Universidad Nacional de Córdoba.

Facultad de Ciencias Exactas, Físicas y Naturales.

ARQUITECTURA DE COMPUTADORAS

Trabajo Práctico Nº 2

UART

Alumnos

- Perez, Esteban Andres ... 39026980
- Carrizo, Facundo ... 39419552

Docentes

- Rodriguez, Santiago
- Pereyra, Martin

OBJETIVO

En este trabajo práctico se nos solicitó desarrollar un sistema de comunicación UART que reciba instrucciones para la ejecución de operaciones en la ALU que desarrollamos en el TP1.

Requerimientos:

- Desarrollar una interfaz que comunique el transmisor y el receptor con la ALU.
- Validar el desarrollo por medio de Test Bench

DESARROLLO

En primer lugar se diseñó el diagrama con los bloques intervinientes para la implementación de la UART conectada con la ALU con el objetivo de tener claro de antemano que entradas y que salidas van a estar implicados en cada módulo para permitirnos desarrollar cada uno de forma eficiente y ordenada.

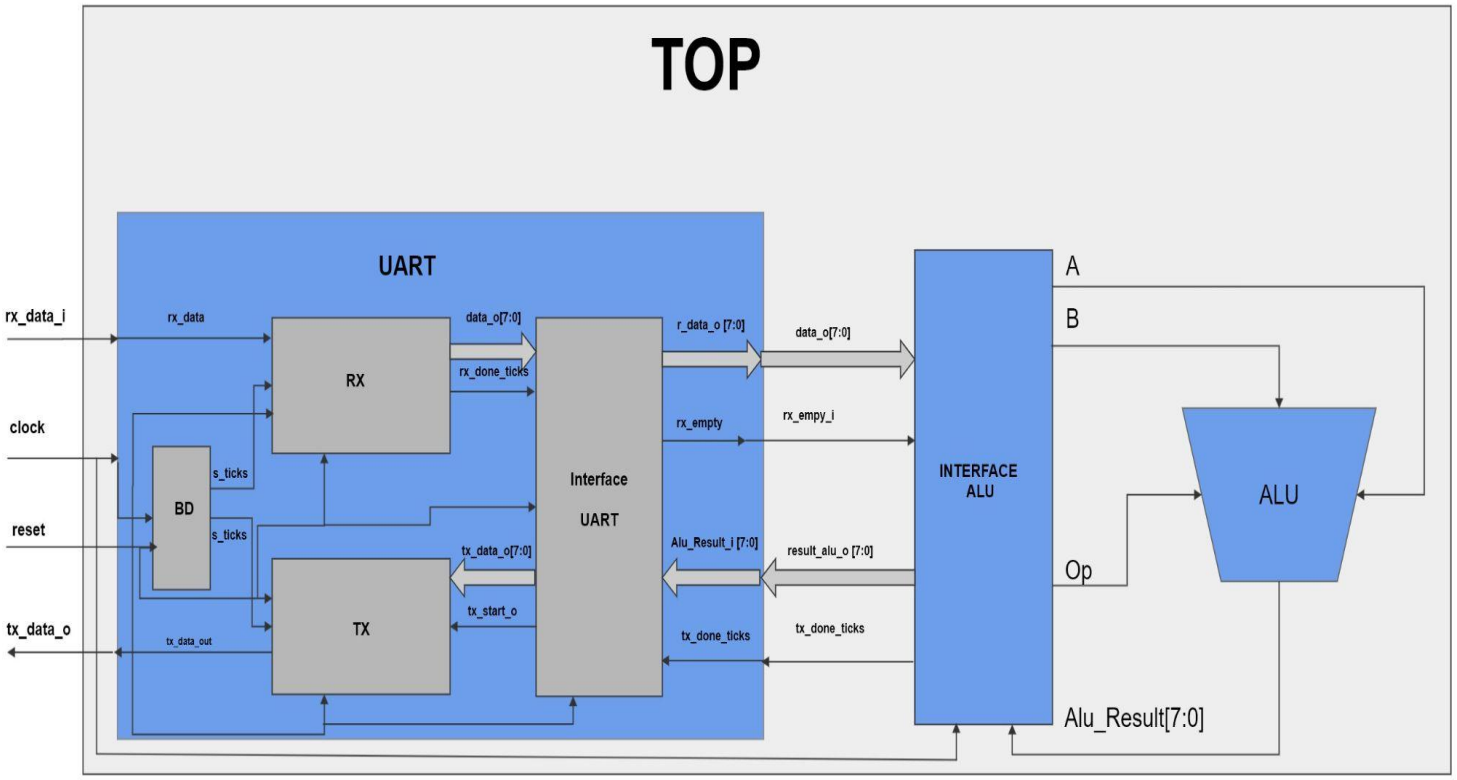


Fig.1 - Modulo TOP (UART + ALU + Interfaz)

Como se puede apreciar en la fig-1, se tienen 5 nuevos módulos, mientras que se reutiliza el módulo ALU implementado anteriormente.

A continuación se presenta una breve descripción de los módulos:

- BD: BaudRate generator

  Se encarga de generar señales “Ticks” para que el Tx y el Rx como señal de sincronización.

Se generan ticks cada N ciclos de clock. En la fórmula de abajo se calcula cada cuantos ciclos se genera el ticks.

$$Ncont = \frac{CLK}{(16 \times BaudRate)}$$

Por ejemplo:

- Para una frecuencia de 50 Mhz
- Con un baudrate de 19200

