Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

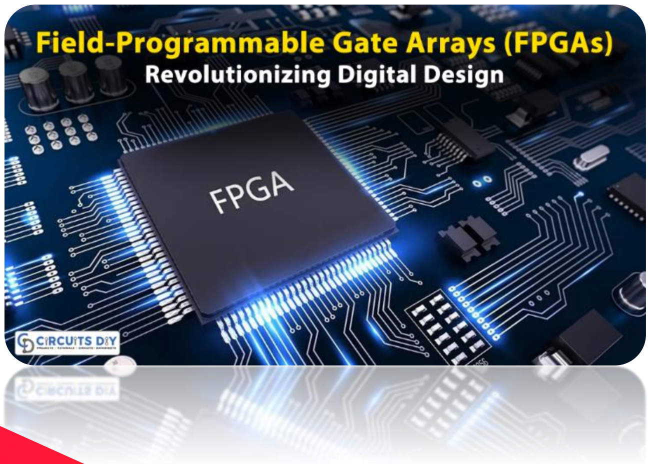
Faculty of Engineering.

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 10: Step-Motor


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

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

## Introduction

In this project, a stepper motor will be controlled using state machines divided into separate components. One component manages the LCD, which will display the input distances. The next component establishes UART_RX communication to receive distances sent by the PC, and another component handles UART_TX communication to send a confirmation message for the received distance. Lastly, a motor component is created to perform the necessary steps corresponding to the specified distances.

## Methodology

### Clock Divider (CLK_DIV)

Purpose: This component divides the FPGA's base clock to obtain a frequency suitable for the required baud rate in UART communication or other timing-sensitive operations.

Implementation:

The CLK_DIV component takes the FPGA's clock frequency and an output frequency as generic parameters.

It divides the input clock signal using a counter, so each cycle of the output clock matches the desired output frequency. This clock division helps control timing in the state machine and data transmission or reception rates.

Code Highlights:

The component contains a counter that increments with each clock pulse.

Once the counter reaches the necessary count to create the desired frequency, it resets and toggles the output clock (o_clk).

### UART_RX Component (UART_RX)

Purpose: Receives serial data asynchronously from the PC and outputs the received byte when the transmission is complete.

Implementation:

The UART_RX component receives the clock signal, reset, and input signal (i_RX) for data reception.

A finite state machine (FSM) manages the reception, starting with detecting a start bit (indicating data transmission has begun), capturing each data bit sequentially, and checking parity before signaling data reception completion.

Code Highlights:

States:

IDLE: Waits for the start bit to begin data reception.

DATA_RECEIVE: Captures data bits as per the baud rate, stores them in the DATA register, and validates the parity bit at the end.

Outputs:

o_RCV signals that a new byte of data is ready.

o_DATA holds the received data byte.

**UART_TX_string Component**

Purpose: Sends a series of characters as a string through UART.

Implementation:

Uses a state machine to manage string transmission, character by character, to the serial output.

When triggered by i_ST, it reads each byte from i_string, passes it to a UART_TX_byte component for transmission, and waits for acknowledgment before sending the next byte.

Code Highlights:

States:

IDLE: Waits for the start signal (i_ST) to begin transmission.

WAIT_RDY: Checks if the previous character has been transmitted.

SEND_CHAR: Sends each character in sequence and transitions back to WAIT_RDY until the entire string is transmitted.

**LCD_putstring Component**

Purpose: Displays a string of text on an LCD by sending ASCII characters one at a time.

Implementation:

This component takes an input string (i_string) and, when triggered by the start signal (i_ST), outputs each character in sequence to the LCD.

State transitions handle LCD initialization, screen clearing, and data loading in line with LCD timing requirements.

Code Highlights:

States:

CONFIG0 and CONFIG1: Initialize the LCD with specific commands.

CLEAR_DISP: Clears the display.

SEND_CHAR: Outputs each character from the input string to the LCD in sequence.

NEW_LINE: Advances to the next line after a set number of characters, ensuring proper display format.

**Stepper Motor Component (Stepper_motor)**

Purpose: Controls the stepper motor based on direction and step frequency.

Implementation:

Receives control signals i_ON (on/off) and i_DIR (direction). Using a state machine, it cycles through a sequence to rotate the motor in the specified direction at the desired speed.

Code Highlights:

States:

IDLE: Waits for the i_ON signal to start stepping.

STEPS: Executes each step in the motor sequence, updating o_STEPS for control and tracking completed steps.

Outputs:

o_STEPS provides the step sequence to drive the motor.

o_done_steps records the number of completed steps.

**Main Code Overview**

The main code (probably Practica_10.vhd) integrates each of these components, coordinating their functions to receive distances via UART, display them on an LCD, and control the motor accordingly.

**Components in the Main Code:**

CLK_DIV for clock generation tailored to the needs of UART and other timing-sensitive tasks.

UART_RX to receive data bytes from the PC, likely distances for the motor.

LCD_putstring to display the received distance on the LCD.

Stepper_motor to drive the motor based on received distance input.

Data Flow:

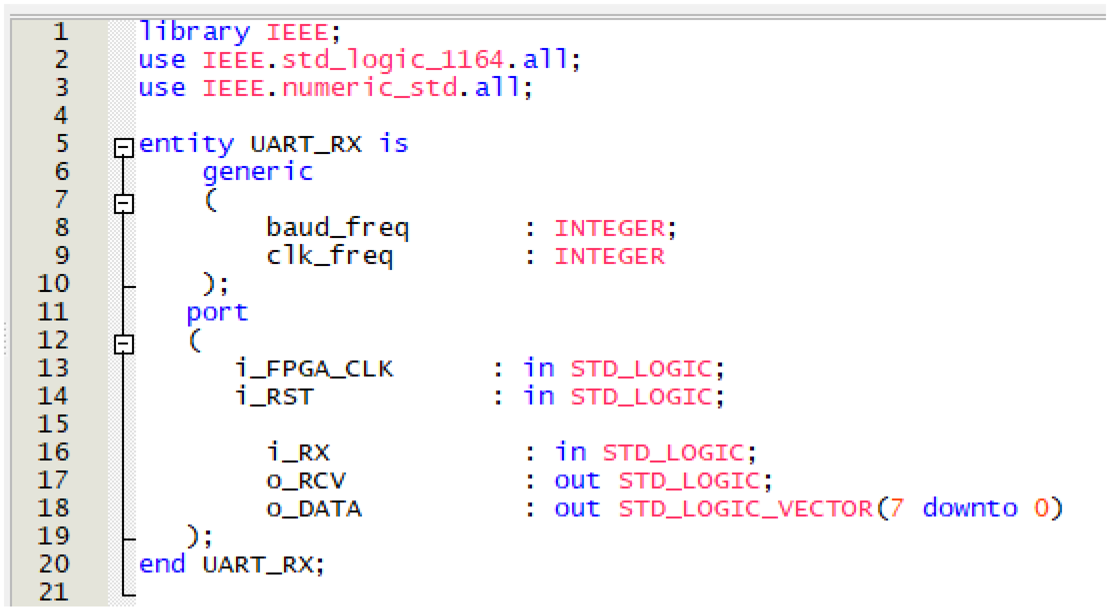
UART_RX captures data from the PC, triggering the display update via LCD_putstring.

