LABORATORIO DE RS232 CON RASPBERRY PI PICO 2W

Vallentina Diaz Valbuena

[est.vallentina.diaz@unimilitar.edu.co](mailto:est.vallentina.diaz@unimilitar.edu.co)

Docente: José de Jesús Rugeles

1. **RESUMEN**
2. **ABSTRACT**
3. **DESARROLLO EXPERIMENTAL**

*Comparación de las Raspberry Pi Pico W y Pi Pico 2W*

En el ámbito de los microcontroladores, las Raspberry Pi Pico W y Pico 2W representan dos de las opciones más populares para el desarrollo de proyectos integrados e IoT. Aunque comparten el mismo núcleo de procesamiento (RP2040), presentan diferencias clave en conectividad, eficiencia energética y capacidades de comunicación inalámbrica. La siguiente tabla comparativa detalla sus especificaciones técnicas, permitiendo una evaluación objetiva para determinar cuál se adapta mejor a las necesidades del laboratorio y evolución de la materia, ya sea por su compatibilidad con WiFi, Bluetooth, consumo energético o coste. Este análisis facilitará la toma de decisiones en aplicaciones que requieran comunicación UART, conectividad inalámbrica o integración con sensores y periféricos.

| Parámetro | Raspberry Pi Pico W | Raspberry Pi Pico 2W |
| --- | --- | --- |
| Procesador | RP2040 (dual-core ARM Cortex-M0+) | RP2040 (dual-core ARM Cortex-M0+) |
| Frecuencia | 133 MHz | 133 MHz |
| Memoria Flash | 2MB | 2MB |
| SRAM | 264 KB | 264 KB |
| Conectividad WiFi | Sí (Infineon CYW43439) | Sí (Infineon CYW43439) |
| Conectividad Bluetooth | No | Sí (Bluetooth 5.2) |
| GPIO | 26 pines | 26 pines |
| Interfaces UART | 2 × UART | 2 × UART |
| Interfaces SPI | 2 × SPI | 2 × SPI |
| Interfaces I2C | 2 × I2C | 2 × I2C |
| ADC | 3 × 12-bit | 3 × 12-bit |
| PWM | 16 canales | 16 canales |
| Tensión operación | 1.8-5.5V | 1.8-5.5V |
| Consumo energía | ~100mA (activo) | ~120mA (activo con BT) |
| Precio | ~$6-$8 | ~$8-$10 |
| Lanzamiento | Junio 2022 | Enero 2023 |

*Tabla 1. Comparación técnica*

A partir de la anterior tabla se puede realizar un análisis de las diferencias claves:

* Conectividad Inalámbrica

Pico W: Solo WiFi 2.4GHz (802.11n)

Pico 2W: WiFi 2.4GHz (802.11n) + Bluetooth 5.2

Permite comunicación con dispositivos Bluetooth (sensores, periféricos) y mayor versatilidad en proyectos IoT

* Consumo Energético

Pico 2W tiene un consumo ligeramente mayor cuando Bluetooth está activo

* Compatibilidad de Software

Ambos son 100% compatibles con MicroPython y CircuitPython. Las librerías para WiFi son idénticas y Pico 2W requiere drivers adicionales para Bluetooth

* Rendimiento en Comunicaciones

Velocidad WiFi: Igual en ambos (hasta 72Mbps teóricos)

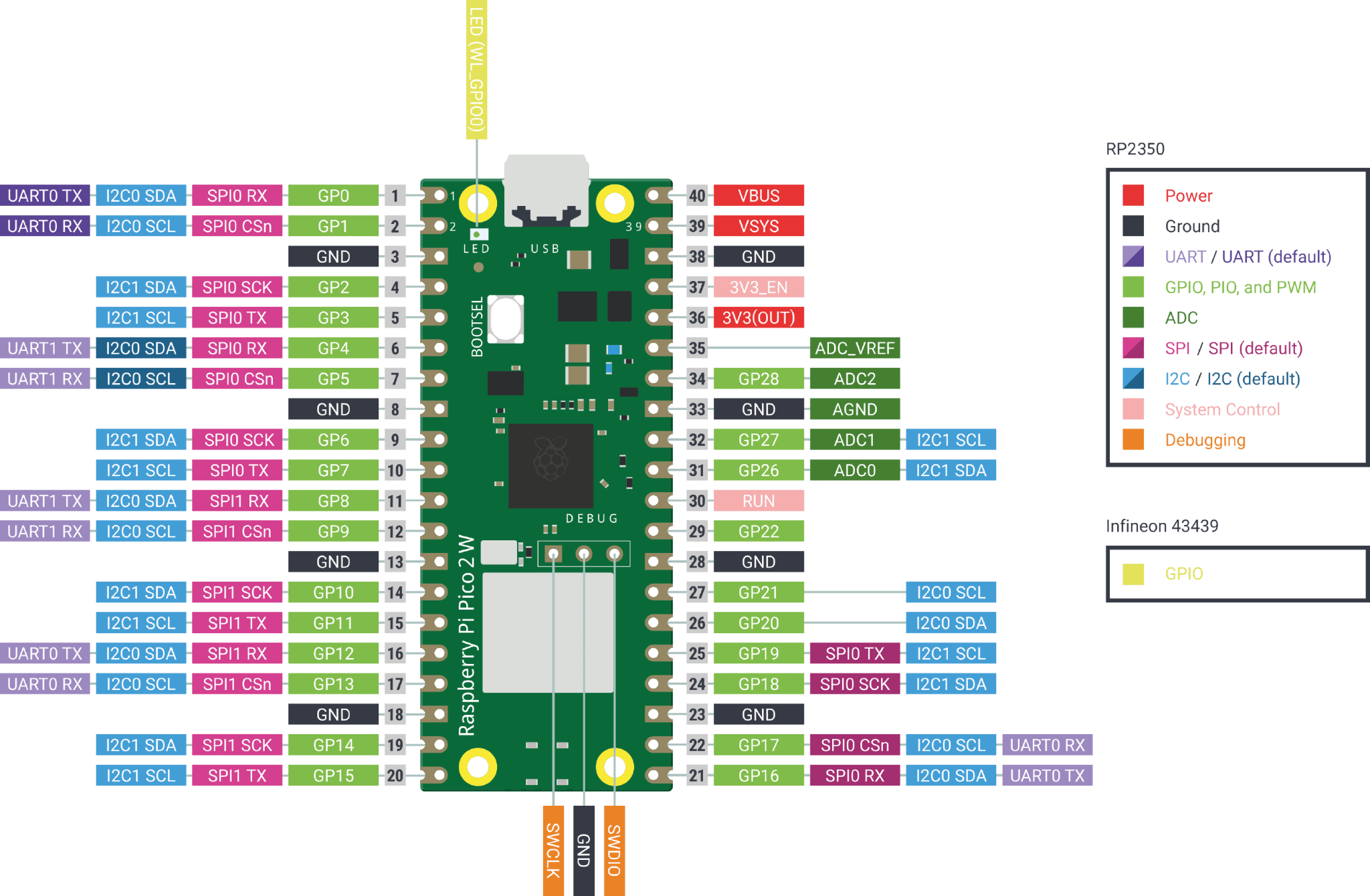
Latencia: Pico 2W puede manejar más conexiones simultáneas gracias al Bluetooth

