Laboratorio RS232

Belky Valentina Giron Lopez
est.belky.giron@unimilitar.edu.co
Docente: José De Jesús Rúgeles

Resumen — En esta práctica se estudió la comunicación serial RS232 utilizando una Raspberry Pi Pico y un osciloscopio digital. Se configuró la UART en Micropython, se enviaron caracteres ASCII y se midieron los tiempos de bit para diferentes tasas de baudios, comprobando la relación teórica con los resultados experimentales. También se analizó la estructura de las tramas RS232, identificando los bits de inicio, datos, paridad y parada. Finalmente, se desarrolló un ejercicio de interconexión entre dos dispositivos mediante programas de transmisión y recepción, lo que permitió validar la comunicación en modo Half-Duplex y Full-Duplex.

En general, la práctica permitió entender el funcionamiento del estándar RS232, utilización del osciloscopio y uso de la programación básica en Micropython para la transmisión de datos.

Abstract -- In this practical exercise, RS232 serial communication was studied using a Raspberry Pi Pico and a digital oscilloscope. The UART was configured in Micropython, ASCII characters were sent, and bit times were measured for different baud rates, verifying the theoretical relationship with experimental results. The structure of RS232 frames was also analyzed, identifying the start, data, parity, and stop bits. Finally, an interconnection exercise was developed between two devices using transmission and reception programs, which allowed for the validation of communication in half-duplex and full-duplex modes.

Overall, the practical exercise provided an understanding of how the RS232 standard works, the use of the oscilloscope, and basic programming in Micropython for data transmission.

I. Introducción

En esta práctica trabajamos con la comunicación serial **RS232**, un protocolo muy utilizado para transmitir información entre dispositivos. El objetivo fue entender de manera sencilla cómo funciona este tipo de comunicación, midiendo en el osciloscopio las señales generadas y comprobando la relación entre la tasa de baudios y el tiempo de bit.

Para lograrlo utilizamos una Raspberry Pi Pico 2W programada en Micropython, desde la cual enviamos diferentes caracteres ASCII y analizamos cómo cambiaban las tramas al modificar la paridad o la velocidad de transmisión. Además, realizamos una interconexión entre dos microcontroladores, creando programas de transmisión y recepción que nos permitieron observar en la práctica cómo se establece la comunicación entre equipos.

II. Desarrollo de experimentos

Para iniciar la práctica se configuró la Raspberry Pi Pico con el firmware de Micropython. Se descargó el archivo .UF2 desde la página oficial de Raspberry Pi y se cargó en el dispositivo. Una vez instalada la interfaz de programación, se utilizó el programa Thonny como entorno principal para escribir y ejecutar los códigos en Micropython.

Luego se identificaron los pines de la UART0 de la Raspberry Pi Pico 2W, en donde el pin 1 corresponde al TX y el pin 3 a GND. Posteriormente, se conectó el osciloscopio digital entre TX y tierra para observar las señales transmitidas.

Continuamos programando la Pico desde Thonny para enviar caracteres ASCII a través del terminal TX. Al ejecutar el código, el osciloscopio capturó las señales generadas. Con esto fue posible medir el tiempo de bit y compararlo con el valor teórico calculado a partir de la tasa de baudios configurada.

Respondiendo las preguntas del punto 3.4

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

En Micropython, la clase disponible para trabajar con comunicación serial es machine.UART. Esta clase permite inicializar, configurar y utilizar los puertos UART de la Raspberry Pi Pico 2W para enviar y recibir datos en el estándar RS232.

- Cree una tabla con los métodos disponibles para la clase UART. Explique cada uno de ellos.

Método	Descripción
UART (id, baudrate, ...)	Crea e inicializa un objeto UART. El parámetro id selecciona el puerto (ejemplo: 0 o 1) y baudrate define la velocidad de transmisión.
.init (baudrate, bits, parity, stop, …)	Configura los parámetros del puerto UART: tasa de baudios, número de bits de datos, paridad (ninguna, par,

	impar) y número de bits de parada.
. read(nbytes)	Lee hasta nbytes caracteres del UART. Si no se especifica, lee todos los datos disponibles.
. write(buf)	Envía datos al puerto UART. El argumento puede ser un string, un byte o un arreglo de bytes.

- ¿Como se modifica la tasa de baudios?

La tasa de baudios se puede establecer o modificar al momento de crear el objeto UART o mediante el método. init(), que para este caso seria el baudrate en el código.

Continuando con el desarrollo de la práctica se envió el carácter ASCII “W” a diferentes tasas de baudios (de 300 a 921600). Se midieron los tiempos de bit en el osciloscopio, se tabularon los resultados y se calculó el porcentaje de error frente a los valores teóricos.

Empezamos con las graficas obtenidas en el osciloscopio donde podemos observar el voltaje máximo y mínimo y también el tiempo de bit experimental.

Baudios	Tiempo de bit (Teo) S	Tiempo de bit (Exp)S	Δt	% error	Voltaje máximo	Voltaje mínimo
300	3.33 m	3.300 m		0.9	-40 mV	3.40V
1200	833.3 u	830.0 u		0.9	-40 mV	3.44V
19200	52.08 u	52.00 u		0.15	-120 mV	3.48 V
57600	17.36 u	17.40 u		0.2	-80 mV	3.48 V
115200	8.681 u	8.700 u		0.21	-80 mV	.3.48 V
921600	1.085 u	1.080 u		0.4	-80 mV	3.48 V

Tabla 1. Diferentes tasas de baudios.

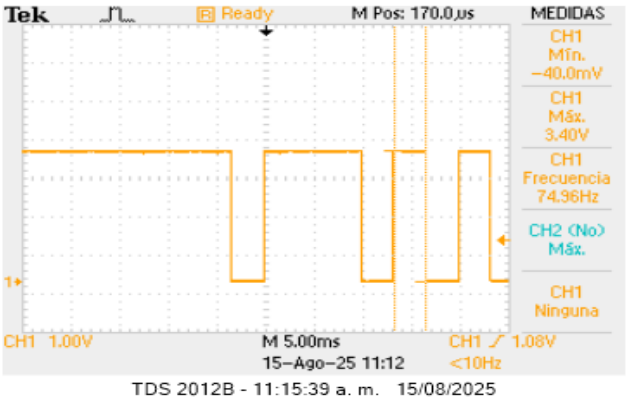


Fig. 1. W a 300 baudios Voltaje

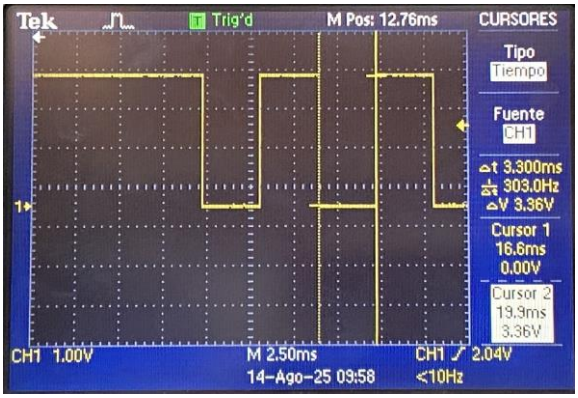


Fig. 2. W a 300 baudios tiempo de bit.

