# Laboratorio 07: Transmisor serial asíncrono

Por:
Ian Gabriel Cañas Fernández, 1092228

.
.

**Transmisor serial asíncrono**

Transfiriendo bytes desde la FPGA hasta una terminal.

**Objetivos:**

- Diseñar un sistema de transmisión asíncrona de datos seriales.

**Procedimiento:**

En el presente laboratorio estaremos utilizando el puerto UART de la tarjeta de desarrollo para la transmisión de datos mediante una forma serial en el momento solicitado. Se estará utilizando una máquina de estados para ello y reconociendo la introducción de los datos. Estaremos trabajando con la codificación ASCII de 8 bits para los caracteres introducidos.

## El transmisor serial asíncrono

Empezamos generando los puertos, funciones y señales necesarias para el laboratorio:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity Lab07 is
    port (
            clk : in std_logic; -- reloj
            reset : in std_logic; -- reset asincrono, activo high

            pdata : in std_logic_vector(7 downto 0); -- datos en paralelo (dato a enviar)
            load : in std_logic; -- cargar TBR

            txd : out std_logic; -- salida de transmision de datos
            empty : out std_logic); -- TBR vacio, active high
end Lab07;
```

```
architecture Behavioral of Lab07 is
  signal clk_state : std_logic := '0';
  signal clk_counter : integer range 0 to 5208;

  signal tx_bit : integer range 0 to 7;

  type tx_state_t is (IDLE, START, DATA, STOP); -- Estados: inactivo, inicia, datos, para
  signal tx_state : tx_state_t;

  signal char_tx : std_logic_vector(7 downto 0);

  signal wait_50 : integer range 0 to 50;

begin

char_tx <= pdata;
```

Generamos, entonces, una función que genera la frecuencia deseada que va a representar la velocidad de transmisión de datos, mediante el uso de la siguiente ecuación:

$$f_2 = \frac{100 \times 10^6\ Hz}{2 \cdot 9600\ baud} \approx 5208.33\ veces/grado$$

