Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

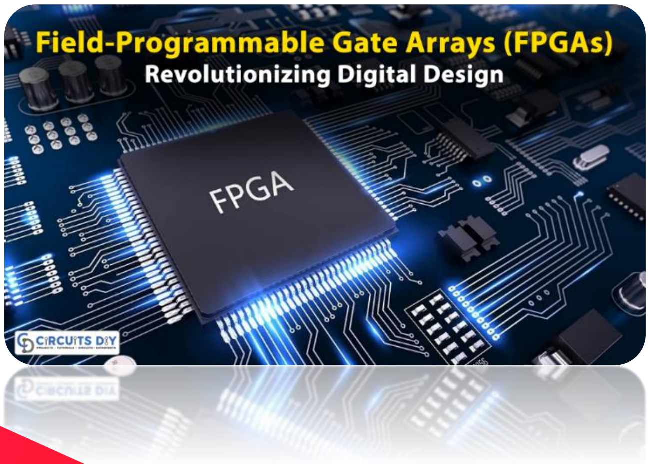
Faculty of Engineering.

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 11: DAC


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

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Introduction

In this practice, a Digital-to-Analog Converter (DAC) was implemented using an FPGA to demonstrate the conversion of digital signals into precise analog outputs. The project uses the MCP4802 DAC, controlled via the SPI protocol, to generate analog voltage levels based on digital inputs. Additionally, a 7-segment display is integrated to show the selected voltage level in real-time.

The design is structured into components:

- DAC Control: Manages communication with the MCP4802 DAC, sending 12-bit data for voltage configuration.
- Display Management: Handles the 7-segment display, providing visual feedback on the selected output voltage.
- Clock Divider: Generates the necessary clock signals for SPI communication and display multiplexing.
- This practice highlights the importance of combining hardware components like DACs and displays with the logic of an FPGA to create a versatile and programmable system for analog signal generation.

Methodology

Step 1: Clock Divider Design

A clock divider (CLK_DIV) was implemented to generate specific frequencies required for different system components.

- For SPI communication with the MCP4802 DAC, the clock frequency was set to 1 MHz.
- For the 7-segment display multiplexing, a frequency of 1 kHz was generated.

The clock divider adjusts the FPGA’s base clock (50 MHz) to these desired frequencies using a counter-based logic.

Step 2: DAC Communication Component

- A DAC_MCP4802 component was created to control the MCP4802 DAC using the SPI protocol.

➢ Data Flow:

1. On receiving a start signal (i_ST), the DAC configuration (i_DAC_CONF) and voltage (i_DAC_VOLT) are concatenated to form the 12-bit SPI data.
2. The 12-bit data is sent bit-by-bit to the DAC via the o_SDI pin, synchronized with the SPI clock (o_SCK).
3. A latch signal (o_LDAC) is triggered to update the DAC output after the data transmission.

- State Machine:

➢ IDLE: Waits for the start signal.

➢ WAIT_ST: Ensures no multiple activations occur.

➢ SEND_DATA: Sends 12 bits of data sequentially to the DAC.

➢ LATCH_DAC: Triggers the latch signal to finalize the DAC output update.

Step 3: Display Management Component

- The DISPLAY_PRINT component was developed to handle a 2-digit 7-segment display.

➢ Data Flow:

1. Input voltage levels (i_NUM0 and i_NUM1) are formatted into two 8-bit binary values representing decimal digits.
2. The values are multiplexed and displayed on the 7-segment display.

- State Machine:

1. NUM0: Activates the first digit and displays the lower voltage value.
2. NUM1: Activates the second digit and displays the higher voltage value.

Step 4: Main Code Integration

- The P11_DAC entity integrated the clock divider, DAC communication, and display components.

➢ The DAC voltage levels are predefined (DAC_VOLT) and mapped to specific button inputs (i_SEL).

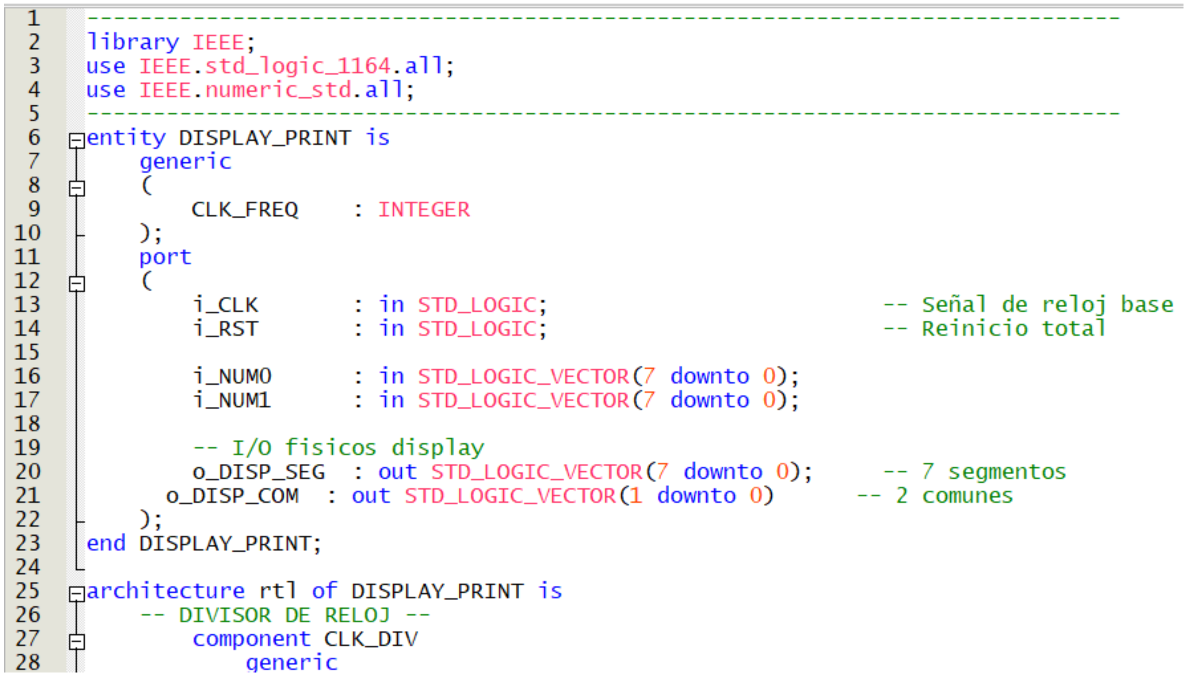
- On a start signal (i_ST):

