Programa 3:
Lectura de datos analógicos via I^2C en la Raspberry Pi

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

Entrega Martes 10 de Octubre, 2021

1. Objetivo

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

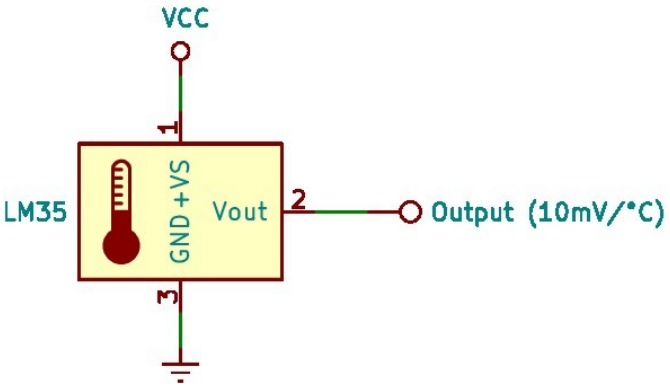
2. Introducción

La presente sección resume los pasos a seguir para leer una señal analógica discretizada con un microcontrolador. En particular, se interesa en la lectura de la temperatura registrada por un sensor LM35 acoplado a un convertidor analógico-digital o ADC vía I^2C. Los datos registrados serán posteriormente enviados a una Raspberry Pi para llevar una bitácora de temperatura que podrá ser desplegada en un navegador web.

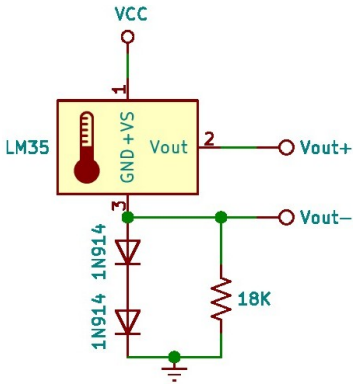
2.1. El sensor LM35

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



(a) Configuración básica: 2°C a 150°C



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

Figura 1: Configuraciones típicas del LM35

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.

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:

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

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. Es por esto que la mayoría de los convertidores analógico-digital incluyen pines para voltajes de referencia V_{Ref+} y V_{Ref-}, para lo cual se hace uso de un divisor de voltaje usando la fórmula:

$$V_{out} = \frac{R_2}{R_1 + R_2} \times V_{IN} \tag{3}$$

Por ejemplo, supóngase que se utilizará un Arduino UNO alimentado a $5V_{CC}$ que cuenta sólo con un pin voltaje de referencia positivo V_{Ref+}, denominado *AREF* según las especificaciones [2]. Usando un par de resistencias $R_1 = 10k\Omega$ y $R_2 = 12k\Omega$ (véase Figura 2) para generar un voltaje de referencia para un LM35 en configuración típica de rango completo se tendría:

$$V_{AREF} = \frac{12\Omega}{12\Omega + 10\Omega} \times 5V = \frac{12\cancel{\Omega}}{22\cancel{\Omega}} \times 5V = \frac{60V}{22} = 2.72V \tag{4}$$

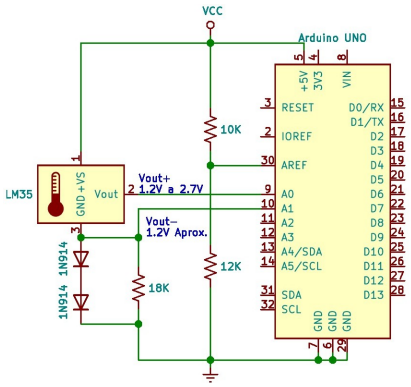


Figura 2: Circuito medidor de temperatura LM35 con Arduino

Con este nuevo valor de referencia 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:

$$\Delta V = \frac{2.73V}{1024} = 0.00267V \tag{5}$$

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 I²C

I²C 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].

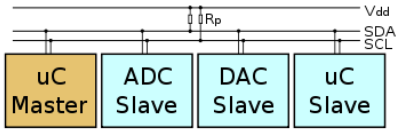


Figura 3: Bus I²C

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].

En I²C todas las operaciones son iniciadas por un dispositivo maestro que provee el reloj (se permiten múltiples maestros) e indica el tipo de operación junto con la dirección del esclavo en un sólo byte. Si la operación es de lectura, el dispositivo maestro cambiará el pin SDA como entrada y escuchará la respuesta del dispositivo esclavo. Si, por el contrario, la operación es de lectura, el dispositivo maestro enviará los datos vía SDA y esperará un bit de OK (0) como respuesta del dispositivo esclavo, leyendo un valor cero cuando el esclavo no contesta o hubo un error en la transmisión de datos.

