

Automatic VITAL Cell Library Generation for Logic and Timing Tools

Richard Edelman and Bulent Ay
SimQuest, Inc.
3235 Kifer Road, Suite 350
Santa Clara, CA 95051
rich@simquest.com

Abstract

The popularity of "cell based design" is increasing due to shortened design cycles and ease of use. Chip design tools are proliferating as small EDA companies innovate. In the past, the combination of these two trends has led to an explosion of proprietary, non-standard macrocell libraries.

This paper discusses automatic generation of VITAL compliant ASIC macrocell libraries from an existing intermediate format, such as Verilog, or CDF. The VITAL initiative is an attempt to "accelerate the development of sign-off quality ASIC macrocell simulation libraries written in VHDL by leveraging existing methodologies of model development". This paper discusses automatic generation of macrocell libraries for logic simulators, synthesis tools and timing analyzers. The concepts discussed in this paper are implemented with a software tool called the Advanced Library Tool (ALT). Because the conceptual model built into ALT matches published VITAL modeling standards so closely, VITAL compliant ASIC macrocell libraries can be generated relatively painlessly.

Introduction

The popularity of cell based design is increasing due to shortened design cycles, and ease of use. Cell based design tools generally require cell based model libraries. The cell based design tools include schematic capture, logic simulation, timing analysis, synthesis, fault simulation, place and route, floor planning, etc. A logic simulator will have a logic simulation library; a schematic capture tool will have a symbol library. While the design is cre-

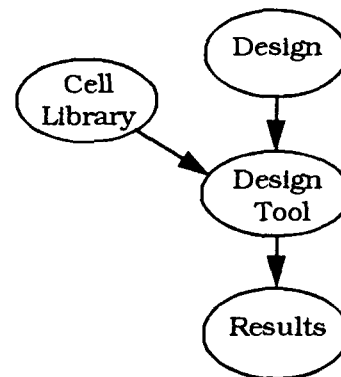


Figure 1. Cell Based Design.

ated by the designer, usually the cell library is created by either the supplier (ASIC foundry) or the design tool manufacturer (EDA company).

Many design tools have been created that have technical merit, ease of use, and superior general features - but lack cell library support. This paper discusses the automatic generation of cell models for use with timing tools such as logic simulators, synthesis tools, and timing analyzers. The concepts discussed in this paper are implemented with a software tool called the Advanced Library Tool (ALT).

Within the ALT environment all models are generated from an intermediate format called the Cell Description File (CDF), where a CDF may be generated either

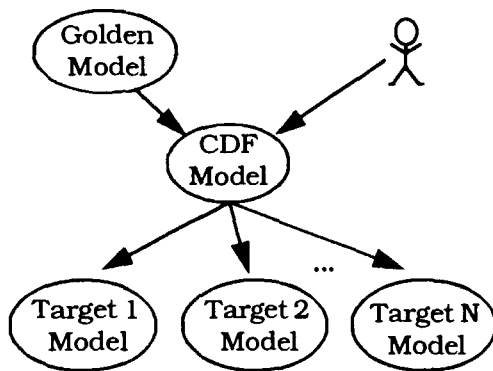


Figure 2. Cell Description File

manually or from other 'golden models' through synthesis and transformations.

Cell Description File - CDF

The Cell Description File is the intermediate modeling format adopted by the Advanced Library Tool. It is not a breakthrough in modeling, but rather is a clear, consistent way to describe a cell. CDF is an extensible format used to describe function, timing and attribute data for cells. Each CDF encapsulates all the known information about that cell in one place.

CDF models are easy to read, and easy to write. They can be checked for correctness with relative ease, even for complicated cells.

The CDF file is the source for each target environment. A CDF of a cell may be generated in two ways:

- by hand, or
- through automatic transformations from a "golden model".

A golden model is a model that has passed certification, and is considered to be an accurate representation of the timing and behavior of the cell. The golden model and the golden simulator together make up a golden environment.

