**SISTEMAS ELECTRÓNICOS DIGITALES**

**TRABAJO VHDL**

**NOMBRE:** LUIS PEDRERO MORCILLO, 52541

**NOMBRE:** JAVIER LASERNA MORATALLA, 52432

**NOMBRE:** ALEJANDRO MORA SÁNCHEZ, 52496

**CURSO:** 18/19

Índice

[Introducción 3](#_Toc535845961)

[Algoritmos y estrategias de diseño 4](#_Toc535845963)

[Problemas y soluciones adoptadas 4](#_Toc535845964)

[Mejoras implementadas. 5](#_Toc535845965)

[Diagramas 6](#_Toc535845966)

[Funcionamiento de los componentes 7](#_Toc535845967)

[Decodificador BCD a 7 segmentos 7](#_Toc535845968)

[Descripción 7](#_Toc535845969)

[Máquina de estados 8](#_Toc535845970)

[Descripción 8](#_Toc535845971)

[Clk divider 10](#_Toc535845972)

[Descripción 10](#_Toc535845973)

[Pulse register 11](#_Toc535845974)

[Descripción 11](#_Toc535845975)

[Anexo 12](#_Toc535845976)

[Código fuente y testbench 12](#_Toc535845977)

[Ascensor 12](#_Toc535845978)

[BCD 15](#_Toc535845979)

[Pulse\_register 17](#_Toc535845980)

[Máquina de estados 21](#_Toc535845981)

[Clk\_divider 26](#_Toc535845982)

# Introducción

El objetivo de este trabajo es diseñar el controlador de un ascensor único en una vivienda de 4 pisos. Las entradas al circuito serán: el piso al que el usuario desea ir (4 botones), y el piso en el cual se encuentra el ascensor en un momento dado. Las salidas del circuito serán: por un lado, la del motor (2bits) y por otro la de la puerta (1 bit). El funcionamiento normal será: el ascensor se moverá al piso indicado por los botones, cuando alcance dicho piso abrirá las puertas que permanecerán abiertas hasta que reciba otra llamada. Durante el movimiento del ascensor entre pisos, si se pulsan los botones, el ascensor no modificará su movimiento.

# 

# Algoritmos y estrategias de diseño

La idea inicial de diseño de nuestro ascensor estuvo basada en el diseño esquemático proporcionado en el enunciado que se nos adjudicó. En un primer lugar se desarrolló la idea general de realizar la implementación de un ascensor, en el cuál pulsando una serie de botones específicos, cada uno asignado a un piso distinto, éste se moviese al piso indicado. Una vez presionado un botón, transcurrido cierto tiempo, aparecía el número del piso en el display y se encenderían dos leds, uno para indicar si la puerta está abierta o cerrada y otro para indicar si el motor del ascensor está en funcionamiento o no. Con esta idea general, se procedió a su implementación mediante el desarrollo de distintos módulos/componentes que lo llevasen a cabo. En primer lugar se creó una serie de componentes igual a los indicados en el enunciado: una máquina de estado encargada de determinar en qué piso (estado) se encuentra el ascensor, controlado por una señal de entrada generada por los botones; un clock divider, el cuál realiza una división de la frecuencia con el objetivo de crear una señal de reloj distinta a la de la placa, adecuada a nuestras necesidades; y un decodificador BCD de 7 segmentos encargado de mostrar la salida del piso en el que se encuentra a través del display. Sin embargo, este primer diseño no era del todo correcto ya que se nos presentaron una serie de problemas que impedían un buen funcionamiento del ascensor.

### Problemas y soluciones adoptadas

1) Se detectó un problema en la máquina de estados, el cuál impedía que el ascensor fuese a un piso indicado mediante un botón si en el transcurso de dicho proceso se presionaba otro botón distinto. Por ejemplo, encontrándose en el piso 0 y pulsando el botón para que fuese al piso 3, si en dicho transcurso se pulsaba el botón para ir al piso 2, se paraba en éste, por lo que no cumplía la especificación de que el ascensor no modificase su movimiento.

La solución adoptada fue crear un componente adicional, un pulse register, el cuál nos permite mediante una máquina de estados más sencilla, registrar el botón presionado para después pasarlo a la máquina de estados principal, evitando así modificar la entrada de la máquina de estado principal directamente.

2) Por otro lado, se detectó un fallo a la hora de realizar la acción de reset. En un principio al pulsar el botón de reset, la máquina de estados lo reconocía como un estado ‘0’ y el ascensor descendía al piso 0 de igual forma que si se pulsaba el botón de piso 0.

Para solucionar esto, la señal de reset se puso independiente de la señal de reloj.

