



**UNIVERSIDAD
POLITÉCNICA DE
MADRID**



POLITÉCNICA

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL**

**PROYECTO FINAL
ELEVADOR**

MATERIA:
SISTEMAS ELECTRONICOS DIGITALES

INTEGRANTES:

- ESCALERA TELLEZ ERICK.
- GÓMEZ ALVARADO RICARDO DANIEL.
- QUIRÓZ HERNÁNDEZ IAN EDUARDO
- VAZQUEZ LOPEZ LUIS ALBERTO

PROFESOR:
DRA. REBECA HERRERO MARTIN

GRUPO:

404

FECHA DE ENTREGA:
MIERCOLES 22 DE ENERO DE 2025

Contenido

INTRODUCCIÓN	3
OBEJTIVOS	3
Tecnologías y Herramientas	3
METODOLOGÍA.....	4
DESARROLLO.....	4
Máquina de Estados	5
Diagrama Esquemático	6
CÓDIGOS.	7
Código de divisor de frecuencia para la Nexys.....	7
Testbench	9
Diagrama - TestBench.....	10
Diagrama RTL.....	10
Código Interfaz con Displays	11
Testbench.	14
Diagrama - TestBench.....	15
Diagrama RTL.....	15
Código Máquina de Estados.....	16
Testbench	22
Diagrama- Testbench	24
Diagrama RTL.....	25
Código Final VHDL	26
Testbench	28
ARCHIVO CONSTRAINTS.....	29
Diagrama RTL	30
CÓDIGO MICROCONTROLADOR	31
Explicación del código.....	34
CONCLUSION	39
BIBLIOGRAFIA	40

INTODUCCIÓN.

El presente proyecto explora el diseño y la implementación de un sistema basado en hardware programable y microcontroladores, empleando la tarjeta Nexys4 DDR con FPGA Artix-7 y el microcontrolador STM32F411 Discovery. La finalidad principal es desarrollar una solución que combine las capacidades de procesamiento paralelo de un FPGA con la versatilidad y facilidad de programación de un microcontrolador STM32.

OBEJTIVOS.

1. **Integración de tecnologías:** Utilizar la tarjeta Nexys4 DDR para tareas de procesamiento intensivo mediante programación en VHDL en el entorno Vivado, y el STM32F411 para control y comunicación utilizando STM32Cube IDE y HAL.
2. **Exploración de interfaces:** Desarrollar la interacción entre ambos sistemas a través de interfaces como UART y SPI , garantizando una comunicación eficiente y sincronizada.
3. **Aplicación práctica:** Implementar un sistema funcional que aproveche las fortalezas de ambos dispositivos, orientado a resolver problemas específicos en el ámbito de la electrónica integrada.

Tecnologías y Herramientas

- **FPGA Nexys4 DDR:** Basado en el FPGA Artix-7 de Xilinx, con características avanzadas como memoria DDR2 integrada, múltiples periféricos y capacidades de alta velocidad
- **STM32F411 Discovery:** Un microcontrolador ARM Cortex-M4 diseñado para aplicaciones de control y procesamiento eficiente, con soporte extensivo de bibliotecas HAL para facilitar el desarrollo
- **Entornos de desarrollo:**
 - **Vivado:** Para la síntesis y simulación de diseños VHDL en el FPGA.
 - **STM32Cube IDE:** Para la configuración y programación del STM32 utilizando la capa HAL.

METODOLOGÍA

El desarrollo se llevó a cabo mediante un enfoque iterativo, comenzando con la configuración básica de hardware, seguida de la implementación de módulos específicos y culminando en pruebas integrales del sistema. Se adoptaron prácticas de documentación exhaustiva y depuración sistemática para garantizar un diseño robusto y reproducible.

Este informe detalla el diseño, implementación, desafíos enfrentados y resultados obtenidos, proporcionando una referencia valiosa para futuros desarrollos en proyectos similares.

DESARROLLO.

El desarrollo del proyecto consiste en elaborar un modelo de elevador funcional de 3 plantas. Antes de continuar con la explicación de los componentes del elevador, debemos mencionar que, para este proyecto, debido a nuestras limitantes en cuanto a equipo y material (ya que somos de intercambio), muchas de las partes se realizaran de manera simulada.

Continuando con la explicación de las características o partes que conforman nuestro elevador, tenemos:

- Nuestro elevador servirá para poder comunicar 3 plantas (piso 1 , 2 y 3) es por ello que se requieren botones para poder hacer la elección del piso.
- Este proyecto, para aprovechar los sistemas que incluyen el microcontrolador, se usará un sistema en el cual el elevador entrará en modo de paro por seguridad, esto por medio del giroscopio del mismo. Además de contar con un botón exclusivo para entrar en modo de paro y un botón para reiniciar el elevador.
- También contará con los botones para abrir y cerrar puertas.
- El movimiento del elevador se simulará por medio de un led de la placa Nexys, esto con el fin de poder sustituir un motor. Además de que las puertas se abrirán por medio de un servo motor.
- Hay sensores infrarrojos para poder detectar la llegada del elevador al piso correspondiente además de tener otro sensor para detectar la presencia de una persona en la entrada del mismo.
- Tenemos un potenciómetro para poder simular la cantidad de peso que está experimentando el elevador, esto arroja una señal y hace el paro del elevador.
- Se cuenta además de la disposición del display de la tarjeta Nexys para poder identificar el piso que se solicita.

Máquina de Estados

Conociendo los requerimientos del proyecto, podemos dibujar la máquina de estado para poder entender el funcionamiento que tendrá internamente los programas.

En este diagrama se cuenta con los estados: IDLE, MOVING_UP, MOVING_DOWN, ARRIVED, DOOR_OPEN, DOOR_CLOSE. Como en primer paso, el elevador se encuentra en un modo de espera. Aquí se evalúa si el elevador se encuentra en óptimas condiciones para poder operar; las cuales son: que en el elevador no esté sobrecargado, que se encuentre estable o que no se haya presionado el botón de stop. Estas condiciones se analizan durante todos los estados del proceso. Posteriormente, mediante la selección de un piso, el propio elevador evalúa el piso que se encuentra y así pueda decidir si ir hacia abajo o hacia arriba dependiendo de lo que se haya solicitado. Como consiguiente, pasa al estado de abrir puertas, en el cual se entra en acción el sensor infrarrojo que se encuentra en la puerta para tener un tiempo de espera, al igual que en el estado de puerta cerrada. Y como final, se regresa al estado de espera.

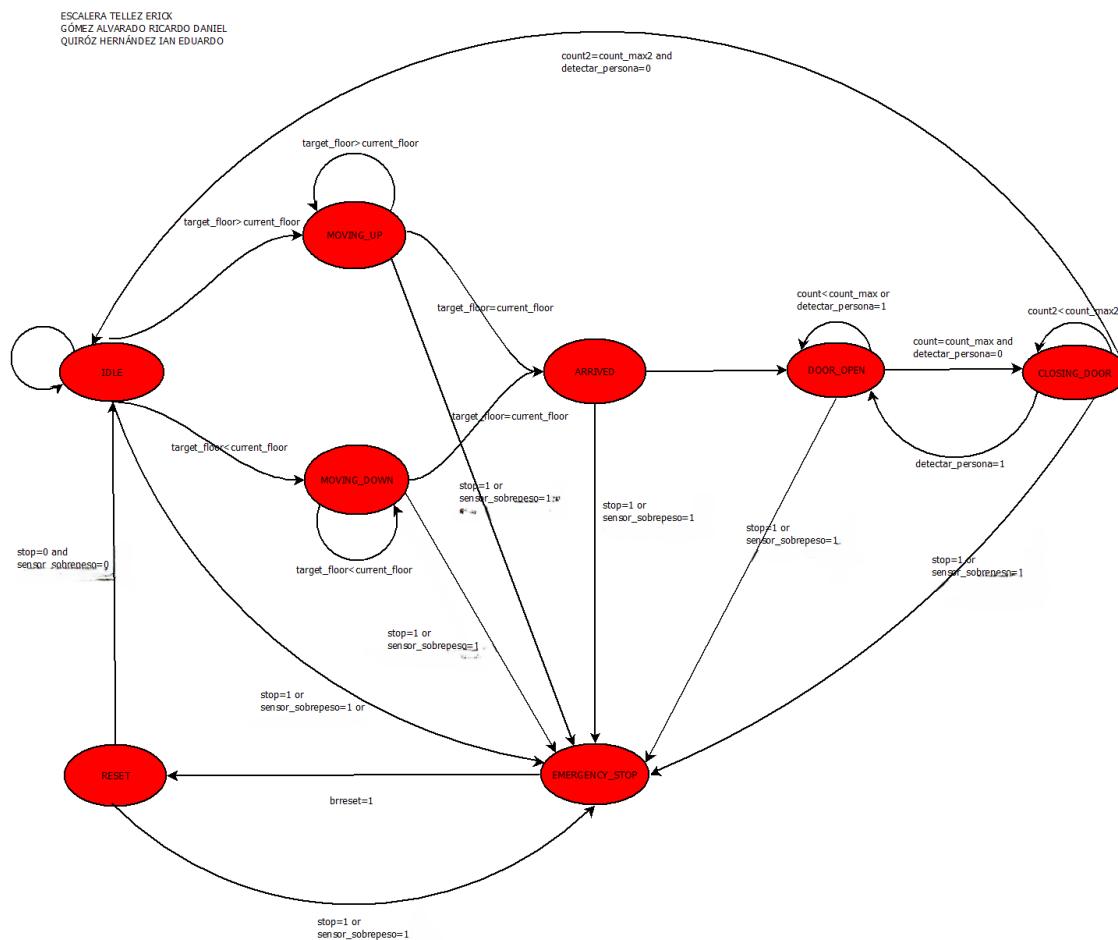


Ilustración 1 Diagrama de la máquina de estados.

Diagrama Esquemático

Para poder continuar con la explicación , necesitamos hacer la conexión de nuestros circuitos junto con la Nexys y el Microcontrolador.

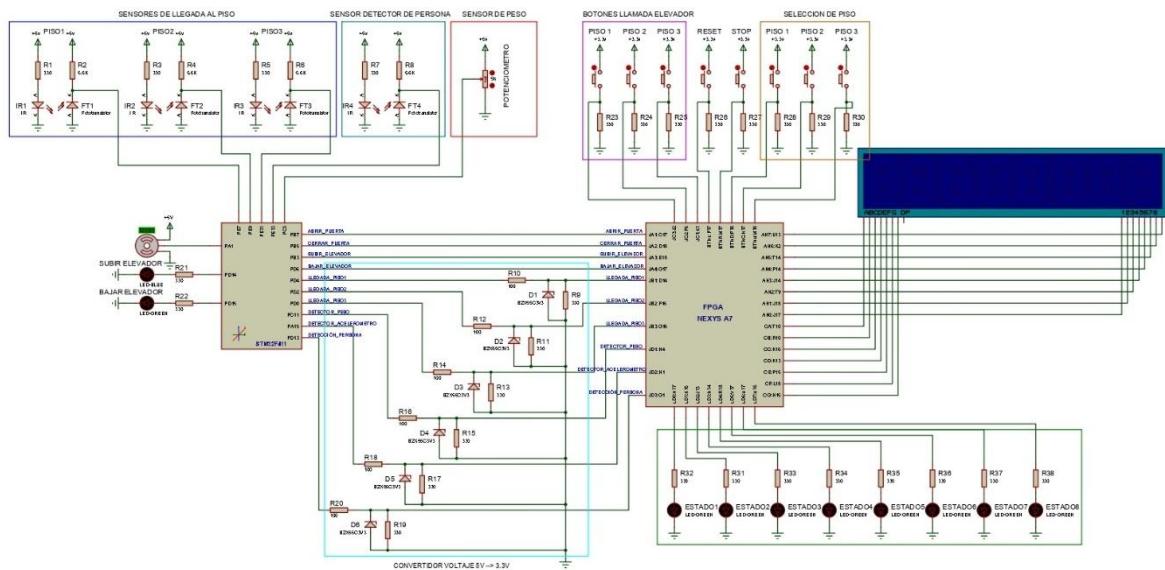


Ilustración 2 Diagrama de conexión.

