ECE 554 – Digital Engineering Laboratory

FPGA Design Tutorial

Version 3.1 - Fall 2001

Surin Kittitornkun and Charles R. Kime

Table of Contents

Changes to V.3.0

- Post-synthesis simulation has been added.
- Flow diagrams have been added.

1. Introductions and Preparation

The FPGA design flow can be divided into the following stages:

1. Design Entry
   a) Performing HDL coding for synthesis as the target (Xilinx HDL Editor)
   b) Using Cores (Xilinx Core Generator)
2. Functional Simulation of synthesizable HDL code (MTI ModelSim)
3. Design Synthesis (FPGA Express)
4. Design Implementation (Xilinx Design Manager)
5. Timing (Post Implementation) Simulation (MTI ModelSim)

This design flow is based on the assumption that the student:

1. is familiar with HDL coding using either Verilog HDL or VHDL. See Appendix A: Design Wizard and Language Assistant, and
2. recognizes the difference between HDL coding for synthesis and for simulation. See Appendix B: Syntax checking for HDL Synthesis vs. HDL Simulation.

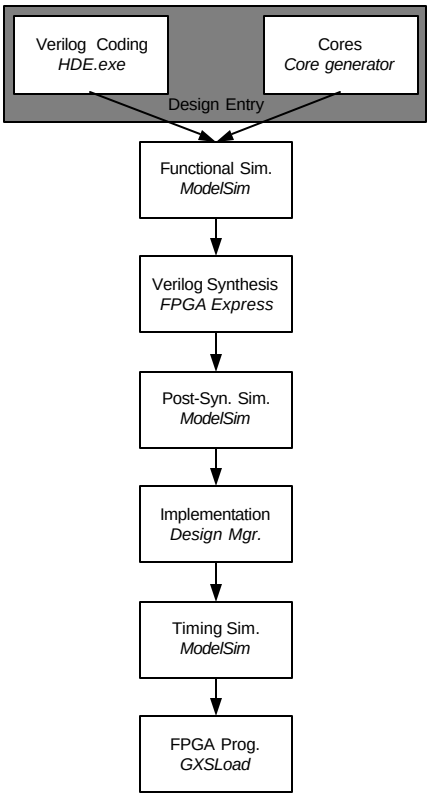
The final ECE 554 project usually contains a big/complex subsystem, which takes a long time to synthesize (>10 min.) and is unchanged or infrequently changed during the system debugging loop. See Appendix C: Incremental Design Synthesis for further details.

Preparation

- Log on an NT workstation (in 3628 Engineering Hall only).
- Create "I:\xilinx\tutorial\mac" and "I:\xilinx\tutorial\cores" directories
- Copy files from http://www.cae.wisc.edu/~ece554/s01/mac and http://www.cae.wisc.edu/~ece554/s01/cores respectively.

File name	Detail
Mltring.v	The top most file contains the "mltring" module and other interfaces.
Memctrl.v	A simple memory controller
mac.v	The top-level file contains "mac_test" module. (MAC = Multiply-ACcumulate)
Mltring.ucf	User constraint file contains port names and their corresponding pin location assignments.
force.do	Script file to simulate "mac.v" in ModelSim (functional)
Timing_force.do	Script file to simulate "mltring.v" in ModelSim (timing)
mult_4x4.*	4-by-4 bit multiplier core: .v for functional simulation, .edn for netlist
reg8b.*	8-bit register core: .v for functional simulation, .edn for netlist

2. Design Entry



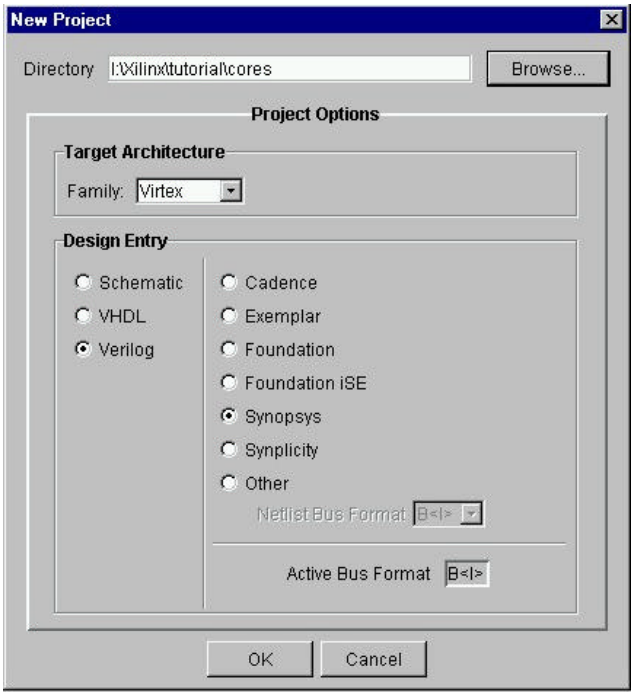
Since you're required to do the projects in HDL (hardware description language) only, the HDL editor is provided. All the keywords are highlighted depending on the file extension either "*.vhd" for VHDL or "*.v" for Verilog HDL.

- **Using HDL Editor**
  - Open the HDL Editor by double clicking on the **HDE.exe** icon on the desktop.
  - Open the top-level file: "mac.v" in "I:\xilinx\tutorial\mac" and try to understand the structure of the multiply-accumulator. You must be able to draw a simple diagram with some useful details described by this HDL code and show it to the instructor.
  - Open "mltring.v" in the same directory and find where the "mac_test" module is instantiated at a specific line number to the instructor.
- **Using Cores from Core Generator**

The pre-designed functional units such as adder/subtractor, multiplier, divider, etc. are available in such a way that they can be customized for a particular use. These functional units are called "core". They can be customized and generated using "Xilinx Core Generator" which can be used in the following procedure.

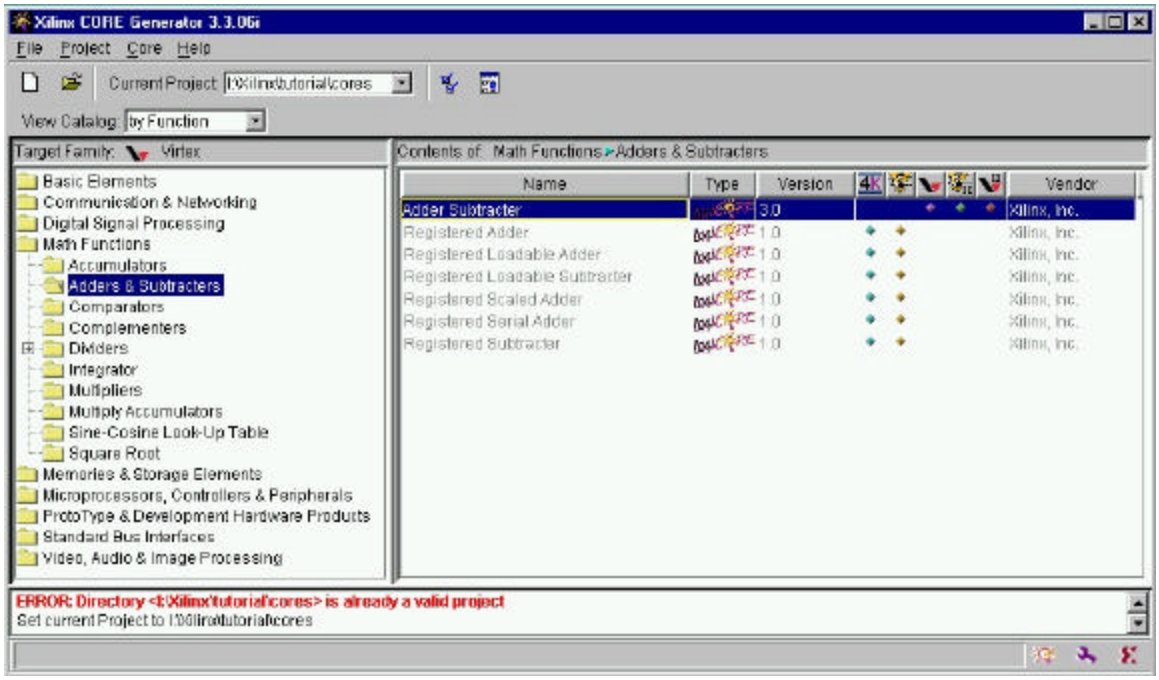
- Launch Core Generator:

- Select **Start=>Programs=>Xilinx Foundation Series 3.1i=>Accessories=>CORE Generator System**
- Select “Create a new project,” then click OK
- Create a new project in “I:\xilinx\tutorial\cores” as shown in the figure below and click OK.



