HIGH SCHOOL EDITION
Journal of Student Research

Volume 11 Issue 3 (2022)

DeepSPICE: Accelerating Digital Cell Characterization Using Deep Learning

Ishwar Suriyaprakash[1] and Greg Burroughs[#]

[1] Homestead High School, USA
[#]Advisor

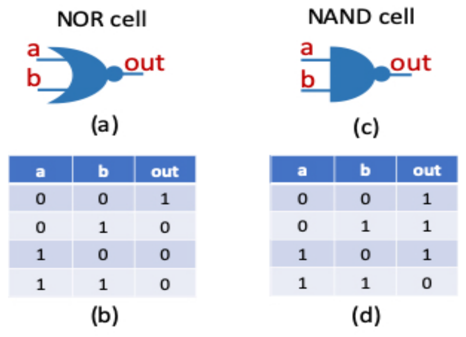
ABSTRACT

This paper introduces DeepSPICE, a machine learning approach to accelerate the characterization of the building blocks, or cells, of digital integrated circuits. In contrast to the current approach of computing the input-to-output propagation delays of a cell by simulating all possible input event combinations, DeepSPICE employs a Deep Neural Network (DNN) to learn from the propagation delays obtained by simulating a small subset of input event combinations and to predict those for the rest. The DNN for training and prediction is created using the Keras framework, and Python is used for automating the entire flow. The effectiveness of the DeepSPICE approach is demonstrated on 14 CMOS logic cells with the number of inputs ranging from 2 to 7. Simulations to create the training set for each cell are performed using the open source NGSPICE circuit simulator on their transistor-level circuit descriptions. Experiments using two different train:test ratios of 0.25:0.75 and 0.3:0.7 demonstrate the promise of reduction in time with DeepSPICE, especially for large cells where simulation costs are expensive. For cells with at least 6 inputs, DeepSPICE computes propagation delays for all input event combinations 2 to 2.2 times faster than baseline while limiting the error between 6.3% and 12.3%.

Introduction

Digital integrated circuits, or chips, are used to implement complex arithmetic and logic functions on silicon. Usually, a single chip is designed, and then several identical such chips are manufactured on a silicon wafer. The individual chips from each wafer are then separated and packaged to be sold to customers.

The process of designing a chip has several steps. The design of the functionality to be manufactured is first specified behaviorally using a hardware description language, such as Verilog, in a hierarchical manner. Subsequently, using software tools, this functional description is then synthesized to a structural implementation, or a circuit netlist, that is an interconnection of smaller building blocks called cells. Cells are used to implement small primitive Boolean functions such as NAND, NOR, and other complex functions using an interconnection of semiconductor devices called transistors, which act as switches. As examples, the symbols for cells that implement 2-input NAND and NOR functions along with their Boolean input-output relations expressed as truth



a	b	out
0	0	1
0	1	0
1	0	0
1	1	0

a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

tables are shown in Figure 1(a) and 1(b), and Figure 1(c) and 1(d), respectively. An example of a circuit netlist comprised of such cells is shown in Figure 2. To enable creation of the circuit netlist, a standard library of cells is designed by the owners of the manufacturing process and made available to the designers of integrated circuits.
Figure 1. (a) NOR cell symbol, (b) NOR truth table, (c) NAND cell symbol, (d) NAND truth table.

As part of the creation and validation of a circuit netlist, certain properties need to be determined to ensure that they meet the desired specifications. Examples of these properties are the maximum time taken by the circuit to complete its desired function and the power consumed during the circuit's operation. The time taken by a circuit to complete its function is usually calculated by measuring the time for changes at inputs to propagate to its outputs, which is known as propagation delay. Changes at inputs of a digital circuit are caused by Boolean events, which can be transitions from Boolean 0 to 1 (rise) or 1 to 0 (fall). Please refer to Figure 3 for an illustration of events. Determination of the properties of a circuit netlist is usually performed in a hierarchical manner using pre-computed values for the corresponding properties of the component cells that make up the netlist. One of the methods to determine the propagation delays of a circuit netlist hierarchically from those of its constituent cells is called timing analysis [13].

To aid in determining the properties of a circuit netlist, the provider of a cell library characterizes each cell in the library by determining these properties of the cell through analog simulations of its transistor-level circuit description. However, the behavior of each type of transistor can vary across a single chip and from chip-to-chip across a silicon wafer due to variations in the manufacturing process. So, the properties of each cell in the library need to be determined over the range of these manufacturing process variations. A tutorial on cell-based design methods used in practice can be found in [1], and details on the standard cell library characterization process can be found in [8].

The process of characterizing all the cells in a standard library is usually computationally expensive [7] [12]. For a library with typically several thousand cells, this characterization process involves simulating the transistor-level description of each cell (1) for each of its possible input events, and (2) over different operational behaviors of the transistor switches to account for variations in the manufacturing process across different chips on a wafer.

Recently, there have been research works to speed up this cell characterization process using machine learning methods. These works have addressed either accelerating the circuit simulation time for any given input event combination by speeding up the simulator itself [2], or accelerating the characterization of each cell over a range of manufacturing process variations (the second aspect mentioned in the previous paragraph) [7].

