Simulations with VHDL
Simulations with VHDL

• Testbench
• Analog vs. digital simulation
• VHDL for simulation
  – Simple simulation example
  – \texttt{wait} in \texttt{process} for simulations
  – Delaying signals (\texttt{after}, \texttt{'delayed})
  – Text I/O
  – Reporting - \texttt{assert}
  – Advanced simulation example
  – Recommended directory structure and example of Makefile for ModelSim
  – The free simulator GHDL
How to simulate – Testbench

- Instantiate the design under test (DUT) into the so called testbench
- All signals to the DUT are driven by the testbench, all outputs of the DUT are read by the testbench and if possible analyzed

- Some subset of all signals at all hierarchy levels can be shown as a waveform
- The simulation is made many times at different design stages – functional, after the synthesis, after the placing and routing, sometimes together with the other chips on the board
- Many VHDL constructs used in a testbench can not be synthesized, or are just ignored when trying to make a synthesis
Analog vs. digital simulation

Pspice Simulation

Digital Simulation (ModelSim)

© V. Angelov
Simple test bench example

```vhdl
entity tb_proc1 is
end tb_proc1;

architecture tb of tb_proc1 is
    component proc1 is
        port(
            clk : in std_logic;
            rst_n : in std_logic;
            en_n : in std_logic;
            d : in std_logic;
            q : out std_logic);
    end component;

    signal rst_n : std_logic;
    signal d : std_logic;
    signal en_n : std_logic;
    signal clk : std_logic:= '1';

begin
    clk <= not clk after 50 ns;

    rst_n <= '1' after 0 ns,
            '0' after 300 ns,
            '1' after 400 ns;

    d <= '0' after 0 ns,
         '1' after 300 ns,
         '0' after 600 ns;

    en_n <= '0';

    u1: proc1
    port map(
        clk => clk,
        rst_n => rst_n,
        en_n => en_n,
        d => d,
        q => q);
end;
```

Initial value – only in simulations!

Component declaration

Component instantiation

Clock signal

Stimuli

No ports

© V. Angelov
wait for simulations

process
begin
  wait on clk;  \iff \text{wait until clk'event};
  ...
  wait on clk, reset;
  ...
  wait until clk'event \text{ and clk='1'};  \text{edge detection}
  ...
  wait until d(0)='1';
  ...
  wait for 20 ns;  \text{time}
  ...
  wait until d(4)='0' \text{ for 200 ns};
  ...
  wait;  \text{until the boolean expression is TRUE}
  ...
end process;  \text{for ever, suspend the process}
Delaying signals

• There are two possible delay models: **inertial** (if not specified) and **transport**
• In the **inertial** model the pulses with width below the delay are rejected, but the width can be specified independently with **reject**

Examples:

```vhdl
constant T : time := 2 ns;

... 

d(0) <= transport pulse after 2.5*T;
d(1) <= inertial pulse after 2.5*T;

d(2) <= reject 1.25*T inertial pulse after 2.5*T;
d(3) <= pulse'delayed(2.5*T); -- uses transport model
```

<table>
<thead>
<tr>
<th>Pulse Type</th>
<th>Time</th>
<th>Signal After Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td></td>
<td>d(0)</td>
</tr>
<tr>
<td>Inertial</td>
<td>2.5T</td>
<td>d(1)</td>
</tr>
<tr>
<td>Reject</td>
<td>1.25T</td>
<td>d(2)</td>
</tr>
<tr>
<td>&quot;delayed&quot;</td>
<td></td>
<td>d(3)</td>
</tr>
</tbody>
</table>

