Simulation with ModelSim-Altera from Quartus II

Quick Start Guide

Embedded System Course

LAP – IC – EPFL – 2010

Version 0.6 (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”

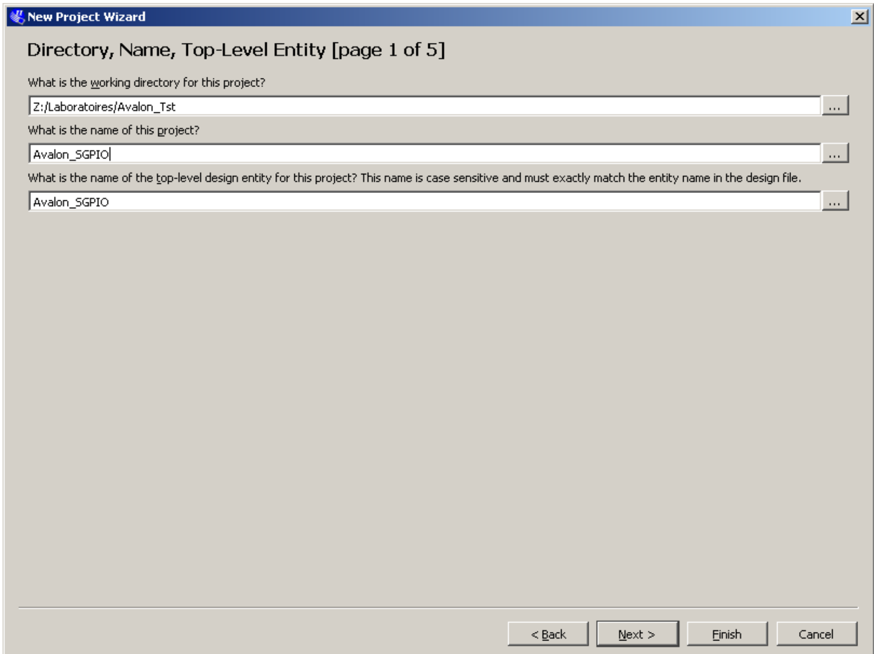


