Diseño Electrónico. Ingeniería en Tecnologías de la Telecomunicación

Manuel Montoya Catalá

Marcos Baptista Ríos

Curso 3ºA

Grupo A1

**Índice:**

**1-Implementación del proyecto en general:**

**2- Funcionamiento del circuito anti rebote:**

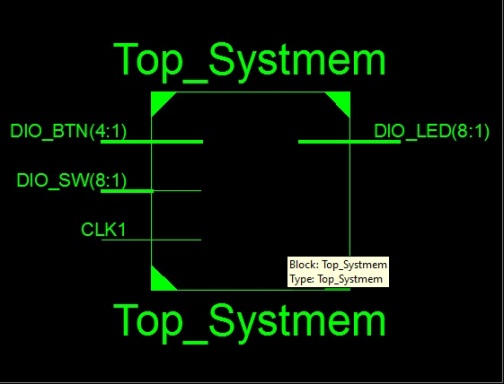
**3- Funcionamiento de la máquina de estados FSM\_Lapso**

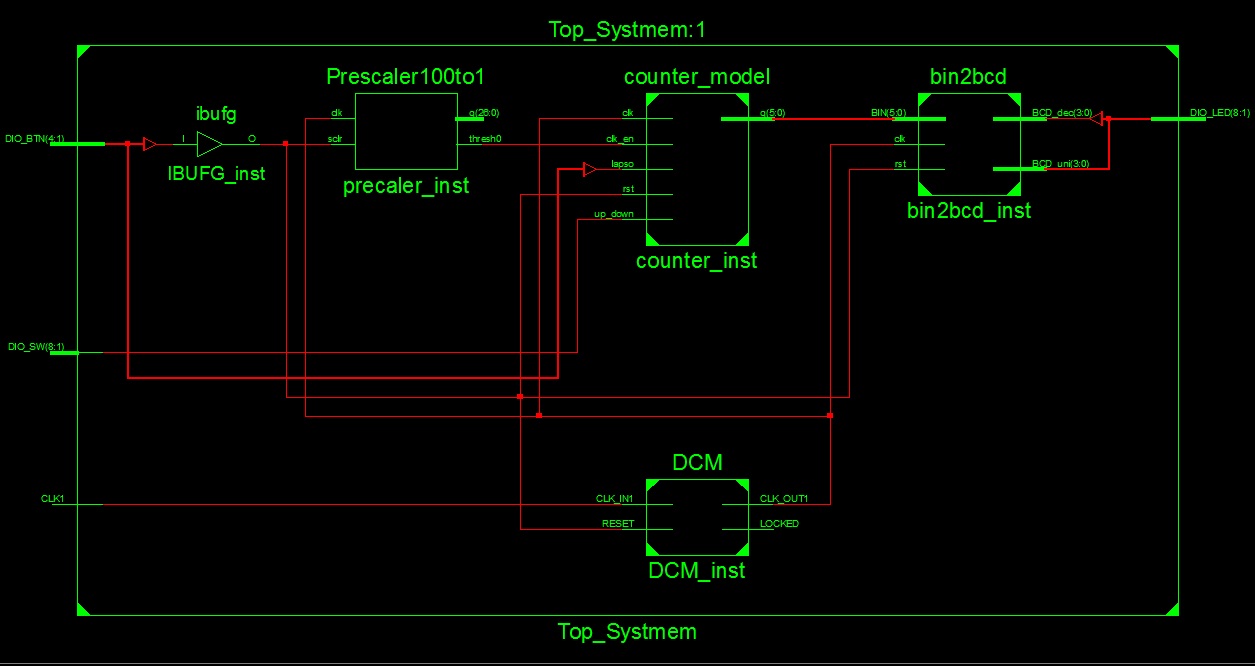
**4- Funcionamiento del "Contador con el lapso incluido":**

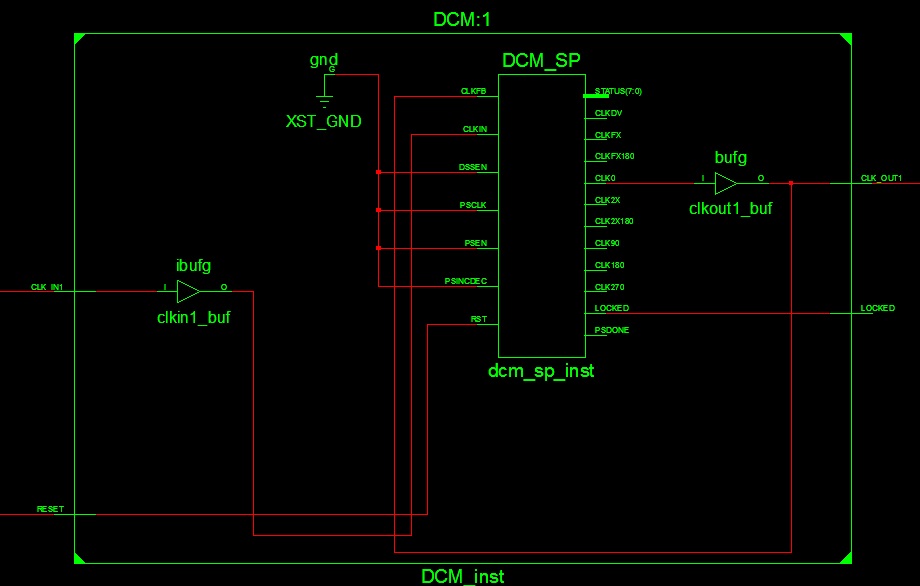
**5- Simulación del contador con lapso y el bin2bcd:**

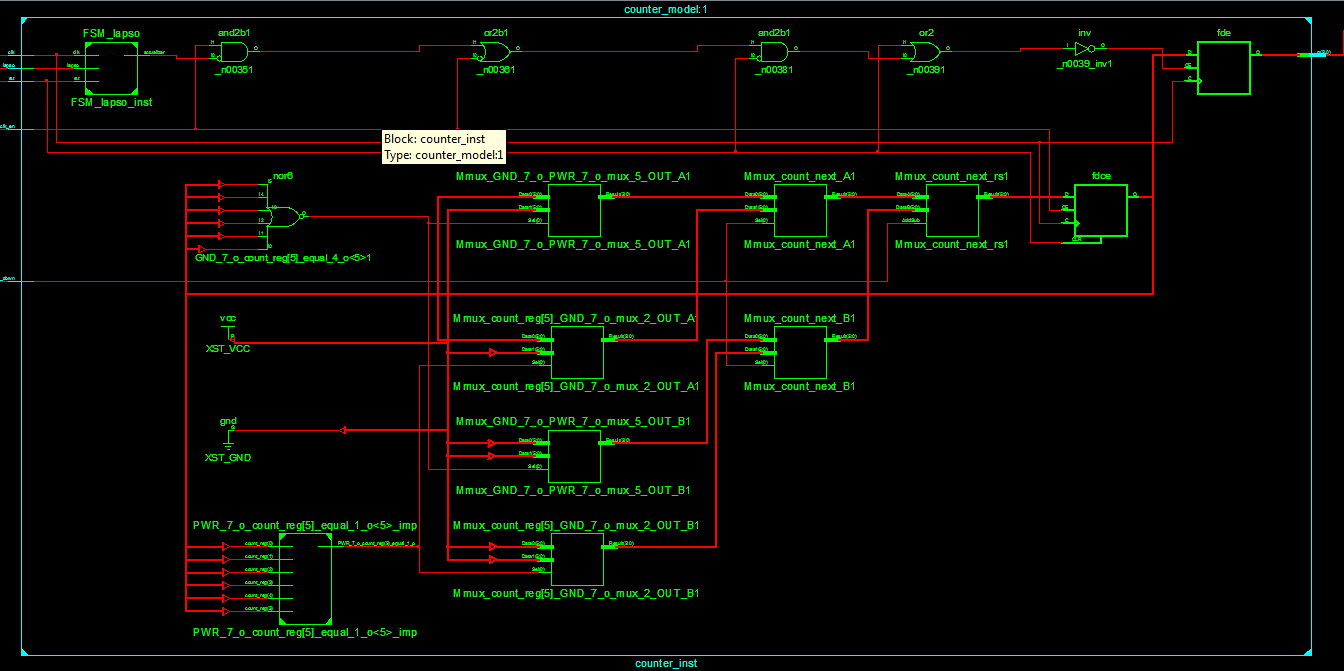
**6 - Simulación temporal**

**1-Implementación del proyecto en general:**

**# ESQUEMÁTICOS DEL CIRCUITO**

****





* Descripción de las principales conexiones:

En los anteriores esquemáticos se observa que se ha realizado la creación de un sistema llamado Top\_System con 3 entradas (DIO\_BTN (pulsadores de reset y lapso), DIO\_SW (para la señal de up/down) y el reloj CLK1) y una salida tipo bus (DIO\_LED (8:1)).

