Tutorial on
FPGA Design Flow
based on
Aldec Active HDL

Ver 1.5
This tutorial assumes that you have basic knowledge on how to use ActiveHDL and its functional simulation. The example codes used in this tutorial can be obtained from http://ece.gmu.edu/coursewebpages/ECE/ECE448/S10/experiments/448_lab3.htm.

The current version of the tutorial was tested using the following tools:

CAD Tool
- ActiveHDL Version : 8.2
- Synthesis Tool
  - Synplicity Synplify PRO Version : 8.6
  - ISE&Webpack Synthesis&Implementation Version : 9.1
- Implementation Tool
  - Xilinx ISE/WebPack Version : 9.1

FPGA Board
- Digilent Basys2
Table of Contents

1 Project Settings 4

2 Synthesis
2.1 Synthesis using Xilinx XST
   2.1.1 Synthesis Options 7
   2.1.2 Synthesis Reports Analysis 8
2.2 Synthesis using Synplicity Synplify Pro
   2.2.1 Synthesis Options 11
   2.3.2 Synthesis Report Analysis 14
   2.3.3 RTL & Technology View 16
2.3 Post –Synthesis Simulation 21

3 Implementation
3.1 Implementation Options 22
3.2 Implementation Reports Analysis 23
3.3 Post-Implementation Simulation 26

4 Uploading Bitstream to FPGA Board 27
1. **Project Settings**

Start the workspace normally, but make sure you select *Create an Empty Design with Design Flow.*

Then press **Next**. You will see a picture similar to the one shown on the next page.
Verify that **Flow Settings** are defined as followed:

- **Synthesis Tool**
  - *ISE&Webpack Synthesis&Implementation* or
  - *Synplicity Synplify Pro.*

- **Implementation Tool**
  - *Xilinx ISE/WebPack 9.1*

- **Default Family**
  - *Xilinx9x SPARTAN3E*

If not, click at the **Flow Settings** button and adjust appropriately.

Also choose,

- **Block Diagram Configuration**
  - Default HDL Language
  - VHDL

Once done, select **Next→Finish**
Now you should see a familiar empty space with a Flow panel on the right side. If you do not see the Flow panel on the right side as shown in the picture, you can press **Alt+3** or **View → Flow** from the top menu bar to open the panel.

Specify the new design name. Download to your hard drive all VHDL files provided to you at the website for lab3 demo.

Add and compile all files from lab3 demo. Then, test your design if it works correctly in the functional simulation as you would normally do. If you are following the tutorial by using lab3demo, make sure you change the *slow_clock_period* located inside `Lab3Demo_package.vhd` to a number suitable for simulation. It will take a long time to simulate otherwise.
2. Synthesis

Synthesis can be done using two different tools: Xilins XST and Synplify Pro. The former can be used both at home and in school, the latter only in school. Please follow Sections 2.1 and 2.3 if you are using Xilinx XST, and Sections 2.2 and 2.3 if you are using Synplify Pro.

2.1 Synthesis using Xilinx XST

2.1.1 Synthesis Options

Click at the options button next to the synthesis icon. Under Synthesis Option select *Update synthesis order*. Arrange your files in the order from the bottom to the top of the design hierarchy. Exclude your non-synthesizable files, such as testbench. Also select a correct *Top-level Unit*, which is Lab3_demo in this example.

Make sure that your settings under General tab are as follows:

- **Family**: Xilinx9x Spartan3E
- **Device**: 3s100cp132
- **Speed Grade**: -4
Under **Std Synthesis** and **Adv Synthesis** tabs, you can adjust optimization goal of the synthesis tool for various results. Most notably, you can tell the synthesis tool to optimize for either *area* or *speed*. To select either one of them, choose **Std Synthesis → Optimization Goal → select Speed or Area**.

### 2.1.2 Synthesis Report Analysis

Minimum clock period, critical path and resource utilization can be found from the log file generated after synthesis. To view the log file, click at the *reports* button next to the Synthesis icon.

Minimum clock period, maximum frequency and critical path can be found under Timing Summary section. Looking at the critical paths can give you an idea of which portions of your code to change in order to improve the circuit performance.