En conclusión, la Pico 2W es una evolución natural de la Pico W, manteniendo todas sus características pero añadiendo un canal Bluetooth, la monitorización inalámbrica del sistema y la conexión con periféricos Bluetooth (como sensores móviles), por esta razones en este laboratorio y en los próximos se va a trabajar con la Raspberry Pi Pico 2W.

Teniendo en cuenta lo anterior, se realiza la configuración inicial del dispositivo, descargando el Firmware en <https://www.raspberrypi.com/documentation/microcontrollers/micropython.html> y siguiendo las instrucciones descritas en esta página para descargar el archivo .UF2 correspondiente al dispositivo.

*Datasheet de la Raspberry Pi Pico 2W*

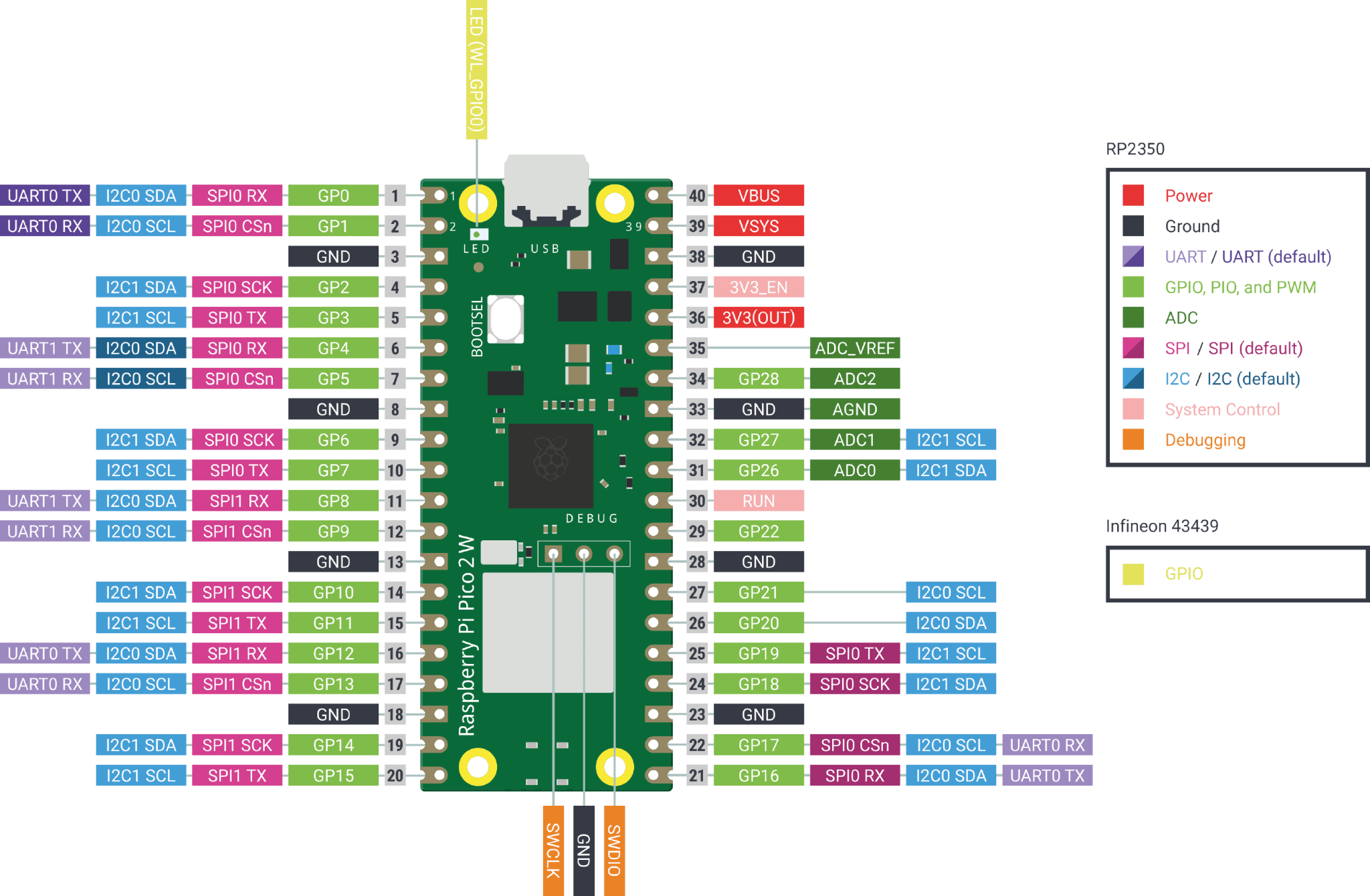
El datasheet destaca sus 26 GPIO multifunción (con ADC de 12 bits, UART, SPI e I2C), capacidad de operar entre 1.8V y 5.5V, y modos de bajo consumo energético. Además, su diseño mantiene compatibilidad con las librerías estándar de MicroPython. Esta combinación de potencia, conectividad y eficiencia la posiciona como una solución versátil para prototipado rápido y sistemas integrados avanzados. A continuación se muestra el datasheet descrito:



