PRÁCTICA 2

Diseño

Para el proceso de diseño me he basado en la FSM de ejemplo que se dá en las diapositivas del tema 4. La estructura del diseño está formada por dos procesos, uno secuencial y otro combinacional.

El registro tiene 5 entradas (reloj, reset, enable, dato y control) y 1 única salida. En la arquitectura, se han definido dos señales de tipo std logic vector para representar el estado en el que estamos. Estas señales de estado, le entrada del dato y la salida tienen un rango variable que depende de un parámetro genérico n.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Reg_Uni is
    generic( n : integer );
    Port ( clk, reset, CKE : in  STD_LOGIC;
           control         : in  STD_LOGIC_VECTOR(2 downto 0);
           dato            : in  STD_LOGIC_VECTOR(n-1 downto 0);
           salida          : out STD_LOGIC_VECTOR(n-1 downto 0));
end Reg_Uni;

architecture Behavioral of Reg_Uni is

--000 Carga el dato de entrada
--001 Contador ascendente
--010 Contador descendente
--011 Desplaza a la izquierda
--100 Desplaza a la derecha
--101 a 111 Conserva el dato

signal Estado_Actual, Proximo_Estado  : std_logic_vector(n-1 downto 0) := (others => '0');
```

El proceso combinacional es el que controla las operaciones que se deben hacer con estado próximo y estado actual para que a la salida se refleje la operación que se ha seleccionado por el control.

En la lista de sensibilidad del proceso se ha incluido la señal dato, control y estado actual. Ya que se debe recalcular la salida y el próximo estado cada vez que una de estas señales cambien.

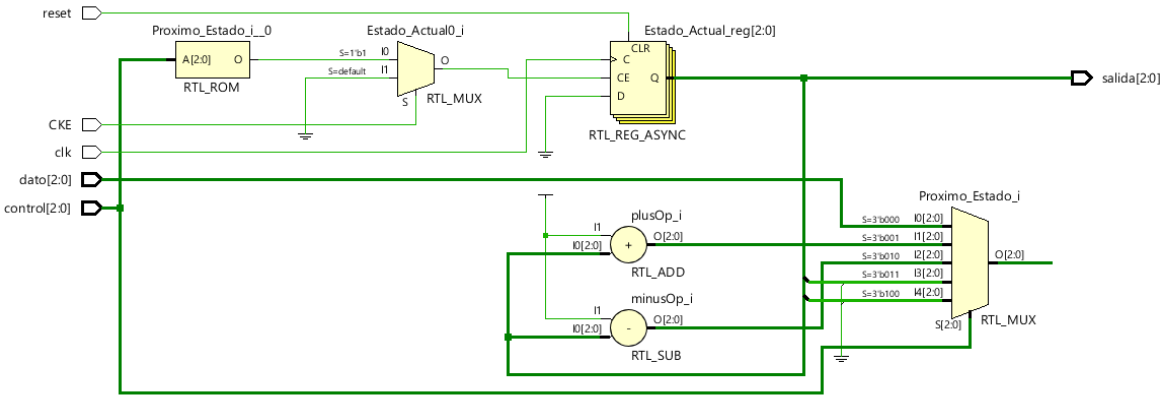
Se he hecho uso de los castings en las operaciones de suma y resta, para hacerlo directamente con la señal estado actual. Para los desplazamientos se ha asignado la parte de estado actual que corresponde a próximo estado y se ha añadido un cero por la izquierda o por la derecha, según el desplazamiento que corresponda.

```
Combinacional: process(control, dato, Estado_Actual)
    begin

        case control is
        when "000" =>
            Proximo_Estado <= dato;
        when "001" =>
            Proximo_Estado <= std_logic_vector( unsigned(Estado_Actual) + 1 );
        when "010" =>
            Proximo_Estado <= std_logic_vector( unsigned(Estado_Actual) - 1 );
        when "011" =>
            Proximo_Estado(n-1 downto 1) <= Estado_Actual(n-2 downto 0);
            Proximo_Estado(0)            <= '0';
        when "100" =>
            Proximo_Estado(n-2 downto 0) <= Estado_Actual(n-1 downto 1);
            Proximo_Estado(n-1)          <= '0';
        when others =>
            Proximo_Estado <= Estado_Actual;
        end case;

        salida <= Estado_Actual;

end process Combinacional;
```

Para el proceso secuencial se ha seguido el modelo de la FSM del tema 4. La lista de sensibilidad incluye el reloj y el reset. Cuando reset está a 1 el estado actual pasa a valer 0. Cuando reset no sea 1, es cuando la máquina de estados se actualiza cuando enable está a 1 y hay un flanco de subida del reloj.

