Digital CMOS Library Design

João Lucas Munhão
joao.s.munhao@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2017

Abstract

Digital cell characterization is essencial in modern integrated circuits digital design flow. Liberty files that contain information about propagation time delay, timing constraints in sequencial cells and both dynamic and static power consumption are the cornerstone of this type of work. They can be obtained by simulating standard cells with different combinations of supply voltage, temperature, load capacity and MOS propagation speed (corners) . The simulation time is substantially reduced with liberty files in backannotation, compared with analog, SPICE, simulations. The goal therefore is to create a methodology and respective software capable of assisting a digital design engineer in characterizing digital cell libraries.
Keywords: digital design flow, digital cell library, cell characterization, corner analysis, liberty file

1. Introduction

The increased complexity of ICs and the increasing transistor density per unit area have greatly increased the time needed to simulate this type of circuits. Therefore a simplified model for delay, power, capacitance and function was developed on cell/gate level, where simulation with completely modelled transistors is a lot simpler and less time consuming.

By simulating the small building blocks first in different conditions, the time needed to create and validate large ICs is reduced considerably, cutting costs and allowing faster development of newer designs. The data from this simulations are compiled in timing and power characterization files.

In a partnership with SiliconGate®, a company specialized in developing state-of-the-art IPs, a proposal was made to develop a methodology and a software capable of characterizing standard cell libraries.

The aim of this work is to develop a powerful tool and respective methodology capable of creating library characterization files.

Section 2 discusses previous work done concerning library characterization and explains the Synopsys Liberty file format, section 3 gives an introduction on the developed methodology, section 4 describes the testbench design flow, section 5 explains the methods' scripts, section 6 illustrates the process of characterizing a standard cell and discusses the validity of the method, and, finally, section 9 compiles the methods merits, problems and limitations and discusses future work.

2. Background

2.1. Previous Work

In [9], Sulistyo uses the Non-Linear Delay Model (NLDM) for cell characterization, giving detailed description for time delay analysis and power characterization using SPICE testbenches for various types of cells.

In [4], Goyal and Kumar discuss limitations of characterizing sub 90 nm technologies. Increased wiring delays, interconnect resistances and non-linearity of waveforms are not taken into account by the NLDM. Cadence's ECSM models and Synopsys' CCS models are possible solutions which this paper analyzes and compares with SPICE models.

The smaller cell footprint also meant short channel effects became extremely important as stated in [1]. With this issue becoming relevant in two dimensional (2D) transistor design, three dimensional (3D) transistor design, FinFETs, with better leakage current characteristics were developed, [5]. In [12], Yuan et al. propose a method for characterizing this type of cells using a surface potential based model called BSIM-CMG.

In [3], Kenza and Ouardi describe a methodology based on the so called Scalable Polynomial Delay Model (SPDM), which is a equation based characterization method, capable of reducing simulation time by staggering values as it only needs to write a single library for all operating conditions and operating ranges of a particular technology. However it loses accuracy as it models cell delay behaviour using a very limited number of variables.

In [6], Nareshkumar estimates the setup and hold

times of a cell through Statistical Static Timing Analysis (SSTA), claiming high accuracy and faster simulations times for the cells tested.

2.2. Synopsys Liberty Format

During digital design flow, the simulation and P%R software can refer to time characterization files to estimate propagation delay through the different critical and non-critical circuit paths. This characterization is done based on process corners, which are different combinations of input slew rate, output load capacitance, operating temperature, supply voltage, transistor technology, among others. Power consumption can also be estimated using the liberty files. The Synopsys Liberty format is the most widely adopted by the industry and therefore is the one chosen for this project. The files have a very specific syntax that must be respected, as seen in Code 1

Code 1: Liberty file library and general parameters definition.

```
library (name) {
  technology (name) ; /* library-level attributes
       */
  delay_model : table_lookup;
  bus_naming_style : string;
  ...
  time_unit : unit ;
  voltage_unit : unit ;
  current_unit : unit ;
  pulling_resistance_unit : unit ;
  capacitive_load_unit (value, unit) ;
  leakage_power_unit : unit ;

  nom_process                    : value ;
  nom_temperature                : value ;
  nom_voltage                    : value ;

  operating_conditions (name) {
    /* operating conditions */
  }
  lu_table_template (name) {
    /* time lookup table template information */
  }
  ...
  cell (name1) {/* cell definitions */
    /* cell information */
  }
...
```

This files are primarily constructed by group statements inside curly brackets ({). Each group statement has its own subgroups and attributes. Example of groups are *library*, *cell* and *pin*.

3. Liberty File Generation Methodology

The developed methodology is capable of characterizing a cell's time, power and pin capacitance characteristics. It is the direct continuation of the work done in SiliconGate, concerning digital library characterization. Figure 1 shows the developed methodology for characterizing a standard cell library.