Fig. 1. Create a QuartusII project

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

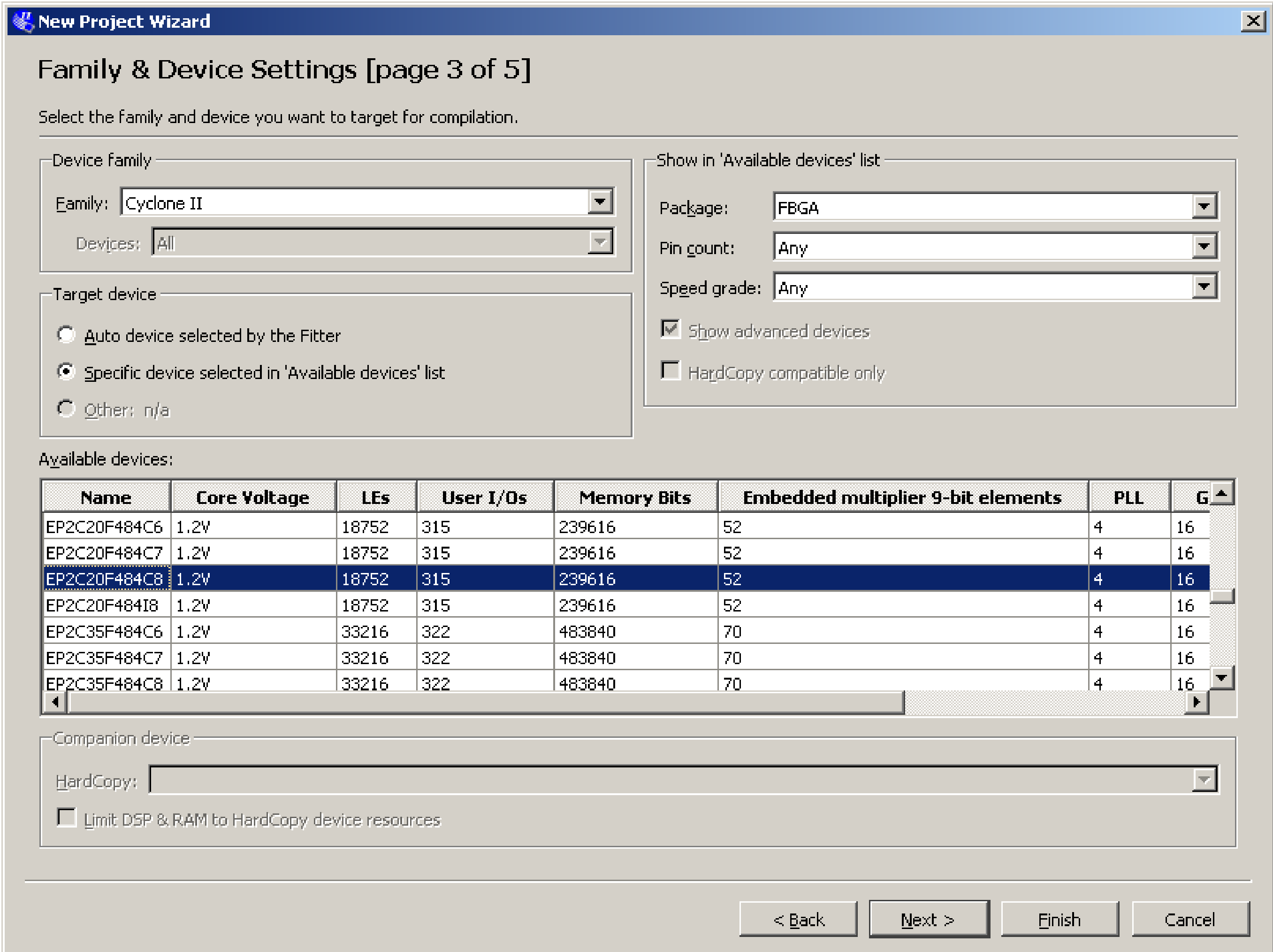


Fig. 2. FPGA selection

For FPGA4U:

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

For Cyclone Robot:

