Diseño Automático de Sistemas Fiables
Memoria de práctica

David Martínez Campos

ÍNDICE

Introducción

Memoria para exponer los resultados y explicaciones de las prácticas de diseño automáticos de sistemas fiables. Aclarar que parte del código de estas prácticas ha sido hecho en la versión 2019.2.1 y otra parte en la version 2021.2.
El código se encuentra en el repositorio:
https://github.com/dmartinezc4/Dis_automatico_sistemas

Práctica 1. Debouncer

Como bien dice la práctica vamos a tener tres ficheros: el sincronizador, el debouncer y el fichero que los une a los dos llamado top_practica1. Aparte de esto tenemos las constraints que hay que cambiar para que en caso de ser puesto en la placa este funcione.

Sincronizador

Este es el código del sincronizador hecho por el profesor

```
-- 1 bit synchronizer based of n flip-flops for clock-domain crossings.
-- drive constant `data_in` to use as a reset synchronizer
--
-- Original author: Colm Ryan
--
-- Copyright (c) 2016 Raytheon BBN Technologies
-- Modified by pablo Sarabia to add reset in negate form

library ieee;
use ieee.std_logic_1164.all;

entity synchronizer is
  generic (
    RESET_VALUE    : std_logic := '0'; -- reset value of all flip-flops in the chain
    NUM_FLIP_FLOPS : natural := 2 -- number of flip-flops in the synchronizer chain
  );
  port(
    rst      : in std_logic; -- asynchronous, high-active
    clk      : in std_logic; -- destination clock
    data_in  : in std_logic;
    data_out : out std_logic
  );
end synchronizer;

architecture arch of synchronizer is

  --synchronizer chain of flip-flops
  signal sync_chain : std_logic_vector(NUM_FLIP_FLOPS-1 downto 0) := (others => RESET_VALUE);

  -- Xilinx XST: disable shift-register LUT (SRL) extraction
  attribute shreg_extract : string;
  attribute shreg_extract of sync_chain : signal is "no";

  -- Vivado: set ASYNC_REG to specify registers receive asynchronous data
  -- also acts as DONT_TOUCH
  attribute ASYNC_REG : string;
  attribute ASYNC_REG of sync_chain : signal is "TRUE";

begin

  main : process(clk, rst)
  begin
    if rst = '0' then
      sync_chain <= (others => RESET_VALUE);
    elsif rising_edge(clk) then
      sync_chain <= sync_chain(sync_chain'high-1 downto 0) & data_in;
    end if;
  end process;

  data_out <= sync_chain(sync_chain'high);

end architecture;
```

Debouncer

Debido al código ser muy grande primero vemos como se instancia la source del debouncer y las signals y constantes necesarias.

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

entity debouncer is
    generic(
        g_timeout          : integer   := 5;        -- Time in ms
        g_clock_freq_KHZ   : integer   := 100_000   -- Frequency in KHz of the system
    );
    port (
        rst_n       : in    std_logic; -- asynchronous reset, low -active
        clk         : in    std_logic; -- system clk
        ena         : in    std_logic; -- enable must be on 1 to work (kind of synchronous reset)
        sig_in      : in    std_logic; -- signal to debounce
        debounced   : out   std_logic  -- 1 pulse flag output when the timeout has occurred
    );
end debouncer;


architecture Behavioural of debouncer is

    -- Calculate the number of cycles of the counter (debounce_time * freq), result in cycles
    constant c_cycles           : integer := integer(g_timeout * g_clock_freq_KHZ) ;
    -- Calculate the length of the counter so the count fits
    constant c_counter_width    : integer := integer(ceil(log2(real(c_cycles))));

    -- ------------------------------------------------------------------------
    type state_type is (idle,btn_prs,valid,btn_unprs);--estados
    signal current_state,next_state: state_type;--registros de estados
    signal counter: unsigned(c_counter_width-1 downto 0);-- contador
    signal time_elapsed: std_logic;--ha pasado el tiempo
    -- ------------------------------------------------------------------------
```

Nada mas terminar tenemos los dos primeros process el del contador y el de cambiar de estado respectivamente.

```
begin
    --Timer
    process (clk, rst_n)
    begin
    -- ------------------------------------------------------------------------
    -- Completar el timer que genera la señal de time_elapsed para trancionar en las maquinas de estados
    if(rst_n='0') then
        time_elapsed<='0';
        counter<=(others=>'0');

    elsif(rising_edge(clk))then
            if(counter < to_unsigned(c_cycles,counter'length))then
                counter<=counter+1;
                time_elapsed<='0';
            else
                time_elapsed<='1';
                counter<=(others=>'0');
            end if;

    end if;

    -- ------------------------------------------------------------------------
    end process;

