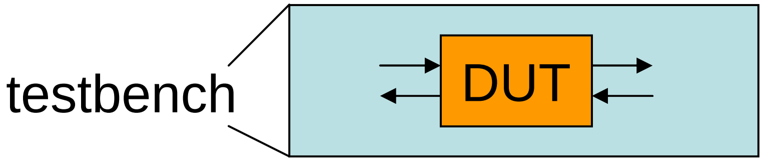
Simulations with VHDL

Simulations with VHDL

- Testbench
- Analog vs. digital simulation
- VHDL for simulation
  - Simple simulation example
  - **`wait`** in **`process`** for simulations
  - Delaying signals (**`after`**, **`'delayed`**)
  - Text I/O
  - Reporting - **`assert`**
  - Advanced simulation example
  - Recommended directory structure and example of Makefile for ModelSim
  - The free simulator GHDL

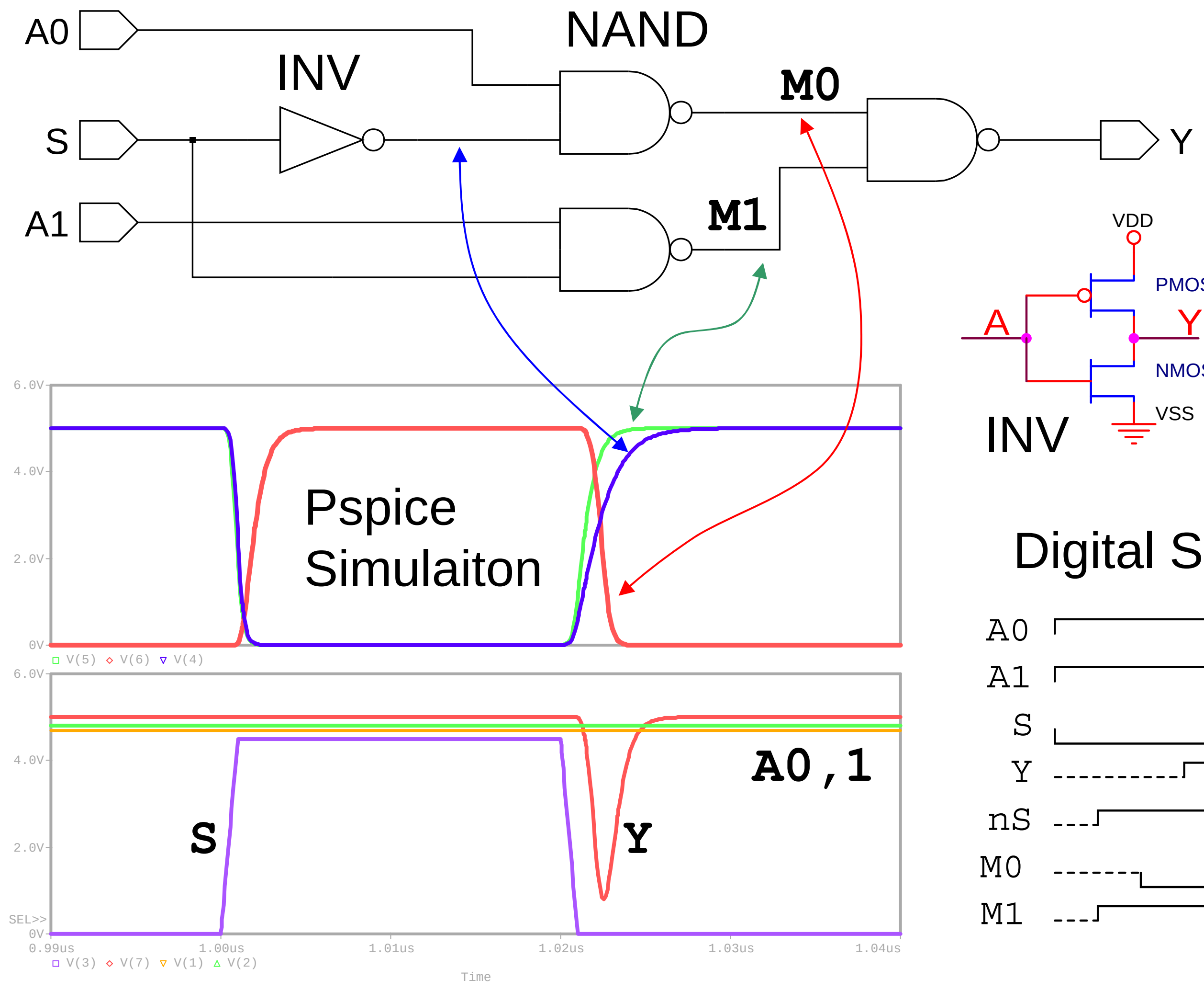
How to simulate – Testbench

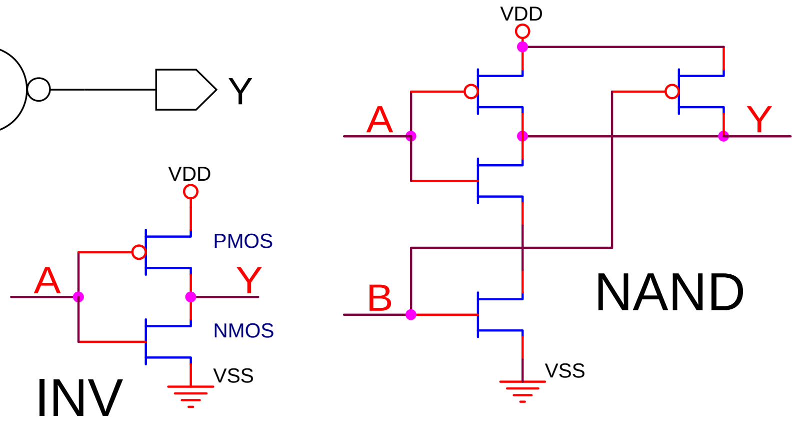
- Instantiate the design under test (DUT) into the so called **testbench**
- All signals to the DUT are driven by the **testbench**, all outputs of the DUT are read by the testbench and if possible analyzed



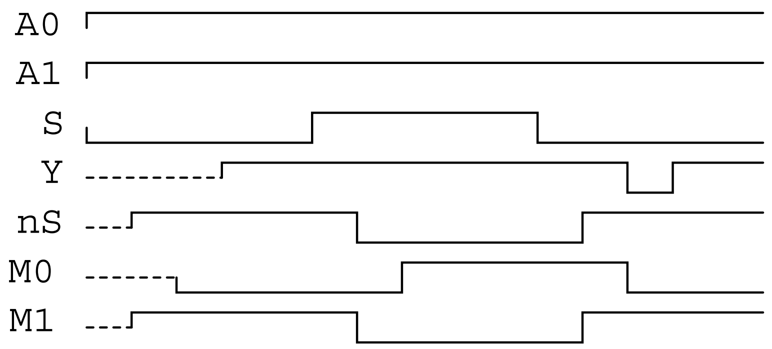
- Some subset of all signals at all hierarchy levels can be shown as a waveform
- The simulation is made many times at different design stages – functional, after the synthesis, after the placing and routing, sometimes together with the other chips on the board
- Many VHDL constructs used in a testbench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

Analog vs. digital simulation





Digital Simulation (ModelSim)



Simple test bench example

```
entity tb_proc1 is
end tb_proc1;
```

no ports

```
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;
```

component declaration

```
signal rst_n : std_logic;