1. The selected voltage level is updated on the 7-segment display using DISPLAY_PRINT.
2. The DAC is configured and updated with the selected voltage using DAC_MCP4802.

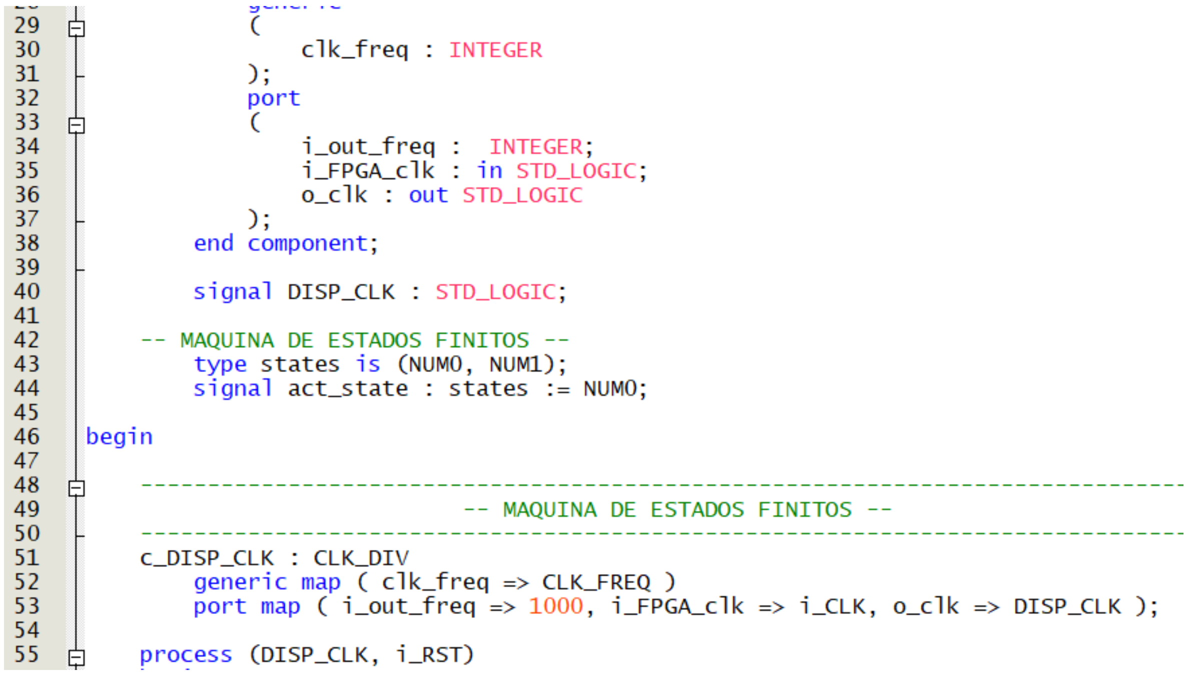
- A ready signal (o_RDY) indicates when the system is prepared for new inputs.

