Laboratorio de Sistemas Electrónicos Digitales

Práctica 1

INTRODUCCIÓN A LA PLACA NEXUS 4 DDR Y PRIMEROS EJEMPLOS

Vega Ochoa, Lydia (52654)

Perea Vanguelov, Francisco Javier (52545)

Martínez Trapiello, Alberto (52713)

A309

Grupo B

Lunes 11:30 – 13:30

Grado en Ingeniería Electrónica Industrial y Automática

Curso 2018/2019

16 de octubre de 2018

**1. Tareas**

Comenzamos esta práctica continuando la primera con la creación de entidades de nivel superior, las cuales nos servirán para aproximar más el código al de un componente a medida. De esta manera la interfaz superior hace las veces de interfaz para su uso más cómodo por parte de potenciales clientes.

En esta práctica usaremos los displays de 7 segmentos con los que representaremos números. Cada display necesitaría de 7 pines, por lo que para representar los 4 dígitos se necesitaría una cantidad indecente de pines, por lo que para optimizar se usan los mismos pines de entrada para alimentar a los 4 displays. Esto implica que cada vez que le pasemos un número a un display este saldrá en todos aquellos que tengan activada la entrada “enable”, por lo que se recurre a la multiplexación para representar los dígitos de modo que a una frecuencia mayor a la cual puede detectar cambios el ojo humano, se van mostrando un número en cada display de manera consecutiva quedando a simple vista como que están encendidos todos a la vez.

Para poder representar los números requerimos de un decodificador para poder pasar de los valores de entrada (número binario representado con interruptores) a los valores de salida (los correspondientes LEDS que se han de encender para cada uno de los casos según el número a representar. Una vez implementado creamos un testbench que nos arroja los resultados de la simulación.

*Tarea 1*



Se puede comprobar que la respuesta se corresponde con la del testbench y que para valores mayores que 9 toma un valor de "1111110" equivalente al 7e que se ve en la imagen, así como que para el 8 están todos los leds a cero (que ilumina todas las luces ya que son activos a nivel bajo).

*Tarea 2*

Para implementar la entidad superior simplemente conectamos las variables de entrada con las de nuestro decoder (componente creado a partir de la entidad del decodificador).

----------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

---------------------------------------------------------------------------------

entity top is

PORT (

code : IN std\_logic\_vector(3 DOWNTO 0);

digsel : IN std\_logic\_vector(3 DOWNTO 0);

digctrl : OUT std\_logic\_vector(3 DOWNTO 0);

segment : OUT std\_logic\_vector(6 DOWNTO 0));

end top;

---------------------------------------------------------------------------------

architecture arch\_top of top is

COMPONENT decoder

PORT (

code : IN std\_logic\_vector(3 DOWNTO 0);

led : OUT std\_logic\_vector(6 DOWNTO 0));

END COMPONENT;

begin

Inst\_decoder: decoder PORT MAP (

code => code,

led => segment

);

digctrl<= not digsel;

end arch\_top;

Finalmente modificamos las contrains para poder cargarlas a la placa y poder probar su funcionamiento en la FPGA.

Primero preparamos los Switches como la entrada de nuestro código, el cual será descodificado y pasado a los leds para poder dibujar un número del 0 al 9 (el código puede llegar a 15 pero dado que no se pueden representar en el display se descartan estos valores dando como resultado un valor erróneo).

set\_property -dict {PACKAGE\_PIN J15 IOSTANDARD LVCMOS33} [get\_ports {code[0]}]

set\_property -dict {PACKAGE\_PIN L16 IOSTANDARD LVCMOS33} [get\_ports {code[1]}]

set\_property -dict {PACKAGE\_PIN M13 IOSTANDARD LVCMOS33} [get\_ports {code[2]}]

set\_property -dict {PACKAGE\_PIN R15 IOSTANDARD LVCMOS33} [get\_ports {code[3]}]

A continuación, preparamos el resto de Switches para seleccionar el display que va a representar el número.

set\_property -dict {PACKAGE\_PIN H6 IOSTANDARD LVCMOS33} [get\_ports {digsel[0]}]

set\_property -dict {PACKAGE\_PIN U12 IOSTANDARD LVCMOS33} [get\_ports {digsel[1]}]

set\_property -dict {PACKAGE\_PIN U11 IOSTANDARD LVCMOS33} [get\_ports {digsel[2]}]

set\_property -dict {PACKAGE\_PIN V10 IOSTANDARD LVCMOS33} [get\_ports {digsel[3]}]

