



# Universidad Autónoma de San Luis Potosí

Facultad de Estudios Profesionales de la Zona Media

**Materia:** Microcontroladores

**Profesor:** Ing. Jesús Padrón

**Alumno:** Jimena Balderas Coronado

**Título de la Práctica:** Práctica 11: Medición de temperatura y visualización gráfica

**Fecha:** 25 de abril de 2025

## Introducción

La serie LM35 son dispositivos de temperatura de circuito integrado de precisión con un voltaje de salida linealmente proporcional a la temperatura centígrada. Tiene un rango de temperatura de 55 °C a 150 °C. Como el dispositivo LM35 extrae solo 60 A del suministro, tiene un autocalentamiento muy bajo de menos de 0.1 °C en aire quieto. El dispositivo LM35 está clasificado para funcionar en un rango de temperatura de 55 °C a 150 °C.

En esta práctica, observaremos la función del sensor conectada al microcontrolador Raspberry Pi Pico y cómo se grafica su temperatura por medio de un programa en PyCharm.

## Desarrollo

Para comenzar generando el código que nos ayudará al funcionamiento debemos crear una carpeta que llevará como nombre “Practica\_11”, aquí importaremos la carpeta `.vscode` y el archivo `pico_sdk_import.cmake` mediante el programa Pico-Developer PowerShell usando los comandos.

```
copy ${env:PICO_SDK_PATH}\external\pico_sdk_import.cmake .  
copy ${env:PICO_EXAMPLES_PATH}\.vscode . -recurse
```

Figura 1: Importación de vscode

```

cmake_minimum_required(VERSION 3.13)

include(pico_sdk_import.cmake)
set(PICO_BOARD pico_w)

project(Practica_11)

pico_sdk_init()

add_executable(ADC2
                ADC2.c)

target_link_libraries(ADC2
                      pico_stdlib
                      hardware_adc)

pico_enable_stdio_usb(ADC2 1)
pico_enable_stdio_uart(ADC2 0)

pico_add_extra_outputs(ADC2)

```

Figura 2: Importación de CMake y configuración inicial

Una vez importados estos documentos, debemos crear un archivo de texto, en el cual se creará el `CMakeLists.txt`:

Como primer paso, deberemos establecer la versión mínima requerida. Incluimos nuestro archivo `pico_sdk_import.cmake`. Establecemos la variable `PICO_BOARD` en `pico_w` para compilar en la placa Raspberry.

Establecemos el nombre de nuestro proyecto, “Practica\_11”.

Se inicializa el SDK de la Raspberry Pi Pico.

```

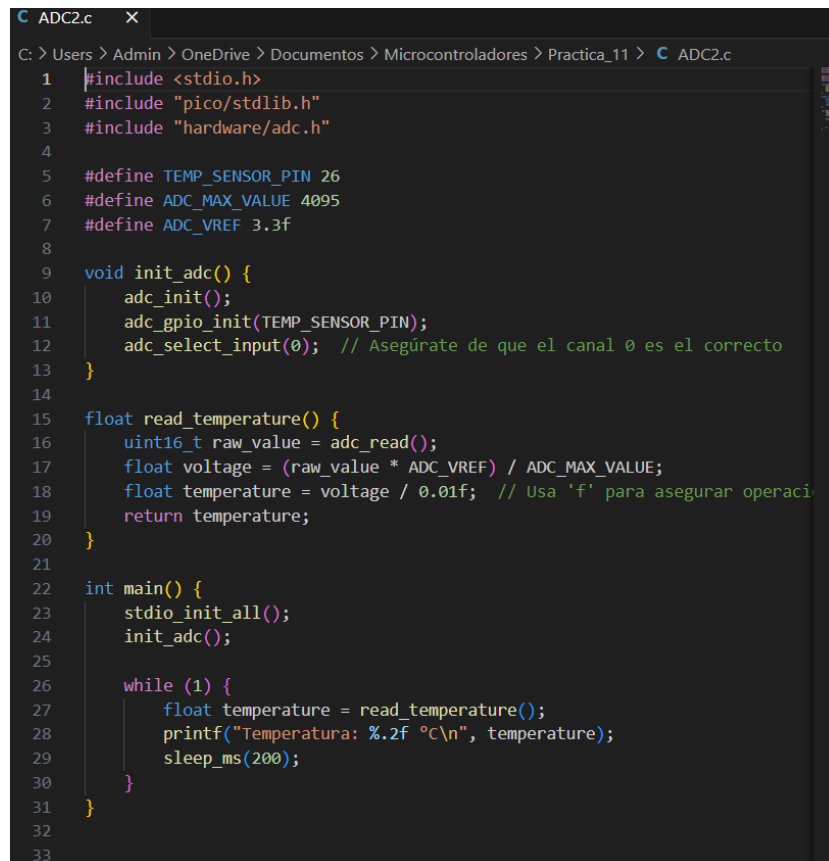
add_executable(ADC2 ADC2.c)
target_link_libraries(ADC2 pico_stdlib)

```

Agregamos un ejecutable llamado `ADC2` que se compilará a partir de nuestro archivo `.c`. La biblioteca `pico_stdlib` proporciona funciones para trabajar

con la placa Pico.

