**Universidad Nacional Autónoma de México**

Facultad de Ingeniería

**Materia:** Diseño Digital VLSI

**Profesor:** Dr. Emiliano Ehecatl Garcia Unzueta

**Semestre:** 2026-1

**Grupo:** 1

**Proyecto:** Validador de código binario


**Integrantes:**

Jaramillo Rodriguez Leslie Citlali

Jiménez Ayala Yordi Josué

Salas Hernández Camila Alexandra


**Fecha de entrega:** 2 de diciembre del 2025

# 1 Objetivo

- Diseñar una FSM que realice un validador de código binario
- Implementar la solución con el método de ecuación algorítmica

# 2 Introducción

## 2.1 Máquina de estados

### 2.1.1 Concepto

Una máquina de estados es un proceso secuencial o autómata en el cual pueden intervenir las entradas (o el estado actual) $Q(t)$ de este sistema. Estos estados $Q(t)$ pueden definirse como un conjunto de variables dentro del sistema que contribuyen a la evolución del mismo. Si los estados son conocidos, se puede conocer la evolución y la salida de la máquina de estados en un momento dado.

Para construir una máquina de estados se dispone de un gráfico que muestra su evolución, conocido como **diagrama de estado**. Cada uno de los estados puede cambiar o evolucionar en función de las entradas y estado.

### 2.1.2 Métodos

Una máquina de estados se puede hacer a partir de diferentes métodos que definen los diferentes estados y la evolución del sistema, estos métodos se conocen como:

- Método de ecuación algorítmica
- Método aritmético. **Diagrama en el anexo.**
- Método tabular PS/NS (estado presente/estado futuro). **Tabla en el anexo.**

## 2.2 Método de ecuación algorítmica

Este proyecto se va a implementar a partir del método de ecuación algorítmica. En este método se utilizan pasos predefinidos para obtener las ecuaciones de excitación del flip-flop y posteriormente se dibuja el diagrama de estados. El diagrama obtenido se utiliza para escribir el código adecuado en VHDL para el diseño.

Todo inicia modelando el sistema con un diagrama de estados y su tabla de transición para identificar cómo las variables actuales ($Q$) deben evolucionar a sus estados futuros ($Q^+$). Posteriormente, se esquematiza el circuito representando la lógica de control conocida como "nubes de lógica combinacional" (**Figura 1**) que alimentan a los Flip-Flops (tipo D).

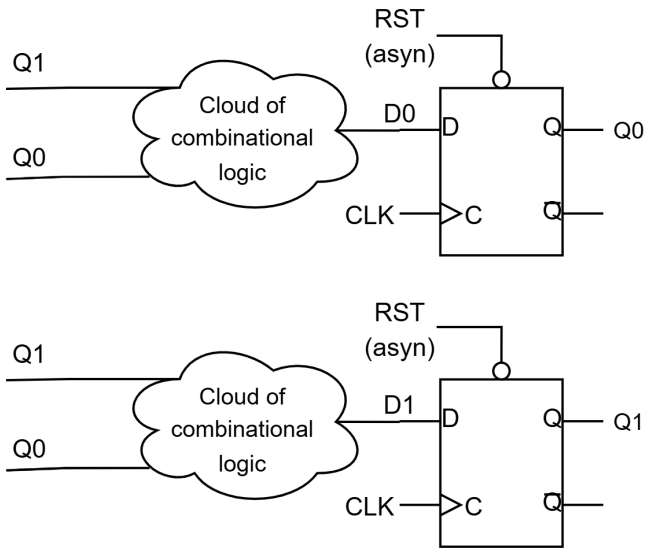


Figura 1: Nubes de lógica combinacional

El paso central del método consiste en resolver estas nubes, se analiza la tabla de verdad asumiendo que la excitación necesaria ($D$) es idéntica al estado futuro deseado, lo que permite extraer las ecuaciones booleanas para cada entrada del Flip-Flop en función de los estados presentes. Finalmente, se reemplazan las nubes dibujando las compuertas lógicas resultantes y se traduce esa estructura directamente a código VHDL, describiendo las ecuaciones concurrentes y los procesos secuenciales del reloj.

# 3 Desarrollo

## 3.1 Validador de código binario.

FSM analiza un valor binario (4 bits) ingresado. Muestra en display si es par/impar. El objetivo de este proyecto es crear un validador de códigos binarios mediante una FSM. El usuario ingresa un código binario de 4 bits, y el sistema determina si el número es par o impar. El resultado se muestra en un display. El sistema analiza los 4 bits ingresados. Dependiendo del bit menos significativo (LSB), determina si el número es par (LSB=0) o impar (LSB=1).

## 3.2 Análisis

Para solucionar este ejercicio, se plantea que los dispalys se encuentren apagados en el estado de espera, para proceder a la validación se va a presionar un botón para el cambio de estado, una vez en el estado de validación, con base en el LSB, si es 0 pasará al estado de visualización de par, mientras que si es 1 pasará al estado de visualización de impar, una vez termine la visualización se pasará al estado de espera nuevamente.

