Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

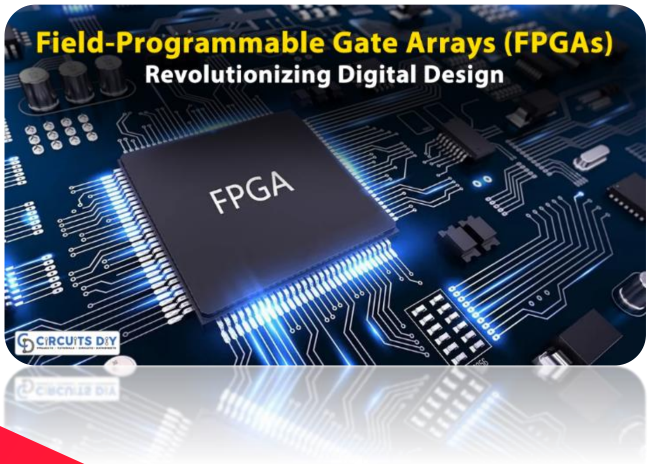
Faculty of Engineering.

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 9: UART_RX


Field-Programmable Gate Arrays (FPGAs)
Revolutionizing Digital Design
FPGA
CIRCUITS DIY

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 practice, an FPGA was programmed to implement the UART protocol, specifically developing UART_RX communication using a state machine. The FPGA was configured to receive data from a computer terminal via USB-TTL and display it on an LCD. A series of words were transmitted from the computer to the FPGA, and the data was displayed on the LCD screen.

Methodology

Step 1: State machine definition

A state machine was designed with multiple states to handle UART_RX data reception and subsequent display on an LCD. The machine includes states for LCD configuration, waiting for data, and character printing.

Step 2: Clock configuration

A clock divider was implemented to control the UART_RX transmission speed, ensuring data reception at 9600 baud. The clock also controls the state transitions within the machine.

Step 3: UART data reception

The UART_RX component was designed to receive asynchronous serial data from a computer terminal via USB-TTL. As the data arrives at the FPGA, it is temporarily stored until reception is complete.

Step 4: LCD configuration state

Before displaying the data on the LCD, the system goes through several configuration states to initialize and prepare the LCD screen, ensuring it is ready to receive data.

Step 5: Character printing state

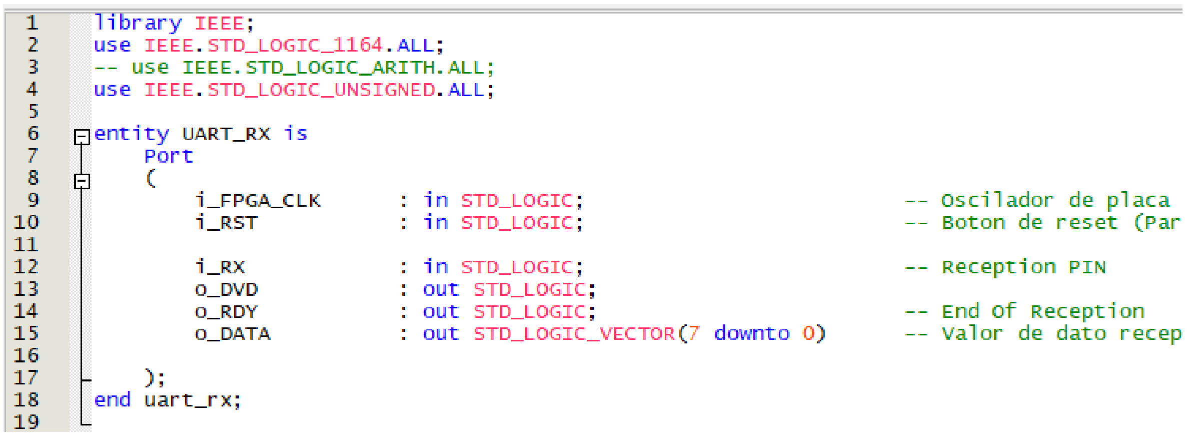
Once the data is received by UART_RX, the system enters the printing state, where the ASCII data is sent to the LCD screen. The state machine controls the printing process bit by bit, ensuring each character is correctly displayed.

