Proyecto Integrador I

Actividad semana 2:

1. **Consigna Nro 1:** Investigación de las plataformas de desarrollo:

Documentar las características principales de Arduino (PlatformIO) y MicroPython (RT-Thread), destacando sus diferencias y aplicaciones en IoT.

PlatformIO es en esencia un IDE profesional y de código abierto que brinda facilidades de editar, comprender el código y desarrollar código para microcontroladores. Se puede integrar como extensión en otros IDEs. Es compatible con más de 400 placas de desarrollo. Está pensado para el desarrollos de aplicaciones IoT. Y facilita para esto, el uso y gestión de librerías. Este además, entre otros, viene con el entorno Arduino que facilita el desarrollo en placas, siendo compatible con múltiples placas. Provee una estructura de proyectos en el que se generan varios directorios y archivos que vienen a formar la estructura básica del proyecto.

Por otra parte, Micro Python de RT Thread es una implementación de python 3.x, diseñado para microcontroladores y sistemas embebidos. Es un entorno de desarrollo profesional potente, y ofrece funciones como la conexión conveniente del modo de desarrollo de la placa (puerto serie, red, USB), soporte para la finalización inteligente de código basado en MicroPython y verificación de sintaxis. Es compatible con múltiples placas, permite sincronizar proyectos completos, y se ayuda de muchos ejemplos de códigos y programas de demostración.

Así también, estos dos entornos tienen diferencias entre sí que los hacen más viables en diferentes situaciones o proyectos. Las principales diferencias serían:

- Lenguaje de programación, PlatformIO soporta variedad de lenguajes (C, C++, python, etc.) mientras que Micro Python solo funciona con python versión 3/superior.
- Compatibilidad con placas: PlatformIO es compatible con más/menos 400 placas, mientras Micro Python (RT Thread) solo puede ejecutarse en sistemas embebidos equipados con RT Thread.
- El entorno de desarrollo que en el primer caso tiene funciones de autocompletado, verificación de sintaxis, y gestión de múltiples proyectos a la par. Y microPython (RT Thread) tiene funciones como la conexión conveniente del modo de desarrollo de la placa y soporte de finalización inteligente de código.
- Gestión de librerías que plataformIO facilita el uso y gestión, mientras que micro Python (RT Thread) proporciona ejemplos y programas de demostración

---

Investigar sobre diferentes módulos shield disponibles para ESP32 y su aplicación en controladores IoT.

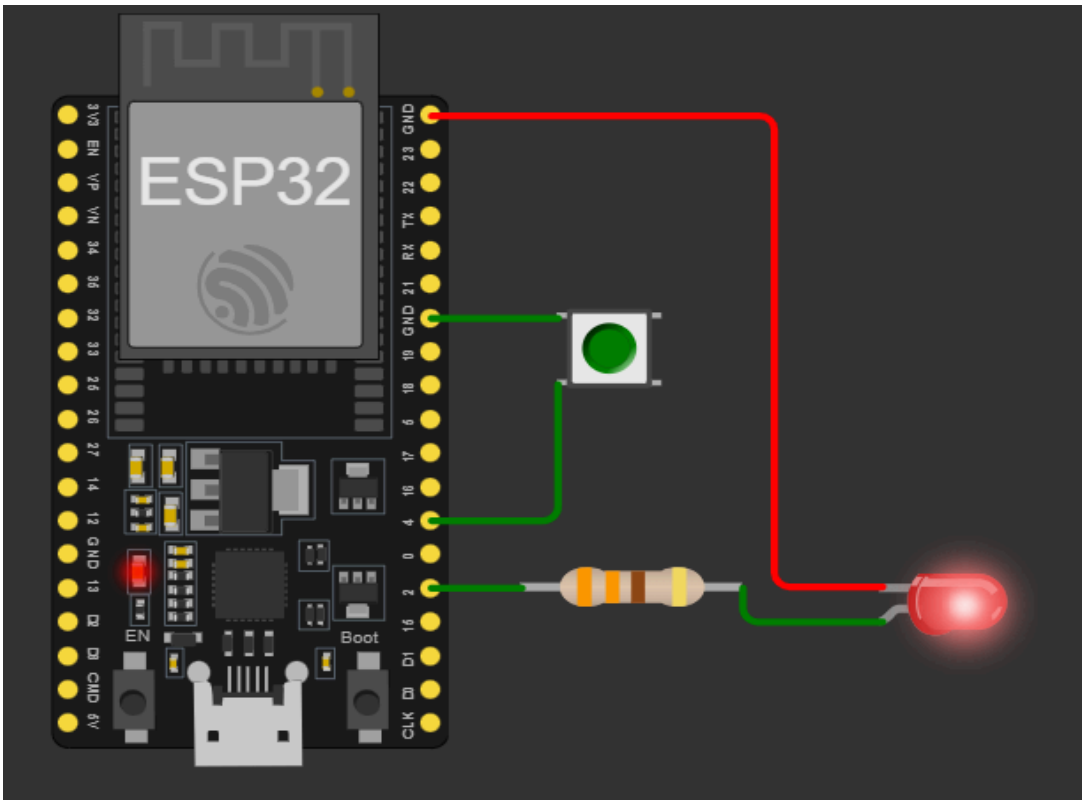
Los módulos shield son placas de expansión que extienden las capacidades del microcontrolador proporcionando funcionalidad adicional y facilitando la conexión con otros componentes. Hay una enorme variedad. Podemos diferenciarlos según categorías por su función o características, como funcionalidad específicas como agregar conectividad wifi, bluetooth, GPS, sensores varios, periféricos como salidas de interfaz de usuario, cámaras, módulos de comunicación, etc.
Alguno de los modulos shield más populares serían: Módulo Shield Base para ESP32 30 Pines IO Expansion que proporciona una entrada de fuente de voltaje de 5V y de 6.5V - 16V que amplía la gama de opciones para llevar a cabo proyectos que requieren más potencia, Módulo sensor de temperatura y humedad DHT11 o modulo de presion y temperatura BMP180 que permiten monitorizar parámetros de amplio uso en proyectos, Módulo sensor de movimiento PIR o módulo sensor de luz BH1750, Módulo sensor de gas MQ-2, Módulo de pantalla OLED de 0.96 pulgadas o módulo de pantalla TFT que permiten agregar pantallas, Módulo GPS NEO-6M