Results

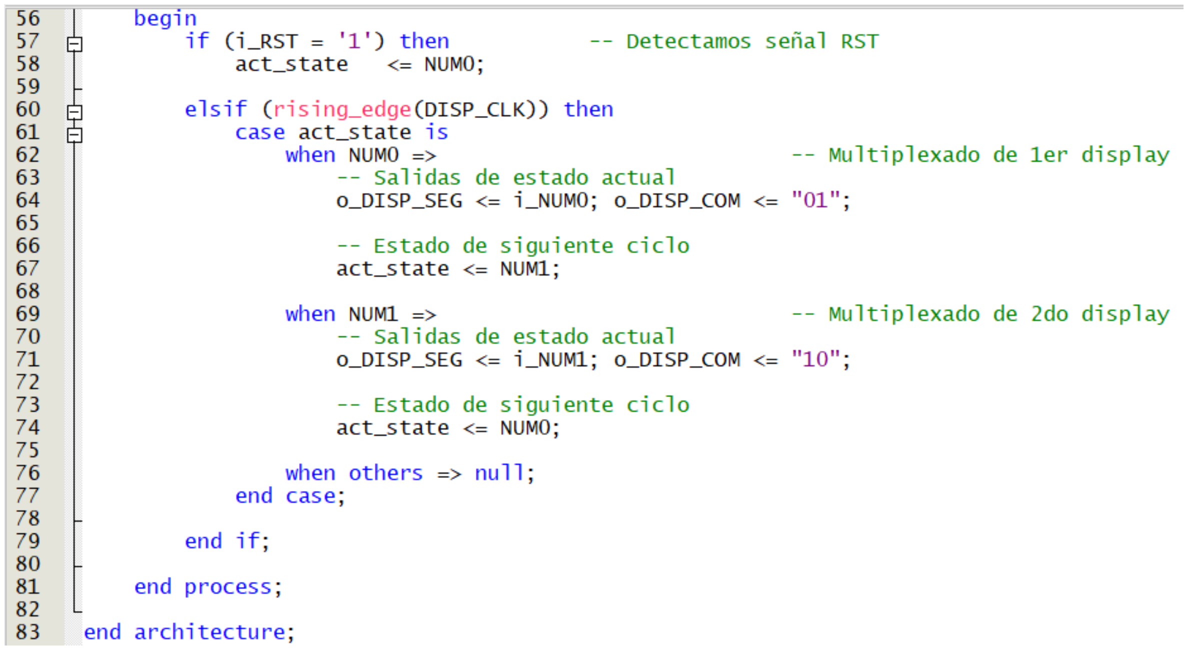
Base algorithm LCD



```vhdl
------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------------------------------------------------------------------------------
entity DISPLAY_PRINT is
    generic
    (
        CLK_FREQ    : INTEGER
    );
    port
    (
        i_CLK       : in STD_LOGIC;                         -- Señal de reloj base
        i_RST       : in STD_LOGIC;                         -- Reinicio total

        i_NUM0      : in STD_LOGIC_VECTOR(7 downto 0);
        i_NUM1      : in STD_LOGIC_VECTOR(7 downto 0);

        -- I/O fisicos display
        o_DISP_SEG  : out STD_LOGIC_VECTOR(7 downto 0);     -- 7 segmentos
      o_DISP_COM  : out STD_LOGIC_VECTOR(1 downto 0)      -- 2 comunes
    );
end DISPLAY_PRINT;

architecture rtl of DISPLAY_PRINT is
    -- DIVISOR DE RELOJ --
        component CLK_DIV
            generic
```



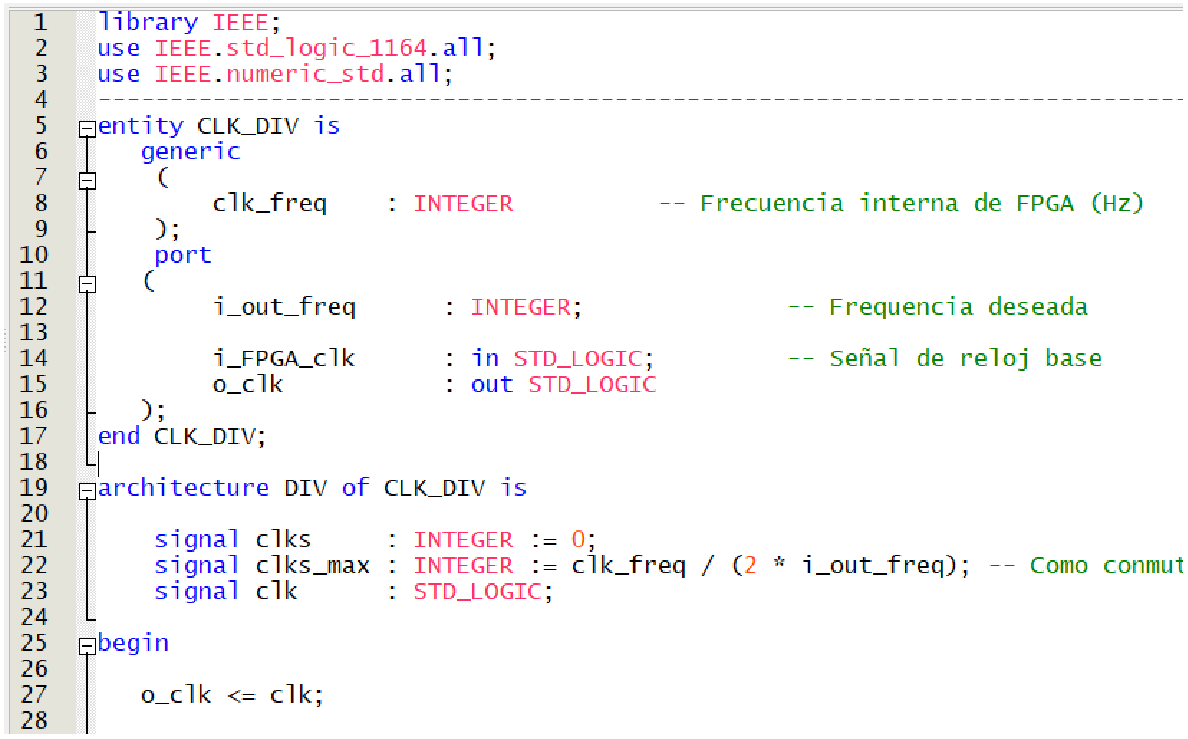
```
            (
                clk_freq : INTEGER
            );
            port
            (
                i_out_freq :  INTEGER;
                i_FPGA_clk : in STD_LOGIC;
                o_clk : out STD_LOGIC
            );
        end component;

        signal DISP_CLK : STD_LOGIC;

    -- MAQUINA DE ESTADOS FINITOS --
        type states is (NUM0, NUM1);
        signal act_state : states := NUM0;

begin

    -------------------------------------------------------------------------------
                             -- MAQUINA DE ESTADOS FINITOS --
    -------------------------------------------------------------------------------
    c_DISP_CLK : CLK_DIV
        generic map ( clk_freq => CLK_FREQ )
        port map ( i_out_freq => 1000, i_FPGA_clk => i_CLK, o_clk => DISP_CLK );

    process (DISP_CLK, i_RST)
```