    --FSM Register of next state
    process (clk, rst_n)
    begin

    if(rst_n ='0') then
        current_state<=idle;
    elsif(rising_edge(clk)) then
            current_state<=next_state;
    end if;
    end process;
```

Tras esto implementaremos la siguiente máquina de Moore propuesta en un process

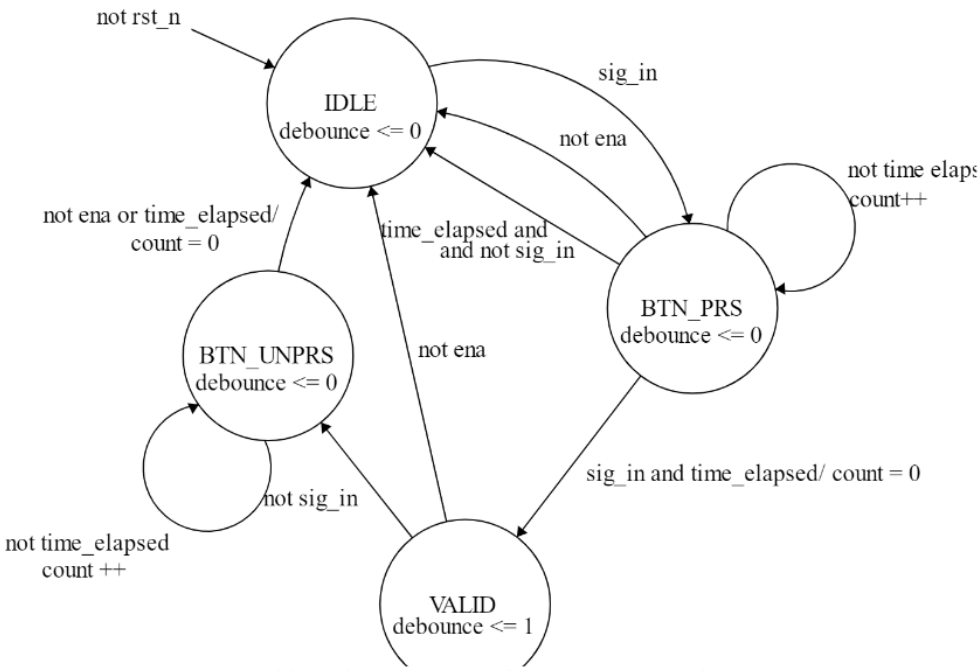


*Figura 1.*Posible implementación mediante una máquina de Moore.

La lista de sensibilidad del process será: el current state (porque es lo necesario para todos los process de fsm tras hacer los de clk); y luego sig_in, rst_n y ena porque, como se ve en el diagrama, si una de estas se activa puede cambiar el flujo de trabajo.
Al ser hecho con case en vez de if, necesitamos un when others, que en este caso hace que la señal de debounced sea 0 y hace que el siguiente estado al que saltar sea el de idle.

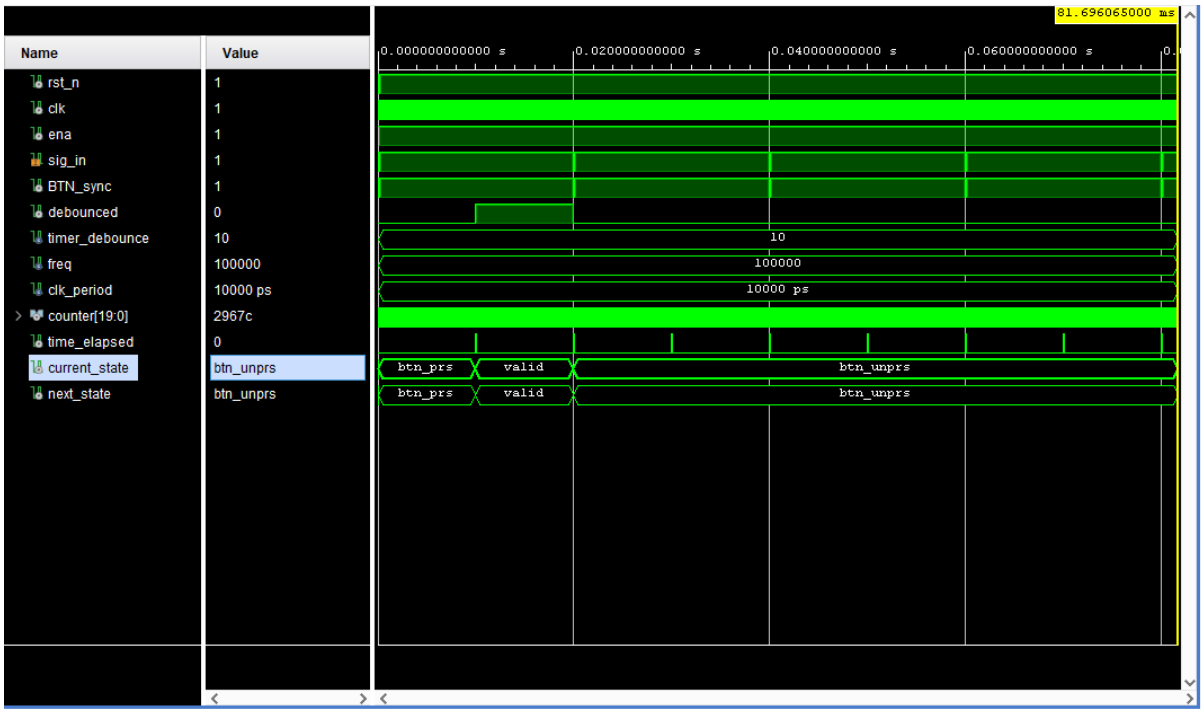
```
89      process (current_state,sig_in,rst_n,ena)--sensitivity list)
90      begin
91          case current_state is
92              when idle => --idle
93                      debounced<='0';
94                      if(sig_in='1') then
95                          next_state<=btn_prs;
96                      elsif(rst_n='0') then
97                          next_state<=idle;
98                      end if;
99              when btn_prs => --btn_prs
100                 debounced<='0';
101
102                 if(ena='0') then
103                     next_state<=idle;
104
105                 elsif(time_elapsed<='0') then
106                     next_state<=btn_prs;
107
108                 elsif(time_elapsed<='1' and sig_in<='0')then
109                     next_state<=idle;
110                 elsif(time_elapsed<='1' and sig_in<='1') then
111                     next_state<=valid;
112                 end if;
113             when valid => --valid
114                 debounced<='1';
115                 if(ena<='0')then
116                     next_state<=idle;
117                 elsif(sig_in<='0')then
118                     next_state<=btn_unprs;
119                 end if;
120             when btn_unprs => --btn_unprs
121                 debounced<='0';
122                 if(time_elapsed<='0')then
123
124                     next_state<=btn_unprs;
125                 elsif(ena='0' or time_elapsed<='0') then
126                     next_state<=idle;
127                 end if;
128             when others=>
129                 next_state<=idle;debounced<='0';
130         end case;
131     end process;
132 end Behavioural;
```

Respecto a los testbench, debido a como lo hemos hecho nunca pasa a idle tras btn_unprs. Además como el clk depende del reloj que es muy grande cambia cada mucho tiempo, pero vamos a mostrar la imagen general y un poco de instantes específicos.

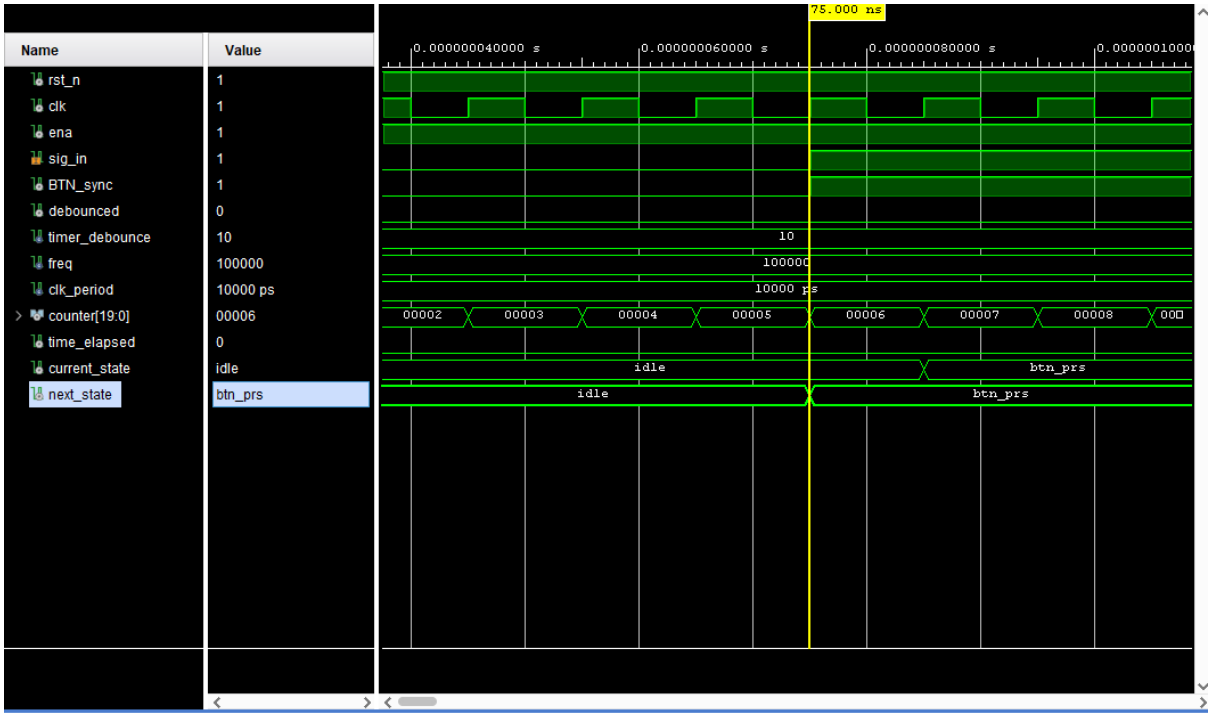
Primero el código del testbench usado para el debouncer