Por último preparamos los displays, para lo cual conectamos la salida del decodificador a las entradas correctas de los LEDs de los displays:

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { segment[0]}]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { segment[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { segment[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { segment[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { segment[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { segment[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { segment[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

Por último tenemos que conectar los “enable” de los displays a nuestras entradas de control:

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { digctrl[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { digctrl[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { digctrl[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { digctrl[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

*Tarea 3*

Pues tenemos en el esquema la señal de entrada “botón” que es una señal asíncrona de un bit, la cual tras pasar por el sincronizador se volverá una señal síncrona “botón\_sinc”. Posteriormente se toman varios valores dentro del debouncer para sacar únicamente el valor en régimen permanente y este se le pasa al contador.

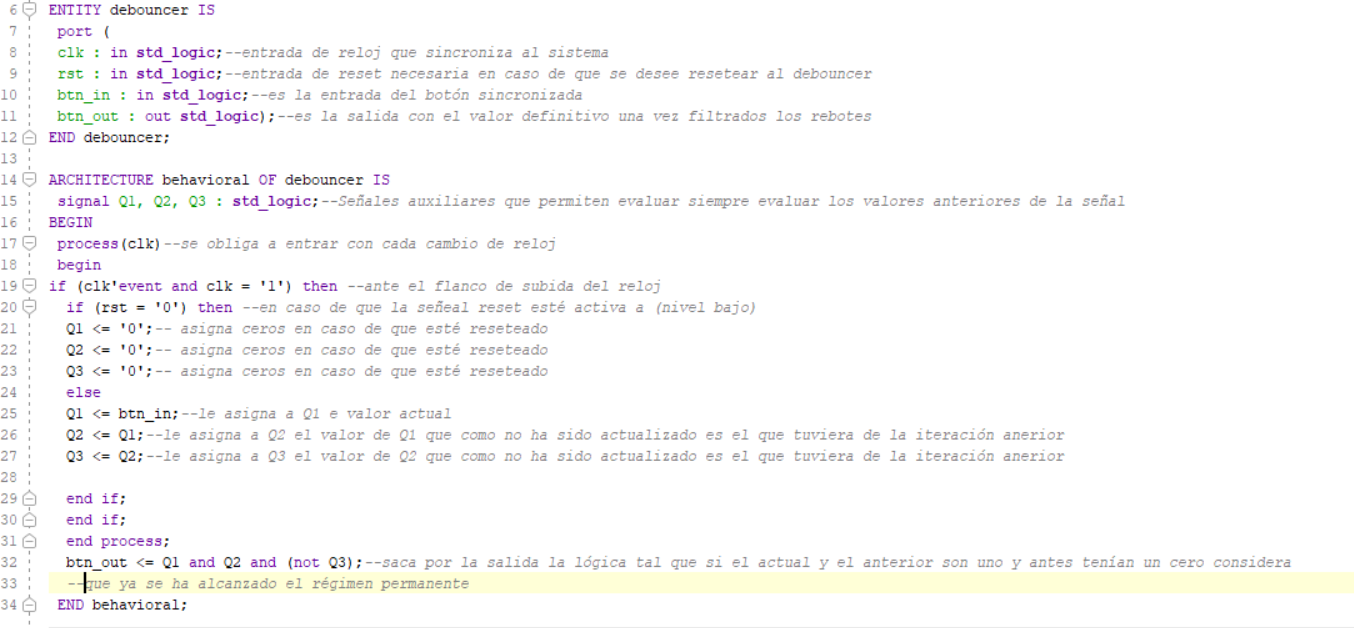
Este contador tiene a la salida un código “code”, que se corresponde con el número en binario a representar. Por lo que es una señal de 4 bits, la cual se pasa por el decodificador que saca en una señal de 7 bits los leds que se deben encender en la el display que se haya seleccionado.

Por último, digsel y digctr son señales de cuatro bits opuestas, que eligen el display a encender.