Tenemos que contar 163 ciclos de clock para que el módulo BD genere un ticks enviado a los módulos rx y tx.

- rx: Receptor

Es el módulo receptor de la UART, encargado de recibir los bits provenientes del transmisor tx y de guardarlos en un registro para su posterior envío a la interfaz de la uart, siempre que la lectura del bit de stop esté en alto.
También envía un bit de flag “rx_done_ticks” cuando se ha leído el bit de stop correctamente para que la interfaz de la uart lo registre.

- tx: Transmisor

Es el módulo de transmisor de la UART, se encarga de enviar los bits en serie de los datos recibidos en paralelo.

- Interface UART:

Este módulo se encarga de ofrecer una interfaz estándar entre los módulos de transmisión y receptor con la interfaz de la ALU.

- Interface Alu: Este módulo recibe los datos provenientes de la interfaz de la uart y los envía a la alu y recibe el resultado de la ALU para enviarlo a la interfaz de la uart.

También envía un bit de flag “tx_done_ticks” para avisarle a la uart que ya está el resultado de la ALU para que el transmisor lo empiece a transmitir.

MÁQUINAS DE ESTADOS

Para el desarrollo del transmisor, receptor y de la interfaz de la ALU se utilizó el concepto de máquina de estado finito (FSM).

Las máquinas de estado nos permite visualizar de forma gráfica el comportamiento de los sistemas digitales (en nuestro caso los módulos) de acuerdo a los eventos de entrada y con su correspondiente estado en el momento dado.
A continuación se muestran los diagramas de estado tanto del tx,rx como de la interfaz de la ALU.

