AutoLibGen: An Open Source Tool for Standard Cell Library Characterization at 65nm Technology

Rachit I. K. and M. S. Bhat
Dept. of Electronics and Communication Engineering
NITK Surathkal, Karnataka, INDIA.
Email: rachit.ik@gmail.com, msbhat@ieee.org

Abstract

In this paper, we present the development of an open source tool, ***AutoLibGen****, for characterising a standard cell library comprising of basic combinational circuits. The cells are initially laid out and the parasitic netlists are extracted. Unlike the traditional method of computing timing and power data using non linear delay and power models we use more accurate Composite Current Source (CCS) based characterization for very deep sub-micron technologies. We tested our tool with a library for 65nm. The library file generated by our tool was successfully compiled by Synopsys Library Compiler and is used to synthesize a Verilog code using Synopsys Design Compiler.*

1. Introduction

In the semi-custom design flow, the designs are built from a library which comprises of the basic circuits, called cells, which are used in the synthesis process to build a larger circuit. The library has to be characterized well to aid the process of synthesis and to get a reasonable idea of the post layout performance. In the ASIC design flow, the synthesis tool prepares a circuit netlist of the RTL code using the components present in the standard cell library. The timing and optimization constraints set during synthesis are used to pick the appropriate cell for a particular gate. Thus the standard cell library forms the basic building block of the circuit. The performance of individual cells in the library have to be robust enough to ensure satisfactory performance of the overall circuit. Further, if well characterized, it can give an idea of the post layout performance of the circuit just after the stage of synthesis. A library is generally represented in the Synopsys Liberty (.lib) format [1], [13]. It contains the characterization data in the form of look-up tables for different values of input slew and output capacitance. The process of Library creation can be summarized as follows.

1) Layouts of the cells are prepared, their netlists extracted.
2) The netlists are simulated for different values of input slew, output load and various other operating conditions.
3) The simulation output is read in. Processing and validation checks are performed using this data.
4) The processed data is written into a text file in the .lib format.

The proposed tool aids the process of characterization from stage 2. It comprises of Tcl scripts, which build simulation scripts for the parasitic annotated netlists of the cells in the library. A shell script then invokes Synopsys HSpice and executes the simulation scripts. The output data files generated after simulation are parsed by C++ programs which extract all the relevant information. The programs then process the data, validate it, and finally writes them into a text file in the *.lib* format (Synopsys library format). Traditionally, characterization data such as delays, transition times and power for different values of input slew and output load for each standard cell are stored in the form of look-up tables. These are called Non Linear Delay models (NLDM) and Non Linear Power Models (NLPM). But these are found to be inaccurate for technologies below 130 nm. Hence, we have used the Synopsys Composite Current Source characterization methodology. The characterization method follows the guidelines put down by Synopsys for CCS based characterization and performs validation checks on the characterized data. We have also included the Non Linear Delay Models in our library, as it is needed to annotate the timing data to the VITAL[1] libraries (VHDL Simulation libraries) that will be generated after compilation of the library [12]-[16].

[1] VHDL Initiative Towards ASIC Libraries

978-1-4244-2315-6/08/$25.00 ©2008 IEEE.

Rest of the paper is organized as follows. Section 2 describes files / formats required for the automation of the simulations using scripts. Section 3 describes the actual automation process using script files. Section 4 mentions the processing done on the simulation output data before they are written into the library. Section 5 describes the procedure of writing the processed data onto the library file and preparing the file for synthesis. Section 6 shows the experimental results.

2. SIMULATION SCRIPTS

In this section, we present how to incorporate the CCS characterization guidelines into HSpice and the process of automation of HSpice command generation using Tcl scripts.

A. Inputs

The tool takes as input the files containing the extracted netlists of the cells and generates a spice netlist which comprises of the excitation sources, analyses statements and data recording statements required for characterization of the cell. The files need to follow a nomenclature standard so that the software can recognize the gate and write an appropriate spice command file. We suggest the following nomenclature be followed. The files be named as
"[gate][no. of inputs] [drivng strenths (x1, x2 etc)].sp" e.g. if the cell is a 3 input AND gate with a drive strength of x4, it will be named as "and3_x4.spi". The names to be used for different types of gates is as shown in table I.

