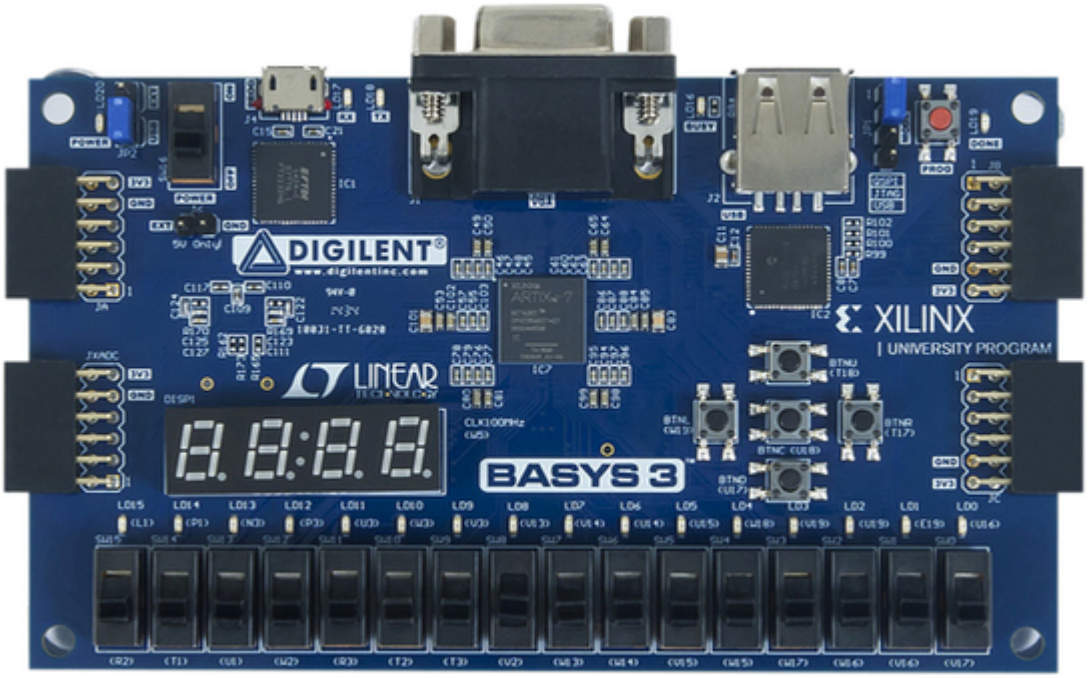
UNC Universidad Nacional de Córdoba

FCEFyN Facultad de Ciencias Exactas Físicas y Naturales



Trabajo Práctico Nº 2

UART: Universal Asynchronous Receiver Transmitter

Docentes:

- Ing. Santiago Rodriguez
- Ing. Martín Pereyra

Autores:

- Federico Joaquín Coronati
- Francisco Adrián De la Cruz

Arquitectura de Computadoras - Año 2021

Índice

Objetivo

Este trabajo práctico de laboratorio consiste en implementar de manera simple el envío y recepción de información a través del protocolo **UART (Universal Asynchronous Receiver Transmitter)**, el cual emplea una conexión serial, para luego utilizar esos datos como entrada al módulo **ALU (Arithmetic and Logic Unit)** desarrollado previamente de manera aislada como un módulo combinacional. De esta manera queda en evidencia que el trabajo es incremental ya que emplea los módulos construidos anteriormente.

Introducción

En primer lugar, se presenta un esquema de los módulos que forman parte de este laboratorio, para posteriormente pasar a detallar cada uno por separado.