The motor component receives control signals derived from the distance data, performing the required steps as commanded.

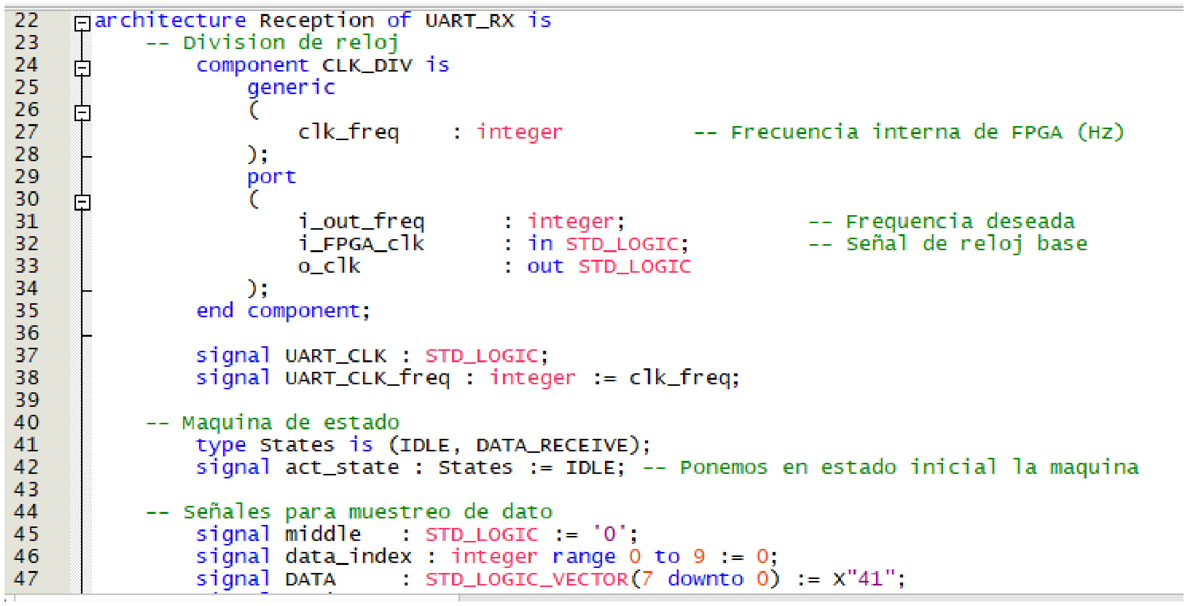
This component-based design allows for modular and efficient control of each subsystem (UART reception, LCD display, motor control) within the FPGA environment.

## Results

Base algorithm UART_RX



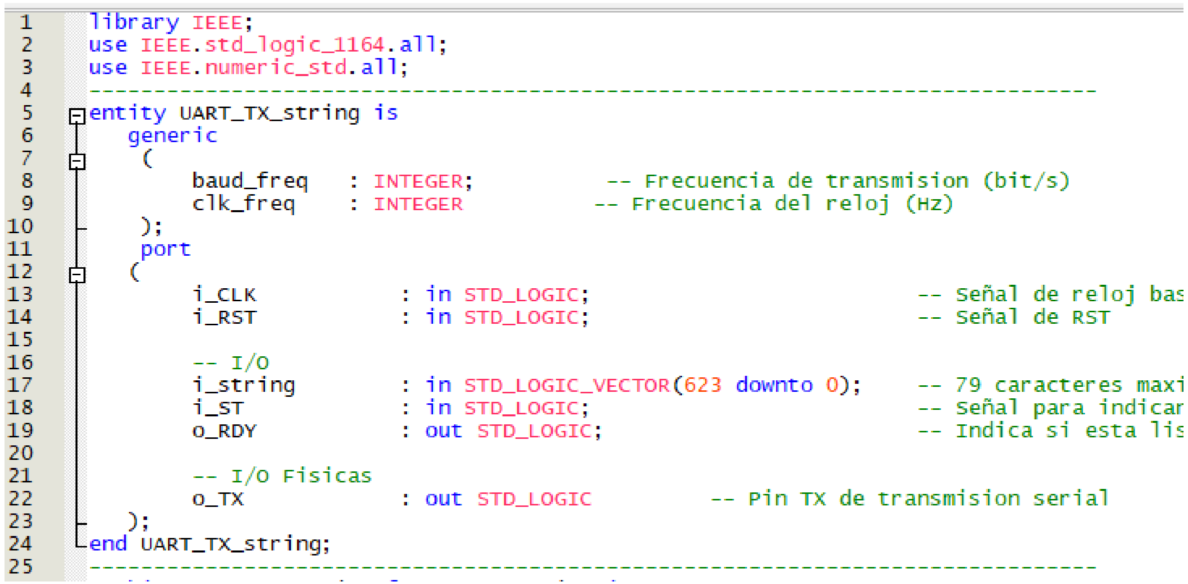
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity UART_RX is
    generic
    (
        baud_freq       : INTEGER;
        clk_freq        : INTEGER
    );
   port
   (
      i_FPGA_CLK      : in STD_LOGIC;
      i_RST           : in STD_LOGIC;

