Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0523 – Circuitos Digitales II
II ciclo 2024

Proyecto Final

Protocolo UART

Integrantes:
Isaí González S - C03457
Daniel De La O Rojas - B82528
Javier Elías Saca - B66418

Profesora: Ing. Ana Eugenia Sanchez Villalobos,

4 de diciembre de 2024

Índice

Índice de figuras

1. Introducción del Protocolo

UART, o Transmisor/Receptor Asíncrono Universal, no es un protocolo de comunicación como SPI o I2C, sino un circuito físico en un microcontrolador o un IC independiente. Su propósito principal es transmitir y recibir datos de forma serial. En nuestro caso, es un protocolo UART full duplex, que como se estudió en clase, consiste en dos módulos que funcionan como transmisor y receptor cada uno. En el siguiente vamos a detallar su comportamiento, basándonos en la información de su funcionamiento.

1.1. ¿Cómo funciona la comunicación UART?

En una comunicación UART, dos dispositivos con UART se comunican directamente entre sí. El UART transmisor convierte los datos paralelos (generalmente provenientes de una CPU o microcontrolador) en formato serial. Luego, envía esos datos serializados al UART receptor, que los convierte nuevamente en formato paralelo para su uso. Esta transmisión requiere solo dos cables:

- Tx (transmisión): transmite datos.
- Rx (recepción): recibe datos

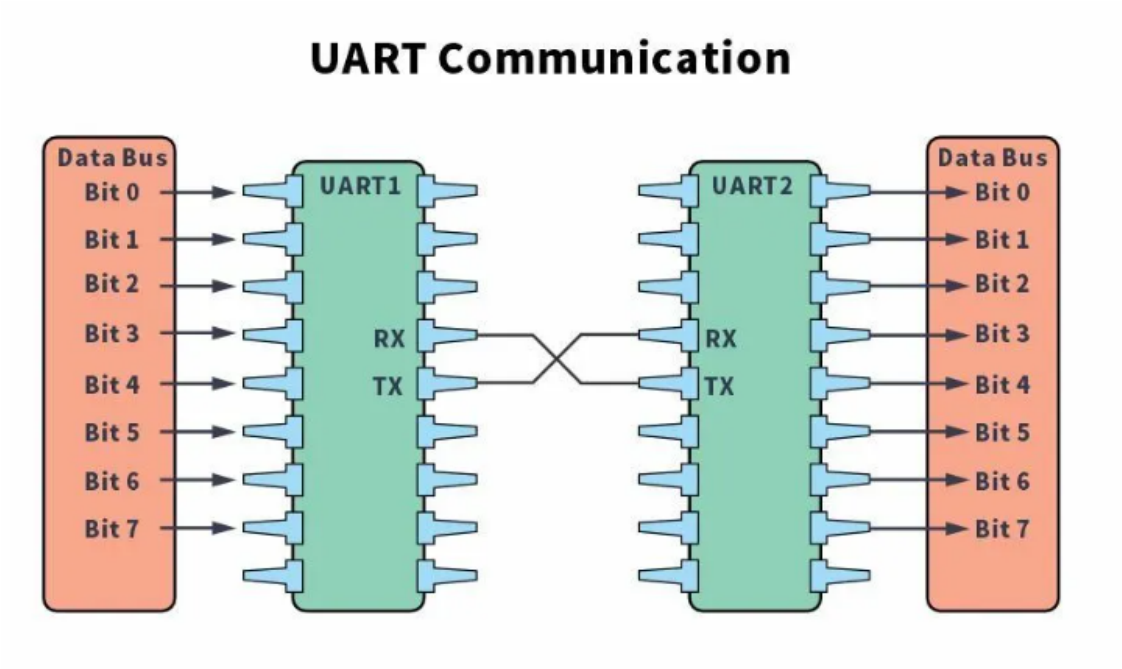

Figura 1: Protocolo UART

Características:

1. Asincronía: La comunicación no requiere una señal de reloj compartida. En lugar de esto, el UART transmisor agrega bits especiales (inicio y parada) al paquete de datos para sincronizar la transmisión con el UART receptor.
2. Bits de inicio y parada:

   El bit de inicio señala el comienzo de un paquete de datos y cambia la línea de transmisión de alta (1) a baja (0).

