TRABAJO VHDL: ASCENSOR

SISTEMAS ELECTRÓNICOS DIGITALES

GRUPO B, A-408, LUNES 11:30

Pablo Sanz Hernández - 52625

Ricardo Andrés Baños Rojas - 52240

1. Objetivo del trabajo

El objetivo del trabajo será, tal y cómo se indica en el guion de Moodle, diseñar el control de un ascensor para una vivienda compuesta por cuatro pisos. Las entradas al circuito serán el piso al que el usuario desea ir mediante cuatro botones, y el piso en el que el ascensor está en un momento dado. Por otra parte, habrá otras dos salidas: la del motor (2 bits), y la de la puerta (1 bit).

El funcionamiento se basa en los siguientes apartados:

- El ascensor debe ir al piso indicado por los botones, cuando llegue abrirá las puertas y permanecerán así hasta que se reciba la siguiente llamada.
- Si, mientras el ascensor se mueve, se pulsan otros botones, se ignorará y no se deberá hacer caso.
- Utilizar los LEDs y los Displays para visualizar la información.

2. Requisitos del trabajo

Los requisitos que se deberán presentar en el trabajo serán:

- Empleo de entradas manuales (pulsadores, switches, etc.) y salidas (LEDs, Displays de 7 segmentos, etc.).
- Sincronización de las entradas.
- Uso de la señal de reloj de la placa.
- Uso de señal de RESET.
- Uso de una máquina de estados.

Por otra parte, se subirá a Moodle una documentación que deberá contener:

- MEMORIA (en formato PDF): Recogerá los siguientes apartados:
  - Una breve introducción (resumen del enunciado).
  - Descripción de la estrategia y algoritmos desarrollados para la realización del trabajo, justificando las soluciones adoptadas.
  - Diagramas: de bloques, de estados, de tiempos, etc.
  - Explicación detallada del funcionamiento de cada uno de los bloques funcionales y de su interfaz.
- DISEÑO: Ficheros relacionados con el diseño:
  - Código fuente VHDL.
  - Fichero del proyecto (Vivado).
  - Otros ficheros relacionados con el diseño (restricciones, pruebas).

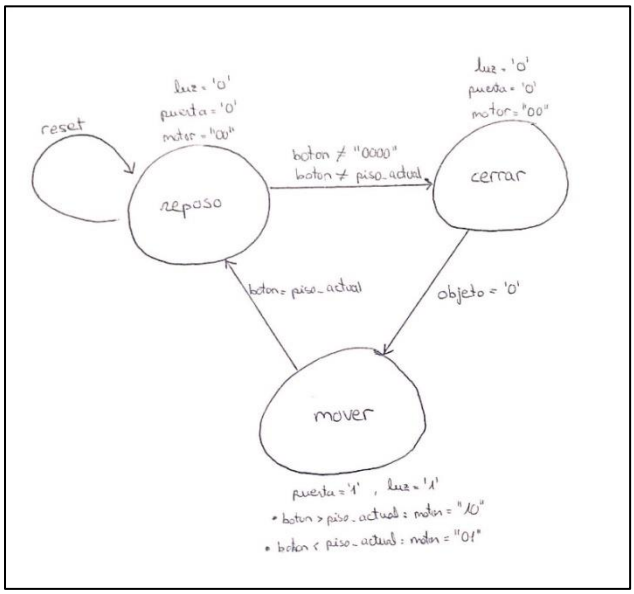
3. Realización del trabajo

El funcionamiento del ascensor es bastante básico: si no pulsamos ningún botón se mantendrá en el estado inicial, y si pulsamos un botón entonces se cerrará la puerta y el motor se pondrá en marcha en cierta dirección según el botón pulsado (‘01’ hacia derecha, ‘10’ hacia izquierda, y ‘00’ parado). Al llegar al piso marcado, se abrirán las puertas y permanecerán abiertas hasta una nueva llamada.

