BRADLEY DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING AT
VIRGINIA POLYTECHNIQUE INSTITUTE AND STATE UNIVERSITY

Towards Automation of ASIC TSMC 0.18 μm Standard Cell Library Development

DJIGBENOU, Jeannette Donan

Thesis Submitted to the Faculty of the Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements for the degree of

Master

In

Electrical Engineering

Dr. HA, Dong S. Chairman
Dr. SCHAUMONT, Patrick Robert
Dr. TRONT, Joseph G.

May 5, 2008
Blacksburg, Virginia

Keywords: VLSI design, CAD Tools, Standard cell library, Automation, ASIC design, Quality Assurance, CMOS technologies

Towards Automation of ASIC TSMC 0.18 μm Standard Cell Library Development

By Djigbénou, Jeannette Donan
Dr. Ha, Dong S., Chairman
Bradley Department of Electrical and Computer Engineering

(Abstract)

Cell-based design is a widely adopted design approach in current Application Specific Integrated Circuits (ASIC) and System-on-Chip (SOC) designs. A standard cell library is a collection of basic building blocks that can be used in cell-based design. The use of a standard cell library offers shorter design time, induces fewer errors in the design process, and is easier to maintain.

Development of a cell library is laborious, prone to errors and even a small error on a library cell can possibly be disastrous due to repeated use of the cell in a design. In this thesis, we investigated ways to automate the process for development of a cell library, specifically TSMC 0.18-micron CMOS standard cell library. We examined various steps in the design flow to identify required repetitive tasks for individual cells. Those steps include physical verification, netlist extraction, cell characterization, and generation of Synopsys Liberty Format file. We developed necessary scripts in Skill, Tcl, Perl and Shell to automate those steps. Additionally, we developed scripts to automate the quality assurance process of the cell library, where quality assurance consists of verifying the entire ASIC design flow adopted for the Virginia Tech VLSI Telecommunications (VTVT) lab. Our scripts have been successfully used to develop our TSMC 0.18-micron library and to verify the quality assurance. The first version of the cell library was released on November 1, 2007 to universities worldwide, and as of March 2008, 20 universities have received the library from us.

Acknowledgments

My utmost gratitude goes to Jesus-Christ, my Lord and Savior.

I would like to express my deep and sincere appreciation to my committee chairman and advisor Dr. Dong S. Ha for his detailed and very constructive input in this work. I am deeply grateful to my committee members Dr. Joseph Tront and Dr. Patrick Schaumont for their resourceful support.

I would also like to thank all the undergraduates who have worked with me diligently. Special thanks are due to the Virginia Tech VLSI for Telecommunications (VTVT) Lab members for their technical support and help. Thanks to Shaimaa Abdallah from the Virginia Tech Alexandria campus in Egypt for her encouragement and the "virtual café" discussions on research and life.

I owe special thanks to the Virginia Tech staff and faculty, especially John Harris, the ECE department System Administrator for his support, and for the long hours he put on installing numerous software tools I needed for this research. I would also like to thank Dr. Timothy Pratt in the Communications department for his encouragements. Thank you for being such an inspiration.

Cypress Semiconductor Inc. is also due my thanks for providing me with work experience through internships and co-op opportunities during my undergraduate and graduate studies. Thanks to Ronald Kalim as well as my former managers: Dragomir Nikolic, Robert Peddenpohl, Brian Evans and Derrick Savage, and all the design, modeling and CAD teams in Lexington, Kentucky and Starkville, Mississippi. I would like to acknowledge that this material is based upon work supported, in part, by the National Science Foundation under Grant No. 0551652, and also by the honor attributed to me through the Webber fellowship.

Finally I would like to express a very special gratitude to my entire family and friends, namely Luc Nounagnon, Firmin and Annah Djigbénou, Antoine and Salomé Djigbénou, my brother and sisters for being a constant and vital support to me through the years. I dedicate this work to all of you.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Cell-based design is a widely adopted design approach in current Application Specific Integrated Circuits (ASIC) and System-on-Chip (SOC) designs. A standard cell library is a collection of basic building blocks that can be used in cell-based design. The use of a standard cell library offers shorter design time, induces fewer errors in the design process, and is easier to maintain.

The Virginia Tech VLSI for Telecommunications (VTVT) Lab is one of the leading research labs in the nation providing standard cell libraries to academic institutions worldwide. In 2002, the VTVT Lab released its first standard cell library on TSMC 0.25 μm. As of March 2008, 281 universities worldwide have used VTVT's cell libraries. Other external sites introduce our cell libraries to specific design communities. On one hand, Graham Petley, author of the coming book: "The Art of Standard Cell Library Design", provides information to the industrial community about the VTVT standard cell libraries [1]; while MOSIS, on the other hand, provides resources to the academic community about the VTVT standard cell libraries [2].

These cell libraries have a significant impact on education and research worldwide. As commercial cell libraries are not freely accessible and contain propriety information, their use in academia impedes VLSI research. The VTVT lab took the initiative to develop standard cell libraries using up-to-date commercial tools. These libraries are freely accessible to the academic community worldwide. Since the release of the TSMC 0.25 μm library in 2002, we have received many requests to provide cell libraries for other technologies. The impact of our cell libraries in the academic community and users' requests are the driving force for this research.

To address the need for faster development of standard cell libraries with higher quality assurance, we introduce a method to automate the design process of standard cell libraries and their quality assurance. In Chapter 2, we describe an overview of standard cell libraries and present previous works in standard cell library development within the academic community. In Chapter 3, we elaborate on the development of VTVT 0.18-micron library from an updated release of VTVT 0.25-micron library. Finally, in Chapter 4, we expand on the automation approach proposed.

This thesis research aims to develop a 0.18-microns standard cell library while automating the process as much as possible. The automation of the development flow consists of generating Skill, Tcl, Perl and Shell scripts that perform physical verification, netlist extraction, cell characterization, and generation of Synopsys Liberty Format file. In this thesis, we also address a verification procedure of our cell libraries through automation. The scripts developed to perform quality assurance on each cell of the library execute the entire ASIC design flow adopted by our lab.

In this thesis, we present results from our automation process for quality assurance and development of our cell libraries. We updated supported tools for VTVT 0.25 μm library and verified the library using the automated quality assurance procedure presented in this thesis. We developed VTVT 0.18 μm library based on the automated development procedure from this thesis and verified the library using our quality assurance scripts. The development process of VTVT 0.18 μm library using automation was initiated in August 2007. On November 1, 2007, the TSMC 0.18 μm library was released.

2. Background

Standard cells are basic building blocks for ASIC design, which improves designers' productivity through reduced design time and debugging. The following section describes the ASIC design flow, the advantages and limitations of ASICs and the trends in design methodologies as well as the motivation for automating the development process of standard cell libraries of ASIC designs. Additionally, previous works on standard cell libraries in academia are presented, including VTVT 0.25-microns standard cell library.

2.1 ASIC Design with Standard Cell Libraries

This first section presents an overview of standard cell libraries, their advantages and disadvantages in comparison with full custom designs and their applications and trends.

2.1.1 ASIC Cell-Based Design Flow

An ASIC design flow typically starts with a VHDL or Verilog description of an intended design. The description is synthesized to a gate level circuit and placed-and-routed to generate its layout. Standard cells are the building blocks of the gate level circuit and the layout.

The design is described in VHDL for the ASIC design flow adopted by the VTVT lab,. It is simulated for verification and then synthesized into a Verilog netlist. If the Verilog netlist meets the designer's specifications, the netlist is imported into Cadence Integrated Circuit Front to Back (ICFB) as a schematic view and into Cadence SOC Encounter to generate a place-and-route layout view. Once the layout view is imported into Cadence ICFB, physical verification; consisting of Design Rule Check (DRC) and Layout versus Schematic (LVS) are run to verify that the synthesized schematic and layout match. The I/O pads are added manually to the design and submitted to the foundry for fabrication. Power estimations can also be executed after the design synthesis stage.

Figure 1 describes the details of VTVT's design flow, which was also used in the verification process for our standard cell library development; specifically TSMC 0.18 μm.

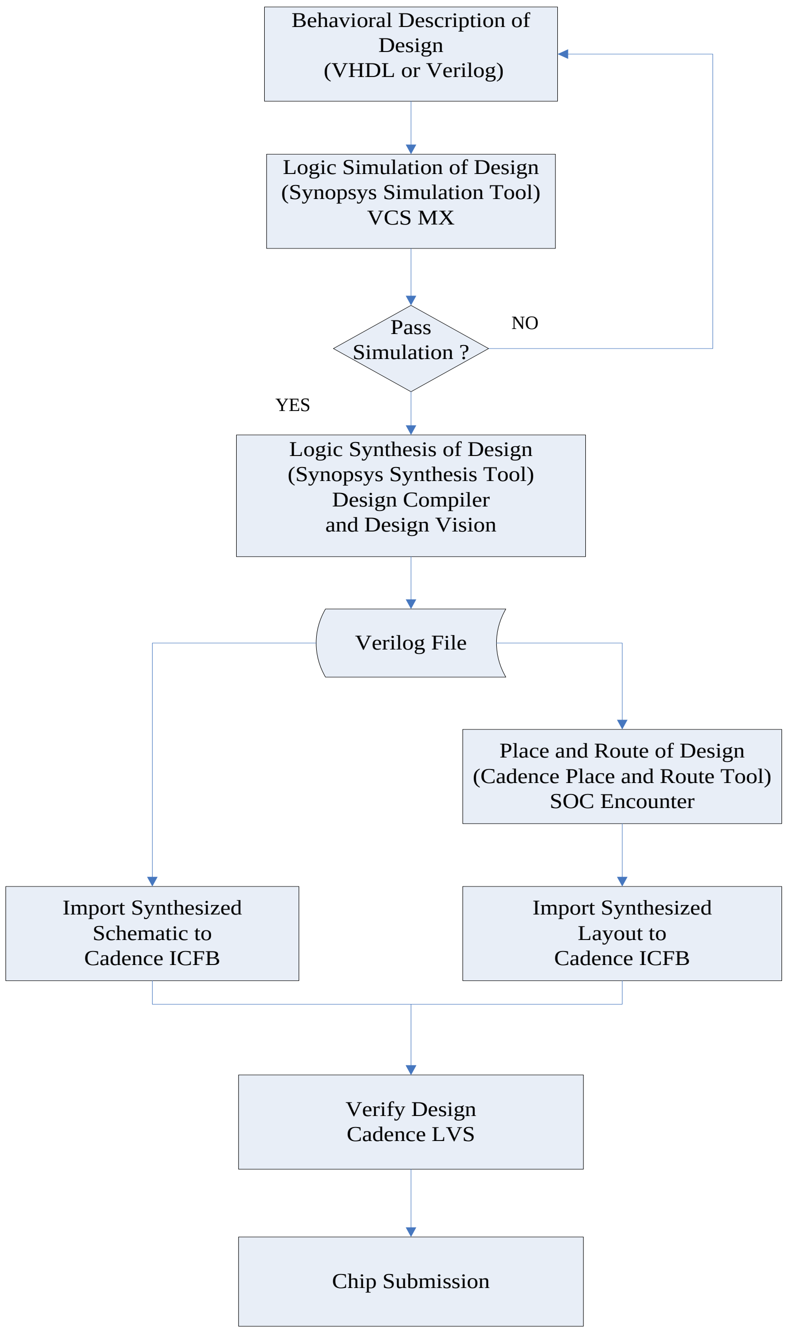

Behavioral Description of Design (VHDL or Verilog)
Logic Simulation of Design (Synopsys Simulation Tool) VCS MX
Pass Simulation ?
NO
YES
Logic Synthesis of Design (Synopsys Synthesis Tool) Design Compiler and Design Vision
Verilog File
Place and Route of Design (Cadence Place and Route Tool) SOC Encounter
Import Synthesized Schematic to Cadence ICFB
Import Synthesized Layout to Cadence ICFB
Verify Design Cadence LVS
Chip Submission

Figure 1: ASIC Design Flow

2.1.2 Advantages and Limitations of ASIC

Cell based ASIC top-down design approach shortens the design time compared with full custom design and promotes design reuse to lower the cost. However, ASICs' performance is less remarkable than full custom design with regard to speed performance. Indeed, a summary of

Chinnery's study indicates that a full custom microprocessor based on a 180 nm process would operate at 1-2 GHz, while the same design using ASIC would operate at 200-350 MHz [3].

2.1.3 Applications and Trends

ASIC design has been used in multiple applications including: telecommunications, home appliance electronics, aerospace and medical applications. The demands for more capabilities in smaller devices within a shorter design time, as shown on Figure 2, will metamorphose ASIC design methodology from a single processor with a specific application to multiple systems and processors on a single chip. This new generation of ASIC, called SOC, is the new trend that will eventually lead the design community in coming years.

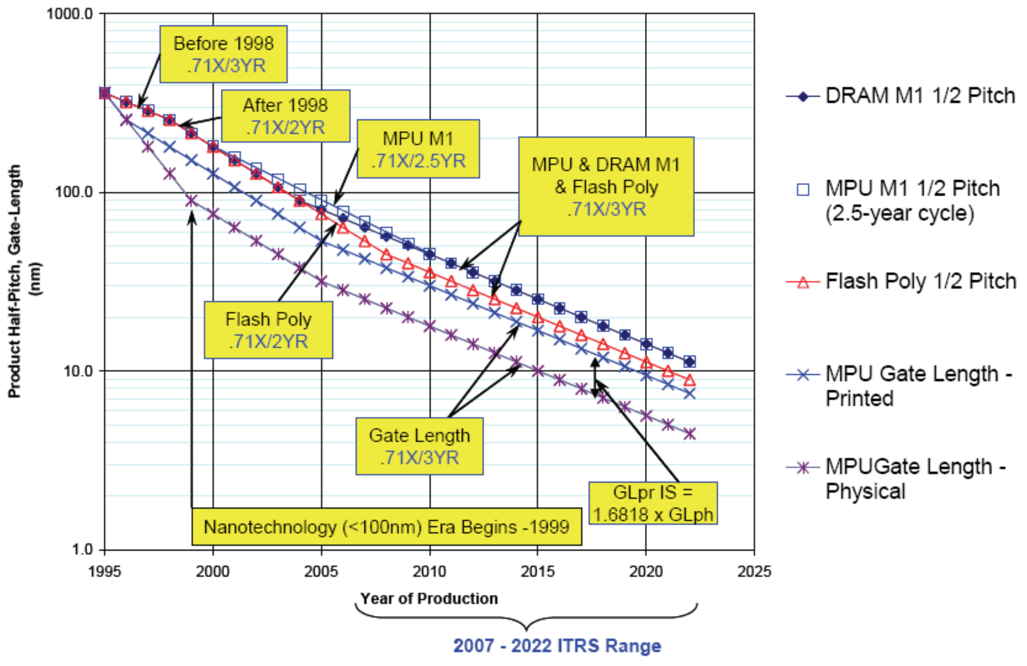


Figure 2: 2007 ITRS Product Technology Trends - Half Pitch, Gate Length [4]

As designers move from ASIC to SOC, the need for standard cell libraries regains interest, because cell libraries can be used in both design approaches. Also, standard cell libraries remain the basic design building blocks. Numerous reports target productivity improvements in design through automation of the ASIC design flow and the layout of standard cells. This thesis research intends automation of the development process for standard cell libraries, which reduces the time and cost for development of cell libraries..

2.2 Standard Cell Libraries in Industry

Independent of the design methodology adopted, Integrated Circuits (IC) design companies use standard cell libraries to reduce the design time and improve the reusability of the designs. Due to large resources needed to develop standard cell libraries, many companies rely on standard cell libraries developed by other companies. Some other companies develop their own standard cell libraries for internal purposes.

2.3 Motivation for Development of Standard Cell Libraries in Academia

Commercial cell libraries are proprietary information of the suppliers who usually impose restrictions on the access and use of their library cells. Those restrictions on commercial library cells hamper VLSI research and teaching activities of academia. In order to address this problem, the VTVT Lab took an initiative to develop and distribute standard cell libraries under the sponsorship of the National Science Foundation. The VTVT Lab currently distributes standard cell libraries based on CMOS TSMC 0.18 μm and TSMC 0.25 μm free of charge to academic institutions around the world. As of March 2008, approximately 280 universities worldwide have received the cell library from the VTVT group.

2.4 Previous Works on Standard Cell Library Development in Academia

This section presents previous works on academic development of standard cell libraries and focuses on the evolution of VTVT Lab's work on standard cell library development.

2.4.1 Academic Standard Cell Libraries

Several academic institutions have offered open source standard cell libraries. Dr. Graham Petley develops a website that includes information on academic open source standard cell libraries [1]. Table 1 summarizes it.

Table 1: Sample List of Academic Institutions Developing Open Source Standard Cell Libraries

Academic Institutions	Number of Cells	Supported Technologies
Université Pierre et Marie Curie	91	0.13 µm
Illinois Institute of Technology/Oklahoma State University	30-35	AMI 0.5 µm, AMI 0.35 µm, TSMC 0.25 µm, TSMC 0.18 µm
University of Tennessee	14	AMI 0.6 µm
Virginia Tech	83	TSMC 0.25 µm, TSMC 0.18 µm

2.4.2 VTVT 0.25 Standard Cell library

VTVT 0.25 µm cell library was initially developed by a team of undergraduate and graduate students lead by former VTVT member Dr. Jos B. Sulistyo under the supervision of Dr. Dong Ha. The original release supported the following tools:

- Synopsys Design Compiler and Design Analyzer, as library compilation and logic synthesis tools, respectively.
- Synopsys VCS-MX, as a logic simulation tool.
- Cadence Silicon Ensemble, as a place-and-route tool.
- Cadence Virtuoso, as a verification and layout tool.

In the design flow adopted for the VTVT standard cell library, the design entry is a VHDL or Verilog behavioral description. Using Synopsys synthesis and simulation tools, the behavioral description of the design is simulated and then synthesized into a gate-level circuit. The synthesized design is extracted in the Verilog format and imported into the Cadence physical design environment. After place-and-route of the design, the synthesized layout is compared against its schematic. This flow is described on Figure 1.

The target technology for the VTVT standard cell library is TSMC 0.25 µm CMOS with a supply voltage of 2.5 V. The cells were developed using the NCSU Cadence Design Kit [4] and following MOSIS deep rules. The cell library contains 84 cells with various primitive gates, multiplexers, and flip-flops with a variety of drive strengths and dummy pads. It includes layout, schematic views and VHDL functional models of all the cells, to support logic simulation and synthesis, place-and-route, and LVS (Logic Versus Schematic). The list of entire cells is available in Appendix A.

The remainder of this thesis focuses on the steps taken toward the development of a TSMC 0.18-micron standard cell library and the automation of its implementation and verification process. With the idea of reuse in mind, VTVT 0.25 μm standard cell library was used to develop VTVT 0.18 standard cell library.

3. Development of VTVT 0.18 μm Standard Cell Library

Prior to using this cell library for developing a new cell library, named VTVT 0.18-microns, the original library VTVT 0.25-microns has been updated with newer design tools and more features, and the updated version was released in December 2006. We briefly describe update of VTVT 0.25-microns standard cell library in Section 3.1. The major tasks for the development of a cell library are: layout of cells, characterization, and LEF file generation. We elaborate the procedure to develop VTVT 0.18 μm standard cell library based on our VTVT 0.25 μm cell library. Section 3.2 presents an overview of the new technology standard cell library. Section 3.3 describes the tools used to develop the VTVT 0.18-micron cell library. Finally Section 3.4 uncovers the development procedure for the new library.

3.1 Update of VTVT 0.25 μm Standard Cell Library

Our 0.25-microns cell library was updated using up-to-date commercial design and simulation tools with the intention to promote users as well as developers to experience the real world design environment. A design synthesis tool from Synopsys, Design Analyzer, was replaced by Design Vision as Synopsys discontinued support of Design Analyzer. The place-and-route tool, Cadence Silicon Ensemble was also replaced by a newer Cadence tool, SOC Encounter. The updated version of the VTVT 0.25-microns standard cell library supports the following tools:

- Synopsys Design Compiler and Design Vision, as library compilation and logic synthesis tools, respectively.
- Synopsys VCS-MX, as a logic simulation tool.
- Cadence SOC Encounter, as a place-and-route tool.
- Cadence Virtuoso, as a verification and layout tool.

The original VTVT 0.25-micron library did not include a symbol library. A symbol library for VTVT 0.25-microns cell library was developed through the process described in Section 3.1.1. As a symbol library is technology independent, it was necessary to modify only the cell library name in the ASCII file (vtvt_tsmc250.slib) prior to being compiled into vtvt_tsmc250.sdb. The new release includes a symbol library named vtvt_tsmc250.slib and vtvt_tsmc250.sdb. We describe development of the symbol library in the following.

3.1.1 Development of a Symbol Library

Symbols are gate level views of layout and schematic cells. A symbol library is the combination of all symbol views of a given standard cell library. A symbol library allows users to view customized cell symbols rather than generic symbols, typically black boxes in either a design synthesis tool or Cadence Virtuoso Schematic Editor. A symbol library also allows synthesized designs to be exported back from the synthesis tools into CAD tools for further design verification. The symbols for a 2-input multiplexer and an inverter are illustrated in Figure 3 .

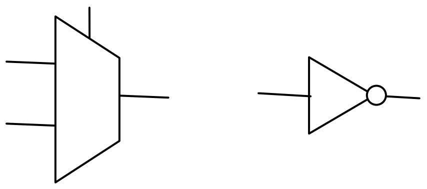


Figure 3: A 2-Input Multiplexer and Inverter Symbol Views

3.1.2 Procedure for Generation of a Symbol Library

Our symbol library was developed using similar symbol views as Artisan's library cells. Symbols were drawn using Cadence Virtuoso Schematic and Symbol editing tools. Then, symbols were exported into an EDIF file, which was compiled into an ASCII symbol library (.slib file) and eventually .sdb file using Synopsys Design Compiler. Figure 4 shows the process for generating a symbol library. Details on this process are provided in Appendix G.

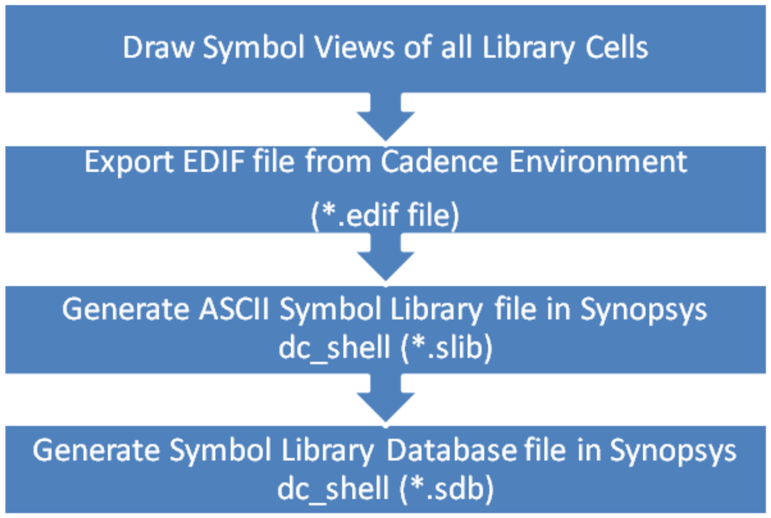

Draw Symbol Views of all Library Cells
Export EDIF file from Cadence Environment (*.edif file)
Generate ASCII Symbol Library file in Synopsys dc_shell (*.slib)
Generate Symbol Library Database file in Synopsys dc_shell (*.sdb)

Figure 4: Generation of a Symbol Library

3.2 Overview of VTVT 0.18 μm Standard Cell Library

According to TSMC 0.18 μm platform documentation, TSMC 0.18 μm process has numerous capabilities, such as embedded memories, mixed- signal/RF, high voltages and CMOS image sensor processes. Device applications include high performance 3D graphics, digital TV and high capacity programmable logic devices. The natural design rules accommodate densities up to 160,000 gates per square millimeter. The process performance boasts logic and on-chip memory speed of over 500 MHz and power gate dissipation less than 3 nW/MHz [5].

Development of the VTVT 0.18 μm standard cell library targets the CMOS standard logic CL018 process based on TSMC 0.18 μm technology available through MOSIS. The technology supports 6 Metals and 1 poly layer, and the supply voltage is 1.8 V. MOSIS deep rules SCN6M_DEEP uses a lambda value of 0.09 μm [6]. Our VTVT 0.25-micron standard cell library is composed of 83 digital cells including combinational and sequential cells. Those cells have been modified to meet the specifications of TSMC 0.18 μm technology. The list of cells is available in Appendix A.

3.3 EDA Tools Used for VTVT 0.18-microns Library Development

VTVT TSMC 0.18 μm cell library was developed using commercial design and simulation tools, so students involved in the project could experience the real world design environment. The tools used to develop the standard cell library are described in this section.

3.3.1 Cadence Design Kit: NCSU_CDK

The Cadence Design Kit, NCSU CDK, developed by Professors Paul Franzon and Rhett Davis' team at North Carolina State University supports both analog and digital full custom designs. It provides Diva rules for Design Rule Check (DRC), Layout Versus Schematic (LVS), extraction of parasitic capacitance and HSpice simulation parameters. It supports multiple technologies including the TSMC 0.18 μm design rules from MOSIS' processes. A download of the NCSU CDK is available on the NCSU web site [4]. We used the NCSU design kit to generate both VTVT 0.25 and 018 μm standard cell libraries.

3.3.2 Cadence Virtuoso

Cadence Virtuoso Layout, Schematic and Symbol tools are used to generate layout, schematic and symbol views for a specific design respectively. The tool provides the graphical user interface to draw integrated circuit designs and run physical verifications (DRC, LVS) and netlist extraction. Our cells were laid out using Cadence Virtuoso Schematic, Layout and Symbol Editors.

3.3.3 Cadence Abstract Generator

The Cadence Abstract Generator tool is used to generate information on the library that will be used in the Place-and-Route process. It uses layout and technology information from the standard cell library to generate abstract views of each cell, then models and combines the library information into a Layout Extraction Format (LEF) file that is imported to the Place-and-Route tool. Cadence Abstract was used to generate abstract views and the LEF file of our libraries.

3.3.4 Cadence SOC Encounter

The tool provides automated floorplaning, routing and physical verification of the design. It can be used for design synthesis, and incorporates analysis for efficient timing, area and power performance of a full-chip. Cadence SOC Encounter is the Place-and-Route tool used to develop and test the performance of our cell libraries.