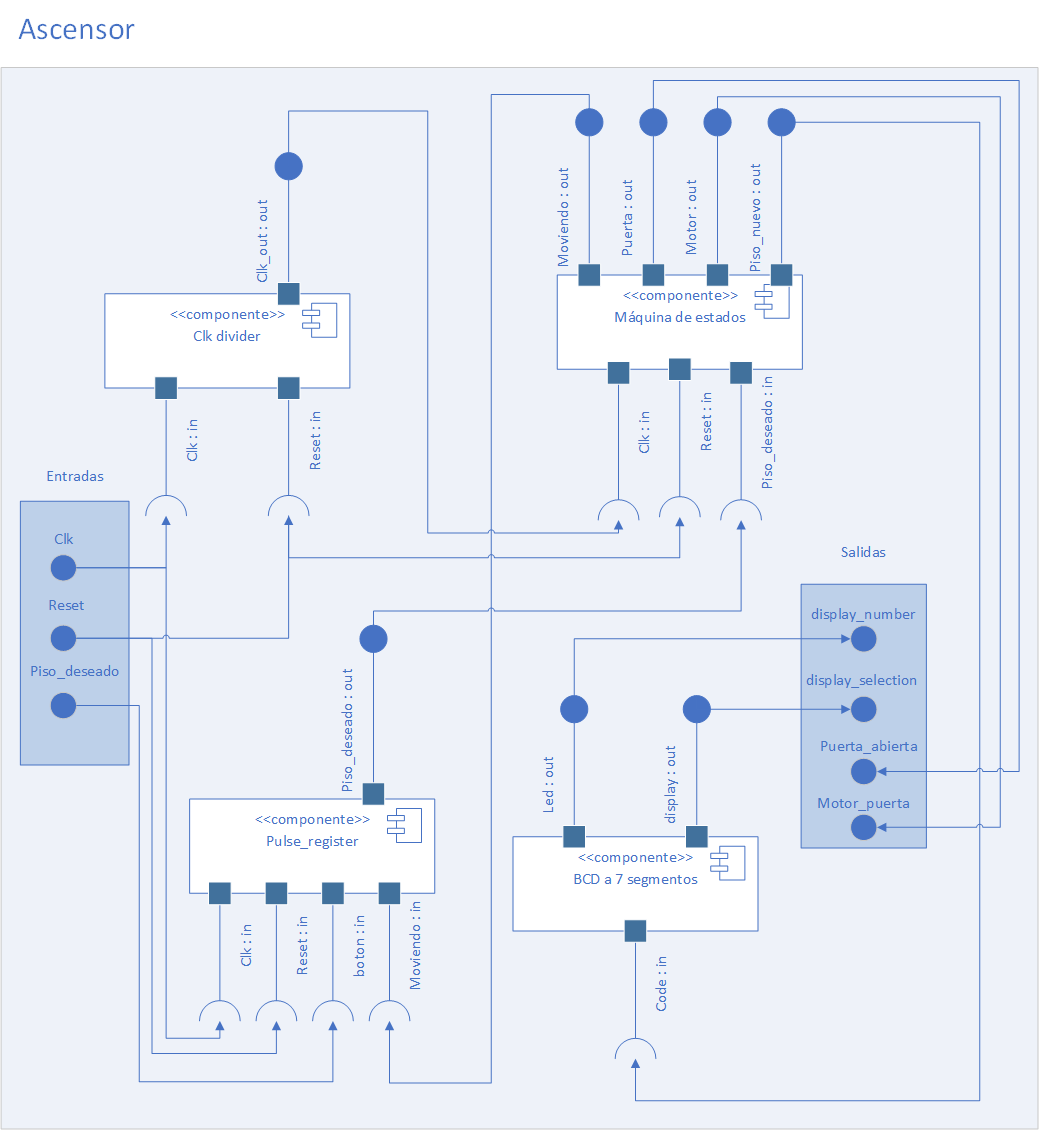
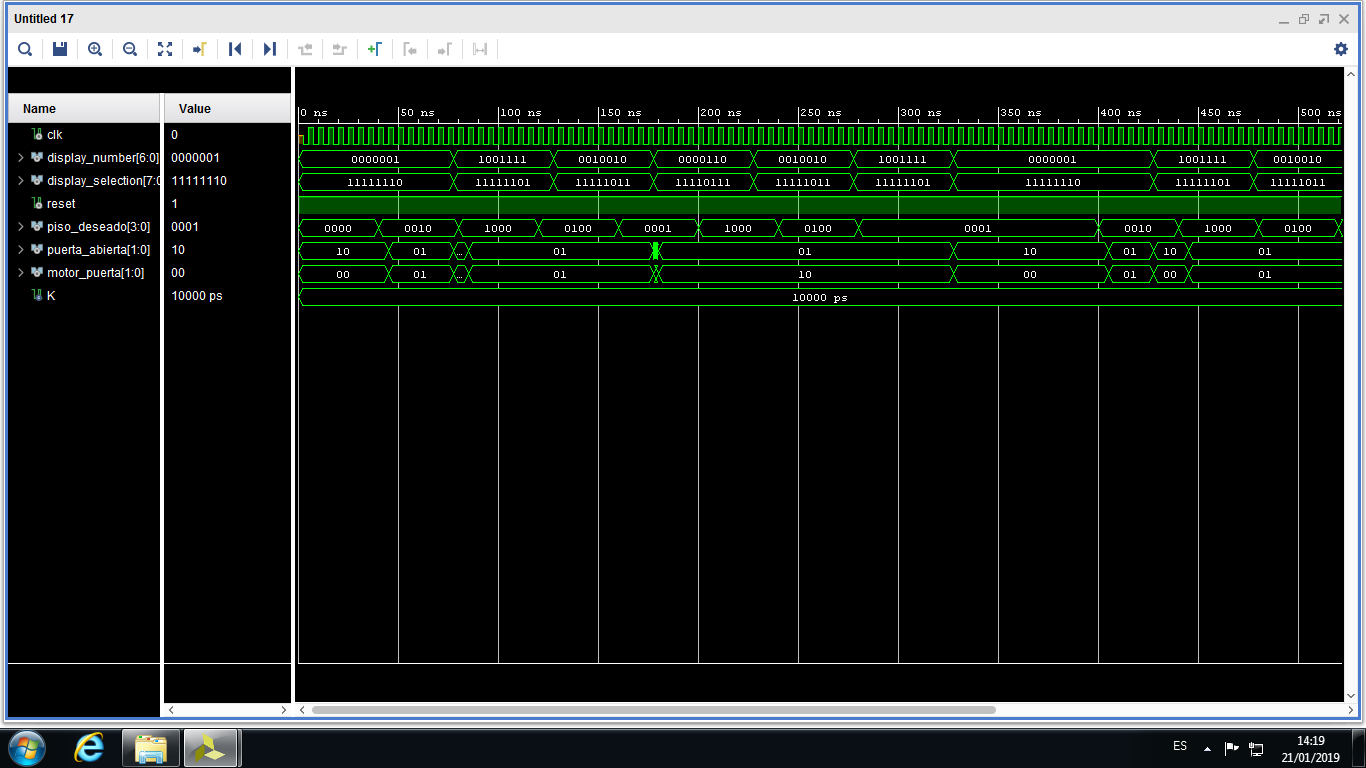
### Mejoras implementadas.

Por otra parte, se han implementado de forma adicional una serie de mejoras del diseño proporcionado:

1) Imitando el funcionamiento de un ascensor, cada número de cada piso se muestra en un dígito distinto, de forma que el dígito más bajo correspondiente al piso 0 se muestra en la derecha y el más alto en la posición de la izquierda.