However, reducing the high computation time for simulating all possible input event combinations for a given cell has not been explored, and this work aims to address this goal. This work introduces DeepSPICE, an approach that uses supervised machine learning to reduce the total time needed to compute the input-to-output propagation delays of a cell. We also show that the speed-up provided by DeepSPICE is more for larger cells.

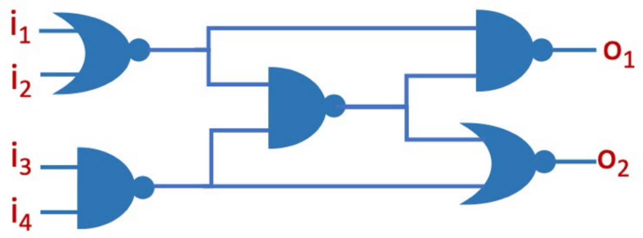


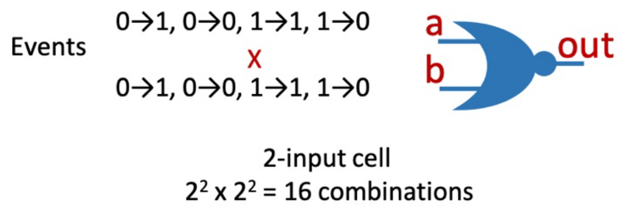
Figure 2. Example of a circuit composed of gates.

The paper is organized as follows. The motivation and background information are provided in Section II. Section III describes the DeepSPICE approach. Experiments on a set of test cases along with results are detailed in Section IV. Next steps in this research are outlined in Section V. The paper concludes in Section VI.

Motivation and Background

Cell delay computation

As mentioned in the Introduction (Section I), each cell in a library with typically several thousand cells needs to be characterized using simulations of its transistor-level circuit description to determine the propagation delay and power. The number of cell inputs in a typical library can range from 1, for a non-inverting buffer and an inverting NOT cell, to more than 15 for complex cells. For a cell with *n* inputs, the number of combinations of Boolean values that can be applied and simulated is 2^n, which is the number of rows of the function's truth table. However, to compute delays which involve propagation of rise (0 to 1) or fall (1 to 0) events from inputs to the output, the number of all possible input event combinations to be simulated is 2^n x $2^n = 4^n$. An example for a 2-input NOR gate is shown in Figure 3 where the number of input event combinations is 16. For a 12-input cell, the number of possible input event combinations is 4^{12}, or 16.8 million. Assuming one second per simulation of one input event combination, these simulations would take ~194 days to complete. One possible option to reduce the execution time of this process is to partition this input event set into subsets and execute the simulation of these subsets on multiple machines in parallel. While this approach decreases the overall time, it comes at the cost of increased computation resources. This paper introduces DeepSPICE, an approach that simulates only a small subset of input event combinations and then employs supervised machine learning on the results to predict

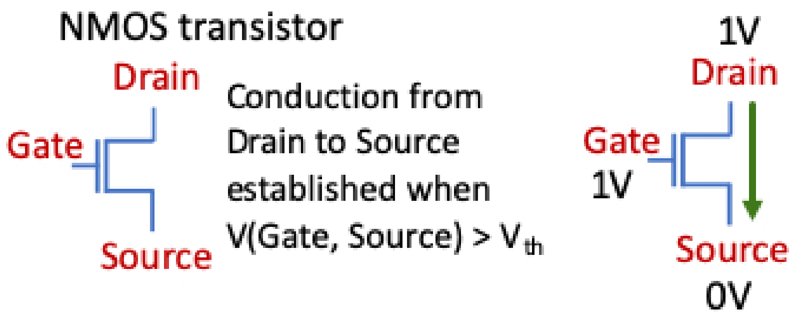


the results for the rest of the input event combinations. This work addresses speed-up only for estimating delays (not power) of each cell. However, the approach can be applied towards speed-up for estimating power as well. The next section provides background information on two types of transistors and how they are used as switches in a cell.

Figure 3. Events at inputs of a cell and input event combinations.

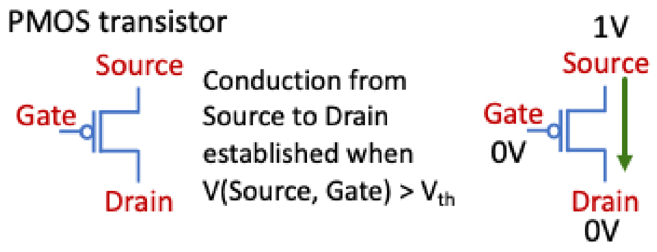
Transistors as switches

Cells are implemented using transistors as switches. Two predominant types of transistors used are the N-channel Metal-Oxide Semiconductor (NMOS) field effect transistor (FET) and the P-channel Metal-Oxide Semiconductor (PMOS) field effect transistor. Each of these two types of transistors has 4 terminals: gate, drain, source and bulk, where the bulk is usually connected to either the source or drain. The gate terminal serves as the switch that turns on electrical conduction between the drain and the source terminals. The transistor conducts when the voltage differential between the gate terminal and the source terminal exceeds a certain threshold voltage, and when there is a potential difference between the drain and the source terminal. An NMOS transistor turns "ON" when the voltage at its gate terminal exceeds that at the source terminal by the threshold voltage. A PMOS transistor turns "ON" when the voltage at its gate terminal falls below that at the source terminal by the threshold voltage. The symbol for an NMOS transistor and a simplified description of its use as a switch is shown in Figure