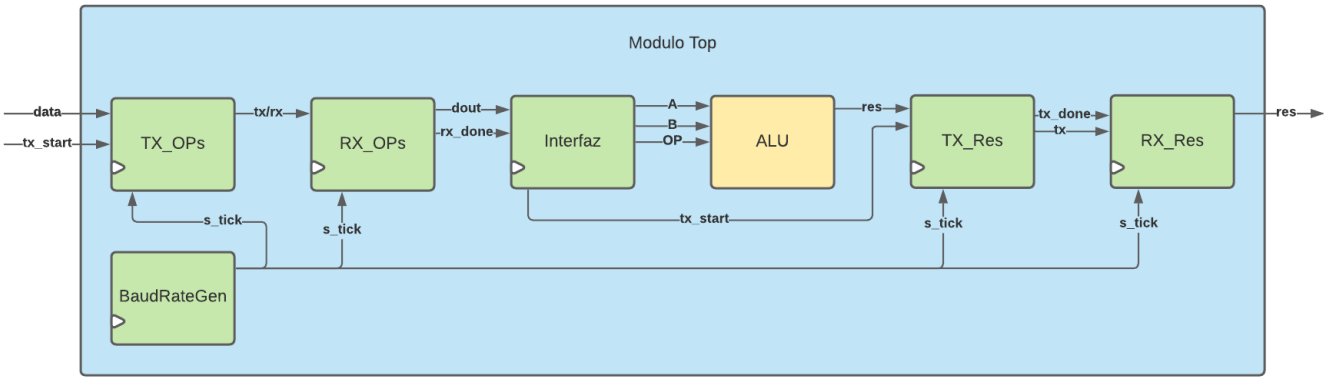


Figura 1: Esquema General de los Módulos de trabajo

Como se puede apreciar en la figura, se tienen 4 nuevos módulos, mientras que se reutiliza el módulo ALU del práctico anterior, motivo por el cual no se detalla este último sino que simplemente se observan sus entradas y salidas en el diagrama. A continuación se presenta una breve descripción de los demás módulos:

baudrategen.v: Es el Generador de Baud Rate, es decir se encarga de enviar una señal **"s_tick"** para que los módulos de recepción y transmisión (RX y TX respectivamente) tomen una muestra del estado del dato en ese instante. Para una señal de clock de 50 MHz, un requerimiento de 16 muestras por bit y un baud rate deseado de 19200 baudios, se requiere tomar una muestra cada 163 ciclos de clock, partiendo de la ecuación.

$$\frac{50x10^{6}}{19200 \cdot 16} \simeq 163\ ciclos\ para\ cada\ muestra$$

uart_rx.v: Es el Receptor de UART, el cual analiza la señal de entrada "rx" y la muestrea idealmente al centro. Presenta los bits de datos en un registro "dout" y una vez finalizada la recepción y verificada la paridad se indica con una bandera "rx_done" para que la interfaz pueda leer el dato confiable.

interface.v: La interfaz entre el módulo ALU y la transmisión y recepción de datos. Se espera que los datos recibidos sean los operandos de una operación aritmética o lógica, y el resultado obtenido se transmita también. Para esto, la interfaz recibe, ordena y presenta los datos adecuadamente. En caso de haber un error se solicitan todos los operandos nuevamente. En caso de correcto funcionamiento se indica mediante la señal "tx_start" que avisa al módulo tx que ya puede leer y transmitir el dato "res", es decir el resultado de la lógica combinacional del módulo ALU.

uart_tx.v: Es el Transmisor de UART, el cual se encarga de leer el resultado de la interfaz y comenzar a transmitir bit a bit, como corresponde según el protocolo. Esto se explica en detalle en el apartado de máquinas de estado.

uart_top.v: Es el módulo principal o Top, cuyo objetivo principal es instanciar e interconectar los módulos antes mencionados. Aquí se indican parámetros interesantes para el sistema UART.

Máquinas de Estado

Para este trabajo de laboratorio se tratará a alguno de los módulos como **Máquinas de Estados Finitos**, conocidas también por la sigla **FSM (Finite State Machine)**. El objetivo principal de las FSM es plantear un modelo matemático en el que se pueda plasmar el comportamiento del sistema dado un evento en específico.

En concreto, los módulos que serán construidos en base a una FSM son 3, el receptor, el transmisor y la interfaz. Para el caso del **transmisor y el receptor**, se usa una misma máquina de estados, la cual posee 5 estados y se ilustra en la figura a continuación.