Resource utilization is located in the Final Report section.

**Example Report: Resource Utilization**

```
# Final Report

Final Results
Top Level Output File Name: lab0_demo
Output Format: NGC
Optimization Goal: speed
Keep Hierarchy: no

Design Statistics
# I/Os: 9

Cell Usage:
# EILS: 139
# GND: 1
# INV: 7
# LUT1: 1
# LUT2: 4
# LUT3: 35
# LUT4: 9
# MUXCY: 43
# I/O: 1
# XORCY: 32
# FlipFlops/Latches: 37
# FDC: 37
# Clock Buffers: 1
# REGs: 1
# I/O Buffers: 9
# I/O BUF: 1
# DBUF: 7

Device utilization summary:

Selected Device: Csl500f5G20-4

Number of Slices: 29 out of 12312 0%
Number of Slices Flip Flops: 37 out of 26524 0%
Number of 4 input LUTs: 56 out of 26524 0%
Number of I/Os: 9
Number of bonded IOBs: 5 out of 221 4%
Number of OCEBs: 1 out of 6 12%
```
## Example Report: Minimum Clock Period and Critical Path

### Timing Detail:
- All values displayed in nanoseconds (ns)

### Timing constraint: Default period analysis for Clock 'clock'
- Clock period: 18.21ns (frequency: 55.39MHz)
- Total number of paths / destination ports: 1070 / 88

### Delay
- Source: clk_post_reset_clk شبكة (FF)
- Destination: clk_post_reset_clk شبكة (FF)
- Source Clock: clock rising
- Destination Clock: clock rising

### Path: slow_clock_gen/clock to slow_clock_gen/clock

<table>
<thead>
<tr>
<th>Cell</th>
<th>Source</th>
<th>Destination</th>
<th>Path</th>
<th>Delay (ns)</th>
<th>Logical Name (Net Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.511</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.508</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.505</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.498</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.495</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.492</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.489</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.486</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.483</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.480</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.477</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.474</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.471</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.468</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.465</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.462</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.459</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.456</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.453</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.450</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.447</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.444</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
<tr>
<td>FF</td>
<td>clk_post_reset_clk</td>
<td>clk_post_reset_clk</td>
<td>slow_clock_gen/clock</td>
<td>0.441</td>
<td>slow_clock_gen/clock_gen</td>
</tr>
</tbody>
</table>
2.2 Synthesis using Synplicity Synplify Pro

2.2.1 Synthesis Options

Click at the options button next to the synthesis icon. Under Synthesis Options select Update synthesis order. Arrange your files in the order from the bottom to the top of the design hierarchy. Exclude your non-synthesizable files, such as your testbench. Also select a correct Top-level Unit, which is Lab3_demo in this example.

Make sure that your settings under General tab are as follows:

Family : Xilinx9x Spartan3
Device : 3s100ecp132
Speed Grade : -4
Run Mode : GUI
Then, select **settings** tab and choose the frequency of your device to 50 MHz instead of Auto Constraint. Press **OK** and click at the **synthesis** button.

Select the first license and click Run to run the synthesis.
After synthesis, you can view the report by selecting view log button.
2.2.2 Synthesis Report Analysis

Minimum clock period (requested and estimated), slack (requested clock period minus estimated clock period), and resource utilization can be found from the log file generated after synthesis. To view the log file, click at the **reports** button next to the **synthesis** icon.

Minimum clock period can be found under *Performance Summary* section of the report. Respectively, one can determine the critical path by looking at the *Worst Path Information*. The report provides you with the 5 worst critical paths in your design. Looking at the critical paths can give you an idea of which portions of your code to change in order to improve the circuit performance.

Similarly, the resource utilization is located at the bottom of the log file. The report tells you the amount of resources the FPGA needs for the design.