Con lo anterior, contamos con 4 posibles estados para el diagrama de estados, **00** es el estado de **espera**, **01** es el estado de **validación** y los estados **10** y **11** son los estados de **visualización** para par e impar respectivamente.

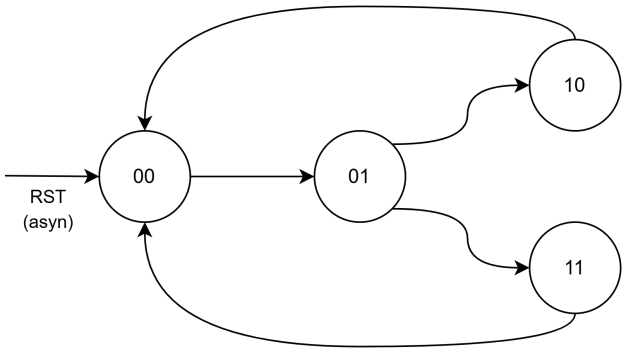


Figura 2: Diagrama de estados, método de ecuación algorítmica

Es importante mencionar que, como necesitamos contar hasta 3 por nuestros estados (00, 01, 10 y 11) vamos a requerir de 2 flip flops para nuestra implementación.

Ya con todo esto, realizamos la tabla con las posibles transiciones, considerando que, como entrada tambien tenemos un botón (para realizar la validación) y el switch de LSB (el número es de 4 bits, sin embargo con el que realizamos la validación es con el menos significativo y por ello solo requerimos de este para la tabla). **Nota.** El botón usa lógica negada.

| **PS** | | **Botón** | **Switch** | **NS** | | **Salida** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | $Q_0$ | $B$ | $S_0$ | $Q_1^+$ | $Q_0^+$ | Disp6 | Disp5 | Disp4 | Disp3 | Disp2 | Disp1 |
| 0 | 0 | x | x | 0 | 0 | - | - | - | - | - | - |
| 0 | 0 | 0 | x | 0 | **1** | - | - | - | - | - | - |
| 0 | 1 | x | 0 | **1** | 0 | - | - | - | - | - | - |
| 0 | 1 | x | 1 | **1** | **1** | - | - | - | - | - | - |
| 1 | 0 | x | x | 0 | 0 | Num | - | - | P | A | r |
| 1 | 1 | x | x | 0 | 0 | Num | I | n | P | A | r |

Tabla de posibles transiciones

Lo que se planteó para realizar la tabla fue que

- Mientras no se presione el botón, nos mantenemos en el estado de espera (**00**)
- Si se presiona el boton (ya tiene que haber una combinacion de 4 bits para el número) pasamos al estado de validación (**00** $\rightarrow$ **01**)

- En el estado de validación, a partir del valor de LSB se pasa al estado de par o impar
  - Si LSB es 0 se pasa al estado par (**01** $\rightarrow$ **10**)
  - Si LSB es 1 se pasa al estado impar (**01** $\rightarrow$ **11**)
- Luego de mostrar el resultado en el estado de visualizar (ya sea par o impar) por un tiempo, se regresa al estado de espera (**10** $\rightarrow$ **00** o **11** $\rightarrow$ **00**)

Nótese que, en la parte de **NS** tenemos los 1 resaltados en negro ya que con estos vamos a plantear nuestras ecuaciones de excitación D. Al emplear 2 flip flops, vamos a tener 2 ecuaciones ($D_0$ y $D_1$). Recordemos que

$$D \;=\; Q^{+} \;=\; Q(NS)$$

### 3.2.1 Ecuación D1

$$D_1 \;=\; Q_1^{+} \;=\; \overline{Q_1}Q_0 \;+\; \overline{Q_1}Q_0$$

$$D_1 \;=\; \overline{Q_1}Q_0$$

### 3.2.2 Ecuación D0

$$D_0 \;=\; Q_0^{+} \;=\; \overline{Q_1}\;\overline{Q_0}\;\overline{B} \;+\; \overline{Q_1}\;Q_0\;S_0$$

$$D_0 \;=\; \overline{Q_1}\;(\overline{Q_0}\;\overline{B} \;+\; Q_0\;S_0)$$

De modo que estas ecuaciones son las de las nubes de lógica combinacional de cada flip flop, quedando de la siguiente manera

