

Trabajo de VHDL:

Ascensor

INTEGRANTES DEL GRUPO	
APELLIDOS, NOMBRE	MATRICULA
Marianini Rios, Gregorio	55963
Alonso Geijo, Javier	55714
Vicente Gordón, Luis	56147

GRUPO TEORIA	A404
GRUPO DE TRABAJO	21
PROFESOR	ALBERTO BRUNETE

Índice

1. Descripción de la funcionalidad.....	3
2. Entidades: Atribución de responsabilidades.....	3
2.1 Diagrama de bloques.....	4
3. Detalle de las entidades.....	7
3.1 Entidad top.....	7
3.2 Entidad fsm.....	10
3.3 Entidad Display.....	14
3.4 Entidad SYNCHRNZR.....	16
3.5 Entidad EDGETCTR.....	17
3.6 Entidad esclavo	18
4. Simulaciones.....	19
5. Pruebas en la placa.....	25
6. Conclusión.....	26

1. Descripción de la funcionalidad

El sistema desarrollado controla un ascensor de 4 plantas utilizando VHDL en el entorno Vivado. Este sistema permite gestionar el movimiento del ascensor entre las diferentes plantas, mostrar información visual sobre su estado, y manejar situaciones de emergencia mediante un mecanismo robusto. Siguiendo el documento correspondiente a las ofertas de trabajos, tenemos los siguientes requisitos mínimos:

- El ascensor debe ir al piso indicado por los botones.
- El ascensor debe ignorar los botones mientras se mueve.
- Utilizar LEDs y el display para visualizar la información.

Siguiendo estos requisitos mínimos hemos desarrollado las siguientes especificaciones principales:

- **Selección de piso:** El usuario selecciona el piso mediante botones dedicados.
- **Indicadores LED:** Los LEDs muestran el estado del ascensor, incluyendo la planta actual y alertas.
- **Modo de emergencia:** Un interruptor especial activa un modo de emergencia que prioriza la seguridad del sistema y los LEDs de la placa empiezan a parpadear.
- **Sincronización de entradas:** Se implementó un sincronizador para evitar rebotes en las señales de entrada.
- **Display de 7 segmentos:** Proporciona una representación clara del piso actual, el piso objetivo al que se quiere ir, si el ascensor está en movimiento o parado, incluyendo el sentido del movimiento, y si las puertas están abiertas o cerradas .

Durante el desarrollo del trabajo hemos ido implementando mejoras. Estas son algunas de las principales mejoras:

1. Uso de un esclavo para hacer el contador para los pisos y las puertas.
2. Un estado de emergencia, en el cual el ascensor se bloquea al pulsarlo y al salir del estado sigue en el estado anterior a la emergencia.
3. Uso de sincronizadores y detectores de flancos para un funcionamiento mejor del programa.

El sistema se programó para ejecutarse en la NEXYS 4 DDR. y se probó mediante simulaciones exhaustivas.

2. Entidades: Atribución de responsabilidades

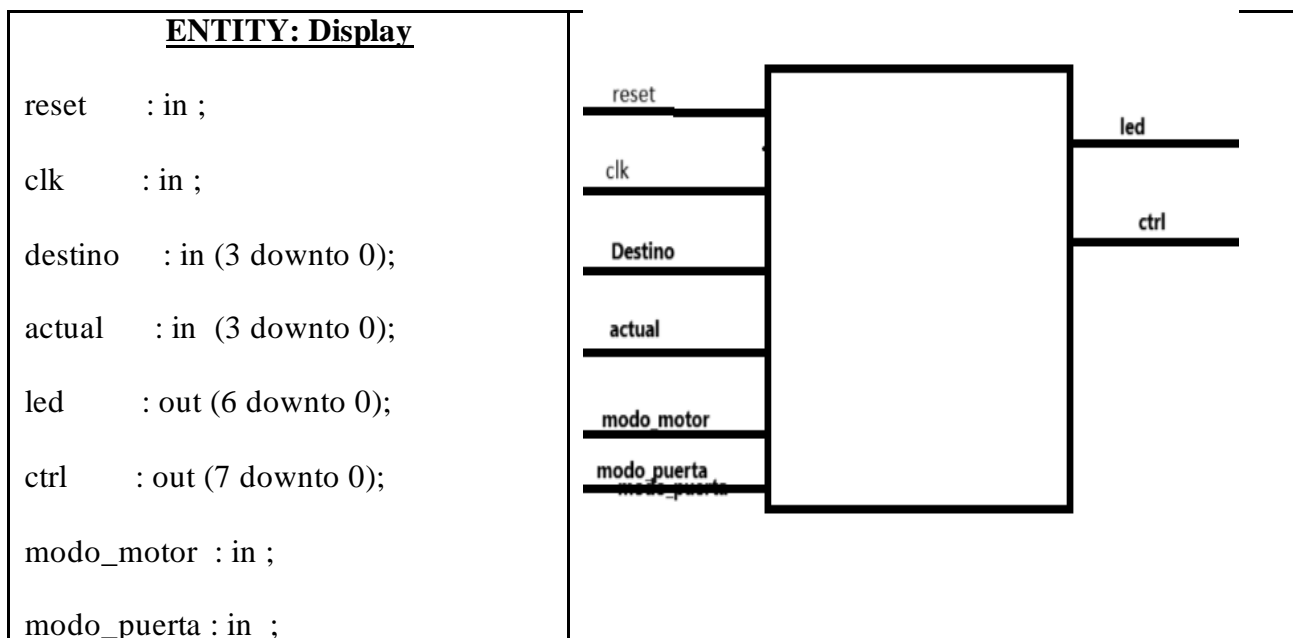
Las entidades utilizadas para desarrollar el trabajo son las indicadas a continuación.

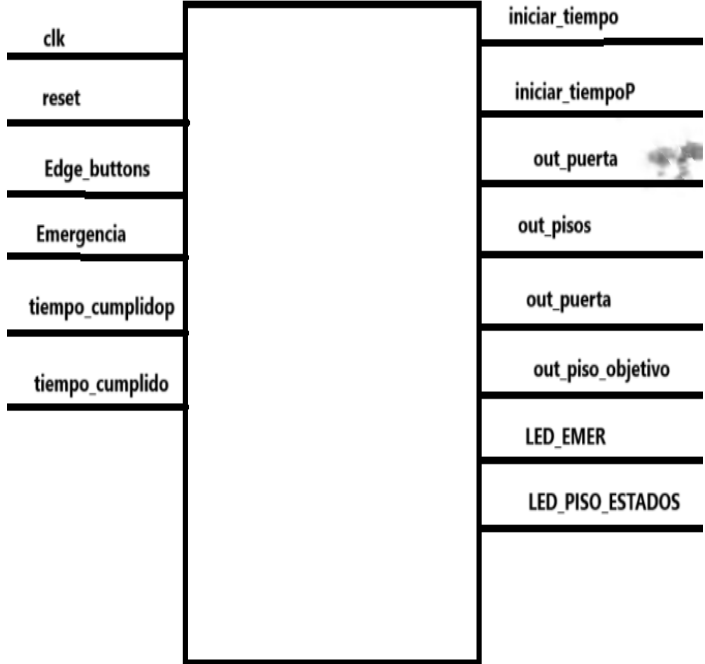


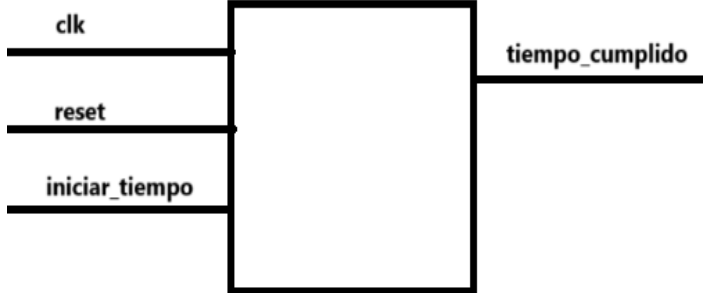
1. **Entidad top:** Interfaz principal que conecta las entradas y salidas del usuario con las entidades internas.
2. **Entidad fsm:** Controla los estados del ascensor y gestiona el modo de emergencia.
3. **Entidad Display:** Decodifica y muestra información en un display de 7 segmentos.
4. **Entidad SYNCHRNZR:** Sincroniza las señales de entrada para evitar rebotes.
5. **Entidad EDGETCTR:** Detecta flancos en las señales de entrada.
6. **Entidad esclavo:** Gestiona tareas auxiliares, como temporizadores y control de estados secundarios.