- Cyclone Family
- EP1C12Q240C8 device

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

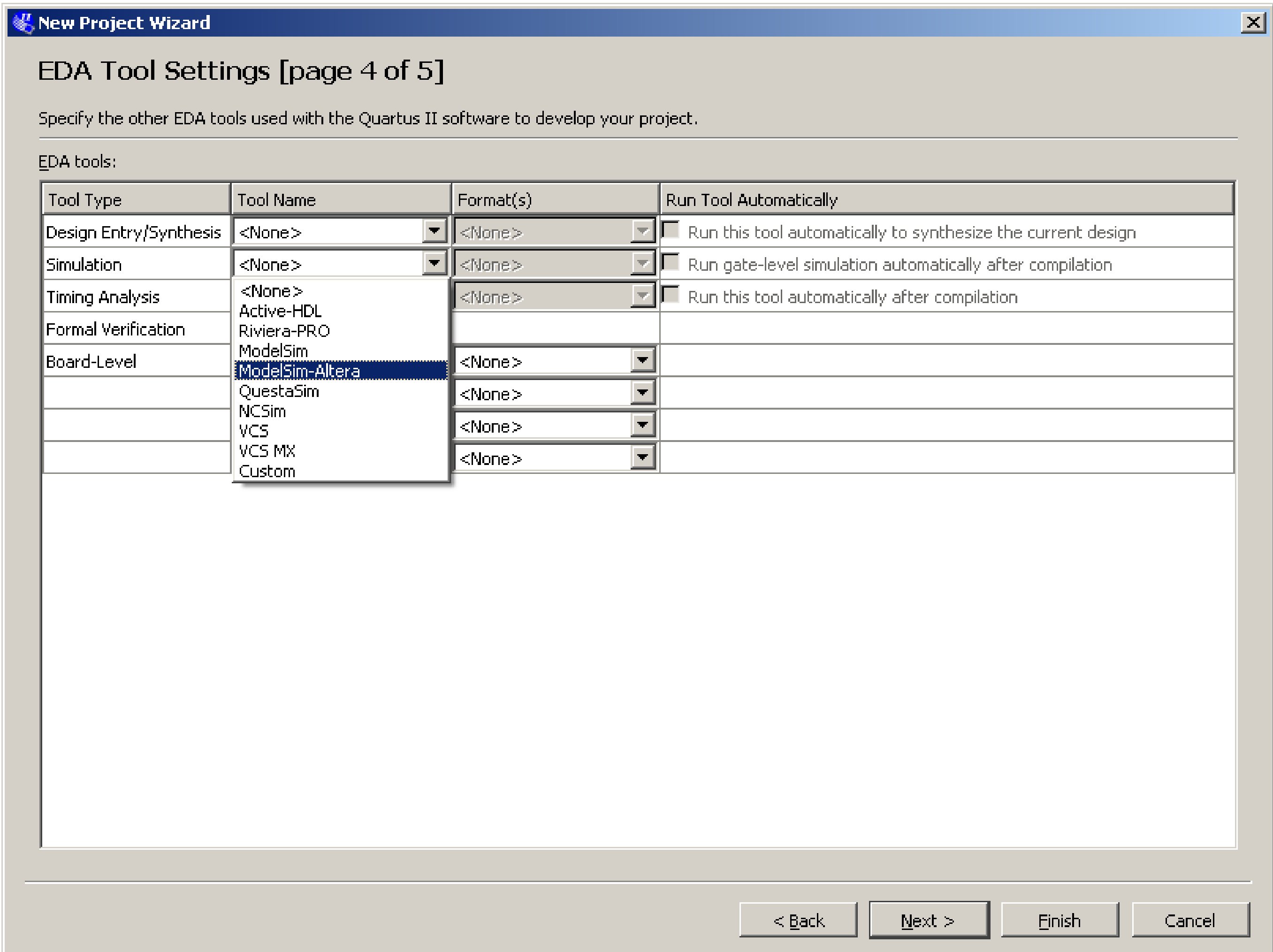


Fig. 3. Simulator selection

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

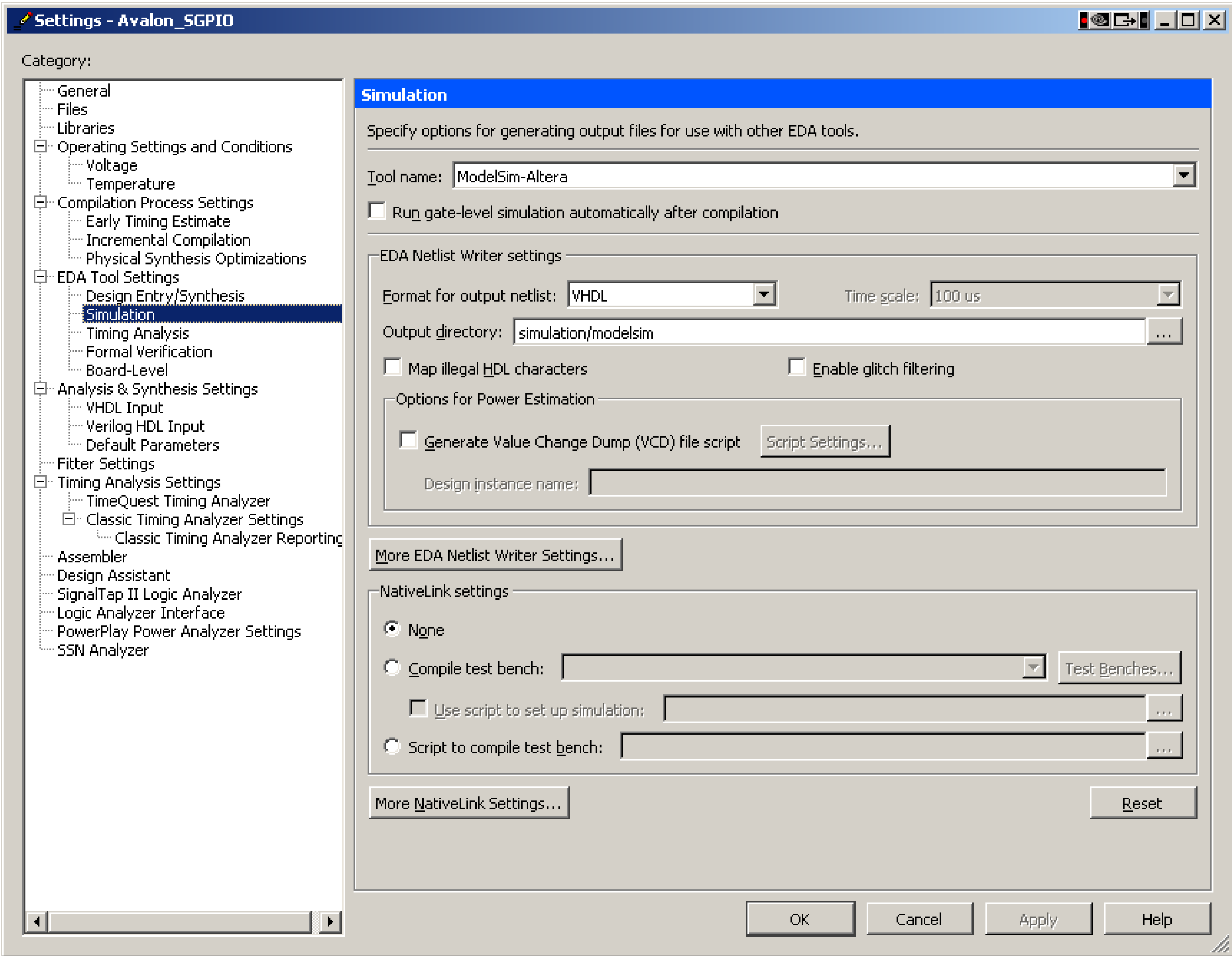


Fig. 4. Settings Tool selection for simulation

Tools → Options → General → EDA Tool Options

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

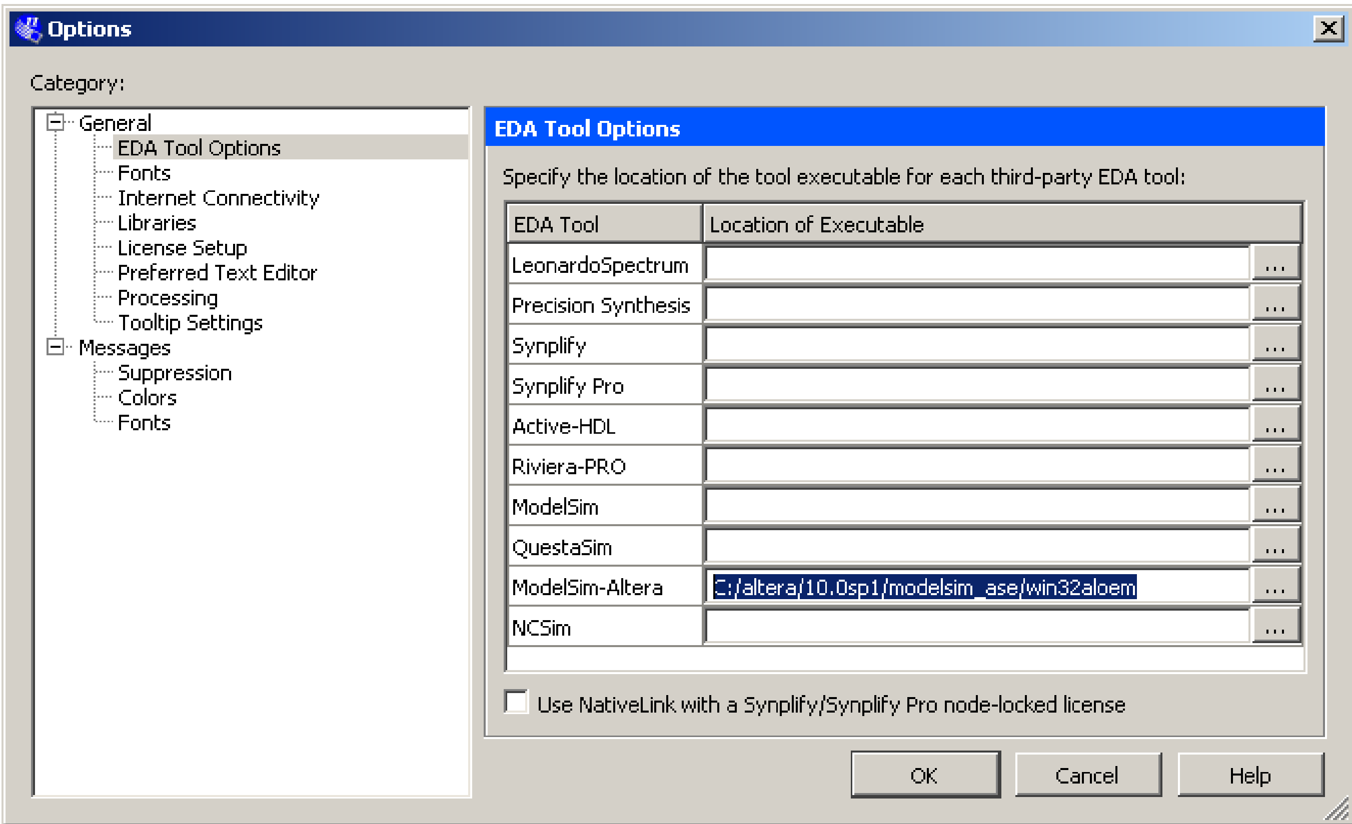


Fig. 5. Eda Tools path

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

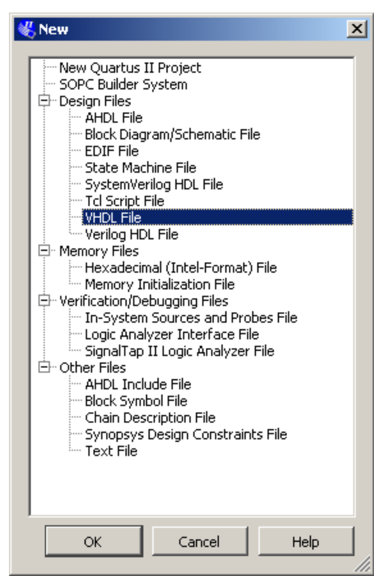


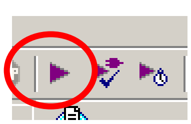
Fig. 6. Create VHDL File

```
-- 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***

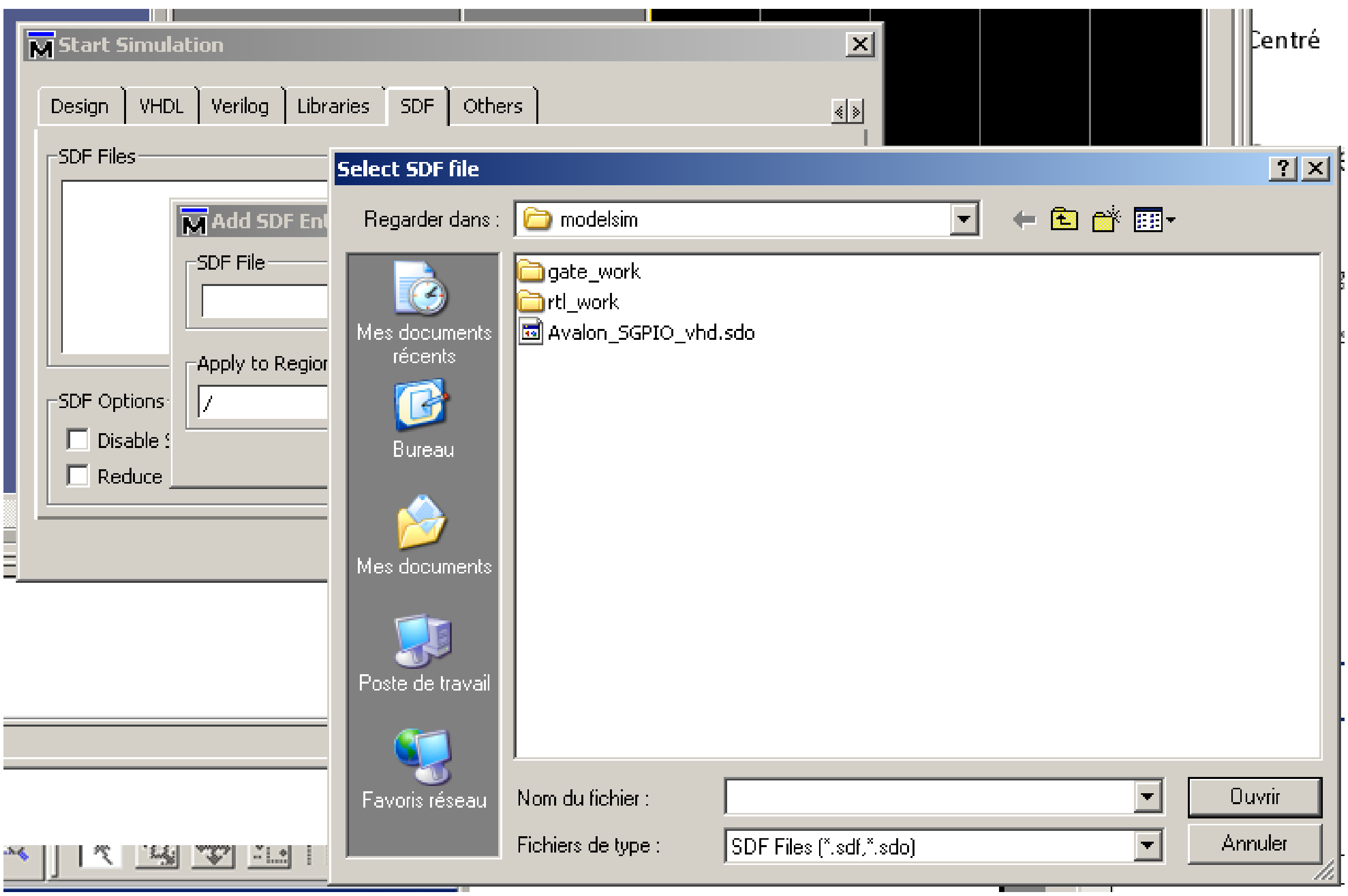