Table I. File Nomenclature

Gate	Name
NOT	inv_x*.sp
BUFFER	buf_x*.sp
NAND	nand*_ x*.sp
NOR	nor*_ x*.sp
AND	and*_ x*.sp
OR	or*_ x*.sp
XOR	xor*_ x*.sp
XNOR	xnor*_ x*.sp

B. Script structure

The characterization task for each cell is divided into three spice files as given below.

File 1: Dynamic simulations and Noise: This file performs simulations for computation of dynamic current waveforms for timing and power, along with computation of CCS Noise DC Current tables [8]. It also records the values of delays and transition times for building the Non Linear Delay Model. This file is named as *gate_name_char.sp*. e.g., and2_x1_char.sp.

File2: Noise and Leakage currents: This file performs simulations for recording the leakage currents at the gate inputs and at the power and ground pins. Also, it records the parameters for calculation of the miller capacitances. This file is named as *gate_name_noise.sp*. e.g., and2_x1_noise.sp.

File3: Intrinsic parasitics and Noise: This file performs simulations for computation of the intrinsic parasitics of the power and ground pins in additions to the simulations performed for building the CCS Noise stage output voltage table. This file is named *gate_name_lkg.sp*. e.g., and2_x1_lkg.sp.

3. AUTOMATION

The process of automation of HSpice script generation is done by Tcl scripts. Two things need to be known for automatically generating spice commands: the functionality of the cell and the number of inputs. Both these parameters can be obtained from the name of the parasitic annotated netlist file. Tcl procedures go through the netlist files and from the file names extract the functionality and the number of inputs. Using this information, input patterns are generated, which will be used to put the circuit into all the possible states required for simulation. Each input vector in the set of input patterns is such that one input pin has either a falling or rising input and the other inputs are held at a non controlling value. Both types of transitions on all input pins are covered in the input pattern. Library characterization guidelines advise not to have multiple input transitions simultaneously. The characterization bounds for input slew and output load are also computed [2].

For computing the minimum transition time, the largest inverter cell in the library is made to drive the smallest inverter and the output transition time is recorded. For the sake of simplicity, we compute the min. transition time as the output transition time, t_o, of the smallest inverter in the library, when driven by an input slew of a minimal value, say 1ns. The min. input transition time, t_{min}, is then taken as t_o. A list of values of input slew for which the cells will be characterized is then formed as [t_{min}, $5t_{min}$, $25t_{min}$, $100t_{min}$]. The maximum transition time is then decided as 50 times the min. input transition [16]. The min. output capacitance, C_{min}, is the input capacitance of the smallest inverter and the max. output capacitance, C_{max}, is the larger of 15 times the min. capacitance and four times the largest input capacitance. The list of output loads for characterization is then created as [C_{min}, $2.5C_{min}$, $5C_{min}$, C_{max}].

A. File1: Dynamic simulations and noise (* char.sp)

Once, the input pattern is generated, the process of writing the spice commands begin. A new file is opened for writing with the name as *gate_name_char.sp.*

First, the simulator settings are written down into the file. The simulator settings are such that a good trade-off between speed and accuracy is achieved. The input transition time, t_d and the output capacitance, C_x are defined as parameters, the values of which will be given at the end of the file. In an actual design, the waveform seen at a cell's input pin arrives from another cell (of the library) through a complex RC network. All of the input waveforms along an actual design path will differ from the pre-set stimulus used for characterization (in the library). Therefore it is necessary to create a stimulus waveform that minimizes the error caused due to this difference. The CCS characterization guidelines [2] recommend the use of what is called the Synopsys pre-driver waveform. It tries to place the introduced error in between the two extremes, viz.,

- Fast input slew with no RC network effect
- Slow input slew with significant RC network effect.