Aquí podemos ver como es que se hace la conexión entre el microcontrolador y la Nexys, además de las conexiones periféricas que necesitan para poder funcionar. El primer bloque se trata de la conexión que se realiza para los sensores infrarrojos que se encargan de detectar si el elevador llega al piso correspondiente. El siguiente bloque se encarga de el sensor infrarrojo para detectar una persona en la entrada del ascensor y así evitar que se cierren las puertas mientras entra. El tercer bloque se encarga de simular un sensor de peso con un potenciómetro. El cuarto bloque, son los botones de la placa Nexys para poder hacer tanto la llamada del elevador al piso en el que te encuentras, así como al piso al que se desea ir. También se encuentra el botón de reset y de stop.

También es importante mencionar que de necesita realizar un convertidor de voltaje de 5V a 3.3V, mediante el uso de diodos Zener y con un calculo de resistencias debido a que las señales de salida que genera la tarjeta del microcontrolador son a 5V, mientras que las entradas para la Nexys son a 3.3V.

Finalmente, tenemos en el diagrama, la conexión con los periféricos de la Nexys, como es el caso de los botones y del display de 7 segmentos.

CÓDIGOS.

Código de divisor de frecuencia para la Nexys.

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18.01.2025 14:52:37
6  -- Design Name:
7  -- Module Name: DivisorFrecuencia - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity DivisorFrecuencia is
28     Port ( clk_in : in STD_LOGIC; -- Reloj de entrada (100 MHz)           clk_out :
29             out STD_LOGIC; -- Reloj dividido
30             clk_out : out STD_LOGIC; -- Reloj dividido2
31             clk_out2 : out STD_LOGIC; -- Reloj dividido2
32             clk_out3 : out STD_LOGIC -- Reloj dividido3
33 );
34 end DivisorFrecuencia;
35
36 architecture Behavioral of DivisorFrecuencia is
37
38     --Variables del Primer Divisor
39     constant DIVISOR : integer := 1000000; -- Valor de división (para dividir por 1 MHz)
40     signal contador : integer := 0;
41     signal temp_clk : STD_LOGIC := '1';
42
43     --Variables del Segundo Divisor
44     constant DIVISOR2 : integer := 10000000; -- Valor de división (para dividir por 1
45     MHz)
46     signal contador2 : integer := 0;
47     signal temp_clk2 : STD_LOGIC := '1';
48
49     --Variables del Tercer Divisor
50     constant DIVISOR3 : integer := 100000000; -- Valor de división (para dividir por 1
51     MHz)
52     signal contador3 : integer := 0;
53     signal temp_clk3 : STD_LOGIC := '1';
54
55 begin
56
57     --Proceso para Generar el Primer Divisor
58     process(clk_in)
59     begin
60         if rising_edge(clk_in) then
61             contador <= contador + 1;
62             contador2 <= contador2 + 1;
63             contador3 <= contador3 + 1;
64             if contador = (DIVISOR) then
65                 contador <= 0;
66                 temp_clk <= not temp_clk;
67             end if;
68             if contador2 = (DIVISOR2) then

```

Proyecto Elevador

```
67         contador2 <= 0;
68         temp_clk2 <= not temp_clk2;
69     end if;
70     if contador3 = (DIVISOR3) then
71         contador3 <= 0;
72         temp_clk3 <= not temp_clk3;
73     end if;
74 end if;
75 end process;
76
77 --Asignar las señales temporales a las salidas
78 clk_out <= temp_clk;
79 clk_out2 <= temp_clk2;
80 clk_out3 <= temp_clk3;
81
82 end Behavioral;
83
84
```

El VHDL que se presenta crea un divisor de frecuencia. Toma una señal de reloj de 100 MHz y produce tres señales de salida que son más lentas: 100 kHz, 50 kHz y 33.33 kHz. Esto se logra mediante contadores que funcionan de manera independiente. Cada contador se incrementa con cada subida del reloj de entrada. Cuando llega a la mitad del número de división (DIVISOR/2), se reinicia y cambia la señal de salida (clk_out, clk_out2, clk_out3).

Testbench

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18.01.2025 14:55:13
6  -- Design Name:
7  -- Module Name: DivisorFrecuencia_tb - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity DivisorFrecuencia_tb is
26   -- No tiene puertos ya que es un testbench
27 end DivisorFrecuencia_tb;
28
29 architecture Behavioral of DivisorFrecuencia_tb is
30   -- Componentes a instanciar
31   component DivisorFrecuencia
32     Port (
33       clk_in : in STD_LOGIC;
34       clk_out : out STD_LOGIC;
35       clk_out2 : out STD_LOGIC;
36       clk_out3 : out STD_LOGIC
37     );
38   end component;
39
40   -- Señales internas
41   signal clk_in_tb : STD_LOGIC := '0';
42   signal clk_out_tb : STD_LOGIC;
43   signal clk_out_tb2 : STD_LOGIC;
44   signal clk_out_tb3 : STD_LOGIC;
45
46   -- Constante para la simulación del reloj
47   constant CLOCK_PERIOD : time := 0.01 ns; -- 100 MHz
48
49 begin
50   -- Instancia del divisor de frecuencia
51   uut: DivisorFrecuencia
52     Port map (
53       clk_in => clk_in_tb,
54       clk_out => clk_out_tb,
55       clk_out2 => clk_out_tb2,
56       clk_out3 => clk_out_tb3
57     );
58
59   -- Generación del reloj de entrada
60   clk_process : process
61 begin
62   begin
63     while true loop
64       clk_in_tb <= '0';
65       wait for CLOCK_PERIOD / 2;
66       clk_in_tb <= '1';
67       wait for CLOCK_PERIOD / 2;
68     end loop;
69   end process;

```

Proyecto Elevador

```

70
71      -- Proceso de estímulo
72      stimulus_process : process
73      begin
74
75          -- Simular durante un tiempo para observar la salida
76          wait for 10 ms;
77
78          -- Finalizar la simulación
79          assert false report "Fin de la simulación" severity failure;
80      end process;
81
82  end Behavioral;
83

```

Diagrama - TestBench

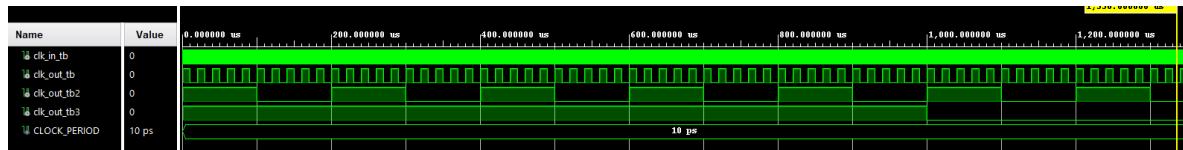


Ilustración 3 Testbench del divisor de frecuencia.

Aquí podemos apreciar como es que los contadores al llegar a su valor máximo alternan sus salidas, cada contador tiene un valor máximo distinto para diferenciar el tiempo que debe transcurrir en cada uno antes de alternar su valor, ya sea en 0 en 1. Los valores propuestos fueron utilizados para ver que se note la diferencia de tiempo de encendido de cada LED en la Nexys.

Diagrama RTL

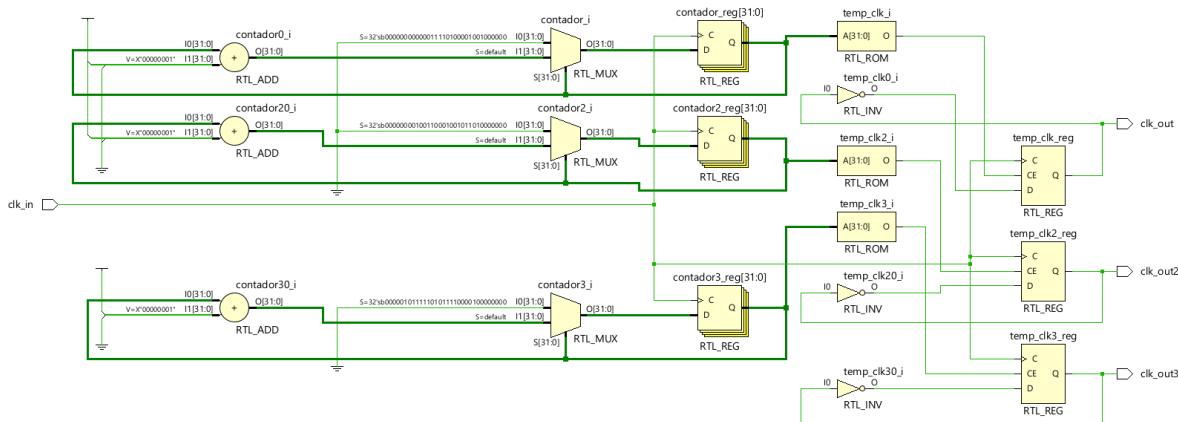


Ilustración 4 Diagrama RTL del divisor de frecuencia.

Código Interfaz con Displays

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18.01.2025 05:46:12
6  -- Design Name:
7  -- Module Name: Display - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_UNSIGNED.ALL;
25
26 -- Entidad que controla la visualización en displays de 7 segmentos
27 entity Display is
28     Port ( codigo_display : in STD_LOGIC_VECTOR (2 downto 0); -- Código de estado del
            elevator
29             seg : out STD_LOGIC_VECTOR (6 downto 0); -- Salida para los segmentos del
            display
30             anodo : out STD_LOGIC_VECTOR (7 downto 0); -- Control de los ánodos de los
            displays
31             clk : in STD_LOGIC; -- Reloj del sistema
32             clk2 : in STD_LOGIC -- Reloj del sistema
33         );
34 end Display;
35
36 architecture Behavioral of Display is
37
38     -- Señales internas
39     signal mensaje : STD_LOGIC_VECTOR(31 downto 0); -- Almacena el mensaje a mostrar
40     signal contador : INTEGER := 0; -- Contador para el divisor
41     signal contador2 : INTEGER := 0; -- Contador para el divisor
42     signal sel: STD_LOGIC_VECTOR(2 downto 0):="000"; -- Control de selección de display
        activo
43     signal caracter: STD_LOGIC_VECTOR (3 downto 0); -- Caracter actual mostrado en un
        display
44 begin
45
46
47
48     -- Control de los ánodos para activar cada display individualmente
49     process(clk2)
50 begin
51         if rising_edge(clk2) then
52             if contador2 = 7 then -- Ajusta el divisor según sea necesario
53                 contador2 <= 0;
54             else
55                 contador2 <= contador2 + 1;
56             end if;
57         end if;
58     end process;
59
60     -- Selección del carácter a mostrar en el display activo
61     process(contador2)
62 begin
63         case contador2 is
64             when 0 => sel <= "000";

```

Proyecto Elevador