Step 6: Compilation and programming

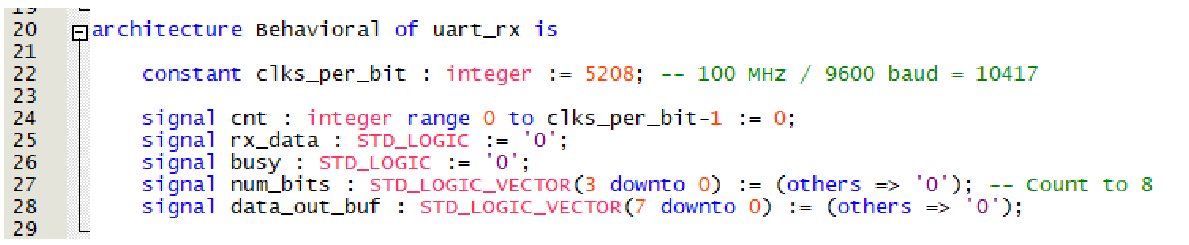
The code was compiled in the development environment and programmed onto the FPGA. Tests were conducted to verify that data sent from the computer was correctly displayed on the LCD screen.

Results

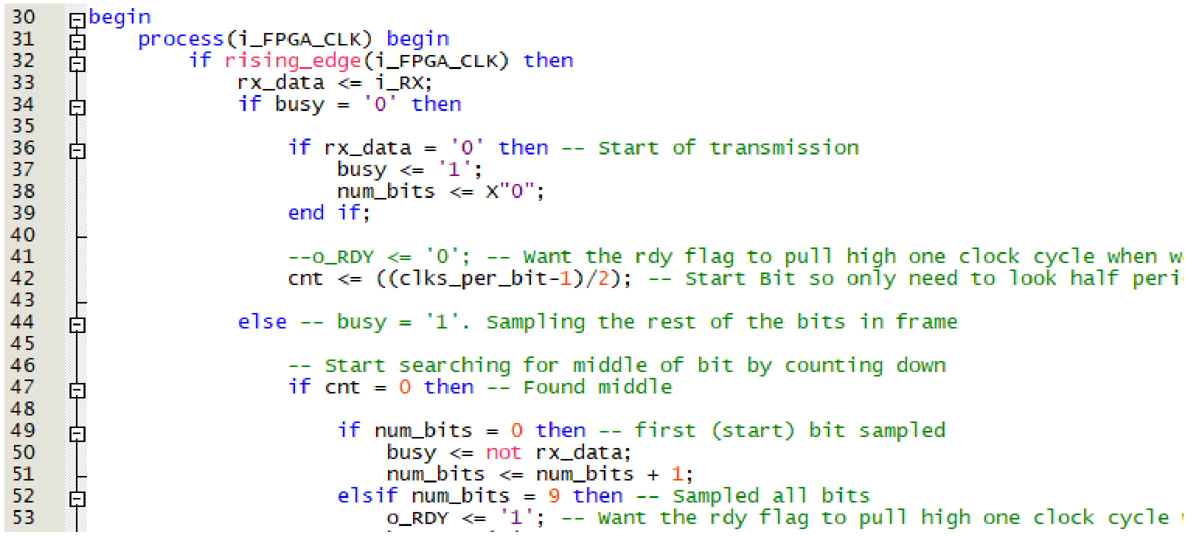
Base algorithm UART_RX



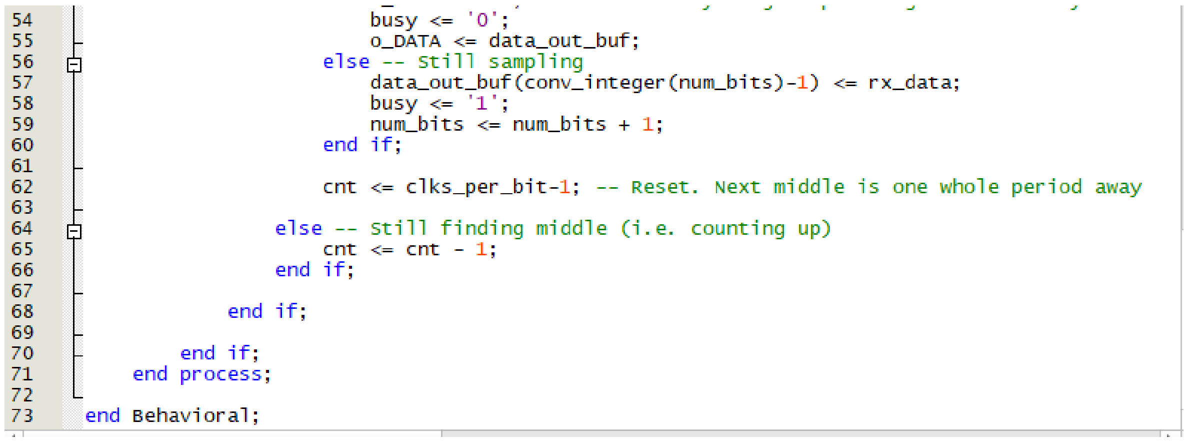
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UART_RX is
    Port
    (
        i_FPGA_CLK      : in STD_LOGIC;                      -- Oscilador de placa
        i_RST           : in STD_LOGIC;                      -- Boton de reset (Par

        i_RX            : in STD_LOGIC;                      -- Reception PIN
        o_DVD           : out STD_LOGIC;
        o_RDY           : out STD_LOGIC;                     -- End of Reception
        o_DATA          : out STD_LOGIC_VECTOR(7 downto 0)   -- Valor de dato recep

