Management and analysis of physics datasets, Part. 1

Third Laboratory

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

25/11/2020

Laboratory Introduction

Goals

Mainly a recap of the last lecture

- GHDL for larger projects
- Introduce processes
- introduce sequential circuits and sensitivity list

GHDL for Larger Projects

GHDL for Larger 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 (usuallly 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)
```

Multiplexer, doubts?

Multiplexer, doubts?

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;
```

Sequential circuits

Processes

- Statements are executed one after the other sequentially, in order
- Processes are used to describe sequential behavior
- All processes in an architecture behave concurrently

Process Statements

The process includes the “sensitivity list”, which is a set of signals on which any change triggers the process itself

- Process runs when any of the signals in the sensitivity list changes
- A process with a sensitivity list must not contain any “wait” statement
- The sensitivity list can only include signals and input ports
- The execution of a process consists in the execution of its whole sequence of statements

```
library ieee;
use ieee.std_logic_1164.all;
entity process_ex is
  port (
    a : in  std_logic;
    b : in  std_logic;
    c : in  std_logic;
    y : out std_logic
    );
end entity process_ex;
architecture rtl of process_ex is
  signal x : std_logic;
begin  -- architecture rtl
  process (a, b, c, x)
  begin
    x <= (a and b) or c;
    y <= x;
  end process;
end architecture rtl;
```

Sequential Clocked Process

- Generates synchronous logic circuits
- All signals are evaluated at the rising edge of the clock, no need to add other signals on the sensitivity list

```
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;
```

Classwork

Suggested excercise

- Build a testbench for the D Flip-Flop
- Design the architecture of a Toggle Flip-Flop
- Build a testbench to check its behavior

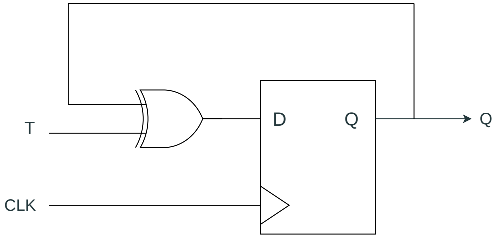
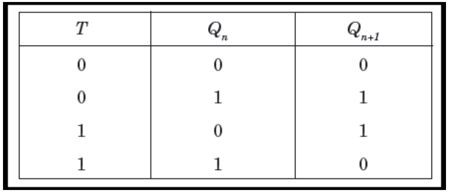


Figure 1: Toggle Flip-Flop Model



T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2: Toggle Flip-Flop Truth Table