Exploración de la comunicación RS-232

Luis Carlos Leal Gamboa
est.luis.cleal@unimilitar.edu.co
Docente: José de Jesús Rúgeles

***Resumen*— En este laboratorio se utilizó la Raspberry Pi Pico 2W programada en MicroPython para implementar una comunicación serial según el estándar RS-232. Se analizaron aspectos de la capa física, como los niveles de voltaje y la sincronización entre transmisor y receptor. Se exploró la estructura de la trama serial, identificando sus partes: bit de inicio, bits de datos, paridad (cuando se usa) y bits de parada. Mediante mediciones prácticas, se verificó la relación inversa entre la tasa de baudios y el tiempo de cada bit. El desarrollo de scripts en MicroPython permitió gestionar la UART para enviar y recibir datos, reforzando el entendimiento de los protocolos de comunicación asíncrona y fortaleciendo habilidades en programación de sistemas embebidos.**

***Abstract*— In this lab, the Raspberry Pi Pico 2W was programmed using MicroPython to implement a serial communication system based on the RS-232 standard. Key aspects of the physical layer were analyzed, including voltage levels and signal synchronization. The structure of the serial frame was studied, identifying its components: start bit, data bits, parity bit (when used), and stop bit(s). Practical tests were conducted to verify the inverse relationship between the baud rate and the bit duration. Writing and running MicroPython code enabled data transmission and reception via UART, enhancing understanding of asynchronous communication protocols and strengthening embedded systems programming skills.**

I. Introducción

La comunicación serial es una de las formas más comunes y útiles de intercambiar datos entre dispositivos electrónicos, y aunque es una tecnología que lleva mucho tiempo, sigue siendo muy relevante en proyectos de microcontroladores. En este laboratorio usamos el estándar RS-232, uno de los más clásicos, para entender cómo funciona la transmisión de datos de forma práctica. Para eso, trabajamos con la Raspberry Pi Pico 2W, que es una tarjeta muy accesible y que se puede programar en MicroPython, lo que la hace ideal para aprender.

Aprendimos cómo se configura un puerto UART, cómo se relaciona la tasa de baudios con el tiempo que dura cada bit y cómo está estructurada una trama serial: con su bit de inicio, los bits de datos, el de paridad (si se usa) y los bits de parada. Lo más interesante fue poder ver las señales en el osciloscopio, medir tiempos reales y compararlos con los valores teóricos. Nos permitió confirmar que, aunque en la teoría todo cuadra, en la práctica hay pequeñas variaciones que también hay que tener en cuenta. Fue una buena forma de unir conceptos teóricos con experimentación real.

II. Configuración inicial

A partir de las indicaciones del fabricante, se realiza la debida configuración de la Rasperry pi pico 2W, esto mediante el proceso de descarga del archivo de configuración de esta, para ello se redirige a la pagina oficial de la compañía, y se realiza la descarga del archivo correspondiente.

Ilustración 1 Archivo de configuración para la Rasperry.

A partir del archivo obtenido, se conecta el dispositivo al computador, con el fin de compartir el archivo descargado a este, para esto, se selecciona el archivo y se mueve al dispositivo, que de fábrica, se muestra como un periférico del computador, mas no como un microcontrolador como se observa a continuación.

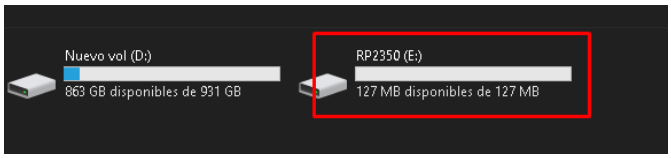

Ilustración 2 RASPERRY PI PICO 2W PREVIAMENTE A SU CONFIGURACIÓN.

A partir de esto, la rasperry pasa a presentarse como el microcontrolador, esto se evidencia en la aplicación de Thonny, como se observa a continuación.

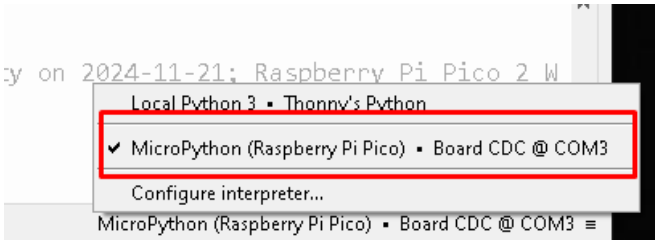

Ilustración 3 reconocimiento de la rasperry en la aplicación de Thonny.

A partir de esto, es necesario evidenciar el pinout del dispositivo en cuestión, esto con el fin de observar la función de cada uno de los pines, en este caso, como se observa a continuación, se hará uso del GP 0, 1, 2 y el del que viene integrado en esta, como se observa a continuación.

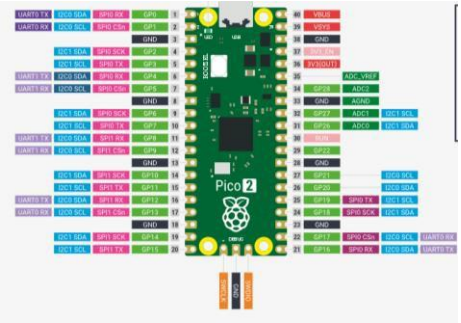

Ilustración 4 Pinout de la Rasperry pi pico 2W

A partir de lo previamente establecido, se hace uso del siguiente código, con el fin de observar el adecuado comportamiento del micro.

```
import machine
import utime

from machine import Pin, UART

led = machine.Pin("LED", machine.Pin.OUT)
uart = UART(0, baudrate=9600, bits=8, parity=None, tx=Pin(0), rx=Pin(1))

while True:
led.on()
uart.write("O")
utime.sleep(1)
led.off()
utime.sleep(1)
```

Este codigo cumple la función de prender el led integrado cada vez que el caracter “U” es enviado, este en código ASCII equivale a 85 en hexadecimal y en binario 1010101, con el fin de comprobar que la teórica con lo observado, se mide la salida en el pin GP0 con la sonda del osciloscopio, lo cual se observa a continuación.

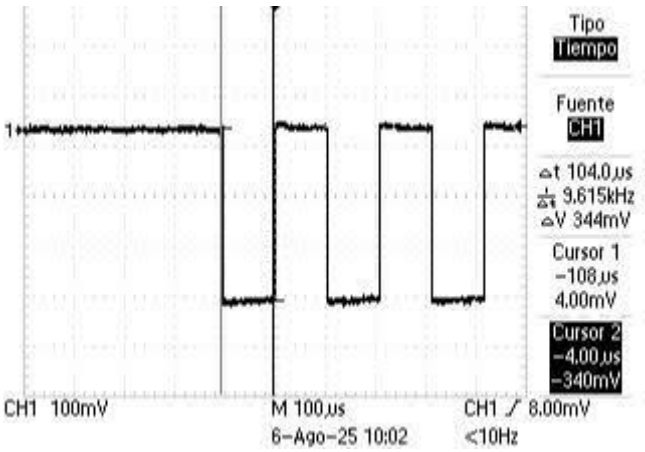

Ilustración 4 letra u en osciloscopio.

Como se observa el delta de tiempo corresponde a 104µs, esto representa el tiempo de bit de manera experimental, para obtener su valor de manera teórica, se hace uso de la siguiente ecuación.

$$Tiempo\ de\ bit = \frac{1}{Cantidad\ baudios}$$

Ecuación 1.

