DISSERTATION

Γ (GAMMA): CLOUD-BASED ANALOG CIRCUIT DESIGN SYSTEM

Submitted by

Yishai Arie Statter

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2016

Doctoral Committee:

Advisor: Tom W. Chen

Ross McConnell
George Collins
Sudeep Pasricha

Copyright by Yishai Arie Statter 2016

All Rights Reserved

ABSTRACT

Γ (GAMMA): CLOUD-BASED ANALOG CIRCUIT DESIGN SYSTEM

With ever increasing demand for lower power consumption, lower cost, and higher performance, designing analog circuits to meet design specifications has become an increasing challenging task, on one hand; analog circuit designers must have intimate knowledge about the underlining silicon process technology's capability to achieve the desired specifications. On the other hand, they must understand the impact of tweaking circuits to satisfy a given specification on all circuit performance parameters. Analog designers have traditionally learned to tackle design problems with numerous circuit simulations using accurate circuit simulators such as SPICE, and have increasingly relied on trial-and-error approaches to reach a converging point. However, the increased complexity with each generation of silicon technology and high dimensionality of searching for solutions, even for some simple analog circuits, have made trial-and-error approaches extremely inefficient, causing long design cycles and often missed market opportunities. Novel rapid and accurate circuit evaluation methods that are tightly integrated with circuit search and optimization methods are needed to aid design productivity.

Furthermore, the current design environment with fully distributed licensing and supporting structures is cumbersome at best to allow efficient and up-to-date support for design engineers. With increasing support and licensing costs, fewer and fewer design centers can afford it. Cloud-based software as a service (SaaS) model provides new opportunities for CAD applications. It enables immediate software delivery and update to customers at very low cost. SaaS tools benefit from fast feedback and sharing channels between users and developers and

run on hardware resources tailored and provided for them by software vendors. However, web-based tools must perform in a very short turn-around schedule and be always responsive.

A new class of analog design tools is presented in this dissertation. The tools provide effective design aid to analog circuit designers with a dash-board control of many important circuit parameters. Fast and accurate circuit evaluations are achieved using a novel lookup-table transistor models with novel built-in features tightly integrated with the search engine to achieve desired speed and accuracy. This enables circuit evaluation time several orders faster than SPICE simulations. The proposed architecture for analog design attempts to break the traditional analog design flow using SPICE based trial-and-error methods by providing designers with useful information about the effects of prior design decisions they have made and potential next steps they can take to meet specifications. Benefiting from the advantages offered by web-hosted architectures, the proposed architecture incorporates SaaS as its operating model. The application of the proposed architecture is illustrated by an analog circuit sizer and optimizer. The Γ sizer and optimizer show how web-based design-decision supporting tool can help analog circuit designers to reduce design time and achieve high quality circuit.

ACKNOWLEDGEMENTS

The road to a Ph.D. started with a family walk on Saturday afternoon. My simple remark that: “If I went back to school, I’d like to get a Ph.D.” was taken by my wife too seriously. Erin, thanks for setting me on this path and for not letting me quit on so many breaking points along the way. The sacrifices you and the kids made for this program are appreciated and will never be forgotten. This achievement is yours as much as it is mine.

Thanks to my advisor, Dr. Tom Chen for accepting me to his team and entrusting me with this interesting research topic. It’s been awhile since I had a chance to be so freely creative.

The following needs to be done in my native language:

להוריי: אני יודע שכשאמא בקשה שאהיה "דוקטור" היא לא התכוונה לכזה, אבל גם לזה לקח יותר מדי זמן. תודה ששלחתם אותי לטכניון ותמכתם בי בשבע השנים הרזות של הלימודים ושירות החובה. תודה על העידוד ותודה על הסבלנות הרבה בזמן שאנחנו פה בגולה עם הנכדים.

Acknowledgement and Disclaimer: The research reported in this dissertation was partially supported by a National Science Foundation Grant No. 0841259. Any opinions, findings, conclusions or recommendations expressed in this dissertation are the author’s and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iv