3.3.5 Synopsys Design Compiler

Design Compiler was used to generate the simulation files and functional models of the library cells. It was also used to compile the technology file (.lib file), the symbol library file (.slib file) and all setup files for design synthesis and logic simulations.

3.3.6 Synopsys Design Vision

Synopsys Design Vision is Synopsys Design Compiler's graphical user interface for synthesis. It can be used to perform power, area and timing estimation of full integrated circuits. Design Vision was used to test the design synthesis performance of the library.

3.3.7 HSpice

HSpice is a SPICE simulator of Synopsys. HSpice was used in simulations for characterization purposes.

3.4 VTVT 0.18-micron Standard Cell Library Development Procedure

Similarly to an IC development, implementation of a cell library also requires organized design stages as shown in Figure 5. The first step consists of creating layout and schematic views of each cell of the library. Secondly, the cells are tested for physical verification and their netlist extracted. Then, each cell is simulated for capacitance, timing delay and power dissipation. The following step consists of generating simulation and functional models of each cell. During the fifth phase, abstract views of cells and a LEF file for the library are created; followed by the design of a symbol library in the sixth. Finally, the entire library designed can be tested prior to being released to design engineers for the implementation of a full-custom design. Each design stage is detailed in the subsequent sections. Some of the steps have been automated in the development process and will be discussed in Chapter4.

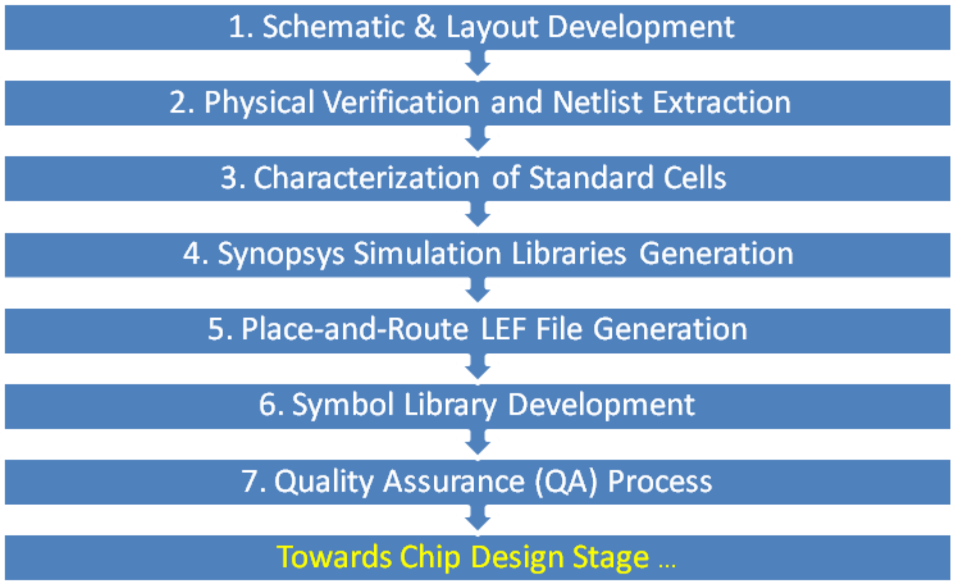

1. Schematic & Layout Development
2. Physical Verification and Netlist Extraction
3. Characterization of Standard Cells
4. Synopsys Simulation Libraries Generation
5. Place-and-Route LEF File Generation
6. Symbol Library Development
7. Quality Assurance (QA) Process
Towards Chip Design Stage ...

Figure 5: Standard Cell Library Development Cycle Stage

3.4.1 Schematic and Layout Libraries Development

The layout and the schematic of each cell were originally created by a former VTVT member, Dr. Jos Sulistyo, who developed and released VTVT standard cell library based on TSMC 0.25 µm [7]. The list of the cells is available in Appendix A. The 0.25 µm cells were implemented with a lambda of 0.12 µm, using MOSIS SCN5M_DEEP rules included in a NCSU Cadence Design Kit. In order to generate VTVT TSMC 0.18 µm standard cells layout views, and with the intention of design reuse, the layout cells from the 0.25 µm technology were scaled down manually to develop the 0.18 µm cells with a new lambda value of 0.09 µm. Then, the transistor sizes in the schematic views were updated based on changes applied to the layout views. The schematic view was modified based on the TSMC 0.18 µm transistor model and the new device size. A tutorial was written to assist a team of students working on this part of the project and is available in Appendix D.

Our cells were laid out manually with Cadence Virtuoso and follow MOSIS DEEP rules (SCN6M_DEEP). Though TSMC 0.18 µm technology includes 6 metal layers, only Metal 1 and Metal 2 layers were used to draw the layouts of cells. This constraint allows Place and Routing (P&R) tools to use the remaining metal layers for routing. In addition to the design rules, the layout of cells follows rules shown in Table 2, which are necessary for P&R. The cells were scaled down by reducing the layout layers by a ratio of both technologies' lambda values: 0.12

and 0.09. A tutorial and a checklist in Appendix B and Appendix C, respectively, include a set of rules that must be taken into account when generating a scaled down standard cell layout view. A summary of these rules is given in Table 2, where lambda is 0.09-microns for VTVT 0.18-micron library.

Table 2: Rules for Cell Layouts for VTVT 0.25 μm and 0.18 μm Libraries

Cell Design Settings	Values
Cell Height	108 λ
Cell Width	Multiple of 9 λ
Metal Pitch	9 λ for metal 1 through metal 4, 18 λ for metal 6
Metal Width	4 λ for all layers
Metal Offset	0 for all layers
Power/Ground Pins/Rails	11λ for VDD and VSS

Note: λ is 0.12 μm (0.09 μm) for VTVT 0.25 μm (0.18 μm) library

The layout and schematic scaling down process could have been implemented with a new Cadence tool: Virtuoso Layout Migration (VLM). This idea was considered initially, but dropped due to tool compatibility problems faced at Virginia Tech.

3.4.2 Physical Verification and Netlist Extraction

The layout cells, once scaled down, were verified using the Diva Design Rule Check (DRC) provided with the NCSU CDK. All cells had to be DRC clean prior to taking any further actions. This ensured that a chip follows the recommended metal size, spacing and orientation proposed by the foundry, i.e. TSMC for this technology. When the layout cells were DRC clean, Layout Versus Schematic (LVS) was run to guarantee that the scaled down schematic and layout netlists match. These LVS rules were implemented using Diva and were also included in the NCSU CDK. The physical verification process was completed when the design was LVS clean. To ensure the quality of the verification process and to reduce the design time, this stage of the development of the library was automated. Details on the automation process will be presented in Chapter 4.

3.4.3 Characterization of Standard Cells

Characterization of cells consists of capturing key parameters of cells such as propagation delay, rise and fall delays, and power dissipation. It was performed by extracting SPICE netlists from layouts and then simulating them using a SPICE simulator, HSPICE.

The characterization of VTVT's standard cell libraries is processed using a method developed by Dr. Jos Sulistyo as a Master's thesis [7]. The method proposed by Dr. Jos Sulistyo consisted of simulating the extracted netlists and using the Synopsys linear CMOS model to represent the library cells for input capacitance, dynamic and static power dissipation, cell area and cell rise and fall delays.

3.4.3.1 HSpice Simulations

The netlists of the library cells extracted after physical verification are simulated using HSpice. The N-MOS and P-MOS used for simulation on TSMC 0.18 µm library cells characterization are TSMC18DN and TSMC18DP Spice models level 49, respectively. The simulation setup for the library cells is also included in Dr. Jos Sulistyo's Master's thesis. These HSpice simulations have been automated in the development process of VTVT 0.18-microns library.

3.4.3.2 Synopsys Liberty File Format

The results from the HSpice simulation for all library cells are combined into one Synopsys Liberty File Format or commonly called ".lib file". The Liberty format file from Synopsys supports multiple types of models on standard library cells. The Synopsys linear CMOS model used in VTVT's characterization is the first and simplest model in comparison with current models available on the EDA market. Moreover, Synopsys Liberty file format is the most popular and widely used format file by standard library designers. The process for generating the ".lib file" has also been automated.

3.4.4 Synopsys Simulation Libraries Generation

Simulation libraries are used to generate VHDL, Verilog and logic simulation models of a given standard cell library. The Synopsys library file format is the database used to generate all simulation libraries required for using the standard cell library in the Synopsys environment. These simulation libraries are generated either with Library Compiler or Design Compiler. VTVT standard cell libraries use VHDL models for design synthesis. The steps to generating simulation libraries mentioned in Appendix E are also included in the VTVT Synopsys_Libraries directory of the VTVT release package for the standard cell library.

3.4.5 Place-and-Route (P&R) LEF File Generation

This design stage consists of generating a Layout Exchange File (LEF file) which is supported by Cadence SOC Encounter, the place-and-route tool supported by VTVT 0.18 μm library. A LEF file contains detailed information on the location of each cell's layers and their dimensions. It is created from the combination of the technology information of the library and the abstract views of each cell. Mapping files, which enable Cadence Virtuoso to import the layouts from Cadence SOC Encounter, were generated manually based on the Cadence SOC Encounter mapping format. This section describes how the LEF file was generated for VTVT 0.18 μm library.

3.4.5.1 Technology File Update

The original NCSU CDK does not support Place-and-Route, and we modified the kit to support it. We created a technology file (TF file) containing technology information such as: lambda value, metal layers' resistance, interconnects, layer available for the technology, and layers' mapping number.

3.4.5.2 Generation of a LEF file

The TF file is inserted into Cadence Abstract in order to generate abstract views of the library cells. The abstract cells are then used to generate the LEF file for the specific standard cells library. Appendix F contains the steps undertaken to generate the LEF file for TSMC 0.18-μm library.

3.4.6 Symbol Library Development

In the new release of VTVT 0.25 μm standard cell library, we created and included a symbol library. Since symbol libraries are technology independent, they can be ported to other cell libraries supporting different technologies. Therefore, the symbol views from VTVT 0.25-microns library were copied into VTVT 0.18-micron standard cell library, and the library name in the ASCII file of the symbol library was modified to vtvt_tscm180 and compiled into vtvt_tsmc180.sdb file. The symbol library component of VTVT 0.18-microns was generated. Details on this symbol library process are provided in Appendix G and H.

3.5 Quality Assurance Process

Quality Assurance (QA) is an important process of the standard cell library development as these cells are the building blocks of ICs. It is to verify that the cell library performs as expected. The overall performance of the library cells is tested with a 4-bit counter from a VHDL description to the chip submission stage. The QA adopted for the development of the VTVT cell library is described below:

3.5.1 Physical Verification

Physical verification is performed by running Design Rule Check (DRC) and Layout versus Schematic (LVS) on each cell of the library by executing a Skill script (alllib_drc.il) that will be presented in the next chapter. This step guarantees that proper instantiations of the cell library in a full custom design will not fail design rules.

3.5.2 Design Synthesis

All cells must be properly instantiated in the design synthesis tool. Therefore multiple VHDL scripts were implemented to test the instantiation of each cell in design synthesis. A golden model to implement each VHDL file was created, and all VHDL files were automatically tested through design synthesis. The automation process within design synthesis is detailed in Section 4.2.1.

3.5.3 Place-and-Route

Each of the synthesized designs from the previous section are automatically imported into the place-and-route tool (Cadence SOC Encounter). A QA on place-and-route is achieved by running place-and-route on all the cells and ensuring that they get properly instantiated within the place-and-route phase of an IC design A correct instantiation of cells within a design occurs when the full-custom design passes connectivity and geometry checks within Cadence SOC Encounter. This QA process is automated for each cell, which ensures that the process is not dependent on human errors. Details in the automation process are shown in Section 4.2.2.

3.5.4 Design Example - 4-bit Counter

A 4-bit counter whose block diagram is shown in Figure 6 is used as a verification of the design flow for VTVT 0.18-microns standard cell library. A VHDL description for the counter is given in Figure 6. The VHDL description has been synthesized into a gate level circuit using Synopsys Design Vision and placed and routed with Cadence SOC Encounter.

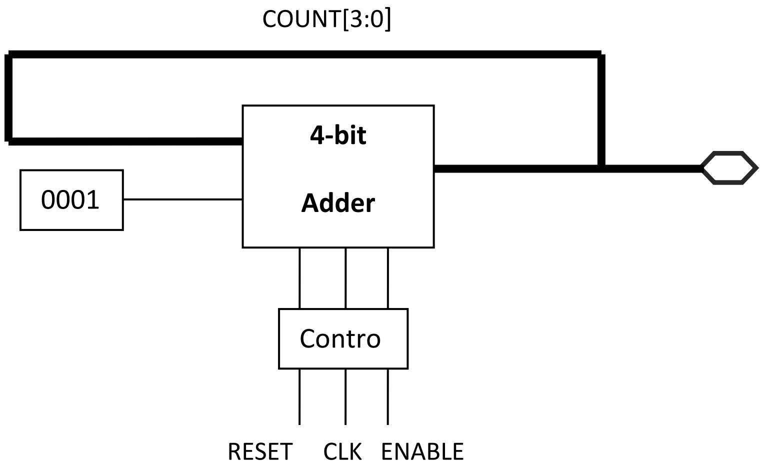


Figure 6: Block Diagram of a 4-bit Counter

```
-- file name: cnt_bhv.vhd
-- A four bit up-counter which counts up when ENABLE is 1.
-- RESET resets the counter to 0.
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package PKG_INC is
function INC(X :STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
end PKG_INC;

package body PKG_INC is
  function INC(X : STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
   variable XV: STD_LOGIC_VECTOR(X'LENGTH-1 downto 0);
  begin
   XV := X;
   for I in 0 to XV'HIGH loop
    if XV(I) = '0' then
      XV(I) := '1';
      exit;
    else XV(I) := '0';
    end if;
   end loop;
   return XV;
  end INC;
end PKG_INC;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.PKG_INC.all;
entity CNT_BHV is
  port(CLK,ENABLE,RESET: in STD_LOGIC;
       COUNT: inout STD_LOGIC_VECTOR(3 downto 0));
end CNT_BHV;
```

```
architecture BEHAVIOR of CNT_BHV is
  begin
  process(CLK,ENABLE,RESET)
    begin
      if RESET = '1' then
      COUNT <= "0000";
      elsif CLK'EVENT and CLK='1' then
      if ENABLE = '1' then
                      COUNT <= INC(COUNT);
      end if;
      end if;
  end process;
end BEHAVIOR;
```

Figure 7: VHDL Script Describing a 4-bit Counter

The synthesized design layout view is shown on Figure 8. The size of the circuit is 150 μm by 150 μm. I/O Pads are not added in the layout. I/O pads can be obtained from MOSIS and can be inserted manually as described in the tutorial available on the VTVT web site [15]. Once I/O pads are inserted, the counter is complete for fabrication. The steps for the submission of a chip design are also available on the VTVT tutorials site [16].

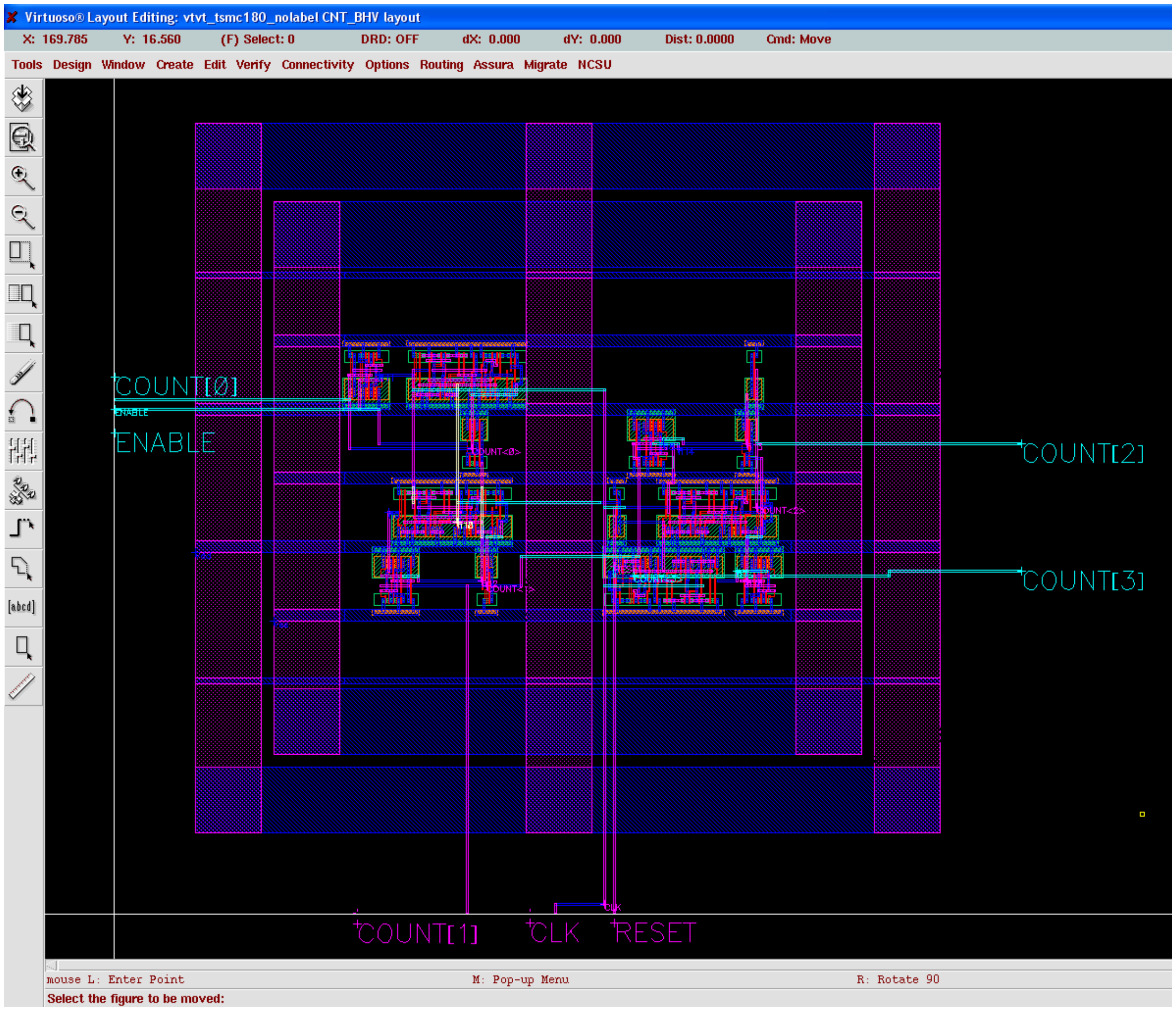

Virtuoso® Layout Editing: vtvt_tsmc180_nolabel CNT_BHV layout
X: 169.785 Y: 16.560 (F) Select: 0 DRD: OFF dX: 0.000 dY: 0.000 Dist: 0.0000 Cmd: Move
Tools Design Window Create Edit Verify Connectivity Options Routing Assura Migrate NCSU
COUNT[0]
ENABLE
COUNT[2]
COUNT[3]
COUNT[1]
CLK
RESET
mouse L: Enter Point
M: Pop-up Menu
R: Rotate 90
Select the figure to be moved:

Figure 8: Layout of a 4-bit Counter using VTVT 0.18-microns Standard Cell Library

4. Toward Automation of TSMC 0.18-micron Cell Library Development

Figure 9 summarizes the development flow used to generate VTVT_TSMC180 cell library. Scripts were developed and used in two stages of the development flow: the generation of the Synopsys Liberty format file (.lib file) and the quality assurance (QA) process. These scripts are written in Perl, Skill, Tcl or Shell, and were executed within multiple design environments and systems including Cadence Virtuoso, HSpice, Shell, Synopsys Design Compiler and Cadence SOC Encounter. In this chapter, the scripts created to automate and verify the standard cell library development are discussed.

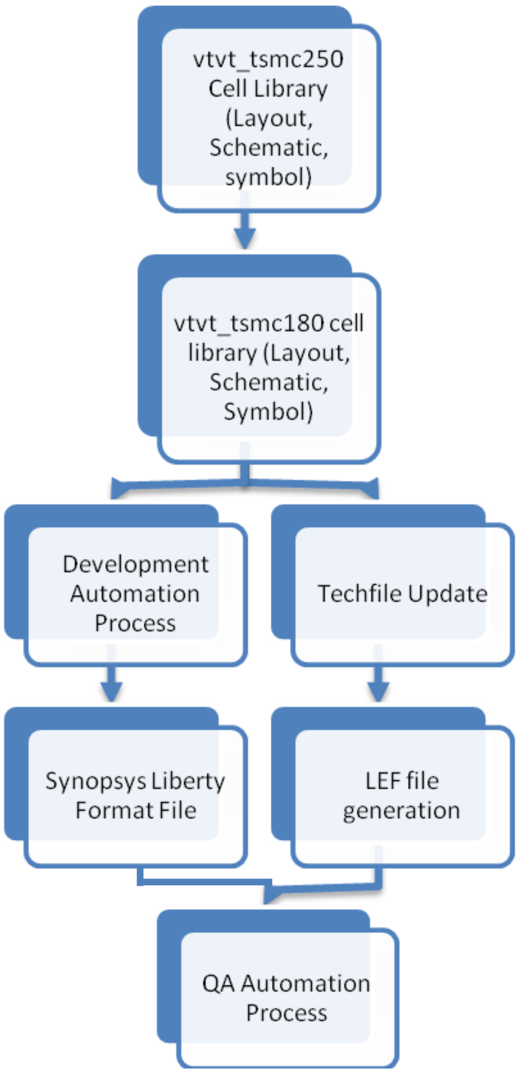

vtvt_tsmc250 Cell Library (Layout, Schematic, symbol)
vtvt_tsmc180 cell library (Layout, Schematic, Symbol)
Development Automation Process
Techfile Update
Synopsys Liberty Format File
LEF file generation
QA Automation Process

Figure 9: VTVT_TSMC180 Development Flow

4.1 From Library Cells to Synopsys Liberty Format File

A series of scripts was developed to generate the Synopsys Liberty Format file (.lib file) from the layout and schematic views of the VTVT_TSMC180 cells. The steps of this flow are detailed in this subsection and presented overall on Figure 10.

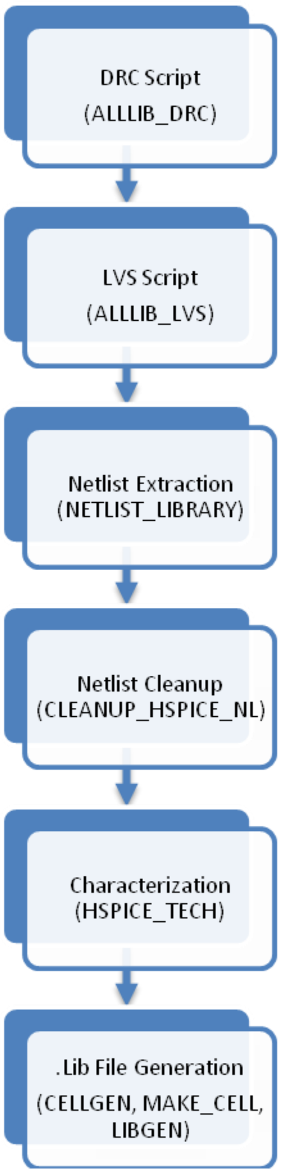


Figure 10: From Library cells toward Synopsys Liberty Format File

4.1.1 Design Rule Check (DRC) Automation

A skill script named alllib_drc was implemented to automatically run DRC for an entire cell library. The script verifies that each standard cell is DRC clean and generates a log file that contains the status of each cell checked. In the occurrence of cells failing DRC, the log file enables the user to easily identify and therefore modify appropriate cells prior to moving to the next design step. This step is successfully completed when the log file indicates that the library is DRC clean. The alllib_drc script and its documentation are attached in appendices I and J at the end of this thesis. It is used within the Command Interface Window (CIW) of the design kit by first loading the script file then invoking the command for the library of interest. A summary of this script is shown in Table 3.

Table 3: ALLLIB_DRC Script Summary for vtvt_tsmc180

Calling Environment	Command Interface Window (CIW) in the Cadence environment
Usage	➢ load "alllib_drc.il" ➢ alllib_drc("vtvt_tsmc180")
Output	Logfile "vtvt_tsmc180_drc.log"

4.1.2 Layout versus Schematic (LVS) Automation

The LVS skill script named alllib_lvs is a skill script that performs LVS check on each cell library based on the LVS rules included in the design kit. It also generates a log file which lists the LVS run status of each cell. When a cell fails LVS, the library fails the LVS script run. The developer must, therefore, check the logfile and modify failing cells prior to running the next script. The documentation and content of the alllib_lvs.il script are attached in appendices K and L. This script is loaded then invoked within the CIW in the NCSU_CDK. Table 4 summarizes the usage of this script.

Table 4: ALLLIB_LVS Script Summary for vtvt_tsmc180

Calling Environment	Command Interface Window (CIW) in the Cadence environment
Usage	➢ load "alllib_lvs.il" ➢ alllib_lvs("vtvt_tsmc180")
Output	Logfile "vtvt_tsmc180_lvs.log"

4.1.3 Netlist Extraction

The netlist extraction Skill script netlist_library.il extracts the netlists of all the layout cells of a given library. It is loaded and invoked within the CIW of the design kit. This script should be run once the cells are declared DRC and LVS clean. The netlists extracted are saved in directory vtvt_tsmc180netlist, as cell_name.hspice_nl files, where cell_name is the name of each cell within the library. A log file vtvt_tsmc180_netlist.log is also generated and includes the status of each cell's extraction. Appendices M and N contain the netlist_library.il description and script.