The pre-driver waveform can be analytically shown as:

$$vramp(t) = V\,dd*(t- t_{start})$$
$$vstep(t) = V\,dd* (1- e^{-(t-t_{start})}/RC)$$
$$\mathbf{vpredriver(t) = 0.5\ (vramp(t) + vstep(t))}$$

The commands for generating pre-driver input wave are then written into the file.

The parasitic annotated netlist of the cell is then defined as a subcircuit and these subcircuits are then instantiated, once for each vector in the input pattern.Each subcircuit has its own excitation sources, output capacitance, Vdd and Gnd pins. Having separate power pins for each subcircuit ensures accuracy of measurement of current through these pins. *Gate_name_char.sp* spice file performs simulations for computing the dynamic currents at the input, output and power pins. The dynamic currents at the output are recorded to build what is called the Driver Model [2] for CCS Timing. The input currents are stored to calculate the values of two capacitances for the CCS Timing Receiver Model [2]. The simulations for the computation of the noise dc current tables are also performed in this file. A DC voltage source is swept from 0 to 1.1Vdd in steps of 0.1Vdd at the input of the cell the current at the output of the cell is recorded for each simulation point.

For *File1*, there are *2n* subcircuits for transient analysis and *n* subcircuits for dc analysis, hence the total computational complexity can be given as $O_{trans}(n) + O_{dc}(n)$, where $O_{trans}()$ and $O_{dc}()$ indicate the order of complexity for a transient analysis and dc analysis respectively.

B. File2: Noise and leakage currents(* noise.sp)

In this file, simulations are performed for recording the leakage currents at the gate inputs and at the power and ground pins. For measuring the leakage currents, the input is held constant for sometime to record the steady state current values. Gate leakage currents are measured for cases where the input is high as well as low. Power pin leakage currents are measured for two cases, when the output is high and when the output is low. The procedure given in [8] is adopted for performing simulations to get the miller capacitances between the output pin and each of the input pins for storing in the library file for noise analysis. The total number of subcircuits (is a function of the number of inputs) in this file is given by $2n + 2$. Therefore, the total computational complexity of this file can be considered as $O_{trans}(n)$.

C. File3: Intrinsic parasitics and Noise(* lkg.sp)

This file aids the computation of intrinsic parasitics and the CCS noise stage output voltage tables. For computation of intrinsic resistances at the power and ground pins, a dc source having a voltage of around 0.1Vdd is applied at these pins and the current through this source for both the output states is calculated [5]. The resistance is the ratio of the voltage to the current. For intrinsic capacitances, an ac source of around 0.1Vdd is applied at the power and ground pins and the magnitude and phase of the current through this source for both the output states is recorded. The capacitance is then calculated as

$$C = \frac{I_0}{\omega V_0 \cos(\phi)} \tag{1}$$

AC analysis is performed from 10MHz to 1GHz and the average of all the values is computed and taken as the final capacitance [5]. CCS Noise specifications demand the response of a cell to a ramp input [8], [10]. Transient analysis is performed to get this information. Distinct subcircuit instantiations are created for measurement of noise timing data for each input pin and for each type of transition at the output.

The total number of subcircuits in this file is a function of the number of inputs in the cell and can be given as $2n+4$, with transient analysis performed on $2n$ subcircuits, dc analysis and ac analysis on 2 subcircuits each. Therefore, the total computational complexity of this file given by $O_{trans}(n)+O_{dc}(1)+O_{ac}(1)$, where $O_{trans}()$, $O_{dc}()$ and $O_{ac}()$ are the order of complexity for a transient, dc and ac analysis respectively.

4. DATA PROCESSING

As mentioned earlier, there are three spice files for each cell, viz. *_char.sp, *_noise.sp, *_lkg.sp. After simulation, each of these spice files generate a *.lis* file with the same name for each cell, i.e., *_char.lis, *_noise.lis and *_lkg. These files are parsed by a C++ program to extract the required information as explained in the following sections. The CCS validation guidelines [4], [7], [10] have been followed for processing and validating the simulation data.