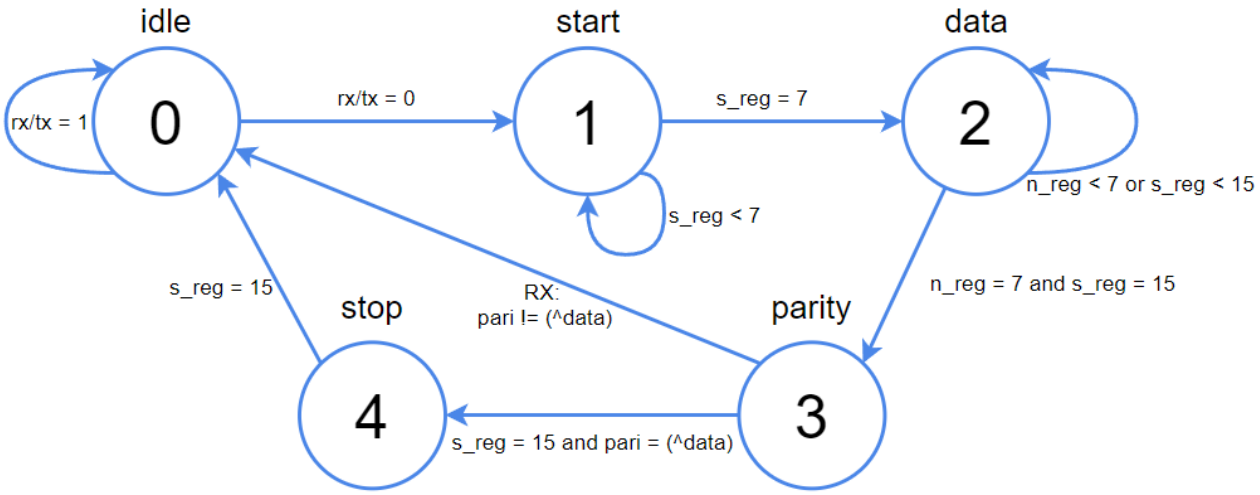


Figura 2: Diagrama de Estados de las FSM de RX y TX

Los estados que se observan en la figura consisten en:

0) **idle:** Inactividad, es decir nada se está transmitiendo o recibiendo.
1) **start:** Un bit en bajo que indica que viene o se manda una nueva trama de datos.
2) **data:** Bits de datos, habitualmente 8 bits.
3) **parity:** Un bit especial que indica la cantidad de bits seteados en “1” de los datos, sirve para detección de errores de transmisión.
4) **stop:** Un bit en alto que indica que finalizó la transmisión de esa trama.

Cabe destacar que tanto el transmisor como el receptor funcionan de manera análoga a excepción del estado de paridad. La diferencia sustancial es que el transmisor solamente debe calcular la paridad y enviar ese bit en el momento correspondiente mientras que el receptor debe recibir ese bit, calcular la paridad de la trama y verificar que coincidan, en caso de no coincidir se vuelve al estado de idle para recibir toda la trama nuevamente.

Para **calcular la paridad**, se emplea la lógica de **usar compuertas XOR** entre los bits de datos de la trama y verificar si la cantidad de bits en “1” son pares o impares. En caso de ser pares el bit de paridad será “0”, en caso de ser impares será “1”. Esta lógica se puede negar agregando otra compuerta XOR a la salida

pero se decidió trabajar de esta manera que resulta más intuitiva. La figura 3 muestra un esquema de lo explicado anteriormente.