Por lo cual reemplazando en la ecuación número uno con respecto a la cantidad de baudios establecida en el código, se obtiene.

$$Tiempo\ de\ bit = \frac{1}{9600}$$

$$Tiempo\ de\ bit = 104{,}1\mu s$$

De manera teórica, el valor de duración del tiempo de bit corresponde a 104,1µs. A partir de esto, se realiza el cálculo del porcentaje de error, como se observa a continuación.

$$\%Error = \frac{|Valor\ teorico - Valor\ experimental|}{|Valor\ teorico|} x100$$

$$\%Error = \frac{|104 - 104{,}1|}{|104|} x100$$

$$\%Error = 0{,}09\%$$

Como se observa, el porcentaje de error es muy bajo, lo que indica que la teórica con respecto a la realidad es acertada.

III. Parámetros de la UART.

En esta parte consultaremos dos documentos, la configuración de la UART de los dispositivos en MicroPython y la página oficial de MicroPython para poder respondes las siguientes preguntas.

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

En MicroPython, la comunicación serial RS-232 se realiza usando la clase machine.UART, que permite configurar y controlar un puerto UART para transmitir y recibir datos de forma asíncrona.Cree una tabla con los métodos disponibles para la clase UART. Explique cada uno de ellos.

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

Método	Descripción
UART.deini()	Desactiva y libera el bus UART.
UART.any()	Devuelve el número de bytes disponibles para leer. Retorna 0 si no hay datos.

UART.read([nbytes])	Lee hastanbytesde datos. Si no se especifica, intenta leer todos los disponibles. Devuelve los datos comobytesoNonesi hay error o timeout.
UART.readinto(buf[,nbytes])	Lee datos directamente en un buffer existente. Puede leer hastanbyteso el tamaño del buffer. Devuelve el número de bytes leídos
UART.readline()	Lee una línea completa terminada en\n. Si no llega el salto de línea, devuelve lo recibido al agotarse el tiempo de espera
UART.write()	Envía los datos del buffer por UART. Devuelve la cantidad de bytes enviados oNonesi ocurre un timeout.
UART.sendbreak()	Envía una señal de*break*, manteniendo la línea en estado bajo durante más tiempo del normal (usado en ciertos protocolos de control).

Tabla 1 métodos disponibles clase UART

- ¿Como se modifica la tasa de baudios?
  En MicroPython, la velocidad del UART (tasa de baudios) se configura con el parámetro baudrate al crear el objeto, como UART(1, baudrate=9600), o se puede cambiar después con uart.init(baudrate=115200). Es clave que el valor elegido sea

IV. MEDIDA DE LOS TIEMPO EN BIT.

En esta parte del laboratorio se transmitirá el carácter ASCII "W" utilizando seis tasas de baudios diferentes. Para cada una, se medirá el tiempo de bit experimental con el osciloscopio y se calculará el error respecto al valor teórico, completando una tabla con los siguientes datos: baudios, tiempo de bit teórico, tiempo de bit experimental, Δt, % de error, voltaje máximo y voltaje mínimo. Sin embargo, solo se incluirán en el informe las mediciones de dos baudios representativos (empezando por 300 baudios) para evitar saturar el documento con imágenes. En el osciloscopio se observará la forma de onda para obtener la amplitud, el tiempo de bit real y la diferencia de tiempo (Δt).

- 300 baudios.

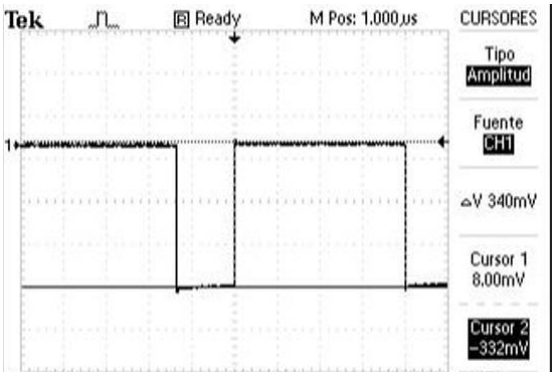

Ilustración 5 Voltaje máximo y mínimo 300 baudios.

A partir de lo que se observa, se puede determinar que el valor de voltaje máximo corresponde a 8mV y el valor mínimo corresponde a -332mV.

Así mismo se realizó la medición para el tiempo de bit de manera experimental, como se observa a continuación.

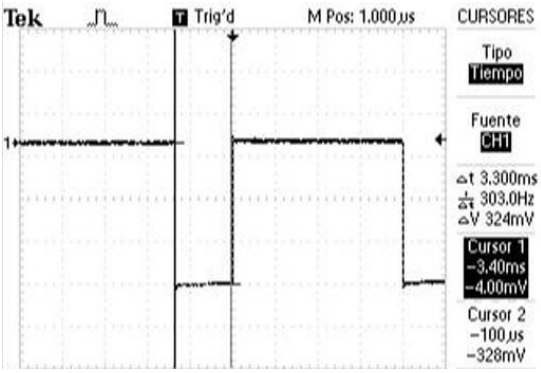

Ilustración 6. Tiempo de bit experimental para 300 baudios.

Como se observa, el tiempo de bit medido de manera experimental corresponde a 3,3ms, así mismo se calcula el tiempo total (Δt) que se demora el carácter ASCII en ser transmitido.

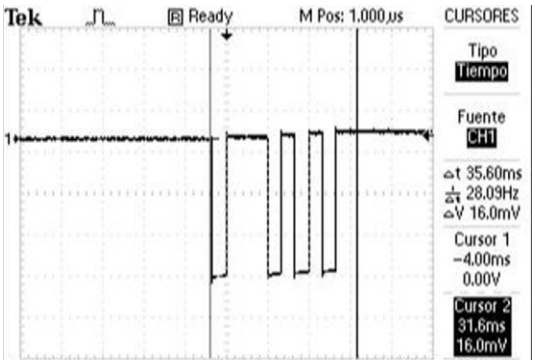

Ilustración 7. Delta t para 300 baudios.

Como se observa, el tiempo total de transmisión del carácter, corresponde a 35,6ms.
Este proceso se repite variando la cantidad de baudios por transmisión, por lo siguiente, se observa la cantidad para 9800 baudios.

- 9800 baudios.
-

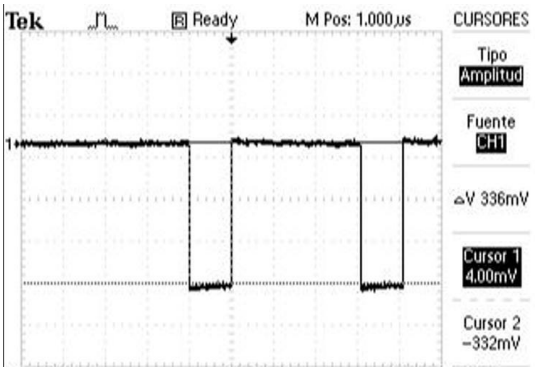

Ilustración 8. Voltaje máximo y mínimo para 9800 baudios.

Así mismo se realiza la medición del tiempo de bit para esta señal.

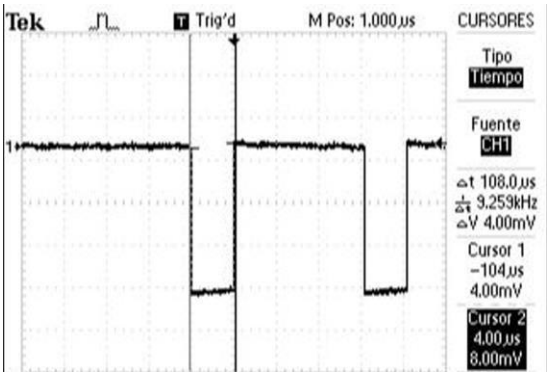

Ilustración 9. Tiempo de bit para 9800 baudios.

