Simulation with ModelSim-Altera from Quartus II

Quick Start Guide

Embedded System Course

LAP – IC – EPFL – 2010

Version 0.5 (Preliminary)

René Beuchat, Cagri Onal

1 Installation and documentation

Main information in this document has been found on:

http:\\www.altera.com

This guide has been prepared to help students following the Embedded System Course in I&C by René Beuchat at EPFL. A development board FPGA4U is use during the laboratories with Quartus II froma Altera and ModelSim-Altera from Mentor.

Copy of the tools can be found at LAP for personal installation:

```
\\lapsrv1\distribution\Altera\Tools_For_Windows\To_install_QuartusII_10_0\
```

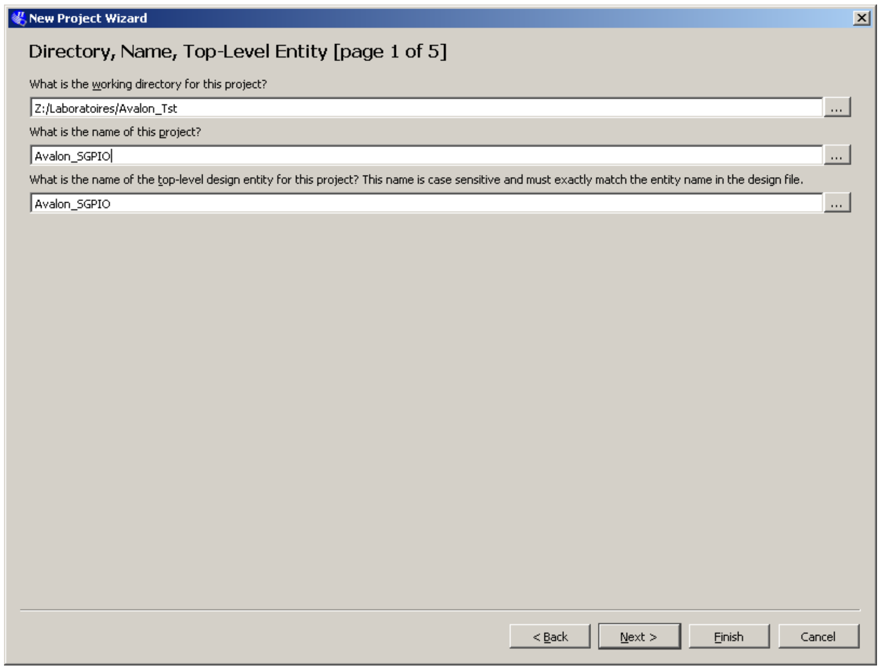
(or you can follow http:\\www.altera.com to download the install files after registration).

2 Launching QuartusII

This document is to be used with QuartusII with external tools for simulation. That is necessary from version 10.0 of QuartusII.

2.1 New QuartusII Project

To create new QuartusII project, select menu "File→ Create New Project Wizard"



Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

Z:/Laboratoires/Avalon_Tst

What is the name of this project?

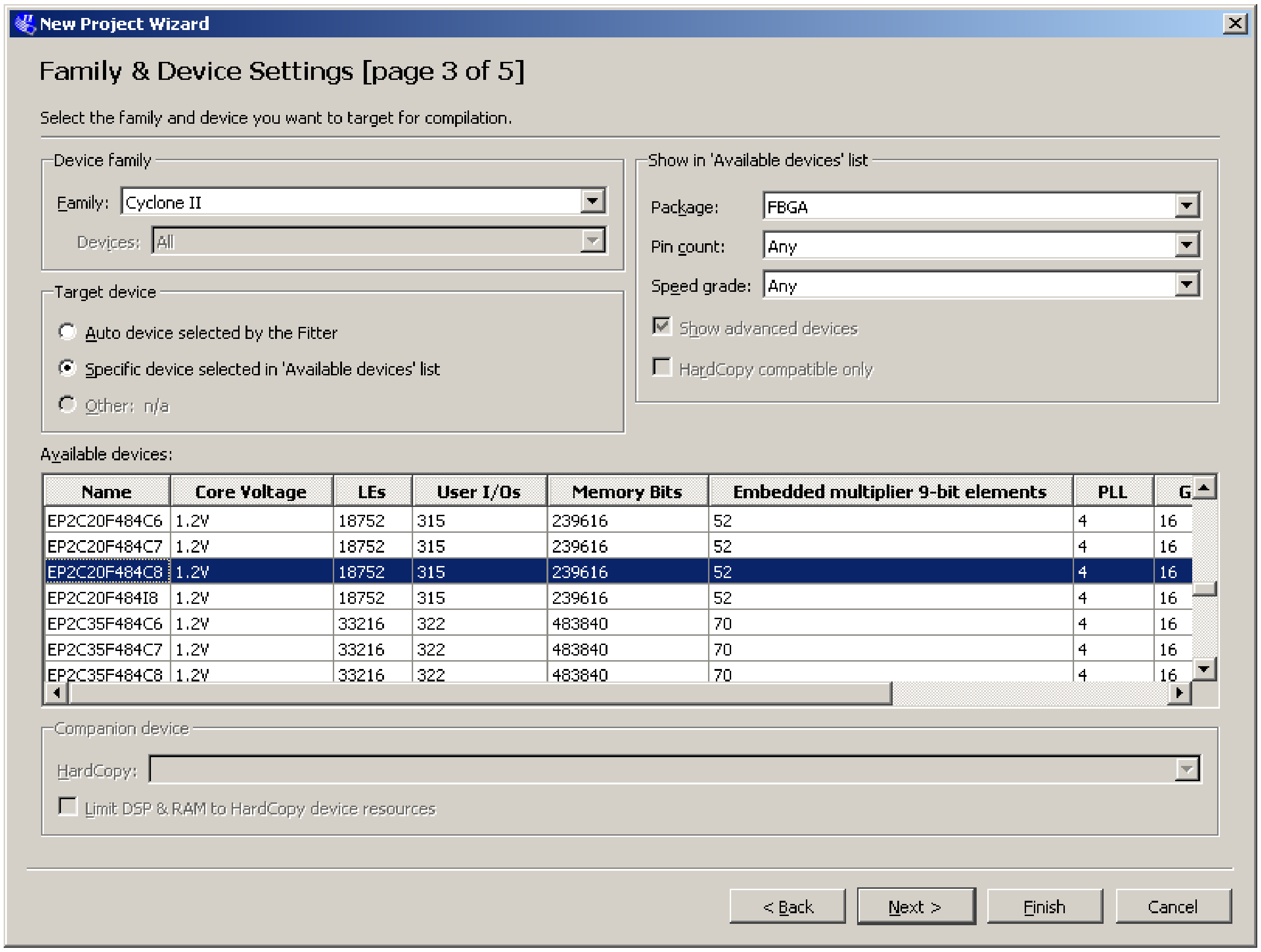
Avalon_SGPIO

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

Avalon_SGPIO

Fig. 1. Create a QuartusII project

NO space or special characters in your project, directory or files names.