Table 5: NETLIST_LIBRARY Script Summary for vtvt_tsmc180

Calling Environment	Command Interface Window (CIW) in the Cadence environment
Usage	➢ load "netlist_library.il" ➢ netlist_library("vtvt_tsmc180")
Output	Logfile "vtvt_tsmc180_netlist.log" Directory "vtvt_tsmc180_netlist" Files "vtvt_tsmc180_netlist/<cellname>.hspice_nl

4.1.4 Netlist Cleanup

The extracted netlists from the previous stage will be included in an upper level netlist file named cellname.l, where cellname is the name of the corresponding library cell. The cellname.l file located in a directory stdlib_netlist, contains subcircuit netlists that will be instantiated within the circuit description files for characterization purposes through HSpice simulations. Figure 11 illustrates an inclusion of inv_1.hspice_nl within its corresponding upper level netlist file inv_1.l, located in directory "stdlib_netlists". For documentation purposes, the output directory from the netlist_library script, vtvt_tsmc180_netlist is renamed as hspice_extracted_netlist.

```
.subckt inv_1 vdd ip op
.inc ../hspice_extracted_netlist/inv_1.hspice_nl
.ends inv_1

.subckt nonloadinv_1 dd ip op
ein in gnd ip gnd 1
xinv_1 dd in op inv_1
.ends nonloadinv_1

.subckt nonload_noninv_1 dd ip op
x1 dd ip in1 nonloadinv_1
x2 dd in1 op nonloadinv_1
.ends nonload_noninv_1
```

Figure 11: stlib_netlists/inv_1.l file

The extracted netlists generated by the netlist_library script had to be modified, hence the need for the cleanup_hspice_nl Perl script. Among other changes, the script replaces the 0 node by GND for the ground node and removes any temperature and probe information. It generates new files cellname.hspice_nl files. Its description is attached in appendix O. The modified directory is named hspice_extracted_netlists. It contains all the cellname.hspice_nl netlist files for each cell. Table 6 shows a summary of this command.

Table 6: CLEANUP_HSPICE_NL Script Summary for vtvt_tsmc180

Calling Environment	Command Line from Unix/Linux system
Usage or Command	➢ cleanup_hspice_nl hspice_extracted_netlists
Output	Modified Files “hspice_extracted_netlist/<cellname>.hspice_nl

4.1.5 Characterization Automation

The characterization process consists of running HSpice on multiple circuit configurations for each library cell with the intention to capture specific cell characteristics such as: pin capacitance, delays, power dissipation. It requires a directory structure described on Figure 12 shows organization of the HSpice simulation files based for each cell. Originally, all cells and their simulation results were stored in a single directory vtvt_tsmc180_characterization. A script named stdlib_org was executed to organize all cells in the directory structure shown below. The hspice_extracted_netlists directory contains all cells netlists. Directory stdlib_netlists contains all subcircuits for all cells as well as some loading information for the entire simulation circuitry. Each cell has a simulation directory which contains spice simulation files (*.sp files) and simulation results files (*.results files).

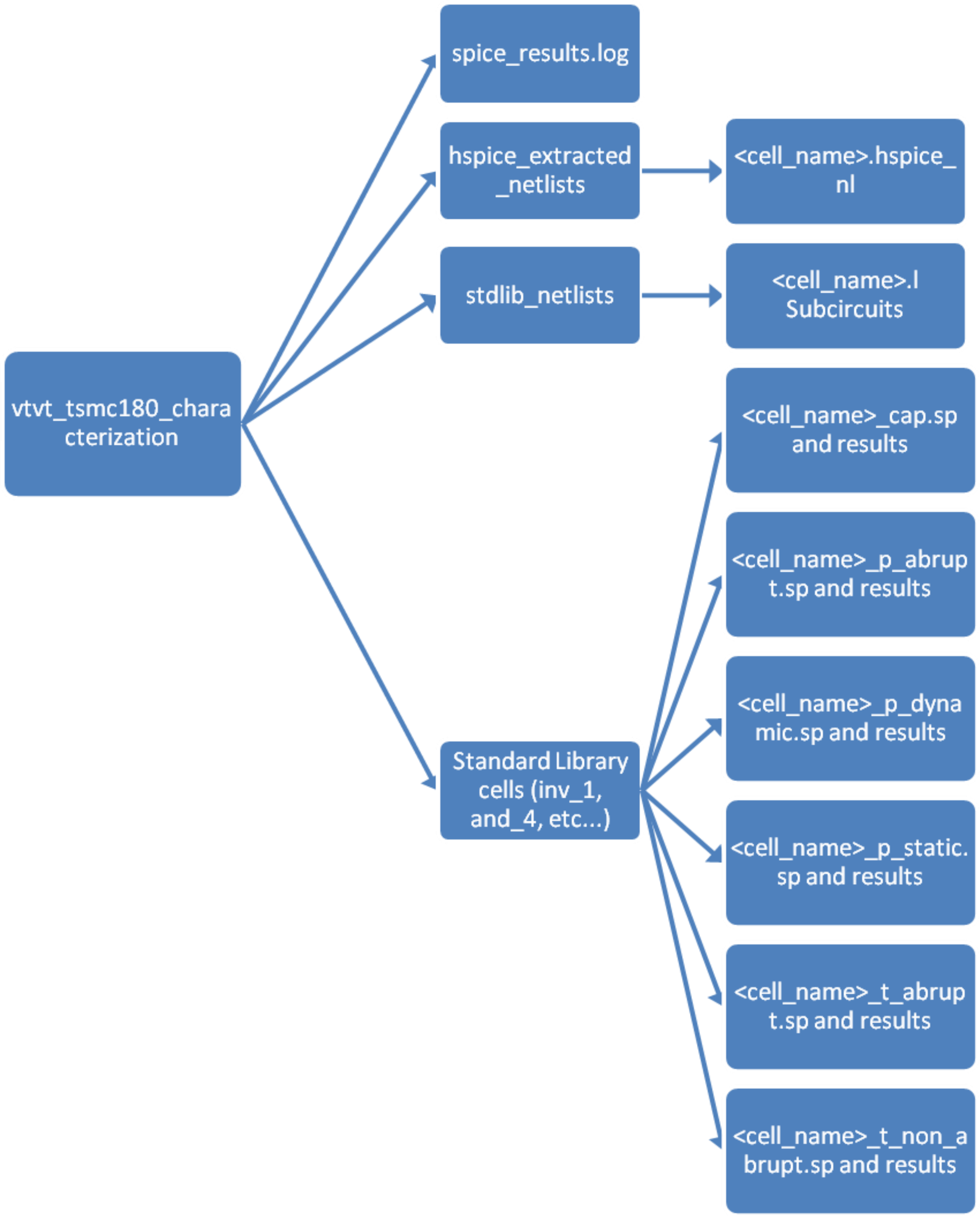

vtvt_tsmc180_chara cterization
spice_results.log
hspice_extracted _netlists
<cell_name>.hspice_ nl
stdlib_netlists
<cell_name>.l Subcircuits
Standard Library cells (inv_1, and_4, etc...)
<cell_name>_cap.sp and results
<cell_name>_p_abrup t.sp and results
<cell_name>_p_dyna mic.sp and results
<cell_name>_p_static. sp and results
<cell_name>_t_abrup t.sp and results
<cell_name>_t_non_a brupt.sp and results

Figure 12: Directory Structure generated by STDLIB_ORG Script for Cell Characterization

A summary of stdlib_org command is shown on Table 7. Its documentation is available on Appendices P and Q.

Table 7: STDLIB_ORG Script Summary for vtvt_tsmc180

Calling Environment	Command Line from Unix/Linux system
Usage or Command	➢ stdlib_org vtvt_tsmc180_characterization
Output	None

A Perl script named hspice_tech was developed to automatically run HSpice on the entire library. The hspice_tech script was invoked on the vtvt_tsmc180_characterization directory and the spice_results.log file was generated and contains all run information for each cell. The log file indicates all successful and failing HSpice simulations. Details on the characterization script are attached in appendices R and S. The process of invoking the hspice_tech script is shown below (Table 8).

Table 8: HSPICE_TECH Script Summary for vtvt_tsmc180

Calling Environment	HSpice Environment
Usage or Command	➢ hspice_tech vtvt_tsmc180_characterization
Output	spice_results.log

4.1.6 Library Format File Generation

Once the cells have been characterized, the cellgen and libgen Perl scripts are used to generate the Synopsys Liberty format. These scripts have already been developed by a former team of students working on vtvt_tsmc250 standard cell library. The cellgen script reads all the cells simulation results files from characterization and generates a corresponding cell file for each library cell. Since the directory structure has been modified, another script named make_cell was implemented as a wrapper to the cellgen script. It runs cellgen on all the cells characterization results and moves all cell files to a cellgen_cells directory. Information on the make_cell script is available in appendix T and its command summary on Table 9.

Table 9: MAKE_CELL Script Summary for vtvt_tsmc180

Calling Environment	Command Line from Unix/Linux system
Usage or Command	➢ make_cell
Output	cellgen_cells directory

The libgen script is then invoked on the cellgen_cells directory. It combines all cell files into a single file and generates the Synopsys Liberty format file for library vtvt_tsmc180. The process of automation for the development process of the cell library from the library cells to the Synopsys Liberty Format file is completed.

4.2 Quality Assurance of Standard Cell Library

The QA scripts are part of a verification flow that encompasses the entire ASIC design flow mentioned in Chapter 2. Figure 13 shows the automation flow for quality assurance of the entire cell library. The Design Synthesis QA phase consists of running a script that automatically performs design synthesis on multiple VHDL scripts, which are implemented based on a gold model. The second phase, Place-and-Route QA, is achieved by running place-and-route on the synthesized designs generated by the first phase. Finally, in the third step, namely physical verification QA all the placed-and-routed designs are verified by running Design Rule Check (DRC) and Layout Versus Extraction (LVS). Quality assurance for design synthesis, P&R and Physical Verification are described below.

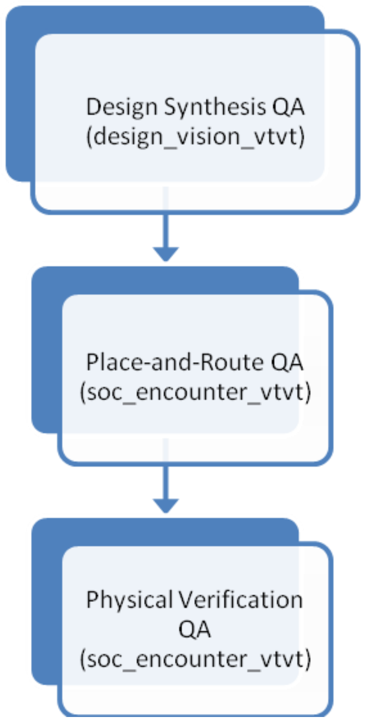


Figure 13: Quality Assurance Automation Flow

4.2.1 Quality Assurance within Design Synthesis

A generic testcase was implemented for each cell of the standard cell library. The golden model for the testcase shown in Figure 14 is described in VHDL for each file and stored in a directory named “vhdlFiles”.

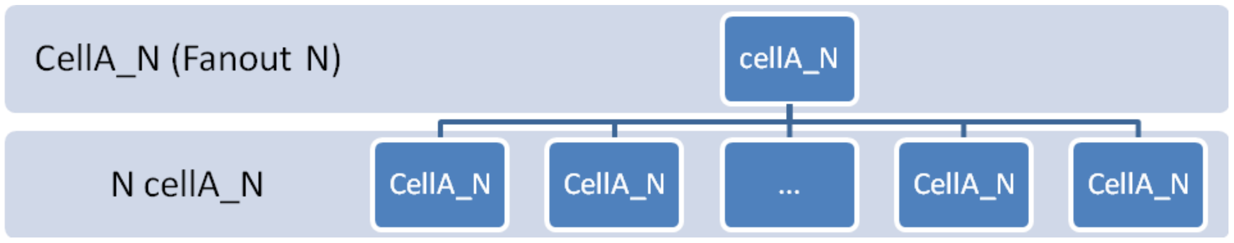


Figure 14: Golden Model for Generic Testcase

The design_vision_vtvt script, which is based on the Design Vision tutorial available on the VTVT website, is a Tcl script developed to run design synthesis using the generic testcase. The script ensures that each library cells can be properly instantiated by Synopsys Design Synthesis tools. Once all VHDL scripts are successfully synthesized, a Verilog file is generated for each cell, containing the synthesized netlist. A summary of this command is shown on Table 10. The description and script for design_vision_vtvt is attached on Appendix U.

Table 10: DESIGN_VISION_VTVT Script Summary for vtvt_tsmc180

Calling Environment	Synopsys Design Compiler (dc_shell_xg_t)
Usage or Command	➢ source design_vision_vtvt.tcl
Output	- vFiles directory containing all Verilog synthesized netlists - vtvt_tsmc180_design_vision_vtvt.log

4.2.2 Quality Assurance from Place-and-Route to Physical Verification

The synthesized Verilog netlists are imported into Cadence SOC Encounter for place-and-route. The Perl script soc_encounter_vtvt executes place-and-route on the synthesized Verilog netlists, imports the place-and-routed layout design into Cadence Virtuoso Layout Editor, imports the Verilog netlists into Cadence Composer Schematic Editor, and performs DRC and LVS on the imported designs. A library is considered to pass our quality assurance process when the soc_encounter_vtvt script generates no errors for any library cell. This script tries to ensure that the cells can be used by any user following the VTVT tutorials for an ASIC design that includes the previously mentioned steps. See references [10], [11], [12], [13] and [14]

More documentation on the soc_encounter_vtvt script is available on Appendix V. Its command summary table is shown below.

Table 11: SOC_ENCOUNTER_VTVT Script Summary for vtvt_tsmc180

Calling Environment	Cadence Environment
Usage or Command	➢ ./soc_encounter_vtvt.pl vFiles
Output	vtvt_tsmc180_soc_encounter_vtvt.log

4.3 Summary

Commercial cell libraries include hundreds of standard cells. Therefore, manually developing each cell not only impedes the performance, but also degrades the quality assurance of the design. Multiple scripts described in this chapter will reduce the development time for the VTVT group to expand its current cell libraries.

5. Conclusion

In this thesis, we presented a procedure for the development of VTVT 0.18-micron CMOS standard cell library and our efforts to automate the procedure. The 0.18-micron cell library is based on the existing VTVT cell library for TSMC 0.25-micron and was released on November 1, 2007.

The development flow of VTVT 0.18-micron library was initiated with the implementation of schematic and layout cells. The cell views were implemented by scaling down the current 0.25-micron library cells. Then physical verification procedures (Design Rule Check, and Layout versus Schematic) were executed on each cell of the library prior to netlist extractions of each cell's layout view via automation. The extracted netlists were simulated through HSpice for the characterization of the cells. The results from simulations were compiled into a Synopsys Liberty format (.lib) file to generate VHDL models of the cells. Information from the layout views and library technology file were combined to generate abstract views of each cell and a Layout Extraction Format (LEF) file for use in place-and-route tools. A symbol library was also generated to ease the user interface with the cells in a graphic environment for design synthesis or schematic capturing of an integrated circuit design. The tools used to generate the VTVT 0.18-micron library are up-to-date commercial tools to provide real world experience to teams of undergraduate and graduate students working on this project. The library developed supports features such as: Design Synthesis through Synopsys Design Compiler and Design Vision; Place-and-Route through Cadence SOC Encounter and Logic Simulation through Synopsys VCS-MX simulation tool. Additionally, layout and schematic views of each cell are available.

The original purpose for the design of cell library was to allow academia to improve research in VLSI provided free-of-access and diverse standard cell libraries based on current technologies. Our cell libraries contain 83 cells comprised of combinational and sequential cells with different drive strengths. In the design flow adopted for our standard cell library, the design entry is a VHDL or Verilog behavioral description. Using Synopsys synthesis and simulation tools, the behavioral description of the design is simulated and then synthesized into a gate-level circuit. The synthesized design is extracted in the Verilog format and imported into the Cadence physical design environment. After place-and-route of the design, the synthesized layout is compared against its schematic and ready for chip submission. A 4-bit counter is presented in this thesis as a verification of our design flow for VTVT 0.18-microns library.

The efforts to automate the development procedure of standard cell libraries have also been presented in this thesis. They consist of executing Skill, Tcl, Perl and Shell scripts on each cell. The following steps in the design flow have been automated and presented in this thesis: physical verification, netlist extraction, cell characterization, and generation of Synopsys Liberty Format file. The procedure to a complete automation of the development of standard cell libraries would consist of automatically generating layout and schematic views for each cell. However, this thesis did not address this shortcoming because of the problem of tool compatibility faced at Virginia Tech. Future research options from this work would consist of automatically generating layout and schematic cell views for each library cell.

In addition to automating the standard library development, we also proposed a method for automation of quality assurance of our libraries. The quality assurance scripts, namely design_vision_vtvt and soc_encounter_vtvt, execute the entire design flow proposed by VTVT for each cell of the library. The script design_vision_vtvt performs design synthesis on each cell based on multiple VHDL scripts which guarantee that each cell is instantiated in design synthesis. The script design_vision_vtvt then outputs multiple Verilog netlists, specific to each VHDL scripts synthesized. The script soc_encounter_vtvt uses each synthesized Verilog netlists to execute Place-and-Route. The script soc_encounter_vtvt saves the placed-and-routed design into a GDSII file that is imported into Cadence Virtuoso for physical verification. Prior to releasing a library, we ensured that the quality assurance scripts are executed and successful for all cells in a library. This automated quality assurance flow has been executed on both VTVT 0.25 and 0.18 microns libraries.

In summary, we presented the development of TSMC 0.18 μm standard CMOS cell library distributed by the Virginia Tech VLSI for Telecommunications (VTVT) Group and the procedure toward its automation. We also proposed an automation process for its quality assurance. VTVT’s standard cell libraries can be acquired via our website at http://www.vtvt.org. The VTVT Lab website also provides intensive up-to-date tutorials on the entire design flow process for a complete ASIC design based on our cell libraries. We believe the academic impact of this thesis is significant, because it aims to successfully provide open source standard cell libraries for research and teaching within a shorter design time and for various CMOS technologies.

A. Appendix: VTVT TSMC 0.18um Standard Cell Library Documentation (Release 1- November 1, 2007)

The VTVT (Virginia Tech VLSI for Telecommunication) group has developed a standard cell library kit based on the TSMC 0.18 μm technology. In this first release, the layout, symbols, and schematic libraries were developed and re-compiled using a modified version of the NCSU kit version 1.5.1. The layouts were developed using the MOSIS DEEP design rules, namely SCN6M_DEEP. The kit, as distributed here, includes:

- Layouts, symbols and schematic in Cadence dfII format
- Synopsys Synthesis (.db and .sdb) and VHDL simulation libraries, as well as the source code (.lib and .slib) for the synthesis libraries.
- LEF files for the PNR tools

  The layouts include only simple cells (2- and 3-input combinational cells, tristate buffers, flip-flops, and latches).

- Symbols and Schematic libraries for improved user-interface during synthesis and import from synthesis tool to Cadence.
- Design supported by up-to-date testing and simulation tools:
    - Cadence SOC Encounter
    - Synopsys Design Vision
    - Synopsys Design Compiler
    - Synopsys VCS MX

  There are some other things the VTVT lab plans to provide, but as of now (11/1/2007) has not yet provided, such as:

- Arithmetic macro cells
- TLF files, timing-driven placement
- Support for TSMC 0.13 μm technology

Cells Contained in the Library

The cell library vtvt_tsmc180 contains the cells listed in Table 12.

Table 12: Cells Contained in the Library vtvt_tsmc180

Cell Name	Function
buf_[1,2,4]	Noninverting buffer, drive strength 1, 2, or 4
inv_[1,2,4]	Inverter, drive strength 1, 2 or 4
and2_[1,2,4]	2-input AND gate, drive strength 1, 2, or 4
and3_[1,2,4]	3-input AND gate, drive strength 1, 2, or 4
and4_[1,2,4]	4-input AND gate, drive strength 1, 2, or 4
or2_[1,2,4]	2-input OR gate, drive strength 1, 2, or 4
or3_[1,2,4]	3-input OR gate, drive strength 1, 2, or 4
or4_[1,2,4]	4-input OR gate, drive strength 1, 2, or 4
nand2_[1,2,4]	2-input NAND gate, drive strength 1, 2, or 4
nand3_[1,2,4]	3-input NAND gate, drive strength 1, 2, or 4
nand4_[1,2,4]	4-input NAND gate, drive strength 1, 2, or 4
nor2_[1,2,4]	2-input NOR gate, drive strength 1, 2, or 4
nor3_[1,2,4]	3-input NOR gate, drive strength 1, 2, or 4
nor4_[1,2,4]	4-input NOR gate, drive strength 1, 2, or 4
xor2_[1,2]	2-input XOR gate, drive strength 1 or 2
xnor2_[1,2]	2-input XNOR gate, drive strength 1 or 2
mux2_[1,2,4]	2-to-1 multiplexer, drive strength 1, 2, or 4
mux3_2	3-to-1 multiplexer, drive strength 2

mux4_2	4-to-1 multiplexer, drive strength 2
ABnorC	(ip1*ip2+ip3)’, drive strength 1
ABorC	ip1*ip2+ip3, drive strength 1
ab_or_c_or_d	ip1*ip2+ip3+ip4, drive strength 1
Not_ab_or_c_or_d	(ip1*ip2+ip3+ip4)’, drive strength 1
fulladder	One-bit ripple-carry adder, drive strength 1
bufzp_2	noninverting tristate buffer, low-enabled, drive strength 2
invzp_[1,2,4]	inverting tristate buffer, low-enabled, drive strength 1, 2, or 4
cd_8	clock driver, drive strength 8
cd_12	clock driver, drive strength 12
cd_16	clock driver, drive strength 16
lp_[1,2]	high-active D latch, drive strength 1 or 2
Lrp_[1, 2, 4]	high-active D latch with asynchronous low-active reset and drive strength 1, 2, or 4
Lrsp_[1, 2, 4]	high-active D latch with asynchronous low-active reset and asynchronous high-active set, drive strength 1, 2, or 4
Dp_[1,2,4]	rising-edge triggered D flip-flop (with 1, 2, or 4 drive strength)
Drp_[1,2,4]	rising-edge triggered D flip-flop with asynchronous low-active reset (1, 2, or 4 drive strength)
drsp_[1,2,4]	rising-edge triggered D flip-flop with asynchronous low-active reset and asynchronous high-active set
dksp_1	rising-edge triggered D flip-flop with asynchronous active high set and extra inverted output.
dtsp_1	rising-edge triggered D flip-flop with asynchronous active high set input and serial scan input.

dtrsp_2	rising-edge triggered D flip-flop with asynchronous low-active reset, asynchronous high-active set, and serial scan input
jkrp_2	rising-edge triggered JK flip-flop with asynchronous active-low reset and extra inverted output, drive strength 2.
Filler	filler cell (empty cell with power and ground rails and nwell)

Downloading the Cell Library

After you have downloaded the kit from the web and un-tarred it, you will have the following files listed in Table 13.

It is recommended that before the layout, symbols and schematic views (in Cadence_Libraries.tar) are used, a SCMOS_DEEP-based technology library for the TSMC 0.18 μm technology, named TSMC_CMOS018_DEEP, be created first. One way is by following the installation procedures on http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_unix_env.php#install.

Another way is to follow the instructions in the README file for the Cadence_Libraries directory of this release package.

Table 13: Contents of the Cell Library Tar file vtvt_tsmc180_release.tar.gz

File	***Contents***	**Containing its own README?**
cdk_modifications.tar.gz	Modifications to NCSU kit; in tar format	Y
Synopsys_Libraries.tar.gz	Synopsys synthesis layout and symbol libraries (.db and .sdb) and VHDL simulation libraries; in tar format	Y
Cadence_Libraries.tar.gz	Standard cell layout, symbol and schematic in dfII and GDSII formats; in tar format	Y
vtvt_tsmc180_lef.tar.gz	LEF files, GDSII-to-dfII map for Cadence SOC Ensemble and Virtuoso; in tar format	Y
tutorial_files.tar	Tutorials' input files for design flow; in tar format	Y

Installing the Cell Library Kit

A possible sequence for installing the cell library kit is as follows:

1. Download the kit and untar it.
2. Untar cdk_modifications.tar and make modifications to your NCSU kit. If you intend to generate a new LEF file, then, re-compile the NCSU Kit, following directions from the NCSU site (see http://www.cadence.ncsu.edu/wiki/Techfiles), or use the technology file (.tf file) provided in the release package.
3. Untar Cadence_Libraries.tar and copy the contained libraries to the user's working directory for layout, symbols and schematic tools (e.g. ICFB)
4. Untar vtvt_tsmc180_lef.tar and copy the contents to the user's working directory for placement-and-routing tools (e.g. SOC Encounter™ from Cadence).

5. Untar Synopsys_Libraries.tar and place it in a directory accessible by the users for use with Synopsys® Synthesis (Design Compiler, and Design Vision) and Simulation tools (Scirocco, Virsim).

Overview of Parts

The standard cell library vtvt_tsmc180 is intended for use with Cadence SOC Encounter Placement-and-Routing (PNR) tool. The layouts were developed with Cadence Virtuoso custom layout tool (using ICFB). The MOSIS DEEP design rules were chosen since the I/O pads available from MOSIS (designed by Tanner Research – see http://www.mosis.org/Technical/Designsupport/pad-library-scmos.html) followed that particular set of rules. The symbols and schematic were developed with Cadence Composer schematic.

The Synopsys .db synthesis library was obtained by characterizing the layout using HSPICE. The Synopsys .sdb synthesis library was obtained by extracting the symbols library from Cadence to Synopsys Design Compiler using EDIF 2.0.0. The .synopsys_dc.setup and .synopsys_vss.setup files used by the VTVT group in creating the libraries are also included in the tar file.

Further Documentation

After you untar the tar files, some of them contain their own README files. They will provide further documentation for the specific directory. However, they do not contain guidelines or tutorials about the tools. For tutorials and guidelines to modify the standard cell library, and/or use the simulation and design tools, refer to our site at: http://www.vtvt.ece.vt.edu/vlsidesign/index.php.

B. Appendix: Tutorial on Scaling Down Layout Cells

The goal of this tutorial is to show you how to scale down a cell library from VTVT_TSMC250 to VTVT_TSMC180.

You must be familiar with creating a new library and a new cell through Cadence. If you need help running DRC, LVS, and all other instructions mentioned in this tutorial without any details, please refer to the online tutorials available on VTVT Lab's website at www.vtvt.org.

Creating a new library and Setting Layout Editing Options

- In the .cdsenv file in your home directory (create the file if it does not exist), add or modify the following lines to set your grid options
    - layout xSnapSpacing float 0.045
    - layout ySnapSpacing float 0.045
    - layout gridMultiple float 3.24
    - layout gridSpacing float 0.81
    - layout drawDottedGridOn boolean nil

- Start icfb
- Create a new library named vtvt_tsmc180_<your initials>. Refer to the online tutorial if you do not know how to create a cell or a library.
- Attach the new library vtvt_tsmc180_<your initials> to TSMC 0.18 technology provided by the NCSU_CDK, namely NCSU_TechLib_tsmc02d.
- Create a new cell in this library (the cell you intend to scale down)

Drawing a Cell Layout

Take the following details into account as you start working on the cells.