```
65      when 1 => sel <= "001";
66      when 2 => sel <= "010";
67      when 3 => sel <= "011";
68      when 4 => sel <= "100";
69      when 5 => sel <= "101";
70      when 6 => sel <= "110";
71      when others => sel <= "111";
72  end case;
73 end process;
74
75 -- Selección del carácter a mostrar en el display activo
76 process(sel, mensaje)
77 begin
78  case sel is
79    when "000" => carácter <= mensaje(3 downto 0);
80    when "001" => carácter <= mensaje(7 downto 4);
81    when "010" => carácter <= mensaje(11 downto 8);
82    when "011" => carácter <= mensaje(15 downto 12);
83    when "100" => carácter <= mensaje(19 downto 16);
84    when "101" => carácter <= mensaje(23 downto 20);
85    when "110" => carácter <= mensaje(27 downto 24);
86    when others => carácter <= mensaje(31 downto 28);
87  end case;
88 end process;
89
90 -- Asignación del mensaje a mostrar según el estado del elevador (64 bits para 8
91 -- caracteres)
92 process(código_display)
93 begin
94  case código_display is
95    when "001" => mensaje <= "0011010001010110111011100000"; -- "PISO ---1"
96    when "010" => mensaje <= "0011010001010110111011100001"; -- "PISO ---2"
97    when "011" => mensaje <= "0011010001010110111011100010"; -- "PISO ---3"
98    when "100" => mensaje <= "010001110001001110111011101110"; -- "IDLE----"
99    when "101" => mensaje <= "100110101010011010101101110110"; -- "ERROR---"
100   when "110" => mensaje <= "110001100011100110101001011100"; -- "--OPENING"
101   when "111" => mensaje <= "1101101100001100101010010111100"; -- "--CLOSING"
102   when others => mensaje <= "11111111111111111111111111111111"; -- Apagar
103   displays
104 end case;
105 end process;
106
107 -- Decodificación del carácter seleccionado a segmentos de display
108 process(carácter)
109 begin
110  case carácter is
111    when "0000" => seg <= "1001111"; -- '1'
112    when "0001" => seg <= "0010010"; -- '2'
113    when "0010" => seg <= "0000110"; -- '3'
114    when "0011" => seg <= "0011000"; -- 'P'
115    when "0100" => seg <= "1111001"; -- 'I'
116    when "0101" => seg <= "0100100"; -- 'S'
117    when "0110" => seg <= "0000001"; -- 'O'
118    when "0111" => seg <= "1000010"; -- 'D'
119    when "1000" => seg <= "1110001"; -- 'L'
120    when "1001" => seg <= "0110000"; -- 'E'
121    when "1010" => seg <= "0001000"; -- 'R'
122    when "1011" => seg <= "0001001"; -- 'N'
123    when "1100" => seg <= "0100000"; -- 'G'
124    when "1101" => seg <= "0110001"; -- 'C'
125    when "1110" => seg <= "1111110"; -- '-'
126    when others => seg <= "1111111"; -- Apagar display
127  end case;
128 end process;
129
130 -- Control de los ánodos para activar cada display individualmente
131 process(sel)
132 begin
133  case sel is
```

Proyecto Elevador

```
132      when "000" => anodo <= "11111110";
133      when "001" => anodo <= "11111101";
134      when "010" => anodo <= "11111011";
135      when "011" => anodo <= "11110111";
136      when "100" => anodo <= "11101111";
137      when "101" => anodo <= "11011111";
138      when "110" => anodo <= "10111111";
139      when others => anodo <= "01111111";
140  end case;
141 end process;
142
143
144
145 end Behavioral;
146
```

Este código se encarga de controlar displays de 7 segmentos, lo que permite mostrar cómo está funcionando un elevador en la Nexys . Usa una señal de 3 bits llamada `codigo_display` para elegir un mensaje que ya está definido y que se guarda en un vector de 32 bits llamado `mensaje`, capaz de representar hasta 8 caracteres. Un contador llamado `contador2`, que está sincronizado con `clk2`, se ocupa de encender los 8 displays de forma cíclica con la señal `anodo`, lo que ayuda a mostrar los caracteres de manera correcta. Cada carácter se convierte en segmentos utilizando una tabla de conversión, permitiendo que se muestren números, letras y símbolos dependiendo del estado del elevador, como `PISO 1`, `IDLE`, `ERROR`, entre otros.

Testbench.

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 18.01.2025 07:19:41
6  -- Design Name:
7  -- Module Name: Display_TB - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_UNSIGNED.ALL;
25
26 entity Display_TB is
27 end Display_TB;
28
29 architecture testbench of Display_TB is
30
31     -- Señales para conectar con el Decodificador
32     signal codigo_display : STD_LOGIC_VECTOR (2 downto 0);
33     signal seg : STD_LOGIC_VECTOR (6 downto 0);
34     signal anodo : STD_LOGIC_VECTOR (7 downto 0);
35     signal clk : STD_LOGIC := '0';
36
37     -- Instancia del Decodificador a probar
38     component top
39         Port ( codigo_display : in STD_LOGIC_VECTOR (2 downto 0);
40                 seg : out STD_LOGIC_VECTOR (6 downto 0);
41                 anodo : out STD_LOGIC_VECTOR (7 downto 0);
42                 clk : in STD_LOGIC
43             );
44     end component;
45
46 begin
47
48     -- Instanciar el DUT (Device Under Test)
49     DUT: top
50         port map (
51             codigo_display => codigo_display,
52             seg => seg,
53             anodo => anodo,
54             clk => clk
55         );
56
57     -- Proceso para generar el reloj
58     process
59     begin
60         while true loop
61             clk <= '0';
62             wait for 0.001 ns;
63             clk <= '1';
64             wait for 0.001 ns;
65         end loop;
66     end process;
67
68
69

```

Proyecto Elevador

```

70      -- Proceso para recorrer todos los casos de codigo_display
71      process
72      begin
73          wait for 1 ns; -- Esperar antes de iniciar la simulación
74
75          -- Prueba de los 7 estados del elevador
76          codigo_display <= "001"; -- "PISO ---1"
77          wait for 5 us;
78
79          codigo_display <= "010"; -- "PISO ---2"
80          wait for 5 us;
81
82          codigo_display <= "011"; -- "PISO ---3"
83          wait for 5 us;
84
85          codigo_display <= "100"; -- "IDLE----"
86          wait for 5 us;
87
88          codigo_display <= "101"; -- "ERROR---"
89          wait for 5 us;
90
91          codigo_display <= "110"; -- "-OPENING"
92          wait for 5 us;
93
94          codigo_display <= "111"; -- "-CLOSING"
95          wait for 5 us;
96
97          wait;
98      end process;
99
100 end testbench;

```

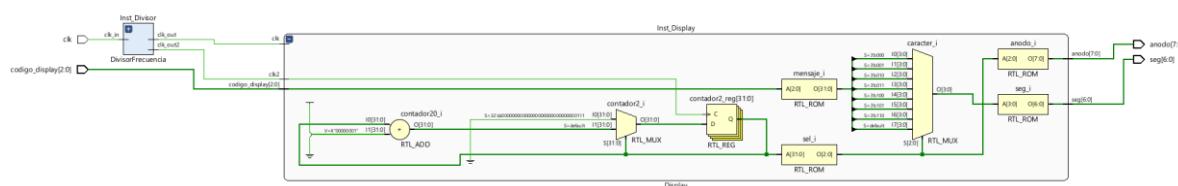
Diagrama - TestBench



Ilustración 5 Testbench displays

El Diagrama de tiempos nos muestra el cambio en el ánodo y en los segmentos, si miramos la tabla de decodificación y lo juntamos con su segmento y la letra que debería entregarnos, notamos la concordancia entre estos 3 anteriores y cuando termina con el ultimo display, vuelve a repetirse, notamos que esta secuencia cambia dependiendo del valor de la entrada, ya sea 0 a 7.

Diagrama RTL.



Código Máquina de Estados.

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 17.01.2025 01:24:44
6  -- Design Name:
7  -- Module Name: ElevatorFSM - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use IEEE.STD_LOGIC_ARITH.ALL;
24 use IEEE.STD_LOGIC_UNSIGNED.ALL;
25
26
27 entity ElevatorFSM is
28     Port (
29         clk : in STD_LOGIC;
30         btresest : in STD_LOGIC;
31
32         -- Entradas de botones y sensores
33         piso1, piso2, piso3 : in STD_LOGIC;
34         llamar_piso1, llamar_piso2, llamar_piso3 : in STD_LOGIC;
35         llegada_piso1, llegada_piso2, llegada_piso3 : in STD_LOGIC;
36         detector_persona, sensor_sobrepeso, sensor_movimiento : in STD_LOGIC;
37         stop : in STD_LOGIC;
38         clk_div : in STD_LOGIC;
39
40         -- Salidas de control
41         abrir_puerta, cerrar_puerta, subir_ascensor, bajar_ascensor : out STD_LOGIC;
42
43         -- Salidas de monitoreo
44         estado_actual : out STD_LOGIC_VECTOR(7 downto 0);
45         codigo_display : out STD_LOGIC_VECTOR(2 downto 0)
46     );
47 end ElevatorFSM;
48
49 architecture Behavioral of ElevatorFSM is
50
51     -- Definición de estados
52     type state_type is (
53         IDLE, MOVING_UP, MOVING_DOWN, ARRIVED,
54         DOOR_OPEN, CLOSING_DOOR, EMERGENCY_STOP, RESET
55     );
56
57     signal state, next_state : state_type;
58
59     -- Señales combinacionales para "próximo piso" y "próximo destino"
60     signal Scurrent_floor, Starget_floor : integer range 1 to 3 := 1;
61
62     -- Registros que guardarán el piso actual y el piso destino
63     signal current_floor, target_floor : integer range 1 to 3 := 1;
64
65     -- Contadores para la puerta
66     signal count : integer range 0 to 100 := 0;
67     signal count_max : integer range 1 to 100 := 30;
68     signal count2 : integer range 0 to 100 := 0;
69     signal count_max2 : integer range 1 to 100 := 50;

```

Proyecto Elevador

```
70      -- Señal de error
71      signal error: STD_LOGIC;
72
73 begin
74 -----
75      -- PROCESO SÍNCRONO PRINCIPAL DE LA FSM (REGISTRO DEL ESTADO)
76
77      process(clk, btreset)
78 begin
79          if btreset = '0' then
80              state <= IDLE;
81          elsif rising_edge(clk) then
82              state <= next_state;
83          end if;
84      end process;
85
86 -----
87      -- PROCESO PARA DETECTAR ERROR (combinacional)
88
89      process(stop, sensor_sobrepeso, sensor_movimiento)
90 begin
91          if stop = '1' or sensor_sobrepeso = '1' or sensor_movimiento = '1' then
92              error <= '1';
93          else
94              error <= '0';
95          end if;
96      end process;
97
98 -----
99      -- PROCESO PARA ACTUALIZAR CONTADORES (DEPENDE DE clk_div, SÍNCRONO A clk_div)
100
101     process(clk_div, btreset, state)
102 begin
103     if btreset = '0' then
104         count <= 0;
105         count2 <= 0;
106     elsif rising_edge(clk_div) then
107         if state = DOOR_OPEN then
108             if count < count_max then
109                 count <= count + 1;
110             else
111                 count <= 0;
112             end if;
113         else
114             count <= 0;
115         end if;
116     else
117         if state = CLOSING_DOOR then
118             if count2 < count_max2 then
119                 count2 <= count2 + 1;
120             else
121                 count2 <= 0; -- Se reinicia cuando llega a count_max2
122             end if;
123         else
124             count2 <= 0; -- Se reinicia cuando no está en CLOSING_DOOR
125         end if;
126     end if;
127 end process;
128
129 -----
130      -- ASIGNACIÓN DE ESTADO_ACTUAL (solo monitoreo)
131
132      process(state)
133 begin
134     case state is
135         when IDLE      => estado_actual <= "00000001";
136         when MOVING_UP => estado_actual <= "00000010";
137         when MOVING_DOWN => estado_actual <= "00000100";
```

Proyecto Elevador