        i_RX            : in STD_LOGIC;
        o_RCV           : out STD_LOGIC;
        o_DATA          : out STD_LOGIC_VECTOR(7 downto 0)
   );
end UART_RX;

```



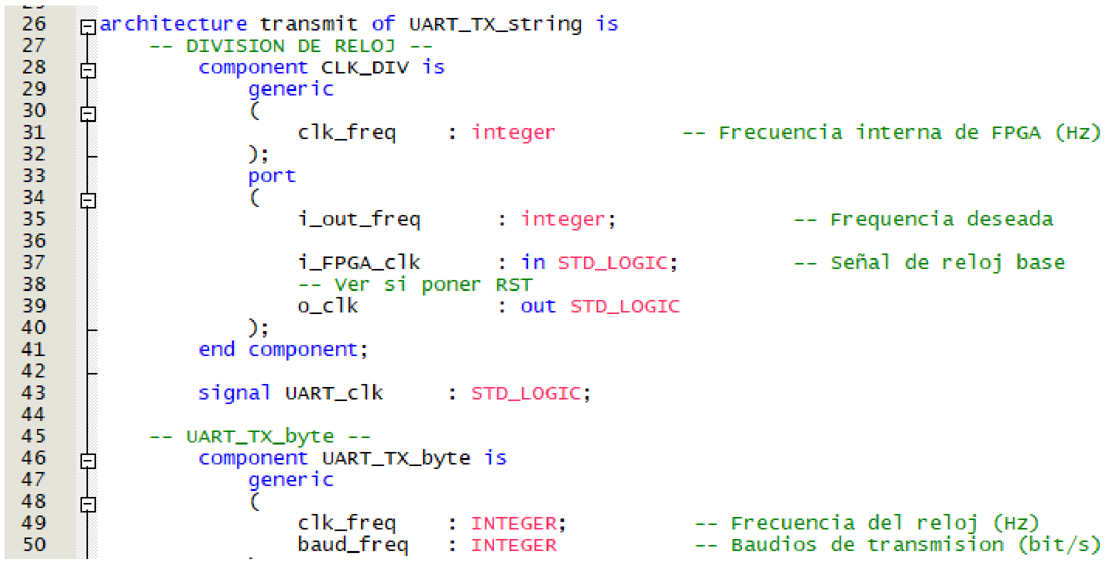
```vhdl
architecture Reception of UART_RX is
    -- Division de reloj
       component CLK_DIV is
           generic
           (
               clk_freq    : integer          -- Frecuencia interna de FPGA (Hz)
           );
           port
           (
               i_out_freq      : integer;              -- Frequencia deseada
               i_FPGA_clk      : in STD_LOGIC;         -- Señal de reloj base
               o_clk           : out STD_LOGIC
           );
       end component;

       signal UART_CLK : STD_LOGIC;
       signal UART_CLK_freq : integer := clk_freq;

    -- Maquina de estado
       type States is (IDLE, DATA_RECEIVE);
       signal act_state : States := IDLE; -- Ponemos en estado inicial la maquina

    -- Señales para muestreo de dato
       signal middle    : STD_LOGIC := '0';
       signal data_index : integer range 0 to 9 := 0;
       signal DATA      : STD_LOGIC_VECTOR(7 downto 0) := x"41";
```

Universidad Autónoma De Querétaro
Facultad de Ingeniería

## Base algorithm UART_TX



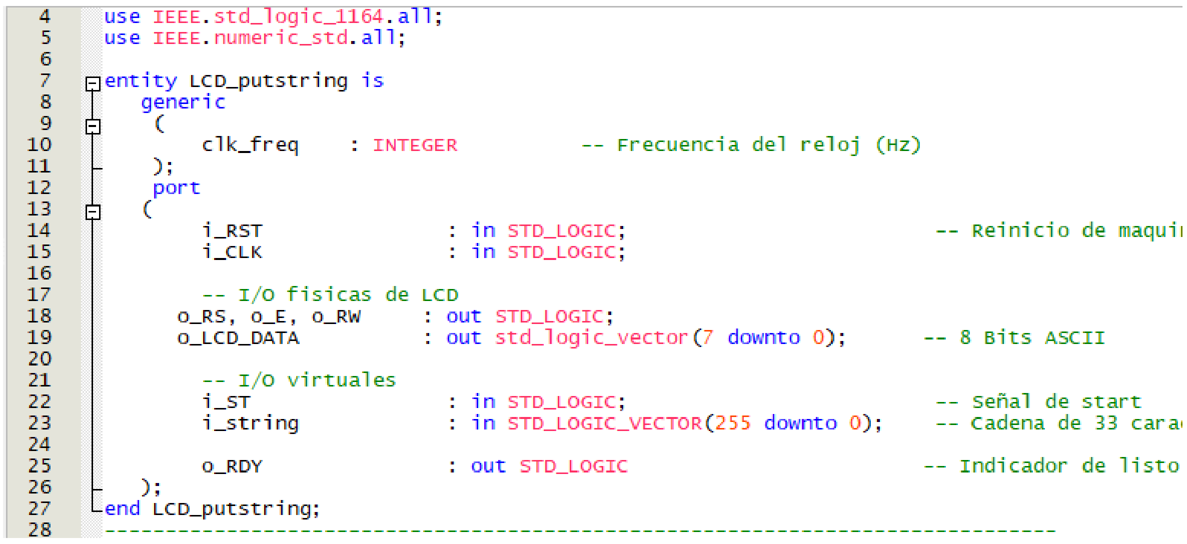
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
-------------------------------------------------------------------------------
entity UART_TX_string is
   generic
    (
        baud_freq   : INTEGER;          -- Frecuencia de transmision (bit/s)
        clk_freq    : INTEGER          -- Frecuencia del reloj (Hz)
    );
    port
   (
        i_CLK           : in STD_LOGIC;                           -- Señal de reloj bas
        i_RST           : in STD_LOGIC;                           -- Señal de RST

        -- I/O
        i_string        : in STD_LOGIC_VECTOR(623 downto 0);      -- 79 caracteres maxi
        i_ST            : in STD_LOGIC;                           -- Señal para indicar
        o_RDY           : out STD_LOGIC;                          -- Indica si esta lis

        -- I/O Fisicas
        o_TX            : out STD_LOGIC         -- Pin TX de transmision serial
   );
end UART_TX_string;
-------------------------------------------------------------------------------
```



