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 12: ADC

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Introduction

In this practice, an Analog-to-Digital Converter (ADC) module, the ADC0804, was implemented and controlled using an FPGA to convert analog signals into digital outputs. The system uses state machines to manage the ADC's conversion and data read processes, while the converted data is displayed in real-time on a 7-segment display.

The Practice is divided into modular components to simplify the design and functionality:

- Clock Divider: Generates the clock signals needed for ADC operation and display multiplexing.
- ADC Control: Implements the communication protocol with the ADC0804, handling the conversion and retrieval of digital values from analog inputs.
- 7-Segment Display: Displays the converted digital value, decoded into a human-readable format using a Binary-Coded Decimal (BCD) system.

This practice highlights the essential role of ADCs in bridging the analog and digital domains, making it possible to process analog sensor data within digital systems. By leveraging the FPGA's capabilities, the system demonstrates precise control and real-time interaction between components.

Methodology

Step 1: Clock Divider Design

- A CLK_DIV module was implemented to generate the necessary clock signals for different system components.
- A 1 kHz clock signal was created to drive the ADC0804 (o_CLK_IN), ensuring proper sampling frequency for analog-to-digital conversion.
- Another 1 kHz clock was used for multiplexing the 7-segment display.

The clock divider adjusted the FPGA’s base clock (50 MHz) to these lower frequencies using a counter mechanism.

Step 2: ADC Control Component

- The ADC_0804LCN component was designed to manage the ADC0804 module.

➢ Data Flow:

1. When the conversion request (i_GET) is triggered, the component sends control signals to initiate an analog-to-digital conversion.
2. The ADC's interrupt signal (i_INTR) indicates the completion of the conversion.
3. Upon a read request (i_READ), the digital data (i_DATA) is retrieved and stored in an internal signal (o_DATA).

- State Machine:

➢ IDLE: Waits for a conversion (i_GET) or read (i_READ) request.
➢ GET_ADC: Sends a write signal to initiate the ADC conversion process.
➢ READ_ADC: Sends a read signal to retrieve the converted data and updates the output signal (o_DATA).

Step 3: Display Management Component

- The DISPLAY_BCD component was implemented to decode the 8-bit digital output from the ADC into a Binary-Coded Decimal (BCD) format and display it on a 4-digit, 7-segment display.

➢ Data Flow:

1. The ADC output is passed through a series of BCD decoders to generate decimal digits.
2. These digits are sent sequentially to the display using multiplexing, ensuring efficient use of the 7-segment display.

- State Machine:

1. NUM0 to NUM3: Activates each digit of the 4-digit display in sequence and updates its value.

Step 4: Main Code Integration

- The P12_ADC entity integrates the clock divider, ADC control, and display components:

➢ The clock divider provides the necessary clock signals for both the ADC and the display.

- The ADC control component manages conversion and data retrieval from the ADC0804.
- The display component decodes and visualizes the 8-bit digital output in real time.
- User interaction is managed via input buttons:
  - i_GET triggers an analog-to-digital conversion.
  - i_READ retrieves the converted data and updates the display.