```
23  library IEEE;
24  use IEEE.STD_LOGIC_1164.ALL;
25  use IEEE.NUMERIC_STD.ALL;
26
27  entity tb_debouncer is
28  end tb_debouncer;
29
30  architecture testBench of tb_debouncer is
31    component debouncer is
32      generic(
33          g_timeout          : integer   := 5;         -- Time in ms
34          g_clock_freq_KHZ   : integer   := 100_000    -- Frequency in KHz of the system
35      );
36      port (
37          rst_n       : in    std_logic;
38          clk         : in    std_logic;
39          ena         : in    std_logic;
40          sig_in      : in    std_logic;
41          debounced   : out   std_logic
42      );
43    end component;
44
45    constant timer_debounce : integer := 10; --ms
46    constant freq : integer := 100_000; --KHZ
47    constant clk_period : time := (1 ms/ freq);
48    -- Inputs
49    signal  rst_n       :   std_logic := '0';
50    signal  clk         :   std_logic := '0';
51    signal  ena         :   std_logic := '1';
52    signal  BTN_sync      :   std_logic := '0';
53    -- Output
54    signal  debounced   :   std_logic;
55
56  begin
57    UUT: debouncer
58      generic map (
59        g_timeout        => timer_debounce,
60        g_clock_freq_KHZ => freq
61      )
62      port map (
63        rst_n     => rst_n,
64        clk       => clk,
65        ena       => ena,
66        sig_in    => BTN_sync,
67        debounced => debounced
68      );
69    clk <= not clk after clk_period/2;
70    process is
71    begin
72      wait until rising_edge(clk);
73      wait until rising_edge(clk);
74      rst_n <= '1';
```

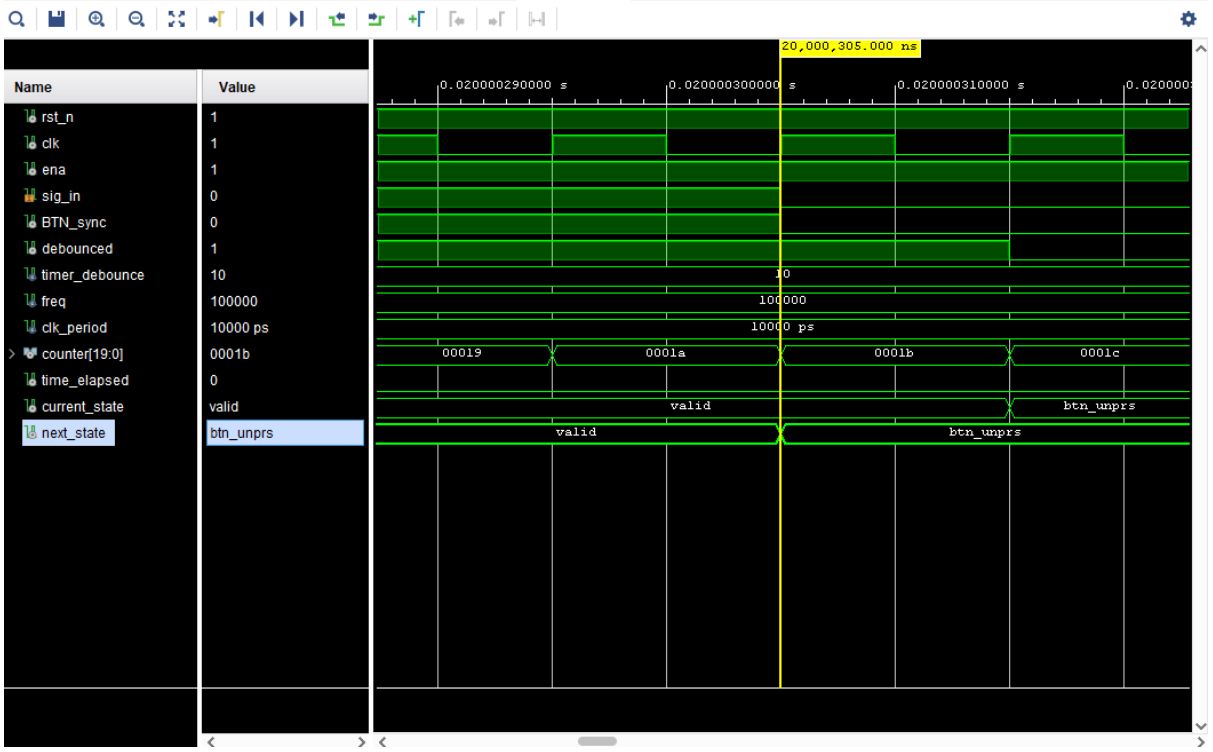
```
76        wait for 50 ns;
77        wait until rising_edge(clk);
78        BTN_sync <='1';
79        wait for 100 ns;
80        wait until rising_edge(clk);
81        BTN_sync <= '0';
82        wait for 100 ns;
83        wait until rising_edge(clk);
84        BTN_sync <='1';
85        wait for 20 ms;
86        wait until rising_edge(clk);
87        BTN_sync <='0';
88     end process;
89  end testBench;
```



Esto sería la imagen general, seleccionando el estado btn_unprs, donde se cumple que el debounced es 0 y debido a que ena y time elapsed es 1 el siguiente estado es btn unprs.



Esta sería la imagen del principio del testbench en el que está en estado idle, y tras 5 segundos este pasará al estado btn_prs.



Y por último un instante de tiempo donde ocurren muchos cambios muy seguidos

Top_practica1

De nuevo al ser un fichero bastante grande lo mostraremos en partes.
Primero la instanciación de la entidad en si y la instanciación de los componentes que esta usa, además de las signals necesarias.

