UNIVERSITY OF CALIFORNIA
SANTA CRUZ

STATISTICAL MODELING OF SRAMS

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Hunter Nichols

March 2022

The thesis of Hunter Nichols
is approved:

Professor Matthew Guthaus, Chair

Professor Jose Renau

Assistant Professor Heiner Litz

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Statistical Modeling of SRAMs

by

Hunter Nichols

Characterizing static random access memories (SRAMs) is difficult but necessary to understand its properties. Choosing an optimal memory requires critical characteristics such as delay, power, and area. Characterization can be done accurately using SPICE but is slow. Analytical models aim to provide quick results to allow for rapid design iterations with the memory. These models do not require perfect accuracy but must maintain fidelity. This thesis presents the implementation of two Elmore-based models and statistical models for SRAMs. In addition, this thesis assesses the models' accuracy, fidelity, speed, and additional costs.

Acknowledgments

I want to thank my parents, grandparents, brother, and significant other for supporting me while pursuing higher education. I would also like to thank my advisor Matthew Guthaus for supporting and guiding my graduate studies. Lastly, I would like to thank my committee members Prof. Jose Renau and Assistant Prof. Heiner Litz for reading my thesis and providing invaluable feedback.

Chapter 1

Introduction

Static random access memory (SRAM) is a standard component in all modern System on Chip (SoC) and microprocessor designs. Closed source tools and license restrictions have gated experimentation with SRAMs causing a majority of designs to use a limited set of SRAM configurations. The advent of open-source memory compilers, such as OpenRAM [8, 9], has attracted system designers to explore new architectures to measure the impact on design power, performance, and area (PPA) [3, 4]. Characterization of the SRAMs can dictate the turnaround of these designs, so methods that produce fast, accurate, and faithful results are desired.

Characterization gauges the performance of the SRAM. Delay, power, and slew are typical measurements from the SRAM during characterization. Multiple sets of these metrics are measured for the SRAMs while performing different operations such as read or write. Characterization is processed around several corners to define its worst-case conditions. Optimization tools use the metrics from characterization to pick memory configurations that best fit the project's goals.

Several types of characterizations can produce this information. Silicon characterization directly measures a fabricated circuit. SPICE characterization simulates the behavior of an SRAM given its circuit description, device models, and operation conditions. Analytical characterization uses simplified device models and statically estimates only the critical portions of the SRAM. Statistical modeling uses previously generated results to create a predictive characterization model.

Each characterization method balances accuracy and cost. Silicon measurements pro-

vide the most accurate characterizations, but the fabrication delay and monetary cost make this infeasible to rely on solely. SPICE simulation is the second-most accurate characterization method, but the speed is dependent on the size of the circuit. SRAMs can have a device count that ranges from tens of thousands to hundreds of thousands which is slow to simulate. In addition, characterization needs multiple simulations which further slows the process. Analytical characterization for SRAMs uses simplified models and only evaluates the necessary components associated with the operation to reduce the devices modeled significantly. Both impact the accuracy. The statistical model's speed is independent of the SRAM size, but the accuracy and speed depend on the amount of data and model type.

The firmly-established Elmore delay model [6] works well for analytical characterization. The fundamental equation only requires the circuit's load capacitance and output resistance to estimate. Elmore delay is pessimistic but maintains strong fidelity with the actual delay. The Elmore model needs to be modified or combined with other models for characterization. The simplicity and applicability of this analytical model has allowed itself to permeate into many tools.

CACTI [12] is a tool for modeling caches and memories and exclusively uses analytical models to calculate delay and power. It reduces the characterization size by estimating several predefined critical paths of the memory and ignoring unrelated devices. The Elmore delay model is used extensively in CACTI for calculating gate delays but with an alteration to account for slew.

Statistical methods are based on fitted data. Supervised models such as linear regression and neural networks can be used to represent the delay of the SRAM by using prior SPICE characterization results. These models are accurate enough to predict complete memory characterizations including different loads and process corners. Statistical models provide many attractive properties with some drawbacks that are not present in traditional Elmore models. Statistical models have extensive support in many Python packages [2, 13], and little hand-tuning is required to generate an accurate model. The base models provided in these packages can be used without modification or adjusted through model hyperparameters such as model inputs, model type, neural network layers, optimization algorithms, etc. Several works have begun to explore the use of neural network models in order to speed up SRAM parameter optimization [10, 11] with a focus on improving the accuracy by tuning hyperparameters of the

model.

This work aims to evaluate modeling techniques for a memory compiler to provide a quick and faithful estimation of simulation results. Two Elmore-based analytical characterizations were added to OpenRAM including a logical effort [14] based delay model and a delay model based on the CACTI memory modeling tool. Additionally, statistical model characterization was added and integrated into OpenRAM. This thesis aims to display statistical models as a competitive method of characterization that can be easily automated. This contribution to OpenRAM is open-source and available at:

https://github.com/VLSIDA/OpenRAM

The thesis is laid out as follows. Chapter 2 covers background on memory characterization which includes OpenRAM SPICE simulation and analytical characterization, CACTI delay modeling, and statistical models. Chapter 3 discusses the implementation of the three models in OpenRAM including the addition of a static timing graph within OpenRAM to facilitate these models. Chapter 4 displays the results of analytical characterizations from all three implemented analytical methods and compares them their fidelity, accuracy, and trade-offs with SPICE simulation.

Chapter 2

Background

Different characterization methods produce results in the same format but in vastly different ways. The output data from characterization can be represented using a Liberty file (LIB) [1]. LIB files contain the component's delay, slew, power, and area and a separate LIB file exists for each process, voltage, temperature (PVT) corner. SPICE simulation and analytical Elmore models are highly dependent on the structure of the SRAM for its characterization, while statistical models are dependent on the characterized SRAMs used to generate its data. The LIB file is valid as long as it has the necessary information, but there will be accuracy differences.

SPICE simulation, analytical models, and statistical models are the focus of this thesis, with OpenRAM and CACTI used as primary examples. General background is presented for statistical models to give insight into how they can be used for characterization.

2.1 SPICE Characterization

SPICE characterization uses SPICE circuit simulations to measure the performance of the SRAM. There are several available SPICE simulators including the open-source Ngspice or hspice from Synopsys which requires a license. The simulators will differ in speed, accuracy, and functionality. A SPICE file and stimulus file are required to simulate the SRAM. The SRAM SPICE file is the SPICE representation of the SRAM circuit primarily made of transistors but can include resistors and capacitors to model extracted parasitic elements. The stimulus file generates inputs to the SRAM SPICE file and defines measurements for the SRAM. These

measurements become the delays, slew, and powers represented in the LIB file.

Designing a stimulus file for characterizing an SRAM requires prior knowledge to simulate successfully. The stimulus controls the input control signals for the SRAM and input clock frequency, but the SRAM can fail depending on the settings. The control signals require knowledge about the SRAM itself, but the clock frequency is unknown and can require successive simulations to set a working period. The delay and slew of a read operation and powers of all operations are measured for the LIB file using the maximum frequency. Delays of write operations are unused in a LIB file but can affect the maximum frequency if the delay is larger than a read operation.

2.1.1 Verification

Important information regarding the memory is often not included in LIB files but is helpful to memory designers when understanding memory failures. From the perspective of a SPICE simulation, any condition that causes a measurement to fail indicates a memory failure. Most of these failures are solved by increasing the period in the stimulus file. Other failures that cause this behavior require tedious measurements and inspection of signals within SPICE. Write failures, read failures, and bit-flips are common SRAM issues that can be detected automatically with specific measurements in the stimulus file.

SPICE characterization is improved significantly by adding debug measurements. For example, the stimulus file can have voltage measurements on the bitcell storage nodes to check the functionality of write operations. These failures cannot be fixed within the stimulus file and will require changes to the SRAM. Detection of one of these failures will allow the characterization to fail quickly. Failure stop points can be defined to prevent infinitely repeating failed simulations. However, this stop point could consume hours of simulation time to reach. Adding additional measurements for these failures does not affect simulation time and should always be done if possible.

2.1.2 Simulation Example

OpenRAM provides SPICE characterization and several SPICE verification tests for its SRAMs. The memories are tested on several PVT corners to produce the corresponding LIB files. In addition to debug measurements in characterization, OpenRAM also maintains

functional tests and regression tests to catch errors when the SRAM is updated.

OpenRAM SPICE characterization will perform multiple simulations to get the necessary data for the LIB file. A feasible clock period is determined through simulations which is used as a starting point in a binary search of simulations to find the minimum clock period. This minimum clock period is used to find the delay, power, and slew of the SRAM over varying output load and input slew combinations where each case is a separate simulation. The leakage power is measured with a short simulation while the SRAM is disabled. Finally, the D flip-flops (DFF) used on the address, data, and control signals are simulated to find the setup and hold times using a binary search method.

