ECSE-487 Computer Architecture Laboratory
ModelSim and Leonardo Spectrum
Tutorial Manual, Version 1.0

TA: Hsin-Yun Yao, Prof: W.J.Gross

September 8, 2004

1 Introduction

This document provides the necessary information to get started with the design tools used in this course: ModelSim and Leonardo Spectrum. They are widely used in the industry for hardware design and are well made generally speaking. This document is written based on the tutorials and user guides listed in the reference section. Most of the documents can be found in the directory where the tool is installed. To learn more, you are strongly encouraged to explore the details of these tools from these documents.

1.1 Overview of Design Tools

Traditionally, in hardware design, the entries were done with schematics. The rapid developments in the manufacture of programmable gate arrays in the past years have made the use of hardware description language more and more extensive.

VHDL is a strongly typed hardware description language. Beside describing synthesizable hardware, it is also flexible enough for testbench design. To make the testing automated, testbenches often use features such as reading/writing text files, timing manipulation and scripting.

ModelSim is a simulation tool extensively used in industry because of the full support of the VHDL language and scripting capacity. On the contrary, Altera packages (MAX+Plus II and Quartus) do not provide the possibility of test benching. Only a subset of the VHDL language is supported, which imposes serious limitations on testbench writing. In addition, the functionality can only be tested after the synthesis stage, which can take up to 30 minutes for an average final project in this course.

Leonardo Spectrum is a logic synthesis tool by Mentor Graphics. It takes VHDL or Verilog design entries and produce vendor-independent and gate-level netlist output, as well as an estimation of timing and area consumption. It does not perform place and route, which would be provided by the hardware vendor such as Xilinx, Altera, and Lattice/Vantis. The synthesis tool does produce

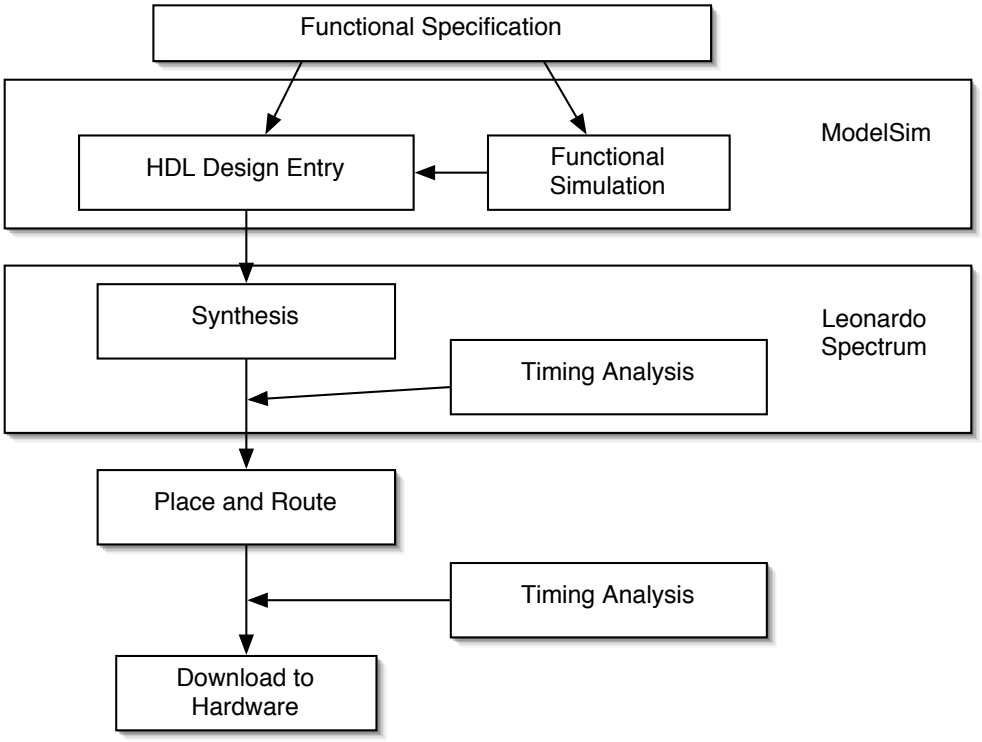

Figure 1: Programmable Logic Design Flow

an estimation of timing based on the hardware device of your choice, but it is rather rough and optimistic.