   El bit de parada marca el final de la transmisión, regresando la línea a su estado alto (1).

3. Baud Rate (Tasa de Baudios):

   Define la velocidad de transmisión de datos en bits por segundo (bps).

   Ambos UART deben operar a la misma tasa de baudios (o con una diferencia menor al 10 %) para evitar errores.

4. Paridad (opcional):

   Verifica la integridad de los datos. El UART receptor compara los bits del marco de datos con el bit de paridad para detectar posibles errores.

2. Explicación del wavedrom

Para mayor orden y claridad se expicará el Wavedrom en orden de cada Módulo
UART1 (Transmisor):
Cómo se observa las señales, estas se comportan de modo que primero se debe armar el paquete de datos que se envían hacía el receptor a través de tx1, por tanto que se irá armando el paquete del protocolo a medida que se van activando una serie de señales que indican el posicionamiento de: start_bit, data, parity_bit, y stop_bit.

- Primero, al haber un reset, se inicializan los valores, con idle_bit en uno, este valor será uno siempre que el protocolo no esté activo.
- Seguidamente se activa la señal start_bit que da inicio a la transmisión.
- Una vez que start_bit ha indicado el inicio del protocolo, UART1 recibe el dato en paralelo y mediante data_in, la máquina de estados desplaza los datos usando la configuración PISO (Parallel-in, serial-out), de modo que como se observa, la señal de tx1, va desplazando serialmente los datos que ingresan en paralelo, siguiendo el orden LSB.
- Cuando acaba de desplazarse el dato de 8bits, la señal de parity_bit entra en acción, la cuál calcula la paridad par, e indica si la cantidad de unos en el dato es par o impar, de modo que coloca el valor correspondiente al dato en la señal.
- Por último, se activa la señal para el stop_bit, el cual indica que se llegó al fin del envío de datos. Una vez sucede esto, se activa IDLE, quedando en uno la señal.

Como bien se indica, a medida que se van activando todas estas señales, tx1 va reflejando este comportamiento, siendo que por este cable es por dónde se hace la transferencia de datos.

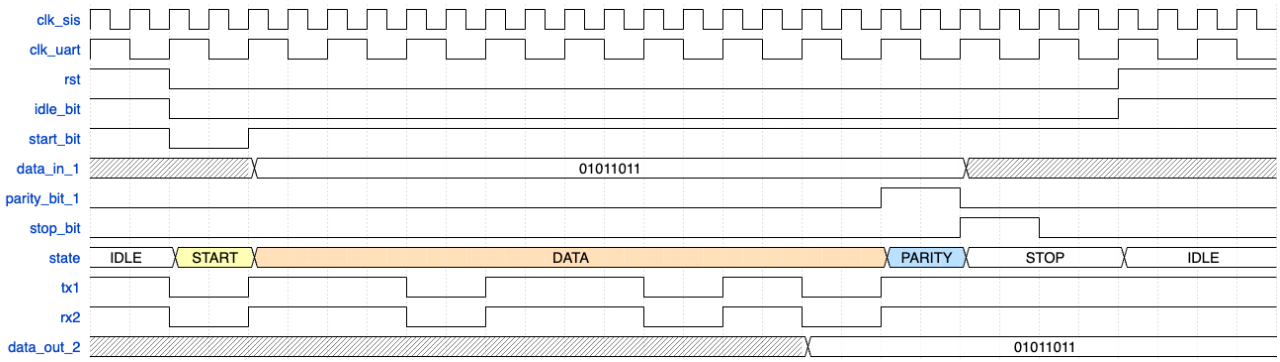

Figura 2: Wavedrom de la tranmisión de UART1 a UART2.

UART2 (Receiver):
El comportamiento del segundo módulo como receptor, tiene algunas diferencias respecto al primero, y es que este recibe los datos através de tx1, que en este módulo se llama rx2, y convierte los datos en serie a paralelo, siguiendo la configuración SIPO (Serial-in, parallel-out).

- Se recibe la señal mediante rx2.
- El dato recibido mediante rx2 se desempaqueta y se muestra el valor del dato en la salida data_out_1

El resto del protocolo invierte el comportamiento de los UART, en el que el transmisor recibe, y el receptor envía. Solo cambian los nombres de las señales pero se comporta exactamente igual.