```
139      when ARRIVED      => estado_actual <= "00001000";
140      when DOOR_OPEN     => estado_actual <= "00010000";
141      when CLOSING_DOOR => estado_actual <= "00100000";
142      when EMERGENCY_STOP => estado_actual <= "01000000";
143      when RESET         => estado_actual <= "10000000";
144      when others        => estado_actual <= "00000001";
145  end case;
146 end process;
147
148 -----
149 -- ASIGNACIÓN DE codigo_display (solo monitoreo)
150
151 process(state, current_floor)
152 begin
153   case state is
154     when IDLE =>
155       codigo_display <= "011"; -- ejemplo
156     when MOVING_UP | MOVING_DOWN =>
157       case current_floor is
158         when 1 => codigo_display <= "000";
159         when 2 => codigo_display <= "001";
160         when 3 => codigo_display <= "010";
161         when others => codigo_display <= "111"; -- por seguridad
162       end case;
163     when DOOR_OPEN =>
164       codigo_display <= "101";
165     when CLOSING_DOOR =>
166       codigo_display <= "110";
167     when EMERGENCY_STOP =>
168       codigo_display <= "100";
169     when others =>
170       codigo_display <= "111";
171   end case;
172 end process;
173
174 -----
175 -- PROCESO COMBINACIONAL PARA CALCULAR Scurrent_floor
176 -- (PISO ACTUAL BASADO EN SEÑALES llegada_pisoX)
177
178 process(llegada_piso1, llegada_piso2, llegada_piso3, current_floor)
179 begin
180   if (llegada_piso1 = '1') and (llegada_piso2 = '0') and (llegada_piso3 = '0') then
181     Scurrent_floor <= 1;
182   elsif (llegada_piso1 = '0') and (llegada_piso2 = '1') and (llegada_piso3 = '0')
183   then
184     Scurrent_floor <= 2;
185   elsif (llegada_piso1 = '0') and (llegada_piso2 = '0') and (llegada_piso3 = '1')
186   then
187     Scurrent_floor <= 3;
188   else
189     -- Mantener el piso si no hay una señal válida de llegada
190     Scurrent_floor <= current_floor;
191   end if;
192 end process;
193
194 -----
195 -- PROCESO COMBINACIONAL PARA CALCULAR Starget_floor
196 -- (PISO DESTINO BASADO EN BOTONES pisoX / llamar_pisoX)
197
198 process(piso1, piso2, piso3, llamar_piso1, llamar_piso2, llamar_piso3, target_floor)
199 begin
200   if (piso1 = '1' or llamar_piso1 = '1') and
201     (piso2 = '0' and llamar_piso2 = '0') and
202     (piso3 = '0' and llamar_piso3 = '0') then
203     Starget_floor <= 1;
204
205   elsif (piso2 = '1' or llamar_piso2 = '1') and
206     (piso1 = '0' and llamar_piso1 = '0') and
207     (piso3 = '0' and llamar_piso3 = '0') then
208
209
```

Proyecto Elevador

```
206     Starget_floor <= 2;
207
208     elsif (piso3 = '1' or llamar_piso3 = '1') and
209         (piso2 = '0' and llamar_piso2 = '0') and
210         (piso1 = '0' and llamar_piso1 = '0') then
211         Starget_floor <= 3;
212
213     else
214         Starget_floor <= target_floor;
215     end if;
216 end process;
217
218
219 -- AQUI ESTÁ EL CAMBIO IMPORTANTE:
220 -- PROCESO SÍNCRONO PARA REGISTRAR current_floor
221
222 process(clk, btreset)
223 begin
224     if btreset = '0' then
225         current_floor <= 1; -- Piso inicial
226     elsif rising_edge(clk) then
227         current_floor <= Scurrent_floor;
228     end if;
229 end process;
230
231
232 -- PROCESO SÍNCRONO PARA REGISTRAR target_floor
233
234 process(clk, btreset)
235 begin
236     if btreset = '0' then
237         target_floor <= 1; -- O el piso que consideres inicial
238     elsif rising_edge(clk) then
239         target_floor <= Starget_floor;
240     end if;
241 end process;
242
243
244 -- PROCESO COMBINACIONAL DE LA FSM PARA CALCULAR next_state
245 -- (USAMOS current_floor, target_floor, error, ETC.)
246
247 process(
248     btreset, state, count, count_max, count2, count_max2,
249     pisol, stop, detector_persona, sensor_sobrepeso, sensor_movimiento,
250     current_floor, target_floor, error
251 )
252 begin
253     -- Valores por defecto de las salidas
254     abrir_puerta <= '0';
255     cerrar_puerta <= '0';
256     subir_ascensor <= '0';
257     bajar_ascensor <= '0';
258
259     case state is
260
261         when IDLE =>
262             if error = '1' then
263                 next_state <= EMERGENCY_STOP;
264             else
265                 abrir_puerta <= '0';
266                 cerrar_puerta <= '0';
267                 subir_ascensor <= '0';
268                 bajar_ascensor <= '0';
269
270             if target_floor < current_floor then
271                 next_state <= MOVING_DOWN;
272             elsif target_floor > current_floor then
273                 next_state <= MOVING_UP;
274             else
```

Proyecto Elevador

```
275          -- target_floor = current_floor
276          next_state <= IDLE;
277      end if;
278  end if;

279
280  when MOVING_UP =>
281      if error = '1' then
282          next_state <= EMERGENCY_STOP;
283      else
284          subir_ascensor <= '1';
285          if current_floor = target_floor then
286              next_state <= ARRIVED;
287          else
288              next_state <= MOVING_UP;
289          end if;
290      end if;

291
292  when MOVING_DOWN =>
293      if error = '1' then
294          next_state <= EMERGENCY_STOP;
295      else
296          bajar_ascensor <= '1';
297          if current_floor = target_floor then
298              next_state <= ARRIVED;
299          else
300              next_state <= MOVING_DOWN;
301          end if;
302      end if;

303
304  when ARRIVED =>
305      if error = '1' then
306          next_state <= EMERGENCY_STOP;
307      else
308          subir_ascensor <= '0';
309          bajar_ascensor <= '0';
310          cerrar_puerta <= '1';
311          next_state <= DOOR_OPEN;
312      end if;

313
314  when DOOR_OPEN =>
315      if error = '1' then
316          next_state <= EMERGENCY_STOP;
317      else
318          abrir_puerta <= '1';
319          cerrar_puerta <= '0';

320          if detector_persona = '1' then
321              -- Mantener la puerta abierta si hay persona
322              next_state <= DOOR_OPEN;
323          elsif count = count_max then
324              next_state <= CLOSING_DOOR;
325          else
326              next_state <= DOOR_OPEN;
327          end if;
328      end if;

329
330
331  when CLOSING_DOOR =>
332      if error = '1' then
333          next_state <= EMERGENCY_STOP;
334      else
335          abrir_puerta <= '0';
336          cerrar_puerta <= '1';

337          if detector_persona = '1' then
338              -- Reabrir puerta si detecta persona
339              next_state <= DOOR_OPEN;
340          elsif count2 = count_max2 then
341              next_state <= IDLE;
342          else
343
```

Proyecto Elevador

```
344           next_state <= CLOSING_DOOR;
345       end if;
346   end if;
347
348   when EMERGENCY_STOP =>
349       -- Forzamos ascensor detenido y puerta cerrada
350       subir_ascensor <= '0';
351       bajar_ascensor <= '0';
352       abrir_puerta <= '0';
353       cerrar_puerta <= '1';
354
355       -- Volver al RESET si se sueltan las condiciones de error
356       if btreset = '0' and
357           sensor_sobrepeso = '0' and
358           sensor_movimiento = '0' and
359           stop = '0' then
360           next_state <= RESET;
361       else
362           next_state <= EMERGENCY_STOP;
363       end if;
364
365   when RESET =>
366       cerrar_puerta <= '1';
367       if sensor_sobrepeso = '0' and
368           sensor_movimiento = '0' and
369           stop = '0' then
370           next_state <= IDLE;
371       else
372           next_state <= RESET;
373       end if;
374
375   when others =>
376       next_state <= IDLE;
377
378   end case;
379 end process;
380 end Behavioral;
```

Este código crea una Máquina de Estados Finitos (FSM) destinada a manejar el ascensor en una FPGA. Se definen varios estados, que incluyen IDLE, MOVING_UP, MOVING_DOWN, DOOR_OPEN, CLOSING_DOOR y EMERGENCY_STOP. La máquina cambia de estado según las entradas, como los botones de llamada, sensores de peso y movimiento, y la detección de llegada a los pisos. La FSM está siempre atenta al piso actual y al de destino (current_floor y target_floor), y toma decisiones sobre subir, bajar, abrir o cerrar las puertas según las circunstancias. También se encarga de errores, entrando en un estado de emergencia si detecta sobrepeso, movimiento inusual o si se activa una parada de emergencia, hasta que todo esté seguro de nuevo.

Además, el código regula los tiempos de apertura y cierre de las puertas mediante contadores (count y count2) que aumentan con cada pulso del reloj (clk_div). También ofrece salidas de estado (estado_actual y codigo_display), lo que permite ver el estado del ascensor en un display de 7 segmentos. La FSM asegura que el ascensor funcione de manera ordenada, evitando conflictos en los comandos y manteniendo la seguridad en primer lugar. Si hay un reinicio (btreset), el sistema regresa a IDLE y espera nuevas órdenes, garantizando así un funcionamiento organizado y fiable.

Testbench

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 17.01.2025 05:57:59
6  -- Design Name:
7  -- Module Name: ElevatorFSM_TB - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity ElevatorFSM_TB is
28 end ElevatorFSM_TB;
29
30 architecture Behavioral of ElevatorFSM_TB is
31   signal clk : STD_LOGIC := '0';
32   signal btreset : STD_LOGIC := '0';
33   signal piso1, piso2, piso3 : STD_LOGIC := '0';
34   signal stop : STD_LOGIC := '0';
35   signal llamar_piso1, llamar_piso2, llamar_piso3 : STD_LOGIC := '0';
36   signal llegada_piso1, llegada_piso2, llegada_piso3 : STD_LOGIC := '0';
37   signal detector_persona, sensor_sobrepeso, sensor_movimiento : STD_LOGIC := '0';
38   signal abrir_puerta, cerrar_puerta, subir_ascensor, bajar_ascensor : STD_LOGIC;
39   signal estado_actual : STD_LOGIC_VECTOR(7 downto 0);
40   signal codigo_display : STD_LOGIC_VECTOR(2 downto 0);
41
42
43 component top
44   Port (
45     clk : in STD_LOGIC;
46     btreset : in STD_LOGIC;
47     piso1, piso2, piso3 : in STD_LOGIC;
48     stop : in STD_LOGIC;
49     llamar_piso1, llamar_piso2, llamar_piso3 : in STD_LOGIC;
50     llegada_piso1, llegada_piso2, llegada_piso3 : in STD_LOGIC;
51     detector_persona, sensor_sobrepeso, sensor_movimiento : in STD_LOGIC;
52     abrir_puerta, cerrar_puerta, subir_ascensor, bajar_ascensor : out STD_LOGIC;
53     estado_actual : out STD_LOGIC_VECTOR(7 downto 0);
54     codigo_display : out STD_LOGIC_VECTOR(2 downto 0)
55   );
56 end component;
57
58 begin
59   -- Instancia del elevador
60   uut: top
61     port map (
62       clk => clk,
63       btreset => btreset,
64       piso1 => piso1,
65       piso2 => piso2,
66       piso3 => piso3,
67       stop => stop,
68       llamar_piso1 => llamar_piso1,
69       llamar_piso2 => llamar_piso2,