NMOS transistor
Drain
Gate
Source
Conduction from Drain to Source established when V(Gate, Source) > V_{th}
1V
Drain
Gate
1V
Source
0V

4. Similarly, Figure 5 shows the symbol for a PMOS transistor and its use as a switch.
Figure 4. NMOS transistor as a switch.

To simplify description, we can consider that an NMOS transistor turns "ON" when a Boolean value of 1 is applied at its gate terminal, and turns "OFF" when a Boolean value of 0 is applied at its gate terminal. Similarly, a PMOS transistor turns "ON" when a Boolean 0 is applied at its gate terminal, and turns "OFF" when a Boolean 1 is applied at its gate terminal. More information on transistor operation and their use as switches can

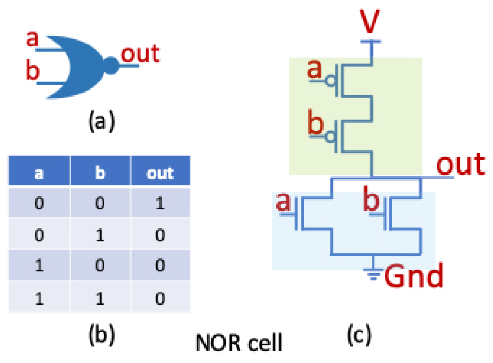

PMOS transistor
Source
Gate
Drain
Conduction from Source to Drain established when V(Source, Gate) > V_{th}
1V
Source
Gate
0V
Drain
0V

be found in [3] and [10].
Figure 5. PMOS transistor as switch.

Cell implementation using transistors

Cells that implement primitive Boolean functions are constructed using transistors as switches. In Complementary MOS cells, or CMOS cells, a network of PMOS switches is used to implement the rows of truth table of the function for which the output value is a Boolean 1, and a network of NMOS switches is used to implement the rows of the truth table for which the output value is a Boolean 0. In the CMOS implementation of a cell, each output wire of the cell has some capacitance to the ground terminal. The value at the output of the cell depends on whether this output capacitor is charged to the supply voltage or is discharged to the ground (or 0 volts). The PMOS network of a cell is used to implement the Boolean 1 value at the output by establishing a conducting path from the supply voltage to the output wire through "ON" PMOS switches for the output capacitor to be charged to the supply voltage. The NMOS network of a cell is used to implement the Boolean 0 value at the output by establishing a conducting path from the output wire to the ground through "ON" NMOS switches for the output capacitor to be discharged to the ground.

The implementation of a CMOS NOR cell, for which the symbol and truth table are given in Figure 6(a) and 6(b) respectively, is shown in Figure 6(c). In Figure 6(c), the PMOS network is shown shaded in light green

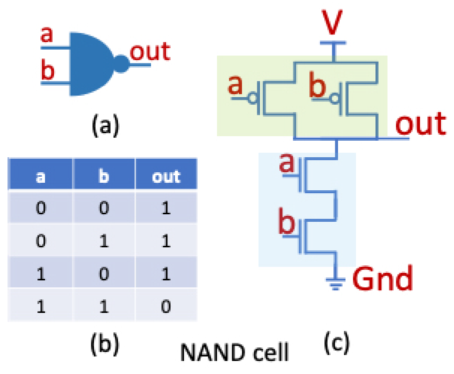


a	b	out
0	0	1
0	1	0
1	0	0
1	1	0

Figure 1. (a) NOR cell symbol (b) NOR truth table (c) CMOS NOR cell.

and the NMOS network is shown shaded in light blue. To understand the operation of this cell, consider the first row of the truth table in Figure 6(b) where a = 0, b = 0, and out = 1. As discussed in Section II-B, since a PMOS transistor turns on when its gate terminal is a Boolean 0, applying a = 0 and b = 0 will turn on the two PMOS transistors connected in series thereby connecting the out terminal to the supply voltage. This results in the capacitor at the out terminal being charged to the supply voltage, and hence attaining a Boolean value of 1 as specified in the truth table.

Similarly, Figure 7(c) shows the implementation of a CMOS NAND cell, for which the symbol and truth table are given in Figure 7(a) and 7(b) respectively. More information on CMOS logic gates can be found in [9].



a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

Figure 2. (a) NAND cell symbol (b) NAND truth table (c) CMOS NAND cell.

Simulation of a cell to determine properties

Each cell in a standard cell library is characterized through analog simulations of its transistor-level circuit description to determine its properties, such as propagation delays of events at its inputs to its outputs, and the power it consumes during its use. In this paper, we only focus on the propagation delays.