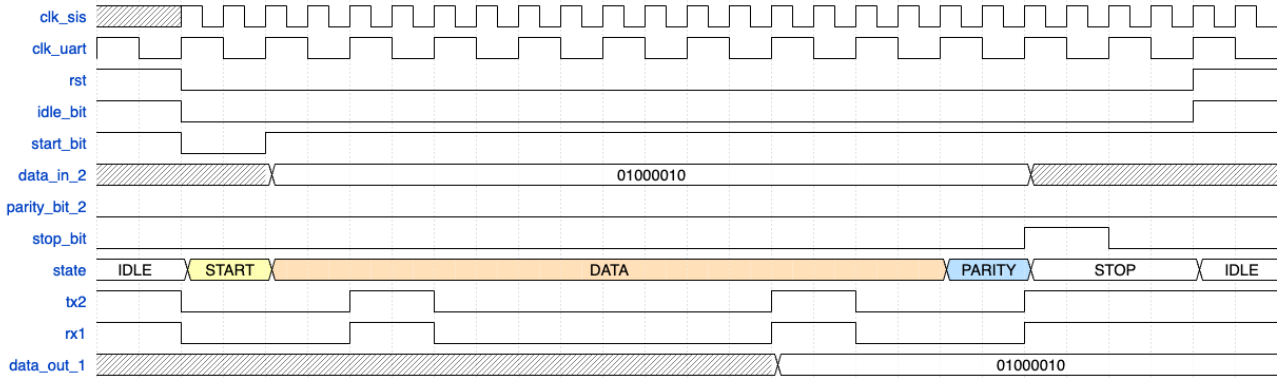

Figura 3: Wavedrom de la tranmisión de UART2 a UART1.

3. Como se corren los archivos

Se desarrolla un documento MAKEFILE que se busca que no tenga ningun problema para correr el programa en Linux.

```
uart:
        iverilog -o tb.vvp UART_tb.v
        vvp tb.vvp
        gtkwave GTKFINAL.gtkw

clean:
        rm -rf tb.vvp UART_tb.vcd
```

Para correrlo, solo hace falta escribir en la terminal

```
make uart
```

Cabe recalcar que este makefile se desarrolló con la intención de que se puedan abrir todas las señales de manera más rápida, por lo que es necesario tener el archivo GTKFINAL.gtkw en el directorio (el cual si viene incluido en el github).

4. Trabajo en Github

El repositorio de Github es el siguiente:
https://github.com/IGonzalez16/ProyectoDigitales2
En dicho repositorio se puede observar el aporte de cada uno de los integrantes del grupo, en el cual además, se observa que el código presentó muchas modificaciones, ya que nos parecía que

algunas cosas eran innecesarias, luego muy complejas. A último momento se decidió empezar de nuevo todo el código de cero, con calma y más precisión, aún así se logró un resultado óptimo.

