TRABAJO

DE

SISTEMAS ELECTRÓNICOS DIGITALES

CURSO 2021/2022

FPGA:

ASCENSOR DE 4 PLANTAS

Pablo Pérez 54157

Mario Mondejar 54103

Rubén Pleiter 54808

FECHA: 15-01-2021

Tabla de contenido

PROPÓSITOS

El principal objetivo de este trabajo ha sido familiarizarnos aún más con las FPGA y en este caso, con el lenguaje VHDL y la aplicación física en la placa correspondiente.

Como diseño de trabajo, hemos elegido un ascensor único en una vivienda de 4 pisos. Este diseño proponía los siguientes requisitos:

- Entradas del circuito: Piso deseado (4 botones) y Piso actual (4 botones).
- Salidas del circuito: Motor (2 bits) y Puerta (1 bit).

Funcionalidades:

- El ascensor debe ir al piso indicado por los botones, cuando llegue abrirá las puertas, que permanecerán abiertas hasta que reciba otra llamada.
- Si el ascensor se está moviendo, ignora los botones de piso deseado.
- Utilizar LEDs y Displays para visualización de información.

INTRODUCCIÓN

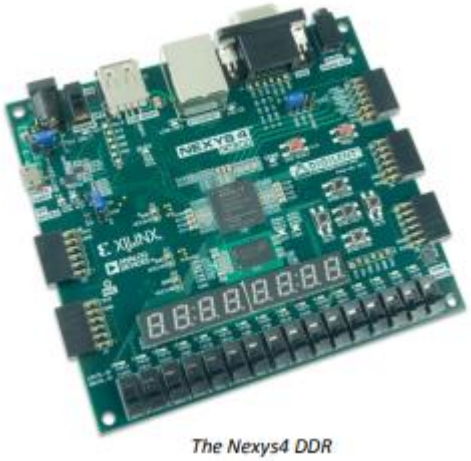
VHDL es un lenguaje que empieza a desarrollarse en el Departamento de Defensa de los Estados Unidos, llamado VHSIC (Very High Speed Integrated Circuits) [1]. En este lenguaje, la principal idea es que no se programa, se describen circuitos electrónicos.

Surge principalmente por la crisis del ciclo de vida del Hardware, cada vez los circuitos integrados contenían más complicaciones, y la sustitución de estos era bastante compleja, ya que no existía una base de documentación. Por ello, nació como una manera de documentar circuitos.

Este lenguaje tiene como principal función modelar circuitos. Estos modelos, pueden ser simulados a través de un test bench para comprobar si la funcionalidad es la que queremos o también se pueden sintetizar para crear un circuito funcione como el modelo.

CARACTERÍSTICAS DE LA PLACA

La placa utilizada en el proyecto es la Nexys4 DDR. Esta contiene un reloj de 100MHz. Con el clk_divider conseguiremos un reloj de 200 Hz que usaremos para refrescar el propio display de la placa. También contiene un botón de reset que usaremos para volver al estado inicial, con el piso actual en el 1, con las puertas abiertas y motor apagado.



The Nexys4 DDR

DISEÑO

MAQUINA DE ESTADOS

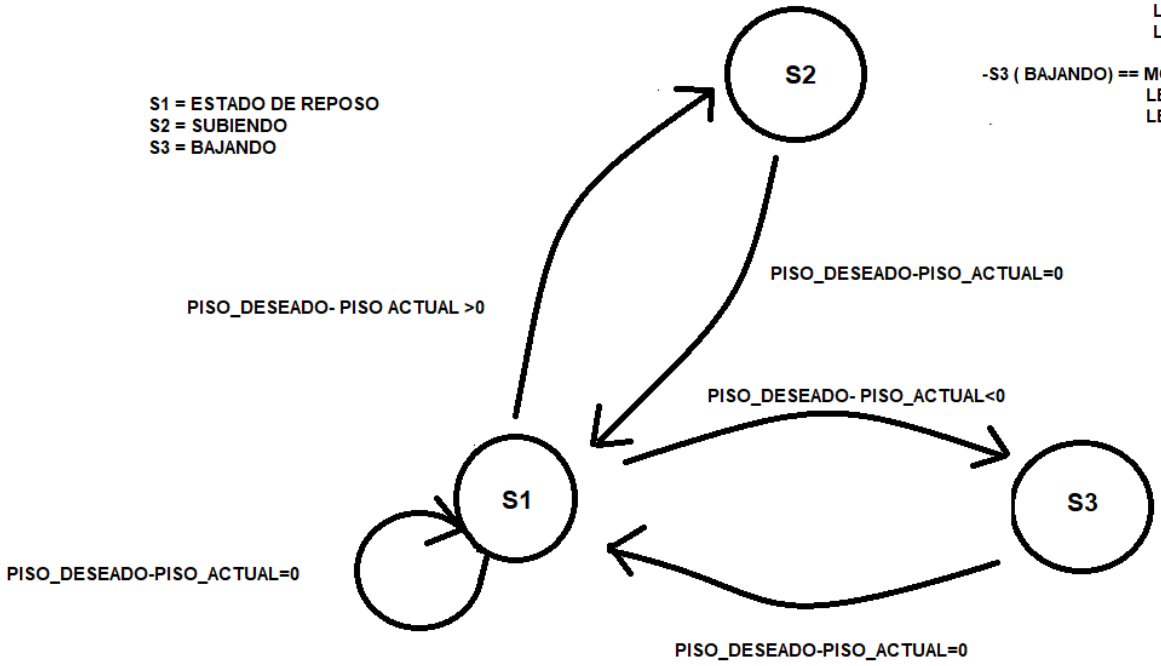
CADA ESTADO LLEVA ASOCIADO UNAS SALIDAS :