```
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use IEEE.NUMERIC_STD.ALL;
24
25  entity top_practical is
26    generic (
27        g_sys_clock_freq_KHZ  : integer := 100e3; -- Value of the clock frequencies in KHz
28        g_debounce_time       : integer := 20;  -- Time for the debouncer in ms
29        g_reset_value         : std_logic := '0'; -- Value for the synchronizer
30        g_number_flip_flps    : natural := 2  -- Number of ffs used to synchronize
31    );
32    port (
33        rst_n         : in std_logic; --Negate reset must be connected to the switch 0
34        clk100Mhz     : in std_logic; -- Connect to the main clk
35        BTNC           : in std_logic; -- Connect to the BTNC
36        LED           : out std_logic --Connect to the LED 0
37    );
38  end top_practical;
39
40  architecture behavioural of top_practical is
41    component debouncer is
42      generic(
43          -- In a generic declaration the value is overrided by the top module
44          g_timeout         : integer  := 5; -- Time for debouncing (value overrided)
45          g_clock_freq_KHZ  : integer  := 100_000 -- Frequency in KHz of the system (value overrided)
46      );
47      port (
48          rst_n       : in    std_logic; -- asynchronous reset, low -active
49          clk         : in    std_logic; -- system clk
50          ena         : in    std_logic; -- enable must be on 1 to work (kind of synchronous reset)
51          sig_in      : in    std_logic; -- signal to debounce
52          debounced   : out   std_logic  -- 1 pulse flag output when the timeout has occurred
53      );
54    end component;
55
56    component synchronizer is
57    generic (
58      RESET_VALUE    : std_logic  := '0'; -- reset value of all flip-flops in the chain
59      NUM_FLIP_FLOPS : natural    := 2 -- number of flip-flops in the synchronizer chain
60    );
61    port(
62      rst      : in std_logic; -- asynchronous, low-active
63      clk      : in std_logic; -- destination clock
64      data_in  : in std_logic; -- data that wants to be synchronized
65      data_out : out std_logic -- data synchronized
66    );
67    end component;
68
69    signal BTN_sync : std_logic; -- Synchronized signal of the BTNC
70    signal Toggle_LED : std_logic; -- Internal signal to connect between the debouncer and the toggle process
71    signal LED_register, state_LED : std_logic; -- The output signal of the LED registered and unregisterd
```

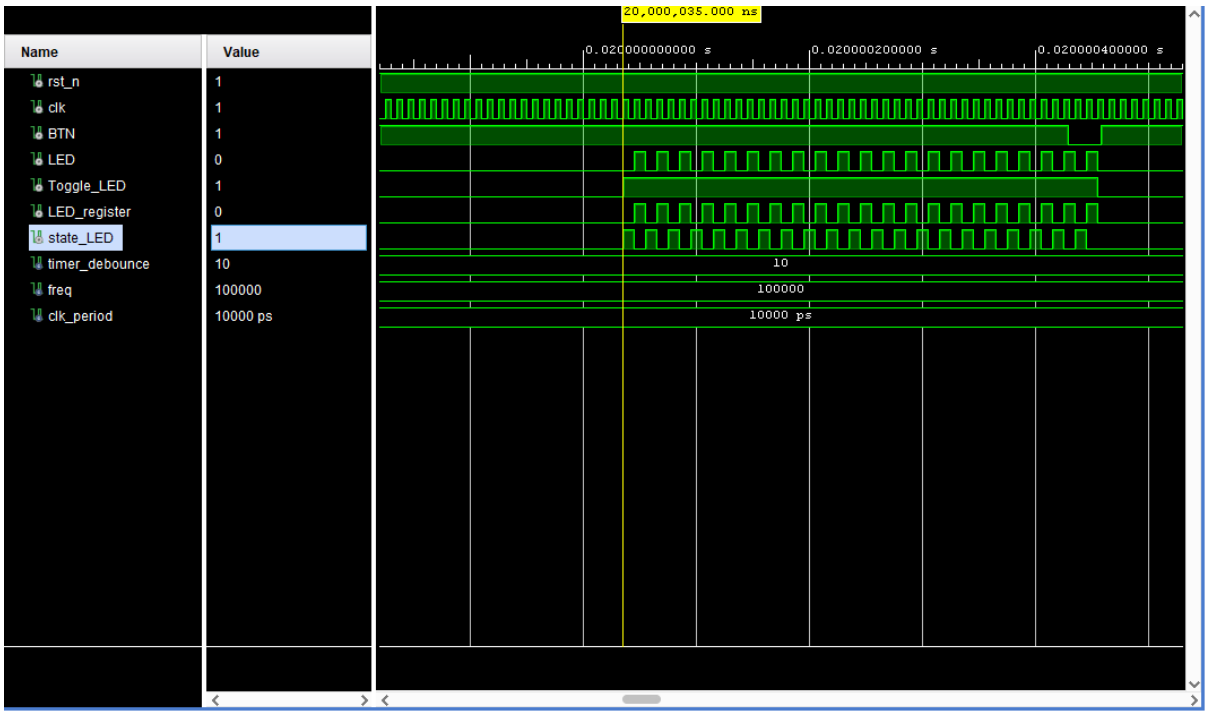
Tras esto empezamos el begin de la architecture.
Desde la línea 73 hasta la 97 hacemos un mapeado de las variables para que se puedan comunicar, así como de los genéricos.

```
begin
  -- DEBOUNCER
  debouncer_inst: debouncer
    generic map (
      g_timeout        => g_debounce_time,
      g_clock_freq_KHZ => g_sys_clock_freq_KHZ
    )
    port map (
      rst_n     => rst_n,
      clk       => clk100Mhz,
      ena       => '1', -- Always enabled for this lab
      sig_in    => BTN_sync, -- Synchronized input from the synchronizer
      debounced => toggle_LED -- Output to the toggle LED process
    );
  -- SYNCHRONIZER
  synchronizer_inst: synchronizer
    generic map (
      RESET_VALUE    => g_reset_value,
      NUM_FLIP_FLOPS => g_number_flip_flps
    )
    port map (
      rst      => rst_n,
      clk      => clk100Mhz,
      data_in  => BTNC,
      data_out => BTN_sync
    );
  --  PROCESS to register LED output
  registerLED: process(clk100Mhz, rst_n) is
  begin
    if (rst_n = '0') then --Cuando el reset es 0 el estado ha de ser 0
      LED_register<='0';

    elsif rising_edge(clk100Mhz) then --en caso de no ser cero y haber flanco de subida, el registro de led será el toggle
      LED_register<=state_LED;

    end if;
  end process;