Results

Base algorithm LCD

```
5   library IEEE;
6   use IEEE.std_logic_1164.all;
7   use IEEE.numeric_std.all;
8   ----------------------------------------------------------------------------------------------
9   entity DISPLAY_BCD is
10      generic
11      (
12          CLK_FREQ    : INTEGER
13      );
14      port
15      (
16          i_CLK       : in STD_LOGIC;
17          i_DATA      : in STD_LOGIC_VECTOR(7 downto 0);
18
19          -- I/O fisicos display
20          o_DISP_SEG  : out STD_LOGIC_VECTOR(7 downto 0);     -- 7 segmentos
21        o_DISP_COM  : out STD_LOGIC_VECTOR(3 downto 0)      -- 4 comunes
22      );
23  end DISPLAY_BCD;
```

```
25  architecture rtl of DISPLAY_BCD is
26
27      -- DIVISOR DE RELOJ --
28          component CLK_DIV
29              generic
30              (
31                  clk_freq : INTEGER
32              );
33              port
34              (
35                  i_out_freq :  INTEGER;
36                  i_FPGA_clk : in STD_LOGIC;
37                  o_clk : out STD_LOGIC
38              );
39          end component;
40          signal DISP_CLK : STD_LOGIC;
41          signal SUM_CLK : STD_LOGIC;
42
43      -- MULTIPLEXADO DE DISPLAY --
44          -- Secuencia de comunes de display
45          type array_4_t is array (0 to 3) of std_logic_vector(3 downto 0); -- Tipo para manejar los comunes
46          constant com_array : array_4_t :=
47          (
```

```
46          constant com_array : array_4_t :=
47          (
48              "1110",
49              "1101",
50              "1011",
51              "0111"
52          );
53          signal comun_index : integer range 0 to 4;
54
55          -- Impresion de digitos en display
56          type array_10_t is array (0 to 9) of std_logic_vector(7 downto 0);
57          constant num_array : array_10_t :=
58          (
59              "11000000",  -- 0
60              "11111001",  -- 1
61              "10100100",  -- 2
62              "10110000",  -- 3
63              "10011001",  -- 4
64              "10010010",  -- 5
65              "10000010",  -- 6
66              "11111000",  -- 7
67              "10000000",  -- 8
68              "10011000"   -- 9
69          );
70
71          type int_array is array (0 to 3) of integer;
72          signal numbers : int_array := (0, 0, 0, 0);
```

```
76          component BCD_4bit
77              port
78              (
79                  i_DATA : in std_logic_vector(3 downto 0);
80                  o_DATA : out std_logic_vector(3 downto 0)
81              );
82          end component;
83
84          signal BCD_DATA_0, BCD_DATA_1, BCD_DATA_2, BCD_DATA_3, BCD_DATA_4, BCD_DATA_5, BCD_DATA_6 : STD_LOG
85
86          signal BCD_DATA     : std_logic_vector(9 downto 0) := "0000000000";      -- Salida de decoder
87
88  begin
89
90      ---------------------------------------------------------------------------
91                              -- DECODER BCD 8 BITS --
92      ---------------------------------------------------------------------------
93      BCD_4bit_0 : BCD_4bit
94          port map ( i_DATA => '0' & i_DATA(7 downto 5), o_DATA => BCD_DATA_0 );
95      BCD_4bit_1 : BCD_4bit
96          port map ( i_DATA => BCD_DATA_0(2 downto 0) & i_DATA(4), o_DATA => BCD_DATA_1 );
97      BCD_4bit_2 : BCD_4bit
98          port map ( i_DATA => BCD_DATA_1(2 downto 0) & i_DATA(3), o_DATA => BCD_DATA_2 );
99      BCD_4bit_3 : BCD_4bit
100         port map ( i_DATA => '0' & BCD_DATA_0(3) & BCD_DATA_1(3) & BCD_DATA_2(3), o_DATA => BCD_DATA_3 );
101     BCD_4bit_4 : BCD_4bit
102         port map ( i_DATA => BCD_DATA_2(2 downto 0) & i_DATA(2), o_DATA => BCD_DATA_4 );
103     BCD_4bit_5 : BCD_4bit
```

```
114     ------------------------------------------------------------------------------
115                             -- MULTIPLEXADO DE DISPLAY --
116     ------------------------------------------------------------------------------
117     c_DISP_CLK : CLK_DIV                                        -- Divisor de reloj
118         generic map ( clk_freq => CLK_FREQ )
119         port map ( i_out_freq => 1000, i_FPGA_clk => i_CLK, o_clk => DISP_CLK);
120
121     multiplexado : process (DISP_CLK)
122     begin
123         if rising_edge(DISP_CLK) then
124             -- Enviamos informacion a display seleccionado
125             o_DISP_COM  <= com_array(comun_index);
126             o_DISP_SEG  <= num_array(numbers(comun_index));
127
128             -- Seleccionamos display para proxima impresion
129             if (comun_index = 3) then
130                 comun_index <= 0;
131             else
132                 comun_index <= comun_index + 1;
133             end if;
134
135         end if;
136
137     end process;
138
139 end rtl;
```