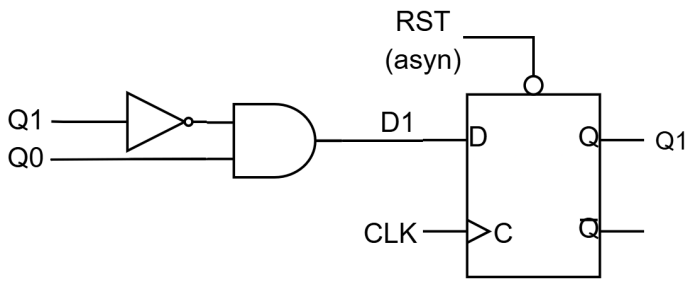


Figura 3: Circuito de la nube de lógica combinacional $D_1$

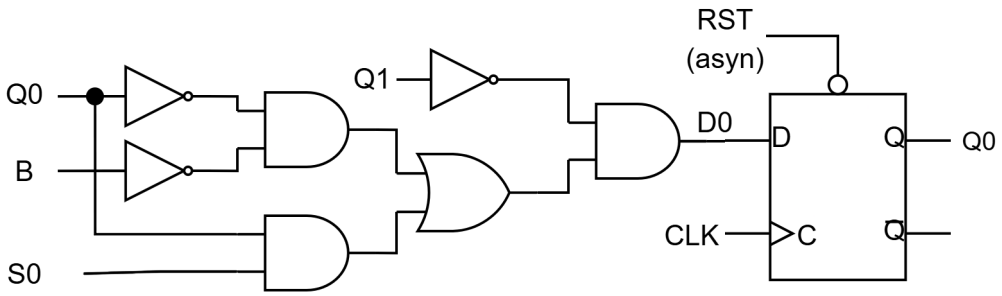


Figura 4: Circuito de la nube de lógica combinacional $D_0$

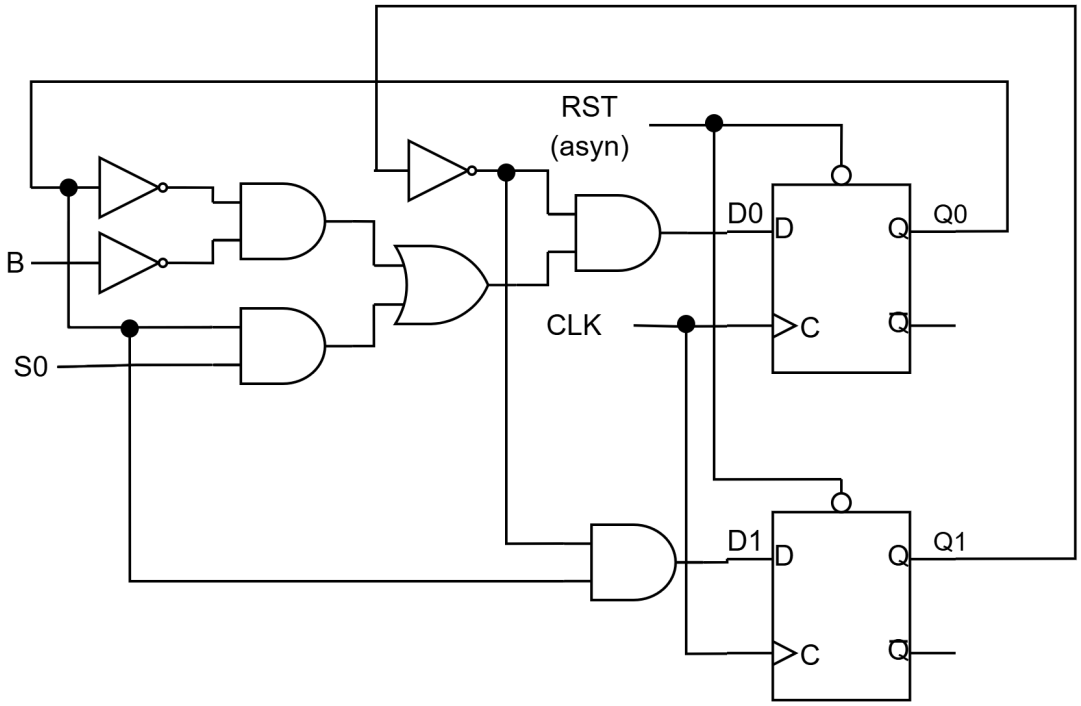


Figura 5: Circuito final

Con este circuito ya podemos pasar a la programación de nuestra FSM.

## 3.3 Código

### 3.3.1 Entidad

En nuestra entidad vamos a declarar 4 entradas (el reloj, el reset, el boton para activar la validación y S que es un vector para 4 bits que representaran 4 switches para el número binario que se va a ingresar) y 6 salidas (los 6 displays para mostrar el número y el mensaje de si es par o impar).

```
entity Proyecto is
    Port (
        clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        B   : in  STD_LOGIC;
        S   : in  STD_LOGIC_VECTOR(3 downto 0);
        Display1 : out STD_LOGIC_VECTOR(6 downto 0);
        Display2 : out STD_LOGIC_VECTOR(6 downto 0);
        Display3 : out STD_LOGIC_VECTOR(6 downto 0);
        Display4 : out STD_LOGIC_VECTOR(6 downto 0);
        Display5 : out STD_LOGIC_VECTOR(6 downto 0);
        Display6 : out STD_LOGIC_VECTOR(6 downto 0));
end Proyecto;
```

### 3.3.2 Arquitectura

Primero declaramos las señales que vamos a usar

```
signal q1, q0 : STD_LOGIC;
signal d1, d0 : STD_LOGIC;
signal num : STD_LOGIC_VECTOR(3 downto 0);
signal counter : integer range 0 to 100000000 := 0;
```

- **q1**, **q0**, **d1**, y **d0** son parte de las entradas y salidas de los flip flops.
- **num** va a almacenar el numero ingresado antes de presionar el boton para que se muestre en pantalla (hacemos esto para que no haya conflicto y se muestre un número diferente por mover los switches).
- **counter** va a ser un contador que va a estar activo cuando estemos en el estado de visualizar y que el mensaje se vea por un tiempo considerable.