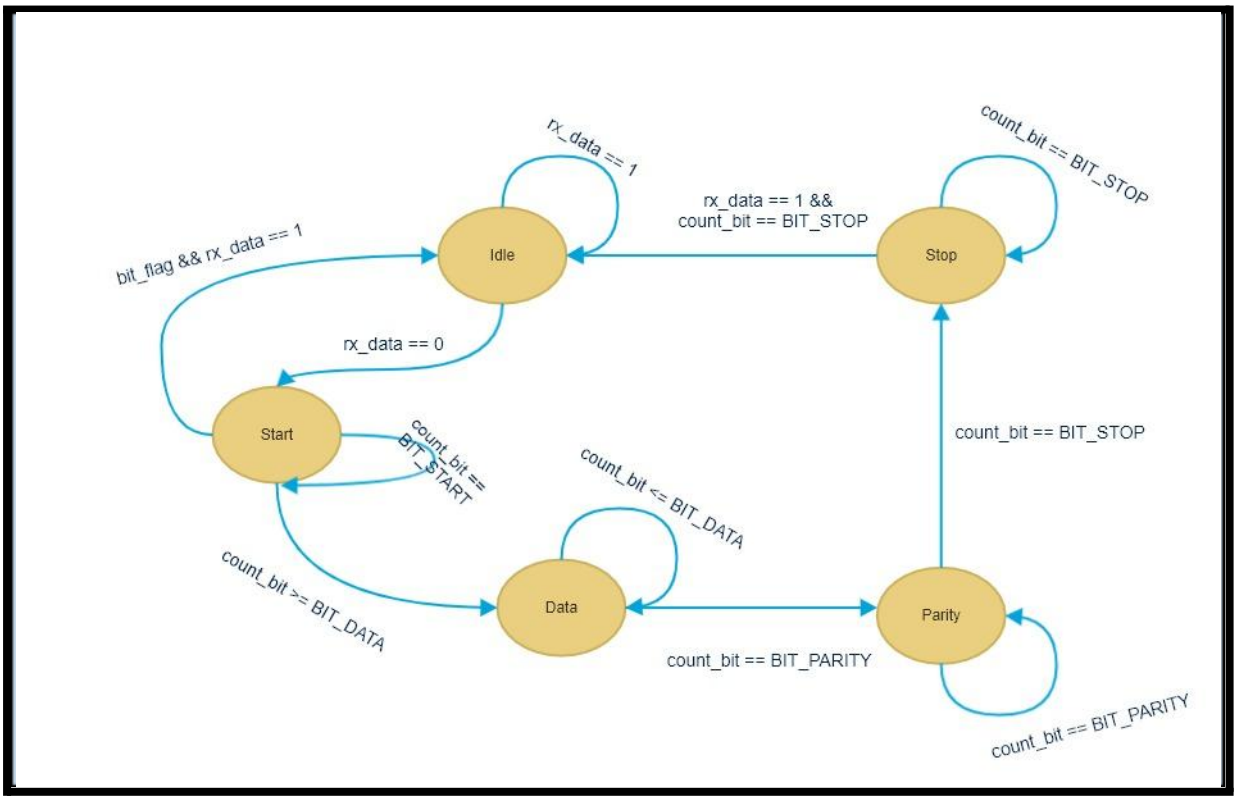


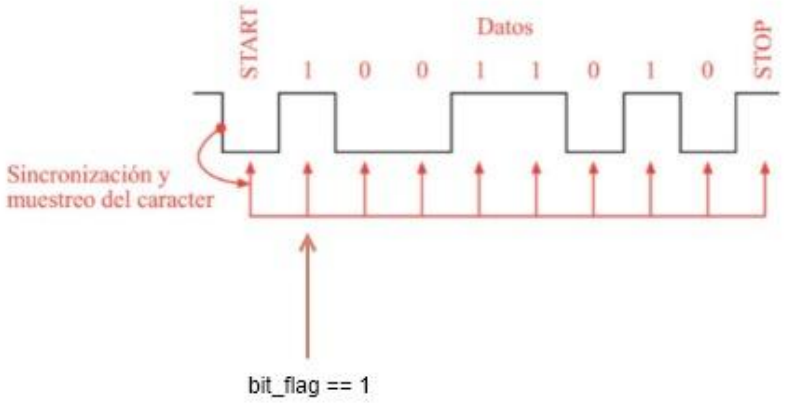
Fig.2 - FSM del receptor (rx)

En la fig-2 se puede ver el diagrama con los estados intervinientes y el evento de entrada en el cual va a cambiar de estado en el momento correspondiente.

Estados

- Idle: Se espera recibir el bit de inicio, es decir cuando rx esté en bajo.
- Start: Este estado nos permite que pasados 7 ticks verificar si realmente hubo un bit de inicio.
- Data: En este estado se reciben los bit de datos uno a uno de forma serial y se lo almacena en un registro.
- Parity: En este estado se verifica si hubo paridad o no para cumplir con el protocolo uart
- Stop: Se espera el bit de stop para validar si la trama recibida es correcta o no.

Lectura de bits



El bit_flag nos marca el tiempo de sampleo de todos los bits (cada 7 ticks en el bit correspondiente), como también el chequeo del bit de inicio y de stop.