Cada componente o módulo tiene una gama de características y especificaciones, que se encuentran generalmente, bien definidas en los respectivos sitios web de los fabricantes. Pero también se pueden verificar estas características y detalles en bibliotecas de componentes como Digikey o Mouser, sitios de compra y distribuidores de componentes electrónicos con amplio detalles en las especificaciones y en español. O en bases de datos de componentes electrónicos como Octopart o Findchips

---

2. **Consigna Nro 2: Ejercicios de Implementación:**

Controlador de Entradas Digitales: Crear un sketch en Arduino y un script en Micro Python que lea el estado de un botón y encienda un LED cuando el botón esté presionado..



Sketch en micro Python, para el encendido de un led, cuando el pulsador se encuentra presionado. Defino el pin 2 para el led, y el 4 para el pulsador. Luego defino la función “encendido led” con un bucle while, cada vez que el pulsador se presiona (`machine.Pin.IN`) su valor es igual a 0, se enciende . Y cuando se deja de presionar (`machine.Pin.PULL_UP`) su valor igual 1, se apaga

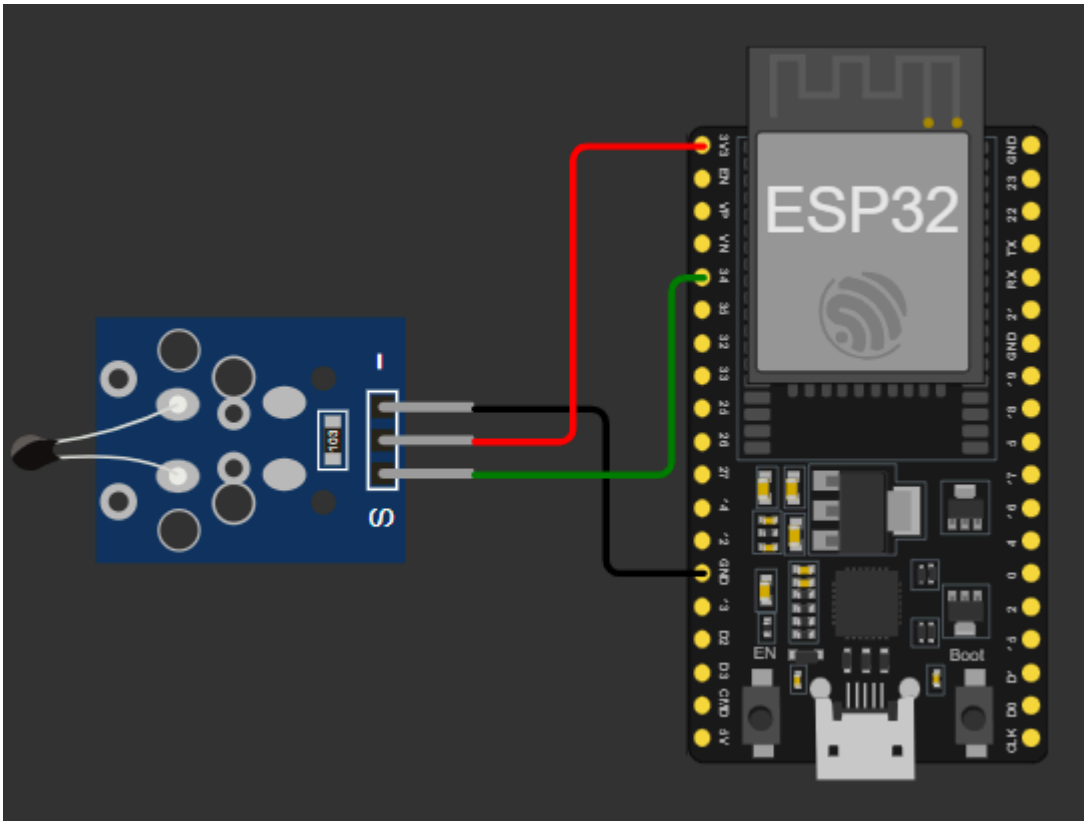
```
import machine
import time

led_pin=2
switch_pin=4

led=machine.Pin(led_pin, machine.Pin.OUT)
switch=machine.Pin(switch_pin, machine.Pin.IN, machine.Pin.PULL_UP)

def encendido_led():
  while True:
    if switch.value()==0:
      led.on()
    elif switch.value()==1:
      led.off()
      time.sleep(0.2)

encendido_led()
```