Automatic CDF Generation

For users already possessing golden models, entering yet another model description is not realistic. These users must have a way to streamline the path from their existing golden models to target models. CDF can be generated from existing golden models using exhaustive simulation techniques and transformations.

Currently, ALT includes automatic CDF generation procedures for Verilog and Spice golden environments. This process consists of two phases:

- logic characterization, and
- timing characterization.

Before a CDF can be generated, a pre-CDF must exist which describes certain information necessary for the logic characterization to be correct. Pre-CDF may be created using heuristics from a Verilog model, a Spice model or by translating supplier data directly. The pre-CDF of a cell contains required and optional information. Required information includes:

- cell Type (combinational or sequential),
- pin Type (clock, control, enable, tri-state).

The optional information which may be needed for generation of certain target models includes:

- cell Area,

- cell Power,
- capacitance,
- fanout.

Correct CDF generation cannot occur until the pre-CDF information is complete and correct. Optional information is important only for target model generation.

Logical Characterization

Once pre-CDF has been created, and a golden simulation model has been supplied, "logical characterization" can begin, as shown in Figure 3.

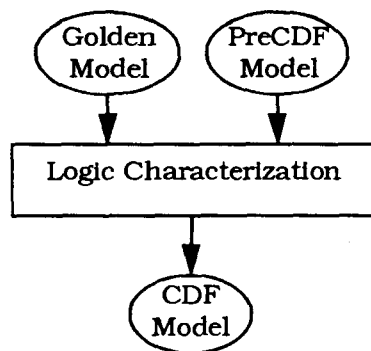


Figure 3. CDF Model Generation

Logical characterization is the process of taking a cell model of unknown behavior and synthesizing that model.

A golden model can be a Verilog model or a Spice netlist of a cell. Spice golden models are processed by a "switch level netlist transformer" for proper operation. This transformation converts a Spice netlist of a CMOS cell into a netlist of switches, represented in Verilog format. The generated netlist includes appropriate delay insertions to overcome "zero delay feedback" problems. The resulting switch level netlist is then treated as a Verilog golden model.

The pre-CDF information facilitates automatic simulation of the golden model. Simulation is performed using automatically generated simulation vectors, and the golden simulation model. The simulation results are synthesized by a Karnaugh map minimizer, which then produces a CDF functional model. Once logical characterization is complete, a new CDF is generated which includes the synthesized function and the pre-CDF information.

The automatically generated simulation vectors used in logic characterization are "exhaustive". For combinational cells, every possible functional combination of inputs is simulated. For sequential cells, the inputs are divided into three sets:

- data,
- control and
- clock.

Each set is treated individually to determine the state transitions. The sequential simulation vectors are skewed for every transition between data and clock signals. In addition, there are simulation vectors generated for determining high-Z outputs.

The minimization techniques are a combination of standard boolean minimization algorithms and proprietary heuristics. Combinational cells are represented as two level "sum of products" or "product of sums" terms. Sequential cells are represented as views of standard flip flops or latches. The logic characterization tool understands standard D, JK, and T flip flops, and a simple latch. These are known as internal sequential elements. Each internal sequential element has a set and reset control pin, and a single clock.

When a sequential element is logically characterized, it is mapped into an internal sequential element. Figure 4 shows a D flip flop (DFLOP1) with no set or reset, and two data pins implemented as an internal DFF with set and reset tied to ground, and an OR gate feeding from the

two D inputs to the single internal D input.

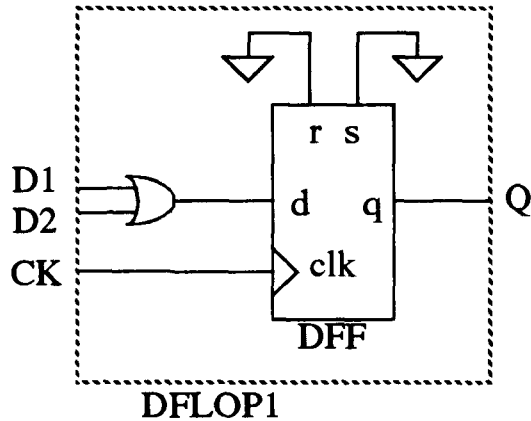


Figure 4. Implementation of cell DFLOP1

The logical characterization procedures are capable of characterizing all combinational functions and all one memory element sequential cells. The sequential cells include flip-flops, latches and their "scanned" versions. The characterization procedures also include recognition of tri-state, pull-up, pull-down, clocked, and enabled types of behavior.