  -- PROCESS to toggle LED
  toggleLED: process(LED_register, Toggle_LED)
  begin
    state_LED <= toggle_LED xor LED_register;

  end process;
  -- Connect LED_register to the output
  LED <= LED_register;
end behavioural;
```

Como se puede observar hay dos process uno que lo que hace es hacer registro del output led y otro que lo que hace es encender el led. Fuera de todo process se hace la asignación para que la salida LED valga lo que haya en el LED_register.

Respecto al testbench

Esto sería la imagen general del testbench que como nos pasaba en el testbench anterior.



Esta sería la única parte (por como hemos hecho el debouncer), en la que el led cambia en ciertos instántes a 1.

Constraints

Comentarios/Observaciones

Respecto a los testbench, principalmente del debouncer: no sabemos donde poner time_elapsed<=’0’ sin que nos saliese unsigned y nos permitiese un muestreo cómodo en los tb. La idea sería poner time_elapsed<=’0’ en el process del timer en el momento en el que el counter todavia no ha llegado a tomar el tiempo; pero eso nos dejaría (en un testbench) un instánte de tiempo ínfimo con el que muestrear.
El testbench del debouncer que muestra que no pasa a idle por como

Práctica 2. Micrófono DDR

En esta práctica no le he podido dedicar todo el tiempo necesario y solo tengo los .vhd de la práctica.
Aparte he considerado un reset asíncrono al no especificarse nada en la memoria.

Generador de reloj síncrono

Este módulo ocupa muchas líneas lo primero que vamos a ver es la entidad que simplemente es la declarada por el profesor y las señales creadas para poder hacer un reloj síncrono mas las señales para la máquina de estados.
Necesitamos la libreria mathreal para poder hacer el redondeo “floor”.

```
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.numeric_std.all;
28  use IEEE.MATH_REAL.ALL;
29
30  entity fsm_sclk is
31      generic(
32          g_freq_SCLK_KHZ : integer := 1_500; -- Frequency in KHz of the
33                                              --synchronous generated clk
34          g_system_clock  : integer := 100_000 --Frequency in KHz of the system clk
35      );
36      port (
37          rst_n       : in std_logic; -- asynchronous reset, low active
38          clk         : in std_logic; -- system clk
39          start       : in std_logic; -- signal to start the synchronous clk
40          SCLK        : out std_logic;-- Synchronous clock at the g_freq_SCLK_KHZ
41          SCLK_rise   : out std_logic;-- one cycle signal of the rising edge of SCLK
42          SCLK_fall   : out std_logic -- one cycle signal of the falling edge of SCLK
43      );
44  end fsm_sclk;
45
46  architecture behavioural of fsm_sclk is
47
48      -- Tenéis que hacer la conversion necesaria para sacar la constante que indique
49      --  el número de ciclos para medio periodo del SCLK. Debéis usar floor
50      -- para el redondeo (que opera con reales)
51
52      --1 Hz = 1 ciclo
53      -- f=(1/periodo)   periodo=(1/f)
54
55      constant c_cycles : integer :=integer((1/g_freq_SCLK_KHZ)*g_system_clock); --Los ciclos serían el tiempo de este módulo * la frequencia del sistema total
56
57
58      constant c_half_T_SCLK : integer := integer((1/g_freq_SCLK_KHZ)*g_system_clock); --constant value to compare and generate the rising/falling edge
59      constant c_counter_width :integer :=integer(floor(log2(real(c_cycles)))); -- the width of the counter, take as reference the debouncer
60
61      type state_type is (sclk0,sclk1);                       --Estados  (respectivamente sclk=0 y sclk=1)
62      signal current_state,next_state: state_type;       --Cambiar estados
63      signal CNT: unsigned(c_counter_width-1 downto 0);  --Contador
64      signal time_elapsed: std_logic;                    --Ha pasado el tiempo
65
```

Despues tenemos los dos siguientes process: el del clk con con el contador y otro tambien de clk pero para poder cambiar de estados.

```
begin


process (clk,rst_n) begin --process contador
   if(rst_n='0') then
    CNT<=(others=>'0');
    time_elapsed<='0';
   elsif(rising_edge(clk))then
       if(CNT<to_unsigned(c_half_T_SCLK,CNT'length))then
           CNT<=CNT+1;
       else
           time_elapsed<='1';
           CNT<=(others=>'0');
       end if;
    end if;

end process;

process (clk,rst_n) begin --Registro de siguiente estado
if(rst_n='0')then
   current_state<=sclk0;
else
    current_state<=next_state;
end if;

end process;
```

Tras estos llega el process final que nos dice el comportamiento en cada estado.
He supuesto que si alguna de las señales sclk está activada la otra no puede estarlo; no puede haber un flanco de subida y bajada al mismo tiempo.

```
process (current_state)begin
   case current_state is

   when sclk0=>
       if(time_elapsed<='0')then
           next_state<=sclk0;
           sclk_fall<='0';
       elsif(time_elapsed<='1')then
           next_state<=sclk1;
           sclk_rise<='1';
       end if;
   when sclk1=>
       if(time_elapsed<='0')then
           next_state<=sclk1;
           sclk_rise<='0';
       elsif(time_elapsed<='1')then
           next_state<=sclk0;
           sclk_fall<='1';
       end if;

   when others=>
       next_state<=sclk0;
       sclk_fall<='0';
       sclk_rise<='0';
   end case;
   --Yo entiendo que pese a que no lo ponga en el pdf cuando hay un fall el rise tiene que acabar y viceversa
   --Si no estarían ambas a 1 de manera constante rise y fall de sclk
end process;
```

Registro de desplazamiento con carga en paralelo

La entidad en si no es muy complicada se explica basicamente con la imagen proporcionada en la práctica. He decidido que el generico sea de 8 bits porque en la memoria “palabras de bits”; entiendo que se refiere a de 8 bits. En cualquier caso en caso de necesitar otro tamaño simplemente se puede cambiar el número tras el integer de g_n, ya que din y dout dependen de este g_n.

```
entity Shift_register is
    generic(g_n:integer:=8
    );
    port ( clk: in std_logic;
           rst_n: in std_logic;
           s0,s1: in std_logic;
           din: in std_logic_vector(g_n-1 downto 0);
           dout: out std_logic_vector(g_n-1 downto 0)
           );
end Shift_register;
```

La arquitectura requiere que nos declaremos una variable para poder hacerle cambios sin problema durante el único process. Esta variable también tiene el la anchura igual que los din y dout.
En el reset “limpiamos” o ponemos todo a 0’s; luego los casos son los expuestos en la memoria.

