COMUNICACIÓN DIGITAL - EXPLORANDO LA COMUNICACIÓN RS232 (agosto de 2025)

Fernando Javier Riaño Rios
Est.fernando.riano@unimilitar.edu.co

Resumen - La comunicación serial sigue siendo clave en la instrumentación y el control de dispositivos electrónicos. En este trabajo comprobamos, de manera práctica, la relación entre la tasa de baudios (velocidad en bits por segundo) y el tiempo por bit (tb=1/baudios) También identificamos cómo se arma la trama UART/RS-232: bit de inicio, datos (de LSB a MSB), paridad y bits de parada, y cómo cada parte afecta el tiempo total de transmisión. Usamos una Raspberry Pi Pico programada con MicroPython y un osciloscopio digital (acoplamiento DC y disparo en flanco negativo) para medir anchos de pulso a diferentes velocidades y comparar valores teóricos con los medidos. Además, conectamos dos placas para enviar y responder mensajes, con registro en archivo y aviso por LED, validando el intercambio en semidúplex y las pautas para hacerlo en *full-duplex*. Los resultados concuerdan con la teoría dentro de los límites del instrumento y muestran que la paridad y los bits de parada aumentan de forma lineal el tiempo por carácter y la duración total de la trama.

Abstract – Serial communication remains essential for instrumentation and control in electronic devices. This paper provides a hands-on validation of the relationship between baud rate (bit-per-second speed) and bit time (tb=1/baud) and examines the UART/RS-232 frame—start bit, data (LSB to MSB), parity, and stop bits—highlighting how each field impacts total transmission time. A Raspberry Pi Pico running MicroPython and a digital oscilloscope (DC coupling, negative-edge trigger) were used to measure bit widths at multiple speeds and to compare theoretical versus measured values. We also linked two boards to exchange messages with logging and LED signaling, validating half-duplex operation and outlining practical steps toward stable full-duplex. Results agree with theory within instrument limits and confirm that parity and stop bits increase character time and overall frame duration in a roughly linear way.

Índice de Términos - RS-232; UART; MicroPython; Raspberry Pi Pico; baudios; tiempo de bit; paridad; bits de Parada; osciloscopio; trama serial.

INTRODUCCIÓN

La comunicación serial asíncrona es muy usada por su cableado sencillo y estabilidad temporal. En la práctica conviven UART (TTL 3,3 V) y RS-232 (niveles ±V e inversión), por lo que distinguir sus diferencias eléctricas y de codificación evita errores. En UART cada carácter viaja en una trama con start, n datos (LSB→MSB), paridad opcional y stop; el tiempo de bit cumple t_b = 1/baudios y el tiempo por carácter es (1 + n_datos + n_paridad + n_stop)*t_b. Este laboratorio configura la UART de una Raspberry Pi Pico en MicroPython, captura tramas con osciloscopio y compara configuraciones con y sin paridad y distintos bits de parada. Por último, se enlazan dos placas para intercambio bidireccional y registro, validando los cálculos temporales, la idea es entender, con medidas reales, cómo cambia el tiempo por bit y por carácter según la configuración, y evitar errores típicos al conectar UART vs. RS-232.

I. DESARROLLO DEL LABORATORIO

1. Realizamos las configuraciones necesarias para conectar nuestro Raspberry Pi Pico 2w al PC y poder trabajar desde el entorno Tonny - Python, procedemos a enviar el carácter ASCII “U”, se conecta a un osciloscopio digital entre la terminal TX y tierra del dispositivo para medir el tiempo de bits del carácter.

Código usado:

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

# Configuración del LED integrado
led = machine.Pin("LED", machine.Pin.OUT)

# Configuración de la UART:
# UART 0
# Baudrate: 9600
# 8 bits de datos
# paridad (parity=0)
# TX en Pin 0, RX en Pin 1
uart = UART(0, baudrate=9600, bits=8, parity=0, tx=Pin(0), rx=Pin(1))