3. Instrucciones

1. Descargue y pruebe la tarjeta simuladora siguiendo los pasos de la subsección 3.1.
2. Revise cuidadosamente las subsecciones 3.1 a 3.4.
3. Calcule las resistencias para un divisor de voltaje óptimo para el circuito simulado siguiendo las instrucciones de la subsección 3.2.
4. Con base en el voltaje de referencia, calcule el factor de conversión de valores discretos a centígrados tal como lo indica la subsección 3.3.
5. Tomando como base el programa de ejemplo del Apéndice A y lo aprendido en las subsecciones 3.1 a 3.4 realice el programa descrito en la sección 4 y, con los resultados obtenidos, responda el cuestionario de la sección 6.

3.1. Paso 1: Configuración del Simulador

Descargue el simulador de `https://github.com/kyordhel/RPiVirtualBoards` ejecutando la siguiente línea de comandos:

```
git clone https://github.com/kyordhel/RPiVirtualBoards.git
cd RPiVirtualBoards
```

A continuación instale todas las dependencias requeridas por el simulador usando *pipenv*[2]:

```
pipenv install
```

Finalmente, pruebe el simulador ejecutando la siguiente línea:

```
pipenv run python temperature.py
```

Si, por otro lado, no desea mantener el simulador conmo un proyecto aislado, puede instalar los paquetes con *pip* después de clonar el proyecto:

```
pip install  --user -r requirements.txt
./temperature.py
```

Si la configuración es correcta, verá una ventana similar a la de la Figura 4 con los valores de temperatura simulados cambiando.

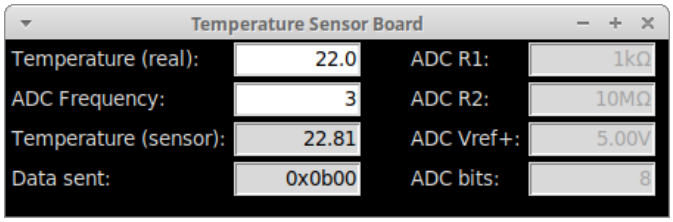


Figura 4: Simulador de tarjeta lectora de LM35 con DAC I^{2}C

3.2. Paso 2: Cálculo del divisor de voltaje

El simulador configurado en la Subsección 3.1 implementa la simulación de un sensor de temperatura LM35 en configuración básica acoplado a un circuito ADC con un divisor de voltaje en V_{Ref+} y V_{Ref-} a tierra, mismo que se configura con la línea:

Código ejemplo 1: Configuración del ADC del sensor LM35 virtual I^{2}C (`temperature.py:58`)

```
1  run_temperature_board(r1=1, r2=10000, p8bits=True)
```

[2] *pipenv* es una herramienta que facilita la creación y administración de entornos virtuales en cualquier proyecto escritos en Python, llevando un control riguroso de los paquetes de los que depende dicho proyecto. Se puede instalar fácilmente con la línea `pip install -user pipenv`

La función `run_temperature_board` recibe tres parámetros, los valores de las resistencias R_1 y R_2 para la alimentación del V_{Ref+} y un boleano que configura el módulo ADC para operar con una presición de 8 bits (`p8bits=True`) o de 10 bits (`p8bits=False`).

Sin embargo, los valores por defecto de las resistencias usadas no son óptimos, ya que hacen que el módulo ADC opere en rango completo de $0-5V$ mientras que el LM35 en configuración básica opera en el rango de $0.02-1.50V$, motivo por el cual deben proporcionarse otros valores para las resistencias.

Para calcular las resistencias del divisor de voltaje, lea cuidadosamente la Subsección 2.2 y utilice la Ecuación (3) tomando un valor fijo para R_1 (ej. $10k\Omega$) y despejando R_2.

IMPORTANTE

Verifique que al calcular los valores de R_1 y R_2 aproxima sus resultados a los valores de resistencias comerciales, por ejemplo cambiando una resitencia de 1273.5Ω por una resistencia comercial de $1k2$.