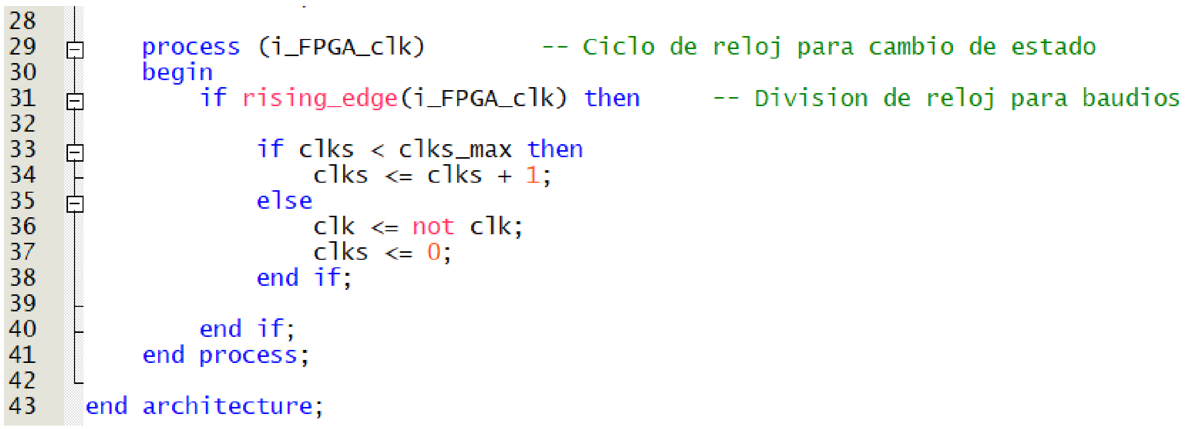
```
    begin
        if (i_RST = '1') then           -- Detectamos señal RST
            act_state   <= NUM0;

        elsif (rising_edge(DISP_CLK)) then
            case act_state is
                when NUM0 =>                        -- Multiplexado de 1er display
                    -- Salidas de estado actual
                    o_DISP_SEG <= i_NUM0; o_DISP_COM <= "01";

                    -- Estado de siguiente ciclo
                    act_state <= NUM1;

                when NUM1 =>                        -- Multiplexado de 2do display
                    -- Salidas de estado actual
                    o_DISP_SEG <= i_NUM1; o_DISP_COM <= "10";

                    -- Estado de siguiente ciclo
                    act_state <= NUM0;

                when others => null;
            end case;

        end if;

    end process;

end architecture;
```

Universidad Autónoma De Querétaro
Facultad de Ingeniería

Base algorithm CLK

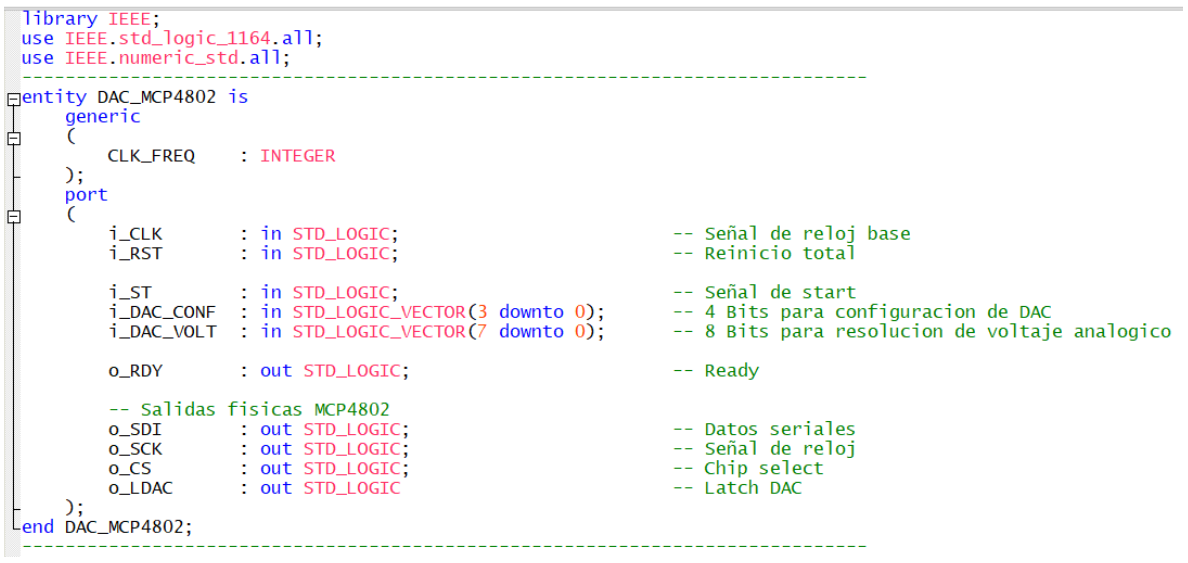


```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
-----------------------------------------------------------------------------
entity CLK_DIV is
   generic
    (
         clk_freq    : INTEGER          -- Frecuencia interna de FPGA (Hz)
    );
    port
   (
         i_out_freq      : INTEGER;              -- Frequencia deseada

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

architecture DIV of CLK_DIV is

    signal clks     : INTEGER := 0;
    signal clks_max : INTEGER := clk_freq / (2 * i_out_freq); -- Como conmut
    signal clk      : STD_LOGIC;

begin

   o_clk <= clk;