New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family

Family: Cyclone II

Devices: All

Show in 'Available devices' list

Package: FBGA

Pin count: Any

Speed grade: Any

[✓] Show advanced devices

[ ] HardCopy compatible only

Target device

( ) Auto device selected by the Fitter

(●) Specific device selected in 'Available devices' list

( ) Other: n/a

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	G
EP2C20F484C6	1.2V	18752	315	239616	52	4	16
EP2C20F484C7	1.2V	18752	315	239616	52	4	16
EP2C20F484C8	1.2V	18752	315	239616	52	4	16
EP2C20F484I8	1.2V	18752	315	239616	52	4	16
EP2C35F484C6	1.2V	33216	322	483840	70	4	16
EP2C35F484C7	1.2V	33216	322	483840	70	4	16
EP2C35F484C8	1.2V	33216	322	483840	70	4	16

Companion device

HardCopy:

[ ] Limit DSP & RAM to HardCopy device resources

Fig. 2. FPGA selection

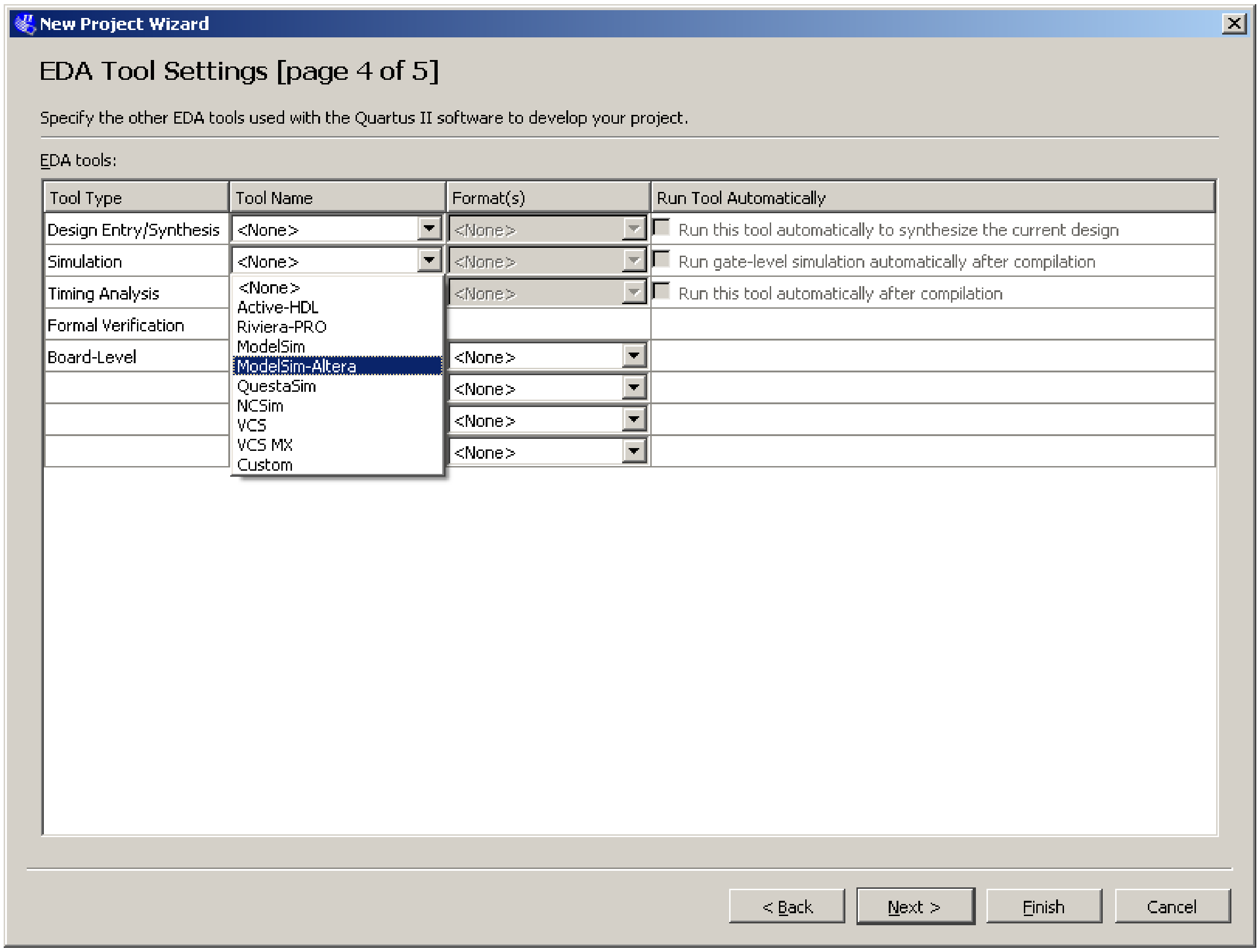
For FPGA4U:

- **Cyclone II** family
- **EP2C20F484C8** device

For Cyclone Robot:

- Cyclone Family
- EP1C12Q240C8 device

In creating the project, specify the **ModelSim-Altera** simulation option.



New Project Wizard

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	<None>	<None>	Run gate-level simulation automatically after compilation
Timing Analysis		<None>	Run this tool automatically after compilation
Formal Verification			
Board-Level		<None>	
		<None>	
		<None>	
		<None>	