*Tarea 4*

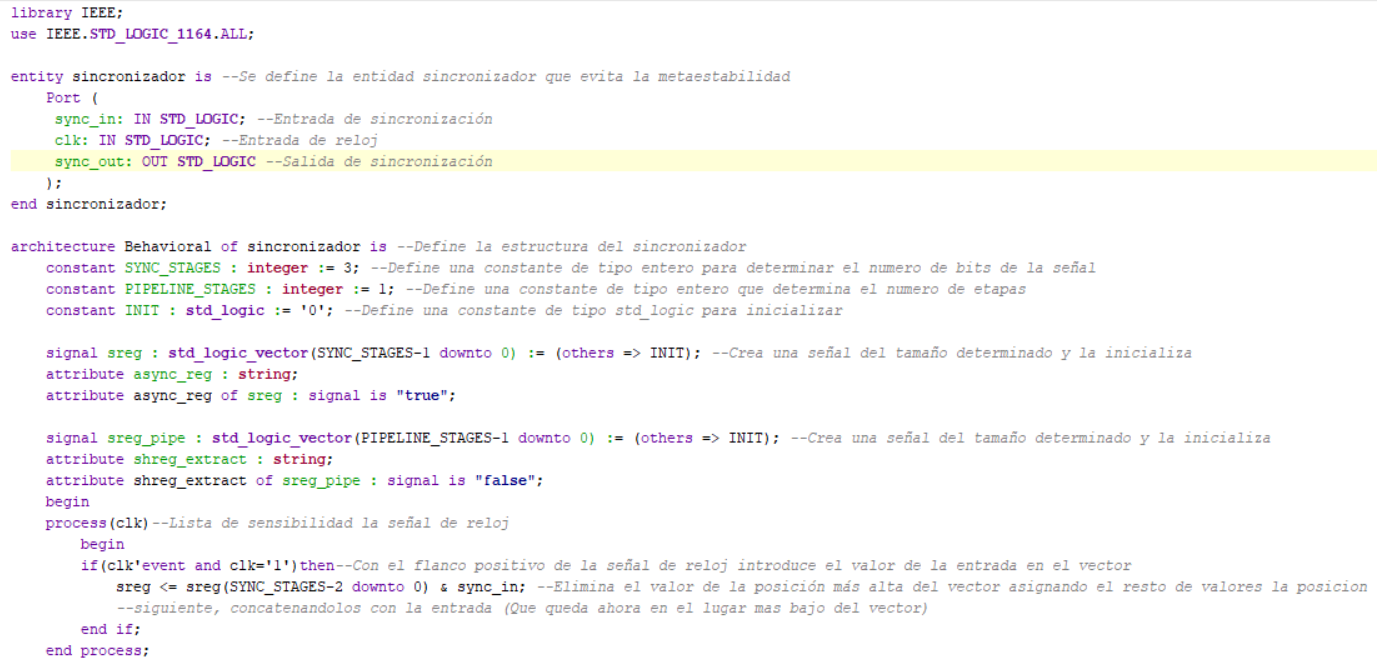
Para esta parte modificamos el código para añadir un botón que permita aumentar los números representados tras cada pulsación. Esto plantea un problema clásico de casi todos los micros, FPGAs, etc y es el efecto rebote. Este se da cuando ante la pulsación de un botón la señal tarda en alcanzar el régimen permanente y por tanto si se lee antes de que llegue éste puede ocurrir que la señal leída sea falsa, por lo que no es fiable. Además, toda señal del exterior se debe sincronizar debido a que la placa se rige según los pulsos del reloj integrado.