© V. Angelov
Text out in VHDL simulation (example)

```vhdl
use std.textio.all;
use IEEE.std_logic_textio.all;

file outfile_wr : TEXT open write_mode is "dm.log";
signal timecnt : Integer;

process(clk)
  variable outline : line;
  begin
    if clk'event and clk='1' then
      if WE = '1' then
        WRITE(outline, timecnt);
        WRITE(outline, string'(" 0x"));
        hwrite(outline, addr);
        WRITE(outline, string'(" 0x"));
        hwrite(outline, din);
        WRITELINE(outfile_wr, outline);
      end if;
    end if;
  end process;
end;
```

logging in a text file of all memory writes

<table>
<thead>
<tr>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
</tr>
<tr>
<td>4</td>
<td>0x04</td>
</tr>
<tr>
<td>5</td>
<td>0xFF</td>
</tr>
<tr>
<td>6</td>
<td>0x00FF</td>
</tr>
</tbody>
</table>

package to work with text files

`addr(7:0)`

`clk`

`din(15:0)`

`rst`

`we`
process
  variable s : line;
  variable goodw, gooda, goode : Boolean;
  variable addrv : std_logic_vector(addr'range);
  variable wev : std_logic;
  variable oev : std_logic;
begin
  if endfile(infile) then
    wait;  -- stop reading
  end if;
  readline(infile, s);
  if s(s'low) /= '#' then
    read(s, wev, goodw);
    read(s, oev, goode);
    read(s, addrv, gooda);
    if gooda and goodw and goode then
      addr <= addrv;
      we <= wev;
      oe <= oev;
      wait until falling_edge(clk);
    else
      wait;  -- stop reading
    end if;
  end if;
end process;
function is_1_or_0(src : std_logic_vector) return boolean is
begin
    for i in src'range loop
        if src(i) /= '0' and src(i) /= '1' then
            return false;
        end if;
    end loop;
    return true;
end is_1_or_0;

ram_proc: process(clk)
begin
    ...
    if we = '1' then
        -- synthesis off
        assert is_1_or_0(addr)
        report "Attempt to write with undefined address"
        severity WARNING;
        -- synthesis on
        mem_data(conv_integer(addr))<= din;
    ...
    Condition; if not fulfilled, the message after report will be printed

    The synthesis tools generally ignore the assert, but this can be specified explicitly

In the simulator it is possible to mask/show the messages or to break the simulation, depending on the severity (NOTE, WARNING, ERROR, FAILURE)

Check if all bits are good: '1' or '0'
Simulation of the registerfile(1)

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
use std.textio.all;
use IEEE.std_logic_textio.all;
entity reg_file_tb is
generic 
(Na : Positive := 3;
 Nd : positive := 16);
end reg_file_tb;
architecture sim of reg_file_tb is
component reg_file is
generic 
(Na : Positive := 3;
 Nd : positive := 16);
port(
 clk : in std_logic;
 rst_n : in std_logic;
 we : in std_logic;
 waddr : in std_logic_vector(Na-1 downto 0);
 din : in std_logic_vector(Nd-1 downto 0);
 raddra : in std_logic_vector(Na-1 downto 0);
 raddrb : in std_logic_vector(Na-1 downto 0);
 rdata : out std_logic_vector(Nd-1 downto 0);
 rdatb : out std_logic_vector(Nd-1 downto 0))
end component;

procedure check_read(
 signal clk : in std_logic;
 signal dout : in std_logic_vector(Nd-1 downto 0);
 variable dexp : in std_logic_vector(Nd-1 downto 0)) is

use std.textio.all;
use IEEE.std_logic_textio.all;

entity reg_file_tb is
generic 
(Na : Positive := 3;
 Nd : positive := 16);
end reg_file_tb;
architecture sim of reg_file_tb is
component reg_file is
generic 
(Na : Positive := 3;
 Nd : positive := 16);
port(
 clk : in std_logic;
 rst_n : in std_logic;
 we : in std_logic;
 waddr : in std_logic_vector(Na-1 downto 0);
 din : in std_logic_vector(Nd-1 downto 0);
 raddra : in std_logic_vector(Na-1 downto 0);
 raddrb : in std_logic_vector(Na-1 downto 0);
 rdata : out std_logic_vector(Nd-1 downto 0);
 rdatb : out std_logic_vector(Nd-1 downto 0))
end component;

