Bits and Pieces of CS250's Toolflow

CS250 Tutorial 2 (Version 091210a)
September 12, 2010
Yunsup Lee

In this tutorial you will learn what each VLSI tools used in class are meant to do, how they flow, file extensions of their inputs and outputs.

Figure 1 shows the CS250 toolflow you will be using for the class. You will use Synopsys VCS (`vcs`) to *simulate* and *debug* your RTL design. After you get your design right, you will use Synopsys Design Compiler (`dc_shell-xg-t`) to *synthesize* the design. Synthesis is the process of transforming an RTL model into a gate-level netlist. You will use Synopsys Formality (`fm_shell`) to *formally verify* that the RTL model and the gate-level model *match*. VCS is used again to simulate the synthesized gate-level netlist. After obtaining a working gate-level netlist, you will use Synopsys IC Compiler (`icc_shell`) to *place and route* the design. Placement is the process by which each standard cell is positioned on the chip, while routing involves wiring the cells together using various metal layers. The tools will provide feedback on the performance and area of your design after both synthesis and place and route. The results from place and route are more realistic but require much more time to generate. After place and route, you will generate and simulate the final gate-level netlist using VCS. Finally you will use this gate-level simulation as a final test for correctness and to generate transition counts for every net in the design. Synopsys PrimeTime PX (`pt_shell`) takes these transition counts as input and correlate them with the capacitance values in the final layout to produce estimated power measurements.

Each section or subsection has a list of documents which are provided from Synopsys. The documents are located in the cs250 course locker (`~cs250/manuals`) which can be accessed through the instructional machines.

Tools

Synopsys VCS

VCS is used to simulate your design. The design could be expressed in several different languages, however, we encourage you to use Verilog for this class. For more information on Verilog, take a look at the Language section in this tutorial. VCS takes a set of Verilog files as input and produces a simulator. When you execute the simulator you need some way to observe your design so that you can measure its performance and verify that it is working correctly. You can instruct the simulator to automatically write transition information about each signal in your design to a file. There is a standard text format for this type of signal transition trace information called the Value Change Dump format (VCD). Unfortunately, these textual trace files can become very large quickly, so Synopsys uses a proprietary compressed binary trace format called VCD Plus (VPD). You can view VPD files using a waveform viewer called Synopsys Discover Visual Environment (DVE).

- `vcs-user-guide.pdf` - VCS User Guide
- `vcs-quick-reference.pdf` - VCS Quick Reference

