UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

ARQUITECTURA DE COMPUTADORAS

TRABAJO PRÁCTICO N° 2 UART

Apellido	Nombre	Matrícula
Guglielmotti	Bruno	43474558
Rodríguez	Franco Aníbal	42994188

Índice

1.Introducción..... 1
2.Desarrollo..... 1
2.1.UART.....1
2.1.1 UART Baud Rate Generator.....2
2.1.2 UART Receiver.....4
2.1.3 UART Transmitter.....5
2.2 Interfaz..... 6

1.Introducción

En este laboratorio se hará el desarrollo de un código en Verilog con el fin de sintetizar una FPGA Digilent Basys3 para que implemente una Unidad Aritmética Lógica (ALU) similar a la del trabajo pasado. Esta vez se implementará un Universal Asynchronous and Transmitter (UART) junto a un baud rate generator necesario para llevar la cuenta de los datos, estos luego pasarán a una interfaz que los llevarán a la ALU desarrollada para hacer el cálculo, y el resultado entregado por la ALU será transmitido de nuevo por el UART.

2.Desarrollo

2.1.UART

UART es un protocolo de comunicación asíncrono que no utiliza una señal de reloj compartida entre el transmisor y el receptor para sincronizar los datos, en su lugar, acuerdan una velocidad de transmisión denominada baud rate que define la cantidad de bits transmitidos por segundo. La comunicación se realiza en serie, lo que implica que los bits se envían uno tras otro a través de un único cable de transmisión.

Componentes Principales del Protocolo UART

11. **Transmisor (TX)**: Convierte los datos paralelos en bits seriales y los envía por la línea de transmisión.
22. **Receptor (RX)**: Recibe los bits seriales y los convierte nuevamente en datos paralelos.
33. **Línea de transmisión (TX Line)**: Canal unidireccional que transporta los datos del transmisor al receptor.
44. **Línea de recepción (RX Line)**: Canal unidireccional que transporta los datos del receptor al transmisor.
55. **Bits de control**:
    - **Bit de inicio**: Señaliza el comienzo de una transmisión. Generalmente siempre la línea está en alto y un bit bajo indica el inicio.
    - **Bits de datos**: Usualmente los datos se envían en bloques de 7, 8 o 9 bits.
    - **Bit de paridad**: Opcional, utilizado para verificar la integridad de los datos (control de errores).
    - **Bit de parada**: Marca el final de un bloque de datos a través de uno o dos bits en estado alto.

El transmisor es esencialmente un shift register que carga los datos en paralelos y luego los va lanzando uno a uno a una determinada tasa de envío. Por otro lado, el receptor recibe los bits uno a uno y los reensambla de nuevo.

Como no hay información del clock involucrada, antes de que la comunicación comience el transmisor y el receptor deben ponerse de acuerdo en una serie de parámetros tales como el baud rate, el número de bits de datos y stop, así como también parity bit.

2.1.1 UART Baud Rate Generator

El baud rate generator se encarga de generar una señal de sincronización para controlar la tasa de transmisión y recepción de los bits. Produce una señal de “tick” que determina cuándo se debe muestrear el estado de la línea de transmisión o cuándo debe enviar el próximo bit. La frecuencia de este tick debe coincidir con la tasa de baudios acordada entre los dispositivos de comunicación.

Necesita tomar la señal de reloj del sistema y dividirla para generar una frecuencia de muestreo que coincida con la tasa deseada, en este casó se usó 9600 baudios por lo que en la fórmula queda de la siguiente manera.