```
-------------------------------------------------------
-- Divisor de reloj de 100Mhz a 9600 bps
-------------------------------------------------------
process(clk, load)
begin
  if (load = '0') then
    clk_state <= '0';
  elsif(rising_edge(clk)) then
        if clk_counter = 5208 then
            clk_state <= not clk_state;
            clk_counter <= 0;
        else
            clk_counter <= clk_counter + 1;
        end if;
  end if;
 end process;
```

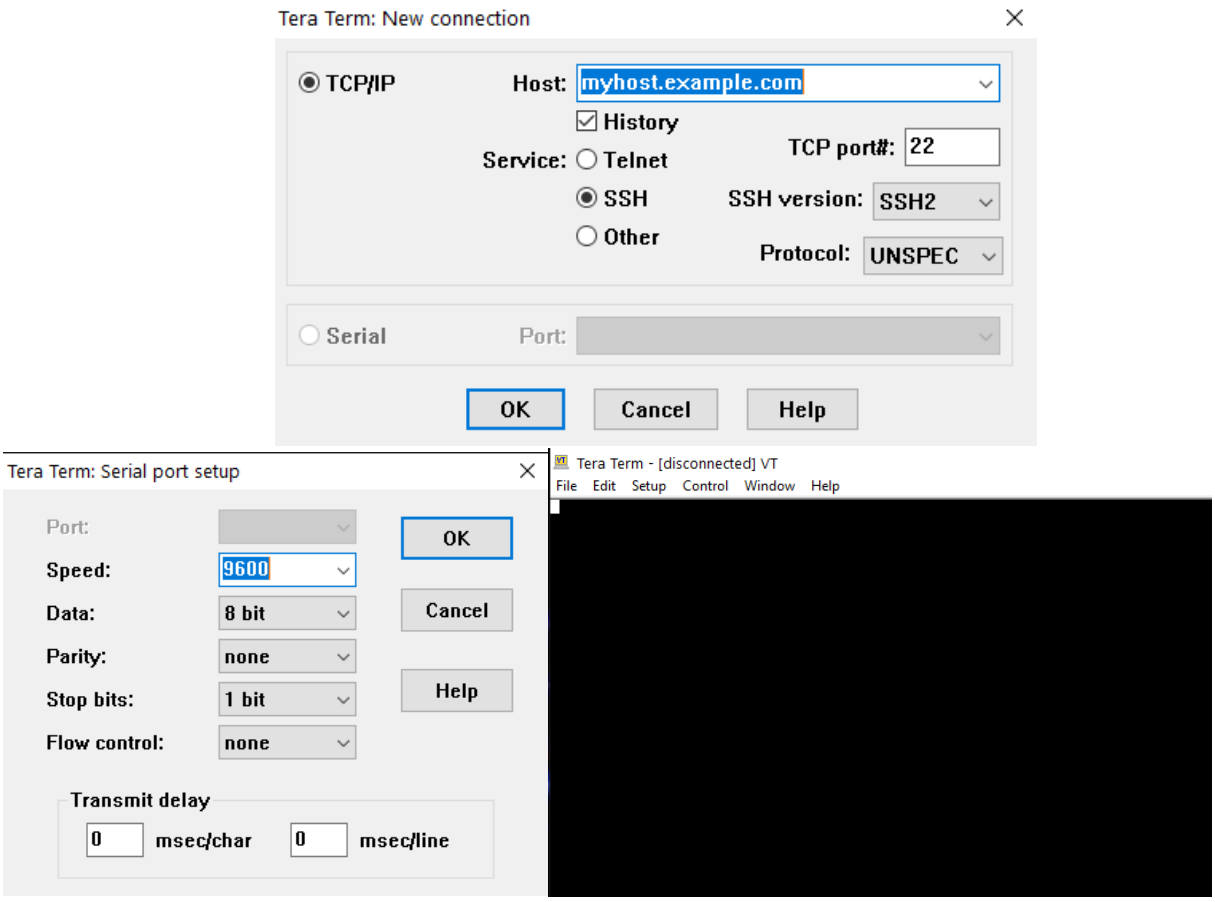
Obsérvese que no siempre se estará generando dicha frecuencia, sino que estamos dependiendo de que se esté pulsando el botón que active la señal de carga (load), pues en dicho caso se activa el siguiente proceso.

```
-----------------------------------------------------
-- Máquina de estados que forma la trama
-----------------------------------------------------
 process(clk_state, reset)
 begin
    if (reset = '1') then
        txd <= '1';
        tx_state <= IDLE;
        tx_bit <= 0;
        empty <= '1';
--        char_pointer <= 0;
        wait_50 <= 0;

    elsif(rising_edge(clk_state)) then
        case(tx_state) is
          when IDLE =>
            txd <= '1';
            tx_bit <= 0;

            if(wait_50 = 50) then
              tx_state <= START;
              wait_50 <= 0;
            else
              wait_50 <= wait_50 + 1;
            end if;

          when START => -- Space = 0; Mark = 1
            txd <= '0';
            tx_state <= DATA;

          when DATA => -- Carga byte (TSR)
            txd <= char_tx(tx_bit);

            if(tx_bit = 7) then
              tx_state <= STOP;
            else
              tx_bit <= tx_bit + 1;
            end if;