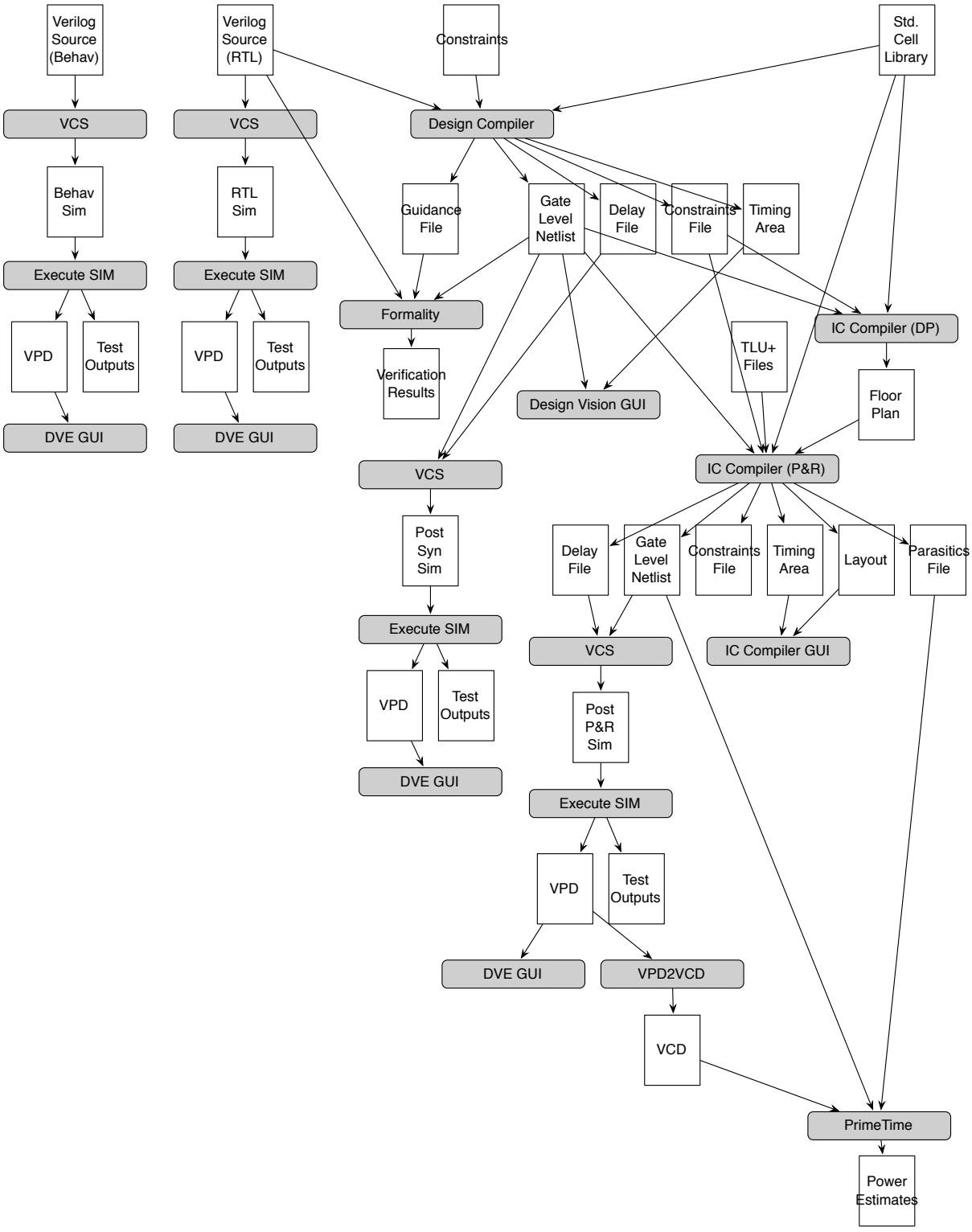

Figure 1: CS250 Toolflow

- `vcs_dve-user-guide.pdf` - Discovery Visual Environment User Guide
- `vcs_ucli-user-guide.pdf` - Unified Command Line Interface User Guide

Synopsys Design Compiler

Design Compiler takes an RTL hardware description, timing constraints, and a standard cell library as input and produces a gate-level netlist as output. To take a closer look at the standard cell library you are using for the class, consult the Process section of this tutorial. Internally, a synthesis tool performs many steps including high-level RTL optimizations, RTL to unoptimized Boolean logic, technology independent optimizations, and finally technology mapping to the available standard cells. A synthesis tool is only as good as the standard cells which it has at its disposal. You get reports about the critical path of your synthesized design, area and power estimates, and resource mapping. You can also take advantage of Design Vision to visualize critical paths of your synthesis results, and schematics of the gate-level netlist.

- `dc-user-guide.pdf` - Design Compiler User Guide
- `dc-quick-reference.pdf` - Design Compiler Quick Reference
- `dc-user-guide-cli.pdf` - Design Compiler Command-Line Interface Guide
- `dc-user-guide-lp.pdf` - Synopsys Low-Power Flow User Guide
- `dc-user-guide-verilog.pdf` - HDL Compiler for Verilog User Guide
- `dc-user-guide-sysverilog.pdf` - HDL Compiler for SystemVerilog User Guide
- `dc-user-guide-tcl.pdf` - Using Tcl With Synopsys Tools
- `dc-user-guide-tco.pdf` - Synopsys Timing Constraints and Optimization User Guide
- `dc-reference-manual-opt.pdf` - Design Compiler Optimization Reference Manual
- `dc-reference-manual-rt.pdf` - Design Compiler Register Retiming Reference Manual
- `dc-application-note-sdc.pdf` - Synopsys Design Constraints Format Application Note
- `dc_dv-user-guide.pdf` - Design Vision User Guide
- `dc_dv-tutorial.pdf` - Design Compiler Tutorial Using Design Vision

Synopsys Formality

Formality is used to formally verify whether or not your RTL implementation and the synthesized gate-level netlist match. Formal verification utilizes mathematical techniques to compare the logic to be verified against either a logical specification or a reference design. Unlike verification through simulation, formal verification does not require input vectors. As it considers only logical functions during comparisons, it is independent of the design's physical properties, such as its layout and timing.