## Código de ADC2



```
C ADC2.c X
C: > Users > Admin > OneDrive > Documentos > Microcontroladores > Practica_11 > C ADC2.c
1  #include <stdio.h>
2  #include "pico/stdlib.h"
3  #include "hardware/adc.h"
4
5  #define TEMP_SENSOR_PIN 26
6  #define ADC_MAX_VALUE 4095
7  #define ADC_VREF 3.3f
8
9  void init_adc() {
10     adc_init();
11     adc_gpio_init(TEMP_SENSOR_PIN);
12     adc_select_input(0); // Asegúrate de que el canal 0 es el correcto
13 }
14
15 float read_temperature() {
16     uint16_t raw_value = adc_read();
17     float voltage = (raw_value * ADC_VREF) / ADC_MAX_VALUE;
18     float temperature = voltage / 0.01f; // Usa 'f' para asegurar operaci
19     return temperature;
20 }
21
22 int main() {
23     stdio_init_all();
24     init_adc();
25
26     while (1) {
27         float temperature = read_temperature();
28         printf("Temperatura: %.2f °C\n", temperature);
29         sleep_ms(200);
30     }
31 }
32
33
```

Figura 3: Código fuente del archivo ADC2.c

Empezaremos creando un archivo de texto en la carpeta de nuestra práctica, y posteriormente cambiaremos su extensión a “.c”, le pondremos el nombre de ADC2.

Empezaremos incluyendo las librerías:

- `stdio.h`: Librería estándar de C para entrada/salida, necesaria para `printf()`.
- `pico/stdlib.h`: Librería del SDK de la Raspberry Pi Pico, para inicialización básica y retardos.
- `hardware/adc.h`: Control del ADC de la Pico.

Ahora definiremos el canal que corresponde al ADC que será el `TEMP_SENSOR_PIN` al GPIO 26. Haremos la definición del valor máximo de 12 bits del ADC con `ADC_MAX_VALUE`. Contemplaremos una referencia de voltaje del ADC que será 3.3V con `ADC_VREF`.

Para inicializar el ADC tendremos en un `void` las funciones que inicializan el hardware del ADC (`adc_init()`), habilitan el pin GPIO 26 para el ADC (`adc_gpio_init(...)`) y seleccionan el canal 0 del ADC (`adc_select_input(0)`).

Para la lectura y cálculo de la temperatura, tendremos dentro de `float read_temperature()` la conversión del valor leído del ADC a voltaje real.

Se convierte a **voltaje real** con la fórmula:

$$V = \frac{\text{raw\_value} \times V_{REF}}{4095}$$

Luego se convierte a temperatura:

$$T = \frac{V}{0.01} = V \times 100$$

Figura 4: Fórmula para convertir voltaje real

En la función principal, `int main()`, se realiza la inicialización con `stdio_init_all()`, y `init_adc()`. En el bucle infinito se lee la temperatura, se imprime con `printf()` mostrando 2 decimales, y se espera 200 ms con `sleep_ms(200)`.

## Código para gráfica de temperatura

```
main.py ×
1  import serial
2  import matplotlib.pyplot as plt
3  import matplotlib.animation as animation
4  from collections import deque
5
6  # Configuración del puerto serial
7  SERIAL_PORT = "COM3"
8  BAUD_RATE = 115200
9  ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
10
11 # Configuración de datos
12 temps = deque(maxlen=50)
13 times = deque(maxlen=50)
14 time_counter = 0
15
16 # Configurar la gráfica
17 fig, ax = plt.subplots()
18 ax.set_title("Temperatura en Tiempo Real")
19 ax.set_xlabel("Tiempo (s)")
20 ax.set_ylabel("Temperatura (°C)")
21 ax.set_ylim(0, 50)
22 line, = ax.plot(*args: [], [], 'r-', label="LM35")
23 ax.legend()
24
25 # Función para actualizar la gráfica
26 def update(frame): 1usage
27     global time_counter
28     try:
29         data = ser.read_until(b'\n').decode('utf-8').strip()
30         if data:
31             parts = data.split()
32             if len(parts) >= 2:
```

Figura 5: Configuración del puerto y librerías

Empezaremos abriendo nuestro programa de PyCharm, e importaremos las siguientes librerías:

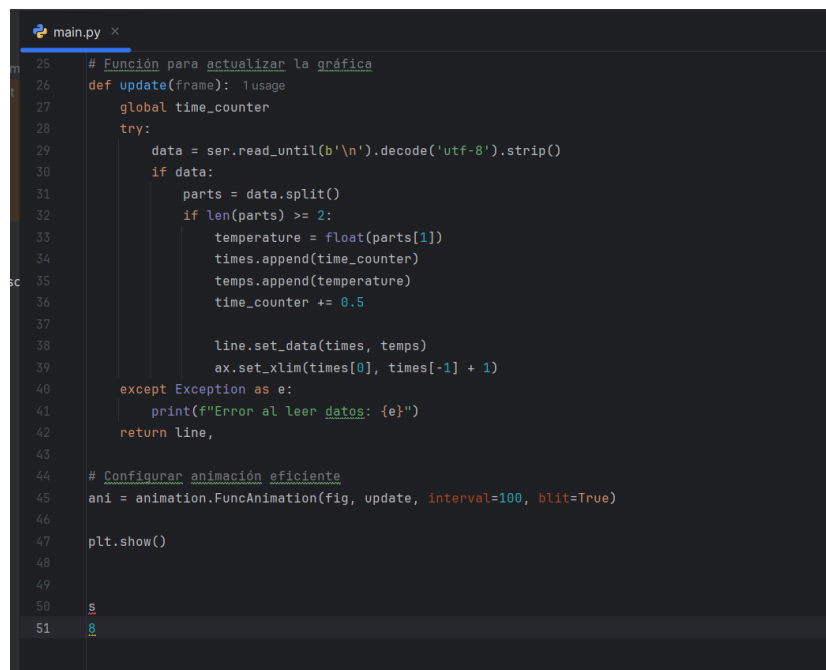
- `serial`: Para comunicación con el puerto COM.
- `matplotlib.pyplot`: Para graficar.
- `matplotlib.animation`: Para actualizar la gráfica en tiempo real.
- `deque`: Cola eficiente para mantener los últimos N datos.

Configuramos el puerto serial abriendo COM3 a 115200 baudios con un timeout. Este punto debe coincidir con el usado en la Raspberry Pi Pico.

Las variables serán:

- `temps`: Guarda hasta 50 lecturas de temperatura.
- `times`: Guarda los tiempos correspondientes.
- `time_counter`: Se usa como eje X.

Para la gráfica, se crea un gráfico con eje Y de 0 a 50 °C, se inicializa una línea para los datos que se actualizará continuamente.



```

25 # Función para actualizar la gráfica
26 def update(frame): 1 usage
27     global time_counter
28     try:
29         data = ser.read_until(b'\n').decode('utf-8').strip()
30         if data:
31             parts = data.split()
32             if len(parts) >= 2:
33                 temperature = float(parts[1])
34                 times.append(time_counter)
35                 temps.append(temperature)
36                 time_counter += 0.5
37
38             line.set_data(times, temps)
39             ax.set_xlim(times[0], times[-1] + 1)
40     except Exception as e:
41         print(f"Error al leer datos: {e}")
42     return line,
43
44 # Configurar animación eficiente
45 ani = animation.FuncAnimation(fig, update, interval=100, blit=True)
46
47 plt.show()
48
49
50
51

```

Figura 6: Animación y actualización de la gráfica

Se lee una línea desde el puerto serial, se procesa para obtener la temperatura, se actualiza el gráfico y se maneja cualquier error con un mensaje.

La función `FuncAnimation` llama a `update` cada 100 ms, y con `blit=True` se optimiza la actualización del gráfico. `plt.show()` muestra la ventana.

Cuando terminemos de compilar, haremos las conexiones de nuestro microcontrolador al sensor y posteriormente cargaremos nuestro programa al microcontrolador por medio de un cable tipo b, para comenzar nuestra gráfica, agregaremos calor a nuestro sensor, para esto prenderemos un encendedor y lo colocaremos en el sensor.

## Resultados

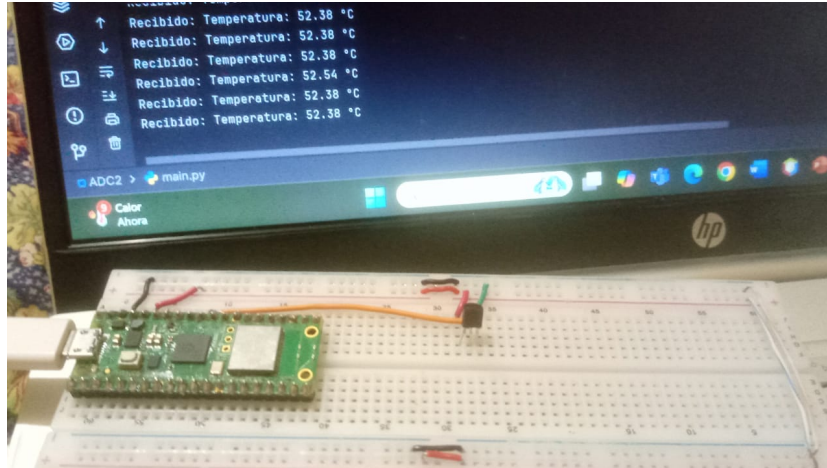


Figura 7: Resultado

## Conclusiones

Esta práctica fue interesante, ya que dio seguimiento al comportamiento de ADC en los microcontroladores, además de que hizo que fuera didáctica al estar haciendo que el sensor recibiera calor para observar como es que la gráfica cambiaba de valores.