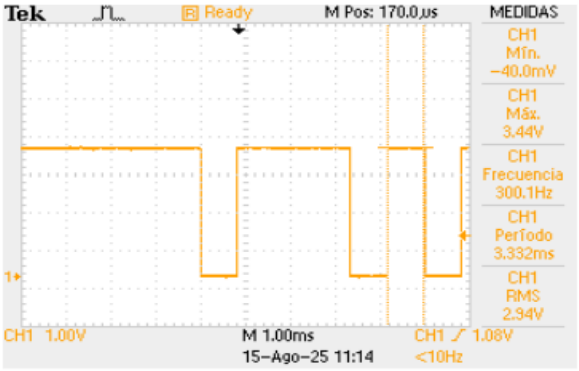


Fig. 3. W a 1200 baudios Voltaje.

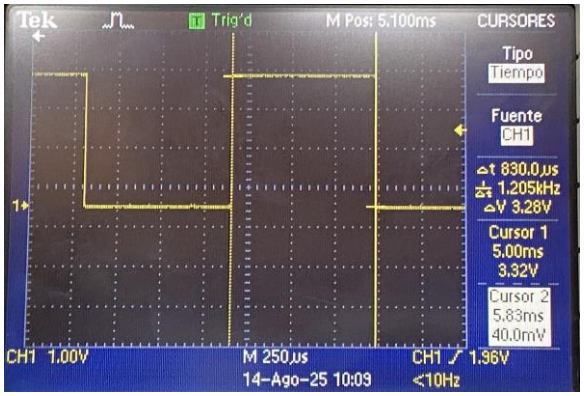


Fig. 4. W a 1200 baudios tiempo de bit.

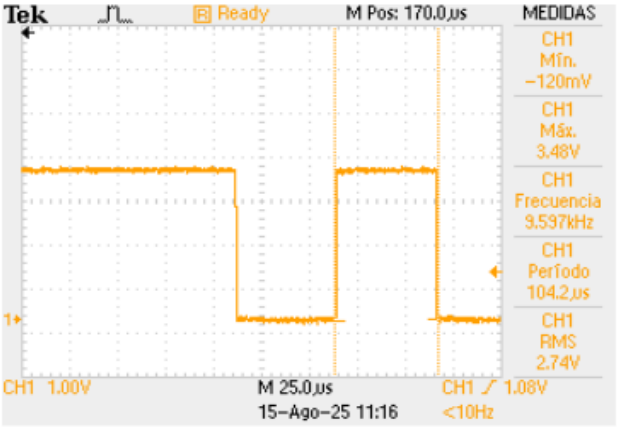


Fig. 5. W a 19200 baudios voltaje.

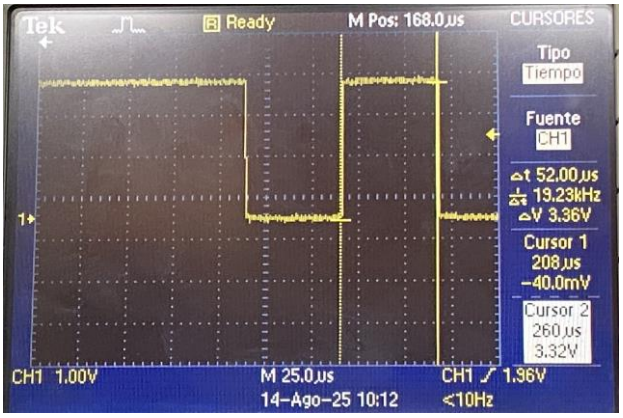
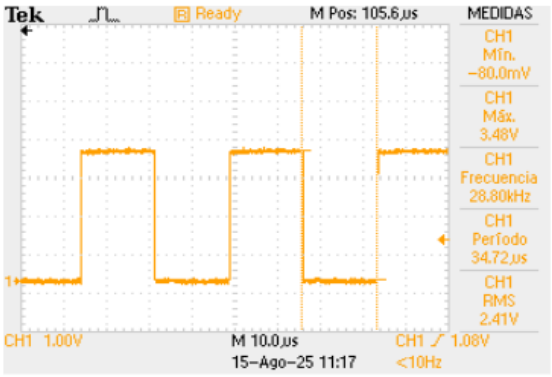


Fig. 6. W a 19200 baudios tiempo de bit.



TDS 2012B - 11:21:03 a. m. 15/08/2025

Fig. 7. W a 57600 baudios voltaje.

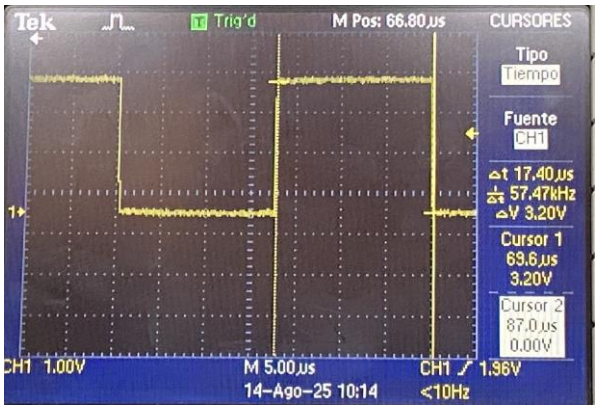
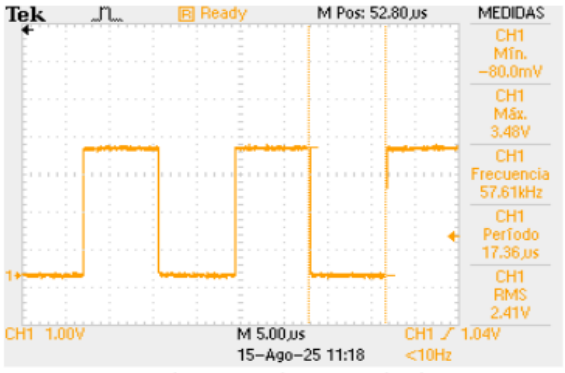


Fig. 8. W a 57600 baudios tiempo de bit.