    );
end uart_rx;
```



```
architecture Behavioral of uart_rx is

    constant clks_per_bit : integer := 5208; -- 100 MHz / 9600 baud = 10417

    signal cnt : integer range 0 to clks_per_bit-1 := 0;
    signal rx_data : STD_LOGIC := '0';
    signal busy : STD_LOGIC := '0';
    signal num_bits : STD_LOGIC_VECTOR(3 downto 0) := (others => '0'); -- Count to 8
    signal data_out_buf : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

```

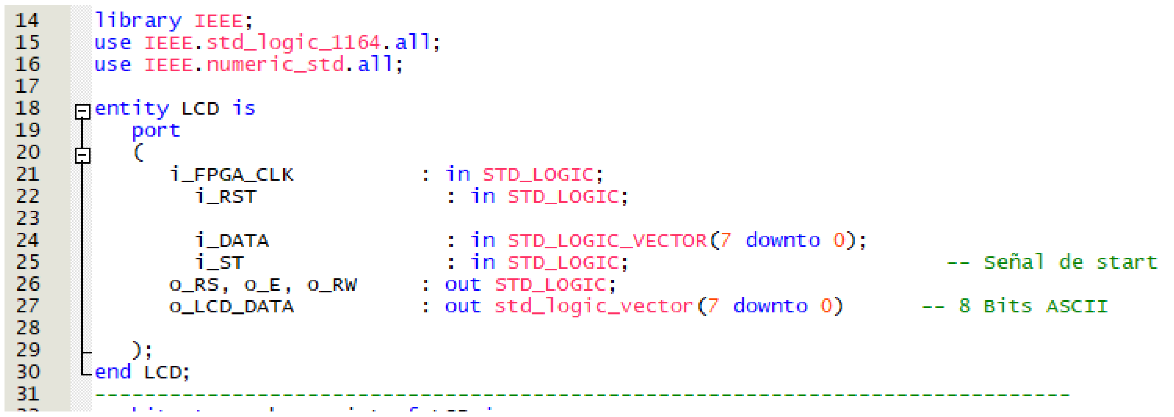


```
begin
    process(i_FPGA_CLK) begin
        if rising_edge(i_FPGA_CLK) then
            rx_data <= i_RX;
            if busy = '0' then

                if rx_data = '0' then -- Start of transmission
                    busy <= '1';
                    num_bits <= X"0";
                end if;

                --o_RDY <= '0'; -- Want the rdy flag to pull high one clock cycle when w
                cnt <= ((clks_per_bit-1)/2); -- Start Bit so only need to look half peri

            else -- busy = '1'. Sampling the rest of the bits in frame

                -- Start searching for middle of bit by counting down
                if cnt = 0 then -- Found middle

                    if num_bits = 0 then -- first (start) bit sampled
                        busy <= not rx_data;
                        num_bits <= num_bits + 1;
                    elsif num_bits = 9 then -- Sampled all bits
                        o_RDY <= '1'; -- Want the rdy flag to pull high one clock cycle
```

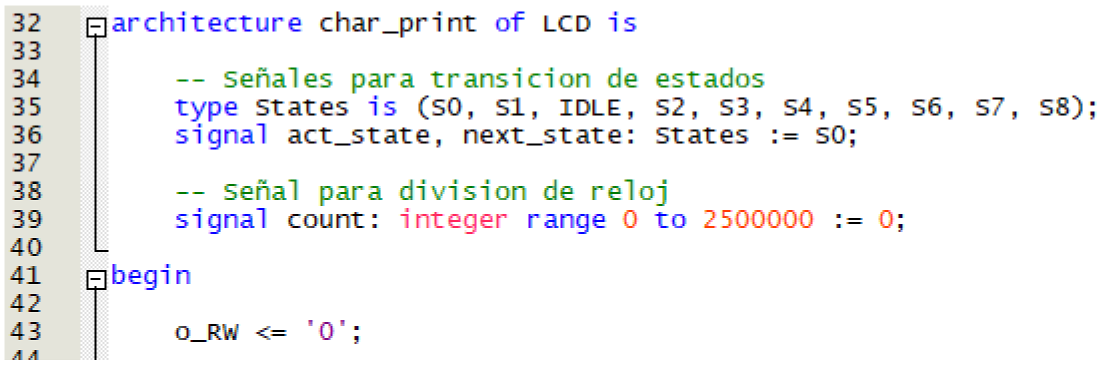