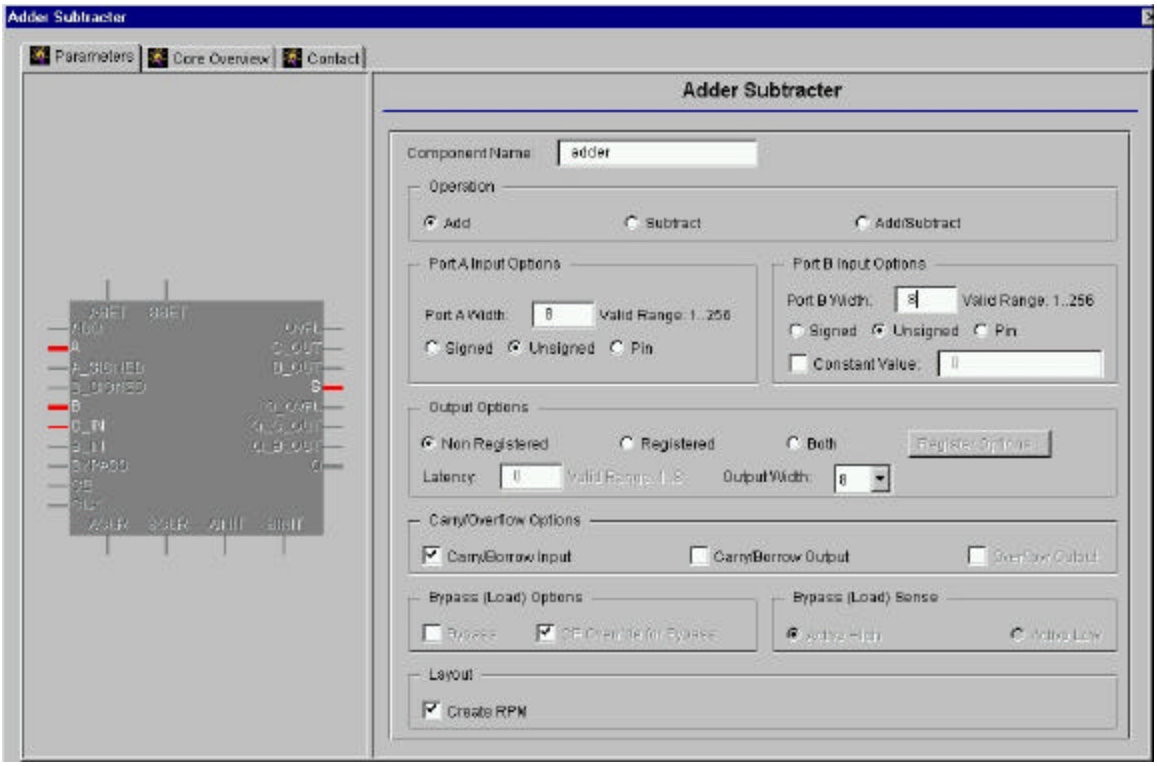
(Coregen_newproject.jpg)

- Make sure that you read the message and understand it and Click OK
- Double click on “Math Functions”=>”Adders & Subtractors”=>”Adder Subtractor” as shown.



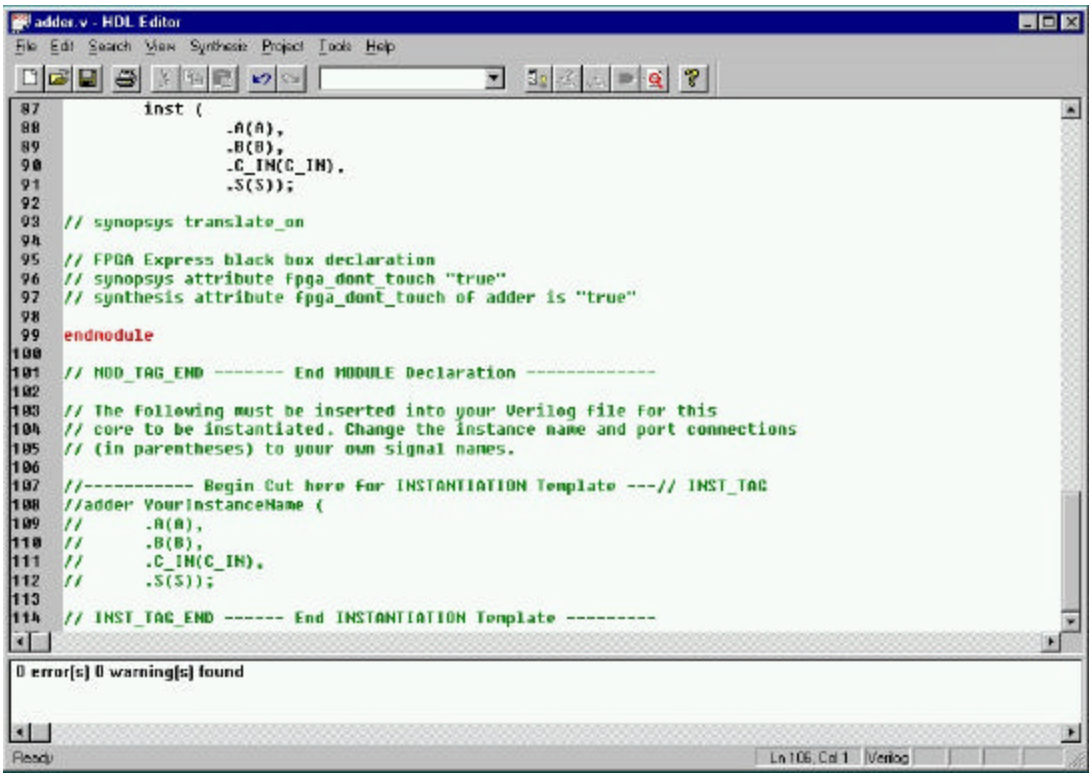
coregen_math_add.jpg

- Double Click “Adder Subtractor.” You will see the following display. Note that the core type is called “logicore” version 3.0 and the vendor is Xilinx, Inc.



coregen_add_sub8.jpg

- Cores from other vendors are usually unavailable unless we pay for them.
- Customize and generate the core
  - Input the parameters exactly as shown then above click “Generate” button.
  - Use Window Explorer to view “I:\xilinx\tutorial\cores” directory. You should see the following files: *adder.edn*, *adder*. *veo*, and *adder.xco*, as generated by Core Generator System.
  - Copy “adder.veo” and paste it in the same directory.
  - Rename “copy of adder.veo” to “adder.v” and use the HDL Editor to open it.
  - Locate this statement “`` `include "XilinxCoreLib/C_ADDSUB_V4_0.v" ``” at the beginning of the file. It indicates that “adder.v” includes `C_ADDSUB_V4_0.v` in “D:\xilinx\fndtn\verilog\src\XilinxCoreLib\” at compile time. This is a behavioral model used in simulation.
  - Scroll down to the bottom of the file and put “//” in front of every line below “endmodule” like this. Note that “//” is the comment symbol in Verilog HDL.

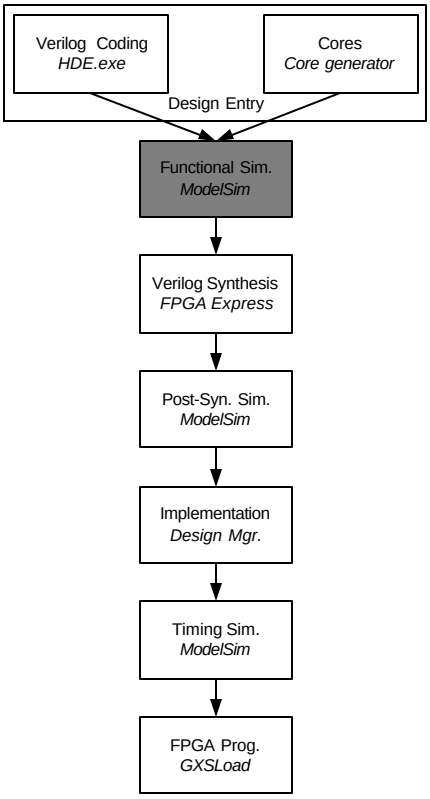


Coregen_rename.jpg

- Note that “adder” instantiation that you commented out has already been put in “mac.v”.
- The “adder.v” file you just edited is the behavioral model for the adder core to be used in functional simulation to be performed next.

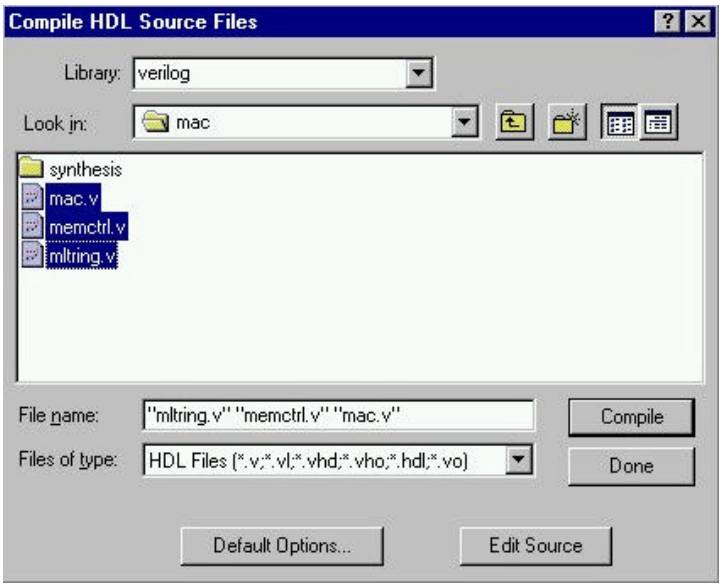
- Quit HDL Editor
  - **File=>Exit**

3. Functional Simulation (ModelSim)



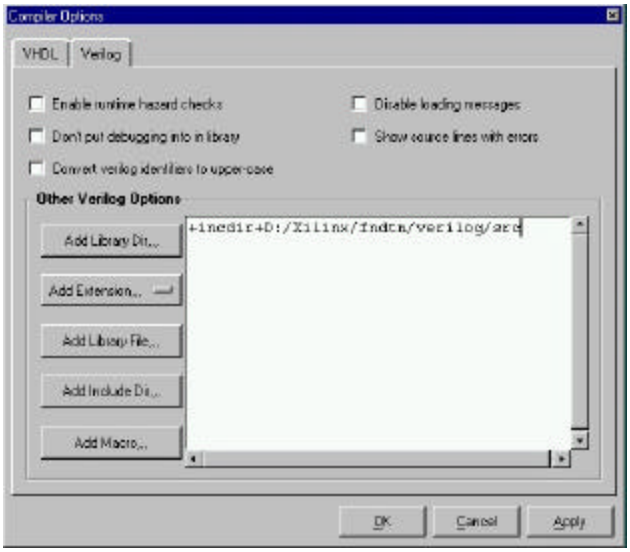
This step is equivalent to software debugging. The HDL simulator (debugger) is provided for both *functional* and *timing* simulations. The HDL simulator is "ModelSim" from ModelTech, Inc. It has both a graphic user interface (GUI) and scripting capability.