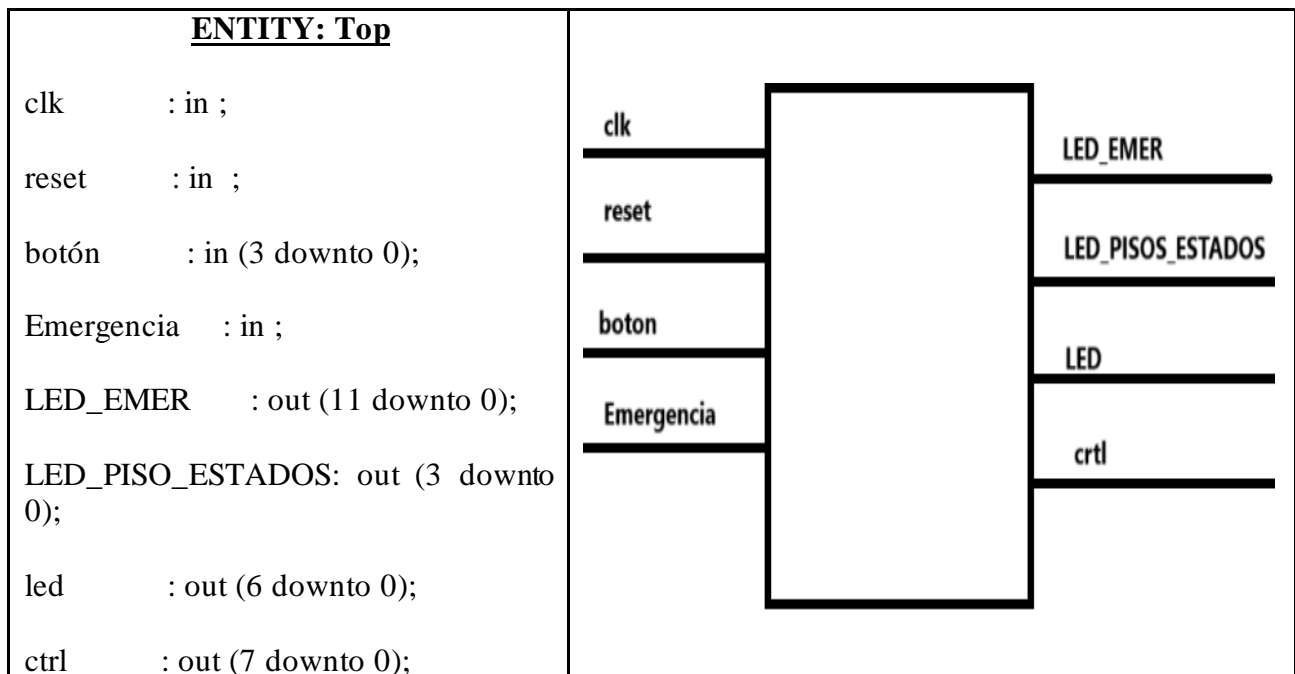
La funcionalidad del programa se puede dividir en la fsm que es la encargada de los cambios de estado, además otras acciones como el encendido y apagado de los LEDs de la emergencia, la entidad esclavo que es la encargada del contador de tiempo para pasar de piso a piso y de abrir y cerrar las puertas (lo que tarda el ascensor en ir de un piso a otro y lo que tarda el ascensor en abrir y cerrar las puertas). La entidad display que es la encargada de mostrar en el display de la placa las diferentes fases por las que pasa el ascensor (piso actual, piso objetivo, estado de las puertas y el movimiento). Además, estas entidades anteriores están auxiliadas por la entidad EDGETCTR y SYNCHRNZR que se encargan de sincronizar las señales y detectar los flancos.

2.1 Diagrama de bloques

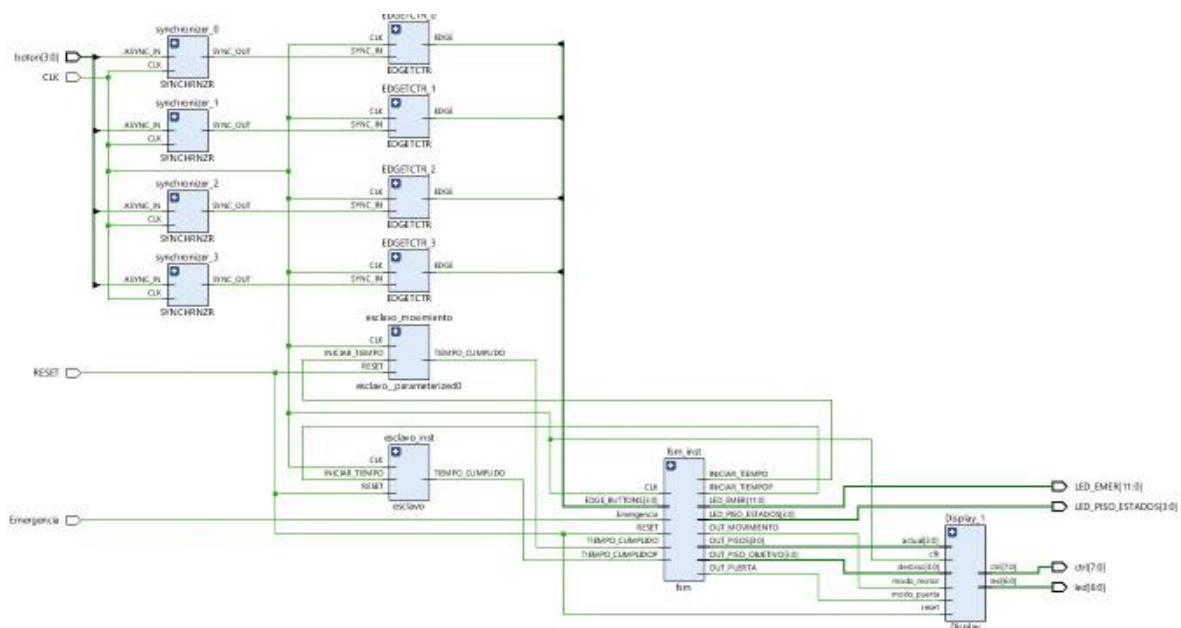
El sistema se organiza según el siguiente diagrama de bloques:



<p align="center"><u>ENTITY: fsm</u></p> <p>clk : in ; reset : in ; edge_buttons : in (3 downto 0);</p> <p>Emergencia : in ; tiempo_cumplido : in ; tiempo_cumplidoP : in ;</p> <p>iniciar_tiempo : out ; iniciar_tiempoP : out ; out_puerta : out ;</p> <p>out_pisos : out (3 downto 0); out_movimiento : out ; out_piso_objetivo: out (3 downto 0);</p> <p>LED_EMER : out (11 downto 0); LED_PISO_ESTADOS : out (3 downto 0);</p>	
<p align="center"><u>ENTITY: SYNCHRNZR</u></p> <p>clk : in ; async_in : in ; sync_out : out ;</p>	
<p align="center"><u>ENTITY: EDGETCTR</u></p> <p>clk : in ; sync_in : in ; edge : out ;</p>	
<p align="center"><u>ENTITY: Esclavo</u></p> <p>generic (N_CICLOS_5SEG : natural);</p> <p>clk : in ; reset : in ; iniciar_tiempo : in ; tiempo_cumplido : out ;</p>	



Cada bloque representa una entidad en el sistema y está interconectado con las demás mediante señales internas. La jerarquía asegura un diseño modular y escalable. Se puede ver como la entidad top no es la que más componentes tiene en su interfaz, esto se debe a que hemos utilizado señales intermedias para los diferentes eventos o valores. A continuación, está el diagrama de bloques entero de proyecto.



3. Detalle de las entidades

3.1 Entidad top

Función: La entidad principal (top) se encarga de todas las entidades que hemos nombrado anteriormente y conecta todas las entradas y salidas externas. Es el punto de integración del sistema. La entidad TOP actúa como la interfaz física entre la placa y el usuario, ya que contiene todas las entradas que dependen directamente del usuario.

Interfaz:

entity top is

port (

CLK : in std_logic;

RESET : in std_logic;

botón : in std_logic_vector (3 downto 0);

Emergencia : in std_logic;

LED_EMER : out std_logic_vector (11 downto 0);

LED_PISO_ESTADOS: out std_logic_vector (3 downto 0);

led : out std_logic_vector(6 downto 0);

ctrl : out std_logic_vector(7 downto 0)

);

end top;

Señales internas:

- sync_buttons: Sincroniza las entradas de los botones.
- iniciar_tiempo, tiempo_cumplido: Gestionan el temporizador.
- estado_emergencia: Controla el modo de emergencia.