TDS 2012B - 11:22:03 a. m. 15/08/2025

Fig. 9. W a 115200 baudios voltaje.

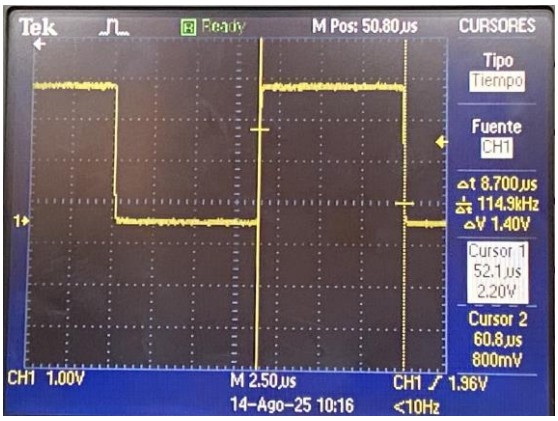
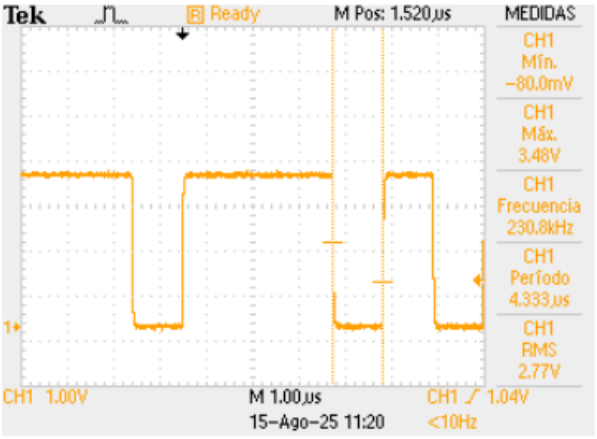


Fig. 10. W a 115200 baudios tiempo de bit.



TDS 2012B - 11:24:07 a. m. 15/08/2025

Fig. 11. W a 921600 baudios voltaje.

Cuando envío el mismo carácter ASCII y solo cambio la tasa de baudios, la información en sí no cambia, sigue siendo el mismo carácter y los mismos bits. Lo que cambia es la velocidad con la que viaja ese dato.

Si los baudios bajan, cada bit dura más tiempo y la señal se ve más ancha en el osciloscopio, por lo que el mensaje tarda más en llegar. En cambio, si aumento los baudios, los bits se transmiten más rápido y la trama ocupa menos tiempo; en el osciloscopio los pulsos se ven más estrechos.

Para calcular el tiempo de bit utilizamos la siguiente operación:

$$t_b = \frac{1}{\text{baudios}}$$

Fig. 13. Calculó tiempo de bit.

También utilizamos la siguiente operación para calcular el tiempo de trama la cual sería nuestro Δt en la tabla 1:

$$T_{\text{trama}} = \frac{N_{\text{bits}}}{\text{baudios}}$$

Fig. 14. Calculó tiempo de trama.

Debemos que el número de bits puede variar de acuerdo a la estructura de la trama, en este caso vamos a tener en cuenta la siguiente estructura:

$$N_{\text{bits}} = 1\ (\text{start}) + N_{\text{datos}} + N_{\text{paridad}} + N_{\text{stop}}$$

Fig. 15. Calculó número de bits.

Continuando con la practica realizamos el envió de 10 caracteres ASCII empleando una tasa de baudios de 9600 y para la trama una estructura de 8 bits de datos y paridad par.

Para realizar esto empleamos el siguiente código en el cual solo cambiamos el carácter enviado en la línea 8 que es la de uart.write (“”).

```
1  import machine
2  import utime
3  from machine import Pin,UART
4  led = machine.Pin("LED", machine.Pin.OUT)
5  uart = UART(0, baudrate=9600,bits=8,parity=0, tx=Pin(0), rx=Pin(1))
6  while True:
7   led.on()
8   uart.write("U")
9   utime.sleep(1)
10  led.off()
11  utime.sleep(1)
```

Fig. 16. Código en Thonny para envió de caracteres ASCII.

Los 10 caracteres enviados son: gLSyCcilaj

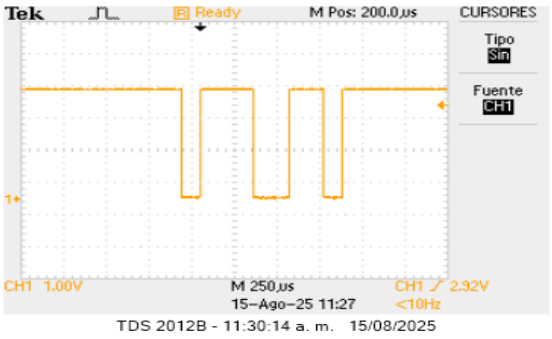


Fig. 17. Carácter g.

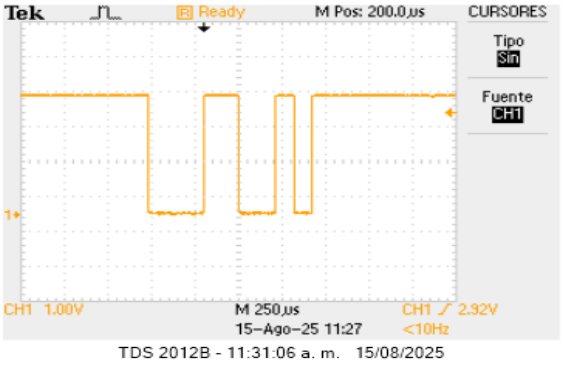


Fig. 18. Carácter L.

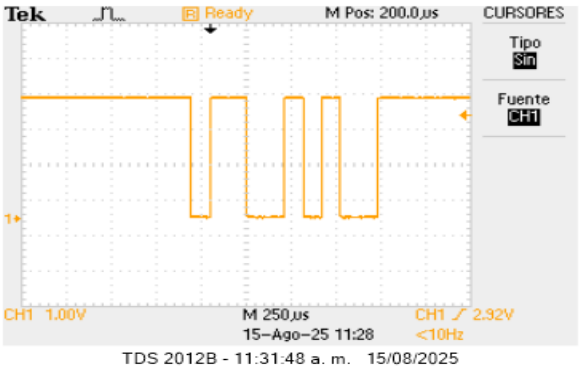


Fig. 19. Carácter S.