- Launch ModelSim.
  - **Start=>Programs=>ModelSim SE 5.5d=>ModelSim**
- Change the directory to the top-level of the design that we want to simulate.
  - **File=>Change Directory** and browse to "I:\Xilinx\tutorial\mac" then click **Open**.
- Compile all the design (source) files to a specific library which, in particular, is the "Verilog" library.
  - **Design=>Compile**
  - Change the "Library" to "Verilog"
  - Hold "Ctrl" key and click on "mltring.v", "memctrl.v", and "mac.v" to compile all Verilog files
- Make sure that the library is set to Verilog.



modelsim_compile.jpg

- Compile "cores" generated by Core Generator
  - Navigate the "Compile HDL Source Files" window to "I:\xilinx\tutorial\cores".
  - Push the "Default Options" button and select "Verilog" tab as shown next.
  - Type in "+incdir+D:/xilinx/fndtn/verilog/src" as shown to specify the include directory of the available cores in Verilog HDL format. Note that ModelSim will recognize "/" as "\" internally.
  - Launch "Window Explorer" to navigate to D:\xilinx\fndtn\verilog\src\XilinxCoreLib". You can see all the behavioral models for the cores.

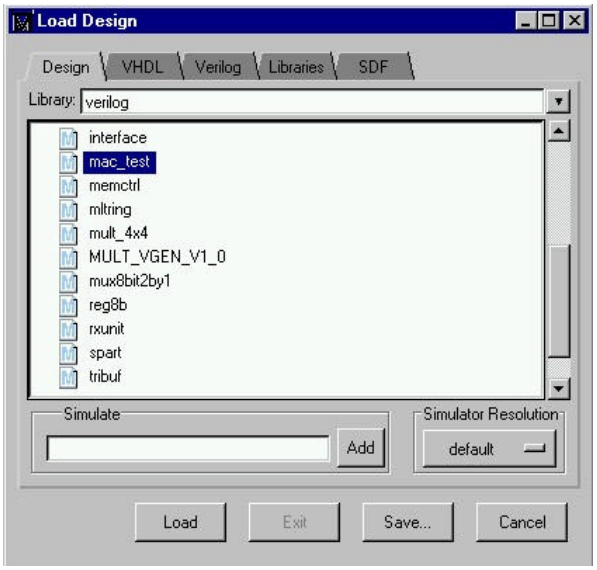


Modelsim_compile_include.jpg

  - Note that typing this string of characters is equivalent to clicking "Add Include Dir" and browsing to "D:\Xilinx\fndtn\verilog\src\".
  - Push Apply and OK buttons respectively.

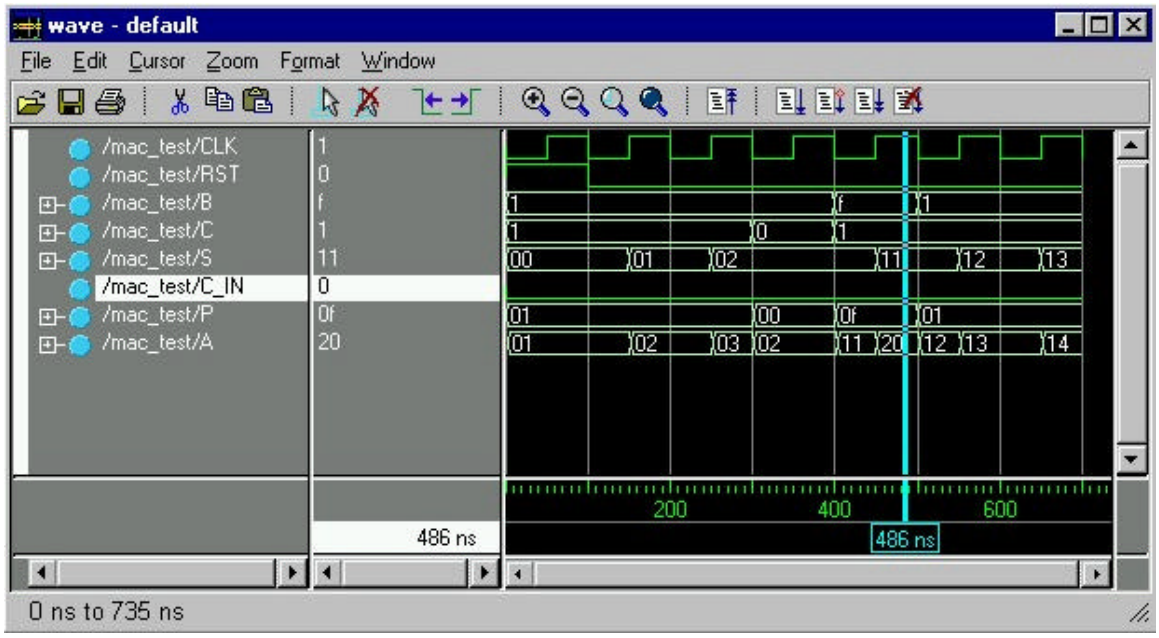
- Select and compile the following files one by one or all together: “adder.v”, mul4x4.v”, and “reg8b.v”. Click “Done” button if ModelSim finishes the compilations.
- In the mean time, you should look at messages listed on the ModelSim main window. Some are the actual commands reflecting what you did to the GUI and some are warning and error messages.

- Load the top-level (mac_test) design
  - **Design=>Load Design**
  - Choose source Library=”Verilog”
  - Click on “mac_test” module and then click “Load” to load the “mac_test” module.



Modelsim_load_design.jpg

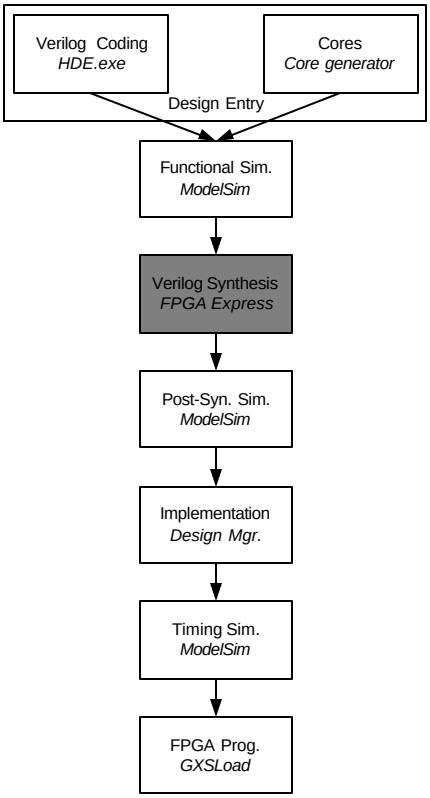
- Create a simulation environment, and apply stimuli to “mac_test” module from a script file “force.do”.
  - **Macro=>Execute Macro**=>force.do
  - Open “force.do” and understand what it does.
- Functional verification
  - The waveform in the “wave” window should look similar to this figure.
  - The module just stimulated is a multiply-accumulator with two unsigned 4-bit inputs, b and c. The product P gets added (accumulated) to the current register value S every positive clock edge.



Modelsim_waveform.jpg

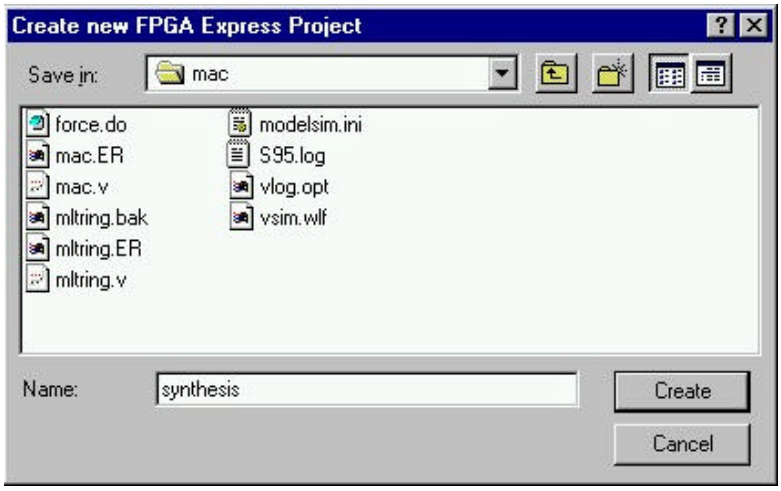
- Quit the simulation
  - At ModelSim main window type "`quit -sim`" to quit the simulation only.
  - Minimize the main ModelSim window.

4. Design Synthesis (FPGA Express)



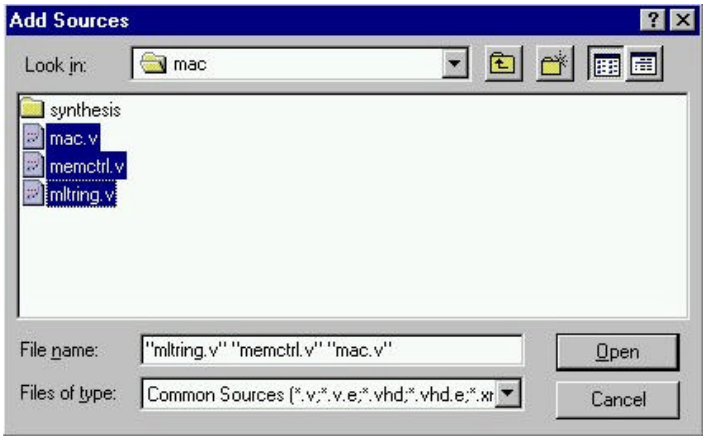
After we get the correct functionality of our top-level ("mac_test") module, we must convert these top-level design files and all generated cores to the programming file for the FPGA. The first step is called *design synthesis.* In ECE554, we use FPGA Express as our synthesis tool.

