Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

Faculty of Engineering.

Career: Automation Engineer.

Subject: Systems with reconfigurable logic.

Student: Martínez Murillo Omar Yarif.
Student: Diego Joel Zúñiga Fragoso.
Student: Daniela del Carmen Manríquez Navarro.
Student: Joselyn Gallegos Abreo.

Practice 8: UART_TX

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Introduction

UART (Universal Asynchronous Receiver/Transmitter) communication is a serial communication method that enables data exchange between two devices asynchronously, meaning it does not require a shared clock signal. This is achieved through a specific protocol that defines how data bits are initiated, transmitted, and terminated. Before communication can begin, both devices must be configured with the same parameters, such as baud rate, data bit count, parity bits, and stop bits.

In this project, an FPGA was programmed to implement the UART protocol, specifically developing UART_TX communication using a state machine. The FPGA was configured to send data to a computer terminal via USB-TTL. A series of letters were transmitted based on a combination of switches on the FPGA, and the data was only sent once until the switches were pressed again.

Methodology

Step 1: State machine definition

A state machine was designed with 11 states: 10 for bit transmission (1 start bit, 8 data bits, and 1 parity bit) and a perpetual IDLE state. The main inputs and outputs were defined, including signals for character selection and the transmission start bit.

Step 2: Clock configuration

A clock divider was implemented to generate the required baud rate (9600 baud) using a counter to adjust the internal FPGA clock.

Step 3: Character selection

The characters to be transmitted were declared in a 7-element array, containing the binary representations of the letters 'A', 'K', 'R', 'I', 'J', 'm', and 'o'. The character to be sent is selected using input switches.

Step 4: Transmission state implementation

Each state of the state machine is responsible for transmitting one bit of the selected character. The initial states handle data bit transmission, followed by the parity and stop bits.

Step 5: State transitions

The state machine transitions from one state to another with each clock cycle until the entire character is transmitted. It then returns to the IDLE state and waits for a new transmission command.

Step 6: Compilation and programming

The code was compiled in the development environment and programmed onto the FPGA. Tests were conducted to verify data transmission through the UART port using the FPGA switches.

Results

Base algorithm UART_TX

```
 7  ----------------------------------------------------------------------------
 8  library IEEE;
 9  use IEEE.std_logic_1164.all;
10  use IEEE.numeric_std.all;
11
12  entity UART_TX is
13      port
14      (
15          i_FPGA_clk      : in STD_LOGIC;
16
17            -- Señales I/O de maquina de estado de comunicacion serial
18            i_char_select   : in STD_LOGIC_VECTOR(2 downto 0);
19            i_ST            : in STD_LOGIC;
20
21            o_EOT           : out STD_LOGIC;
22            o_M             : out STD_LOGIC
23      );
24  end UART_TX;
25  ----------------------------------------------------------------------------
```

```
architecture Transmition of UART_TX is

    -- Señales para transicion de estados
    type Estados is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, IDLE);
    signal act_state, next_state: Estados := IDLE; -- Ponemos en estado
    --signal UART_clk : STD_LOGIC;

    -- Señal para division de reloj
    signal count: integer range 0 to (5208);

    -- 7 palabras a enviar
    type array_7_t is array (0 to 6) of std_logic_vector(7 downto 0); --
    constant chars : array_7_t :=
    (
        "01000001",  -- A
        "01001011",  -- K
        "01010010",  -- R
        "01001001",  -- I
        "01001010",  -- J
        "01101101",  -- m
        "01101111"   -- o
    );
```

```
    signal char_index   : integer range 0 to 7 := 0;
    signal prev_i_ST    : STD_LOGIC := '0';
    signal flag         : STD_LOGIC := '0';

begin
-------------------------------------------------------------------
            -- DIVISION DE RELOJ PARA BAUDIOS DE TRANSMISION --
-------------------------------------------------------------------
    process (i_FPGA_clk)
    begin
        if rising_edge(i_FPGA_clk) then
            if count = (5208) then   -- 9600 Baudios
                --UART_clk = not UART_clk;
                act_state <= next_state;     -- Cambio de estado
                count <= 0;
            else
                count <= count + 1;
            end if;
        end if;
    end process;
```