```

Proyecto Elevador

```
70      llamar_piso3 => llamar_piso3,
71      llegada_piso1 => llegada_piso1,
72      llegada_piso2 => llegada_piso2,
73      llegada_piso3 => llegada_piso3,
74      detector_persona => detector_persona,
75      sensor_sobrepeso => sensor_sobrepeso,
76      sensor_movimiento => sensor_movimiento,
77      abrir_puerta => abrir_puerta,
78      cerrar_puerta => cerrar_puerta,
79      subir_ascensor => subir_ascensor,
80      bajar_ascensor => bajar_ascensor,
81      estado_actual => estado_actual,
82      codigo_display => codigo_display
83  );
84
85  -- Generación del clock
86  process
87  begin
88    while true loop
89      clk <= '0'; wait for 0.001 ns;
90      clk <= '1'; wait for 0.001 ns;
91    end loop;
92  end process;
93
94
95  -- Proceso de prueba
96  process
97  begin
98    -- Reset del sistema
99    btreset <= '1'; wait for 1 us;
100   btreset <= '0';
101
102   -- Llamada desde piso 1 al piso 3
103   llamar_piso3 <= '1';
104   llamar_piso3 <= '0' after 2 us;
105   llegada_piso1 <= '1'; wait for 10 us;
106
107   -- Simula llegada al piso 1
108   llegada_piso1 <= '0';
109
110   -- Ascensor subiendo hasta el piso 3
111   llegada_piso2 <= '1'; wait for 4 us;
112   llegada_piso2 <= '0';
113   llegada_piso3 <= '1';
114
115
116
117   -- Espera que termine el ciclo de apertura y cierre de puerta
118   wait for 500 ns;
119
120   -- Ahora se presiona el botón interno para ir al piso 2
121   llegada_piso3 <= '0' after 1 us;
122   piso2 <= '1'; wait for 2 us;
123   piso2 <= '0'; wait for 6 us;
124
125   -- Ascensor bajando hasta el piso 2
126   llegada_piso2 <= '1'; wait for 4 us;
127   wait for 20 us;
128
129   -- Finaliza la simulación correctamente
130   llegada_piso2 <= '0' after 1 us;
131   pisol <= '1'; wait for 2 us;
132   pisol <= '0'; wait for 6 us;
133
134
135   llegada_piso1 <= '1';
136   wait for 500 ns;
137
138
```

Proyecto Elevador

```
139      llegada_piso1 <= '0' after 1 us;
140      piso3 <= '1'; wait for 2 us;
141      piso3 <= '0'; wait for 6 us;
142
143      llegada_piso2 <= '1'; wait for 8 us;
144      llegada_piso2 <= '0';
145      llegada_piso3 <= '1'; wait for 10 us;
146
147      llegada_piso3 <= '0' after 1 us;
148      piso1 <= '1'; wait for 2 us;
149      piso1 <= '0'; wait for 6 us;
150
151      llegada_piso2 <= '1'; wait for 8 us;
152      stop <= '1'; wait for 10 us;
153      stop <= '0'; wait for 10 us;
154      btreset <= '1'; wait for 10 us;
155      btreset <= '0'; wait for 10 us;
156
157
158
159
160
161      wait;
162  end process;
163 end Behavioral;
```

Diagrama- Testbench

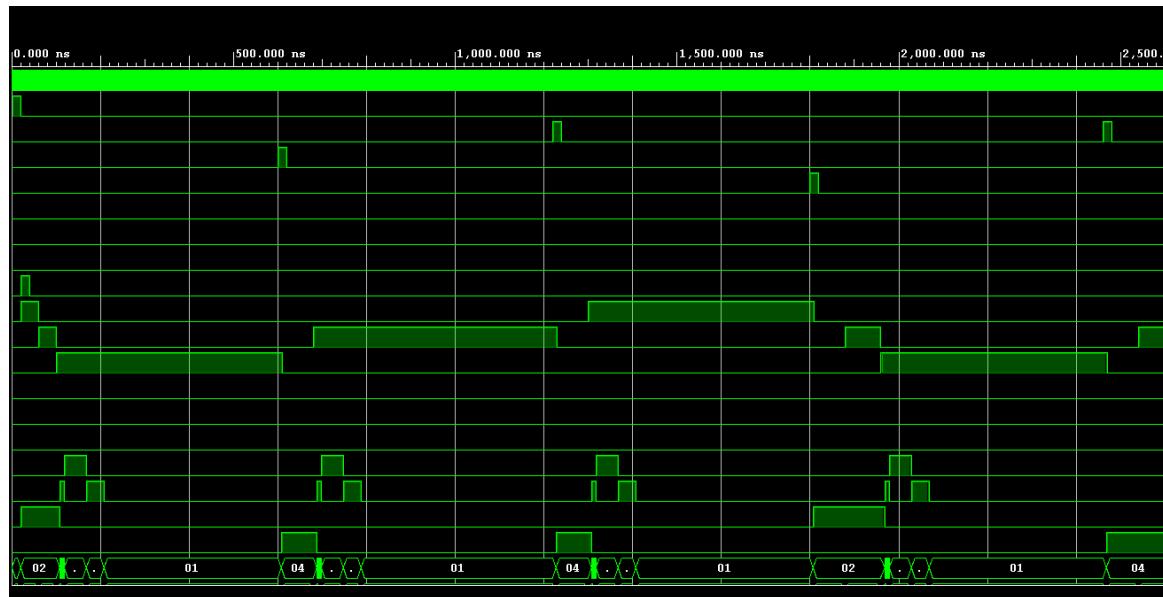


Ilustración 6 Gráfica de testbench de la maquina de estados.

Proyecto Elevador

Diagrama RTL

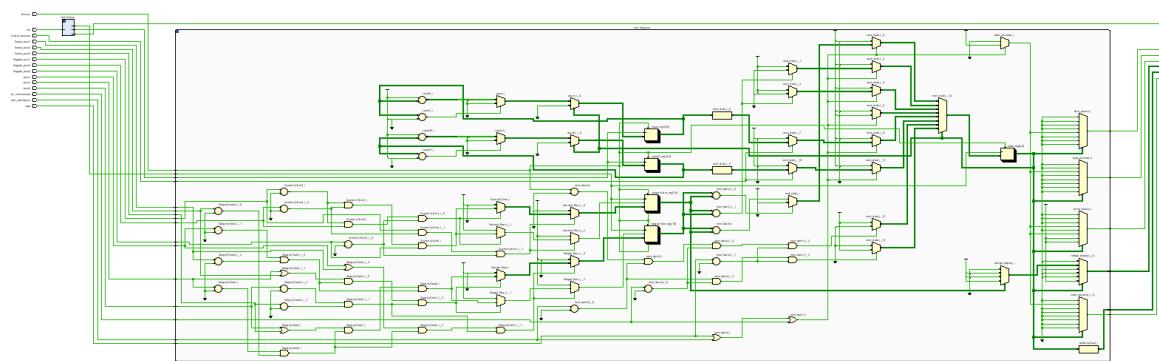


Ilustración 7 Diagrama RTL de la máquina de estados

Código Final VHDL

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 19.01.2025 04:34:10
6  -- Design Name:
7  -- Module Name: top - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity top is
35   Port (
36     clk : in STD_LOGIC;
37     btreset : in STD_LOGIC;
38     pisol, piso2, piso3 : in STD_LOGIC;
39     stop : in STD_LOGIC;
40     llamar_pisol, llamar_piso2, llamar_piso3 : in STD_LOGIC;
41     llegada_pisol, llegada_piso2, llegada_piso3 : in STD_LOGIC;
42     detector_persona, sensor_sobrepeso, sensor_movimiento : in STD_LOGIC;
43     abrir_puerta, cerrar_puerta, subir_ascensor, bajar_ascensor : out STD_LOGIC;
44     estado_actual : out STD_LOGIC_VECTOR(7 downto 0);
45     seg : out STD_LOGIC_VECTOR (6 downto 0); -- Salida para los segmentos del
46     display
47     anodo : out STD_LOGIC_VECTOR (7 downto 0) -- Control de los ánodos de los
48     displays
49   );
50 end top;
51
52 architecture Behavioral of top is
53
54 component ElevatorFSM
55   Port (
56     clk : in STD_LOGIC;
57     btreset : in STD_LOGIC;
58     pisol, piso2, piso3 : in STD_LOGIC;
59     stop : in STD_LOGIC;
60     llamar_pisol, llamar_piso2, llamar_piso3 : in STD_LOGIC;
61     llegada_pisol, llegada_piso2, llegada_piso3 : in STD_LOGIC;
62     detector_persona, sensor_sobrepeso, sensor_movimiento : in STD_LOGIC;
63     abrir_puerta, cerrar_puerta, subir_ascensor, bajar_ascensor : out STD_LOGIC;
64     clk_div : in STD_LOGIC;
65     estado_actual : out STD_LOGIC_VECTOR(7 downto 0);
66     codigo_display : out STD_LOGIC_VECTOR(2 downto 0)
67   );
68 end component;

```

Proyecto Elevador

```
68
69 component DivisorFrecuencia
70   Port ( clk_in : in STD_LOGIC; -- Reloj de entrada (100 MHz)
71         clk_sys : out STD_LOGIC; -- Reloj dividido
72         clk_states : out STD_LOGIC; -- Reloj dividido2
73         clk_dis : out STD_LOGIC -- Reloj dividido3
74       );
75 end component;
76
77 component Display
78   Port ( codigo_display : in STD_LOGIC_VECTOR (2 downto 0); -- Código de estado del
79          elevador
80          seg : out STD_LOGIC_VECTOR (6 downto 0); -- Salida para los segmentos del
81          display
82          anodo : out STD_LOGIC_VECTOR (7 downto 0); -- Control de los ánodos de los
83          displays
84          clk2 : in STD_LOGIC -- Reloj del sistema
85        );
86 end component;
87
88
89
90
91
92
93
94
95
96 begin
97
98   --ESTABLECER LA INSTANCIA PARA OBTENER LAS SEÑALES DE RELOJ
99   Inst_Divisor: DivisorFrecuencia PORT MAP (
100     clk_in => clk,
101     clk_sys => Sclk_sys,
102     clk_states => Sclk_states,
103     clk_dis => Sclk_dis
104   );
105
106   --Instancia para declarar la maquina de estados
107   Inst_Maquina: ElevatorFSM PORT MAP (
108     clk => Sclk_sys,
109     btreset => btreset,
110     pisol => pisol,
111     piso2 => piso2,
112     piso3 => piso3,
113     stop => stop,
114     llamar_pis01 => llamar_pis01,
115     llamar_pis02 => llamar_pis02,
116     llamar_pis03 => llamar_pis03,
117     llegada_pis01 => llegada_pis01,
118     llegada_pis02 => llegada_pis02,
119     llegada_pis03 => llegada_pis03,
120     detector_persona => detector_persona,
121     sensor_sobrepeso => sensor_sobrepeso,
122     sensor_movimiento => sensor_movimiento,
123     abrir_puerta => abrir_puerta,
124     cerrar_puerta => cerrar_puerta,
125     subir_ascensor => subir_ascensor,
126     bajar_ascensor => bajar_ascensor,
127     clk_div => Sclk_states,
128     estado_actual => estado_actual,
129     codigo_display => Scodigo_display
130   );
131
132   --Instancia para declarar el controlador de los displays
133   Inst_Display: Display PORT MAP (
```

Proyecto Elevador