- Launch FPGA Express
  - **Start=>Programs=>Xilinx Foundation Series 3.1i=>Accessories=>FPGA Express**
- Create a new project
  - **File=>New Project**
  - Navigate to "I:\xilinx\tutorial\mac"
  - Type in the project name "synthesis" and click the "Create" button as shown below.
  - Note that the "synthesis" directory is created by FPGA Express to hold its project file "synthesis.exp" as well as our synthesized hardware (netlist file: "*.edf").



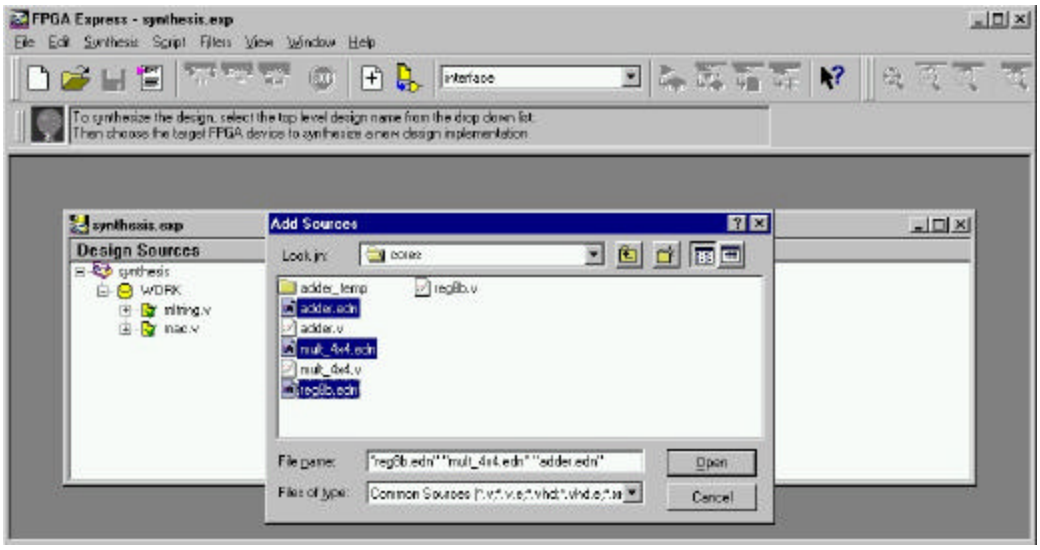
fpga_new_project.jpg

- Add source (design) files to the "synthesis" project
  - Hold "Ctrl" key and select all "*.v" files in "I:\xilinx\tutorial\mac" directory and then click **Open** as shown.



Fpga_add_sources.jpg

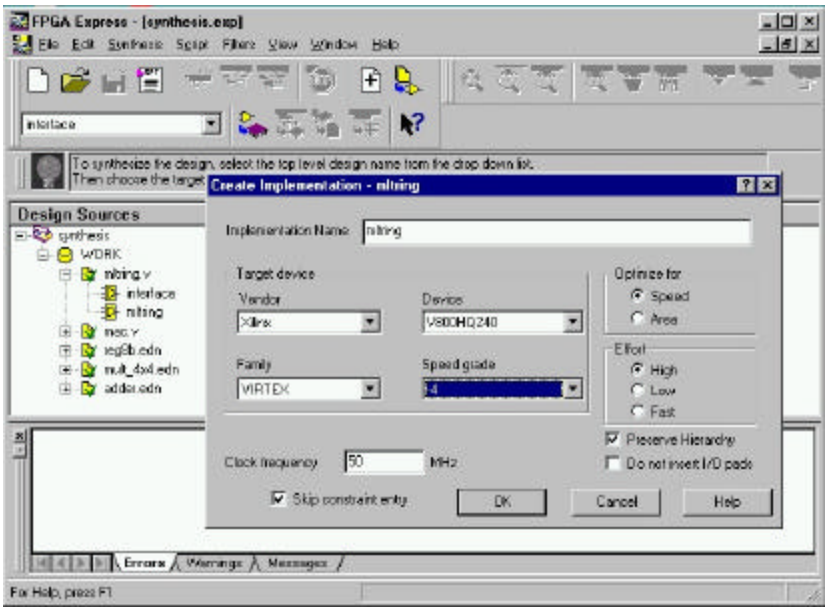
- Add more source files (cores)
  - Click on the "WORK" library symbol.
  - **Synthesis=>Add Source**
  - Navigate the window to "cores" directory
  - Hold "Ctrl" key and select the following files "adder.edn", "mult_4x4.edn", and "reg8b.edn" as shown.



Fpga_add_edif.jpg

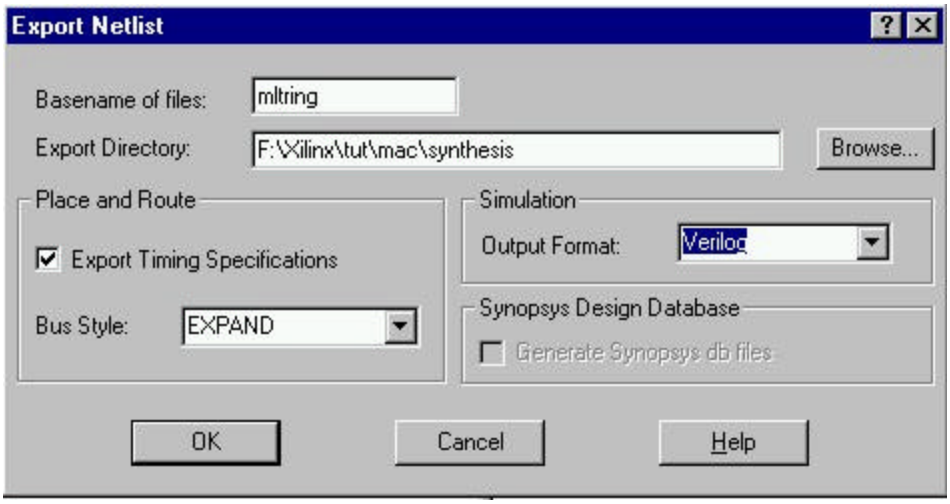
Now all the design (source) files have been entered and analyzed by the FPGA Express. The check mark will appear to indicate that each file is correct according to the FPGA Express. If not, the cross mark will appear instead. The next step is to synthesize the top most ("mltring") module.

- Synthesize the top-level ("mltring") module
  - Make sure that all the sources have a "check" mark.
  - Expand the "mltring.v" and select "mltring" module.
  - **Synthesis=>Create Implementation**
  - Fill in the following parameters as shown then click OK.
  - Note that the **Target device** is as follows:
    - Family: VIRTEX (Xilinx Virtex FPGA)
    - Device: V800HQ240 (800,000 gates and 240-pin package)
    - Speed grade: -4



Fpga_synthesis.jpg

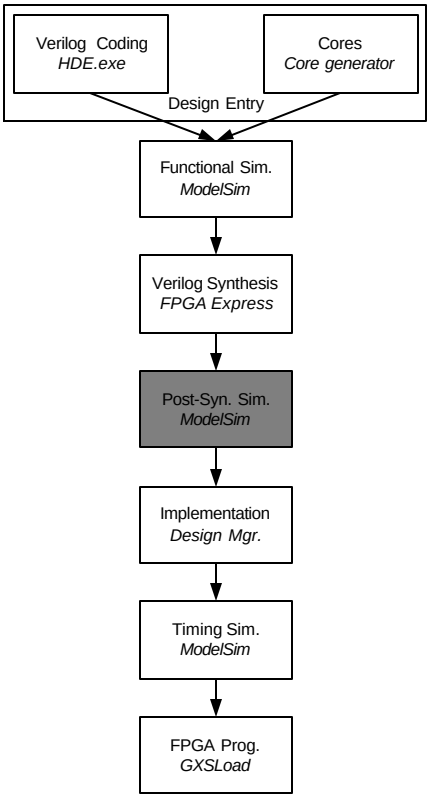
- View the schematic (optional).
  - Select “mltring-Optimized”
  - **Synthesis=>View Schematic**
- Export the netlist file (“mltring.edf”)
  - Select “mltring-Optimized”
  - **Synthesis=>Export Netlist**
  - Note that the output file is named “mltring.edf” in “I:\xilinx\tutorial\mac\synthesis\” directory as shown.



Fpga_export_netlist.jpg

- Minimize the FPGA Express window.