```
architecture Behavioral of Shift_register is
    begin

    process (clk,rst_n)

        variable reg: std_logic_vector(g_n-1 downto 0) := (others =>'0');
        begin

            if(rst_n='0')then
               reg:=(others =>'0');
            elsif(rising_edge(clk))then
                if(s0='0' and s1='0')then       --no cambios
                    reg:=reg;

                elsif(s0='0' and s1='1') then   --rotar a derecha
                    reg:=reg(g_n-1 downto 1);

                elsif(s0='1' and s1='0') then   --rotar a izquierda
                    reg:=reg(g_n-2 downto 0);

                elsif(s0='1' and s1='1') then   --cargar en paralelo
                    reg:=din;

                end if;

            end if;
            dout<=reg;
    end process;

end Behavioral;
```

MEMS

No ha sido posible terminar esta entidad a tiempo, y en estos momentos tiene errores de sintaxis y compilación.
MEMS es un entidad estructural, es decir toma los otros archivos mencionados y los usa como componentes.
Nuestra idea es que la entidad es de este estilo

```
entity MEMS is
    port (clk: in std_logic;
          lrsel: in std_logic;
          data: out std_logic
          );
end MEMS;
```

Como la mostrada en el diagrama, faltaría añadirle los genericos y constantes usados en las otras entidades para poder hacer un port correcto.

A parte de esto declaramos los componentes.

```
architecture structural of MEMS is

component Shift_register is
    generic(g_n:integer:=8
    );
    port ( clk: in std_logic;
           rst_n: in std_logic;
           s0,s1: in std_logic;
           din: in std_logic_vector(g_n-1 downto 0);
           dout: out std_logic_vector(g_n-1 downto 0)
           );
end component;

component fsm_sclk is
    generic(
        g_freq_SCLK_KHZ : integer := 1_500; -- Frequency in KHz of the
                                            --synchronous generated clk
        g_system_clock  : integer := 100_000 --Frequency in KHz of the system clk
    );
    port (
        rst_n       : in std_logic; -- asynchronous reset, low active
        clk         : in std_logic; -- system clk
        start       : in std_logic; -- signal to start the synchronous clk
        SCLK        : out std_logic;-- Synchronous clock at the g_freq_SCLK_KHZ
        SCLK_rise   : out std_logic;-- one cycle signal of the rising edge of SCLK
        SCLK_fall   : out std_logic -- one cycle signal of the falling edge of SCLK
    );
end component;
```

Y el resto que no se ha terminado y tiene errores sería; portear y describir el diseño de este source

Constraints

Hemos descomentado las líneas y porteado las conexiones a las señales correctas respecto al fichero de constraints

```
##Omnidirectional Microphone

set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports { CLK }]; #IO_25_35 Sch=m_clk
set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports { DATA }]; #IO_L24N_T3_35 Sch=m_data
set_property -dict { PACKAGE_PIN F5    IOSTANDARD LVCMOS33 } [get_ports { LRSEL }]; #IO_0_35 Sch=m_lrsel
```

Comentarios/Observaciones

Obviamente no he podido trabajar en esta práctica como en las otras por la falta de tiempo y disposicion a trabajar de mi compañero de equipo.

Práctica 3. Verificación VHDL

Testbench (tb_top1_csv)

Estas son las librerías que necesitamos para leer archivos y la entidad vacia habitual de los testbenchs

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;


entity tb_top1_csv is
end tb_top1_csv;
```

El resto simplemente se ha realizado de acuerdo a los pasos de la guia 3 de vhdl provista en el campus.
Declarar las signals para portearlas al UUT y luego ya un process para leer el fichero csv que se integrado como source

```
architecture testBench of tb_top1_csv is

    --File handler
    file file_input : text;
    -- ports for the top
    signal rst_n:      std_logic; -- Reset
    signal clk100Mhz:  std_logic; -- Reloj
    signal BTNC:       std_logic; -- Boton
    signal LED:        std_logic; -- LED

    component top_practical is
      port (
          rst_n         : in std_logic;
          clk100Mhz     : in std_logic;
          BTNC           : in std_logic;
          LED           : out std_logic
      );
    end component;

    constant freq : integer := 100_000; --KHZ
    constant clk_period : time := (1 ms/ freq);
begin

    --Instanciar top
    UUT: top_practical
        port map (
          rst_n     => rst_n,
          clk100Mhz => clk100Mhz,
          BTNC       => BTNC,
          LED       => LED
        );
        clk100Mhz <= not clk100Mhz after clk_period/2;
proc_sequencer : process
    -- Process to read the data
        file text_file :text open read_mode is "inputs.csv";
        variable text_line : line; -- Current line
        variable ok: boolean; -- Saves the status of the operation of reading
        variable char : character; -- Read each character of the line(used when using comments)
        variable delay: time; -- Saves the desired delay time
        variable data: std_logic; --Generates a variable of the first operand rst_n type (which is the same for the rest)
        variable expected_led: std_logic;
```

Tras esto vendría la parte del process donde se le dice que satarse del inputs.csv y que leer

```
begin

    while not endfile(text_file) loop
        readline(text_file, text_line);
            -- Skip empty lines and commented lines
            if text_line.all'length = 0 or text_line.all(1) = '#' then
                next;
            end if;
        -- Read the delay time
        read(text_line, delay, ok);
        assert ok
            report "Read 'delay' failed for line: " & text_line.all
            severity failure;

        -- Read first operand (rst_n)
        read(text_line, data, ok);
        assert ok
            report "Read 'rst_n' failed for line: " & text_line.all
            severity failure;
        rst_n <= data;

         -- Read first operand (BTNC)
        read(text_line, data, ok);
        assert ok
            report "Read 'A' failed for line: " & text_line.all
            severity failure;
        BTNC <= data;

        -- Read the second operand (Expected LED value)
        read(text_line, data, ok);
        assert ok
            report "Read 'LED' failed for line: " & text_line.all
            severity failure;
            LED <= data;

            -- Wait for the delay
            wait for delay;
```

Tras leer el fichero se espera el delay, y se hace un assert del led y led esperado para comprobar el resultado.

```
    --Verify the LED_expected is correct
    expected_led := (rst_n and BTNC);


    assert expected_led = LED
        report "Unexpected result: " &
            "rst_n = "& std_logic'image(rst_n) & "; "&
            "clk100Mhz = "& std_logic'image(clk100Mhz) & ";
            "BTNC = "& std_logic'image(BTNC) & "; "&
            "LED = " & std_logic'image(LED)
        severity error;


end loop;
report "Finished" severity FAILURE;
```