```
                        busy <= '0';
                        o_DATA <= data_out_buf;
                    else -- Still sampling
                        data_out_buf(conv_integer(num_bits)-1) <= rx_data;
                        busy <= '1';
                        num_bits <= num_bits + 1;
                    end if;

                    cnt <= clks_per_bit-1; -- Reset. Next middle is one whole period away

                else -- Still finding middle (i.e. counting up)
                    cnt <= cnt - 1;
                end if;

            end if;

        end if;
    end process;

end Behavioral;
```

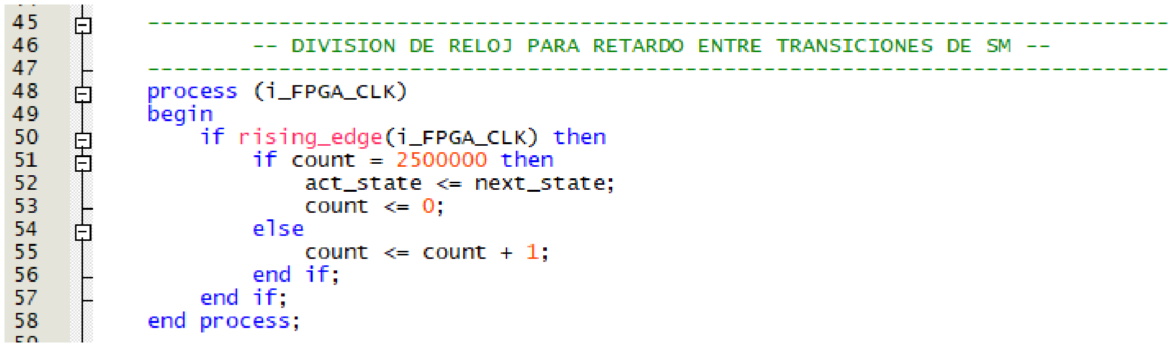
Base algorithm LCD



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity LCD is
   port
   (
      i_FPGA_CLK          : in STD_LOGIC;
        i_RST               : in STD_LOGIC;

        i_DATA              : in STD_LOGIC_VECTOR(7 downto 0);
        i_ST                : in STD_LOGIC;                          -- Señal de start
      o_RS, o_E, o_RW     : out STD_LOGIC;
      o_LCD_DATA          : out std_logic_vector(7 downto 0)       -- 8 Bits ASCII

   );
end LCD;
------------------------------------------------------------------------------
```

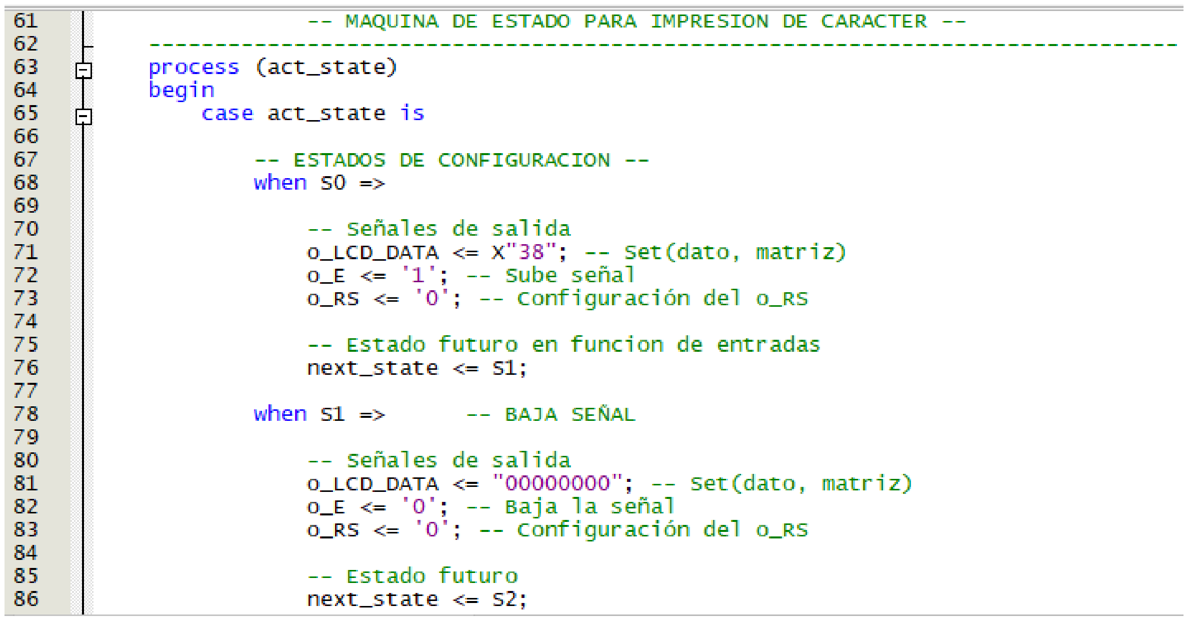


```vhdl
architecture char_print of LCD is

    -- Señales para transicion de estados
    type States is (S0, S1, IDLE, S2, S3, S4, S5, S6, S7, S8);
    signal act_state, next_state: States := S0;

    -- Señal para division de reloj
    signal count: integer range 0 to 2500000 := 0;

begin

    o_RW <= '0';
```