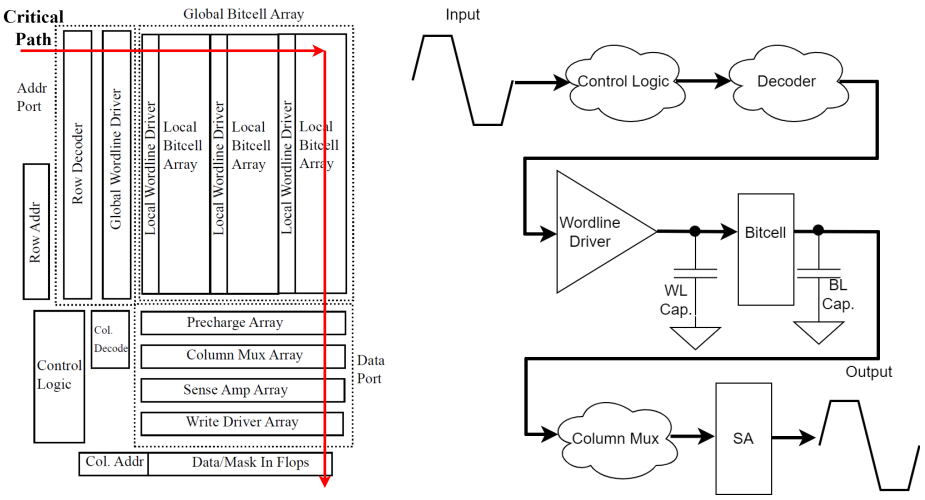
Each SPICE simulation that measures the power, delay, and slew requires multiple operations by the SRAM. The largest bit delay is chosen to represent all delays and powers for the SRAM as shown in Figure 2.1(a). Bitcells not connected to this path can be trimmed from the SPICE file to reduce simulation time while having a low impact on characterization accuracy. The fall delay is measured by reading a 0, and the rise delay is measured from reading a 1. Cycles to write these values to the cell are included along with reads and writes to neighboring cells to ensure the bitline is cleared of its previous value. Slew is measured in the same read cycles as the delay. Power is measured for the read and write operations of 0 and 1 to the target cell.

OpenRAM's functional tests are intended for potential failures that cannot be easily checked during characterization. The functional tests perform many random reads and writes to random locations in the SRAM and check if any operation failed. These tests help debug issues such as cross talk between wordlines and bitlines which can corrupt data between neighboring cells. Only specific bits are written to test the worst-case delay during characterization as additional SRAM cycles would make characterization even slower.

OpenRAM contains a set of regression tests that run when OpenRAM is updated. Delay characterization, functional simulations, DRC, and LVS checks are performed during regression. Any update that causes a failure in the SRAMs is likely to be caught by these tests.

2.1.3 Analytical Characterization

Analytical characterization serves as a faster option than SPICE characterization at the cost of developing the models. SPICE simulation is accurate but time-consuming, causing



(a) SRAM with critical path

(b) Critical Path Isolated

Figure 2.1: Instead of simulating the entire bitcell array, analytical methods estimate the wordline and bitline by lumping the capacitance.

characterization to take over 24 hours for large SRAMs potentially. However, running SPICE simulations only requires defining or acquiring transistor models. Analytical models require developing a large set of parameters that replicates a simplified SPICE simulation to provide a fast and faithful characterization.

Analytical characterization first aims to reduce the circuit to only necessary components and reduce the complexity of the calculations. Reducing the circuit involves only evaluating the critical path while accounting for capacitive loads affecting that path. The equations are simplified from SPICE to reduce calculation time. Elmore delay is a common choice because it uses minimal parameters and has good fidelity. This model will cause accuracy loss, but this is accepted as long as fidelity is maintained.

Knowing the SRAM structure is critical to building an analytical model. Using knowledge about the structure, such as array size, allows the analytical model to estimate and reduce many components in the SRAM. As shown in Figure 2.1, the wordline and bitline in the critical path of the SRAM. Rather than simulating all unused bitcells connected to the bitline, the analytical model can reduce them to the equivalent capacitance they impose on the bitline but requires knowledge of the array dimensions. Additionally, the dimensions of the array will depend on structures within the SRAM such as the column mux sizing. This information about

the structure must be incorporated to maintain the accuracy and fidelity of the analytical model.

2.1.4 Analytical Examples

Prior to this thesis work, OpenRAM used a basic Elmore model for its analytical delay. The model used Elmore delay for all calculations while considering low-swing bitline delays and PVT corners. Every module in the SRAM defined a delay function, while the top module summed the delays along the critical path. A variation of the Π RC model is assumed for the wordline and bitline wire delays. This work details the logical effort based analytical model that replaced OpenRAM's original model.

The delay model in CACTI is very similar to OpenRAM's Elmore delay model. The CACTI delay model is a modified Elmore model by Horowitz that considers the input slew. The SA and bitcell are modeled differently from gates to improve the accuracy. Like OpenRAM, CACTI defines a delay for each module and sums all delays within the critical path. The Π RC model is used in CACTI for improved accuracy over the lumped wire model.

2.2 Statistical Delay Modeling

Statistical, regression, and supervised machine learning (ML) models use pre-existing data to train a model to predict outputs of new inputs. Creating a statistical model for SRAM characterization requires previously characterized results. The training data should span the potential range of SRAM configuration inputs. SPICE characterization data is chosen for the most accurate model but can be expensive to obtain.

Choosing inputs for the model to use will depend on the model and task. Linear regression (LR) creates a prediction with a sum of inputs multiplied by trained weights. The equation below represents a basic form of LR

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + ... + w_m x_m \tag{2.1}$$

The inputs to the model, $\boldsymbol{x}$, can be anything that represents the model. For SRAMs, portions of the configuration are used as inputs such as the number of words and wordsize. The inputs can include any information assumed relevant including global characteristics such as PVT corner or configuration-specific inputs such as delay chain size. The number of model inputs

will increase its complexity, size, and prediction speed. The most effective inputs for model accuracy are chosen, and redundant inputs are trimmed.

The weights are chosen to minimize an objective loss function which is typically the sum of square errors

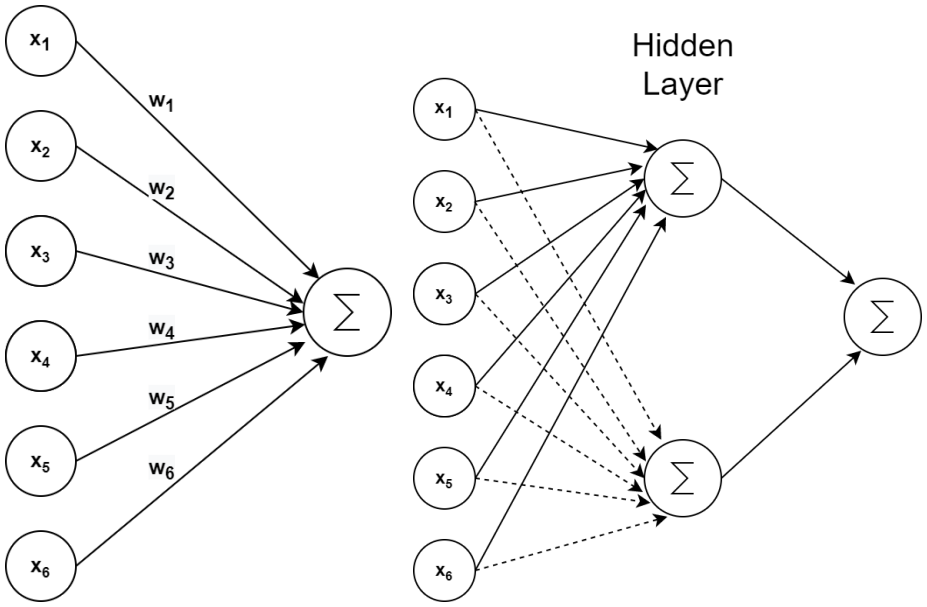
$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(\boldsymbol{x_n}, \boldsymbol{w}) - l_n]^2 \tag{2.2}$$

The loss equation is quadratic, so the derivative with respect to the weight will be linear, and one solution will exist. This process defines the optimal weights given a set of N inputs and output labels, l_n, from a given dataset.

LR is simple but can be used to create accurate SRAM models. Elnaz [5] explored using LR models as a compact representation of the Pareto-optimal database for SRAM and cache designs. The data used for their model comes from CACTI. Each configuration generates many delays and powers, but only a subset near the best energy-delay product is used for the model. The inputs to the LR model were tuned using logarithms, square root, and polynomial functions to improve the model's accuracy. They achieved an average accuracy of 2.99% and a worst-case of 17.98% when predicting the delay of SRAMs. This thesis explores the accuracy of LR models built from a limited set of SPICE characterizations.