Con el sketch de Arduino, seguimos una lógica similar. Definimos los pines en el void setup. Y luego saltamos al loop, definimos “pulsador”. Luego definimos el if/else, cuando el valor está en high, se enciende el led, cuando el valor está en low (no presionado), se apaga.

```
void setup() {
 pinMode(15, OUTPUT);
 pinMode(2, INPUT);
}

void loop() {
  int pulsador=digitalRead(2);

  if (pulsador == HIGH){
    digitalWrite(15, HIGH);
  }
  else{
    digitalWrite(15, LOW);
  }
delay(20);
}
```

Controlador de Entradas Analógicas: Desarrollar un programa que lea valores de un sensor de temperatura y los muestre en el Serial Monitor/consola.

Se une el sensor de temperatura analógico NTC que brinda wokwi. Primero hice el código en la fórmula de celsius indicada en el while, pero el valor que me daba en consola era 2095. El mismo wokwi tiene un apartado donde indica que el sensor no lee la temperatura de forma directa, y brinda un bloque de código para hacer la conversión del sensor a, en este caso, grados celsius. Importe math, plasme la linea de codigo ajustando las variables como temperatura, y defino fuera del while el valor Beta que corresponde a este sensor



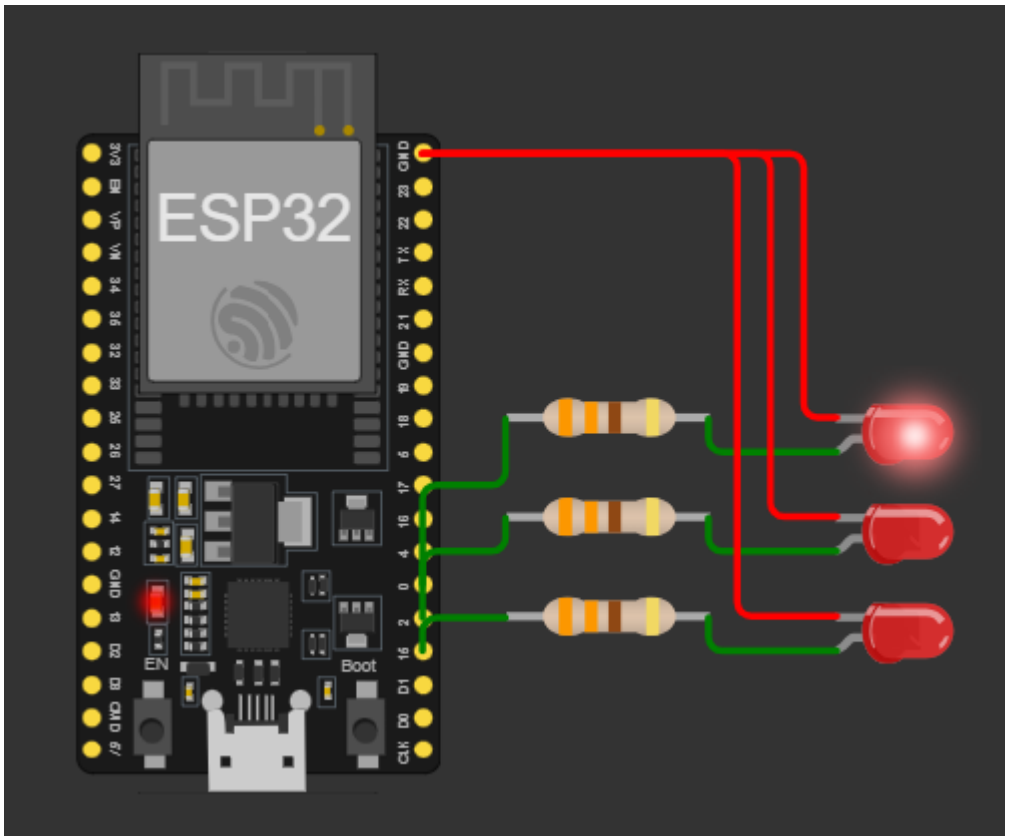

```
from machine import ADC, Pin
import time
import math

sensor= ADC(Pin(34))
sensor.atten(ADC.ATTN_11DB)
Beta= 3950

while True:
    temperatura=sensor.read()
    celsius = 1 / (math.log(1 / (4095. / temperatura - 1)) / Beta
+ 1.0 / 298.15) - 273.15
    print ("la temperatura leida por el sensor es: ", celsius,
"C°")

    time.sleep(1)
```