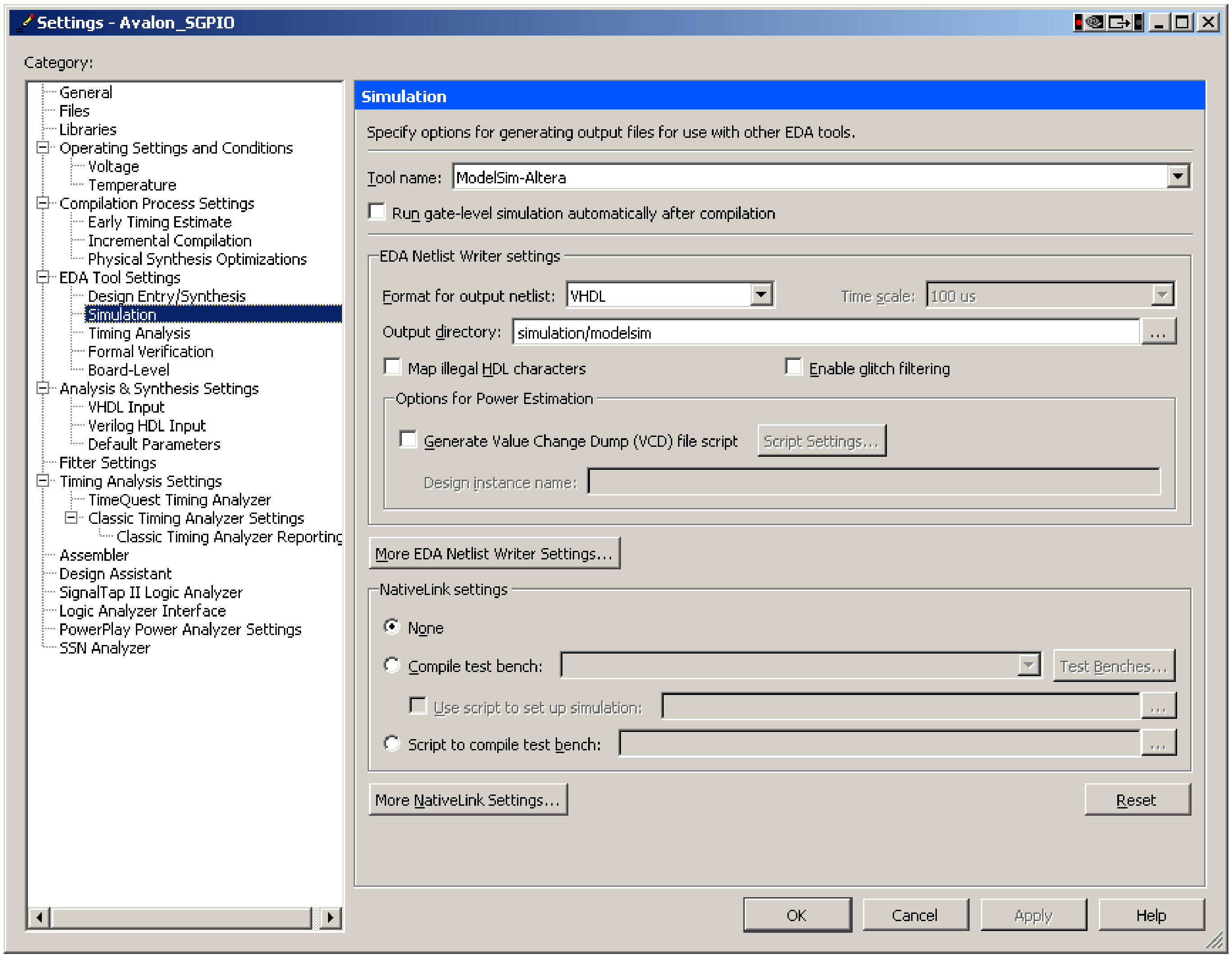
Fig. 3. Simulator selection

Notes:

- Some other external tools could be added here if available.
- ModelSim complete version could be used if licenses are available, but the libraries need to be build in this case. With ModelSim-Altera, all the libraries are directly available and linked to the tools.

It can be specified later with :

Assignments → Settings → EDA Tool Settings → Simulation


Settings - Avalon_SGPIO

Category:

- General
- Files
- Libraries
- Operating Settings and Conditions
  - Voltage
  - Temperature
- Compilation Process Settings
  - Early Timing Estimate
  - Incremental Compilation
  - Physical Synthesis Optimizations
- EDA Tool Settings
  - Design Entry/Synthesis
  - Simulation
  - Timing Analysis
  - Formal Verification
  - Board-Level
- Analysis & Synthesis Settings
  - VHDL Input
  - Verilog HDL Input
  - Default Parameters
- Fitter Settings
- Timing Analysis Settings
  - TimeQuest Timing Analyzer
  - Classic Timing Analyzer Settings
    - Classic Timing Analyzer Reporting
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- PowerPlay Power Analyzer Settings
- SSN Analyzer

Simulation

Specify options for generating output files for use with other EDA tools.

Tool name: ModelSim-Altera

☐ Run gate-level simulation automatically after compilation

EDA Netlist Writer settings

Format for output netlist: VHDL　　Time scale: 100 us

Output directory: simulation/modelsim

☐ Map illegal HDL characters　　☐ Enable glitch filtering

Options for Power Estimation

☐ Generate Value Change Dump (VCD) file script

Design instance name:

NativeLink settings

[✓ None]

Compile test bench:

☐ Use script to set up simulation:

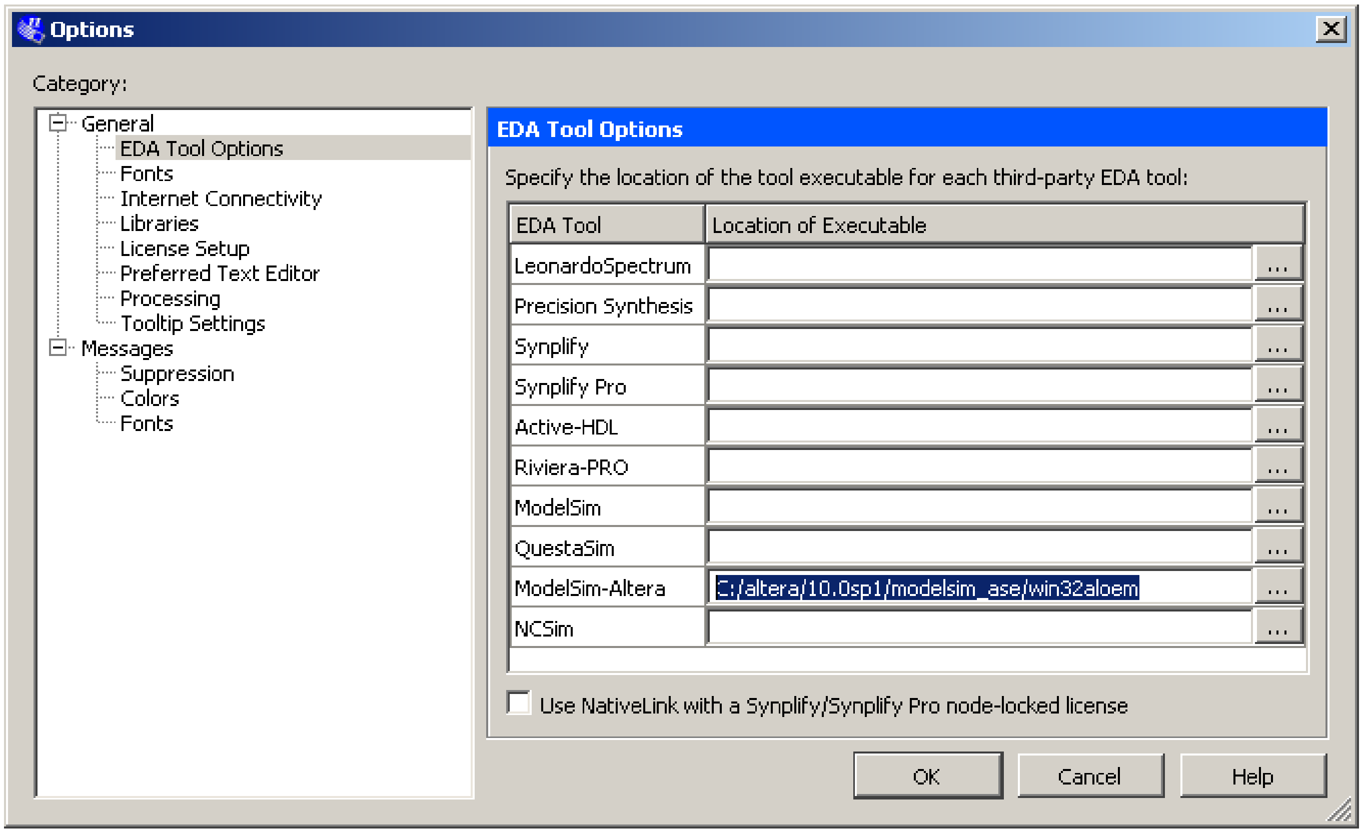
Script to compile test bench:

Script Settings...
More EDA Netlist Writer Settings...
Test Benches...
More NativeLink Settings...
Reset
OK
Cancel
Apply
Help

Fig. 4. Settings Tool selection for simulation

Tools → Options → General → EDA Tool Options

allows the specification of the tools path. Assign the ModelSim-Altera directory



EDA Tool Options

Specify the location of the tool executable for each third-party EDA tool:

EDA Tool	Location of Executable
LeonardoSpectrum	
Precision Synthesis	
Synplify	
Synplify Pro	
Active-HDL	
Riviera-PRO	
ModelSim	
QuestaSim	
ModelSim-Altera	C:/altera/10.0sp1/modelsim_ase/win32aloem
NCSim	

☐ Use NativeLink with a Synplify/Synplify Pro node-locked license

Fig. 5. Eda Tools path

3 Program example to start

A very simple parallel port with direction programmable for each bit is created for the Avalon slave bus.

File → New → Design Files → VHDL File

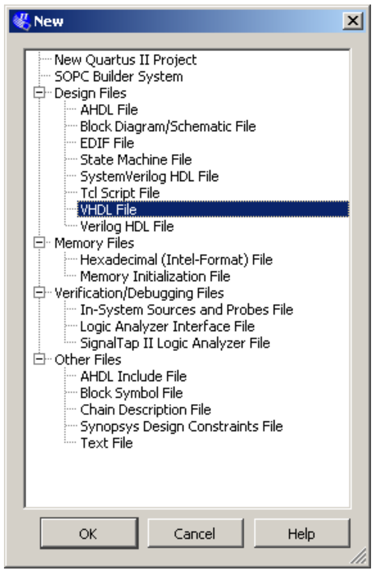


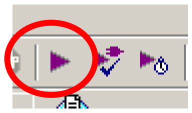
Fig. 6. Create VHDL File

```
-- Design of a simple parallel port
-- Avalon slave unit
-- Parallel Port with programmable direction bit by bit on 8 bits
--
-- 3 address:
--   0: data
--   1: direction 0: input (reset state), 1: output
--   2: read data pin Read only

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Avalon_SGPIO IS
   PORT(
      Clk      : IN  std_logic;
      nReset   : IN  std_logic;
      CS       : IN  std_logic;
      Rd       : IN  std_logic;
      Wr       : IN  std_logic;
      RDData   : OUT std_logic_vector (7 DOWNTO 0);
      WRData   : IN  std_logic_vector (7 DOWNTO 0);
      Adr      : IN  std_logic_vector (1 DOWNTO 0);
      PortP    : INOUT  std_logic_vector (7 DOWNTO 0)
);
END Avalon_SGPIO ;
```

```
ARCHITECTURE bhv OF Avalon_SGPIO IS
   signal      iRegPort    :  std_logic_vector (7 DOWNTO 0);   -- internal registers
   signal      iRegDir     :  std_logic_vector (7 DOWNTO 0);   -- internal registers
   signal      iRegPin     :  std_logic_vector (7 DOWNTO 0);   -- Driver for reading pin value
BEGIN

-- Process to write internal registers through Avalon bus interface
-- Synchronous access in rising_edge of clk
-- Addresses allows to select write registers if CS and Wr activated
WrReg:            -- Write by Avalon slave access
   Process(Clk, nReset)
   Begin
      if nReset = '0' then
         iRegDir     <= (others => '0');  -- input at reset
         iRegPort    <= (others => '0');  -- Port value = 0 at reset
      elsif rising_edge(Clk) then
         If (CS = '1') and (Wr = '1') then
            case Adr is
               when "00"  =>
                  iRegPort <= WRData;
               When  "01" =>
                  iRegDir  <= WRData;
               When others =>
                  null;
            End case;
         End if;
      End if ;
    end process WrReg ;
```

```
-- interfal buffer for reading external pin value

iRegPin <= PortP;       -- Parallel Port direct access

-- Process to read the different sources of data by the Avalon bus interface
-- could be sometimes better with synchronous access on rising_edge of clk with 1 wait cycle

RdReg:            -- Read by Avalon slave access
   Process(CS, Rd, Adr, iRegPort, iRegDir, iRegPin)
   Begin
      RDData <= (others => '0');
      If (CS = '1') and (Rd = '1') then
         case Adr is
            when "00"  =>
               RDData   <= iRegPort ;
            when "01"  =>
               RDData   <= iRegDir ;
            when "10" =>
               RDData   <= iRegPin;
            When others =>
               RDData <= (others => '0');
         End case;
      End if;
   End process RdReg;

-- Process to control the buffer output for external output accesses or selecting input direction
-- and putting the output in Z (tri-state) state

PortIO:              -- Effect on Parallel port
   process(iRegPort, iRegDir)
   begin
      for i in 0 to 7 loop
         if iRegDir(i) = '1' then
            PortP(i) <= iRegPort(i);
         else
            PortP(i) <= 'Z';
         end if;
      End loop;
   end process;

END bhv;
```

The file can then be compiled:

Processing → Start compilation or click the 

If there are no error the design can be simulated.

4 Simulation

4.1 Preparation to simulation

The simulator ModelSim-Altera can be launched from QuartusII in 2 modes:

- ***RTL simulation***, without real delay, only functional simulation
- ***Gate level simulation***, can have "real" delay from place and route timing generation

A file with **.sdo** extension is created after compilation in QuartusII and contains the delay from technology and place & route. It is necessary to specify it in ModelSim to have gate level delay.