while True:
    led.on()          # Enciende el LED
    uart.write("U")    # Envía el carácter "U" por UART
    utime.sleep(1)     # Espera 1 segundo
    led.off()         # Apaga el LED
    utime.sleep(1)     # Espera 1 segundo
```

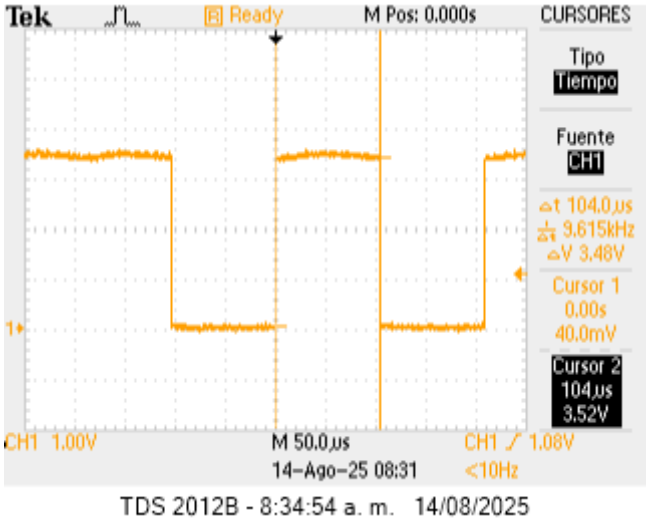

Ilustración 1 grafica del osciloscopio t bit caracter "U" 9600 baudios.

Calculamos el tiempo de bit:

$$Tb = \frac{1}{baudios} = \frac{1}{9600} = 104.1\ \mu s$$

Comparando el resultado teórico de tiempo de bit= 104,1 µs, es concordante con el mostrado en la grafica del osciloscopio, El porcentaje de error para esta medición es de:

$$\%error = \frac{|104.1\ \mu s - 104\ \mu s|}{104\ \mu s} * 100\% = 0{,}09\%$$

Es un porcentaje de error muy bajo y el cual se podría deber a la manipulación del osciloscopio el cual, al generar una mayor amplitud visual de la señal, esta se perdía y no permitía realizar la medición más exacta.

2.

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

La clase es machine.UART. Permite configurar y usar la comunicación serial asíncrona (UART/USART) desde el microcontrolador.

Es el periférico de serial asíncrona del microcontrolador (UART/USART). Te permite configurar velocidad (baudios), número de bits, paridad, bits de parada, pines TX/RX, buffers y timeouts, y hacer read/write de bytes.

¿Como se modifica la tasa de baudios?

Dos caminos válidos:

A. Fijarla al crear el objeto:
from machine import UART, Pin
uart = UART(0, baudrate=9600, bits=8, parity=None, stop=1, tx=Pin(0), rx=Pin(1))

B. Cambiarla “en caliente” con init()

uart.init(baudrate=57600) # solo velocidad
uart.init(baudrate=115200, bits=8, parity=None, stop=1) # velocidad + formato

Tabla con los métodos disponibles para la clase UART.

Metodo /parametro	Detalles
UART(id, ...) (constructor)	Crea/configura la UART de hardware. id: 0 ó 1 (UART0/UART1). Parámetros típicos: baudrate, bits (7/8; en algunos puertos 9), parity (None, 0=EVEN, 1=ODD), stop (1/2), tx=Pin(...), rx=Pin(...), txbuf/rxbuf
init(...)	Reconfigura “en caliente”. Puedes cambiar baudios, formato, pines, timeouts sin crear un objeto nuevo. Útil para barrer baudios en tus mediciones.
deinit()	Apaga la UART. Libera el periférico. Tras deinit() crea un objeto nuevo si quieres reusarla.
any()	Bytes disponibles en RX (no bloquea). Devuelve enteros. Patrón típico: if uart.any(): data=uart.read(...). Evita bloqueos.
read([n])	Lee hasta n bytes; si no das n, lee lo disponible. Retorna bytes o None si se agota timeout. Úsalo con any() para lectura no bloqueante.
readinto(buf[, n]	Lee directo a un buffer existente. Evita crear objetos (menos GC). Devuelve n bytes leídos o None. Bueno para alto rendimiento.
readline()	Lee hasta \n. Útil para protocolos “por línea”. Devuelve bytes o None en timeout.
write(buf)	Envía bytes por TX. Acepta bytes/bytearray/str (este último lo convierte). Devuelve n bytes escritos.
flush()	Espera a que TX quede vacío. Garantiza que todo salió por el cable antes de continuar (p. ej., antes de dormir el MCU).
txdone()	¿Terminó la cola de TX? True si no hay transmisión en curso. Útil para temporizar capturas en el osciloscopio.
sendbreak()	Envía un BREAK (línea en 0 sostenido). Algunos equipos RS-232 lo usan como señal especial de arranque/atención.
irq(handler, trigger, hard=False)	Interrupciones de UART. Permite callbacks por eventos (p. ej., RX). Útil si quieres reacción inmediata sin polling.

3. ahora se envía el caracter ASCII “W”. con 6 diferentes tasas de baudios (300, 600, 5000, 9600, 115200).

Tomamos los datos arrojados por el osciloscopio para luego proceder a hacer los respectivos cálculos teóricos y realizar las comparaciones.

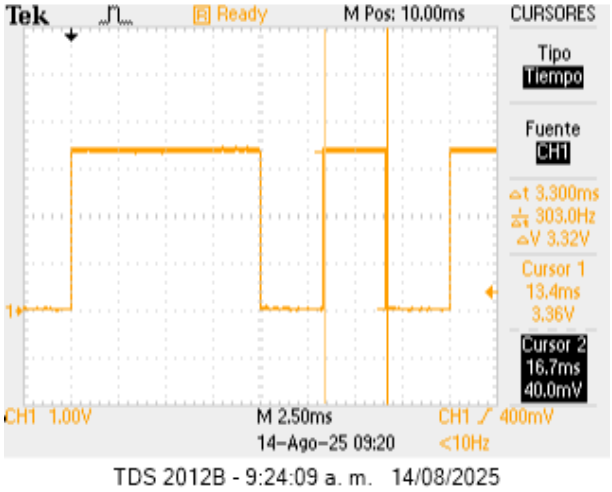

Ilustración 2 grafica del osciloscopio t bit caracter "W" 300 baudios.

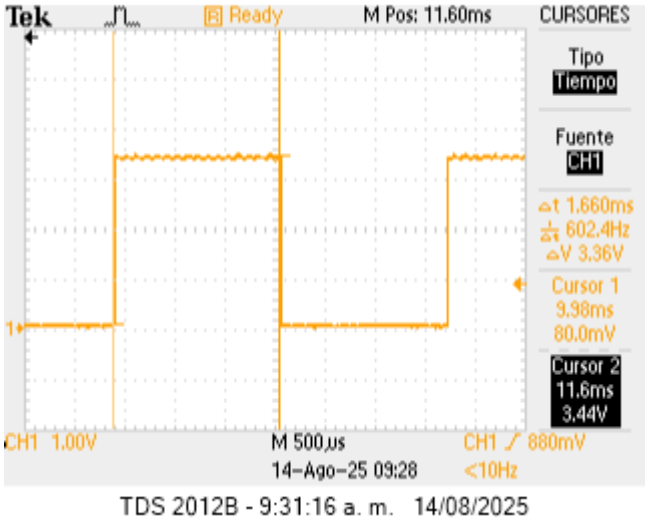

Ilustración 3 grafica del osciloscopio t bit caracter "W" 600 baudios.