- Metal1 and Metal2 pins should be centered at the grid points.
- Use paths and not rectangles for poly, metal1 and metal2 routing

Guidelines for Drawing Standard Cells

Generate a template cell that will be used to create all other cells. The template should include the specific cell height, power supply height, cell width multiple, nwell height, and all other necessary specifications for the design of all the other cells. This will allow other users working on the same project to design cells with the same characteristics to be a part of a standard cell library. The rules applied to generate a template cell are listed in the checklist that is attached in the following appendix and shown on the figure below.

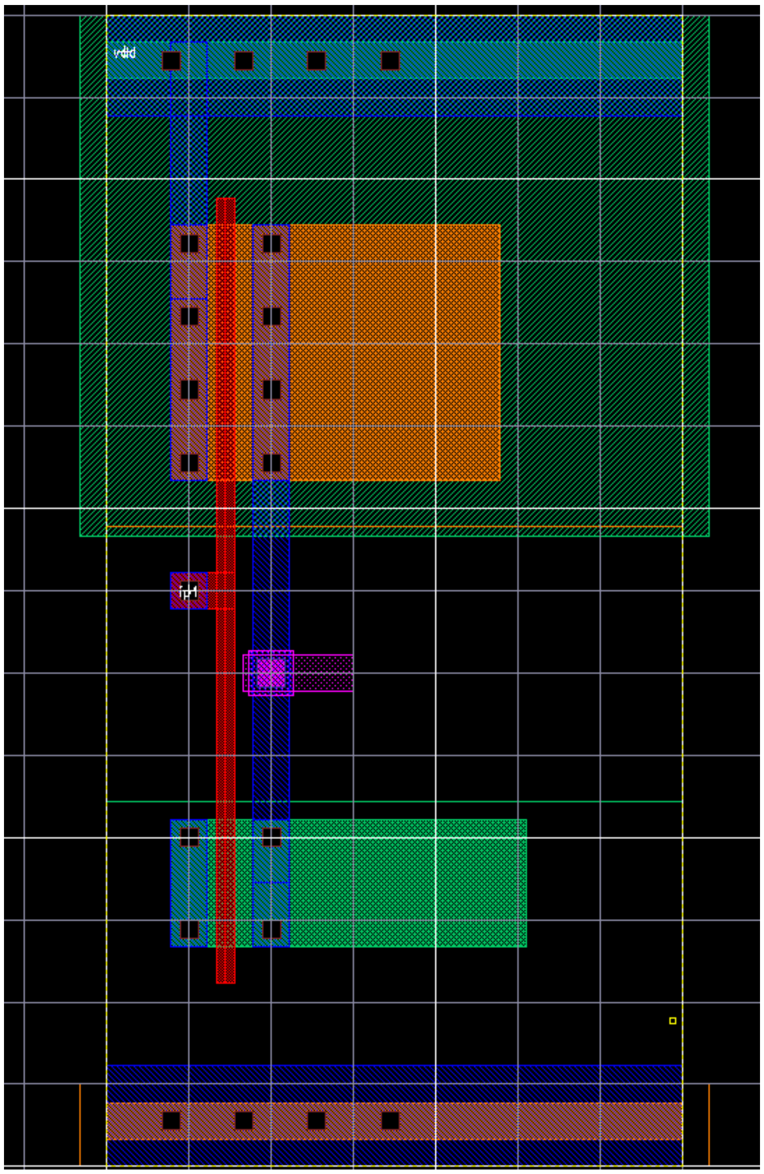


Figure 15: Overall Design rules – Template Cell

A few guidelines for implementing a TSMC 0.18-micron cell by scaling down its corresponding TSMC 0.25-micron cell are presented in this section. In addition to these restrictions, all layers such as poly, metal1 and metal2, must be scaled down by a factor of 0.09/0.12.

- The Height of all cells is fixed
- The width of cells varies but should be an integer multiple of the grid lines.
- Metal1 from power/ground rails, active regions in the power/ground rails, as well as the Select layers for n/p transistors should be aligned with the grid line. They determine the width of the cell.

Rules on Transistors

- The N-Select layer for n-transitors and the p-select layer for the p-transistor should start from the left edge at the y-axis: x=0.
- For different drive strengths, maintain the length from active to select edge to 0.54um.
- Increase size of n-transistors from the topmost edge. Increase size of p-transistor from the bottom edge.

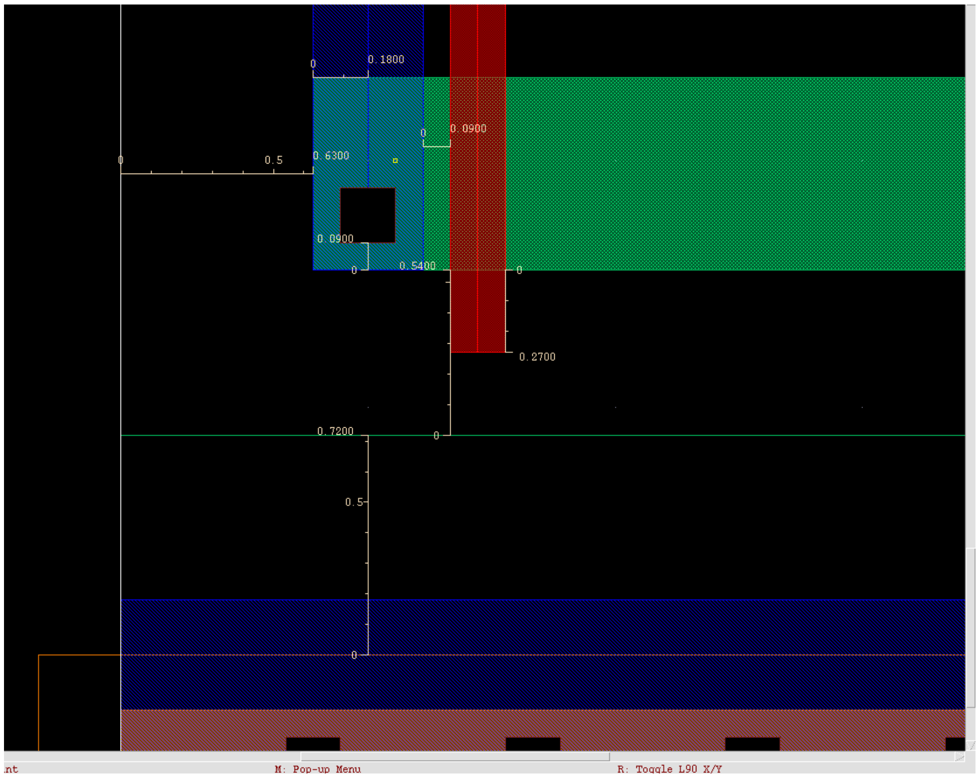


Figure 16: N-transistor measures

Pins, Poly Extensions and Supply Nets Rules

- The extension of the n-well and the p-select for the ground rail beyond the y-axis is 0.27um.
- Any heights and position of the following layers: nwell, pselect, nselect, power and ground rails must not be modified.
- Metal1 path is 0.36 wide with center at 0.18um.
- Poly path is 0.18um wide centered at 0.09um.
- Any metal should always be 9λ to center or 7λ to edge.
- Metal2 path is 0.36 wide with center at 0.18um.
- Always make sure the metal path layer is centered on grid line
- Run DRC on the cell often to make sure no errors are added.
- Pins must be centered at grid intersection point. Metal1 and metal2 routing paths should be centered on grid lines. Metal2 is always horizontal. Metal1 is vertical except for power and ground rails.
- Always place a connection or routing to another layer at grid intersections. Metal routing is always on grid lines.

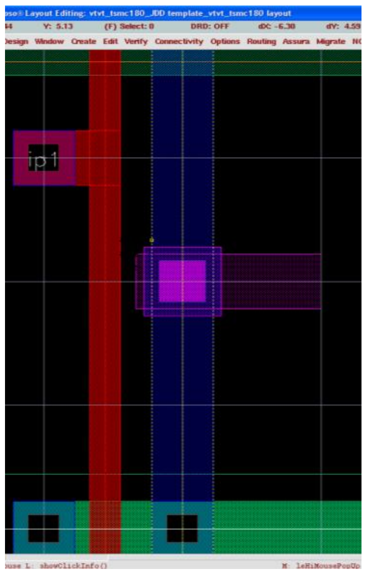

Layout Editing: vtvt_tsmc180_JDD template_vtvt_tsmc180 layout
Y: 5.13
(F) Select: 0
DRD: OFF
dX: -6.30
dY: 4.59
Design Window Create Edit Verify Connectivity Options Routing Assura Migrate
ip1
showClickInfo()
leHiMousePopUp

Figure 17: Metal2 pin connection

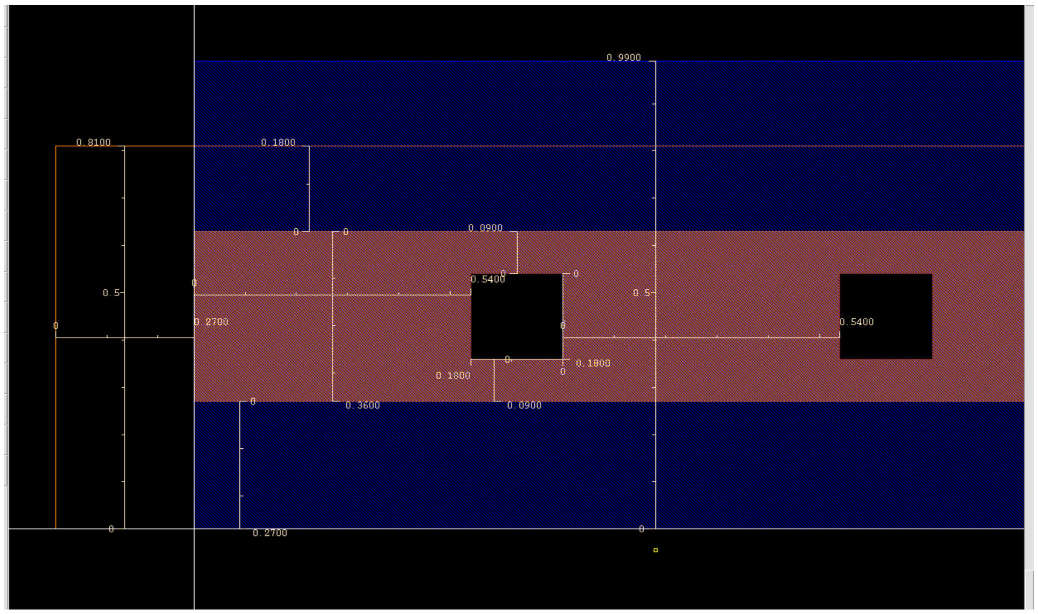


Figure 18: Ground rail information

- The minimum distance between metal1 path and poly path is 0.09um.
- Poly extends to exactly 0.27um from active region on p-transistor and n-transistors.
- The p-active region is always 0.54 um away from the p-select layer boundary.

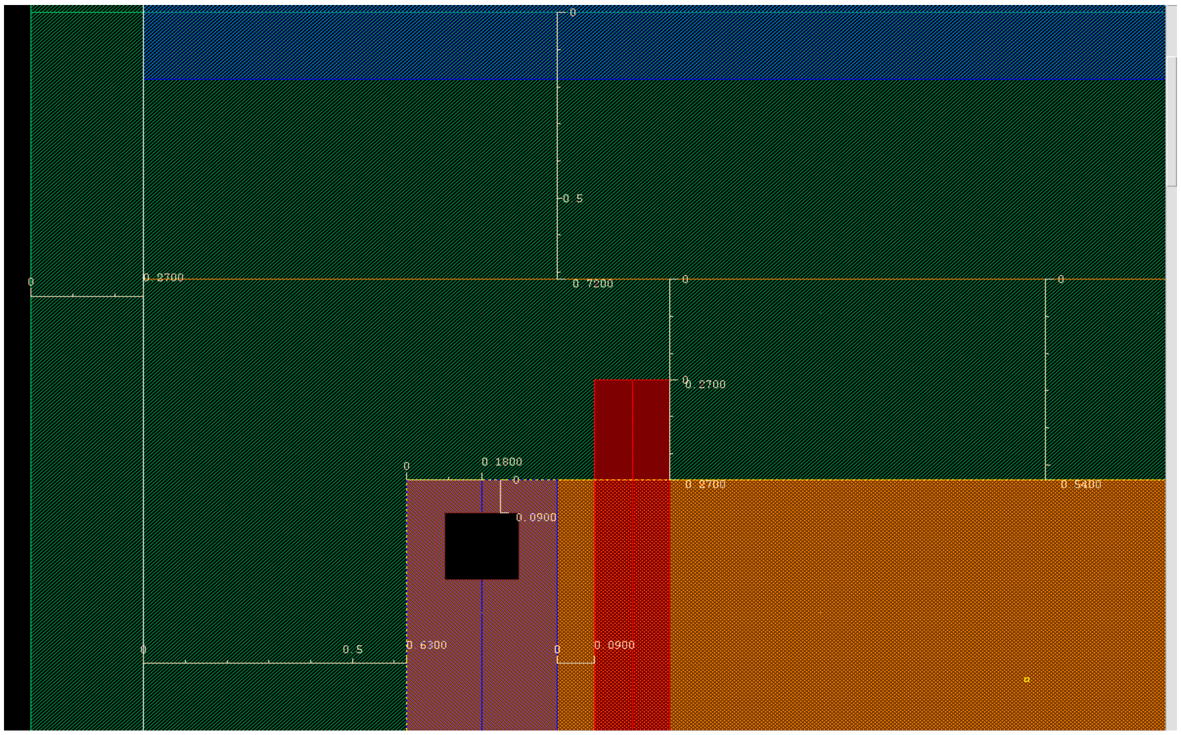
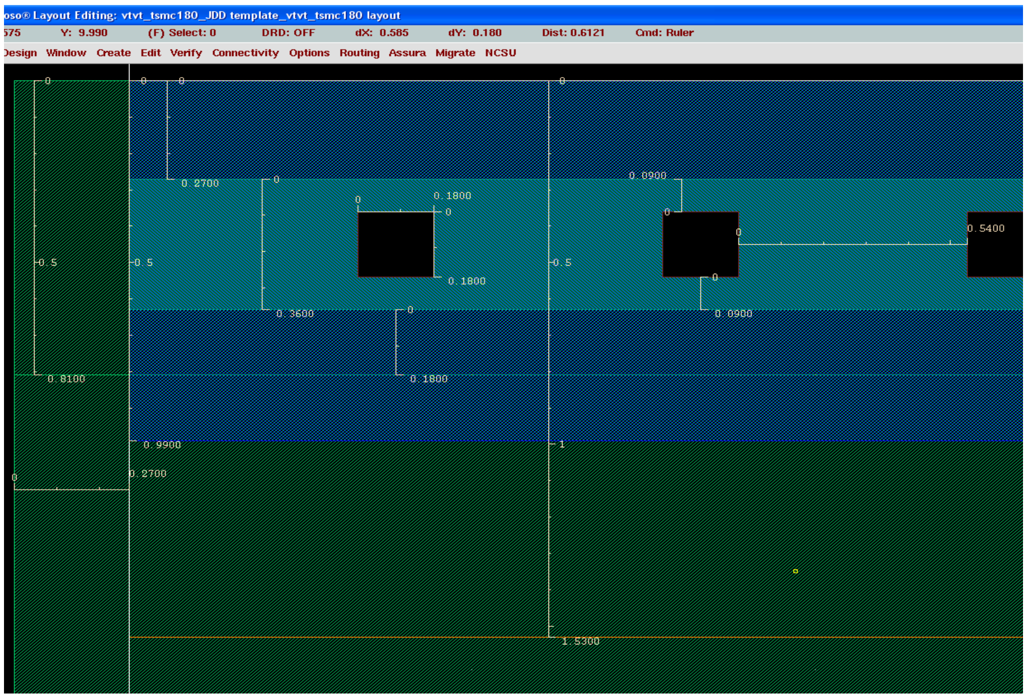


Figure 19: Poly Extension


oso® Layout Editing: vtvt_tsmc180_JDD template_vtvt_tsmc180 layout

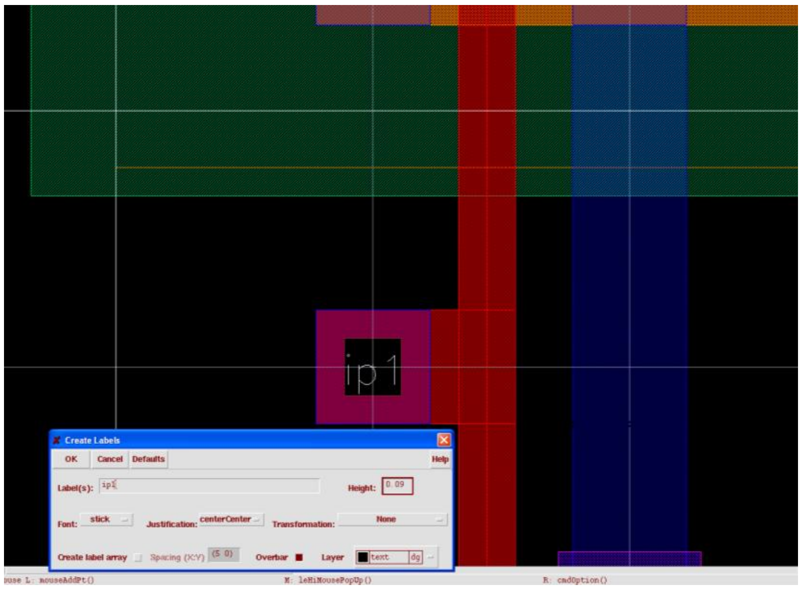
575 Y: 9.990 (F) Select: 0 DRD: OFF dX: 0.585 dY: 0.180 Dist: 0.6121 Cmd: Ruler

Figure 20: Power Rail dimensions and contact spacing

- The width of the metal1 layer is fixed to 0.99um.
- Spacing between each cc layer is 0.54um
- Cc layers are 0.18um by 0.18um
- N-active layer in power rail is 0.36um wide with left edge on y-axis.

Pin Naming Convention

- In the LSW window, select “metal1 dg” or “metal2 dg”
- Type ‘l’ for label to insert a label. In the label form, enter the text for the pin name and set the height of the text to 0.09
- Place the label within the boundary of the cell. Do not place vdd and vss at the boundary exactly, but inside.
- No labels should cross the cell’s height and width boundaries.



Label(s): ip1
Height: 0.09
Font: stick
Justification: centerCenter
Transformation: None
Create label array
Spacing (X:Y) (5 0)
Overbar
Layer text dg

```
ouse L: mouseAddPt()          M: leHiMousePopUp()          R: cadOption()
```

Figure 21: Adding a label to a pin

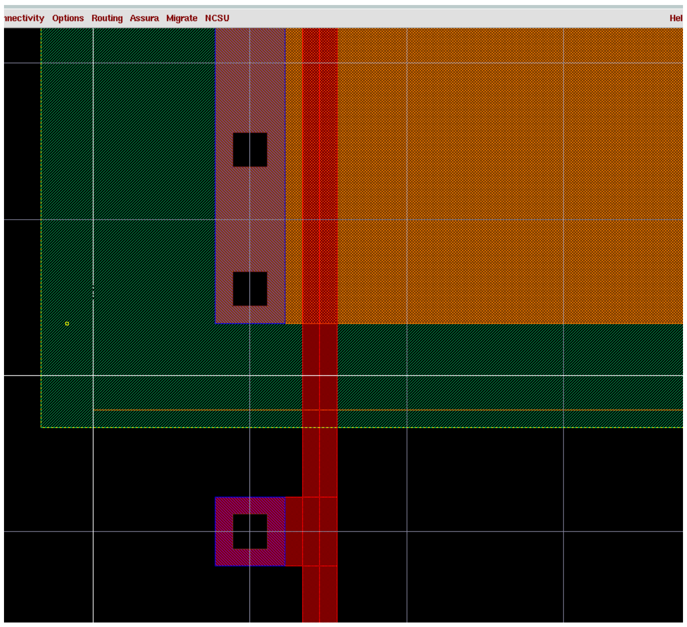


Figure 22: Pin must be centered at grid lines intersections

C. Appendix: Checklist for Layout Scale Down of Cells

This appendix contains a checklist of rules for a successful implementation of scaled down layout views from VTVT 0.25-micron library to VTVT 0.18-micron library. This checklist verifies that the layout cells are properly scaled down. Some of these rules are flexible. Their application would ease or fasten the place-and-route process of designs based on our cell library.

The grid line rules are established to avoid that pin connections in the place-and-route tool generate snap grid errors.

Table 14: Checklist for Standard Cell Layout Views Scale Down

	Checklist for Standard Cell Layouts **Use this checklist to make sure your layouts are correct.** **All these points must be checked in order to complete any layout cells.**	Completed (Y/N)
1	Editing Window Options set (Minor spacing 0.81, Major Spacing = 4.05, X snap spacing = 0.045, Ysnap spacing = 0.045	
2	Pass DRC	
3	Pass LVS	
4	All units are in microns	
5	Height of cell 11.34	
6	Bottom at y=0 line	
7	Origin At (0,0)	
8	Limit of cell (left = 0, right = vertical grid line, bottom = 0, top = 11.34	

9	Extension of nwell 0.27 (from left and right)	
10	Extension of pselect on vdd rail 0.27 (from left and right)	
11	Nwell layer height (bottom=6.21, top=11.34, left = -0.27, right = (vertical grid line + 0.27)	
12	N-select containing vdd rail (Bottom=10.53, Top=11.34, Left = -0.27, Right = (vertical grid line + 0.27)	
13	N-active in vdd rail (bottom=10.71, top=11.07, left=0, right= vertical grid line)	
14	Metal1 on vdd rail (Bottom=10.35, top = 11.34, left = 0, right = vertical grid line)	
15	Distance from metal1 (on vdd rail) to pactive of pmos (1.08)	
16	P-select containing pmos (Bottom=6.3, Top=9.72 left = 0, right = vertical grid line)	
17	P-active for pmos (bottom = varies, top = 9.27, left = 0.63 for first mos	
18	Distance from pactive (for pmos) to p-select (for pmos) is 0.45	
19	Extension of poly on pmos = 0.27 and nmos = 0.27	
20	Metal1 and metal2 distance from the edge of the cell to edge of metal= 0.63	
21	N-active for nmos (Bottom = 2.16, top = varies, left = 0.63 (for first mos))	
22	Poly width for transistors = 0.18	
23	N-select for nmos (bottom= 1.62, top =3.6, left = 0, right = vertical grid line	
24	Distance from n-select (for nmos) to bottom of nactive (for nmos) is 0.54	
25	Distance from metal1 (on vdd rail) to n-select of nmos (0.63)	

26	Metal1 on vss rail (bottom=0, top=0.99, left = 0, right = vertical grid line	
27	Pselect on vss rail (bottom = 0, top = 0.81, left = -0.27, right = vertical grid line+0.27)	
28	Pactive on vss rail (bottom = 0.27, top = 0.63, left = 0, right = vertical grid line	
29	Nactive for nmos (Bottom = 2.16, top = varies)	
30	First Contacts (cc) on vss and vdd rails (0.54 away from edge)	
31	Spacing between cc on vss and vdd rails (0.54)	
32	Omit last cc at edge of vss and vdd rails if distance from cc to edge less than 0.54	
33	Added all labels.	
34	Routing metal1 and metal2 (including contacts) are 0.63 away from edge of cell. (check contacts on mos)	
35	No alignment of pins on same horizontal grid line, or consecutive pins on horizontal grid lines should be separated by one vertical grid line.	
36	Metal1, metal2 width of paths: 0.36	
37	Metal2 lines must be horizontal (as much as possible)	
38	Metal 2 lines are centered on horizontal grid lines. (as much as possible)	
39	Labels are in their metal layer + Additive pin square	

D. Appendix: Tutorial on Scaling Down a Schematic Cell

Once the layout is complete and it passes DRC, you can create a schematic cell view of the scaled down layout. This tutorial provides directives for scaling down the VTVT TSMC 0.25-micron schematic cell into a valid schematic cell view for VTVT 0.18-micron.

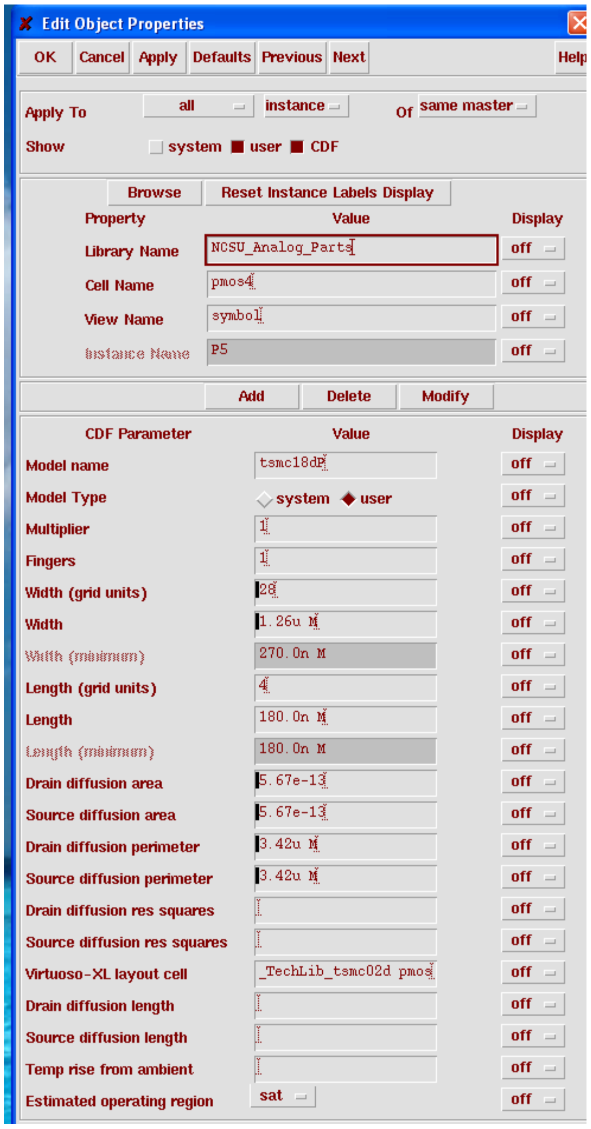
- Create a new schematic view for the cell to scale down in the VTVT 0.18 library.
- Open the schematic cell view from the TSMC 0.25 cell.
- Copy and paste the schematic to the VTVT 0.18 cell. Do not use "Copy" from the Library manager, but rather select and copy the schematic cell view across the screens.
- Measure the width of the Pmos (Nmos) transistors from the layout. It should be 1.26um (2.52um). The length of the transistors is 0.18um (The Poly width for this technology).