La forma más sencilla de realizar el código será a partir de una máquina de estados. Esta estará compuesta por 3 estados: *reposo, cerrar y mover*.

- El estado ‘reposo’ es el estado por defecto y espera a que se pulse un botón para almacenarlo y pasar a cerrar las puertas. El ascensor esperará con la puerta abierta y con el motor que mueve el ascensor parado. Permaneceremos en este estado hasta que se pulse el botón de un piso distinto al piso donde se encuentra el ascensor para pasar al siguiente estado, o hasta la pulsación de reset.
- El estado ‘cerrar’ comprueba que la puerta del ascensor no detecte ningún obstáculo. Cuando el sensor de la puerta indica que está libre de obstáculos, pasaremos entonces a cerrar la puerta e ir al siguiente estado.
- El estado ‘mover’ moverá el ascensor en función del botón pulsado: si el piso indicado es mayor al piso presente ascenderá, y en caso de ser inferior descenderá. El ascensor permanece en movimiento hasta que llegue al piso indicado momento en el cual regresa al estado de reposo.

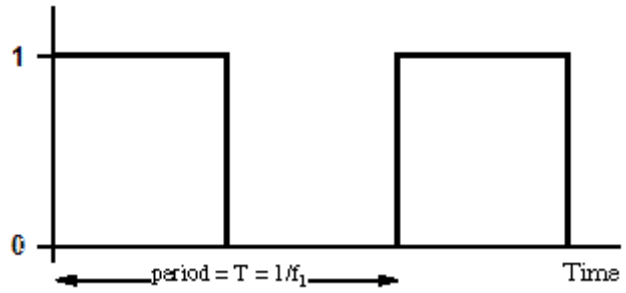
El uso de máquinas de estados de Moore será de gran ayuda para controlar aquellos diseños que presentan distintos estados durante su funcionamiento, como ocurre en nuestro caso.La máquina de estados del ascensor será una máquina de Mealy, ya que las salidas dependen tanto del estado actual como de las entradas. El esquema de esta máquina se refleja a continuación:



- Entradas del sistema: Botón, piso actual, objeto.
- Salidas del sistema: Motor, puerta, luz.
- Puerta será ‘0’ cuando se encuentre abierta y ‘1’ cuando esté cerrada.
- Botón y piso actual tendrán valores comprendidos entre 1 y 4.
- Motor será ‘10’ cuando gire hacia la derecha (ascensor subiendo), ‘01’ cuando gire hacia la izquierda (ascensor bajando), y ‘00’ cuando esté en reposo.

Aparte de las entradas y salidas del sistema se han añadido dos señales para ver el estado en el que nos encontramos, y para capturar la pulsación del botón. La captura del botón realiza una comprobación de que no se pulsan dos simultáneamente, ya que esto supondría un error para el ascensor, y únicamente se registrará si la pulsación es correcta.

Para el cálculo del periodo de la señal del reloj de la placa simplemente realizamos la inversa de la frecuencia especificada en la señal del reloj de la placa. En nuestro caso la frecuencia tendrá un valor de f = 100 MHz, y por tanto el periodo tomará un valor de clk_period = 10ns.



De esta forma, deberemos especificar en el código que el reloj cambie de flanco cada clk_period/2 = 5 ns. La inclusión de este factor en el código se realizaría así:

```
clk_process: PROCESS
BEGIN
	clk <= '0';
	WAIT FOR clk_period/2;
	clk <= '1';
	WAIT FOR clk_period/2;
END PROCESS;
```

3.1. Versiones

El funcionamiento del ascensor programado no es del todo realista (las puertas se cierran inmediatamente al llamar al ascensor desde otro piso, el movimiento del mismo no realiza esperas y lo hace de forma muy brusca…).

Las versiones siguientes han sido añadidas para agregar un punto de realidad al funcionamiento del ascensor, con funciones como la detección de objetos que impiden cerrar las puertas mediante un sensor, o un uso eficiente y rentable de la luz del ascensor que evite gastos innecesarios.

