CS152 FPGA CAD Tool Flow
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

Compiled: 4/3/2003 for CS152 Spring 03, Prof. John Kubiatowicz
Updated: 9/4/2003 for CS152 Fall 03, Prof. Dave Patterson
Updated: 2/10/2004 for CS152 Spring 04, Prof. John Kubiatowicz
Sources: lab3.pdf from CS150 9/17/2002, Xilinx Reference Manual, Xilinx University Program
Please send any errors, corrections, or comments to cs152-staff@imail.eecs.berkeley.edu

1 Introduction to the Design Tool Flow
2 How to Use the Design Tool Flow
3 Basic Project Tutorial
4 ChipScope
5 Tips and Hints

1 Introduction to the Design Tool Flow

Refer to the illustration on the next page for the steps involved in the CAD tool flow we will use.

1.1 Design Entry

The first step in logic design is to conceptualize your design. Once you have a good idea about the function and structure of your circuit, you can start the implementation process by specifying your circuit. In this class we will use a Hardware Description Language (HDL) called Verilog. HDLs have several advantages over other methods of circuit specification: ease of editing (files can be written using any text editor), ease of management when dealing with large designs, and the ability to use a high-level behavioral description of a circuit. If you are familiar with emacs, you may find it convenient for writing and editing Verilog code. Alternatively, you may use the editor provided in Xilinx's Project Navigator or ModelSim.

1.2 Synthesis

Once your design is entered, the next step in the implementation path is synthesis. In our case, the function of the synthesis program is to translate the Verilog description of the circuit into an equivalent circuit comprising a set of primitive circuit components that can be directly implemented on an FPGA. In a way, the synthesis tool is almost like a compiler. Where a compiler translates to a sequence of primitive commands that can be directly executed on a processor, synthesis translates to primitive circuit components that can be directly implemented in FPGA. The final product of a synthesis tool is a netlist file, a text file that contains a list of all the instances of primitive components in the translated circuit and a description of how they are interconnected. The synthesis tool we will be using in this class is called Synplify. Note that Xilinx's software suite can also perform this synthesis procedure. The reason we use Synplify is because Synplify is an industrial strength CAD tool program. It's faster, produces better logic, and will give many more synthesis warnings—something very useful for students!

1.3 Placement, Routing

The next step in the implementation flow is to take the netlist of components generated by the synthesis tool and turn it into bits that are need to configure the LUTs, muxes, Flipflops, and other configurable resourses in the FPGA. To do that, first the primitive circuit components in the netlist need to be assigned to a specific *place* on the FPGA. For example, a 4LUT implementing the function of a 4 input NAND gate in a netlist could be implemented with any of the about 40,000 4LUTs in a Xilinx Virtex 2000E FPGA chip. Clever choice of placement will make the subsequent routing easier and result in a faster overall circuit. Once the components are placed, the proper connections must be made according to the netlist description obtained from the synthesis step. That step is called *routing*. Unlike synthesis, which only requires a set of