Fig. 7. Specify timing information

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

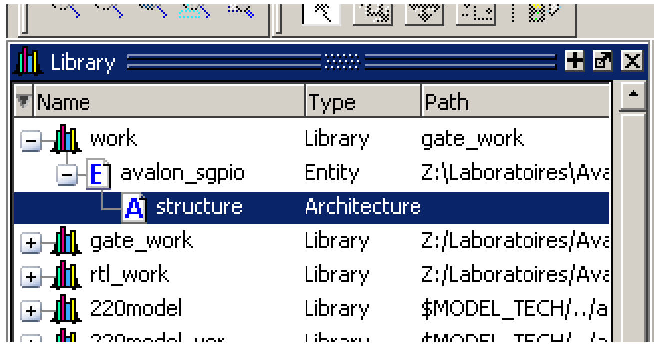


Fig. 8. File to simulate

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***

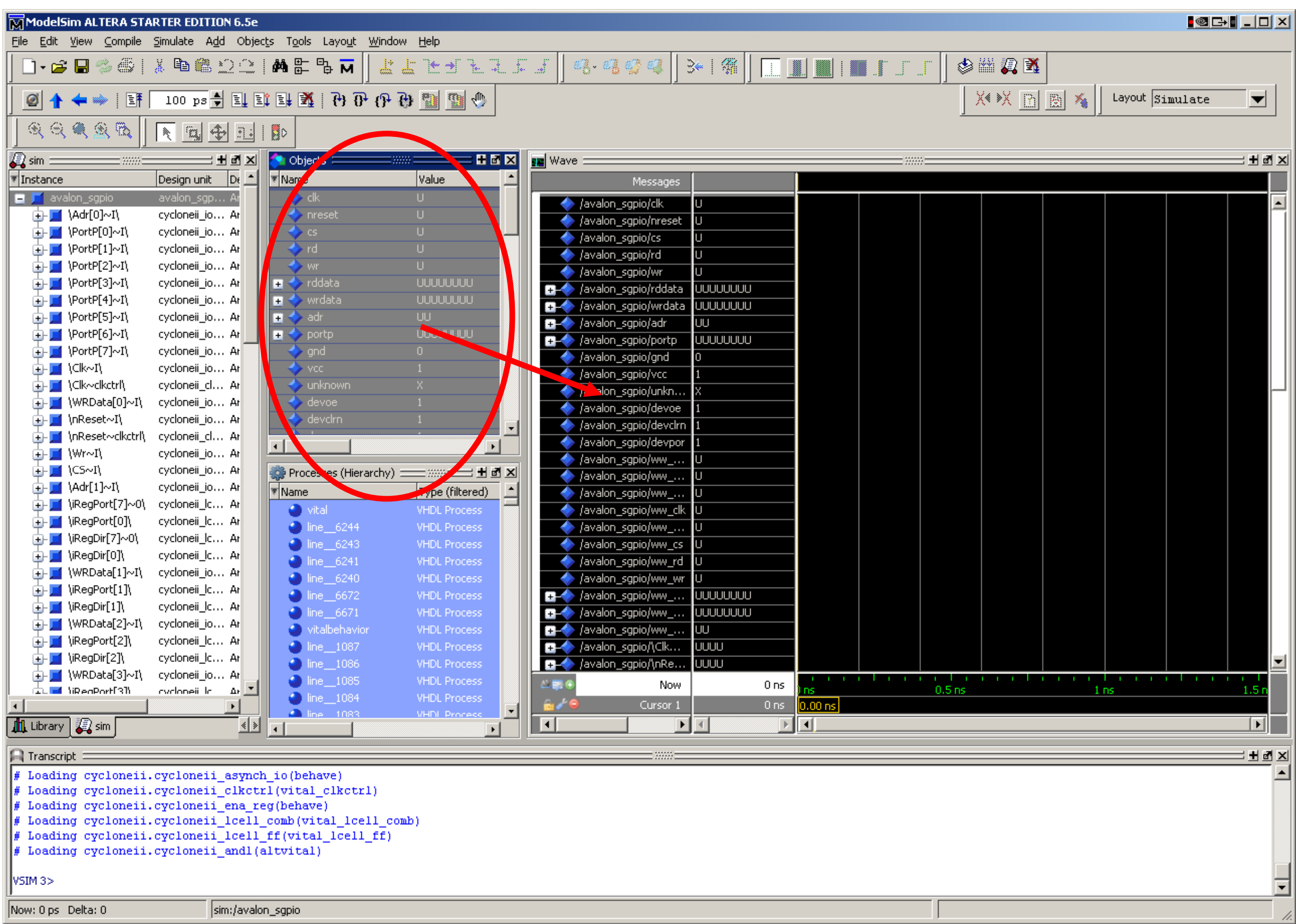


Fig. 9. ModelSim windows

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.

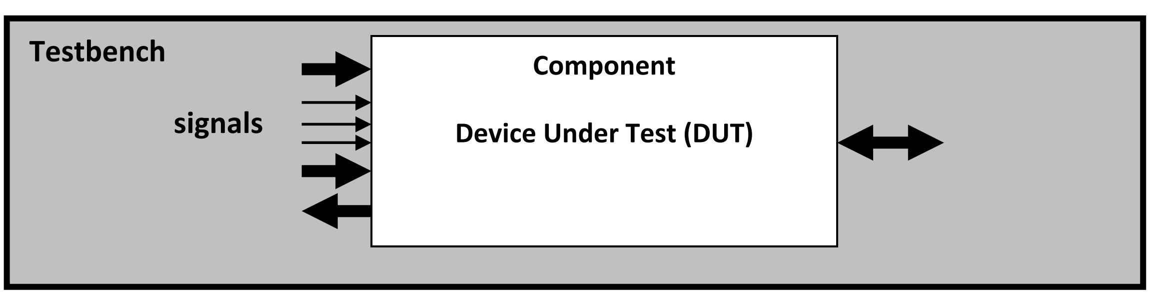


Fig. 10. Testbench structure

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: WrBus8, WrBus16, WrBus32
- a read procedure simulating a simplified Avalon slave read transfer: RdBus8, RdBus16, RdBus32

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. For each procedure there are 3 different versions for 8bit, 16bit, 32 bit data access. The user should use the one suitable