*A. *_char.lis*

The Noise dc current values are stored in a 2-D array and they do not require any processing. The dynamic currents have to be segmented and validated for their correctness before they are stored and the CCS Timing Receiver model capacitances have to be calculated. These are done as follows:

Dynamic currents (output): The output current is the current flowing in charging the output capacitor. The output current waveform has a lot of data samples and only a few data samples need to be stored to get the required level of accuracy and to reduce the size of the library file. We have followed the Synopsys characterization guidelines for segmentation of the current waveform with a few minor modifications. The algorithm used for segmentation is given below.

1) Let $n_s = n/15$, where n is the total number of samples of the current waveform.
2) Obtain one value from every set of n_s values in the waveform data.
3) Integrate the sampled waveform using equation 2.

$$V_{out} = \frac{0.5}{C_{out}}(I_n + I_{n-1})\,(t_n - t_{n-1}) \qquad (2)$$

4) Check whether the result is within 5% of the expected value (Vdd for rising transitions and 0 for falling transitions).
5) If yes, STOP, else decrement n_s by 1
6) Repeat steps 2-5.

We will have a minimum of 15 values in every *output_current_ rise* and *output_current_fall* group.

Dynamic currents (Power and Ground (PG) pins): The current through the power and ground pins also have to be stored. These values are stored in the *pg_current_rise* and *pg_current_fall* groups in the *.lib* file. Just like the output current data, these too have a large number of values and are segmented before committing it to the library file.

Unlike output currents, since the PG currents have bigger variations due to short circuit component during segmentation process, they are first converted to a step-wise format by dividing the original waveform into a number of buckets. For each bucket the average current is captured as a list of current values (I, t), as shown in figure 1.

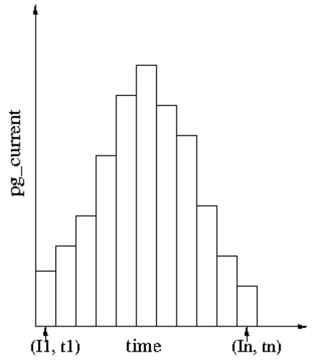


Fig. 1. Stepwise Power and Ground (PG) current waveform

Smaller the bucket size is, higher the accuracy of the current waveform approximation, however requiring more number of current points to be saved in the library. Points are selected from the stepwise current waveform such that the charge difference between the piecewise linear current and the stepwise current is within the recommended error tolerance on the charge (2%). Charge is calculated by integrating the current waveform using the GNU scientific library. We begin with an initial bucket size of *n/20* points, where *n* is the total number of values stored for PG current, and keep decreasing the bucket size till we achieve the desired accuracy. Figure 2 shows the segmentation schematic.

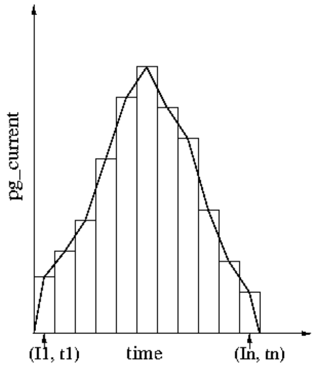
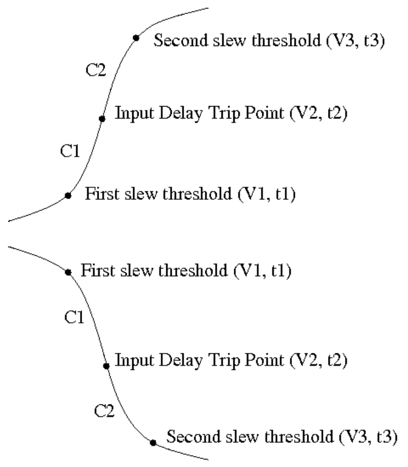


Fig. 2. Segmenting the PG current waveform for piecewise linear representation

Receiver Model capacitances: The receiver model capacitances are calculated from the input current waveform using the expressions given below (also refer to figure 3). The GNU scientific libraries were used for computation of the integral.

$$C_1 = \frac{\int_{t1}^{t2} I_{in} dt}{V_2} \text{ and } C_2 = \frac{\int_{t2}^{t3} I_{in} dt}{V_{dd} V_2}$$