For a Pmos, follow the following steps.

a. Instantiate a Pmos (from NCSU_Analog_Parts, cell pmos4, view Name: symbol). We will call it "temporary cell" in this tutorial
b. In the properties of the the "temporary cell", the model is already tsmc18dP.
c. Edit the cell property of "temporary cell" as shown on the Figure below.
    - o Get the dimensions of the transistors from the VTVT 0.25 schematic and scale them by a factor of 0.09/0.12.
    - o Modify the "temporary cell" width and length based on the scaled down dimensions
    - o In the same edit form, set "Apply to" to "all instance of Same Master". This will replace all similar cells by the properties assigned to "temporary cell".
d. Repeat steps (a) through (c) for the Nmos.

At the end of the schematic update, all cells must have a model name: tsmc18dP or tsmc18dN. The temporary cell should then be deleted from the schematic view.

For verification purposes, run LVS to ensure that layout and schematic netlists match.



Edit Object Properties

OK | Cancel | Apply | Defaults | Previous | Next | Help

Apply To: all, instance, Of same master

Show: system, [✓ user], [✓ CDF]

Browse | Reset Instance Labels Display

Property	Value	Display
Library Name	NCSU_Analog_Parts	off
Cell Name	pmos4	off
View Name	symbol	off
Instance Name	P5	off

Add | Delete | Modify

CDF Parameter	Value	Display
Model name	tsmc18dP	off
Model Type	system [✓ user]	off
Multiplier	1	off
Fingers	1	off
Width (grid units)	28	off
Width	1.26u M	off
Width (minimum)	270.0n M	off
Length (grid units)	4	off
Length	180.0n M	off
Length (minimum)	180.0n M	off
Drain diffusion area	5.67e-13	off
Source diffusion area	5.67e-13	off
Drain diffusion perimeter	3.42u M	off
Source diffusion perimeter	3.42u M	off
Drain diffusion res squares		off
Source diffusion res squares		off
Virtuoso-XL layout cell	_TechLib_tsmc02d pmos	off
Drain diffusion length		off
Source diffusion length		off
Temp rise from ambient		off
Estimated operating region	sat	off

Figure 23: Edit Properties Form for Schematic Scale Down Process

E. Appendix: Generating Simulation Libraries

This appendix presents the procedure undertaken to generate Synopsys simulation libraries for VTVT 0.18-micron standard cell library. It is assumed that a Synopsys Liberty Format file, commonly called .lib file has already been generated prior to initiating the following procedures.

1. Verify that following statements are included in .synopsys_vss.setup

```
WORK > vtvt_tsmc180
vtvt_tsmc180 :  ./vtvt_tsmc180
```

2. Making simulation and synthesis files

2.1 Start dc_shell then read the library and get the db file and vhdl models

```
$ dc_shell-xg-t
> read_lib vtvt_tsmc180.lib
> write_lib vtvt_tsmc180 -o vtvt_tsmc180.db
> write_lib -f vhdl -o vtvt_tsmc180.vhd vtvt_tsmc180
> quit
```

2.2. Move db file to directory libs

```
mv vtvt_tsmc180.db libs/.
```

2.3 Update the *Vcomponents file

Open the *_Vcomponents file and replace string VCOMPONENTS by COMPONENT

2.4 Analyze (vhdlan) the resulting vhdl files for simulation purposes

```
>mkdir vtvt_tsmc180
```

Analyze the files in this order:

```
>vhdlan -w vtvt_tsmc180 *_Vtables.vhd *_Vcomponents.vhd *_VITAL.vhd
```

2.5 Generate Standard cells Components packages.(COMPONENTS.syn file)

Start dc_shell

```
$ dc_shell-xg-t

analyze -library vtvt_tsmc180 -format vhdl {vtvt_tsmc180_Vcomponents.vhd}

quit
```

There should be no errors.

<u>3. Setup for Synthesis and Simulation</u>

3.1 When vtvt_tsmc180 directory is used as library for synthesis and simulation

update .synopsys_vss.setup as follow:

```
WORK          > DEFAULT

DEFAULT        : work

vtvt_tsmc180    : ./vtvt_tsmc180
```

3.2 Add libs directory to search path

```
set search_path {. ./libs /software/Synopsys/Y-2006.06-SP2/libraries/syn}
```

3.3 Include technology libraries

```
set link_library {"vtvt_tsmc180.db"}

set target_library {"vtvt_tsmc180.db"}
```

F. Appendix: Generating a LEF file

This appendix describes the procedure used to generate the Layout Extraction Format (LEF) file for VTVT 0.25-micron library. Similar steps can be applied to generate the LEF file for VTVT 0.18-micron library. To generate a LEF file, you first need to modify the technology file for the Cadence library vtvt_tsmc250. Initially, the vtvt_tsmc250 library is attached to NCSU_TechLib_tsmc03d.

A LEF file combines layout and technology information for each cell in an ASCII format for place-and-route tools. The NCSU CDK used to develop our standard cell libraries does not support Place-and-Route. Therefore, the technology information in the kit had to be modified to support Place-and-Route. A complete Technology file (.tf file) supporting place-and-route is available in the release package of either VTVT_TSMC250 or VTVT_TSMC180 libraries.

In this document, we first present the changes required for the technology file, then provide the steps to generate a LEF file based on Cadence Abstract Generator.

Creating a TF file Supporting Place-and-Route

Use the file vtvt_tsmc250.tf given in the package. It contains the following changes. The file should contain:

- Control section: tsmc03d.tf
- layerDefinition function: Copy simply from the tsmc03d.tf LayerDefinition section
- LayerRules section: layerRules.tf . Remove viaLayer ca information.
- PhysicalRules section: physicalRules.tf:
- Devices section: devices.tf
- Lx and dlr Rules section: physicalDesingApp.tf
- prRules section: prRules.tf
- electricalRules: Added electrical rules per http://www.mosis.org/cgi-bin/cgiwrap/umosis/swp/params/tsmc-025/n94s-params.txt values for sheetRes.

The prRules.tf has been modified to support PNR. A sample of

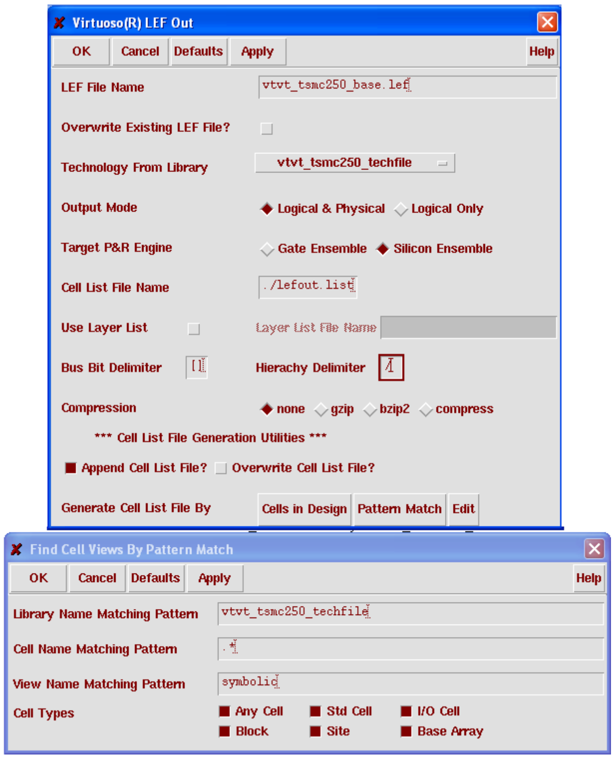
Creating a LEF file

Now that you have the new and valid vtvt_tsmc250.tf file, create a new library in Cadence in Technology File Manager->New…

- The new lib is vtvt_tsmc250_techfile

- Attach the vtvt_tsmc250 and pad library to vtvt_tsmc250_techfile
    - CIW->Technology File Manager->Attach
    - Design Library: vtvt_tsmc250 or pad library
    - Technology Library: vtvt_tsmc250_techfile

- Extract LEF base information from new techfile: From the modified technology file, extract the LEF base information by doing in CIW File->Export->LEF…as follow:



LEF File Name	vtvt_tsmc250_base.lef
Overwrite Existing LEF File?	☐
Technology From Library	vtvt_tsmc250_techfile
Output Mode	[✓ Logical & Physical] Logical Only
Target P&R Engine	Gate Ensemble [✓ Silicon Ensemble]
Cell List File Name	./lefout.list
Use Layer List	☐ Layer List File Name
Bus Bit Delimiter	[]
Hierachy Delimiter	/
Compression	[✓ none] gzip bzip2 compress

*** Cell List File Generation Utilities ***

■ Append Cell List File? ☐ Overwrite Cell List File?

Generate Cell List File By: Cells in Design | Pattern Match | Edit

Library Name Matching Pattern	vtvt_tsmc250_techfile
Cell Name Matching Pattern	.*
View Name Matching Pattern	symbolic
Cell Types	■ Any Cell ■ Std Cell ■ I/O Cell ■ Block ■ Site ■ Base Array

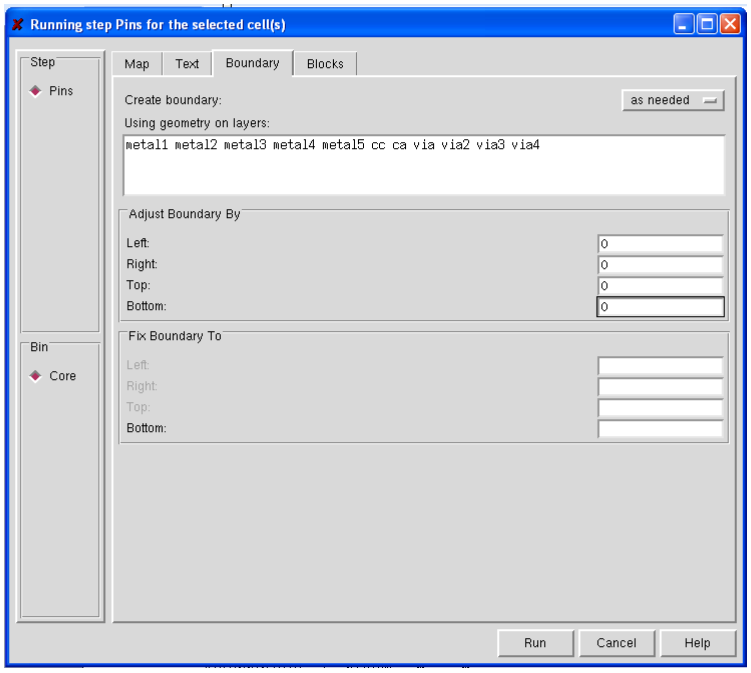
- The file vtvt_tsmc250_base.lef is generated. It contains the basic tech information and via_array information for P&R.

- Import the LEF file right back into library vtvt_tsmc250_techfile. You can dump the techfile to make sure that all your changes have been made (no more ca layers in the layer Rules section). The CIW window should say: Done reading file. Now you can use the vtvt_tsmc250 library in abstract.

Generating Abstract Views

Now, we want to get the macros of standard cells and I/O pads (if applicable).

- Start Abstract by typing "abstract" in your Cadence environment
- For the standard cells and/or I/O pads, do the following:
    - Go to File-> Library -> Open…
    - Select the library to work on
    - Abstract should not give you any errors. If you get errors, resolve them before moving to the next step, by modifying the TF file.
- **<u>In the Pin Section</u>**
    - Then, select all the cells and go to Flow->Pin
    - Fill up the form as follow:



Create boundary: as needed

Using geometry on layers:

```
metal1 metal2 metal3 metal4 metal5 cc ca via via2 via3 via4
```

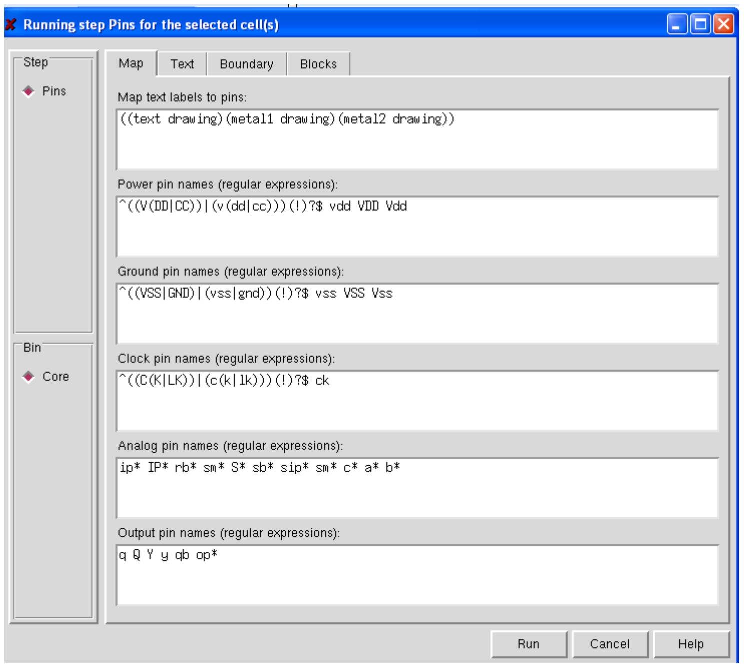
Adjust Boundary By

Left:	0
Right:	0
Top:	0
Bottom:	0

Fix Boundary To

Left:	
Right:	
Top:	
Bottom:	

    - In the Boundary Tab, no need to adjust the boundaries, so set it to 0. You may change it if you wish.
    - In the Map file, make your changes as follow:


Running step Pins for the selected cell(s)

Map text labels to pins:
```
((text drawing)(metal1 drawing)(metal2 drawing))
```
Power pin names (regular expressions):
```
^((V(DD|CC))|(v(dd|cc)))(!)?$ vdd VDD Vdd
```
Ground pin names (regular expressions):
```
^((VSS|GND)|(vss|gnd))(!)?$ vss VSS Vss
```
Clock pin names (regular expressions):
```
^((C(K|LK))|(c(k|lk)))(!)?$ ck
```
Analog pin names (regular expressions):
```
ip* IP* rb* sm* S* sb* sip* sm* c* a* b*
```
Output pin names (regular expressions):
```
q Q Y y qb op*
```

- These changes are as follow for the standard cells
  - Text layer: ((text drawing)(metal1 drawing)(metal2 drawing))
  - Power Pins: ^((V(DD|CC))|(v(dd|cc)))(!)?$ vdd* VDD* Vdd*
  - Ground Pins: ^((VSS|GND)|(vss|gnd))(!)?$ vss* VSS* Vss*
  - Clock Pins: ^((C(K|LK))|(c(k|lk)))(!)?$ ck*
  - Analog Pin: ip* IP* rb* sm* S* sb* sip* sm* c* a* b* j k
  - Output Pin: q Q Y y qb op*
- Run

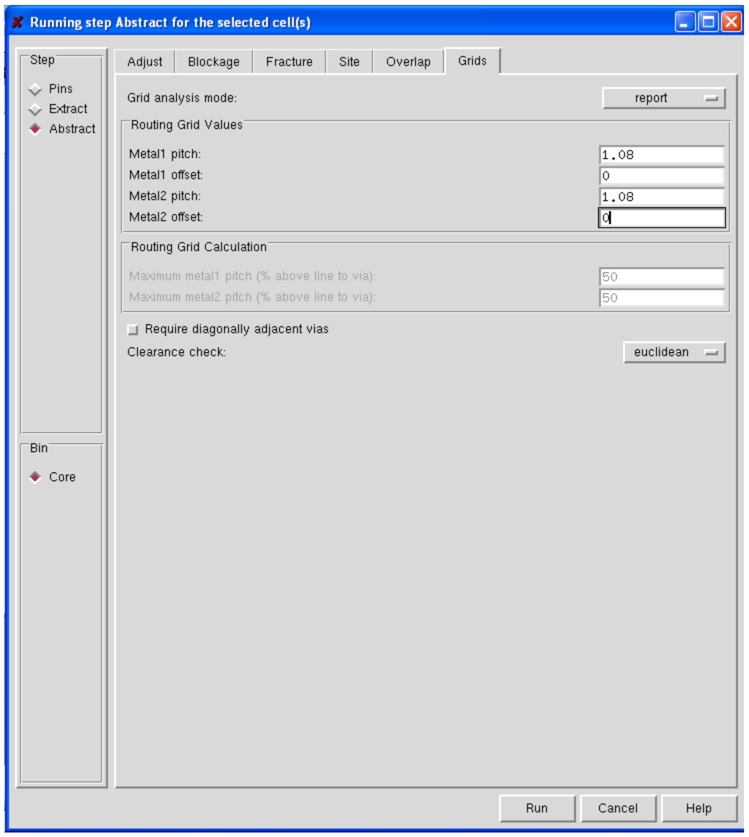
In the Pin section of the flow, you may get some 515 warnings. You may disregard these warnings.

- **In the Extract Section**

  - Use the default format and it should pass

- **In the Abstract Section**

  - Using the default settings, this step should pass as well.
  - Use Extract Signals selection
  - In the Grid tab, set the offset to 0.



Grid analysis mode: report

Routing Grid Values

Metal1 pitch: 1.08
Metal1 offset: 0
Metal2 pitch: 1.08
Metal2 offset: 0

Routing Grid Calculation

Maximum metal1 pitch (% above line to via): 50
Maximum metal2 pitch (% above line to via): 50

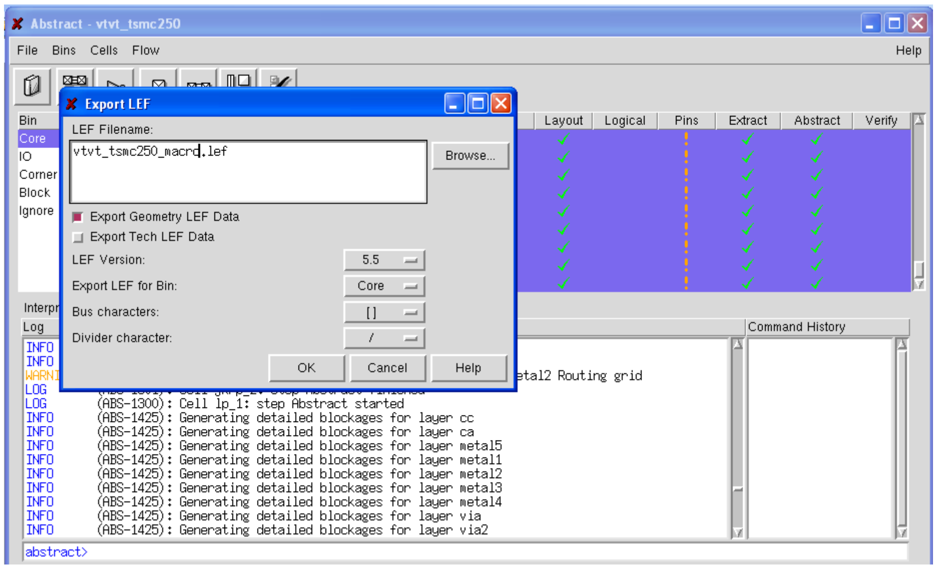
Require diagonally adjacent vias

Clearance check: euclidean

- Leave the Site blank or set it to SiteCore for standard cells. Set it to IOSite and/or CornerSire for the pads. I recommend you leave it empty.
- You may get some Warnings: 1076 about the Grid resolution. Modify the grid or leave it…If the Grid tab is set to 0, there are no more warnings.

- **Exporting MACRO LEF file**

  - Then move to File -> Export and get the LEF file for the bins you selected. Name it vtvt_tsmc250_macro.lef and pad_macro.lef



```
Export LEF
LEF Filename: vtvt_tsmc250_macro.lef
[✓] Export Geometry LEF Data
[ ] Export Tech LEF Data
LEF Version: 5.5
Export LEF for Bin: Core
Bus characters: []
Divider character: /
```

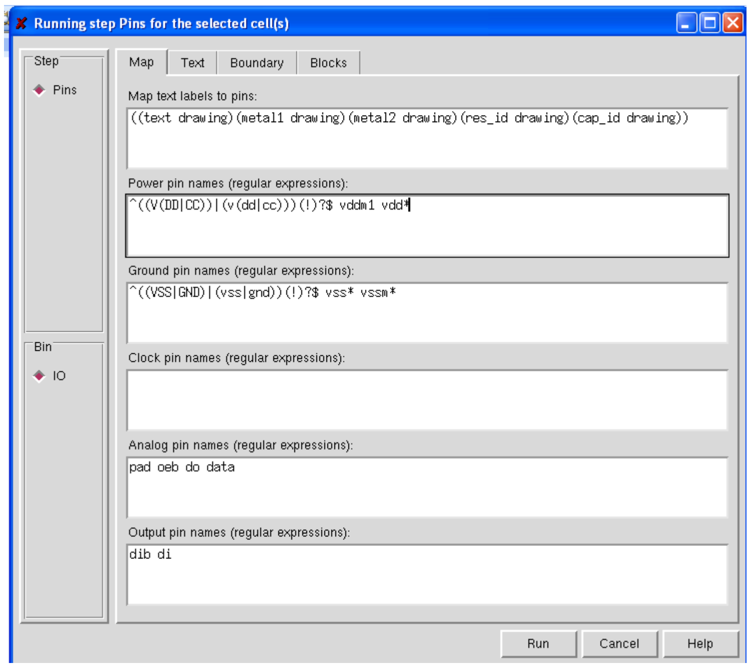
```
INFO
INFO
WARNI                                                        etal2 Routing grid
LOG
LOG     (ABS-1300): Cell lp_1: step Abstract started
INFO    (ABS-1425): Generating detailed blockages for layer cc
INFO    (ABS-1425): Generating detailed blockages for layer ca
INFO    (ABS-1425): Generating detailed blockages for layer metal5
INFO    (ABS-1425): Generating detailed blockages for layer metal1
INFO    (ABS-1425): Generating detailed blockages for layer metal2
INFO    (ABS-1425): Generating detailed blockages for layer metal3
INFO    (ABS-1425): Generating detailed blockages for layer metal4
INFO    (ABS-1425): Generating detailed blockages for layer via
INFO    (ABS-1425): Generating detailed blockages for layer via2
abstract>
```

- This LEF file will contain macros (information for each layout cell)

- **<u>Including I/O Pads</u>**

If you intend to include I/O pads, do the following using the dummymosispads.

- To get the pads' MACROs, attach the dummy library to vtvt_tsmc250_techfile. Move the padcorner into Corner Bin, all the others into IO Bin. And redo the Pins, Extract and Abstract steps mentioned above. The only changes occur in the Pin section. The naming goes as follow:
- In I/O and Corner bins, fill out the form as such:
  - Text Layers: ((text drawing)(metal1 drawing)(metal2 drawing)(metal5 drawing)(res_id drawing)(cap_id drawing))
  - Power Pin: ^((V(DD|CC))|(v(dd|cc)))(!)?$ vddm1 vdd1 vdd2 vddm2
  - Ground Pin: ^((VSS|GND)|(vss|gnd))(!)?$ vss1 vss2 vssm1 vssm2
  - Analog: pad oeb do data
  - Output Pin: dib di



Map text labels to pins:
```
((text drawing)(metal1 drawing)(metal2 drawing)(res_id drawing)(cap_id drawing))
```
Power pin names (regular expressions):
```
^((V(DD|CC))|(v(dd|cc)))(!)?$ vddm1 vdd*
```
Ground pin names (regular expressions):
```
^((VSS|GND)|(vss|gnd))(!)?$ vss* vssm*
```
Clock pin names (regular expressions):

Analog pin names (regular expressions):
```
pad oeb do data
```
Output pin names (regular expressions):
```
dib di
```

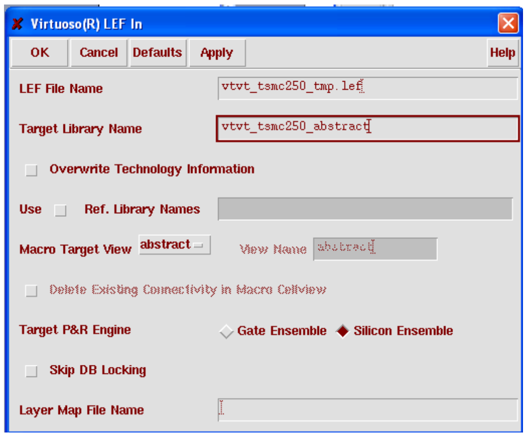
- Save the pad LEF macro file as pad_macro.

