

Management and analysis of physics datasets, Part. 1

Second Laboratory

Antonio Bergnoli bergnoli@pd.infn.it - Filippo Marini filippo.marini@pd.infn.it

24/11/2020

Laboratory Introduction

- Recap of the last lecture
- make an overview of the synthesis
- understand in depth hierarchical design
- introduce sequential circuits
- study some combinatorial and sequential circuits

1. **case when**
2. **type** definition
3. **Synchronous processes**
4. other uses of **wait**
5. **if then else**

Web resources

1. [https : //www.edaplayground.com/](https://www.edaplayground.com/)
2. [https : //vhdlwhiz.com/](https://vhdlwhiz.com/)
3. [https : //surf-vhdl.com/](https://surf-vhdl.com/)

Free Books and Handbooks

4. VHDL Handbook **old but still good!!**
5. Free Range VHDL

a smart way to build larger simulation projects

```
student@MAPD-machine : ~$ ghdl -i heartbeat.vhd # include source file in the project
student@MAPD-machine : ~$ ghdl -i heartbeat_top.vhd # include source file in the project
student@MAPD-machine : ~$ ghdl -d # check the working directory

# Library work
# Directory :
entity heartbeat
architecture behaviour of heartbeat
entity heartbeat_top
architecture str of heartbeat_top

student@MAPD-machine : ~$ ghdl -m heartbeat_top # make the selected entity (usually the top)
analyze heartbeat_top.vhd
analyze heartbeat.vhd
elaborate heartbeat_top

student@MAPD-machine : ~$ ghdl -r heartbeat_top --wave=wave.ghw --stop-time=1us # run the simulation

./heartbeat_top : info : simulation stopped by --stop-time @1us

student@MAPD-machine : ~$ gtkwave wave.ghw #inspect the result (waveform)
```

Overview of Synthesis (some examples just to have an idea of the complexity)

to create a logic circuit starting from a source code description

1. not every VHDL is synthesizable
2. a synthesizer tool typically creates a logic circuit using elementary “logic cells” e.g: **and, or, not, flip flop**
3. a Simulator tool is **not** a synthesizer.

Simple Xor Gate

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

```
entity xor_port is
```

```
    port (  
        a : in  std_logic;  
        b : in  std_logic;  
        y : out std_logic);
```

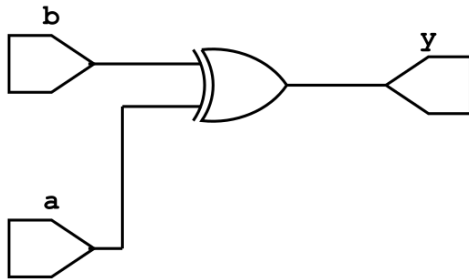
```
end entity xor_port;
```

```
architecture str of xor_port is
```

```
begin  -- architecture str
```

```
    y <= a xor b;
```