5. Post-synthesis Simulation

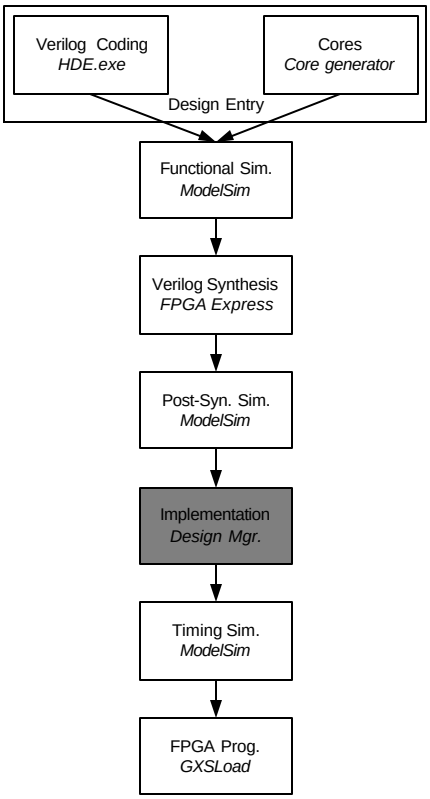


- In the same directory I:\xilinx\tutorial\synthesis\, you should be able to locate “mltring.v”.
- Double-click on “mltring.v” to open by HDL Editor.
- The file header should contain the following statements.

```
// Synopsys FPGA Express automatically generated file
// This file will be overwritten by each chip export
// Author: kittitor
// Program: FPGA Express
// Version: 3.5.0.5831
```

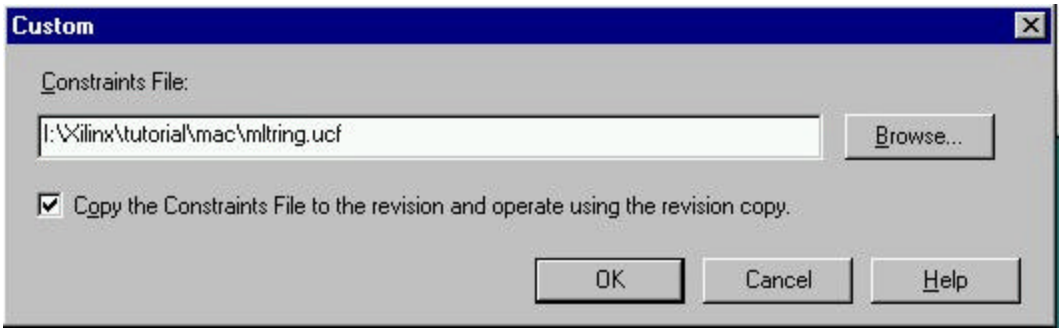
- This “mltring.v” was also generated by FPGA Express along with the “mltring.edf”.
- We can use this file for “post-synthesis” simulation.
- Launch ModelSim again and change the directory to “I:\xilinx\tutorial\synthesis\”.
- Compile “mltring.v” to Verilog library.
- Load “mltring” module and Execute Macro in “force_timing.do”.
- Compare the current waveforms to the ones from functional simulation.
- Let the TAs know whether they are the same.

6. Design Implementation (Design Manager)



The second step in producing the programming file for the FPGA is Design Implementation. In this process, netlists (*.edf) are translated, followed by mapping, placement, and routing. Finally, the corresponding hardware configuration bit stream (*.bit) for programming the FPGA is generated. Design implementation is extremely important to the success of your project. Please pay careful attention to the errors and warnings that show up, consult with the instructors to investigate the causes if necessary, and perform the fixes needed.

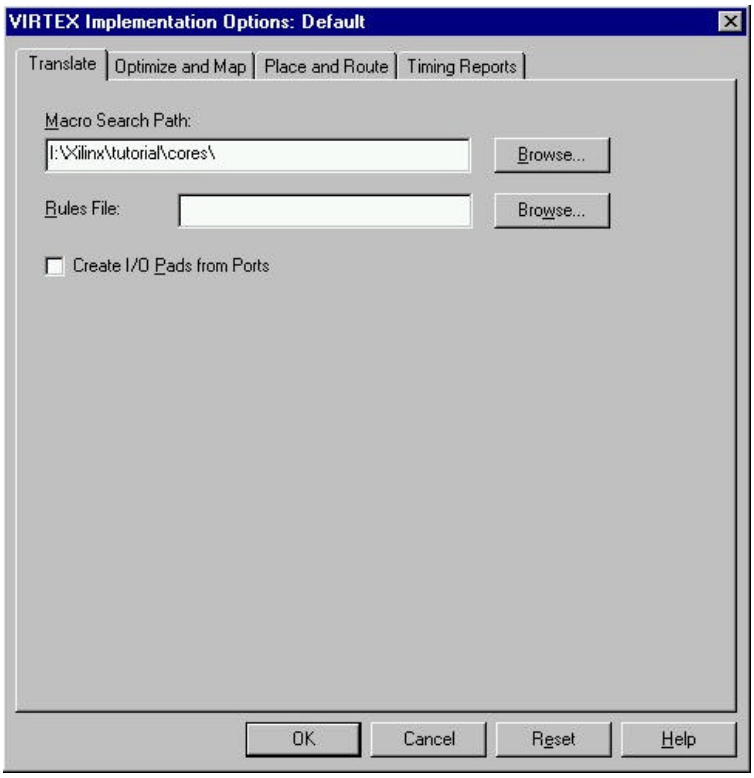
- Launch Design Manager
  - **Start=>Programs=>Xilinx Foundation Series 3.1i=>Accessories=>Design Manager**
- Create a new project
  - **File=>New Project**
  - Select "Input Design"=>Browse to "I:\xilinx\tutorial\mac\synthesis\mltring.edf"
  - Note that the "Working Directory" is "I:\xilinx\tutorial\mac\synthesis\xproj" and then click OK.
  - Xilinx Design Manager creates the "I:\xilinx\tutorial\synthesis\xproj" directory specified as the "Working Directory".
- Add a Universal Constraint File (UCF).
  - "Constraints File"=>Custom=>Browse to "I:\xilinx\tutorial\mac\mltring.ucf"



dmgr_ucf.jpg

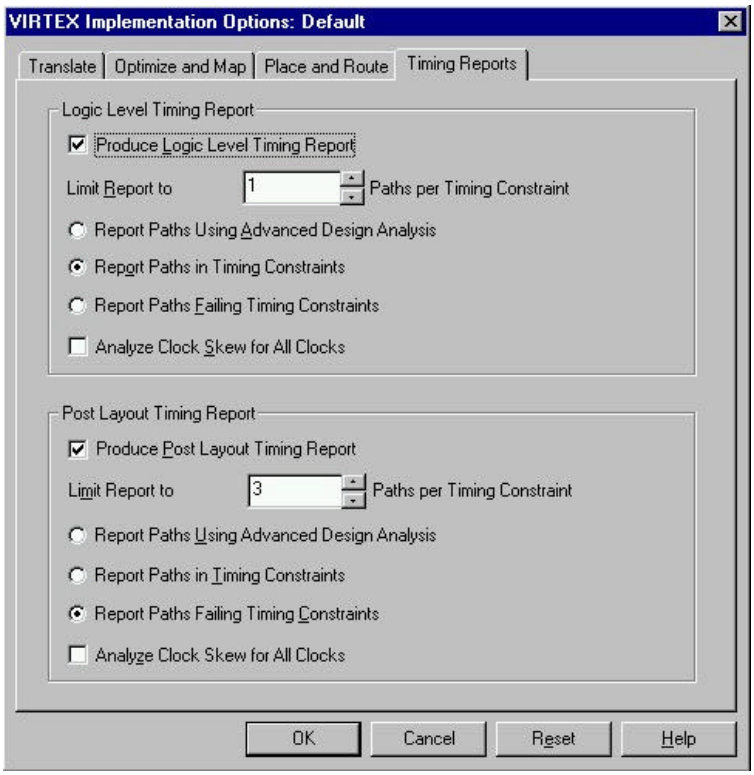
WARNING: Failure to use this file or other specified files in every Xilinx FPGA design could cause active outputs to be connected together and result in costly damage to the FPGAs or other lab components!

- Specify implementation options.
  - Click on "**rev1 (New, OK).**"
  - Click **Select Design=>Options**.
  - At "Program Options, Implementation" Click "**Edit Options**".
  - Select "Translate"=>Macro Search Path=>Browse to "I:\xilinx\tutorial\cores\," and click OK.



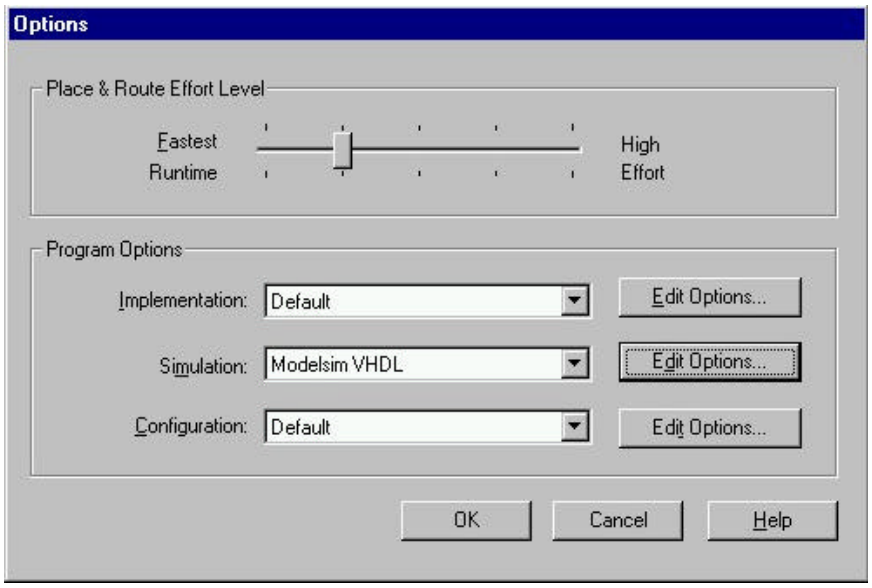
dmgr_translate.jpg

- Produce timing reports for performance estimation
  - Click on "Timing Reports"=>check "Produce Logic Level Timing Report"