A su vez, este sistema se divide en varios modulos conectados como se muestra en el segundo esquemático. Se puede ver que las señales DIO\_BTN (1) (reset) y CLK1 no se utilizan directamente. El reset se hace pasar por un Buffer IBUFG y de ahí se distribuye para las demás entradas de reset de todos los módulos. La señal CLK1 se une al módulo DCM. Este módulo crea una nueva señal de reloj CLK0 que es la que utiliza el resto de los módulos del circuito.

El módulo prescaler100to1 se utiliza para configurar el valor de cuenta del circuito y así conseguir la frecuencia de cuenta perfecta del módulo Counter\_model.

Dentro del módulo Counter\_model se, puede ver que está incluido el módulo FSM\_Lapso el cual realiza la función Lapso; este módulo tiene incluido en él una función antirrebote para evitar fallos por los rebotes del pulsador DIO\_BTN(2), (lapso).

El último módulo es el bin2bcd, el cual simplemente convierte el valor de cuenta binario del contador a BCD para que se muestre en los led el valor de cuenta en un rango de 0 a 59.

**2 - Funcionamiento del circuito anti rebote:**

Para la implementación del circuito anti-rebote hemos creado una nueva entidad "antirebote", la cual tiene posee, aparte de las entradas del reloj "clk" y reset "rst",

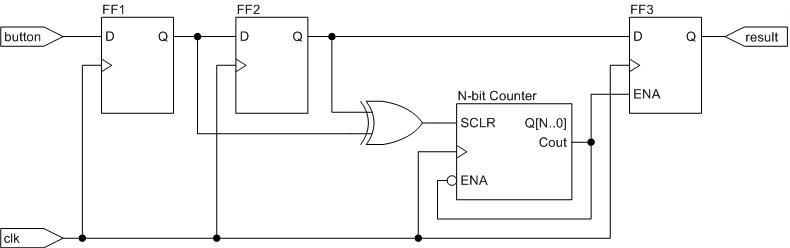
la entrada "entrada" donde se debe conectar la señal con rebote; y la salida "salida"

que es la señal de entrada sin rebote.

- A grandes rasgos, la entidad rebote lo que hace es detectar un cambio en la entrada mediante una XOR entre la señal de entrada en un flanco de rejoj y la señal de entrada del siguiente.

- Cuando lo detecta, acciona un contador binario, cuando acaba el contador binario, pone a la salida la entrada estable y el contador se para.

- Su esquema es el siguiente pero con un reset asíncrono:



# Si el contador binario es de N bits, la entidad tarda 2^N + 2 ciclos de reloj en actualizar la salida estable:

- 1 ciclo para detectar el cambio

- 2^N ciclos para que termine el contador

- 1 ciclo para asignar la entrada a la salida

# Siendo N el número de bits del contador, el tiempo que detectarse un rebote es:

siendo Frec\_reloj = 100MHz, si establecemos N = 20

Entonces el máximo tiempo entre rebotes sería de T = 10,5 ms

Este también es el tiempo que tardara darle valor a la salida desde que se produce el ultimo rebote

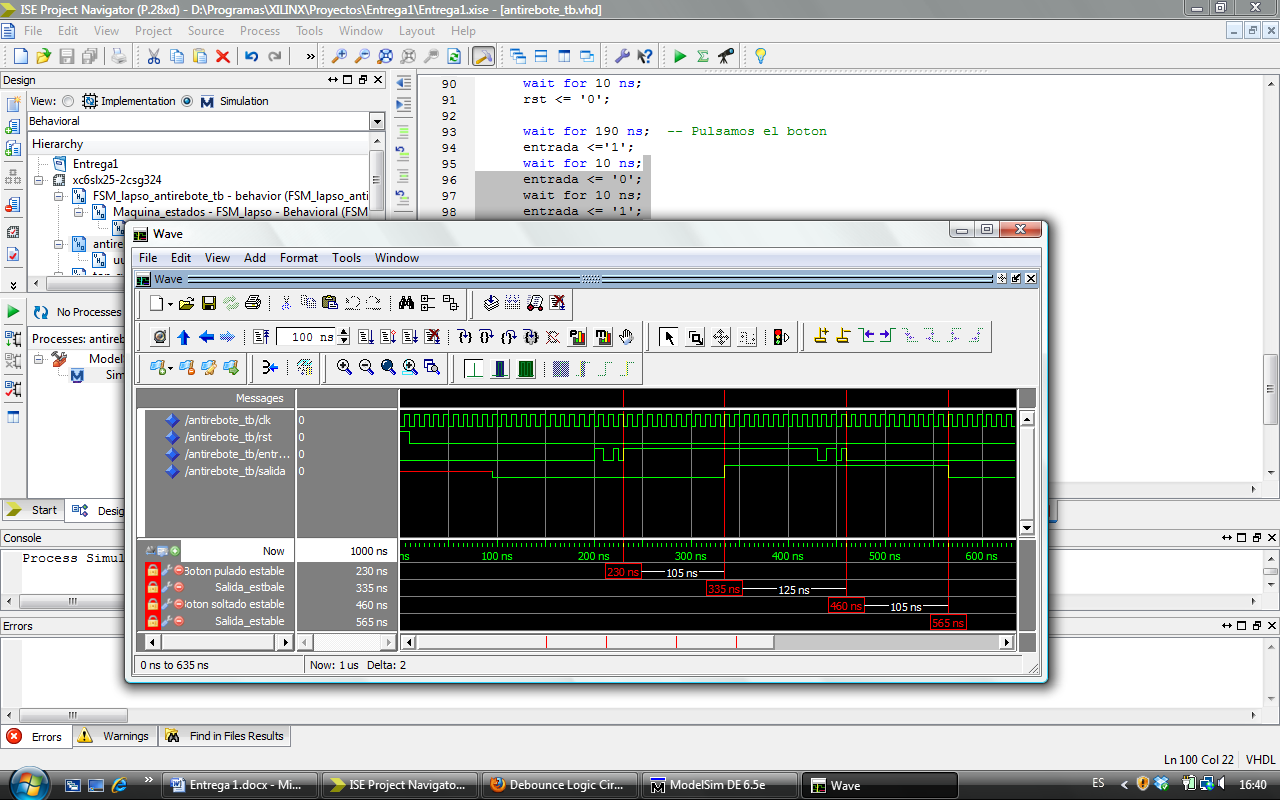
# Para el testbench hemos utilizado un tamaño de contador N = 3

- Siendo el periodo del clk 10 ns:

El tiempo que tarda en estabilizarse la entrada desde el último rebote es:

T\_rebote = (2^3 + 2 )\* 10 ns = 100 ns

- El test bench que simula pulsar y soltar el botón con rebote es el siguiente:



- Como podemos ver, el tiempo que tarda en cambiarse la salida desde el último rebote es 105 ns y no 100 ns,

esto es debido a que, la asignación se produce en los flancos de subida, que son a mitad del ciclo

**# Codigo de "antirebote.vhd":**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.std\_logic\_unsigned.all;

entity antirebote is

generic(

bits\_contador : integer := 20); -- 10 ciclos de reloj

PORT(

clk : in STD\_LOGIC; -- Reloj del sistema

rst : in STD\_LOGIC; -- Reset asincrono

entrada : in STD\_LOGIC; -- Señal de entrada con rebote

salida : out STD\_LOGIC); -- Señal de salida sin rebote

end antirebote;

architecture algoritmica of antirebote is

signal biestables\_D : STD\_LOGIC\_VECTOR(1 downto 0); -- Biestables a la entrada para detectar cambio en

-- La señal de entrada

signal clear: STD\_LOGIC; -- Clear sincrono del contador que se activa

-- Cuando cambia la señal de entrada

signal cuenta : STD\_LOGIC\_VECTOR(bits\_contador downto 0) := (others => '0'); -- salida del contador

begin

clear <= biestables\_D(0) xor biestables\_D(1); -- Deteca un cambio en la señal de entrada

--

process(clk,rst)

begin

if(rst = '1') then --Reset asincrono

cuenta <= (others => '0'); -- Activamos y reiniciamos la cuenta, congelando la salida

elsif(clk'EVENT and clk = '1') then -- Cuando flanco de subida

biestables\_D(0) <= entrada; -- La entrada del primer biestable\_D es la señal con rebote

biestables\_D(1) <= biestables\_D(0); -- La entada del segundo biestable\_D es la salida del primer biestable

-- Esto hace que haya un ciclo de reloj entre la salida de ambos biestables

-- haciendo posible medianta la puerta xor detectar cambios en la entrada

if(clear = '1') then -- Si la señal de entrada a cabiado

cuenta <= (others => '0'); -- Reestablecemos la cuenta

elsif(cuenta(bits\_contador) = '0') then --Si no ha cambiado pero cambió y aun no ha acabado el contador

-- por lo que se considera como señal no estables

cuenta <= cuenta + 1; -- Aumentamos el contador (binario)

-- Tener en cuenta que este contador, si es de 20 bits, "cuenta" tiene 21 bits

-- El contador solo cuenta hasta "0\_1111\_1111\_1111\_1111"

-- Al llegar al siguiente ("1\_0000\_0000\_0000\_0000") deja de contar

-- De esta manera el contador trabaja mucho mas rapido

-- Cuando el contador llega a "1\_0000\_0000\_0000\_0000", se para ahi y al haber flanco de subida

-- no se cumplira la condicion "cuenta(bits\_contador) = '0'", por lo que la salida sera siempre igual

-- a la entrada, por lo que de alguna manera el bit de mayor peso del contador es ENABLE de la salida

-- y el ENABLE del contador, al llegar a '1', el contador se desactiva y la salida se activa

-- cuando es '0', el contador cuenta y se activa y la salida se desactiva

else --Si cambio la entrada hace tiempo y el contador acaba de terminar

salida <= biestables\_D(1); --Llevamos la entrada a la salida

end if; -- if del rst o valor del contador

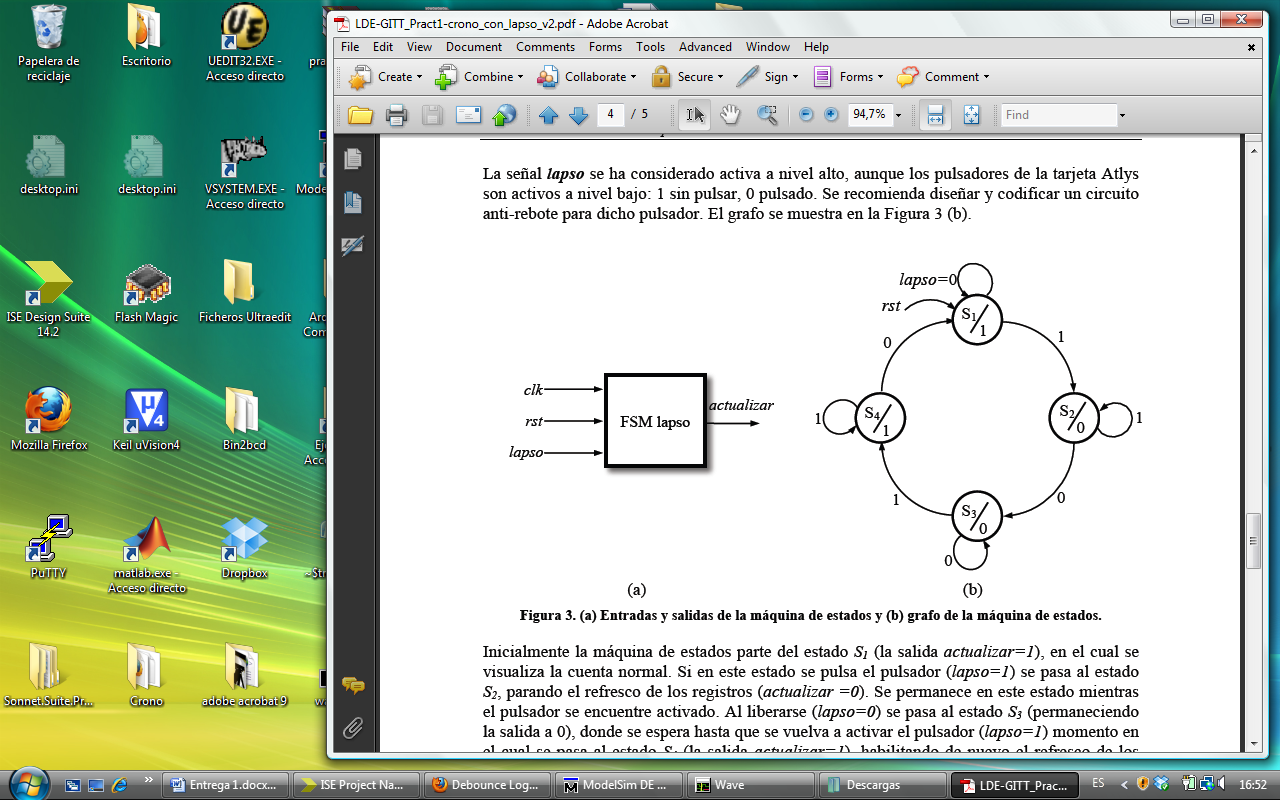
end if; -- if del flanco de subida

end process;

end algoritmica;

**3- Funcionamiento de la máquina de estados FSM\_Lapso**

# Para la máquina de estados, hemos creado una entidad "FSM\_lapso" la cual simula el circuito



# Para ello hemos creado una señal de tipo enumeral con los estados, y mediando un proceso sensible al cambio de la señal de entrada "lapso" hemos creado la maquina de estados.