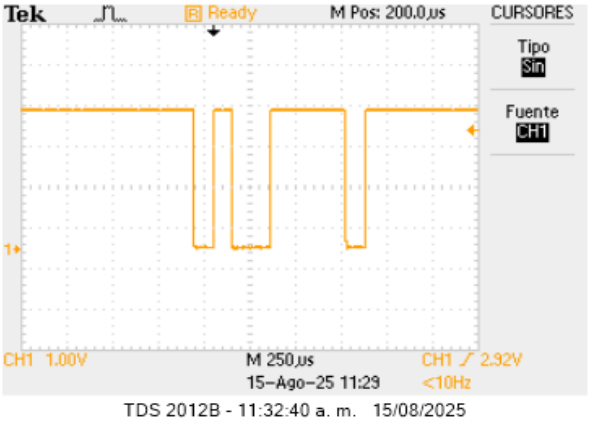


Fig. 20. Carácter y.

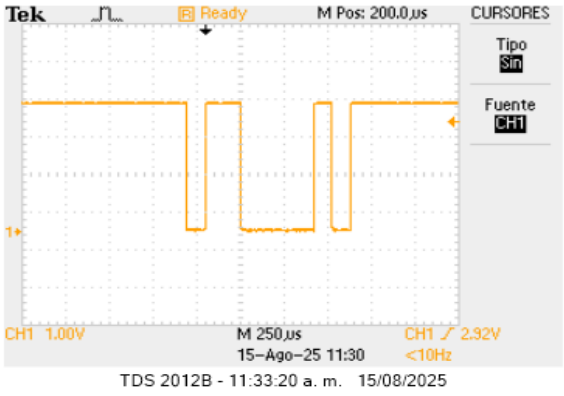


Fig. 21. Carácter C.

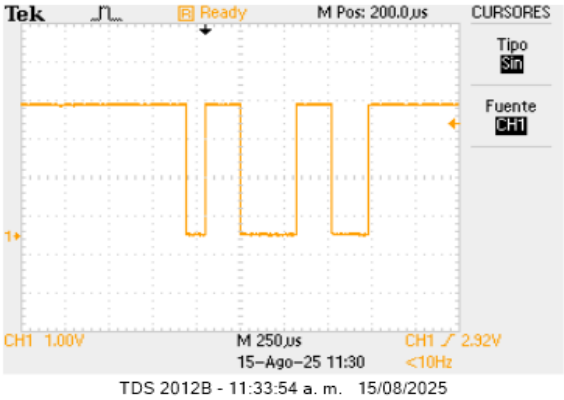


Fig. 22. Carácter c.

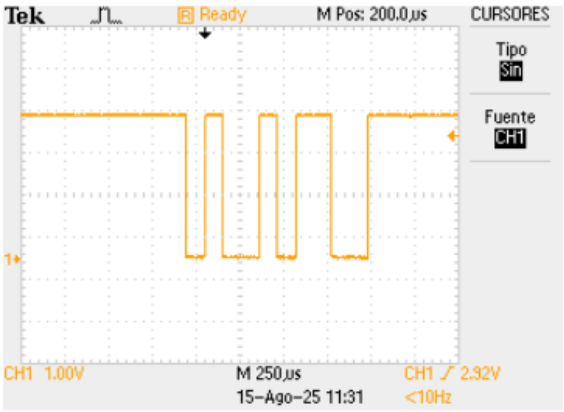


Fig. 23. Carácter i.

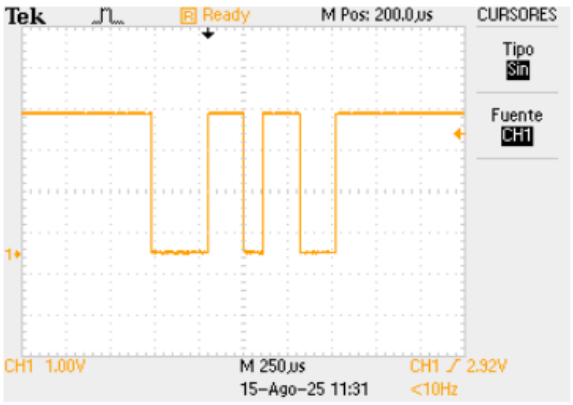


Fig. 24. Carácter l.

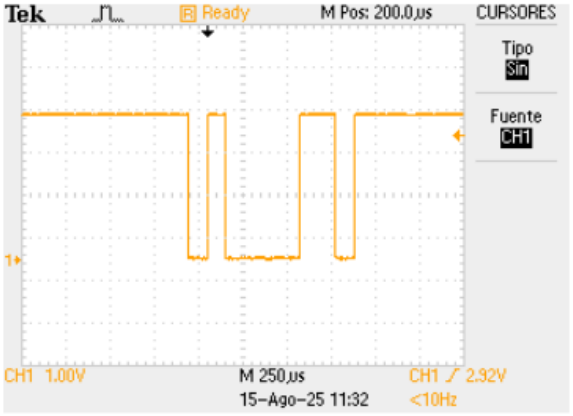


Fig. 25. Carácter a.

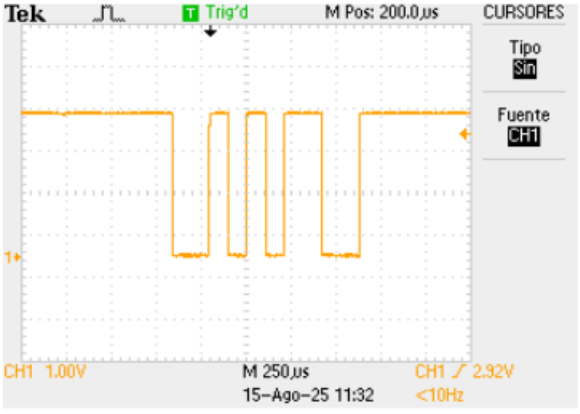


Fig. 26. Carácter j.

Cuando observamos la señal en el osciloscopio, lo que realmente estamos viendo son los ceros y unos del carácter enviado, pero representados como pulsos. Cada parte de la trama tiene un papel importante dentro del protocolo RS232:

- Bit de inicio (start bit): siempre es un 0 lógico que indica al receptor que viene un nuevo dato.