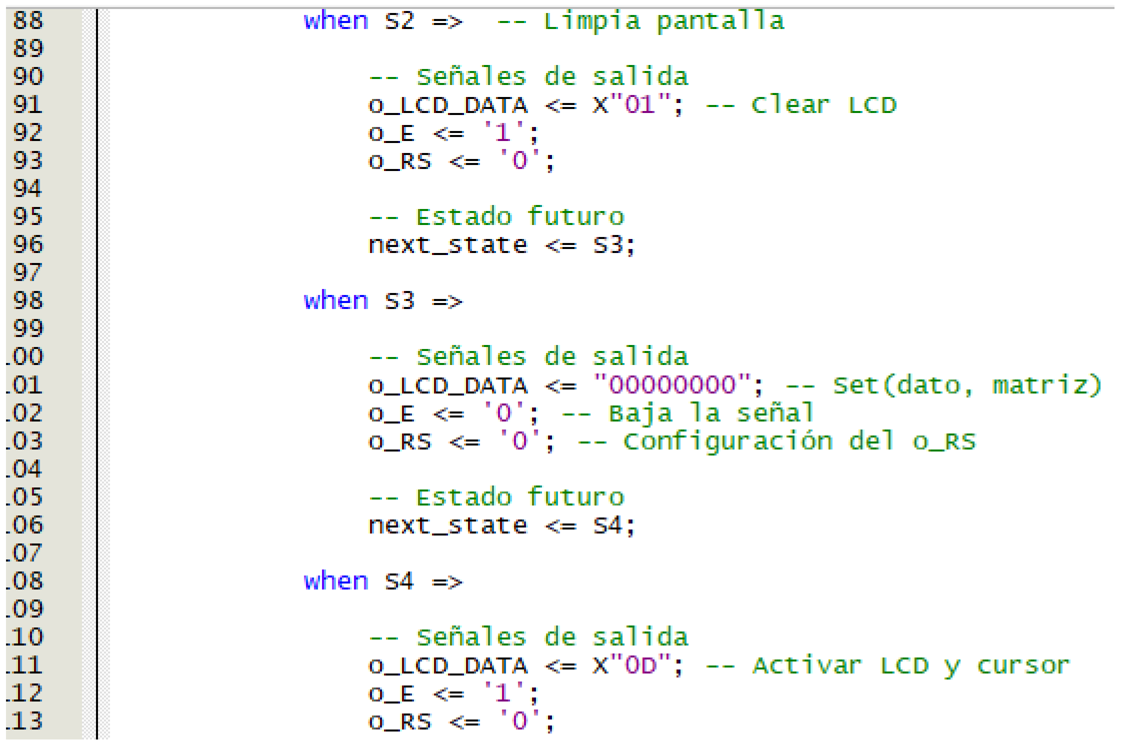


```vhdl
    ------------------------------------------------------------------------------
            -- DIVISION DE RELOJ PARA RETARDO ENTRE TRANSICIONES DE SM --
    ------------------------------------------------------------------------------
    process (i_FPGA_CLK)
    begin
        if rising_edge(i_FPGA_CLK) then
            if count = 2500000 then
                act_state <= next_state;
                count <= 0;
            else
                count <= count + 1;
            end if;
        end if;
    end process;
```