Base algorithm BCD

```
 1  library IEEE;
 2  use IEEE.std_logic_1164.all;
 3  use IEEE.numeric_std.all;
 4
 5  entity BCD_4bit is
 6      port
 7      (
 8          i_DATA      : in STD_LOGIC_VECTOR(3 downto 0);
 9          o_DATA      : out STD_LOGIC_VECTOR(3 downto 0)
10      );
11  end BCD_4bit;
12  ----------------------------------------------------------------------
13  architecture rtl of BCD_4bit is
14  begin
15      process (i_DATA)
16      begin
17          if (unsigned(i_DATA) >= 5) then
18              o_DATA <= STD_LOGIC_VECTOR(unsigned(i_DATA) + 3);
19          else
20              o_DATA <= i_DATA;
21          end if;
22
23      end process;
24
25  end rtl;
```

Base CLK

```vhdl
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 conmutamos el re
    signal clk      : STD_LOGIC;

    .
```

```vhdl
architecture DIV of CLK_DIV is

    signal clks     : INTEGER := 0;
    signal clks_max : INTEGER := clk_freq / (2 * i_out_freq); -- Como conmutamos el reloj ocu
    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 code ADC

```
 7  library IEEE;
 8  use IEEE.std_logic_1164.all;
 9  use IEEE.numeric_std.all;
10  ----------------------------------------------------------------------------------------------------
11  entity ADC_0804LCN is
12      generic
13      (
14          CLK_FREQ    : INTEGER
15      );
16      port
17      (
18          i_CLK       : in STD_LOGIC;
19          i_RST       : in STD_LOGIC;
20
21          i_GET       : in STD_LOGIC;                         -- Boton para solicitar una conversion
22          i_READ      : in STD_LOGIC;                         -- Boton para solicitar una lectura
23
24          o_DATA      : out STD_LOGIC_VECTOR(7 downto 0);
25
26          -- I/O fisicos ADC0804
27          i_INTR      : in STD_LOGIC;                         -- Interrupcion
28          i_DATA      : in STD_LOGIC_VECTOR(7 downto 0);      -- 8 Bits de datos
29          o_CS        : out STD_LOGIC;                        -- Chip select
30          o_RD        : out STD_LOGIC;                        -- Lectura
31          o_WR        : out STD_LOGIC;                        -- Escritura
32          o_CLK_IN    : out STD_LOGIC                        -- Clock
33      );
34  end ADC_0804LCN;
```

```
36  architecture rtl of ADC_0804LCN is
37
38      -- DIVISOR DE RELOJ --
39          component CLK_DIV
40              generic
41              (
42                  clk_freq : INTEGER
43              );
44              port
45              (
46                  i_out_freq :  INTEGER;
47                  i_FPGA_clk : in STD_LOGIC;
48                  o_clk : out STD_LOGIC
49              );
50          end component;
51
52          signal ADC_CLK : STD_LOGIC;
53
54      -- MAQUINA DE ESTADOS FINITOS --
55          type states is (IDLE, GET_ADC0, GET_ADC1, READ_ADC0, READ_ADC1, READ_ADC2);
56          signal act_state : states := IDLE;
57
58  begin
59
60      o_CLK_IN <= ADC_CLK;
61
```