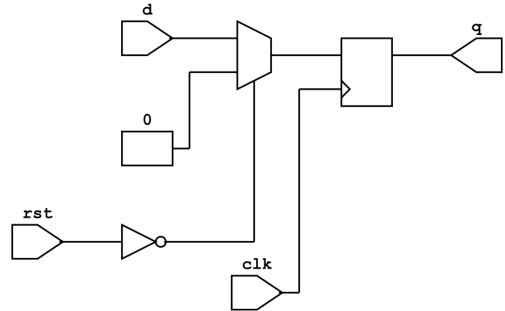
```
end architecture str;
```



simple D-type Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    d    : in  std_logic;
    q    : out std_logic);
end entity dff;
architecture rtl of dff is
begin -- architecture rtl
  flipflop : process (clk) is
  begin -- process flipflop
    if rising_edge(clk) then
      if rst = '0' then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process flipflop;
end architecture rtl;
```

-- rising clock edge



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

entity adder is

    generic (
        width : integer := 8);

    port (
        a      : in  std_logic_vector(width - 1 downto 0);
        b      : in  std_logic_vector(width - 1 downto 0);
        sum    : out std_logic_vector(width - 1 downto 0));

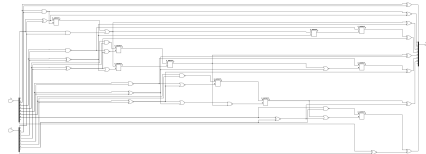
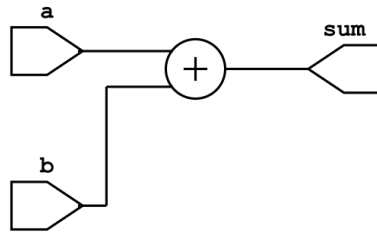
end entity adder;

architecture rtl of adder is

begin -- architecture rtl

    sum <= std_logic_vector(unsigned(a) + unsigned(b));

end architecture rtl;
```



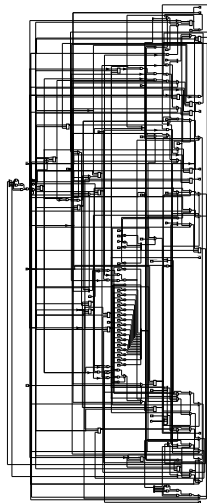
Fir Filter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fir_filter_4 is
  port (
    i_clk      : in  std_logic;
    i_rstb     : in  std_logic;
    i_coeff_0   : in  std_logic_vector(7 downto 0);
    i_coeff_1   : in  std_logic_vector(7 downto 0);
    i_coeff_2   : in  std_logic_vector(7 downto 0);
    i_coeff_3   : in  std_logic_vector(7 downto 0);
    i_data      : in  std_logic_vector(7 downto 0);
    o_data      : out std_logic_vector(9 downto 0));
end fir_filter_4;
architecture rtl of fir_filter_4 is
  type t_data_pipe is array (0 to 3) of signed(7 downto 0);
  type t_coeff is array (0 to 3) of signed(7 downto 0);
  type t_mult is array (0 to 3) of signed(15 downto 0);
  type t_add_st0 is array (0 to 1) of signed(15+1 downto 0);
  signal r_coeff : t_coeff;
  signal p_data : t_data_pipe;
  signal r_mult : t_mult;
  signal r_add_st0 : t_add_st0;
  signal r_add_st1 : signed(15+2 downto 0);
begin
  p_input : process (i_rstb, i_clk)
  begin
    if(i_rstb = '0') then
      p_data <= (others => (others => '0'));
      r_coeff <= (others => (others => '0'));
    elsif(rising_edge(i_clk)) then
      p_data <= signed(i_data)&p_data(0 to p_data'length-2);

```

```
    p_mult : process (i_rstb, i_clk)
    begin
      if(i_rstb = '0') then
        r_mult <= (others => (others => '0'));
      elsif(rising_edge(i_clk)) then
        for k in 0 to 3 loop
          r_mult(k) <= p_data(k) * r_coeff(k);
        end loop;
      end if;
    end process p_mult;
    p_add_st0 : process (i_rstb, i_clk)
    begin
      if(i_rstb = '0') then
        r_add_st0 <= (others => (others => '0'));
      elsif(rising_edge(i_clk)) then
        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 (i_rstb, i_clk)
    begin
      if(i_rstb = '0') then
        r_add_st1 <= (others => '0');
      elsif(rising_edge(i_clk)) then
        r_add_st1 <= resize(r_add_st0(0), 18)
          + resize(r_add_st0(1), 18);
      end if;
    end process p_add_st1;
    p_output : process (i_rstb, i_clk)
    begin