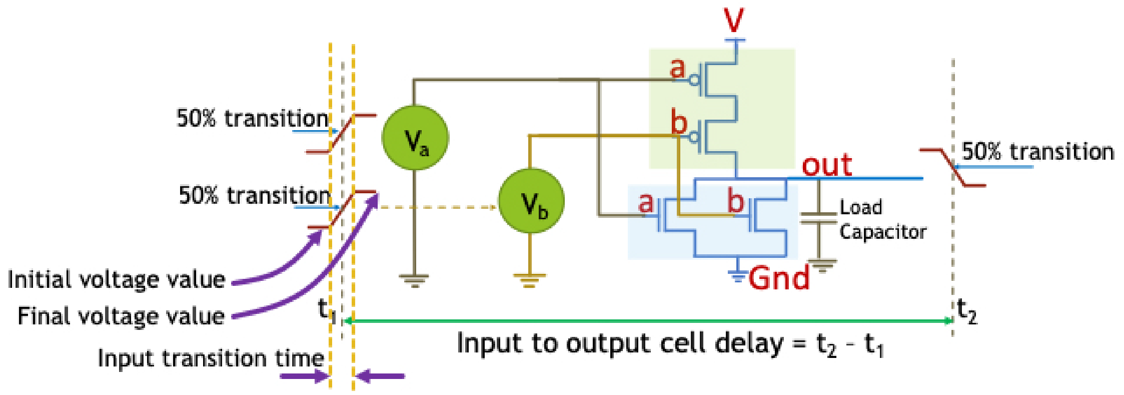


Figure 3. Circuit simulation setup for cell delay determination.

The circuit that is simulated includes (a) the transistor-level description of the cell that provides the connections between the transistors that make up the cell, (b) the connection of the cell's power supply to a DC voltage source (labeled as V in Figure 8), (c) the ground connection (labeled as Gnd in Figure 8), (d) capacitor connected to the output terminal to model the load provided by other cells when this cell is part of a circuit netlist, and (e) a voltage source connected to each of the cell's input terminals to provide the input event for the cell. Each voltage source is modeled as a piece-wise linear voltage waveform corresponding to one of four events that can be applied to that input, namely 0 to 0, 0 to 1, 1 to 0, or 1 to 1 (shown in Figure 3). Each 1 to 0 or 0 to 1 event has a specified transition time and is modeled as a linear ramp of the voltage source from the initial voltage value to the final voltage value over this transition time. The piece-wise linear voltage waveforms at each input are set up such that the start and end points of transitions at all the inputs are aligned in time.

The circuit description along with an input event combination is created in a specific file format along with commands to measure the input-to-output event propagation delay. This file is then provided to an analog circuit simulator that simulates the circuit in incremental time steps. At each time step, the simulator (a) uses models for transistors and other circuit components (b) uses appropriate process technology parameter values for these models, and (c) solves network equations to determine the voltage and current values at locations within the circuit in response to applied input event. In this manner, the voltage waveform at the output terminal is generated as a time series in response to the input event waveforms. The circuit file format and the analog circuit simulator that is used is usually either an open source or commercial version of the original Berkeley SPICE [11] file format and simulator. This work uses the open source NGSPICE [4] simulator for the experiments instead of commercial simulators such as HSPICE [5] and SPECTRE [6] due to cost and licensing considerations.

In this work, the input-to-output event propagation delay is measured as the difference between the time at which a transition at the cell's output reaches 50% of its final voltage value and the time at which the transition at each of its inputs reach 50% of its final voltage value. An example simulation setup for the NOR cell in Figure 6(c) is shown in Figure 8 for the case where each of the cell inputs, a and b, has a rising transition (a 0-to-1 event). The delay is measured as ($t_2 – t_1$). In this work, we also measure this delay for different transition times for the voltage sources at the input terminals.

As described above, determination of the input-to-output propagation delays of a cell involves simulating all possible input event combinations at its input terminals. As mentioned in Section IIA, this work (DeepSPICE) aims to reduce the overall time needed for the determination of these delays by simulating only a subset of input event combinations and using machine learning on these results to predict the delays for the rest of the input event combinations. DeepSPICE is based on the following observations. In each cell, there are paths through its PMOS transistor network from power supply to output that charge the output, and paths through its NMOS transistor network from output to the ground terminal that discharge the output voltage. One or more of these paths can be activated by an input event combination and can contribute to a different input-to-output delay. As the size of a cell increases, the number of paths to charge and discharge the output can increase exponentially with the number of inputs while the number of transistors in the cell increases at a significantly smaller rate. The hypothesis is that even though the number of charge (or discharge) paths can increase exponentially with the number of inputs, these paths share a significant number of transistors since the number of transistors does not increase as much. So, it should be possible to learn the charge (or discharge) patterns of a cell, and therefore its delays, by exercising only a subset of these paths using the corresponding input event combinations. The Deep-SPICE approach is described in the next section.

DeepSpice Approach

The DeepSPICE approach for a cell is illustrated in Figure 9. There are two parts to this approach. The first part, depicted in steps within the shaded box in Figure 9, deals with creating the training set to be used for machine learning. The total set of input event combinations (set T) is computed, and a small random subset (set TR) is chosen from set T for determining the input-to-output delays through circuit simulations as described in Section IID. The simulations are repeated for 3 different transition times, and the propagation delays are recorded.