```
62      ------------------------------------------------------------------------
63                              -- MAQUINA DE ESTADOS FINITOS --
64      ------------------------------------------------------------------------
65      c_ADC_CLK : CLK_DIV
66          generic map ( clk_freq => CLK_FREQ )
67          port map ( i_out_freq => 1000, i_FPGA_clk => i_CLK, o_clk => ADC_CLK );
68
69      process (ADC_CLK, i_RST)
70      begin
71          if (i_RST = '1') then           -- Detectamos señal RST
72              act_state   <= IDLE;
73
74          elsif (rising_edge(ADC_CLK)) then
75              case act_state is
76                  when IDLE =>
77                      -- Salidas del estado
78                      o_CS <= '1'; o_WR <= '1'; o_RD <= '1';
79                      -- NO chip select | NO write | NO read
80
81                      -- Estado proximo
82                      if (i_GET = '1') then                       -- Se solicito c
83                          act_state <= GET_ADC0;
84                      elsif (i_READ = '1' and i_INTR = '0') then  -- Se solicito l
85                          act_state <= READ_ADC0;
86                      else
87                          act_state <= IDLE;
88                      end if;
```

```
121                 when READ_ADC1 =>
122                     -- Salidas del estado
123                     o_CS <= '0'; o_WR <= '1'; o_RD <= '0';
124                     -- SI chip select | NO write | SI read
125
126                     -- Estado proximo
127                     act_state <= READ_ADC2;
128
129                 when READ_ADC2 =>
130                     -- Salidas del estado
131                     o_CS <= '0'; o_WR <= '1'; o_RD <= '1';
132                     -- SI chip select | NO write | SI read
133
134                     o_DATA <= i_DATA;
135
136                     -- Estado proximo
137                     act_state <= IDLE;
138
139                 when others => null;
140             end case;
141
142         end if;
143
144     end process;
145
146
147 end architecture;
148
```

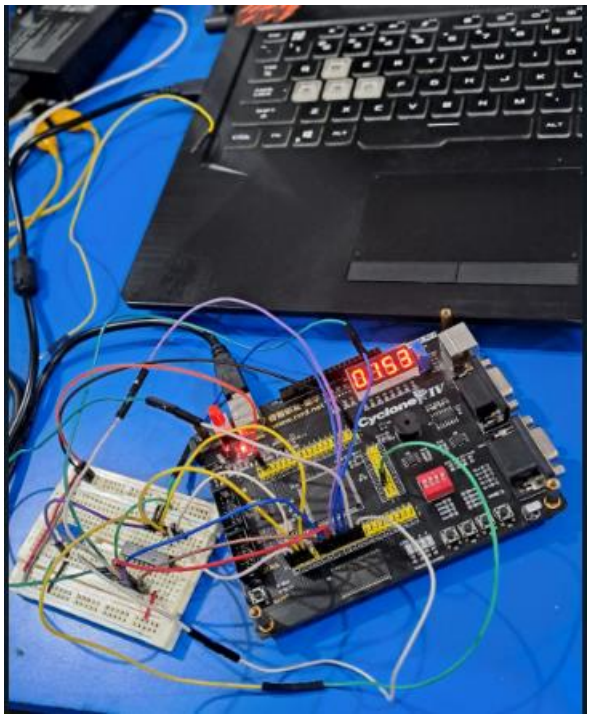
Base code main

```
 7  library IEEE;
 8  use IEEE.std_logic_1164.all;
 9  use IEEE.numeric_std.all;
10  ----------------------------------------------------------------------------------
11  entity P12_ADC is
12      generic
13      (
14          CLK_FREQ    : INTEGER := 50000000
15      );
16      port
17      (
18          i_CLK       : in STD_LOGIC;                         -- Señal de reloj bas
19          i_RST       : in STD_LOGIC;                         -- Reinicio total
20
21          i_GET       : in STD_LOGIC;                         -- Boton para solicit
22          i_READ      : in STD_LOGIC;                         -- Boton para solicit
23
24          -- I/O fisicos ADC0804
25          i_INTR      : in STD_LOGIC;                         -- Interrupcion
26          i_DATA      : in STD_LOGIC_VECTOR(7 downto 0);      -- 8 Bits de datos
27          o_CS        : out STD_LOGIC;                        -- Chip select
28          o_RD        : out STD_LOGIC;                        -- Lectura
29          o_WR        : out STD_LOGIC;                        -- Escritura
30          o_CLK_IN    : out STD_LOGIC;                        -- Clock
31
32          -- I/O fisicos display
33          o_DISP_SEG  : out std_logic_vector(7 downto 0);     -- 7 segmentos
34        o_DISP_COM  : out std_logic_vector(3 downto 0)      -- 4 comunes
35      );
36  end P12_ADC;
37
38  architecture rtl of P12_ADC is
39
40      signal RDY : STD_LOGIC := '0';
41
42      -- DIVISOR DE RELOJ --
43          component CLK_DIV
44              generic
45              (
46                  clk_freq : INTEGER
47              );
48              port
49              (
50                  i_out_freq :  INTEGER;
51                  i_FPGA_clk : in STD_LOGIC;
52                  o_clk : out STD_LOGIC
53              );
54          end component;
55          signal ADC_CLK : STD_LOGIC;
56
57      -- MAQUINA DE ESTADOS FINITOS --
58          type states is (IDLE);
59          signal act_state : states := IDLE;
60
61      -- DISPLAY CON DECODER BCD DE 8 BITS --
```