Luego, ya dentro del begin, declaramos nuestras ecuaciones que obtuvimos en el análisis previo

```
d1 <= (not q1) and q0;
d0 <= not q1 and ((not q0 and not B) or (q0 and S(0)));
```

Seguido de esto, vamos a realizar un process en donde vamos a integrar lo que previamente hicimos con los flip flops, por lo que este process va a ser el encargado de los cambios de estados, a su vez, vamos a colocar un contador que se va a activar cuando $q_1 = 1$ puesto que esto nos indica que estamos en un estado de visualización (par $\rightarrow$ 10 o impar $\rightarrow$ 11) para que el mensaje se muestre por un tiempo mas largo antes de pasar nuevamente al estado de espera.

```
process(clk, rst)
begin
    if rst = '0' then
        q1 <= '0';
        q0 <= '0';
        num <= "0000";
        counter <= 0;
    elsif rising_edge(clk) then
        if (q1 = '1') then
            if counter < 100000000 then
                counter <= counter + 1;
                q1 <= q1;
                q0 <= q0;
            else
                counter <= 0;
                q1 <= d1;
                q0 <= d0;
            end if;
        else
            counter <= 0;
            q1 <= d1;
            q0 <= d0;
            if (q1 = '0' and q0 = '0' and B = '0') then
                num <= S;
            end if;
        end if;
    end if;
end process;
```

En este mismo process introducimos la siguiente sentencia

```
if (q1 = '0' and q0 = '0' and B = '0') then
    num <= S;
end if;
```

Esto quiere decir que si estamos en el estado de espera **00** ($Q_1 = 0$ y $Q_0 = 0$) y presionamos el botón ($B = 0$) vamos a almacenar el valor que tenga S en num, esto porque, el número se va a mostar hasta el estado de visualización y lo queremos es asegurarnos de que el numero ingresado antes de apretar el botón sea el mismo que se muestre en el display una vez que se valide.

Después tenemos un process para la visualización del número que se ingresó, como solo debe mostrarse cuando estamos en el estado de visualización se agrega un if para que cuando $q_1 = 1$ se muestre el número, si no se cumple entonces el display estará apagado.

```
process(q1, num)
begin
    if (q1 = '1') then
        case num is
            when "0000" => Display6 <= "1000000"; -- 0
            when "0001" => Display6 <= "1111001"; -- 1
            when "0010" => Display6 <= "0100100"; -- 2
            when "0011" => Display6 <= "0110000"; -- 3
            when "0100" => Display6 <= "0011001"; -- 4
            when "0101" => Display6 <= "0010010"; -- 5
            when "0110" => Display6 <= "0000010"; -- 6
            when "0111" => Display6 <= "1111000"; -- 7
            when "1000" => Display6 <= "0000000"; -- 8
            when "1001" => Display6 <= "0010000"; -- 9
            when "1010" => Display6 <= "0001000"; -- A
            when "1011" => Display6 <= "0000011"; -- b
            when "1100" => Display6 <= "1000110"; -- C
            when "1101" => Display6 <= "0100001"; -- d
            when "1110" => Display6 <= "0000110"; -- E
            when "1111" => Display6 <= "0001110"; -- F
            when others => Display6 <= "1111111"; -- Apagado
        end case;
    else
        Display6 <= "1111111";
    end if;
end process;
```

Por último, tenemos un process para visualizar si es par o impar el número, esto lo realizamos con un case para 10 (par) e impar (11), antes de esto, debemos concatenar q1 y q0 puesto que los manejamos como dos señales independientes y case solo evalua una sola señal. una vez esto implementamos el case con cada estado de visualización.

```
process(q1, q0)
    variable ps : STD_LOGIC_VECTOR(1 downto 0);
begin
    ps := q1 & q0;

    Display5 <= "1111111"; Display4 <= "1111111"; Display3 <= "1111111";
    ↪  Display2 <= "1111111"; Display1 <= "1111111";

    case ps is
        when "10" =>
            Display3 <= "0001100"; -- P
            Display2 <= "0001000"; -- A
            Display1 <= "0101111"; -- r