The first thing needed to characterize a standard cell, is a testbench. The testbench is a test environment for the cell, in it, different input pin stimuli are

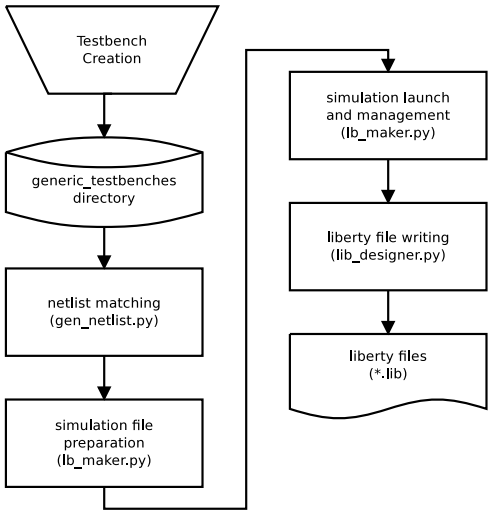

Figure 1: Methodology for digital library characterization.

fed into the cell, and each input signal combination will induce a certain behaviour on the cell's outputs. These testbenches have to be designed manually, due to their complexity. Therefore, an useful feature is reusability. Two cells from two different digital cell libraries may share the same functionality, input pin name and number, e.g. an inverter, therefore, the same testbench can be used for both cells. Anytime a new testbench is designed, it is also stored in a repository for future use.

After designing the necessary, non-existing, testbenches, *gen_netlist.py* makes a matchmaking between the cell's netlist and the testbenches in the repository directory. If a cell's schematic is updated, it's only necessary to reextract its netlist and rerun the script to have the files ready for simulation.

Next, a script manages and launches the corners' HSPICE simulations. The script can launch the simulations in multiple Central Processing Unit (CPU) cores, speeding up the total simulation time.

After obtaining the simulation results, *lib_designer.py* can be run. The script needs an empty liberty file shell, which is a pre-made file containing keywords that the script identifies and substitutes with measured data. This facilitates the writing of liberty files, minimizing code complexity.

The obtained liberty files can then be verified by Synopsys' Design Vision software, to ensure the files syntax are correct.

The following sections give a more in-depth look into the method, section 4 describes the testbench design and creation procees, while section 5 goes through each created script explaining its functionality.

4. Digital Cells Characterization Testbench

If a new, never before characterized cell is present in a digital library. There are two possible ways to create its testbench. A new one can be created from scratch, or it can be adapted from an similar one. The following measures should be included in a testbench, for full charactetization of the cell.

4.1. Propagation Time Delays and Transition Times

There are two important timing measurements to characterize. Timing arcs and timing constraints. Propagation timing delay refers to the time it takes an input pin signal to drive the output pin signal to a certain value.

Figure 2 shows the definition of delay and transition times for logic circuits. The input rise and fall times are labeled t_r and t_f, whereas t_{LH} and t_{HL} are the output rise and fall time. t_{PLH} and t_{PHL} are the propagation delay for the rising and falling edge respectively. The propagation delay is measured from a 50% value change on the input pin to a 50% value change on the output pin, while the slew rate or transition time of the output is the time the signal takes to vary from 10% to 90% on a rising edge, while on a falling edge it's from 90% to 10% [2].

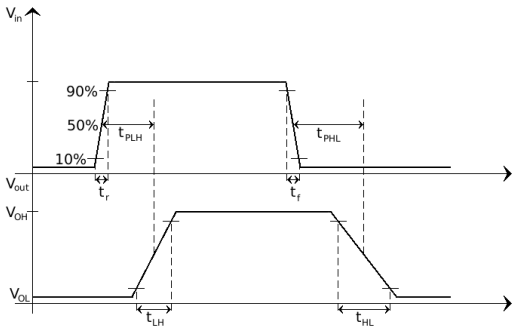

Figure 2: Delay and transition times definition.

The propagation delay and slew rate are proportional to the input transition and to the output load. Due to this reason, the liberty file exhibits propagation delays and timing transitions as a 2 dimensional table of input net transition versus output net capacitance.

4.2. Timing Violations

Sequential cells have other important time characteristics. Minimum pulse width can be defined for an input pin as the time between rising/falling edge at 50% VDD till falling/rising edge at 50% VDD, that guarantees a logic change at output. It depends specifically on the input transition rate, so its liberty file table is unidimensional.

Setup time is defined as the time the input needs to be stable before a capturing edge clock, while hold time is the time the input must remain stable after a capturing clock edge.

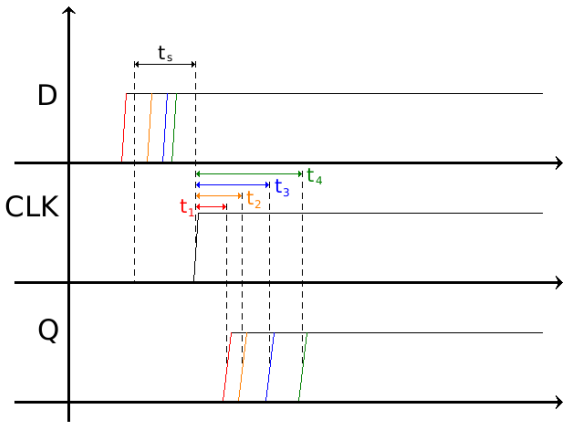

Figure 3: Clock to Q delay for different D pin transitions before and after setup time.