1.2 Scope of This Course

In this course, the design tool you need to learn in depth is ModelSim. For your assignments and the final projects, you should simulate the functionality of your design as throughly as possible with automated testbenches.

In order to show that your design can become a piece of hardware, you need to synthesize your device with Leonardo Spectrum. The scope of this course only covers the area and timing aspects in the hardware synthesis. However, feel free to explore more advance features in Leonardo Spectrum with on-line manuals listed at the reference section.

2 Getting Started with ModelSim

ModelSim provides both graphical and command line user interface. Although it takes some time to learn, the command line interface is fast and easily script-able. When you perform actions with GUI, most of the time the equivalent command will appear on the command-line prompt. In addition, you can append all the commands with ";" (semicolon) like in unix shell. You can also put the commands in a script file and execute it with "do" command.

2.1 Editing the Code

To edit VHDL files, you can use the editor provided by ModelSim, or any external code editors such as emacs or vim. ModelSim can understand both DOS and UNIX coded text files. Maybe you would prefer one with VHDL syntax coloring. The editor in ModelSim can sometimes be slow or unstable (i.e. crashes without reason), especially when the simulation is big.

This tutorial provides an example of an iterative 8-bit Multiplier. A testbench is also written to test the full functionality of the multiplier.

2.2 Creating a Project

ModelSim can be found under "FPGA Advantage" design suite. Once you start ModelSim, you can, but don't have to, create a project to contain your VHDL files.

GUI

- File -> New -> New Project
- Put the project in your home directory.
- You can add or create new VHDL files

2.3 Creating a Library

When you create a project, a default library "work" is also created for you in your project directory by default. The library is where ModelSim reads the data during simulation. If you put the library in your home directory, i.e. on the network, when the network is unstable, ModelSim may have trouble reading it. This is particularly annoying when the simulation is big. One alternative is to create a library locally, e.g. in the c:/temp to put the results of compilation.

GUI

- If you need to, right click on the default library to remove it.

- File -> New -> New Library
- The "Library Name" is what the simulator sees (example: work).
- The "Library Physical Name" is what it really is on the hard drive (example: c:/temp/487tutorial/work).

Command Line In command-line, you create a library first, then map it to "work":

```
> vlib c:/temp/487tutorial/work
> vmap work c:/temp/487tutorial/work
```

2.4 Compile Your Code

GUI

- Menu "Compile", or
- in "Project" tab, right click on the file(s), or
- Press the "Compile" button on the toolbar.

Command Line

```
> vcom h:/(path)/mult.vhd
> vcom h:/(path)/mult_tb.vhd
```

To compile a list of .vhd files in order, put the filenames in a file and use "vcom -f" command.

2.5 Starting and Stopping the Simulator

GUI

- Menu "Simulate", or
- in "Library" tab, right click on the entity, or
- Press the "Simulate" button on the toolbar.
- To stop the simulation, go to menu "Simulate -> End Simulation".

Command Line

```
> vsim work.mult_testbench
> quit -sim
```

2.6 Running Simulation

The simulator is the core of ModelSim. You can control the whole simulation through command line. Most options should be also be available through GUI menus and buttons.

GUI

- Menu "View -> All" to see all windows including Waveform, Signals, Variables, Processes, etc.
- From the "Signal" window, select the desired signals and drag them to the waveform window.
- If you don't have an automated testbench, you can force the signals in the "Signal -> Edit" menu. Follow the on-screen description for different options of forcing the signals.
- To run the simulation, press the button "Run". You can change the elapsed for each "run" at menu "Simulate -> Simulation Options".
- Right click on each signals for additional options.
- If you recompile the code, you need to restart the simulation by pressing the "Restart" button on the toolbar.