- `fm-user-guide.pdf` - Formality User Guide
- `fm-quick-reference.pdf` - Formality Quick Reference

Synopsys IC Compiler

IC Compiler takes as input a gate-level netlist, timing constraints, physical and timing libraries, and it generates as output a layout. It first determines how each gate should be placed on the chip. Then it uses several heuristic algorithms to group related gates together and thus hopefully minimize routing congestion and wire delay. This tool will focus its effort on minimizing the delay through the critical path. To this end, it can resize gates, insert new buffers, and even perform local resynthesis.

- `icc-user-guide.pdf` - IC Compiler Implementation User Guide
- `icc-quick-reference.pdf` - IC Compiler Quick Reference
- `icc_dp-user-guide.pdf` - IC Compiler Design Planning User Guide

Synopsys PrimeTime PX

PrimeTime PX is an add-on feature to PrimeTime that accurately analyzes power dissipation of cell-based designs. PrimeTime PX supports two types of power analysis modes: averaged mode and time-based mode. Averaged mode calculates averaged power based on toggle rates. Time-based mode lets you know the peak power as well as the averaged power using gate-level simulation activity.

- `pt-user-guide.pdf` - PrimeTime Fundamentals User Guide
- `pt_px-user-guide.pdf` - PrimeTime PX User Guide
- `pt-quick-reference.pdf` - PrimeTime Suite Quick Reference

Libraries and Process

You are using the Synopsys 90nm educational library for the class. It is a 1P9M (1 poly, 9 metal layers) 1.2V/2.5V process.

The standard cells and other cells have been built using this design rules. The standard cell library has been created aimed at optimizing the main characteristics of designed ICs by its help. The cell library includes typical miscellaneous combinational and sequential logic cells for different drive strengths. Besides, the cell library contains the cells (isolation cells, level shifters, retention flip-flops, always-on buffers, and power gating cells) which are required for different styles of low power (multi-voltage, multi-threshold) designs. The presence of all these cells provides the support of IC design with different core voltages to minimize dynamic and leakage power.

We made an SRAM model compiler which works with the CS250 toolflow. This SRAM model compiler generates a behavioral verilog source file, a technology library, and a milkyway library for an arbitrary sized SRAM block. We also made a cache model compiler which works with the CS250 toolflow as well. The parameters to the cache model compiler are: size, number of sets, ways, MSHRs, secondary misses, access port width, refill port width, behavior (blocking/non-blocking).

- `synopsys-90nm-databook-stdcells.pdf` - Digital Standard Cell Library Databook
- `tut8-ucbmc.pdf` - UC Berkeley SRAM Model Compiler
- `tut9-ucbcc.pdf` - UC Berkeley Cache Model Compiler

DesignWare

DesignWare library is a collection of reusable circuit-design building blocks (components) that are tightly integrated into the Synopsys synthesis environment. DesignWare components that implement many of the built-in HDL operators are provided by Synopsys. These operators include +, −, ∗, <, >, <=, >=, and the operations defined by if and case statements.

- `designware-intro.pdf` - DesignWare Building Block IP Documentation Overview
- `designware-user-guide.pdf` - DesignWare Building Block IP
- `designware-quick-reference.pdf` - DesignWare Building Block IP Quick Reference
- `designware-datasheets` - Directory containing datasheets on each DW component

Language

The CS250 toolflow supports Verilog and SystemVerilog. For language specifications consult the following documents.

- `ieee-std-1364-1995-verilog.pdf` - Language specification for the original Verilog-1995
- `ieee-std-1364-2001-verilog.pdf` - Language specification for Verilog-2001
- `ieee-std-1364-2005-verilog.pdf` - Language specification for Verilog-2005
- `ieee-std-1364.1-2002-verilog-synthesis.pdf` - Standard for Verilog Register Transfer Level Synthesis
- `ieee-std-1800-2005-sysverilog.pdf` - Language specification for the original SystemVerilog-2005
- `ieee-std-1800-2009-sysverilog.pdf` - Language specification for SystemVerilog-2009

File Extensions

- `*.v` - Verilog source file. Normally it's a source file your write. Design Compiler, and IC Compiler can use this format for the gate-level netlist.
- `*.sv` - SystemVerilog source file. Normally it's a source file your write.
- `*.vg, .g.v` - Verilog gate-level netlist file. Sometimes people use these file extension to differentiate source files and gate-level netlists.
- `*.svf` - Automated setup file. This file helps Formality process design changes caused by other tools used in the design flow. Formality uses this file to assist the compare point matching and verification process. This information facilitates alignment of compare points in the designs that you are verifying. For each automated setup file that you load, Formality processes the content and stores the information for use during the name-based compare point matching period.
- `*.ddc` - Synopsys internal database format. This format is recommended by Synopsys to hand gate-level netlists.
- `*.vcd` - Value Change Dump format. This format is used to save signal transition trace information. This format is in text format, therefore, the trace file in this format can get very large quickly. There are tools like `vcd2vpd`, `vpd2vcd`, and `vcd2saif` switch back and forth between different formats.