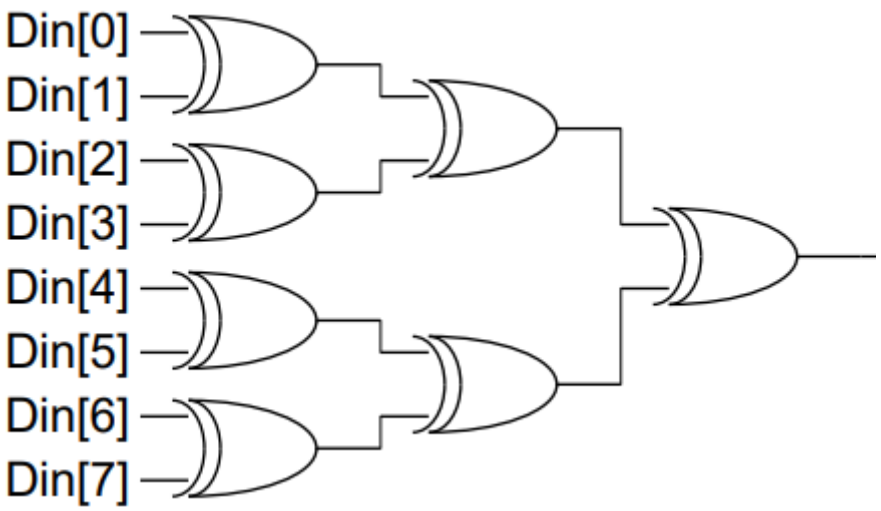


Figura 3: Compuertas XOR para calcular paridad

En cambio, **la máquina de estados de la interfaz** es mucho más simple. Consta de 3 estados, A, B y OP, en los que se recibe una trama de 8 bits para A, una vez se recibio correctamente se recibe B y luego OP. Una vez que se tienen los 3 datos necesarios para el módulo ALU, se envía la señal tx_start_next indicando que los 3 datos son confiables y que la salida del módulo ALU ya tiene el valor esperado, ya que, como la ALU es combinacional estará entregando el resultado con datos parciales y no será el esperado, hasta que los 3 datos A, B y OP hayan sido entregados correctamente.

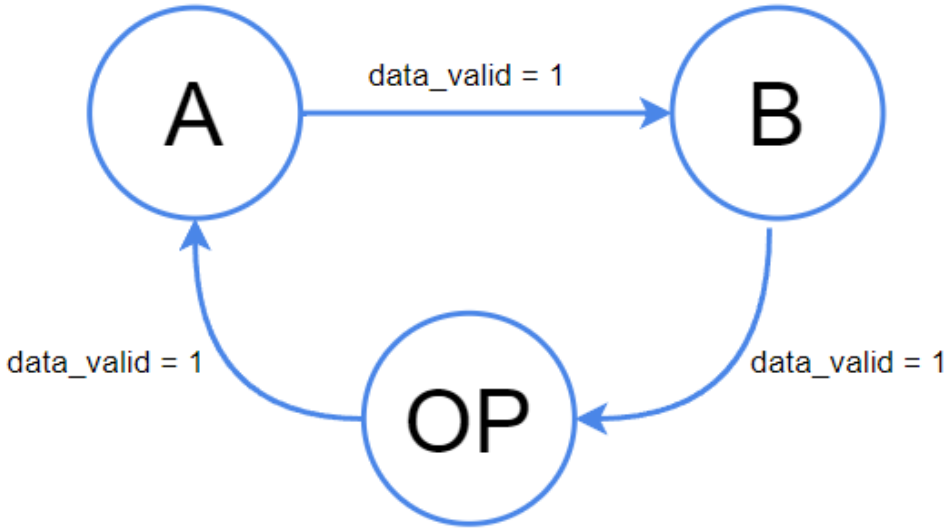


Figura 4: Diagrama de Estados de la FSM del módulo Interfaz

Análisis Temporal de las Señales

Una vez codificados los módulos, que pueden accederse a través del repositorio de Github referenciado al final de este documento, se desarrollan los **Testbenches** correspondientes a cada módulo para comprobar el correcto funcionamiento de los mismos dada una entrada deseada. Estos archivos se encuentran en una carpeta separada de nombre Testbenches.

Luego de tener cada módulo testeado por separado, llega el momento de reunirlos a todos con ayuda del módulo top, que se encarga de instanciar los módulos explicados en la primera parte de este artículo. Se aclara que se crean 2 instancias de los módulos transmisor y receptor, ya que se usa un TX y RX para el envío de los **operadores** a la ALU, y otro par TX y RX para el envío del **resultado**.