- **<u>Combining Macro Cells</u>**
  - Next, you need to combine the base LEF and the (pad+stdcell) MACRO LEF into one file: vtvt_tsmc250_tmp.lef
  - Modify the vtvt_tsmc250_tmp.lef by :
    - Add any missing “SHAPE FEEDTHRU ;” by the dummy pads after any USE GROUND or USE POWER statements.
    - Correct Site informations as follow:
      - SITE CoreSite
      -   SYMMETRY Y ;
      -   CLASS core ;
      -   SIZE 1.08 BY 15.12 ;
      - END CoreSite

      - SITE IOSite
      -   SYMMETRY Y ;
      -   CLASS pad ;
      -   SIZE 90.00 BY 300.00 ;
      - END IOSite

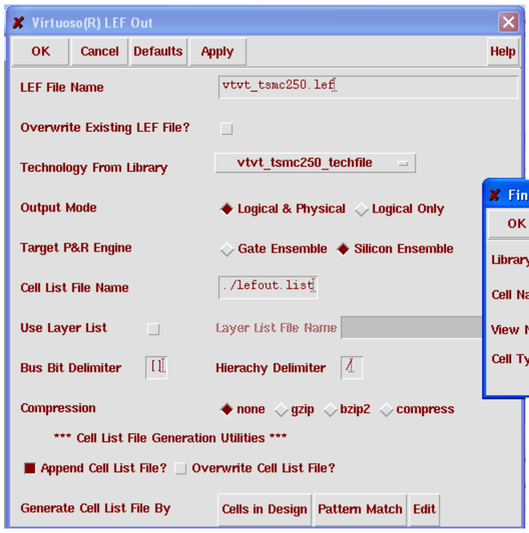
- SITE CornerSite
-     SYMMETRY R90 ;
-     CLASS pad ;
-     SIZE 300.00 BY 300.00 ;
- END CornerSite

- **<u>Create LEF file</u>**
    - Import this file back into Cadence and load it in a new library: vtvt_tsmc250_abstract which will contain all the abstract views of the standard cells and pads.
        - CIW->File->Import->LEF

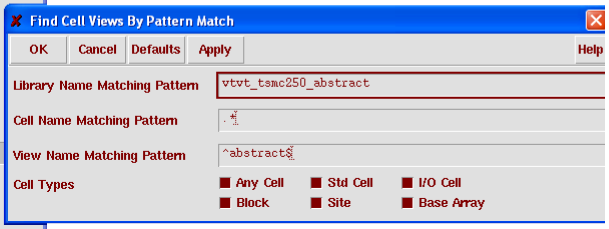


LEF File Name: vtvt_tsmc250_tmp.lef
Target Library Name: vtvt_tsmc250_abstract
☐ Overwrite Technology Information
Use ☐ Ref. Library Names
Macro Target View: abstract   View Name: abstract
☐ Delete Existing Connectivity in Macro Cellview
Target P&R Engine: ◇ Gate Ensemble [✓ Silicon Ensemble]
☐ Skip DB Locking
Layer Map File Name:

    - Once the CIW displays: "Done reading LEF file", you have now all the abstracted views for the standard cell library and the pads.
    - In Cadence CIW->Export->LEF



LEF File Name: vtvt_tsmc250.lef
Overwrite Existing LEF File? ☐
Technology From Library: vtvt_tsmc250_techfile
Output Mode: [✓ Logical & Physical] ◇ Logical Only
Target P&R Engine: ◇ Gate Ensemble [✓ Silicon Ensemble]
Cell List File Name: ./lefout.list
Use Layer List ☐   Layer List File Name
Bus Bit Delimiter: []   Hierachy Delimiter: /
Compression: [✓ none] ◇ gzip ◇ bzip2 ◇ compress
*** Cell List File Generation Utilities ***
☑ Append Cell List File? ☐ Overwrite Cell List File?
Generate Cell List File By: Cells in Design | Pattern Match | Edit



Library Name Matching Pattern: vtvt_tsmc250_abstract
Cell Name Matching Pattern: .*
View Name Matching Pattern: ^abstract$
Cell Types: ☑ Any Cell ☑ Std Cell ☑ I/O Cell ☑ Block ☑ Site ☑ Base Array

- Export back this LEF file and save it as vtvt_tsmc250.lef. Set the bus bit delimiters and hierarchy delimiter as shown above.
- You now have a valid LEF file for Place N Route.
- Test it in Cadence SOC encounter by following the proposed VTVT ASIC Design Flow.

G. Appendix: Generating a Symbol Library

This appendix presents the procedure undertaken to generate a symbol library for VTVT 0.25-micron standard cell library. A symbol library is technology independent. Therefore, this symbol library can be used with VTVT 0.18-micron cell library. Only the library name in the ASCII symbol library file will have to be modified.

For the design of the symbol library we are using the EDIF 200 Interface from Cadence Composer to Synopsys. The Design Compiler EDIF reader must be configured to receive the symbol library EDIF file.

The design plan is described on Figure 24 below. A technology library file must be available in order to make a symbol library. The symbol library can contain symbols of other cells not present in a technology library file, but it must contain a symbol description of all the cells present in the working technology library file.

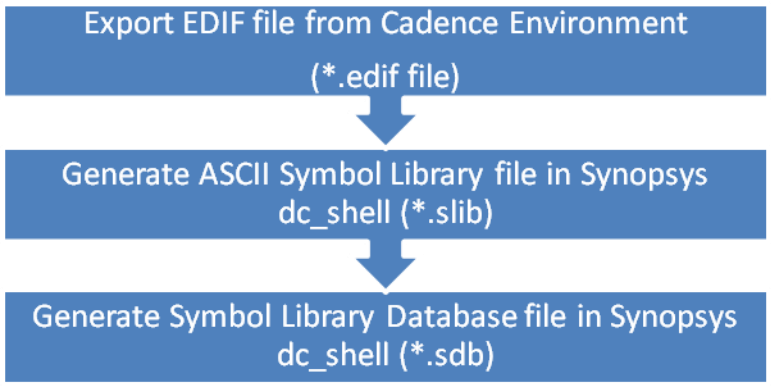

Export EDIF file from Cadence Environment
(*.edif file)
Generate ASCII Symbol Library file in Synopsys dc_shell (*.slib)
Generate Symbol Library Database file in Synopsys dc_shell (*.sdb)

Figure 24: Design Flow Chart

Export Cadence Composer EDIF file (*.edif file)

We are using Cadence to export an EDIF file of the symbol library to be designed. To export EDIF 200 of the symbol library, follow the steps below:

- For each of the cells of the library make sure vss and vdd are used as power supply nets
  - Make a schematic per the layout measurement
  - Create the symbol cell view from the schematic view of the corresponding cell
- Run LVS between schematic and extracted views from cell for verification
- Once all your schematics and symbols are generated, run a script that will generate a unique Composer Schematic cell view that contains all instantiations of the symbols in the library.

- Generate an edifOut.il file that will be used to export and EDIF file. Bellow is an EDIF file model:

```
edifOutKeys = list(nil

        'runDirectory           "/home/jddjig01/v/tech/ncsu/symbol_testing/"

        'outputFile             "OUT_symbol_lib_test.edif"

        'flattenDir             ""

        'library        "symbol_lib_test"

        'cell                   "total_cell_symbol"

        'viewName               "schematic"

        'externalLibList ""

        'scalarOption           "FALSE"

        'interpretInhConn       "FALSE"

        'scalarWithPortsUnexpanded      "FALSE"

        'netlistOption          "FALSE"

        'outputCDFData                  "FALSE"

        'skipDupInstNameCk      "FALSE"

        'replaceBundleWithArray         "TRUE"

        'netlistMode            "Hierarchical"

        'hierarchyFile          ""

        'stopCellExpansionFile  ""

        'ILFile                 ""

        'design                 ""

        'techFile               "tsmc_03d.tf"

        'ripperLibraryName      "ripper"

        'ripperCellName                 "ripper"

        'ripperViewName                 "symbol"

)
```

- Before exporting the EDIF file, the dot.synopsys_dc.setup file must be updated. Below is a model of the .synopsys_dc.setup file.

```
/*

  This is an example .synopsys_dc.setup file which should be

  used when transferring designs to Cadence through EDIF.

*/

/* Library and Search Path variables */

search_path = {. /software/synopsys/2003.06/libraries/syn
/home/jddjig01/v/tech/ncsu/symbol_testing }

link_library = {"symbol_lib_test.db"}

target_library = {"symbol_lib_test.db"}

symbol_library = {"symbol_lib_test.sdb" "basic.sdb" "ripper.sdb" "US.8ths.sdb"}

/* Verilog */

verilogout_no_tri = "true"

/* Read Symbol */

edifin_lib_route_grid = 1024

vhdlout_use_packages = {IEEE.std_logic_1164, IEEE.std_logic_arith, IEEE.std_logic_textio,
symbol_lib_test.components}

/* EDIF read symbol library variables */

/* EDIF variables for Importing Special Connectors for Composer*/

    edifin_lib_in_port_symbol = "ipin"

    edifin_lib_out_port_symbol = "opin"

    edifin_lib_inout_port_symbol = "iopin"

    edifin_lib_in_osc_symbol = "iosc"

    edifin_lib_out_osc_symbol = "oosc"

    edifin_lib_inout_osc_symbol = "oosc"

    edifin_lib_logic_1_symbol = "vdd"

    edifin_lib_logic_0_symbol = "vss"

/* EDIF variables for Importing a Bus Ripper from Composer*/

    edifin_lib_ripper_bits_property = "schPatchExpr"
    edifin_lib_ripper_bus_end = "bus_end"

    edifin_lib_ripper_cell_end = "ripper"

    edifin_lib_ripper_view_name = "symbol"

    edifin_lib_route_grid = 1024

/* EDIF variables for Importing Sheet Templates from Composer*/

    edifin_lib_templates = {{A,landscape,Asize},{A,portrait,Asize.book}, \

    {B,landscape,Bsize},{C,landscape,Csize},{D,landscape,Dsize}, \

    {E,landscape,Esize},{F,landscape,Fsize}}

    /* Bus Naming variables */

    bus_naming_style = "%s<%d>"

    bus_dimension_separator_style = "><"

    bus_range_separator_style = ":"

    bus_extraction_style = "%s<%d:%d>"

    bus_minus_style = "-%d"

    edifout_no_array = "false"

    /* Read design variagles */

    edifin_autoconnect_offPageConnectors = "true"

    edifin_delete_empty_cells = "true"

    edifin_delete_ripper_cells = "true"

    /* Power and Ground variables */

    edifin_ground_net_name = "vss!"

    edifin_ground_net_property_name = ""

    edifin_ground_net_property_value = ""

    edifout_ground_name = "vss!"

    edifout_ground_net_name = "vss!"

    edifout_ground_net_property_name = ""

    edifout_ground_net_property_value = ""

    edifout_ground_pin_name = "vss!"
```

```
edifin_power_net_name = "vdd!"

edifin_power_net_property_name = ""

edifin_power_net_property_value = ""

edifout_power_name = "vdd"

edifout_power_net_name = "vdd!"

edifout_power_net_property_name = ""

edifout_power_net_property_value = ""

edifout_power_pin_name = "vdd!"

edifout_power_and_ground_representation = "net"

/* Net to Port Connection variables */

edifin_autoconnect_ports = "true"

compile_fix_multiple_port_nets = "true"

gen_match_ripper_wire_widths = "true"

edifout_name_rippers_same_as_wires = "false"

link_force_case = "case_insensitive"

single_group_per_sheet = "true"

use_port_name_for_oscs = "false"

write_name_nets_same_as_ports = "true"

/* Output variables */

edifout_netlist_only = "false"

edifout_external = "true"

edifout_translate_origin = "center"

edifout_display_instance_names = "false"

edifout_display_net_name = "false"

edifout_target_system = "cadence"

edifout_instantiate_ports = "true"

edifout_pin_name_property_name = "pinName"

edifout_designs_library_name = "SYNOPSYS"
```

- The files basic.slib, ripper.slib, US.Sheet.8ths must also be available in the working directory. They can be obtained from the directory: $SYNOPSYS_ROOT/doc/syn/interfaces/cadence/libraries/
- In the CIW, go to File->Export->EDIF200 and load the edifOut.il file as the template file and click OK to export an EDIF file of the symbol library symbol_lib_test (per the edifOut.il file above).
- The file OUT_symbol_lib_test.edif is generated.

Generating ASCII Symbol Library (*.slib file)

Once the *.edif file is available, the ASCII symbol library must be generated. To create the ASCII symbol library file, the user simply needs to enter:

```
dc_shell> read_lib Your_EDIF_File -format edif -symbol Your_Library.slib
```

Verifying Content of ASCII Symbol Library

- The command compare_lib will make sure that all the cells in the technology library file are all represented by a symbol cell in the corresponding symbol library file.
- Verify that the scale, power nets values, and all the special symbols (pins, sheets) are present in the .slib file.
- Write a script that will verify the validity of the symbol library.

Generating the Symbol Library (.sdb file)*

Once the *.slib file has been verified, type within the Synopsys environment:

```
dc_shell > write_lib Your_Library.slib –o Your_Library.sdb
```

Add this file Your_Library.sdb to your dot_synopsys_dc.setup file.

The Symbol library file is generated.

Testing Symbols in Synopsys Synthesis

To test the design, make a VHDL (or Verilog) test case that contains the connectivity of each then or all the cells in the symbol library (if possible).

For the symbol_lib_test library used in this case, the file symbol_lib_testcase.vhd is created as a tescase. Figure 25 shows the content of symbol_lib_testcase.vhd:

```
-- File name: symbol_lib_testcase.vhd
-- a testcase for symbol_lib_test libray
-- It implements (not(a)) nor (b) and outputs (c)
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity symbol_lib_testcase is
        port (a,b: in STD_LOGIC;
                c: out STD_LOGIC);
end;
architecture BEHAVIOR of symbol_lib_testcase is
        begin
                c <= ((not a) NOR b);
end BEHAVIOR;
```

Figure 25: symbol_lib_tescase.vhd File Content

Compile the symbol_lib_testcase file by typing: vhdlan symbol_lib_testcase.vhd

Then in your Synopsys environment type: design_analyser&

Go to File->Read and select symbol_lib_testcase.vhd

Click once on the cell and go to Edit->Uniquify->Hierarchy

Go to Tools-> Design Optimization->Select Verify Design-> Click OK

Then double click on the cell and the expected schematic of the design should be represented.

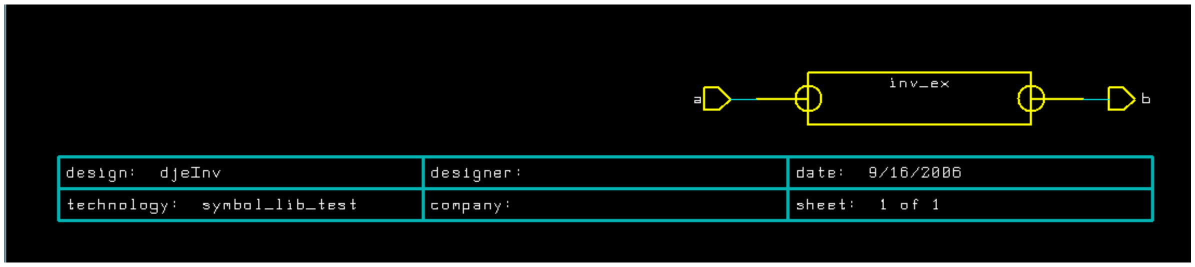
To illustrate this test, I use the symbol_lib_test library which contains two cells, a NOR and an inverter named nor_ex, and inv_ex respectively.

Inverter Test Case : Cell inv_ex

The file djeInv.vhd is shown below

```
-- File name: djeInv_bhv.vhd

-- An inverter

library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity djeInv is

            port (a: in STD_LOGIC;

                        b: out STD_LOGIC);

end;

architecture BEHAVIOR of djeInv is

            begin

                        b <= (not a);

end BEHAVIOR;
```

The file is compiled and read in design_analyser. The result is shown on Figure 26.



design: djeInv	designer:	date: 9/16/2006
technology: symbol_lib_test	company:	sheet: 1 of 1

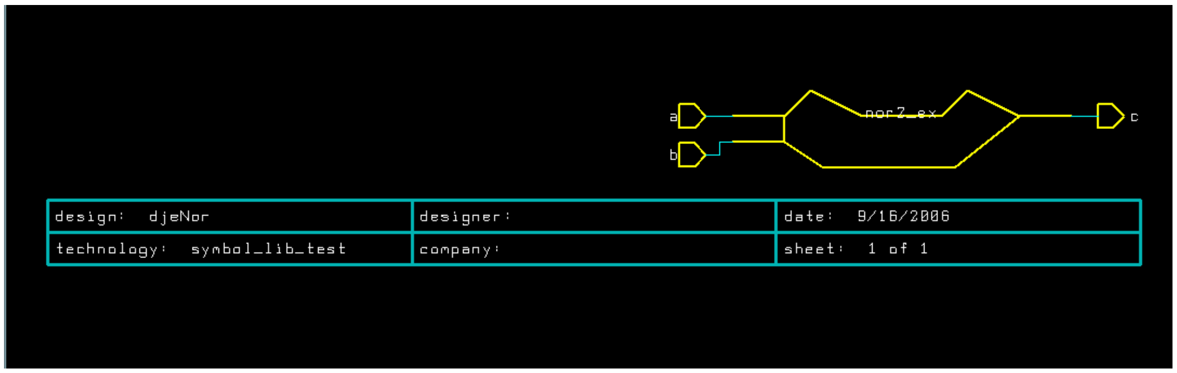
Figure 26: djeInv.vhd schematic

Nor Cell Test Case: Cell Name nor2_ex

The vhdl file is shown below:

```
-- File name: djeNor_bhv.vhd

-- A nor gate

library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity djeNor is

        port (a,b: in STD_LOGIC;

                c: out STD_LOGIC);

end;

architecture BEHAVIOR of djeNor is

        begin

                c <= (a NOR b);

end BEHAVIOR;
```

The file is compiled and the schematic is shown below on Figure 27.



```
design:   djeNor                  designer:      date:   9/16/2006
technology:   symbol_lib_test     company:       sheet:   1 of 1
```

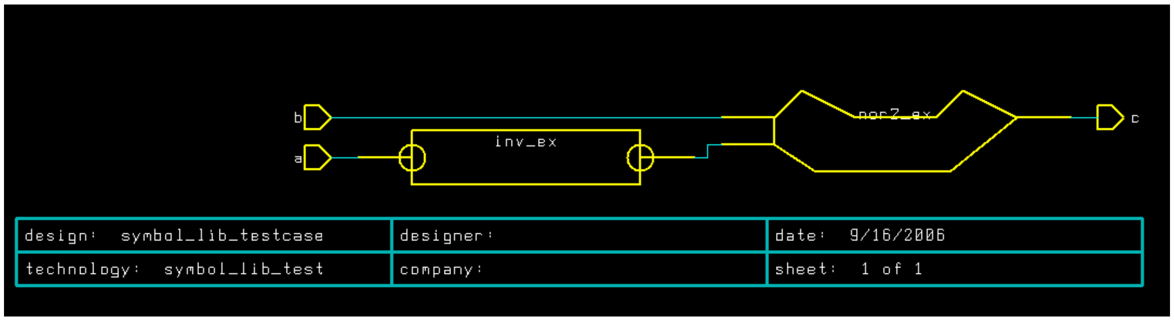
Figure 27:djeNor.vhd

Combination of Cells Test Case

A description of the combination of both cells is shown in symbol_lib_testcase.vhd below.

```
-- File name: symbol_lib_testcase.vhd
-- a testcase for symbol_lib_test libray
-- It implements (not(a)) nor (b) and outputs (c)
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity symbol_lib_testcase is
        port (a,b: in STD_LOGIC;
                c: out STD_LOGIC);
end;
architecture BEHAVIOR of symbol_lib_testcase is
        begin
                c <= ((not a) NOR b);
end BEHAVIOR;
```

The schematic shown in Synopsys Synthesis is displayed below on Figure 28.



design: symbol_lib_testcase	designer:	date: 9/16/2006
technology: symbol_lib_test	company:	sheet: 1 of 1

Figure 28: Testcase for symbol_lib_test Library

H. Appendix: Compiling a Symbol Library

This appendix contains instructions to compile a symbol library for VTVT 0.18-micron library. Compiling a symbol library consists of generating a database format (.sdb) file from the symbol library ASCII format (.slib) file.

These steps are also included in the release package for VTVT 0.18-micron library.

1 . Read the *.slib file within lc_shell

```
$lc_shell

>read_lib vtvt_tsmc180.slib
```

Disregard the WARNING messages about SCALE and ROUTE_GRID redefinitions. This should not be an issue if you verify that the redefined values are the same. The statements to check for are:

- Symbol library 'NCSU_Analog_Parts' read successfully
- Symbol library 'basic' read successfully
- Symbol library 'vtvt_tsmc180' read successfully

2. Generate the *.sdb

For each of the symbol libraries read do the following:

- write_lib NCSU_Analog_Parts -o NCSU_Analog_Parts.sdb
- write_lib basic -o basic.sdb
- write_lib vtvt_tsmc180 -o vtvt_tsmc180.sdb

3. Move all files to libs directory

- mv basic.s* libs/.
- mv NCSU_Analog_Parts.sdb libs/.
- mv US.8ths.s* libs/.
- mv vtvt_tsmc180.s* libs/.

<u>4. Update file dotsynopsys_dc.setup to include these libraries:</u>

<u>4.1 Add libs directory to search path:</u>

```
set search_path {. ./libs <path_to_Synopsys_installation_dir>/Y-2006.06-SP2/libraries/syn}
```

<u>4.2 Include symbols libraries</u>

```
set symbol_library {"vtvt_tsmc180.sdb" "basic.sdb" "NCSU_Analog_Parts.sdb" "US.8ths.sdb"}
```

I. Appendix : ALLLIB_DRC Script Description

Overview

This is a SKILL script that will automatically run DRC on all the layout views of any given library. The command stores the run summary in a default log file: <library_name>_drc.log. The command name is **alllib_drc**. It must be a SKILL file named **alllib_drc.il**.

It should be invoked within the Cadence Design Kit (NCSU_CDK) CIW as follow:

- ➢ load “alllib_drc.il”
- ➢ alllib_drc “vtvt_tsmc250”

A call to the command would be:

alllib_drc *library_name*

Example: Calling alllib_drc vtvt_tsmc250

This call runs DRC on all the standard cell libraries present in library vtvt_tsmc250. It outputs the run summary in the standard output and in file vtvt_tsmc250_drc.log

Specifications

These are the specifications to follow to generate the FINAL alllib_drc script.

Content of the Library:

It should be assumed that once the library is found, it contains layout views of cells. If the library does not contain any layout views of a cell, do not comment on it. Skip the cell and move to the next cell. However, in the run summary, only include the cells that have a layout view.

Library not found message

If the library is not in the current directory, output the following statement in standard output AND in the log file:

```
Library <library_name> not found.
Aborted DRC run.
```

Format of the log file

The log file: <library_name>_drc.log and standard output should follow the following format.

1. Introductory Title line. (Command entered)
2. Run Status for each cell (with a list of cells)
3. Write a Run Summary
4. End of Summary and Date run.

An example of log file is shown below. One contains errors and the other does not.

```
vtvt_tsmc250_drc.log: alllib_drc.il vtvt_tsmc250 on January 19, 2007 4pm


Running DRC for library vtvt_tsmc250…

inv_1…Pass

inv_2…Pass

inv_4…Pass

buf_1...Fail

buf_2....Pass

and2_1...Fail

.

.
```

```
vtvt_tsmc250_drc.log: alllib_drc.il vtvt_tsmc250 on January 19, 2007 4:05pm


Running DRC for library vtvt_tsmc250…

inv_1…Pass

inv_2…Pass

inv_4…Pass

buf_1...Pass

buf_2....Pass

and2_1...Pass
```

Design Steps

In order to get all the specifications, the steps below propose a design approach for this assignment in a progressive order.

1. Run a SKILL script that will print out the following:

“Running DRC for standard cell library <name of the standard cell library>…”

List all the cells of the library … Success / Fail

Summary:

Your standard cell library is DRC –clean/Fail

Or

DRC Errors in the following cells

(List the errors)

2. If the library is not in the run directory, print error message on stdout AND in log file.

Library <library_name> not found.

