\pm 5\,\%$) y de las impedancias de los demás componentes conectados, incluyendo al Arduino mismo.

Se recomienda usar siempre un multímetro para corroborar el voltaje entre V_{AREF} y tierra y poder actualizar dicho valor en el código.[5]

El programa de ejemplo para el Arduino[6] se presenta a continuación:

Código ejemplo 1: `arduino-code.cpp`:40–54, `read_temp` function

```
float read_temp(void){
    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:

[5] El valor real de V_{AREF} variará entre 2.4V y 3.1V con resistencias con tolerancia de 5 %, equivalente a una variación de 70°C.

Código ejemplo 2: `arduino-code.cpp`:59–64, `read_avg_temp` function

```
float read_avg_temp(int count){
  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 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 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 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 python3-smbus
# adduser pi i2c
$ pip install smbus2
```

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: -- -- -- -- -- -- -- --
```

[6] Para compilar los archivos de código del Arduino utilice el Arduino IDE (no olvide cambiar la extensión de los archivos de `cpp` a `ino`) o bien instale las herramientas de compilación por línea de comandos con

```
# apt install gcc-avr binutils-avr gdb-avr avr-libc avrdude
```

y posteriormente utilice gcc-avr para compilar los archivos fuente.
Recuerde que no es posible programar el arduino via I²C.

4.4. Paso 4: Bitácora de temperatura via I²C

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 I²C, tal como se muestra en los Códigos de Ejemplo 4 y 5. Las comunicaciones via I²C 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 I²C 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 I²C mediante el uso de la librería `smbus`[7]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 I²C

```
1 import smbus2
2 import struct
3 # Initialize the I2C bus;
4 # RPI version 1 requires smbus.SMBus(0)
5 i2c = smbus2.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 I²C 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:28-35` — Lectura de flotantes del bus I²C

```
1  def readTemperature():
2    try:
3      msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
4      i2c.i2c_rdwr(msg)  # Performs write (read request)
5      data = list(msg)   # Converts stream to list
6      # list to array of bytes (required to decode)
7      ba = bytearray()
8      for c in data:
9        ba.append(int(c))
10     temp = struct.unpack('<f', ba)
```

[7]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.

```
    print('Received temp: {} = {}'.format(data, temp))
    return temp
  except:
```

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.

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

```
1
2  // Update this value with the voltage measured in AREF
3  // Do remember that resistors have 5% tolerance and that voltage
4  // in the pin may drop once you connect the Arduino
5  #define VAREF 2.7273
6
7  // Prototypes
8  float read_temp(void);
9  float read_avg_temp(int count);
10
11 /**
12 * Setup the Arduino
13 */
14 void setup(void){
15   // Configure ADC to use voltage reference from AREF pin (external)
16   analogReference(EXTERNAL);
17   // Set ADC resolution to 10 bits
18   // analogReadResolution(10);
19
20   // Setup the serial port to operate at 56.6kbps
21   Serial.begin(9600);
22   pinMode(13, OUTPUT);
23 }
24
25 /**
26 * Reads temperature in C from the ADC
27 */
28 float read_temp(void){
29   // The actual temperature
30   int vplus  = analogRead(0);
31   // The reference temperature value, i.e. 0ºC
32   int vminus = analogRead(1);
33   // Calculate the difference. when V+ is smaller than V- we have negative temp
34   int vdiff = vplus - vminus;
35   /* Now, we need to convert values to the ADC resolution, AKA 2.72V/1024
36   *  We also know that 1C = 0.01V so we can multiply by 2.72V / (0.01V/ºC) = 272ºC
37   *  to get C instead of V. Analogously we can multiply VAREF by 100 but
38   *  since we will divide per 1024, it suffice with dividing by 10.24
39   */
40   float temp = vdiff * VAREF / 10.24f;
41   return temp;
42 }
43
44 /**
45 * Gets the average of N temperature reads
46 */
47 float read_avg_temp(int count){
48   float avgtemp = 0;
49   for(int i = 0; i < count; ++i)
50     avgtemp += read_temp();
51   return avgtemp / count;
52 }
53
54 void loop(){
55   float temp = read_avg_temp(5);
56   Serial.print((int)temp);
57   Serial.print(".");
58   Serial.println((int)(10 * temp)  % 10);
59   digitalWrite(13, HIGH);
60   delay(250);
61   digitalWrite(13, LOW);
62   delay(250);
63 }
```

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 = smbus2.SMBus(1)

def readTemperature():
  try:
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 4)
    i2c.i2c_rdwr(msg)  # Performs write (read request)
    data = list(msg)   # Converts stream to list
    # list to array of bytes (required to decode)
    ba = bytearray()
    for c in data:
      ba.append(int(c))
    temp = struct.unpack('<f', ba)
    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

```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);
}
```