Inputs can be tuned and scaled to improve the accuracy of LR, but some errors will always exist because the outputs are non-linear. Quadratic regression or any other polynomial regression can yield better models but can grow exponentially complex with the number of inputs. These models require hand tuning and reducing the polynomial features as they increase the model complexity while providing minor improvements to the accuracy. For SRAMS, quadratic regression can produce better results because power and delay are inherently non-linear. Compilers like OpenRAM add new features and will need new inputs to the statistical models. Continuously adding inputs to the model may not be maintainable over time, so LR is used as the default regression model while other options are explored to address more accurately modeling non-linear outputs.

More complex models such as neural networks can also represent non-linear functions by adding intermediate layers between input and output. Linear and polynomial regression can be viewed as having a single input layer and a single output layer. In contrast, neural



(a) Linear Regression (b) Neural Network

Figure 2.2: Linear regression and neural networks share many of the same functions, but hidden layers add additional complexity for learning patterns in the inputs.

network models have additional layers between the input and output layers. These are referred to as hidden layers. The number of hidden layers are variable and have their own weights that determine the output value as shown in Figure 2.2. Models with more than three layers are referred to as deep neural networks (DNN). The number of hidden layers, number of nodes in hidden layers, and connectivity of the layers are all hyperparameters left up to the designer to choose that best fits their desired accuracy and model complexity. Last and Schlichtmann [11] explored how changing the weights are shared between outputs affects the accuracy in models predicting SRAM outputs.

These models do not have readily available closed-form solutions. Optimization techniques such a gradient descent can find the minimum to these functions but require iterative steps. This process is referred to as model training, and the training time is dependent on the model complexity and the amount of training data.

Statistical models are much faster than SPICE simulation but at the cost of data acquisition. In the case of memory characterization, the data is expensive because SPICE simulation is slow for large circuits. Therefore, generating the data could be infeasible when a large set

of SPICE simulations could potentially take days. However, assuming data collection for the model is infrequent, the cost is low if there is a long gap between data collecting. The main benefit of this model is that minimal information about the SRAM internals is needed to create accurate predictions. The SRAM configuration is presented as inputs to the model such as the number of words, word size, words per row. The model is trained with the characterization results from SPICE simulations of those configurations. SRAM configurations can be drastically different, but as long as enough data is provided to the model, it can learn complex patterns that are typically directly specified in an Elmore model.

Chapter 3

Implementation

This chapter details the implementation of the analytical SRAM models and statistical models within OpenRAM. The analytical models, Logical effort (LE) and CACTI-OpenRAM (CO), perform characterization using a combination of static timing and a graph. The statistical model is maintained in OpenRAM using automated scripts to generate SPICE simulation training data.

3.1 Analytical Models

The analytical SRAM models implemented for this thesis use a static timing graph. The graph determines the paths from input to output, and the timing functions implemented per model statically estimate the delay of each stage in those paths. This section will detail how the graph is implemented to accommodate the analytical models and enable different features within OpenRAM.

3.1.1 Static Timing Graph

OpenRAM generates SRAMs using modular structures which can be easily replicated in a graph. OpenRAM frequently changes the SRAM structure due to improvements in timing reliability, layout, or added features such as partial word writes to accommodate RISCV architectures. These structures are modular to easily incorporate and support the many possible configuration options OpenRAM offers. A graph can represent any of these configurations and

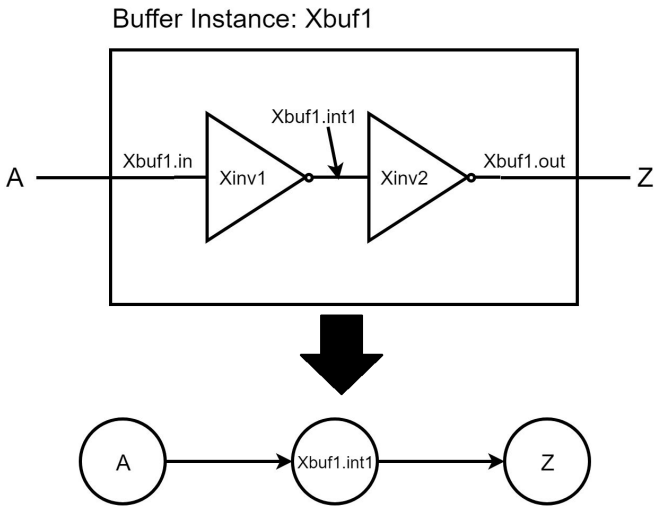


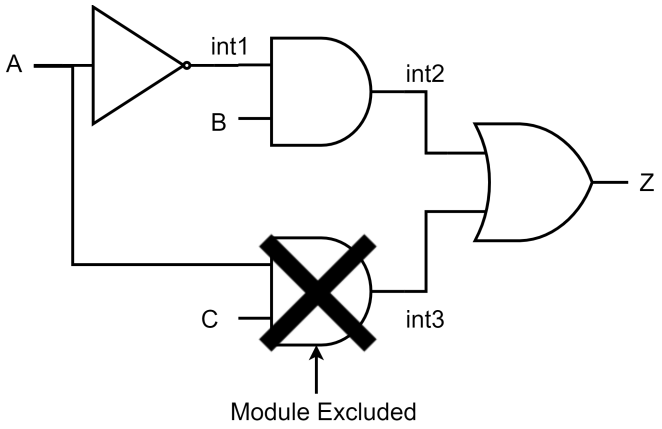
Figure 3.1: Example of an OpenRAM buffer graph represented as two inverters. The internal port net names are discarded and instantiated nets are used instead.

only relies on the connections between the modules.

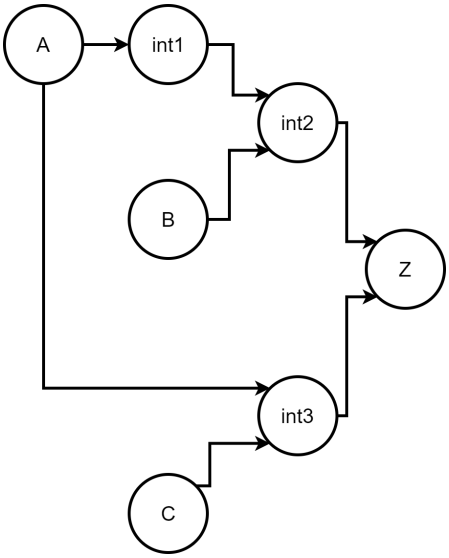
The graph was implemented to support static timing for analytical characterization and is inspired by SPICE hierarchy. It is a directed graph with wire nodes representing vertices and edges representing devices as shown in figure 3.1. For example, a graph representing a 2-input NAND gates have three vertices A, B, and Z and two edges from (A,Z) and (B,Z). Any module that defines a delay function can become part of the graph, or its components must define delay functions i.e. an AND gate made up of a NAND and NOT gate must either define its own delay function or delay functions must be defined by the NAND and NOT gates. Higher-level modules such as the bank or bitcell array do not require a delay definition as long their component sub-modules define delays.

The graph is first generated as a representation of the SRAM. A graph can be generated starting from any module, but generating a graph from the SRAM module is needed for characterization. The module recursively calls a graph formation function on its submodules. Inputs, outputs, and intermediate signals become vertices on the graph, while connections between modules define the graph's edges. By default, every input is connected to every output, and the module must override this function if it does not match its behavior.

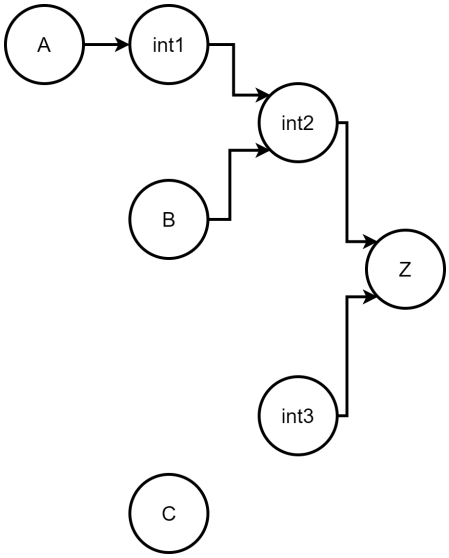
False paths in the graph are handled prior to their formation. Modules unrelated to



(a) Original Circuit



(b) Without exclusion



(c) With exclusion

Figure 3.2: Excluding specified modules prevents false paths in the timing graph.

the read delay are excluded from the formation process, as shown in Figure 3.2, to prevent false paths in the timing such as the precharge circuitry which connects the complementary bitlines in each column. Precharging occurs before the SRAM read operation is possible and cannot be part of the critical path. Path exclusion is handled on a case-by-case basis as excluding too many modules can affect paths of interest.