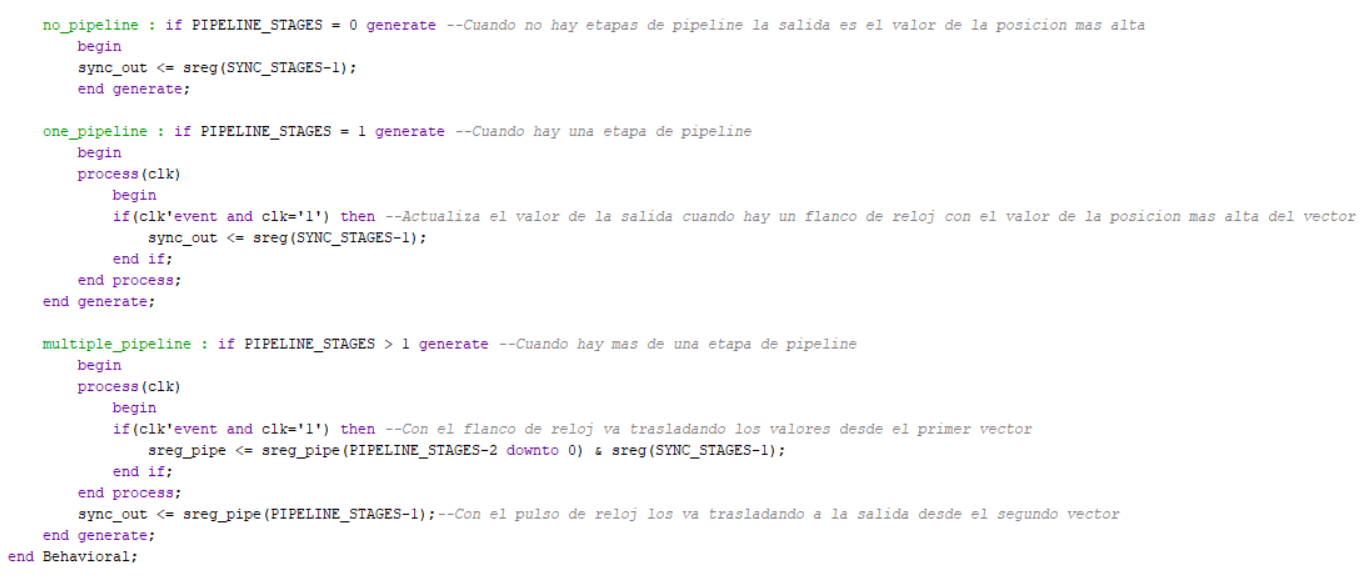
Para solventar este problema se plantea inicialmente planteamos un sistema “debouncer” se establecen tres señales que se van pasando el dato de entrada de modo que tiene en cuenta los valores anteriores de la señal de entrada para comparar y que a la salida sólo tome el caso en el cual la señal es seguro que se ha establecido.



*Tarea 5*

Posteriormente proseguimos con el sincronizador, el cual se encargará de que la señal asíncrona que llega del exterior entre dentro de nuestro circuito como señal síncrona. Para ello se sirve de un vector al cual va actualizando desplazando los bits y finalmente lo saca en la salida que se conectará a nuestro dispositivo. En función del número de PIPELINES\_STAGES realizará distinto código tal como está descrito mediante comentarios en el siguiente código:





*Tarea 6 y 7*

Generamos el código correspondiente al contador que junto con el debouncer y el sincronizador podemos hacer que los displays vayan representando números ascendentes con cada vez que se pulse el botón.

-----------------------------------------------------------------------------------------------

Entity contador is

port ( rst: in std\_logic;

boton: in std\_logic;

code: out std\_logic\_vector (3 downto 0));

end entity;

Architecture cont of contador is

signal c: signed (code'range);

begin

process ( boton, rst)

begin

if (rst = '0') then

code <= (others=>'0');

elsif (boton'event and boton ='1') and(c < 10) then

c <= c + 1;

end if;

end process;

code<=std\_logic\_vector (c);

end architecture;

Este contador se une con los otros bloques desarrollados en una última entidad top que se desarrolla a continuación, contemplando desde el principio la necesidad de una señal de reset:

-----------------------------------------------------------------------------------

entity top2 is

port( button, rst, clk: in std\_logic;

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

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

digctrl: out std\_logic\_vector (3 downto 0));

end entity;

Architecture top2 of top2 is

component sincronizador

Port (

sync\_in: IN STD\_LOGIC;

clk: IN STD\_LOGIC;

sync\_out: OUT STD\_LOGIC

);

end component;

component debouncer

port (

clk : in std\_logic;

rst : in std\_logic;

btn\_in : in std\_logic;

btn\_out : out std\_logic);

END component;

component contador

port ( rst: in std\_logic;

boton: in std\_logic;

code: out std\_logic\_vector (3 downto 0));

end component;

component decoder

PORT (

code : IN std\_logic\_vector(3 DOWNTO 0);

led: OUT std\_logic\_vector(6 DOWNTO 0)

);

end component;

signal boton\_sinc, boton\_deb: std\_logic;

signal code: std\_logic\_vector (3 downto 0);

begin

sinc: sincronizador port map (button, clk, boton\_sinc);

antireb: debouncer port map (clk, rst, boton\_sinc, boton\_deb);

cont: contador port map (rst, boton\_deb, code);

dec: decoder port map (code, segment);

digctrl <= not digsel;

end architecture;