**Example Report: Timing**

<table>
<thead>
<tr>
<th>Worst clock in design: 13.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Clock</td>
</tr>
<tr>
<td>Lab3_demo.clock</td>
</tr>
</tbody>
</table>

**Example Report: Resource Utilization**

```
### END OF TIMING REPORT ###

---------------------------------------------
Resource Usage Report for Lab3_demo

Mapping to part: xc3s1000csp132-4

Cell usage:
FDC 37 uses
GND 2 uses
MUXCY 1 use
MUXCY_L 37 uses
VCC 2 uses
XORCY 31 uses
LUT1 32 uses
LUT2 12 uses
LUT3 1 use
LUT4 17 uses

I/O ports: 9
I/O primitives: 9
IBUF 1 use
IBUFG 1 use
OBUF 7 uses

BUFG 1 use

I/O register bits: 0
Register bits not including I/Os: 37 (1%)

Global Clock Buffers: 1 of 24 (4%)
```
Example Report: Worst Path Information

Worst Path Information
View Worst Path in Analyst
************************

Path information for path number 1:
- Requested Period: 20.000
- Setup time: -0.339
- Clock delay at ending point: 0.000 (ideal)
- Required time: 20.339
- Propagation time: 6.595
- Clock delay at starting point: 0.000 (ideal)
- Clock (critical): 13.814

Number of logic level(s): 25
Starting point: slow_clock_gen.counter[0] / 0
Ending point: slow_clock_gen.counter[25] / D
The start point is clocked by Lab3_dcm.clock [rising] on pin C
The end point is clocked by Lab3_dcm.clock [rising] on pin C

<table>
<thead>
<tr>
<th>Instance / Net</th>
<th>Type</th>
<th>Pin</th>
<th>Pin</th>
<th>Delay</th>
<th>Arrival</th>
<th>No. of Fan Out(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>slow_clock_gen.counter[0]</td>
<td>RC</td>
<td>0</td>
<td>Out</td>
<td>0.592</td>
<td>0.592</td>
<td>-</td>
</tr>
<tr>
<td>counter[0]</td>
<td>Net</td>
<td>-</td>
<td>-</td>
<td>0.980</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>slow_clock_gen.un4_countercry_1</td>
<td>MUXCY_L</td>
<td>CI</td>
<td>in</td>
<td>-</td>
<td>1.572</td>
<td>-</td>
</tr>
<tr>
<td>slow_clock_gen.un4_countercry_1</td>
<td>MUXCY_L</td>
<td>LO</td>
<td>Out</td>
<td>1.236</td>
<td>2.808</td>
<td>-</td>
</tr>
<tr>
<td>un4_countercry_1</td>
<td>Net</td>
<td>-</td>
<td>-</td>
<td>0.000</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>slow_clock_gen.un4_countercry_2</td>
<td>MUXCY_L</td>
<td>CI</td>
<td>in</td>
<td>-</td>
<td>2.808</td>
<td>-</td>
</tr>
<tr>
<td>slow_clock_gen.un4_countercry_2</td>
<td>MUXCY_L</td>
<td>LO</td>
<td>Out</td>
<td>0.065</td>
<td>2.373</td>
<td>-</td>
</tr>
</tbody>
</table>
### 2.2.3 RTL & Technology View

Investigation of the internal structure of your design after synthesis can be done by looking at the RTL and Technology views of your circuit.

RTL view is the schematic representation of the design in terms of generic logic components that are independent of the target technology (specific Xilinx FPGA), for example, in terms of multiplexers, adders, comparators, registers, counters, and logic gates.

Technology view is the schematic representation of the design in terms of components available in the target technology (specific Xilinx FPGA), for example, in terms of LUTs, flip-flops, fast carry logic, I/O blocks.

Hence, Technology view generally has more detailed/bigger diagram than RTL view. Viewing either one of them can be done by using Synplify Pro, which can be opened using **RTL schematic** button of the Flow panel.

Below is the basic layout of Synplify Pro.
Once open, selecting *.srs file will open RTL view of your design. Similarly, selecting *.sms file will open Technology view of your design.
RTL View:
Navigating through the components can be done by a right-click on any blank area and selecting Push/Pop Hierachy. Your mouse icon should now change from a cross sign to arrow sign, allowing you to click and navigate through the component, if possible.

Investigating the critical path of your circuit can be done by selecting HDL-Analyst → Technology → Hierarchical Critical Path. Once clicked, a page similar to the one below will be shown for lab3 demo.
A zoomed diagram of LUT3_01 is shown below. The red number on top of the component is showing delay and slack time of this circuit.