```
134      codigo_display => Scodigo_display,
135      seg => seg,
136      anodo => anodo,
137      clk2 => Sclk_dis
138  );
139
140 end Behavioral;
141
```

Este código actúa como la mente del ascensor, incluimos los anteriores 3 códigos en esta composición. Está formado por diferentes partes que colaboran para que todo funcione de manera fluida y segura. Tenemos el siguiente modulo TOP, recibe información de botones, sensores de peso y movimiento, y luego decide qué pasos seguir. Cuando alguien presiona un botón para subir o bajar, el código primero revisa si todo está en orden: las puertas deben estar cerradas, no puede haber exceso de peso y no debe haber problemas detectados por los sensores. Solo cuando todo está correcto, el ascensor arranca en la dirección solicitada.

Mientras esto ocurre, hay otro módulo, que se encarga de las pantallas. Este módulo se asegura de que las personas, tanto dentro como fuera del ascensor, comprendan lo que sucede. Convierte señales internas en números y mensajes que son visibles, indicando el piso actual o si las puertas están abiertas. Todo esto se coordina gracias a un reloj interno que mantiene el ritmo, funcionando como un metrónomo que asegura que cada acción se realice de manera ordenada. Este diseño modular no solo facilita entender y mantener el sistema, sino que también permite adaptarse a nuevas necesidades en el futuro, como agregar más pisos o funciones adicionales.

Testbench

En este caso, la simulación del testbench es la misma que el código anterior. Por ende, para evitar ser repetitivos no se incluye.

ARCHIVO CONSTRAINTS

```

1 ## This file is a general .xdc for the Nexys A7-100T
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5 ## Note: As the Nexys 4 DDR was rebranded to the Nexys A7 with no substantial changes, this XDC file will also work for the Nexys 4 DDR.
6
7 ## Clock signal
8 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVC MOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
9 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];
10
11
12 ## Switches
13 #set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVC MOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_R50_15 Sch=sv[0]
14 #set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVC MOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCLK_14 Sch=sv[1]
15 #set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVC MOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
16 #set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVC MOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
17 #set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVC MOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
18 #set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVC MOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sv[5]
19 #set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVC MOS33 } [get_ports { illegada_piso1 }]; #IO_L24N_T3_R50_15 Sch=sv[0]
20 #set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVC MOS33 } [get_ports { illegada_piso2 }]; #IO_L3N_T0_DQS_EMCLK_14 Sch=sv[1]
21 #set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVC MOS18 } [get_ports { illegada_piso3 }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
22 #set_property -dict { PACKAGE_PIN U08 IOSTANDARD LVC MOS18 } [get_ports { stop }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
23 #set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVC MOS33 } [get_ports { llamar_piso1 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
24 #set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVC MOS33 } [get_ports { llamar_piso2 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
25 #set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVC MOS33 } [get_ports { llamar_piso3 }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
26 #set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVC MOS33 } [get_ports { detector_persona }]; #IO_L5N_T0_D07_14 Sch=sv[7]
27 #set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVC MOS33 } [get_ports { sensor_sobrepeso }]; #IO_L24N_T3_34 Sch=sv[8]
28 #set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVC MOS33 } [get_ports { sensor_movimiento }]; #IO_25_34 Sch=sv[9]
29
30 ## LEDs
31 #set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
32 #set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[1] }]; #IO_L24P_T3_R51_15 Sch=led[1]
33 #set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
34 #set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
35 #set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
36 #set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
37 #set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
38 #set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVC MOS33 } [get_ports { estado_actual[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
39 #set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVC MOS33 } [get_ports { codigo_display[0] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
40 #set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVC MOS33 } [get_ports { codigo_display[1] }]; #IO_L14P_T2_SRCC_14 Sch=led[9]
41 #set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVC MOS33 } [get_ports { codigo_display[2] }]; #IO_L22P_T2_A05_D21_14 Sch=led[10]
42 #set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVC MOS33 } [get_ports { abrir_puerta }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
43 #set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVC MOS33 } [get_ports { abrir_puerto }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[12]
44 #set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVC MOS33 } [get_ports { cerrar_puerta }]; #IO_L16P_T2_CSI_B_14 Sch=led[13]
45 #set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVC MOS33 } [get_ports { subir_ascensor }]; #IO_L22N_T3_A04_D20_14 Sch=led[14]
46 #set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVC MOS33 } [get_ports { bajar_ascensor }]; #IO_L20N_T3_A07_D23_14 Sch=led[15]
47 #set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVC MOS33 } [get_ports { led_clk }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[16]
48
49 ## RGB LEDs
50 #set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVC MOS33 } [get_ports { LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
51 #set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVC MOS33 } [get_ports { LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
52 #set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVC MOS33 } [get_ports { LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
53 #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVC MOS33 } [get_ports { LED17_B }]; #IO_L15N_T2_DOS_ADV_B_15 Sch=led17_b
54 #set_property -dict { PACKAGE_PIN R11 IOSTANDARD LVC MOS33 } [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
55 #set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVC MOS33 } [get_ports { LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r
56
57 ###7 segment display
58 #set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVC MOS33 } [get_ports { seg[6] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
59 #set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVC MOS33 } [get_ports { seg[5] }]; #IO_25_14 Sch=cb
60 #set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVC MOS33 } [get_ports { seg[4] }]; #IO_25_15 Sch=cc
61 #set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVC MOS33 } [get_ports { seg[3] }]; #IO_L17P_T2_A26_15 Sch=cd
62 #set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVC MOS33 } [get_ports { seg[2] }]; #IO_L13P_T2_MRCC_14 Sch=ce
63 #set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVC MOS33 } [get_ports { seg[1] }]; #IO_L18P_T3_A10_D26_14 Sch=cf
64 #set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVC MOS33 } [get_ports { seg[0] }]; #IO_L4P_T0_D04_14 Sch=cg
65 #set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVC MOS33 } [get_ports { DP }]; #IO_L18N_T3_A21_VREF_15 Sch=dp
66 #set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVC MOS33 } [get_ports { anodo[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
67 #set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVC MOS33 } [get_ports { anodo[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
68 #set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVC MOS33 } [get_ports { anodo[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
69 #set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVC MOS33 } [get_ports { anodo[3] }]; #IO_L18P_T3_A22_15 Sch=an[3]
70 #set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVC MOS33 } [get_ports { anodo[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
71 #set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVC MOS33 } [get_ports { anodo[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
72 #set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVC MOS33 } [get_ports { anodo[6] }]; #IO_L23P_T3_35 Sch=an[6]
73 #set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVC MOS33 } [get_ports { anodo[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
74
75 ##CPU Reset Button
76 #set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVC MOS33 } [get_ports { btreset }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
77
78 ##Buttons
79 #set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVC MOS33 } [get_ports { piso2 }]; #IO_L9P_T1_DQS_14 Sch=btnc
80 #set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVC MOS33 } [get_ports { piso3 }]; #IO_L4N_T0_D05_14 Sch=bttnu
81 #set_property -dict { PACKAGE_PIN P17 IOSTANDARD LVC MOS33 } [get_ports { btreset }]; #IO_L12P_T1_MRCC_14 Sch=bttnl

```

Proyecto Elevador

```

81: #set_property -dict { PACKAGE_PIN P17  IOSTANDARD LVCMS33 } [get_ports { btreset }]; #IO_L12P_T1_MRCC_14 Sch=btndl
82: set_property -dict { PACKAGE_PIN M17  IOSTANDARD LVCMS33 } [get_ports { stop }]; #IO_L10N_T1_D15_14 Sch=btndr
83: set_property -dict { PACKAGE_PIN P18  IOSTANDARD LVCMS33 } [get_ports { pisol }]; #IO_L9N_11_D05_D13_14 Sch=btnd
84:
85:
86: ##Pmod Headers
87: ##Pmod Header JA
88: set_property -dict { PACKAGE_PIN C17  IOSTANDARD LVCMS33 } [get_ports { abrir_puerta }]; #IO_L20N_T3_A19_15 Sch=j[a][1]
89: set_property -dict { PACKAGE_PIN D18  IOSTANDARD LVCMS33 } [get_ports { cerrar_puerta }]; #IO_L21N_T3_D05_A18_15 Sch=j[a][2]
90: set_property -dict { PACKAGE_PIN E18  IOSTANDARD LVCMS33 } [get_ports { subir_ascensor }]; #IO_L21P_T3_D05_15 Sch=j[a][3]
91: set_property -dict { PACKAGE_PIN G17  IOSTANDARD LVCMS33 } [get_ports { bajar_ascensor }]; #IO_L18N_T2_A23_15 Sch=j[a][4]#set_property -dict { PACKAGE_PIN D17  IOSTANDARD LVCMS33 } [get_ports { JA[8] }]; #IO_L16P_T2_A28_15 Sch=j[a][8]
92: #set_property -dict { PACKAGE_PIN E17  IOSTANDARD LVCMS33 } [get_ports { JA[9] }]; #IO_L22N_T3_A16_15 Sch=j[a][9]
93: #set_property -dict { PACKAGE_PIN F18  IOSTANDARD LVCMS33 } [get_ports { JA[10] }]; #IO_L22P_T3_A17_15 Sch=j[a][10]
94: #set_property -dict { PACKAGE_PIN G18  IOSTANDARD LVCMS33 } [get_ports { JA[11] }]; #IO_L12P_T2_MRCC_15 Sch=j[a][11]
95:
96: ##Pmod Header JB
97: set_property -dict { PACKAGE_PIN D14  IOSTANDARD LVCMS33 } [get_ports { llegada_pis01 }]; #IO_L16P_TO_AD0B_15 Sch=jb[1]
98: set_property -dict { PACKAGE_PIN F16  IOSTANDARD LVCMS33 } [get_ports { llegada_pis02 }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
99: set_property -dict { PACKAGE_PIN G16  IOSTANDARD LVCMS33 } [get_ports { llegada_pis03 }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
100: #set_property -dict { PACKAGE_PIN H14  IOSTANDARD LVCMS33 } [get_ports { JB[4] }]; #IO_L15P_T2_D05_15 Sch=jb[4]
101: #set_property -dict { PACKAGE_PIN E16  IOSTANDARD LVCMS33 } [get_ports { JB[7] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
102: #set_property -dict { PACKAGE_PIN F13  IOSTANDARD LVCMS33 } [get_ports { JB[8] }]; #IO_L5P_TO_AD0B_15 Sch=jb[8]
103: #set_property -dict { PACKAGE_PIN G13  IOSTANDARD LVCMS33 } [get_ports { JB[9] }]; #IO_Q_15 Sch=jb[9]
104: #set_property -dict { PACKAGE_PIN H16  IOSTANDARD LVCMS33 } [get_ports { JB[10] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]
105:
106: ##Pmod Header JC
107: set_property -dict { PACKAGE_PIN K1  IOSTANDARD LVCMS33 } [get_ports { llamar_pis01 }]; #IO_L23N_T3_35 Sch=jc[1]
108: set_property -dict { PACKAGE_PIN F6  IOSTANDARD LVCMS33 } [get_ports { llamar_pis02 }]; #IO_L19N_T3_VREF_35 Sch=jc[2]
109: set_property -dict { PACKAGE_PIN J2  IOSTANDARD LVCMS33 } [get_ports { llamar_pis03 }]; #IO_L22N_T3_35 Sch=jc[3]
110: #set_property -dict { PACKAGE_PIN G6  IOSTANDARD LVCMS33 } [get_ports { JC[4] }]; #IO_L19P_T3_35 Sch=jc[4]
111: #set_property -dict { PACKAGE_PIN E7  IOSTANDARD LVCMS33 } [get_ports { JC[7] }]; #IO_L6P_TO_35 Sch=jc[7]
112: #set_property -dict { PACKAGE_PIN J3  IOSTANDARD LVCMS33 } [get_ports { JC[8] }]; #IO_L22P_T3_35 Sch=jc[8]
113: #set_property -dict { PACKAGE_PIN J4  IOSTANDARD LVCMS33 } [get_ports { JC[9] }]; #IO_L21P_T3_D05_35 Sch=jc[9]
114: #set_property -dict { PACKAGE_PIN E6  IOSTANDARD LVCMS33 } [get_ports { JC[10] }]; #IO_L5P_TO_AD13P_35 Sch=jc[10]
115:
116: ##Pmod Header JD
117: set_property -dict { PACKAGE_PIN H4  IOSTANDARD LVCMS33 } [get_ports { sensor_sobrepeso }]; #IO_L01N_T3_D05_35 Sch=jd[1]
118: set_property -dict { PACKAGE_PIN H1  IOSTANDARD LVCMS33 } [get_ports { sensor_movimiento }]; #IO_L17P_T2_35 Sch=jd[2]
119: set_property -dict { PACKAGE_PIN G3  IOSTANDARD LVCMS33 } [get_ports { detector_persona }]; #IO_L17N_T2_35 Sch=jd[3]
120: #set_property -dict { PACKAGE_PIN G3  IOSTANDARD LVCMS33 } [get_ports { JD[4] }]; #IO_L20N_T3_35 Sch=jd[4]

```

Diagrama RTL

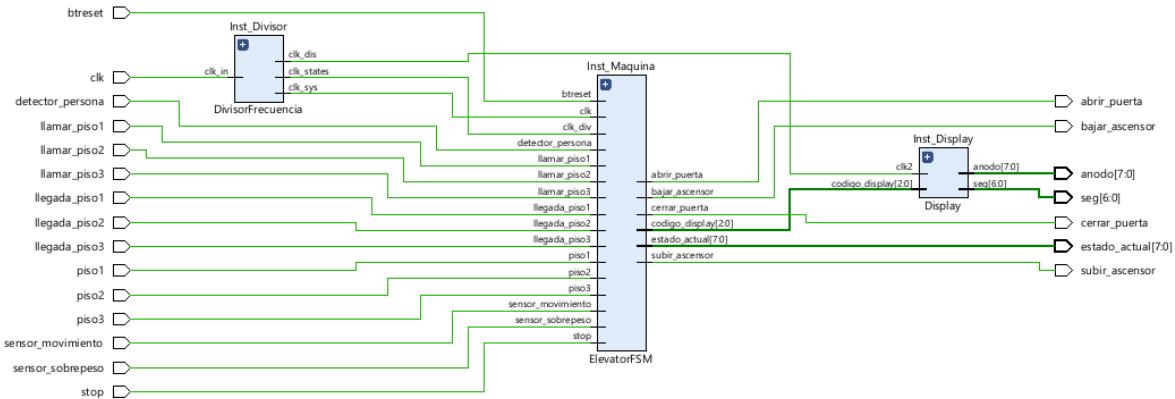


Ilustración 8 Diagrama RTL del código final

CÓDIGO MICROCONTROLADOR