- `*.vpd` - VCD Plus. This is a proprietary compressed binary trace format from Synopsys. This file format is used to save signal transition trace information as well.
- `*.saif` - Switching Activity Interchange Format. It's another format to save signal transition trace information. SAIF files support signals and ports for monitoring as well as constructs such as generates, enumerated types, records, array of arrays, and integers.
- `*.tcl` - Tool Command Language (Tcl) scripts. Tcl is used to drive Synopsys tools.
- `*.sdc` - Synopsys Design Constraints. SDC is a Tcl-based format. All commands in an SDC file conform to the Tcl syntax rules. You use an SDC file to communicate the design intent, including timing and area requirements between EDA tools. An SDC file contains the following information: SDC version, SDC units, design constraints, and comments. Consult `dc-application-note-sdc.pdf` for further information.
- `*.sdf` - Standard Delay Format. An SDF file stores the timing data generated by the tools for use at any stage in the design process. The data in the SDF file is represented in a tool-independent way and can include delays, timing checks (including setup time, hold time), timing constraints, timing environment, incremental and absolute delays, conditional and unconditional module path delays and timing checks, etc.
- `*.lib` - Technology Library source file. Technology libraries contain information about the characteristics and functions of each cell provided in a semiconductor vendors library. Semiconductor vendors maintain and distribute the technology libraries. In our case the vendor is Synopsys. Cell characteristics include information such as cell names, pin names, area, delay arcs, and pin loading. The technology library also defines the conditions that must be met for a functional design (for example, the maximum transition time for nets). These conditions are called design rule constraints. In addition to cell information and design rule constraints, technology libraries specify the operating conditions and wire load models specific to that technology.
- `*.db` - Technology Library. This is a compiled version of `*.lib` in Synopsys database format.
- `*.plib` - Physical Library source file. Physical libraries contain process information, and physical layout information of the cells. This information is required for floor planning, RC estimation and extraction, placement, and routing.
- `*.pdb` - Physical Library. This is a compiled version of `*.plib` in Synopsys database format.
- `*.slib` - Symbol Library source file. Symbol libraries contain definitions of the graphic symbols that represent library cells in the design schematics. Semiconductor vendors maintain and distribute the symbol libraries. Design Compiler uses symbol libraries to generate the design schematic. You must use Design Vision to view the design schematic. When you generate the design schematic, Design Compiler performs a one-to-one mapping of cells in the netlist to cells in the symbol library.
- `*.sdb` - Symbol Library. This is a compiled version of `*.slib` in Synopsys database format.
- `*.sldb` - DesignWare Library. This file contains information about DesignWare libraries.
- `*.def` - Design Exchange Format. This formats is often used in Cadence tools to represent physical layout. Synopsys tools normally use Milkyway format to save designs.
- `*.lef` - Library Exchange Format. Standard cells are often saved in this format. Cadence tools also often uses this format. Synopsys tools normally use Milkyway format for standard cells.
- `*.rpt` - Reports. This is not a proprietary format, it's just a text format which saves generated reports by the tools when you use the automated makefiles and scripts.

- `*.tf` - Vendor Technology File. This file contains technology-specific information such as the names, characteristics (physical and electrical) for each metal layer, and design rules. These information are required to route a design.
- `*.itf` - Interconnect Technology File. This file contains a description of the process cross-section and connectivity section. It also describes the thicknesses and physical attributes of the conductor and dielectric layers.
- `*.map` - Mapping file. This file aligns names in the vendor technology file with the names in the process `*.itf` file.
- `*.tluplus` - TLU+ file. These files are generated from the `*.itf` files. TLUPlus models are a set of models containing advanced process effects that can be used by the parasitic extractors in Synopsys place-and-route tools for modeling.
- `*.spef` - Standard Parasitic Exchange Format. File format to save parasitic information extracted by the place and route tool.
- `*.sbpf` - Synopsys Binary Parasitic Format. A Synopsys proprietary compressed binary format of the `*.spef`. Size of the file shrinks quite a bit using this format.
- `Milkyway database`

  The Milkyway database consists of libraries that contain information about your design. Libraries contain information about design cells, standard cells, macro cells, and so on. They contain physical descriptions, such as metal, diffusion, and polygon geometries. Libraries also contain logical information (functionality and timing characteristics) for every cell in the library. Finally, libraries contain technology information required for design and fabrication.