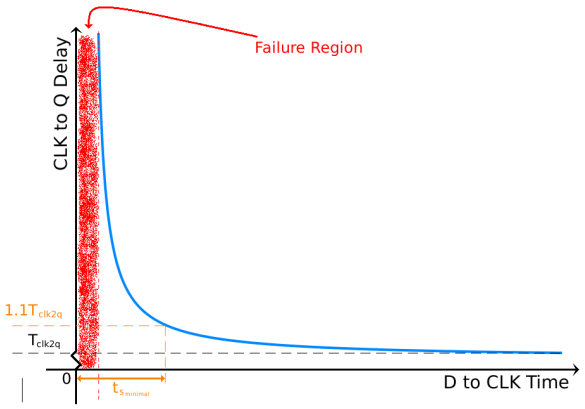

Figure 4: Clock to Q delay dependence with D to clock time.

In Figure 3 clock to Q delay is exemplified for different time intervals between a D pin transition and the capture clock edge, while in Figure 4 this dependence is plotted. A safe estimation of setup time, can be done, by finding the D to CLK edge time that causes a 10% increase in CLK to Q delay, relative to it's nominal value. As this timing measurements relate two input pins, their representation on liberty format will be a two dimensional table of the reference pin input net transition versus the related pin input net transition.

Recovery and removal time are measures that characterize minimum separation times between asynchronous pins changes and clock edges in sequential cells. Recovery time is the time an asynchronous pin signal must be stable before a clock edge, while removal time is the time such signal must be stable after a clock edge. Their LUT in-

dexes are the same ones of setup time

4.3. Leakage Power

Leakage power is the power a standard cell dissipates when both its inputs and outputs are not making signal transitions. There are three main reasons for leakage power, i) Junction Leakage, which results from the diffused p and n regions forming parasitic diodes [8]; ii) Gate-Oxide Leakage, as in CMOS technology below 90nm, gate oxide tunneling is an important player in static power leakage as carriers (electrons) can tunnel from the channel to the gate electrode [10]; iii) Subthreshold Leakage, became more important with the constant miniaturization of transistors and the resulting increase in power dissipation per chip unit area, the supply voltage has been scaled down, which led to the reduction of the threshold voltage (V_{th}) [7].

For measuring the static power in a cell, we can use the following formula:

$$P_{SP} = V_{DD} \times I_{DD} \tag{1}$$

The Liberty files display leakage power values for every input and output pin combination.

4.4. Dynamic Power

Dynamic power is obtained by adding switching power consumption (P_{SW}) and internal power consumption (P_I). Being this the total power, it can be calculated as energy by:

$$E_{tot} = \int_{t_1}^{t_2} V_{dd} \times I_{V_{dd}} \, \mathrm{d}t \tag{2}$$

Switching power concerns the power necessary to charge or discharge an external load. For characterization purposes the load is replaced by a capacitor whose value during characterization is chosen according to typical cell fan-out. The switching energy can be written as:

$$E_{sw} = \int_{t_1}^{t_2} V_{dd} I_{sw} = \int_{0}^{V_{dd}} C V_{dd} \, \mathrm{d}v = \frac{1}{2} C V_{dd}^2 \tag{3}$$

Internal power not only represents charging and discharging power of the cells' internal nets, but also power that is wasted due to momentary turn on of NMOS and PMOS network that connects the cells' supply and ground pins.

The internal power can be obtained by using equations 2 and 3:

$$E_I = E_{tot} - E_{sw} = \int_{t_1}^{t_2} V_{dd} \times I_{V_{dd}} \, \mathrm{d}t - \frac{1}{2} C V_{dd}^2 \tag{4}$$

By summing both switching energy and internal energy we get the total dynamic energy consumed by a cell:

Code 2: Generic testbench directory tree.

```
.
‘-- generic_netlist
|-- _and2
|       ‘-- comb
|           |-- footer.cfg
|           |-- header.cfg
|           ‘-- input.ckt
|-- _nand2
|       ‘-- comb
|           |-- footer.cfg
|           |-- header.cfg
|           ‘-- input.ckt
...
```

$$E_D = E_{sw} + E_I \tag{5}$$

When an input transition does not cause a variation on the output, power consumption will only depend on the input transition time, meaning a unidimensional table can represent the power consumption.

When an input pin transition leads to an output transition, the power now will not only depend upon the rise or fall transition of the input, but also on the load capacitance of the output, therefore a 2 dimension table is required.

4.5. Capacitance

The input pin capacitance can be measured on rise and fall transitions that causes output pin transitions, and can be measure using:

$$C = \frac{\int_{t_1}^{t_2} I_{rise/fall} \, \mathrm{d}t}{V_{dd}} \tag{6}$$

Current integration measures made for power calculations in 4.4 can again be used. The pin capacitance corresponds to the average value of the measures, of rise and fall capacitance, through the different input net transitions and output load capacitances.