El tiempo de bit correspondiente, es de 108µs. Así mismo se realizo el calculo con respecto al tiempo completo en el que se demora todo el carácter ASCII en ser transmitido a 9800 baudios.

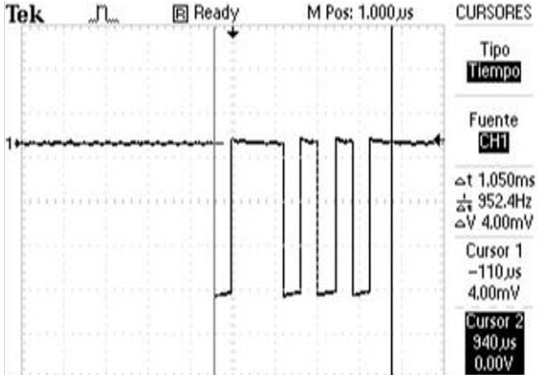

Ilustración 10 Tiempo de trama.

Como se observa, el tiempo de trama en el que el carácter ASCII es transmitido, corresponde a 1,05ms.
Este proceso se repite nuevamente con los siguientes rangos de baudios.

- 2400
- 4800
- 115200
- 230400

A partir de los resultados obtenidos, se completa la siguiente tabla.
Como se evidencia, esta posee el valor de tiempo teórico, esto se calculo directamente, haciendo uso de la ecuación número 1.

Baudios	Tiempo teórico	Tiempo de bit experimental	Δt	Voltaje Max	Voltaje min
300	3,3 ms	3,3 ms	35,6 ms	0.008	-0.332
2400	416,6µs	420 µs	4,2 ms	0.004	-0.332
4800	208,3µs	210 µs	2,1 ms	0.004	-0.332
9800	102,04 µs	108 µs	1,05 ms	0.004	-0.332
115200	8,6 µs	8,8 µs	90 µs	0.004	-0.332
230400	4,3 µs	4,2 µs	44 µs	0.004	-0.332

Tabla 2 Valores medidos.

A partir de esto, se realiza el calculo para obtener el porcentaje de error de acuerdo con los datos obtenidos, con respecto a cada uno de los datos calculados.

- 300 baudios.

$$Tiempo\ de\ bit = \frac{1}{300} = 3{,}3ms$$

- 2400 baudios.

$$Tiempo\ de\ bit = \frac{1}{2400} = 416{,}6\mu s$$

- 4800 baudios.

$$Tiempo\ de\ bit = \frac{1}{4800} = 208{,}3\mu s$$

- 9800 baudios.

$$Tiempo\ de\ bit = \frac{1}{9800} = 102{,}04\mu s$$

- 115200 baudios.

$$Tiempo\ de\ bit = \frac{1}{115200} = 8{,}6\mu s$$

- 230400 baudios.

$$Tiempo\ de\ bit = \frac{1}{230400} = 4{,}3\mu s$$

A partir de estos datos calculados, se complementa la siguiente tabla, observando el % de error con respecto a la teoría de la realidad.

Baudios	Tiempo teórico	Tiempo de bit exp	% de error
300	3,3 ms	3,3 ms	0
2400	416,6µs	420 µs	0.8
4800	208,3µs	210 µs	0.8
9800	102,04 µs	108 µs	5
115200	8,6 µs	8,8 µs	2.3
230400	4,3 µs	4,2 µs	2.3

Tabla 3 Porcentaje de error tiempo de bit.

A partir de esto, se puede inferir de manera tanto teórica como experimental, que la cantidad de baudios utilizada es inversamente proporcional con respecto al tiempo de bit de transmisión de un dato ASCII, lo que quiere decir que a mayor cantidad de baudios utilizados, el tiempo de transmisión será menor, así mismo podemos inferir que tanto las mediciones como los cálculos, corresponden directamente, esto debido al bajo porcentaje de error presente en la práctica.

V. ANÁLISIS DE LA ESTRUCTURA DEL PROTOCOLO RS232.

En esta parte del laboratorio se utilizarán 10 caracteres distintos —A, O, C, D, E, F, G, H, I, J, K— transmitidos a una tasa constante de 4800 baudios. Cada carácter se convertirá a su representación binaria, y se analizará si el número de unos (bits en 1) en su código es par o impar. Para organizar esta información, se presentará una tabla con los resultados.

Carácter en ASCII	Decimal	Binario	Impar	Par
A	65	0.1.0.0.0.0.0.1	1	0
O	79	0.1.0.0.1.1.1.1	0	1
C	67	0.1.0.0.0.0.1.1	0	1
D	68	0.1.0.0.0.1.0.0	1	0
E	69	0.1.0.0.0.1.0.1	0	1
F	70	0.1.0.0.0.1.1.0	0	1
G	71	0.1.0.0.0.1.1.1.	1	0
H	72	0.1.0.0.1.0.0.0	1	0
I	73	0.1.0.0.1.0.0.1	0	1
J	74	0.1.0.0.1.0.1.0	0	1

Tabla 4 Caracteres ASCII utilizados.

Es importante tener en cuenta que, en la transmisión serial, cada carácter se envía con un bit de inicio y un bit de parada, por lo que una secuencia de 8 bits se convierte en una trama de 10 bits en total. Además, en el osciloscopio, los datos se visualizan en orden inverso (del bit menos significativo al más significativo), es decir, que una secuencia como 10001111 se observará como 11110001 (sin incluir los bits de inicio y parada en la lectura directa de la forma de onda).

Para esta parte, se analizarán únicamente los caracteres A, O, F y J, mostrando sus señales en el osciloscopio. Se seleccionan solo cuatro caracteres para evitar saturar el informe con demasiadas imágenes, manteniendo una presentación clara y representativa.

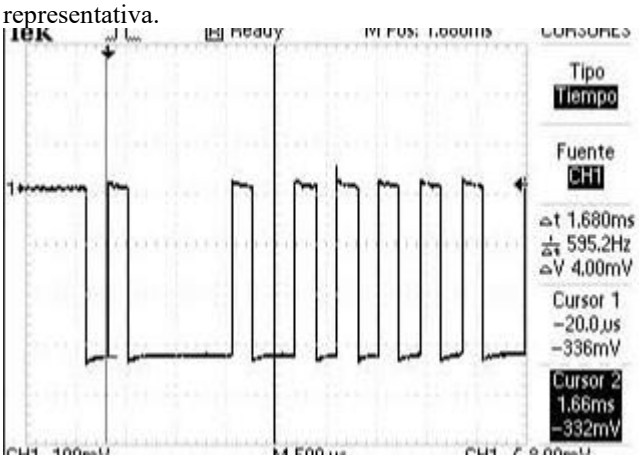

Ilustración 11 Carácter A par.

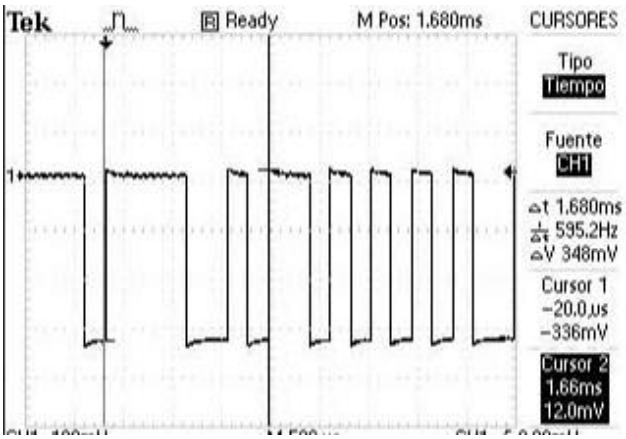

Ilustración 12 Carácter O par.