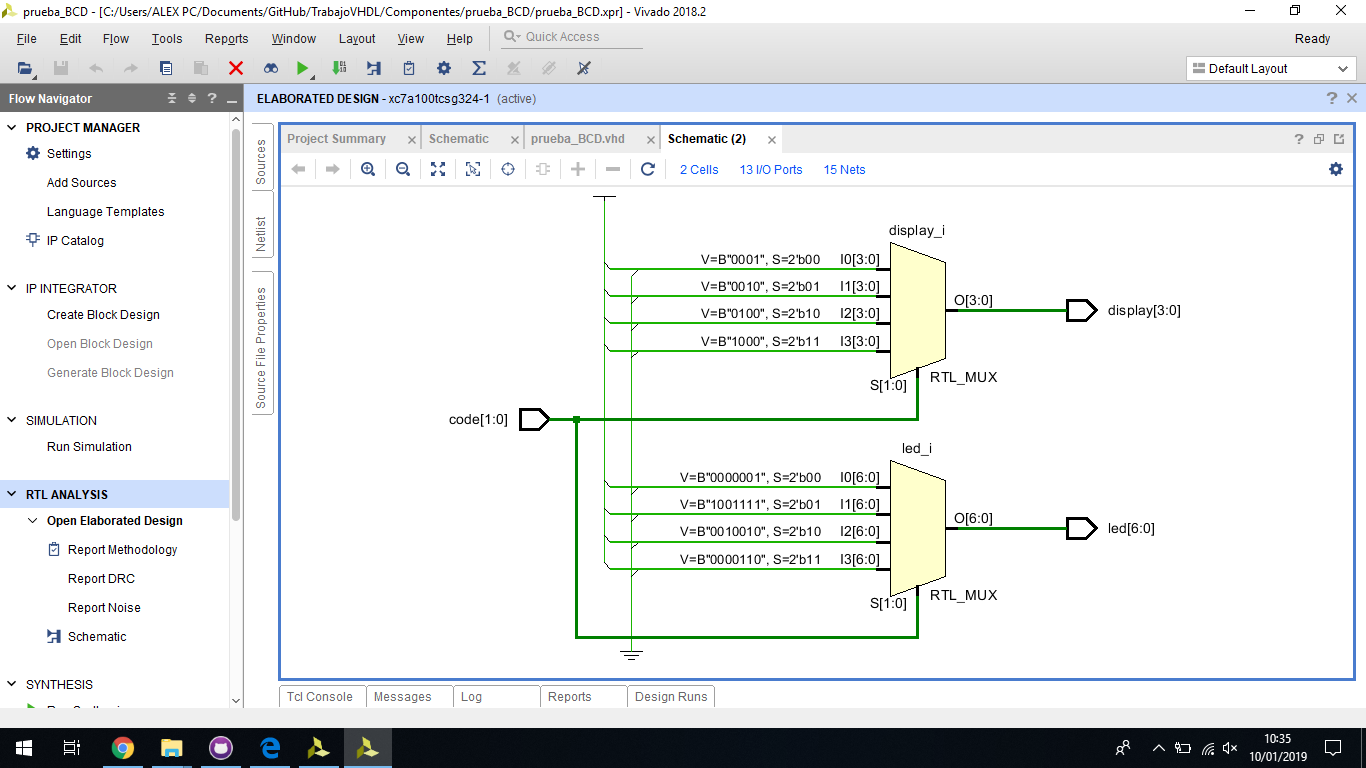
2) Para mostrar la salida de los actuadores (puerta y motor) decidimos utilizar los leds RGB por su sencillez a la hora de reconocer la información mostrada, en comparación con los leds simples utilizados en las prácticas.

# Diagramas



# Funcionamiento de los componentes

## Decodificador BCD a 7 segmentos

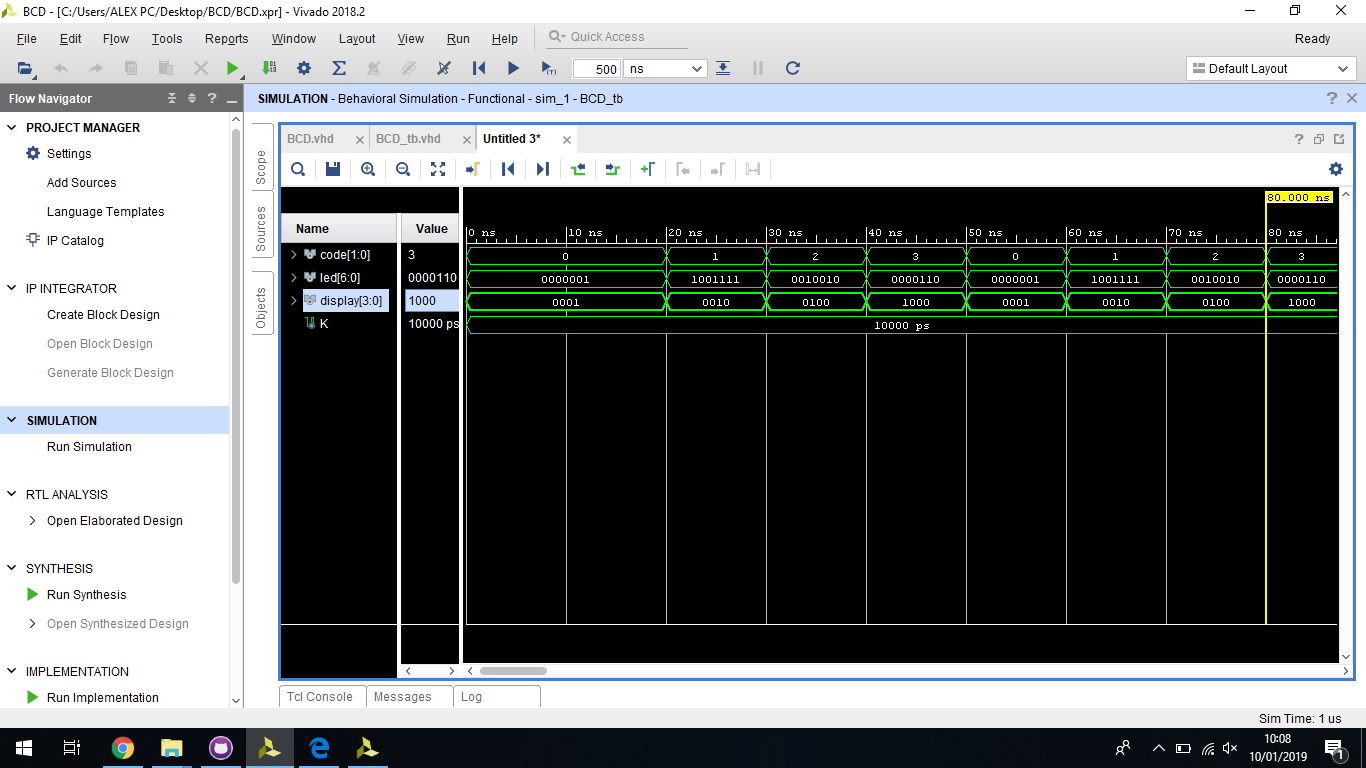


### Descripción

Un decodificador es un circuito combinacional que realiza la operación inversa a la de un codificador de datos, por eso también se puede definir un decodificador como circuito que convierte un código binario concreto en una forma sin codificar.

Los dígitos de la tarjeta son de tipo LED. Cada segmento es un LED de cuyos terminales es común para todos los segmentos. En nuestro caso se trata de un ánodo común, por lo que para que luzca un segmento debemos aplicar al terminal no común un ‘0’. La forma de mostrar números distintos en cada dígito consiste en irlos encendiendo en secuencia haciendo que las líneas de control de los segmentos reflejen el número correspondiente al dígito activo en ese momento.

El decodificador BCD de 7 segmentos lo utilizamos para mostrar en el display de la placa el número del piso en el que nos encontramos. Tenemos una entrada la cual recibe el piso en el que se encuentra el ascensor (code) y dos salidas, una para seleccionar cual, de los 8 dígitos de ambos display se enciende (display); y otra para mostrar el número en decimal del piso en el que se encuentra (led).