- Bits de datos: corresponden al valor binario del carácter ASCII enviado. Estos se transmiten comenzando por el LSB (bit menos significativo) y terminando en el MSB (bit más significativo).
- Bit de paridad: sirve como verificación para saber si hubo un error en la transmisión. Puede ser par, impar o no usarse.
- Bit de parada (stop bits): siempre son 1 lógico y marcan el final de la trama.

En el osciloscopio, esto se ve como una serie de pulsos que bajan y suben.

Luego para realizar el análisis de las señales vamos a hacerlas de manera manual para comprender la estructura de la trama mediante el dibujo de la señal teniendo en cuenta los caracteres de la trama.

Y para esto tendremos en cuenta la siguiente tabla donde cada carácter ASCII tiene un valor en binario para la parte de bit de datos en la estructura de la trama.

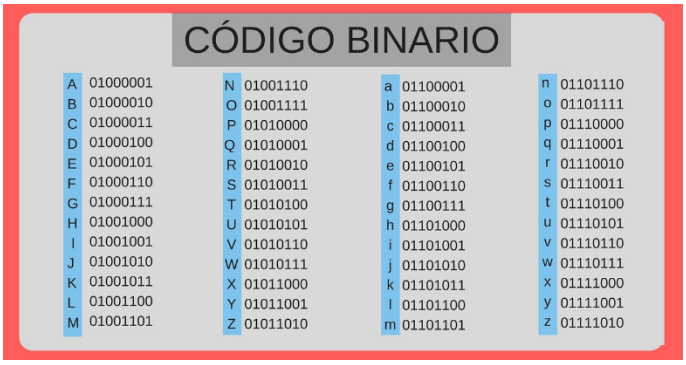


Fig. 27. Tabla caracteres ASCII en binario.

Para este análisis manual utilizaremos las señales de los caracteres ASCII gLl.

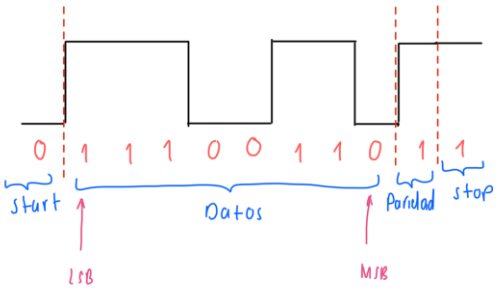


Fig. 28. Grafica carácter g manual.

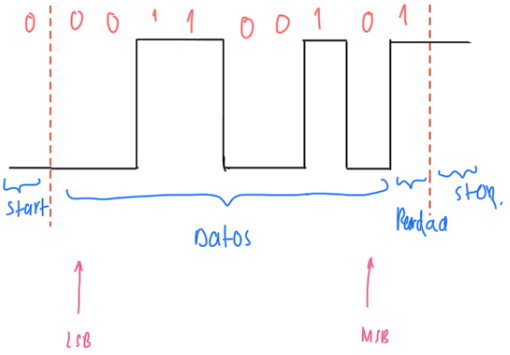


Fig. 29. Grafica carácter L manual.

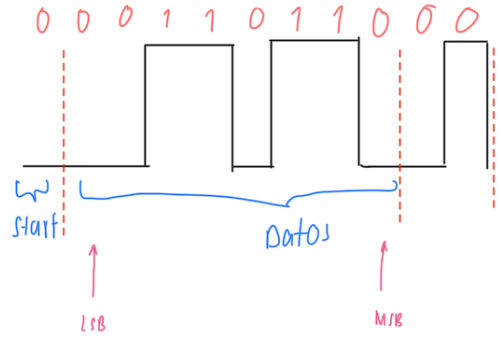


Fig. 30. Grafica carácter l manual.

Para continuar con la practica analizaremos los mismos 3 caracteres ASCII con paridad impar y ver como esta influye en la señal.

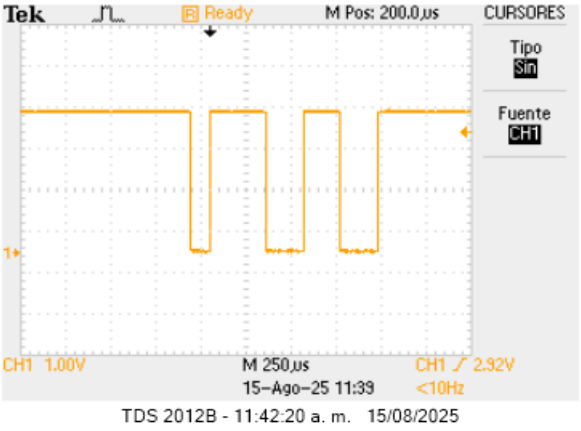


Fig. 31. Carácter g paridad impar.

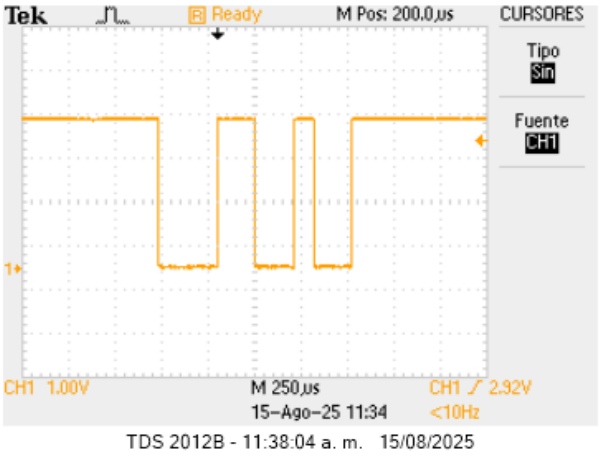


Fig. 32. Carácter L paridad impar.

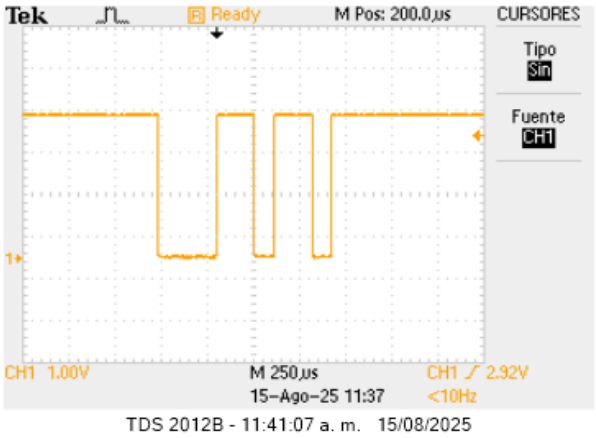


Fig. 33. Carácter l paridad impar.