After the graph has been generated, paths can be generated between an SRAM input and output to the graph. The graph performs a depth-first search and records every path that

connects the input and output. For the purpose of characterization, the timing from clock to target output bit will be used to define the SRAM delay. The graph returns a list of nodes in the path, and edge delays of these nodes can be queried to determine the total path delay.

3.1.2 General Model Implementation

Implementing an analytical model with the static timing graph requires at least defining delay and load for all base modules i.e. a module that does not contain any submodules. These modules are the parametrized gates (pgates) and custom cells. The graph provides external loads to the model's delay function by querying the input load of the modules connected to the current stage. Analytical models can override the minimum requirements and add more if needed.

Pgates are gates in OpenRAM that generate an automated layout, and this information also helps build its delay function. Each model in OpenRAM defines delay and load functions for the NAND, NOR, and NOT pgates. Single transistors like those used for pass gates also have a delay defined. OpenRAM generates pgates automatically, so all dimensions of the gates and transistor sizes can easily be queried within the module to define these functions.

In OpenRAM, there are a collection of handmade gates that are required by each technology. These gates either have a significant effect on the delay or area of the SRAM and would be too inefficient to be generated parametrically. These include the bitcells, replica bitcells, dummy bitcells, sense amplifiers, tri-gates, DFFs, and write drivers. The bitcells must be created by hand to optimize for area which causes the dimensions of the bitcells to ripple to other gates and in the SRAM enforcing that they stay within specific dimensions relative to the bitcells. The non-bitcell modules affected are the SA, tri-gates, and write drivers. These devices are placed vertically below (or above) the bitcells requiring widths matching the bitcells. The bitcells in OpenRAM are as thin as possible, making the other devices too challenging to design parametrically.

Custom cells require a more explicit definition in this scheme because the corresponding modules do not define and generate these cells. OpenRAM maintains modules for these custom cells, but they are skeletons that calculate the dimensions of the provided GDS to provide layout information to the other OpenRAM modules i.e. no information about the transistors is maintained in the module. Therefore, each technology needs to specify information about the

custom cells to create their delay functions. Each custom cell relevant to the critical path has parameters defined in corresponding technology files. These are almost entirely the transistor sizes of the cells but depend on what the model needs.

3.1.3 LE Model Implementation

Implementing the LE model using the graph requires defining each base module's input capacitance and delay. The LE model delay function uses the gate's electrical effort, logical effort, and parasitic delay. These can primarily be derived by using values within each module or requires definitions within the target technology file. The latter is required for all custom cells in OpenRAM.

The logical effort of a gate is dependent on its function. The LE model implements this as $\frac{C_{\text{in}}}{size*C_{\text{inv}}}$. C_{in} is dependent on the size of the gate, so that is factored away, and scaling by the capacitance of the minimum sized inverter is by the definition of logical effort. Non-CMOS circuits in OpenRAM, such as bitcells or sense amplifiers, use estimated values for logical effort parameters.

The LE model also depends on the parasitic delay of the gates. All gates in the LE model have parasitic delays as multiple of the minimum sized inverter parasitic delay. These are the same as provided by Sutherland [14]. Estimates are made for all other circuits not defined here. The minimum sized inverter parasitic delay is measured in SPICE for each technology in OpenRAM.

The LE model uses τ_{LE} time units for all calculations. τ_{LE} is defined as the delay of an ideal inverter with no parasitic delay driving another ideal inverter. This delay is measured in SPICE for each technology in OpenRAM. τ_{LE} is measured by finding the total delay of the circuit then subtracting the parasitic delay.

PVT corners and slews are not directly incorporated into the LE model delay calculations. Instead, all corners are applied linearly to the model's delay as a difference from the nominal. For example, the nominal voltage for the SCMOS technology is 5 Volts, and the corner being characterized is 4.5 Volts or a 10% difference. This factor is applied multiplicatively to the final delay in addition to temperature and process differences. Slew is not used in logical effort, so it is not used in this model. This provides a reasonable corner model without many parameters needed for each technology.

The LE model only produces the information needed to generate a correct LIB file for the SRAM. Many of the debugging statements included in SPICE characterizations are not modeled here, and the critical path is assumed to be along the decoder-bitline path and the SA enable (SAE) timing is not considered.

3.1.4 CO Model Implementation

The CO model is an adaption of the CACTI delay model within OpenRAM. Adding the model to OpenRAM allows comparison against SPICE as CACTI does not generate a SPICE file for the memories it models. CACTI is a well-established modeling tool, so OpenRAM benefits from including a model that is thoroughly documented and maintained. The LE and CO models do not supersede each other, so both models are made available when using OpenRAM's analytical characterizer.

Memory Structure

The CO model is only based on a portion of the architecture modeled by CACTI. The arrays are made of banks that contain subbanks. A subbank is then made of multiple mats. OpenRAM currently does not support multi-banking, so a CACTI mat with one subarray best represents the SRAMs produced by OpenRAM. Figure 3.3 shows both the CACTI subarray and OpenRAM array. They closely resemble each other, except OpenRAM has options for local arrays and does not offer column muxing after the SAs.

Delay Characterization

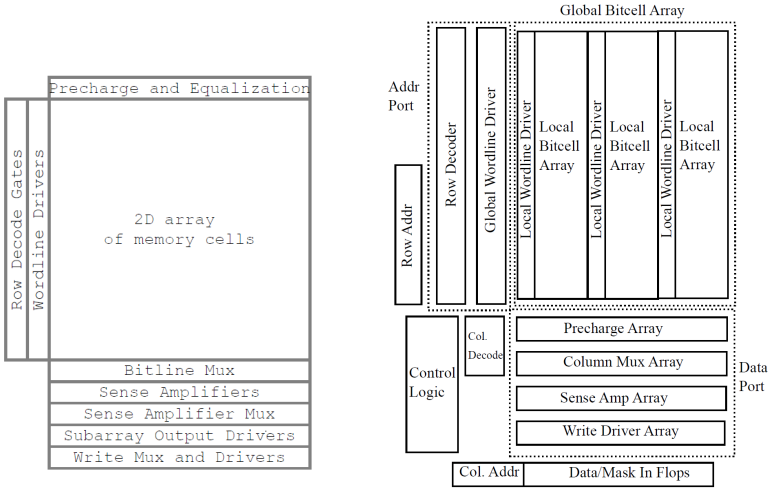
Within a mat, the CO model only considers a subset of available paths that are equivalent to OpenRAM's architecture. The following is how CACTI compares the delays of critical paths to calculate a maximum delay:

$$T_{\text{mat}} = \max(T_{\text{row-decoder-path}}, T_{\text{bit-mux-decoder-path}}, T_{\text{senseamp-decoder-path}}) \tag{3.1}$$

$$T_{\text{row-decoder-path}} = T_{\text{row-predec}} + T_{\text{row-dec-driver}} + T_{\text{bitline}} + T_{\text{senseamp}} \tag{3.2}$$

$$T_{\text{bit-mux-decoder-path}} = T_{\text{bit-mux-predec}} + T_{\text{bit-mux-dec-driver}} + T_{\text{senseamp}} \tag{3.3}$$

$$T_{\text{senseamp-decoder-path}} = T_{\text{senseamp-mux-predec}} + T_{\text{senseamp-mux-dec-driver}} \tag{3.4}$$



(a) CACTI Subarray (b) OpenRAM Array

Figure 3.3: The bitcell array and peripheral circuitry of a CACTI subarray [12] and OpenRAM array [7].

Equations 3.3 and 3.4 represent the delay of the bitline mux decoding and delay of the SA mux decoding path, respectively. The SA mux is an optional structure in CACTI that is used if the maximum size of the bitline mux is not sufficient to achieve the desired words per row. OpenRAM does not have a SA mux, so equation 3.4 is unused. In addition, OpenRAM access time is dependent on the start of the negative edge of the clock. However, decoding is assumed completed before the clock's negative edge, causing equation 3.3 to be irrelevant. The CO simplifies the delay to

$$T_{\text{mat}} = T_{\text{row-decoder-path}} \tag{3.5}$$

Wire Delay

Wires in the CO model are either ideal or non-ideal. Ideal wires have zero resistance and capacitance, while non-ideal wires have finite capacitance and resistance. Most wires within the CACTI are considered ideal, while the wordline and bitline are modeled as non-ideal wires. The CO model uses the the Π RC model to estimate wire delays as shown in Figure 3.4.