Implementación: En la arquitectura, top instancia las entidades necesarias y conecta las señales entre ellas.

Fragmentos de código

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    port (
        CLK           : in  std_logic;
        RESET         : in  std_logic;
        boton         : in  std_logic_vector(3 downto 0);
        Emergencia     : in  std_logic;
        LED_EMER      : out std_logic_vector(11 downto 0) ;
        LED_PISO_ESTADOS : out std_logic_vector(3 downto 0) ;
        led : OUT STD_LOGIC_VECTOR (6 downto 0);
        ctrl : out std_logic_vector (7 downto 0)
    );
end top;

architecture Behavioral of top is

    -- Señales internas para conectar la FSM con el esclavo
    signal iniciar_tiempo : std_logic;
    signal iniciar_tiempo_p : std_logic;
    signal tiempo_cumplido : std_logic;
    signal tiempo_cumplidop : std_logic;
    signal tiempo_emergencia : std_logic;
    signal iniciar_emergencia : std_logic;
    -- Señales para almacenar los valores sincronizados
    signal sync_buttons      : std_logic_vector(3 downto 0); -- Salidas sincronizadas de los botones
    signal edge_detected     : std_logic_vector(3 downto 0); -- Salidas de flancos
    signal piso_objetivo_signal : std_logic_vector(3 downto 0);
    signal actual_signal : std_logic_vector (3 downto 0);

    signal motor_signal: std_logic; --El modo_motor es si subimos, bajamos o nos paramos
    signal puerta_signal : std_logic;
    -- Señales para sincronizar y detectar el flanco

    component fsm
        Port (
            CLK           : in  STD_LOGIC;
            RESET         : in  STD_LOGIC;
            EDGE_BUTTONS  : in  STD_LOGIC_VECTOR(3 downto 0);
            Emergencia     : in  STD_LOGIC;
            TIEMPO_CUMPLIDO : in  STD_LOGIC;
            TIEMPO_CUMPLIDOP : in  STD_LOGIC;

            INICIAR_TIEMPO_P : out STD_LOGIC;
            INICIAR_TIEMPO : out STD_LOGIC;

            OUT_PUERTA      : out STD_LOGIC;
            OUT_PISOS      : out STD_LOGIC_VECTOR(3 downto 0);
            OUT_MOVIMIENTO : out STD_LOGIC;
            OUT_PISO_OBJETIVO : out STD_LOGIC_VECTOR(3 downto 0); -- Nueva salida
            LED_EMER      : out STD_LOGIC_VECTOR(11 downto 0);
            LED_PISO_ESTADOS : out std_logic_vector(3 downto 0)
        );
    end component;

    component esclavo
        generic (

```



```

    N_CICLOS_5SEG : natural -- Ajusta este valor según la frecuencia de reloj y el tiempo deseado
  );
  port (
    CLK          : in std_logic;
    RESET        : in std_logic;
    INICIAR_TIEMPO : in std_logic;
    TIEMPO_CUMPLIDO : out std_logic
  );
end component;

component SYNCHRONIZER
  port(
    CLK : in std_logic;
    ASYNC_IN : in std_logic;
    SYNC_OUT : out std_logic
  );
end component;

component EDGEDETECTOR
  port(
    CLK : in std_logic;
    SYNC_IN : in std_logic;
    EDGE : out std_logic
  );
end component;

component Display
  Port {
    reset : in std_logic;
    clk : in std_logic;
    destino : IN STD_LOGIC_VECTOR (3 downto 0);
    actual : in std_logic_vector (3 downto 0);
    led : OUT STD_LOGIC_VECTOR (6 downto 0);
    ctrl : out std_logic_vector (7 downto 0);
    modo_motor : in std_logic; --El modo motor es si subimos, bajamos = nos paramos
    modo_puerta : in std_logic
  };
end component;

begin
  fsm_inst : entity work.fsm
    port map (
      CLK => CLK,
      RESET => RESET,
      EDGE_BUTTONS => edge_detected,
      Emergencia => Emergencia,
      TIEMPO_CUMPLIDO => tiempo_cumplido,
      TIEMPO_CUMPLIDOS => tiempo_cumplidos,

      INICIAR_TIEMPOS => iniciar_tiempo_p,
      INICIAR_TIEMPO => iniciar_tiempo,

      OUT_PUERTA => puerta_signal,
      OUT_PISOS => actual_signal,
      OUT_MOVIMIENTO => motor_signal,
      OUT_PISO_OBJETIVO => piso_objetivo_signal,
      LED_EMER => LED_EMER,
      LED_PISO_ESTADOS => LED_PISO_ESTADOS
    );

  esclavo_inst : entity work.esclavo
    generic map (
      N_CICLOS_5SEG => 200_000_000 -- Ajuste según la frecuencia de reloj 2 segundos
    )
    port map (
      CLK => CLK,
      RESET => RESET,
      INICIAR_TIEMPO => iniciar_tiempo_p,
      TIEMPO_CUMPLIDO => tiempo_cumplido
    );

  esclavo_movimiento : entity work.esclavo
    generic map (
      N_CICLOS_5SEG => 500_000_000 -- Ajuste según la frecuencia de reloj 5 segundos a 100 MHz
    )
    port map (
      CLK => CLK,
      RESET => RESET,
      INICIAR_TIEMPO => iniciar_tiempo,
      TIEMPO_CUMPLIDO => tiempo_cumplido
    );

  -- Instancia del Sincronizador para el botón 0
  synchroniser_0 : SYNCHRONIZER
    port map (
      CLK => CLK,
      ASYNC_IN => boton(0), -- Entrada asincrónica (Botón 0)
      SYNC_OUT => sync_buttons(0) -- Salida sincronizada
    );

  -- Instancia del Sincronizador para el botón 1
  synchroniser_1 : SYNCHRONIZER
    port map (
      CLK => CLK,
      ASYNC_IN => boton(1), -- Entrada asincrónica (Botón 1)

```

```

        SYNC_OUT => sync_buttons(1)  -- Salida sincronizada
    );
    -- Instancia del Sincronizador para el botón 2
    synchronizer_2 : SYNCHRNZR
    port map (
        CLK      => CLK,
        ASYNC_IN => boton(2),  -- Entrada asincrona (botón 2)
        SYNC_OUT => sync_buttons(2)  -- Salida sincronizada
    );
    -- Instancia del Sincronizador para el botón 3
    synchronizer_3 : SYNCHRNZR
    port map (
        CLK      => CLK,
        ASYNC_IN => boton(3),  -- Entrada asincrona (botón 3)
        SYNC_OUT => sync_buttons(3)  -- Salida sincronizada
    );

    EDGETCTR_0 : EDGETCTR
    port map (
        CLK      => CLK,
        SYNC_IN  => sync_buttons(0),
        EDGE     => edge_detected(0)
    );

    -- Instancia del Edge Detector para el botón 1
    EDGETCTR_1 : EDGETCTR
    port map (
        CLK      => CLK,
        SYNC_IN  => sync_buttons(1),
        EDGE     => edge_detected(1)
    );

    -- Instancia del Edge Detector para el botón 2
    EDGETCTR_2 : EDGETCTR
    port map (
        CLK      => CLK,
        SYNC_IN  => sync_buttons(2),
        EDGE     => edge_detected(2)
    );

    -- Instancia del Edge Detector para el botón 3
    EDGETCTR_3 : EDGETCTR
    port map (
        CLK      => CLK,
        SYNC_IN  => sync_buttons(3),
        EDGE     => edge_detected(3)
    );