          when STOP => -- Space = 1; Mark = 1
            txd <= '1';
--            tx_state <= IDLE;
            empty <= '0';
        end case;
    end if;
end process;
```

Como se puede observar, mientras no se active la señal de RESET, no se va a limpiar el TBR, por lo que no se va a poder cargar otro byte a la terminal. Incluso, si no se activa LOAD, tampoco va a generarse la frecuencia de 9600 baudios, por lo que el proceso estaría haciendo nada. Sin embargo, al pulsarse dicho botón, sucedería la siguiente frecuencia:

1. Se carga un '1' que pueda garantizar que la terminal detecte un *falling_edge* en el siguiente estado.
2. En el estado START se carga un '0' para generar el *falling_edge* recién mencionado.
3. En DATA, se carga el primer bit de la secuencia de datos, y se lleva un conteo. Se prosigue con el segundo bit, así hasta llegar al último.
4. En STOP, se carga el bit '1' que indica que la secuencia ha sido cargada.

Mediante este simple código, podemos fácilmente cargar caracteres a la terminal. En este caso específico estaremos trabajando con la terminal Tera Term.




Además, se ha generado un testbench para confirmar la evolución de las señales a lo largo del ciclo, precisamente para saber si se está logrando la secuencia esperada en el orden solicitado.

Empezamos introduciendo los puertos y librerías del banco de pruebas:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component Lab07 is
port(
      clk : in std_logic; -- reloj
      reset : in std_logic; -- reset asincrono, activo high

      pdata : in std_logic_vector(7 downto 0); -- datos en paralelo (dato a enviar)
      load : in std_logic; -- cargar TBR

      txd : out std_logic; -- salida de transmision de datos
      empty : out std_logic); -- TBR vacio, active high
end component;
```

Generando, a continuación, las señales a simular y asignándolas en orden a sus respectivos puertos.

```
signal clk_in : std_logic;
signal reset_in : std_logic;

signal pdata_in : std_logic_vector(7 downto 0);
signal load_in : std_logic;

signal txd_out : std_logic;
signal empty_out : std_logic;

signal stop : boolean := false;
constant period: TIME := 10NS;

begin
  -- Connect DUT
  DUT: Lab07 port map(clk_in, reset_in, pdata_in, load_in, txd_out, empty_out);
```

Luego, generamos un proceso que simule el comportamiento cíclico del cristal que funciona como reloj interno de la tarjeta de desarrollo.

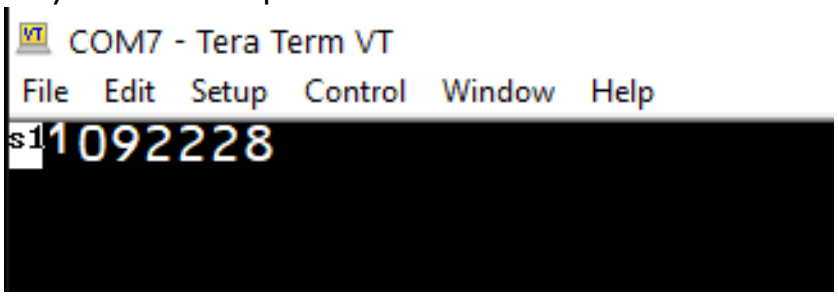
```
clock_gen : process
begin
    while not stop loop
        clk_in <= '0';
        wait for period/2;
        clk_in <= '1';
        wait for period/2;
    end loop;
    wait;
end process clock_gen;
```

Finalmente, generamos un proceso que empiece reiniciando las señales para preparar la carga de información y luego cargar los ocho bits que serían representados mediante switches, para luego cargar la información mediante la señal de load.

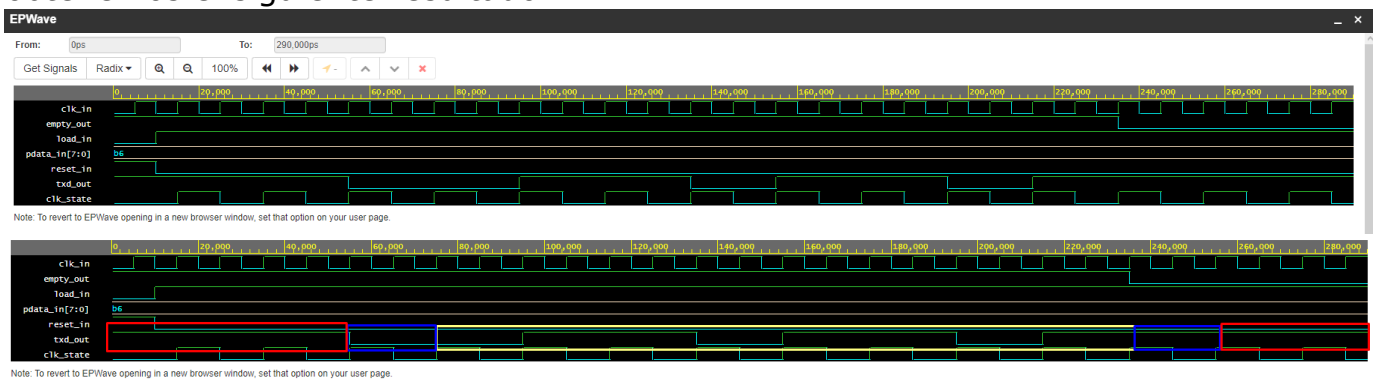
```
stimulus : process
  begin
    --wait for period;
    reset_in <= '1';
    load_in <= '0';
    pdata_in <= "10110110";

    wait for period;

    reset_in <= '0';
    load_in <= '1';

    wait for period*27;


    stop <= true;
    wait;
end process stimulus;
```