The CO model does not estimate the unit capacitance and resistance from technology

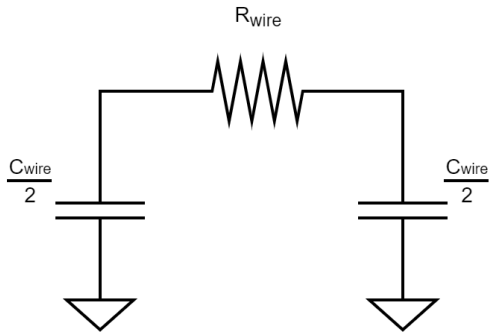


Figure 3.4: Π RC model used by CACTI to estimate wire delay more accurately than a lumped model [12].

parameters like CACTI. This information is sometimes provided by the technology in a Liberty Exchange Format (LEF) file and is true in the case of OpenRAM's available technologies: FreePDK45 [15] and scn4m_subm. RPERSQ and CPERSQDIST within the LEF define the unit resistance and capacitance, respectively. To translate these to unit length of the wire, the RPERSQ is divided and CPERSQDIST is multiplied by the wire minimum width the assumed layer.

Gate Delay

The CO model uses a Horowitz based delay function for all modules by default (Appendix A.1). This is an extension of Elmore delay but includes factors for the input slew. The equation is dependent on the τ, slew time, and charging thresholds. τ refers to the RC time constant of the stage. The charging threshold is the percentage voltage charge expected at that state and is typically 50%. The bitcell and SA stages have do not use Horowitz delay and have specialized delay functions.

The CO model calculates τ at each stage requiring each base module to define functions for on-resistance (R_{on}), drain parasitic capacitance, and gate capacitance. R_{on} is calculated in the CO model by calculating the effective current per width of the gate. This process is

detailed in CACTI and measured in SPICE for each technology using the following equations:

$$I_{eff} = \frac{I_H + I_L}{2} \tag{3.6}$$

$$I_H = I_{DS}(V_{GS} = V_{DD}, V_{DS} = \frac{V_{DD}}{2}) \tag{3.7}$$

$$I_L = I_{DS}(V_{GS} = V_{DS} = \frac{V_{DD}}{2}, V_{DS} = V_{DD}) \tag{3.8}$$

$$R_{\text{on-min-width}} = V_{DD}/I_{eff} \tag{3.9}$$

$$R_{\text{on}} = \text{stack} * R_{\text{on-min-width}}/\text{width} \tag{3.10}$$

A minimum-sized transistor is used in each simulation with a nominal corner. R_{on} is then calculated in the module based on the width of the transistor and stack size. Stack size is the number of transistors in series within a gate. For example, a 2-input NAND would have a stack size of two because of the two NMOS gates in the pull-down network.

Capacitances in the CO model are estimated based on CACTI's process for calculating drain dimensions from layout design rules. The two cases distinguished in the CO model are for folded and non-folded transistors. OpenRAM transistors increase in the vertical dimension when increasing transistor width. Folded transistors increase their width in the horizontal dimension for cases where the module cannot exceed a specific height.

The CO model uses several DRC rules to re-create these capacitance estimations. Table 3.1 describes the DRC rules and values within the OpenRAM technologies. Only the parameters used to estimate the diffusion dimensions were adapted from CACTI as displayed in Figure 3.5. The CO model uses OpenRAM's backend to query the height of the gate and does not require estimation.

Table 3.1: Parameters for estimating gate-drain dimensions.

W_{contact}	Contact Width
$L_{\text{poly\_to\_poly}}$	Minimum poly-to-poly spacing
$L_{\text{active\_contact\_to\_gate}}$	Minimum poly-to-contact spacing

Sources of Error

The CO model contains many parameters re-created from CACTI. Many parameters have detailed instructions in CACTI's documentation while others have little to none. These pa-

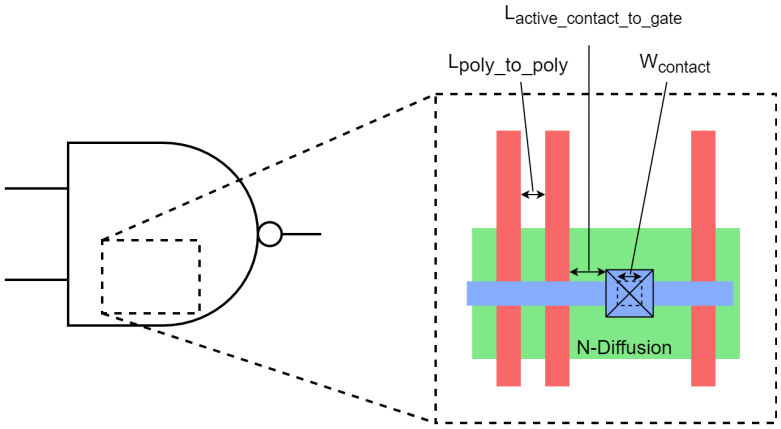


Figure 3.5: Assumed gate layout in CACTI for estimating drain dimensions [12].

rameters are estimated from CACTI and are a potential source of error when comparing the CO model against SPICE. Table 3.2 lists these parameters, their value in OpenRAM's scn4m_subm technology, and a description of the parameter.

Table 3.2: Estimated parameters in the CO model.

Parameter	Value	Description
i_on_p	0.000108 A/um	On current of PMOS. Calculated based on documentation but not used in CACTI.
n_to_p eff_curr_drv_ratio	N/A	Tuning parameter to estimate PMOS on current in CACTI. Not re-created in CO.
nmos_effective resistance_multiplier	N/A	Tuning parameter for NMOS on current in CACTI. Not re-created in CO.
eps_ox	0.00859e-14 F/um	Estimated from CACTI 180nm tech.
c_fringe	0 F/um	Not defined in scn4m_subm.
cpolywire	0 F/um	Defined as 0 in CACTI.
Vdsat	0.256 V	Estimated from CACTI 180nm tech.

3.1.5 Other Graph Applications

The graph provides an interface for the SPICE characterization to probe specific points in the SRAM. Using SPICE to debug common issues for SRAMs, such as write failures, requires probing the voltage of the bitcell storage nodes and comparing it with the operation that occurred within that cycle. The main barrier to debugging is knowing the names of nodes within the SRAM SPICE file as the SPICE stimulus file only knows input and output names to

the SRAM. The graph can be used to determine any SPICE name as all node names in the graph are equivalent to the node names in the SPICE file.

The graph is then queried for the nodes of interest. This process requires some assumptions, but they are much more general and do not pertain to large structures in the SRAM. To get the bitcell internal storage nodes: the timing graph is generated, the bitcell module in the critical path is isolated, and the SPICE name for the storage node is queried from the bitcell. The only assumption here is that the bitcell is part of the critical path which is enforced in the regression tests for OpenRAM. A similar process is repeated to add SAE timing checks and communicate changes to the SRAM's delay line.

3.2 OpenRAM Statistical Models

Several statistical models are implemented in OpenRAM which provide complete characterization with equivalent output to SPICE simulation or Elmore analytical models. These models require occasional SPICE data collection, but generating and maintaining the model requires little to no knowledge of underlying SRAM mechanics. The models are implemented in OpenRAM by generating a large set of SPICE simulations of many different configurations, parsing OpenRAM's output for what is relevant to the model, and using that data to train models and generate predictive delays for the LIB file.

OpenRAM often updates, so many minor updates or a single significant update can cause the model accuracy to drift and requires regenerating the data. Regression tests are present within OpenRAM which test the accuracy of the statistical models and will fail if accuracy dips below the threshold. The makefile can be run at this point to regenerate the data.

A series of scripts were added to OpenRAM to generate and collect SPICE data. OpenRAM expects a directory of configuration files that will have SPICE simulation performed. The configuration file expects setting at least word_size and num_words, and the rest can be automatically generated. The model itself uses the inputs described in Table 3.3. All training configurations define words_per_row and local_array_size, while OpenRAM's characterization automatically sets the PVT corners, input slew, and load. These can also be overridden in the configuration, but since all characterizations use these inputs, we want the model to represent the predetermined ranges better. OpenRAM provides a makefile to generate the model data. The data is then extracted and saved within the target technology's directory in OpenRAM.

Table 3.3: Inputs to OpenRAM statistical models.

num_words	Number of words in SRAM
word_size	Number of bits per word
words_per_row	Column mux size
local_array_size	Divisions of bitcell array along wordline
Process	Transistor model corner (slow, fast, nominal)
Voltage	Voltage corner (between +-10% of nominal)
Temperature	Temperature corner (between 0-100 Celsius)
Slew	Input slew to SRAM
Load	Output load on SRAM data bits

Different packages were tested, but scikit-learn [13] was chosen to create the model for OpenRAM. The package has extensive model variety, a small footprint, and is open-source. A drawback of this package is the inability to design precise models and training algorithms, but the models provided can attain accuracies well within OpenRAM's other models. Scikit-learn is also very small compared to packages such as TensorFlow, where loading the package typically takes more time than the training and predictions. The analytical models can typically generate characterizations under 1 second, so OpenRAM aimed to maintain that similar time frame with the statistical models.