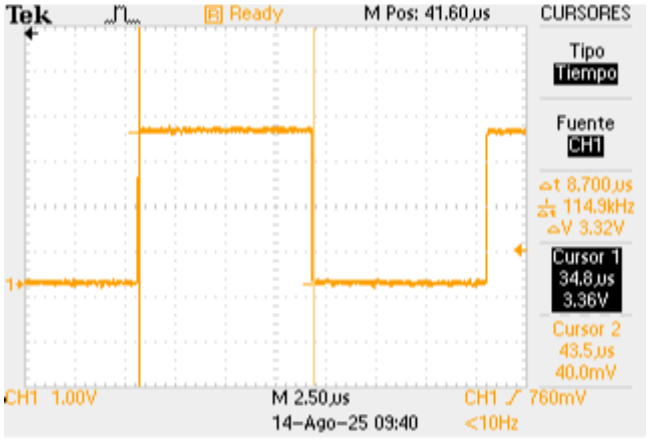

Ilustración 4 grafica del osciloscopio t bit caracter "W" 115200 baudios

Procedemos a calcular el tiempo de bit para cada tasa de baudios y tiempo de bit pata la trama competa, en este caso 11 bits, con la siguientes formulas:

$$tbit = \frac{1}{tasa\ de\ baudios}$$

$$ttrama = \frac{1}{tasa\ de\ baudios} * \#\ \text{bits trama}$$

La letra W en el código ASCII en binario es el 01010111,

Para este caso, como los 8 bits de datos tienen 5 en alto, número impar, si quisiéramos que la paridad fuera impar el bit de paridad seria 0 y la trama quedaría así:

1 bit start: 0
8 bit de datos: 11101010
1 bit paridad: 0
1 bit stop:1
Trama completa = 0- 11101010-0- 1 = 11 bits.

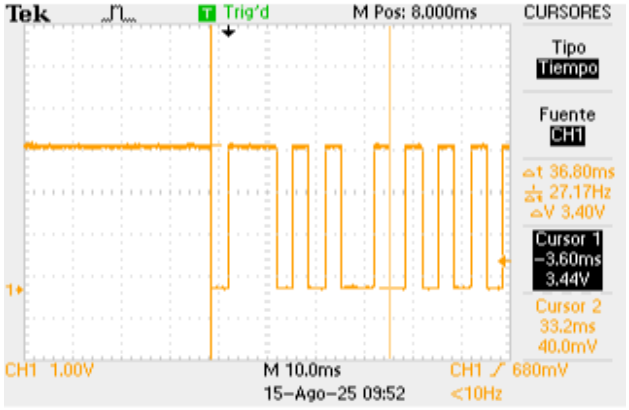

Ilustración 5 grafica del osciloscopio t trama caracter "W" 300 baudios

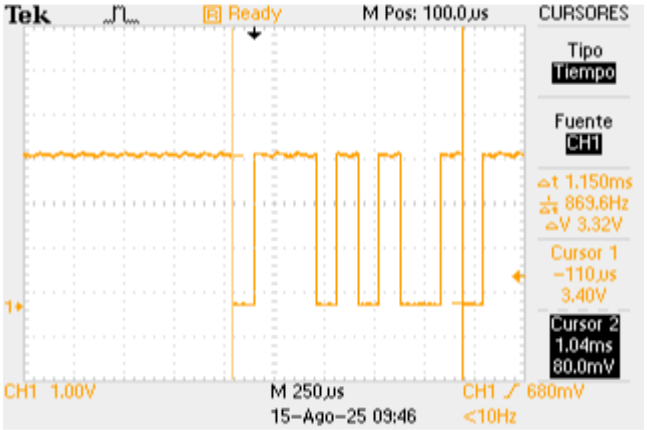

Ilustración 6 grafica del osciloscopio t trama caracter "W" 9600 baudios

De acuerdo a esto se procedió a calcular los valores teóricos tanto de tiempo de bit como tiempo de trama y se compararon con los medidos en el osciloscopio dando los siguientes resultados:

	1 BIT						
	BAUDIOS	T BIT TEORICO	T BIT EXP	Δt (t de bit)	% error	v max	v min
W	300	3,33 ms	3,30 ms	0,03 ms	0,90%	3,32 V	40 mV
	600	1,66 ms	1,66 ms	0,00 ms	0,00%	3,36 V	80 mV
	1200	833µs	832µs	1µs	0,12%	3,36 V	80 mV
	5000	200µs	200µs	0,00 µs	0,00%	3,36 V	80 mV
	9600	104 µs	104µs	0,00 µs	0,00%	3,4 V	0V
	115200	8,68µs	8,7 µs	0,02 µs	0,23%	3,32 V	40 mV
	TRAMA (11 BITS)						
	BAUDIOS	T TRA (teorico)	TTRA EXP	Δt (t trama)	% error	v max	v min
W	300	36,6 ms	36,80 ms	0,2 ms	0,54%	3,32 V	40 mV
	600	18,33 ms	18,34 ms	0,01 ms	0,05%	3,36 V	80 mV
	1200	9,16 ms	9,16 ms	0,00 ms	0,00%	3,36 V	80 mV
	5000	2,2 ms	2,2ms	0,00 ms	0,00%	3,36 V	80 mV
	9600	1,144 ms	1,150 ms	6 µs	0,52%	3,4 V	0V
	115200	95,48µs	95,48µs	0,00 ms	0,00%	3,32 V	40 mV

Ilustración 7 Tabla de datos, resultados caracter "W"

Los tiempos medidos experimentalmente para 300, 600, 1200, 5000, 9600 y 115200 baudios siguen la ley tb=1/baudiost con errores ≤ 0,90 %.

Los tiempos de la trama experimentales también son concordantes con la teoría que dice que al multiplicar el número de bits de la trama, para este caso 11 bits, por la duración de tiempo de bit, nos arroja este tiempo de trama completo, el cual presenta gran coincidencia con los valores teóricos, es un porcentaje de error muy bajo y el cual se podría deber a la manipulación del osciloscopio el cual, al generar una mayor amplitud visual de la señal, la precisión se perdía y no permitía realizar la medición más exacta.

Vmax se mantuvo entre 3,28–3,32 V, coherente con TTL 3,3 V de la Pico, La amplitud se mantuvo estable al variar la velocidad, lo que indica que el driver de salida no está limitado por la frecuencia de conmutación en este rango.

En 115200 baudios, el tiempo por bit es tan corto que pequeñas imprecisiones por cuantización del osciloscopio y del generador de baudios (redondeos de reloj) producen un error relativo cercano al 0,23 %, aunque el error absoluto sea de solo 20 ns.

4. Ahora procederemos a enviar 10 caracteres ASCII, para este caso (O, V, d, F, g, x, 3, =, *, &), los cuales se enviarán con paridad par, impar y sin paridad.

Iniciamos con el carácter “O”, 9600 baudios, en binario 01001111, para paridad par tendremos:

1 bit start: 0
8 bit de datos: LSB – 11110010 - MSB
1 bit paridad: 1
1 bit stop:1
Trama completa = 0- 11110010-1- 1 = 11 bits.

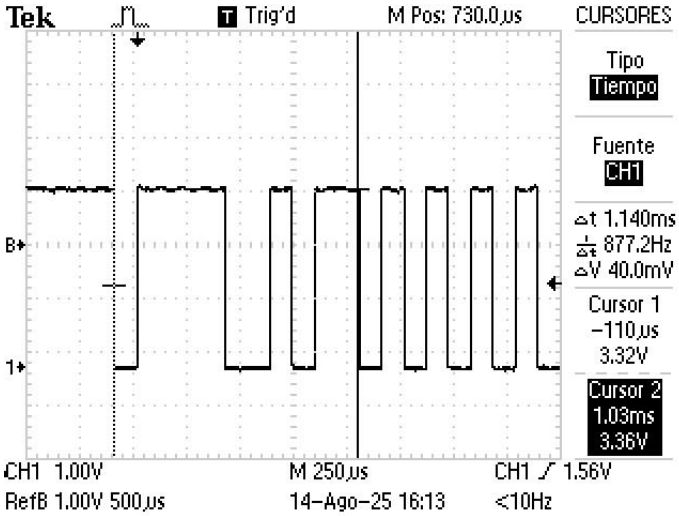

Ilustración 8 t trama paridad par caracter "O" 9600 baudios.

para paridad impar tendremos:

1 bit start: 0
8 bit de datos: LSB – 11110010 - MSB
1 bit paridad: 0
1 bit stop:1
Trama completa = 0-11110010-0- 1 = 11 bits.

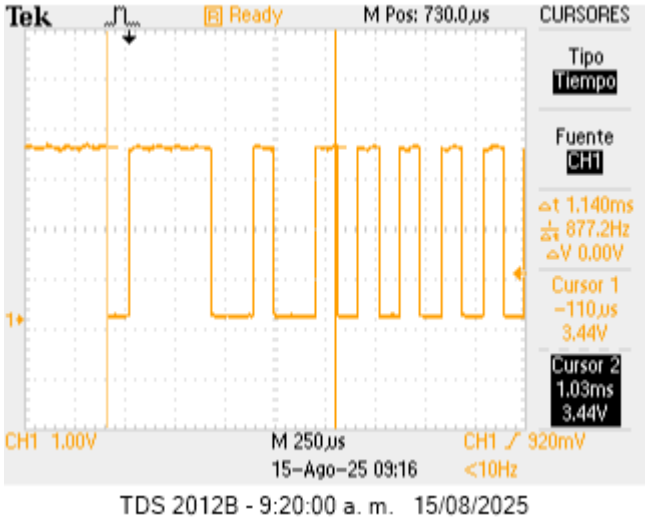

Ilustración 9 t trama paridad impar caracter "O" 9600 baudios.

Sin paridad tendremos:

1 bit start: 0
8 bit de datos: LSB – 11110010 - MSB
1 bit sin paridad: 0
1 bit stop:1
Trama completa = 0-11110010-1 = 11 bits.

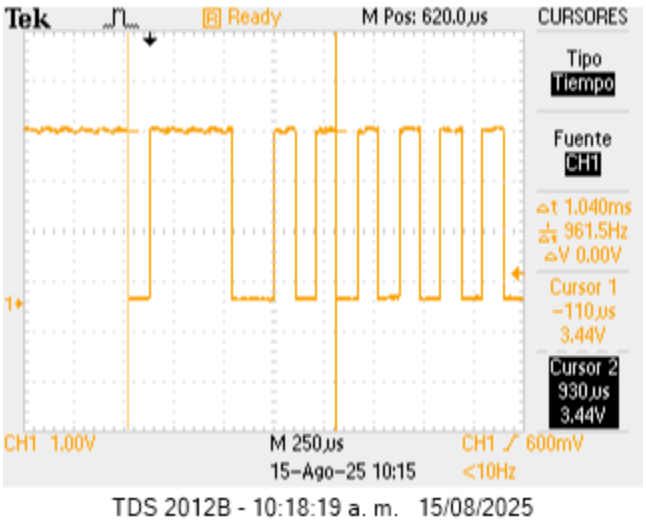

Ilustración 10 t trama sin paridad, caracter "O" 9600 baudios.

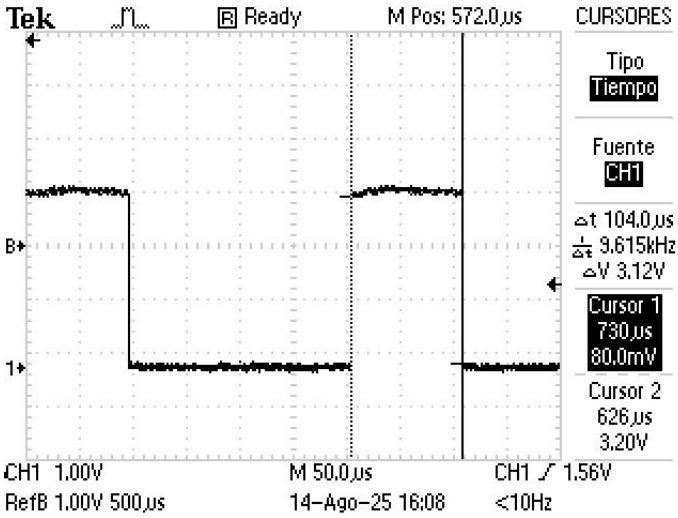

Ilustración 11 tiempo de bit, caracter "O" 9600 baudios.

De acuerdo a las graficas anteriormente mostradas podemos observar fácilmente en la trama un bit en alto demás cuando es paridad par ya que los bits de datos paras el carácter "O" tienen 5 bits en 1 por lo que se hace necesario que el de paridad sea 1, mientras que la grafica de paridad impar tiene el bit de paridad en 0, así mismo se observa que la grafica sin paridad la trama solo tiene 10 bits en total, así mismo la relación entre tiempo de bit y tiempo de trama es concordante.

tbit=1/9600= **104 µs**
ttrama (paridad par e impar) = 104 µs*11= **1,144 ms**
ttrama (sin paridad) = 104 µs*10= **1,04ms**

A continuación, se realizarán las respectivas graficas a mano.

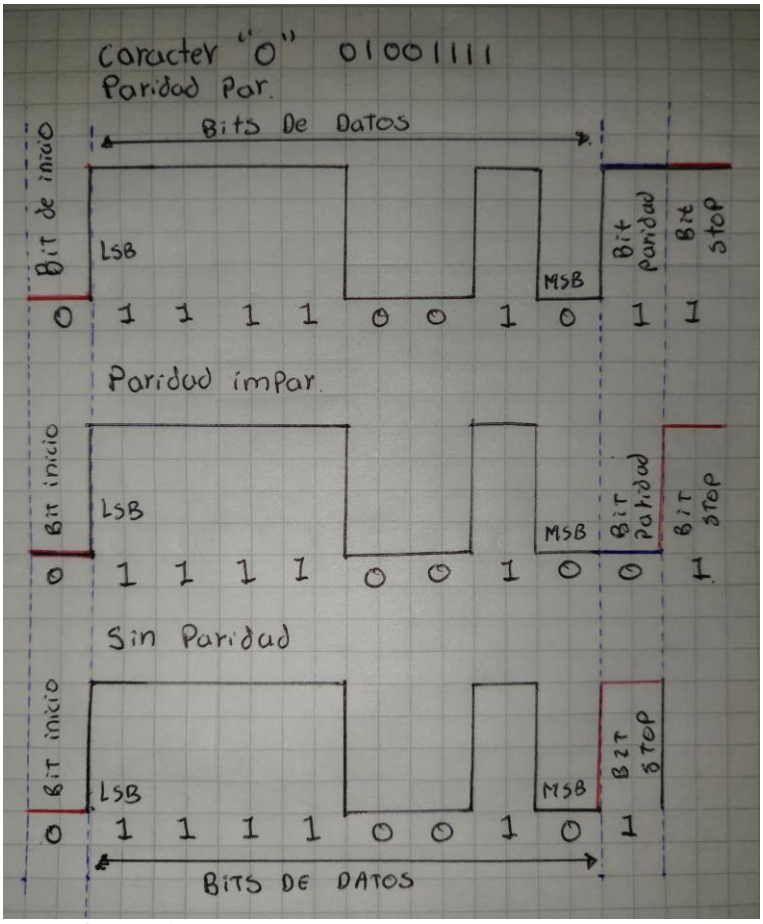

Ilustración 12 grafica a mano caracter "O" diferentes paridades.

Ahora seguimos con la letra **"V",** 300 baudios, en binario 01010110, para paridad par tendremos:
1 bit start: 0
8 bit de datos: LSB – 01101010 - MSB
1 bit paridad: 0
1 bit stop:1
Trama completa = 0- 01101010-0- 1 = 11 bits.

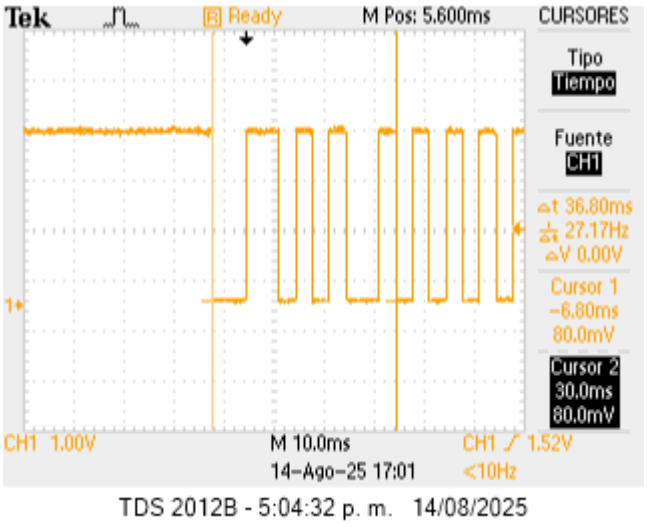

Ilustración 13 t trama paridad par caracter "V" 300 baudios.

"V" para paridad impar tendremos:
1 bit start: 0
8 bit de datos: LSB – 01101010 - MSB
1 bit paridad: 1
1 bit stop:1
Trama completa = 0- 01101010-1- 1 = 11 bits.

También medimos amplitud para verificar el voltaje mínimo y voltaje máximo.

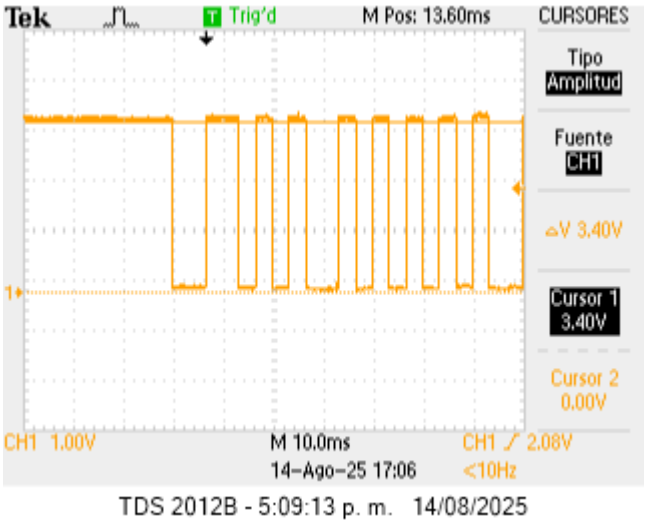

Ilustración 14 grafica amplitud "V" 300 baudios.

"V" sin paridad tendremos:
1 bit start: 0
8 bit de datos: LSB – 01101010 - MSB
1 bit paridad: no
1 bit stop:1
Trama completa = 0- 01101010- 1 = 10 bits.

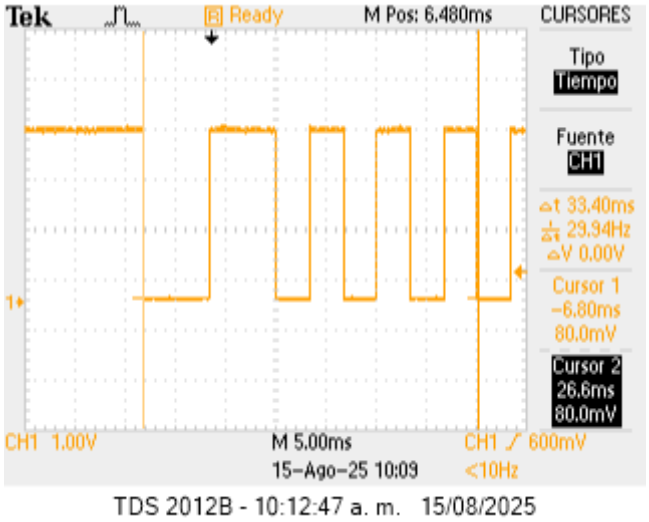

Ilustración 15 t trama sin paridad caracter "V" 300 baudios

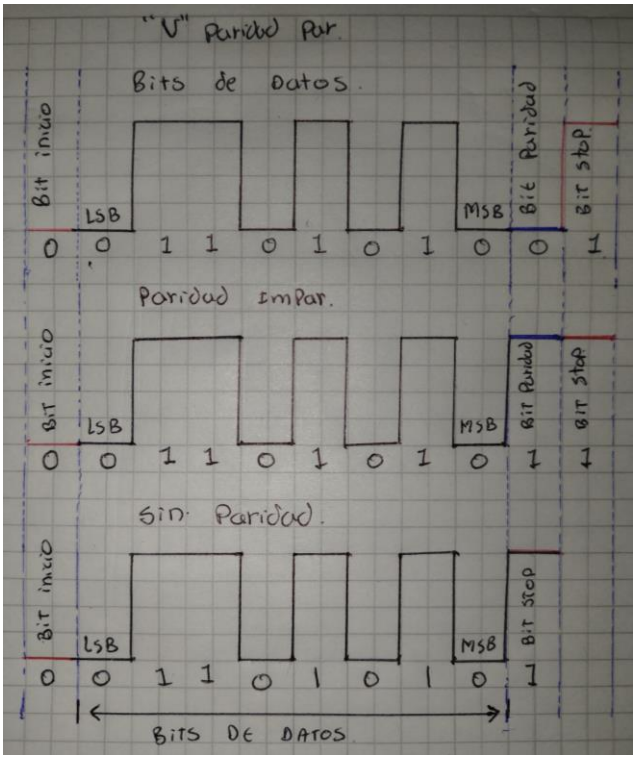

Ilustración 16 grafica a mano carácter "V" diferentes paridades.

De acuerdo a las gráficas anteriormente mostradas podemos observar que para el carácter "V" los bits de datos tienen cuatro bits en 1 por lo que para paridad par se hace necesario el bit de paridad en 0, se observa lo contario para lograr la paridad impar, se debe tener el bit de paridad en 1, así mismo se observa que la gráfica sin paridad la trama solo tiene 10 bits en total, así mismo la relación entre tiempo de bit y tiempo de trama es concordante.

tbit=1/300= **3,33 ms**
ttrama (paridad par e impar) = **3,33 ms** *11= **36,66 ms**
ttrama (sin paridad) = **3,33 ms** ***10**= **33,3 ms**

Ahora seguimos con la letra **"d",** 600 baudios

Carácter	Paridad	Binario	Bit inicio	Bits Datos LSB MSB	Bit Paridad	Bit Stop
d	Par	01100100	0	00100110	1	1
	Impar	01100100	0	00100110	0	1
	None	01100100	0	00100110	None	1

tbit=1/600= **1,66 ms**
ttrama (paridad par e impar) = **1,66 ms** *11= **18,33 ms**
ttrama (sin paridad) = **1,66 ms** ***10**= **16,66 ms**

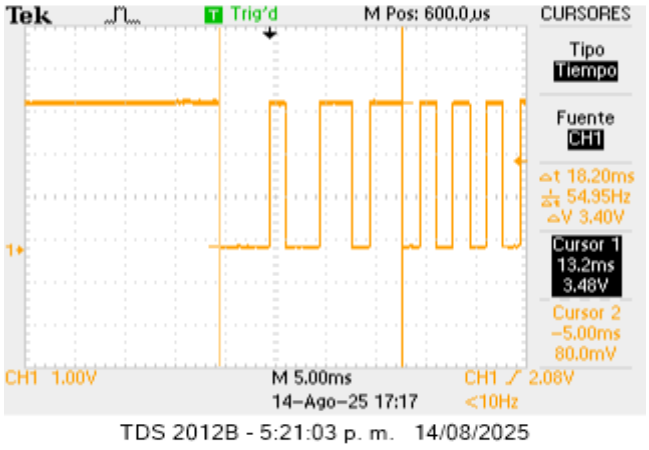

Ilustración 17 t trama paridad par caracter "d" 600 baudios

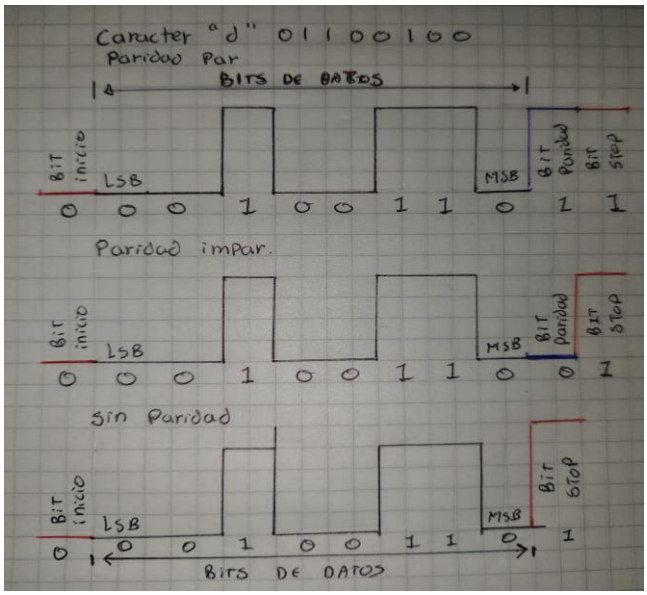

Ilustración 18 grafica a mano carácter "d" diferentes paridades.

De igual manera se calcularon las tramas para los demás caracteres de acuerdo a las paridades:

Carácter	Paridad	Binario	Bit inicio	Bits Datos LSB MSB	Bit Paridad	Bit Stop
F	Par	01000110	0	01100010	1	1
	Impar	01000110	0	01100010	0	1
	None	01000110	0	01100010	None	1
g	Par	01100111	0	11100110	1	1
	Impar	01100111	0	11100110	0	1
	None	01100111	0	11100110	None	1
x	Par	01111000	0	00011110	0	1
	Impar	01111000	0	00011110	1	1
	None	01111000	0	00011110	None	1
3	Par	00110011	0	11001100	0	1
	Impar	00110011	0	11001100	1	1
	None	00110011	0	11001100	None	1