Command Line

- Adding the signals in the design and signals in the component (e.g. U1):

```
>add wave /*
>add wave U1/*
```

- If you don't have an automated testbench, you would need to force the signals. The first command forces the "clk" to the value "1" at 50 ns after the current time, then to "0" at 100 ns after the current time, and repeats cycle every 100 ns. The second command forces the "reset" to "1" after 50 ns, then to "0" after 200 ns.

```
>force clk 1 50, 0 100 -repeat 100
>force reset 1 50, 0 200
```

- To run the simulation, use the "run" command with the desired time, or run -all to run forever (this is useful when you need to single-step the code):

```
> run 1us
```

- One useful command to set everything in hexadecimal format:

```
> radix hex
```

- After you re-compile your code, you need to restart the simulation using "restart" command. "-f" option forces everything to be re-started.

```
> restart -f
```

2.7 Code Debugging

It is possible to put break points and single-step the code. Generally, it works like a software debugger. To stop the simulation, press the "Break" button on the tool bar. Note that if you have multiple instances of the same unit, the breakpoint may occur repeatedly (once for each instance).

GUI

- To put a break point, go to the Source window and click on the line number of the desired stopping point.
- To resume the simulation, press the "Continue Run" button, or use menu "Run -> Continue".
- To single step the code, press the "Step" button, or use menu "Run ->Step".

Command Line The following commands 1) put a break point; 2) resume the simulation; 3) single-step in the code:

```
> bp mult_tb.vhd (line number)
> run -continue
> run -step
```

3 Getting Started with Leonardo Spectrum

Synthesis is a complex process and requires in-depth knowledge of programmable logic. Leonardo Spectrum tool offers three different paths for logic synthesis, where "Level 1" provides a simplified flow for first-time users, and "Level 3", full control of all the advanced features. The scope of this course stops at "Level 1".

The example in the tutorial also comes with the testbench, which cannot be synthesized. Synthesize only the device itself.

3.1 Design Wizard

The easiest way to start using Leonardo Spectrum is with the Design Wizard. It hides most of the details of the synthesis process and provide a straightforward interface for first-time users.

- Start the wizard by pressing the "Wizard" button or from menu "Flows -> SynthesisWizard";
- Choose the desired hardware target;
- Setting the working directory. Since we do not work with the synthesizer tool for a long period of time, it can be your home directory without problem.
- In the Global Constraint window, put the desired clock frequency. Put to a higher value, up hundreds of Mhz, if you wish to test the limit of your design.
- The output file format and parameters do not matter if the design will not be downloaded to hardware.
- Click Finish to start the run flow.

One alternative way to the Design Wizard is to the "Quick Setup" Tab. In the tab, simply set all the appropriate parameters and press "Run Flow" button.

3.2 Schematic View

Once synthesized, you can see the schematic of your design by pressing the "Schematic Viewer" button. The RTL schematic view is the synthesized logic of your unit, and is independent of the hardware device you chose. It should give you the same circuit for Xilinx or Altera. The Technology view is displayed in function of the basic logic element of the hardware you chose. A different model of FPGA or ASIC gives you totally different views since the internal architecture of the various vendor's devices differ substantially. You can also see the critical path of you design. In this view, you can access options such as tracing forward/backward by right-clicking on the module or pin. This viewer provides important insights to logic synthesis, and you should always have a look and compare it with your estimation.

3.3 Optimization

With the "Optimization" tab, you can choose to optimize for speed or area. You should verify the results with "Schematic Viewer" and compare the difference. Sometimes when you optimize for a smaller area, the speed may actually increase. To understand this phenomenon, verify how the synthesizer maps the logic into LUTs.