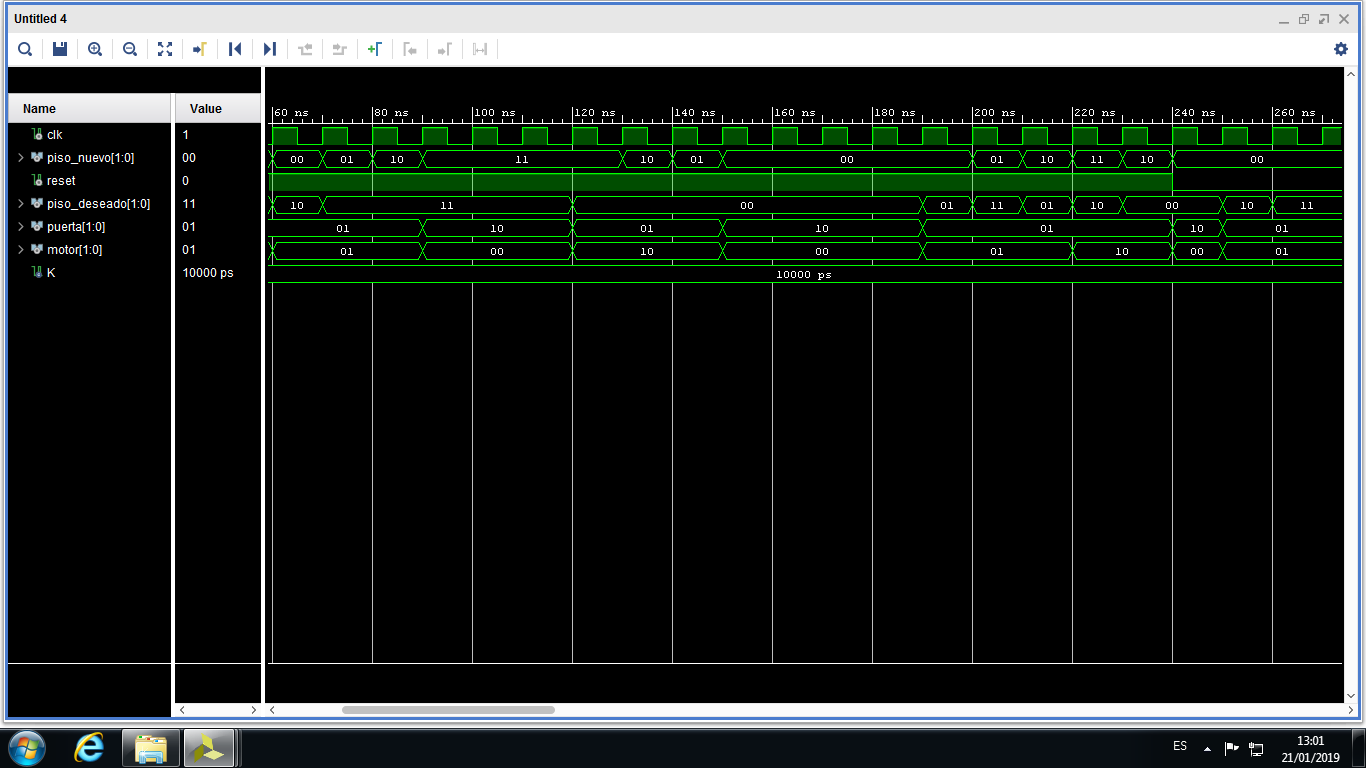
## Máquina de estados



### Descripción

Las máquinas de estado finito son circuitos secuenciales utilizados en muchos sistemas digitales para controlar el comportamiento de sistemas y las rutas de flujo de datos.

Es el componente más importante del ascensor, gracias al cual podemos determinar el piso del ascensor en el que nos encontramos, si la puerta está abierta o cerrada y si el ascensor sube o baja. Nuestro ascensor tiene 4 pisos por lo que tenemos 4 estados: S0 para el piso 0, S1 para el piso 1, S2 para el piso 2, S3 para el piso 3. Por otro lado, disponemos de una entrada la cual nos determina si se produce cambio de estado o se permanece en el mismo (piso\_deseado), una señal de reloj (clk) y un reset (reset). En cuanto a las salidas contamos con una que determina si la puerta está abierta o cerrada (puerta), otra que nos determina si el ascensor sube o baja de piso (motor), otra que nos indica si el ascensor se está movimiento o no (moviendo) y una última que nos determina el piso nuevo (piso\_nuevo).

Desde el punto de vista de la funcionalidad, en la máquina se produce un cambio de estado cuando pulsamos un botón de la placa, por ejemplo, para alcanzar el estado S1 mirando los contraints pulsaremos el botón correspondiente de la placa (pin M18). Mientras se produce la transición de estado la puerta está cerrada (adquiriendo el valor de ‘0’), en este caso el led (pin N16) estaría de color rojo; y en el caso de que no haya una transición en la máquina la puerta estará abierta (valor ‘1’), por lo que el led (pin R11) se encuentra de color rojo. Por último, cuando se produce una transición en la máquina de un estado a otro, la salida del motor adquiere un valor, de “01” si sube el ascensor (representado con el led correspondiente al pin M16, de color verde) y de “10” si baja el ascensor (representado con el mismo led correspondiente al pin R12, de color azul). Sin embargo, si no se produce ningún cambio de estado el motor está parado (“00”) y dicho led no se enciende.