=	Par	00111101	0	10111100	1	1
	Impar	00111101	0	10111100	0	1
	None	00111101	0	10111100	None	1
*	Par	00101010	0	01010100	1	1
	Impar	00101010	0	01010100	0	1
	None	00101010	0	01010100	None	1
&	Par	00100110	0	01100100	1	1
	Impar	00100110	0	01100100	0	1
	None	00100110	0	01100100	None	1

Tabla 1 tramas de los caracteres de acuerdo a las paridades.

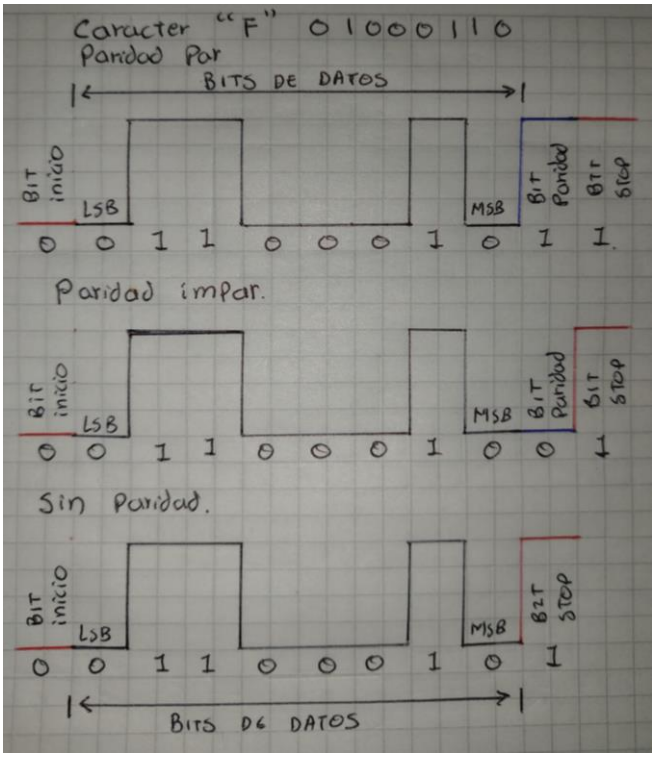

Ilustración 19 grafica a mano carácter "F" diferentes paridades.

también procedemos a realizar las tablas de resultados obtenidos de acuerdo a las respectivas mediciones en el osciloscopio de todos los caracteres, tomando tiempo de bit y tiempo de trama y amplitud en cada caso, y los respectivos cálculos teóricos, estos valores se observarán en las siguientes tablas:

TABLA PARA LETRAS CON PARIDAD PAR							
CARÁCTER	BAUDIOS	TBIT TEORICO	TBIT EXP	TTRAMA (teorico)	TTRAMA (exp)	Δt (tiempo de bit)	% error
V	300	3,33 ms	3,32 ms	36,66 ms	36,8 ms	ico-bit exp)(	0,30%
d	600	1,66 ms	1,68 ms	18,33 ms	18,2 ms	0,02	1%
F	1200	833, 3 µs	830 µs	9,16 ms	9,20 ms	3,3	0,30%
g	5000	200 µs	200 µs	2,2 ms	2, 2ms	0	0%
x	8000	125 µs	126µs	1,37 ms	1,38 ms	1	0,80%
O	9600	104 µs	104µs	1,144 ms	1,144 ms	0	0%
3	10000	100 µs	102 µs	1,1 ms	1,2 ms	2	2%
=	15000	66,6 µs	66,6 µs	733,3µs	733,8µs	0	0%
*	20000	50 µs	50,2 µs	550µs	558µs	0,2	4%
&	30000	33,3 µs	33,4 µs	366 µs	364 µs	0,1	0,30%

Tabla 2 tiempo de bit y trama teóricos y experimentales-paridad par.

TABLA CARACTRES - CON PARIDAD IMPAR							
CARÁCTER	BAUDIOS	TBIT TEORICO	TBIT EXP	TTRAMA (teorico)	TTRAMA (exp)	Δt (tiempo de bit)	% error
V	300	3,33 ms	3,40 ms	36,66 ms	33,3ms	0,07 ms	2,10%
d	600	1,66 ms	1,64 ms	18,33 ms	18,4 ms	0,02 ms	1,20%
F	1200	833, 3 µs	833,4µs	9,16 ms	9,16 ms	0,1 µs	0,01%
g	5000	200 µs	200µs	2,2 ms	2,2ms	0,0 µs	0,00%
x	8000	125 µs	126µs	1,37 ms	1,38ms	1,0 µs	0,80%
O	96000	104 µs	104µs	1,144 ms	1,140 ms	0,0 µs	0,00%
3	10000	100 µs	100µs	1,1 ms	1,12 ms	0,0 µs	0,00%
=	15000	66,6 µs	66,8µs	733,3µs	734 µs	0,2 µs	0,30%
*	20000	50 µs	50µs	550µs	552µs	0,0 µs	0,00%
&	30000	33,3 µs	33,4µs	366 µs	368µs	0,1 µs	0,30%

Tabla 3 tiempo de bit y trama teóricos y experimentales-paridad impar.

TABLA CARACTERES - SIN PARIDAD							
CARÁCTER	BAUDIOS	TBIT TEORICO	TBIT EXP	TTRAMA (teorico)	TTRAMA (expl)	Δt (tiempo trama)	% error
V	300	3,33 ms	3,32 ms	33,3 ms	33,4ms	0,10 ms	0,30%
d	600	1,66 ms	1,68 ms	16,6 ms	16,6 ms	0,00 ms	0,00%
F	1200	833, 3 µs	833,4µs	8,33 ms	8,32 ms	0,01 ms	0,12%
g	5000	200 µs	200µs	2 ms	2 ms	0,00 ms	0,00%
x	8000	125 µs	126µs	1,25 ms	1,24 ms	0,01 ms	0,80%
O	96000	104 µs	104µs	1,04 ms	1,04ms	0,00 ms	0,00%
3	10000	100 µs	100µs	1 ms	1 ms	0,00 ms	0,00%
=	15000	66,6 µs	66,8µs	666 µs	668µs	2 µs	0,30%
*	20000	50 µs	50µs	500 µs	500µs	0 µs	0,00%
&	30000	33,3 µs	33,2µs	333 µs	332µs	1 µs	0,30%

Tabla 4 tiempo de bit y trama teóricos y experimentales- Sin paridad.

En las tres tablas el tiempo de bit medido coincide con el teórico con errores muy pequeños (0–0,3%), Esto confirma que el reloj de la UART y la medición con el osciloscopio están bien ajustados, los errores entre teoría y práctica en tiempo de trama también son muy bajos (0–0,5%).

El Contenido" del carácter no cambia el tiempo de la trama, solo cambia cuando se quiere sin paridad (None) o con el cambio de los baudios, entre la tabla Par e Impar los tiempos son prácticamente iguales (ambas tienen 11 bits).

En bajo baudio, el bit dura mucho y cualquier paso de los cursores se nota, en alto baudio, el bit es muy corto y aparece el efecto del redondeo del reloj del micro y del osciloscopio.

5. Procedemos a modificar el programa para enviar una trama de 60 caracteres enviados a 600 baudios, paridad par y 8 bis de datos. Capture la señal resultante en el osciloscopio y mida el tiempo total de la trama generada.

En este caso se envió la frase: "Las telecomunicaciones conectan al mundo en un instante!! " con el siguiente código:

```
import machine
import utime
from machine import Pin, UART
led = machine.Pin("LED", machine.Pin.OUT)

# Configuración UART (8N1: 600 baudios, 8 bits, sin paridad, 1 stop bit)
uart = UART(0, baudrate=600, bits=8, parity=1, stop=1, tx=Pin(0), rx=Pin(1))

# Frase exacta de 60 caracteres (sin relleno adicional)

frase = "Las telecomunicaciones conectan al mundo en un instante!! "

while True:
led.on()
uart.write(frase)
led.off()
utime.sleep(2)
```

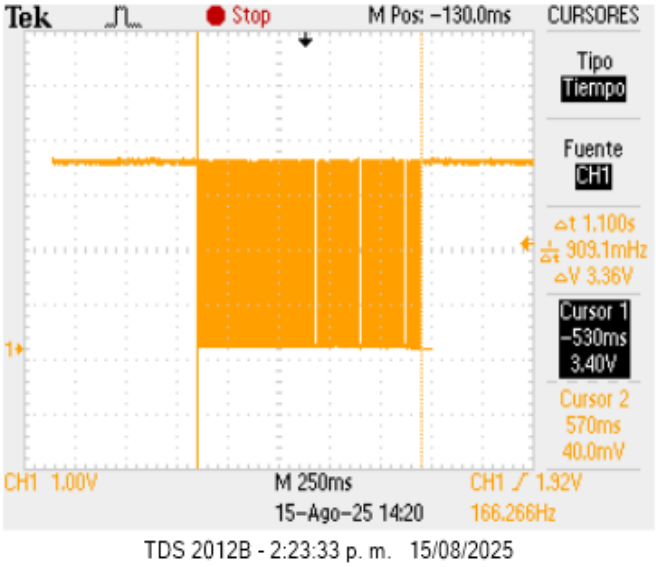

Ilustración 20 tiempo de trama frase con 60 caracteres, 600 baudios.

tbit=1/600= **1,66 ms**
tiempocaracter = **1,66 ms *11**= **18.33 ms**
tiempotrama = **18.33 ms** ***60**= **1,1 s**

De acuerdo a estos cálculos se puede apreciar que la grafica mostrada de la medición en el osciloscopio es concordante al medir el tiempo de trama y comparar con los datos teóricos calculados, no hay error.

Ahora procedemos a realizar el mismo ejercicio, pero esta vez sin paridad, para esto en el código en la parte de paridad se cambia; parity=None.

Los resultados obtenidos para esta cadena de caracteres, pero sin paridad, son los siguientes:

tbit=1/600= **1,66 ms**

tiempocaracter = **1,66 ms *10**= **16,66 ms**
tiempotramafrase = **16,66 ms** ***60**= **1 s**

Este resultado es exactamente igual a la medición echa en el osciloscopio donde se observa claramente el tiempo total de la trama generada es igual a 1s.

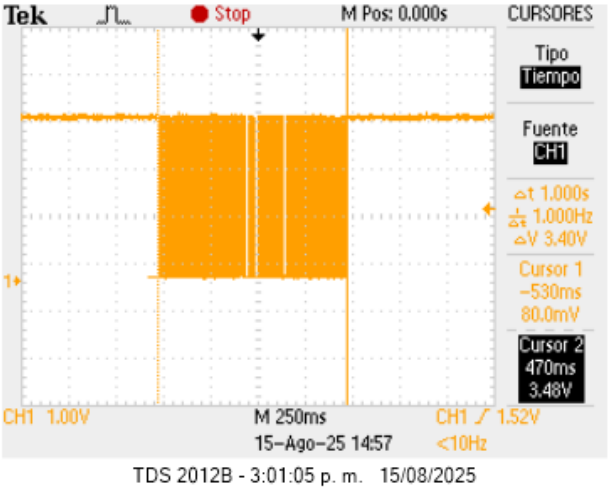

Ilustración 21 tiempo de trama frase con 60 caracteres, 600 baudios, sin paridad.

Ahora se procede a Capturar la trama “UMNG LIDER EN INGENIERIA EN TELECOMUNCACIONES” enviada a 57600 baudios, 7 bits de datos, paridad par y dos bits de parada, para este ejercicio los datos calculados son los siguientes:

1 bit start, 7 bit de datos, 1-bit paridad, 2 bits stop.
Trama completa = 11 bits.
Caracteres de la frase=45

tbit=1/57600= **17,3 µs**
tiempocaracter = **17,3 µs *11= 190,9 µs**
tiempotramafrase = **190,9 µs *45= 8,59 ms**

En el código se procede a Cambiar la velocidad de baudios de 600 a 57600, Cambiar los bits de datos de 8 a 7, Cambiar la paridad de (ninguna) a 1 (par), Cambiar los bits de parada de 1 a 2, Actualizar el mensaje a "UMNG LIDER EN INGENIERIA EN TELECOMUNCACIONES", Ajustar la función de conversión ASCII para manejar 7 bits, generando el siguiente código con el que se procede a tomar las medidas en el osciloscopio:

```
import machine
import utime
from machine import Pin, UART
led = machine.Pin("LED", machine.Pin.OUT)