```

```
        when "11" =>
            Display5 <= "1111001"; -- I
            Display4 <= "0101011"; -- n
            Display3 <= "0001100"; -- P
            Display2 <= "0001000"; -- A
            Display1 <= "0101111"; -- r

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

Ya con todo esto explicado, podemos mostrar todo el código completo.

### 3.3.3 Código completo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Proyecto is
    Port (
        clk      : in  STD_LOGIC;
        rst      : in  STD_LOGIC;
        B        : in  STD_LOGIC;
        S        : in  STD_LOGIC_VECTOR(3 downto 0);
        Display1 : out STD_LOGIC_VECTOR(6 downto 0);
        Display2 : out STD_LOGIC_VECTOR(6 downto 0);
        Display3 : out STD_LOGIC_VECTOR(6 downto 0);
        Display4 : out STD_LOGIC_VECTOR(6 downto 0);
        Display5 : out STD_LOGIC_VECTOR(6 downto 0);
        Display6 : out STD_LOGIC_VECTOR(6 downto 0)
    );
end Proyecto;

architecture Behavioral of Proyecto is

    signal q1, q0  : STD_LOGIC;
    signal d1, d0  : STD_LOGIC;
    signal num     : STD_LOGIC_VECTOR(3 downto 0);
    signal counter : integer range 0 to 100000000 := 0;

begin

    d1 <= (not q1) and q0;
    d0 <= not q1 and ((not q0 and not B) or (q0 and S(0)));

    process(clk, rst)
    begin
```

```
33          if rst = '0' then
34              q1      <= '0';
35              q0      <= '0';
36              num     <= "0000";
37              counter <= 0;
38
39          elsif rising_edge(clk) then
40              if (q1 = '1') then
41                  if counter < 100000000 then
42                      counter <= counter + 1;
43                      q1      <= q1;
44                      q0      <= q0;
45                  else
46                      counter <= 0;
47                      q1      <= d1;
48                      q0      <= d0;
49                  end if;
50              else
51                  counter <= 0;
52                  q1      <= d1;
53                  q0      <= d0;
54
55                  if (q1 = '0' and q0 = '0' and B = '0') then
56                      num <= S;
57                  end if;
58              end if;
59          end if;
60      end process;
61
62      process(q1, num)
63      begin
64          if (q1 = '1') then
65              case num is
66                  when "0000" => Display6 <= "1000000"; -- 0
67                  when "0001" => Display6 <= "1111001"; -- 1
68                  when "0010" => Display6 <= "0100100"; -- 2
69                  when "0011" => Display6 <= "0110000"; -- 3
70                  when "0100" => Display6 <= "0011001"; -- 4
71                  when "0101" => Display6 <= "0010010"; -- 5
72                  when "0110" => Display6 <= "0000010"; -- 6
73                  when "0111" => Display6 <= "1111000"; -- 7
74                  when "1000" => Display6 <= "0000000"; -- 8
75                  when "1001" => Display6 <= "0010000"; -- 9
76                  when "1010" => Display6 <= "0001000"; -- A
77                  when "1011" => Display6 <= "0000011"; -- b
78                  when "1100" => Display6 <= "1000110"; -- C
79                  when "1101" => Display6 <= "0100001"; -- d
80                  when "1110" => Display6 <= "0000110"; -- E
```

```
81                  when "1111" => Display6 <= "0001110"; -- F
82                  when others => Display6 <= "1111111"; -- Apagado
83              end case;
84          else
85              Display6 <= "1111111";
86          end if;
87      end process;
88
89      process(q1, q0)
90          variable ps : STD_LOGIC_VECTOR(1 downto 0);
91      begin
92          ps := q1 & q0;
93
94          Display5 <= "1111111"; Display4 <= "1111111"; Display3 <= "1111111";
            ↪ Display2 <= "1111111"; Display1 <= "1111111";
95
96          case ps is
97              when "10" =>
98                  Display3 <= "0001100"; -- P
99                  Display2 <= "0001000"; -- A
100                 Display1 <= "0101111"; -- r
101
102             when "11" =>
103                 Display5 <= "1111001"; -- I
104                 Display4 <= "0101011"; -- n
105                 Display3 <= "0001100"; -- P
106                 Display2 <= "0001000"; -- A
107                 Display1 <= "0101111"; -- r
108
109             when others => null;
110         end case;
111     end process;
112
113 end Behavioral;
```

## 3.4 Asignación de pines

| Node Name | Direction | Location |
|---|---|---|
| B | Input | PIN_B8 |
| clk | Input | PIN_P11 |
| rst | Input | PIN_A7 |
| S[3] | Input | PIN_C12 |
| S[2] | Input | PIN_D12 |
| S[1] | Input | PIN_C11 |
| S[0] | Input | PIN_C10 |

| | | |
|---|---|---|
| Display1[6] | Output | PIN_C17 |
| Display1[5] | Output | PIN_D17 |
| Display1[4] | Output | PIN_E16 |
| Display1[3] | Output | PIN_C16 |
| Display1[2] | Output | PIN_C15 |
| Display1[1] | Output | PIN_E15 |
| Display1[0] | Output | PIN_C14 |
| Display2[6] | Output | PIN_B17 |
| Display2[5] | Output | PIN_A18 |
| Display2[4] | Output | PIN_A17 |
| Display2[3] | Output | PIN_B16 |
| Display2[2] | Output | PIN_E18 |
| Display2[1] | Output | PIN_D18 |
| Display2[0] | Output | PIN_C18 |
| Display3[6] | Output | PIN_B22 |
| Display3[5] | Output | PIN_C22 |
| Display3[4] | Output | PIN_B21 |
| Display3[3] | Output | PIN_A21 |
| Display3[2] | Output | PIN_B19 |
| Display3[1] | Output | PIN_A20 |
| Display3[0] | Output | PIN_B20 |
| Display4[6] | Output | PIN_E17 |
| Display4[5] | Output | PIN_D19 |
| Display4[4] | Output | PIN_C20 |
| Display4[3] | Output | PIN_C19 |
| Display4[2] | Output | PIN_E21 |
| Display4[1] | Output | PIN_E22 |
| Display4[0] | Output | PIN_F21 |
| Display5[6] | Output | PIN_F20 |
| Display5[5] | Output | PIN_F19 |
| Display5[4] | Output | PIN_H19 |
| Display5[3] | Output | PIN_J18 |
| Display5[2] | Output | PIN_E19 |
| Display5[1] | Output | PIN_E20 |
| Display5[0] | Output | PIN_F18 |
| Display6[6] | Output | PIN_N20 |
| Display6[5] | Output | PIN_N19 |
| Display6[4] | Output | PIN_M20 |
| Display6[3] | Output | PIN_N18 |
| Display6[2] | Output | PIN_L18 |
| Display6[1] | Output | PIN_K20 |
| Display6[0] | Output | PIN_J20 |