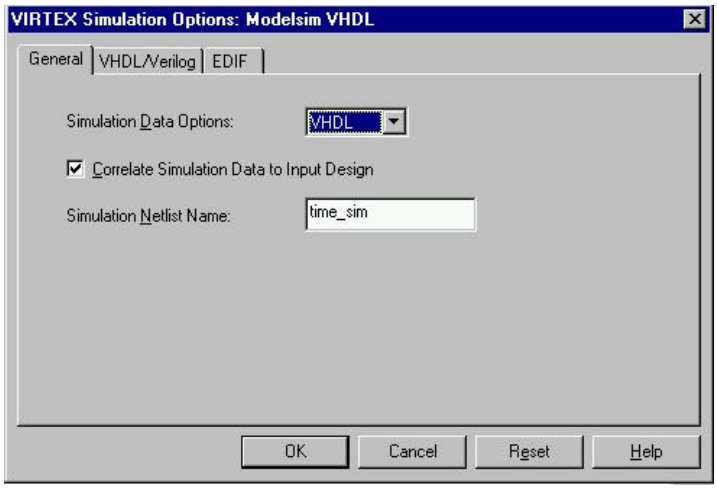
dmgr_timing_reports.jpg

- Specify simulation options to generate output file for the final timing simulation stage in order to verify the functionality whether it meets the specification.
  - "Simulation"=>Choose "ModelSimVHDL"=>Click "Edit Options"



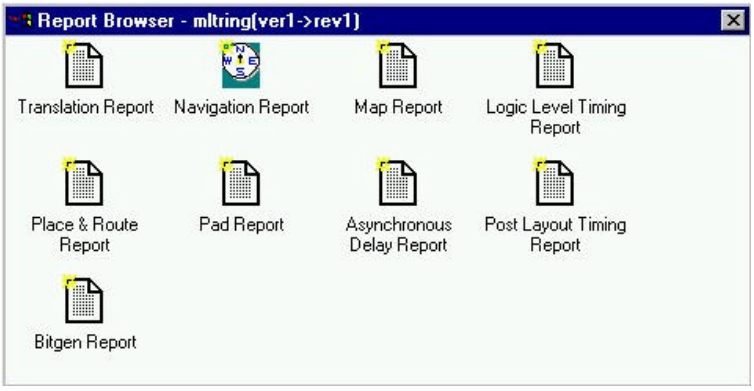
dmgr_simulation_options.jpg

  - Click on "General"=>Check "Correlate Simulation Data to Input Design"
  - Note that the output file will be named "time_sim.vhd" according to the chosen "ModelSim VHDL" choice.



dmgr_simulation_correlate.jpg

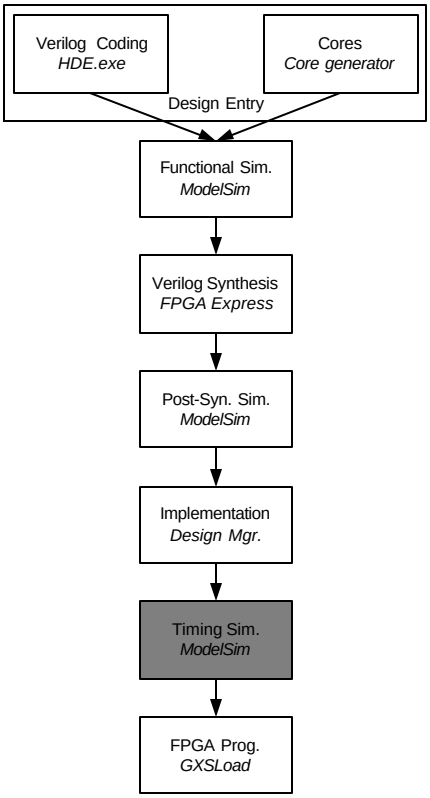
- Start implementation when every parameter is set properly.
  - Select “rev1 (New, OK)”
  - **Design=>Implement**
  - The Flow Engine should run through five implementation steps: Translate, Map, Place & Route, Timing (Sim), and Configure, which generates the configuration bit stream. Watch the log window for warnings and errors.
- View the report files
  - Click on **Utilities=>Report Browser**
  - In particular, examine the **Pad Report** for I/O pin assignments. Double check if the pins assigned in “mltring.ucf” file correspond to those in “Pad Report” and make sure that no other pins have been assigned.



dmgr_reports.jpg

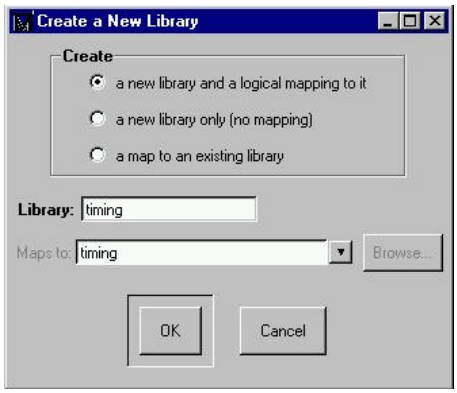
- Examine the output files
  - Check if the following files are up to date: “mltring.bit”, “time_sim.vhd”, and “time_sim.sdf”.
  - Double Click on “Pad Report”.
  - Open “mltring.ucf” and compare if the pin assignments are matched. Don’t continue until you show them to the instructor.

7. Timing (Post Implementation) Simulation (ModelSim)



Now, we will use **ModelSim** to run a full timing simulation using Xilinx's Simprim and Unisim libraries. **ModelSim** reads the "time_sim.sdf" file that contains the timing information from the implementation process and annotates the timing information into the "time_sim.vhd" file.

- Change the directory to where "time_sim.vhd" is.
  - **File=>Change Directory**=>Browse to "I:\xilinx\tutorial\mac\synthesis\"
- Create a new library for timing simulation
  - **Design=>Create a New Library**=>Timing

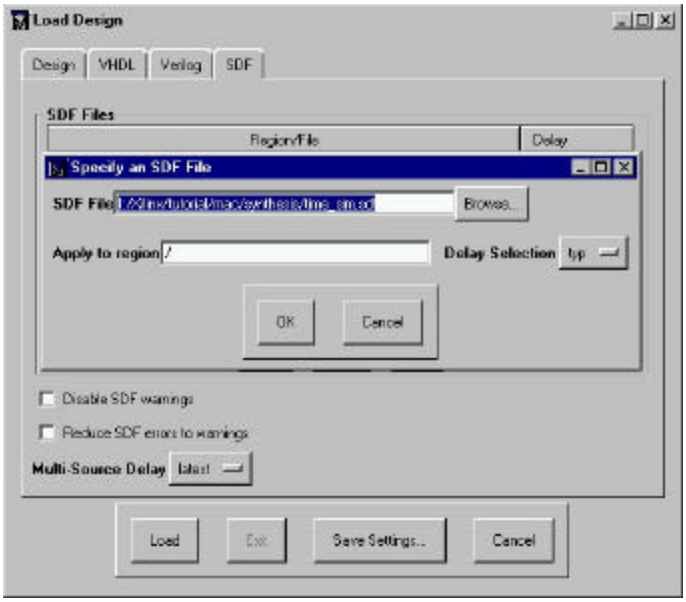


timing_new_lib.jpg

- Compile the "time_sim.vhd" to the "timing" library

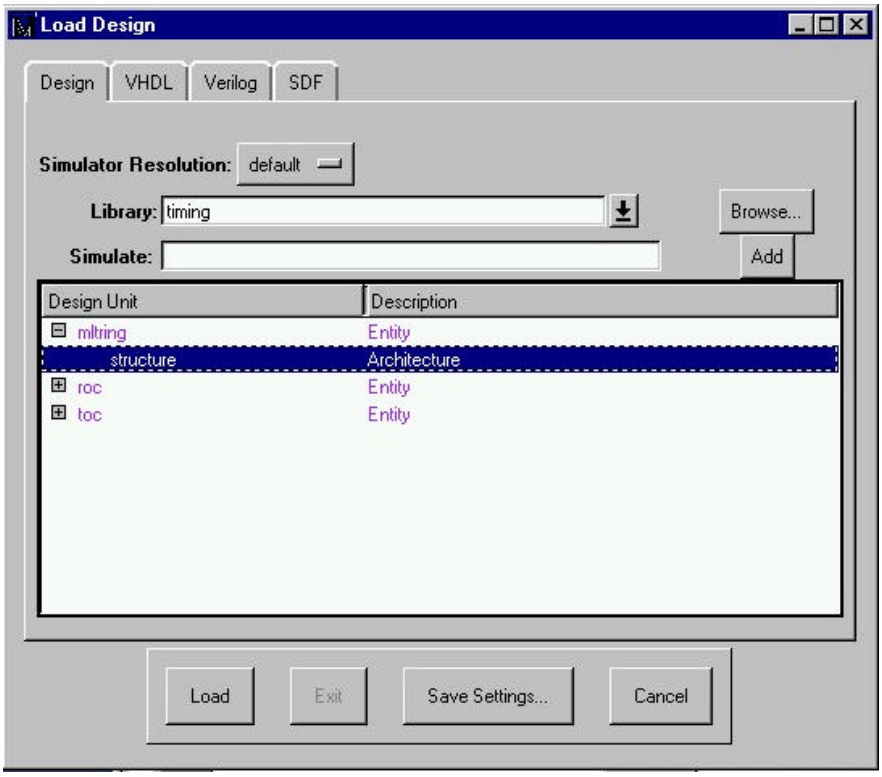
- **Design=>Compile**

- Load the top-level(“mltring”) module from “timing” library
  - **Design=>Load Design**
  - Click “SDF”=>”Add...”=>”SDF File”=>Browse ”I:\xilinx\tutorial\mac\synthesis\time_sim.sdf”