In ModelSim:

Simulate → Start Simulation → SDF → Add… and search in your *project_directory*\simulation\\xxx***.sdo***

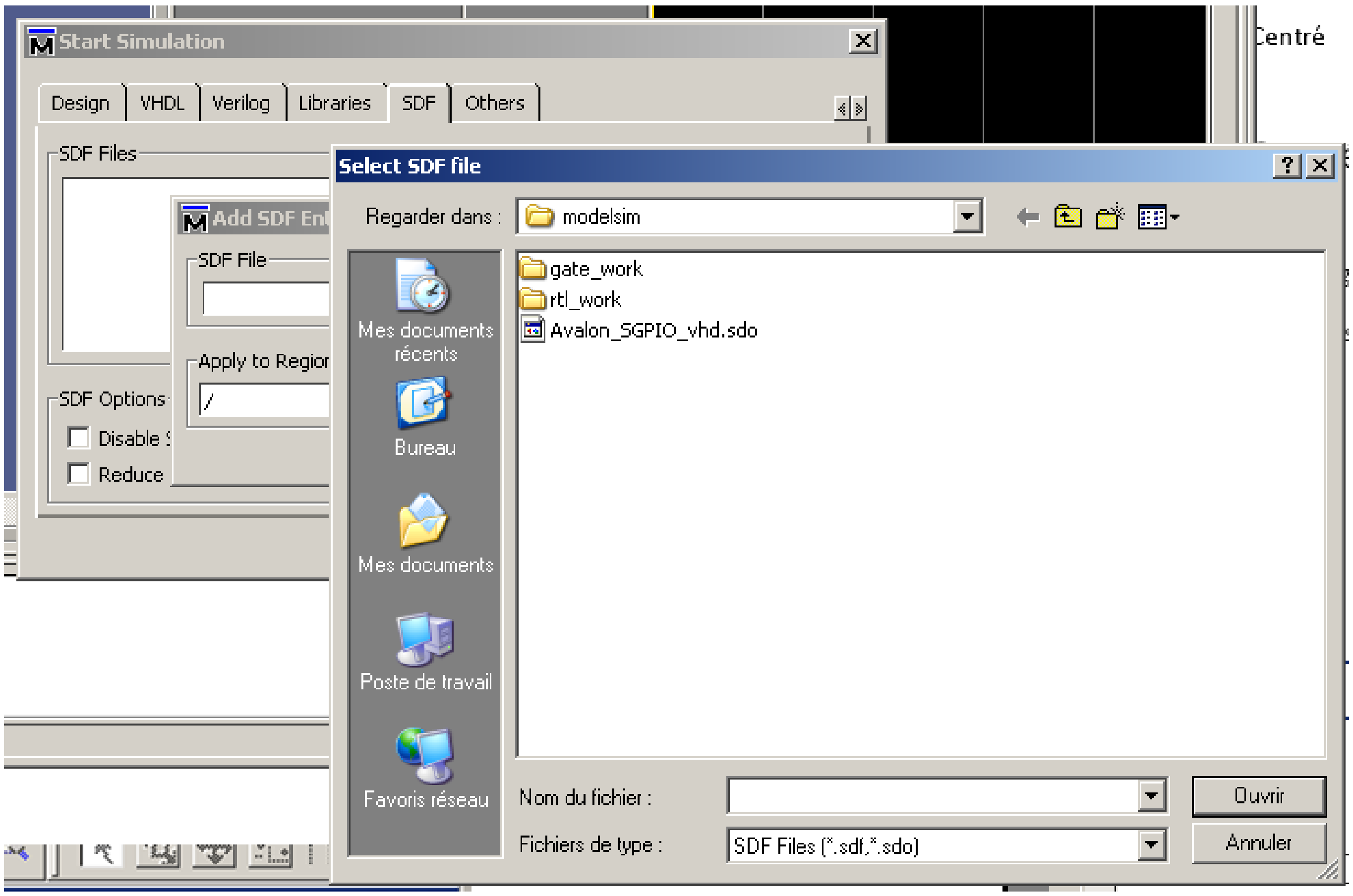
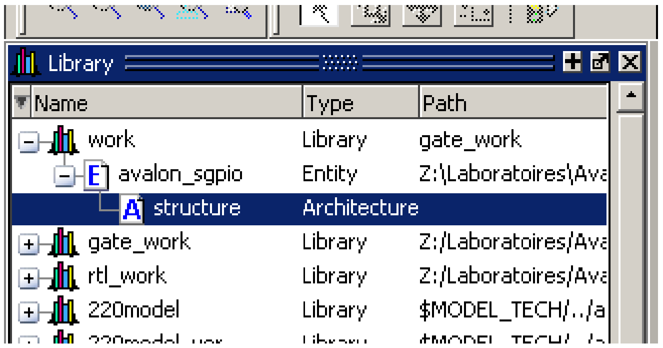


Fig. 7. Specify timing information

To open the simulation file, Open in the navigator windows the work library and select the architecture of the entity to simulate.