En el caso del entorno arduino, primero definimos el valor de Beta. Luego en el setup según la plantilla de wokwi, hay que establecer un serial, en que se indica la velocidad en baudios, con la que se debe iniciar. Luego en el loop, definimos la temperatura, para que tome la lectura del valor del sensor de la misma manera que con el código anterior, y luego se

coloca la misma cuenta que convierte el valor de sensor a temperatura real. Pero lo que se imprime en la consola es el valor seria, a la temperatura leída después de la ecuación.

```
#define BETA 3950

void setup() {
  Serial.begin(9600);
}

void loop() {
  int temperatura = analogRead(A0);
  float celsius = 1 / (log(1 / (1023.0 / temperatura - 1)) / BETA
+ 1.0 / 298.15) - 273.15;
  Serial.print("La temperatura leída por el sensor es: ");
  Serial.println(celsius);
  delay(1000);
}
```

Controlador de Salidas Digitales: Implementar un sistema que alterne el encendido de un conjunto de LEDs en intervalos regulares.



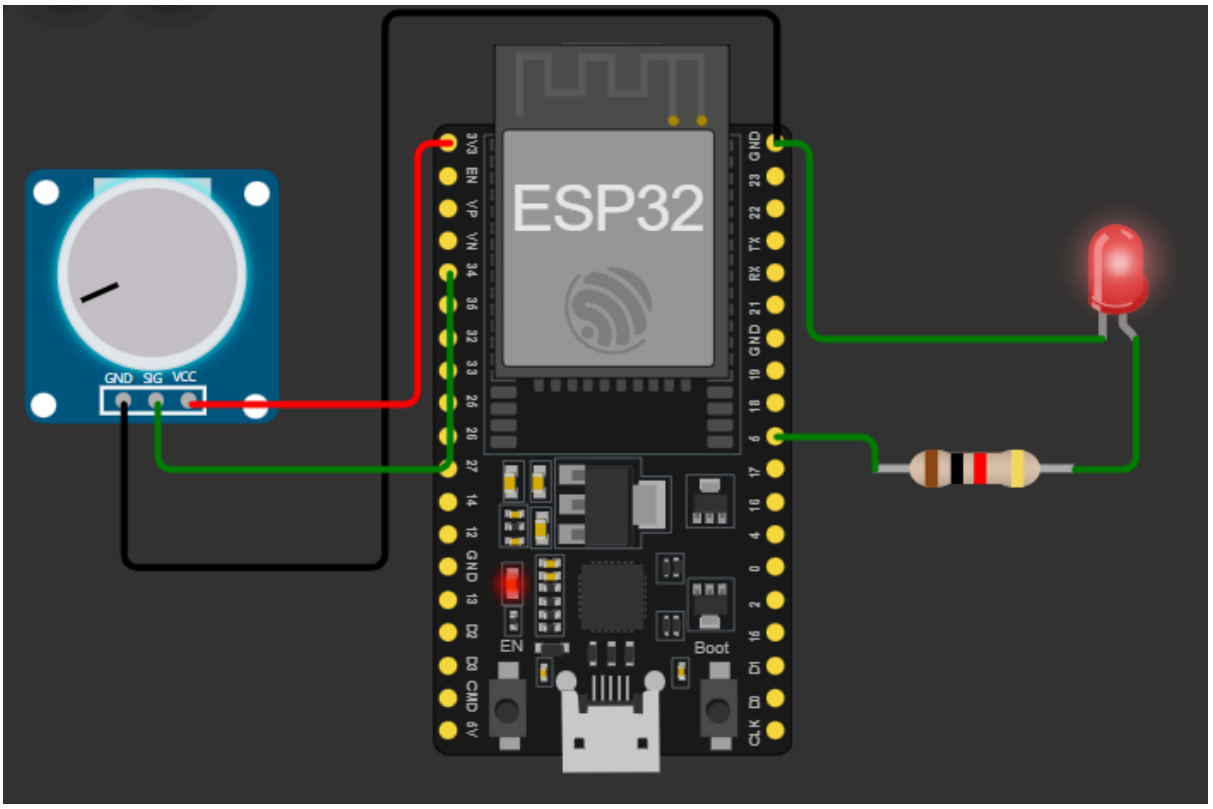
Similar al uno,se definen todos los pines en una lista y se utiliza en este caso, solo un bucle while. que mediante un for in, encienda y apague cada uno de los leds, en el lapso indicado, en este caso definido como 2 segundos.

```
import machine
import time

led_pins=[15,2,0]
leds=[machine.Pin(pin, machine.Pin.OUT) for pin in led_pins]

def glow_led():
  while True:
   for led in leds:
    led.on()
    time.sleep(2)
    led.off()

glow_led()
```