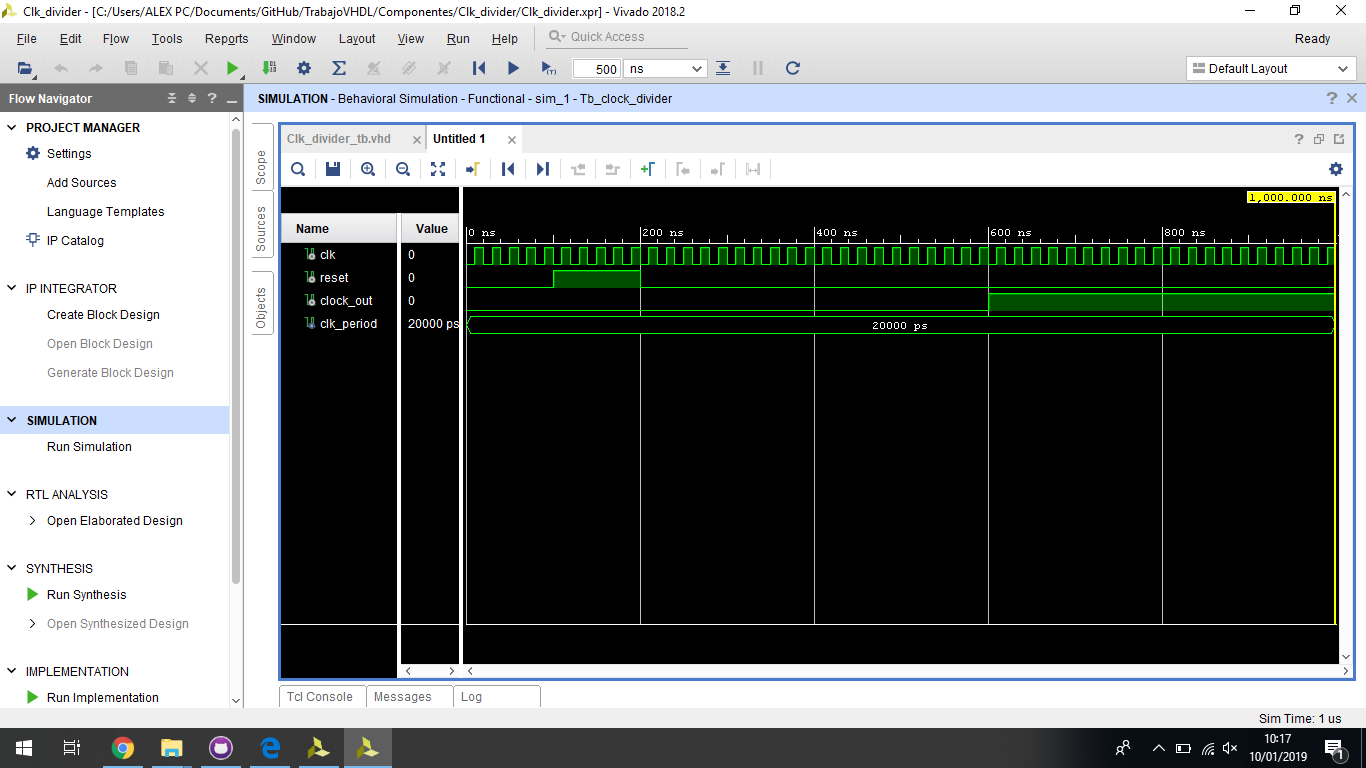
## Clk divider



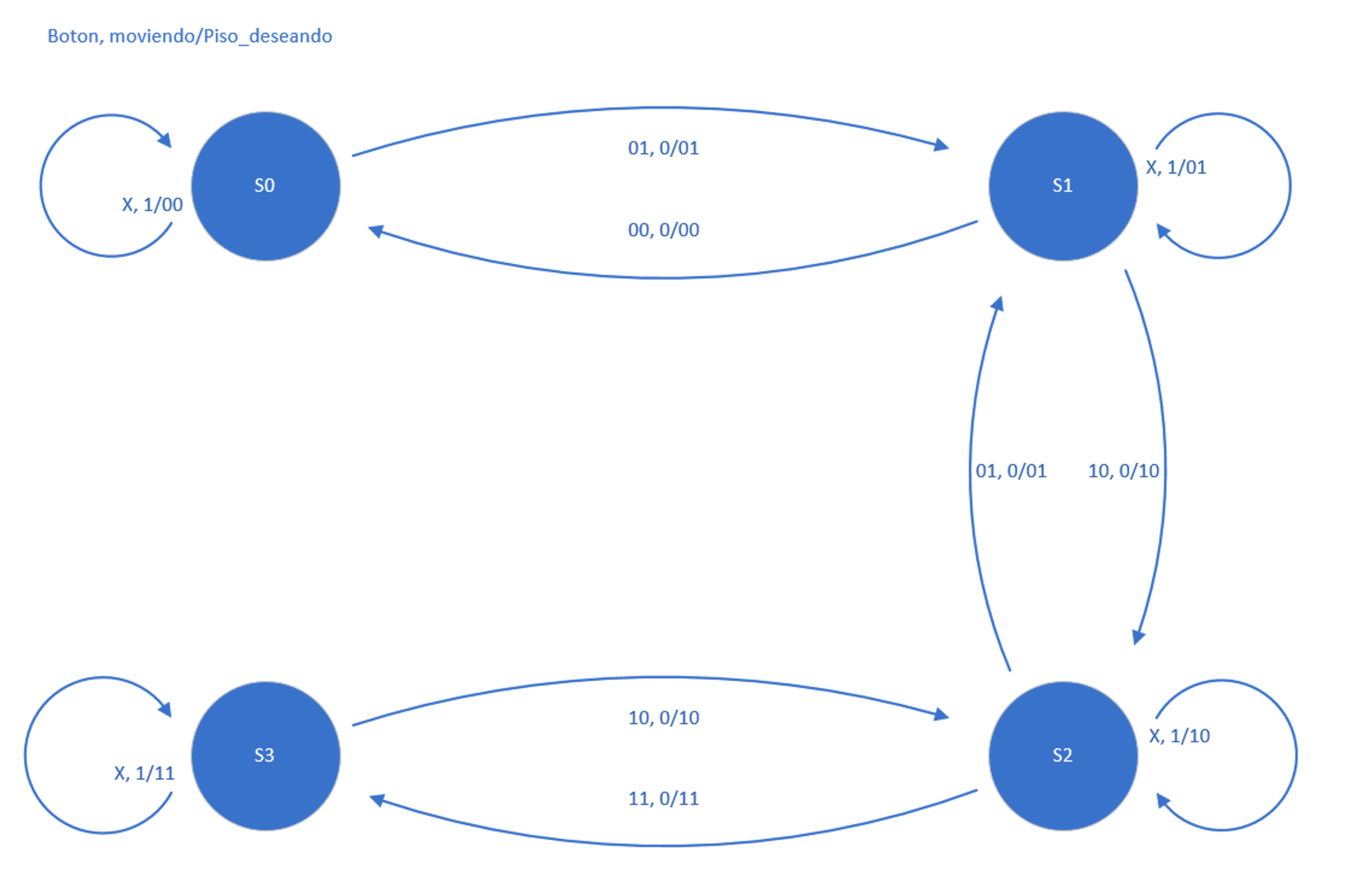
### Descripción

Con este componente creamos una señal de reloj distinta al reloj de la placa. El objetivo es reducir la frecuencia con la que se produce un pulso de reloj. Si miramos las especificaciones de la placa Nexys 4 DDR, observamos que su reloj interno tiene una frecuencia de hasta 100 MHz. Gracias a la implementación de este componente podemos ajustar la frecuencia a la que queremos que trabaje nuestro reloj.

Este módulo dispone de dos entradas, la propia señal de reloj de la placa (clk) y el reset (reset); y de una salida, la cual nos establece la frecuencia a la que queremos que trabaje nuestro reloj (clk\_out).