After creating the testbench for the new cell, its files can be added to a testbench repository, whose directory tree is similar to code 2. Each testbench is divided in three files. The header contains the type of simulation. The input contains the stimuli, and the footer contains all the time, power and capacitance measurements.

Scripts

5. Method Scripts

After having the testebenches prepared and in the repository directory, we can now run the scripts responsible for simulating the corners and generating the liberty files.

Figure 5 contains the general steps the script takes to generate the liberty files.

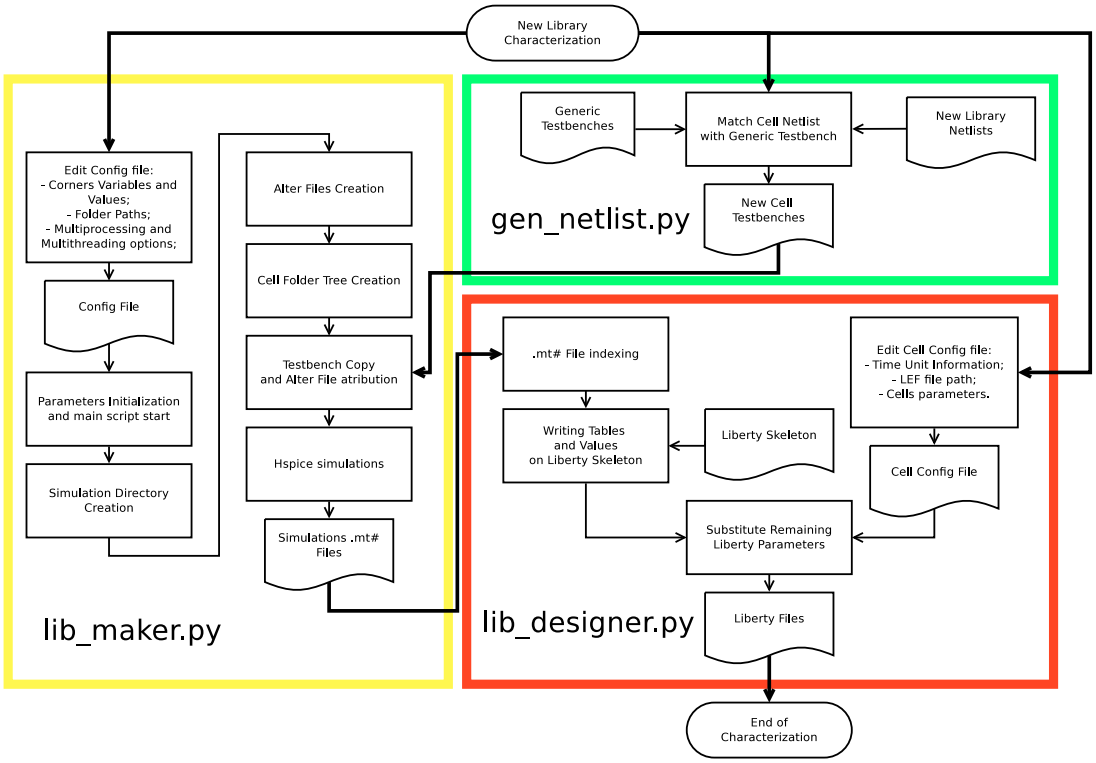

Figure 5: Automatic part of the characterization Flow.

After the creation of the testbench and their inclusion on a testbench repository, *gen_netlist.py* is able to match and concatenate a cell netlist with the correct testbench. The script lists all the cells in the digital library folder and matches their names with the existing testbenches in the repository of code 2. If a match is found, the script copies the files digital library directory and concatenates the cells' netlist with them.

After the testbench matchmaking process, *lib_maker.py* takes over. This is the main script of the method, and it's run in the following manner:

$ ./lib_maker.py 'path_netlist_folder' 'path_empty_lib_shells' 'execution_path'

Where 'path_netlist_folder' is the digital library folder, in which its cell's directories contain the respective netlists, 'path_empty_lib_shells' is the directory containing the empty shell liberty files and 'execution_path' is where the simulations are run and the liberty files are created.

A configuration file must be edited with the corner parameters and other necessary simulation configurations, like the number o processors the script can use. The script creates different types of alter files based on the corners parameters on the configuration file, on the number of processors and on the type of simulation they shall be used to run. This is necessary as each type of measurement, explained in section 4, requires different parameters combinations to produce its corners.

The script will then prepare the 'execution_path' folder by populating it with the necessary files for simulation, as in code 3.

Code 3: Simulation folder directory tree.

```
.
|-- alter1_25.inc
|-- ...
`-- hspiceSim
|-- BASICLIB_nand2
|   |-- alter1_25
|   |   |-- header.cfg
|   |   |-- input.ckt
|   |   |-- footer.cfg
|   |   |-- ...
...
|-- BASICLIB_dffnr
|   |-- alter1_25
... ... ...
```

After running the simulations, the cell's alter directories will have .mt# files with the simulations results, ready to be written to a liberty file.

