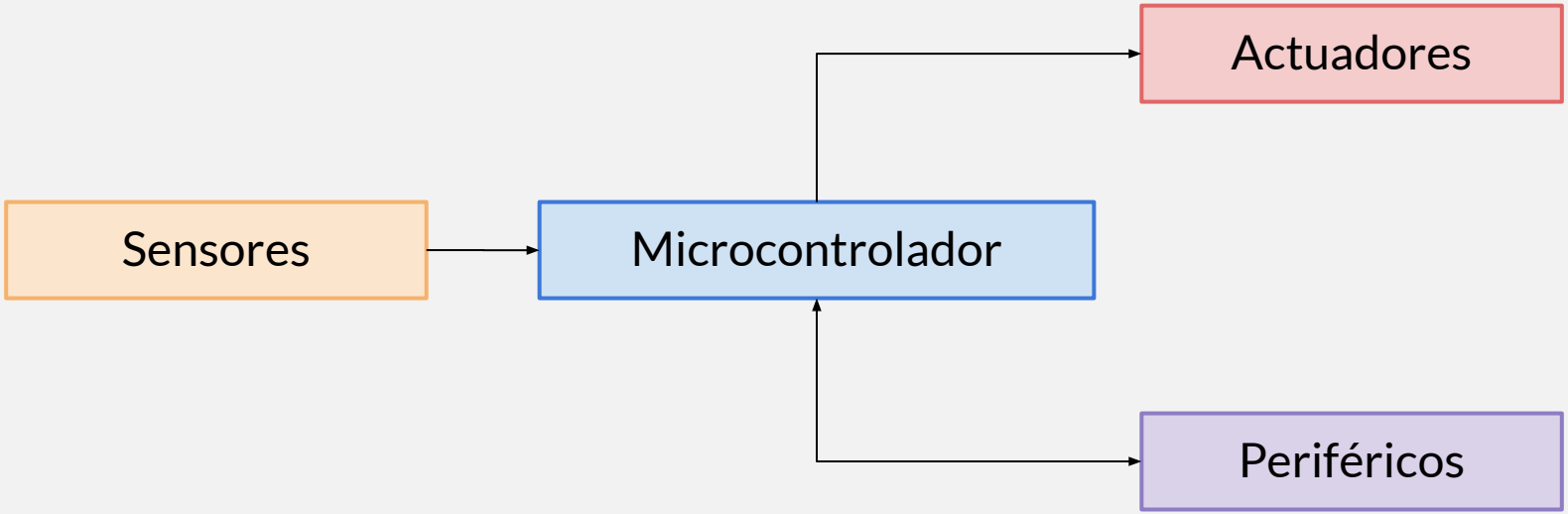
NEON
Network of Competence on Internet of Things