Second slew threshold (V3, t3)
C2
Input Delay Trip Point (V2, t2)
C1
First slew threshold (V1, t1)
First slew threshold (V1, t1)
C1
Input Delay Trip Point (V2, t2)
C2
Second slew threshold (V3, t3)

Fig. 3. Receiver Model Capacitance calculation

*B. *_ lkg.lis*

Intrinsic parasitics: The *_lkg.lis file has the values of current from the disturbance dc source (section III-C). The value of resistance is obtained by dividing the voltage of the disturbance source by the current measured. On similar lines, the magnitude and phase of the current through the disturbance ac source are stored in the *.lis* file, from which the intrinsic capacitance is computed using equation 1.

Noise output voltage timing tables: As mentioned in section III-C, the response of the cell for a ramp input needs to be recorded. The *.lis* file stores the values of time for five different values of output voltage during the transition. These values are read and systematically stored in an array, for the sake of ease of processing.

*C. *_noise.lis*

The static leakage currents and the values of miller capacitances are computed from the data in this file.

Leakage currents: The leakage currents at the gates and at the power and ground pins for different states of the standard cell are stored directly in the *.lis* file in a suitable array format.

Miller capacitances: The *.lis* file contains the values of voltage change at the input for a particular voltage change at the output. Miller capacitances for all the inputs for both rising and falling output transitions are computed using these values as mentioned in [8].

5. LIBRARY PREPARATION

The job of writing the processed data into the library file (*.lib* – Liberty format of Synopsys) is done by C++ programs [2-3], [6], [9]. The data of a cell is processed by executing a shell script which processes the C code and writes the cell data into the library one at a time.

The data from the *.lis* files are not acquired in the order in which they are supposed to be written. Also it is not possible to collect all the data at one go and then write them together, since many variables are reused between iterations. Hence a number of intermediate files are created during processing and before storing the final library data. The cell group information like area, leakage currents and intrinsic parasitics is written into a file *mylib.lib*. The pin group information including receiver model groups are written into a file *pins.lib*. The dynamic currents at the power and ground pins are written into files *mylibdynij.lib* where i indicates whether the pin is Vss (0) or Vdd (1), and j indicates whether the output state is 0 or 1.The timing group information is written into files named as *mylibtimeij.lib*, where i indicates the input pin number (1, 2, etc.) and j indicates the nature of output transition, fall (f) or rise (r). e.g., *mylibtime1f.lib* indicates input pin 1 - output falling transition file.

The *mylibdyn*.lib* files are concatenated in the order 00, 01, 10, 11 into a file called *mydyn.lib*. The files *mylib.lib mydyn.lib pins.lib* are concatenated into a file *mylibtemp.lib*, which is then concatenated with the timing information files, *mylibtime1f.lib mylibtime1r.lib mylibtime2f.lib...* to give a file *finallibn.lib*, wherein *n* indicates the number of the cell being processed. The file *file1.lib* contains the library group information that applies to all the cells in the library. Once all the files have been processed, the files *file1.lib finallib*.lib* are concatenated to give the final library *finallibfinal.lib*, which can then be renamed according to the name given by the user.

A. Library Compilation

To make the library file usable for synthesis, we need to compile it into the *.db* format using Library Compiler from Synopsys [11] for use with Design Compiler It also generates libraries for post-synthesis simulations. The script for generating synthesis and simulation libraries is shown below.

```
read_lib mylib65.lib
#Synthesis library - mylib65.db
write_lib mylib65 -format db \
-output mylib65.db
#VHDL simulation library - mylib65.vhd
write_lib mylib65 -format vhdl \
-output mylib65.vhd
```

Successful compilation confirms the syntactical correctness and the correctness of the values stored.