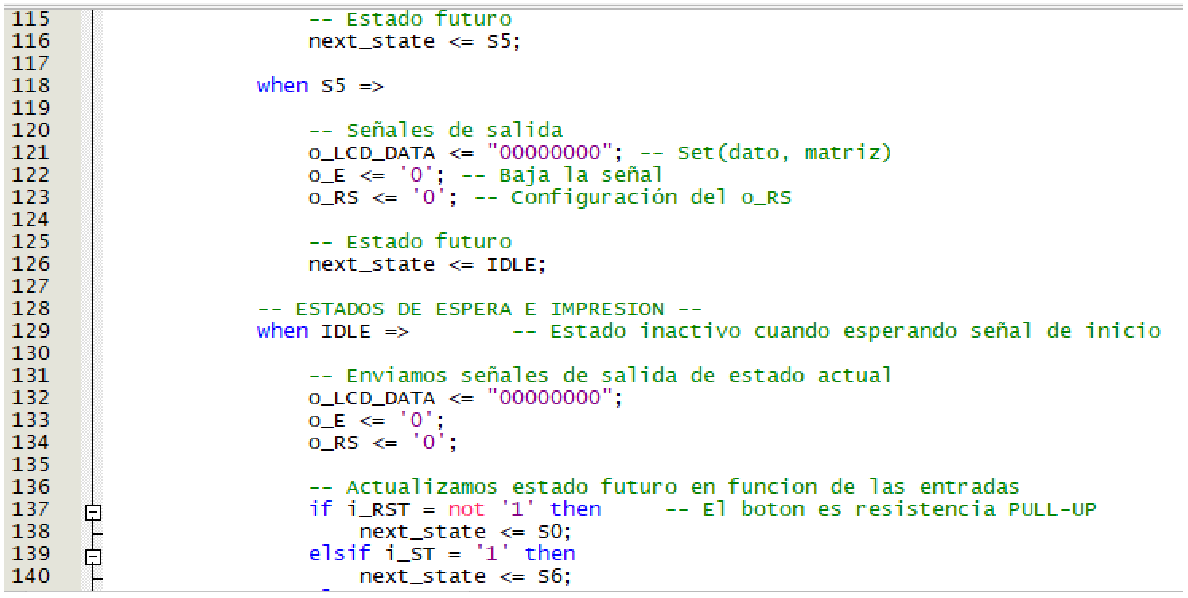
Universidad Autónoma De Querétaro
Facultad de Ingeniería



```
            -- MAQUINA DE ESTADO PARA IMPRESION DE CARACTER --
---------------------------------------------------------------------------------
process (act_state)
begin
    case act_state is

        -- ESTADOS DE CONFIGURACION --
        when S0 =>

            -- Señales de salida
            o_LCD_DATA <= X"38"; -- Set(dato, matriz)
            o_E <= '1'; -- Sube señal
            o_RS <= '0'; -- Configuración del o_RS

            -- Estado futuro en funcion de entradas
            next_state <= S1;

        when S1 =>      -- BAJA SEÑAL

            -- Señales de salida
            o_LCD_DATA <= "00000000"; -- Set(dato, matriz)
            o_E <= '0'; -- Baja la señal
            o_RS <= '0'; -- Configuración del o_RS

            -- Estado futuro
            next_state <= S2;
```

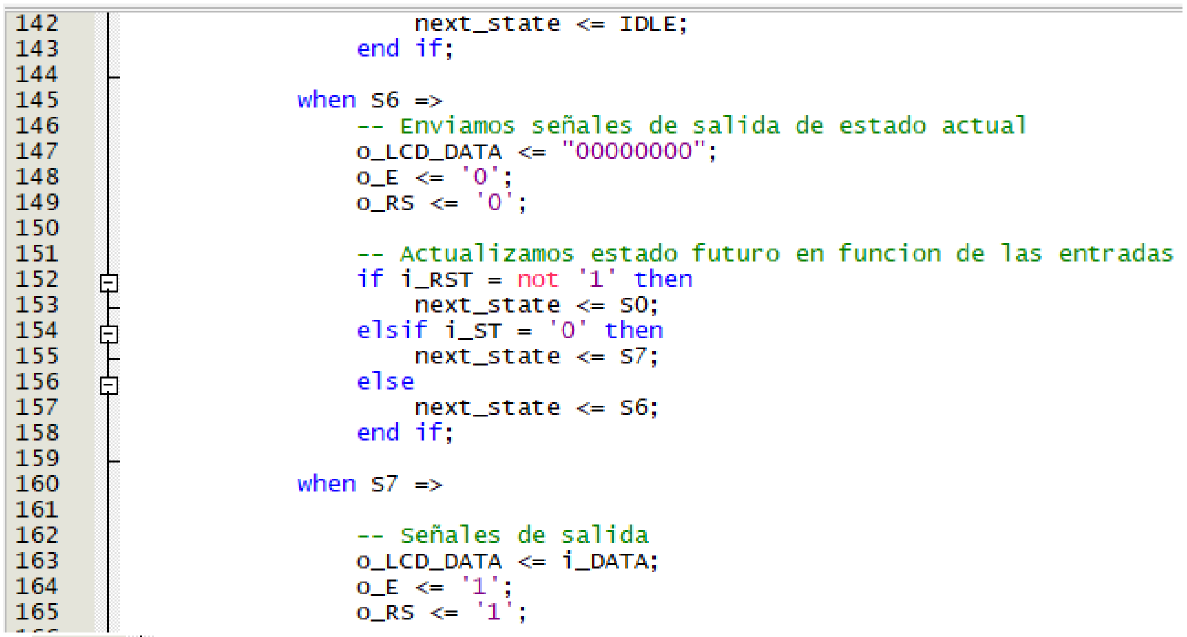


```
        when S2 =>  -- Limpia pantalla

            -- Señales de salida
            o_LCD_DATA <= X"01"; -- Clear LCD
            o_E <= '1';
            o_RS <= '0';

            -- Estado futuro
            next_state <= S3;

        when S3 =>

            -- Señales de salida
            o_LCD_DATA <= "00000000"; -- Set(dato, matriz)
            o_E <= '0'; -- Baja la señal
            o_RS <= '0'; -- Configuración del o_RS

            -- Estado futuro
            next_state <= S4;

        when S4 =>

            -- Señales de salida
            o_LCD_DATA <= X"0D"; -- Activar LCD y cursor
            o_E <= '1';
            o_RS <= '0';
```