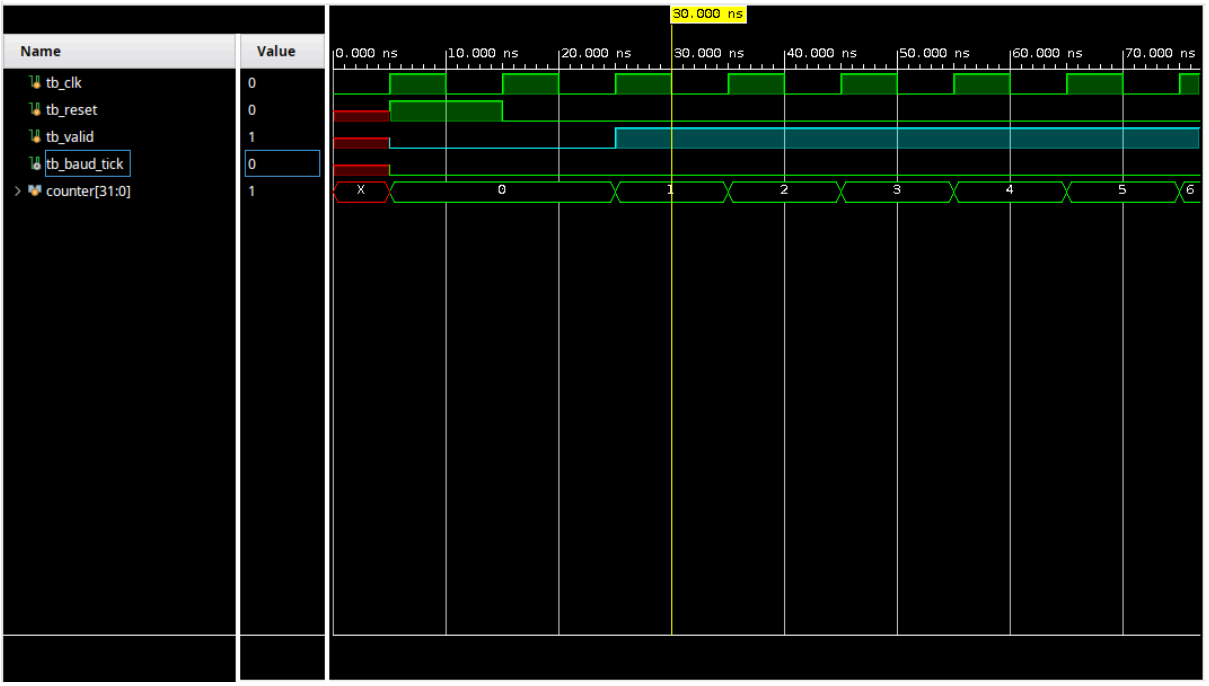
$$\frac{Clock}{BaudRate * 16} = \frac{100\ Mhz}{9600 * 16} \approx 651$$