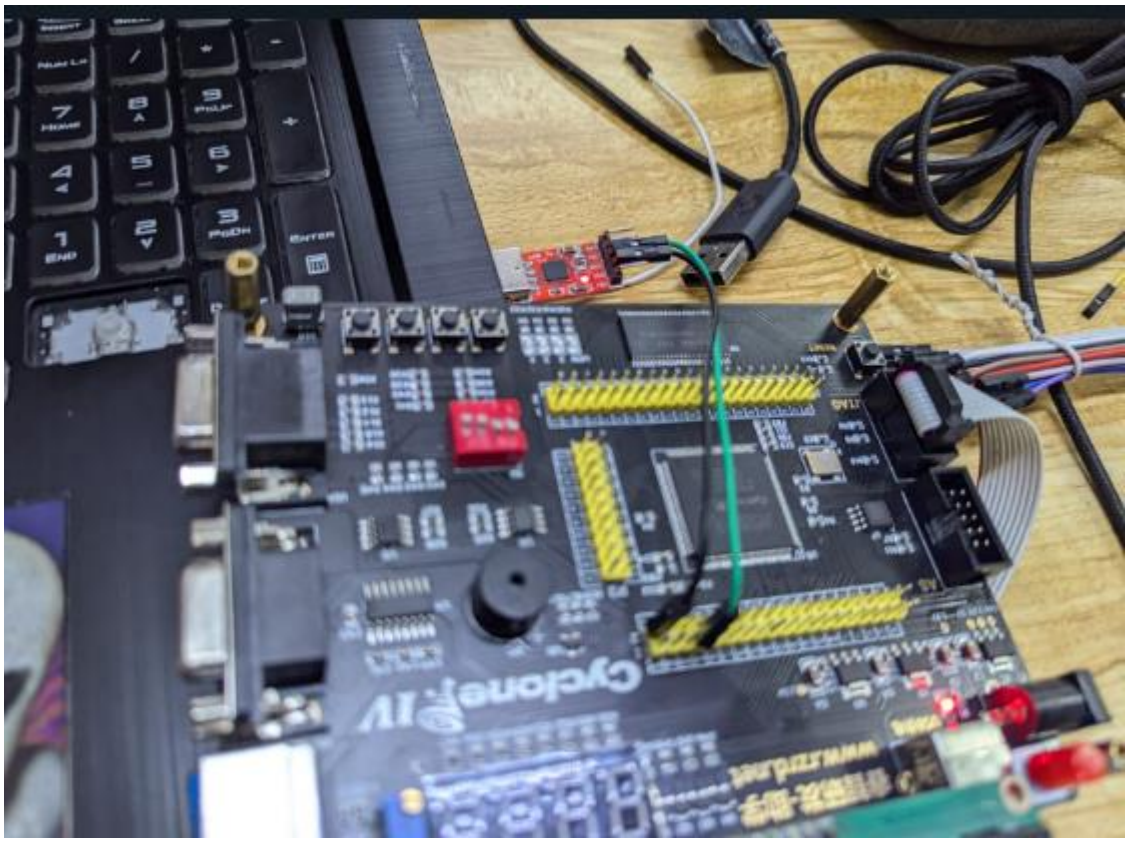
```
73      process (act_state)
74      begin
75          case act_state  is
76              when IDLE =>
77                  -- Enviamos señales de salida de estado actual
78                  o_M         <= '1';                                         -- P
79                  o_EOT       <= not '1';
80                  char_index  <= to_integer(unsigned(i_char_select));     -- S
81
82                  -- Actualizamos estado futuro en funcion de las entradas
83                  if i_ST = '1' then              -- Detectamos boton de inicio
84                      next_state <= S0;
85                  else
86                      next_state <= IDLE;
87                  end if;
88
89              when S0 =>      -- BIT DE START
90                  -- Enviamos señales de salida de estado actual
91                  o_M     <= '0';
92                  o_EOT   <= not '0';
93
94                  -- Actualizamos estado futuro en funcion de las entradas
95                  next_state <= S1;
96
97              when S1 =>      -- BIT 0
98                  -- Enviamos señales de salida de estado actual
99                  o_M     <= chars(char_index)(0);
00                  o_EOT   <= not '0';
01
02                  -- Actualizamos estado futuro en funcion de las entradas
03                  next_state <= S2;
04
05              when S2 =>      -- BIT 1
06                  -- Enviamos señales de salida de estado actual
07                  o_M     <= chars(char_index)(1);
08                  o_EOT   <= not '0';
09
10                  -- Actualizamos estado futuro en funcion de las entradas
11                  next_state <= S3;
12
13              when S3 =>      -- BIT 2
14                  -- Enviamos señales de salida de estado actual
15                  o_M     <= chars(char_index)(2);
16                  o_EOT   <= not '0';
17
18                  -- Actualizamos estado futuro en funcion de las entradas
19                  next_state <= S4;
20
```