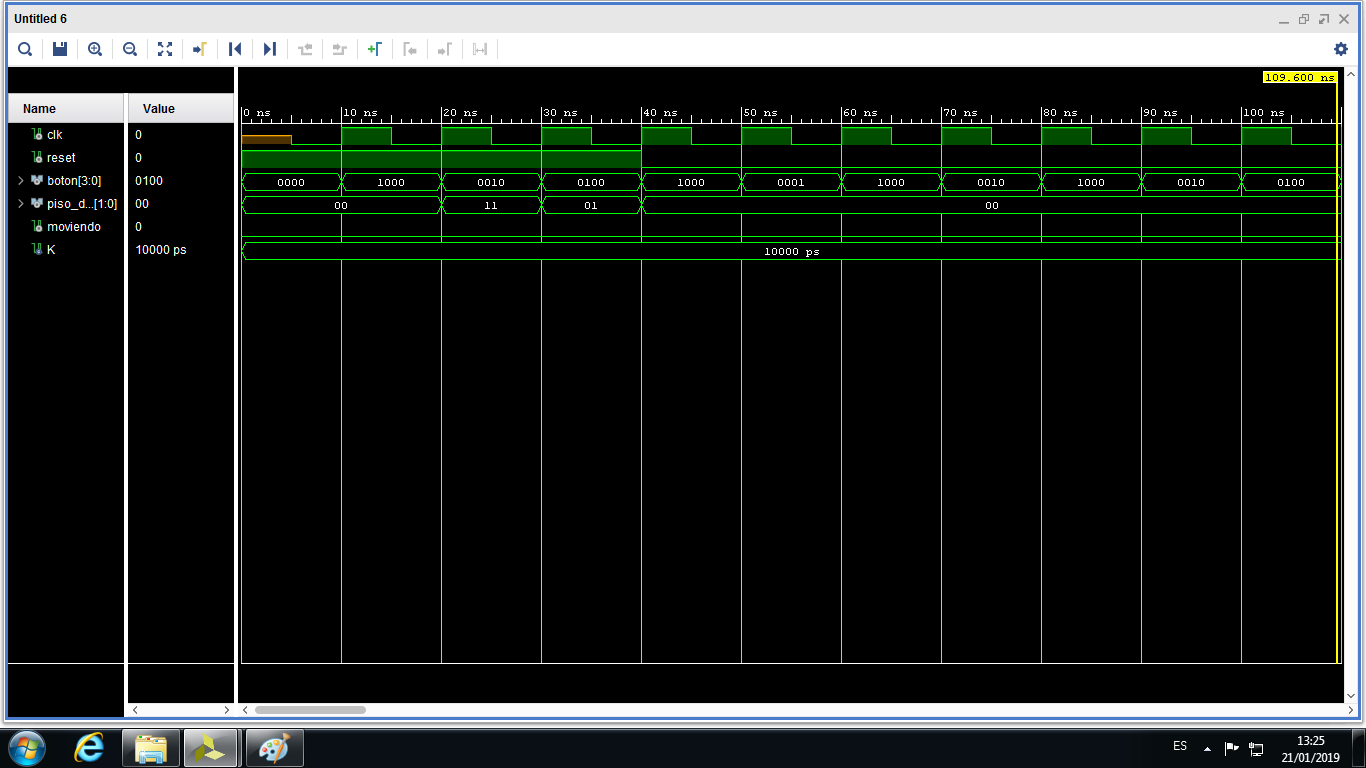


## Pulse register



### Descripción

Es una máquina de estado que utilizamos para registrar el piso al que queremos ir. Dispone de los mismos cuatro estados que la máquina principal. Tenemos una entrada la cual nos determina si se produce cambio de estado o se permanece en el mismo (boton), una señal de reloj (clk), un reset (reset) y otra que nos indica si el ascensor se está movimiento o no (moviendo). En cuanto a las salidas contamos con una que nos determina el piso al que se quiere ir (piso\_deseado).



# Anexo

## Código fuente y testbench

### Ascensor

#### Código fuente

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ascensor is

Port ( clk : in std\_logic;

reset : in std\_logic;

piso\_deseado : in std\_logic\_vector (3 downto 0);

puerta\_abierta : out std\_logic\_vector (1 downto 0);

display\_number : out std\_logic\_vector (6 downto 0);

display\_selection : out std\_logic\_vector (7 downto 0);

motor\_puerta : out std\_logic\_vector (1 downto 0)

);

end Ascensor;

architecture structural of Ascensor is

signal clk\_sinc : std\_logic;

signal piso\_actual : std\_logic\_vector(1 downto 0);

signal piso\_d : std\_logic\_vector (1 downto 0);

signal mov : std\_logic;

component Maquina\_Estado is

port ( clk : in std\_logic;

reset : in std\_logic;

piso\_deseado : in std\_logic\_vector (1 downto 0);

piso\_nuevo : out std\_logic\_vector (1 downto 0);

puerta: out std\_logic\_vector (1 downto 0);

motor: out std\_logic\_vector(1 downto 0);

moviendo: out std\_logic

);

end component;

component Pulse\_register is

port ( clk : in std\_logic;

reset : in std\_logic;

boton : in std\_logic\_vector(3 downto 0);

piso\_deseado: out std\_logic\_vector(1 downto 0);

moviendo : in std\_logic

);

end component;

component Clk\_divider is

port ( clk,reset: in std\_logic;

clk\_out: out std\_logic

);

end component;

component BCD is

port ( code : in std\_logic\_vector (1 downto 0);

led : out std\_logic\_vector (6 downto 0);

display: out std\_logic\_vector (7 downto 0)

);

end component;

begin

inst\_Maquina\_Estado: Maquina\_Estado port map(

clk => clk\_sinc,

reset => reset,

piso\_deseado => piso\_d,

piso\_nuevo => piso\_actual,

puerta => puerta\_abierta,

motor => motor\_puerta,

moviendo => mov

);

inst\_Pulse\_register: Pulse\_register port map(

clk => clk,

reset => reset,

boton => piso\_deseado,

piso\_deseado => piso\_d,

moviendo => mov

);

inst\_Clk\_divider: Clk\_divider port map(

clk => clk,

reset => reset,

clk\_out => clk\_sinc

);

inst\_BCD: BCD port map(

led => display\_number,

code => piso\_actual,

display => display\_selection

);

end;

#### Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Ascensor\_tb is

end Ascensor\_tb;

architecture Behavioral of Ascensor\_tb is

component Ascensor

Port (

clk : in std\_logic;

reset : in std\_logic;

piso\_deseado : in std\_logic\_vector (3 downto 0);

puerta\_abierta : out std\_logic\_vector (1 downto 0);

display\_number : out std\_logic\_vector (6 downto 0);

display\_selection : out std\_logic\_vector (7 downto 0);

motor\_puerta : out std\_logic\_vector (1 downto 0)

);

end component;

signal clk: std\_logic;

signal display\_number: std\_logic\_vector(6 downto 0);

signal display\_selection: std\_logic\_vector(7 downto 0);

signal reset : std\_logic := '1';

signal piso\_deseado : std\_logic\_vector(3 downto 0) := "0000";

signal puerta\_abierta: std\_logic\_vector (1 downto 0);

signal motor\_puerta: std\_logic\_vector(1 downto 0) := "00";

constant K: time := 10 ns;

begin

uut: Ascensor port map (

clk => clk,

reset => reset,

piso\_deseado => piso\_deseado,

display\_number => display\_number,

display\_selection => display\_selection,

puerta\_abierta => puerta\_abierta,

motor\_puerta => motor\_puerta

);

stim\_proc: process

begin

wait for 0.25 \* K;

clk <= '0';

wait for 0.25 \* K;

clk <= '1';

end process;

stim\_proc2: process

begin

wait for 4 \* K;

piso\_deseado <= "0010";

wait for 4 \* K;

piso\_deseado <= "1000";

wait for 4 \* K;

piso\_deseado <= "0100";

wait for 4 \* K;

piso\_deseado <= "0001";

wait for 4 \* K;

piso\_deseado <= "1000";

wait for 4 \* K;

piso\_deseado <= "0100";

wait for 4 \* K;

piso\_deseado <= "0001";

wait for 4 \* K;

piso\_deseado <= "0001";

wait for 4 \* K;

end process;

stim\_proc3: process

begin

wait until display\_number = "1001111";

wait until display\_number = "0000110";

wait until display\_number = "0010010";

wait until display\_number = "0000001";

wait until display\_number = "0000110";

wait until display\_number = "0000001";

assert false

report "Simulation finished."