- S1 (REPOSO) == MOTOR "00"
LED_LLEGADA "1"
LED_PUERTA "1"

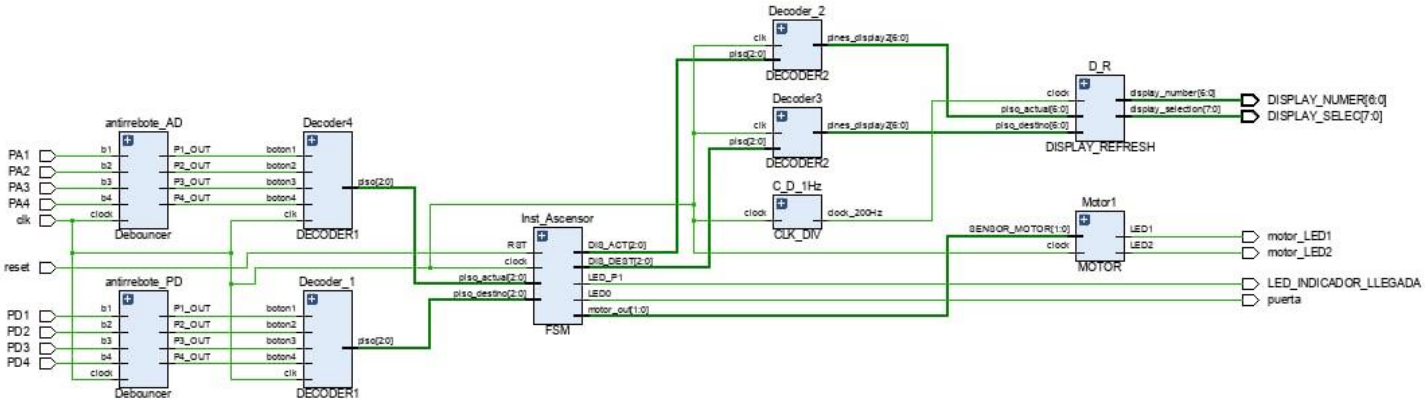
- S2 (SUBIENDO) == MOTOR "01"
LED_LLEGADA "0"
LED_PUERTA "0"

-S3 (BAJANDO) == MOTOR "10"
LED_LLEGADA "0"
LED_PUERTA "0"

S1 = ESTADO DE REPOSO
S2 = SUBIENDO
S3 = BAJANDO



DIAGRAMA



1. Los botones pasan por un debouncer eliminando los rebotes.
2. Estos pasan por un decoder para generar el código del piso deseado y piso actual.
3. El código se dirige a la máquina de estados FSM, que tiene como salidas el piso actual, el piso deseado y el estado del motor y de la puerta como una señal de 2 y 1 bit respectivamente.
4. El piso deseado y piso actual pasa por el decoder del display y por el refrescador de display.
5. El estado del motor pasa por una entidad Motor que gestiona los leds dedicados al motor.
6. La señal de los led de llega y led de la puerta pasa directamente a ser expuesta por los leds.

ANÁLISIS DE ENTIDADES

Todo el código se encuentra en https://github.com/pabloprzgil/Trabajo_FPGA .

DEBOUNCER:

El debouncer ha sido una implementación de una muestra de código de internet.[2]
Tiene como entradas los botones y el reloj y como salidas los propios botones.