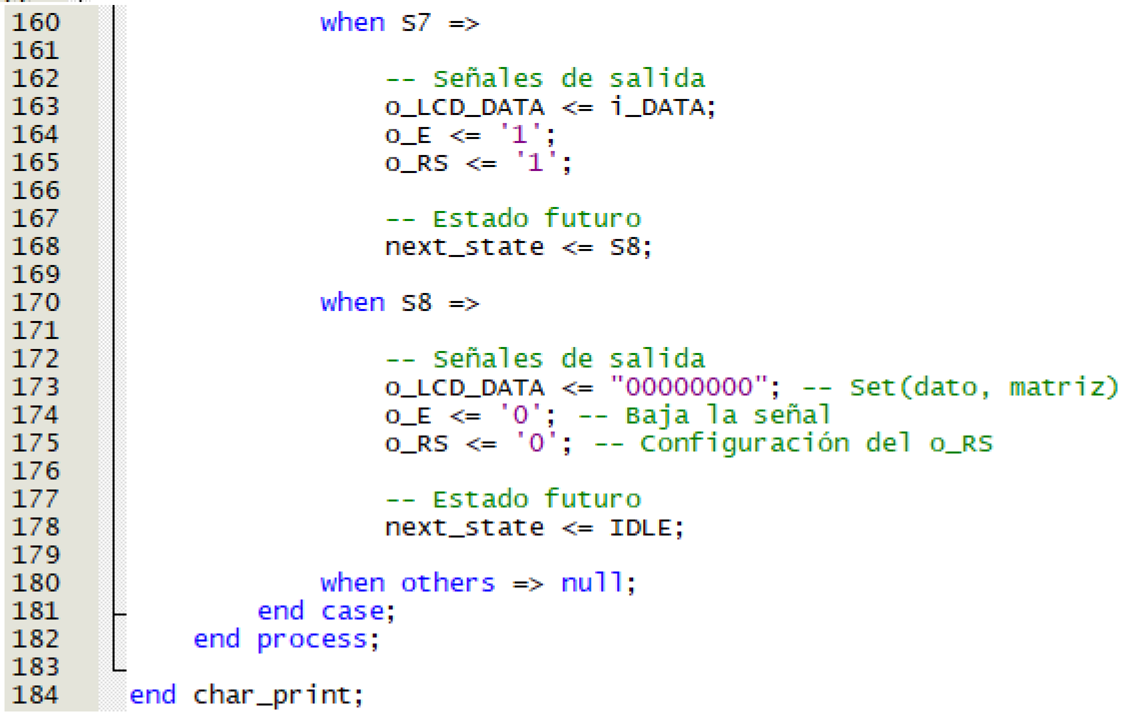


```
                -- Estado futuro
                next_state <= S5;

            when S5 =>

                -- Señales de salida
                o_LCD_DATA <= "00000000"; -- Set(dato, matriz)
                o_E <= '0'; -- Baja la señal
                o_RS <= '0'; -- Configuración del o_RS

                -- Estado futuro
                next_state <= IDLE;

            -- ESTADOS DE ESPERA E IMPRESION --
            when IDLE =>        -- Estado inactivo cuando esperando señal de inicio

                -- Enviamos señales de salida de estado actual
                o_LCD_DATA <= "00000000";
                o_E <= '0';
                o_RS <= '0';

                -- Actualizamos estado futuro en funcion de las entradas
                if i_RST = not '1' then     -- El boton es resistencia PULL-UP
                    next_state <= S0;
                elsif i_ST = '1' then
                    next_state <= S6;
```