severity failure;

end process;

end Behavioral;

### BCD

#### Código fuente

library IEEE;

use IEEE.std\_logic\_1164.all;

entity BCD is

port (

code : in std\_logic\_vector (1 downto 0);

led : out std\_logic\_vector (6 downto 0);

display: out std\_logic\_vector (7 downto 0)

);

end BCD;

architecture Behavioral of BCD is

begin

with code select

led <= "0000001" when "00",

"1001111" when "01",

"0010010" when "10",

"0000110" when "11",

"1111110" when others;

with code select

display <= "11111110" when "00",

"11111101" when "01",

"11111011" when "10",

"11110111" when "11",

"11111111" when others;

end Behavioral;

#### Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity BCD\_tb is

end BCD\_tb;

architecture Behavioral of BCD\_tb is

component BCD

port(

code : in std\_logic\_vector(1 downto 0);

led : out std\_logic\_vector(6 downto 0);

display : out std\_logic\_vector(3 downto 0)

);

end component;

signal code : std\_logic\_vector(1 downto 0):="00";

signal led : std\_logic\_vector(6 downto 0):="0000000";

signal display : std\_logic\_vector(3 downto 0):="0000";

constant K: time := 10 ns;

begin

uut: BCD port map(

code => code,

led => led,

display=> display

);

tb\_1:process

begin

wait for 1 \* K;

code <= "00";

wait for 1 \* K;

code <= "01";

wait for 1 \* K;

code <= "10";

wait for 1 \* K;

code <= "11";

end process;

stim\_proc3: process

begin

wait until display = "11";

assert false

report "Simulation finished."

severity failure;

end process;

end Behavioral;

### Pulse\_register

#### Código fuente

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Pulse\_register is

Port ( clk : in std\_logic;

reset : in std\_logic;

boton : in std\_logic\_vector(3 downto 0);

piso\_deseado: out std\_logic\_vector(1 downto 0);

moviendo : in std\_logic := '0'

);

end Pulse\_register;

architecture Behavioral of Pulse\_register is

type state\_type is (S0, S1, S2, S3);

signal state, next\_state : state\_type;

begin

SYNC\_PROC: process (clk)

begin

if rising\_edge(clk) then

if (reset = '0') then

state <= S0;

elsif moviendo = '0' then

state <= next\_state;

end if;

end if;

end process;

OUTPUT\_DECODE: process (state)

begin

case (state) is

when S0 => piso\_deseado <= "00";

when S1 => piso\_deseado <= "01";

when S2 => piso\_deseado <= "10";

when S3 => piso\_deseado <= "11";

when others => piso\_deseado <= "00";

end case;

end process;

NEXT\_STATE\_DECODE: process (state, boton)

begin

next\_state <= state;

case (state) is

when S0 =>

if(boton = "0001") then

next\_state <= S0;

elsif(boton = "0010") then

next\_state <= S1;

elsif(boton = "0100") then

next\_state <= S2;

elsif(boton = "1000") then

next\_state <= S3;

end if;

when S1 =>

if(boton = "0001") then

next\_state <= S0;

elsif(boton = "0010") then

next\_state <= S1;

elsif(boton = "0100") then

next\_state <= S2;

elsif(boton = "1000") then

next\_state <= S3;

end if;

when S2 =>

if(boton = "0001") then

next\_state <= S0;

elsif(boton = "0010") then

next\_state <= S1;

elsif(boton = "0100") then

next\_state <= S2;

elsif(boton = "1000") then

next\_state <= S3;

end if;

when S3 =>

if(boton = "0001") then

next\_state <= S0;

elsif(boton = "0010") then

next\_state <= S1;

elsif(boton = "0100") then

next\_state <= S2;

elsif(boton = "1000") then

next\_state <= S3;

end if;

end case;

end process;

end Behavioral;

#### Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Pulse\_register\_tb is

end Pulse\_register\_tb;

architecture Behavioral of Pulse\_register\_tb is

component Pulse\_register

port(

clk : in std\_logic;

reset : in std\_logic;

boton : in std\_logic\_vector(3 downto 0);

piso\_deseado: out std\_logic\_vector(1 downto 0);

moviendo : in std\_logic := '0'

);

end component;

signal clk : std\_logic;

signal reset : std\_logic := '1';

signal boton : std\_logic\_vector(3 downto 0) := "0000";

signal piso\_deseado: std\_logic\_vector(1 downto 0);

signal moviendo : std\_logic := '0';

constant K: time := 10 ns;

begin

uut: Pulse\_register port map (

clk => clk,

reset => reset,

boton => boton,

piso\_deseado => piso\_deseado,

moviendo => moviendo

);

stim\_proc: process

begin

wait for 0.5 \* K;

clk <= '0';

wait for 0.5 \* K;

clk <= '1';

end process;

stim\_proc2: process

begin

wait for 1 \* K;

boton <= "1000";

wait for 1 \* K;

boton <= "0010";

wait for 1 \* K;

boton <= "0100";

wait for 1 \* K;

boton <= "1000";

reset <= '0';

wait for 1 \* K;

boton <= "0001";

wait for 1 \* K;

boton <= "1000";

wait for 1 \* K;

boton <= "0010";

end process;

stim\_proc3: process

begin

wait until piso\_deseado = "00";

wait until piso\_deseado = "01";

wait until piso\_deseado = "10";

wait until piso\_deseado = "11";

wait until piso\_deseado = "00";

wait until piso\_deseado = "00";

wait until piso\_deseado = "01";

wait until piso\_deseado = "10";

wait until piso\_deseado = "11";

wait until piso\_deseado = "00";

assert false

report "Simulation finished."

severity failure;

end process;

end Behavioral;

### Máquina de estados

#### Código fuente

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Maquina\_Estado is

port ( clk : in std\_logic;

reset : in std\_logic;

piso\_deseado : in std\_logic\_vector (1 downto 0);

piso\_nuevo : out std\_logic\_vector (1 downto 0);

puerta: out std\_logic\_vector (1 downto 0);

motor: out std\_logic\_vector(1 downto 0);

moviendo: out std\_logic

);

end Maquina\_Estado;

architecture Behavioral of Maquina\_Estado is

type state\_type is (S0, S1, S2, S3);

signal state, next\_state : state\_type;

signal temp : std\_logic\_vector (1 downto 0);

begin

SYNC\_PROC: process (clk, reset)

begin

if (reset = '0') then

state <= S0;

elsif rising\_edge(clk) then

state <= next\_state;

end if;

end process;

