Chapter 1

Introduction

THE DESIGN process for digital integrated circuits is extremely complex. Unfortunately, the Electronic Design Automation (EDA) and Computed Aided Design (CAD) tools that are essential to this design process are also extremely complex. Finding a combination of tools and a way of using those tools that works for a particular design is know as finding a "tool path" for that project. This book will introduce one path through these complex tools that can be used to design digital integrated circuits. The tool path described in this book uses tools from Cadence (www.cadence.com) and Synopsys (www.synopsys.com) that are available to university students through special arrangements that these companies make with universities. Tool bundles that would normally cost hundreds of thousands or even millions of dollars if purchased directly from the companies are made available through "university programs" at small fixed fees.

In this book I'll call the tools "CAD tools" and the companies "EDA companies"

In order to justify these small fees, however, the EDA companies typically reduce their costs by offering very limited support for these tools to university customers. In an industrial setting there would likely be an entire CAD support department whose job it is to get the tools running and to develop tool flows for projects within the company. Few universities, however, can afford that type of support for their CAD tools. That leaves universities to sink or swim with these complex tools making it all the more important to find a usable tool path through the confusing labyrinth of the tool suites. This book is an attempt to codify at least one working tool path for a Cadence/Synopsys flow that students and researchers can use to design digital integrated circuits. It includes tutorials for specific tools, and an extended example of how these tools are used together to design a simple integrated circuit.

Instructions for installing the CAD tools can be found in the appendices

In addition to the CAD tools from Cadence and Synopsys, The tutorials assume that you have some sort of CMOS standard cell library avail-

able. The specific examples in this book will use a cell library developed at the University of Utah specifically for our VLSI classes known as the UofU_Digital library. This library, and the technology information available through the NCSU CDK (North Carolina State University Cadence Design Kit), are freely available from the University of Utah and North Carolina State University respectively. If you don't have these libraries you should be able to follow most of the tutorials with your own library, but you must have a library of some sort.

Details about these libraries can also be found in the appendices

1.1 Cad Tool Flows

The general tool flow described here uses CMOS standard cells with automatic place and route to design the chip, but also includes details of how to design custom cells as layout and add those cells to a library. This custom portion of the flow could, of course, be used to design a fully custom chip. It can also be used to design your own cell library. Designing a cell library involved not only designing the individual cells, but characterizing those cells in terms of their delay in the face of different output loads and input slopes, and codifying that behavior in a way that the synthesis tools can use. The cells also must have their physical parameters characterized so that the place and route tools have enough information to assemble them into a chip. Finally, simulation at a variety of levels of detail and timing accuracy is essential if a functional chip is to result from this process.

CAD tools typically do not include any technology or cell data. This data comes directly from the chip and cell vendors and contains information specific to their technologies.

This entire tool flow will use a large number of tools from both Cadence and Synopsys, a large number of different file formats and conversion programs, and a lot of different ways of thinking about circuits and systems. This is inevitable in a task as complex as designing a large integrated circuit, but it can be intimidating. One ramification of the type of complexity inherent in VLSI design is that the tools, designed as they are to handle very large collections of cells and transistors, aren't much simpler to use on just 4 transistors than they are on 4,000, 400,000, or 4,000,000 transistor designs. It is not easy to simply start small and add features. One must, in some ways, do it all right from the start which makes the learning curve quite steep. There are lots of pieces of the flow that must be available right from the start which can be overwhelming at first. Hopefully by breaking the tool flow into individual tool tutorials, and with detailed walk-through tutorials with lots of screen shots of what you should see on the screen, this can be made less intimidating.

File types for the complete flow are described as they are used in the flow and documented in the appendices. In addition to Cadence database files, they include .lib, .db, .lef, .gds, .sdf, .def, .v, .sdc, and .tcl files.

Of course, as with any tool with a steep learning curve, once you've made it up the steep part of the curve, you may not want to refer back to the level of detail contained in the tutorial descriptions. For that stage of the process I've included slimmed down "highlights" versions of the tool

instructions in the appendices of this volume. If you need to refer back to a tool that you've used before but haven't used for a while, you may just need to glance at the highlights in the appendices rather than walk through the entire tutorial again.

I need at least one figure of the whole tool flow here, and perhaps individual pictures of flows for different purposes. For example, a flow for cell design, a flow for standard cell from Verilog descriptions, and a general flow.