Functional Table Characterization

At the end of Logical Characterization, the extracted boolean representation is further processed to generate table representations. These table representations are utilized during target model generations such as VITAL, Verilog UDP and Mentor QuickPart Table models.

The combinational outputs are simply represented as a minimized truth table of the boolean function synthesized for that output.

The sequential outputs such as flip-flops and latches are handled by "template state tables". A template state table for each of the ALT internal sequential elements is expanded using the boolean

equations synthesized for data, control, clock and output pins of the internal sequential element. The state table for the entire sequential model is generated by combining the minimized truth table for each internal pin.

Existence of multiple functional representations found in CDF simplifies the target model generations efforts.

Timing Representations

In CDF, timing is represented as user definable data attached to paths, ports, or internal nodes. Examples of timing data include:

- Pin to Pin (path delays),
- Setup and Hold limits,
- Recovery and Release limits, and
- Input and Output delays (lumped delays).

This attached timing data can then be treated in a user specific or tool specific way. Each user defined data type also includes its own set of transformation routines. For example, timing data may be represented in CDF as a piecewise linear curve - a simple set of straight lines. A target environment may only be able to handle a single, simple straight line. During target model generation, the model generator for this tool requests all timing data to be converted into linear approximations. At this time, a user specific conversion routine, such as a curve fitting algorithm, is applied, and the data is regenerated. Some accuracy may be lost due to the linear approximation, but this is an inherent limitation of the design tool, and not a limitation that is built into every cell model.

Timing Transformations

Generating a timing abstraction requires the timing model to be entered as pre-CDF. This can usually be accomplished by

writing a simple external interface between the vendor data and the pre-CDF.

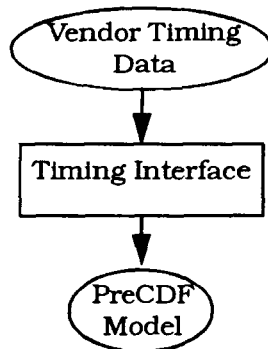


Figure 5. Generating PreCdf

Many timing abstractions can be represented in the existing Verilog golden simulation model with the use of "PLI" interfaces. These timing abstractions are extracted automatically by a simple model reader program.

More complex timing abstractions may need a special pre-CDF interface reader. The pre-CDF, including the timing abstraction, is merged with the generated functional CDF to produce a complete CDF. The final CDF model describes the functionality of the cell, along with the timing of the cell.

Model Generation - Targeting

Once a CDF has been created, models may be generated for target environments. Each target environment has its own model generator. For example:

- cdf2vital
- cdf2synopsys
- cdf2mentor

Generating a target model requires only the CDF of a cell. Each generator shares the same parser and a library of access

routines to generate target models, as shown in Figure 6.

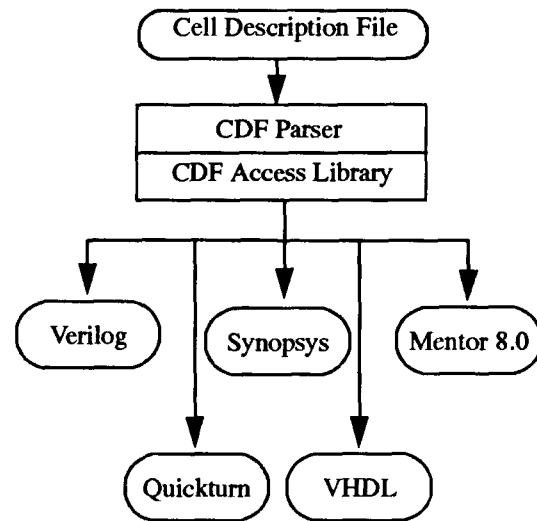


Figure 6. Access to CDF Data.