3.4 Synthesis Report

After you synthesize the design for Altera Flex10K, you will get a report like this:

```
****************************************************

 Number of ports :                     36
 Number of nets :                     190
 Number of instances :                168
 Number of references to this view :    0


Total accumulated area :

 Number of CARRYs :                    19

 Number of DFFs :                      57

 Number of LCs :                       81

 Number of accumulated instances :    168
************************************************
Device Utilization for EPF10K70RC240
************************************************
Resource                Used    Avail   Utilization
------------------------------------------------
IOs                     36      189      19.05%
LCs                     81      3744      2.16%
DFFs                    57      4096      1.39%
Memory Bits             0       18432     0.00%
CARRYs                  19      3744      0.51%
CASCADEs                0       3744      0.00%


------------------------------------------------
                        Clock Frequency Report

    Clock                : Frequency
      ------------------------------------


    Clk_p                : 59.7 MHz


                        Critical Path Report



Critical path #1, (path slack = 33.3):

NAME                                    GATE              ARRIVAL              LOAD
------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                  0.00 (ideal)



reg_ShiftedMcand_s(0)/Q                 DFF         0.00  0.26 up             0.00
modgen_add_0_ix55/O                     CARRY2      0.42  0.68 up             0.00
modgen_add_0_ix59/O                     CARRY3      0.42  1.11 up             0.00
modgen_add_0_ix63/O                     CARRY3      0.42  1.53 up             0.00
modgen_add_0_ix67/O                     CARRY3      0.42  1.95 up             0.00
modgen_add_0_ix71/O                     CARRY3      0.42  2.38 up             0.00
modgen_add_0_ix75/O                     CARRY3      0.42  2.80 up             0.00
modgen_add_0_ix79/O                     CARRY3      0.42  3.23 up             0.00
```

```
modgen_add_0_ix83/O                    CARRY3     0.42  3.65 up            0.00
modgen_add_0_ix87/O                    CARRY3     0.42  4.08 up            0.00
modgen_add_0_ix91/O                    CARRY3     0.42  4.50 up            0.00
modgen_add_0_ix95/O                    CARRY3     0.42  4.93 up            0.00
modgen_add_0_ix99/O                    CARRY3     0.42  5.35 up            0.00
modgen_add_0_ix103/O                   CARRY3     0.42  5.78 up            0.00
modgen_add_0_ix107/O                   CARRY3     0.42  6.21 up            0.00
modgen_add_0_ix111/O                   CARRY3     0.42  6.63 up            0.00
modgen_add_0_ix113/O                   F3_LUT     3.48  10.12 up           0.00
ix939/Y                                MUX        0.00  10.12 up           0.00
nx1482/O                               F4_LUT     3.48  13.60 up           0.00
reg_Product_s(15)/D                    DFF        0.00  13.60 up           0.00
data arrival time                                       13.60


data required time  (default specified - setup time)       46.85
--------------------------------------------------------------------------------
data required time                                         46.85
data arrival time                                          13.60
                                                        ----------
slack                                                     33.25
--------------------------------------------------------------------------------
```

You can find the information about the area and the speed here. The first table tells you how the resources in the FPGA are used. For Flex10K, the basic logic elements are IO (input/output), LC(Logic Cells), DFF(D Flip-Flop), Memory, CARRY (carry propagation logic) and CASCADE. One thing worth mentioning is the CARRY chain. It represents physical cells in the FPGA dedicated to accelerating propagation of critical signals. For example, adders will most of the time have the critical path in the carry logic. Having dedicated logic on the FPGA to accelerate the propagation of the carry is essential for the maximum performance of the real circuit once it is placed and routed on the FPGA. Note that in order for carry chains to be used efficiently, the layout of the circuit in the FPGA is also very important. The scope of this course only covers up to the synthesis stage. For those interested in completing the design process, you can talk to the professor or the TA for additional information.

The next table tells you the speed and the critical path of your unit. The gates displayed are not the signals you may recognize because they are generated by the synthesizer. You can trace down the critical path in Schematic View. If you need to speed up your design, the critical path is what you should focus on. The last table tells you the "slack" you have compared to the desired speed of your circuit. If your desired speed is greater than the speed given by the synthesizer, the "slack" will be negative, indicating that you should optimize your critical path (by pipelining, re-ordering logic, etc.) to make the slack positive. A negative slack essentially means that the data do not have time to stabilize before the next clock edge samples the result at the next flip-flop level.