The third and last script, *lib_designer.py* is responsible for writing the simulation data into the liberty file. The script requires that the testbench uses a standarized variable naming scheme. This is useful to identify the type of measure, the input pin that causes a transition and the output pin that is affected. Facilitating the writing of the liberty file.

Here empty liberty shells are necessary. Like the general testbenches described in section 4. Liberty

shells are created manually and can be used across similar cells of different technologies as they are custom made for each cell. A keyword system is used to facilitate completion of the liberty file. The script will look for keywords inside < and >, on the simulation results or in a cell configuration file that contains cell specific parameters.

The simulated data is indexed in three databases, one that links the .mt# file number to its specific corner, another that groups .mt# files indexes that contain data that will be part of the same liberty file, and the last one that contains the data on each .mt# file. This database, together with a digital librariy cells configuration file, containing cell specific information like pin number and name, cell function, among others, are required to successfully write a liberty file. After each cell liberty file is written, they are concatenated based on the corners they contain.

6. Evaluation and Validation

In subsection 7 the characterization of a negative edge D flip-flop is described, referring difficulties and the necessary hours to execute the job, while in subsection 8 a circuit using standard cells from a 22nm standard cell library is simulated with HSPICE and a verilog backannotaded with a SDF file, generated using the liberty files. Both simulations are compared concernign the delay of various corners measured.

7. Evaluation

To evaluate the methodology created, the characterization of a negative edge D flip-flop was done. This cell was chosen due to the various type of time analysis it requires.

The flip-flop was designed in-house by the engineers of SiliconGate, so no previous characterization existed.

The following subsection describes the steps taken to characterize the cell and the difficulties found during the realization of this task.

7.1. Negative edge D flip-flop characterization

The first step for the characterization is the creation of the flip-flop testbench.

The most time consuming task is designing an stimuli that permits measurements of all the necessary cell delay combinations, power consumption and pin capacitance. This usually takes around 9 hours, as one needs to write the signal input on the testbench, simulate, and then verify the measures correctness. Nonetheless, with the system of testbench reusability developed, this is, at most, a one time ocurrence. A signal viewer software and the simulation .lis file are the main tools in this phase. A designer will need to be going back and forward between the two of them, to ensure the measures are done with the right signal edges at the right time. It is also necessary to make sure the output signals have time to fully change it's voltage value between input pin transitions, as slow corners (high load capacitance, slow transistors and high temperature), can lead to wrongful or failed measurements.

With the testbench created, the next step is to break it apart to form the files that constitute the generic testbench repository. This task can be done together with the creation of the empty liberty shell. These tasks, together, need around 15 minutes to be completed.

With the testbench repository updated, the *gen_netlist.py* script can be run anytime changes on the cell design are made. Ensuring the characterization uses the latest cell netlist.

Next, preparation to launch the main function, *lib_maker.py*, can start. The first thing to do is to change the parameters on the configuration file. After this, execution of the script will create the simulation folder, prepare the simulation files, the tree structure of code 3 and launch the simulations. Simulation time varies with the ammount of CPU processors dedicated to the script and the with number of corners.

Simulations for 25 corners, which corresponds to a liberty file with 5 values of input net transition and 5 values of output net capacitance, take around 30 minutes to complete.

During script development, a problem appeared in the simulation management algorithm. HSPICE simulations can include .lib files, called library files. HSPICE library files can make high-level statements calls, contain netlists, model parameters, test vectors, types of analysis and option macros [11]. In characterization jobs, this files are used mainly to include transistors model types.

During corner simulations the libraries to be included can be changed. During characterization, as the simulations progressed, a new library include statement can be asserted with each alter. This haa the unexpected result of gradually leading the simulations to a halt.

To exemplify this issue, simulations of a corner with typical values were done. An alter with 25 combinations of input slew rates and output capacitances was run, while mantaining temperature and voltage constant at 25°C and 0.88 V respectively.

Each simulation run 10 times to ensure the simulation time was indeed correlated with the library inclusion statements. Two libraries, one for a MOS fast-fast model and another for layout characteristics, were included.

Tables 1 and 2 display simulation time of the 25 corners using two cells, the negative edge D flip flop and the NAND with 2 inputs. The tables contain two situations of library statement inclusions. On

Table 1: Negative edge D flip-flop simulation time, using different alter files.

dffnr	Simulation Time (s)					Average
1 Inst	93.049	93.318	93.056	93.213	93.338	93.350 ± 0.129
	93.279	93.250	93.665	93.866	93.468	
25 Inst	244.168	244.842	249.947	248.267	249.457	244.168 ± 1.105
	245.026	250.374	249.359	251.154	249.766	

Table 2: NAND2 simulation time, using different alter files.

nand2	Simulation Time (s)					Average (s)
1 Inst	46.274	46.419	45.950	46.081	45.862	46.192 ± 0.125
	46.071	46.232	46.294	46.089	46.652	
25 Inst	166.649	166.649	166.588	166.795	166.976	167.044 ± 0.476
	166.626	167.133	168.721	165.709	168.598	

the first, the library is only instantiated on the first alter, on the second they are instantiated on every alter statement. Note that temperature and voltage parameters are the same in each alter file, however that does not influence simulation time.