    -- Instancia del Edge Detector para la señal sincronizada de Emergencia
    Display_1 : Display
    port map (
        reset => RESET,
        clk  => CLK,
        destino => piso_objetivo_signal,
        actual => actual_signal,
        led   => led,
        ctrl  => ctrl,
        modo_motor => motor_signal,
        modo_puerta => puerta_signal
    );
end Behavioral;

```

3.2 Entidad fsm

Función: Gestionar la lógica del ascensor a través de una máquina de estados finitos. Esta entidad controla las transiciones entre los estados del ascensor, que son:

1. ABRIR: Indica que las puertas del ascensor están abiertas.
2. CERRAR: Indica que las puertas del ascensor se están cerrando.
3. MARCHA: El ascensor está en movimiento hacia el piso seleccionado.
4. EMERGENCIA: El ascensor se detiene inmediatamente debido a una señal de emergencia.
5. REPOSO: El ascensor está inactivo, esperando una acción del usuario.

Entradas:

- CLK: Señal de reloj.
- RESET: Reinicio del sistema.
- botón: Vector de selección del piso.
- Emergencia: Señal de emergencia.

Salidas:

- piso_actual: Indica el piso actual del ascensor.
- estado_emergencia: Señal de activación para el modo emergencia.

Fragmento de código

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity fsm is
6     Port (
7         CLK                : in  STD_LOGIC;
8         RESET              : in  STD_LOGIC;  -- Activo en '0'
9         EDGE_BUTTONS      : in  STD_LOGIC_VECTOR(3 downto 0);
10        Emergencia         : in  STD_LOGIC;
11        TIEMPO_CUMPLIDO    : in  STD_LOGIC;  -- Para movimiento
12        TIEMPO_CUMPLIDOP  : in  STD_LOGIC;  -- Para puertas
13        INICIAR_TIEMPO    : out STD_LOGIC;  -- Activa el timer de movimiento
14        INICIAR_TIEMPOP   : out STD_LOGIC;  -- Activa el timer de puerta
15        OUT_FUERTA        : out STD_LOGIC;
16        OUT_PISOS         : out STD_LOGIC_VECTOR(3 downto 0);
17        OUT_MOVIMIENTO    : out STD_LOGIC;
18        OUT_PISO_OBJETIVO : out STD_LOGIC_VECTOR(3 downto 0);
19        LED_EMER          : out STD_LOGIC_VECTOR(11 downto 0);
20        LED_PISO_ESTADOS  : out STD_LOGIC_VECTOR(3 downto 0)
21    );
22 end fsm;
23
24 architecture Behavioral of fsm is
25
26     type estado_type is (reposo, cerrar, marcha, abrir, emer);
27     signal estado, estado_siguiete : estado_type := reposo;
28
29     signal piso_memoria : std_logic_vector(3 downto 0) := "0001";
30     signal piso_real    : std_logic_vector(3 downto 0) := "0001";
31     signal led_estados  : std_logic_vector(3 downto 0) := (others => '0');
32     signal iniciar_tiempo_siguiete : std_logic := '0';
33     signal iniciar_tiempo_puertas  : std_logic := '0';
34     signal puerta_abierta : std_logic := '1';
35     signal tiempo_cumplido_prev : std_logic := '0';
36     signal tiempo_cumplido_p    : std_logic := '0';  -- Para puertas
37     signal aux : std_logic := '0';
38
39     constant C_MAX_CONTADOR : natural := 50_000_000;  -- 0.5s en un reloj
40     signal CONTADOR : natural range 0 to C_MAX_CONTADOR := 0;
41     signal SENAL_emer : std_logic := '0';
42
43 begin
44
45     process (CLK, RESET)
46     begin
47         if RESET = '0' then
48             estado <= reposo;
49             piso_memoria <= "0001";
50             piso_real <= "0001";
51             puerta_abierta <= '1';
52             INICIAR_TIEMPO <= '0';
53             INICIAR_TIEMPOP <= '0';
54             tiempo_cumplido_prev <= '0';
55             tiempo_cumplido_p <= '0';
56             aux <= '0';
57
58             CONTADOR <= 0;
59             SENAL_emer <= '0';
60
61             elsif rising_edge(CLK) then
```

```

62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |

```

```

if estado = emer then
    if CONTADOR < C_MAX_CONTADOR then
        CONTADOR <= CONTADOR + 1;
    else
        CONTADOR <= 0;
        SENAL_emer <= not SENAL_emer;
    end if;
else
    CONTADOR <= 0;
    SENAL_emer <= '0';
end if;

-- Guardamos estado actual => siguiente
estado <= estado_siguiente;

-- Para detectar flancos en TIEMPO_CUMPLIDO
tiempo_cumplido_prev <= TIEMPO_CUMPLIDO;
tiempo_cumplido_p <= TIEMPO_CUMPLIDOP;

-- Actualizamos enable de timers
INICIAR_TIEMPO <= iniciar_tiempo_siguiente;
INICIAR_TIEMPOP <= iniciar_tiempo_puertas;

if estado = reposo then
    if EDGE_BUTTONS(0) = '1' then
        piso_memoria <= "0001";
    elsif EDGE_BUTTONS(1) = '1' then
        piso_memoria <= "0010";
    elsif EDGE_BUTTONS(2) = '1' then
        piso_memoria <= "0100";
    elsif EDGE_BUTTONS(3) = '1' then
        piso_memoria <= "1000";
    end if;
end if;

if (estado = cerrar or estado = abrir) then
    if (TIEMPO_CUMPLIDOP = '1' and tiempo_cumplido_p = '0') then
        puerta_abierta <= not puerta_abierta;
        INICIAR_TIEMPOP <= '0';
    end if;
end if;

if estado = marcha then
    -- Flanco de subida
    if (TIEMPO_CUMPLIDO = '1' and tiempo_cumplido_prev = '0') then
        aux <= '1';
        INICIAR_TIEMPO <= '0';
    -- Flanco de bajada
    elsif (TIEMPO_CUMPLIDO = '0' and tiempo_cumplido_prev = '1') then
        aux <= '0';
    end if;
end if;

if (estado = marcha) and
(TIEMPO_CUMPLIDO = '1' and tiempo_cumplido_prev = '0') and
(iniciar_tiempo_siguiente = '1') then

```

```

121         (iniciar_tiempo_siguiente = '1') then
122
123         if piso_real < piso_memoria then
124             piso_real <= std_logic_vector(unsigned(piso_real) sll 1);
125         elsif piso_real > piso_memoria then
126             piso_real <= std_logic_vector(unsigned(piso_real) srl 1);
127         end if;
128     end if;
129
130 end if;
131 end process;
132
133
134 process (estado, piso_memoria, piso_real, puerta_abierta, Emergencia, aux)
135 begin
136     estado_siguiente <= estado;
137     iniciar_tiempo_puertas <= '0';
138     iniciar_tiempo_siguiente <= '0';
139
140     led_estados <= "0000";
141
142     case estado is
143     when reposo =>
144         led_estados <= "0001";
145
146         if piso_memoria /= piso_real then
147             estado_siguiente <= cerrar;
148             iniciar_tiempo_puertas <= '1';
149         end if;
150
151     when cerrar =>
152         led_estados <= "0010";
153         if puerta_abierta = '1' then
154             iniciar_tiempo_puertas <= '1';
155         else
156             estado_siguiente <= marcha;
157             iniciar_tiempo_puertas <= '0';
158             iniciar_tiempo_siguiente <= '1';
159         end if;
160
161     when marcha =>
162         led_estados <= "0100";
163
164         if Emergencia = '1' then
165             estado_siguiente <= emer;
166             iniciar_tiempo_siguiente <= '0';
167         else
168             if aux = '1' then
169                 iniciar_tiempo_siguiente <= '0';
170             elsif aux = '0' then
171                 if piso_real /= piso_memoria then
172                     iniciar_tiempo_siguiente <= '1';
173                 else
174                     estado_siguiente <= abrir;
175                     iniciar_tiempo_siguiente <= '0';
176                 end if;
177             end if;
178         end if;
179
180     when abrir =>
181         led_estados <= "1000";
182         if puerta_abierta = '0' then
183             iniciar_tiempo_puertas <= '1';
184         else
185             estado_siguiente <= reposo;
186             iniciar_tiempo_puertas <= '0';
187         end if;
188
189     when emer =>
190         led_estados <= "1111";
191         -- Comportamiento de LED Emer controlado por blink_emer en el proc
192         if Emergencia = '0' then
193             estado_siguiente <= marcha;
194             iniciar_tiempo_siguiente <= '1';
195         end if;
196
197     when others =>
198         estado_siguiente <= reposo;
199     end case;
200 end process;
201
202 OUT_MOVIMIENTO <= '1' when (estado = marcha) else '0';
203 OUT_PUERTA <= puerta_abierta;
204 OUT_PISOS <= piso_real;
205 OUT_PISO_OBJETIVO <= piso_memoria;
206 LED_PISO_ESTADOS <= led_estados;
207
208
209 LED_EMER <= (others => SENAL_emer) when (estado = emer) else (others => '0');
210
211 end Behavioral;

```

3.3 Entidad Display

Función: Decodificar el piso actual del ascensor y mostrarlo en un display de 7 segmentos.