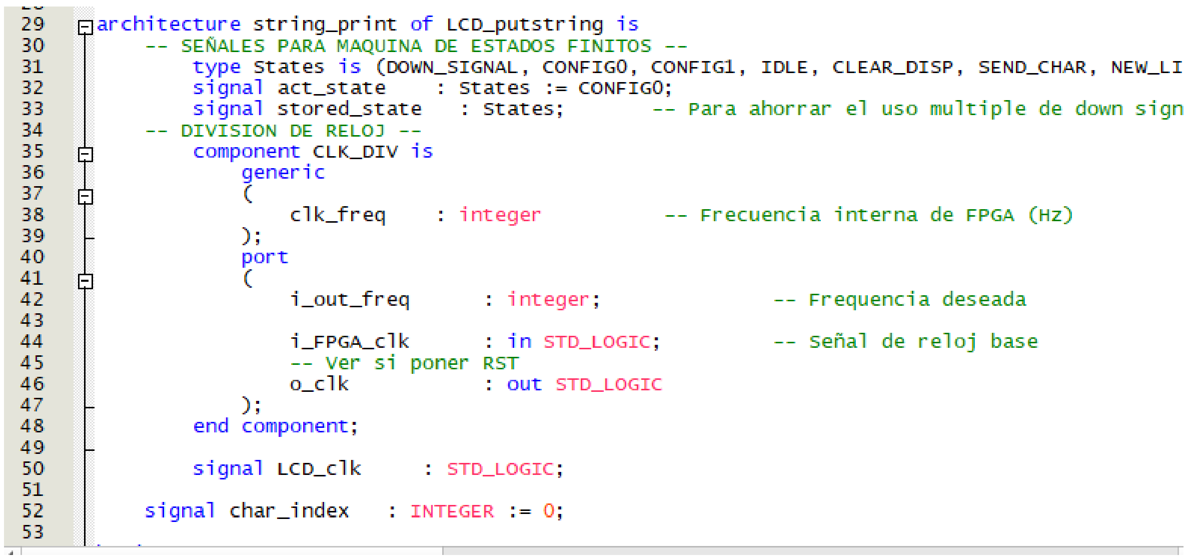
```vhdl
architecture transmit of UART_TX_string is
    -- DIVISION DE RELOJ --
        component CLK_DIV is
            generic
            (
                clk_freq    : integer           -- Frecuencia interna de FPGA (Hz)
            );
            port
            (
                i_out_freq      : integer;               -- Frequencia deseada

                i_FPGA_clk      : in STD_LOGIC;          -- Señal de reloj base
                -- Ver si poner RST
                o_clk           : out STD_LOGIC
            );
        end component;

        signal UART_clk     : STD_LOGIC;

    -- UART_TX_byte --
        component UART_TX_byte is
            generic
            (
                clk_freq    : INTEGER;          -- Frecuencia del reloj (Hz)
                baud_freq   : INTEGER           -- Baudios de transmision (bit/s)
```

Universidad Autónoma De Querétaro
Facultad de Ingeniería

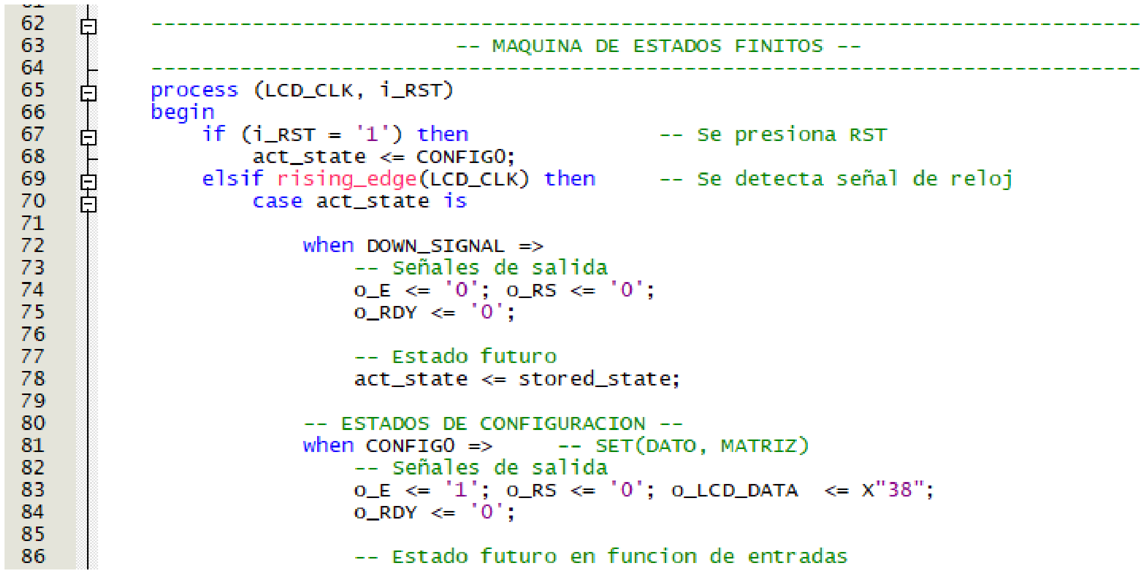
## Base algorithm LCD



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

entity LCD_putstring is
   generic
    (
        clk_freq    : INTEGER          -- Frecuencia del reloj (Hz)
    );
    port
   (
        i_RST                : in STD_LOGIC;                         -- Reinicio de maqui
        i_CLK                : in STD_LOGIC;

        -- I/O fisicas de LCD
      o_RS, o_E, o_RW      : out STD_LOGIC;
      o_LCD_DATA           : out std_logic_vector(7 downto 0);      -- 8 Bits ASCII

        -- I/O virtuales
        i_ST                 : in STD_LOGIC;                         -- Señal de start
        i_string             : in STD_LOGIC_VECTOR(255 downto 0);    -- Cadena de 33 cara

        o_RDY                : out STD_LOGIC                        -- Indicador de listo
   );
end LCD_putstring;
--------------------------------------------------------------------------------
```

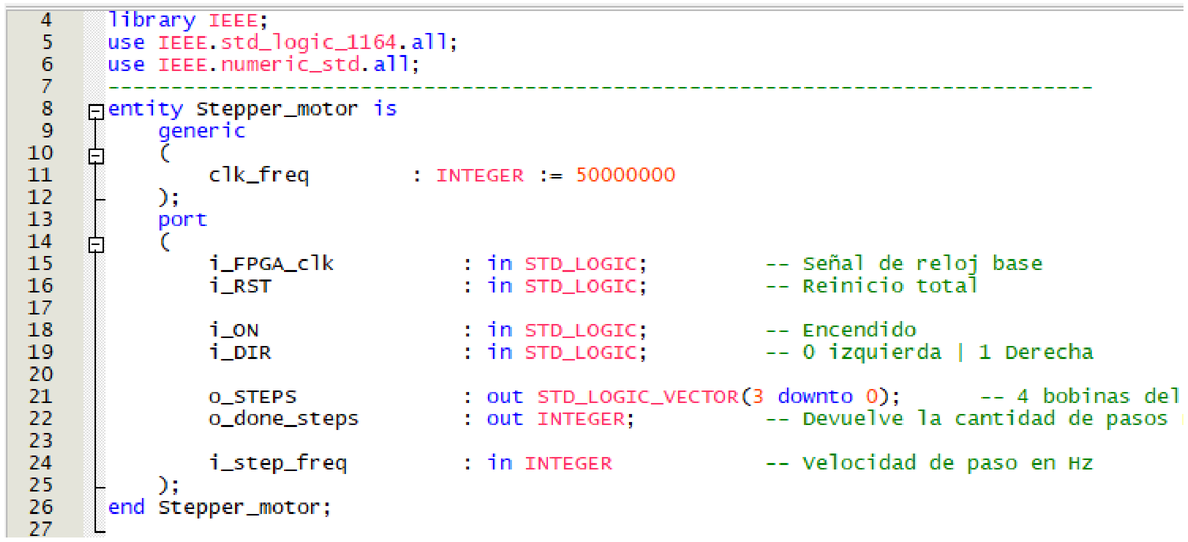