```
62          component DISPLAY_BCD
63              generic
64              (
65                  CLK_FREQ : INTEGER
66              );
67              port
68              (
69                  i_CLK : in STD_LOGIC;
70                  i_DATA : in STD_LOGIC_VECTOR(7 downto 0);
71                  o_DISP_SEG : out STD_LOGIC_VECTOR(7 downto 0);
72                  o_DISP_COM : out STD_LOGIC_VECTOR(3 downto 0)
73              );
74          end component;
75
76      -- Modulo ADC0804LCN --
77          component ADC_0804LCN
78              generic
79              (
80                  CLK_FREQ : INTEGER
81              );
82              port
83              (
84                  i_CLK : in STD_LOGIC;
85                  i_RST : in STD_LOGIC;
86                  i_GET : in STD_LOGIC;
87                  i_READ : in STD_LOGIC;
88                  o_DATA : out STD_LOGIC_VECTOR(7 downto 0);
```

```
89                  i_INTR : in STD_LOGIC;
90                  i_DATA : in STD_LOGIC_VECTOR(7 downto 0);
91                  o_CS : out STD_LOGIC;
92                  o_RD : out STD_LOGIC;
93                  o_WR : out STD_LOGIC;
94                  o_CLK_IN : out STD_LOGIC
95              );
96          end component;
97
98          signal DATA : STD_LOGIC_VECTOR(7 downto 0);
99
100 begin
101
102     ------------------------------------------------------------------------
103                 -- IMPRESION DE BITS DE ADC EN DISPLAY DE 7 SEG --
104     ------------------------------------------------------------------------
105     c_DISPLAY_BCD : DISPLAY_BCD
106         generic map ( clk_freq => CLK_FREQ )
107         port map ( i_CLK => i_CLK, i_DATA => DATA, o_DISP_SEG => o_DISP_SEG, o_DISP_COM => o_DISP_COM);
108
109     ------------------------------------------------------------------------
110                                 -- MODULO ADC0804LCN --
111     ------------------------------------------------------------------------
112     c_ADC_0804LCN : ADC_0804LCN
113         generic map ( CLK_FREQ => CLK_FREQ )
114         port map ( i_CLK => i_CLK, i_RST => not i_RST, i_GET => i_GET, i_READ => i_READ, o_DATA => DAT
115
116 end architecture;
```

Hardware of ADC



Conclusion

The implementation of an Analog-to-Digital Converter (ADC) system using an FPGA highlighted the critical role of ADCs in bridging analog inputs and digital processing. By modularizing the design into clock management, ADC control, and display components, the system achieved precise and efficient operation.

ADC Control: The ADC0804 integration demonstrated reliable conversion of analog signals into digital values, showcasing the importance of standardized communication protocols in interfacing with external modules.

Display Management: The use of a BCD decoder and multiplexed 7-segment display provided an intuitive and real-time visualization of the

converted data, emphasizing the need for user-friendly outputs in embedded systems.

Modular Design: Dividing the project into distinct components enabled seamless integration, scalability, and ease of debugging, which are essential for complex digital designs.