This procedure is used to check the read data and to report any errors.
Simulation of the registerfile (2)

type rftype is array(0 to 2**Na-1) of  
  std_logic_vector(Nd-1 downto 0);

signal din : std_logic_vector(Nd-1 downto 0);
signal waddr : std_logic_vector(Na-1 downto 0);
signal raddra: std_logic_vector(Na-1 downto 0);
signal raddrb: std_logic_vector(Na-1 downto 0);
signal rdata : std_logic_vector(Nd-1 downto 0);
signal rdatb : std_logic_vector(Nd-1 downto 0);
signal rst_n : std_logic;
signal clk : std_logic:= '1';
signal we : std_logic;

begin
  clk <= not clk after 50 ns;
rf: reg_file
  generic map(Na => Na,
              Nd => Nd)
  port map(
    clk => clk,
    rst_n => rst_n,
    we => we,
    waddr => waddr,
    din => din,
    raddra => raddra,
    raddrb => raddrb,
    rdata => rdata,
    rdatb => rdatb);

process
  variable rfile : rftype;
begin
  rst_n <= '0';
  raddra <= (others => '0');
  raddrb <= (others => '0');
  wait until falling_edge(clk);
  wait until falling_edge(clk);
  rst_n <= '1';
  we <= '1';
  for i in rftype’range loop
    waddr <= conv_std_logic_vector(i, waddr'length);
    din <= conv_std_logic_vector(i + 16*(i+1),
                                 din'length);
    wait until falling_edge(clk);
    rfile(i) := din;
  end loop;
)}
Simulation of the registerfile(3)

we <= '0';
rfile(5) := (others => '0'); -- emulate error, delete later
for i in rftype'range loop
  raddra <= conv_std_logic_vector(i, raddra'length);
  check_read(clk, rdata, rfile(i));
end loop;
for i in rftype'range loop
  raddrb <= conv_std_logic_vector(i, raddrb'length);
  check_read(clk, rdatb, rfile(i));
end loop;
wait until falling_edge(clk);
rst_n <= '0';
for i in rftype'range loop
  rfile(i) := (others => '0');
end loop;
wait until falling_edge(clk);
rst_n <= '1';
for i in rftype'range loop
  raddra <= conv_std_logic_vector(i, raddra'length);
  check_read(clk, rdata, rfile(i));
end loop;
for i in rftype'range loop
  raddrb <= conv_std_logic_vector(i, raddrb'length);
  check_read(clk, rdatb, rfile(i));
end loop;
wait;
end process;
end;

Read all from port A, then from port B, the procedure check_read waits for one clock period and reports any errors observed.

Read again after the reset to check if all registers are cleared.
Functional simulation – directory structure

- Store the project files in a clear structure
- Do not mix your sources with any other files created by some simulation or synthesis tool
- Try to use scripts or Makefile(s), instead of GUI

This is only an example! It will be extended later

<project directory>

```
  SIM
  ├── SRC
  │    └── functional
  │         └── work
  │              └── SRC
```

- SIM contains all testbench sources of the project
- SRC contains all source files of the project
- functional and work are created by the Make script to store the compiled ModelSim libraries
# The top level design name
design=cnt3
# The source file(s), the last is the top
src_files=../SRC/my_and.vhd ../SRC/my_or.vhd ../SRC/cnt2bit.vhd ../SRC/$(design).vhd
# The testbench file(s), the last is the top
testbench=./SRC/clk_gen.vhd ./SRC/$(design)_tb.vhd
# Compile the sources for functional simulation
functional: $(src_files)
  vlib $@
  vcom -quiet -93 -work $@ $(src_files)
# Functional Simulation
simfun: $(testbench) functional
  vmap libdut functional
  rm -rf work
  vlib work
  vcom -quiet -93 -work work $(testbench)
  vsim 'work.$(design)_tb' -t 1ns -do 'wave_fun.do'
# Clean all library directories
clean:
  rm -rf functional work modelsim.ini transcript vsim.wlf
.PHONY: simfun clean

For larger designs use a separate compile script
Free VHDL simulator – GHDL (example)

• Analyze the source file(s):
  ghdl -a <design>.vhd

• Analyze the testbench file(s):
  ghdl -a <design>_tb.vhd

• Generate executable file:
  ghdl -e <design>_tb

• Run the simulation:
  ghdl -r <design>_tb --vcd=<design>_tb.vcd
  --stop-time=3us

• View the waveform:
  gtkwave <design>_tb.vcd