Después de comparar las gráficas de paridad par e impar podemos concluir que el bit de paridad se añade después de los 8 bits de datos y su objetivo es que la cantidad total de unos en la trama solo en los bits de datos + paridad sea impar.

La única diferencia entre paridad par e impar es el valor del bit de paridad nivel alto = 1 o bajo = 0, todo lo demás bit de start, bits de datos y bit de stop quedan igual.

En el osciloscopio podíamos observar que en las anteriores graficas con paridad par el bit de paridad era 1, al cambiar a impar ese bit se convierte en 0, así que en la zona donde aparece el bit de paridad se verá un pulso alto o bajo distinto. La anchura de los pulsos duración de cada bit no cambia porque la velocidad (baudios) sigue siendo la misma.

Esto lo podemos observar con un ejemplo en concreto que es el carácter g: los datos de g tienen 5 unos.

- Con paridad par la paridad fue 1 lo que da como resultado 5+1 = 6 → par.
- Con paridad impar la paridad es 0 lo cual da como resultado 5+0 = 5 → impar).

En el osciloscopio, el bit justo antes del bit de stop aparecerá en nivel alto con paridad par, y en nivel bajo con paridad impar.

Continuamos con el análisis de las mismas señales, pero sin paridad.

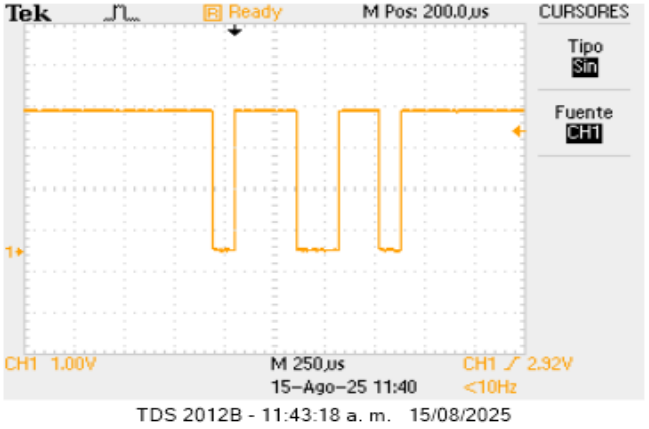


Fig. 34. Carácter g sin paridad.

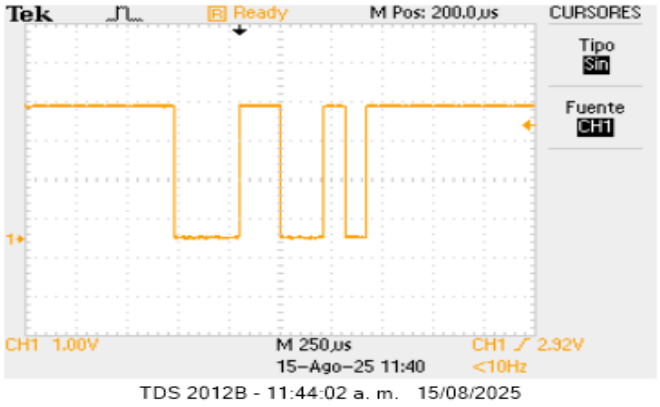


Fig. 35. Carácter L sin paridad.

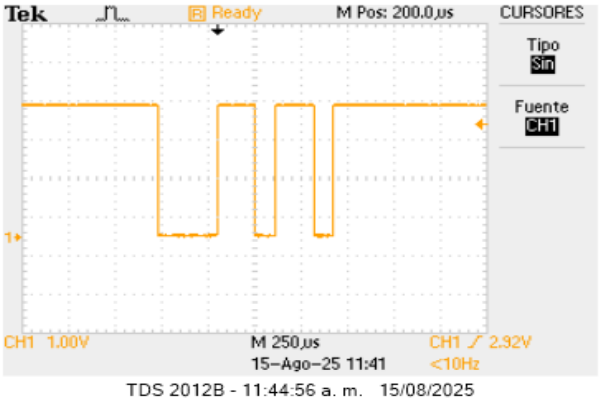


Fig. 36. Carácter l sin paridad.

Cuando quitamos el bit de paridad la trama pasa de ser bit de start + 8 bits de datos + bit de paridad + bit de stop **a ser** bit start + 8 bits de datos + bit de stop.

En la señal podemos observar que cambia:

- La duración total de la trama disminuye porque hay un bit menos.
- En el osciloscopio en la gráfica de la señal ya no se verá ese pulso extra entre los datos y el stop, el stop aparece inmediatamente después del último bit de datos.
- La forma de los bits de datos no cambia, solo el orden temporal lo cual hace que la trama termina antes.

Ejemplificando lo dicho anteriormente para comprender mejor el efecto de quitar el bit de paridad obtenemos:

- (8 bits + 1 stop)

  N° bits = 10 → T=10/9600 = 1,0417 ms

  T=10/9600=1,0417 Ms

Es decir, que el carácter tarda 0.104 mS menos sin paridad.

Luego continuamos con la medición del tiempo total requerido para el envió del carácter, para ello empleamos la siguiente formula:

$$T_{\text{carácter}} = \frac{N_{\text{bits}}}{\text{baudrate}}$$

Fig. 37. Calculo tiempo de carácter.

Cuando medimos en el osciloscopio el tiempo total de cada carácter transmitido, lo que estamos haciendo es sumar la duración de todos los bits que lo componen: el bit de inicio, los bits de datos, el bit de paridad (si está activado) y el bit de parada.

En la práctica, vimos que el tiempo total depende directamente de la tasa de baudios. A menor tasa de baudios, cada bit dura más tiempo, por lo tanto, el carácter completo tarda más en transmitirse. En cambio, a mayor tasa de baudios, los bits son más cortos y el mismo carácter viaja mucho más rápido.

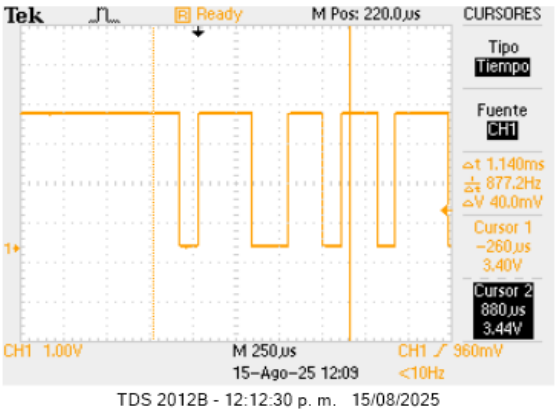


Fig. 38. Tiempo de carácter g con paridad par.