*Imagen 1. Datasheet Raspberry Pi Pico 2W*

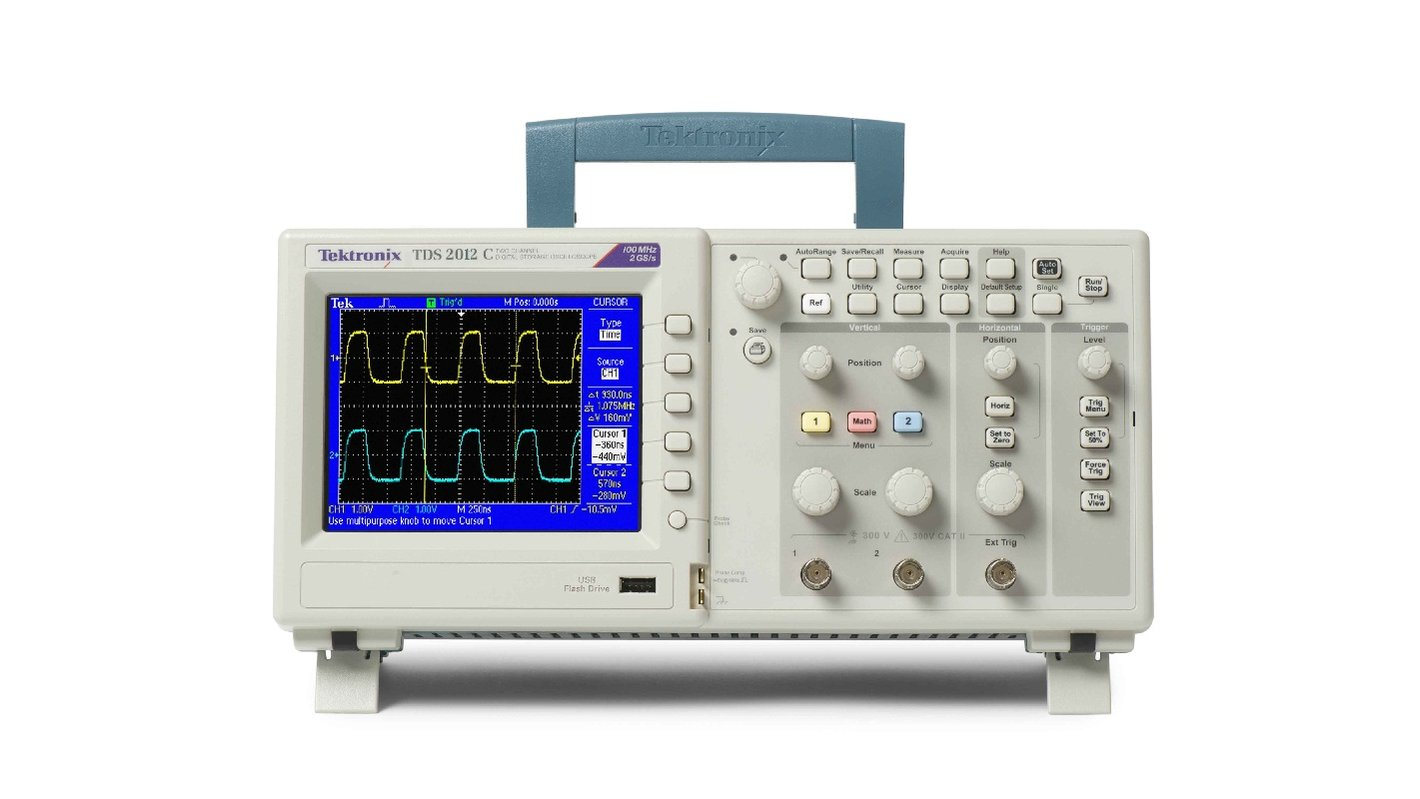
*Primer programa básico*

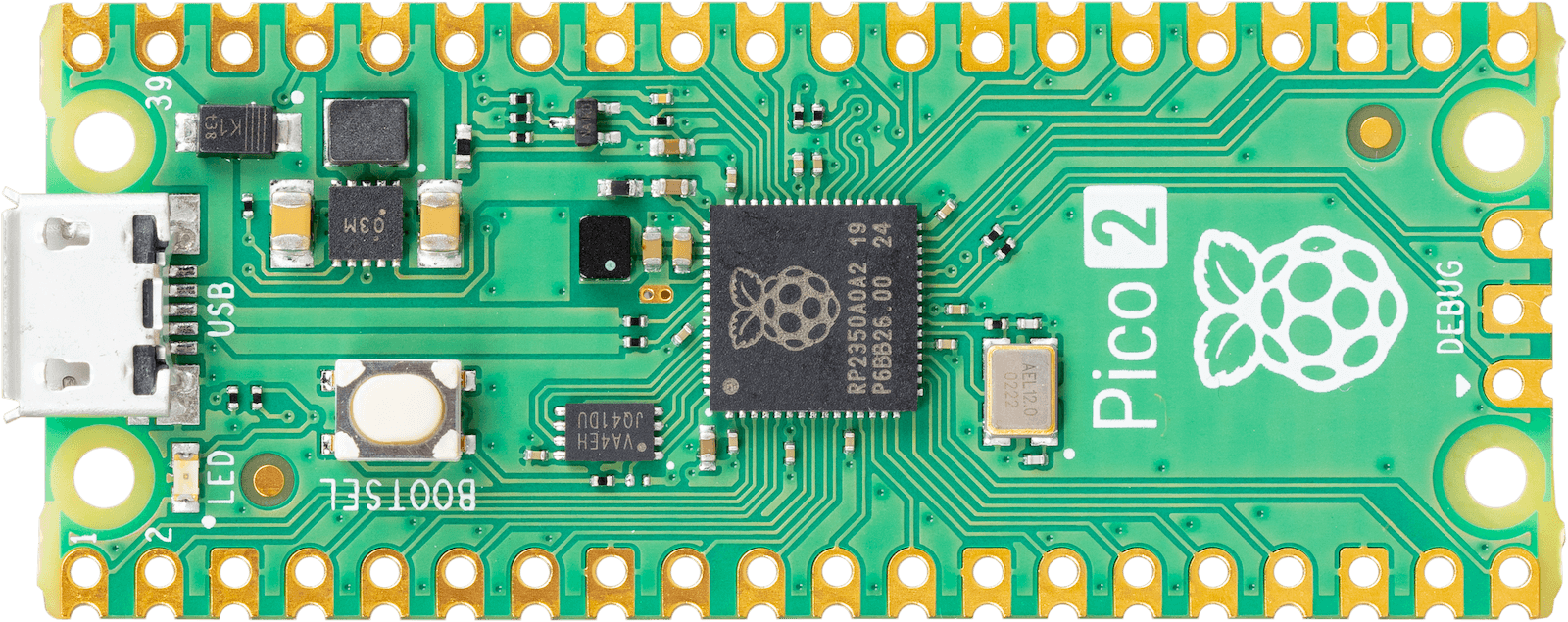
Ahora bien, con toda la información técnica, se procede a identificar los terminales TX y RX en las UART del dispositivo (1 y 2 para el UART0). Ya determinados se conecta un osciloscopio digital Tektronix TDS2012B del laboratorio, entre la terminal TX y tierra del dispositivo, por medio de una sonda. En la siguiente imagen se observa que el Terminal TX es el pin 1 y el terminal GND es el pin 3.