OpenRAM maintains two statistical models: LR and multi-layer perceptron (MLP). LR is simple to set up, simple to use, and generates accurate results in most cases. However, delays and powers of the SRAM tend to be non-linear, so the model cannot be perfectly accurate. The MLP model is a neural network that uses the perceptron as an activation function between layers. The number of layers and hidden nodes per layer was adapted from Last and Schlichtmann [11] where they achieved a good accuracy by using 8x the number of model inputs and four hidden layers. However, each output is maintained by a separate model, and nodes are not shared as described in their work.

The main goal of the statistical models is that they are fast like the other analytical models. The runtime of characterization depends not only on the type of model but also on the amount of training data. A large model such as a convolutional neural network for image processing can take days to train even with state-of-the-art hardware. These models may contain gigabytes of internal parameters, and input sizes can be in the order of megabytes. Models that require large inputs were not considered for this reason.

Data from 79 configurations was added to OpenRAM for the statistical models. Of

these 79 configurations, 9 are common configurations surveyed from related works, and 70 are random configurations. The configurations were randomized based on four variables detailed in Table 3.4. The inputs num_words and word_size are independently picked from their ranges while words_per_row cannot reduce the numbers of rows below 16, and local_array_size cannot be larger than the number of columns. The OpenRAM configurations are detailed in Appendix B.

Table 3.4: Inputs to OpenRAM statistical models.

Parameter	Description	Range
num_words	Number of words in SRAM	16-1024 bits
word_size	Number of bits per word	1-32 bits
words_per_row	Column mux size	1, 2, 4, 8, 16 bits
local_array_size	Divisions of bitcell array along wordline	0-512 bits

The number of random configurations was chosen through model cross validation where a portion of the available data is used train a model and the remaining data was used to test the model. Enough models were added, so the worst-case accuracy of the statistical model's delay was within 25% of SPICE measured delay. The data amount can be configured and changed later in cases where specific corner configurations are not well incorporated into the model.

OpenRAM benefits from maintaining a relatively small set of training data. Accuracy mainly depends on the data amount, but the time cost of simulating the data cannot be ignored. From the OpenRAM random configurations, an average characterization will take 24 minutes. Small SRAMs may only require 15 minutes to characterize, while large SRAMs need nearly two hours. OpenRAM currently assumes the data can be simulated within roughly eight hours, and eight simulations can be run in parallel. These assumptions are chosen from the perspective of an average developer. The time implies data can be generated every day, and the hardware that can allow eight parallel simulations is modest and does not require a specialized setup.

Chapter 4

Results

This section first compares the accuracy and fidelity of the models. Delay and power are evaluated with all the analytical models. Area is modeled using two different statistical models. Good fidelity and speed are more desirable in these models than accuracy. Faithful models allow can be used to compare many SRAMs, and SPICE is used when high accuracy is needed on select SRAMs.

The results are concluded with a discussion on the cost of maintaining statistical models for SRAM characterization. Analysis of the necessary data amounts for an accurate model and a cost discussion on getting this data is also presented.

4.1 Delay

Delay is one of the most critical characteristics of SRAMs. Accurate models for delay are preferred, but the model must at least maintain fidelity to be useful. The models' accuracy and fidelity are calculated and compared against SPICE.

4.1.1 Experimental Methods

The random configurations detailed in section 3.2 are used to evaluate the accuracy of each model. This totals 70 SRAM configurations detailed in Appendix B.2. The configuration under evaluation is not included in the training set of the statistical model for that evaluation. Slew, load, and PVT are not included in this randomization, and the OpenRAM nominal corner

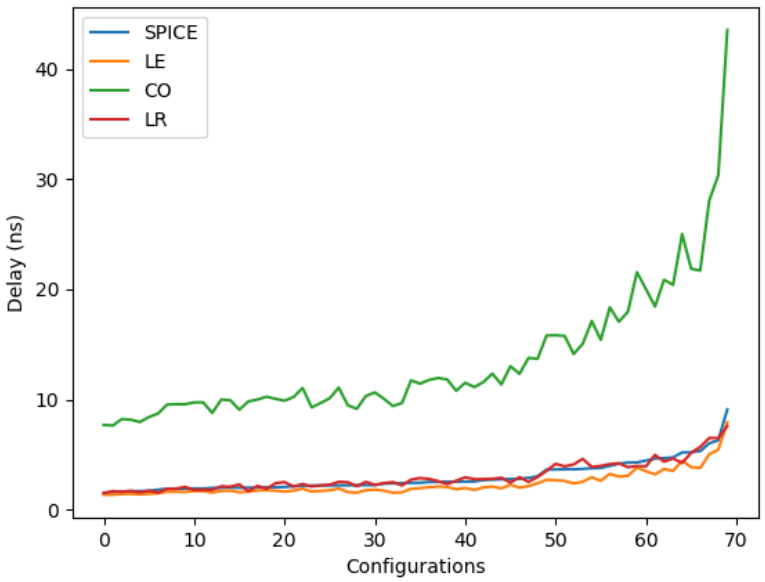


Figure 4.1: SRAM delays of 70 SRAM configurations. The LR model is nearly aligned with SPICE in this view.

is always chosen because its the most common choice for characterization. Nine slew/load pairs, dependent on technology, are chosen for each configurations. All SPICE simulations were run using hspice.

4.1.2 Accuracy

SPICE simulation is the golden characterization and sets the standard of accuracy for all models. The accuracy of the Elmore based models is compared with SPICE as detailed in Figure 4.1. The SPICE delay is sorted from least to greatest while maintaining the respective delay of the model along the x-axis. The delays are sorted to identify the model's trend with SPICE as the delay increases. The CO model tends to overestimate SPICE while the LE model underestimates SPICE. Some configurations exhibit noise in the LR model, but each model tracks SPICE simulation well. The absolute error relative to SPICE is shown and is bounded between 260% and 408% for CO, while the error for LE is bounded between 11% and 40%. LE is the clear winner in accuracy between these two Elmore models. Neither model accurately replaces SPICE, but this was expected of the models. Knowing the accuracy gives an

understanding of how to interpret the analytical models.

This comparison shows that the CO model is extremely pessimistic with SPICE. This significant difference could come from tuning parameters used in CACTI but is absent in the CO model. For example, CACTI contains tuning parameters for the transistor current which were not included in the CO model due to lack of documentation. Table 3.2 details potential parameters that could have resulted in accuracy loss.

The delay of the statistical model is displayed in Figure 4.2 and is closely aligned with the SPICE delay. The error is bounded between 0.08% and 25% which is the lowest of any model. The model is trained using SPICE data which allows for this accuracy but requires costly simulations.

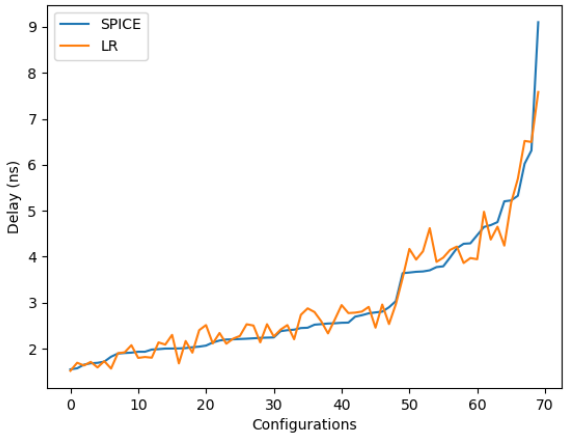
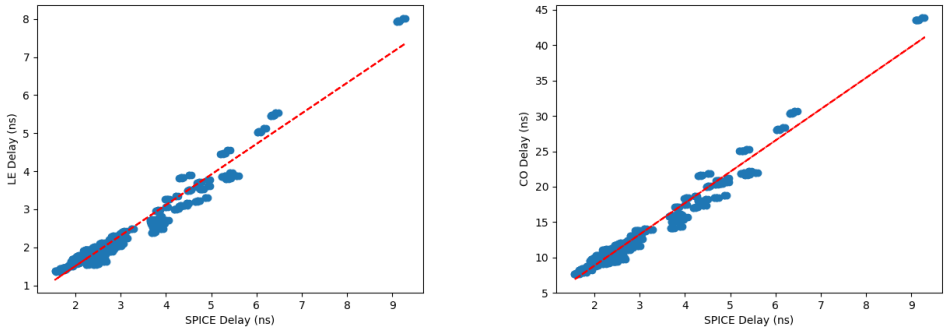


Figure 4.2: Linear regression statistical model resembles SPICE delays with some noise.