Co-funded by the Erasmus+ Programme of the European Union

Sensores, Actuadores y Periféricos

Sensores, Actuadores y Periféricos

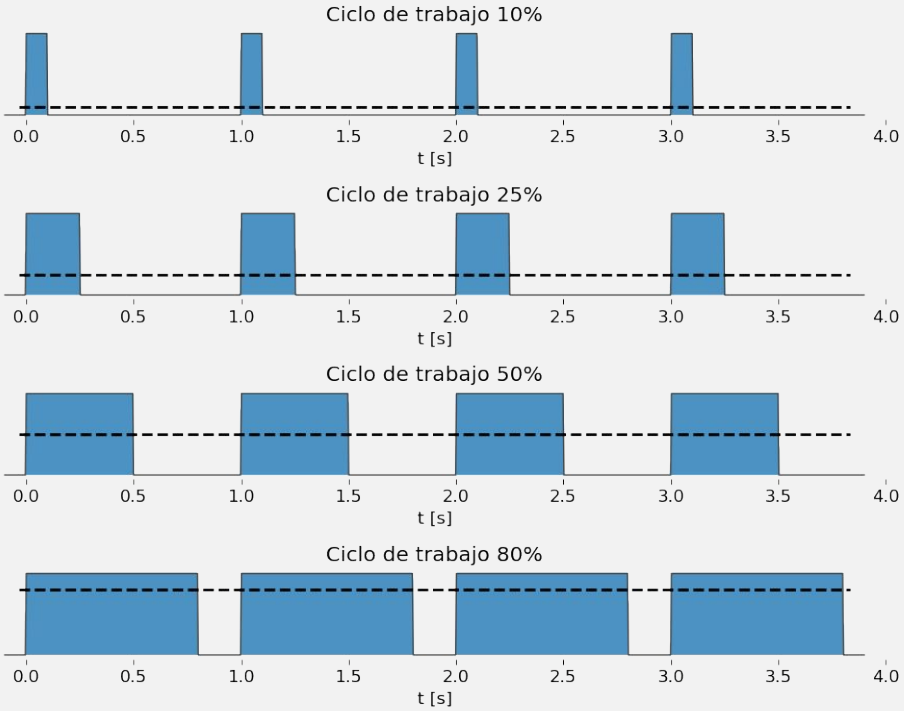


- **Conexiones externas**
  - Entrada/salida digital (1 o más bits)
  - PWM
  - Entrada/salida analógica
  - Comunicación
    - UART
    - I2C
    - SPI

PWM

- Modulación de ancho de pulso
- Onda pulsada de frecuencia fija y ciclo de trabajo variable
- La información se transmite en la duración (o ancho) del pulso
- Aplicaciones:
    - Brillo regulable para LEDs
    - Control de motor de continua
    - Control de giro del servomotor

PWM


Ciclo de trabajo 10%
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
t [s]
Ciclo de trabajo 25%
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
t [s]
Ciclo de trabajo 50%
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
t [s]
Ciclo de trabajo 80%
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
t [s]

PWM

- Pines disponibles en kit de desarrollo:
  - LEDs: GPIO0, GPIO2, GPIO15
  - GPIO14
- Configuración:

```
from machine import Pin, PWM
pwm = PWM(Pin(0), freq=10000)
pwm.duty(512)
```

→ Ciclo de trabajo: 0 – 1023

PWM – Control de motor de continua

- Valor medio determina la velocidad de giro
- Para ciclo de trabajo 0: mínima velocidad (valor medio nulo)
- Para ciclo de trabajo 1023: máxima velocidad (valor medio máximo)
- Interfaz de potencia: puente H (circuitería adicional)

PWM – Control de servomotor

- Se configura a 50Hz (pulsos separados 20 ms)
- Para la posición 0°: pulso de 1ms (ciclo de trabajo 51)
- Para la posición 90°: pulso de 1.5ms (ciclo de trabajo 76)
- Para la posición 180°: pulso de 2ms (ciclo de trabajo 102)

Entrada/Salida analógica

- Conversor analógico-digital de 12 bits
- Conversor digital-analógico de 8 bits
- Aplicaciones:
  - Leer tensiones variables de sensores como temperaturas, luminosidad, etc.
  - Generar formas de onda arbitrarias o valores de tensión específicos

Configuración ADC

```
from machine import Pin, ADC
adc = ADC(Pin(35), atten=ADC.ATTN_11DB)
lectura = adc.read_uv()
```

Configuración ADC

```
from machine import Pin, ADC
adc = ADC(Pin(35), atten=ADC.ATTN_11DB)
lectura = adc.read_uv()
```

Factor de atenuación:

- ADC.ATTN_0DB: 100mV ~ 950mV
- ADC.ATTN_2_5DB: 100mv ~ 1250mV
- ADC.ATTN_6DB: 150mV ~ 1750mV
- ADC.ATTN_11DB: 150mV ~ 2450mV

Configuración DAC

```
from machine import Pin, DAC
dac = DAC(Pin(25))
dac.write(127)
```


Amplitud entre 0 - 255

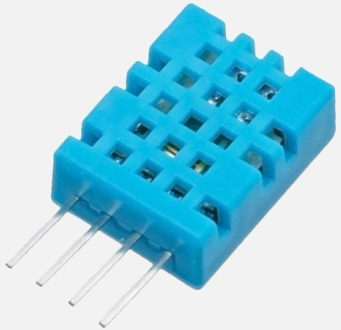
Protocolos

- **UART**: comunicación serie asincrónica
- **I2C**: comunicación serie sincrónica de baja velocidad
- **SPI**: comunicación serie sincrónica de alta velocidad