*Imagen 2. TX, RX y GND*

A continuación, se muestra cómo sería esa conexión física:





*Imagen 3. TX y GND al osciloscopio*

Posteriormente, se implementa el siguiente programa en el entorno de desarrollo integrado Thonny con Python, para poder implementar una comunicación básica mediante UART.

import machine

import utime

from machine import Pin, UART

led = machine.Pin("LED", machine.Pin.OUT)

uart = UART(0, baudrate=9600, bits=8, parity=0, tx=Pin(0), rx=Pin(1))

while True:

led.on()

uart.write("U")

utime.sleep(1)

led.off()

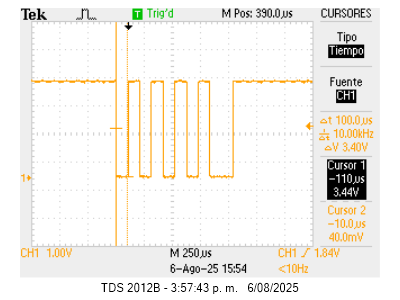
utime.sleep(1)

Ahora bien, el código anterior está diseñado para enviar periódicamente el carácter "U" (ASCII) cada segundo mientras controla el led integrado de la placa. Utiliza el puerto UART0 configurado a 9600 baudios, con 8 bits de datos y paridad par, empleando los pines GPIO0 (TX) y GPIO1 (RX). El led se enciende durante el envío del dato y se apaga posteriormente, creando un patrón visual sincronizado con la transmisión serial. Este ejemplo es ideal para entender los fundamentos de la comunicación serie en entornos integrados con MicroPython.

Se ejecuta el programa, se mide en el osciloscopio la señal recibida y se configura adecuadamente el disparo (trigger) del osciloscopio para que se capture la señal, teniendo en cuenta:

* Nivel de Trigger: Ajustar al 50% del voltaje de la señal (ej.: ~1.65V para 3.3V lógicos).
* Tipo de Trigger: Normal (para evitar que la pantalla se congele si no hay señal).

Luego se mide el tiempo de bit y se verifica si corresponde a la medida teórica esperada. En la siguiente imagen se observa la señal recibida y la medición del tiempo de bit, correspondiente a :



*Imagen 4. Carácter “U”, tiempo de bit*

Teóricamente, el tiempo de bit, se halla a partir de la siguiente ecuación:

Reemplazando los baudios utilizados:

Entonces, se logra concluir que el tiempo de bit medido en el osciloscopio coincide con el tiempo de bit calculado. Por otro lado, también se puede medir y calcular el tiempo de la trama, es decir, al realizar la conversión de código ASCII a hexadecimal a binario, se obtiene lo siguiente:

Y el tiempo de la trama se va a ver reflejado de la siguiente forma:

| Bit de inicio | Datos | Paridad | Bit de parada |
| --- | --- | --- | --- |

+1 bit +8 bits +1 bit +1 bit

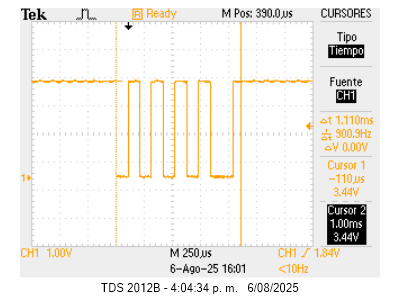
Para los datos de este primer programa se tiene que:

| 0 | 01010101 | 0 | 1 |
| --- | --- | --- | --- |

Y con este análisis, se puede decir que:

Reemplazando los datos utilizados:

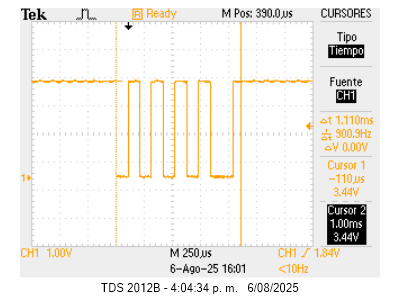
Al medirlo con el osciloscopio se obtiene lo siguiente:



*Imagen 5. Carácter “U”, tiempo de la trama*

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado. Ahora bien, los datos se transmiten en cierto orden, inicialmente va el LSB (least significant bit) y al final el MSB (most significant bit), esto se aprecia mejor en la siguiente imagen:

Bit de inicio Paridad



LSB MSB Bit de parada

*Imagen 6. Analisis carácter “U”*

La imagen anterior muestra la estructura típica de una trama UART al transmitir el carácter “U”, destacando sus componentes esenciales.

*Parámetros de la UART en Micropython*

Para poder continuar con el desarrollo del laboratorio, se debe hacer una pausa para poder responder las siguientes preguntas:

* ¿Cuál es la clase disponible en Micropython para la comunicación serial RS 232?

La clase UART implementa el protocolo estándar de comunicaciones serie dúplex UART/USART. A nivel físico, consta de dos líneas: RX y TX. La unidad de comunicación es un carácter (no confundir con una cadena de caracteres) que puede tener 8 o 9 bits de ancho.

* ¿Cuáles son los métodos disponibles para la clase UART?. Explique cada uno de ellos.