- Dentro de este circuito instanciamos el "antirebote" para evitar el efecto rebote

# Para el testbench, usamos un circuito antirebote con N = 3:

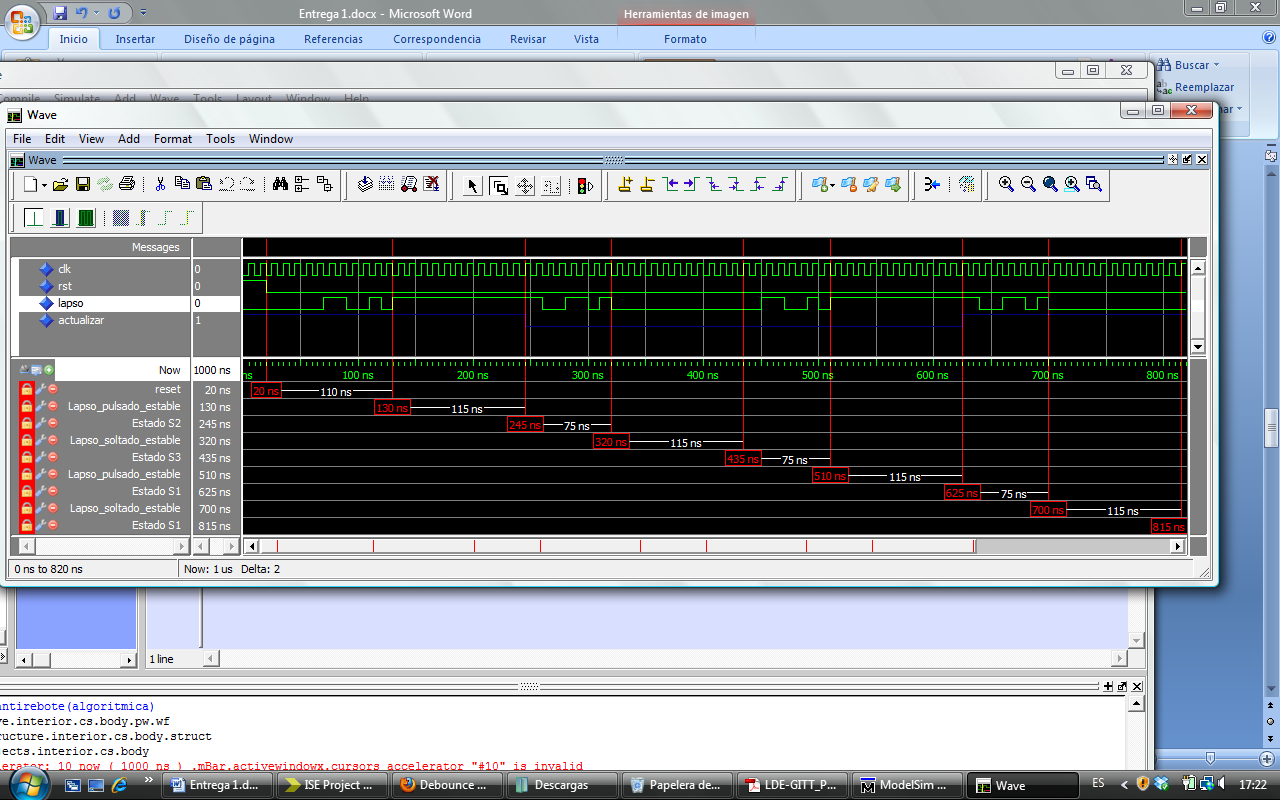
- En este caso el circuito antirebote tardara 10 ciclos de reloj en dar la salida del lapso estable

Además tardara otro ciclo en que la máquina de estado pueda trabajar con el nuevo valor estable del lapso.

- Así pues se tardan 11 ciclos = 110 ns desde el ultimo rebote hasta que la máquina de estados tenga el nuevo valor estable de la señal lapso.

# En el código viene instrucción por instrucción como esta implementado

# En el test\_bench, partimos del reset y simulamos 2 pulsaciones del botón con rebote:



# Como se puede ver, pasan 115 ns y no 110 ns desde que se estabilizan los rebotes hasta que a la máquina de estados le llega el cambio de lapso, esto es debido a que, la asignación se produce en los flancos de subida, que son a mitad del ciclo.

- Los ficheros del testbench no vemos necesario ponerlos ya que ocupan mucho pero ponemos aquí las primeras líneas del proceso del testbench:

stim\_proc: process

begin

rst <= '1';

-- hold reset state for 100 ns.

wait for 20 ns;

rst <= '0'; -- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 5\*clk\_period; -- Pulsamos el boton

lapso <= '1'; -- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '0'; -- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '1'; -- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '0'; -- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '1'; -- Valor estable de lapso

-- El contador se inicializa y se tarda 11 ciclos en que afecte a salida

**# Código de "FSM\_lapso.vhd"**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_logic\_arith.all;

entity FSM\_lapso is

Port ( clk: in STD\_LOGIC;

rst : in STD\_LOGIC;

lapso : in STD\_LOGIC;

actualizar: out STD\_LOGIC);

end FSM\_lapso;

architecture Behavioral of FSM\_lapso is

-- Declaramos sistema anti-rebote

COMPONENT antirebote

PORT(

clk : IN std\_logic;

rst : IN std\_logic;

entrada : IN std\_logic;

salida : OUT std\_logic );

END COMPONENT;

-- Señales y estados

type estadoFSM is (S1, S2, S3, S4); --Listado de estados

attribute ENUM\_ENCODING: STRING;

attribute ENUM\_ENCODING of estadoFSM:type is "00 01 10 11"; --Codificamos estados

signal estado: estadoFSM;

signal lapso\_estable : std\_logic;

begin

-- Instanciamos citcuito antirebote

antirebote\_inst: antirebote PORT MAP (

clk => clk,

rst => rst,

entrada => lapso,

salida => lapso\_estable );

-- Maquina de estados del lapso

FSM\_lapso: process(clk,rst)

begin

if (rst='1') then --Reset asincrono

estado <= S1; -- Estado inicial.

actualizar <= '1'; --Salida del estado inicial

elsif (clk'event and clk='1') then --Si no hay reset y flanco de subida

case estado is

when S1 =>

if (lapso\_estable='1') then

estado <= S2; -- Vamos a este estado

actualizar <= '0'; -- Salida en ese nuevo estado

end if;

when S2 =>

if (lapso\_estable='0') then

estado <= S3; -- Vamos a este estado

actualizar <= '0'; -- Salida en ese nuevo estado

end if;

when S3 =>

if (lapso\_estable='1') then

estado <= S4; -- Vamos a este estado

actualizar <= '1'; -- Salida en ese nuevo estado

end if;

when S4 =>

if (lapso\_estable='0') then

estado <= S1; -- Vamos a este estado

actualizar <= '1'; -- Salida en ese nuevo estado

end if;

end case;

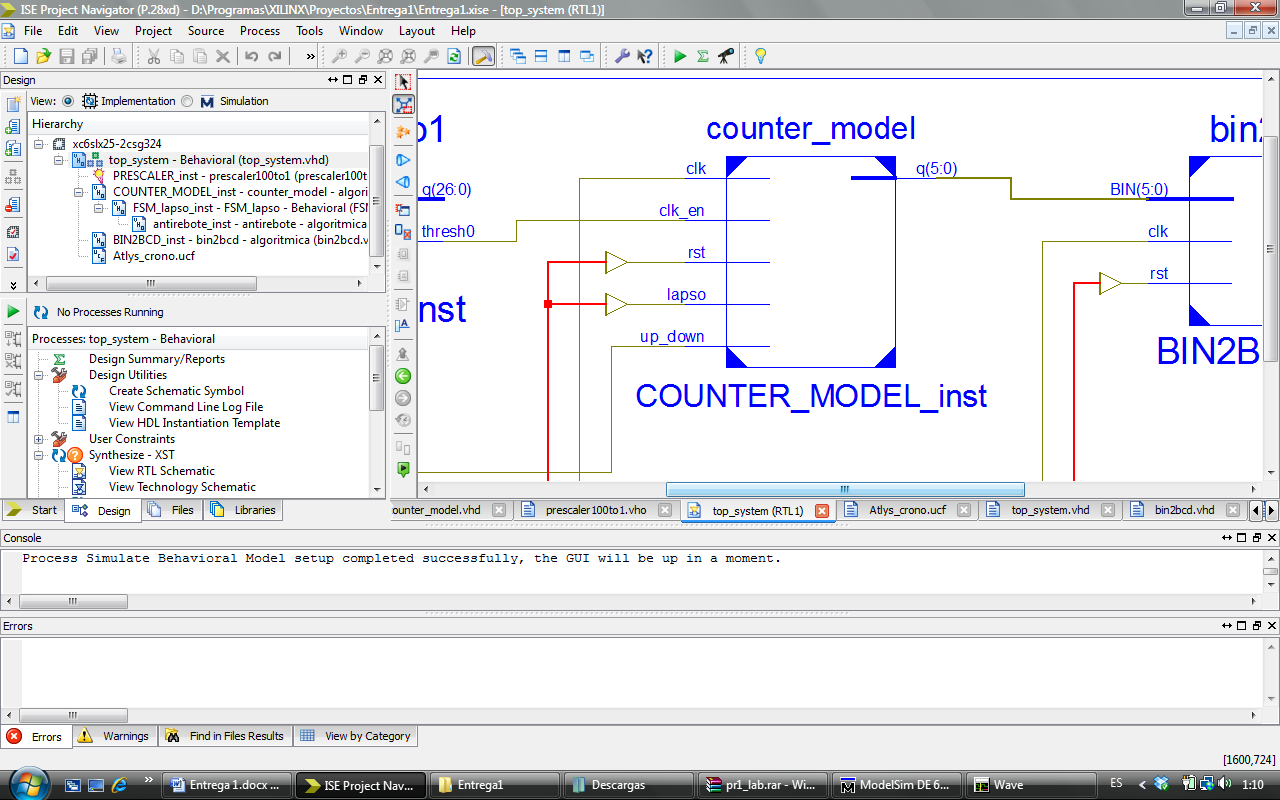
end if;