timing_sdf.jpg

  - Click on “Design”=>mltring=>structure then click Load

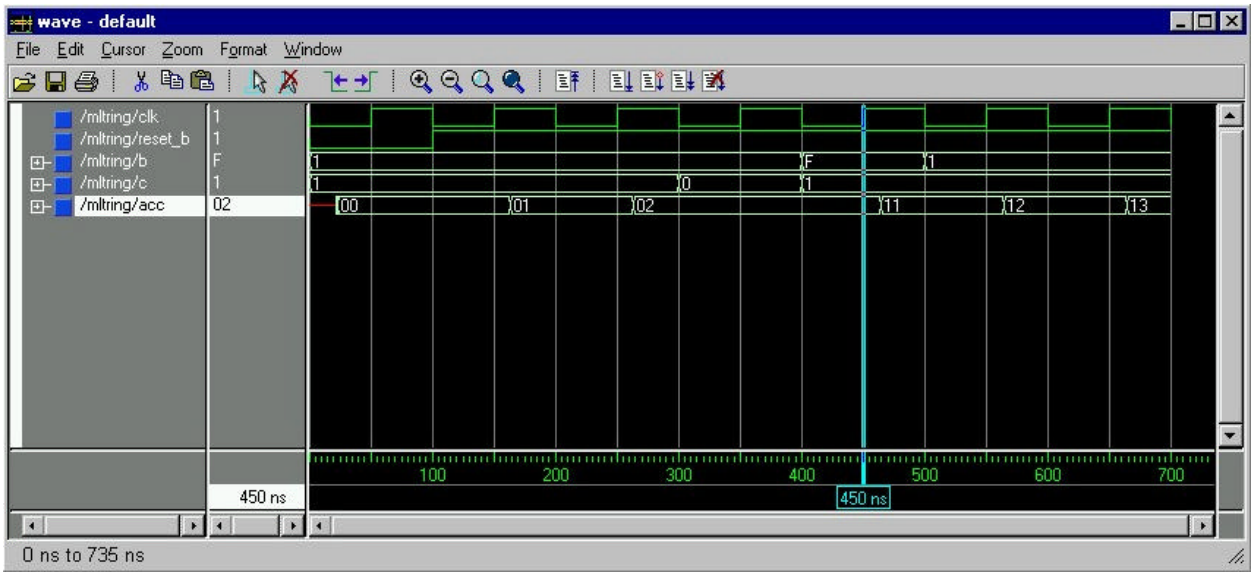


timing_load_sdf.jpg

- Create the simulation environment and apply the stimuli.

- **Macro=>Execute Macro**=>Browse=>”timing_force.do”
- Open ”timing_force.do” and understand what it does.

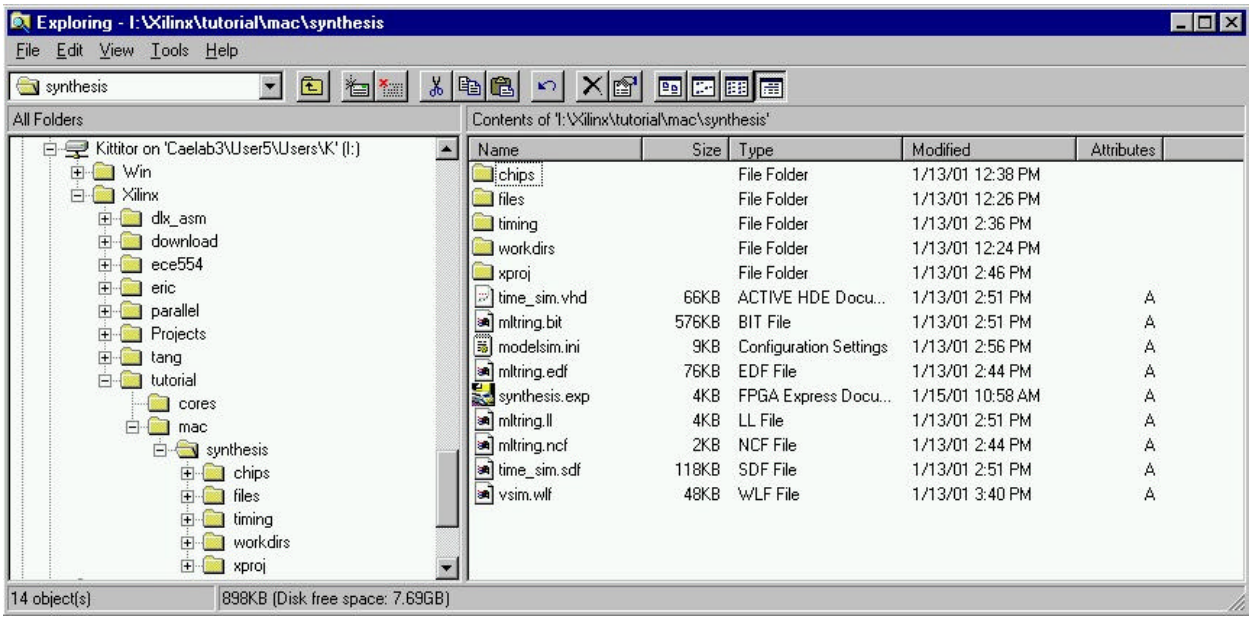
- Functional verification: Verify if the results are correct and determine the propagation time of the 8-bit register from the script file “timing_force.do”



timing_waveform.jpg

7. Conclusion

The final directory structure of the tutorial should be similar to this figure.



(dir_structure.jpg)

Directory	Created by	Purpose
cores	Student	To hold generated cores and the “Core Gen” project file
mac	Student	To hold other files in the design hierarchy
mac\synthesis	FPGA Express	FPGA Express working directory
mac\synthesis\chips	FPGA Express	To hold information on the synthesized modules
mac\synthesis\files	FPGA Express	For internal use
mac\synthesis\timing	ModelSim	To hold compiled module for timing simulation
mac\synthesis\workdirs	FPGA Express	For internal use
mac\synthesis\xproj	Design Manager	To hold several versions and revisions of implementation

8. Refe rences

a. *ModelSim SE/EE User's Manual*

ModelSim=>Help=>SE/EE Documentation=>SE/EE Bookcase

b. *FPGA Compiler II/FPGA Express VHDL Reference Manual*

FPGA Express=>Help=>VHDL Reference Manual

c. *FPGA Compiler II/FPGA Express Verilog HDL Reference Manual*

FPGA Express=>Help=>HDL Reference Manual

d. *Xilinx Foundation 3.1i On-line Documentation*

Start=>Programs=>Xilinx Foundation 3.1i=>Online Documentation

e. *Xilinx Design Manager On-line Help*

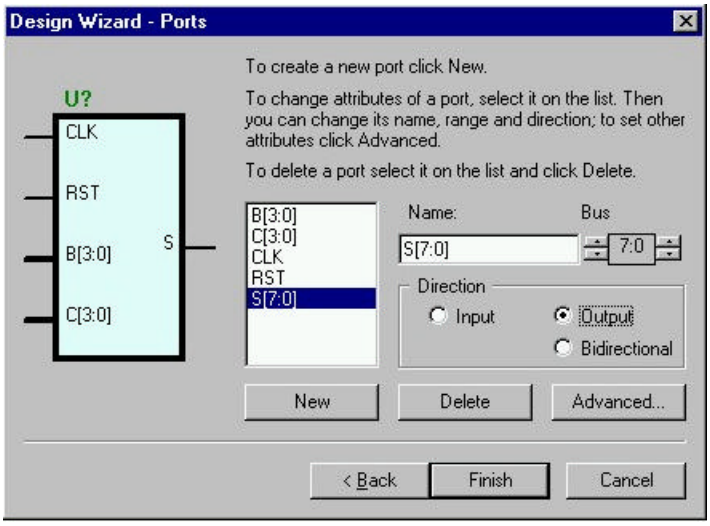
Design Manager=>Help=> Help Topics

Appendixes

A. HDL Design Wizard and Language Assistant

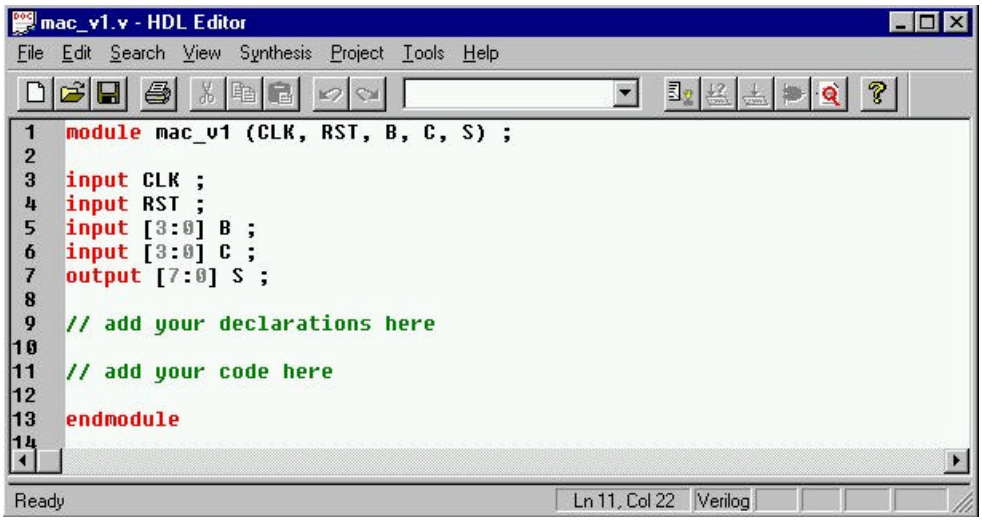
HDL Design Wizard is a GUI tool to help the designers visually declare the interfaces of an HDL "module" (in Verilog) and "entity" (in VHDL). Input, output or bidirectional ports are defined.