The MOS HSPICE models can be extremely complex, even with a low number of corners, adding the libraries in each alter statement dramatically increases the simulation time. The solution found was to change the way alter files are created, where libraries definitions are only included in the first corner they appear. This hinted to the possibility of HSPICE actually mantaining in memory every library included since the first simulation.

With the simulations finished, the last script can be run. *lib_designer.py* will compile the data and write the liberty file. As this script deals mainly with reading and writing data on disk, its execution time is minimal compared with creating the testbench and simulating the corners.

In the end around 10 hours are needed to characterize the flip-flop. However 90 % of this time was used to create the testbench, its now visible why testbench reusability is so important. Is also important to notice that 25 corners is a small number of corners. On typical characterization jobs, thousands of corners are needed to fully characterize a standard cell, requiring hours or days of simulation to generate the necessary data to write the liberty files for an entire digital cell library.

8. Validation

To validate the timing characterization done, a comparison between a circuit's analog HSPICE simulation and the verilog model, with a SDF file backannotaded, can be used.

The HSPICE models the standard cells transistors, with great detail, so its simulations are closely matched to reality, except the delays, caused by the circuits interconnections, not being modeled. The backannotaded verilog model on the other hand, includes an SDF file generated after a digital circuit's P&R , this file indicates not only the circuits cell delay, provided by the liberty files, but also the cell's interconnections delays. If we edit the file to not include the latter, a comparison can be made between the circuit's delay with HSPICE and with the verilog model.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: XOR truth table.

The first step, therefore, is to choose a circuit to be designed. The logic function XOR is a simple logic function that can be obtained using NANDs and NORs logic gates, table 3 displays the truth table of a XOR. The XOR output changes to high exclusively if a single input is high and if both inputs are low or high, the outputs is low.

To synthesize the circuit, Synopsys Design Compiler is used. Design Compiler uses the verilog RTL model of the XOR, as in code 4, and the liberty files of a 22nm digital cell library to generate a gate-level netlist, which is a completely structural description of the circuit using standard cells only.

Code 4: XOR logic RTL description.

```
module XOR_RTL(y   ,
               a   ,
               b
               );

   //PORTS ------------
   output y   ;
   input  a   ;
   input  b   ;

   //LOGIC ------------
   assign y = ( a & ~b) | ( ~a & b);

endmodule
```

The resulting circuit is displayed in figure 6. The circuit can be reextracted into a verilog model that contains instantiations of the standard cells, code 5 displays the extracted model.

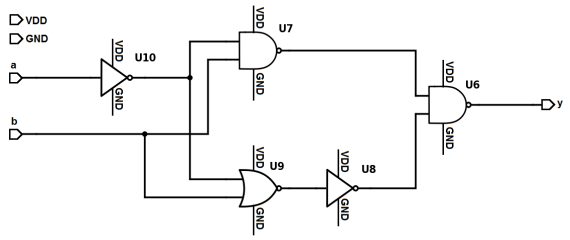

Figure 6: XOR synthesized circuit.

Code 5: XOR logic synthesized description.

```
module XOR_SYNT ( a, b, y, vdd, vss );
   output y   ;
   input  a   ;
   input  b   ;
   input  vdd ;
   input  vss ;

   wire   n5  ;
   wire   n6  ;
   wire   n7  ;
   wire   n8  ;

   BASICLIB_nand2 U6 ( .vdd(vdd), .vss(vss), .a(
        n5), .b(n6), .yz(y)  );
   BASICLIB_nand2 U7 ( .vdd(vdd), .vss(vss), .a(b
        ) , .b(n7), .yz(n6) );
   BASICLIB_inv   U8 ( .vdd(vdd), .vss(vss), .y(
        n8), .yz(n5)         );
   BASICLIB_nor2  U9 ( .vdd(vdd), .vss(vss), .a(
        n7), .b(b), .yz(n8)  );
   BASICLIB_inv   U10( .vdd(vdd), .vss(vss), .y(a
        ) , .yz(n7)          );

endmodule
```

The Synthesis tool used three different types of standard cells to generate the design, inverters, NANDs and NORs. This will allow to evaluate the accuracy of the characterization done using the liberty files of this cells.

At this step a testbench was created for the designed XOR. To simulate the XOR circuit integration in a larger design, other XOR cells were connected to its inputs and output, this resembles the typical driving strengths and output load capacitance the cell would receive if included in a larger circuit. In figure 7 the evaluated cell is U3.