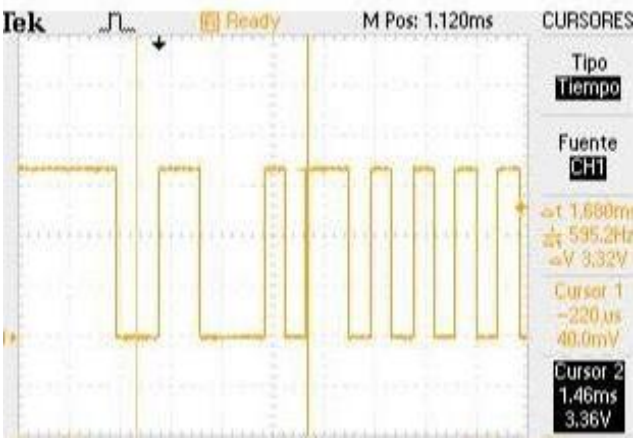

Ilustración 13 Carácter F par.

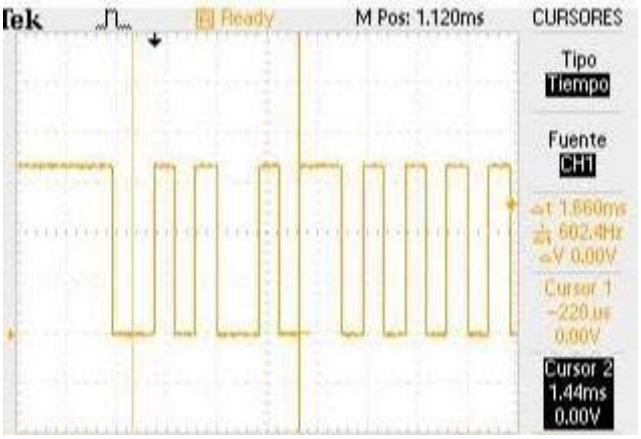

Ilustración 14 Carácter J par.

A partir de lo observad en las ilustraciones 11,12,13 y 14 se observa el comportamiento de la transmisión de datos de los caracteres ASCII, aquí se observa tanto el bit de inicio, el bit de parada y el bit de paridad, en este caso, dado con el fin de que todos los caracteres se mantengan de manera par.

A partir de eso, se realizarán en PowerPoint las gráficas de manera teórica, con respecto a cada uno de los datos, de esta manera comprobando lo observado en el osciloscopio.

- Carácter A par.

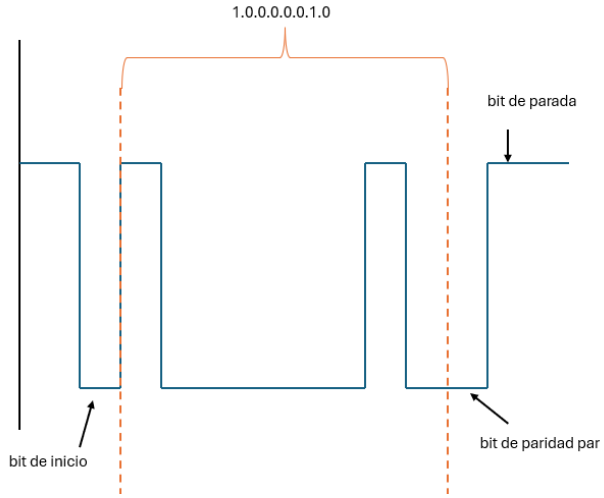

Ilustración 15 Carácter ASCII A de manera par.

- Carácter O par.

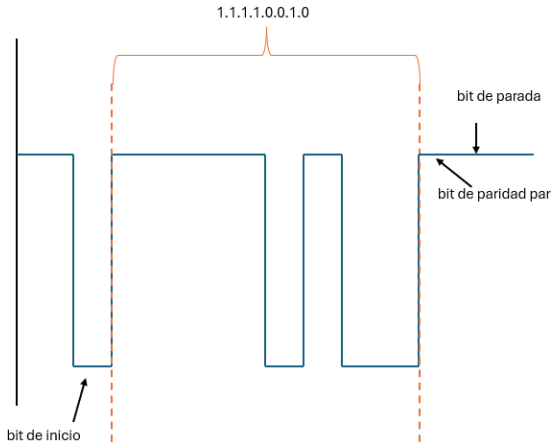

Ilustración 16 Carácter ASCII O de manera par.

Como se observa, el bit de paridad par corresponde a un uno, esto debido a la cantidad de unos presentes por el codigo ASCII, así mismo, se observa adecuadamente cada uno de los bits que componen el dato transmitido.

- Carácter F par.

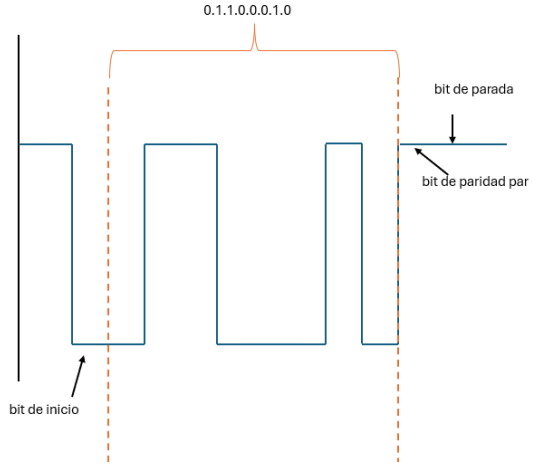

Ilustración 17 Carácter ASCII F par.

- Carácter J par.

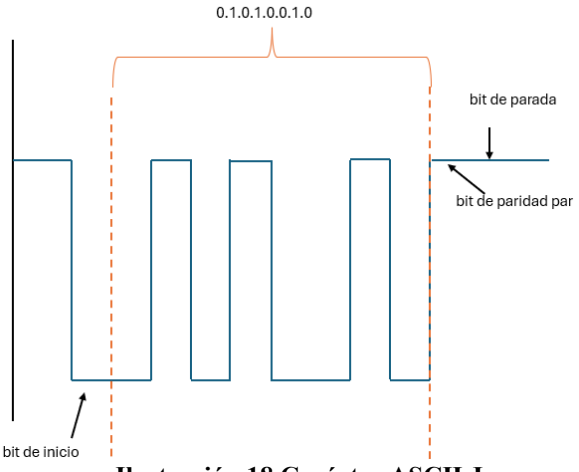

Ilustración 18 Carácter ASCII J par.

Así mismo se desea observar estos mismos caracteres de manera en que se encuentre la paridad impar, para ello se establece en el código “parity=1”, de esta manera, el comando dado, dará como resultado un bit de uno u otro comportamiento dependiendo del carácter ASCII a observar, por ende, los resultados obtenidos y vistos desde el osciloscopio son los siguientes.

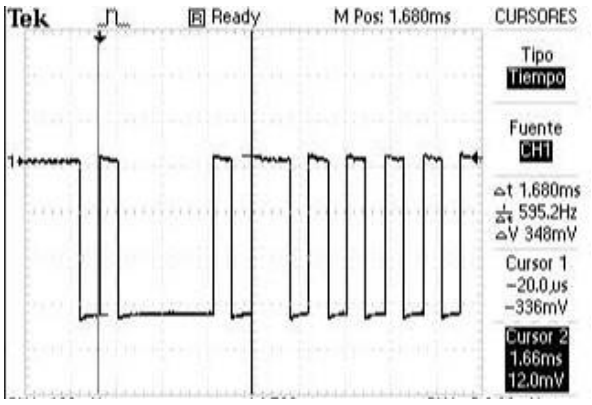

Ilustración 19 carácter ASCII A impar

Así mismo se realiza su diagrama de manera teorico, como se observa a continuacion.