A continuación se muestra una captura de pantalla de la simulación de comportamiento, discriminando las señales involucradas en el sistema en diferentes colores, para poder explicarlas con mayor claridad.

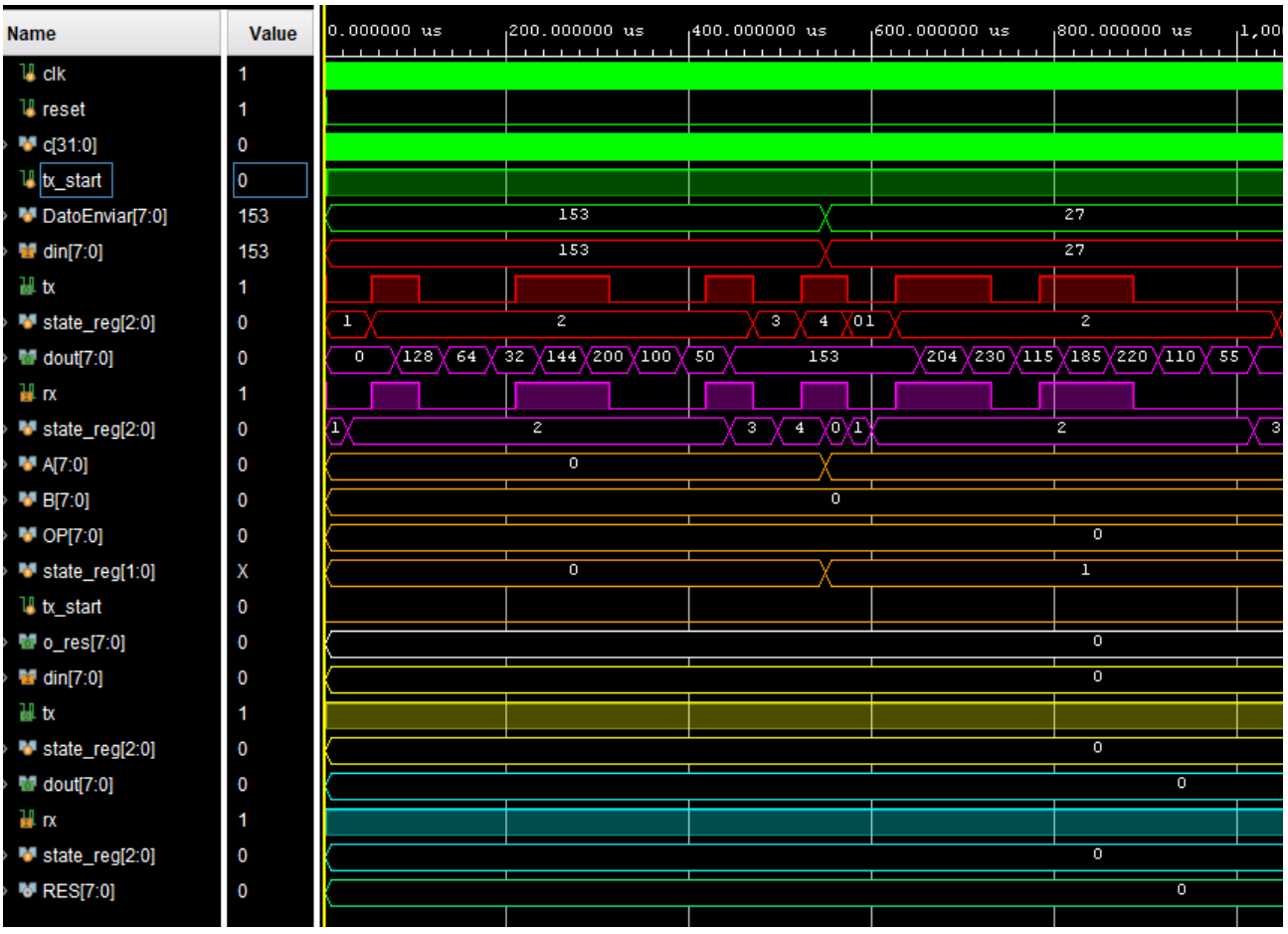


Figura 5: Simulación de todo el Sistema

Tanto esta como las demás figuras se incluirán en el **repositorio** para poder observarse con mayor claridad.