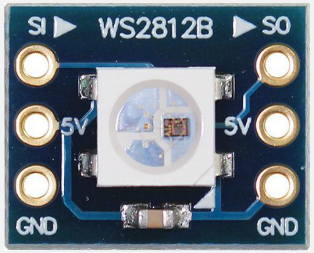
Aplicaciones:

- **UART**: terminal en PC usando conversor UART a USB
- **I2C**: pantallas LCD, sensores varios
- **SPI**: tarjetas SD, pantallas LCD

Otros sensores y actuadores



Sensor de temperatura
y humedad DHT11/22



LED RGB NeoPixel
(ws2812b)

Configuración DHT11/22

```
from dht import DHT11
from machine import Pin

dht = DHT11(Pin(32))
try:
    dht.measure()
    print(f"Temperatura: {dht.temperature()}°C")
    print(f"Humedad: {dht.humidity()}°C")
except OSError:
    print("Error al leer medición")
```

Configuración DHT11/22

```
from dht import DHT11
from machine import Pin

dht = DHT11(Pin(32))
try:
    dht.measure()
    print(f"Temperatura: {dht.temperature()}°C")
    print(f"Humedad: {dht.humidity()}°C")
except OSError:
    print("Error al leer medición")
```

Configuración DHT11/22

```
from dht import DHT11
from machine import Pin

dht = DHT11(Pin(32))
try:
    dht.measure()
    print(f"Temperatura: {dht.temperature()}°C")
    print(f"Humedad: {dht.humidity()}°C")
except OSError:
    print("Error al leer medición")
```

Configuración NeoPixel

```
from neopixel import NeoPixel
from machine import Pin

np = NeoPixel(Pin(27), 3)

np[0] = (255, 0, 0)  # Rojo
np[1] = (0, 255, 0)  # Verde
np[2] = (0, 0, 255)  # Azul
np.write()
```

Configuración NeoPixel

```
from neopixel import NeoPixel
from machine import Pin

np = NeoPixel(Pin(27), 3)

np[0] = (255, 0, 0)  # Rojo
np[1] = (0, 255, 0)  # Verde
np[2] = (0, 0, 255)  # Azul
np.write()
```

Configuración NeoPixel

```
from neopixel import NeoPixel
from machine import Pin

np = NeoPixel(Pin(27), 3)

np[0] = (255, 0, 0)  # Rojo
np[1] = (0, 255, 0)  # Verde
np[2] = (0, 0, 255)  # Azul
np.write()
```

Watchdog

- Se usa para reiniciar el sistema al congelarse el programa para garantizar que el equipo siga en funcionamiento al encontrarse con una situación no recuperable
- Requiere ser *alimentado* periódicamente. Al superar un tiempo determinado, se produce el reinicio del equipo

Watchdog - Configuración

```
from machine import WDT
wdt = WDT(timeout=2000)  # timeout en milisegundos

while True:
    # Resto del código
    wdt.feed()
```

Watchdog - Configuración

```
from machine import WDT
wdt = WDT(timeout=2000)  # timeout en milisegundos

while True:
    # Resto del código
    wdt.feed()
```