```

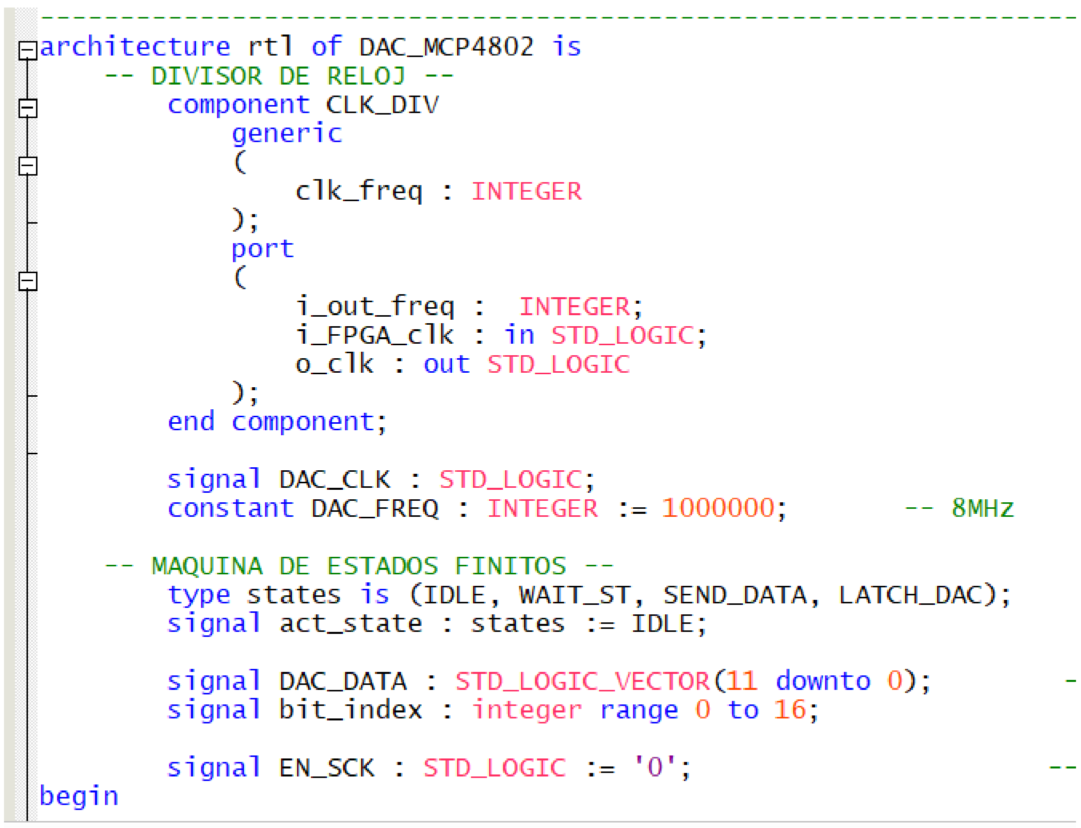


```

   process (i_FPGA_clk)        -- Ciclo de reloj para cambio de estado
   begin
       if rising_edge(i_FPGA_clk) then      -- Division de reloj para baudios

           if clks < clks_max then
               clks <= clks + 1;
           else
               clk <= not clk;
               clks <= 0;
           end if;

       end if;
   end process;

end architecture;
```

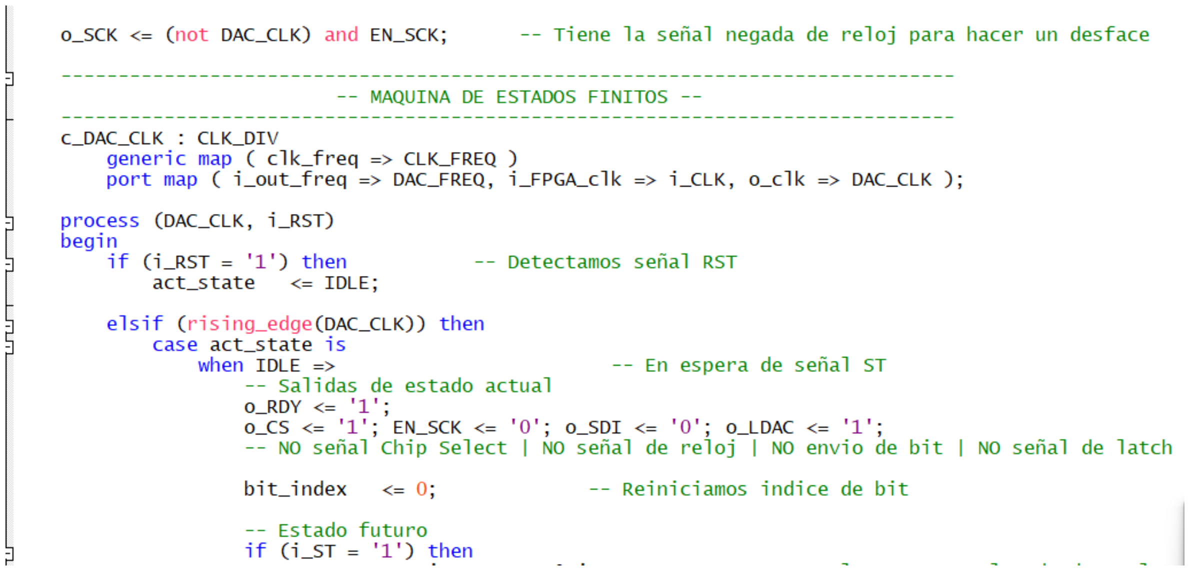
Base DAC



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------------------------------------------------------------------------------
entity DAC_MCP4802 is
    generic
    (
        CLK_FREQ     : INTEGER
    );
    port
    (
        i_CLK        : in STD_LOGIC;                          -- Señal de reloj base
        i_RST        : in STD_LOGIC;                          -- Reinicio total