A diferencia de microPython, en el entorno de arduino hay que declarar cada pin/led. Luego se realiza un setup individual para cada caso, Y en el loop se indica prendido, el delay que determina el tiempo que permanece encendido, y el apago, continua con la linea arranca el mismo proceso con el próximo. Facilita la gestión y manipulación individual, pero lo hace significativamente menos escalable.

```
#define LED_PIN1 15
#define LED_PIN2 2
#define LED_PIN3 0

void setup() {
 pinMode(LED_PIN1, OUTPUT);
 pinMode(LED_PIN2, OUTPUT);
 pinMode(LED_PIN3, OUTPUT);
}

void loop() {
digitalWrite(LED_PIN1, HIGH);
delay(1000);
digitalWrite(LED_PIN1, LOW);

digitalWrite(LED_PIN2, HIGH);
delay(1000);
digitalWrite(LED_PIN2, LOW);

digitalWrite(LED_PIN3, HIGH);
delay(1000);
digitalWrite(LED_PIN3, LOW);
}
```

Controlador de Salidas Analógicas: Escribir un código que controle la intensidad de un LED usando PWM basado en la lectura de un potenciómetro.



Para el caso del sketch de arduino definimos el led y el potenciómetro. Buscando se recomienda configurar el potenciómetro con la línea `pot.atten(ADC.ATTN_11DB)` para regular la entrada de voltaje a 3.3V. Dentro del bucle while se utiliza la función map para mapear los valores del potenciometro que va de 0 a 4095. Pero como el puerto PWM solo acepta valores que vayan de 0 A 1023 debe mapearse para encajar en ese rango. Para eso es la función map_value donde colocamos el valor minimo y maximo de entrada, y el valor minimo y maximo de salida.

```
from machine import Pin, PWM, ADC
import time

led = PWM(Pin(5))

pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)

def map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min

while True:
    pot_value=pot.read()
    pwm_value = map_value(pot_value, 0, 4095, 0, 1023)
    led.duty(int(pwm_value))
```

```
    time.sleep(0.1)
```

En la variante del entorno Arduino, definimos los pines para el led,y para el potenciómetro. En el setup sólo se inicializan el led como entrada, y el potenciómetro como salida. Y en el loop, primero se lee el potenciómetro, y devuelve un valor entre 0 y 1023. Y con el map() se mapea a un valor que va de 0 a 255, segun encontre por que en arduino, el ciclo de trabajo PWM puede variar entre un valor 0 osea que esta siempre apagado, a 255, siempre prendido.El analogWrite es para controlar la intensidad con la que enciende el led, según el valor en el que se posicione el potenciómetro.

```
const int LED_PIN(5);
const int POT_PIN (34);

void setup () {
    pinMode(LED_PIN, OUTPUT);
    pinMode(POT_PIN,INPUT);
}

void loop() {
    int potValue=analogRead(POT_PIN);
    int pmwValue = map(potValue, 0, 1023, 0, 255);

    analogWrite(LED_PIN, pmwValue);
    delay(100);
}
```