5. Resultados

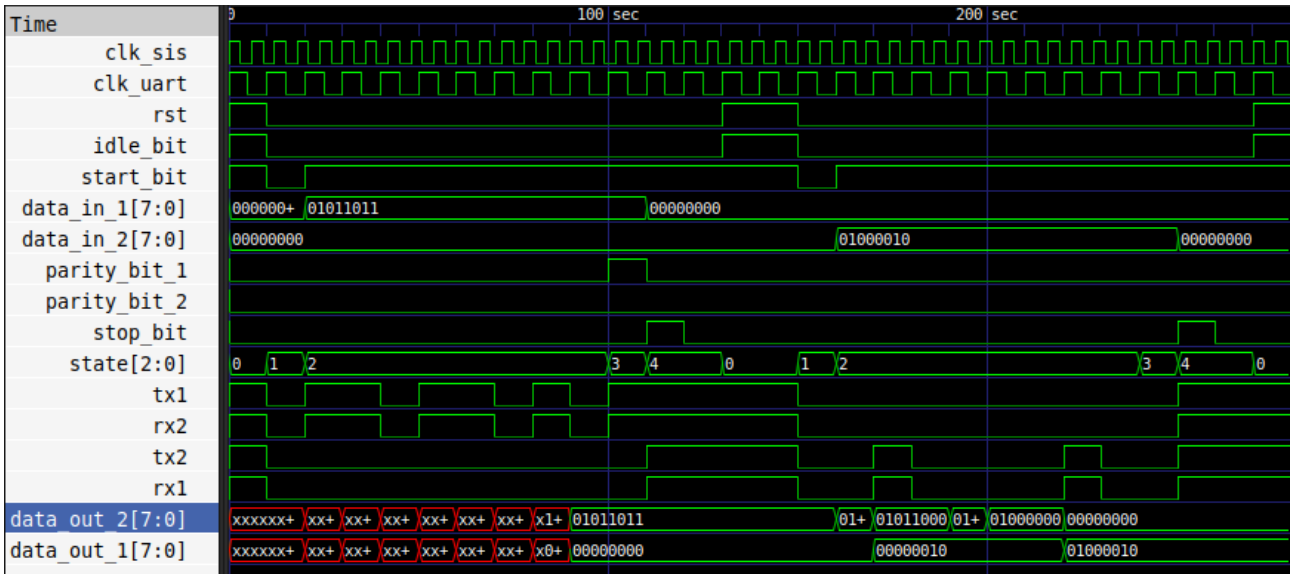

Figura 4: Respuesta de GTKWave

Para la primer iteración o envío, como se observa en el comportamiento de las señales de gtkwave, estas se comportan como se esperaba en el Wavedrom, donde el paquete de datos se va armando. Primero el idle_bit empieza en alto. La señal 01011011 de start_bit, indica que los datos en paralelo pueden ingresar. Pasado un tiempo se activa el parity_bit de ser necesario y justo después se activa el stop_bit. Lo que detiene el proceso con Idle en alto.

La señal de tx1, refleja el bus de datos recibido en parelelo, convirtiendo el dato a serial con la configuración de LSB, como se oberva, además se conectan esta señal con rx2, del módulo 2, como se ejemplifica en la imagen. Dicho comportamiento es ingresado en serial por el UART2 y desplazado a paralelo por su misma lógica. Al final del primer envío se obtiene el mismo dato enviado, en la señal de data_out, la cual está desempaquetada, es decir, sin los bits de paridad y demás.

Para la segunda iteración el comportamiento de los módulos es exactamente el mismo, a diferencia de que se envía un bus de datos distinto. Esta iteración empieza luego de aplicar un reset a la entrada del sistema, el cuál reinicia el proceso pero de modo inverso, es decir, el UART2 transfiriendo la señal al UART1, que como se observa igualmente, el dato ingresado, se empaqueta, se transmite mediante tx2 hacia rx1, y este último es tomado por el módulo uno, y lo desempaqueta, mostrando el dato en la segunda salida de data_out. En esta segunda iteracion el dato transmitido es "01000010".

6. Conclusiones

A lo largo del proyecto, se logró un entendimiento detallado del funcionamiento y características del protocolo UART. Se demostró su importancia en la comunicación serial asíncrona y cómo los bits de inicio, datos, paridad y parada permiten una transmisión eficiente entre dispositivos sin necesidad de una señal de reloj compartida. Fue posible simular tanto el transmisor como el

receptor de manera efectiva, garantizando la conversión de datos paralelos a seriales y viceversa. El uso de herramientas como GTKWave y el diseño de un Makefile eficiente facilitaron la visualización y análisis del comportamiento del sistema. Esto permitió verificar que las señales funcionaran conforme a las especificaciones del protocolo y que la simulación se ejecutara de manera automatizada, optimizando el tiempo de desarrollo. El uso de GitHub como repositorio compartido fue fundamental para el trabajo colaborativo. La constante revisión y optimización del código ayudaron a mantener la calidad del proyecto, permitiendo corregir errores y simplificar el diseño cuando fue necesario.

Los resultados obtenidos en GTKWave confirmaron que la implementación respeta el protocolo UART. Las señales y datos mostraron un comportamiento acorde a lo esperado, validando el diseño del sistema. Finalmente, el desarrollo del proyecto definitivamente nos permitió mejorar nuestros conocimientos en Verilog y en la implementación de protocolos de comunicación. Fue un proyecto difícil pero resultó en un aprendizaje significativo.