# 4 Resultados

El botón superior en las fotos es $B$, mientras que el de abajo es $rst$. Se usan los últimos 4 switches de la derecha, se lee de izquierda a derecha el número binario ($S_3,\ S_2,\ S_1,\ S_0$).

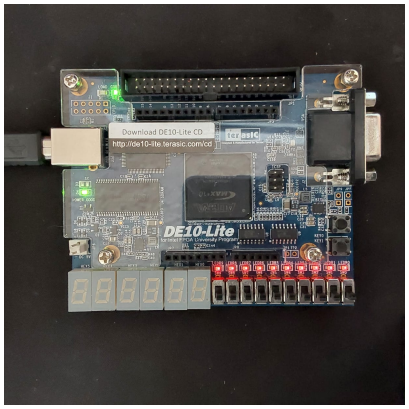

Figura 6: Estado de espera

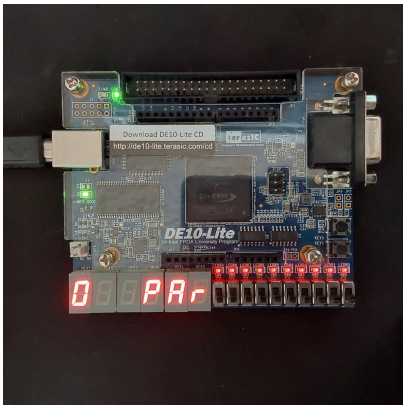


Figura 7: Validación 0

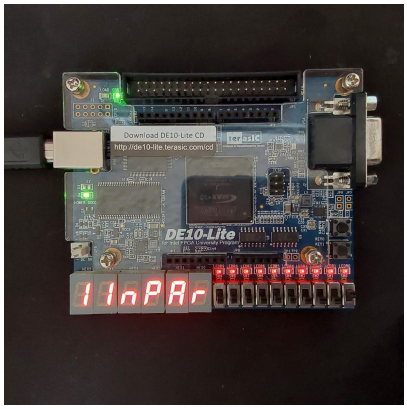


Figura 8: Validación 1

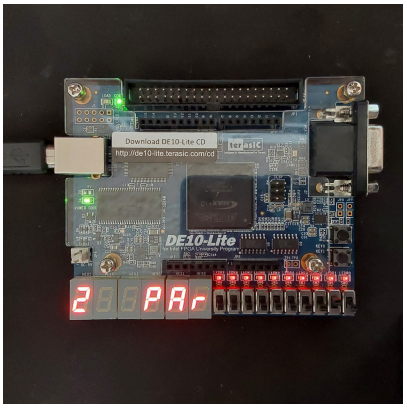


Figura 9: Validación 2

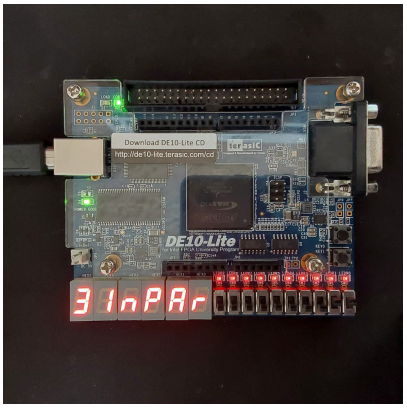


Figura 10: Validación 3

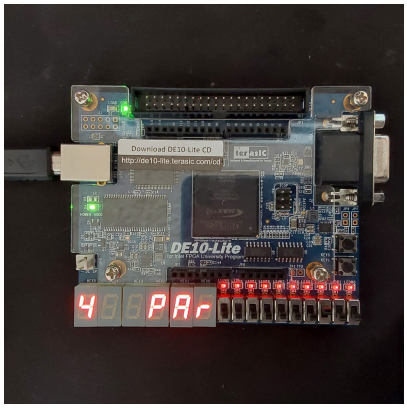


Figura 11: Validación 4