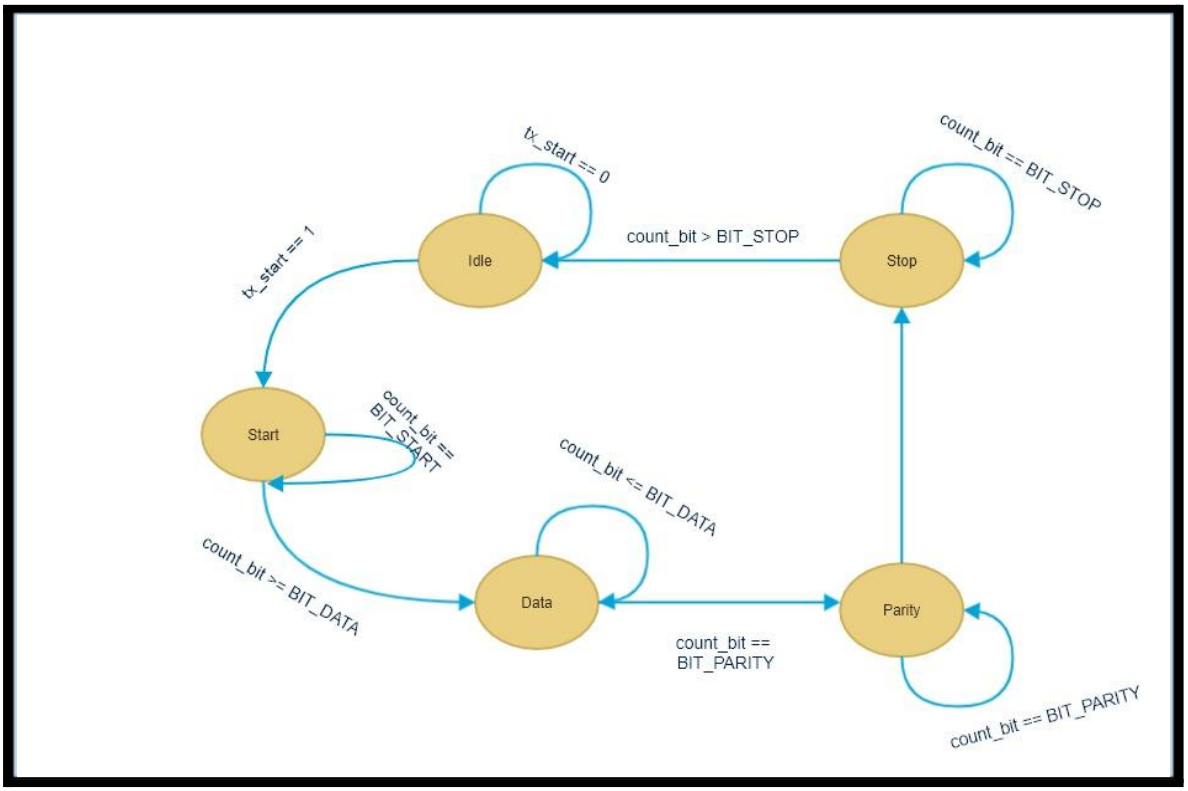


Fig.3 - FSM del transmisor (tx)

En la fig.3 se aprecia la máquina de estado del transmisor, es idéntico al del receptor explicado anteriormente solo cambia los eventos.

Cuando haya un start de entrada se guarda el valor a transmitir en un registro auxiliar y se envía el bit de inicio, después los bits de datos, el de paridad y por último el bit de stop. Cabe destacar que cada bit de envío dura 16 ticks.