**Resultados:**

Hemos visto cómo mediante una simple combinación de ceros y unos en un bus de switches, hemos podido cargar información de caracteres a una terminal sin mucha complicación mediante tan solo un puerto de salida; utilizando, claramente una lógica secuencial y codificada que activaría su detección.

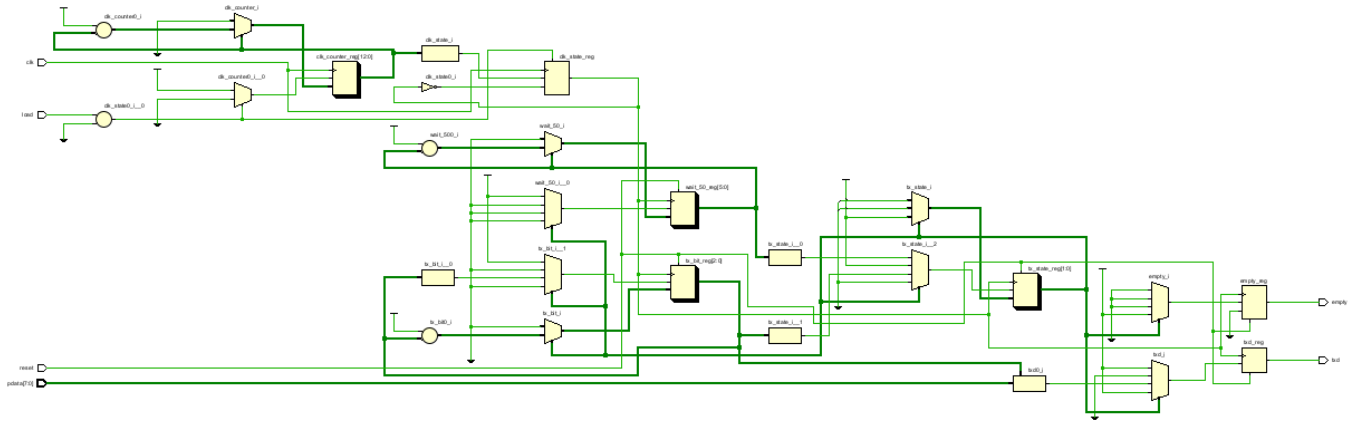


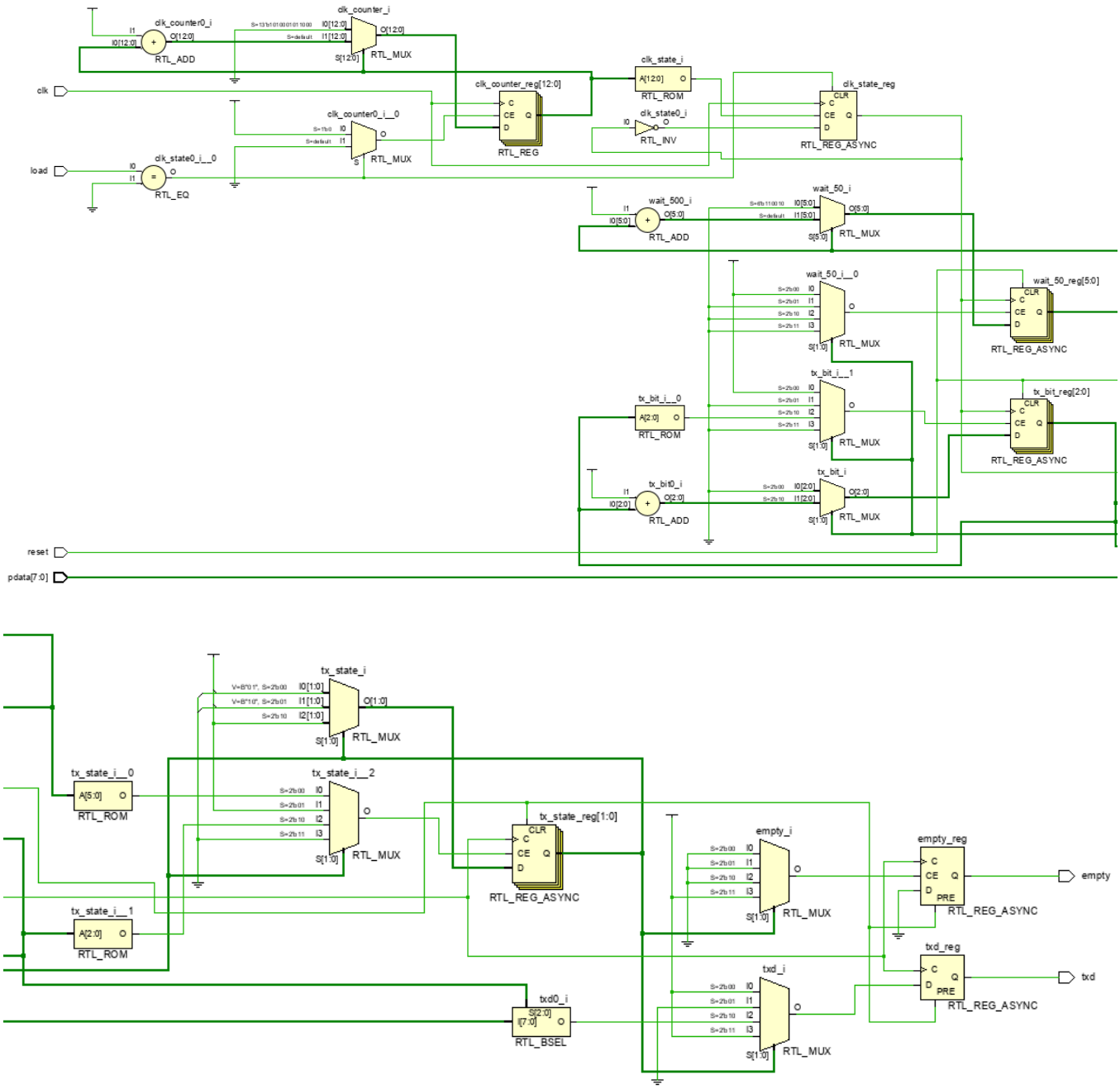

En el testbench adaptado para no trabajar con muchos ciclos de reloj obtenemos el siguiente resultado:




Se puede observar a simple vista las secciones de la señal, empezando por el descanso en las secciones rojas, observando los bits de START y STOP, respectivamente '0' y '1' y, finalmente, la sección marcada de amarillo que va leyendo y asignando cada bit del vector de derecha a izquierda, como corresponde su lectura.

Además, obtenemos el siguiente diagrama del hardware descrito:









**Análisis:**

Se pudo comprender el funcionamiento de la trasmisión serial y cómo esta asegura una correcta transmisión mediante bits que anuncian la llegada de un bus de datos. Además, mediante la simulación, se llega a observar muy detalladamente la señal transmitida y cómo esta responde perfectamente a la combinación deseada.

Finalmente, se pudo percibir cómo se transmite a la perfección el carácter enviado sin fallo alguno, a pesar de no estar trabajando exactamente a la frecuencia solicitada, pues dicha exactitud, con el reloj dado no es posible, sino aproximable. Lográndose, incluso con una incertidumbre menor al 1 %.