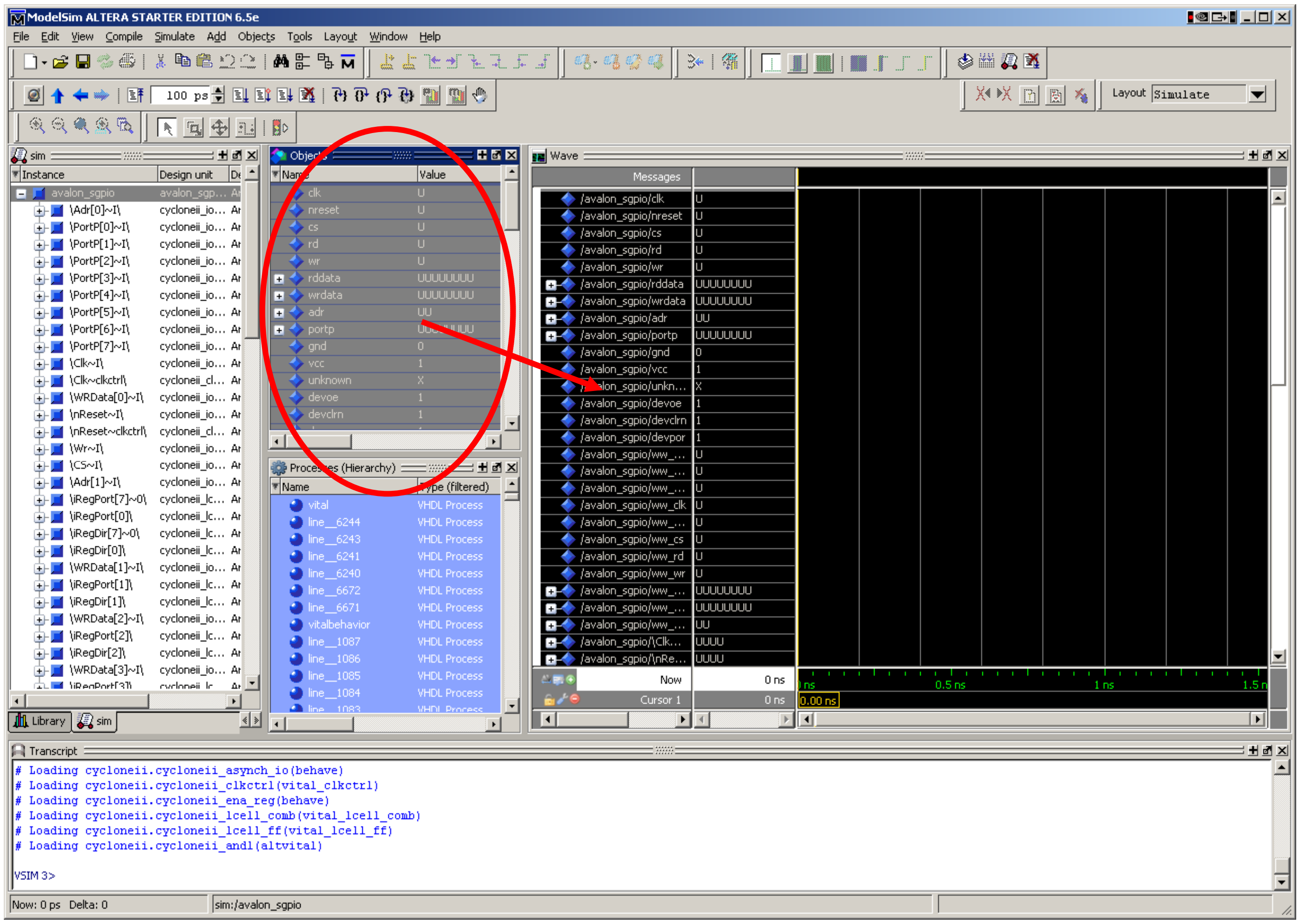
Library

Name	Type	Path
work	Library	gate_work
avalon_sgpio	Entity	Z:\Laboratoires\Ava
structure	Architecture	
gate_work	Library	Z:/Laboratoires/Ava
rtl_work	Library	Z:/Laboratoires/Ava
220model	Library	$MODEL_TECH/../a

Fig. 8. File to simulate

4.2 Signals to simulate

You select from the object windows all the signals to simulate and drag and drop on the Wave windows. If the Wave window is not displayed, select them from ***View→ Wave***



ModelSim ALTERA STARTER EDITION 6.5e

Objects

Name	Value
clk	U
nreset	U
cs	U
rd	U
wr	U
rddata	UUUUUUUU
wrdata	UUUUUUUU
adr	UU
portp	UUUUUUUU
gnd	0
vcc	1
unknown	X
devoe	1
devclrn	1

Wave

Messages	
/avalon_sgpio/clk	U
/avalon_sgpio/nreset	U
/avalon_sgpio/cs	U
/avalon_sgpio/rd	U
/avalon_sgpio/wr	U
/avalon_sgpio/rddata	UUUUUUUU
/avalon_sgpio/wrdata	UUUUUUUU
/avalon_sgpio/adr	UU
/avalon_sgpio/portp	UUUUUUUU
/avalon_sgpio/gnd	0
/avalon_sgpio/vcc	1
/avalon_sgpio/unkn...	X
/avalon_sgpio/devoe	1
/avalon_sgpio/devclrn	1
/avalon_sgpio/devpor	1
/avalon_sgpio/ww_...	U
/avalon_sgpio/ww_...	U
/avalon_sgpio/ww_...	U
/avalon_sgpio/ww_clk	U
/avalon_sgpio/ww_...	U
/avalon_sgpio/ww_cs	U
/avalon_sgpio/ww_rd	U
/avalon_sgpio/ww_wr	U
/avalon_sgpio/ww_...	UUUUUUUU
/avalon_sgpio/ww_...	UUUUUUUU
/avalon_sgpio/ww_...	UU
/avalon_sgpio/\Clk...	UUUU
/avalon_sgpio/\nRe...	UUUU
Now	0 ns
Cursor 1	0 ns

Transcript

```
# Loading cycloneii.cycloneii_asynch_io(behave)
# Loading cycloneii.cycloneii_clkctrl(vital_clkctrl)
# Loading cycloneii.cycloneii_ena_reg(behave)
# Loading cycloneii.cycloneii_lcell_comb(vital_lcell_comb)
# Loading cycloneii.cycloneii_lcell_ff(vital_lcell_ff)
# Loading cycloneii.cycloneii_andl(altvital)

VSIM 3>
```

Now: 0 ps Delta: 0 sim:/avalon_sgpio

Fig. 9. ModelSim windows

With commands from Modelsim it is possible to simulate the design.

Some useful commands:

➢ **restart**	#clear timing diagram and reload SDF files, put timing at 0
➢ **force -r 20ns, clk 0, 1 10ns**	#force the clk signal with a repeat periode of 20ns, 0 now and 1 at 10ns
➢ **run** *xx*	#run for xx time ex: 200ns, all initialized values are in RED as U or X
➢ **force** nreset 1 10ns, 0 100ns, 1 200ns	# specify the level and when to put it
➢ **force -deposit** sim:/avalon_sgpio/portp LHLHLHLH 100ns	# just put the signal, but can be modified by the simulation result

The commands can be send through the commands windows or from a script file. In this case it has the ***.do*** extension.

```
Example that can be put in a xxx.do file:
```

To run it: ***do xxx.do***

```
restart
force -r 20ns, clk 0, 1 10ns
force nreset 1 10ns, 0 100ns, 1 200ns
run 200 ns
force rd 0
force cs 1, 0 40ns, 1 80ns
force wr 1, 0 40ns, 1 80ns
force -freeze sim:/avalon_sgpio/wrdata 11110000 0
force -freeze sim:/avalon_sgpio/adr 00 0
force -deposit sim:/avalon_sgpio/portp LHLHLHLH 100ns
run 200ns
force cs 1, 0 40ns, 1 80ns
force wr 1, 0 40ns, 1 80ns
force -freeze sim:/avalon_sgpio/wrdata 00111100 0
force -freeze sim:/avalon_sgpio/adr 01 0
run 200ns
force -deposit sim:/avalon_sgpio/portp ZZZZZZZZ 0
run 100ns
```

4.3 Simulation by Test bench

A testbench can be written in VHDL. This VHDL doesn't needs to be synthesizable and can contain ***WAIT UNTIL*** structure. This means that it is possible to wait on some signals activation before continuing the simulation.