for the corresponding hardware architecture.The procedures are proposed under the ways of a package and a package body. The VHDL files containing the packages are on moodle.

4.3.2 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
generic(
         N: natural := 3;
         L: natural := 8);
end testbench;


ARCHITECTURE bhv OF testbench IS

component ParallelPort is
     Port (
      Clk          : in std_logic;
      nReset       : in std_logic;
      Address      : in std_logic_vector(N-1 downto 0);
```

```
      ChipSelect   : in std_logic;
      Read         : in std_logic;
      Write        : in std_logic;
      ReadData     : OUT std_logic_vector (L-1 DOWNTO 0);
      WriteData    : IN std_logic_vector (L-1 DOWNTO 0);
      ParPort      : INOUT std_logic_vector (7 DOWNTO 0)
     );
   end component;

   signal Clk           : std_logic;
   signal nReset        : std_logic;
   signal Address       : std_logic_vector(31 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 (L-1 DOWNTO 0);
   signal SReadData     : std_logic_vector (L-1 DOWNTO 0);
   signal WriteData     : std_logic_vector (L-1 DOWNTO 0);
   signal ParPort       : std_logic_vector (7 DOWNTO 0);

   constant HalfPeriod : TIME := 10 ns;  --50 MHZ


BEGIN

UUT : ParallelPort

     Port MAP(
       Clk           => Clk,
       nReset        => nReset,
       Address       => Address(N-1 downto 0),
       ChipSelect    => ChipSelect,
       Read          => Read,
       Write         => Write,
       ReadData      => ReadData,