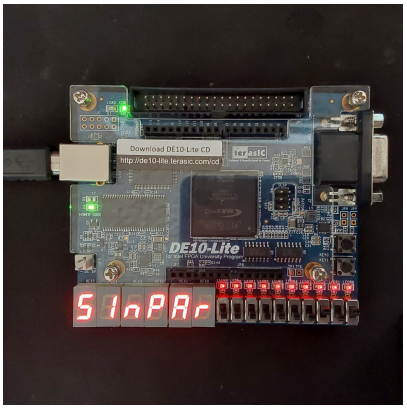


Figura 12: Validación 5

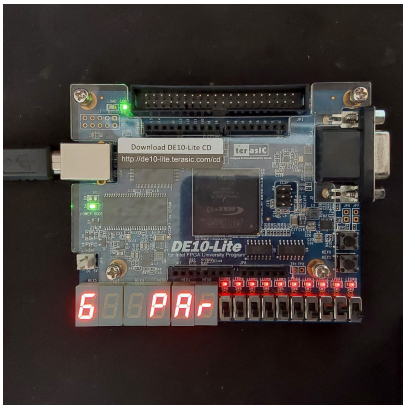


Figura 13: Validación 6

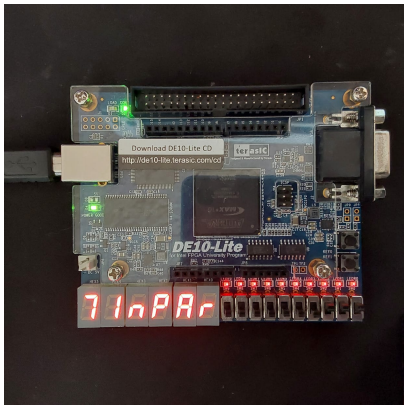


Figura 14: Validación 7

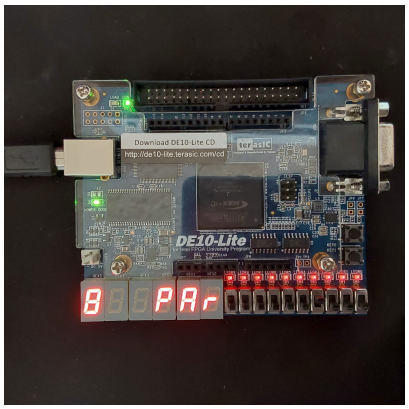


Figura 15: Validación 8

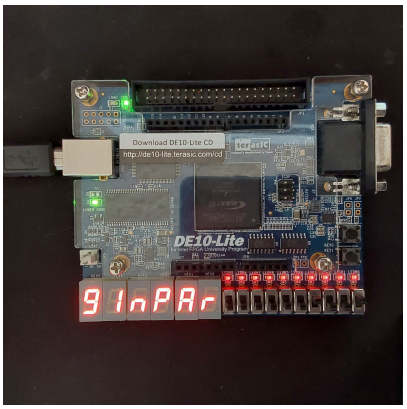


Figura 16: Validación 9

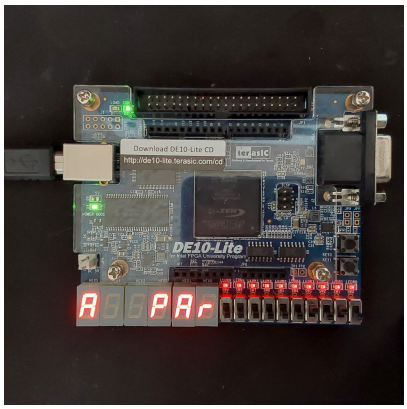


Figura 17: Validación 10 (A)

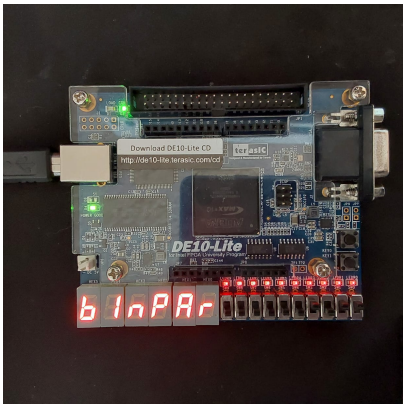


Figura 18: Validación 11 (b)

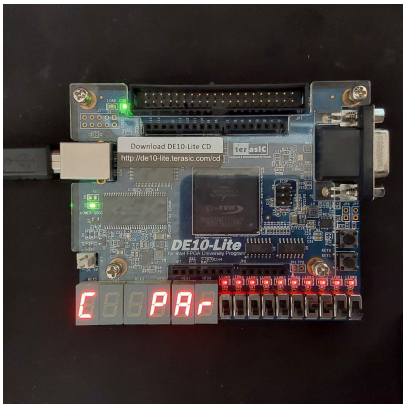


Figura 19: Validación 12 (C)

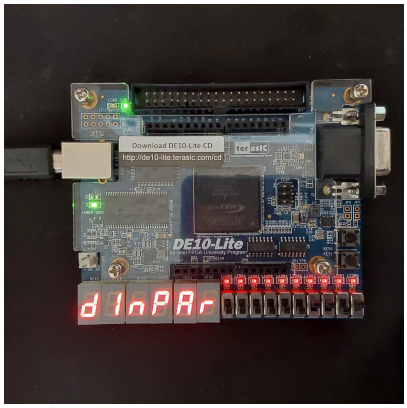


Figura 20: Validación 13 (d)