3. Print the List of cells in the library that have a layout view.
4. Run DRC for all cells
5. List Status and Cells
6. Write a summary

J. Appendix: ALLIB_DRC.IL SCRIPT

This appendix contains the content of the DRC script “alllib_drc.il”

```
procedure( alllib_drc( LibName )

; syntax:
; load "alllib_drc.il"
; alllib_drc "LibName"

prog( ( currDir RulePath Path firsttime Port Line pattern filelist
dumpport restoreport cellName oport tempport cellHandle logport LogLine
passfail errorCellList errorCountList sumPort cellCount techName errorName
) ;local vars

;;>> initialize variables
currDir = getWorkingDir()
RulePath = strcat( getShellEnvVar( "CDK_DIR" ) "/techfile/divaDRC.rul" )
Path = nil
firsttime = t
cellCount = 0

;;>> clear out old log files
if( isDir( "DRClogs") then sh( "rm -f ./DRClogs/*.log" ) else sh( "mkdir
DRClogs" ))

sumPort = outfile( strcat( currDir "/DRClogs/" LibName "_drc.log" ) )

fprintf( sumPort "\nSetting up DRC run for %s\n" LibName )

;;>> check for the presence of cds.lib in the current directory
if( not( isFile( concat( currDir "/cds.lib" ) ) )
      then
            printf( "ERROR: cds.lib was not found within the current
working directory.\n" )
            fprintf( sumPort "ERROR: cds.lib was not found within the
current working directory.\n" )
            close( sumPort )
            return( nil )
      else
            fprintf( sumPort "Searching cds.lib for library definition.\n"
)
)


;;>> look in cds.lib for path definition of specified library
Port = infile( concat( currDir "/cds.lib" ) )
while( gets( Line Port )
      rexCompile( strcat( LibName '"[ \t]+" ) )
      if( rexExecute( Line )
      then
            pattern = strcat( '"DEFINE[ \t]*" LibName '"[ \t]*" )
            rexCompile( pattern )
            Path = rexReplace( Line "" 1 )
            rexCompile( "\n" )
            Path = rexReplace( Path "" 0 )
      )
)
close( Port )

;;>> if we get to here and the path variable is still nil, then the
library is not defined in the cds.lib
if( not( Path )
      then
            printf( "The library %s was not found.\nAborted DRC run.\n"
LibName )
            fprintf( sumPort "The library %s was not found.\nAborted DRC
run.\n" LibName )
            close( sumPort )
            return( nil )
      else
            fprintf( sumPort "Library definition found, verifying path.\n"
)
)

;;>> check for the existence of the specified library
if( not( isDir( Path ))
      then
            printf( "ERROR: The library definition in cds.lib may not
contain the correct path.\n" )
            fprintf( sumPort "ERROR: The library definition in cds.lib may
not contain the correct path.\n" )
            close( sumPort )
            return( nil )
      else
            fprintf( sumPort "Path is valid.\n" )
);end if


;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Files check out, so now loop
through all cells and run DRC
printf( "Running DRC for %L \n" LibName )

;;>> get a list of cells from the filesystem directory listing
filelist=getDirFiles( Path )


;;>> redirect output to the batch runtime log
dumpport= outfile( strcat( currDir "/DRClogs/batch_runtime_master.log" ) )
restoreport=poport         ;;<< saves pointer to stdout so that it may be
restored later
poport=dumpport            ;;<< remap port

errorCellList=list()
errorCountList=list()

techName=techGetTechLibName( techGetTechFile( ddGetObj( LibName ) ) )
```

```
fprintf( sumPort "\nRunning DRC for %s:\n\n" LibName )

;;>> now we have a filelist, so loop through each and run drc if it is a
directory with a subdirectory named "layout"
for( i 0 ( length( filelist ) - 1 )
      cellName = nth( i filelist )
      if( isDir( strcat( Path "/" cellName "/layout" )) ; checks that cell
has layout subdirectory
      then
            cellCount=1+cellCount
      ;;>> due to the way ivDRC works in our environment, there needs to
be a Virtuoso window open
      ;;>> for those functions which expect a window handle. So, the first
cell which is located
      ;;>> is opened in Virtuoso. This window will be closed upon
completion of the DRC checks

            cellHandle =  dbOpenCellViewByType( LibName cellName "layout"
nil "a" )

            if( firsttime
            then
                  winhandle=ddsServOpen( LibName cellName "layout" "edit" )
                  firsttime=nil
            )

      ;;>> remove runtime specific rules if the file exists
            if( isFile( "runspecific.il" ) then sh( "rm runspecific.il" )
)


      ;;>>create the runtime specific rules file in order to specify a
unique log file for each cell.
      ;;>>silly, but the drcLogFile can not be specified from the calling
skill script
            string = strcat( "drcLogFile( \"DRClogs/" cellName ".log\" )"
)
            oport = outfile( strcat( currDir "/runspecific.il" ) )
            fprintf( oport "%s\n" string )
            close( oport )

      ;;>>create the logfile to be used by drc (unique for each cell,
stored in ./DRClogs/<CELLNAME>.log)
            tempport = outfile( strcat( "DRClogs/" cellName ".log" ) )
            close( tempport )

      ;;>> run DRC for the cell, poport is redirected to
batch_runtime_master.log, DRC log is redirected to <cellname>.log
            ivDRC( ?cell cellHandle ?full t ?rulesLibName techName ?rsf
strcat( currDir "/runspecific.il" ) )

      ;;>> now check generated logfile to see if DRC completed without
errors
```

```
            logport = infile( strcat( currDir "/DRClogs/" cellName ".log"
) )
            count="1"
            while( gets( LogLine logport )
                  if( nindex( LogLine "Total error count is" )
                  then
                        pattern = strcat( "[ \t]*Total error count is[
\t]*")
                        rexCompile( pattern )
                        count = rexReplace( LogLine "" 1 )
                  )
            )
            close( logport )

            if( (count == "0")
            then
                  passfail = "pass"
            else
                  passfail = strcat( "fail ( " count " error(s) )" )
                  rexCompile( "\n" )
                  passfail=rexReplace( passfail "" 0 )
                  count=rexReplace( count "" 0 )
                  errorCountList=cons( count errorCountList )
                  errorCellList=cons( cellName errorCellList )
            )

            poport = restoreport

            printf( "%s" cellName )
            for( i 0 (30-strlen( cellName )) printf( "." ))
            printf( "  %s\n" passfail )

            poport = dumpport

            fprintf( sumPort "%s" cellName )
            for( i 0 (30-strlen( cellName )) fprintf( sumPort "." ))
            fprintf( sumPort "  %s\n" passfail )

      )
)

poport=restoreport
close( dumpport )

;summary

printf( "\n\nDRC summary for library %s:\n" LibName )
fprintf( sumPort "\nDRC summary:\n\n" )

if( length( errorCellList ) == 0
      then
            printf( "%s is DRC clean" LibName )
            fprintf( sumPort "\t%s is DRC clean" LibName )
      else
```

```
            for( i 0 (length( errorCellList ) - 1)
                  logport = infile( strcat( currDir "/DRClogs/" nth( i
errorCellList ) ".log" ) )
                  printf( "%s has %s errors:\n" nth( i errorCellList ) nth(
i errorCountList ) )
                  fprintf( sumPort "\t%s has %s errors\n" nth( i
errorCellList ) nth( i errorCountList ) )
                  while( gets( LogLine logport )
                        if( nindex( LogLine "RULE LABEL" )
                              then
                                    rexCompile( strcat( "RULE LABEL:[ \t]*"
) )
                                    errorName = rexReplace( LogLine "" 1 )
                                    printf( "\t%s" errorName )
                                    fprintf( sumPort "\t\t%s" errorName )
                        )
                  )
            close( logport )
            )
)

drain( poport )
close( sumPort )
hiCloseWindow(winhandle)

); prog
); procedure
```

K. Appendix: ALLLIB_LVS Description

This appendix describes the LVS script mentioned in this thesis. The following documentation provides details on the command.

Overview

This is a SKILL script that will automatically run LVS on all the extracted/schematic views of any given library. The command stores the run summary in a default log file: <library_name>_lvs.log. The command name is **alllib_lvs**. It must be a SKILL file named **alllib_lvs.il**.

A call to the command would be:

alllib_lvs *library_name*

Example: Calling

alllib_lvs vtvt_tsmc250

This call runs LVS on all the standard cell libraries present in library vtvt_tsmc250. It outputs the run summary in the standard output and in file vtvt_tsmc250_lvs.log

Specifications

These are the specifications to follow to generate the FINAL alllib_lvs script.

Content of the Library:

It should be assumed that once the library is found, it contains layout and schematic views of cells. If the library does not contain any layout and schematic views of a cell, do not comment on it. Skip the cell and move to the next cell. However, in the run summary, only include the cells that have a layout view.

File not found message

If the library is not in the current directory, output the following statement in standard output AND in the log file:

```
Library <library_name> not found.

Aborted LVS run.
```

Format of the log file

The log file: <library_name>_lvs.log and standard output should follow the following format.

Introductory Title line. (Command entered)

Run Status for each cell (with a list of cells)

Write a Run Summary

End of Summary and Date run.

```
vtvt_tsmc250_lvs.log: alllib_lvs.il vtvt_tsmc250 on January 19, 2007 4pm



Running LVS for library vtvt_tsmc250…

inv_1…Pass

inv_2…Pass

inv_4…Pass

buf_1...Fail

buf_2....Pass

and2_1...Fail

.
```

An example of log file is shown below. One contains errors and the other does not.

```
vtvt_tsmc250_lvs.log: alllib_lvs.il vtvt_tsmc250 on January 19, 2007 4:05pm



Running LVS for library vtvt_tsmc250…

inv_1…Pass

inv_2…Pass

inv_4…Pass

buf_1...Pass

buf_2....Pass

and2_1...Pass
```

Design Steps

In order to get all the specifications, the steps below propose a design approach for this assignment in a progressive order.

1. Run a SKILL script that will print out the following:
   "Running LVS for standard cell library <name of the standard cell library>…"

   List all the cells of the library … Success / Fail

   Summary:

   Your standard cell library is LVS –clean/Fail

   Or

   LVS Errors in the following cells

   (List the errors)
2. If the library is not in the run directory, print error message on stdout AND in log file.
   Library <library_name> not found.
3. Print the List of cells in the library that have a layout and a schematic view.
4. Run Extraction then LVS for all cells
5. List Status and Cells
6. Write a summary

L. Appendix: ALLLIB_LVS.IL SCRIPT

This appendix contains the content of the LVS script “alllib_lvs.il”

```
procedure( alllib_lvs( LibName @rest CellList )

;;>> This script will run LVS.
;;>> syntax:
;;>> load "alllib_lvs.il"
;;>> alllib_lvs( "LibName" "Cell1" "Cell2" ... )
;;>> Cell names are optional.  If they are absent, LVS runs for the whole
library.

prog( ( currDir LVSRules Path firsttime Port Line pattern filelist
dumpport restoreport cellName oport logport LogLine passfail error1 error2
error3 error4 sumPort cellCount ) ;local vars

;;>> initialize variables
currDir = getWorkingDir()
LVSRules=strcat( getShellEnvVar( "CDK_DIR" ) "/techfile/divaLVS.rul" )
Path=nil
firsttime=t
cellCount=0

;;>> check for the presence of cds.lib in the current directory
if( not( isFile( concat( currDir "/cds.lib" ) ) )
then
      printf("ERROR: cds.lib was not found within the current working
directory.\n")
      return( nil )
)

;;>> look in cds.lib for path definition of specified library
;;>> this works in Jeanette's cds.lib even though it's FUBAR
Port = infile( concat( currDir "/cds.lib" ) )
while( gets( Line Port )
      rexCompile( strcat( LibName '"[ \t]+" ) )
      if( rexExecute( Line )
      then
            pattern = strcat( '"DEFINE[ \t]*" LibName '"[ \t]*" )
            rexCompile( pattern )
            Path = rexReplace( Line "" 1 )
            rexCompile( "\n" )
            Path = rexReplace( Path "" 0 )
      )
)
close( Port )

;;>> if we get to here and the path variable is still nil, then the
library is not defined in the cds.lib
```

```
if( not( Path )
      then
            printf("ERROR: The library was not found in cds.lib.\n")
            return( nil )
)

;;>> check for the existence of the specified library
if( not( isDir( Path ))
      then
            printf("ERROR: The library definition in cds.lib may not
contain the correct path.\n")
            return( nil )
)


;;>>>>>>>>>>>>>>>>>>>>>>>>>>>> Library found, so now loop through all cells
and run LVS

;;>> clear out old log files
if( isDir( "LVSlogs") then sh( "rm -f ./LVSlogs/*.log" ) else sh( "mkdir
LVSlogs" ))


if( length(CellList)==0
      then
      ;;>> get a list of cells from the filesystem directory listing
            filelist=sort( getDirFiles( Path ) nil )
            printf( "Running LVS for %s...\n" LibName )
      else
      ;;>> use provided list
            filelist=CellList
            printf( "Running LVS for given cells in %s...\n" LibName )
)

printf( "Ignore any warnings and BE PATIENT...\n" )

;;>> redirect output to the batch runtime log
dumpport= outfile( strcat( currDir "/LVSlogs/batch_runtime_master.log" ) )
poport=dumpport           ;;<< remap port

error1 = nil  ;;<< failed to match
error2 = nil  ;;<< execution failed
error3 = nil  ;;<< missing view
error4 = nil  ;;<< not a cell

NCSU_parasiticCapIgnoreThreshold = 1e-18

;;>> now we have a filelist, so loop through each and run lvs if it is a
directory with a subdirectory named "layout"
foreach( cellName filelist
      ;;>> checks that the cell has both layout and schematic views
      if( and( isDir( strcat( Path "/" cellName "/layout/" )) isDir(
strcat( Path "/" cellName "/schematic/" )) )
      then
```

```
            cellCount++
      ;;>> due to the ways ivExtract and ivLVS work in our environment,
there needs to be a Virtuoso window open
      ;;>> for those functions which expect a window handle. So, the first
cell which is located
      ;;>> is opened in Virtuoso. This window will be closed upon
completion of the LVS checks
            if( firsttime
                  then
                        winhandle = ddsServOpen( LibName cellName "layout"
"edit" )
                        firsttime = nil
            )

            passfail = nil

      ;;>> run LVS for the cell, poport is redirected to
batch_runtime_master.log, DRC log is redirected to <cellname>.log
            ivExtract( ?cell dbOpenCellViewByType( LibName cellName
"layout" nil "a" ) ?set "Extract_parasitic_caps" )
            currWin=hiGetCurrentWindow()

            ivLVS( "LVSrundir" LibName cellName "extracted" t LibName
cellName "schematic" t t nil t nil " " nil 20 t " " LVSRules nil )

      ;;>> The script now checks the LVS log file until it sees a fatal
error or LVS completes successfully
            done=nil

            while( not( done )
                  sh( strcat( "cp ./LVSrundir/si.log ./LVSlogs/" cellName
".log" ) )
                  logport = infile( strcat( currDir "/LVSlogs/" cellName
".log" ) )
                  while( gets( LogLine logport )
                        if( nindex( LogLine "Comparison program completed
successfully." )  then done=1 )
                        if( nindex( LogLine "*Error*" ) then done=2 )
                  )
                  ipcSleep( 1 )
            )

      ;;>> LVS is done, generates output
            if( done == 1
                  then
                        logport = infile( strcat( currDir "/LVSlogs/"
cellName ".log" ) )
                        while( gets( LogLine logport )
                              if( nindex( LogLine "The net-lists " )
                                    then
                                          if( nindex( LogLine "The net-lists
match" ) then passfail="matched" )
                                          if( nindex( LogLine "The net-lists
failed to match" )
```

```
                                        then
                                             passfail = "failed to
match"
                                             error1 = cons(
cellName error1 )
                                   )
                         )
                    )

                    close( logport )

               else
                    passfail="LVS failed to complete"
                    error2=cons( cellName error2 )
          )

          ipcSleep( 1 )

          poport=stdout

          printf( "%s" cellName )
          for( i 0 (30-strlen( cellName )) printf( "." ) )
          printf( "  %s\n" passfail )

          poport=dumpport

          ;;>> If you can figure out why this command works, I'll give
you a dollar. (it closes the box saying LVS is done)
          hiDBoxOK( simNetNoOp6 )

     else
          if( not( nindex( cellName "." ) )
               then
                    if( isDir( strcat( Path "/" cellName ))
                         then
                              cellCount++
                              error3=cons( cellName error3 )
                         else
                              error4=cons( cellName error4 )
                    )
          )
     )
)

poport=stdout
close( dumpport )

;;>> summary

sumPort=outfile( strcat( currDir "/LVSlogs/summary.log" ) )

printf( "\n\nError summary:\n" )
fprintf( sumPort "\nLVS summary:\n\n" )
fprintf( sumPort "\t%d cells checked.\n\n" cellCount )
fprintf( sumPort "\t%d cells matched.\n\n" ( cellCount - length( error1 )
- length( error2 ) - length( error3 ) ) )
fprintf( sumPort "\t%d cells did not run LVS.\n\n" length( error3 ) )
fprintf( sumPort "\tError summary:\n\n" )

if( error1==nil
      then
            printf( "Library is LVS clean\n" )
            fprintf( sumPort "\t\tLibrary is LVS clean\n" )
)

foreach( cellName error1
      printf( "%s failed to match\n" cellName )
      fprintf( sumPort "\t\t%s failed to match\n" cellName )
)

foreach( cellName error2
      printf( "%s LVS did not complete successfully\n" cellName )
      fprintf( sumPort "\t\t%s LVS did not complete successfully\n"
cellName )
)

foreach( cellName error3
      printf( "%s Layout or Schematic view missing\n" cellName )
      fprintf( sumPort "\t\t%s Layout or Schematic view missing\n"
cellName )
)

foreach( cellName error4
      printf( "%s is not a cell\n" cellName )
      fprintf( sumPort "\t\t%s is not a cell\n" cellName )
)

close(sumPort)
hiCloseWindow( winhandle )

); prog
); procedure
```

M. Appendix: NETLIST_LIBRARY Description

Layout extraction consists of using Cadence tools to extract Capacitance and device models of a layout view of a cell. In the process of characterizing standard cells, I use HSpice to determine rise/fall time delays, power dissipation and other parameters that will be discussed in another section. The Netlist extraction should be completed on individual cells. Since this process is very time consuming, it must be automated.

Overview

Write a script that generates a netlist from all the extracted views of the cells of a given library. The script will be named: netlist_cell. It uses Analog Environment within Cadence to get the netlist for each cell. The command should be called:

```
netlist_cell <library_name>
```

At the end of the script, a directory named <library_name>_netlist is generated. The directory contains netlist files for all the cells in the library <library_name>. The netlist files is named <cell_name>.l

Here is an illustration of the command call. Assume we are trying to get the netlist of all the cells in library vtvt_tsmc250. The command to run will be:

```
netlist_cell vtvt_tsmc250
```

After this run, a directory named vtvt_tsmc250_netlist will be generated. If one of the cells in vtvt_tsmc250 was inv_1, the directory vtvt_tsmc250_netlist will contain the file inv_1.l. This file is the netlist of cell inv_1.

At the end of the run for an entire library, a summary of the run will be generated in a file named <library_name>_netlist.log.

Specifications

Assumptions

- As the command is run, it will check that an extracted view is available for a given cell. If the extracted view is missing, display the message below on the screen or in the summary file and move to the next cell.

    <cell_name> … Extracted view missing.

If the extracted view is available, continue the run for that cell view and output the message below in the summary file:

    <cell_name> … Done

- This script is only about running Analog Environment on extracted cell views of cells of a given library. If a cell does not have an extracted view, inform the user by displaying a list of cells that do not have an extracted cell view in the summary run (see Format of Summary File below for more detail on the Summary file format).

- The script must be run in a Cadence environment directory that is already set to run ICFB.

Log Directory

The netlist generated for each cell is stored in a directory named <library_name>_netlist.

The netlist generated are saved as <cell_name>.l files in directory <library_name>_netlist.

If directory <library_name>_netlist already exists, remove it and re-create it for every new run of command netlist_cell.

Log directory: <library_name>_netlist

Log file: <cell_name>.l

Library not found message

If the library entered is not found, output the message:

    Library <library_name> not found in this directory.
    Aborted Netlist Extraction Process

Format of Summary File

At the end of the run for an entire library, display a summary of the run. This summary can also be found in file: <library_name>_netlist.log. As the command is running, display the following information.

```
vtvt_tsmc250_netlist.log: netlist_cell vtvt_tsmc250 on January 19, 2007 4:05pm



Running netlist_cell for library vtvt_tsmc250…

inv_1…Done

inv_2…Done

inv_4…Done

buf_1...Done

buf_2....Done

and2_1...Done

mux2_1…No extracted view

mux2_2…No extracted view

xor2_1
```

N. Appendix: NETLIST_LIBRARY.IL SCRIPT

This appendix contains the content of the Netlist extraction script "netlist_library.il"

```
;;##########################################################################
########
;;##  Filename : netlist_library.il
;;##
;;##  Version  1.0
;;##  Revision 031407-5
;;##
;;##  Description : Runs netlisting on an entire library
;;##  Usage :
;;##    netlist_library( "libName" )
;;##
;;##  Author :
;;##    Christopher Wilson (wilchr@vt.edu)/Jeannette Djigbenou
;;##                   (jddjig01@vt.edu)
;;##  Department :
;;##    Bradley Department of Electrical and Computer Engineering's
Virginia
;;##    Tech VLSI for Telecommunications Lab
;;##
;;##########################################################################
########

procedure( netlist_library( LibName )

;;>> This script will run netlisting on an entire library.
;;>> syntax:
;;>> load "netlist_library.il"
;;>> netlist_library( "LibName" )


prog( ( currDir Path Port pattern logFile filelist sessionName formName
log ) ;;local vars

;;>>>>>>>>>>> initialize variables <<<<<<<<<<<<<<<
currDir = getWorkingDir()
Path=nil

;;>> check for the presence of cds.lib in the current directory
if( not( isFile( concat( currDir "/cds.lib" ) ) )
then
      printf("ERROR: cds.lib was not found within the current working
directory.\n")
      return( nil )
)

;;>> look in cds.lib for path definition of specified library
Port = infile( concat( currDir "/cds.lib" ) )
```

```
while( gets( Line Port )
      rexCompile( strcat( LibName '"[ \t]+" ) )
      if( rexExecute( Line )
      then
            pattern = strcat( '"DEFINE[ \t]*" LibName '"[ \t]*" )
            rexCompile( pattern )
            Path = rexReplace( Line "" 1 )
            rexCompile( "\n" )
            Path = rexReplace( Path "" 0 )
      )
)
close( Port )

;;>> if we get to here and the path variable is still nil, then the
library is not defined in the cds.lib
if( not( Path )
      then
            printf("ERROR: The library was not found in cds.lib.\n")
            return( nil )
)

;;>> check for the existence of the specified library
if( not( isDir( Path ))
      then
            printf( "Library not found.  Aborted netlisting process." )
            return( nil )
)


;;>>>>>>>>>>>>>>>>>>> Library found: Set up the proper files and
directories

;;>> clear out old log files and create the proper directory
  sh( strcat( "rm -rf " currDir "/" LibName "_netlist" ) )
  sh( strcat( "mkdir " LibName "_netlist" ) )

;;>>Set the currDir and logFile directories
  currDir=strcat( currDir "/" LibName "_netlist" )
  logFile=strcat( currDir "/" LibName "_netlist.log" )

;;>>Create the logfile and open a port for output...
  sh( strcat( "echo  > " logFile ) )
  log=outfile(logFile)

;;>> Create the missing cells log (temp file)
  sh( "echo Cells missing extracted views: > missing" )

;;>> Initialize the logfile
  fprintf( log strcat( LibName "_netlist.log: netlist_library " LibName "
on " ) )
  close(log) ;;we must close the log file to allow access by the shell
  sh( strcat( "date >> " logFile ) )
  log=outfile(logFile) ;; reopen the file... this may be inefficient and
can be modified later
```

```
  fprintf( log strcat( "\n Running netlist_library for library: " LibName
"\n" ) ) ;;log file
  printf( strcat( "Running netlist_library for library: " LibName "\n" ) )

;;>> create an array of all the cells in the library
  filelist=sort( getDirFiles( Path ) nil )

;;>> attempt to perform netlist on all cells - set up the sevSession for
each cell and run
  foreach( cellName filelist

      ;;WE SHOULD MAKE SURE THAT (. .. prop.xx etc ) are not added to the
testing.
      if( cellName != "prop.xx" && cellName != "." && cellName != ".." &&
cellName != "cdsinfo.tag"
      then

       fprintf( log strcat( cellName "..." ) )  ;;log file
       printf( strcat( cellName "..." ) )       ;;CIW

        if( isDir( strcat( Path "/" cellName "/extracted/" ) )
        then
           ;;set up the session and perform the netlist
          sessionName=sevStartSession( ?lib LibName ?cell cellName ?view
"extracted" )
          envSetVal( "hspiceS1.envOpts" "switchViewList" 'string
"extracted hspiceS spice cmos_sch cmos.sch schematic" )
          envSetVal( "hspiceS1.envOpts" "stopViewList" 'string "ivpcell
hspiceS spice" )
          envSetVal( "hspiceS1.envOpts" "includeSyntax" 'string "hspice" )
              sevNetlistFile( sessionName 'createFinal )

               ;;Clean up the windows
              hiCloseWindow( hiGetCurrentWindow() )
              hiCloseWindow( hiGetCurrentWindow() )

               ;;Update the log / CIW
              fprintf( log "Done\n" )
              printf( "Done\n" )

               ;;Move created netlist to the working directory
              sh( strcat( "cp ~/cadence/simulation/" cellName
"/hspiceS/extracted/netlist/hspiceFinal " currDir "/" cellName
".hspice_nl" ) )

        else
        ;;>> if a cell doesn't have an extracted view - add cell name to
file
         fprintf( log "No extracted view\n" )
             printf( "No extracted view\n" )
             ;;>> Add the missing cells to the final log file...
             sh( strcat( "echo " cellName ">> missing" ) )

       );if then else
      ); if not
  );foreach

;;>> finalize log report by outputting the cell array
    fprintf( log "\n \n Completed netlisting on library.\n\n" )
    printf( "Netlisting complete, view the logfile for the results." )
    close(log)
    sh( strcat( "cat missing >> " logFile ) )
    sh( strcat( "echo >> " logFile ) )
    sh( strcat( "echo END OF FILE >> " logFile ) )


;;>> clear the simulation directory, but as a debugging technique we might
leave this on for now...
    sh( "rm missing" )
    ;;sh( "rm -rf ~/cadence" )

); prog
); procedure
```

O. Appendix: NETLIST_CLEANUP Script

This script is used to modify the netlists extracted from the layout cells prior to performing characterization. Its script is presented below.

```
#!/bin/bash
# Company:        The Bradley Department of Electrical and Computer
Engineering
#                 Virginia Tech VLSI for Telecommunications Laboratory
(VTVT Lab)
#
# Author:         Jeannette Donan Djigbenou <jddjig01@vt.edu>
#
# Creation Date: Friday, September 14, 2007
#
# Description: This script cleans up the extracted netlist files.
#
# History:
#   09/14/2007: (jddjig01) File Header Added/Updated
#

# Command line: cleanup_hspice_nl <directory_name>
answer='x'
while [ ${answer} != 'y' -a ${answer} != 'n' ] ; do
    echo ""
    echo "This script will replace VSS node by GND and remove the Block
below from all *.hspice_nl files in directory: "`pwd`
    echo "Removing Block"
    echo " .TEMP    25.0000"
    echo " .OP"
    echo " .save"
    echo " .OPTION  INGOLD=2 ARTIST=2 PSF=2"
    echo " +        PROBE=0"
    echo " .END"
    echo -n "Do you want to remove this block from this directory? (y/n) "
    read answer
    answer=`echo ${answer} | tr '[A-Z]' '[a-z]'`
done

if [ "${answer}" = 'n' ] ; then
    exit
fi

echo ""
echo "... modifying files ..."

perl -ni.bak -e "s/(\+)        PROBE=0//; print unless
/^\.[TEMP|OP|save|OPTION|END]/;" *.hspice_nl
perl -pi.bak -e "s/ 0 / GND /g" *.hspice_nl
perl -pi.bak -e "s/ 0 / GND /g" *.hspice_nl
perl -pi.bak -e "s/VDD\!/VDD/g" *.hspice_nl
```

```
echo ""
echo "... Removed Block ..."
echo "... Removing all .bak files"
rm *.bak
echo "Completed cleaning up extracted netlist script."
```

P. Appendix: STDLIB ORGANIZATION Description

Overview

This is a TCL script that automatically organizes all cells spice files for characterization in the stdlib_characterization directory.

The SHELL command should be stdlib_org.

A call to the stdlib_org is: stdlib_org <spice_directory> <libname>

This call will sort all the cells spice files in directories named <cell_name>

Here is an example:

Command stdlib_org stdlib_characterization vtvt_tsmc250

Directory stdlib_characterization contains the following cells:

```
-r--r--r--  1 vtvt vtvt  770 Jul 25  2006 inv_1_cap.sp
-r--r--r--  1 vtvt vtvt 1850 Jul 25  2006 inv_1_p_abrupt.sp
-r--r--r--  1 vtvt vtvt 2567 Jul 25  2006 inv_1_p_dynamic.sp
-r--r--r--  1 vtvt vtvt  818 Jul 25  2006 inv_1_p_static.sp
-r--r--r--  1 vtvt vtvt 2048 Jul 25  2006 inv_1_t_abrupt.sp
-r--r--r--  1 vtvt vtvt 2797 Jul 25  2006 inv_1_t_nonabrupt.sp
-r--r--r--  1 vtvt vtvt 1145 Jul 25  2006 nor2_2_cap.sp
-r--r--r--  1 vtvt vtvt 2349 Jul 25  2006 nor2_2_p_abrupt.sp
-r--r--r--  1 vtvt vtvt 3312 Jul 25  2006 nor2_2_p_dynamic.sp
-r--r--r--  1 vtvt vtvt 1246 Jul 25  2006 nor2_2_p_static.sp
-r--r--r--  1 vtvt vtvt 5124 Jul 25  2006 nor2_2_t_abrupt.sp
-r--r--r--  1 vtvt vtvt 6099 Jul 25  2006 nor2_2_t_nonabrupt.sp
```

After running the command, directory <libname>_characterization_org is generated. It contains directories inv_1 and nor2_2.