4.1.3 Fidelity

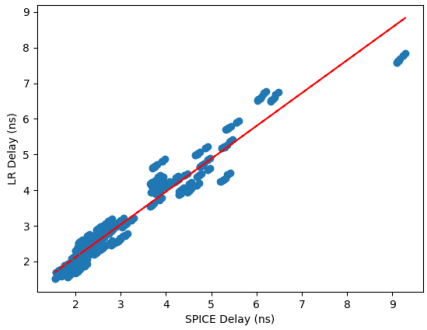
The Elmore delay models shown in the previous section were less accurate than the statistical model, but accuracy is not the only metric of interest. In addition to speed, fidelity to the SPICE model is expected of the analytical models.

Fidelity, or faithfulness, of two datasets tells us how well they trend together i.e. if one increases, the other increases and vice versa. The differing slopes of the datasets do not affect the fidelity but is measured by the accuracy. Pearson's correlation ranges between -1 to 1, and the positive correlation range represents the desired fidelity behavior. Negative correlation



(a) LE model

(b) CO model



(c) LR model

Figure 4.3: Delay correlation between analytical models and SPICE.

implies the datasets increase and decrease inverse to each other which is not desired in the models. Negative and zero correlation is considered unfaithful when comparing the models to SPICE.

The correlation of SPICE delays and model delays are performed and shown in figures 4.3(a), 4.3(b), and 4.3(c) for LE, CO, and the statistical model respectively. The LE model with an average error of 22.6% has a correlation of 0.979, the CO model with an average error of 345.4% has a correlation of 0.984, and the LR model with an average error of 7.3% has a correlation of 0.970. Each model displays strong fidelity to SPICE simulation. CO with the most significant error which was about 15x worse than the LE model has a similar correlation as LE. This displays that the models are faithful and track SPICE well even when the accuracy is poor, and therefore, any of the three models are helpful when designing and comparing SRAM

designs.

Both the CO and LE models contain estimates that are slightly off the expected trend in the graph. These values generated the apparent noise in the accuracy graphs. Both models overestimate the delay of the wordline and bitline. The models incorporate non-ideal wire delays for these portions, while the SPICE simulation does not contain extracted parasitics. The parasitic capacitance is not a significant portion of the delay in this technology, so fidelity is not affected significantly. The statistical model does not make this mistake because it does not explicitly model individual portions of the SRAM.

Table 4.1: Analytical model errors compared with SPICE.

Model	Mean Error (%)	Worst Error (%)	Std Dev. (%)	Fidelity
LE	22.6	39.1	6.4	.979
CO	345.4	407.8	33.3	.984
LR	7.3	24.8	5.4	.970

Table 4.1 summarizes both the accuracy and fidelity of the analytical models. LR has the best accuracy, CO has the best fidelity, and LE is between both in accuracy and fidelity.

4.2 Power

Power is difficult to model as it is to simulate. OpenRAM supports several different SPICE simulators including Ngspice, hspice, and Xyce. Each simulator will output slightly different results. The difference in delay between the simulators is often much smaller than differences in power. For example, OpenRAM maintains separate regression tests for its different simulators, and there is a 0.1% difference between the Ngspice and Xyce delays while there is a 66% difference in dynamic power. There are explicit models for dynamic power, but modeling switching probabilities and unintended power consumption like glitch power makes it difficult to produce accurate and faithful results. Modeling delay is often easier because focus can be put on the critical paths while ignoring the majority of the SRAM, while power concerns the entire circuit. This makes power an excellent candidate to predict with statistical models.

Figure 4.4 shows the accuracy of power models compared to SPICE. Characterization with LE and CO uses the same power model, so only LE represents the Elmore models here. LR tracks SPICE while the LE model fails to capture the trend. Both display spikes in power.

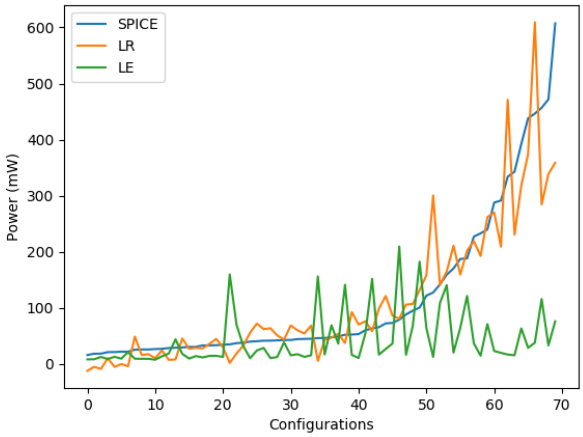


Figure 4.4: Accuracy of LR and LE power models in OpenRAM.

The LE model assumes a linear dependence on the number of devices and power consumption, which is why power is poorly tracked.

The statistical model should only use data from the same simulator. The simulator can have a wide variance in power, so using a consistent simulator to produce data is essential when improving accuracy. More complicated neural network models may be able to consider the simulator used such that any data can be provided, but that is left to future work.

4.3 Area Modeling

Area modeling is helpful even in tools, like OpenRAM, which create layouts for their memories. CACTI only models the area based on estimates of individual modules and using assumed floorplanning. Generating the layout can be slow for larger SRAMs as routing complexity increases. OpenRAM does not contain area modeling but allows layout to be disabled if not needed. Area modeling requires many assumptions about the layout and can take a significant amount of time to develop and maintain. Statistical models can provide an easy way to implement an area model that is accurate and maintains good fidelity.

A linear regression model was trained using the log of the SRAM area, and predictions were generated on 65 of the random configurations as five failed to create a layout. As shown in Figure 4.5. The model has an average error of 29.4% with the worst case of 114.5% accuracy but

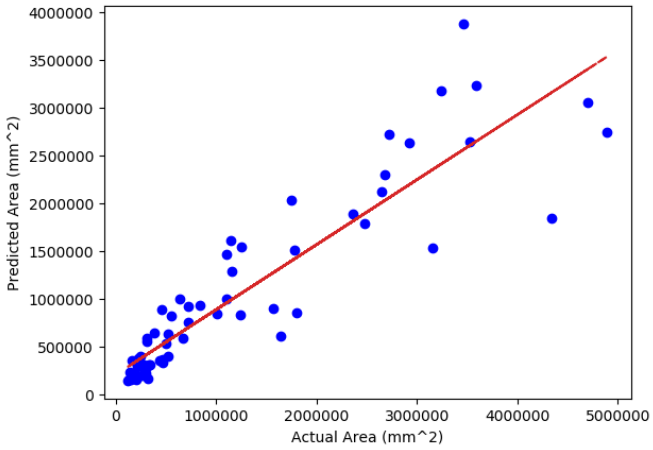


Figure 4.5: Predicted area from LR model displays good fidelity but poor accuracy in larger SRAMs.

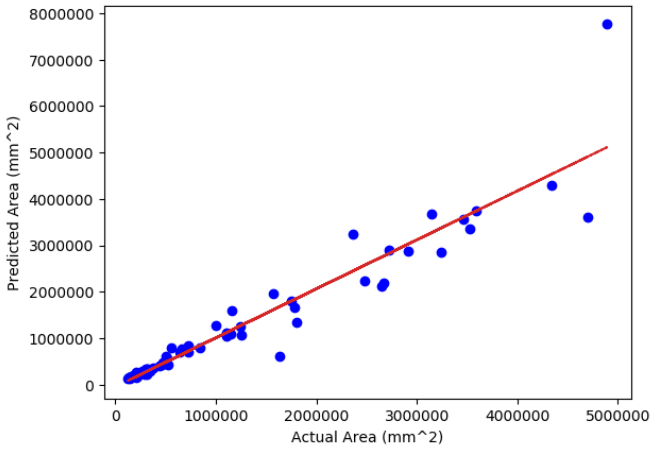


Figure 4.6: A neural network model almost tracks the SRAM area trend much closer than LR.

has a fidelity of 0.909. The area grows quadratically, so linear regression is not expected to have good accuracy. A neural network model that uses the same hyperparameter configuration as the delay models for OpenRAM was also tested. This model achieves 12.9% error with a worst-case accuracy of 64.5%. This model also maintains good fidelity with 0.948 as shown in Figure 4.6. Just changing the model results in nearly doubling the accuracy of the area predictions created by statistical models. Further hyperparameter exploration could help improve the worst-case error, but the fidelity is sufficient for comparing configurations.

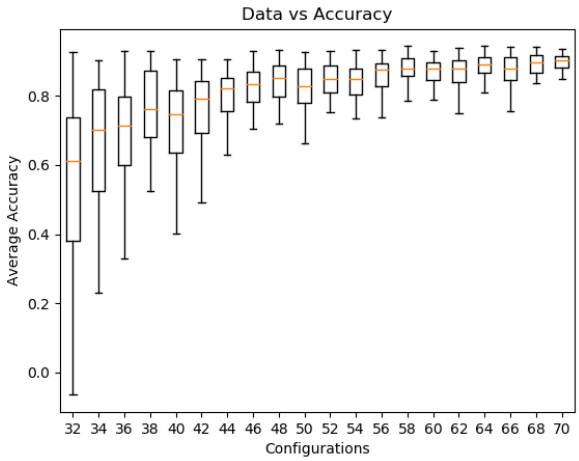


Figure 4.7: Accuracy of the linear regression model saturates around 93%.