After extracting the verilog model from the testbench circuit, the next step is to run Cadence SoC Encounter to make the P&R. To run the software, besides the synthesized verilog file and the digital library liberty files, its also necessary two LEF files, the technology LEF file and the digital library LEF file. The technology LEF files contains specific properties of the technology, i.e. number of metal

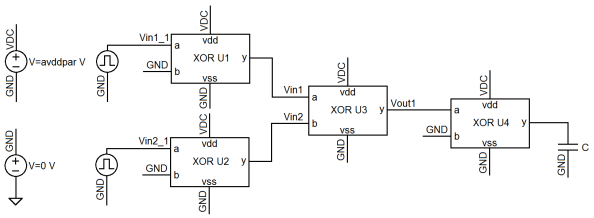

Figure 7: XOR testbench circuit.

layers and design rules, metal capacitances and via definitions. These files are specific to each technology so they are provided by the foundry. The digital library LEF file contains the description of each standard cell layout, for instance, its geometry, the different layers needed, the pins layout and blockages.

With these files, P&R of the testbench is made. With the layout, RC extraction and SDF file generation is done.

A verilog and HSPICE testbench is created, figure 8 exemplifies the stimuli visualized on the pins of XOR U3.

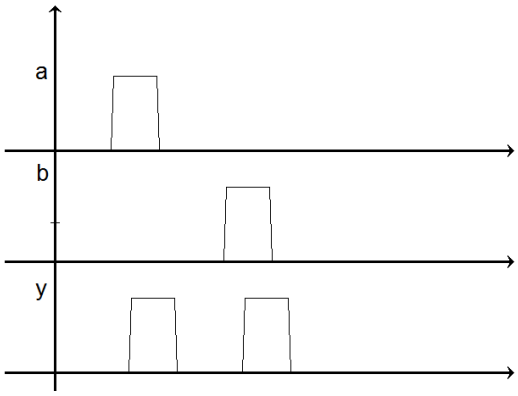

Figure 8: HSPICE and verilog stimuli.

Four corners are used to compare the analog HSPICE simulations and the backannotaded verilog simulations. These corners correspond to combinations of two types of transitors models, typical-typical and fast-fast and two supply voltages, 1.62 V and 1.8 V. The SDF files generated contain cell delays at 125 °C, so the HSPICE simulations were also run at this temperature. Tables 4 and 5 displays the delay between input and output pin transitions of XOR U3, for the stimuli exemplified in figure 8, while table 6 presents the relative error.

Generally, the backannotaded verilog model presents inferior cell delay when compared with the HSPICE simulation, with relative errors around 3.7 %. The inferior cell delay in verilog simulations, can be a consequence of characterizing the standard cells with the black box testbench illustrated

Table 4: XOR input to output delay in HSPICE simulation

HSPICE			Delay (ps)			
			corner			
			mos-tt		mos-ff	
a	b	y	1.62 V	1.8 V	1.62 V	1.8 V
rise	low	rise	173	152	156	139
fall	low	fall	182	163	167	150
low	rise	rise	77.7	67.5	69.8	61.6
low	fall	fall	79.2	68.9	70.8	62.6

Table 5: XOR input to output delay in Verilog simulation

Verilog			Delay (ps)			
			corner			
			mos-tt		mos-ff	
a	b	y	1.62 V	1.8 V	1.62 V	1.8 V
rise	low	rise	186	166	169	153
fall	low	fall	185	168	171	155
low	rise	rise	75.0	66.4	68.1	60.5
low	fall	fall	79.7	69.4	71.7	64.0

in figure 6. In real circuits, the drive strength on a cell pin is limited by the current the cells connected to the pin are capable of providing, as this current needs to charge the gate and the parasitics capacitante of the input pin. In the testbench, there is no limit to the current the Piece-Wise Linear voltage source can provide the cell, therefore the input will rise on any specified time leading to shorter cell delay and consequently affecting also the pin capacitance and dynamic power measures, as this are measured when the cells' input signals change their values. With this results, its possible to infer that in situations where supply voltage is even lower, the relative error will grow, as the supply voltage directly affects the cell output current, and therefore its drive strength capabilities.

9. Conclusions

The aim of this work was to develop a methodology and respective software capable of facilitating the task of characterizing standard cell libraries.

The developed method streamlined the process of library characterization as much as possible, reducing the ammount of manual tasks and maximizing computer run phases. However problems were encountered during the evaluation and validation processes.

The manual creation of testbenches proved to be a laborious task to accomplish, the time and effort needed to create a testbench for a cell's timing arcs, power consumption and pin capacitance varies with the cell's logic function, nonetheless, it takes precious time everytime a new digital cell library, with new standard cells, needs to be characterized. Sequential cells proved to be especially difficult, as optimization testbenches need to be designed to characterize the cells time constraints. The reusability of the testbenches partially solves this problem, but is not the ideal solution.