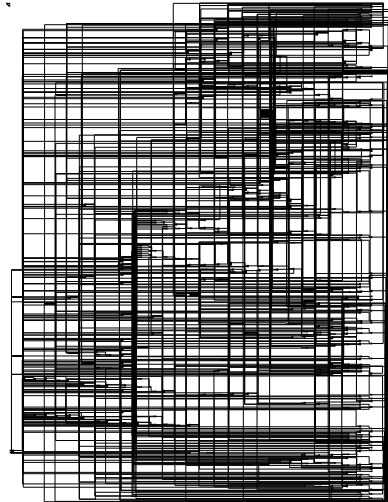
```



Complete I^2Cbus master core

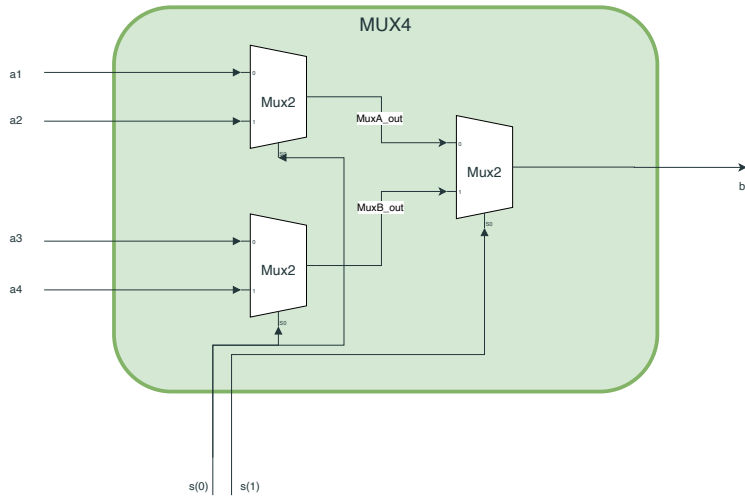
taken from [https : //opencores.org/projects/i2c/](https://opencores.org/projects/i2c/)

1000 VHDL lines of code



Basic combinatorial, hierarchical circuits

Multiplexer (Schematic)



```
MUX2
library IEEE;
use IEEE.std_logic_1164.all;
entity mux2 is
  port(
    a1 : in  std_logic_vector(2 downto 0);
    a2 : in  std_logic_vector(2 downto 0);
    sel : in  std_logic;
    b   : out std_logic_vector(2 downto 0));
end mux2;
architecture rtl of mux2 is
begin
  p_mux : process(a1, a2, sel)
  begin
    case sel is
      when '0'   => b <= a1;
      when '1'   => b <= a2;
      when others => b <= "000";
    end case;
  end process p_mux;
end rtl;
```

```
MUX4
  library IEEE;
  use IEEE.std_logic_1164.all;
  entity mux4 is
    port(
      a1, a2, a3, a4 : in  std_logic_vector(2 downto 0);
      sel             : in  std_logic_vector(1 downto 0);
      b               : out std_logic_vector(2 downto 0));
  end mux4;
  architecture rtl of mux4 is
    component mux2 is
      port (
        a1, a2 : in  std_logic_vector(2 downto 0);
        sel    : in  std_logic;
        b      : out std_logic_vector(2 downto 0));
    end component mux2;
    signal muxA_out, muxB_out : std_logic_vector(2 downto 0);
  begin
    muxA : mux2
      port map ( a1 => a1, a2 => a2,
                 sel => sel(0),
                 b   => muxA_out);

    muxB : mux2
      port map ( a1 => a3, a2 => a4,
                 sel => sel(0),
                 b   => muxB_out);