References

[1] Actel, "ModelSim Tutorial", http://www.actel.com/documents/oem_tutor.pdf

[2] Actel, "ModelSim User's Guide", http://www.actel.com/documents/oem_man.pdf

[3] Exemplar Logic Inc., "LeonardoSpectrum Users Guide", 1999

[4] Exemplar Logic Inc., "LeonardoSpectrum HDL Synthesis", 1999

[5] Peter J. Ashenden, "The Designer's Guide to VHDL", Morgan-Kaufman Publishers, 1995.

Appendix A Example Code: an Iterative Multpilier

```
-------------------------------------------------
-- Filename: mult.vhd
-- Title: Implementation of a iterative multiplier
-- Author: Hsin-Yun Yao, McGil University
-- Date: August 2004
-------------------------------------------------

library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

Entity mult is
port(
  -- Common ports
  Clk_p       : in  std_logic;
  Rst_p       : in  std_logic;
  -- Inputs
  Enable_p    : in  std_logic;
  Mcand_p     : in  std_logic_vector(7 downto 0);
  Mlier_p     : in  std_logic_vector(7 downto 0);
  -- Outputs
  ResValid_p  : out std_logic;
  Product_p    : out std_logic_vector(15 downto 0)
);
end mult;

architecture RTL_iter of mult is

signal Cnt8_s     : std_logic_vector(3  downto 0);
signal ShiftedMcand_s : std_logic_vector(15 downto 0);
signal ShiftedMlier_s : std_logic_vector(7  downto 0);
signal Product_s   : std_logic_vector(15 downto 0);
signal ResValid_s : std_logic;
begin

P1 : process( Clk_p )
begin
  if rising_edge(Clk_p) then
    if Rst_p = '1' then
      -- Reset everything to zero
      Cnt8_s      <= (Others => '0');
      ShiftedMcand_s  <= (Others => '0');
      ShiftedMlier_s  <= (Others => '0');
      Product_s    <= (Others => '0');
      ResValid_s  <= '0';
      Product_p <= (Others => '0');
    else

      -- Accumulate only if LSB of ShiftedMlier_s is '1'
```

```
      if ShiftedMlier_s(0) = '1' then
        Product_s <= ShiftedMcand_s + Product_s;
      end if;

      -- Shifting left, for the next round
      ShiftedMcand_s <= ShiftedMcand_s(14 downto 0) & '0';
      -- Shifting right, removing bits that are done
      ShiftedMlier_s <= '0' & ShiftedMlier_s(7 downto 1);

      if Cnt8_s = "1000" then
        ResValid_s  <= '1';
        Product_p    <= Product_s;
      else
        Cnt8_s      <= Cnt8_s + '1';
      end if;

      -- reads the data in only when Enabled
      if Enable_p = '1' then
        Cnt8_s      <= (Others => '0');
        ShiftedMcand_s  <= (Others => '0');
        ResValid_s  <= '0';
        ShiftedMcand_s(7 downto 0) <= Mcand_p;
        ShiftedMlier_s  <= Mlier_p;
        Product_s <= (Others => '0');
      end if;

    end if;
  end if;
end process;

-- Drive outputs
ResValid_p  <= ResValid_s;


end RTL_iter;
```

```
-------------------------------------------------
-- Filename: mult_testbench.vhd
-- Title: Automated testbench for the iterative multiplier "mult.vhd"
-- Author: Hsin-Yun Yao, McGil University
-- Date: August 2004
-------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Testbench has not inputs nor outputs
Entity mult_testbench is
end mult_testbench;


architecture behavioural of mult_testbench  is

-- This is the DUT (Device Under Test)
component mult
port(
  -- Common ports
  Clk_p       : in  std_logic;
  Rst_p       : in  std_logic;
  -- Inputs
  Enable_p    : in  std_logic;
  Mcand_p     : in  std_logic_vector(7 downto 0);
  Mlier_p     : in  std_logic_vector(7 downto 0);
  -- Outputs
  ResValid_p  : out std_logic;
  Product_p   : out std_logic_vector(15 downto 0)
);
end component;

-- Signals mapping to the DUT in the same order
signal clk_s: std_logic := '0';
signal rst_s: std_logic;
signal enb_s: std_logic;
signal mcd_s: std_logic_vector(7 downto 0);
signal mlr_s: std_logic_vector(7 downto 0);
signal val_s: std_logic;
signal prd_s: std_logic_vector(15 downto 0);

-- 1 -> result is correct, 0 -> incorrect
signal Result_Good : std_logic := '0';

begin
U1: mult port map ( clk_s, rst_s, enb_s, mcd_s, mlr_s, val_s, prd_s);

-- Generate clock and reset signal to facilitate simulation
clk_s <= not clk_s after 20 ns;
rst_s <= '1', '0' after 80 ns;
```

```
P1: process
begin
  -- This process runs every time the multiplier finishes its calculation.

  -- make sure everything is synchronous
  wait until rising_edge(clk_s);

  if rst_s = '1' then
      mcd_s <=(Others => '0');
      mlr_s <=(Others => '0');
      enb_s <= '0';
  else
      enb_s <= '1';

      -- increment mlier
      if mlr_s < 255 then
        mlr_s <= mlr_s + 1;
      else
        mlr_s <= (Others=>'0');
      end if;

      -- increment mcand only when the mplier finishes a round
      if mlr_s = "00000000" then
        if mcd_s < 255 then
          mcd_s <= mcd_s + 1;
        else
          mcd_s <= (Others=>'0');
        end if;
      end if;

      -- set enable to zero after one clock cycle
      wait until rising_edge(clk_s);
      enb_s <= '0';

      -- wait for the calculation to be finished
      wait until rising_edge(clk_s) and val_s = '1';

      -- check the results
      if prd_s = mcd_s * mlr_s then
        Result_Good <= '1';
      else
        Result_Good <= '0';
        -- The following line will print error on modelsim command prompt
        assert false report "incorrect" severity Error;
      end if;

end if;
end process;

end behavioural;
```