```vhdl
architecture string_print of LCD_putstring is
    -- SEÑALES PARA MAQUINA DE ESTADOS FINITOS --
        type States is (DOWN_SIGNAL, CONFIG0, CONFIG1, IDLE, CLEAR_DISP, SEND_CHAR, NEW_LI
        signal act_state     : States := CONFIG0;
        signal stored_state    : States;       -- Para ahorrar el uso multiple de down sign
    -- DIVISION DE RELOJ --
        component CLK_DIV is
            generic
            (
                clk_freq    : integer          -- Frecuencia interna de FPGA (Hz)
            );
            port
            (
                i_out_freq      : integer;              -- Frequencia deseada

                i_FPGA_clk      : in STD_LOGIC;         -- Señal de reloj base
                -- Ver si poner RST
                o_clk           : out STD_LOGIC
            );
        end component;

        signal LCD_clk     : STD_LOGIC;

    signal char_index    : INTEGER := 0;
```

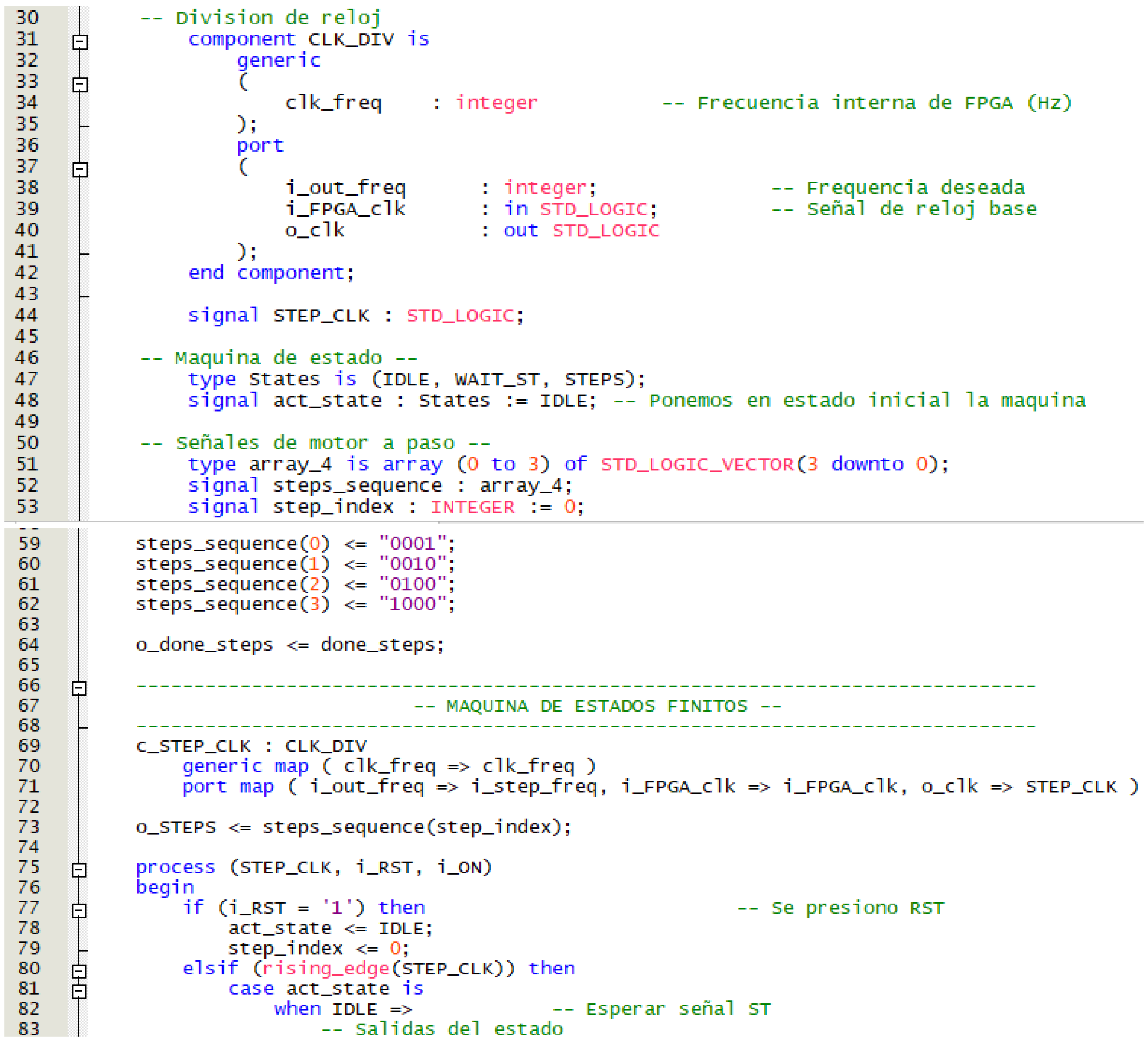


```vhdl
    ------------------------------------------------------------------------------
                            -- MAQUINA DE ESTADOS FINITOS --
    ------------------------------------------------------------------------------
    process (LCD_CLK, i_RST)
    begin
        if (i_RST = '1') then                -- Se presiona RST
            act_state <= CONFIG0;
        elsif rising_edge(LCD_CLK) then      -- Se detecta señal de reloj
            case act_state is

                when DOWN_SIGNAL =>
                    -- Señales de salida
                    o_E <= '0'; o_RS <= '0';
                    o_RDY <= '0';

                    -- Estado futuro
                    act_state <= stored_state;

                -- ESTADOS DE CONFIGURACION --
                when CONFIG0 =>       -- SET(DATO, MATRIZ)
                    -- Señales de salida
                    o_E <= '1'; o_RS <= '0'; o_LCD_DATA  <= X"38";
                    o_RDY <= '0';