6. Results

To test **AutoLibGen**, layouts of *and2_x1, inv_x1, inv_x2, nand3_x1, nor2_x1* and *or2_x2* cells using 65nm technology is prepared. The parasitic annotated netlists were extracted and a library mylib65.lib was created using our software. Verilog code for a small circuit as shown in figure 4 was written and synthesized using Synopsys Design Vision. The expressions for the output is given by,

$$out1 = \overline{\overline{in1} + \overline{(in2 \cdot in3 \cdot in4)}(in5+in6)} \qquad (3)$$

$$out2 = \overline{(in5+in6) \cdot in7} \qquad (4)$$

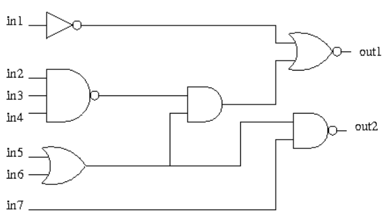


Fig. 4. Logic circuit diagram for the Verilog code

The code is synthesized using mylib65.db with the default optimization options. The synthesized circuit is shown in figure 5. The timing, area and cell reports were successfully generated. This confirms the correctness of the library developed using our tool.

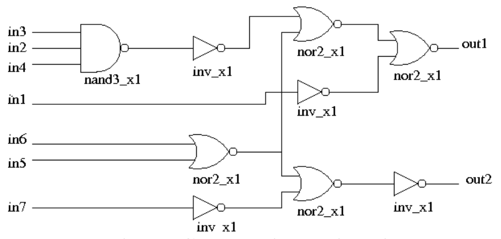


Fig. 5. Synthesized circuit

A. Limitations

Although the synthesis process correctly uses gates from the library generated using **AutoLibGen**, there are some limitations with the library generation process. These are listed below.

1) Currently, **AutoLibGen** tool supports only the following basic gates: NAND, NOR, AND, OR, NOT, BUFFER.
2) Gates should have a symmetric structure.
3) It is necessary to have an inverter in the library (for deciding the points of characterization).
4) Does not support gates with multiple outputs.

7. Conclusions

In this work, we have proposed a software tool, **AutoLibGen** for Library Characterization at 65nm node supporting the basic gates NAND, NOR, AND, OR, NOT and BUFFER cells. It characterizes the cells for timing, power and noise using the Composite Current Source methodology, which is found to be a lot more accurate for VDSM designs than the traditional Non Linear Delay and Power models. The software uses HSpice for performing the simulations for characterization. The tool is tested using HSpice version 2006.03-SP1, Library Compiler version 2007.12, Design Compiler version 2007.12 from Synopsys. The testing was successful and no errors were encountered. Since, the cells are characterized and written into the library sequentially, the memory requirement does not increase with the number of cells in the library. Hence, there is no limit on the number of cells that can be contained by the library.

References

[1] Liberty User Guide, Vol. 1 (Version 2007.03)
[2] Synopsys CCS Timing Library characterization guidelines, Version 3.1
[3] Synopsys CCS Timing Liberty Syntax, Version 1.2
[4] Synopsys CCS Timing Library Validation Guidelines, Version 2.0
[5] Synopsys CCS Power Library characterization guidelines, Version 3.0
[6] Synopsys CCS Power Liberty Syntax, Version 3.0
[7] Synopsys CCS Power Validation Document, Version 2.0
[8] Synopsys CCS Noise characterization guidelines, Version 1.2
[9] Synopsys CCS Noise Liberty Syntax, Version 2.0
[10] Synopsys CCS Noise Validation Document, Version 2.0
[11] Library Compiler User Guide, version 2007.12
[12] HSpice Simulation and Analysis Users Guide, Version Y-2006.09, Sept. 2006
[13] Kai Zhang, Wang Dong-hui and Li Yungang, "Library building for Sub-Micron CMOS process", Proc. Fifth International IEEE Conference on ASIC 2003, pp. 1369–1372.
[14] Binay Ackaloor and Dinesh Gaitonde, "An Overview of Library Characterization in Semi-Custom Design", Proc. IEEE Custom Integrated Circuits Conference, 1998, pp. 305–312
[15] Hashimoto, M., Fujimori, K. and Onodera, H., "Standard Cell Libraries with Various Driving Strength Cells", Proc. Asia and South Pacific Design Automation Conference, 2003, pp. 589–590
[16] www.vlsitechnology.org