Appendix B VHDL Coding Convention

This coding guideline is based on the Appendix B of MAX+PLUS II Tutorial written for this course previously. A good coding style is as important as the code itself, and it doesn't take more time to do. It is not only beneficial for other people reading the code, but also important when you debug your own code. A good coding style is expected in your assignments and projects

- Each VHDL file will start with a consistent professional-style header.
- The name of the file should match the entity name with a .vhd suffix to facilitate easy location and maintenance.
- Use consistent indentation and correct spacing. Keep in mind that the tab space is not necessary the same in all editors. Keep lines shorter than around 80 characters and align colons and port maps.
- Write enough comments. Never comment self-evident statements.
- When applicable, the name of the architecture should be the same as the entity plus the modelling style. If the entity name is "toto" and the architectural modelling style is "structural", then the architecture name would be "toto_struct".
- In general, clock event is always to the rising edge, and direction of bits in a bus is always DOWNTO.
- The header for an entity with a structural description will refer the reader to a structural diagram in the report, unless the entity is trivial. The structural diagram will illustrate every internal component and every I/O of the entity. An entity with a behavioral description may not have a structural diagram. In this case, the entity must be a component of some higher level entity that does have a structural diagram containing this entity. The header will refer the reader to the appropriate higher level entity's structural diagram.
- For highly repetitive blocks, do not use signals to interconnect. Use port connections directly. This will minimize the amount of VHDL code and be easier to understand and maintain.
- Instantiations using the generate statement should be port mapped positionally, not using named association, i.e., no arrows.
- In an entity definition, the ports of the entity should be grouped and listed in the following sequence: inputs, outputs and control.