Programación Asíncrona

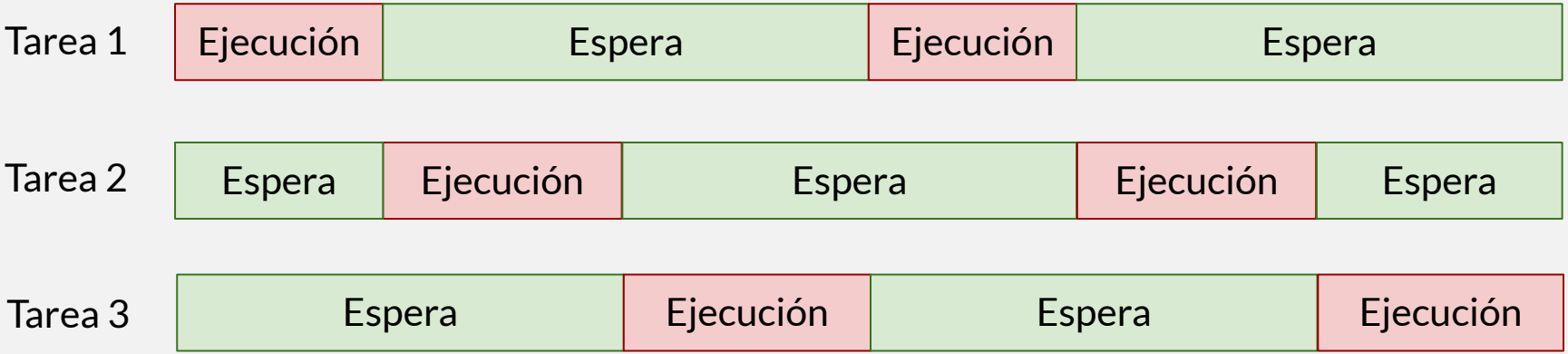
Programación Asíncrona - Problema

- Un solo programa controla muchos periféricos o sensores
- La mayor parte del tiempo se ocupa esperando:
  - Un estímulo externo
  - El momento adecuado para comunicarse con el gateway
- El programa se complejiza al implementar un ciclo que responda a todos los eventos

Programación Asíncrona - Solución

- Repartir el programa en tareas independientes
  - *Tasks*
- Ceder el control en los tiempos de espera
  - *Await*
- Alternar entre las tareas
  - *Scheduling*

Programación Asíncrona



uasyncio

```
import uasyncio

async def tarea1():
    while True:
        print("Tarea 1")
        await uasyncio.sleep_ms(500)
async def tarea2():
    while True:
        print("Tarea 2")
        await uasyncio.sleep_ms(200)
```

uasyncio

```
import uasyncio

async def tarea1():
    while True:
        print("Tarea 1")
        await uasyncio.sleep_ms(2000)
async def tarea2():
    while True:
        print("Tarea 2")
        await uasyncio.sleep_ms(800)
```

uasyncio

```
import uasyncio

async def tarea1():
    while True:
        print("Tarea 1")
        await uasyncio.sleep_ms(2000)
async def tarea2():
    while True:
        print("Tarea 2")
        await uasyncio.sleep_ms(800)
```

uasyncio

```
import uasyncio

async def tarea1():
    while True:
        print("Tarea 1")
        await uasyncio.sleep_ms(2000)
async def tarea2():
    while True:
        print("Tarea 2")
        await uasyncio.sleep_ms(800)
```

uasyncio

```
async def main():
    uasyncio.create_task(tarea1())
    uasyncio.create_task(tarea2())

    while True:
        await uasyncio.sleep_ms(1000)

uasyncio.run(main())
```

uasyncio

```
async def main():
    uasyncio.create_task(tarea1())
    uasyncio.create_task(tarea2())

    while True:
        await uasyncio.sleep_ms(1000)

uasyncio.run(main())
```

uasyncio

```
async def main():
    uasyncio.create_task(tarea1())
    uasyncio.create_task(tarea2())

    while True:
        await uasyncio.sleep_ms(1000)

uasyncio.run(main())
```

Nota sobre variables globales

Para compartir resultados entre diferentes tareas, es posible usar variables globales.

En Python y MicroPython, no es posible modificar variables fuera del alcance de la función pero sí accederlas

Nota sobre variables globales

Ejemplo:

```
var_global = 5

def funcion():
    print(var_global)  # Muestra 5
    var_global = 6     # Crea una nueva variable
                       # dentro del alcance de la función
```

Nota sobre variables globales

Solución:

```
var_global = 5

def funcion():
    global var_global
    print(var_global)  # Muestra 5
    var_global = 6     # Modifica la variable global
```

Ejercicios

- **Ejercicio 1**
  - Hacer parpadear un LED de forma suave, variando su brillo
- **Ejercicio 2**
  - Cambiar el brillo un LED según la posición del potenciómetro

Ejercicios

- **Ejercicio 3**
  - Leer la temperatura del DHT11/22 y encender el LED verde si la temperatura está por debajo de 30°C. Si está por encima, encender el LED rojo
- **Ejercicio 4**
  - Modificar el programa del parpadeo del LED de la clase anterior para que al pulsar un botón, parpadee más rápido y al pulsar el otro botón parpadee más lento incorporando programación asíncrona