El display muestra en dos display el piso objetivo al que se pretende ir mostrando el número de dicho piso, otro dos display indicando el piso real en el que se encuentra el ascensor mostrando el número de dicho piso. Otro dos display indica si las puertas están abiertas o cerradas para que puedan pasar las personas mediante unos dibujos en el display, y los dos últimos dos display indica si está parado o está en movimiento, además indica con flechas si sube hacia arriba o baja hacia abajo indicando el sentido del movimiento.

Entradas:

- piso: Valor del piso actual.
- Piso_objetivo: piso al que se pretende ir.
- Puertas: marca si están abiertas o cerradas.
- Movimiento: dice si se está en movimiento el ascensor.

Salidas:

- segmentos: Señales para controlar el display de 7 segmentos.

Fragmento de código:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 entity Display is
5     Port (
6         reset      : in  std_logic;
7         clk         : in  std_logic;
8         destino     : in  std_logic_vector(3 downto 0);
9         actual      : in  std_logic_vector(3 downto 0);
10        led         : out std_logic_vector(6 downto 0);
11        ctrl        : out std_logic_vector(7 downto 0);
12        modo_motor  : in  std_logic;
13        modo_puerta : in  std_logic
14    );
15 end Display;
16 architecture Behavioral of Display is
17
18     constant DIV_MAX : integer := 20000; -- 100 MHz / 5kHz = 20000
19     signal div_contador : unsigned(15 downto 0) := (others => '0');
20     signal reset_contador : unsigned(2 downto 0) := (others => '0');
21     signal led_reg : std_logic_vector(6 downto 0) := (others => '1');
22     signal ctrl_reg : std_logic_vector(7 downto 0) := (others => '1');
23
24     function decode_pisos(nib : std_logic_vector(3 downto 0))
25     return std_logic_vector is
26     variable seg : std_logic_vector(6 downto 0) := (others => '1');
27     begin
28         case nib is
29             when "0001" => seg := "0000001"; -- '0'
30             when "0010" => seg := "1001111"; -- '1'
31             when "0100" => seg := "0010010"; -- '2'
```

```

32         when "1000" => seg := "0000110"; -- '3'
33         when others => seg := "1111111"; -- Apagado
34     end case;
35     return seg;
36 end function;
37 function decode_PUERTA(b : std_logic) return std_logic_vector is
38     variable seg : std_logic_vector(6 downto 0) := (others => '1');
39 begin
40     if b = '0' then
41         seg := "0110110"; -- '0'
42     else
43         seg := "1001001"; -- '1'
44     end if;
45     return seg;
46 end function;
47 function decode_motor1sub(b : std_logic) return std_logic_vector is
48     variable seg : std_logic_vector(6 downto 0) := (others => '1');
49 begin
50
51     if b = '0' then
52         seg := "1111110"; -- '0'
53     else
54         seg := "0001101"; -- '1'
55     end if;
56     return seg;
57 end function;
58 function decode_motor1baj(b : std_logic) return std_logic_vector is
59     variable seg : std_logic_vector(6 downto 0) := (others => '1');
60 begin
61     if b = '0' then
62         seg := "1111110"; -- '0'
63     else
64         seg := "1000011"; -- '1'
65     end if;
66
67     return seg;
68 end function;
69 function decode_motor2sub(b : std_logic) return std_logic_vector is
70     variable seg : std_logic_vector(6 downto 0) := (others => '1');
71 begin
72     if b = '0' then
73         seg := "1111110"; -- '0'
74     else
75         seg := "0011001"; -- '1'
76     end if;
77     return seg;
78 end function;
79 function decode_motor2baj(b : std_logic) return std_logic_vector is
80     variable seg : std_logic_vector(6 downto 0) := (others => '1');
81 begin
82     if b = '0' then
83         seg := "1111110"; -- '0'
84     else
85         seg := "1100001"; -- '1'
86     end if;
87     return seg;
88 end function;
89 begin
90     process(clk)
91     begin

```

```

91     begin
92         if rising_edge(clk) then
93             if reset = '0' then
94                 div_contador <= (others => '0');
95                 reset_contador <= (others => '0');
96             else
97                 div_contador <= div_contador + 1;
98                 if div_contador = to_unsigned(DIV_MAX, div_contador'length) then
99                     div_contador <= (others => '0');
100                     reset_contador <= reset_contador + 1; -- 000..111
101                 end if;
102             end if;
103         end if;
104     end process;
105     process(reset_contador, destino, actual, modo_puerta, modo_motor)
106     begin
107         ctrl_reg <= "11111111";
108         led_reg <= "11111111";
109
110         case reset_contador is
111             when "000" =>
112                 -- D0 =>
113                 ctrl_reg(0) <= '0';
114                 led_reg <= decode_pisos(destino);
115             when "001" =>
116                 -- D1 =>
117                 ctrl_reg(1) <= '0';
118                 led_reg <= decode_pisos(destino);
119             when "010" =>
120                 -- D2 =>
121                 ctrl_reg(2) <= '0';
122                 led_reg <= decode_FUERTA(modo_puerta);
123
124             when "011" =>
125                 -- D3 =>
126                 ctrl_reg(3) <= '0';
127                 led_reg <= decode_FUERTA(modo_puerta);
128
129             when "100" =>
130                 -- D4 =>
131                 if unsigned(destino) > unsigned(actual) then
132                     ctrl_reg(4) <= '0';
133                     led_reg <= decode_motor2sub(modo_motor);
134                 else
135                     ctrl_reg(4) <= '0';
136                     led_reg <= decode_motor2baj(modo_motor);
137                 end if;
138             when "101" =>
139                 if unsigned(destino) > unsigned(actual) then
140                     -- D5 =>
141                     ctrl_reg(5) <= '0';
142                     led_reg <= decode_motor1sub(modo_motor);
143                 else
144                     -- D5 =>
145                     ctrl_reg(5) <= '0';
146                     led_reg <= decode_motor1baj(modo_motor);
147                 end if;
148             when "110" =>
149                 -- D6 =>
150                 ctrl_reg(6) <= '0';
151                 led_reg <= decode_pisos(actual);
152
153             when "111" =>
154                 -- D7 =>
155                 ctrl_reg(7) <= '0';
156                 led_reg <= decode_pisos(actual);
157             when others =>
158                 ctrl_reg <= "11111111";
159                 led_reg <= "11111111";
160         end case;
161     end process;
162     led <= led_reg;
163     ctrl <= ctrl_reg;
164 end Behavioral;

```

3.4 Entidad SYNCHRNZR

Función: La entidad SYNCHRNZR se encarga de devolver una señal en sincronismo con el reloj del sistema para evitar posibles cambios en la señal de la entrada durante la lectura. Dicha tarea se consigue con dos memorias que guardan los valores intermedios (ya estables),

por lo que la salida siempre se retrasará dos ciclos de reloj. Respecto al código, se ha empleado el sincronizador dado en los guiones de prácticas de la asignatura, no obstante, se ha modificado levemente para que sea capaz de operar con más de una señal simultáneamente. El código de la entidad es el siguiente:

Entradas:

- ASYNC_IN: Señal sin procesar del botón.

Salidas:

- SYNC_OUT: Señal estable y sincronizada.

Lógica implementada:

- Utiliza registros de flip-flops para estabilizar las señales durante varios ciclos de reloj.
- Se asegura de que las señales procesadas sean confiables antes de enviarlas a la FSM.

Fragmento de código:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity SYNCHRNZR is
5 port (
6     CLK : in std_logic;
7     ASYNC_IN : in std_logic;
8     SYNC_OUT : out std_logic
9 );
10 end SYNCHRNZR;
11
12 architecture Behavioral of SYNCHRNZR is
13     signal sreg : std_logic_vector(1 downto 0);
14     begin
15         process (CLK)
16         begin
17             if rising_edge(CLK) then
18                 sync_out <= sreg(1);
19                 sreg <= sreg(0) & async_in;
20             end if;
21         end process;
22     end Behavioral;
23
```

3.5 Entidad EDGETCTR

Función: Como último paso en el tratamiento de la señal de entrada, tenemos la entidad EDGEDTCTR que detecta los flancos de subida. Como en las anteriores entidades, se ha definido una señal matricial para tratar todos los botones simultáneamente:

Entradas:

- SYNC_IN: Señal sincronizada del botón.

Salidas:

- EDGE: Genera un pulso cuando se detecta un cambio en el botón.