# Configuración UART (7E2: 57600 baudios, 7 bits, paridad PAR, 2 stop bits)
uart = UART(0, baudrate=57600, bits=7, parity=1, stop=2, tx=Pin(0), rx=Pin(1))

frase = "UMNG LIDER EN INGENIERIA EN TELECOMUNCACIONES"

while True:
led.on()
uart.write(frase)       # Envía la frase completa
led.off()               # Apaga LED
utime.sleep(1)          # Espera 1 segundo
```

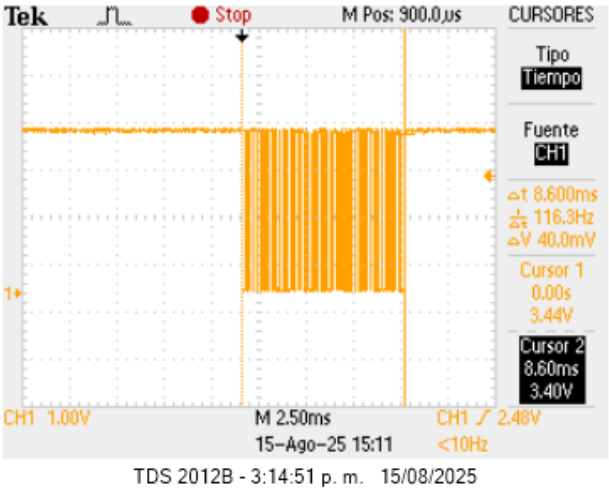

Ilustración 22 tiempo de trama frase con 45 caracteres, 57600 baudios.

La captura de la frase en a 57 600 baudios arrojó Texp 8,60 ms, frente a Tteo 8,594 ms. La diferencia de 6,25 µs (0,073 %) confirma la concordancia con el modelo T=N/baudios, con N=11 y valida la correcta generación de start, datos (LSB→MSB), paridad y dos stop. La amplitud estable (3,3 V) indica que la salida de TX opera bien a esta velocidad.

6. Reto de programación.

Interconectamos dos dispositivos Raspberry Pi Pico cruzando las líneas TX y Rx entre los dispositivos, se diseñaron los siguientes programas para trasmisión Tx y recepción Rx.

tommy 1.py *

```python
# TX.py — LED externo en GPIO16 — versión mínima corregida
from machine import UART, Pin
import time

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))  # UART0 (GP0 TX, GP1 RX)
led = Pin(16, Pin.OUT)  # LED en GPIO16
led.value(0)

PING = b'A'   # Lo que enviamos
ACK  = b'B'   # Lo que esperamos del RX

while True:
    # 1) Enviar 'A' cada 2 s
    uart.write(PING)
    print("Enviado: 'A'")
    time.sleep(2)

    # 2) Revisar si llegó ACK
    if uart.any():
        resp = uart.read(1)  # lee 1 byte
        # Drenar bytes extra si hubiera (ruido, ecos, etc.)
        while uart.any():
            uart.read(1)
        print("Recibido:", resp)

        # 3) Parpadeo SOLO si es 'B'
        if resp == ACK:
            # Parpadeo 3 s (3 ciclos de 0.5 on + 0.5 off)
            for _ in range(3):
                led.value(1); time.sleep(0.5)
                led.value(0); time.sleep(0.5)
```

Ilustración 23 programa para trasmisión Tx, Tonny-Python.

```python
# RX.py — versión mínima corregida (LED en GP16, bloqueante con sleep)
from machine import UART, Pin
import time

uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))
led = Pin(16, Pin.OUT)
led.value(0)

counter = 0

with open('recibidos.txt', 'a') as f:
    while True:
        if uart.any():
            dato = uart.read(1)         # lee 1 byte
            # Drenar posibles bytes residuales
            while uart.any():
                uart.read(1)

            if dato == b'A':            # SOLO si recibió 'A'
                print("Recibido: 'A'")
                led.value(1)            # Enciende LED
                uart.write(b'B')        # Responde de inmediato con 'B'
                print("Enviado: 'B'")
                counter += 1
                f.write(f"{counter}\n") # 1 línea por recepción
                f.flush()

                time.sleep(5)           # Mantiene LED 5 s (bloquea)
                led.value(0)            # Apaga LED

        time.sleep(0.01)  # pequeño respiro
```

Ilustración 24 programa para recepción Rx, Tonny-Python.

Ahora se creo un programa para la conexión full-Duplex.

```python
# full_duplex.py — MicroPython — LED externo en GPIO16 (cargar en ambos Picos)
from machine import UART, Pin
import utime

# ---------- UART ----------
BAUD = 9600
uart = UART(0, baudrate=BAUD, bits=8, parity=None, stop=1, tx=Pin(0), rx=Pin(1), timeout=0)

# ---------- LED externo ----------
led = Pin(16, Pin.OUT)
led.off()

PING = b'P'
ACK  = b'Q'

# Tiempos
PERIODO_ENVIO_MS = 2000
LED_RX_MS = 5000
LED_ACK_BLINK_MS = 3000
PERIODO_BLINK_MS = 200

ultimo_ping = utime.ticks_ms() - PERIODO_ENVIO_MS

# Estados LED
apagar_led_en = None              # LED fijo tras 'P' recibido
blink_hasta = None                # parpadeo tras 'Q' recibido
proximo_toggle = 0
estado_led = 0