Directory inv_1 contains the inv_1*sp and directory nor2_2 contains nor2_2*sp files.

Specifications

These are the specifications to follow to generate the FINAL stdlib_org script.

Content of Input directory stdlib_characterization:

The content of the stdlib_characterization directory should be a list of spice files (*.sp files) of cells present in the standard cell library directory.

This will ensure that all cells are being characterized.

Verify that all cells from library vtvt_tsmc250 have their spice files present in the stdlib_characterization directory. If not, tell the user which files are missing in the summary after the run (see content of Log File for more details).

Library not found message

If the library is not in the current directory, output the following statement in standard output AND in the log file:

Library <library_name> not found.
Aborted verification run

Content of Directory <libName>_stdlib_characterization_org:

The generated directory from the command should contain directories named <cell_name>.

Each directory <cell_name> should contain the following extensions:

<cell_name>_cap.sp
<cell_name>_p_abrupt.sp
<cell_name>_p_dynamic.sp
<cell_name>_p_static.sp
<cell_name>_t_abrupt.sp
<cell_name>_t_nonabrupt.sp

If a file is missing, let the user know by outputting this message in the log file or on stdout:

File <cell_name>_*.sp….missing

Format of the log file

The summary of the organization should tell me how many cells have been organized in the new directory <libname>_characterization_org. It should also tell me if some files were not found for any cells from the standard cell library.

An example of log file is shown below. One contains errors and the other does not.

```
vtvt_tsmc250_stdlib_org.log: stdlib_org stdlib_characterization vtvt_tsmc250 on
January 19, 2007 4pm



inv_1…Done

inv_2…Done

inv_4…Done

buf_1...Done

buf_2....Done

and2_1...Done

xnor2_2…missing characterization files

nand2_3…missing nand2_3_cap.sp
```

Q. Appendix: STDLIB_ORG SCRIPT

This appendix presents the organization script generated in this thesis: “stdlib_org”.

```
#!/usr/bin/tclsh
#
#

set pwd [pwd]
set systemTime [clock seconds]
puts "stdlib_org $argv"
puts "on [clock format $systemTime -format %B\ %d\,\ %Y\ %I%p]\n"
set LibName "[lindex $argv 1]"
set SpDir "[lindex $argv 0]"

set log_file [open "$LibName\_stdlib_org.log" w+]
puts $log_file "stdlib_org $argv"
puts $log_file "on [clock format $systemTime -format %B\ %d\,\ %Y\
%I%p]\n"

if {[file exists $LibName]} {
file delete -force $LibName\_characterization_org
file mkdir $LibName\_characterization_org

cd ./$LibName
exec ls > list
file rename -force list ..
cd $pwd


set original_count_of_cells 0
set count_of_completed_cells 0

set list_file [open "list" r]
gets $list_file line
while {![eof $list_file]} {
    if {$line != "list"} {
       if {$line != "cdsinfo.tag"} {
            if {$line != "prop.xx"} {
              incr original_count_of_cells
              file mkdir $LibName\_characterization_org/$line
              cd ./$SpDir

              set contents [glob -nocomplain $line*]
              foreach item $contents {
                  file copy $item ../$LibName\_characterization_org/$line/
              }

              cd $pwd
              cd $LibName\_characterization_org/$line
              set done_test 0
              set cap 0
set p_abrupt 0
set p_dynamic 0
set p_static 0
set t_abrupt 0
set t_nonabrupt 0
if {[file exists $line\_cap.sp]} {
    incr done_test
    incr cap
}
if {[file exists $line\_p_abrupt.sp]} {
    incr done_test
    incr p_abrupt
}
if {[file exists $line\_p_dynamic.sp]} {
    incr done_test
    incr p_dynamic
}
if {[file exists $line\_p_static.sp]} {
    incr done_test
    incr p_static
}
if {[file exists $line\_t_abrupt.sp]} {
    incr done_test
    incr t_abrupt
}
if {[file exists $line\_t_nonabrupt.sp]} {
    incr done_test
    incr t_nonabrupt
}
if {$done_test == 6} {
    incr count_of_completed_cells
    puts $log_file "$line\...Done"
}
if {$done_test == 0} {
    puts "$line\...missing characterization files"
    puts $log_file "$line\...missing characterization files"
} elseif {$cap == 0} {
    puts "$line\...missing $line\_cap.sp"
    puts $log_file "$line\...missing $line\_cap.sp"
} elseif {$p_abrupt == 0} {
    puts "$line\...missing $line\_p_abrupt.sp"
    puts $log_file "$line\...missing $line\_p_abrupt.sp"
} elseif {$p_dynamic == 0} {
    puts "$line\...missing $line\_p_dynamic.sp"
    puts $log_file "$line\...missing $line\_p_dynamic.sp"
} elseif {$p_static == 0} {
    puts "$line\...missing $line\_p_static.sp"
    puts $log_file "$line\...missing $line\_p_static.sp"
} elseif {$t_abrupt == 0} {
    puts "$line\...missing $line\_t_abrupt.sp"
    puts $log_file "$line\...missing $line\_t_abrupt.sp"
} elseif {$t_nonabrupt == 0} {
    puts "$line\...missing $line\_t_nonabrupt.sp"
    puts $log_file "$line\...missing $line\_t_nonabrupt.sp"
```

```
             }
           }
       }
      cd $pwd
     }
    gets $list_file line
}

close $list_file

puts ""
puts "$count_of_completed_cells out of $original_count_of_cells cells have
been organized for characterization."

puts $log_file ""
puts $log_file "$count_of_completed_cells\ out of
$original_count_of_cells\  cells have been organized for
characterization."
puts $log_file "See directory $LibName\_characterization_org for more
details."
puts $log_file ""
puts $log_file "End of File"

file delete list

} else {
puts "Missing Library $LibName"
puts "Aborted stdlib_org $SpDir $LibName"
}
```

R. Appendix: HSPICE_TECH Description

Overview

This is a TCL script that automatically runs HSPICE on all files in a directory structure.

The command is hspice_tech <directory name>

After running this command, the results of the Hspice simulation should be kept in the directory of each cell name.

Here is an example.

Directory vtvt_tsmc250_characterization_org contains sub-directories like

inv_1

nor2_2

…

Each subdirectory contains cells in the following format.

<cell_name>_cap.sp

<cell_name>_p_abrupt.sp

<cell_name>_p_dynamic.sp

<cell_name>_p_static.sp

<cell_name>_t_abrupt.sp

<cell_name>_t_nonabrupt.sp

The goal of the command would be to run Hspice on all the sub-directories *.sp files.

Specifications

These are the specifications to follow to generate the FINAL hspice_tech script.

Content of the Directory

The directory structure of the input to the command has been described above.

<input directory name>

<cell name>

*.sp files

Run Outputs

After running Hspice for the entire <input directory>, the results files should be stored in the corresponding <cell_name> directory which includes all the *.sp files.

Output Files Naming Convention

The output files from Hspice runs should be <file_name_without_sp_extension>.results.
Follow the Hspice tutorial on the vtvt.org site at
(http://www.vtvt.ece.vt.edu/vlsidesign/tutorialHspiceIndex.php)
To know how to generate spice files and results.

Failed Run Situation

If a run fails, let the user know in the output log file and on stdout.

Location of Help Directories

You may find a sample of the spice directory with Brad. He is implementing this new structure right now…He should be finished with at least half of the cells. Ask him about a manual representation of a few sub-directories within stdlib_characterization_org. He should be able to provide it for you, based on the task he got this week as well!!

Missing Input Directory

If the input directory is missing, inform the user and abort the run

```
Directory <input directory> is missing.
Aborting hspice_tech run.
```

Simulation Process

This is the process for running hspice on all the files
Verify that the input directory is here. If it's not, let the user know and abort the run
If the directory exists, move to each subdirectory.
Foreach subdirectory present in the input directory, run all hspice runs. If you generate all *.results files for that specific sub-directory, output

```
<cell_name>….Done
```

and continue the run.

If you encounter Hspice run errors, let the user know which file had errors.

<cell_name>….error in <file_name>

and continue the run

At the end of the run, let the user know that the runs are completed and generate a *.log file in the run directory.

Format of the log file

The log file: <input directory>_spice_results.log and standard output should follow the following format.

Introductory Title line. (Command entered)

Run Status for each cell (with a list of cells)

Write a Run Summary

End of Summary and Date run.

An example of log file is shown below.

```
vtvt_tsmc250_characterization_org _spice_results.log: hspice_tech
vtvt_tsmc250_characterization_org  on January 19, 2007 4pm



Running hspice_tech for directory vtvt_tsmc250_characterization_org

inv_1…Done

inv_2…Done

inv_4…Done

buf_1....error in buf_1_cap.sp

buf_2....error in buf_2__p_abrupt.sp

.

.
```

S. Appendix: HSPICE_TECH SCRIPT

This appendix contains the HSpice automation script “hspice_tech”.

```
#!/usr/bin/tclsh
#
# Company:       The Bradley Department of Electrical and Computer
Engineering
#                Virginia Tech VLSI for Telecommunications Laboratory
(VTVT Lab)
#
# Author:        Nathan Kees <nkees@vt.edu>/
#                Jeannette Djigbenou <jddjig01@vt.edu>
#
# Creation Date: Wednesday, March 14, 2007
#
# Description: This script runs HSpice for all .sp files within a
directory
#

set user [exec whoami]
puts "\nWelcome $user\n"
set errnum ""

proc getcells {dir log} {
	cd $dir

	set errstatus [catch {set allcells [glob *]}]
	set badcells {}
	set cellcount 0

	if {$errstatus == 0} {
		foreach sub $allcells {
			if {[file isdirectory $sub] == 1} {

				set cellstatus [catch "hs $sub $log"]
				incr cellcount
				cd ..

				if {$cellstatus == 1} {
					lappend badcells $sub
				}
			}
		}

		puts "\nHSpice was run for $cellcount cells."
		puts $log "\nHSpice was run for $cellcount cells."

		if {[llength $badcells] == 0} {
			puts "All cells completed HSpice successfully."
			puts $log "All cells completed HSpice successfully."
```

```
            } else {
                  puts "The following cells did not complete HSpice
successfully."
                  puts "See above for details.\n"

                  puts $log "The following cells did not complete HSpice
successfully."
                  puts $log "See above for details.\n"

                  foreach sub $badcells {
                        puts "\t$sub"
                        puts $log "\t$sub"
                  }
            }

      } else {
            puts "$dir has no sub-directories"
            puts $log "$dir does not contain any sub-directories"
      }
}

proc hs {cell log} {
      cd $cell

      set misslist [list _cap _p_abrupt _p_dynamic _p_static _t_abrupt
_t_nonabrupt]
      set errorlist {}
      set successlist {}

      set errstatus [catch {set spfiles [glob *.sp]}]

      if {$errstatus == 0} {
            foreach sub $spfiles {
                  regexp ($cell)(.*)(\.sp) $sub x cell ext

                  catch {eval exec "hspice $cell$ext.sp >
$cell$ext.results"} hsinfo
                  # the output of the hspice command is contained in hsinfo

                  if {[string match *concluded* $hsinfo]} {

                        # HSpice completed
                        lappend successlist $ext
                        regsub ($ext) $misslist "" misslist

                  } else {

                        # HSpice aborted
                        lappend errorlist $ext
                        regsub ($ext) $misslist "" misslist

                  }

            }
```

```
	}

	if {[llength $errorlist] == 0 && [llength $misslist] == 0} {

		puts "\nHSpice succeeded for $cell"
		puts $log "\nHSpice succeeded for $cell"

	} elseif {[llength $errorlist] == 0} {

		puts "\nHSpice is missing files for $cell"
		puts $log "\nHSpice is missing files for $cell"

	} else {

		puts "\nHSpice failed for $cell"
		puts $log "\nHSpice failed for $cell"

	}

	foreach errorext $errorlist {
		puts "\t$errorext aborted"
		puts $log "\t$errorext aborted"
	}

	foreach missext $misslist {
		puts "\t$missext was missing"
		puts $log "\t$missext was missing"
	}

	foreach successext $successlist {
		puts "\t$successext succeeded"
		puts $log "\t$successext succeeded"
	}

	if {[llength $errorlist] != 0 || [llength $misslist] != 0} {
		error "cell failed"
		# Error gets passed back to getcells and is handled there
	}

}

if {[file isdirectory $argv] == 1} {
	set logfile [open "spice_results.log" w 0666]

	puts "Running HSpice for $argv..."
	puts "This could take a while...\n"

	set starttime [clock seconds]

	puts $logfile "Command: hspice_tech $argv"
	cd $argv
	puts $logfile "Running HSpice for [pwd] on [clock format $starttime
-format "%B %d, %Y, %r"]\n"
```

```
      cd ..

      getcells $argv $logfile
      set endtime [clock seconds]
      puts $logfile "\n...Execution completed on [clock format $endtime -
format "%B %d, %Y, %r"]"
      puts $logfile "Total run time = [clock format [expr {$endtime -
$starttime}] -gmt 1 -format "%T"]\n"

} else {
      puts "ERROR: \"$argv\" is not a directory."
      puts "HSpice run aborted.\n"
      exit
}

puts "\n...Execution complete\n"

close $logfile
```

T. Appendix: MAKECELL SCRIPT

This appendix contains the script "makecell". This script creates cell files of standard cell libraries from a directory structure that contains all cfg_files and from a directory that contains all characterization results. In this thesis, the characterization procedure has not been elaborated. The characterization approach for our cell libraries has been detailed by Dr. Jos Sulistyo [7]. This script requires the use of cellgen scripts developed by Dr. Jos Sulistyo.

```
#!/usr/bin/perl
# Company:       The Bradley Department of Electrical and Computer
Engineering
#                Virginia Tech VLSI for Telecommunications Laboratory
(VTVT Lab)
#
# Author:        Jeannette Donan Djigbenou <jddjig01@vt.edu>
#
# Creation Date: Sunday, September 16, 2007
#
# Description: This script creates the cell files of standard cells from
#           a directory that contains all cfg_files and from a directory
#           that contains all characterization results
#
# History:
#   09/16/2007: (jddjig01) File Header Added/Updated
#
# USAGE: make_cell

print "Welcome to the make_cell script\n";
print "This script generates the Synopsys Liberty Format file for the cfg
files from the cfg_dir you entered in the script \n";
print "USAGE>> make_cell\n\n";


# Insert all these data to a file for this configuration info (EVENTUALLY)
$rundir = "/home/jddjig01/v/gen_cellgen/vtvt_tsmc180_cellgen"; # Update
appropriately this entire section
$spice_char_dir = "$rundir/vtvt_tsmc180_characterization";
$cfg_files_dir = "$rundir/vtvt_tsmc180_cfg_files";
$cellgen_cells = "$rundir/vtvt_tsmc180_cellgen_cells";
$cellgen_script = "$rundir/./cellgen";

# Get the list of cells in the characterization directory
# And remove the dots in the directory list
opendir(CHAR_RESULTS_DIR, $spice_char_dir) or die "Can't read this
directory $spice_char_dir \n";
@cells_list = sort readdir CHAR_RESULTS_DIR;
shift(@cells_list);
shift(@cells_list);
closedir(CHAR_RESULTS_DIR);
print "CELLS : @cells_list\n";


# Run Cellgen on all cells in the characterization directory
# Generate all $cell_name.cell files in directory cellgen_files directory
chdir($spice_char_dir);
foreach $cell_name (@cells_list) {
    if (("$cell_name" ne "hspice_extracted_netlist") and ("$cell_name" ne
"stdlib_netlists") and ("$cell_name" ne "filler")) {
      if (defined opendir(CELL_DIR, $cell_name)) {
          print "WORKING on $cell_name\n";
          chdir($cell_name);
          print "running:perl $cellgen_script
$cfg_files_dir/$cell_name.cfg\n";
          @cellgen_file = `perl $cellgen_script
$cfg_files_dir/$cell_name.cfg`;
          $output_cellfile = "$cellgen_cells/$cell_name.cell";
          open(CELLGEN_OUT,">$output_cellfile") or die "Can't open this
file $output_cellfile\n";
          print "CELLGEN_CELL FILE $output_cellfile for
$cell_name\n\n@cellgen_file";
          print CELLGEN_OUT "@cellgen_file\n";
          close(CELLGEN_OUT);
          chdir($spice_char_dir);
      }
    }
}
print "You are done\n";
```

U. Appendix: DESIGN_VISION_TCL DESCRIPTION

I. Introduction

This script will automatically run a list of vhdl files (*.vhd) from an input directory, analyze, elaborate and compile each of the design. The output of this script is a directory of corresponding verilog synthesized files for each of the vhdl files synthesized. Also this script can run for a single vhdl file and the output is a single synthesized verilog file. Refer to the vtvt website tutorial on how to use Design Vision for logic Synthesis manually if you are interested.

II. Usage

If you are using the standard cell libraries release package, start Synopsys using the Synopsys_Libraries directory of the release package. For directions on how to start Synopsys with the release package, please refer to the Synopsys Installation tutorial.

Create subdirectory named WORK under your current directory (if not already done). All intermediate files will be stored under the directory.

The usage of this script:

Step 1: Type "**Synopsys**" at a UNIX prompt to start your Synopsis Environment.

Step 2: Type "**dc_shell-t**" at the Synopsis prompt. Design Vision will start in tcl mode.

Step 3: Type "**source design_vision_vtvt.tcl**" to run the script.

The script will ask you to select running this script for a directory or a single vhdl design and you need input **the path of the directory** or **the name of the single .vhd file**.

III. Output

Assume you need to run design_vision_vtvt.tcl for directory "Mydir". This directory should contain vhdl files. At the end of the run, directory: "Mydir_synthesized" should be created. That directory contains all the verilog files from the synthesized VHDL files. The log file should be created in directory "Mydir_synthesized".

If you need to run design_vision_vtvt.tcl for a single file "Mydesign.vhd", this file should lie in the directory where you are now. At the end of the run, the verilog file "Mydesign.v" should be created. The log file should be created in directory "Mydesign _design_vision_vtvt.log".

There are two statuses for the log output: PASS, FAIL. The example of logfile is shown below.

```
vtvt_tsmc250_design_vision_vtvt.log: Design Vision report for directory vtvt_tsmc250 on
October 29, 2007 03:59 PM



ABnorC_TC.vhd...Pass

ABorCorD_TC.vhd...Pass

ABorC_TC.vhd...Pass

and2_1_TC.vhd...Fail

and2_2_TC.vhd...Pass
```

V. Appendix: SOC_ENCOUNTER_VTVT DESCRIPTION

Introduction

The perl script "soc_encounter_vtvt.pl" automatically runs a list of vhdl files from an input directory, or a single vhdl file in SOC Encounter, exports the resulting layout into Cadence ICFB. Later it runs Cadence and performs DRC and LVS on the extracted layout generated as well as the schematic of the design.

Possible ways to run the are:

a. ***soc_encounter_vtvt <path to the directory of VHDL files>***
b. ***soc_encounter_vtvt <path to single or multiple VHDL files>***

The above commands should be issued from the directory containing the script, however, the directory of the VHDL files can be at any other location on the same computer. Also, note that the script assumes that the name of top cell in the VHDL files is the same as the name of the VHDL file itself.

Setup

The script uses a couple of files to perform all the above operations. There are three categories of files required to run the script"

a. Skill scripts

There are two Skill scripts, which were written separately by other cell library developers. Therefore description of these scripts is not provided here. However, use of soc_encounter_vtvt.pl doesn't require knowledge of the functionality of these scipts. There are two skill scripts, namely:

1. alllib_drc.il – Runs DRC
2. alllib_lvs.il – Runs LVS

Note that these two scripts should be present in the same directory as "soc_encounter_vtvt.pl" script itself.

b. Cadence Template Files

All these files should be present in the same directory as "soc_encounter_vtvt.pl" script itself. These files contain a set of Cadence commands, executed to perform a particular task in cadence. For each VHDL file processed by the soc_encounter_vtvt,pl script, a copy of the command file is generated with a name as <top_cell_name>_encounter.cmd, <top_cell_name>_import.cmd etc, corresponding to each task. The template files contain identifiers which are modified to customize the template for the VHDL cell being processed. These identifiers are marked by a pattern of exclamation marks, before and after a string and their meaning can be easily understood from the string contained within exclamation marks.

The list of files and their function is given below:

1. "template_encounter.cmd" – This script runs SOC Encouter and generates GDS file output
2. "template_cadence_import.cmd" – It imports the generated GDS file into cadence
3. "template_cadence_drc.cmd" – Runs design rule check
4. "template_cadence_lvs.cmd" – Runs LVS

c. Cell Library Related Files

The tasks performed by "soc_encounter_vtvt.pl" script depend upon some files of the cell library itself, these files are:

1. vtvt_SocE2df2.map
2. vtvt_tsmc250.lef
3. vtvt_tsmc250.lib
4. vtvt_tsmc250_StreamIn.map
5. Cell library – "vtvt_tsmc250"
6. Cell library – "vtvt_tsmc250_nolabel"

Note that all these files should be present in the same directory as "soc_encounter_vtvt.pl" script itself.

Output

The script should generate a logfile named: <directory_soc_encounter_vtvt.log> in the directory where it's been run. The logfile has the following format and contains information mentioned below.

```
vtvt_tsmc250_soc_encounter_vtvt.log: PNR to Cadence ICFB report for directory
vtvt_tsmc250 on January 19, 2007 4:05pm



Running soc_encounter_vtvt for directory vtvt_tsmc250…

Evaluating inv_1.vhdl

            PNR … Pass

            Import… Pass

            Verification …Pass



Evaluating inv_2.vhdl

            PNR … Pass

            Import… Pass

            Verification … Pass

Evaluating mux2_1.vhdl
```

There are three statuses for the log output: PASS, FAIL, ABORTED. A step passes if it is successfully run by the tool and output the expected results. Else, it fails. Once a step fails, all subsequent steps for the VHDL file are aborted. Other than the main log file described above, there are a couple of other log files, which are generated by Skill scripts, which contain the run specific information for each VHDL design processed.

Publications

J. Djigbenou, and D.S. Ha, "Development and Distribution of TSMC 0.25*u*m Standard CMOS Library Cells, "*International Conference on Microelectronic Systems Education*, pp. 27-28, June 2007.

J. Djigbenou, T.V. Nguyen, C.W. Ren, and D.S. Ha, "Development of TSMC 0.25*u*m Standard Cell Library," *IEEE Southeast Conference*, pp. 566-568, March 2007.

Bibliography

1. **Petley, Graham.** Standard Cells, Open Source. *ASIC Standard Cell Libraries Design by Graham Petley.* [Online] January 12, 2008. [Cited: March 25, 2008.] http://vlsitechnology.org/html/libraries05.html.

2. **MOSIS.** SCMOS Design Kits. *MOSIS.* [Online] [Cited: March 25, 2008.] http://www.mosis.com/design/flows/design-flow-scmos-kits.html.

3. **Harris, Neil Weste David.** *CMOS VLSI Design A Circuit and Systems Perspective.* New York : Addison Wesley, 2005. p. 547. 0-321-14901-7.

4. **NCSU_CDK.** NCSU Cadence Design Kit. *NCSU CDK - NCSU EDA Wiki.* [Online] July 12, 2007. [Cited: March 26, 2008.] http://www.eda.ncsu.edu/wiki/NCSU_CDK.

5. **Taiwan Semiconductor Manufacturing Company.** TSMC 0.18 and 0.15-micron Technologies Platform Brochure. *Taiwan Semiconductor Manufacturing Company.* [Online] April 2005. [Cited: March 26, 2008.] http://www.tsmc.com/download/english/a05_literature/0.15-0.18-micron_Brochure.pdf.

6. **MOSIS.** TSMC 0.18 Micron Process. *MOSIS.* [Online] [Cited: March 26, 2008.] http://www.mosis.com/products/fab/vendors/tsmc/tsmc018/.

7. **Sulistyo, Jos Budi.** *On the Characterization of Library Cells.* [Master's Thesis] Blacksburg : Virginia Tech, 2000.

8. **Dr. Dong S. Ha.** VTVT: VLSI Design Overview. [Online] November 24, 2006. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/index.php.

9. **Djigbenou, Jeannette.** Design Vision: A Logic Synthesis Tool. *VTVT- VLSI Design: Synopsys Tutorials.* [Online] March 26, 2008. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialSynopsys_logicSynth.php.

10. —. Place And Route Using Cadence SOC Encounter. *VTVT - VLSI Design: Cadence Tutorials.* [Online] March 26, 2008. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_socEncounter.php.

11. —. Import Synthesized Design Into Cadence Virtuoso Layout View. *VTVT - VLSI: Cadence Tutorials.* [Online] March 26, 2008. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_importGDS.php.

12. —. Import Synthesized Design Into Cadence Composer Schematic View. *VTVT - VLSI: Cadence Tutorials.* [Online] March 26, 2008. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_importVerilog.php.

13. —. Design Rule Check (DRC). *VTVT - VLSI: Cadence Tutorials.* [Online] March 26, 2008. [Cited: April 10, 1008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_DRC.php.

14. —. LVS (Layout-Versus-Schematic) with Virtuoso. *VTVT - VLSI: Cadence Tutorials.* [Online] March 26, 2008. [Cited: April 10, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_lvs.php.

15. —. Pad Insertion. *VTVT- VLSI Design: Cadence Tutorials.* [Online] VTVT, March 26, 2008. [Cited: April 14, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_pad_insertion.php.

16. **Aust, Carrie.** Steps for Chip Submission. *VTVT - VLSI Design.* [Online] VTVT, March 26, 2008. [Cited: April 14, 2008.] http://www.vtvt.ece.vt.edu/vlsidesign/tutorialCadence_submission.php.