Primera versión

- La primera versión del programa incluye únicamente la problemática expuesta en el enunciado del trabajo: en el momento en que pulsamos un botón se cerrará la puerta y el motor se pondrá en marcha en cierta dirección según el botón pulsado, pero si no pulsamos ningún botón, se mantendrá en el estado inicial. Al llegar al piso marcado, se abrirán las puertas y permanecerán abiertas hasta una nueva llamada.

Segunda versión

- Esta versión añade una nueva funcionalidad al mecanismo del ascensor: permite la detección de objetos situados en la puerta que suponen un problema a la hora de cerrar las puertas. Esto lo conseguiremos añadiendo una nueva entrada *objeto*, que tratará de un sensor colocado a la altura de la puerta que comprueba que las puertas se pueden cerrar sin problema.

Tercera versión

- Esta última versión es una mejora de la anterior, ya que le añade el hecho de que el ascensor no gaste más energía de la requerida. Esto se conseguirá encendiendo la luz del ascensor únicamente cuando este se esté desplazando, manteniéndola por tanto apagada cuando el ascensor no esté en uso.

- Además se han añadido dos señales (`destino_fijado y piso_destino`) a la arquitectura del ascensor para que este último vaya al piso que inicialmente pusimos como destino aún cuando pulsemos otro botón durante el trayecto. Una vez que el ascensor llegue a su primer destino, se verificará si los botones siguen marcando ese destino. Si no se da ese caso, el ascensor de desplazara al nuevo destino.

3.2. Asignación de pines

Para adaptar los pines de nuestra placa debemos configurar en el programa que usamos LVCMOS33 I/O estándar. Una vez hecho esto, se ha decidido realizar la siguiente asignación:

- El control de los botones del ascensor lo realizaremos con los switches situados a la izquierda de la placa.
- El control del piso en el que se encuentra el ascensor lo manejaremos con los switches situados a la derecha de la placa.
- La salida del motor la podremos observar con los dos primeros LEDs situados a la izquierda de la placa.
- El estado de la puerta del ascensor se observará en el LED situado más a la derecha de la placa.
- La luz del ascensor mostrará su estado en el segundo LED más a la derecha de la placa (contiguo al LED de la puerta).
- La presencia de objetos que impidan que se cierren las puertas del ascensor la controlaremos con el botón superior de los cinco botones negros.
- El reset se implementará en el botón central de los cinco botones negros.

De esta forma, los pines asignados son los siguientes:

PIN N17: reset

PIN H17: puerta

PIN M18: objeto

PIN K15: luz

PIN R15: piso_actual[0]

PIN M13: piso_actual[1]

PIN L16: piso_actual[2]

PIN J15: piso_actual[3]

PIN V10: boton[0]

PIN U11: boton[1]

PIN U12: boton[2]

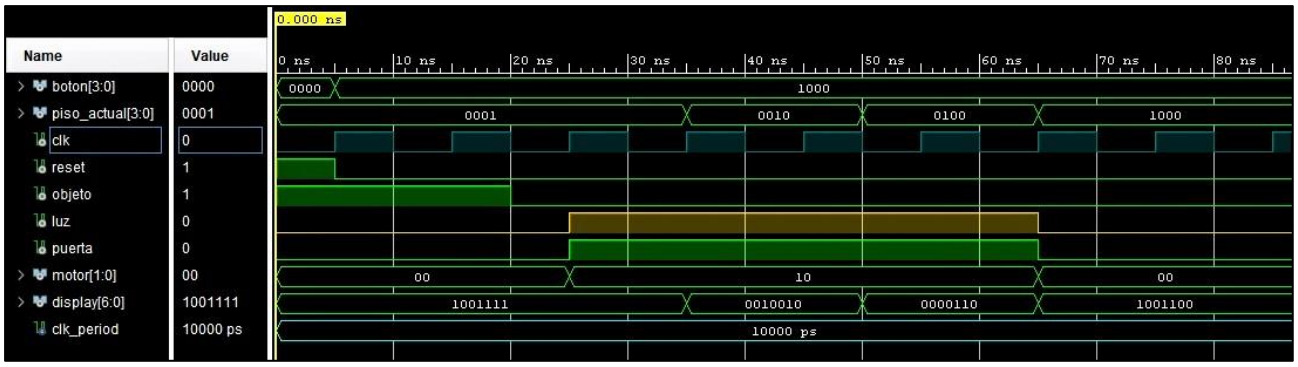
PIN H6: boton[3]