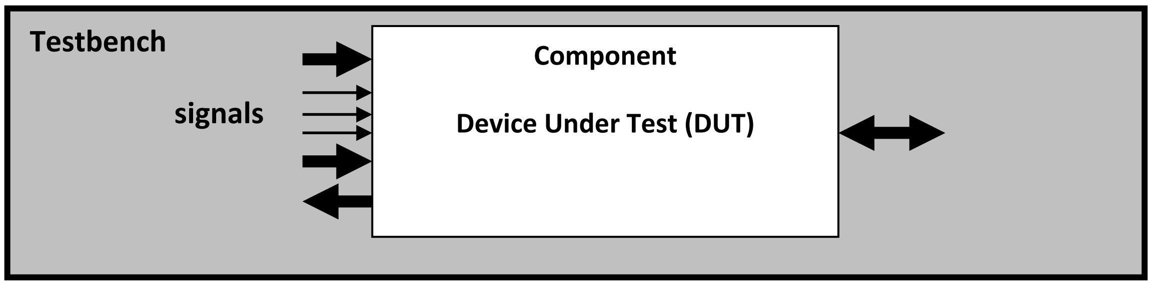


Fig. 10. Testbench structure

The simulation VHDL program can be seen as a sequential suite of instructions to execute.

A testbench contains the sequence of events to send to the tested module inferred as a **component.** Structural connection is done in the main structure to the component through **port map**.

An example is provided with a package and his body package with 2 procedures:

- a write procedure simulating a simplified Avalon slave write transfer: WrBus
- a read procedure simulating a simplified Avalon slave read transfer: RdBus

A procedure is like a process and executed sequentially. Parameters can be passed as signal or constant values.

Put the 2 files in the active directory where the tested architecture is. This is the **work library** place.

4.3.1 A Package to help Avalon access for test bench

2 procedures are provided to simulate Read and Write Access as Avalon slave.

The procedures are proposed under the ways of a package and a package body.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE CycleAvalon IS

-- Procedures
-- ----------
-- need to be adapted depending on address bus size and data size
-- Access by Avalon bus --> FPGA for simulation
procedure WrBus(
   sAdresse                : IN STD_LOGIC_VECTOR (31 downto 0);   -- provide the address to read the data
   sData                   : IN STD_LOGIC_VECTOR (31 downto 0);   -- provide the data to read
   SIGNAL Address          : OUT STD_LOGIC_VECTOR (2 downto 0);   -- Generated address for module
   SIGNAL WriteData        : OUT STD_LOGIC_VECTOR (7 downto 0);   -- data write to module to test
   SIGNAL Clk              : IN STD_LOGIC;                        -- clk generated in a process at tb level
   SIGNAL ChipSelect       : OUT STD_LOGIC;                       -- Avalon CS signal generated
   SIGNAL Write            : OUT STD_LOGIC;                       -- Avalon Write signal generated
   SIGNAL Read             : OUT STD_LOGIC;                       -- Avalon Read signal generated
   SIGNAL nBE              : OUT STD_LOGIC_VECTOR (3 downto 0);   -- nBE generated
   NbWait, NbSetUp, NbHold : IN integer range 0 TO 20;            -- constant values for write timing
   SIGNAL WaitRequest      : IN STD_LOGIC                         -- WaitRequest from another process
   );

-- need to be adapted depending on address bus size and data size
-- Access by Avalon bus --> FPGA for simulation
-- Reading
procedure RdBus(
   sAdresse                : IN STD_LOGIC_VECTOR (31 downto 0);
   SIGNAL sData            : OUT STD_LOGIC_VECTOR (31 downto 0);
   SIGNAL Address          : OUT STD_LOGIC_VECTOR (2 downto 0);
   SIGNAL ReadData         : IN STD_LOGIC_VECTOR (7 downto 0);
   SIGNAL Clk              : IN STD_LOGIC;
   SIGNAL ChipSelect       : OUT STD_LOGIC;
   SIGNAL Write            : OUT STD_LOGIC;
   SIGNAL Read             : OUT STD_LOGIC;
   SIGNAL nBE              : OUT STD_LOGIC_VECTOR (3 downto 0);
   NbWait, NbSetUp         : IN integer range 0 TO 20;
   SIGNAL WaitRequest      : IN STD_LOGIC
   );
END CycleAvalon;
```

4.3.2 Body package:

And the implementation in a body package.

```
LIBRARY STD;
USE STD.TEXTIO.all;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

PACKAGE BODY CycleAvalon IS

-- delay for simulation Avalon cycle
constant tAD : TIME := 5 ns;  --
constant tCS : TIME := 5 ns;  --
constant tDa : TIME := 5 ns;  --
constant tWR : TIME := 5 ns;  --
constant tRD : TIME := 5 ns;  --
constant tBE : TIME := 5 ns;  --

-- Procedures
-- ----------
-- Access by Avalon --> FPGA for simulation
procedure WrBus(
   sAdresse                 : IN STD_LOGIC_VECTOR (31 downto 0);
   sData                    : IN STD_LOGIC_VECTOR (31 downto 0);
   SIGNAL Address           : OUT STD_LOGIC_VECTOR (2 downto 0);
   SIGNAL WriteData         : OUT STD_LOGIC_VECTOR (7 downto 0);
   SIGNAL Clk               : IN STD_LOGIC;
   SIGNAL ChipSelect        : OUT STD_LOGIC;
   SIGNAL Write             : OUT STD_LOGIC;
   SIGNAL Read              : OUT STD_LOGIC;
   SIGNAL nBE               : OUT STD_LOGIC_VECTOR (3 downto 0);
   NbWait, NbSetUp, NbHold  : IN integer range 0 TO 20;
   SIGNAL WaitRequest       : IN STD_LOGIC
   ) is

-- simulation simple write cycle
-- WrBus ( CONV_STD_LOGIC_VECTOR(X"00001,20), CONV_STD_LOGIC_VECTOR(X"1234",16), …)

begin
      Address(2 downto 0)     <= sAdresse(2 downto 0)    after tAD;
      WriteData(7 downto 0)   <= sData(7 downto 0)       after tDa;
      nBE                     <= "0000"                  after tBE;
      ChipSelect              <= '1'                     after tCS;
      Write                   <= '0';
      Read                    <= '0'                     after tRD;
   for i in 0 to NbSetUp loop
      wait until Clk = '0';      -- attend flanc descendant Clk
      wait until Clk = '1';      -- attend flanc montant Clk
   end loop;
      Write    <= '1'         after tWR;
      wait until Clk = '0';
      wait until Clk = '1';
   for i in 0 to NbWait loop
      wait until Clk = '0';
      wait until Clk = '1';
   end loop;
      wait until WaitRequest = '0'; -- attend quittance
      Write       <= '0'         after tWR;
   for i in 0 to NbHold loop
      wait until Clk = '0';
      wait until Clk = '1';
   end loop;
      nBE         <= "1111"         after tWR;
      ChipSelect  <= '0'            after tCS;
      Read        <= '0'            after tRD;

end WrBus;
```

```
-- Reading
procedure RdBus(
   sAdresse               : IN STD_LOGIC_VECTOR (31 downto 0);
   Signal sData           : OUT STD_LOGIC_VECTOR (31 downto 0);
   SIGNAL Address         : OUT STD_LOGIC_VECTOR (2 downto 0);
   SIGNAL ReadData        : IN STD_LOGIC_VECTOR (7 downto 0);
   SIGNAL Clk             : IN STD_LOGIC;