                    -- Estado futuro en funcion de entradas
```

## Base algorithm STEPPER MOTOR



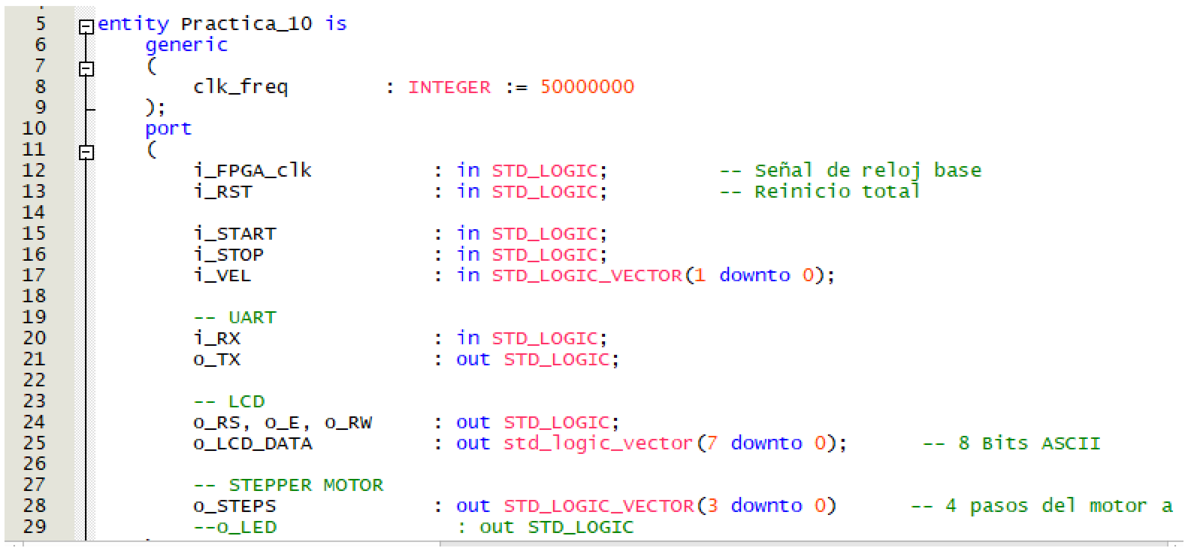
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------------------------------------------------------------------------------
entity Stepper_motor is
    generic
    (
        clk_freq        : INTEGER := 50000000
    );
    port
    (
        i_FPGA_clk          : in STD_LOGIC;         -- Señal de reloj base
        i_RST               : in STD_LOGIC;         -- Reinicio total

        i_ON                : in STD_LOGIC;         -- Encendido
        i_DIR               : in STD_LOGIC;         -- 0 izquierda | 1 Derecha

        o_STEPS             : out STD_LOGIC_VECTOR(3 downto 0);     -- 4 bobinas del
        o_done_steps        : out INTEGER;          -- Devuelve la cantidad de pasos

        i_step_freq         : in INTEGER            -- Velocidad de paso en Hz
    );
end Stepper_motor;
```



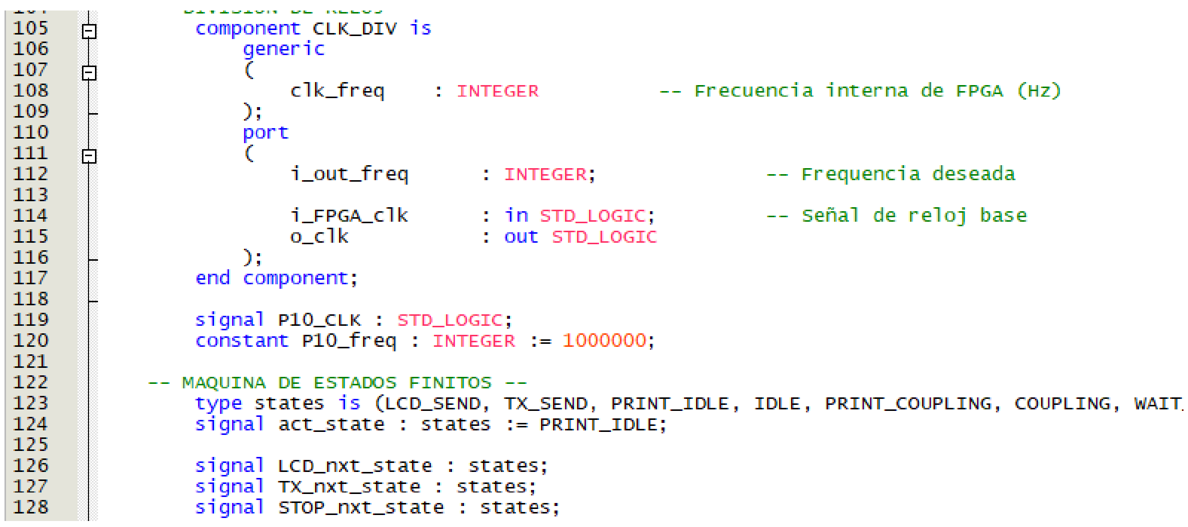
```vhdl
-- Division de reloj
    component CLK_DIV is
        generic
        (
            clk_freq    : integer          -- Frecuencia interna de FPGA (Hz)
        );
        port
        (
            i_out_freq      : integer;              -- Frequencia deseada
            i_FPGA_clk      : in STD_LOGIC;         -- Señal de reloj base
            o_clk           : out STD_LOGIC
        );
    end component;

    signal STEP_CLK : STD_LOGIC;

-- Maquina de estado --
    type States is (IDLE, WAIT_ST, STEPS);
    signal act_state : States := IDLE; -- Ponemos en estado inicial la maquina

-- Señales de motor a paso --
    type array_4 is array (0 to 3) of STD_LOGIC_VECTOR(3 downto 0);
    signal steps_sequence : array_4;
    signal step_index : INTEGER := 0;
```

```vhdl
steps_sequence(0) <= "0001";
steps_sequence(1) <= "0010";
steps_sequence(2) <= "0100";
steps_sequence(3) <= "1000";

o_done_steps <= done_steps;

--------------------------------------------------------------------------
                       -- MAQUINA DE ESTADOS FINITOS --
--------------------------------------------------------------------------
c_STEP_CLK : CLK_DIV
    generic map ( clk_freq => clk_freq )
    port map ( i_out_freq => i_step_freq, i_FPGA_clk => i_FPGA_clk, o_clk => STEP_CLK )

o_STEPS <= steps_sequence(step_index);

process (STEP_CLK, i_RST, i_ON)
begin
    if (i_RST = '1') then                         -- Se presiono RST
        act_state <= IDLE;
        step_index <= 0;
    elsif (rising_edge(STEP_CLK)) then
        case act_state is
            when IDLE =>           -- Esperar señal ST
                -- Salidas del estado
```