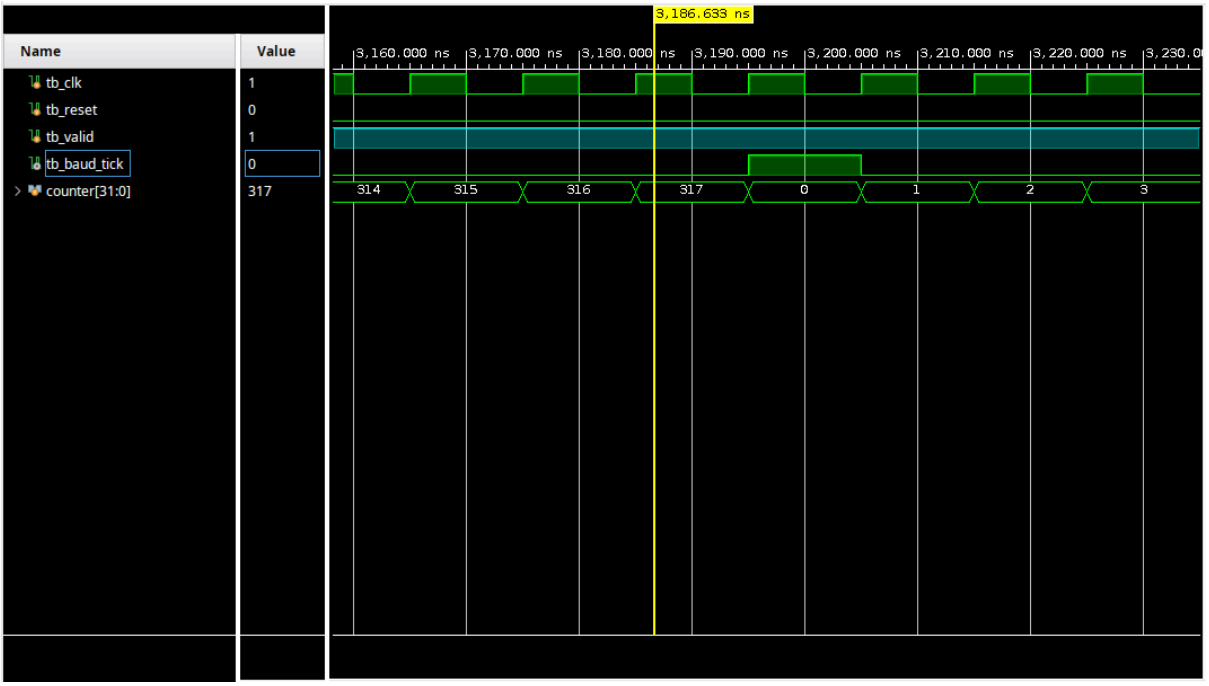
El factor **16** en el baud rate generator es una técnica de oversampling que mejora la precisión y fiabilidad en la recepción de datos UART. Muestreando a una velocidad 16 veces mayor que el baud rate, el sistema UART puede detectar los bits con mayor precisión, lidiar mejor con pequeñas desincronizaciones, y mejorar la estabilidad del sistema frente a variaciones en la velocidad de transmisión.

```
// Baud rate generation
always @(posedge i_clk) begin

   if (i_reset) begin

       counter <= 0;
       o_baud_tick <= 0;

   end else if (i_valid) begin

       if (counter == divisor - 1) begin

           o_baud_tick <= 1; // Generate tick
           counter <= 0;     // Reset counter

       end else begin

           o_baud_tick <= 0;
           counter <= counter + 1;

       end

   end else begin

       o_baud_tick <= 0;
       counter <= 0;

   end
end
```

El módulo básicamente depende de las señales ***i_valid*** y ***i_reset***, la segunda se encarga de llevar el sistema a un estado conocido, mientras que la primera sirve para poner en marcha el contador, el cuál se incrementa siempre y cuándo no alcance el valor de ***divisor***, una vez que lo alcanza se envía el tick por la salida del módulo hacia el transmisor y receptor.