| Método | Descripción |
| --- | --- |
| UART.init(baudrate=9600, bits=8, parity=None, stop=1, \*, ...) | Inicializa/configura la UART con parámetros clave: velocidad (baudrate), bits de datos (bits), paridad (parity: None, 0=par, 1=impar), y bits de stop (stop). |
| UART.deinit() | Desactiva la UART, liberando los recursos asociados (pines y periféricos). Útil para reiniciar la comunicación o cambiar configuraciones. |
| UART.any() | Devuelve un entero que cuenta el número de caracteres que se pueden leer sin bloquear. Devuelve 0 si no hay caracteres disponibles y un número positivo si hay caracteres. El método puede devolver 1 incluso si hay más de un carácter disponible para leer. |
| UART.read([nbytes]) | Lee caracteres. Si nbytes se especifica, lee como máximo esa cantidad de bytes; de lo contrario, lee la mayor cantidad de datos posible. Puede regresar antes si se alcanza un tiempo de espera. El tiempo de espera se configura en el constructor. |
| UART.readinto(buf[, nbytes]) | Leer bytes en el [nombre del objeto buf]. Si nbytes se especifica, se lee como máximo esa cantidad de bytes. De lo contrario, se lee como máximo len(buf). Puede regresar antes si se alcanza un tiempo de espera. El tiempo de espera se configura en el constructor. |
| UART.readline() | Lee una línea que termina en un carácter de nueva línea. Puede regresar antes si se alcanza un tiempo de espera. El tiempo de espera se configura en el constructor. |
| UART.write(buf) | Escribe el buffer de bytes en el bus. El valor de retorno es el número de bytes escritos o None en tiempo de espera. |
| UART.sendbreak() | Envía una condición de interrupción en el bus. Esto activa el bus durante un tiempo mayor al requerido para la transmisión normal de un carácter. |
| UART.flush() | Espera hasta que se hayan enviado todos los datos. En caso de tiempo de espera, se genera una excepción. La duración del tiempo de espera depende del tamaño del búfer de transmisión y de la velocidad en baudios. |
| UART.txdone() | Indica si se han enviado todos los datos o si no hay transferencia. En este caso, devuelve True. Si hay una transmisión de datos en curso, devuelve False. |
| UART.irq(handler=None, trigger=0, hard=False) | Configure un controlador de interrupciones que se llamará cuando se produzca un evento UART. |

*Tabla 2. Métodos de flujo estándar*

* ¿Cómo se modifica la tasa de baudios?

Los objetos UART como la tasa de baudios se pueden crear e inicializar utilizando:

from machine import UART

uart = UART(1, 9600)

uart.init(9600, bits=8, parity=None, stop=1)

*Medida de los tiempos de bit*

Ya aclarados los parámetros anteriores se procede a envíar ahora el carácter ASCII “W”. Se eligieron seis diferentes tasas de baudios, con 8 bits de datos, paridad impar y se midieron los tiempos de bit. Los datos se encuentran tabulados en la siguiente tabla:

| Baudios | Tb (teórico) | Tb (experi.) | Δt | % error |
| --- | --- | --- | --- | --- |
| 300 | 3,33ms | 3,40ms | 0,07ms | 2,1% |
| 1200 | 833,33μs | 840,0μs | 6,67μs | 0,8% |
| 9600 | 104,17μs | 110,0μs | 5,83μs | 5,5% |
| 19200 | 52,08μs | 52,0μs | 0,08μs | 0,1% |
| 57600 | 17,36μs | 17,0μs | 0,36μs | 2,1% |
| 115200 | 8,68μs | 8,80μs | 0,12μs | 1,4% |

*Tabla 3. Comparación tiempo de bit*

La comparación de la tabla anterior muestra una buena concordancia entre los valores teóricos y experimentales, los datos demuestran que el sistema cumple con las especificaciones técnicas, siendo 19200 baudios la velocidad más precisa (error del 0.1%), ideal para aplicaciones que requieren sincronización crítica.

Ahora bien, también se midieron los tiempos de tramas y los datos se encuentran tabulados en la siguiente tabla:

| Baudios | Tt (teórico) | Tt (experi.) | Δt | % error |
| --- | --- | --- | --- | --- |
| 300 | 36,67ms | 36,0ms | 0,67ms | 1,8% |
| 1200 | 9,17ms | 9,20ms | 0,03ms | 0,3% |
| 9600 | 1,15ms | 1,11ms | 0,04ms | 3,5% |
| 19200 | 572,9μs | 572,0μs | 0,9μs | 0,2% |
| 57600 | 190,9μs | 191,0μs | 0,1μs | 0,05% |
| 115200 | 95,49μs | 95,0μs | 0,49μs | 0,5% |

*Tabla 4. Comparación tiempo de trama*

La comparación de la tabla anterior muestra una buena concordancia entre los valores teóricos y experimentales, el hardware y software utilizados reproducen fielmente los tiempos de transmisión teóricos, especialmente en velocidades medias y altas, lo que es crítico para sistemas de comunicación en tiempo real.

Además, se capturaron las imágenes del osciloscopio para documentar esta prueba, estas se muestran a continuación:

| Baudios | Tiempo de bit | Tiempo de trama |
| --- | --- | --- |
| 300 |  |  |
| 1200 |  |  |
| 9600 |  |  |
| 19200 |  |  |
| 57600 |  |  |
| 115200 |  |  |

*Tabla 5. Comparación tiempo de trama*

Para concluir este apartado, los resultados obtenidos en las tablas 3 y 4 muestran un alto grado de concordancia entre los valores teóricos y experimentales para los tiempos de bit (Tb) y los tiempos de trama completa (Tt) en el protocolo UART. Se observa que, independientemente de la velocidad de transmisión, los errores porcentuales se mantienen dentro de márgenes aceptables (≤ 5.5%), esto valida la estabilidad y confiabilidad de la implementación UART en la Raspberry.

*Análisis de la estructura del protocolo RS232*

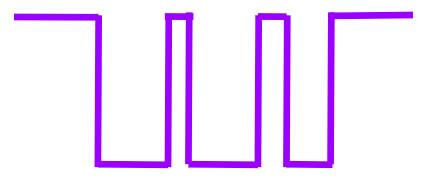
Ahora, se procede a enviar diez caracteres ASCII diferentes a los caracteres empleados anteriormente. Con ayuda del osciloscopio digital se capturaron cada uno de los datos enviados. El docente a cargo del laboratorio asignó el baudrate correspondiente a 19200 baudios, 8 bits de datos y se va a variar la paridad entre par, impar y ninguna. El tiempo de bit teórico se calcula a continuación:

* Carácter ASCII “D”

*Paridad par*

La imagen a continuación muestra la estructura típica de una trama UART al transmitir el carácter “D”, con paridad par, destacando sus componentes esenciales y es lo que se espera observar en el osciloscopio:

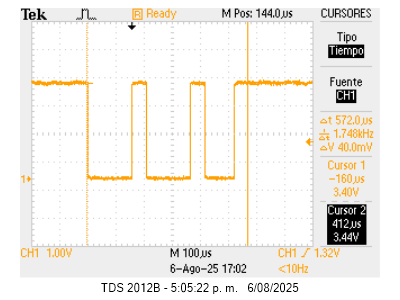
Bit de inicio Paridad

**

LSB MSB Bit de parada

*Imagen 7. Analisis carácter “D”, paridad par*

Al conectar el osciloscopio, la señal medida se observa en la siguiente imagen:



*Imagen 8. Carácter “U”, tiempo de la trama*

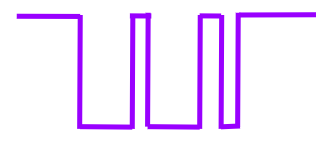
Ahora bien, el tiempo de la trama teórico se calcula a continuación:

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

*Paridad impar*

La imagen a continuación muestra la estructura típica de una trama UART al transmitir el carácter “D”, con paridad impar, destacando sus componentes esenciales y es lo que se espera observar en el osciloscopio:

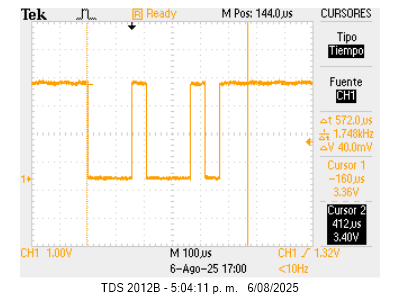
Bit de inicio Paridad

**

LSB MSB Bit de parada

*Imagen 9. Analisis carácter “D”, paridad impar*

Al conectar el osciloscopio, la señal medida se observa en la siguiente imagen:



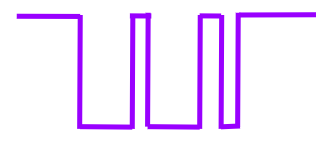
*Imagen 10. Carácter “U”, tiempo de la trama*

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

*Sin paridad*

La imagen a continuación muestra la estructura típica de una trama UART al transmitir el carácter “D”, sin paridad, destacando sus componentes esenciales y es lo que se espera observar en el osciloscopio:

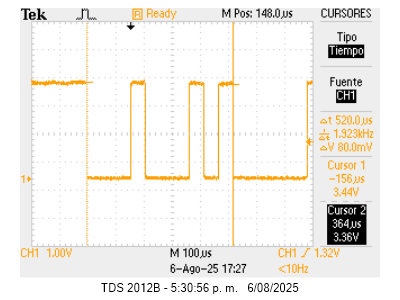
Bit de inicio Bit de parada

**

LSB MSB

*Imagen 11. Analisis carácter “D”, sin paridad*

Al conectar el osciloscopio, se envía la “D” acompañaba de otro carácter, para poder diferenciar el cambio de paridades, la señal medida se observa en la siguiente imagen:



*Imagen 12. Carácter “U”, tiempo de la trama*

Ahora bien, el tiempo de la trama teórico se calcula a continuación:

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

Finalmente, como se mencionó al inicio de este apartado, se enviaron 10 caracteres ASCII, los cuales se reflejan en la siguiente tabla, donde también se encuentra la cadena de bits de cada carácter según su paridad:

| Carácter | Paridad par | Paridad impar | Sin paridad |
| --- | --- | --- | --- |
| A | 01000001001 | 01000001011 | 0100000101 |
| B | 00100001001 | 00100001011 | 0010000101 |
| D | 00010001001 | 00010001011 | 0001000101 |
| G | 01110001001 | 01110001011 | 0111000101 |
| N | 00111001001 | 00111001011 | 0011100101 |
| M | 01011001001 | 01011001011 | 0101100101 |
| P | 00000101001 | 00000101011 | 0000010101 |
| K | 01101001001 | 01101001011 | 0110100101 |
| Y | 01001101001 | 01001101011 | 0100110101 |
| S | 01100101001 | 01100101011 | 0110010101 |

*Tabla 6. Cadena de bits de caracteres según paridad*

*Medida del tiempo de la trama*

Ahora, se modifica el programa para que envíe una trama ASCII de 60 caracteres enviados a 600 baudios, paridad par y 8 bis de datos. Entonces, se implementa el siguiente programa en el entorno de desarrollo integrado Thonny con Python:

import machine

import utime

from machine import Pin, UART

led = machine.Pin("LED", machine.Pin.OUT)

uart = UART(0, baudrate=600, bits=8, parity=0, tx=Pin(0), rx=Pin(1))

trama = "DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD"

while True:

led.on()

uart.write(trama)

utime.sleep(1)

led.off()

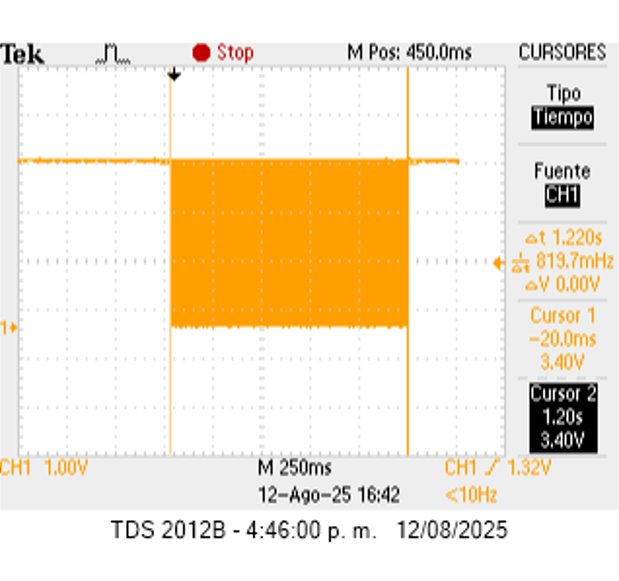
utime.sleep(1)

El tiempo de bit teórico se calcula a continuación:

El tiempo de carácter teórico se calcula a continuación:

El tiempo de trama teórico se calcula a continuación:

Se captura la señal resultante en el osciloscopio y se mide el tiempo total de la trama generada a continuación:

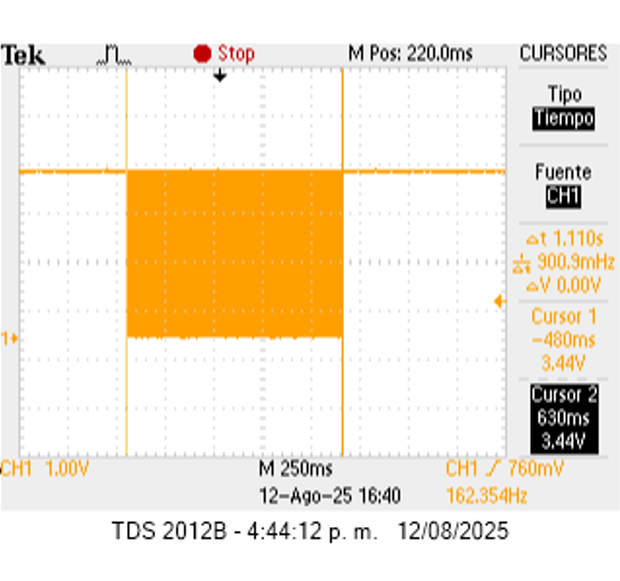


*Imagen 13. 60 caracteres, con paridad*

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado. Luego, se realiza la misma prueba anterior, pero se elimina el bit de paridad. El tiempo de carácter teórico se calcula a continuación:

El tiempo de trama teórico se calcula a continuación:

Se captura la señal resultante en el osciloscopio y se mide el tiempo total de la trama generada a continuación:



*Imagen 14. 60 caracteres, sin paridad*

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado. Para finalizar, se captura la trama “UMNG LIDER EN INGENIERIA EN TELECOMUNICACIONES” enviada a 57600 baudios, 7 bits de datos, paridad par y dos bits de parada. Mida el tiempo total del mensaje. Contraste los valores teóricos contra los experimentales.

El tiempo de bit teórico se calcula a continuación:

Y el tiempo de carácter se va a ver reflejado de la siguiente forma:

| Bit de inicio | Datos | Paridad | Bit de parada |
| --- | --- | --- | --- |

+1 bit +7 bits +1 bit +2 bits

El tiempo de carácter teórico se calcula a continuación:

El tiempo de trama teórico se calcula a continuación:

Se captura la señal resultante en el osciloscopio y se mide el tiempo total de la trama generada a continuación:



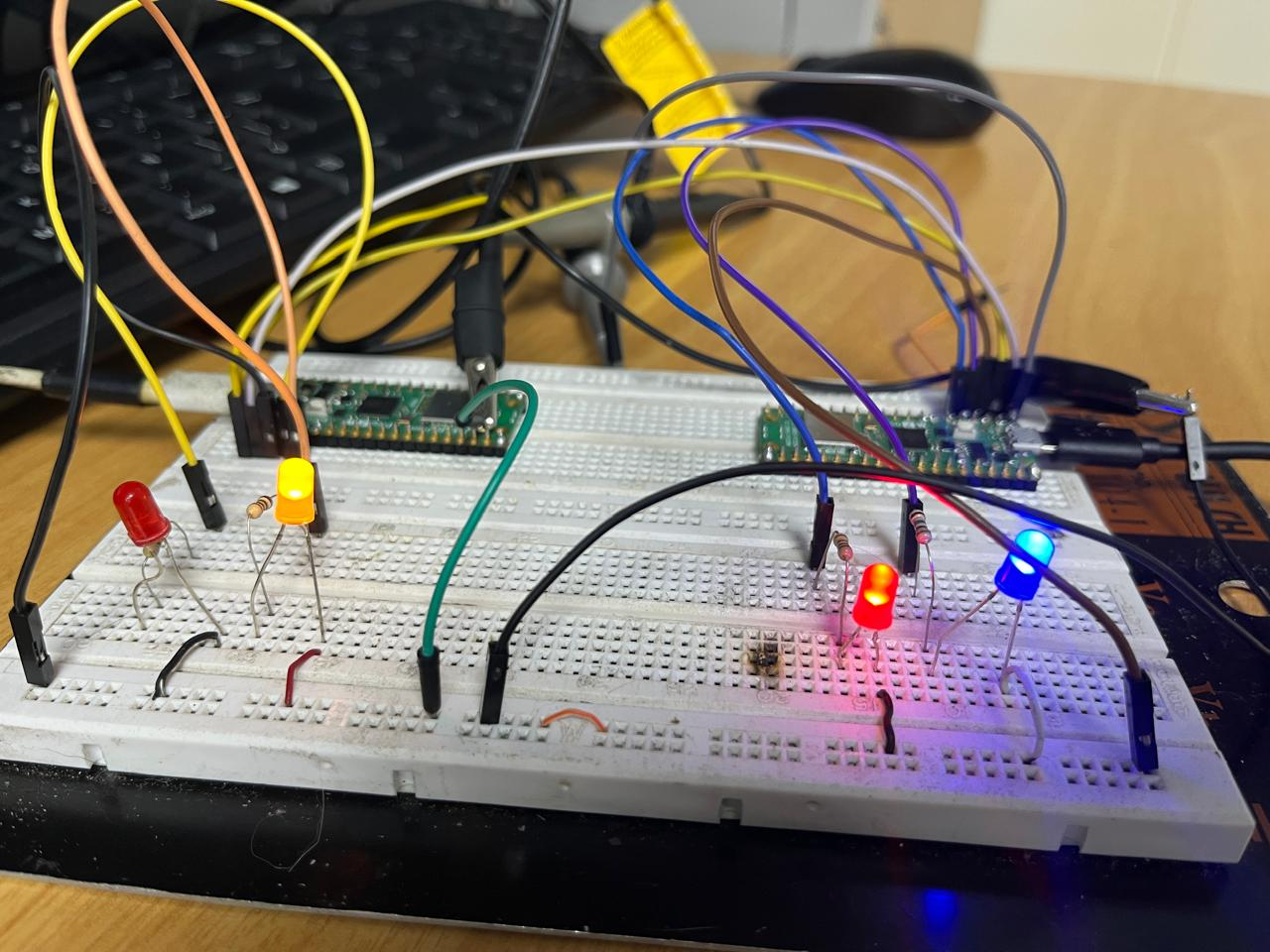
*Imagen 15. Mensaje de 46 caracteres*

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

*Reto de programación*

Ahora bien, se interconectan dos dispositivos Raspberry Pi Pico 2W cruzando las líneas TX y RX entre los dispositivos, con la misma señal GND de referencia entre los dos circuitos.