4.4 Statistical Model Accuracy and Cost

Statistical models are the most accurate among the discussed analytical models. However, this model requires training data from SPICE characterizations. While the type of model can play a role, the accuracy is dependent mainly on the amount of training data. Figure 4.7 details the average accuracy of the model during 10-fold cross validation. This uses 90% of the dataset to train the model while 10% is used to test against and is repeated 10 times across the dataset. The process was repeated 50 times for each data slice. The x-axis represents how many random configurations were combined with OpenRAM's common configuration data, and the y-axis displays the accuracy range for the validation steps of that random slice. This plot gauges the accuracy of the model at varying amounts of data. At 32 configurations, the model has an extreme range of accuracies while the worst case and average accuracy begin to level around 58 configurations. Configurations can be continually added to improve the accuracy, but the gains quickly taper.

Each training set has an associated time cost that cannot be ignored when choosing the amount of training data. OpenRAM makes frequent changes to its memories which warrants replacing the existing training data, so the time to generate this data matters. SPICE characterizations of the random configurations take between 12.5 to 100.2 minutes depending on the size of the SRAM. The average characterization took 24.6 minutes in the 70 configurations. The

average simulation time of each model can be estimated based on the number of configurations and number of parallel characterizations allowed.

The model's amount of data will depend on the desired accuracy and maximum total simulation time. The assumptions from section 3.2 require the worst-case accuracy to be 25% on an 8 hour time budget with at most 8 parallel simulations. All simulations were performed in hspice, and individual simulations were not explicitly made parallel. The sum of simulation time of the 70 configurations was 28.67 hours, so eight parallel simulations could reduce this to 3.58 hours in the best case. The nine common configurations have a simulation time of 36.02 hours which can be reduced to 4.5 hours at best. The best-case total time is 8.08 hours which is near the expected budget, so nearly all the random data can be included. The model can maintain the minimum accuracy with at least 58 configurations, so trimming 1-2 configurations from the 70 maximum still stays within this range. Although perfect parallelization was assumed, the longest simulation within the common configurations required 13.7 hours to simulate. Additional parallelization within that simulation or removal of this configuration would be required to meet the time budget.

4.5 Model Trade offs

A difficult metric to quantify between the analytical models is their maintainability or maintenance cost. All three models presented here are designed to be low maintenance to consider OpenRAM's frequent updates. The majority of maintenance required by LE and CO models add delay, capacitance, and resistance estimations to new custom models or pgates. These changes are infrequent but commonly happen when OpenRAM is ported to a new technology. New technologies require an entirely new set of custom cells, and their characteristics need to be identified within OpenRAM's technology file for these models to function. Statistical models are arguably much easier to maintain. While they may require more frequent data collection, large changes to OpenRAM would only result in a data collection as well. Complicated additions to OpenRAM such as changing custom cells or new technologies would only require re-collecting data, but the model itself does not need to be changed. The only changes to OpenRAM that would warrant changing the model is low accuracy or new model inputs.

Chapter 5

Conclusions

This thesis presents several analytical models that were implemented in OpenRAM including two Elmore delay based models and a statistical model. The well-established memory and cache modeling tool, CACTI, was used as a basis for one of these models. The process of adapting it to OpenRAM using a static timing graph was detailed in this thesis. This thesis also displays the use of a statistical model to generate characterizations, and unlike the Elmore models, it requires little knowledge about the memory as a whole to create. The statistical model achieved the highest accuracy to SPICE while being fast and maintaining strong fidelity. Statistical models present an attractive option to model SRAMs because they can be easily automated and maintain good model properties. The main drawback is the data collection needed for the model which will have a variable cost depending on desired accuracy and how often the model needs to be re-generated.

Appendix A

Model Code

A.1 Horowitz delay function adapted from CACTI.

```
def horowitz(self,
             inputramptime, # input rise time
             tf,            # time constant of gate
             vs1,           # threshold voltage
             vs2,           # threshold voltage
             rise):         # whether input rises or fall

    if inputramptime == 0 and vs1 == vs2:
        return tf * (-math.log(vs1) if vs1 < 1 else \
                     math.log(vs1))

    a = inputramptime / tf
    if rise == True:
        b = 0.5
        td = tf * math.sqrt(math.log(vs1)*math.log(vs1) +\
             2*a*b*(1.0 - vs1)) +\
             tf*(math.log(vs1) - math.log(vs2))
```

```
else:
    b = 0.4
    td = tf * math.sqrt(math.log(1.0 - vs1) *\
         math.log(1.0 - vs1) +\
         2*a*b*(vs1)) +\
         tf*(math.log(1.0 - vs1) - math.log(1.0 - vs2))

return td
```

Appendix B

Results Configurations

B.1 Nine common configurations selected to train the statistical models.

Word size	Number of words	Words per row	Local array size
8	256	8	No Local Buffers
8	512	8	No Local Buffers
8	1024	16	No Local Buffers
32	256	4	No Local Buffers
32	512	4	No Local Buffers
32	1024	8	No Local Buffers
32	2048	8	No Local Buffers
64	1024	4	No Local Buffers
128	1024	4	No Local Buffers

B.2 Seventy random configurations used to train the statistical models.

Word size	Number of words	Words per row	Local array size
2	16	1	2
3	32	1	2
3	1024	2	3
4	16	1	4
4	32	2	5
4	64	4	14
5	256	16	75
5	512	2	8
6	16	1	1
6	32	1	1
6	128	4	8
7	16	1	3
7	64	2	10
7	256	4	25
8	128	4	22
8	256	1	1
9	128	1	4
9	256	4	15
10	64	4	21
12	16	1	1
12	16	1	7
12	16	1	8
12	128	4	38
12	256	8	17
12	256	8	83
12	256	16	186
12	512	4	20
14	16	1	0
14	32	2	23
15	32	1	11
15	512	8	85
16	16	1	9
16	128	8	2
16	512	1	14
16	512	16	107

Word size	Number of words	Words per row	Local array size
16	1024	16	40
17	32	1	13
17	256	16	49
17	1024	16	86
18	32	1	18
18	128	2	7
19	16	1	5
20	16	1	5
20	32	2	5
20	32	2	13
20	128	8	90
21	128	2	10
21	1024	4	54
22	512	16	249
23	32	1	2
23	1024	16	118
24	16	1	23
26	32	1	23
26	64	4	23
26	512	1	14
27	128	1	5
27	256	4	2
27	256	8	191
27	512	4	60
27	1024	4	89
28	16	1	8
28	32	1	14
28	128	8	61
28	1024	2	53
30	1024	4	58
31	64	4	81
31	1024	16	49
32	32	1	31
32	128	2	17
64	512	4	No Local Buffers

Bibliography

[1] Liberty User Guides and Reference Manual Suite, 2020.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] Khalid Al-Hawaj et al. Towards a reconfigurable bit-serial/bit-parallel vector accelerator using in-situ processing-in-sram. In *ISCAS*, 2020.

[4] A. de Gennaro et al. Design and implementation of reconfigurable asynchronous pipelines. *TVLSI*, 28:1527–1539, March 2020.

[5] Elnaz Ebrahimi. *Pareto-optimal methodology for cache and SRAM modeling*. University of California, Santa Cruz, 2011.

[6] William C Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of applied physics*, 19(1):55–63, 1948.

[7] Matthew Guthaus, Hunter Nichols, Jesse Cirimelli-Low, Joseph Kunzler, and Bin Wu. Enabling design technology co-optimization of srams through open-source software. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 41–7. IEEE, 2020.

[8] Matthew R. Guthaus et al. OpenRAM: An open-source memory compiler. In *ICCAD*, 2016.

[9] Matthew R. Guthaus et al. OpenRAM. https://github.com/VLSIDA/OpenRAM, 2020.

[10] Felix Last, Max Haeberlein, and Ulf Schlichtmann. Predicting memory compiler performance outputs using feed-forward neural networks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(5):1–19, 2020.

[11] Felix Last and Ulf Schlichtmann. Partial sharing neural networks for multi-target regression on power and performance of embedded memories. In *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, pages 123–128. IEEE, 2020.

[12] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP laboratories*, 27:28, 2009.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[14] Robert F Sproull and Ivan E Sutherland. Logical effort: Designing for speed on the back of an envelope. *IEEE Advanced Research in VLSI*, 9:219, 1991.

[15] James E Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W Rhett Davis, Paul D Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, et al. Freepdk: An open-source variation-aware design kit. In *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*, pages 173–174. IEEE, 2007.