end process;

end Behavioral;

**4 - Funcionamiento del "Contador con el lapso incluido":**

# Hemos creado una entidad "Counter\_Model" con las entradas y salidas:



# Esta entidad crea un contador de 0 a 59 con rst asíncrono y señal de up\_down e instancia la entidad "FSM\_lapso", de manera que el valor de la cuenta del contador solo se lleva a la salida si la señal "actualizar" de la entidad instaciada "FSM\_lapso" está a 1.

- El funcionamiento del contador de 0 a 59 viene especificado en el código en sí.

- El funcionamiento del "FSM\_lapso" y del "antirebote" que instancia ya están explicados

# Para el test\_bench:

- Pondremos como contador del "antirebote" N = 3 bits

- Pondremos el "clk\_enable" del contador la señal de reloj de 100 MHz

- En este caso, para la señal "lapso", el numero de ciclos que pasa desde el ultimo rebote, hasta que se hace efectivo el cambio en el contador (Se para la salida o se vuelve a contar) son los 11 ciclos que tarda en cambiarse la señal "actualizar" mas el ciclo que tarda en usarlo el contador, por lo que son 12 ciclo

El tiempo que pasa desde que pulamos hasta que se hace efectivo el cambio serán 120 ns,

más los 5ns de siempre por el flanco de subida a mitad de ciclo: 125 ns

- Los estímulos que se darán al sistema son los de la simulación de pulsar el botón con rebote 2 veces y cambiar el up\_down del sistema.

**#Codigo de estimulos del test\_bench:**

stim\_proc: process

begin

rst <= '1';

up\_down<='1'; --Cuenta ascendente

-- hold reset state for 100 ns.

wait for 20 ns;

rst <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 5\*clk\_period; -- Pulsamos el boton

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '1'; -- Valor estable de lapso

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 15\*clk\_period ; -- Soltamos el boton

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '0'; -- Valor estable de lapso

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 15\*clk\_period; -- Pulsamos el boton

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '1'; -- Valor estable de lapso

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

up\_down<='0'; --Cuenta descendente

wait for 15\*clk\_period; -- Soltamos el boton

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk\_period ;

lapso <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk\_period ;

lapso <= '0'; -- Valor estable de lapso

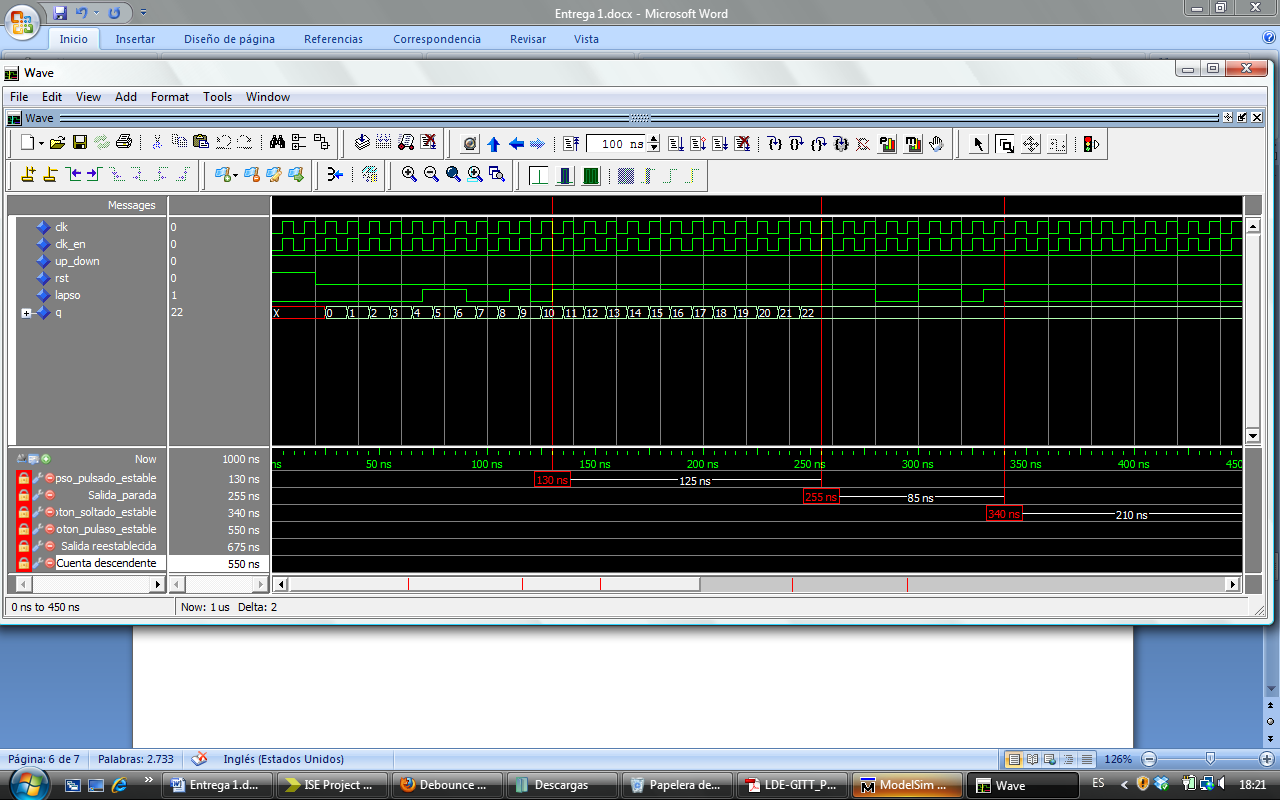
-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait;