- Carácter A impar.

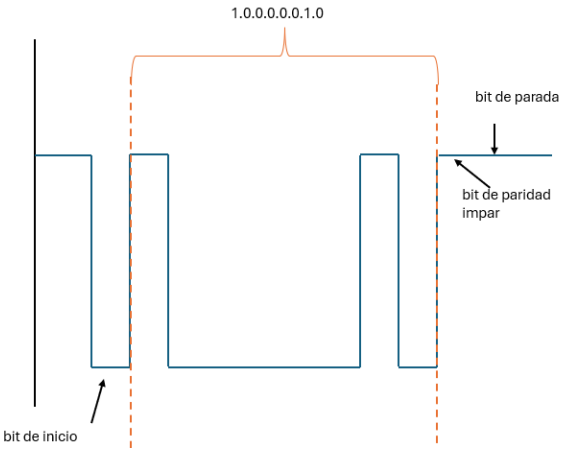

Ilustración 20 Diagrama carácter ASCII A impar.

En este caso, se observa como el comando afecta directamente el comportamiento, ya que para que el dato sea impar se adiciona un uno, para que el total de unos presentes en el dato sea impar.

- Carácter O impar.

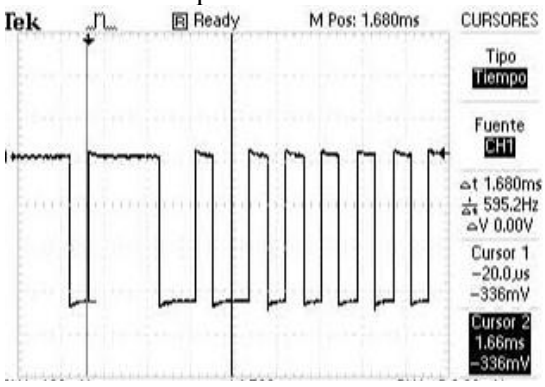

Ilustración 21 carácter ASCII O impar.

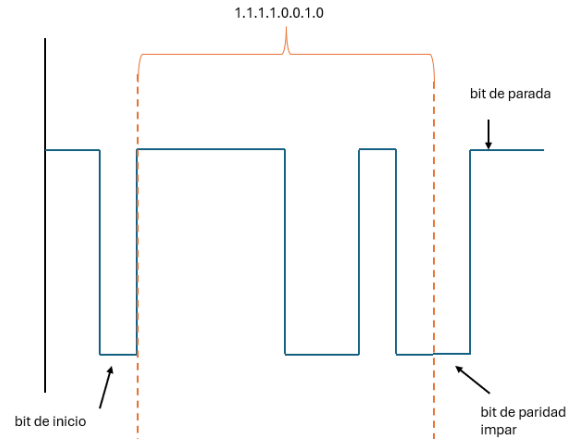

Ilustración 22 Diagrama de carácter ASCII O impar.

Reafirmando lo establecido con anticipación, se puede nuevamente, el comando indica que se comporte como un cero el bit adicional de paridad, debido a que los bits del código ASCII utilizado, originalmente presentan una cantidad de unos impar.

- Carácter F impar.

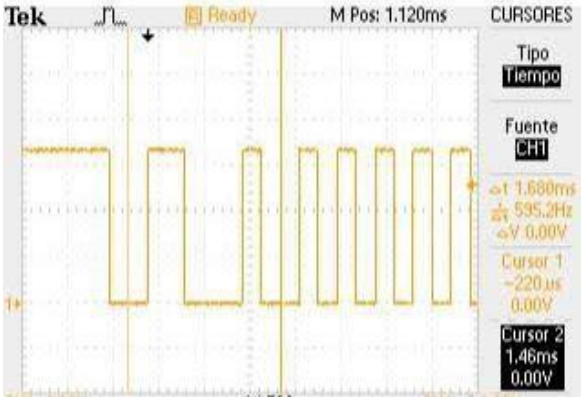

Ilustración 23 Carácter ASCII F impar.

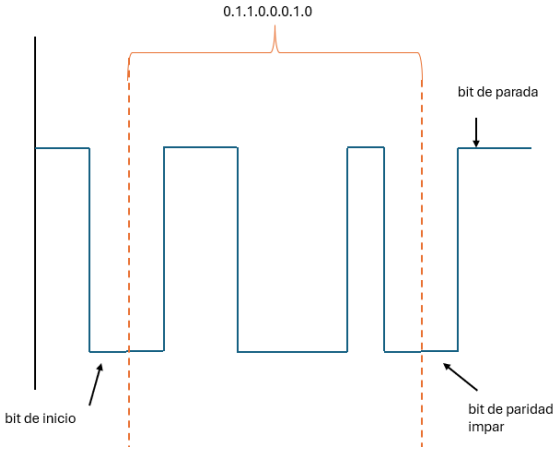

Ilustración 24 Diagrama carácter ASCII F impar.

- Carácter J impar.

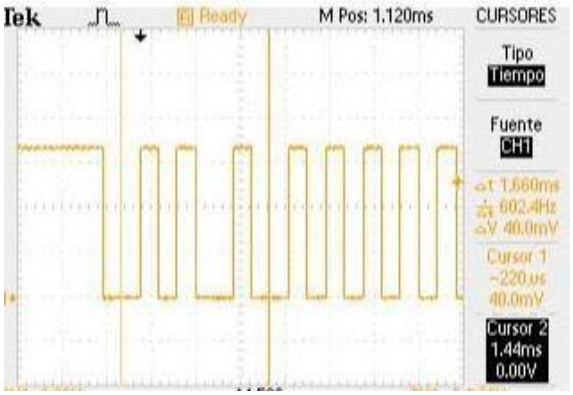

Ilustración 25 Carácter ASCII J impar.

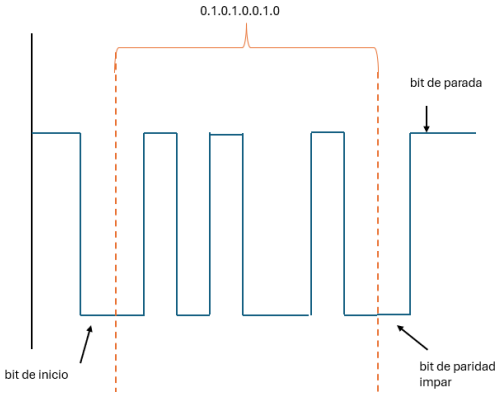

Ilustración 26 Diagrama carácter ASCII J impar.

Así mismo, se observan los anteriores caracteres de manera que no tengan paridad, para ello se establece en el código "parity=None" de esta manera el bit adicional no será agregado.

- Carácter A sin paridad.

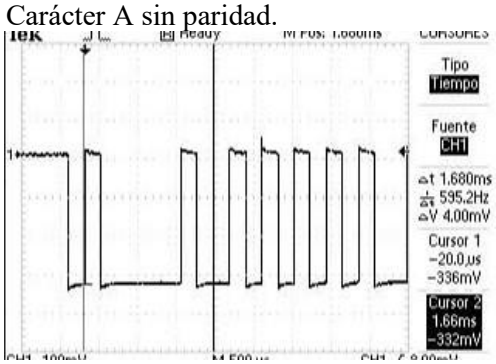

Ilustración 27 carácter ASCII A sin paridad.

- Carácter O sin paridad.