DECODER1:

Tiene como entradas los botones salidos del debouncer y como salidas el piso codificado en 3 bits.
Convierte la pulsación de botón en código:

Botón	Piso
1	001
2	010
3	011
4	100

```
decoder: PROCESS (clk, boton1, boton2, boton3, boton4)
    BEGIN
        IF rising_edge(clk) THEN
            IF boton1='1' THEN
              piso <= "001";
            ELSIF boton2='1' THEN
              piso <= "010";
            ELSIF boton3='1' THEN
              piso <= "011";
            ELSIF boton4='1' THEN
              piso <= "100";
            END IF;
          END IF;
END PROCESS;
```

Este se implementa 2 veces, para decodificar los botones pulsados y los sensores de piso actual.

DECODER2:

Convierte el código sacado de la FSM de piso actual y piso deseado en código para el display.
Convierte la planta actual en los pines del display.

```
decoder2: PROCESS (clk, piso)
    BEGIN
        IF rising_edge(clk) THEN
        IF piso = "001" THEN
              pines_display2 <= "1001111";
        ELSIF piso = "010" THEN
              pines_display2 <= "0010010";
        ELSIF piso = "011" THEN
              pines_display2 <= "0000110";
        ELSIF piso = "100" THEN
              pines_display2 <= "1001100";
              END IF;
            END IF;
     END PROCESS;

end Behavioral;
```

FSM:

Contiene diferentes process.
El primer process, maneja el reset y la transformación de las salidas de los decoder (piso deseado y piso actual) en enteros para poder restarlos.

También contiene el process de gestión de estados, explicado en la máquina de estados mostrada anteriormente.

```
NEXT_STATE_DECODE: PROCESS (clock, state, P_D, P_A)

  BEGIN
    next_state <= state;
     case state is
     when S1 =>
         IF (P_D-P_A=0 ) THEN       -- planta deseada
                    next_state <= S1;       --estado de reposo
         ELSIF (P_D-P_A>0) THEN   -- la planta deseada está arriba
                    next_state <= S2;    -- estado subiendo
         ELSIF (P_D-P_A<0) THEN     -- planta deseada esta abajo
                    next_state <= S3;    -- estado bajando
         END IF;
     when S2=>
       IF (P_D-P_A=0) THEN      -- planta deseada
                next_state <= S1; -- reposo
                END IF;
      when S3=>
         IF (P_D-P_A=0) THEN        -- planta deseada
                next_state <= S1; -- reposo
                END IF;
END CASE;
END PROCESS;
```

Por último, contiene la gestión de las salidas (motor, llegada a destino y puerta) según el estado.

CLK_DIVIDER:

Extraído de los apuntes de clase. Lo hemos utilizado para conseguir una señal de reloj de 200 Hz para refrescar el display.

Primero se cambia la frecuencia del reloj a 1 Hz para después, con una variable count o cnt, conseguir la frecuencia deseada contando los pulsos.

MOTOR:

Gestiona los LEDs referidos al motor, se enciende uno u otro dependiendo si está subiendo o bajando.

TOP:

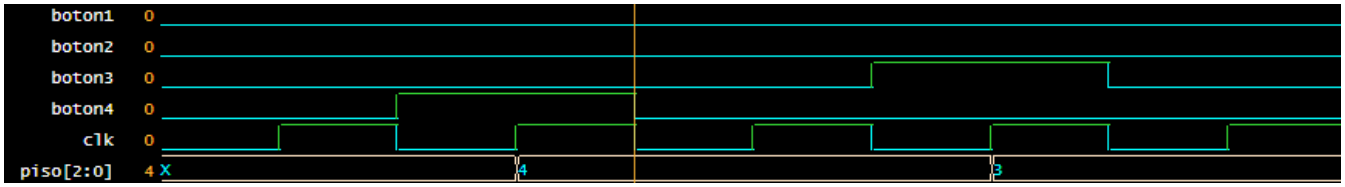
Contiene la definición de los componentes y sus correspondientes instancias. Es la entidad base del proyecto.

IMPLEMENTACIÓN Y SIMULACIÓN EN LA PLACA

1. Cuando la placa se enciende, partimos de un estado inicial donde estamos en el primer piso, las puertas están abiertas y el motor no funciona.
2. Los 4 primeros interruptores simulan los botones para elegir el piso al que queremos ir y los 4 siguientes los sensores de cada planta.
3. Los displays muestran el piso actual y el piso deseado.
4. El usuario selecciona el piso deseado y posteriormente se cierra la puerta y se enciende el led del motor en modo subida o bajada según corresponda
5. Como los sensores son interruptores, tendremos que indicarlos manualmente que va avanzando cada piso.
6. En caso de pulsar un botón mientras subes/bajas, el ascensor lo guarda para que cuando llegue al piso deseado se dirija al siguiente pulsado.