- HDL Wizard
  - Double Click on "HDL.EXE" icon on the desktop
    - "Create new document"=> Select Use HDL Design Wizard => "OK"
    - Click "Next>"
    - Select "Verilog"
    - Type in the file's name to be saved: "mac_v1"
    - Enter the following I/P ports: CLK, RST, B[3:0], C[3:0], and O/P port: S[7:0] as follows
      - Click "New"
      - Enter the port name
      - Click Up/Down button appropriately to get the desired values
      - Select the correct direction
      - If there is no more I/O ports to enter, click "Finish" else go back to the beginning.



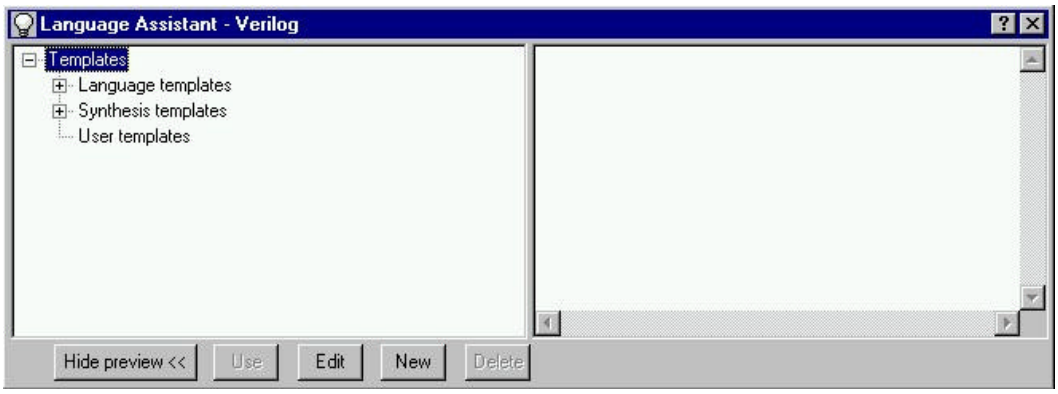
(wizard_ports.jpg)

      - For a registered I/O port, click on the "Advanced…" button to specify.
      - After entering all the I/O ports, you should see a Verilog interface file similar to the following. Note that "mac_v1" is used as both the module name and file name which is not necessary, be convenient and helpful. In practice, the file name has no relationship with the module name.

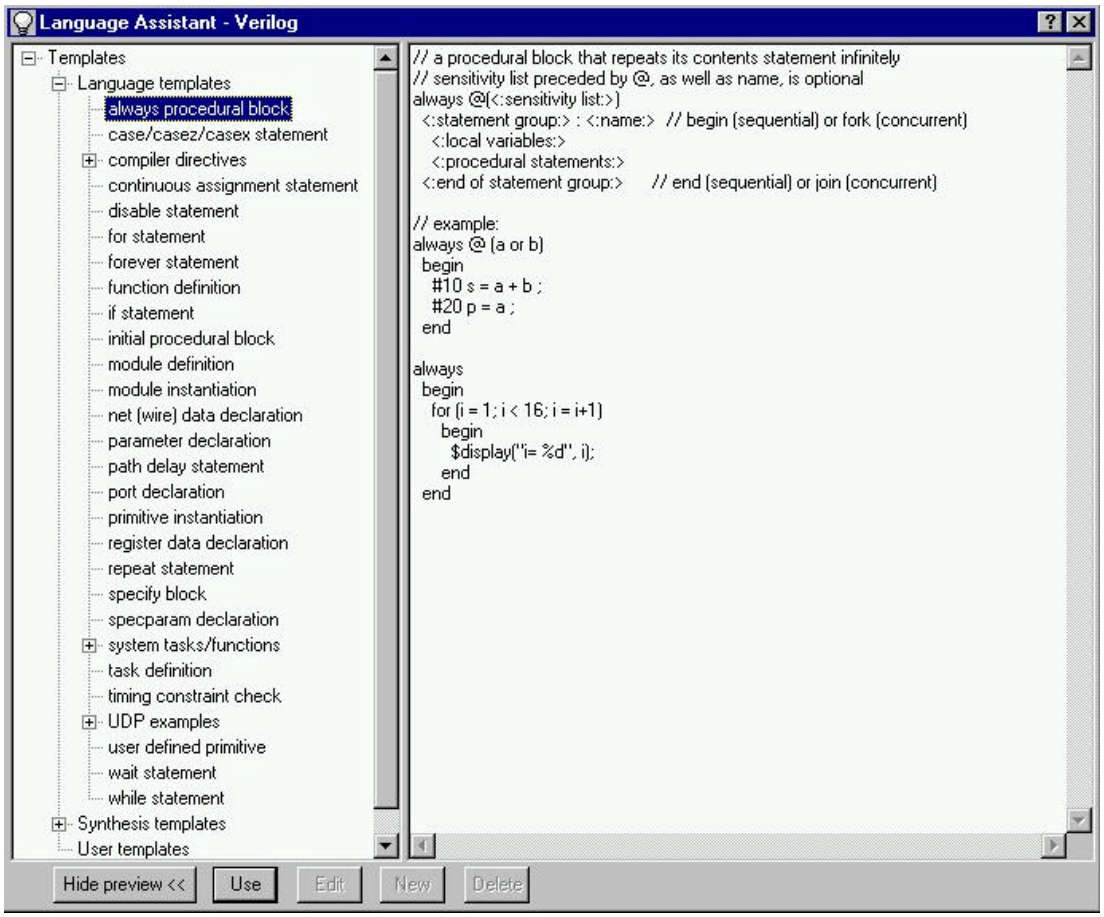


- Add the declarations and code as necessary.

- Language Assistant
  - **Tools=>Language Assistant**



  - The Language Assistant is composed of language, synthesis, and user templates. Both Verilog and VHDL have its own set of templates.
    - The Language templates provide most of the language constructs in alphabetical order: always, case, …
      - Select “always” so that you can see the explanation and example in the right window
      - Click “Use” to paste the content into the HDL file at the location pointed to by the cursor.



(assistant_lang.jpg)

- The synthesis templates provide a set of synthesizable HDL components such as barrel shifter, comparator, etc. However, for efficient and high-speed prototyping in the FPGA, the Core Generator System is suggested since cores use the special features, such as high-speed carry propagation, of the FPGA effectively.
- The user templates can be customized to keep any HDL code as on-line references.

B. Syntax checking for HDL Synthesis vs. HDL Simulation

Note that a syntactically correct HDL module may not be synthesizable by FPGA Express. Therefore, before functional simulation, you MUST have FPGA Express checked the syntax for synthesis. This is the difference between syntax checking for HDL synthesis and HDL simulation. Please take a look at on-line guide for HDL(Verilog) and VHDL coding for synthesis (see References section). The Language Assistant in App. A also provides templates in both Verilog and VHDL.

- Copy a Verilog file "non_syn.v" at http://www.cae.wisc.edu/~ece554/s01/ to your "I:\xilinx\tutorial\mac" directory.
- Let ModelSim check the syntax
  - Change the current dir. of ModelSim to "I:\xilinx\tutorial\mac".
  - Compile "non_syn.v" to "Verilog" library.
  - You should see no error or warning messages.
- Let FPGA Express check the syntax
  - Go back to FPGA Express
  - Add a new source file "non_syn.v" to the current project "synthesis." FPGA Express will automatically performs syntax checking.
  - This file will not pass this step because of the "initial" statement. Look at the error message. It should be exactly like this.

VE-19 (1 Occurrence)

Error: Initial statement not supported near symbol "initial" on line 5 in file non_syn.v

  - Delete "initial Q = 1'b1;" and update the source file again. It should pass the syntax checking by FPGA Express.
  - Note that this is just one of several non-synthesizable constructs in Verilog HDL. See one of the references for either Verilog or VHDL.
  - In addition, passing the FPGA Express syntax checker does always NOT mean that the HDL code can be synthesized correctly since problems can arise later in the synthesis process. Also, synthesis may work, but not produce useful results. For example, you have to search the output schematic carefully to see whether latches are inferred by your incomplete "if" or "case" statements.

C. Incremental Design Synthesis

The final project is usually complicated and large. This is a simple rule of thumb, "the more complex the design is going to be, the longer time spent to execute the design flow". This is for not only functional simulation but also design synthesis and implementation. However, hardware development is not that bad. It is similar to software development in several senses, for example, the concept of *build*. If a module is under development while others are done, the synthesis tool (FPGA Express) doesn't need to re-synthesize all individual modules especially those that are unchanged and time-consuming to synthesize.

- Go back to FPGA Express
- Synthesize "mac_test" module with the Check mark on "Do not insert I/O pads" option.
    - If you can recall the synthesis procedure shown earlier, this is left unchecked. The reason is to let FPGA Express put "I/O pads" on all of the I/O ports.
    - In this case, we don't let FPGA Express do that because this module is under the module hierarchy. No I/O pads are needed. Otherwise, error messages will show up during the translation phase in the implementation. See section 5 Design Implementation.
- Export netlist file "mac_test.edf" to "I:\xilinx\tutorial\synthesis\" with "Export Timing Specifications" unchecked.
- Make sure that "mac_test.edf" is present in "I:\xilinx\tutorial\synthesis\" directory.
- Add the new source file ("mac_test.edf") to the current project.
- Delete the source file ("mac.v") from the current project.
    - Click on "mac.v" icon then "Edit=>Del".

(incremental_add.jpg)

- At this point, the "mltring-optimized" chip must be updated.
    - Select "mltring-optimized" chip icon
    - Synthesis=>Force Update
- Export the netlist file "mltring.edf" of "mltring-optimized" again for the future implementation.
- Don't forget to specify where (which directory) this new netlist file is for Xilinx Design Manager. Please refer to Section 4, Design Implementation for more details.