Fragmento de código:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity EDGETCTR is
4  port (
5      CLK : in std_logic;
6      SYNC_IN : in std_logic;
7      EDGE : out std_logic
8  );
9  end EDGETCTR;
10
11 architecture Behavioral of EDGETCTR is
12     signal sreg : std_logic_vector(2 downto 0);
13     begin
14         process (CLK)
15         begin
16             if rising_edge(CLK) then
17                 sreg <= sreg(1 downto 0) & SYNC_IN;
18             end if;
19         end process;
20         with sreg select
21             EDGE <= '1' when "100",
22                   '0' when others;
23     end Behavioral;
```

3.6 Entidad esclavo

Función: La entidad esclavo es responsable de manejar las tareas auxiliares del sistema, como:

1. **Temporizadores:** Gestionar los tiempos necesarios para que el ascensor complete transiciones específicas, como abrir o cerrar puertas.
2. **Sincronización de estados secundarios:** Apoyar a la FSM (entidad principal) en la coordinación de señales relacionadas con el estado actual del ascensor.

Esta entidad actúa como un módulo independiente que provee señales de control a otras entidades, asegurando que los procesos dependientes de tiempo se realicen correctamente.

Entradas:

- CLK: Señal de reloj para sincronizar las operaciones.
- RESET: Reinicia los valores de los temporizadores.
- iniciar_tiempo: Señal que activa un temporizador específico.

Salidas:

- tiempo_cumplido: Señal que indica que un temporizador ha terminado su cuenta regresiva.

Lógica implementada: La entidad esclavo utiliza contadores para implementar temporizadores. Dependiendo del estado y las entradas, activa o reinicia los temporizadores, generando señales de salida cuando se cumple el tiempo requerido.

Fragmento de código:

```

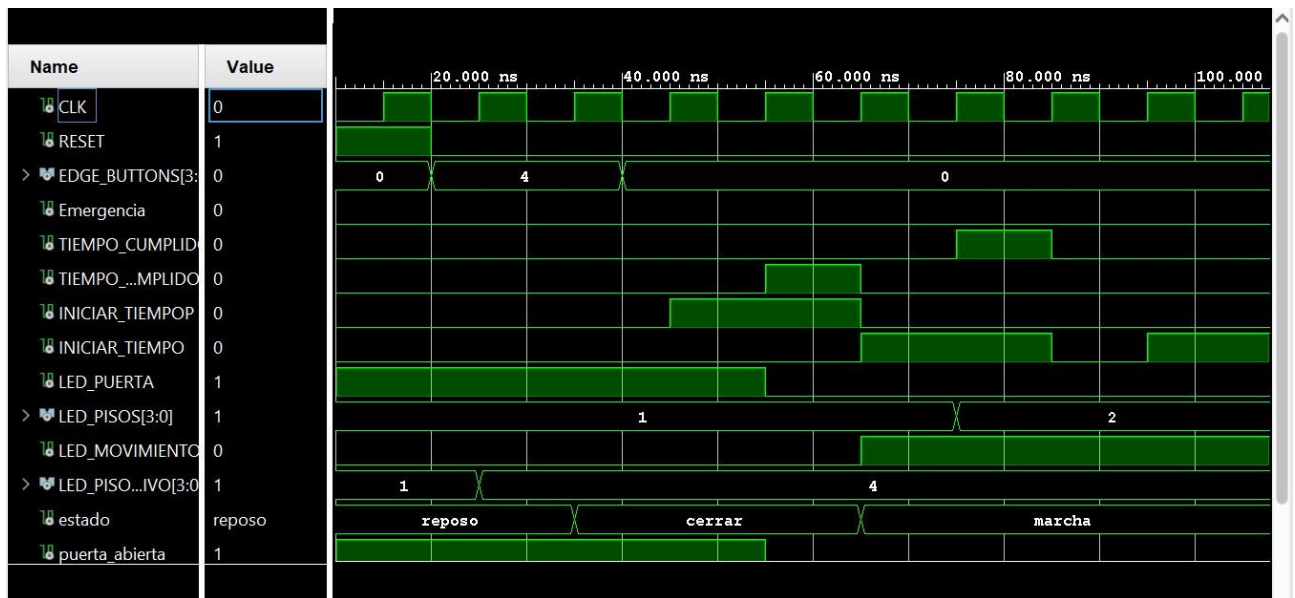
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity esclavo is
6  generic (
7      N_CICLOS_5SEG : natural
8  );
9  port (
10     CLK           : in  std_logic;
11     RESET          : in  std_logic;
12     INICIAR_TIEMPO : in  std_logic;
13     TIEMPO_CUMPLIDO : out std_logic
14 );
15 end esclavo;
16 architecture Behavioral of esclavo is
17     signal contador : unsigned(31 downto 0) := (others => '0');
18     signal activo   : std_logic := '0';
19 begin
20     process(CLK, RESET)
21     begin
22         if RESET = '0' then
23             contador <= (others => '0');
24             activo <= '0';
25         elsif rising_edge(CLK) then
26             if INICIAR_TIEMPO = '1' then
27                 if activo = '0' then
28                     activo <= '1';
29                     contador <= (others => '0');
30                 else
31                     if contador < to_unsigned(N_CICLOS_5SEG - 1, 32) then
32                         contador <= contador + 1;
33                     end if;
34                 end if;
35             else
36                 activo <= '0';
37                 contador <= (others => '0');
38             end if;
39         end if;
40     end process;
41     TIEMPO_CUMPLIDO <= '1' when (activo = '1' and contador = to_unsigned(N_CICLOS_5SEG - 1, 32)) else '0';
42 end Behavioral;
43

```

4. Simulaciones

Las diferentes simulaciones realizadas validaron el funcionamiento general del sistema y sus entidades específicas. Se destacaron los siguientes resultados satisfactorias de las entidades de FSM, DISPLAY y ESCLAVO:

1. **FSM:** Las transiciones entre los estados principales (REPOSO, ABRIR, CERRAR, MARCHA y EMERGENCIA) se realizaron correctamente, verificando que las salidas asociadas respondieran adecuadamente.



El testbench utilizado para esta simulación es el siguiente:

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity fsm_tb is
6  and fsm_tb;
7
8  architecture Behavioral of fsm_tb is
9
10     -- Component under test (CUT)
11     component fsm is
12     Port (
13         CLR          : in  STD_LOGIC;
14         RESET        : in  STD_LOGIC;
15         EDGE_BUTTONS : in  STD_LOGIC_VECTOR(3 downto 0);
16         Emergencia    : in  STD_LOGIC;
17         Sensor_puerta : in  STD_LOGIC;
18         Boton_abrir   : in  STD_LOGIC;
19         TIEMPO_CUMPLIDO : in  STD_LOGIC;
20         TIEMPO_CUMPLIDOP : in  STD_LOGIC;
21         INICIAR_TIEMPO : out STD_LOGIC;
22         INICIAR_TIEMPO : out STD_LOGIC;
23         LED_PUERTA    : out STD_LOGIC;
24         LED_PISOS     : out STD_LOGIC_VECTOR(3 downto 0);
25         LED_MOVIMIENTO : out STD_LOGIC;
26         LED_PISO_OBJETIVO : out STD_LOGIC_VECTOR(3 downto 0)
27     );
28     end component;
29
30     -- Señales
31     signal CLR          : STD_LOGIC := '0';
32     signal RESET        : STD_LOGIC := '0';
33     signal EDGE_BUTTONS : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
34     signal Emergencia    : STD_LOGIC := '0';
35     signal Sensor_puerta : STD_LOGIC := '0';
36     signal Boton_abrir   : STD_LOGIC := '0';
37     signal TIEMPO_CUMPLIDO : STD_LOGIC := '0';
38     signal TIEMPO_CUMPLIDOP : STD_LOGIC := '0';
39     signal INICIAR_TIEMPO : STD_LOGIC;
40     signal INICIAR_TIEMPO : STD_LOGIC;
41     signal LED_PUERTA    : STD_LOGIC;
42     signal LED_PISOS     : STD_LOGIC_VECTOR(3 downto 0);
43     signal LED_MOVIMIENTO : STD_LOGIC;
44     signal LED_PISO_OBJETIVO : STD_LOGIC_VECTOR(3 downto 0);
45
46     -- Constantes
47     constant CLK_PERIOD : time := 10 ns;
48
49 begin
50     -- Instanciar la FSM
51     uut: fsm
52     Port map (
53         CLR          => CLR,
54         RESET        => RESET,
55         EDGE_BUTTONS => EDGE_BUTTONS,
56         Emergencia    => Emergencia,
57         Sensor_puerta => Sensor_puerta,
58         Boton_abrir   => Boton_abrir,
59         TIEMPO_CUMPLIDO => TIEMPO_CUMPLIDO,
60         TIEMPO_CUMPLIDOP => TIEMPO_CUMPLIDOP,
61         INICIAR_TIEMPO => INICIAR_TIEMPO,
62         INICIAR_TIEMPO => INICIAR_TIEMPO,
63         LED_PUERTA    => LED_PUERTA,
64         LED_PISOS     => LED_PISOS,
65         LED_MOVIMIENTO => LED_MOVIMIENTO,
66         LED_PISO_OBJETIVO => LED_PISO_OBJETIVO
67     );
68
69     -- Reloj
70     process
71     begin
72         while true loop
73             CLK <= '0';
74             wait for CLK_PERIOD / 2;
75             CLK <= '1';
76             wait for CLK_PERIOD / 2;
77         end loop;
78     end process;
79
80     -- Proceso de estímulos
81     process
82     begin
83         -- Reset inicial
84         RESET <= '1';
85         wait for 20 ns;
86         RESET <= '0';
87
88         -- Selección de piso 4
89         EDGE_BUTTONS <= "0100";
90         wait for 20 ns;
91         EDGE_BUTTONS <= "0000";
92
93         -- Proceso de cerrar puertas
94         wait until INICIAR_TIEMPO = '1';
95         wait for CLK_PERIOD;
96
97     end process;
98
99 end Behavioral;

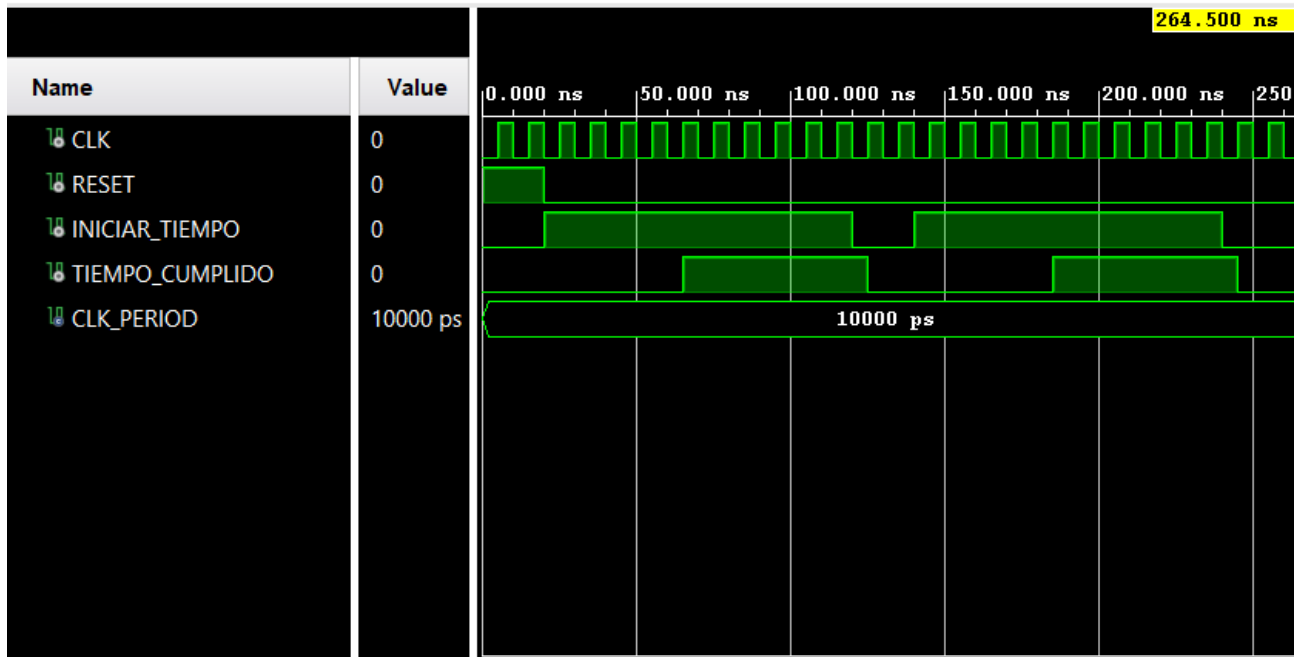
```

```

51     wait for CLK_PERIOD;
52     Sensor_puerta <= '1';
53     EDGE_BUTTONS <= "0100";
54     wait for 20 ns;
55     EDGE_BUTTONS <= "0000";
56     wait for 2 CLK_PERIOD;
57     -- Espera un ciclo de reloj completo
58     TIEMPO_CUMPLIDOP <= '1'; -- Simula tiempo cumplido de las puertas
59     wait for CLK_PERIOD;
100    TIEMPO_CUMPLIDOP <= '0'; -- Baja el tiempo cumplido
101    -- Movimiento del ascensor
102    wait until INICIAR_TIEMPO = '1';
103    wait for CLK_PERIOD; -- Espera un ciclo de reloj
104    TIEMPO_CUMPLIDO <= '1'; -- Activa tiempo cumplido general
105    wait for CLK_PERIOD;
106    TIEMPO_CUMPLIDO <= '0';
107    -- Repetir el tiempo cumplido en marcha
108    wait for 3*CLK_PERIOD;
109    Emergencia <= '1';
110    wait for 3*CLK_PERIOD;
111    Emergencia <= '0';
112    wait until INICIAR_TIEMPO = '1';
113    wait for CLK_PERIOD;
114    TIEMPO_CUMPLIDO <= '1';
115    wait for CLK_PERIOD;
116    TIEMPO_CUMPLIDO <= '0';
117    -- Apertura de puertas
118    wait until INICIAR_TIEMPOF = '1';
119    wait for CLK_PERIOD;
120    TIEMPO_CUMPLIDOP <= '1';
121    wait for CLK_PERIOD;
122    TIEMPO_CUMPLIDOP <= '0'; -- Selección de piso 4
123
124    EDGE_BUTTONS <= "0010";
125    wait for 20 ns;
126    EDGE_BUTTONS <= "0000";
127
128    -- Proceso de cerrar puertas
129    wait until INICIAR_TIEMPOF = '1';
130    wait for CLK_PERIOD; -- Espera un ciclo de reloj completo
131    TIEMPO_CUMPLIDOP <= '1'; -- Simula tiempo cumplido de las puertas
132    wait for CLK_PERIOD;
133    TIEMPO_CUMPLIDOP <= '0'; -- Baja el tiempo cumplido
134    -- Movimiento del ascensor
135    wait until INICIAR_TIEMPO = '1';
136    wait for CLK_PERIOD; -- Espera un ciclo de reloj
137    TIEMPO_CUMPLIDO <= '1'; -- Activa tiempo cumplido general
138    wait for CLK_PERIOD;
139    TIEMPO_CUMPLIDO <= '0';
140
141    wait until INICIAR_TIEMPOF = '1';
142    wait for CLK_PERIOD;
143    TIEMPO_CUMPLIDOP <= '1';
144    wait for CLK_PERIOD;
145    TIEMPO_CUMPLIDOP <= '0'; -- Selección de piso 4
146    -- Fin de la simulación
147    wait for 50 ns;
148    wait;
149 end process;
150 end Behavioral;
151

```

2. **Entidad esclavo:** Los temporizadores funcionaron de manera precisa, generando las señales necesarias para controlar el tiempo de apertura y cierre de puertas.