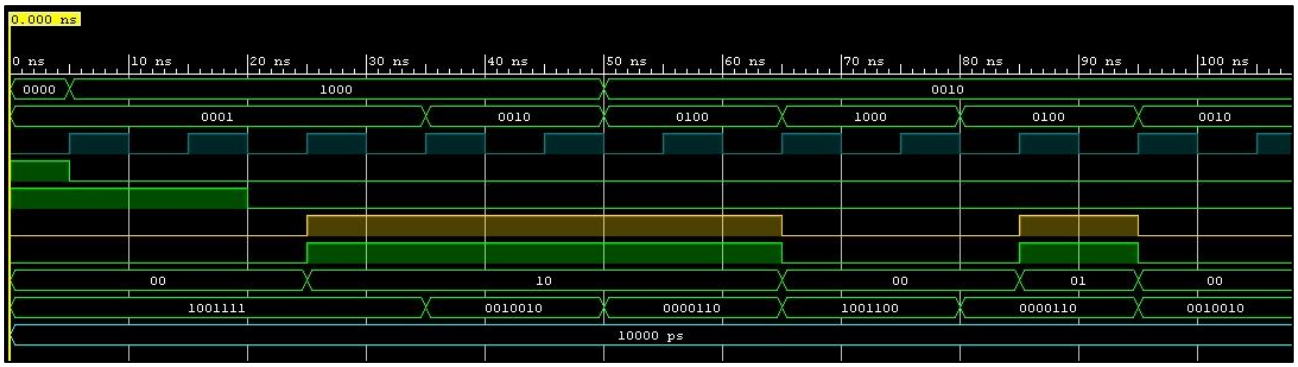
PIN V11: motor[0]

PIN V12: motor[1]

3.3. Simulación

En ambos ejemplos se puede ver que hacemos que el ascensor se desplace del primer al cuarto piso. En el segundo sin embargo se puede apreciar cómo, aun cambiando el piso de destino (se cambia de cuatro a dos) mientras el ascensor está desplazandose, el destino no cambia hasta que llega al final del primer trayecto. Una vez que se detecta que el botón marca el segundo piso, el ascensor baja.





El código del testbench responsable de las formas de onda del segundo ejemplo se encuentra al final del documento.

4. Código VHDL

Entidad y arquitectura de Top

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY top IS
    PORT (
        boton, piso_actual: IN std_logic_vector (3 downto 0);
```

```
          clk, reset: IN std_logic;
          objeto: IN std_logic;
          luz, puerta: OUT std_logic;
          motor: OUT std_logic_vector (1 downto 0);
          display: OUT std_logic_vector (6 downto 0)
          );
    END top;

ARCHITECTURE behavioral OF top IS

    COMPONENT decoder
        port(
            code : in std_logic_vector(3 downto 0);
            led : out std_logic_vector(6 downto 0)
        );
    END COMPONENT;

    TYPE estado IS (reposo, cerrar, mover);
    SIGNAL presente: estado:= reposo;
    SIGNAL boton_pulsado: std_logic_vector(3 DOWNTO 0):=boton;
    SIGNAL destino_fijado: std_logic:= '0';
    SIGNAL piso_destino: std_logic_vector(3 DOWNTO 0);
BEGIN

    INST_DECODER: decoder PORT MAP(
        code => piso_actual,
        led => display);

    estados: PROCESS (reset, clk)
    BEGIN
        IF reset = '1' THEN
            presente <= reposo;