Table 6: Relative error between the HSPICE and Verilog measures

Comparison			Relative Error (%)			
			corner			
			mos-tt		mos-ff	
a	b	y	1.62 V	1.8 V	1.62 V	1.8 V
rise	low	rise	7.5	8.9	8.4	10.3
fall	low	fall	1.8	3.2	2.4	3.1
low	rise	rise	3.5	1.6	2.4	1.8
low	fall	fall	0.6	0.7	1.3	2.2

The HSPICE simulation launch and management script completely fulfills the objective of automatically launching and managing HSPICE simulations in a multicore environment. However, it lacked tools to detect if the simulations failed or if the data obtained was not valid. This limitation, partially hinders the methodology's flow, as it's necessary to manually correct these errors.

The liberty file writing script successfully automizes every step needed to write a liberty file, being an extremely realiable tool.

These scripts are based on a modular structure and this proved to facilitate error detection and correction.

The methodology was successfully validated, showing an average error of 3.7% in timing analysis, when comparing the HSPICE transistor level simulation with the logic level simulation.

A complete digital cell library designed by SiliconGate was characterized. As a consequence of being fully characterized by the corresponding liberty file, this digital cell library was the first to be licensed for use by one of SiliconGate's clients. This result is an evidence that the present work fully achieved its purposes, defining a new period in SiliconGate's digital cell characterization capability.

9.1. Future Work

The general workflow suffers from limitations in the accuracy of the generated liberty files, therefore, one of the first things to improve is the black box test model used to characterize the liberty cells.

A modification that can improve the method's flow is the automation of testbench creation. The ideal software has to be capable of inserting specified input signals, detecting the evaluated cell logic, creating a custom full-fledged test table and ensure the stimuli covers all the necessary cases needed to measure. This process becomes increasingly complex as one progress from combinational logic, to

sequential and tri-state logic. Nonetheless, full automation can only be obtained going down this path. So future work on this area will seek to gradually automize testbench design.

The NLDM used is also not the best existing delay model to characterize standard cells. The Composite Current Source (CCS) model can be the next step in improving the accuracy of the simulations, as these models take into account physical effects, e.g. the effect of Miller capacitance on input pin capacitance and net delay, in technology sizes inferior to 90 nm. The implementation of this model would require changes on the way the liberty files are written.

The automatic identification of failed simulations during characterization and the re-execution of these simulations with different parameters is a feature that needs to be implemented. It's a feature necessary to facilitate a digital designer's job and that can improve vastly the obtained results.

The liberty file creation script, became rather complex with the great number of measures it needed. So simplification of this tool's workflow is necessary.

Scripts that do a relative validation of the generated liberty files can also be designed. These scripts would go through the look-up tables, comparing the timing data to ensure it increases with capacitance and input net transition. This of course, does not verify if the results are exact, nonetheless it would indicate anomalous deviations through tables, indicating wrongful indexing by the software, mishaps on the alter file creation or unexpected cell behaviour.

Furthermore, the ability to display characterization data in other formats besides the Synopsys Liberty would be an important addition.

Acknowledgements

I would like to thank Prof. Marcelino Bicho dos Santos and the employees at SiliconGate for helping me through this process.

References

[1] C. E. Amrutlal, P. J. Hemang, D. N. Nareshbai, and P. R. Mukeshbhai. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Journal of Information, Knowledge and Research in Electronics and Communication Engineering*, 02(02):857–860, October 2013.

[2] R. J. Baker. *CMOS - Circuit Design, Layout and Simulation*. Wiley-Interscience, 2^{nd} edition, 2008. ISBN 978-0-470-22941-5.

[3] K. Charafeddine and F. Ouardi. Fast timing characterization of cells in standard cell library design based on curve fitting. In *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems*, 2017.

[4] R. Goyal and N. Kumar. Current based delay models: A must for nanometer timing. 2005.

[5] D. Hisamoto et al. FinFETA Self-Aligned Double-Gate MOSFET Scalable to 20 nm. *IEEE Transactions on Electron Devices*, 47(12):2320–2325, December 2000.

[6] S. Nareshkumar. Probability Measurement of Setup and Hold Time with Statistical Static Timing Analysis. *Int. Journal of Engineering Research and Applications*, 5(1):38–41, January 2015.

[7] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Proceedings of IEEE*, 91(2):305–327, February 2003.

[8] A. Sarwar. CMOS Power Consumption and C_{pd} Calculation, June 1997. Texas Instruments Application Report.

[9] J. B. Sulistyo. On the Characterization of Library Cells. Master's thesis, Virginia Polytechnic Institute and State University, August 2000.

[10] A. K. Sultania, D. Sylvester, and S. S. Sapatnekar. Gate Oxide Leakage and Delay Tradeoffs for Dual-T_{ox} Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(12):1362–1375, December 2005.

[11] Synopsis Inc. *Hspice® User Guide: Basic Simulation and Analysis*, June 2017. Version M-2017.03-SP1.

[12] Y. Yuan, C. Martin, and E. Oruklu. Standard Cell Library Characterization for FinFET Transistors using BSIM-CMG Models. In *IEEE International Conference on Elecro/Information Technology (EIT)*, 2015.