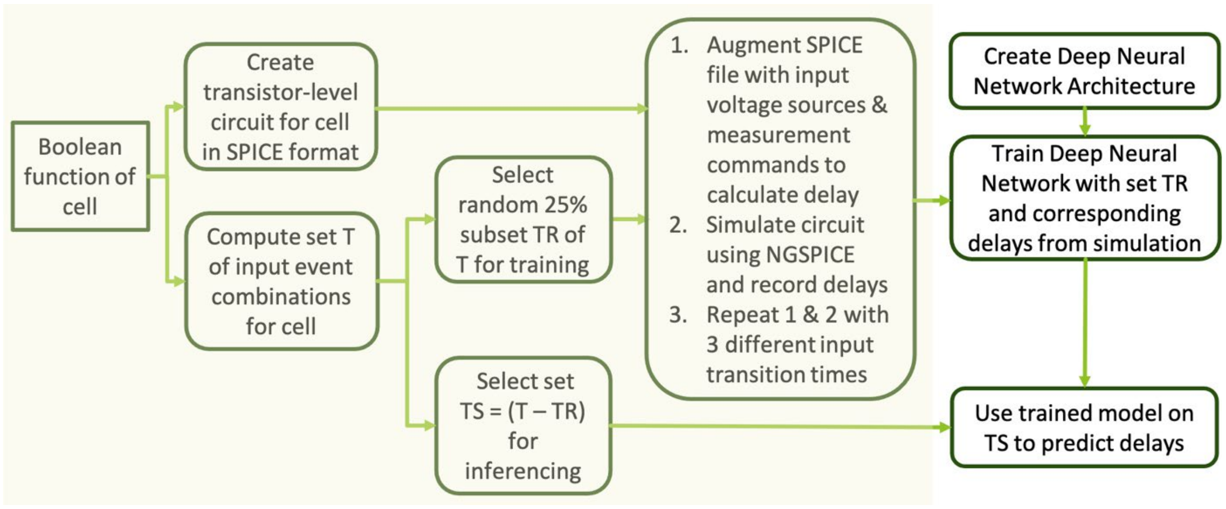

Boolean function of cell
Create transistor-level circuit for cell in SPICE format
Compute set T of input event combinations for cell
Select random 25% subset TR of T for training
Select set TS = (T – TR) for inferencing
1. Augment SPICE file with input voltage sources & measurement commands to calculate delay
2. Simulate circuit using NGSPICE and record delays
3. Repeat 1 & 2 with 3 different input transition times
Create Deep Neural Network Architecture
Train Deep Neural Network with set TR and corresponding delays from simulation
Use trained model on TS to predict delays

Figure 4. DeepSPICE Flow.

In the second part, a Deep Neural Network (DNN) built with dense layers in Keras is constructed and used for training. The neurons per layer and the number of layers are scaled based on the number of inputs in the cell. The ReLU function is used to model the non-linearity inside each neuron. The input features of the DNN are the initial input voltage, the final voltage, and the transition time at each input terminal of the cell, and the output feature is the measured input-to-output delay (as described in Section IID) for the corresponding input event combination and transition time. These features are shown in Figure 8. The DNN is trained on the input event combinations in TR along with their corresponding delays, and the DNN is used to predict the delays for the input event combinations in the remaining set, TS = T – TR.

To measure the effectiveness of the DeepSPICE approach, two metrics are used. The first is the quality metric which represents the error in prediction. This is measured by the Normalized Root Mean Square Error (NRMSE), which is defined below.

$$Error\ (\%) = 100\ x\ NRMSE \quad = \frac{RMSE\ between\ simulated\ and\ predicted\ delays\ for\ TS}{Mean\ delay\ of\ test\ set\ TS}$$

The speed-up obtained by DeepSPICE is measured by the Acceleration Factor which is defined below.

$$Acceleration\ Factor = \frac{time\ for\ baseline}{time\ for\ DeepSPICE}$$

where time for baseline is the time needed to simulate all input event combinations (set T) for a cell, and time for DeepSPICE is the summation of the time taken for simulating the subset TR, the training time, and the time for predicting the delay of subset TS.

Experiments and Results

The DeepSPICE approach was used on 14 CMOS cells that implement different Boolean functions. Two 2-input cells, three 3-input cells, three 4-input cells, three 5-input cells, two 6-input cells, and one 7-input cell were created for this experiment and for each of these cells, the CMOS transistor network was specified in the SPICE file format. The characteristics of the cells used for the experiments are given in Table 1. The function implemented by each cell is given in column 5 of Table 1. Similar to the transistor implementations discussed for a NAND and NOR cell in Section IIC, the Boolean function of each cell in Table 1 is implemented using a PMOS transistor network and an NMOS transistor network. The total number of PMOS and NMOS transistors used to implement each cell is given in column 3 of Table 1. The total number of baseline simulations is given in column 4 of Table 1.

Table 1. Characteristics of cells used for experiments.