       WriteData     => WriteData,
       ParPort       => ParPort
     );

reset_process :process
   begin
   nReset <= '0';
   wait for 5 ns;
   nReset <= '1';
   wait for 10 us;
   end process;

clk_process :process
   begin
    clk <= '0';
    wait for HalfPeriod;
    clk <= '1';
    wait for HalfPeriod;
```

```
   end process;


waitreq_process: process

   begin
      wait until ChipSelect = '1';
      WaitRequest <= '1';
      wait for 10*HalfPeriod;
      wait until Clk = '1';
         WaitRequest <= '0';
   end process;


read_write:
PROCESS
   BEGIN
      wait for 50 ns;
   loop
      ParPort  <= "ZZZZZZZZ";
      WrBus8 ( 0, 255 , Address, WriteData, Clk,  ChipSelect, Write, Read, nBE,1,1,1,WaitRequest);
      wait for 50 ns;
      RdBus8 ( 0, SReadData , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,1, WaitRequest);
      wait for 50 ns;
      WrBus8 ( 2, 155 , Address, WriteData, Clk,  ChipSelect, Write, Read, nBE,1,1,1, WaitRequest);
      wait for 50 ns;
      RdBus8 ( 2, SReadData , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,1, WaitRequest);
      wait for 50 ns;
      WrBus8 ( 0, 0 , Address, WriteData, Clk,  ChipSelect, Write, Read, nBE,1,1,1, WaitRequest);
      wait for 50 ns;
      RdBus8 ( 0, SReadData , Address, ReadData,  Clk,  ChipSelect, Write, Read, nBE,1,1,1, WaitRequest);
      wait for 50 ns;
      ParPort  <= x"7d";
      RdBus8 ( 1, SReadData , Address, ReadData, Clk,  ChipSelect, Write, Read, nBE,1,1,1, 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