        i_ST         : in STD_LOGIC;                          -- Señal de start
        i_DAC_CONF   : in STD_LOGIC_VECTOR(3 downto 0);       -- 4 Bits para configuracion de DAC
        i_DAC_VOLT   : in STD_LOGIC_VECTOR(7 downto 0);       -- 8 Bits para resolucion de voltaje analogico

        o_RDY        : out STD_LOGIC;                         -- Ready

        -- Salidas fisicas MCP4802
        o_SDI        : out STD_LOGIC;                         -- Datos seriales
        o_SCK        : out STD_LOGIC;                         -- Señal de reloj
        o_CS         : out STD_LOGIC;                         -- Chip select
        o_LDAC       : out STD_LOGIC                          -- Latch DAC
    );
end DAC_MCP4802;
------------------------------------------------------------------------------
```

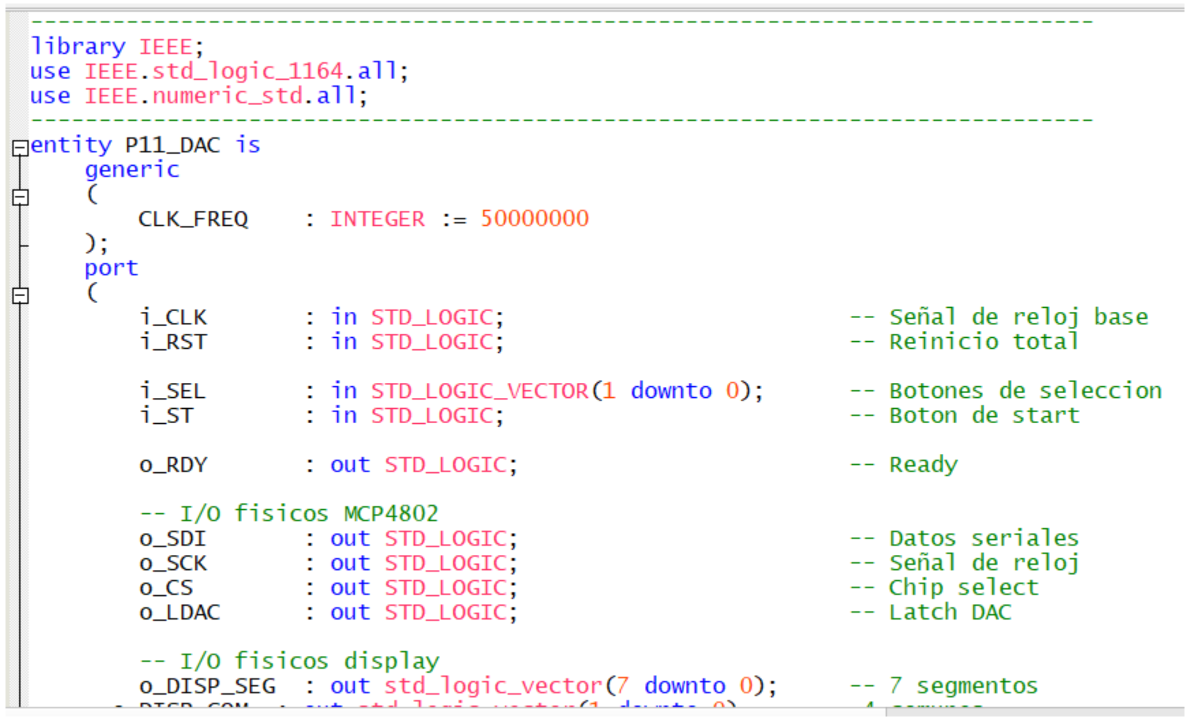


```
-----------------------------------------------------------------
architecture rtl of DAC_MCP4802 is
    -- DIVISOR DE RELOJ --
        component CLK_DIV
            generic
            (
                clk_freq : INTEGER
            );
            port
            (
                i_out_freq :  INTEGER;
                i_FPGA_clk : in STD_LOGIC;
                o_clk : out STD_LOGIC
            );
        end component;

        signal DAC_CLK : STD_LOGIC;
        constant DAC_FREQ : INTEGER := 1000000;        -- 8MHz

    -- MAQUINA DE ESTADOS FINITOS --
        type states is (IDLE, WAIT_ST, SEND_DATA, LATCH_DAC);
        signal act_state : states := IDLE;

        signal DAC_DATA : STD_LOGIC_VECTOR(11 downto 0);        -
        signal bit_index : integer range 0 to 16;

        signal EN_SCK : STD_LOGIC := '0';                      --
begin
```

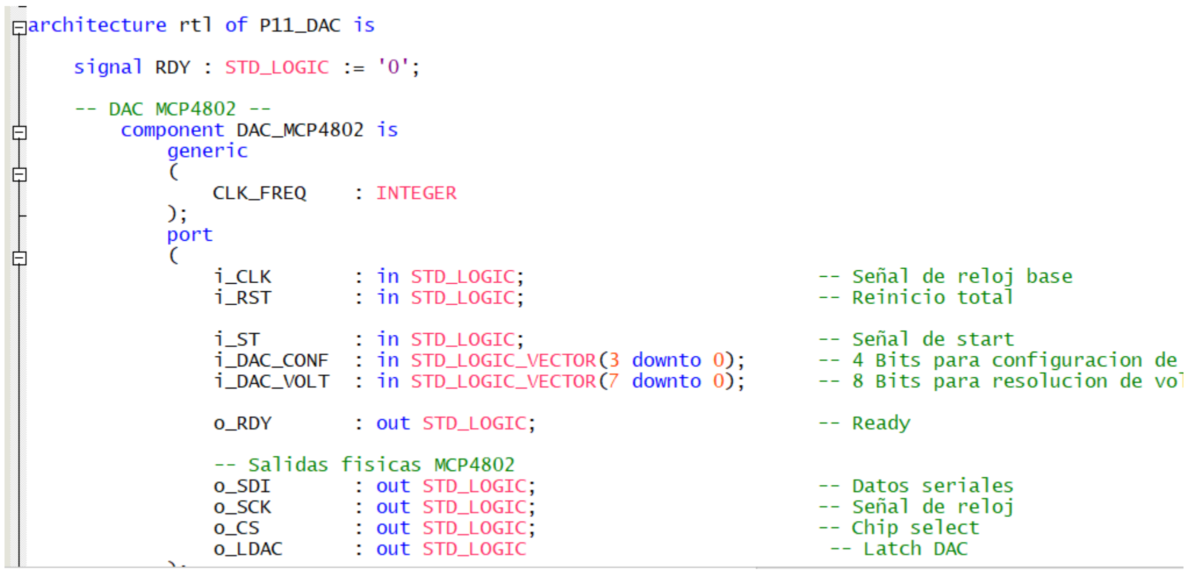