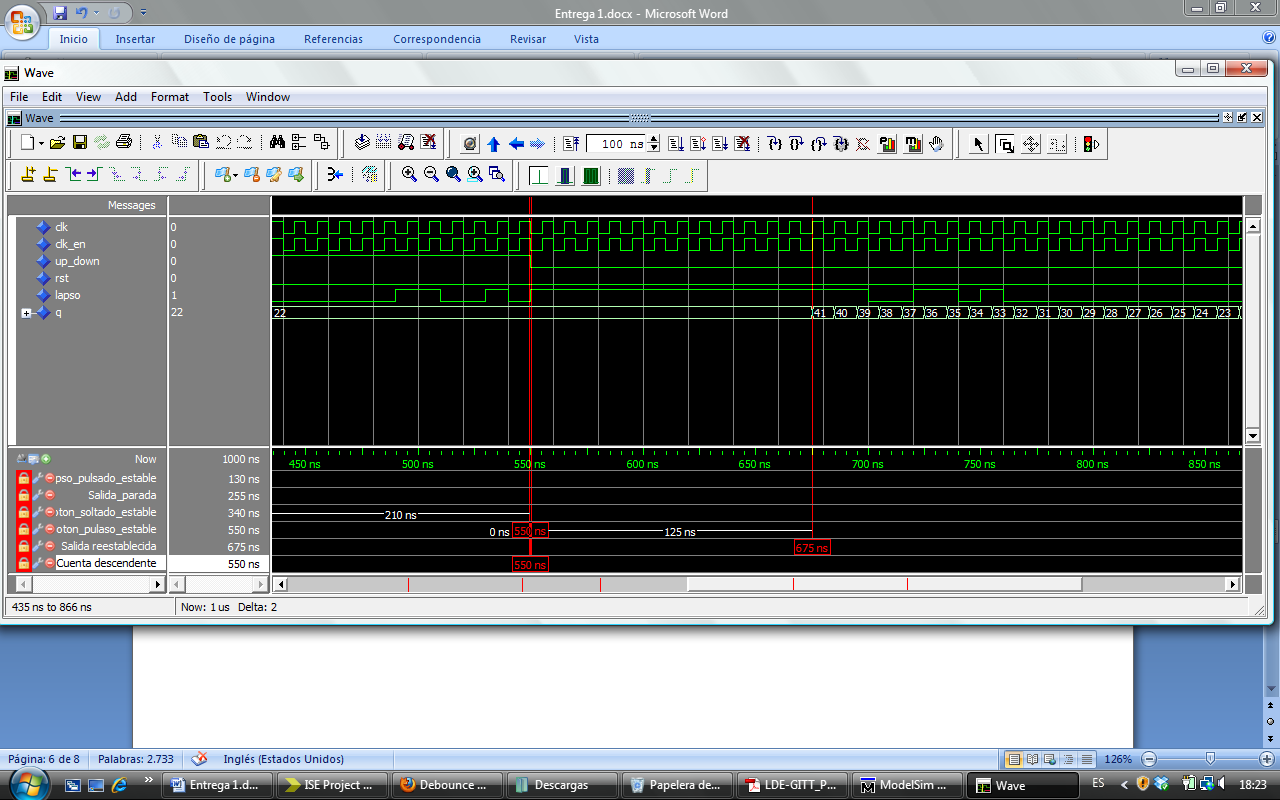
end process;

# Gráfica del test\_bench:

- Gráfica de 0 a 450



- Gráfica de 450 a 900



- Como se puede ver se tarda 125 ns desde que se pulsa el botón hasta que la salida del contador se congela o reactiva, tal como predijimos, también se muestra el funcionamiento de la señal "up\_down" que es cambiado en el nanosegundo 550

**# Código del "Contador con el Lapso" "Counter\_model.vhd"**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_signed.all; -- Operaciones sin signo

use ieee.std\_logic\_arith.all; -- Operaciones aritmeticas

--Contador hacia arriba o hacia abajo de 0 a 59

--Creamos un proceso que calcula el siguiente valor del contador, este proceso se activa

--cuando cambia el valor del contador "count\_reg" o la señal "updwn"

--El valor de la señal "count\_reg" cambiara cuando haya un flanco de reloj

--El segundo proceso se activa con el flanco de reloj o la señal de reset y lo que hace

--es cambiar el valor de la señal "count\_reg" poniendole el proximo valor,que se calculo

--con el primer proceso; este cambio acciona el primer proceo que genera el siguiente valor

--(Debe tardarse menos en generar el siguiente valor "proceso 1" que el tiempo entre flancos)

entity counter\_model is

Port ( clk : in STD\_LOGIC;

clk\_en : in STD\_LOGIC;

rst : in STD\_LOGIC;

lapso : in STD\_LOGIC;

up\_down : in STD\_LOGIC;

q : out STD\_LOGIC\_VECTOR (5 downto 0));

end counter\_model;

architecture algoritmica of counter\_model is

-- Declaramos la maquina de estados del lapso

COMPONENT FSM\_lapso

PORT(

clk : IN std\_logic;

rst : IN std\_logic;

lapso : IN std\_logic;

actualizar : OUT std\_logic

);

END COMPONENT;

-- Señales a utilizar

signal count\_reg : integer range 0 to 59; -- Valor actual del contador

signal count\_next: integer range 0 to 59; -- Valor que tomará el contador en el siguiente flanco

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

begin --Empiezan las instrucicones concurrentes

-- Instanciamos la maquina de estados

FSM\_lapso\_inst: FSM\_lapso PORT MAP (

clk => clk,

rst => rst,

lapso => lapso,

actualizar => actualiza

);

--Proceso que te calcula el siguiente valor de count\_reg, es decir, count\_next

--Cuando cambie el valor del contador o la señal updwn

process(count\_reg,up\_down)

begin

if (up\_down = '1') then -- Si la cuenta es ascendente

if (count\_reg = 59) then --Si hemos llegado al tope

count\_next <= 0;

else

count\_next <= count\_reg + 1;

end if;

else --Si la cuenta es descendente

if (count\_reg = 0) then --Si hemos llegado al tope

count\_next <= 59;

else

count\_next <= count\_reg - 1;

end if;

end if;

end process;

-- Podriamos haber usado variables dentro del proceso para calcular el siguiente valor del contador

-- pero necesitamos señales igualmente para comunicar los procesos

-- Este proceso varia el valor de count\_reg e indica en funcion de la señal "actualiza"

-- si ponemos el valor de count\_reg a la saida

process(clk, rst)

begin

if (rst = '1') then --Reset asincrono

count\_reg <= 0;

elsif (clk = '1' and clk'event) then --Si flanco de subida

if(clk\_en='1') then -- Si esta habilitado (se habilita cuando pasa un seg)

count\_reg <= count\_next; -- Se actualiza la salida

if (actualiza = '1') then

q <= CONV\_STD\_LOGIC\_VECTOR(count\_reg,6); -- Ponemos el valor del contador a la salida

end if;

end if;

end if;

end process;

end algoritmica;

**5 - Simulación del contador con lapso y el bin2bcd:**

# Para la simulación del sistema completo, al no funcionar ni el preescaler ni el DCM en el modelsim, lo que hemos hecho ha sido crear una entidad conectando el contador y el bin2bcd. Para ello:

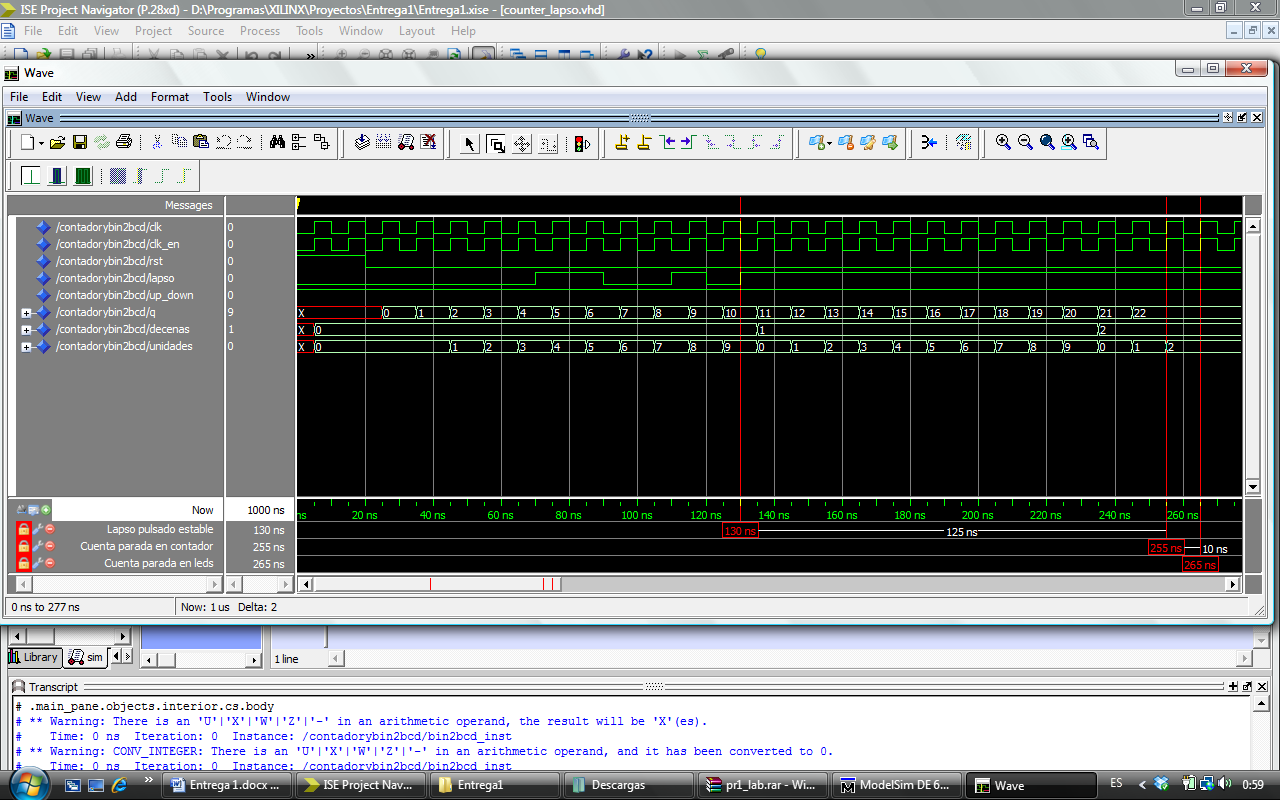
- Hemos conectado la salida binaria "q" del contador a la entrada del "bin2bcd.vhd"

- Hemos conectado al "clk\_en" del contador la señal de reloj "clk" (Para poder ver la simulación)

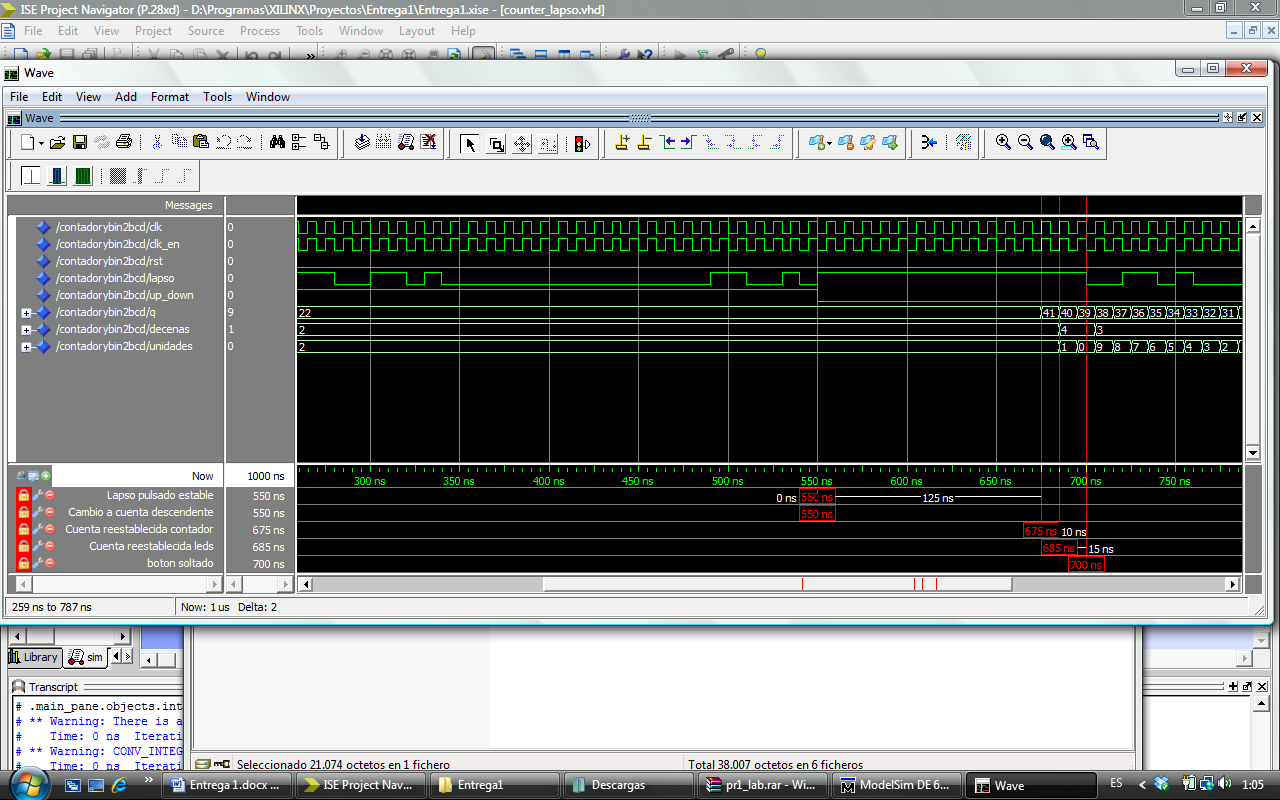
- El contador binario del "antirebote" tiene N = 3 bits

# Aquí están las graficas de lo que ocurre al pulsar dos veces el botón de lapso con rebote

- La primera pulsación hace que el contador deje de poner a su salida el valor de la cuenta.



- La primera pulsación hace que el contador vuelva a poner a su salida el valor de la cuenta.



# Como se puede apreciar, contador sigue contando internamente cuando la salida esta parada

Se puede ver que pasa un ciclo de reloj desde que el contados pone a la salida un valor hasta que este es mostrado en los leds.

**6 - Simulación temporal:**

# La simulación funcional del sistema completo con:

- Preescaler = 3 ciclos

- Contador binario antirrebote N = 3 bits => 8 + 2 ciclos

Debido a esto, recordando que el funcionamiento de nuestro sistema antirrebote, después del ultimo rebote, tardará unos 10 – 11 ciclos en parar la salida, por lo que el valor de cuanta aumenta unos 4 valores desde que presionamos el botón hasta que se hace efectiva la parada o reanudo de la visualización.