En la figura se puede apreciar que el dato que se desea enviar es el número 153 (para ser operando A), seguido del número 27 (operando B). Se recortó una

porción de la figura donde se envía el número 32 que corresponde al código de la operación suma.

El transmisor de operandos se muestra en color rojo, mientras que el receptor de los operandos se muestra en color magenta. Se puede ver además el state_reg, que contiene el número del estado actual según el color antes mencionado.

Para el resultado, el transmisor se muestra en color amarillo y el receptor en color celeste, pero la parte de transmisión del resultado quedó fuera de la figura anterior, por lo tanto se muestra a continuación.

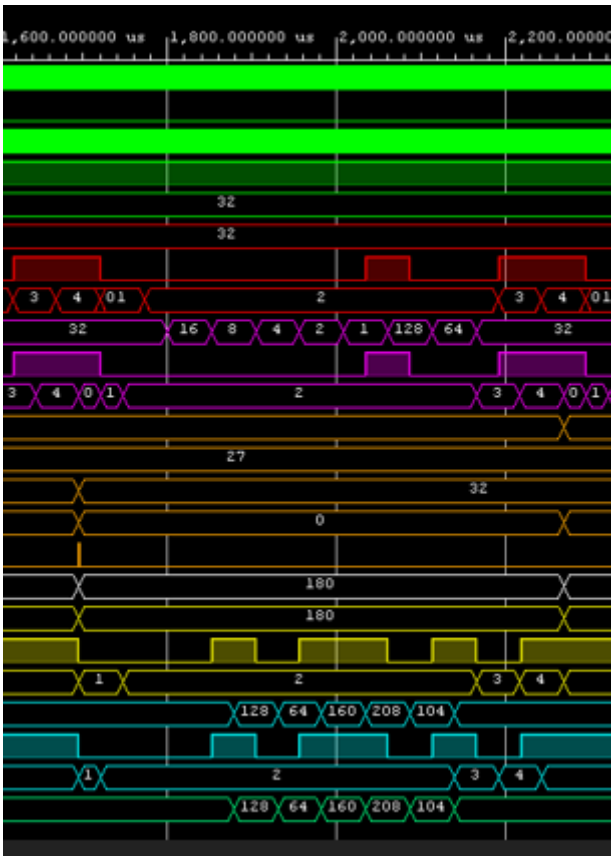


Figura 6: Continuación de la figura 5

En esta figura se puede ver con claridad el momento donde se envía el resultado 180 (en color amarillo) y el receptor del resultado en color celeste.

Conclusiones

Este fue un trabajo incremental ya que integró el módulo diseñado en el trabajo anterior. El módulo ALU es un módulo puramente combinacional es decir que no necesita señal de clock y el tiempo que demora en entregar el resultado es únicamente la demora en actuar de las compuertas internas, de modo que el resultado está listo prácticamente al instante.

En cambio, los demás módulos si necesitan ser temporizados, pero es importante tener en cuenta que el protocolo UART no transmite una señal de clock, sino que se temporiza de manera indirecta gracias al generador de baud rate que indica a los módulos UART cuando deben tomar una muestra y actuar en consecuencia.

El modelo de las máquinas de estado permite simplificar el diseño de estos sistemas ya que se puede ver claramente en los diagramas cuales son las condiciones de cambio de estado y luego llevar esto a la práctica.

Referencias

Repositorio de Github: https://github.com/FranciscoCross/TP2-Arquitectura

“FPGA Prototyping By Verilog Examples - Pong P. Chu”, del cual se basa la implementación.