```
    o_SCK <= (not DAC_CLK) and EN_SCK;      -- Tiene la señal negada de reloj para hacer un desface

    ------------------------------------------------------------------------------
                            -- MAQUINA DE ESTADOS FINITOS --
    ------------------------------------------------------------------------------
    c_DAC_CLK : CLK_DIV
        generic map ( clk_freq => CLK_FREQ )
        port map ( i_out_freq => DAC_FREQ, i_FPGA_clk => i_CLK, o_clk => DAC_CLK );

    process (DAC_CLK, i_RST)
    begin
        if (i_RST = '1') then           -- Detectamos señal RST
            act_state    <= IDLE;

        elsif (rising_edge(DAC_CLK)) then
            case act_state is
                when IDLE =>                        -- En espera de señal ST
                    -- Salidas de estado actual
                    o_RDY <= '1';
                    o_CS <= '1'; EN_SCK <= '0'; o_SDI <= '0'; o_LDAC <= '1';
                    -- NO señal Chip Select | NO señal de reloj | NO envio de bit | NO señal de latch

                    bit_index    <= 0;              -- Reiniciamos indice de bit

                    -- Estado futuro
                    if (i_ST = '1') then
```

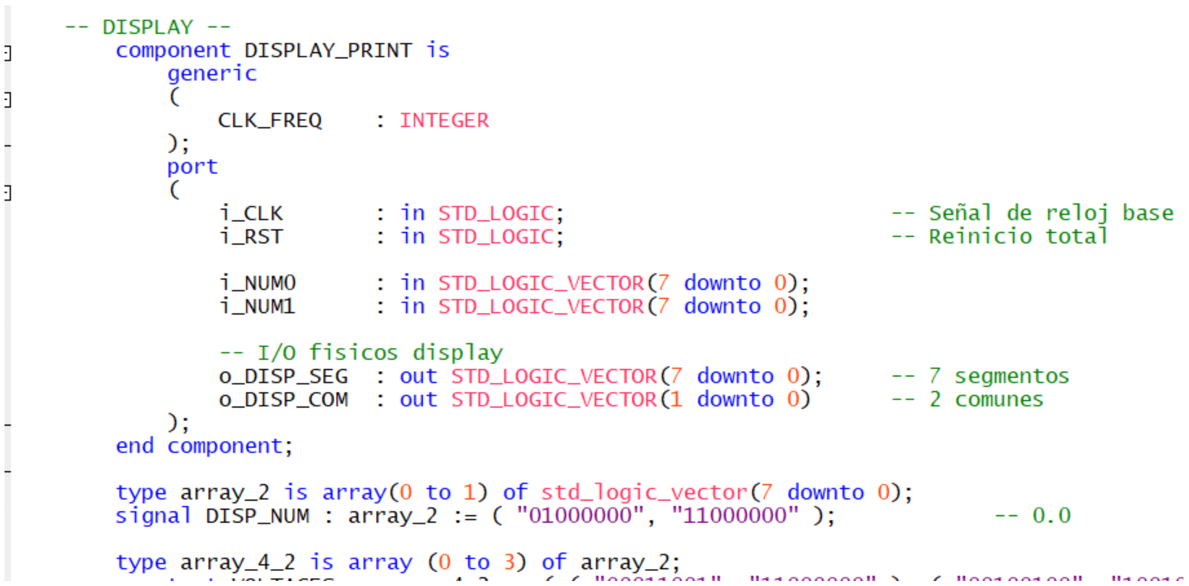
Base code



```
------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------------------------------------------------------------------------------
entity P11_DAC is
    generic
    (
        CLK_FREQ     : INTEGER := 50000000
    );
    port
    (
        i_CLK        : in STD_LOGIC;                          -- Señal de reloj base
        i_RST        : in STD_LOGIC;                          -- Reinicio total

        i_SEL        : in STD_LOGIC_VECTOR(1 downto 0);       -- Botones de seleccion
        i_ST         : in STD_LOGIC;                          -- Boton de start

        o_RDY        : out STD_LOGIC;                         -- Ready

        -- I/O fisicos MCP4802
        o_SDI        : out STD_LOGIC;                         -- Datos seriales
        o_SCK        : out STD_LOGIC;                         -- Señal de reloj
        o_CS         : out STD_LOGIC;                         -- Chip select
        o_LDAC       : out STD_LOGIC;                         -- Latch DAC