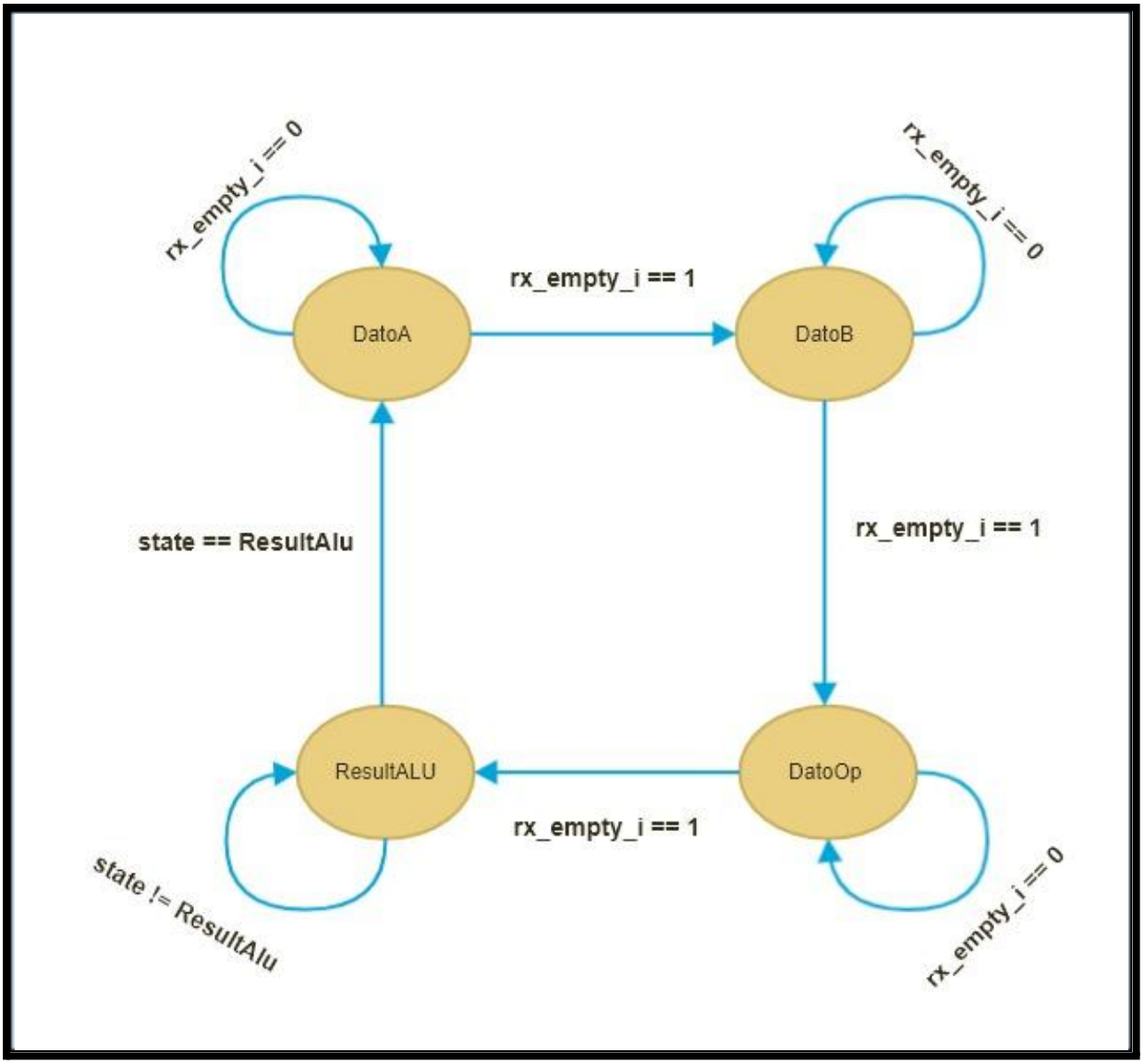


Fig.4 - FSM IntALU

En la fig.4, representa la máquina de estado de la interfaz de la ALU, en el cual consiste de 4 estados.

Estados

- DatoA: se espera que se ingrese el operando A.
- DatoB: se espera que se ingrese el operando B.
- DatoOp: se espera que se ingrese la operación a realizar.
- ResultALU: En este estado se espera el resultado de la operación en la ALU para su posterior envío al módulo uart para ser transmitido.

SIMULACIÓN

Para llevar a cabo la simulación de la uart con la alu se instancio en el test bench el módulo TOP (implementado en la fpga) y también se instancio una uart auxiliar que se va a encargar de enviar los bits al módulo TOP.