3.3. Paso 3: Lectura del sensor LM35

Para leer la temperatura se necesita convertir los valores discretos leídos por el ADC en valores de temperatura. Esto se puede realizar mediante un simple análisis debido a la linearidad del LM35. En un ADC típico se obtienen dos lecturas: 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. Por ejemplo, con el LM35 en una configuración básica (V_{OUT-} en GND), $V_{Ref+} = 2.72V$ y considerando que que $1^oC = 0.01V$ se tendría:

$$\begin{aligned} T[^oC] &= V_{diff} \times \frac{2.72[V]}{1024 \times 0.01[\frac{V}{^oC}]} \\ &= V_{diff} \times \frac{2.72}{10.24}[^oC] \\ &= 0.266 V_{diff}[^oC] \end{aligned}$$

o bien, generalizando para todo voltaje de referencia:

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

Esta fórmula de conversión, o su equivalente según el divisor de voltaje utilizado, deberá programarse en la Raspberry Pi que adquiera los valores discretos de temperatura del sensor.

3.4. Paso 4: Adquisición de datos via I²C

Una forma de leer del puerto serial I²C de la Raspberry Pi usando Python es mediante el paquete `smbus`, o su sucesor `smbus2` que incorpora exactamente las mismas funciones pero está implementada completamente en Python en lugar de ser un envoltorio para la *SMBusLib* de C.

El primer paso es instanciar un objeto de tipo `SMBus` indicando el número de dispositivo a controlar (véase el Código de Ejemplo 2, línea 1).

Código ejemplo 2: Lectura de datos usando I^{2}C (`temperature.py:30-43`)

```
1 i2c = smbus2.SMBus(1)
2
3 def readTemperature():
4   try:
5     msg = smbus2.i2c_msg.read(SLAVE_ADDR, 1)
6     i2c.i2c_rdwr(msg)
7     data = list(msg)
8     temp = struct.unpack('<B', msg.buf)[0]
9     print('Received ADC temp value: {} = {:0.2f}'.format(data, temp))
10    return temp
```

Debido a que las transmisiones de datos en I^{2}C son síncronas y completamente controladas por el dispositivo maestro, no es necesario realizar complicadas rutinas que almacenen datos en búfferes y controlen interrupciones. En su lugar, simplemente se genera un mensaje I^{2}C de tipo lectura o escritura y se realiza la transacción con la función `i2c_rdwr` (en I^{2}C toda lectura requiere de una escritura al canal SDA). Para generar el mensaje es necesario especificar tanto la dirección del dispositivo esclavo colmo los datos a enviar, en el caso de una escritura, o el número de bytes que se leerán de éste. Esta operación se muestra en el Código de Ejemplo 2, líneas 5 y 6.

Un aspecto a tomar en cuenta es que las transmisiones seriales de datos no son otra cosa que flujos de bytes sin significado alguno, mientras que las variables en Python son objetos con un tipo inferido sobre los cuales se pueden realizar un gran número de operaciones; es decir, ambos son incompatibles. Es por esto que es necesario convertir los datos binarios en algo que Python pueda entender. Estas conversiones las realizan las funciones `pack` y `unpack` del paquete `struct`. En particular, la función `unpack` recibe un *bytearray* y un especificador de formato que indica cómo deben interpretarse los bytes en el arreglo para poder generar una tupla de n elementos interpretados.

En el Código de Ejemplo 2 (línea 8), se realiza la lectura de 1 byte como entero no signado (ADC de 8 bits con rango de 0–255), por lo que se utiliza el especificador de formato `'<B'` para leer 1 byte no signado (`unsigned char` en C). Si se quisieran leer 16 bits (ADC de 10 bits) habrpia que especificar un *unsigned half int* con la cadena de formato `'<H'`. El caracter de *menor que* (<) en la cadena de formato se requiere para especificar que los bytes están codificados en *little endian*, es decir, con el byte menos significativo a la izquierda, que es la codificación por defecto en I^{2}C.

4. Programas

Tomando como base el código de ejemplo del Apéndice A, desarrolle un programa llamado `temp_srvr.py` que cumpla con las siguientes especificaciones:

- El programa lee vía I^{2}C valores **discretos** de temperatura de un sensor LM35 proporcionados por un Arduino o cualquier otro ADC compatible con la siguiente configuración.
  - [2 pts] La resolución del convertidor analógico-digital es de 10 bits.
  - [2 pts] El divisor de voltaje para el pin de referencia V_{Ref+} permite tomar lecturas en el rango de 0–150°C con la máxima resolución posible usando resistencias comerciales.
- El programa recibe por línea de comandos tres parámetros opcionales:
  - Los valores de las resistencias R_1 y R_2 del divisor de voltaje en formato: `R1=1,R2=1` (valores de resistencias en kilo Ohms). El rango válido es de $1k\Omega$ a $100M\Omega$.
  - La resolución en bits del convertidor ADC con el formato `b=8` o `b=10`. Valores diferentes no son aceptables.
  - La frecuencia de operación del convertidor ADC con el formato `f=3` (en Hertz). El rango válido es de $1Hz$ a $100Hz$.
- El programa muestra en pantalla la temperatura en grados centígrados cada segundo. La temperatura reportada es la promedio obtenida del sensor.
- [2 pts] El programa almacena en el archivo local `temp_history.py` el historial de temperatura promedio en grados centígrados con una precisión de un segundo (1s).