   SIGNAL ChipSelect      : OUT STD_LOGIC;
   SIGNAL Write           : OUT STD_LOGIC;
   SIGNAL Read            : OUT STD_LOGIC;
   SIGNAL nBE             : OUT STD_LOGIC_VECTOR (3 downto 0);
   NbWait, NbSetUp        : IN integer range 0 TO 20;
   SIGNAL WaitRequest     : IN STD_LOGIC
   ) is


-- simulation simple reading cycle
-- RdBus ( CONV_STD_LOGIC_VECTOR(X"00001,20), CONV_STD_LOGIC_VECTOR(X"1234",16), …)

begin
      Address(2 downto 0)     <= sAdresse(2 downto 0)    after tAD;
      nBE         <= "0000"      after tBE;
      ChipSelect  <= '1'         after tCS;
      Write       <= '0';
      Read     <= '0'            after tRD;
   for i in 0 to NbSetUp loop
      wait until Clk = '0';      -- wait falling edge Clk
      wait until Clk = '1';      -- wait rising edge  Clk
   end loop;
```

```
      Read     <= '1'         after tWR;
   for i in 0 to NbWait loop
      wait until Clk = '0';
      wait until Clk = '1';
   end loop;
      wait until WaitRequest = '0'; -- wait Acknowledge
      nBE                 <= "1111"        after tWR;
      ChipSelect          <= '0'           after tCS;
      Read                <= '0'           after tRD;
      sData(7 downto 0)   <= ReadData(7 downto 0);
      sData(31 downto 8)  <= (others => '0');
end RdBus;

END CycleAvalon;
```

4.3.3 TestBench

The tesbench itself provides processes:

- for clk generation,
- nReset activation/deactivation
- call to the procedures executed in simulation mode only

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY std;
USE std.textio.all;

LIBRARY work;
USE work.CycleAvalon.all;

entity testbench is
   -- Nothing as input/output
end testbench;


ARCHITECTURE bhv OF testbench IS
-- The system to test under simulation

component ParallelPort is
     Port (
      Clk            : in std_logic;
      nReset         : in std_logic;
      Address        : in std_logic_vector(2 downto 0);
      ChipSelect     : in std_logic;
      Read           : in std_logic;
      Write          : in std_logic;
      ReadData       : OUT std_logic_vector (7 DOWNTO 0);
      WriteData      : IN std_logic_vector (7 DOWNTO 0);
      ParPort        : INOUT std_logic_vector (7 DOWNTO 0)
     );
   end component;

-- The interconnection signals:
   signal Clk           : std_logic;
   signal nReset        : std_logic;
   signal Address       : std_logic_vector(2 downto 0);
   signal ChipSelect    : std_logic;
   signal Read          : std_logic;
   signal Write         : std_logic;
   signal nBE           : std_logic_vector(3 downto 0);
   signal WaitRequest   : std_logic;
   signal ReadData      : std_logic_vector (7 DOWNTO 0);
   signal SReadData     : std_logic_vector (31 DOWNTO 0);
   signal WriteData     : std_logic_vector (7 DOWNTO 0);
   signal ParPort       : std_logic_vector (7 DOWNTO 0);

   constant HalfPeriod : TIME := 10 ns;  -- 50 MHz -> 20ns/2 -> 10 ns

BEGIN

DUT : ParallelPort              -- Component to Test as Device Under Test
     Port MAP(
       Clk          => Clk,              -- from component => signals in the architecture
       nReset       => nReset,
       Address      => Address,
       ChipSelect   => ChipSelect,
       Read         => Read,
       Write        => Write,
       ReadData     => ReadData,
       WriteData    => WriteData,
       ParPort      => ParPort
     );

-- All Byte Enable always activated (in this test)
nBE      <= "0000";

-- Reset activation, active low pulse of 50ns every 10us
reset_process :
process
   begin
   nReset <= '0';
   wait for 50 ns;
   nReset <= '1';
   wait for 10 us;                       -- to repeat the cycle again and again at 10 us intervalle
end process;

-- Clock generation for all simulation time
clk_process :
process
   begin
    clk <= '0';
    wait for HalfPeriod;
    clk <= '1';
    wait for HalfPeriod;
end process;

-- WaitRequest generation by a parallel process, start counting clock cycle when ChipSelect activated
```

```
waitreq_process:
process
begin
      wait until ChipSelect = '1';
      WaitRequest <= '1';
      wait for 10*HalfPeriod;    -- 5 Clock cycle with WaitRequest activated
      wait until Clk = '1';
      WaitRequest <= '0';
end process;

-- Bus acces to initialize the GPIO and use it
read_write:
PROCESS
   BEGIN
      wait for 50 ns;
      loop
         ParPort  <= "ZZZZZZZZ";    -- external state Z
         WrBus ( X"00000000", X"000000FF" , Address, WriteData, Clk,  ChipSelect, Write, Read,
nBE,1,1,1,WaitRequest);       -- Write 0xFF @addresse 0, with 1 setup time, 1 wait and 1 hold
         wait for 50 ns;
         RdBus ( X"00000000", SReadData   , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,
WaitRequest);                 -- Read @addresse 0, with 1 setup time and 1 wait
         wait for 50 ns;
         WrBus ( X"00000002", X"0000009b" , Address, WriteData, Clk,  ChipSelect, Write, Read, nBE,1,1,0,
WaitRequest);                 -- Write 0x9B @addresse 2, with 1 setup time, 1 wait and 0 hold
         wait for 50 ns;
         RdBus ( X"00000002", SReadData   , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,
WaitRequest);
         wait for 50 ns;
         WrBus ( X"00000000", X"00000000" , Address, WriteData, Clk,  ChipSelect, Write, Read, nBE,1,1,1,
WaitRequest);
         wait for 50 ns;
         RdBus ( X"00000000", SReadData   , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,
WaitRequest);
         wait for 50 ns;
         ParPort  <= x"7d";
         RdBus ( X"00000001", SReadData , Address, ReadData, Clk,  ChipSelect, Write, Read, nBE,1,1,
WaitRequest);
         wait for 50 ns;
      end loop;
END PROCESS;

END bhv;
```

The test needs to be adapted to the different unit to test on Avalon slave and the tesbend is easy to change.

With the assert test it is easy to verify if an answer is correct:

Syntaxe :

```
    ASSERT condition                        -- Boolean condition
        [REPORT "string"]                   -- if not correct, condition wrong, display the string
        [SEVERITY severity_level];          -- with the severity_level indicated
```

`with TYPE severity_level IS (note, warning, error, failure);`

With the assert condition it is necessary that the developer knows the result of the test !! Yes !

Sommaire

Liste of Figures