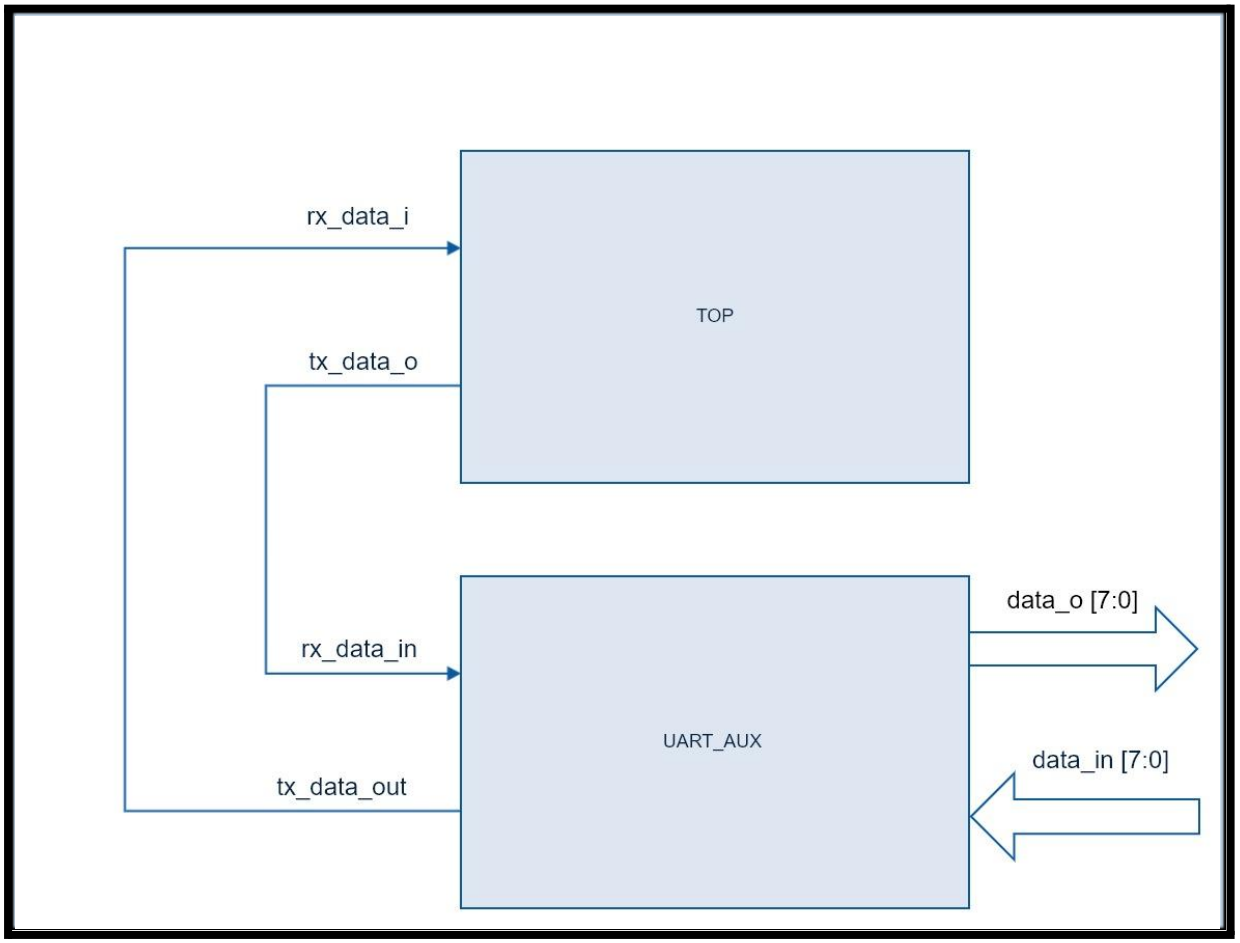


Fig.5. TOP con uart aux

En la uart auxiliar se coloca el dato a enviar por el puerto data_in, y la salida a la entrada del módulo TOP.
Después la salida del TOP va conectada a la entrada de la uart auxiliar (al módulo rx) para recibir el resultado de la ALU.
Por último verificamos el resultado de la ALU en el puerto data_o.