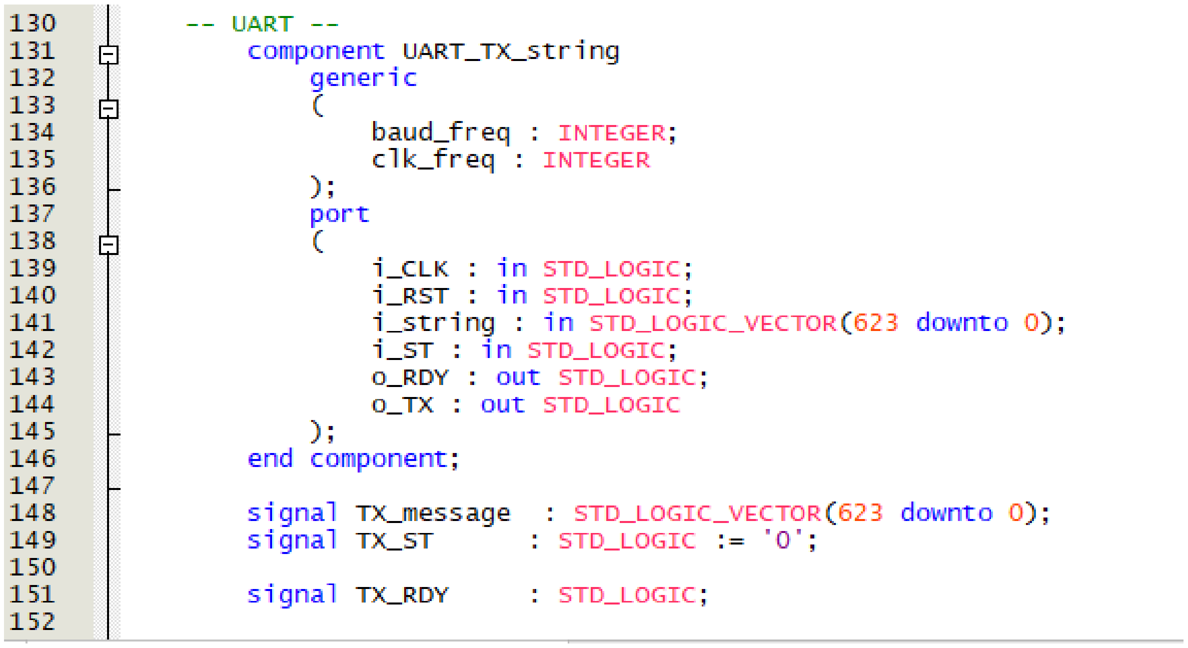
## Base code



```
entity Practica_10 is
    generic
    (
        clk_freq        : INTEGER := 50000000
    );
    port
    (
        i_FPGA_clk          : in STD_LOGIC;         -- Señal de reloj base
        i_RST               : in STD_LOGIC;         -- Reinicio total

        i_START             : in STD_LOGIC;
        i_STOP              : in STD_LOGIC;
        i_VEL               : in STD_LOGIC_VECTOR(1 downto 0);

        -- UART
        i_RX                : in STD_LOGIC;
        o_TX                : out STD_LOGIC;

        -- LCD
        o_RS, o_E, o_RW     : out STD_LOGIC;
        o_LCD_DATA          : out std_logic_vector(7 downto 0);      -- 8 Bits ASCII

        -- STEPPER MOTOR
        o_STEPS             : out STD_LOGIC_VECTOR(3 downto 0)      -- 4 pasos del motor a
        --o_LED              : out STD_LOGIC
```