Cell	Inputs	Number of transistors in cell	Number of baseline simulations	Boolean function implemented by the cell
1	2	4	18	$\overline{ab}$
2	2	4	18	$\overline{(a+b)}$
3	3	6	90	$\bar{a} + \bar{b}\bar{c}$
4	3	6	90	$\bar{a}\ (\bar{b} + \bar{c}\ )$
5	3	10	96	$\bar{b}\ (\bar{a} + \bar{c}\ ) + \bar{a}\bar{c}$
6	4	10	384	$\bar{a}\bar{b} + \bar{d}(\bar{b} + \bar{c})$
7	4	12	378	$\bar{a}\bar{b}\bar{c} + \bar{d}(\bar{b} + \bar{c})$
8	4	12	360	$\bar{a}\bar{d} + \bar{b}\bar{c}(\bar{a} + \bar{d})$
9	5	10	1530	$\bar{b}\bar{c} +\ \bar{a}\bar{d}\bar{e} + \bar{c}\bar{d}$
10	5	12	1512	$\bar{a}\bar{b}\bar{e} + \bar{b}\bar{c} + \bar{a}\bar{d}\bar{e} + \bar{c}\bar{d}$
11	5	16	1536	$\bar{a}\bar{b}\bar{e} + \bar{a}\bar{c}\bar{e} + \bar{b}\bar{c} + \bar{a}\bar{d}\bar{e} + \bar{c}\bar{d}$
12	6	22	5760	$\bar{a}\bar{c}\bar{f} + \bar{a}\bar{c}\bar{e} + \bar{a}\bar{c}\bar{d} + \bar{a}\bar{d}\bar{e} + \bar{b}\bar{c}\bar{e} + \bar{c}\bar{d}\bar{e}$
13	6	30	6120	$\bar{a}\bar{c}\bar{f} + \bar{a}\bar{c}\bar{e} + \bar{a}\bar{c}\bar{d} + \bar{a}\bar{d}\bar{e} + \bar{b}\bar{c}\bar{e} + \bar{c}\bar{d}\bar{e} + \bar{a}\bar{e}\bar{f} + \bar{b}\bar{d}\bar{f}$
14	7	30	24426	$\bar{a}\bar{b}\bar{d} + \bar{b}\bar{c}\bar{e} + \bar{c}\bar{d}\bar{f} + \bar{d}\bar{e}\bar{g} + \bar{a}\bar{e}\bar{f} + \bar{b}\bar{f}\bar{g}$

Table 2. DeepSPICE results for the case when 25% subset is used for training.

For each cell, the transistor length was set to 0.18 micrometer, while the widths of different transistors

Cell	Inputs	Time for baseline (sec)	Trial 1		Trial 2		Trial 3		Mean Acceleration Factor	Mean Error (%)
			Time for DS (sec)	Error with DS (%)	Time for DS (sec)	Error with DS (%)	Time for DS (sec)	Error with DS (%)		
1	2	0.40	2.20	25.25	2.52	27.37	2.26	32.77	0.17	28.46
2	2	0.36	2.10	43.89	2.13	40.97	2.14	48.46	0.17	44.44
3	3	2.97	2.92	33.69	2.95	28.95	2.97	31.39	1.01	31.34
4	3	2.22	2.74	30.39	2.56	26.70	2.61	22.28	0.84	26.46
5	3	2.89	2.55	28.58	2.66	25.50	2.62	26.16	1.11	26.75
6	4	11.21	7.94	22.43	8.03	17.74	8.30	26.39	1.39	22.19
7	4	10.62	7.75	25.36	7.13	27.16	7.33	27.64	1.44	26.72
8	4	12.80	8.02	26.05	7.71	25.76	8.55	29.13	1.58	26.98
9	5	45.16	29.54	9.64	29.47	10.10	29.28	8.69	1.53	9.48
10	5	39.61	27.95	9.70	27.82	9.95	27.85	11.24	1.42	10.30
11	5	49.52	29.70	13.14	29.55	9.76	30.97	10.68	1.65	11.20
12	6	246.55	122.53	16.07	125.60	5.30	122.41	15.63	2.00	12.33
13	6	309.54	147.19	7.11	148.01	8.99	147.14	6.95	2.10	7.68
14	7	1404.80	631.54	6.57	632.72	6.53	628.36	5.82	2.23	6.31

in the cell were specified to provide a reasonable output transition time. A load capacitor of 5fF was connected at each cell's output. The temperature for simulation was set to 27C. For each input event combination for a cell, the delays
were computed for three different input transition times, 10ps, 20ps and 30ps. As mentioned in Section IID, the simulations were performed using the open source NGSPICE circuit simulator and transistor models that were provided with the simulator.

For a given cell, the total number of input event combinations, called NIC, is given by twice the product of the number of truth table rows for which the output is 1 and the number of truth table rows for which the output is 0. This number captures all possible transitions that can occur at the output of that cell. The total number of baseline simulations, BS, for the set T is then NIC x 3, to account for three different transition times at the input terminals.