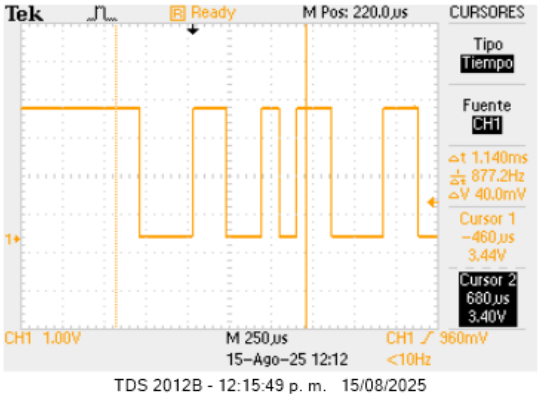


Fig. 39. Tiempo de carácter L con paridad par.

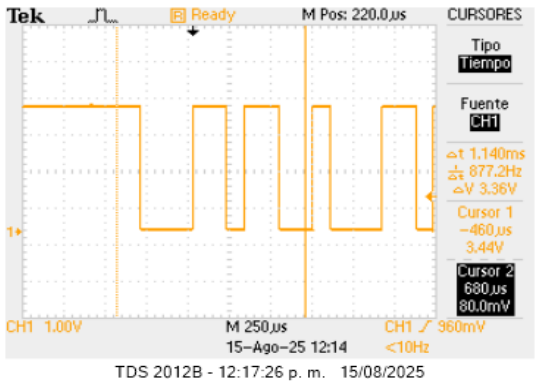


Fig. 40. Tiempo de carácter l con paridad par.

Luego realizamos una modificación en el código para enviar una trama ASCII de 60 caracteres enviados a 600 baudios, paridad par y 8 bis de datos.

Empleamos el siguiente código para la visualización de la señal:

```
Thonny - C:\Users\User\OneDrive\Documentos\SEXTO SEMESTRE UMNG\COMUNICACIONES DIGITALES\PRIMER CORTE\EXP3\CODIGO.py @ 9 : 22
Fichero  Editar  Visualizar  Ejecutar  Herramientas  Ayuda
CODIGO.py
1  import machine
2  import utime
3  from machine import Pin, UART
4  led = machine.Pin("LED", machine.Pin.OUT)
5  uart = UART(0, baudrate=600, bits=8, parity=1, tx=Pin(0), rx=Pin(1))
6
7  while True:
8      led.on()
9      uart.write("Fibra y satelites llevan datos y voz a todo el planeta   :) ")
10     utime.sleep(1)
11     led.off()
12     utime.sleep(1)
13
```

Fig. 41. Código Thonny 60 caracteres.

Luego de ejecutar el código en Thonny obtenemos la siguiente grafica en el osciloscopio:

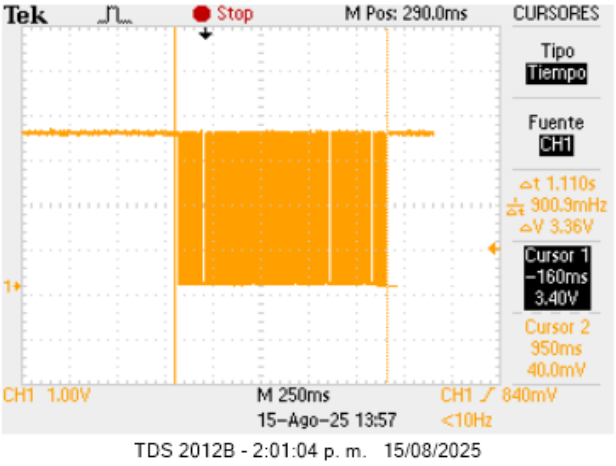


Fig. 42. Frase 60 caracteres.

Luego realizamos el cálculo teórico para compararlo con el obtenido de manera experimental en el osciloscopio, aplicamos las siguientes ecuaciones:

$$t_b = \frac{1}{\text{baudios}} = \frac{1}{600} = 0{,}0016667\ \text{s} = 1{,}6667\ \text{ms}$$

Fig. 43. Tiempo de bit frase de 60 caracteres.

$$T_{\text{car}} = \frac{11}{600} = 0{,}0183333\ \text{s} = 18{,}333\ \text{ms}$$

Fig. 44. Tiempo de carácter frase de 60 caracteres.

$$\text{Total bits} = 60 \times 11 = 660\ \text{bits}$$

$$T_{\text{trama60}} = \frac{660}{600} = 1{,}1\ \text{s}$$

Fig. 45. Tiempo de trama frase de 60 caracteres.

Analizando los resultados experimentales y los teóricos obtenemos un porcentaje de error pequeño.

Luego eliminamos el bit de paridad y analizamos los cambios de estructura de la gráfica y trama.

En el código empleado en la figura 41 solo cambiamos la línea 5 en la parte de parity y lo ponemos = None.

Y obtenemos la siguiente gráfica:

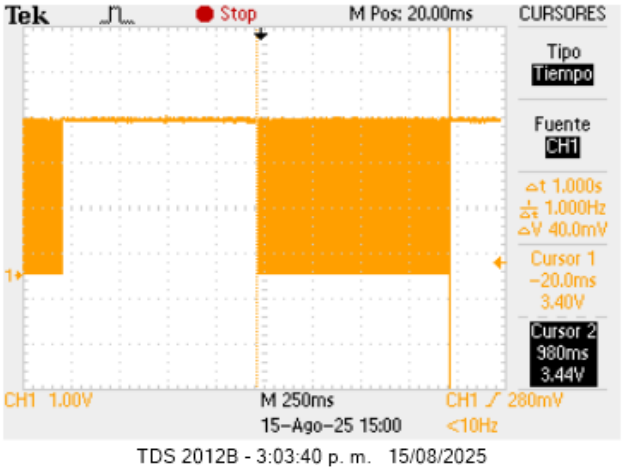


Fig. 46. Frase 60 caracteres sin paridad.

Para comparar el resultado experimental obtenido realizamos los siguientes cálculos:

$$t_b = \frac{1}{600} = 0.0016667\ \text{s} = 1.6667\ \text{ms}$$

Fig. 47. Tiempo de bit frase de 60 caracteres sin paridad.

$$T_{\text{car}} = \frac{10}{600} = 0.0166667\ \text{s} = 16.667\ \text{ms}$$

Fig. 48. Tiempo de carácter frase de 60 caracteres sin paridad.

$$T_{60} = \frac{60 \times 10}{600} = 1.0\ \text{s}$$

Fig. 49. Tiempo de trama frase de 60 caracteres sin paridad.