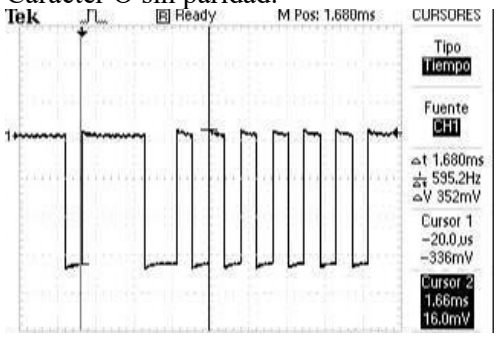

Ilustración 28 carácter ASCII O sin paridad.

- Carácter F sin paridad.

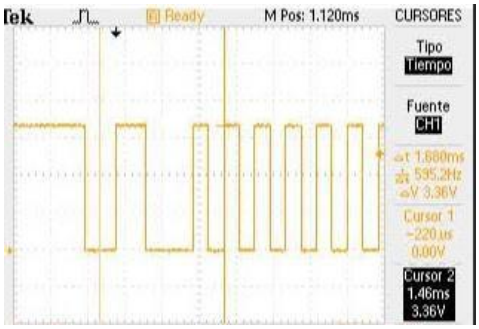

Ilustración 29 carácter ASCII F sin paridad.

- Carácter J sin paridad.

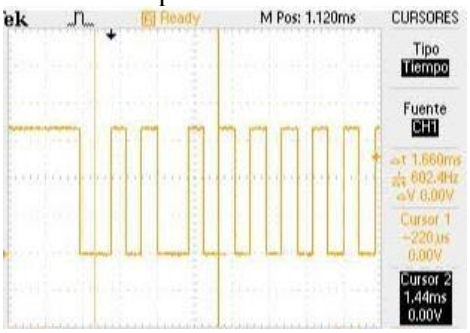

Ilustración 30 carácter ASCII J sin paridad.

Como se evidencia en las ilustraciones 27,28,29 y 30, el bit adicional de paridad se elimina, manteniendo únicamente el bit de stop, dando por concluida la transmisión del carácter establecido, para proseguir con el siguiente, en este caso la letra U.

VI. MEDIDA DEL TIEMPO DE LA TRAMA.

A partir de observar el tiempo total de una transmisión de datos de 60 caracteres ASCII, con un tiempo de bit de 600 baudios con paridad par y con un total de 8 bits de dato por cada uno de los caracteres, se hace uso de la siguiente ecuación.

$$Tiempo\ de\ trama = \frac{1}{baudios} x\ bits\ por\ caracter\ x\ caracteres\ totales$$

Reemplazando en la ecuación a partir de los datos a utilizar se obtiene.

$$Tiempo\ de\ trama = \frac{1}{600} x\ 11\ x\ 60$$

$$Tiempo\ de\ trama = 1{,}1s$$

Se obtiene de manera teórica, que el tiempo de trama para la transmisión de los 60 caracteres con dichas especificaciones corresponde a 1,1segundos, por lo cual para corroborar esto se utiliza el siguiente código, el cual nos permitirá comprobar esto en el osciloscopio.

```
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))

while True:
    led.on()
    uart.write("A"*60)
    utime.sleep(1)
    led.off()
    utime.sleep(1)
```

Se recalca en el código, la paridad igual a 0, de manera que sea par esta última, así mismo se resaltan los baudios establecidos. A partir de esto se obtiene la siguiente señal en el osciloscopio.

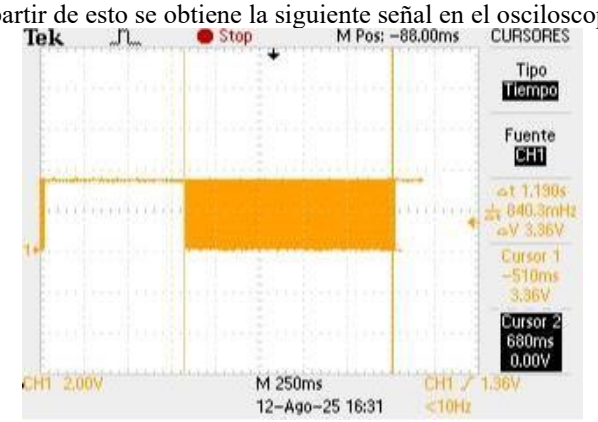

Ilustración 31 Tiempo de trama para los 60 caracteres ASCII A

Como se evidencia el tiempo de trama obtenido de manera experimental corresponde a 1,19segundos. A partir de esto, se realiza el calculo del porcentaje de error.

$$\%Error = \frac{|1{,}1 - 1{,}19|}{|1{,}1|} x100$$

$$\%Error = 8\%$$

El porcentaje de error obtenido nos confirma que la realidad concuerda con lo planteado de manera teórica, este error puede proceder directamente por el ajuste exacto de los cursores, donde procede directamente de error humano.

Por otro lado, se busca observar el tiempo de trama de los 60 caracteres ASCII A, pero esta vez sin paridad, para ello se establece en el código "parity=None", dando como resultado un bit menos por cada uno de los caracteres, por lo cual calcular el tiempo de trama de manera teórica se realiza de la siguiente manera.

$$Tiempo\ de\ trama = \frac{1}{600} x\ 10\ x\ 60$$

$$Tiempo\ de\ trama = 1s$$

Por lo cual lo observado de manera experimental es lo siguiente.

Ilustración 32 Tiempo de trama sin paridad de 60 caracteres ASCII A

Como se observa, el tiempo de trama de los 60 caracteres sin paridad, corresponde a 1,07 segundos, por lo cual se puede

indicar que la teoría corresponde con la realidad, siendo que el error y la pequeña diferencia visible, es directamente al establecer la posición de los cursores.
Así mismo se transmite el mensaje "UMNG LIDER EN INGENIERIA EN TELECOMUNICACIONES" utilizando configuración de 57600 baudios, con 7 bits de datos, paridad par y 2 bits de parada. Cada carácter se envía como una trama de 11 bits en total, incluyendo: 1 bit de inicio, 7 bits de datos, 1 bit de paridad y 2 bits de parada, por lo cual el tiempo de trama calculado de manera teórica corresponde a.

$$Tiempo\ de\ trama = \frac{1}{57600}\ x\ 11\ x\ 45$$

$$Tiempo\ de\ trama = 8{,}59ms$$

Por lo cual teóricamente, el tiempo de transmisión de este mensaje corresponde a 8,59 milisegundos, a partir de esto se medio en el osciloscopio el tiempo de trama para corroborar esto, para ello se utilizó el siguiente código.