The function of a target model is generated to efficiently utilize that target environment. Depending on the target, generated models may use

- built-in simulator primitives,
- user-defined primitives,
- boolean equations, or
- state tables.

Each generator can produce all appropriate representations.

During functional realization of target models, target specific features are optimized for best performance. This process is called "targeting". For instance, Verilog targets use built-in primitives and user-defined primitives, where VITAL targets use VITAL primitives and VITAL tables.

Certain design tools have variable width primitives. Verilog has an 'and' gate that is generic and can have N inputs. For targets that do not have such generic primitives, built-in primitives of AND2, AND3, and AND4 might be used. To optimize for space, in a cell model of a 4 input 'and'

gate, an AND4 is used, and not a combination of AND2, or AND3.

Timing in the Target Environment

Many design tools have built in notions of timing. Each target model generator "understands" how to generate the target timing model, given a CDF timing model. Some target environments are flexible, and can represent timing as equations, relying on built-in netlist analysis. Some target environments do not have built-in equations, but allow timing to be calculated or annotated into "timing slots". Timing slots can be of two types, data timing slots, or actual delay timing slots. Timing slots are place holders for either data items, or actual delay.

Equations are evaluated by the target environment during simulation as defined for that design tool. An equation may be written to take advantage of high level netlist analysis functions that are actually built into the simulation environment. This type of model is sometimes easier to understand, since the timing equations for the timing calculation are actually documented directly in the cell model, do not exist in an external timing calculator, and are not hidden in an internal programming language (like PLI).

Data timing slots are placeholders that must be interpreted by the timing model, or by an internal timing calculator. For instance, a timing calculator may be called, while inside the simulation environment, to calculate the path delay across a cell. The timing calculator assumes that certain data items have been provided in the model, such as slope and intercept. The timing calculator may then perform netlist analysis, such as fanout counting, or it may actually read a layout capacitance file. This fanout information or layout capacitance is then used to turn the slope and intercept into actual delay.

Delay timing slots exist in timing environments where the actual delay must be either specified in the model, or placed in the model by a delay annotator. Delay timing slots depend on an external timing calculator to calculate all the timing information needed by the model. After the instance specific timing has been calculated externally, it is simply annotated to the proper place within the model.

Conclusions

Advanced Library Tool has been in use since September 1991. Using ALT, an entire library of ASIC cell models can be generated in a matter of hours.

The models generated by this methodology are consistent in terms of functionality, naming, attributes and other characteristics, thus providing similar "look and feel" on different platforms and design tools.

The models generated by each generator are considered correct by construction, once the operation of the generator is verified. Therefore, functional verification is reduced to a single task of verification of the generator, compared to the verification of each model generated through manual efforts.

The current implementation of the tool can target about ninety percent of a model library. Support for the remaining cells, such as counters, state machines and memories, is a planned enhancement to the tool and the CDF syntax, now under development. Future enhancements to ALT will include symbol generators, documentation generation, automatic timing characterization, along with more target environments.

The current implementation requires users to provide all timing characteristics. A new tool is under development to overcome this requirement. This new tool will characterize a cell, given only the SPICE netlist and parameters of the cell and its underlying technology.

Acknowledgments

The authors would like to acknowledge the contributions made by CAE vendors: Cadence Design Systems, Mentor Graphics, Synopsys Inc., IKOS Systems, and OrCad. Mentioned trademarks belong to their respective owners.

References

[1] VITAL, "VHDL Initiative Toward ASIC Libraries" Model Development Specification, Version v2.2b, March 1994.

[2] Cadence Design Systems, "Verilog-XL Reference Manual" version 1.6, Mar. 1991, San Jose, California.

[3] Mentor Graphics, "QuickPart Model Development Manual" version 8.0_1, Sept. 1991, Beaverton, Oregon.

[4] Synopsys Inc., "Library Compiler Reference Manual" version 2.0, Mar. 1991, Mt. View, California.