- [Robustez] Si los parámetros introducidos son incorrectos o están fuera de rango, el programa desplegará una ayuda mostrando el uso correcto de los parámetros y no se mostrará ninguna interfaz gráfica.
- [+5 pts] El programa ejecuta un servidor web (ej. `127.0.0.1:8080`) que permite monitorear de manera remota la temperatura con un cliente web, mostrando la información del sensor (resolución, rango y frecuencia).

  Importante: El video-evidencia deberá mostrar ambas ventanas, la interfaz del cliente web y el simulador. Las temperaturas deberán coincidir dentro de lo razonable.
- [+5 pts] El cliente web remoto permite consultar la bitácora de temperatura y utiliza esta información para generar una gráfica *en tiempo real* con el histórico de los valores de temperatura registrados.

Nota: La nota máxima del programa sin elementos adicionales es de 6 puntos (60/100).

5. Especificaciones técnicas de los programas

- No utilice paquetes adicionales.
- El código deberá ser ejecutable con Python versión 3.5 o posterior.
- Todos los programas deberán comenzar con la línea de intérprete o *she-bang* correspondiente
- Todos los programas deberán tener el nombre del autor de la forma:

```
1 # Author: Nombre del Alumno
```

- Cuando se implemente un servidor web los videos-evidencia deberá observarse claramente cómo el alumno controla remotamente desde la interfaz de usuario al simulador de la RaspBerry Pi (ej. desde su celular o desde otra máquina virtual).
- Incluya sólo los videos y el código fuente de los programas ***sin librerías ni paquetes***.
- Los archivos de código python deberán estar en raíz `./`.
- Los videos-evidencia deberán estar en el subdirectorio `./vid/`.
- Los videos-evidencia deberán durar no más de 60 segundos, incluir sólo la ventana del simulador y contar únicamente con *stream* de video comprimido con *codec* h.264 a $15fps$ con una resolución máxima de 1280×720 y con un tamaño máximo de 3MB por archivo (velocidad de datos aproximada de $1500kbps$)[3].
- Los entregables deberán estar empaquetados en un archivo comprimido de nombre `[prefijo]_p03` donde `[prefijo]_p03` corresponde a los primeros 4 caracteres de la CURP del alumno, por ejemplo `hicm_p03.zip`. Los formatos aceptables son *7z*, *rar*, *tar.bz2*, *tar.gz* y *zip*.

6. Cuestionario

1. [2 pts] ¿Cuáles son los valores óptimos para las resistencias comerciales usadas en el divisor de voltaje que alimenta al pin V_{Ref+}? Justifique su respuesta e incluya el análisis matemático (cálculos) correspondiente.
2. [2 pts] Suponga que se tiene un ADC que puede opera a una frecuencia de hasta $1kHz$. ¿Que estrategia se recomienda para maximizar la presición del sensor? ¿Conviene seguir utilizando el promedio simple u otra técnica de filtrado? ¿Es conveniente reducir la frecuencia de muestreo? Justifique su respuesta.

Nota: La nota máxima del cuestionario es de 4 puntos (40/100).

[3] `ffmpeg -i input -an -vf scale=-1:720 -c:v libx264 -crf 28 -r 15 -preset veryslow video.mp4`

7. 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. El archivo `temperature.py`

Código ejemplo 3: Archivo temperature.py

```
import time

# Initializes virtual board (comment out for hardware deploy)
from virtualboards import run_temperature_board

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

# 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():
  """Reads a temperature bytes from the Arduino via I2C"""
  try:
    msg = smbus2.i2c_msg.read(SLAVE_ADDR, 1)
    i2c.i2c_rdwr(msg)
    data = list(msg)
    temp = struct.unpack('<B', msg.buf)[0]
    print('Received ADC temp value: {} = {:0.2f}'.format(data, temp))
    return temp
  except:
    return None
#end def

def log_temp(temperature):
  try:
    with open(LOG_FILE, 'a') as fp:
      fp.write('{} {:0.2f}C\n'.format(
        time.strftime("%Y.%m.%d %H:%M:%S"),
        temperature
      ))
  except:
    return
#end def

def main():
  # Runs virtual board (comment out for hardware deploy)
  run_temperature_board(r1=1, r2=10000, p8bits=True)
  time.sleep(1)

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

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