A continuación se muestran capturas de pantalla de la simulación behavior y post-síntesis.

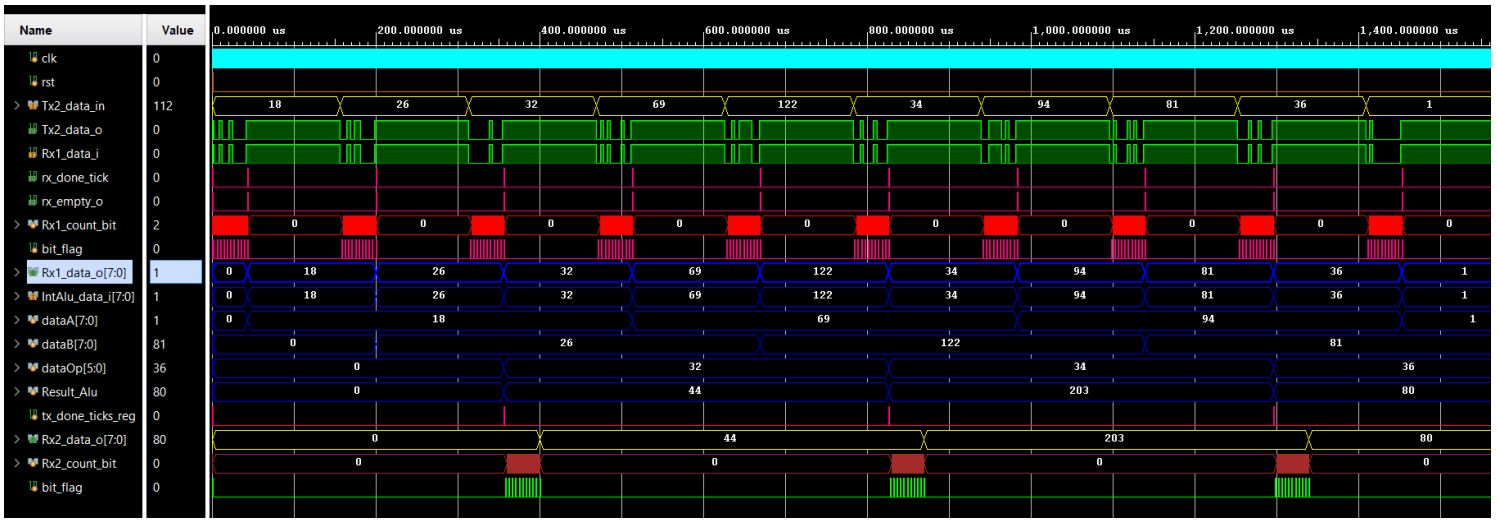


Fig.6. simulación behavior.

En la figura anterior se puede observar:

- El dato A (18), dato B (26) y la operación de la suma (32). Y el resultado 44.
- Dato A (69), Dato B (122) y la operación de la resta (34). Y el resultado 203.

Los datos A y B se generan de forma random.

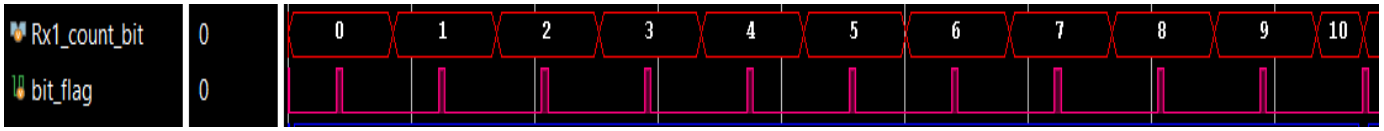


Fig.7. bit de sample

En la fig.5 se ve claramente cuando se lee el bit de la trama, en el momento de que bit flag está en alto. Esto se da a la mitad del bit lo cual corresponde a 7 ticks como se mencionó anteriormente.