signal d     : std_logic;

signal en_n  : std_logic;
signal q     : std_logic;
signal clk   : std_logic:= '1';
```

Initial value – only in simulations!

clock signal

```
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';
```

stimuli

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

Component instantiation

`wait` for simulations

```
process                              no sensitivity list
begin
  wait on clk;   ⟺   wait until clk'event;
  …                                  until event on any signal in the list
  wait on clk, reset;
  …
  wait until clk'event and clk='1';  edge detection
  …                                  boolean expression
  wait until d(0)='1';
  …
  wait for 20 ns;                    time
  …
  wait until d(4)='0' for 200 ns;
  …                                  until the boolean expression is TRUE
                                     but not more than the time
  wait;
  …                                  for ever, suspend the process
end process;
```

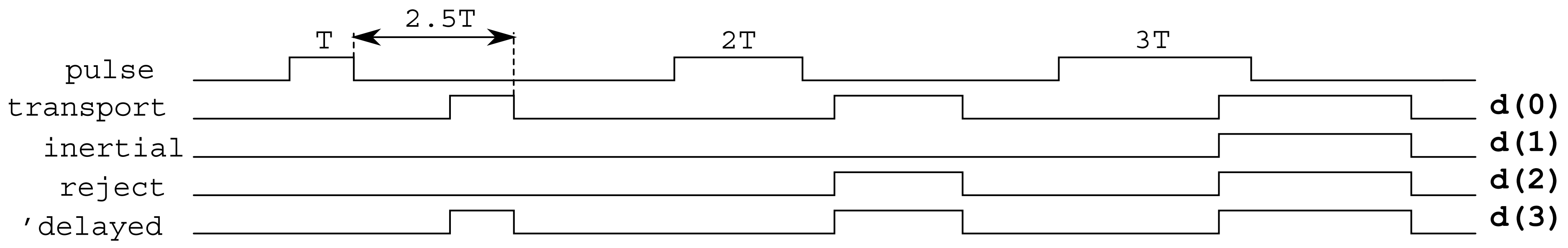
Delaying signals

- 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:

```
constant T : time := 2 ns;
...
d(0) <= transport pulse after 2.5*T;
d(1) <= inertial  pulse after 2.5*T;
          optional
d(2) <= reject 1.25*T inertial pulse after 2.5*T;
d(3) <= pulse'delayed(2.5*T); -- uses transport model
```

```
SIGNAL after TIME
        ⇕
inertial SIGNAL after TIME ⇔ reject TIME inertial SIGNAL after TIME
```