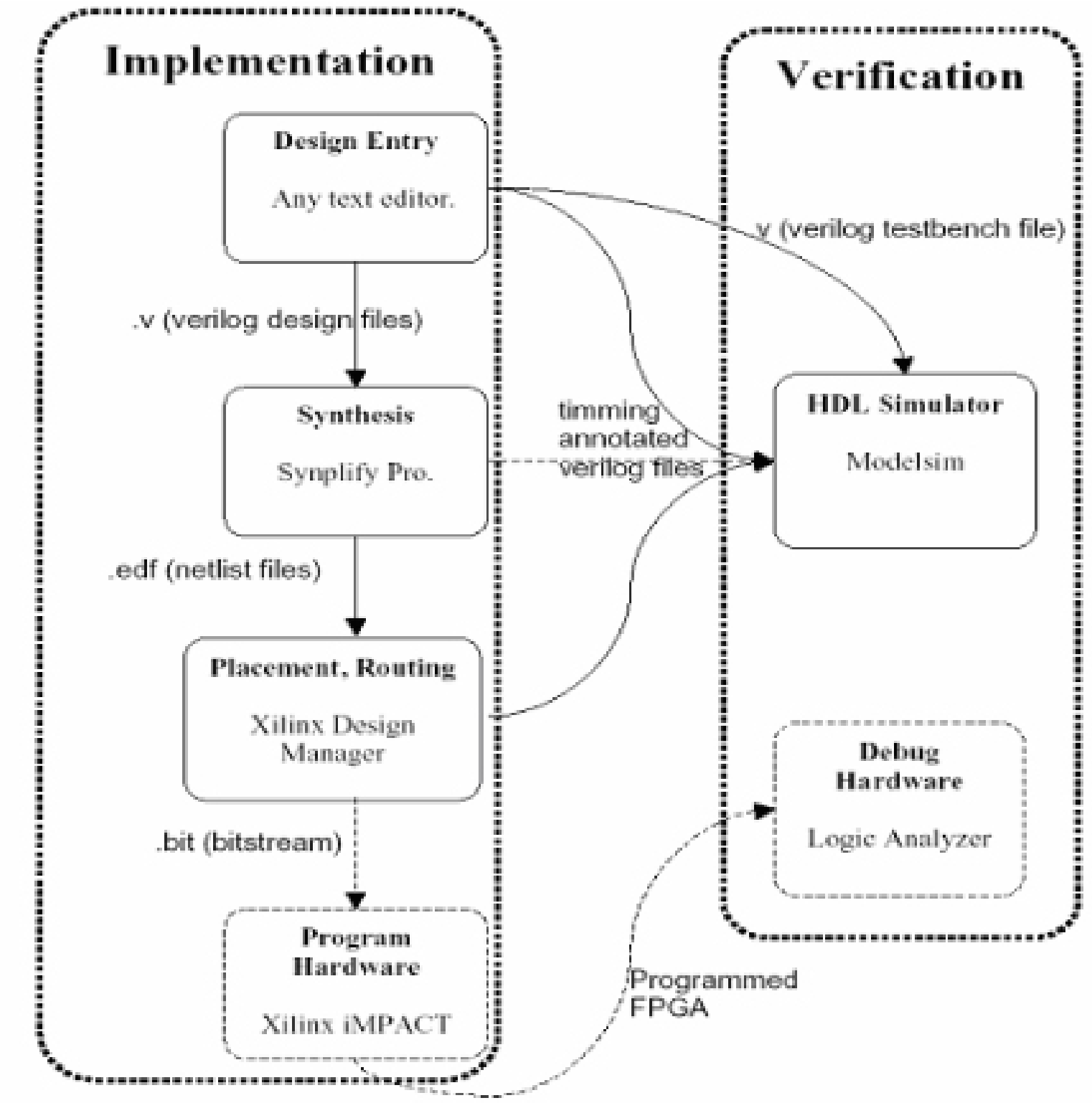
primitive components to translate to; placement and routing are dependent upon the specific size and structure of the target FPGA chip. Due to this reliance, the FPGA vendor usually provides the placement and routing programs. Therefore, we will be using the Xilinx's Project Navigator to perform this step. The end product after placement and routing is a bit file containing the stream of bits used to configure the FPGA.

1.4 Program Hardware

The last step in the implementation flow is the simple act of transporting the configuration bits to the FPGA. There are also many ways of doing this. For this class we will be mostly using the Parallel Cable IV along with the iMPACT software to program the board.

1.5 Verification

As you should have learned from experience, a significant part of the time and effort spent on any sizable project will be spent on debugging, and logic design is no exception. There are two ways to verify the correctness of a design: to program the FPGA with the design and check if the circuit is behaving correctly, or to run simulations of the design in software. While programming the FPGA and physically checking the functionality sounds simple, the whole tool flow requires a significant amount of time to complete, especially as your designs get larger and more complex (20 minutes towards the end of the semester!). Repeatedly tweaking the input design to fix errors would require running the flow repeatedly, a huge waste of time. In addition it can be difficult to physically observe the causes for an error on a FPGA. For these reasons, software simulation is essential in the verification process. There are many places along the tool flow where you can use simulation to verify the correctness of your design. As you progress down the tool flow and more information about the physical implementation on the FPGA becomes available, more accurate timing simulations can be performed. **You should not attempt to verify that your design works on the board until it works in simulation!!!** To do so otherwise would just be a waste of time. In this class, we will be using ModelSim to do our verification.



CAD tool flow. Optional links are showed as dashed lines.

2 How to Use the Design Tool Flow – An Overview

2.1 Design Entry

Design entry will be done either through Verilog or Schematics using Xilinx ISE 5.2i. You can find a short tutorial below. Note that the Synthesis tool Synplify Pro does not synthesize schematic files, so you have to take the functional Verilog code of the schematics instead, so that you can synthesize your project. For the Xilinx primitives in the Verilog code you use, you may be required to created blackboxes for those primitive blocks as well.

To port your schematics into Verilog, follow the instructions in section 3 of this tutorial.

Creating blackboxes: You will need to create a blackbox for each specific Xilinx primitive block that is used in the functional Verilog code. A Xilinx primitive block is one that is predefined in the Xilinx block libraries: for example IBUF, AND3 and RAMB4_S16. To create a blackbox, all