1.1.1 Custom VLSI and Cell Design Flow

This is a tool flow for designing custom VLSI systems where the design goes down to the circuit fabrication layout. This flow starts with transistor schematics at the front end and uses custom layout to design portions of the system. It is used for designing cell librarys as well as for designing performance-critical portions of larger circuits where individual design of the transistor-level circuits is desired. The front end for this flow is transistor-level schematics in Composer. These schematics may be simulated at a functional level using Verilog simulators like Verilog-XL and NC_Verilog, and with a detailed analog simulator like Spectre. The back end is composite layout using Virtuoso and more detailed simulation using analog simulators.

If the final target is a cell library then the cells can be characterized for performance using a simulator like Spectre or by using a library characterizer tool like Signalstorm. These chacterizations are required if you would like to use these cells with a synthesis system later on. Abstract view can also be generated so that the cells can be used with a place and route system.

1.1.2 Hierarchical Cell/Block ASIC Flow

This is a tool flow for system level design using a CMOS standard cell library. The library may be a commercial library or it may be one that you design yourself, or a combination of the two. The front end can be schematics designed using cells from these libraries, or Verilog code. If the system decription is in structural Verilog code which is set of instantiations of standard cells in Verilog, then this can be used directly as the front-end description. If the Verilog is behavioral Verilog, then a synthesis step using Synopsys dc_shell, design_vision or module compiler, or Cadence BuildGates can synthesize the behavioral description into a Verilog structural description.

These descriptions, whether structural, behavioral, or a combination of both, can be simulated for functionality using Verilog-XL or NC_Verilog.

These simulations may use a zero-delay, unit-delay, or extracted delay model. The extracted delays come from the synthesis systems which extract timings based on the cell characterizations.

The back end to the ASIC flow uses SOC Encounter to place and route the structural file into a full chip. This description may be extracted again to get timings that include wiring delays, or the timing can be analyzed using a static timing analyzer like Synopsys PrimeTime. The system can also be simulated in mixed-timing mode where parts of the circuit are simulated at a switch level using a Verilog simulator and parts of the circuit are simulated at a detailed level using an analog simulator like Spectre. The final result is a gds (also known as stream) file that can be sent to a fabrication service such as MOSIS to have the chips built.

Of course, the tool flows described here only scratches the surface of what the tools can do! Please feel free to explore, press on likely looking buttons, and read the manuals to explore the tools further. If you discover new and wonderful things that the tools can do, document those additions to the flow and let me know and I'll include them in subsequent releases of this manual.

1.2 What this Manual is and isn't

This manual includes walk-through tutorials for a number of tools from Cadence and Synopsys, and description of how to combine those tools into a working tool flow for VLSI design. It is *not* a manual on the VLSI design process itself! There are many fine textbooks about VLSI design available. This is a "lab manual" that is meant to go along with those textbooks and describe the nuts and bolts of working with the CAD tools. I will assume that you either already understand general VLSI design, or are learning that as you proceed through the tutorials contained in this manual.

Bugs in the Tools?

Before we dive into the tutorials, here's a quick word about tool bugs. These tools are complex, but so are the systems that you can design with them. They also feel very cumbersome and buggy at times, and at times they are! However, even with the inevitable bugs that creep into tools like this, I encourage you to follow the tutorials carefully and resist the temptation to blame a tool bug each time you run into a problem! I've found in teaching courses with these tools for years that it is almost 100% certaintly that if you're having trouble with a tool in a class setting, that it's something that you've done or some quirk of your data rather than a bug in the tool. It's

amazing how subtle (or sometimes how obvious!) the differences can be in what you're doing and what the procedure specifies. Relax, take a deep breath, and think carefully about what's going on and what might cause it. Read the error messages carefully. Occasionally there is real information in the error message! Try explaining things to a fellow student. Often in the process of explaining what you're doing you'll see what's going on. Let someone else look at it. Let your first instinct be to try to figure out what's going on, not to blame the tool! If the tool turns out the be the problem, at least you will have exhausted the more likely causes of the problem first before you discover this.

1.3 Typographical Conventions

Finally, a word about typographical conventions. I will try to stick to these, but don't promise perfect adherence to these conventions! In general:

- I'll try to use `boxed, fixed width font` for any text that you should type in to the system. This, hopefully, will look a little like the fixed-width font you'll see on your screen while you're typing. So if you are supposed to type in a command like `cad-ncsu` it will look like that.
- I'll try to use **bold face** for things that you should see on the sceen or in windows that the tools pop up. So if you should see **Create Library** in the title bar of the window, it will look like that in the text.
- I'll use *slanted text* in the marginal notes. These are little points of interest that are ancillary or parenthetical to the main text.

This is a margin note

- I'll use a non-serifed face to give the names of the tools that we're working with. Note that the name of the tool, like Composer, is seldom the name of the executable program that you run to get to that program. For those, refer back to the typed commands like `cad-ncsu`.