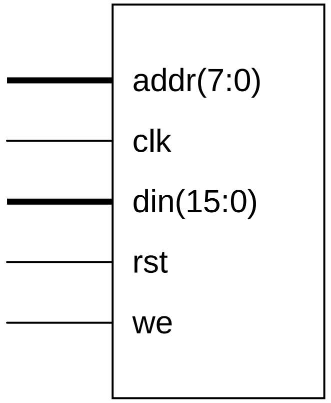


Text out in VHDL simulation (example)

```
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;
```

package to work with text files

counter (timer), not shown here



logging in a text file of all memory writes

```
0 0x00 0x0123
1 0x01 0x0000
2 0x01 0x0000
3 0x02 0x2000
4 0x03 0x3000
5 0x04 0x0000
6 0xFF 0x00FF
```

Text in (example)

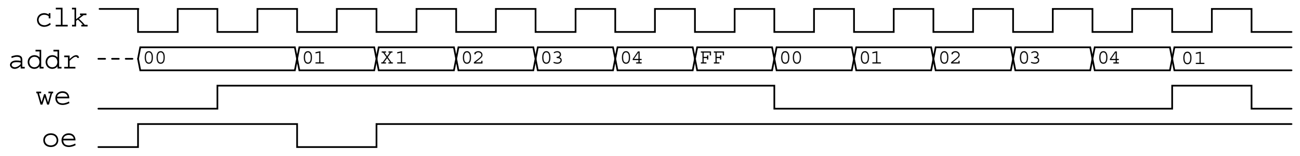
```
process
variable s : line;
variable goodw, gooda, goode : Boolean;
variable addrv : std_logic_vector(addr'range);
variable wev   : std_logic;
variable oev   : std_logic;
begin
    if endfile(infile) then
        wait;   -- stop reading
    end if;
    readline(infile, s);
    if s(s'low) /= '#' then
        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;
```

skip the comment lines

read only in variables!

copy to signals

```
# we oe addr
0 0 XXXXXXXX # nothing
0 1 00000000 # read from addr 0
1 1 00000000 # write to addr 0
1 0 00000001 # write to addr 1
1 1 0X000001 # write to undef addr
1 1 00000010 # write to addr 2
1 1 00000011 # write to addr 3
1 1 00000100 # write to addr 4
1 1 11111111 # write to addr FF
0 1 00000000 # read from addr 0
0 1 00000001 # read from addr 1
0 1 00000010 # read from addr 2
0 1 00000011 # read from addr 3
0 1 00000100 # read from addr 4
1 1 00000001 # write to addr 1
0 1 00000001 # read from addr 1
0 - 00000001 # read from addr 1
0 0 00000001 # read from addr 1
X 0 00000001 # ? from addr 1
0 1 00000001 # read from addr 1
0 1 11111111 # read from addr 1
```



Reporting in VHDL simulations – `assert`

```
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;
...
```

Check if all bits are good: `'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

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

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

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

 variable L : line;
begin
  wait until falling_edge(clk);
  if (dexp /= dout) then
    assert false
    report "Unexpected read data"
    severity WARNING;
    write(L, string'(", expected "));
    write(L, dexp);
    write(L, string'(", but read "));
    write(L, dout);
    writeline(output, L);
  end if;
end;
```

This procedure is used to check the read data and to report any errors.

Simulation of the registerfile(2)

```vhdl
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);
```

two dimensional array type

all signals in the testbench

the registerfile

store here the written data in order to compare later

```vhdl
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;
```

reset

Write some simple pattern

Simulation of the registerfile(3)

```
  we <= '0';
  rfile(5) := (others => '0'); -- emulate error, delete later
```

read & compare

```
  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;
```

reset

```
  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';
```

read & compare

```
  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          contains all testbench sources of the project
 │    ├───SRC
 │    ├───functional ┐ Created by the Make script to store
 │    └───work       ┘ the compiled ModelSim libraries
 └───SRC
         contains all source files of the project
```

Functional simulation - example of ModelSim Makefile

```
# 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)

ghdl
gtkwave

- 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
```

options to use the **std_logic_arith**

```
--ieee=synopsys
-fexplicit
```

- Run the simulation:

```
ghdl -r <design>_tb --vcd=<design>_tb.vcd
--stop-time=3us
```

Value Change Dump (VCD)

- View the waveform:

```
gtkwave <design>_tb.vcd
```