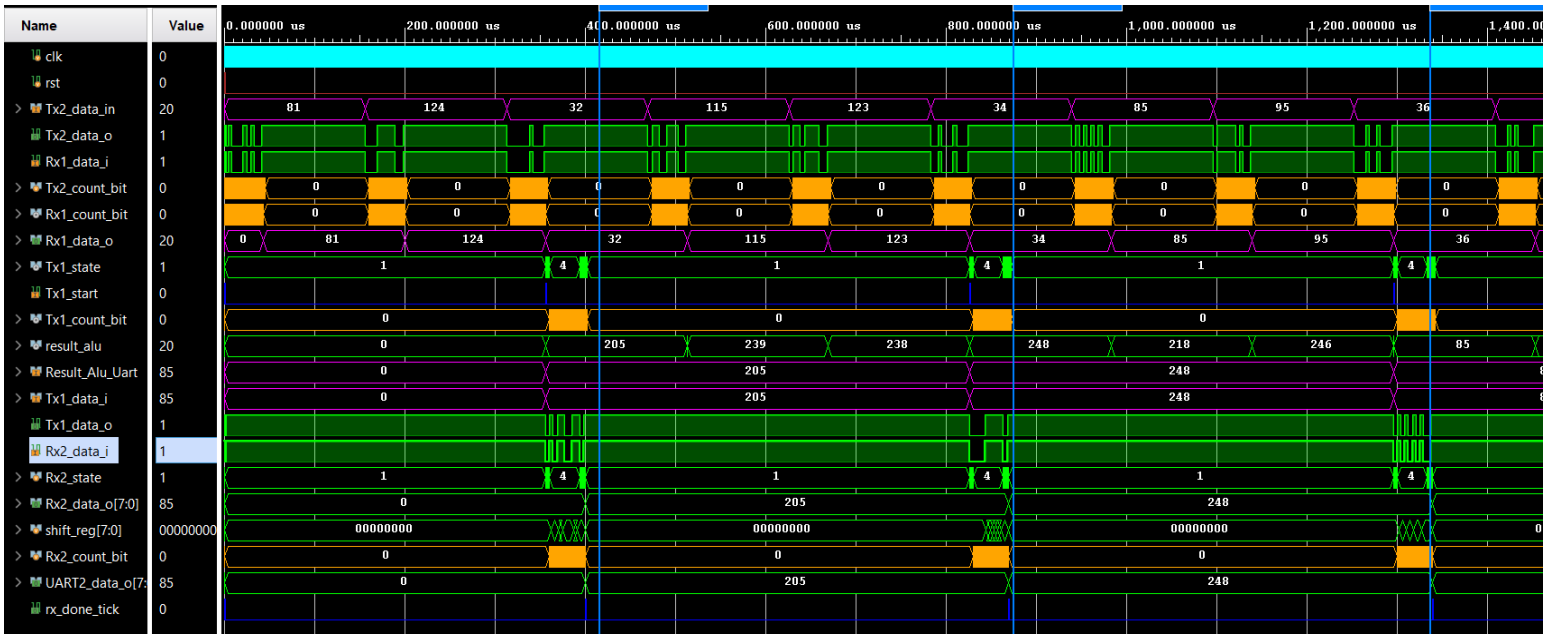


Fig.7. simulación post-síntesis.

Resultados:

1.
   - Dato A: 81
   - Dato B: 124
   - Operación: 32
   - Resultado: 205
2.
   - Dato A: 115
   - Dato B: 123
   - Operación: 34
   - Resultado: 248
3.
   - Dato A: 85
   - Dato B: 95
   - Operación: 36
   - Resultado: 85

CONCLUSIÓN

Este trabajo nos permitió desarrollar máquinas de estado en verilog y comprender mejor el funcionamiento de la comunicación serial. Pudimos obtener el resultado esperado en ambas simulaciones tanto behavior y post-síntesis.

La síntesis nos permitió comprender aún más el lenguaje verilog en tanto que los bloques combinacionales NUNCA se debe trabajar con registros. SOLO se debe cambiar de estado de acuerdo al evento de entrada y al estado en ese momento.

Como conclusión final en un futuro llevaremos a cabo la simulación en un FPGA de manera física, para poder observar las diferencias que pueden llegar a existir.