El testbench utilizado para esta simulación es el siguiente:

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity esclavo_tb is
6  end esclavo_tb;
7
8  architecture Behavioral of esclavo_tb is
9
10     signal CLK          : std_logic := '0';
11     signal RESET        : std_logic := '0';
12     signal INICIAR_TIEMPO : std_logic := '0';
13     signal TIEMPO_CUMPLIDO : std_logic;
14     constant CLK_PERIOD : time := 10 ns;
15
16 begin
17     -- Instancia del esclavo con valor reducido para prueba
18     uut: entity work.esclavo
19         generic map (
20             N_CICLOS_SSEG => 5 -- Ajuste para probar rápidamente
21         )
22         port map (
23             CLK          => CLK,
24             RESET        => RESET,
25             INICIAR TIEMPO => INICIAR TIEMPO,

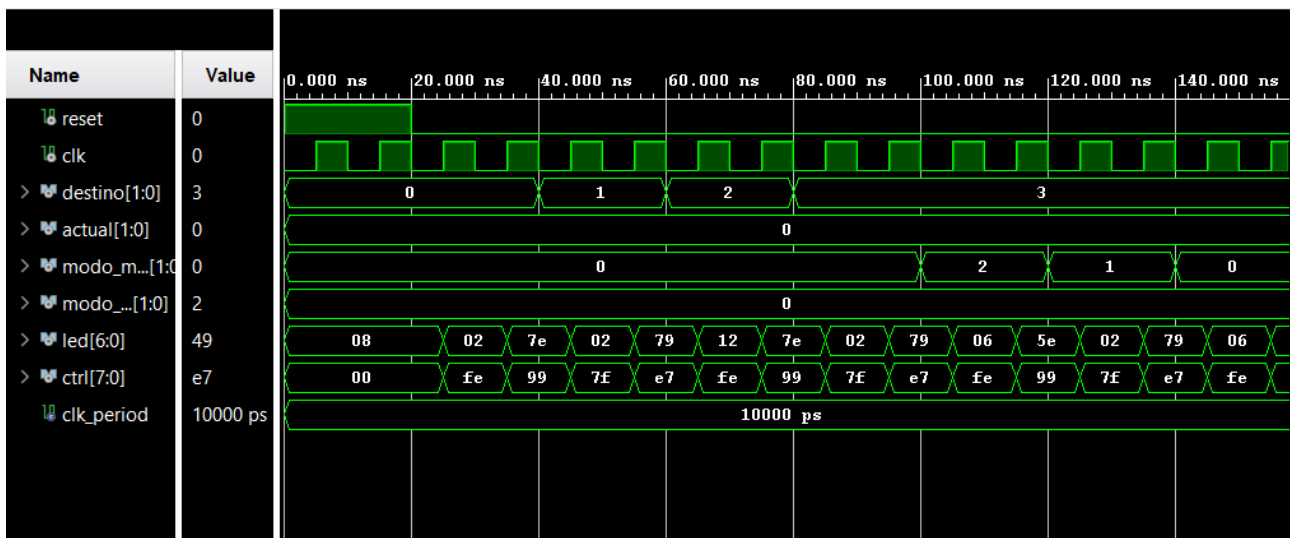
```

```

25      TIEMPO_CUMPLIDO => TIEMPO_CUMPLIDO
26  );
27  -- Generador de reloj
28  process
29  begin
30      while true loop
31          CLK <= '0';
32          wait for CLK_PERIOD/2;
33          CLK <= '1';
34          wait for CLK_PERIOD/2;
35      end loop;
36  end process;
37  -- Estímulos
38  process
39  begin
40      -- Aplicar reset
41      RESET <= '1';
42      wait for 20 ns;
43      RESET <= '0';
44      -- Iniciar tiempo
45      INICIAR_TIEMPO <= '1';
46      -- Esperar suficiente tiempo para que se cumplan los 5 ciclos
47      wait for 100 ns;
48      INICIAR_TIEMPO <= '0';
49      wait for 20 ns;
50
51      -- Iniciar de nuevo
52      INICIAR_TIEMPO <= '1';
53      wait for 100 ns;
54      INICIAR_TIEMPO <= '0';
55      wait;
56  end process;
57 end Behavioral;
58
59
60

```

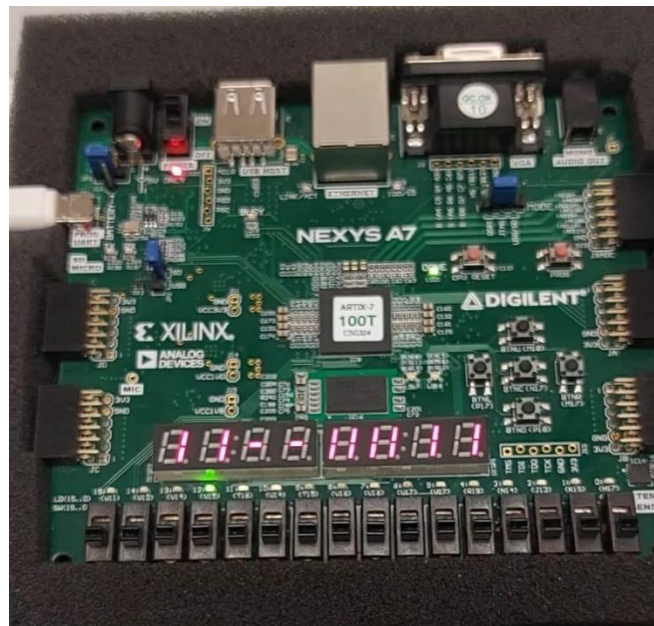
3. **Display:** El piso actual se mostró correctamente en el display de 7 segmentos, y se validaron mensajes de emergencia.



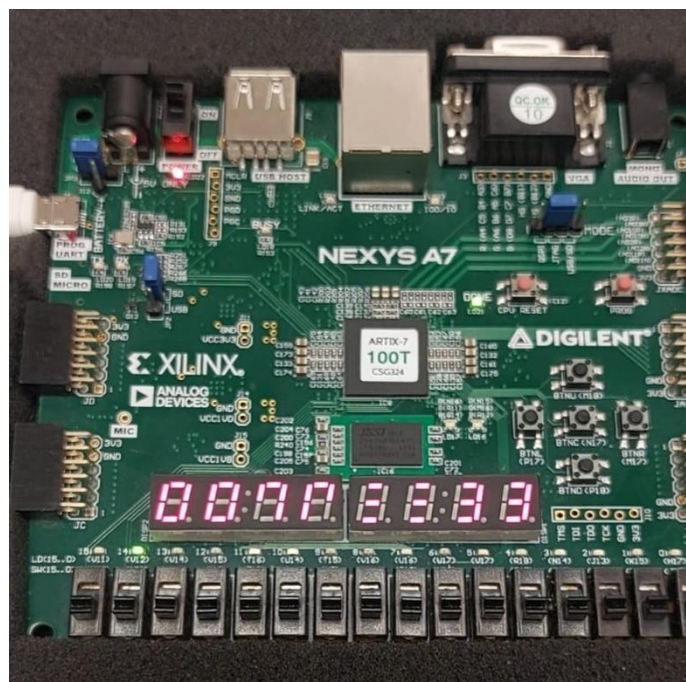
En general, los diferentes entidades funciona de manera estable y cumplió con los requerimientos establecidos en las simulaciones realizadas.

5.Pruebas en la placa

A continuación, se puede ver diferentes fotos del display de la placa con el programa subido en las cuales se puede ver lo comentado anteriormente, tanto los símbolos del display como el funcionamiento de todo el programa.



En esta foto se puede ver el piso actual que en esta foto es el piso uno, el estado del ascensor en este caso está parado (una línea en medio del display), el estado de las puertas en este caso abierto (las dos líneas de los laterales) y por último el piso objetivo que en este caso vuelve a ser el piso número uno. Este caso es que el ascensor ha llegado al piso objeto y se ha parado y las puertas están abiertas.



En esta foto se puede ver el piso actual que en esta foto es el piso cero, el estado del ascensor en este caso está subiendo (una flecha hacia arriba), el estado de las puertas en este caso cerrado (todas las líneas centrales) y por último el piso objetivo que en este caso es el piso número tres. Este caso el ascensor esta subiendo de la planta cero a la planta tres.

6.Conclusión

El sistema de control del ascensor no solo cumple con los requisitos mínimos establecidos en el documento de ofertas de trabajo, sino que también incorpora mejoras adicionales desarrolladas a partir de nuestras propias propuestas. Gracias a un enfoque de diseño modular y la implementación de entidades específicas, se alcanzaron los siguientes objetivos:

1. **Fiabilidad del sistema:** Las pruebas y simulaciones realizadas demostraron un comportamiento sólido y predecible, incluso bajo escenarios operativos complejos o de alta demanda.
2. **Seguridad:** El modo de emergencia garantiza un mecanismo eficaz para detener el ascensor de forma segura ante situaciones críticas, priorizando la protección de los usuarios.
3. **Eficiencia:** La utilización de temporizadores y una adecuada sincronización de señales asegura transiciones rápidas y fluidas entre los distintos estados operativos del sistema.
4. **Interfaz intuitiva:** Los indicadores visuales, como LEDs y displays, ofrecen información clara y precisa sobre el estado del ascensor, mejorando la experiencia del usuario.

Además, la arquitectura modular del sistema facilita la integración de nuevas funcionalidades, como la expansión a más niveles o la incorporación de algoritmos avanzados para gestionar la priorización de las llamadas. Esto no solo amplía las capacidades del sistema, sino que también asegura su adaptabilidad a las necesidades futuras.