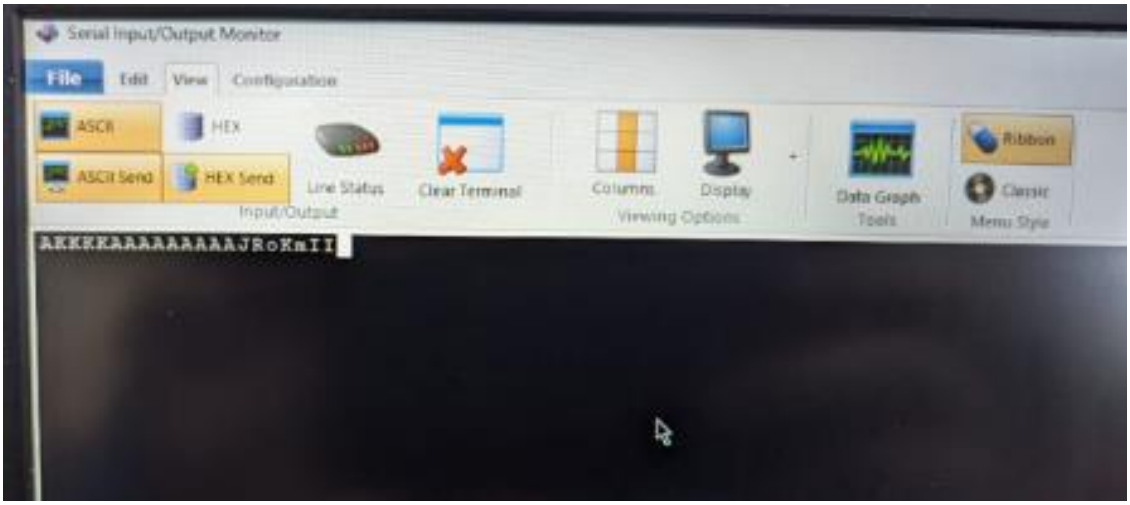
```
121                 when S4 =>      -- BIT 3
122                     -- Enviamos señales de salida de estado actual
123                     o_M     <= chars(char_index)(3);
124                     o_EOT   <= not '0';
125 
126                     -- Actualizamos estado futuro en funcion de las entradas
127                     next_state <= S5;
128 
129                 when S5 =>      -- BIT 4
130                     -- Enviamos señales de salida de estado actual
131                     o_M     <= chars(char_index)(4);
132                     o_EOT   <= not '0';
133 
134                     -- Actualizamos estado futuro en funcion de las entradas
135                     next_state <= S6;
136 
137                 when S6 =>      -- BIT 5
138                     -- Enviamos señales de salida de estado actual
139                     o_M     <= chars(char_index)(5);
140                     o_EOT   <= not '0';
141 
142                     -- Actualizamos estado futuro en funcion de las entradas
143                     next_state <= S7;
```

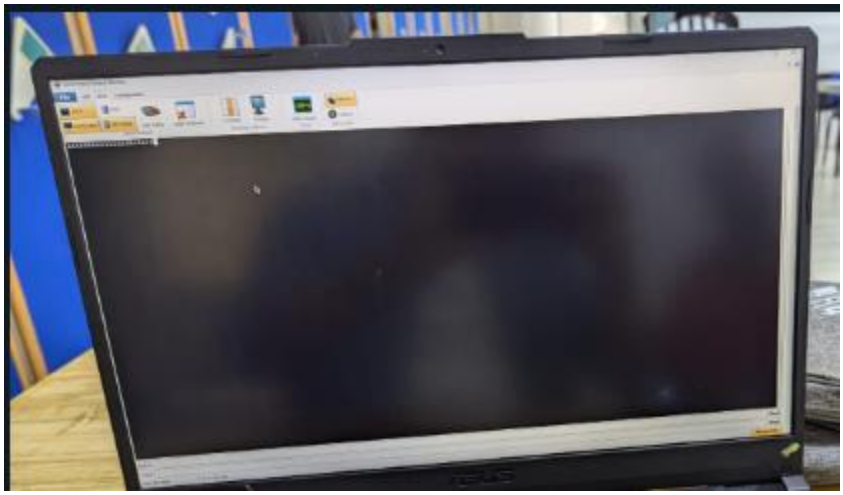
```
145             when S7 =>      -- BIT 6
146                 -- Enviamos señales de salida de estado actual
147                 o_M     <= chars(char_index)(6);
148                 o_EOT   <= not '0';
149 
150                 -- Actualizamos estado futuro en funcion de las entradas
151                 next_state <= S8;
152 
153             when S8 =>      -- BIT 7
154                 -- Enviamos señales de salida de estado actual
155                 o_M     <= chars(char_index)(7);
156                 o_EOT   <= not '0';
157 
158                 -- Actualizamos estado futuro en funcion de las entradas
159                 next_state <= S9;
160             when S9 =>      -- BIT DE PARIDAD
161                 -- Enviamos señales de salida de estado actual
162                 o_M     <= (chars(char_index)(0) xor chars(char_index)(1)) xor (chars(char
163                 o_EOT   <= not '0';
164 
165                 next_state <= S10;
166             when S10 =>     -- BIT DE STOP
167                 -- Enviamos señales de salida de estado actual
168                 o_M     <= '1';
169                 o_EOT   <= not '0';
170 
```

```
171               -- Actualizamos estado futuro en funcion de las entradas
172               if i_ST = '1' then            -- Detectamos boton de inicio de 
173                   next_state <= S10;
174               else
175                   next_state <= IDLE;
176               end if;
177           when others => null;
178       end case;
179   end process;
180 
181 end Transmition;
```

Hardware of FPGA with UART_TX







Conclusion

The implementation of a state machine for UART transmission on an FPGA highlighted the importance of serial communication protocols in digital systems. The use of UART_TX enabled efficient data transmission from the FPGA to a computer terminal, with precise control over the start, data, parity, and stop bits. This type of communication is essential in applications where data needs to be transferred between devices without the need for a shared clock signal.

Applications for such systems include connecting microcontrollers, sensors, and other electronic devices, particularly in embedded systems where UART is used for communication between different modules or for interacting with external peripherals. Implementing this technique in FPGAs also showcases their flexibility and power in designing custom communication systems, which is crucial in industries such as industrial automation, telecommunications, and consumer electronics.