        -- I/O fisicos display
        o_DISP_SEG   : out std_logic_vector(7 downto 0);      -- 7 segmentos
```

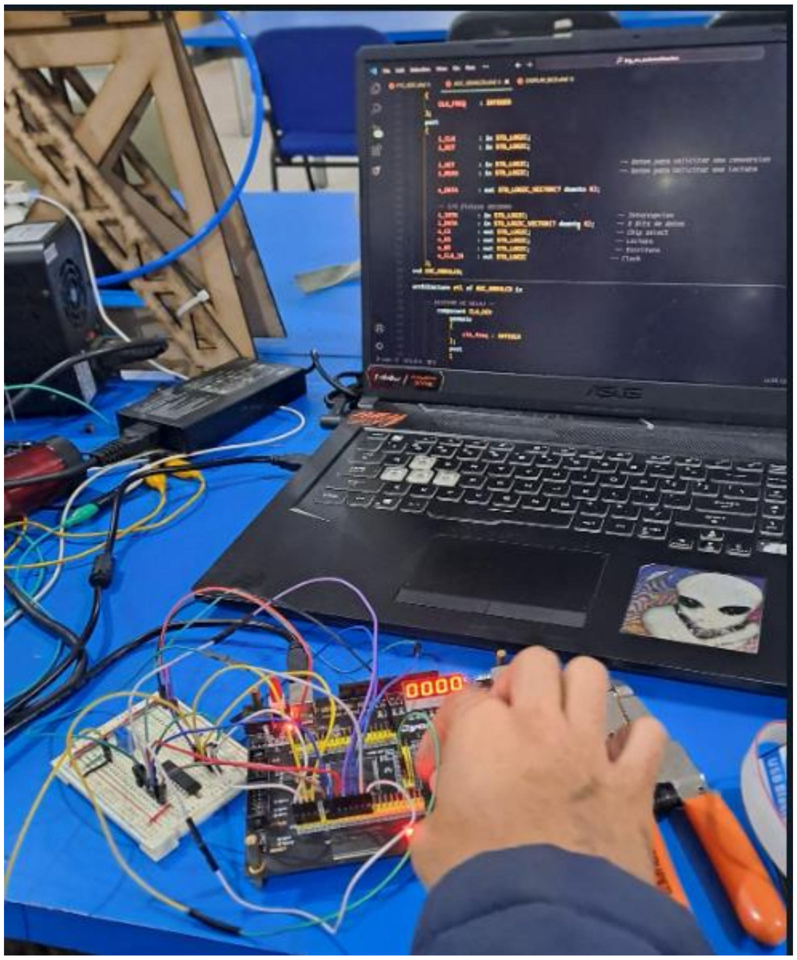


```
architecture rtl of P11_DAC is

    signal RDY : STD_LOGIC := '0';

    -- DAC MCP4802 --
        component DAC_MCP4802 is
            generic
            (
                CLK_FREQ     : INTEGER
            );
            port
            (
                i_CLK        : in STD_LOGIC;                         -- Señal de reloj base
                i_RST        : in STD_LOGIC;                         -- Reinicio total

                i_ST         : in STD_LOGIC;                         -- Señal de start
                i_DAC_CONF   : in STD_LOGIC_VECTOR(3 downto 0);      -- 4 Bits para configuracion de
                i_DAC_VOLT   : in STD_LOGIC_VECTOR(7 downto 0);      -- 8 Bits para resolucion de vo

                o_RDY        : out STD_LOGIC;                        -- Ready

                -- Salidas fisicas MCP4802
                o_SDI        : out STD_LOGIC;                        -- Datos seriales
                o_SCK        : out STD_LOGIC;                        -- Señal de reloj
                o_CS         : out STD_LOGIC;                        -- Chip select
                o_LDAC       : out STD_LOGIC                          -- Latch DAC
```



```
    -- DISPLAY --
        component DISPLAY_PRINT is
            generic
            (
                CLK_FREQ     : INTEGER
            );
            port
            (
                i_CLK        : in STD_LOGIC;                          -- Señal de reloj base
                i_RST        : in STD_LOGIC;                          -- Reinicio total

                i_NUM0       : in STD_LOGIC_VECTOR(7 downto 0);
                i_NUM1       : in STD_LOGIC_VECTOR(7 downto 0);

                -- I/O fisicos display
                o_DISP_SEG   : out STD_LOGIC_VECTOR(7 downto 0);      -- 7 segmentos
                o_DISP_COM   : out STD_LOGIC_VECTOR(1 downto 0)       -- 2 comunes
            );
        end component;

        type array_2 is array(0 to 1) of std_logic_vector(7 downto 0);
        signal DISP_NUM : array_2 := ( "01000000", "11000000" );          -- 0.0

        type array_4_2 is array (0 to 3) of array_2;
```

Hardware of DAC



Conclusion

The implementation of a Digital-to-Analog Converter (DAC) using an FPGA demonstrated the effective integration of digital control and analog signal generation. By dividing the design into modular components—clock management, DAC communication, and display handling—the system achieved precise and synchronized operation.

DAC Communication: The use of the MCP4802 DAC allowed for smooth and accurate conversion of digital signals to analog outputs, showcasing the importance of SPI protocol in interfacing with external devices.

Display Management: The 7-segment display provided real-time feedback, enabling users to visually confirm the selected voltage level, highlighting the role of human-readable interfaces in embedded systems.

Modular Design: The component-based approach ensured flexibility, scalability, and easier debugging, making it a robust foundation for future extensions or adaptations.

This practice underscores the significance of combining hardware and digital logic to bridge the gap between digital processing and analog control, with applications in fields such as signal processing, instrumentation, and control systems. The methodology and results demonstrate how FPGAs can effectively control complex systems with precision and efficiency.