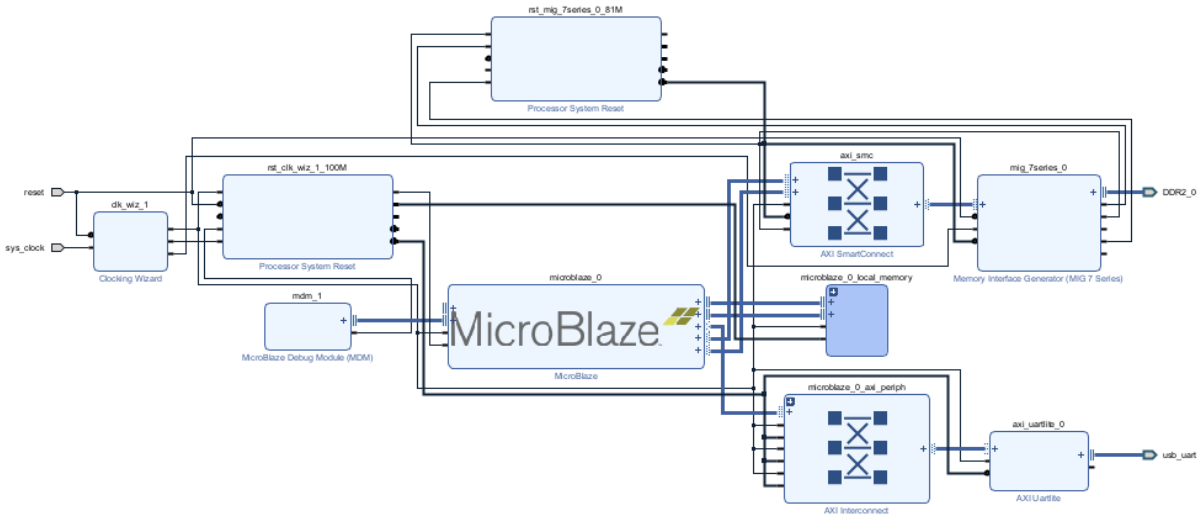
Comentarios

De nuevo debido a la reiterada escasez de tiempo, no ha habido tiempo para hacer un testbench satisfactorio.

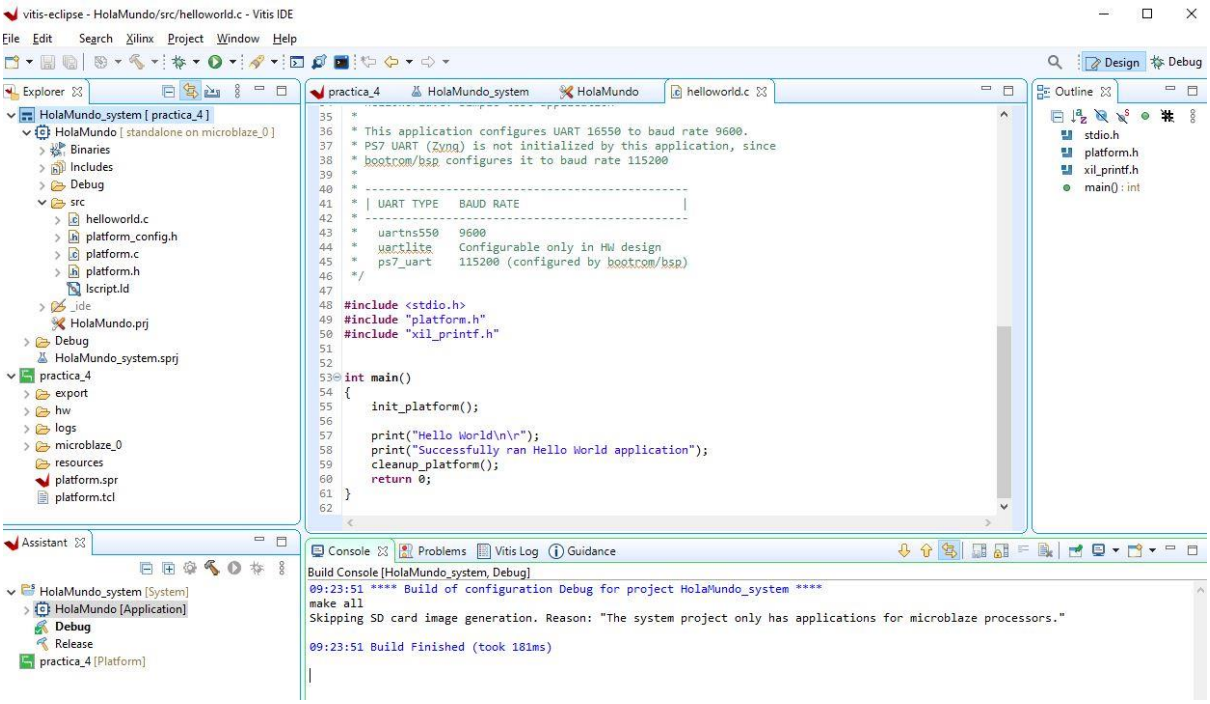
Práctica 4. Softcore

Esta práctica consiste en realizar un proyecto en vitis y ponerlo en la placa.
Como no disponemos de ningún portátil que pueda tener vitis y vivado y funcionar al mismo tiempo, solo puedo argumentar que hemos seguido los pasos de la práctica guiada y que por tanto debería de funcionar.

El diagrama de bloques generado queda de la siguiente manera:



Al exportar este proyecto a vitis y compilar con un hola mundo, nos sale que la compilación es correcta.

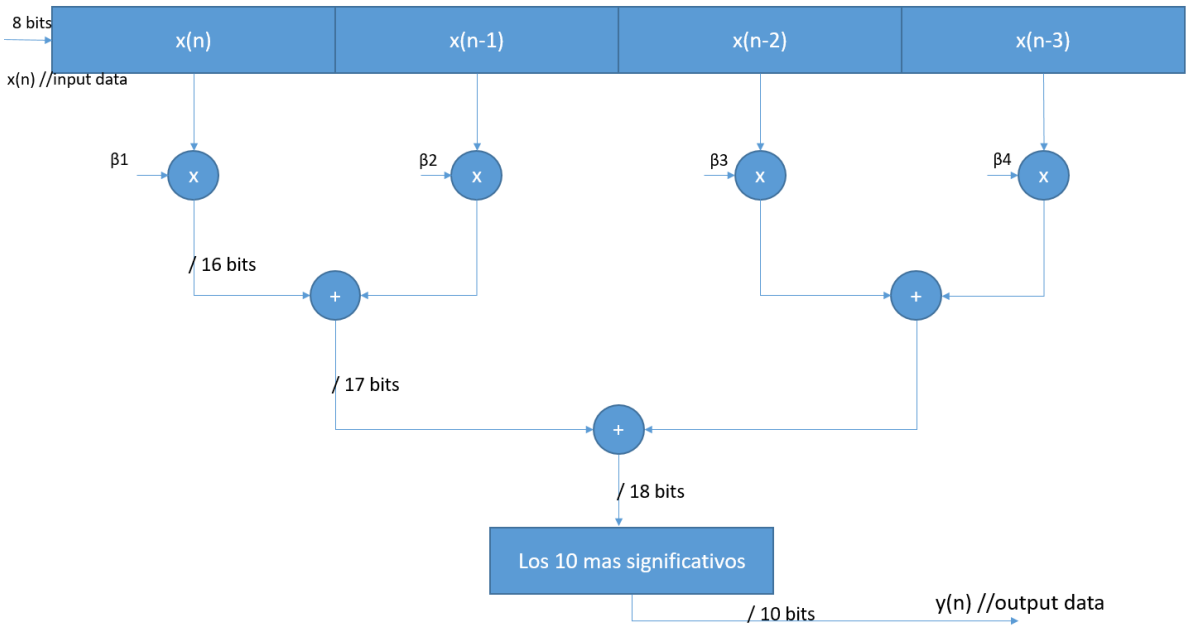


Comentarios

Debido al no disponer de un ordenador, que pueda arrancar vivado y vitis carezco de la capacidad de comprobar los resultados reales en una placa pero, he comentado paso a paso lo que he hecho con un par de compañeros y debería de ejecutarse de manera correcta.