For each cell, DeepSPICE was performed with two training options. The first option used training with a 25% subset of the baseline simulations, and the second option used training with a 30% subset of baseline simulations. For each training option, 3 trials were performed with different randomly selected training sets. For each trial, 3 different DNN training and prediction runs, each with different initial DNN starting states, were performed and the mean results from the 3 runs were computed and recorded.

Results of the experiments with 25% subset of baseline simulations used for training are given in Table 2. Results with 30% subset of baseline simulations used for training are given in Table 3. In each table, each row contains the results for a different cell, with rows ordered in non-decreasing order of number of inputs of cell. The time for baseline simulations for each cell is shown in column 3. Each consecutive pair of the next 6 columns, column 4 through 9, provide the cumulative DeepSPICE time and the associated error in each of 3 trials. The final two columns provide the mean acceleration factor and the mean error with the DeepSPICE approach.

Table 3. DeepSPICE results for the case when 30% subset is used for training.

Cell	Inputs	Time for baseline (sec)	Trial 1		Trial 2		Trial 3		Mean Acceleration Factor	Mean Error (%)
			Time for DS (sec)	Error with DS (%)	Time for DS (sec)	Error with DS (%)	Time for DS (sec)	Error with DS (%)		
1	2	0.40	2.26	33.51	2.18	40.74	2.51	38.39	0.17	37.55
2	2	0.36	2.28	38.78	2.11	58.57	2.25	35.35	0.16	44.23
3	3	2.97	3.06	32.75	3.17	28.22	3.08	27.59	0.96	29.52
4	3	2.22	2.86	30.08	2.61	27.86	2.79	24.31	0.81	27..
5	3	2.89	2.73	28.48	2.84	25.30	3.01	26.47	1.01	26.75
6	4	11.21	9.96	13.83	9.99	18.95	9.95	19.32	1.12	17.37
7	4	10.62	9.61	26.60	9.99	24.24	10.15	21.91	1.07	24.25
8	4	12.80	9.77	25.12	10.46	16.49	10.16	26.94	1.26	22.85
9	5	45.16	36.33	8.82	35.27	8.58	35.93	7.88	1.26	8.43
10	5	39.61	34.64	8.93	34.23	8.11	35.46	8.82	1.14	8.62
11	5	49.52	38.03	9.15	37.03	8.98	37.34	9.30	1.32	9.14
12	6	246.55	146.91	4.94	148.29	4.59	146.69	4.68	1.67	4.74
13	6	309.54	179.00	6.55	176.51	8.08	177.44	8.33	1.74	7.65
14	7	1404.80	753.06	5.74	758.49	6.85	752.47	6.24	1.86	6.28

A graphical view of average acceleration factor (shown as a blue dot) and error (shown as an orange dot) obtained over the 3 trials for each cell using a 25% training subset is shown in Figure 10(a). A similar view is provided in Figure 10(b) for results when using a 30% training subset. In both figures, the X axis represents cell IDs with cells ordered in non-decreasing order of number of inputs.

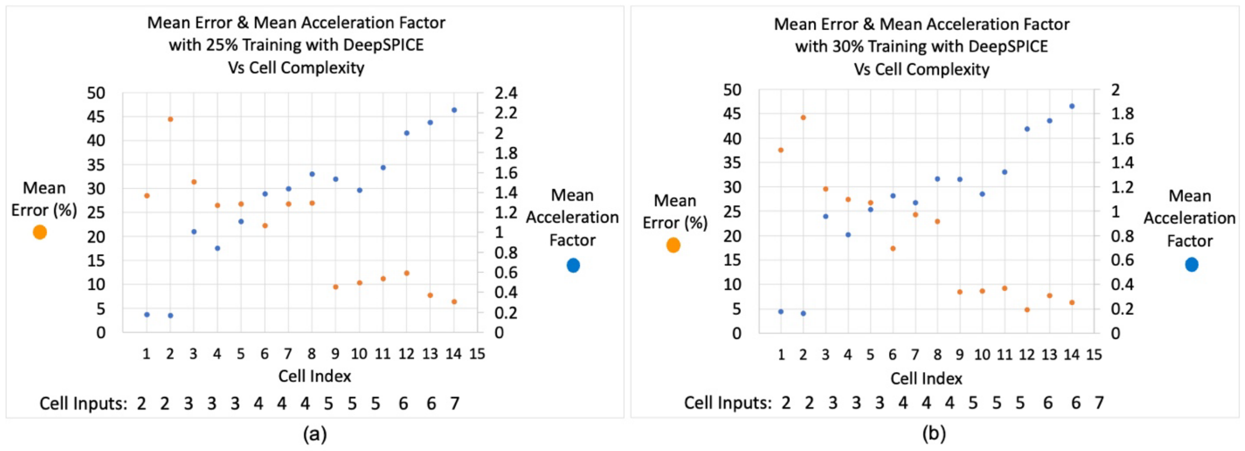


Figure 5. Mean Error and Mean Acceleration factor when using (a) 25% subset for training, (b) 30% subset for training.