        ELSIF clk = '1' AND clk'event THEN
```

```
            CASE presente IS

            WHEN reposo =>
                IF boton /= "0000" AND boton /= piso_actual THEN
                    presente <= cerrar;
                    boton_pulsado <= boton;
                END IF;

            WHEN cerrar =>
                IF objeto = '0' THEN
                    IF destino_fijado = '0' THEN
                        piso_destino <= boton_pulsado;
                        destino_fijado <= '1';
                    END IF;
                    presente <= mover; --Sin obstáculo
                END IF;

            WHEN mover =>
                IF piso_actual = piso_destino THEN
                    destino_fijado <= '0';
                    presente <= reposo; --Ya llego al piso
                END IF;
            END CASE;
        END IF;
    END PROCESS;

    salida: PROCESS (presente)
    BEGIN
        CASE presente IS
            WHEN mover =>
                luz <= '1'; --Luz encendida
                puerta <= '1'; --Cierra puerta

                IF boton_pulsado > piso_actual THEN --Ascensor sube
```

```
                    motor <= "10";

                ELSE --Ascensor baja
                    motor <= "01";
                END IF;

            WHEN OTHERS =>
                motor <= "00";  --Ascensor Parado
                puerta <= '0';  --Puerta abierta
                luz <= '0'; --Luz apagada
        END CASE;
    END PROCESS;
END behavioral;
```

Entidad y arquitectura de Decoder

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY decoder IS
    PORT(
        code: IN std_logic_vector(3 DOWNTO 0);
        led: OUT std_logic_vector(6 DOWNTO 0));
END ENTITY;

ARCHITECTURE dataflow OF decoder IS
BEGIN
    WITH code SELECT
        led <= "0000001" WHEN "0000",
            "1001111" WHEN "0001",
            "0010010" WHEN "0010",
            "0000110" WHEN "0100",
            "1001100" WHEN "1000",
            "0000001" WHEN OTHERS;
END dataflow;
```

Testbench de Top

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY top_tb is
END top_tb;

ARCHITECTURE Behavioral of top_tb is
    COMPONENT top is
        PORT(
            boton, piso_actual: IN std_logic_vector (3 downto 0);
            clk, reset: IN std_logic;
            objeto: IN std_logic;
            luz, puerta: OUT std_logic;
            motor: OUT std_logic_vector (1 downto 0);
            display: OUT std_logic_vector (6 downto 0));
    END COMPONENT;

    SIGNAL boton, piso_actual : std_logic_vector (3 DOWNTO 0);
    SIGNAL clk, reset, objeto: std_logic;
    SIGNAL luz, puerta: std_logic;
    SIGNAL motor: std_logic_vector(1 DOWNTO 0);
    SIGNAL display: std_logic_vector(6 DOWNTO 0);
    CONSTANT clk_period: time := 10 ns;

BEGIN

    utt: top PORT MAP(
        boton => boton,
        piso_actual => piso_actual,
        clk => clk,
        reset => reset,
        objeto => objeto,
```

```
        luz => luz,
        puerta => puerta,
        motor => motor,
        display => display);

    clk_process: PROCESS
    BEGIN
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
    END PROCESS;

    simul_process: PROCESS
    BEGIN
        reset <= '1';
        piso_actual <= "0001";
        boton <= "0000";
        objeto <= '1';
        WAIT FOR 5 ns;

        reset <= '0';
        boton <= "1000";
        WAIT FOR 15 ns;

        objeto <= '0';
        WAIT FOR 15 ns;

        piso_actual <= "0010";
        WAIT FOR 15 ns;

        piso_actual <= "0100";
        boton <= "0010";
        WAIT FOR 15 ns;
```

```
        piso_actual <= "1000";
        WAIT FOR 15 ns;

        piso_actual <= "0100";
        WAIT FOR 15 ns;

        piso_actual <= "0010";
        WAIT;

    END PROCESS;
END Behavioral;
```