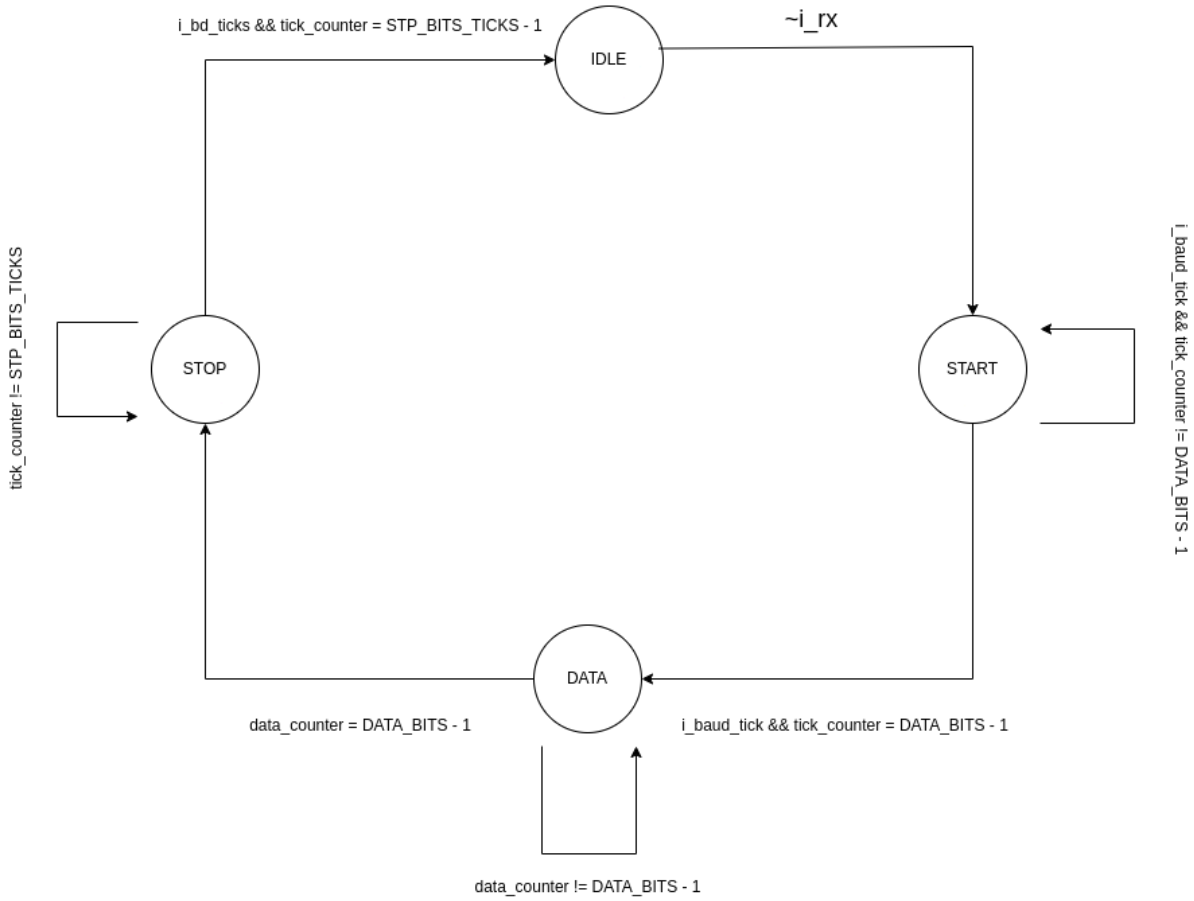
En en testbench correspondiente se puede ver este comportamiento:





2.1.2 UART Receiver

El receptor de UART consiste en la siguiente máquina de estados:



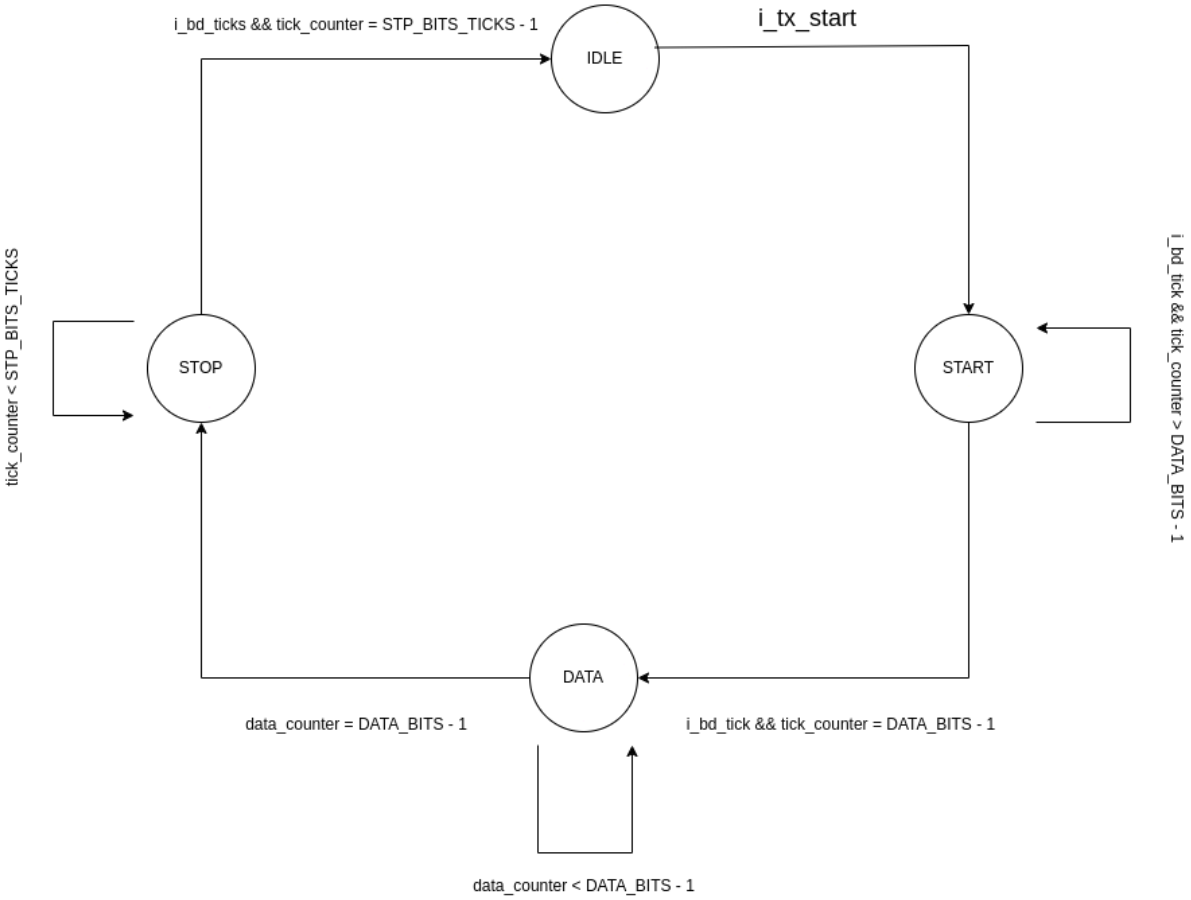
donde:

- *i_baud_tick*: el tick generado por el baud rate generator.
- *i_rx*: la señal de recepción.
- *tick_counter*: el contador de ticks que en el estado START tiene que contar 8 ticks (la mitad de un bit) para empezar a marcar la frecuencia de lectura de los bits cada 16 ticks (esto se hace a partir de que entra en el estado DATA).
- *data_counter*: el contador de bits del dato.
- *DATA_BITS*: señala la cantidad de bits del dato, en este caso, 8.
- *STP_BITS_TICKS*: es la cantidad de ticks necesarios entre la lectura de un bit y otro, en este caso, 16.

Una vez que se terminó de recibir el dato y la máquina de estados pasa del estado STOP al estado IDLE de nuevo, una flag *o_rx_done* se pone en uno para avisarle a la interfaz que el dato ya está listo para su procesamiento.

2.1.3 UART Transmitter

El transmisor de UART sigue una lógica muy similar la receptor, siguiendo este diagrama de estados:

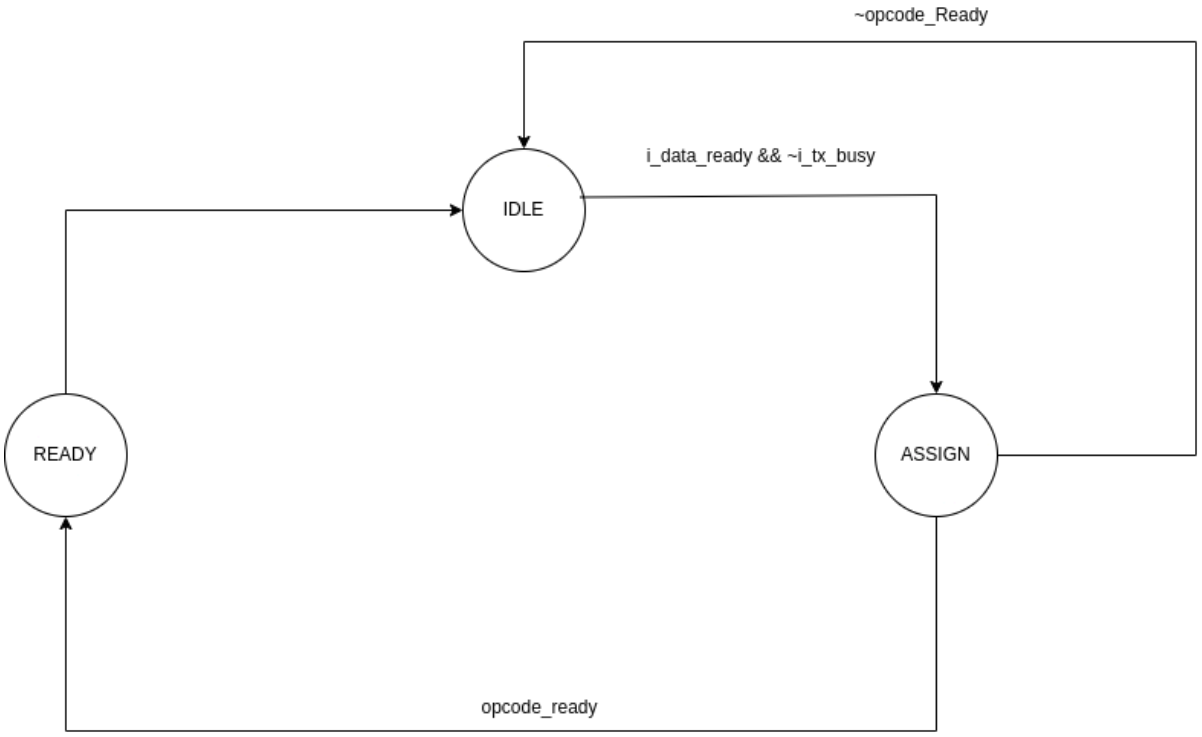


donde los registros tienen el mismo nombre y cumplen las mismas funciones que en el receptor, excepto por *i_tx_start* que es la señal recibida desde la interfaz para indicarle al transmisor que empiece a transmitir el dato que le envía, y por otro lado, cuando está transmitiendo, pone en 1 la salida *o_tx_transmiting* para indicarle a la interfaz que si recibe datos del receptor, no los procese.

2.2 Interfaz

La interfaz tiene como objetivo procesar los datos recibidos desde el receptor para luego enviarselos a la ALU y también indicar al transmisor que cuando estén los datos listos, en el siguiente ciclo de clock transmita lo que esté a la salida de la ALU.

Sigue este diagrama de estados:



donde:

- *i_data_ready*: la flag enviada por el receptor para avisar que terminó de recibir un dato entero y que el mismo ya se puede leer.
- *i_tx_bussy*: flag que se recibe desde el transmisor indicando si el mismo está ocupado transmitiendo o no, si es así la interfaz no procesa ni asigna ningún dato que le llegue del receptor.
- *opcode_ready*: flag interna que indica cuando el dato que corresponde al operador está listo, esto significa, según la lógica implementada, que los 3 datos están listos para ser procesados por la ALU.

La lógica implementada para la asignación de los datos consiste en flags internas que van indicando qué dato está listo y cual no basándose en el orden en el que estos llegan:

```
// Sequential block
ASSIGN: begin
// Assign operands and opcode based on incoming data bits
  if (~operand1_ready) begin
    o_operand1 <= i_data;
    operand1_ready <= 1'b1;
  end else if (operand1_ready && ~operand2_ready) begin
    o_operand2 <= i_data;
    operand2_ready <= 1'b1;
  end else if (operand2_ready) begin
    o_opcode <= i_data[NB_OP-1:0];
```

```
    opcode_ready <= 1'b1;
  end
end

// Combinational block
ASSIGN: begin
  if (opcode_ready) begin
    next_state = READY;
  end else begin
    next_state = IDLE;
  end
end
```

Entonces cuando los 3 datos ya fueron recibidos y asignados a la ALU, la interfaz le avisa mediante *o_data_ready* al transmisor, que transmita lo que está a la salida de la ALU.