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

led = machine.Pin("LED", machine.Pin.OUT)
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 TELECOMUNICACIONES")
    utime.sleep(1)
    led.off()
    utime.sleep(1)
```

Por lo cual lo observado en el osciloscopio corresponde a.

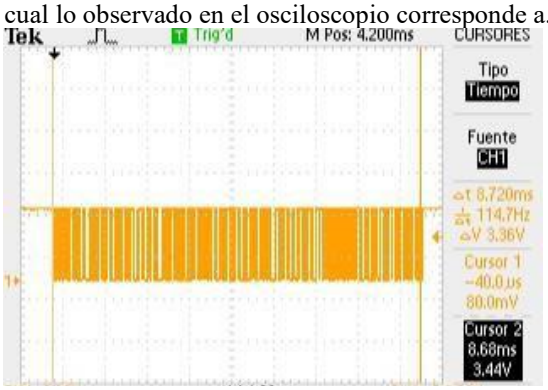

Ilustración 33 Tiempo de trama mensaje UMNG.

Como se observa, el tiempo de trama experimental corresponde a 8,72ms, por lo cual se realiza el calculo del porcentaje de error con respecto a su valor teórico calculado.

$$\%Error = \frac{|8{,}59 - 8{,}72|}{|8{,}59|} x100$$

$$\%Error = 1{,}5\%$$

Como se observa, el porcentaje de error es relativamente bajo, dando de esta manera veracidad con respecto a lo que se observa con lo que se calcula.

VII. RETO DE PROGRAMACIÓN.

El reto de programación consiste en realizar una comunicación entre dos Rasperry Pi pico 2W, inicialmente se busca realizar la comunicación hall dúplex, es decir que uno envía mientras que el otro recibe, para ello se establece el siguiente montaje en la portoboard.

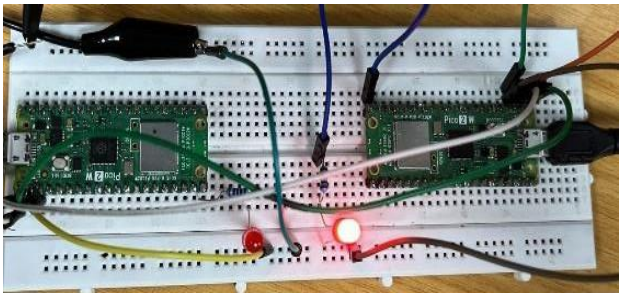

Ilustración 34 comunicación Hall dúplex.

Como se observa, las dos rasperry están interconectadas entre ellas a partir de Rx para Tx y Tx para Rx, manteniendo una tierra común, así mismo se hace uso de dos leds que se utilizan para comprobar que la comunicación se garantice.
De manera que sea posible realizar la comunicación se utiliza el siguiente código, que pertenecerá a la rasperry que enviará el carácter A.

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

# LED en GPIO15
led = Pin(15, Pin.OUT)
# Configurar UART con 9600 bps, 8 bits, sin paridad, 1 bit de stop
uart = UART(0, baudrate=9600, bits=8, parity=None, stop=1, tx=Pin(0), rx=Pin(1))

while True:
    # 1. Enviar "A" cada 2 segundos
    uart.write("A")
    print("Enviado: A")
    utime.sleep(2)

    # 2. Esperar respuesta
    start_wait = utime.ticks_ms()
    while utime.ticks_diff(utime.ticks_ms(), start_wait) < 1000: # Espera hasta 1 segundo
        if uart.any():
            dato = uart.read(1)
            if dato == b"B":
                print("Recibido: B")
                # Parpadeo LED durante 3 segundos
                start_blink = utime.ticks_ms()
                while utime.ticks_diff(utime.ticks_ms(), start_blink) < 3000:
                    led.on()
                    utime.sleep(0.25)
                    led.off()
                    utime.sleep(0.25)
                break
```

En este caso el código configura una comunicación serial UART a 9600 bps y envía el carácter "A" cada 2 segundos; si recibe como respuesta el carácter "B", activa un parpadeo del LED conectado al GPIO 15, alternándolo 6 veces con intervalos de 0.25 segundos, lo que genera un efecto de señalización visual que confirma la recepción exitosa del mensaje.
Así mismo, está presente la contraparte con el otro rasperry, que corresponderá a recibir, el código utilizado en función es.

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

# LED en GPIO15
led = Pin(15, Pin.OUT)

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

# Contador de recepciones
contador = 0

while True:
    if uart.any():
        dato = uart.read(1)
        if dato == b"A":
            contador += 1
            print(f"Recibido: A | Conteo: {contador}")

            # Parpadeo del LED durante 5 segundos
            start_blink = utime.ticks_ms()
            while utime.ticks_diff(utime.ticks_ms(), start_blink) < 5000:
                led.on()
                utime.sleep(0.25)
                led.off()
                utime.sleep(0.25)

            # Enviar respuesta "B"
            uart.write("B")
            print("Enviado: B")

            # Guardar conteo en archivo
            with open("recibidos.txt", "a") as f:
                f.write(f"{contador}\n")
```

Este código monitorea constantemente el puerto UART para detectar el carácter "A". Cada vez que lo recibe, incrementa un contador, activa un parpadeo del LED durante 5 segundos, envía una respuesta "B" y guarda el valor actual del contador en un archivo llamado recibidos.txt. Permite verificar comunicación bidireccional y registrar datos de forma persistente. Este archivo se puede observar a continuación.

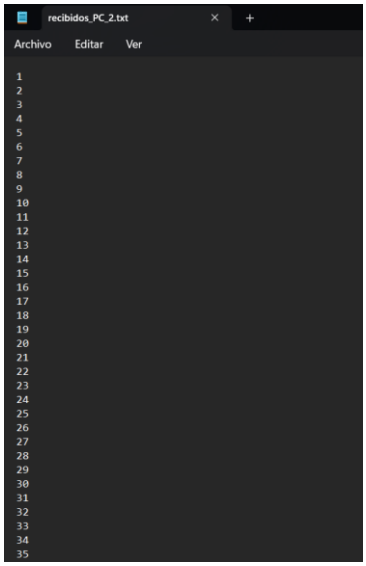

Ilustración 35 Archivo contador de mensajes recibidos.

Como se observa, cada vez que la rasperry recibe el carácter A se incrementa el contador, almacenando esto en el archivo "recibidos.txt".
Así mismo para comprobar adecuadamente la comunicación, se hace uso del osciloscopio con el fin de mirar la señal enviada en cada una de las rasperry.

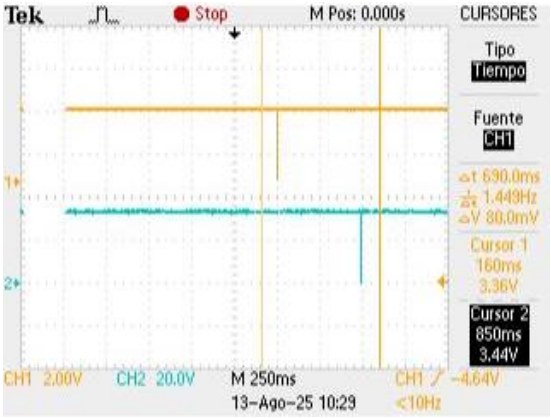

Ilustración 36 comunicación hall dúplex en el osciloscopio.

Como se observa, la rasperry que enviar el carácter A envía el dato y posteriormente la rasperry de recepción envía el carácter B, estos tiempos se observan a continuación.

Ilustración 37 Tiempo de envío y recepción.

Como se observa, el tiempo de envió con respecto al tiempo de recepción corresponde a 500ms, lo que significa que cada 0,5 segundos la rasperry de emisión envía el carácter A, para posteriormente recibir el carácter B de la rasperry de recepción.

Así mismo, se busca realizar la comunicación Full dúplex, es decir que ambas rasperry están enviando y recibiendo datos al mismo tiempo, para ello observar esto, se añade otro led, esto con el fin de monitorear la adecuada comunicación, siendo un led de enviado con su contraparte de recibido, por lo cual el montaje del circuito es el siguiente.

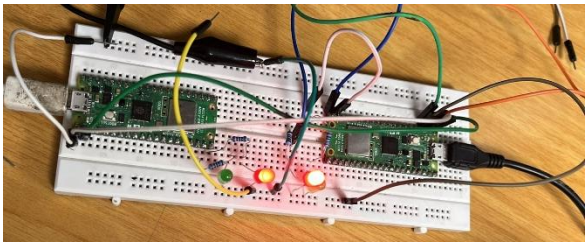

Ilustración 38 Montaje comunicación full dúplex

El código utilizado para ambas rasperry es el siguiente, con la única diferencia que se modifica el carácter enviado en la variable "MI_CARACTER".

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

LED_FIJO = 15
LED_TITILA = 14
MI_CARACTER = "B"
PERIODO = 0.5

led_fijo = Pin(LED_FIJO, Pin.OUT)
led_titila = Pin(LED_TITILA, Pin.OUT)

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

while True:
    led_fijo.on()
    uart.write(MI_CARACTER)
    print(f"Enviado: {MI_CARACTER}")

    start_time = utime.ticks_ms()
    while utime.ticks_diff(utime.ticks_ms(), start_time) < PERIODO * 1000:
        if uart.any():
            dato = uart.read(1)
            if dato:
                print(f"Recibido: {dato.decode()}")
                led_titila.on()
                utime.sleep(0.1)
                led_titila.off()
        utime.sleep(0.05)

    led_fijo.off()

    start_time = utime.ticks_ms()
    while utime.ticks_diff(utime.ticks_ms(), start_time) < PERIODO * 1000:
        if uart.any():
            dato = uart.read(1)
            if dato:
                print(f"Recibido: {dato.decode()}")
                led_titila.on()
                utime.sleep(0.1)
                led_titila.off()
        utime.sleep(0.05)

    MI_CARACTER = "B" if MI_CARACTER == "A" else "A"
```