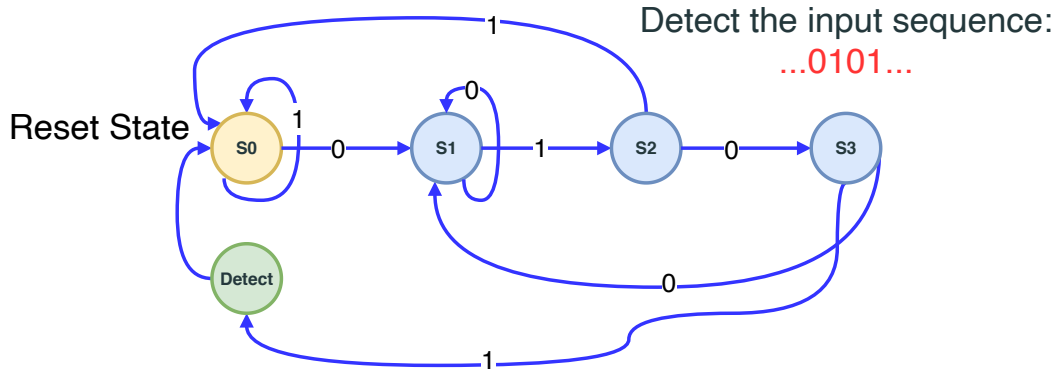
    muxOut : mux2
      port map ( a1 => muxA_out, a2 => muxB_out,
                 sel => sel(1),
                 b   => b);
  end rtl;
```


Basic Sequential circuits

The simplest one : D-type Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    d   : in  std_logic;
    q   : out std_logic);
end entity dff;
architecture rtl of dff is
begin -- architecture rtl
  flipflop : process (clk) is
  begin -- process flipflop
    if rising_edge(clk) then -- rising clock edge
      if rst = '0' then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process flipflop;
end architecture rtl;
```

A simple state machine (state diagram)



A simple state machine

```
library ieee;
use ieee.std_logic_1164.all;
entity patterndetect is
  port (
    a  : in  std_logic;
    clk : in  std_logic;
    rst : in  std_logic;
    y   : out std_logic;
  )
end entity patterndetect;
architecture rtl of patterndetect is
  type state_t is (S0, S1, S2, S3, Detect);
  signal state : state_t := S0;
begin -- architecture rtl
  main : process (clk) is
    begin -- process main
      if rising_edge(clk) then
        if rst = '0' then
          state <= S0;
          y <= '0';
        else
          case state is
            when S0 =>
              y <= '0';
              if a = '0' then
                state <= S1;
              end if;
            when S1 =>
              y <= '0';
              if a = '0' then
                state <= S1;
              elsif a = '1' then
                state <= S2;
              end if;
            when S2 =>
              y <= '0';
              if a = '0' then
                state <= S3;
              elsif a = '1' then
                state <= S0;
              else
                null;
              end if;
            when S3 =>
              y <= '0';
              if a = '0' then
                state <= S1;
              elsif a = '1' then
                state <= Detect;
              else
                null;
              end if;
            when Detect =>
              y <= '1';
              state <= S0;
            when others => null;
          end case;
        end if;
      end if;
    end process main;
  end architecture rtl;
```

```
when S2 =>
  y <= '0';
  if a = '0' then
    state <= S3;
  elsif a = '1' then
    state <= S0;
  else
    null;
  end if;
when S3 =>
  y <= '0';
  if a = '0' then
    state <= S1;
  elsif a = '1' then
    state <= Detect;
  else
    null;
  end if;
when Detect =>
  y <= '1';
  state <= S0;
when others => null;
end case;
end if;
end if;
end process main;
end architecture rtl;
```

A simple state machine (testbench)

```
library ieee;
use ieee.std_logic_1164.all;
entity patterndetect_tb is
end entity patterndetect_tb;
```

```
architecture test of patterndetect_tb is
    signal a    : std_logic;
    signal clk   : std_logic := '0';
    signal rst   : std_logic;
    signal y     : std_logic;
begin -- architecture test
    DUT : entity work.patterndetect
        port map (
            a => a,
            clk => clk,
            rst => rst,
            y => y);
    clk <= not clk after 2 ns;
    WaveGen_Proc : process
    begin
        a <= '0';
        rst <= '1'; wait for 10 ns;
        rst <= '0'; wait for 10 ns;
        rst <= '1';
        wait until rising_edge(clk);
        a <= '0';
        wait until rising_edge(clk);
        a <= '1';
        wait until rising_edge(clk);
        a <= '0';
        wait until rising_edge(clk);
        a <= '1';
        wait for 100 ns;
        wait;
    end process WaveGen_Proc;
end architecture test;
```

Homework

- build another state machine and try to model it in VHDL;
- build a testbench for it.