Práctica 5. Filtro Digital

A la hora de crear un filtro digital solo necesitaremos una source, la cual hemos llamado fir. Pero antes de empezar a programar nos hemos puesto de acuerdo en como hacerlo. También seguiremos la recomendación del profesor de usar arrays para poder hacer las sumas y multiplicaciones con mas tranquilidad
Nuestro diagrama a la hora de realizar las operaciones será el siguiente:



Intentamos hacerlo lo más paralelizable posible. Por tanto siguiendo el diagrama vamos a necesitar vamos a necesitar 8 vectores de tamaño 8 para los coeficientes (betas) y para los datos de i_input (x(n)). Vamos a necesitar otros cuatro arrays de tamaño 16 para las multiplicaciones. Y finalmente vamos a necesitar dos arrays para las sumas uno de 17 para las “sumas intermedias”, y otro de 18 para la suma final. Justo después vamos a hacer un resize del array de 18 para quedarnos con los 10 más significativos y que este sea el valor de o_output (y(n)).

Primero la entidad y declaración de variables, antes del begin, que ya hemos comentado

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fir_filter is
port (
    clk     :in std_logic;
    rst     :in std_logic;
    -- Coeficientes
    beta1   :in std_logic_vector(7 downto 0);
    beta2   :in std_logic_vector(7 downto 0);
    beta3   :in std_logic_vector(7 downto 0);
    beta4   :in std_logic_vector(7 downto 0);
    -- Data input 8 bit
    i_data  :in std_logic_vector(7 downto 0);
    -- Filtered data
    o_data  :out std_logic_vector(9 downto 0)
    );
end fir_filter;

architecture Behavioral of fir_filter is

type t_data_pipe      is array (0 to 3) of signed(7  downto 0); --Array de 4 para
type t_coeff          is array (0 to 3) of signed(7  downto 0); --Array de 4 para los coeficientes
type t_mult           is array (0 to 3) of signed(15    downto 0); --Array de 4 para las multiplicaciones
type t_add_st0        is array (0 to 1) of signed(15+1  downto 0); --Array de 2 para la suma de los dos datos
--Signals porque los datos van a cambiar todo el rato
signal r_coeff              : t_coeff ;  --Variable para almacenar los coeficientes
signal p_data               : t_data_pipe; --Variable para almacenar los datos de inputs, los xn y x(n-1)
signal r_mult               : t_mult; -- Resultado de la multiplicación
signal r_add_st0            : t_add_st0; --Resultado de las dos primeras sumas
signal r_add_st1            : signed(15+2  downto 0); --Resultado de la ultima suma

begin
```

Como todos los process no caben en un mismo recorte vamos poco a poco. Los process que haremos serán: asignación de input y coeficientes, multiplicación, sumas intermedias, suma final y resize.
Todos los process dependen de rst y clk, con un reset asíncrono activo a nivel bajo (hace reset si está a 0). El reset en cada uno de los process pone las variables llenas a 0.

En el input en caso de que no haya reset simplemente asignamos a nuestras signals que son arrays, los valores correspondientes.

Mientras que en el process de multiplicación ponemos en nuestro array de resultados de multiplicación, el proceso de multiplicar los coeficientes por los datos.

```
p_input : process (rst,clk)
begin

  if(rst='0') then
    p_data       <= (others=>(others=>'0'));
    r_coeff      <= (others=>(others=>'0'));
  elsif(rising_edge(clk)) then --Pasamos a signed los coeficientes para hacer la multiplicación mas sencilla
    p_data      <= signed(i_data)&p_data(0 to p_data'length-2);
    r_coeff(0)  <= signed(beta1);
    r_coeff(1)  <= signed(beta2);
    r_coeff(2)  <= signed(beta3);
    r_coeff(3)  <= signed(beta4);
  end if;
end process p_input;

p_mult : process (rst,clk) --Proceso para hacer la multiplicación
begin

  if(rst='0') then
    r_mult <= (others=>(others=>'0'));
  elsif(rising_edge(clk)) then
    for k in 0 to 3 loop
      r_mult(k) <= p_data(k) * r_coeff(k); --Multiplicación de cada input por su correspondiente beta
    end loop;
  end if;
end process p_mult;
```

El siguiente es el process de las “sumas intermedias”; lo que hacemos es un for que suma los resultados de la multiplicación de manera que sea: resultado 1 + resultado 2 y resultado 3 + resultado 4; haciéndoles a todos estos un resize para que sean de tamaño 17.

Después el process de la “suma final” es una suma sin necesidad de for porque solo hay dos cosas que sumar, las dos “sumas intermedias” y estas también sufren un resize pero esta vez a tamaño 18.

Finalmente el process de asignar el output. Aquí asignamos el valor de la “suma final” a el output, pero como solo queremos los 10 bits mas significativos lo que hacemos es asignar como un vector nuestro array pero desde el bit 17 al bit 8.

Tras esto cerramos la arquitectura Behavioral

```
p_add_st0 : process (rst,clk) --Proceso para sumar (xn b1 y x[n-1] b2) y (x[n-2] b3 y x[n-3] b4)
begin

  if(rst='0') then
    r_add_st0     <= (others=>(others=>'0'));
  elsif(rising_edge(clk)) then --loop que suma las salidas (1 y 2) y (3 y 4)
    for k in 0 to 1 loop
      r_add_st0(k)     <= resize(r_mult(2*k),17)  + resize(r_mult(2*k+1),17);
    end loop;
  end if;
end process p_add_st0;

p_add_st1 : process (rst,clk) --Proceso para sumar los dos operandos que quedan
begin

  if(rst='0') then
    r_add_st1     <= (others=>'0');
  elsif(rising_edge(clk)) then
    r_add_st1     <= resize(r_add_st0(0),18)  + resize(r_add_st0(1),18); --Suma los resultados anteriores
  end if;
end process p_add_st1;

p_output : process (rst,clk) --Proceso para igualar el output
begin

  if(rst='0') then
    o_data     <= (others=>'0');
  elsif(rising_edge(clk)) then
    o_data     <= std_logic_vector(r_add_st1(17 downto 8)); --Cojemos los 10 primeros bits del resultado
  end if;
end process p_output;

end Behavioral;
```

Comentarios

Me resulta un poco tarde cambiar la estructura ya, pero, si no me equivoco con los types y los arrays; el r_coeff podria ser del mismo tipo que p_data, de manera que podriamos ahorrarnos la declaración de un tipo especifico de array.