```

1  /* USER CODE BEGIN Header */
2  /**
3   * @file           : main.c
4   * @brief          : Main program body
5   * @attention
6   *
7   * Copyright (c) 2025 STMicroelectronics.
8   * All rights reserved.
9   *
10  * This software is licensed under terms that can be found in the LICENSE file
11  * in the root directory of this software component.
12  * If no LICENSE file comes with this software, it is provided AS-IS.
13  *
14  */
15 /**
16 */
17 /**
18 /* USER CODE END Header */
19 /* Includes -----*/
20 #include "main.h"
21
22 /* Private includes -----*/
23 /* USER CODE BEGIN Includes */
24
25 /* USER CODE END Includes */
26
27 /* Private typedef -----*/
28 /* USER CODE BEGIN PTD */
29
30 /* USER CODE END PTD */
31
32 /* Private define -----*/
33 /* USER CODE BEGIN PD */
34
35 /* USER CODE END PD */
36
37 /* Private macro -----*/
38 /* USER CODE BEGIN PM */
39 uint16_t piso1;
40 uint16_t piso2;
41 uint16_t piso3;
42 uint16_t persona;
43 uint16_t subir;
44 uint16_t bajar;
45 volatile int16_t adcval;
46 uint16_t servo[6] = {700,750,950,1150,1350,1400}; // 6 posiciones distintas
47 uint8_t current_position = 0;
48 uint8_t spiTxBuf[2];
49 uint8_t spiRxBuf[2];
50
51 // Variable para almacenar el valor del eje X
52 int8_t accel_x = 0; // Eje X
53 int8_t accel_y = 0; // Eje Y
54 int8_t accel_z = 0; // Eje Z
55
56 /* USER CODE END PM */
57
58 /* Private variables -----*/
59 ADC_HandleTypeDef hadcl;
60
61 SPI_HandleTypeDef hspil;
62
63 TIM_HandleTypeDef htim2;
64
65 /* USER CODE BEGIN PV */
66
67 /* USER CODE END PV */
68
69

```

Proyecto Elevador

```
70  /* Private function prototypes -----*/
71 void SystemClock_Config(void);
72 static void MX_GPIO_Init(void);
73 static void MX_ADC1_Init(void);
74 static void MX_TIM2_Init(void);
75 static void MX_SPI1_Init(void);
76 /* USER CODE BEGIN PFP */
77
78 /* USER CODE END PFP */
79
80 /* Private user code -----*/
81 /* USER CODE BEGIN 0 */
82
83 // Función para leer los dos registros de 8 bits del eje X (OUT_X_H y OUT_X_L) y
84 // combinarlos en 16 bits
85
86
87 /* USER CODE END 0 */
88
89 /**
90  * @brief The application entry point.
91  * @retval int
92  */
93 int main(void)
94 {
95
96  /* USER CODE BEGIN 1 */
97
98  /* USER CODE END 1 */
99
100 /* MCU Configuration-----*/
101
102 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
103 HAL_Init();
104
105 /* USER CODE BEGIN Init */
106
107 /* USER CODE END Init */
108
109 /* Configure the system clock */
110 SystemClock_Config();
111
112 /* USER CODE BEGIN SysInit */
113
114 /* USER CODE END SysInit */
115
116 /* Initialize all configured peripherals */
117 MX_GPIO_Init();
118 MX_ADC1_Init();
119 MX_TIM2_Init();
120 MX_SPI1_Init();
121 /* USER CODE BEGIN 2 */
122 HAL_ADC_Start(&hadc1);
123 HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
124 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
125 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
126 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
127 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
128 /* USER CODE END 2 */
129
130 /* Infinite loop */
131 /* USER CODE BEGIN WHILE */
132 while (1)
133 {
134     HAL_ADC_Start(&hadc1);
135     if(HAL_ADC_PollForConversion(&hadc1, 100)==HAL_OK)
136     {
137         adcval=HAL_ADC_GetValue(&hadc1);
```

Proyecto Elevador