Universidad Autónoma De Querétaro
Facultad de Ingeniería

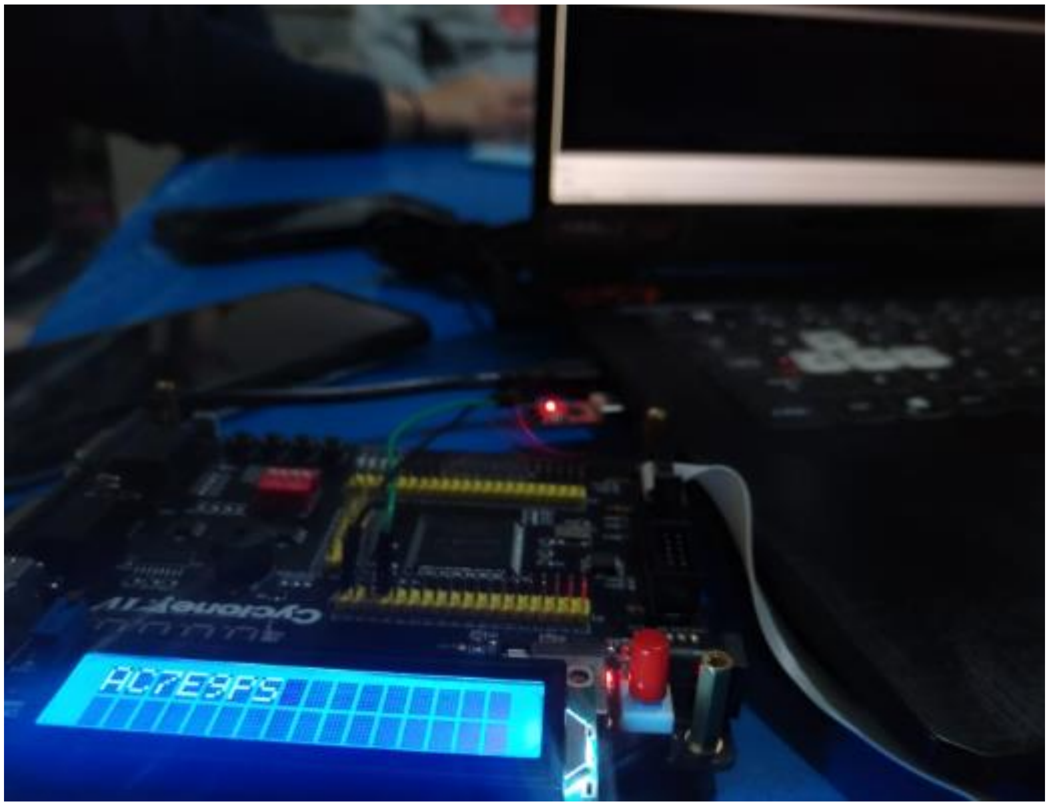


```vhdl
                    next_state <= IDLE;
                end if;

            when S6 =>
                -- Enviamos señales de salida de estado actual
                o_LCD_DATA <= "00000000";
                o_E <= '0';
                o_RS <= '0';

                -- Actualizamos estado futuro en funcion de las entradas
                if i_RST = not '1' then
                    next_state <= S0;
                elsif i_ST = '0' then
                    next_state <= S7;
                else
                    next_state <= S6;
                end if;

            when S7 =>

                -- Señales de salida
                o_LCD_DATA <= i_DATA;
                o_E <= '1';
                o_RS <= '1';
```



```vhdl
            when S7 =>

                -- Señales de salida
                o_LCD_DATA <= i_DATA;
                o_E <= '1';
                o_RS <= '1';

                -- Estado futuro
                next_state <= S8;

            when S8 =>

                -- Señales de salida
                o_LCD_DATA <= "00000000"; -- Set(dato, matriz)
                o_E <= '0'; -- Baja la señal
                o_RS <= '0'; -- Configuración del o_RS

                -- Estado futuro
                next_state <= IDLE;

            when others => null;
        end case;
    end process;
end char_print;
```

Hardware of FPGA with UART_RX



Conclusion

The implementation of a state machine for UART reception (UART_RX) on an FPGA emphasized the importance of serial communication protocols in digital systems. The use of UART_RX allowed efficient data reception from a computer terminal to the FPGA, with precise control over the start, data, parity, and stop bits. This type of communication is crucial in applications where devices need to receive data asynchronously without 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 receiving communication from external modules or peripherals. Implementing this technique in FPGAs highlights their flexibility and power in designing custom communication systems.