TEST BENCH

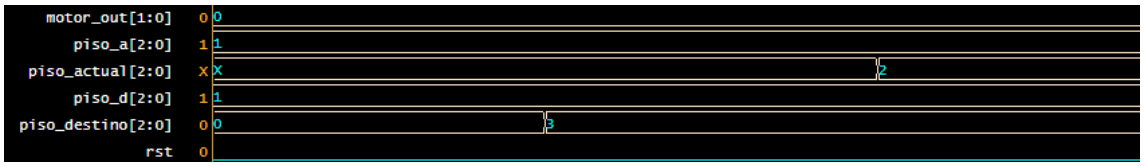
Hemos decidido incluir el testbench del decoder y de la fsm en la memoria. Se han realizado más, incluidos en el repositorio.




```
entity DECODER1_tb is
end DECODER1_tb;
architecture Behavioral of DEC1_tb is
  component DECODER1
     Port ( clk : in STD_LOGIC;
     boton1 : in STD_LOGIC;
     boton2 : in STD_LOGIC;
     boton3 : in STD_LOGIC;
     boton4 : in STD_LOGIC;
     piso : out STD_LOGIC_VECTOR (2 DOWNTO 0));
  end component;
      constant K: time := 100 ns;
      signal b1: std_logic:='0';
      signal b2: std_logic:='0';
      signal b3: std_logic:='0';
      signal b4: std_logic:='0';
      signal ck: std_logic:='0';
      signal p: std_logic_vector (2 downto 0);
  begin

decoder_tb: DECODER1
 port map(
 clk => ck,
 boton1 => b1,
 boton2 => b2,
 boton3 => b3,
 boton4 => b4,
 piso => p
 );
ck <= not ck after 0.5 * K;
b4 <= '1' after 1 * k, '0' after 2 * k;
b3 <= '0', '1' after 3*k, '0' after 4*k;
end Behavioral;
```

Pulsamos el botón 3 y 4 y da como salida 3 y 4 que es lo que buscamos.




```
entity FSM_tb is

end FSM_tb;

architecture Behavioral of FSM_tb is

COMPONENT FSM
    Port ( clock : in STD_LOGIC;
           RST: in STD_LOGIC;
           piso_destino : in STD_LOGIC_VECTOR (2 DOWNTO 0);
           piso_actual : in STD_LOGIC_VECTOR (2 DOWNTO 0);
           LED_P1: out STD_LOGIC;
           LED0: out STD_LOGIC;
           DIS_DEST: out STD_LOGIC_VECTOR (2 DOWNTO 0);
           DIS_ACT: out STD_LOGIC_VECTOR (2 DOWNTO 0);
           motor_out : out STD_LOGIC_VECTOR (1 downto 0)
           );
END COMPONENT;

constant K: time := 100 ns;
SIGNAL clk: STD_LOGIC:='0';
SIGNAL rst: STD_LOGIC:='0';
SIGNAL piso_destino: STD_LOGIC_VECTOR (2 DOWNTO 0):= "000";
SIGNAL motor_out: STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL piso_actual: STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL LED_P1: STD_LOGIC:='0';
SIGNAL LED0: STD_LOGIC:='0';
SIGNAL piso_d: STD_LOGIC_VECTOR (2 DOWNTO 0):= "000";
SIGNAL piso_a: STD_LOGIC_VECTOR (2 DOWNTO 0);


begin

ascensor: FSM
    PORT MAP(
        clock => clk,
        RST => rst,
        piso_destino => piso_destino,
        motor_out => motor_out,
        piso_actual => piso_actual,
        DIS_DEST => piso_d,
        DIS_ACT => piso_a,
        LED_P1 => LED_P1,
        LED0 => LED0
        );


clk <= not clk after 0.5 * K;
piso_destino <= "011" after 0.4 * K, "000" after 8.5 * K;
piso_actual <= "010" after 0.4 * K, "000" after 8.5 * K;


end Behavioral;
```

Cambiamos algunas variables del piso y vemos resultados

BIBLIOGRAFÍA

1. https://vhdl.es/tutorial-vhdl/
2. https://allaboutfpga.com/vhdl-code-for-debounce-circuit-in-fpga/