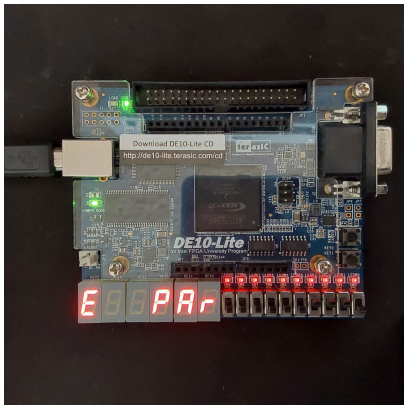


Figura 21: Validación 14 (E)

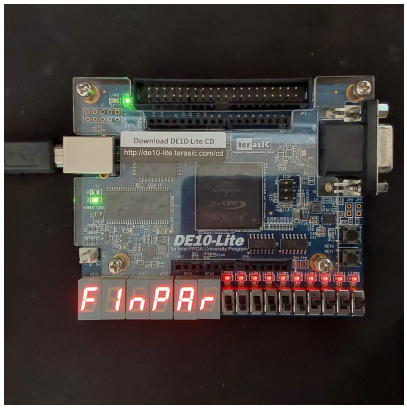


Figura 22: Validación 15 (F)

# 5 Análisis de resultados

Las pruebas realizadas en la tarjeta confirmaron el correcto funcionamiento de la FSM diseñada mediante el método de ecuación algorítmica. De lo observado se destaca que la transición de estados se realiza correctamente al validar pues como se observa en los resultados:

- Cuando $LSB = 0$ pasamos al estado **10** lo que indica que el número es par.
- Cuando $LSB = 1$ pasamos al estado **11** lo que indica que el número es impar.

Esto confirma que las ecuaciones de excitación $D_0$ y $D_1$ fueron implementadas correctamente. Otro punto a rescatar es el funcionamiento en general de la maquina que en los resultados anteriores no puede comprobarse con solo ver las fotos, pero si con el video que se anexa, hacemos énfasis en

- **Estado de Espera:** El sistema inicia y se mantiene en el estado 00 (Figura 6) con los displays apagados o en modo de espera, hasta que se presiona el botón $B$.
- **Visualización:** Se observó que el mensaje de resultado (Par/Impar junto con el número) permanece visible por un lapso perceptible gracias al contador implementado en el `process` del reloj (con un límite de $100,000,000$ ciclos), lo que permite al usuario leer el resultado cómodamente antes de que la FSM regrese automáticamente al estado de espera.
- **Reset Asíncrono:** El botón de *rst* funcionó con prioridad, permitiendo reiniciar la máquina al estado 00 en cualquier momento de la ejecución.

# 6 Conclusión

Se logró cumplir satisfactoriamente con el objetivo de diseñar e implementar un validador de código binario utilizando una SFM. Las pruebas demostraron que el sistema interpreta correctamente el bit menos significativo ($S_0$) para discriminar entre números pares e impares, visualizando el resultado (par/impar) y el valor decimal correspondiente sin errores.

A través del trabajo colaborativo y el análisis de las distintas metodologías de diseño de FSM abordadas en el curso, podemos observar que el método de ecuación algorítmica utilizado en este proyecto resulta ser el más complejo de implementar en comparación con los métodos aritmético y tabular . Esta complejidad radica en la necesidad de plantear y resolver manualmente las "nubes de lógica combinacional" para obtener las ecuaciones booleanas de excitación ($D_0, D_1$) antes de poder describir el circuito. A diferencia del método tabular o el aritmético, el método de ecuación algorítmica exige un mayor rigor en el álgebra booleana previa a la codificación.

Sin embargo, a pesar de su dificultad, este método permitió un control detallado sobre la lógica interna del sistema. Finalmente, la integración del código VHDL consolidó los conocimientos sobre lógica secuencial y máquinas de estados.

# 7 Referencias

[1] García Unzueta, E. E. (2025). *Unidad 4: Diseño e implementación de máquinas de estado finito (FMS)* [Diapositivas de clase]. Facultad de Ingeniería, Universidad Nacional Autónoma de México.

[2] García Unzueta, E. E. (2025). *Unidad 2: Diseño digital VLSI con lenguajes de descripción de hardware, con ambientes de desarrollo CAD - EDA* [Diapositivas de clase]. Facultad de Ingeniería, Universidad Nacional Autónoma de México.

# 8 Anexo

## 8.1 Método aritmético

## 8.2 Método tabular PS/NS (estado presente/estado futuro)

| | **PS** | | | | **NS** | |
|---|---|---|---|---|---|---|
| Reset | $Q_1$ | $Q_0$ | $B$ | $S_0$ | $Q_1^+$ | $Q_0^+$ |
| 0 | X | X | X | X | 0 | 0 |
| 1 | 0 | 0 | X | X | 0 | 0 |
| 1 | 0 | 0 | 0 | X | 0 | 1 |
| 1 | 0 | 1 | X | 1 | 1 | 0 |
| 1 | 0 | 1 | X | 0 | 1 | 0 |
| 1 | 1 | 0 | X | X | 0 | 0 |

Tabla método tabular