  Milkyway provides two types of libraries that you can use: reference libraries and design libraries. Reference libraries contain standard cells and hard or soft macro cells, which are typically created by vendors. Reference libraries contain physical information necessary for design implementation. Physical information includes the routing directions and the placement unit tile dimensions, which is the width and height of the smallest instance that can be placed.

  A design library contains a design cell. The design cell might contain references to multiple reference libraries (standard cells and macro cells). Also, a design library can be a reference library for another design library.

  The Milkyway library is stored as a UNIX directory with subdirectories, and every library is managed by the Milkyway Environment. The top-level directory name corresponds to the name of the Milkyway library. Library subdirectories are classified into different views containing the appropriate information relevant to the library cells or the designs.

  In a Milkyway library there are different views for each cell, for example, `NOR1.CEL` and `NOR1.FRAM`. This is unlike a `.db` formatted library where all the cells are in a single binary file. With a `.db` library, the entire library has to be read into memory. In the Milkyway Environment, the Synopsys tool loads the library data relevant to the design as needed, reducing memory usage. The most commonly used Milkyway views are `CEL` and `FRAM`. `CEL` is the full layout view, and `FRAM` is the abstract view for place and route operations.
- `simv` - Compiled simulator. This is the output of `vcs`. In order to simulate, run the simulator by `./simv` at the command line.
- `alib-52` - characterized target technology library. A pseudo library which has mappings from Boolean functional circuits to actual gates from the target library. This library provides De-

sign Compiler with greater flexibility and a larger solution space to explore tradeoffs between area and delay during optimization.

Bonus: A Glossary from Design Compiler

The Design Compiler User Guide (Chapter GL) has a good glossary on general VLSI stuff. If you are not familiar with the following keywords, we encourage you to take a look. You will see these terminologies in the labs and the tutorials a lot.

- annotation
- back-annotate
- cell
- clock
- clock gating
- clock latency
- clock skew
- clock source
- clock tree
- clock uncertainty
- core
- critical path
- datapath
- current design
- current instance
- design constraints
- false path
- fanin
- fanout
- fanout load
- flatten
- forward-annotate
- generated clock
- hold time
- ideal clock
- ideal net
- input delay
- instance
- leaf cell
- link library
- multicycle path
- netlist

- operating conditions
- optimization
- output delay
- pad cell
- path group
- pin
- propagated clock
- real clock
- reference
- RTL
- setup time
- slack
- structuring
- synthesis
- symbol library
- target library
- technology library
- timing exception
- timing path
- transition delay
- ungroup
- uniquify
- virtual clock
- wire load model

Review

In this tutoiral you have learned what each VLSI tools are meant to do, how they flow, file extensions of their inputs and outputs.

`vcs-sim-behav`	Synopsys VCS	Behavioral simulation
`vcs-sim-rtl`	Synopsys VCS	RTL simulation
`dc-syn`	Synopsys Design Compiler	Synthesis; RTL to gate-level netlist
`dc-syn`	Synopsys Formality	Formal verification
`vcs-sim-gl-syn`	Synopsys VCS	Post synthesis gate-level simulation
`icc-par`	Synopsys IC Compiler	Place and Route; gate-level netlist to layout
`vcs-sim-gl-par`	Synopsys VCS	Post place and route gate-level simulation
`pt-pwr`	Synopsys PrimeTime PX	Power analysis

Acknowledgements

Many people have contributed to versions of this tutorial over the years. The tutorial was originally developed for CS250 VLSI Systems Design course at University of California at Berkeley by Yunsup Lee. Contributors include: Krste Asanović, Christopher Batten, John Lazzaro, and John Wawrzynek. Versions of this tutorial have been used in the following courses:

- CS250 VLSI Systems Design (2009-2010) - University of California at Berkeley
- CSE291 Manycore System Design (2009) - University of California at San Diego