From Figure 10(a), which corresponds to results using a 25% subset for training, it can be seen that for the 6 largest cells, with inputs ranging from 5 to 7, DeepSPICE is 1.42-2.23 times faster than the baseline approach while limiting the error between 6.31% and 12.33%. In fact, for the largest cell with 7 inputs, DeepSPICE performs fastest with 2.23 times acceleration over baseline with an error of 6.31%. For the 3 cells with 4 inputs, the results are mediocre – DeepSPICE is only 1.39-1.58 times faster than baseline while the error is in the range of 22.19-26.75%. For cells with at most 3 inputs, DeepSPICE provides only limited acceleration for 2 cells while it is actually slower than baseline for 3 cells, and the errors are at least 25%. Hence, for smaller cells, the baseline approach is better than DeepSPICE. From Figure 10(b), which corresponds to results using a 30% subset for training, it can be seen that for the 6 largest cells, DeepSPICE performs 1.14-1.86 times faster than the baseline approach, and it limits the error between 4.74% and 9.14%. For the 3 cells with 4 inputs, DeepSPICE is barely faster than the baseline approach by a factor of 1.07-1.26 times, with errors in the range of 17.37-24.25%. For cells with at most 3 inputs, DeepSPICE is slower than the baseline for 4 cells, and the errors are at least 25%. Overall, the trend is similar to the results from the 25% training subset (Figure 10(a)) with a reduction in both the acceleration factor and mean error.

In summary, DeepSPICE shows promising results, especially for large cells for which simulations are computationally expensive. This validates the hypothesis that for large cells, it is possible to learn their charging and discharging patterns, and hence the delays, from simulating a small subset of input event combinations.

Results on smaller cells are not impressive. This can be attributed to the fact that small cells have few input event combinations to begin with, and training with an even smaller set of input event combinations can result in the loss of significant information. Therefore, for small cells, delay computations with a full set of baseline simulations would be the recommended approach.

Future Work

Work is ongoing to improve DeepSPICE to reduce the mean error further. One approach is using the time series waveforms at the input terminals as features. Also, so far, the experiments were performed using a representation of cell using ideal conductors as wires to connect the transistors. In practice, cell implementations contain wires with non-zero resistances and certain pairs of conductors within cells having capacitances. The next step is to evaluate DeepSPICE on such cell implementations.

In this work, DeepSPICE was used only to predict delays. It can be extended to predict the power consumption of a cell for input event combinations. The DeepSPICE concept can also be extended to higher levels of design hierarchies to predict properties of circuits that contain an interconnection of cells as building blocks.

The current version of DeepSPICE used a DNN architecture for machine learning. One of the next steps is to explore a Recurrent Neural Network (RNN) architecture for this application and compare with the current results.

Conclusion

This work introduced DeepSPICE, a machine learning approach to accelerate the determination of the input-to-output propagation delays of cells, which are building blocks of digital circuits. Results using DeepSPICE demonstrate the promise of this approach, especially for large cells for which exhaustive simulations can be computationally very expensive.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

[1] Batten, Christopher. ECE 5745 Complex Digital ASIC Design. https://www.csl.cornell.edu/courses/ece5745/handouts/ece5745-T05-methodology-auto.pdf.

[2] Černý, David, and Josef Dobeš. "Deep Learning Neural Network Algorithm for Computation of Spice Transient Simulation of Nonlinear Time Dependent Circuits." Electronics, vol. 11, no. 1, 2021, p. 15., https://doi.org/10.3390/electronics11010015.

[3] Lee, Chuan-Zheng. Transistors. https://web.stanford.edu/class/archive/engr/engr40m.1178/slides/transistors.pdf.

[4] NGSPICE: Circuit Simulator - Oregon State University. https://web.engr.oregonstate.edu/~traylor/ece391/smith_NGSPICE_USERGUIDE_ECE391.pdf.

[5] "PrimeSim HSPICE the Gold Standard for Accurate Circuit Simulation." The Gold Standard for Accurate Circuit Simulation, https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html.

[6] "Spectre Simulation Platform." Cadence, https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-simulation-platform.html.

[7] Tan, Wei-Lii. "Machine Learning Overcomes Library Challenges at the Latest Process Nodes." Tech Design Forum Techniques, https://www.techdesignforums.com/practice/technique/machine-learning-overcomes-library-challenges-at-newer-process-nodes/.

[8] "What Is Library Characterization? – How It Works & Techniques." Synopsys, https://www.synopsys.com/glossary/what-is-library-characterization.html.

[9] Designing Combinational Logic Gates in CMOS. http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee141_f01/Notes/chapter6.pdf.

[10] MOS Transistors, CMOS Logic Circuits - Stanford University. https://web.stanford.edu/class/archive/engr/engr40m.1178/slides_sp17/lecture08.pdf

[11] The Spice Home Page. http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/

[12] Mentor, ST team on chip process characterisation. (2020, November 20). Eenewseurope.Com. https://www.eenewseurope.com/en/mentor-st-team-on-chip-process-characterisation/

[13] Nowe, Philip. "Timing (Analysis) is Everything – A How-To Guide for Timing Analysis." Circuit Cellar, https://circuitcellar.com/wp-content/uploads/2015/07/CC160-Nowe.pdf