Con estos cálculos podemos concluir que el porcentaje de error entre la medida experimental y el cálculo teórico es del 0%.

Luego enviamos la frase "UMNG LIDER EN INGENIERIA EN TELECOMUNCACIONES" cambiamos la tasa de baudios a

57600, los bits de datos son 7, paridad par y dos bits de parada para esto empleamos el siguiente código en Thonny:

```python
import machine
import utime
from machine import Pin, UART

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

# 57600 baudios, 7 bits de datos, paridad par, 2 bits de parada
uart = UART(0, baudrate=57600, bits=7, parity=0, stop=2, tx=Pin(0), rx=Pin(1))

while True:
    led.on()
    uart.write("UMNG LIDER EN INGENIERIA EN TELECOMUNCACIONES")
    utime.sleep(1)
    led.off()
    utime.sleep(1)
```

Fig. 50. Código Thonny frase umng.

Después de ejecutarlo obtenemos la siguiente grafica en el osciloscopio:

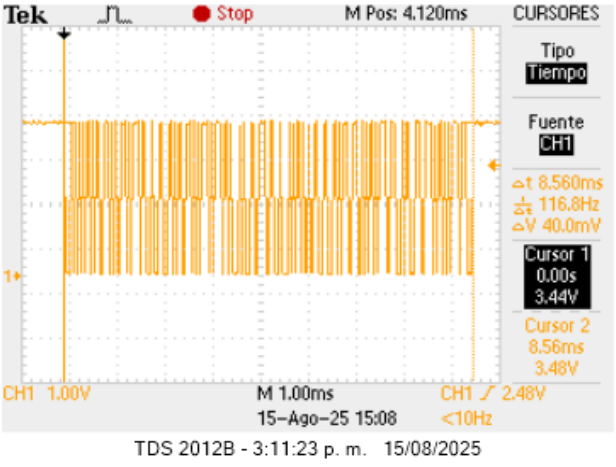


Fig. 51. Frase umng.

Para comparar el resultado experimental obtenido realizamos los siguientes cálculos:

$$t_b = \frac{1}{57600} = 1.7361 \times 10^{-5}\ \text{s} \approx \mathbf{17.36}\ \boldsymbol{\mu}\mathbf{s}$$

Fig. 52. Tiempo de bit frase umng paridad par.

$$T_{\text{car}} = \frac{11}{57600} = 1.9097 \times 10^{-4}\ \text{s} \approx \mathbf{190.97}\ \boldsymbol{\mu}\mathbf{s}$$

Fig. 53. Tiempo de carácter frase umng paridad par.

$$T_{\text{msg}} = \frac{45 \times 11}{57600} = 8.59375 \times 10^{-3}\ \text{s} \approx \mathbf{8.594}\ \text{ms}$$

Fig. 54. Tiempo de trama frase umng paridad par.

Con estos cálculos podemos concluir que el porcentaje de error entre la medida experimental y el cálculo teórico es del 0.395% lo cual es muy bajo y se puede deber a la posición de los cursores.

Continuamos con el desarrollo de la practica en la parte del reto, para la comunicación half dúplex utilizamos los siguientes códigos:

RX

```
from machine import UART, Pin

import utime


# Configuración UART0 (TX=GPIO0, RX=GPIO1)

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


# LED en GPIO14

led = Pin(14, Pin.OUT)

led.value(0)  # LED apagado al inicio


# Archivo de registro

log_file = "rx_log.txt"

count = 0


def registrar_recepcion(n):

    """Guarda el número de recepciones en un archivo de texto"""

    with open(log_file, "a") as f:

        f.write(str(n) + "\n")


while True:

    # Verificar si hay datos disponibles

    if serial.any() > 0:
```

```
        dato = serial.read(1)  # leer un byte
        if dato:
            char = dato.decode("utf-8")
            print("Dato recibido:", char)

            # Encender LED por 5 segundos (en dos intervalos de
2.5s)
            led.on()
            utime.sleep(2.5)
            led.off()
            utime.sleep(2.5)

            # Incrementar y registrar contador
            count += 1
            registrar_recepcion(count)

            # Enviar respuesta
            serial.write("R")
            print("Respuesta enviada: R")
```

TX

```
from machine import UART, Pin
import utime

# Configuración UART0
serial = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))

# LED en GPIO14
led = Pin(14, Pin.OUT)

def parpadear (duracion_ms, intervalo=400):
    """Hace parpadear el LED durante cierto tiempo"""
    fin = utime.ticks_add(utime.ticks_ms(), duracion_ms)
    while utime.ticks_diff(fin, utime.ticks_ms()) > 0:
        led.value(1)
        utime.sleep_ms(intervalo // 2)
        led.value(0)
        utime.sleep_ms(intervalo // 2)

while True:
    # Enviar carácter cada 2 segundos
    simbolo = "X"
    serial.write(simbolo)
    print("Dispositivo TX: enviado ->", símbolo)
    utime.sleep(2)}

    # Revisar si hay respuesta en el buffer
    if serial.any():
        dato = serial.read(1)
        if dato:
            respuesta = dato.decode("utf-8")
            print("Dispositivo TX: recibido <-", respuesta)

            # Hacer parpadear LED por 3 segundos
            parpadear(3000, intervalo=300)
```

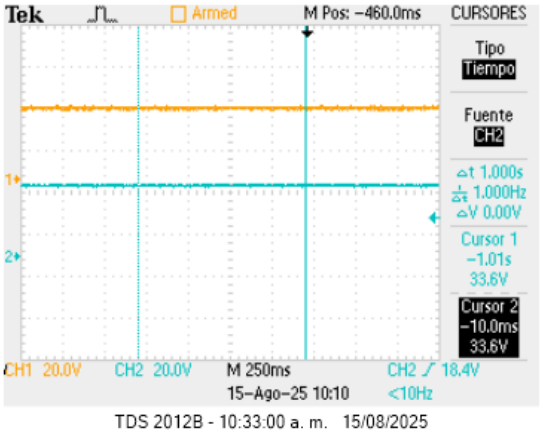


Fig. 55. Grafica Half dúplex.

Link repositorio GitHub: https://github.com/belkyvalentina11/Comunicaci-n-digital-.git