OUTPUT\_DECODE: process (state)

begin

case (state) is

when S0 => piso\_nuevo <= "00";

when S1 => piso\_nuevo <= "01";

when S2 => piso\_nuevo <= "10";

when S3 => piso\_nuevo <= "11";

when others => piso\_nuevo <= "00";

end case;

end process;

NEXT\_STATE\_DECODE: process (state, piso\_deseado)

begin

next\_state <= S0;

case (state) is

when S0 =>

if (piso\_deseado = "00") then

next\_state <= S0;

puerta<="10";

motor<="00";

moviendo<='0';

elsif (piso\_deseado /= "00") then

puerta<="01";

motor<="01";

next\_state <= S1;

temp <= piso\_deseado;

moviendo<='1';

end if;

when S1 =>

if (temp /= "01") then

if(temp = "10") or (temp = "11") then

next\_state <= S2;

motor<="01";

elsif(temp = "00") then

next\_state <= S0;

motor<="10";

end if;

puerta<="01";

moviendo<='1';

else

if (piso\_deseado = "00") then

next\_state <= S0;

puerta<="01";

motor<="10";

temp <= "00";

moviendo<='1';

elsif (piso\_deseado = "01") then

next\_state <= S1;

puerta<="10";

motor<="00";

temp <= "01";

moviendo<='0';

elsif (piso\_deseado = "10") then

next\_state <= S2;

puerta<="01";

motor<="01";

temp <= "10";

moviendo<='1';

elsif (piso\_deseado = "11") then

next\_state <= S2;

temp <= piso\_deseado;

puerta<="01";

motor<="01";

moviendo<='1';

end if;

end if;

when S2 =>

if (temp /= "10") then

if(temp = "00") or (temp = "01") then

next\_state <= S1;

motor<="10";

elsif(temp = "11") then

next\_state <= S3;

motor<="01";

end if;

puerta<="01";

moviendo<='1';

else

if (piso\_deseado = "00") then

next\_state <= S1;

temp <= piso\_deseado;

puerta<="01";

motor<="10";

moviendo<='1';

elsif (piso\_deseado = "01") then

next\_state <= S1;

puerta<="01";

motor<="10";

temp <= "01";

moviendo<='1';

elsif (piso\_deseado = "10") then

next\_state <= S2;

puerta<="10";

motor<="00";

temp <= "10";

moviendo<='0';

elsif (piso\_deseado = "11") then

next\_state <= S3;

puerta<="01";

motor<="01";

temp <= "11";

moviendo<='1';

end if;

end if;

when S3 =>

if (piso\_deseado = "11") then

next\_state <= S3;

puerta<="10";

motor<="00";

moviendo<='0';

elsif (piso\_deseado /= "11") then

puerta<="01";

motor<="10";

next\_state <= S2;

temp <= piso\_deseado;

moviendo<='1';

end if;

end case;

end process;

end Behavioral;

#### Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Maquina\_Estado\_tb is

end Maquina\_Estado\_tb;

architecture Behavioral of Maquina\_Estado\_tb is

component Maquina\_Estado

port(

clk : in std\_logic;

reset : in std\_logic;

piso\_deseado : in std\_logic\_vector (1 downto 0);

piso\_nuevo : out std\_logic\_vector (1 downto 0);

puerta: out std\_logic\_vector (1 downto 0);

motor: out std\_logic\_vector(1 downto 0);

moviendo: out std\_logic

);

end component;

signal clk: std\_logic;

signal piso\_nuevo: std\_logic\_vector(1 downto 0);

signal reset : std\_logic := '1';

signal piso\_deseado : std\_logic\_vector(1 downto 0) := "00";

signal puerta: std\_logic\_vector (1 downto 0);

signal motor: std\_logic\_vector(1 downto 0) := "00";

constant K: time := 1000000000ns;

begin

uut: Maquina\_Estado port map (

clk => clk,

reset => reset,

piso\_deseado => piso\_deseado,

piso\_nuevo => piso\_nuevo,

puerta => puerta,

motor => motor

);

stim\_proc: process

begin

wait for 0.5 \* K;

clk <= '0';

wait for 0.5 \* K;

clk <= '1';

end process;

stim\_proc2: process

begin

wait for 1 \* K;

piso\_deseado <= "11";

wait for 1 \* K;

piso\_deseado <= "01";

wait for 1 \* K;

piso\_deseado <= "10";

wait for 1 \* K;

piso\_deseado <= "00";

wait for 2 \* K;

piso\_deseado <= "10";

wait for 1 \* K;

piso\_deseado <= "11";

wait for 5 \* K;

piso\_deseado <= "00";

wait for 7 \* K;

piso\_deseado <= "01";

end process;

stim\_proc3: process

begin

wait until piso\_nuevo = "10";

wait until piso\_nuevo = "00";

wait until piso\_nuevo = "11";

wait until piso\_nuevo = "00";

wait until piso\_nuevo = "11";

wait until piso\_nuevo = "01";

reset <= '0';

wait until piso\_nuevo = "11";

wait until piso\_nuevo = "10";

wait until piso\_nuevo = "00";

wait until piso\_nuevo = "11";

assert false

report "Simulation finished."

severity failure;

end process;

end Behavioral;

### Clk\_divider

#### Código fuente

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.ALL;

entity Clk\_divider is

port (

clk,reset: in std\_logic;

clk\_out: out std\_logic

);

end Clk\_divider;

architecture behavioral of Clk\_divider is

signal count: integer:=1;

signal tmp : std\_logic := '0';

begin

process(clk,reset)

begin

if(reset='0') then

count<=1;

tmp<='0';

elsif(clk'event and clk='1') then

count <=count+1;

if (count = 200000000) then

tmp <= NOT tmp;

count <= 1;

end if;

end if;

clk\_out <= tmp;

end process;

end behavioral;

#### Testbench

library ieee;

use ieee.std\_logic\_1164.all;

entity Tb\_clock\_divider is

end Tb\_clock\_divider;

architecture behavioral of Tb\_clock\_divider is

component Clock\_Divider

port(

clk : in std\_logic;

reset : in std\_logic;

clock\_out : out std\_logic

);

end component;

signal clk : std\_logic := '0';

signal reset : std\_logic := '0';

signal clock\_out : std\_logic;

constant clk\_period : time := 20 ns;

begin

uut: Clock\_Divider port map (

clk => clk,

reset => reset,

clock\_out => clock\_out

);

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

stim\_proc: process

begin

wait for 100 ns;

reset <= '1';

wait for 100 ns;

reset <= '0';

wait;

end process;

END;