```
        Secuencial: process(clk, reset)
            begin
                if reset='1' then
                    Estado_Actual <= (others => '0');
                elsif cke = '1' then
                    if rising_edge(clk) then
                        Estado_Actual <= Proximo_Estado;
                    end if;
                end if;
            end process Secuencial;

        end Behavioral;
```

Análisis RTL



En el esquemático podemos ver que el único elemento de memoria presente es el que representa el estado actual de la FSM.

Test bench

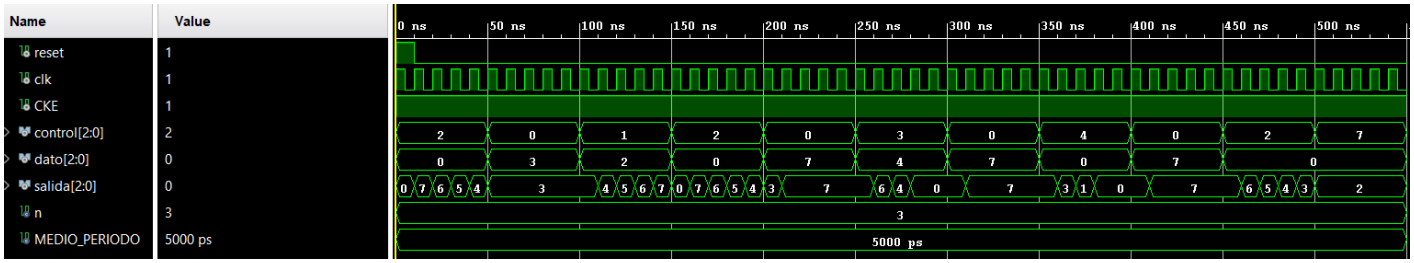
El test bench se ha hecho con manejo de ficheros igual que en los ejercicios con los estímulos en el .txt y el .csv que se ha creado a raíz de estos estímulos,

```
#Estímulos para Reg_Uni_Sinc.
#
# Delay Time (ns)       Input (dato, control).

    50 ns               000010
    50 ns               011000
    50 ns               010001
    50 ns               000010
    50 ns               111000
    50 ns               100011
    50 ns               111000
    50 ns               000100
    50 ns               111000
    50 ns               000010
```

Entrada,Enable
50 ns,000010
50 ns,011000
50 ns,010001
50 ns,000010
50 ns,111000
50 ns,100011
50 ns,111000
50 ns,000100
50 ns,111000
50 ns,000010
50 ns,000111

Simulación

Con los estímulos del apartado anterior hemos obtenido el siguiente resultado.



En esta captura podemos ver la simulación al completo, vamos a analizarla por tramos de tiempo.

0-50ns: Desde el principio vemos reset a 0, por lo que la salida es 0. En cuanto reset deja de valer 1, como control está a 2, el registro funciona como contador descendente. Cuanta de 0 a 7, 6, 5…

50ns-100ns: Control vale 0 y dato vale 3, por lo que en el registro se carga un 3, en este caso no se aprecia demasiado bien porque la salida ya era 3 por el contador ascendente del tramo anterior.

100ns-200ns: En este tramo control empieza valiendo 1, por lo que el registro vemos que se comporta como un contador ascendente y luego pasa a 2 y empieza a comportarse como contador descendente.

200ns-300ns: En este tramos empezamos cargando un 7 con control a 0 y luego hacemos desplazamientos hacia la izquierda. De 7 (111) pasamos a 6 (110) a 4 (100) y por último 0 (000).

300ns-400ns: Volvemos a cargar un 7 en el registro pero, esta vez, hacemos desplazamientos a la derecha. De 7 (111) pasamos a 3 (011) y de 3 a 1(001) y a 0 (000).

400ns-550ns: En este tramo final he vuelto a cargar un 7 y contamos hacia atrás para cambiar el valor de salida. En 500ns ponemos control a 7 (111) para que se conserve el valor.

Implementación

Para la implementación se ha generado una nueva fuente que incluya el registro universal y el sincronizador como componentes. El sincronizador solo se usará para el clock enable. No se usará para el reset ya que debe ser asíncrono.

```
end Reg_Uni_Sinc;

architecture Behavioral of Reg_Uni_Sinc is

component Sincronizador
        generic(n_sin : integer);
        port ( I,reset,clk : in std_logic;
               CKE : out std_logic);
end component Sincronizador;

component Reg_Uni
        generic( n : integer);
        port( reset,clk,cke : in std_logic;
              control         : in  STD_LOGIC_VECTOR(2 downto 0);
              dato            : in  STD_LOGIC_VECTOR(n-1 downto 0);
              salida          : out STD_LOGIC_VECTOR(n-1 downto 0));
end component Reg_Uni;

 signal s1 : std_logic := '0';

 begin

    Sincro: Sincronizador
        generic map ( n_sin )
        port map (I => I, reset => reset, clk => clk, CKE => s1);

    Registro: Reg_Uni
        generic map ( n )
        port map (control => control, dato => dato, salida => salida, reset => reset, clk => clk, cke => s1);

end Behavioral;
```

También se ha usado la fuente de constraints para asociar señales con elementos de la placa. Se ha modificado el archivo ZYBO_MASTER.xdc. Se descomenta el reloj, dos botones, 3 switches y las conexiones necesarias del Pmod JC. Necesitamos 3 conexiones del pmod, una para cada componente de dato.

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { control[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { control[1] }];  #IO_L24P_T3_34 Sch=SW1
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { control[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports I]; #IO_L20N_T3_34 Sch=BTN0
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports reset]; #IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3
```

```
##Pmod Header JC
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { dato[0] }];
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports { dato[1] }];
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { dato[2] }];
#set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { q[2] }]; #I
#set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { jc_p[2] }];
#set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { jc_n[2] }];
#set_property -dict { PACKAGE_PIN T12   IOSTANDARD LVCMOS33 } [get_ports { jc_p[3] }];
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { jc_n[3] }];
```