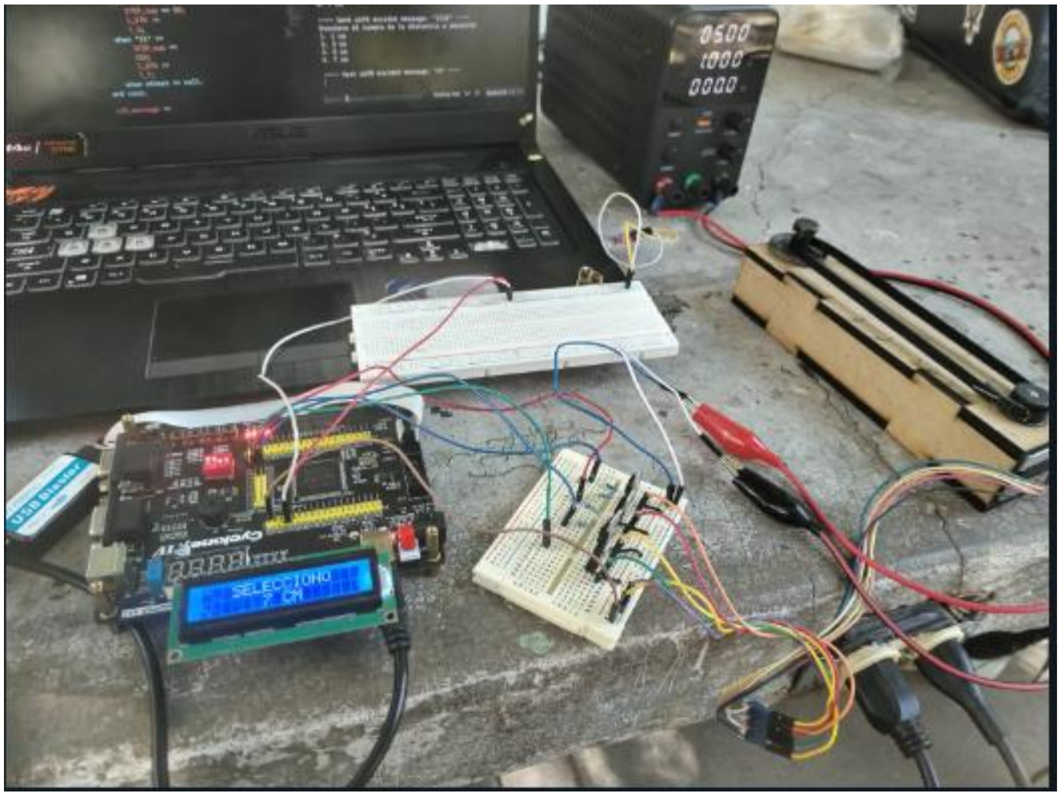


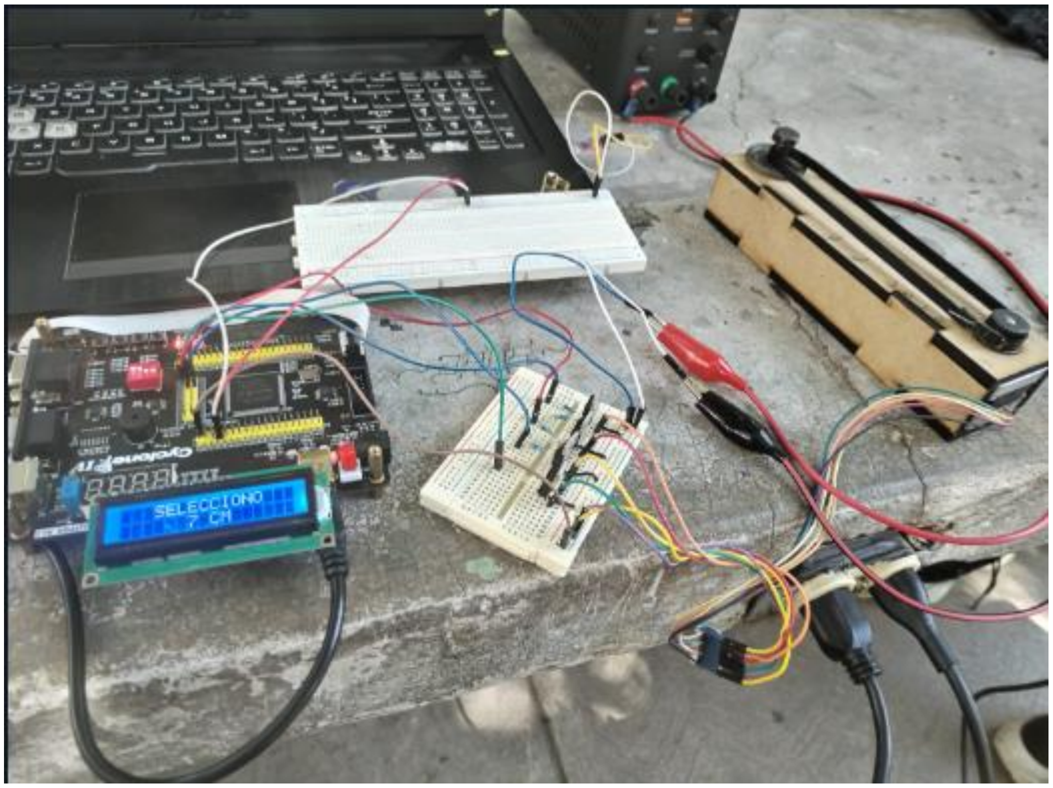
```
        component CLK_DIV is
            generic
            (
                clk_freq    : INTEGER          -- Frecuencia interna de FPGA (Hz)
            );
            port
            (
                i_out_freq      : INTEGER;              -- Frequencia deseada

                i_FPGA_clk      : in STD_LOGIC;         -- Señal de reloj base
                o_clk           : out STD_LOGIC
            );
        end component;

        signal P10_CLK : STD_LOGIC;
        constant P10_freq : INTEGER := 1000000;

    -- MAQUINA DE ESTADOS FINITOS --
        type states is (LCD_SEND, TX_SEND, PRINT_IDLE, IDLE, PRINT_COUPLING, COUPLING, WAIT
        signal act_state : states := PRINT_IDLE;

        signal LCD_nxt_state : states;
        signal TX_nxt_state : states;
        signal STOP_nxt_state : states;
```

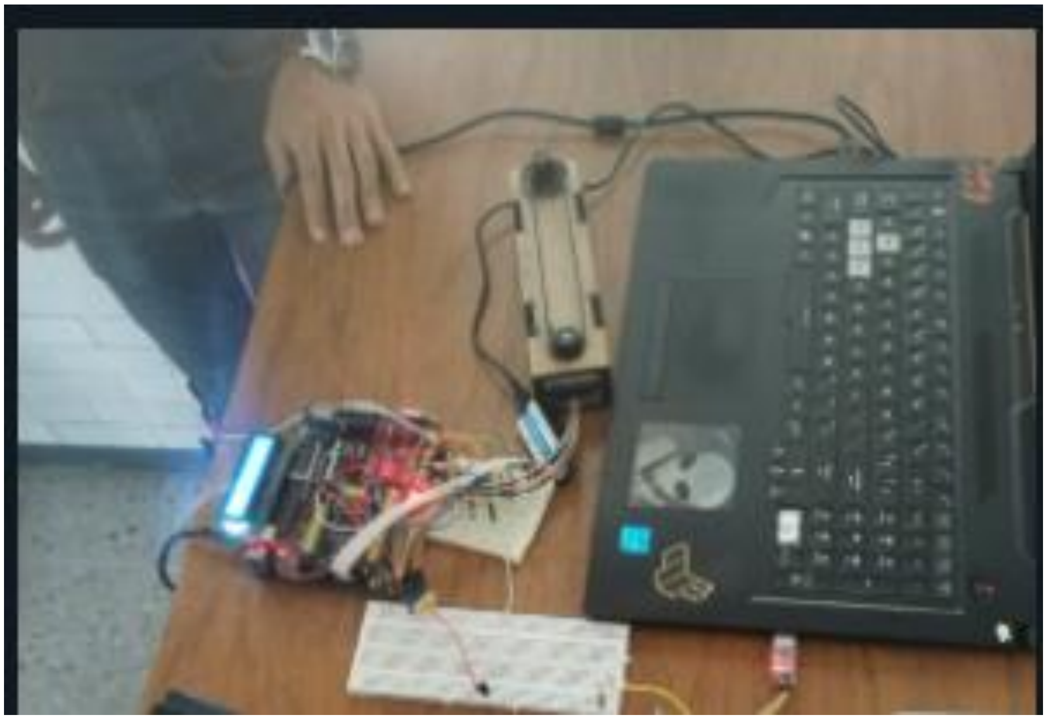


```
    -- UART --
        component UART_TX_string
            generic
            (
                baud_freq : INTEGER;
                clk_freq : INTEGER
            );
            port
            (
                i_CLK : in STD_LOGIC;
                i_RST : in STD_LOGIC;
                i_string : in STD_LOGIC_VECTOR(623 downto 0);
                i_ST : in STD_LOGIC;
                o_RDY : out STD_LOGIC;
                o_TX : out STD_LOGIC
            );
        end component;

        signal TX_message  : STD_LOGIC_VECTOR(623 downto 0);
        signal TX_ST       : STD_LOGIC := '0';

        signal TX_RDY      : STD_LOGIC;
```

Hardware of Stepper motor




SELECCIONO
7 CM



## Conclusion

The modular design of this project emphasizes the importance of managing each component individually to achieve a fully functional main code. By separating the UART, LCD, and motor control into distinct components, the system achieves greater flexibility, clarity, and ease of debugging, which are crucial in complex digital systems.

UART: Implementing the UART as a separate component ensures reliable data reception and transmission, allowing the FPGA to communicate effectively with external devices, such as a PC. This is essential for receiving commands or data and sending back confirmation messages, enabling smooth interaction with other systems.

LCD: The LCD component provides a real-time interface for displaying information, such as distance or status updates. By handling this display as an independent module, the system maintains a user-friendly interface without overloading the main code with display-specific logic.

Stepper Motor Control: Having dedicated motor control allows for precise and responsive step adjustments based on input commands. The modular motor component translates distance data into controlled movements, making it ideal for applications requiring exact positioning.

In summary, designing these components independently supports efficient integration in the main code, where each component's functionality is optimized. This modular approach not only enhances code readability and reusability but is also essential in applications such as automation, where UART, LCD, and motor control play pivotal roles in system interaction and control.