Primero se diseña un programa para transmisión TX.py y un programa para recepción RX.py para ejecutarse en los dispositivos Raspberry. El programa de transmisión envía cada segundo un carácter ASCII “A”. Al ser recibido este carácter el dispositivo de recepción enciende un led durante cinco segundos y envía como respuesta otro carácter “B” que al recibirse en el dispositivo transmisor, enciende un led que parpadea durante tres segundos. El proceso se repite de forma permanente. En las líneas de TX y RX de los dispositivos se conectan los canales del osciloscopio digital para analizar el proceso. En la siguiente imagen, se logra apreciar el montaje realizado:



*Imagen 16. Montaje del reto*

Se implementa el siguiente programa en el entorno de desarrollo integrado Thonny con Python, para el transmisor:

from machine import Pin, UART

import time

led\_tx = Pin(2, Pin.OUT)

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

while True:

uart.write('A')

print("TX: Enviado 'A'")

start\_time = time.time()

while time.time() - start\_time < 2:

if uart.any():

data = uart.read(1)

if data:

print(f"TX: Recibido '{data.decode()}'")

end\_time = time.time() + 3

while time.time() < end\_time:

led\_tx.value(1)

time.sleep(0.3)

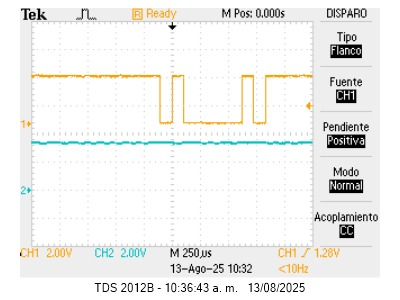
led\_tx.value(0)

time.sleep(0.3)

break

time.sleep(500)

Al conectar la línea del transmisor al osciloscopio por el canal 1, con el trigger en modo normal, se captura el envío del carácter “A”, como se observa a continuación:



*Imagen 17. Carácter “A” del transmisor*

Así mismo, se implementa el siguiente programa en el entorno de desarrollo integrado Thonny con Python, para el receptor:

from machine import Pin, UART

import time

led\_rx = Pin(2, Pin.OUT) # GPIO 2

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

contador = 0

while True:

if uart.any():

data = uart.read(1)

if data:

char = data.decode()

print(f"RX: Recibido '{char}'")

with open("recibidos.txt", "a") as f:

f.write(f"{contador} {char}\n")

led\_rx.value(1)

time.sleep(5)

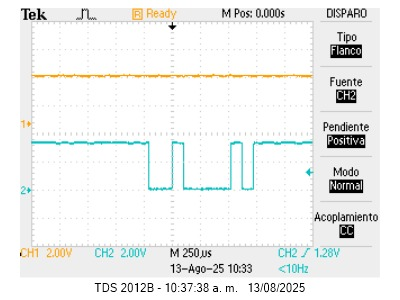
led\_rx.value(0)

uart.write('B')

print("RX: Enviado 'B'")

time.sleep(500)

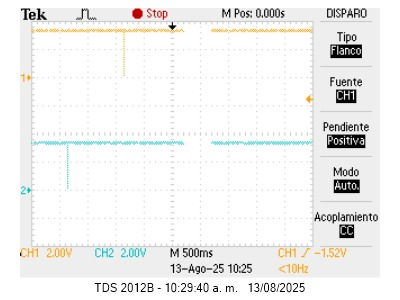
Al conectar la línea del receptor al osciloscopio por el canal 2, con el trigger en modo normal, se captura el envío del carácter “B”, como se observa a continuación:



*Imagen 18. Carácter “B” del receptor*

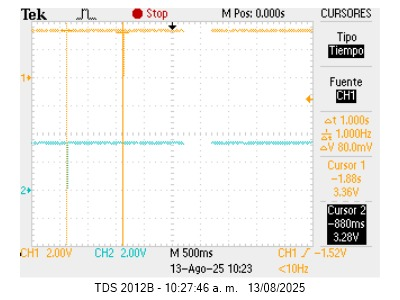
El dispositivo receptor debe además crear el archivo recibidos.txt donde incluye el contador de número de recepciones (Utilice un salto de línea para que quede un solo número por línea en el archivo txt).

Ahora bien, para comprobar adecuadamente la comunicación, se modifica el osciloscopio con el trigger en modo automático, con el fin de mirar las señales enviadas en cada una de las Raspberry, como se observa en la siguiente imagen:



*Imagen 19. Comunicación Half-Duplex*

Además de apreciar las gráficas, se confirma el tiempo de respuesta entre la transmisión y la recepción, el cual a cada código se le agregó un time sleep de 500 por lo cual sería de aproximadamente 1s la demora al momento de recepción del mensaje y su respuesta, esto se puede confirmar y probar de manera experimental como se muestra a continuación:



*Imagen 20. Comunicación Half-Duplex*

Entonces, se logra concluir que el tiempo de respuesta medido en el osciloscopio coincide con el tiempo de respuesta calculado y programado.

Luego se modifican los programas en el entorno de desarrollo integrado Thonny con Python, en ambas Raspberry, para realizar una comunicación Full-Duplex:

from machine import Pin, UART

import time

led\_parpadeo = Pin(2, Pin.OUT)

led\_encendido = Pin(3, Pin.OUT)

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

last\_send\_time = time.ticks\_ms()

blink\_end\_time = None

led\_on\_until = None

while True:

if time.ticks\_diff(time.ticks\_ms(), last\_send\_time) >= 10000:

uart.write('A')

print("Enviado 'A'")

last\_send\_time = time.ticks\_ms()

if uart.any():

data = uart.read(1)

if data:

char = data.decode()

print(f"Recibido '{char}'")

if char == 'A':

led\_encendido.value(1)

led\_on\_until = time.ticks\_add(time.ticks\_ms(), 5000)

print("Enviado 'B'")

elif char == 'B':

blink\_end\_time = time.ticks\_add(time.ticks\_ms(), 3000)

if led\_on\_until and time.ticks\_diff(led\_on\_until, time.ticks\_ms()) <= 0:

led\_encendido.value(0)

led\_on\_until = None

if blink\_end\_time:

if time.ticks\_diff(blink\_end\_time, time.ticks\_ms()) > 0:

led\_parpadeo.toggle()

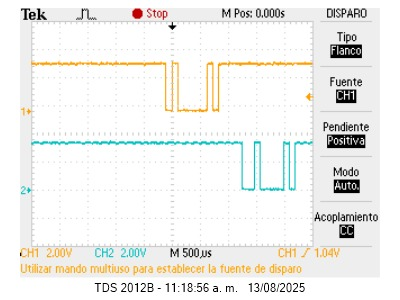
time.sleep(0.3)

else:

led\_parpadeo.value(0)

blink\_end\_time = None

Finalmente se utiliza el osciloscopio digital para validar el funcionamiento, como se observa a continuación:



*Imagen 21. Comunicación Full-Duplex*

Si se desea ver el funcionamiento del reto, en el repositorio se encuentra un video para ambas comunicaciones.

1. **CONCLUSIÓN**
2. **REFERENCIAS**
3. **REPOSITORIO DE GITHUB**

<https://github.com/Vallentincita/Comunicaci-n-digital-TEL-A->