# Contador de recepciones (también guarda archivo)
contador = 0
try:
    with open("recibidos.txt", "r") as f:
        for linea in f:
            pass
        ultima = linea.strip() if 'linea' in locals() else ""
        if ultima.isdigit():
            contador = int(ultima)
```

Ilustración 25 programa para full Duplex, Tonny-Python.

En half-duplex, la idea clave es la no coincidencia temporal de los pulsos, primero transmite A, luego responde B, sin superposición.

En full-duplex, lo fundamental es la superposición de tramas en CH1 y CH2, demostrando que la UART permite transmisión simultánea en canales independientes.

Utilizamos el osciloscopio para ver el comportamiento de los programas dando los siguientes resultados:

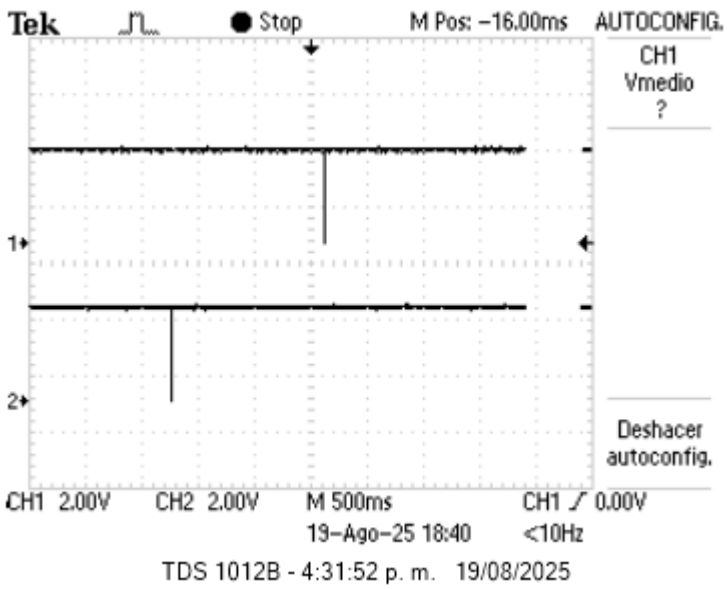

Ilustración 26 tramas observadas en half-Duplex

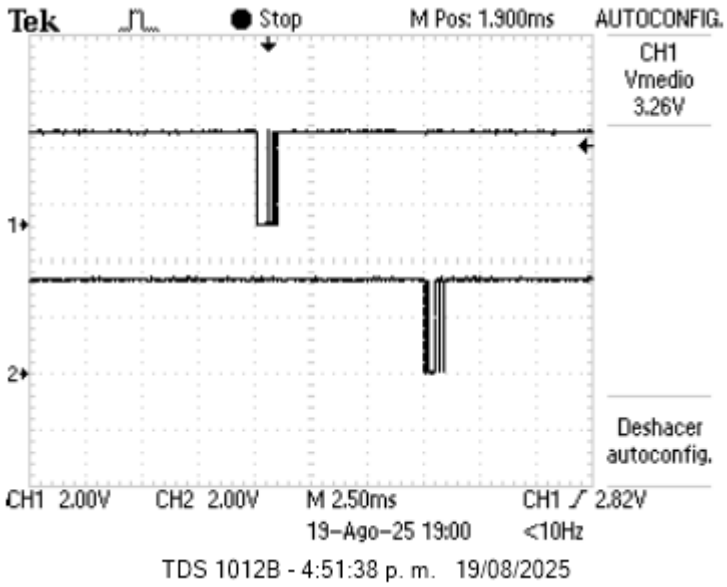

Ilustración 27 tramas observadas en full-Duplex

La primera imagen confirma el comportamiento half-duplex porque los pulsos aparecen en momentos diferentes en cada canal, por la escala no se puede ver la estructura de bits, solo la evidencia de que los envíos ocurren alternados, Se observa que primero aparece un pulso en CH1 (TX de A) y, posteriormente, otro pulso en CH2 (TX de B), en half-duplex la comunicación ocurre en turnos, un dispositivo transmite y el otro espera para responder.

En esta imagen de full duplex se distinguen dos tramas solapadas, mientras uno está mandando, el otro también comienza a transmitir casi de inmediato, demuestra transmisión simultánea en los dos canales, lo que es característico de la comunicación full-duplex.

7. Comparación entre los parámetros técnicos de las Raspberries Pi Pico W y Pi pico 2W.

Las Raspberry Pi de la familia Pico son microcontroladores diseñados para ofrecer alta eficiencia energética, bajo costo y versatilidad en aplicaciones embebidas. El modelo Pico W, lanzado en 2022, introdujo la conectividad inalámbrica Wi-Fi al ecosistema. En 2024, con la llegada del Pico 2W, se incorporó una nueva arquitectura de procesador y mejoras en memoria y capacidades de comunicación, manteniendo la filosofía de bajo costo, pero con un salto cualitativo en desempeño y escalabilidad.

Parámetro	Raspberry Pi Pico W	Raspberry Pi Pico 2W	Analisis de diferencia
Micro-controlador	RP2040 (ARM Cortex-M0+ dual-core, 133 MHz máx.)	RP2350 (ARM Cortex-M33 dual-core, 150 MHz máx. + vector unit)	El RP2040 (Pico W) es un chip de propósito general con gran adopción en educación y prototipado. El RP2350 (Pico 2W) introduce Cortex-M33, con soporte de seguridad avanzada (TrustZone), aceleración de operaciones aritméticas (FPU y SIMD), y mejor manejo de interrupciones. Nuevo núcleo con arquitectura más moderna, soporte de seguridad avanzada y mejor eficiencia.
Memoria RAM	264 KB SRAM	520 KB SRAM	Pico 2W casi duplica la RAM, habilitando aplicaciones más complejas y buffers más grandes.
Memoria Flash	2 MB QSPI	4 MB QSPI	La Flash de 4 MB en el 2W permite almacenamiento de firmware más grande, soporte para OTA, aumenta la capacidad de almacenamiento de firmware y datos persistentes.
Conectividad inalámbrica	Wi-Fi 2.4 GHz (CYW43439, IEEE 802.11n)	Wi-Fi 2.4 GHz y BLE 5.2 (Infineon CYW43439 actualizado)	El 2W incorpora Bluetooth, ampliando su aplicabilidad en IoT y dispositivos móviles.
GPIO	26 pines GPIO (3,3 V lógicos)	26 pines GPIO (3,3 V lógicos)	Igual número, pero con mejoras en periféricos internos y controladores.
Periféricos	2 × UART, 2 × SPI, 2 × I²C, 16 × PWM, 3 × ADC (12 bits)	4 × UART, 4 × SPI, 4 × I²C, PWM extendido, ADC mejorado (12 bits, mayor precisión)	Mientras que el Pico W tiene 2 buses UART/SPI/I²C, el Pico 2W **duplica la disponibilidad** (4 buses cada uno), lo cual permite gestionar más dispositivos sin necesidad de multiplexación.
Consumo energético	~0,6 W típico en operación	~0,7–0,8 W típico (por núcleos más potentes y BLE activo)	El 2W consume más, pero con mejor relación rendimiento/watt.
Seguridad	Sin TrustZone ni acelerador es criptográfic os	ARM TrustZone, soporte para ejecución segura, aceleración de criptografía	integra **TrustZone**, aislamiento de procesos y aceleración criptográfica, lo que lo convierte en un candidato sólido para aplicaciones IoT con **ciberseguridad embebida**: autenticación, cifrado y arranque seguro.
Precio estimado (2025)	~6 USD	~7–8 USD	Se mantiene bajo costo, con un incremento marginal frente a las mejoras ofrecidas.

Tabla 5 parámetros técnicos de las Raspberries Pi Pico W y Pi pico 2W

Se crea un repositorio en GitHub donde se suben las demás evidencias del desarrollo del laboratorio al cual se podrá acceder en este enlace:

https://github.com/fernandoriano/LABORATORIO3--COMUNICACION-DIGITAL.git

CONCLUSIÓNES

La estructura de la trama UART, compuesta por el bit de inicio, los bits de datos (transmitidos del LSB al MSB), el bit de paridad opcional y los bits de parada, se visualizó y midió con claridad en el osciloscopio. Se pudo comprobar empíricamente cómo la adición del bit de paridad y el uso de más de un bit de parada incrementan de manera lineal y predecible la duración total de la trama, Para un mismo baudio, las tramas con paridad duran lo de una ranura de bit más que las sin paridad, añadiendo overhead (sobrecosto) al tiempo de transmisión por carácter. El contenido específico del dato transmitido no altera este tiempo, que depende únicamente de la configuración de bits y la velocidad.

La integridad de la señal se mantuvo excelente en todas las velocidades, con niveles de voltaje TTL estables en 3.3V para el estado alto y muy cercanos a 0V para el estado bajo, sin distorsión apreciable o ruido que comprometiera la legibilidad de los bits, incluso a la máxima velocidad de 115200 baudios, el análisis con el osciloscopio no solo permitió validar los tiempos y la forma de las tramas, sino también verificar el perfecto acople entre la lógica programada y el comportamiento eléctrico del sistema, cerrando el ciclo entre el software, el protocolo de comunicación y el hardware.

En todos los casos, los errores se mantuvieron por debajo de 0,5 % (típicamente 0,1–0,3 %), lo que corrobora la precisión del ajuste de baudios y la coherencia entre teoría y práctica.

III. REFERENCIAS

[1] Raspberry Pi Ltd. (2023). *RP2040 Datasheet*. [En línea]. Disponible en: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[2] Raspberry Pi Ltd. (2024). *Raspberry Pi Pico W and Pico 2W Product Brief*. [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/

[3] MicroPython. (2024). *machine — funciones relacionadas con hardware*. Documentación oficial. [En línea]. Disponible en: https://docs.micropython.org/en/latest/library/machine.UART.html

[4] Texas Instruments. (2020). *UART Interface Basics*. Application Report. [En línea]. Disponible en: https://www.ti.com/lit/an/slla037/slla037.pdf

[5] Horowitz, P. y Hill, W. (2015). *The Art of Electronics*. 3ª ed. Cambridge University Press.

[6] Peatman, J. B. (1998). *Embedded Design with the PIC Microcontroller*. Prentice Hall.

[7] MathWorks, "Transformada Rápida de Fourier - FFT," MATLAB Documentation, 2024. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/fft.html

[8] MathWorks, "Transformada Inversa Rápida de Fourier - IFFT," MATLAB Documentation, 2024. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/ifft.html

[9] M. Oppenheim y A. Willsky. *Señales y sistemas*. 2ª ed., Prentice Hall, 1997.

[10] A. V. Oppenheim, R. W. Schafer, y J. R. Buck. *Discrete-Time Signal Processing*. 3ª ed., Prentice Hall, 2009.

[11] Lathi, B. P. *Linear Systems and Signals*. 2ª ed., Oxford University Press, 2005.

[12] OpenAI. (2025). Respuesta generada por ChatGPT sobre simulación de señales y espectros en MATLAB. Comunicación en línea, 6 de agosto de 2025.

[13] J. G. Proakis y D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. 4ª ed., Pearson, 2007.

[14] T. K. Rawat. *Signal Processing and Linear Systems*. Oxford University Press, 2015.

[15] Universidad Nacional de Colombia. (2022). *Análisis espectral de señales periódicas usando series de Fourier y transformadas*. Curso de Señales y Sistemas. [En línea]. Disponible en: https://www.virtual.unal.edu.co/cursos/ingenieria/4025051/pdf