```
138     if (adcval>300) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, 1);
139     else HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, 0);
140 }
141
142 if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_5) == GPIO_PIN_SET) // Cerrar
143 {
144     HAL_Delay(200); // Anti-rebote
145     if (current_position < 5) {
146         current_position = (current_position + 1) % 6;
147     }
148     // Ajusta el PWM para mover el servo
149     HAL_TIM_Set_COMPARE(&htim2, TIM_CHANNEL_2, servo[current_position]);
150 }
151
152 if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_7) == GPIO_PIN_SET) // Abrir
153 {
154     HAL_Delay(200); // Anti-rebote
155     if (current_position > 0) {
156         current_position = (current_position - 1 + 6) % 6;
157     }
158     // Ajusta el PWM para mover el servo
159     HAL_TIM_Set_COMPARE(&htim2, TIM_CHANNEL_2, servo[current_position]);
160 }
161
162 piso1 = HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_7);
163 piso2 = HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_9);
164 piso3 = HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_11);
165 persona = HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_13);
166 subir = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3);
167 bajar = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_6);
168
169 if (piso1 == GPIO_PIN_SET) {
170     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET); //Llegada piso 1
171 } else {
172     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_RESET);
173 }
174 if (piso2 == GPIO_PIN_SET) {
175     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET); //Llegada piso 2
176 } else {
177     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
178 }
179 if (piso3 == GPIO_PIN_SET) {
180     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0, GPIO_PIN_SET); //Llegada piso 3
181 } else {
182     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0, GPIO_PIN_RESET);
183 }
184 if (persona == GPIO_PIN_SET) {
185     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET); //Detección de persona
186 } else {
187     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
188 }
189 if (subir == GPIO_PIN_SET) {
190     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); //Subir elevador
191 } else {
192     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
193 }
194 if (bajar == GPIO_PIN_SET) {
195     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET); //Bajar elevador
196 } else {
197     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
198 }
199
200 HAL_Delay(200); //Tiempo de espera para evitar rebotes
201
202
203
204 /* USER CODE END WHILE */
205
206 /* USER CODE BEGIN 3 */
```

Explicación del código

Este código su función es manejar las señales de los sensores de piso, los botones que controlan el ascensor, un servo motor que abre y cierra las puertas, y un sensor de aceleración para detectar movimientos inusuales. El programa está organizado de manera que se inicializan los diferentes periféricos del microcontrolador, se recogen las entradas digitales y analógicas, y se realizan acciones basadas en esas lecturas dentro de un bucle sin fin.

1. Configuración de Variables y Periféricos

Se declaran varias variables globales para guardar el estado de los sensores (piso1, piso2, piso3, persona) y los botones del ascensor (subir, bajar). La variable adcval es de tipo volatile int16_t, lo que implica que puede cambiar debido a interrupciones o eventos externos. También se crea un array de posiciones para el servo (servo[6]) con valores de PWM preestablecidos para abrir y cerrar. Se establecen buffers para la transmisión y recepción necesarios para la comunicación SPI con el sensor de aceleración (spiTxBuf y spiRxBuf), además de variables para los valores del acelerómetro en los ejes X, Y y Z (accel_x, accel_y, accel_z).

Asimismo, se declaran manejadores para los periféricos ADC, y Timer PWM (hadc1, hspi1, htim2), que permiten la interacción con los sensores y dispositivos conectados al microcontrolador.

3. Inicialización del Sistema

Dentro de main(), se lleva a cabo la inicialización de los periféricos:

- HAL_Init(); configura la HAL (Hardware Abstraction Layer) del STM32.
- SystemClock_Config(); establece la frecuencia del reloj del sistema.
- MX_GPIO_Init();, MX_ADC1_Init();, MX_TIM2_Init();, MX_SPI1_Init(); se encargan de preparar los pines GPIO, el ADC, el PWM respectivamente.

Se inician los módulos requeridos para que el ascensor funcione correctamente:

- HAL_ADC_Start(&hadc1); inicia la conversión ADC.
- HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); activa la señal PWM para el servo.

Los LEDs del panel de control del ascensor se configuran para que empiecen apagados (RESET).

4. Bucle Principal (while 1)

El sistema entra en un bucle infinito (while(1)) donde se monitorean constantemente los sensores y se toman decisiones en tiempo real.

Proyecto Elevador

Lectura del Sensor ADC

Cada ciclo comienza con HAL_ADC_Start(&hadc1);, permitiendo que el convertidor ADC obtenga valores analógicos. Se espera hasta que la conversión termine (HAL_ADC_PollForConversion(&hadc1, 100)) y se guarda el resultado en adcval. Si este valor supera 300, se enciende un LED (GPIOC, GPIO_PIN_11), lo que podría señalar un problema de sobrepeso o una advertencia.

Control del Servo para la Puerta

El servo motor abre y cierra la puerta del ascensor según la presión de los botones:

- Si se presiona el botón de cierre (GPIOB, GPIO_PIN_5), se incrementa la posición del servo (current_position) y se ajusta el PWM con __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, servo[current_position]);.
- Si se presiona el botón de apertura (GPIOB, GPIO_PIN_7), se decrementa la posición del servo y se actualiza el PWM.

Detección de Sensores de Piso y Control de LEDs

Se leen los sensores de piso (piso1, piso2, piso3), el sensor de presencia (persona) y los botones de subir y bajar (subir, bajar). Dependiendo de si un sensor está activo (GPIO_PIN_SET), se encienden los LEDs correspondientes:

- GPIOD, GPIO_PIN_4 indica la llegada al piso 1.
- GPIOD, GPIO_PIN_2 indica la llegada al piso 2.
- GPIOD, GPIO_PIN_0 indica la llegada al piso 3.
- GPIOD, GPIO_PIN_13 se activa al detectar una persona dentro del ascensor.
- GPIOD, GPIO_PIN_14 se ilumina cuando el ascensor está subiendo.
- GPIOD, GPIO_PIN_15 se enciende cuando el ascensor está bajando.

5. Conclusión

Este código controla nuestro ascensor leyendo los sensores de piso, los botones de control y detectando movimientos con un acelerómetro. La integración del ADC permite vigilar señales analógicas, la comunicación SPI facilita la obtención de datos del sensor de aceleración, y el PWM controla el servo de la puerta. Además, se utilizan LEDs para mostrar el estado del ascensor y mejorar la interacción con los usuarios. En conjunto, el código presenta una solución segura y eficiente para el control de un ascensor en un microcontrolador STM32.

Proyecto Elevador

Circuitos.

Como parte de la conexiones del proyecto, para poder simular la presencia de botones de un elevador usaremos los botoenes que ofrece la Nexys, pero al no ser suficientes, necesitaremos agregar otros tres botones más. Los cuales representan los botones que se ubican fuera del elevador y ser llamado. El uso que se les da a los botones de la Nexys es, para los botones centrales se hace la elección de piso por dentro del elevador. Mientras que el botón derecho se usa para hacer un paro de emergencia (stop).

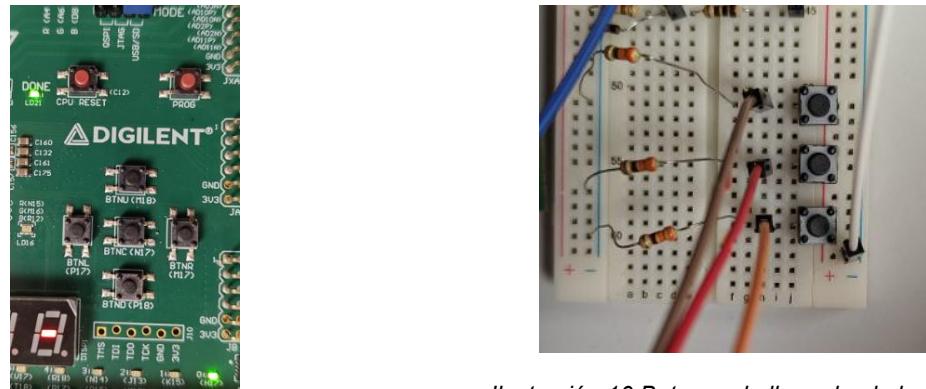


Ilustración 10 Botones de llamada al elevador.

Ilustración 9 Botones internos del elevador, paro y reset

Posteriormente, para simular la llegada del elevador al piso que se ha indicado, se hace uso de sensores infrarrojos, esto se hace mediante algún objeto que interrumpe el paso de la comunicación de los sensores y así poder mandar una señal a la Nexys para poder hacer el cambio de estado.

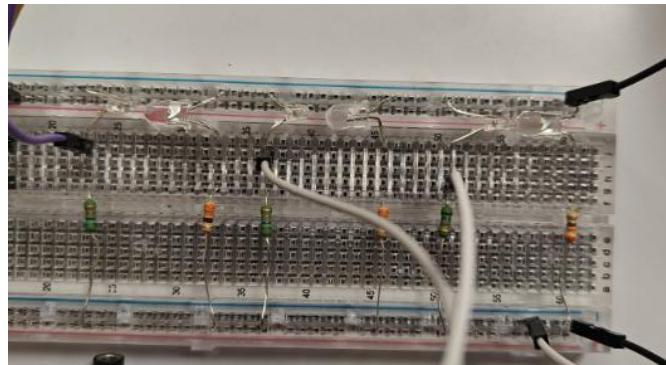


Ilustración 11 Sensores de arriba del elevador.

Para poder hacer las comunicaciones de entradas de la Nexys y salidas del microcontrolador, se necesitó hacer un conversor de 5V a 3.3V para evitar dañar las tarjetas. Esto se hace mediante el uso de diodos Zener.

Proyecto Elevador

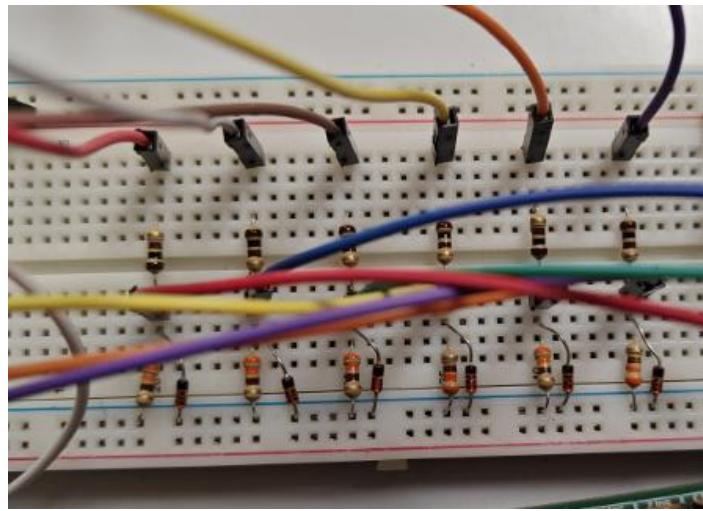


Ilustración 12 Conversor de voltaje entre micro y Nexys

Para poder entender cómo es que trabaja la máquina de estados, además de programar el display para avisar al usuario sobre el estado actual del elevador, también se usaron los leds de la propia Nexys y así tener una mejor visión de como es que funciona.

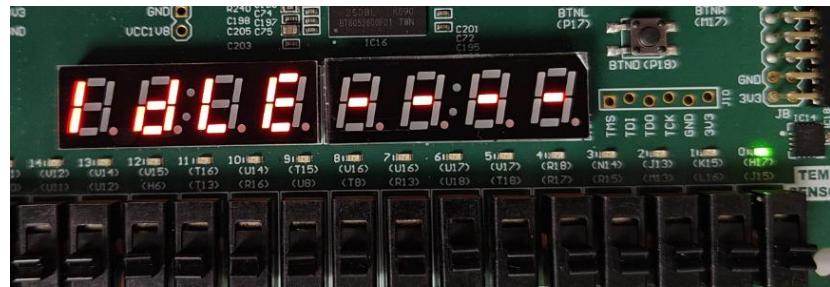


Ilustración 13 Display y leds

Finalmente, los elementos que nos ayudan a simular diferentes condiciones en el elevador son las siguientes:

- Servo motor: Este elemento simula la apertura y cierre de puertas del elevador. Esto lo hace mediante una configuración donde el servo gira 90° para hacer la acción.



Ilustración 14 Servomotor

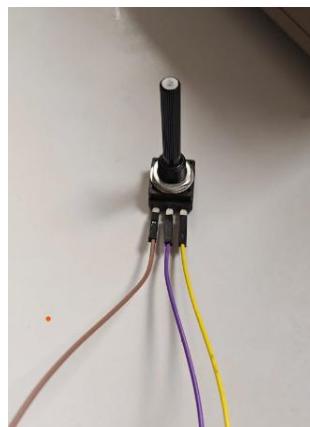
Proyecto Elevador

- Como segundo elemento, tomamos de nueva cuenta un sistema de sensores infrarrojos para poder hacer una detección de persona que se encuentre en la entrada del elevador. Esto con el fin de que si la máquina de estados entra el estado de cierre de puertas, se vuelvan a abrir y evitar un “accidente”



Ilustración 15 Sensor infrarrojo de puerta del elevador.

- Finalmente tenemos la presencia del potenciómetro que simula la presencia de un sensor de peso, el cual tiene un límite para activar el estado de error



CONCLUSION

En conclusión, podemos decir que el objetivo final se cumplió de manera satisfactoria, se logró implementar de buena manera ambas placas de desarrollo, tanto la nexys 4 como la STM32, esto mediante una buena planeación, una buena división de tareas y un buen sistema de organización del trabajo.

Si bien se tuvieron retos complicados, como la aparición de errores relacionados al uso de relojes en el código VHDL o el de la implementación de la comunicación SPI de la STM32, así mismo se pudieron solucionar algunos problemas como lo son, confusiones al momento de implementar la máquina de estados, el uso de ciertos programas que al final tuvieron que eliminarse como lo es el detector de flancos y el sincronizador, que no funcionaron de la manera esperada.

Por otro lado, la mejor ayuda que tuvimos es el uso de los testbench, que nos permitieron poder probar los códigos de manera satisfactoria antes de su implementación física, así mismo se hicieron circuitos físicos de prueba para probar los códigos de la STM32 y poder verificar su correcto funcionamiento.

Finalmente, podemos decir que se cumplió la rúbrica del proyecto, en este caso de uno que implementaba ambas placas, ya que se usaron de manera completa los conocimientos adquiridos durante las clases, como lo son, Máquina de estado, Elaboración de Testbench, Uso de códigos en VHDL, Convertidores Analógico-Digitales, Timers, PWM, etc.

BIBLIOGRAFIA

Vishay Semiconductors, "BPW85A, BPW85B, BPW85C Silicon NPN Phototransistor," Vishay, 2023. [Online]. Available: www.vishay.com.

Vishay Semiconductors, "BZX55-Series Small Signal Zener Diodes," Vishay, 2023. [Online]. Available: www.vishay.com.

J. Castro-Godínez, *Diodo Zener, Lección 03.2*, Instituto Tecnológico de Costa Rica, Escuela de Ingeniería Electrónica, II Semestre 2013.

STMicroelectronics, *Discovery kit with STM32F411VE MCU*, User Manual UM1842, DocID027213 Rev 1, Dec. 2014. [Online]. Available: [www.st.com](http://www.st.com/#contentReference[oaicite:0]{index=0});

STMicroelectronics, *STM32F411xC/E Advanced ARM-based 32-bit MCUs*, Reference Manual RM0383, DocID026448 Rev 2, Sep. 2017. [Online]. Available: [www.st.com](http://www.st.com/#contentReference[oaicite:1]{index=1});

STMicroelectronics, *STM32F411xC, STM32F411xE Datasheet*, DocID026289 Rev 7, Dec. 2017. [Online]. Available: [www.st.com](http://www.st.com/#contentReference[oaicite:2]{index=2});

Digilent, Inc., *Nexys A7 FPGA Board Reference Manual*, Rev. July 10, 2019. [Online]. Available: [www.digilentinc.com](http://www.digilentinc.com/#contentReference[oaicite:3]{index=3});

MMSV, *Analog Servo HD-1160A Specification*, Version V1, 2023.