Lastly, clicking directly on the component in this view can take you straight to the vhdl source code.
2.3 Post-Synthesis Simulation

Click refresh file list next to the synthesis icon. Drill down into the directories and select the .edf file for the netlist and the .vhm file for the simulation file.

Click at the options button next to the post-synthesis simulation icon. Remove the default input file, and select your testbench as an input file by clicking at the button close to the cross sign (marked by a dot). Then, select Recompile Files. Once done, choose the appropriate top-level unit, which is lab3demo_tb.vhd in this example.

Press OK, and then select post-synthesis simulation. Now you should see timing waveforms similar to the ones obtained during functional simulation. The difference is that the components and signals are now mapped into appropriate FPGA hardware.
3 Implementation

3.1 Implementation Options

Click at the options button next to the implementation icon. Select the correct Netlist File which is a file with the same name as your top level VHDL file and the extension .edf. It is normally located in the synthesis folder of your workspace. Use this file to implement your design. Choose the correct FPGA Family, Device and Speed Grade, the same as used during the Synthesis phase:

In our example these are:
Family : Xilinx9x Spartan3E
Device  : 3s100ecp132
Speed Grade : -4
Under **Constraint File**, select **Custom constraint file**. Browse to your .ucf for the lab, lab3_demo.ucf in our example. Then, navigate to the **BitStream** tab by clicking at the right arrow at the top right hand corner. Under **General** tab of **BitStream** deselect **Do Not Run Bitgen**. This will create bitstream, .bit, which you can upload to FPGA.

Also, under **Post-Map STR**, **Post-PAR STR**, and **Simulation** tabs make sure that your device speed grade is set to 4.

Similar to synthesis option for Xilinx XST as synthesis tool, you can specify the implementation tool to use a certain optimization goal. To do this, go to **Advanced Map** → **Optimization Goal** → select either **Area** or **Speed**.

Press **OK**, and then select **implementation**.

### 3.2 Implementation Reports Analysis

Similarly to synthesis, you can access the generated reports by clicking the **reports** button, near the **implementation** icon. Unlike synthesis log, implementation log is divided into several smaller reports, which are named differently. Below is a list of reports in which you can find the most useful information about your design after implementation, such as resource utilization, maximum clock frequency, and critical path:

**Resource Utilization:**
- **Map**: See **Design Summary**
- **Place & Route**: See **Device Utilization Summary**

Note: Place & Route provides overall information about the design after placing and routing. Map provides a more detailed summary of resource utilization.

**Minimum Clock period (Maximum Frequency):**
- **Post-Place & Route Static Timing Report**
  This file describes the worst case scenario in terms of minimum clock period. However, since the implementation tools do not provide complete information, please refer to **Timing Analysis** below for a more detailed report.

Note: Post-Map Static Timing Report can be ignored because it provides timing report before placing & routing, and thus cannot correctly predict interconnect delays.

**Pad file** provides the mapping between FPGA pins and ports of your top-level unit (obtained based on the user constraint file .ucf). Please double check this report before running your design on the FPGA board.

**Example**: Mapping between the FPGA pin P10 and the clock input of the Lab3_Demo unit; the two neighboring pins P9 and P11 are marked as **UNUSED**

```
P9 | DIFFM | IO_L32P_5/GCLK0 | UNUSED | 5 | | | | | | | |
P10 | clock | IOB | IO_L32P_4/GCLK0 | INFUT | LVCMOS25 | 4 | | | | | | |
P11 | DIFFM | IO_L3JP_4/D3 | UNUSED | 4 | | | | | | |
```
Example Report: Minimum Clock Period

Clock to Setup on destination clock clock
--------------------------------------------
| Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
--------------------------------------------
clock | 6.870|
--------------------------------------------

Example Report: Resource Utilization

Device Utilization Summary:
Number of BRAMs: 1 out of 0 12%
Number of External IOBs: 9 out of 221 4%
Number of LOCed IOBs: 9 out of 9 100%
Number of Slices: 30 out of 13312 1%
Number of SLCs:

Design Summary
-----------------
Number of errors: 0
Number of warnings: 0
Logic Utilization:
Number of Slice Flip Flops: 31 out of 26,624 1%
Number of 4 input LUTs: 31 out of 26,624 1%
Logic Distribution:
Number of occupied slices: 30 out of 13,312 1%
Number of Slices containing only related logic: 30 out of 30 100%
Number of Slices containing unrelated logic: 0 out of 30 0%
*See NOTES below for an explanation of the effects of unrelated logic
Total number of 4 input LUTs: 56 out of 26,624 1%
Number used as logic: 31
Number used as a route-thru: 25
Number of blocked IOBs: 9 out of 221 4%
Number of GCLKs: 1 out of 0 12%

Total equivalent gate count for design: 564
Additional CAT6 gate count for IOBs: 432
Peak Memory Usage: 150 MB
Total CPU time to MAP completion: 4 secs
Total CPU time to MAP completion: 1 secs

Timing Analysis (Clock period, Maximum Frequency and Critical Path)

For the detailed analysis of critical path and minimum clock period (or maximum frequency) a separate timing analyzer provided by Xilinx should be used. To generate the report, select Analysis → Static Timing Analyzer from the Flow panel. This will open Xilinx Timing Analyzer. You can also navigate to the program from Windows menu by Start → All Programs → VLSI Tools → Xilinx ISE → Accessories → Timing Analyzer.
Once the program is opened, select **Open**, choose netlist file located in `/implement/ver1/rev1` of your workspace, *nhd*, and press **OK**. Selecting **Analyze against Auto Generated Design Constraints** will generate a static timing report.

Example Report: Clock period, Maximum Frequency and Critical Path

```
Timing constraint: Default OFFSET OUT AFTER analysis for clock 'clock_c'

34 items analyzed, 0 timing errors detected.
Maximum allowed offset is 13.75ns.

Offset: 13.75ns (clock path + data path + uncertainty)
Source: .Glob(s1) (P25)
Destination: .Glob(s2) (P15)
Source Clock: clock_c rising
Data Path Delay: 11.75ns (Levels of Logic = 3)
Clock Path Delay: 2.016ns (Levels of Logic = 2)
Clock Uncertainty: 0.000ns

Clock Path: naming/counting/count s(100)
Delay type   Delay(ns)   Logical Resource(s)
---          ------       ------------------------------
Tpipe        0.629       clock
net (fncnt=1) 0.001      clock.buf/1MBFG
TJ10        0.402       clock.buf/1MBFG
net (fncnt=19) 0.798     clock_c
Total         2.016ns (11.200ns logic, 0.000ns route)

Data Path: counting/count s(1) to G Sep(1)
Delay type   Delay(ns)   Logical Resource(s)
---          ------       ------------------------------
Tnado        0.750       counting/count s(1)
net (fncnt=12) 1.661    count(10)
Tnlo        0.600       seven_sum 0/1/1 1d12_1 12 2
net (fncnt=2) 0.505     seven_sum 0/1 48
Tnlo        0.600       seven_sum 0/1 5 1
net (fncnt=1) 2.442     N 1 1
Tnino        5.131       seven_sum 16 16 16
Total        11.735ns (7.067ns logic, 4.668ns route)
```

(61.6% logic, 39.8% route)
3.3 Post-Implementation Simulation

Click at the options button next to the timing simulation icon. Select your testbench as the Top-Level Unit. Afterwards, select timing simulation, which will generate timing waveforms based on your netlist after implementation. You should notice slight timing delays compared to the waveforms from your post-synthesis simulation & functional simulation.
4. Uploading Bitstream to FPGA Board

Before uploading Bit file, make sure that you change your constant values in all your files to proper values, and re-synthesize/re-implement all the files. In particular, in our example, please change the value of the constant slow_clock_period in the Lab3Demo_package.vhd.

Select the Adept program as shown in the picture above. When the program is opened, a device will be shown if it is connected and recognized. Select the bit file by clicking Browse and finding the appropriate file. Click Program to program the file device.
Good luck! Have fun debugging =)