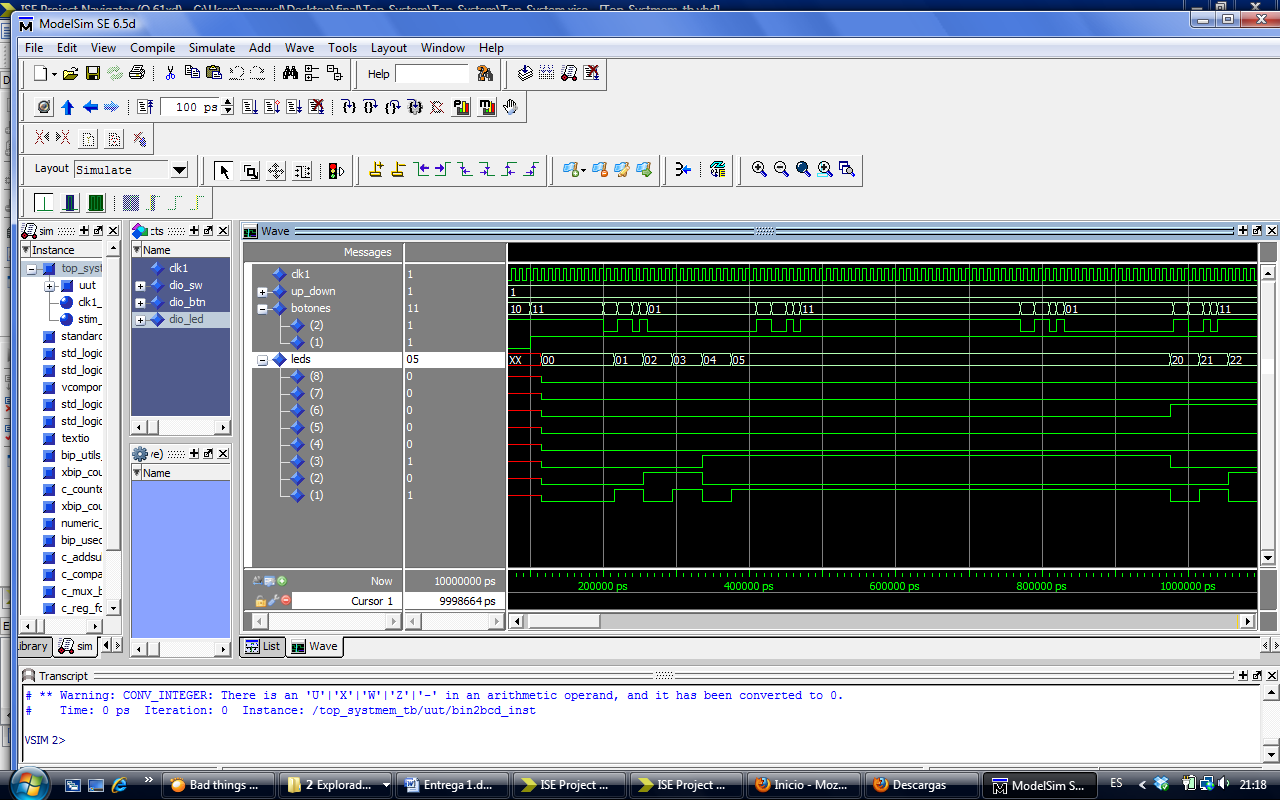
- Dichos parámetros también se usan para la simulación temporal

# Los valores reales del sistema completo serán:

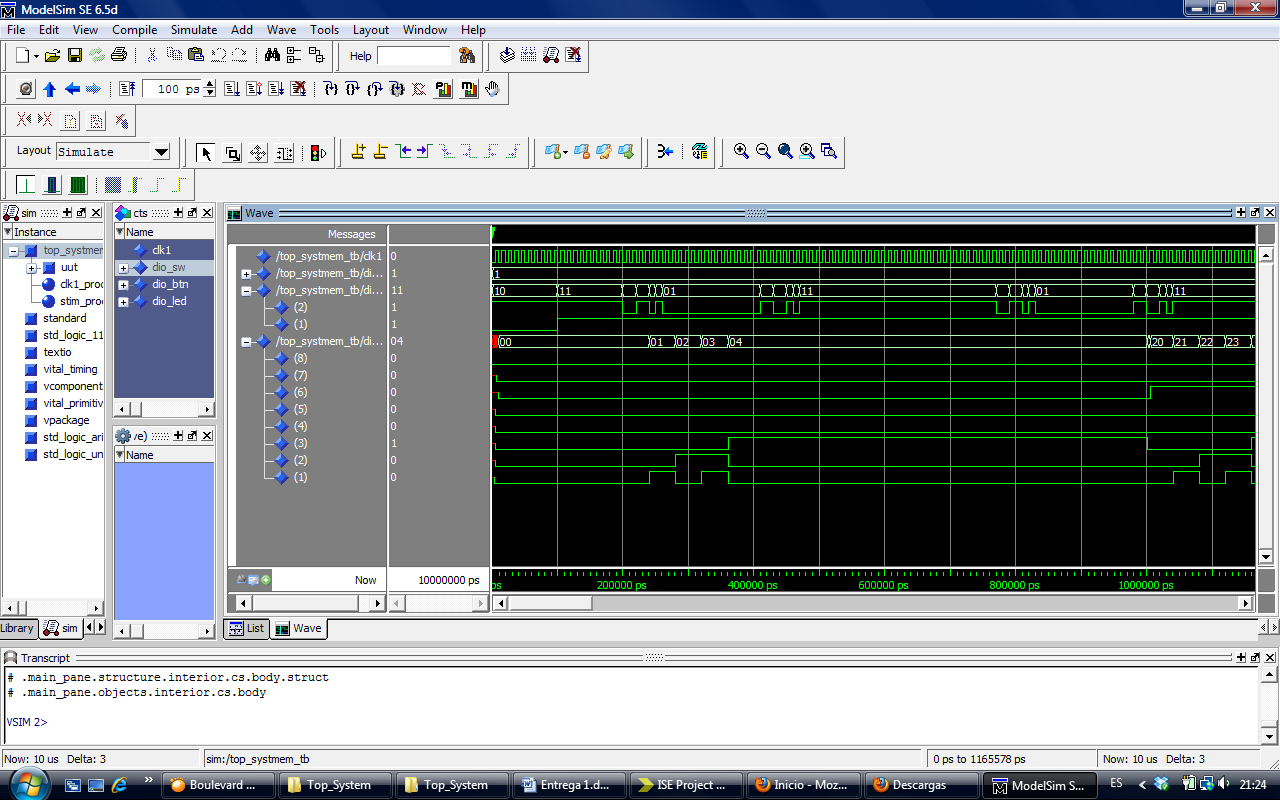
- Preescaler: 100.000.000 ciclos => 1 seg

- Rebote N = 21 bits => 21 mseg

Por lo que en este caso, si que al pulsar, la visualización se para o reanuda sin que el valor de cuenta visualizado varíe.



- Simulacion Temporal:



FUNCIONA !!!!!

El testbench es el siguiente:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric\_std.ALL;

ENTITY Top\_Systmem\_tb IS

END Top\_Systmem\_tb;

ARCHITECTURE behavior OF Top\_Systmem\_tb IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT Top\_Systmem

PORT(

CLK1 : IN std\_logic;

DIO\_SW : IN std\_logic\_vector(1 downto 1);

DIO\_BTN : IN std\_logic\_vector(2 downto 1);

DIO\_LED : OUT std\_logic\_vector(8 downto 1)

);

END COMPONENT;

--Inputs

signal CLK1 : std\_logic := '0';

signal DIO\_SW : std\_logic\_vector(1 downto 1) := (others => '0');

signal DIO\_BTN : std\_logic\_vector(2 downto 1) := (others => '0');

--Outputs

signal DIO\_LED : std\_logic\_vector(8 downto 1);

-- Clock period definitions

constant CLK1\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: Top\_Systmem PORT MAP (

CLK1 => CLK1,

DIO\_SW => DIO\_SW,

DIO\_BTN => DIO\_BTN,

DIO\_LED => DIO\_LED

);

-- Clock process definitions

CLK1\_process :process

begin

CLK1 <= '0';

wait for CLK1\_period/2;

CLK1 <= '1';

wait for CLK1\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

DIO\_BTN(1) <= '0'; -- reset pulsado

DIO\_BTN(2) <= '1'; -- pulso no pulsado

DIO\_SW(1) <='1';

wait for 100 ns;

DIO\_BTN(1) <= '1'; -- adios reset

wait for 100 ns;

-- insert stimulus here

--pulsamos

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- Valor estable de DIO\_BTN(2)

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 15\*clk1\_period ; -- Soltamos el boton

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- Valor estable de DIO\_BTN(2)

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 30\*clk1\_period ; -- Pulsamos de nuevo

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- Valor estable de DIO\_BTN(2)

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 15\*clk1\_period ; -- Soltamos el boton

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 2\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '0'; -- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait for 1\*clk1\_period ;

DIO\_BTN(2) <= '1'; -- Valor estable de DIO\_BTN(2)

-- El contador\_del\_rebote se inicializa y se tarda 11 ciclos en que afecte a salida

wait;

end process;

END;