Siendo que este código implementa una comunicación serial alternada entre dos dispositivos usando UART. Cada dispositivo envía su carácter (A o B) durante un intervalo fijo (0.5 s), mientras monitorea si recibe datos del otro. Durante su fase de transmisión, enciende un LED fijo; al recibir cualquier dato, parpadea un segundo LED brevemente. Tras cada ciclo, alterna el carácter a enviar para mantener una secuencia tipo A-B-A-B. Permite simular un protocolo de comunicación controlado y bidireccional con retroalimentación visual.

Para corroborar esto, se utiliza el osciloscopio, observando la salida de ambas rasperry (Tx), por lo cual lo obtenido es.

Ilustración 39 comunicación Full dúplex

Como se observa, ambas señales se encuentran a la vez que son enviadas, por lo cual se infiere que se esta realizando de manera educada la comunicación.

VIII. Comparación Rasperry Pi Pico y Pi Pico 2W.

La Raspberry Pi Pico 2 W representa una evolución significativa frente a la Pico W, ofreciendo mayor rendimiento gracias a un procesador más rápido (150 MHz vs 133 MHz), el doble de memoria RAM y almacenamiento, así como más interfaces UART, I²C, SPI, canales PWM y recursos PIO. Aunque ambas utilizan el mismo chip inalámbrico, la Pico 2 W mejora el uso de Bluetooth 5.2, incorpora funciones de seguridad avanzada (como TrustZone y arranque seguro) y un regulador de voltaje más eficiente (buck-boost) que optimiza el consumo energético. Con un ligero aumento en el precio, es ideal para proyectos más exigentes, mientras que la Pico W sigue siendo una buena opción para aplicaciones básicas.

Aspecto	Pico W	Pico 2W
Cpu/Arquitectura	Dual Cortex-M0 + @ 133M	Dual Cortex-M33 o RISV-V @

	HZ	150M HZ
Memoria	264 KB SRAM / 2MB Flash	520 KB SRAM / 4 MB Flash
Conectividad	Wi-Fi + Bluetooth 5.2	Wi-Fi + Bluetooth 5.2
Seguridad	Basica	TrustZone, secure boot, TRNG, OTP
UART	2 (UART0 y UART1) con TX y RX en pines asignables	3 UART (UART0, UART1, UART2), mayor flexibilidad para asignar TX/RX
SPI	2 controladores SPI	3 controladores SPI
Precio	35700	38080

TABLA 5 Comparación entre rasperry pi pico y pi pico 2W

IX. ANÁLISIS.

Este laboratorio ofrece una exploración práctica y completa de la comunicación serial RS-232 utilizando la Raspberry Pi Pico 2W y MicroPython. El análisis revela varios puntos clave:

1. Teoría y práctica coinciden

Los cálculos que hicimos en clase coincidieron muy bien con lo que medimos en el laboratorio, con diferencias menores al 5%. Esto nos muestra que las fórmulas que aprendimos realmente funcionan en la práctica, lo cual da confianza para aplicarlas en proyectos futuros.

2. Relación velocidad-tiempo confirmada

Verificamos que a mayor velocidad de transmisión (más baudios), menor es el tiempo que dura cada bit de información. Por ejemplo, a 300 baudios, el valor calculado y el medido fue exactamente el mismo (3.3 ms), demostrando que la relación es precisa y predecible.

3. Funcionamiento de la paridad

Observamos cómo funciona el sistema de detección de errores mediante paridad. Dependiendo de cuántos "unos" tenga un carácter, se añade un bit extra para que el total sea par o impar. Por ejemplo, con el carácter 'O' (que tiene 5 unos), se añadió un bit de paridad igual a 1 para hacer el total par.

4. Programación sencilla y efectiva

Configurar la comunicación serial con MicroPython resultó ser bastante intuitivo. Pudimos probar diferentes configuraciones (cambiar bits de datos, añadir paridad, modificar bits de parada) y todo funcionó como se esperaba, lo que demuestra la versatilidad de la placa.

5. Comunicación entre dispositivos exitosa

Logramos establecer comunicación entre dos placas, tanto en modo "uno habla a la vez" como "ambos hablan simultáneamente". Además, el tiempo de respuesta fue constante (500 ms), lo que hace que la comunicación sea confiable y predecible.

6. Ventajas de la Pico 2W

La nueva versión de la placa (Pico 2W) mejora notablemente a la anterior: es más rápida, tiene el doble de memoria y mejores funciones de seguridad, pero sigue siendo compatible con los proyectos anteriores. Esto la convierte en una excelente opción para proyectos más avanzados sin tener que reiniciar el aprendizaje desde cero.

X. CONCLUSIONES.

1. La comunicación serial RS-232 sigue siendo relevante: A pesar de ser un estándar tradicional, su implementación práctica demuestra su utilidad continua en sistemas embebidos modernos, especialmente para aplicaciones donde la simplicidad y confiabilidad son prioritarias.
2. MicroPython facilita el aprendizaje: La Raspberry Pi Pico 2W con MicroPython proporciona una plataforma accesible y potente para comprender conceptos fundamentales de comunicaciones digitales, permitiendo pasar de la teoría a la práctica de manera inmediata.
3. Precisión en la implementación: Los bajos porcentajes de error entre valores teóricos y experimentales (como el 0% para 300 baudios) validan que la implementación es técnicamente correcta y que los dispositivos funcionan según las especificaciones.
4. La Pico 2W es una mejora significativa: Con su mayor velocidad (150 MHz vs 133 MHz), doble de memoria (520 KB SRAM / 4 MB Flash), y características de seguridad avanzadas (TrustZone, secure boot), representa una evolución importante para proyectos IoT más exigentes.

REFERENCIAS

[1] Raspberry Pi Ltd., *Raspberry Pi Pico W Product Brief*, 2022. [En línea]. Disponible: https://datasheets.raspberrypi.com/picow/pico-w-product-brief.pdf

[2] Raspberry Pi Ltd., *Raspberry Pi Pico 2 W Product Brief*, 2024. [En línea]. Disponible: https://datasheets.raspberrypi.com/pico/pico-2-w-product-brief.pdf

[3] Digi-Key Electronics, “Raspberry Pi Pico 2 vs. Original Pico: What's New?,” *Maker.io*, nov. 2024. [En línea]. Disponible: https://www.digikey.com/en/maker/blogs/2024/raspberry-pi-pico-2-vs-original-pico-whats-new

[4] MicroPython Documentation, “class UART – duplex serial communication bus,” *machine.UART library*, versión más reciente. [En línea]. Disponible: https://docs.micropython.org/en/latest/library/machine.UART.html