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ACRONYMS xxi

1. Introduction 1

1.1. Electronic-Design Automation 1

1.2. Analog Design Automation 3

1.3. The Proposed Γ Analog Design System 6

2. Existing Research 8

2.1. Overview 8

2.2. Symbolic Analysis 8

2.3. Alternative Transistor Modeling 9

2.4. Multi Objective Optimization 11

2.5. Design-Supporting Analysis 11

2.6. Web-based Tools 12

2.7. Hierarchical Design 13

2.8. Commercial Attempts 14

2.9. Summary 15

3. Motivation for Proposed Architecture 16

4. System Architecture 21

4.1. The Γ System 21

4.2. Offline Toolset 24

4.3. Online Toolset 27

4.4. Architecture Layer Model 29

4.5. Software Components 31

4.6. Language and Compilation 32

4.7. Context-Tree Data Base 35

4.8. Foundry Model Translation 38

5. Implementation and Applications of Γ 41

5.1. Transistor-level Modeling for Γ 41

5.1.1. Motivation 41

5.1.2. Implementation 42

5.1.2.1. Look-up Tables 43

5.1.2.2. Geometry Parameters 44

5.1.2.3. Voltage Parameters 45

5.1.2.4. Parallelization 45

5.1.2.5. Reading and Post-Processing Simulation Results 46

5.1.3. Linear Interpolation Table 47

5.1.4. Interpolation Procedures 50

5.1.5. Composite Interpolation 53

5.1.6. Hierarchical Interpolation Tree 54

5.1.7. Resolution Budgeting 58

5.1.8. LUT Interpolation Implementation 60

5.1.9. 4D Compact Model 61

5.2. Circuit Compiler 63

5.2.1. Linear Model of Transistor 64

5.2.2. Nodal Analysis 66

5.2.3. Algebra Engine 70

5.2.4. Generated Equations 74

5.2.4.1. Equilibrium Operating Point Voltages (OP) ... 74

5.2.4.2. Low Frequency Performance: Gain and Rejection Ratios ... 75

5.2.4.3. Output Resistance: Rout ... 76

5.2.4.4. AC Performance: Bandwidth, Settling Time, and Phase Margin ... 77

5.2.4.5. Noise Parameters and Corner Frequency ... 78

5.2.4.6. Additional Performance Parameters ... 79

5.2.5. Internal Circuit Dependencies and Design-Feedback ... 79

5.2.6. Code Generator ... 80

5.2.6.1. *c Templates ... 81

5.2.6.2. ΓCE Shared Object Structure ... 82

5.2.6.3. C Compilation ... 83

5.2.6.4. Manual Code Edits ... 83

5.3. Γ Online Engines ... 84

5.3.1. Γ Circuit Evaluator ... 84

5.3.2. Random Circuit Engine ... 88

5.4. Circuits Pareto Front ... 88

5.4.1. Circuit Inventory Container and Manager - PAT ... 90

5.4.2. PAT Entry 91

5.4.3. Offline PAT Populating 92

5.4.4. Extracting Pareto Fronts 94

5.4.5. Culling 97

5.4.6. PAT Size Requirements and Limitations 99

5.5. Web Applications 99

5.5.1. Data Visualization 99

5.5.2. Single Transistor Sizer 107

5.5.3. FETScape 111

6. Experimental Results 113

6.1. Methodology 113

6.2. Transistor-Level Results 114

6.2.1. Spectre to SPICE Comparison Stage 114

6.2.2. Budgeted Resolution 119

6.2.3. Transistor Level LUT to SPICE Matching Regression 123

6.3. Circuit Level Results 125

6.3.1. Tested Circuits 125

6.4. Run Time Analysis 126
6.4.1. Transistor Level 126
6.4.1.1. Methodology 126
6.4.1.2. Results 127
6.4.2. Circuit Level 129
6.4.2.1. Offline Activities 129
6.4.2.2. Circuit Evaluation Time 130
6.4.2.3. PAT Insertion Time 131
6.5. Γ Website Usage Example 133
6.5.1. Select Topology and Enter Specification 133
6.5.2. Pick an Initial Circuit 135
6.5.2.1. Search for Gain-Valid Circuit 135
6.5.2.2. Reduce Noise to Meet Spec 136
7. Conclusions 139
8. References 142
9. Appendices 147
A. Γ/SPICE Matching Results 148

A. 1. Transistor Level 149

A. 2. Circuit Level 191

A. 2. 1. Operating Point 191

A. 2. 1. 1. Common Source 191

A. 2. 1. 2. Differential Pair 192

A. 2. 1. 3. Differential Pair with Output Stage 192

A. 2. 1. 4. Operational Amplifier 193

A. 2. 2. Low Frequency Gain 194

A. 2. 2. 1. Common Source 194

A. 2. 2. 2. Differential Pair 194

A. 2. 2. 3. Differential Pair with Output Stage 195

A. 2. 2. 4. Operational Amplifier 195

A. 2. 3. Common Mode Rejection Ratio 196

A. 2. 3. 1. Common Source 196

A. 2. 3. 2. Differential Pair 196

A. 2. 3. 3. Operational Amplifier 196

A. 2. 4. Power Supply Rejection Ratio 197

A. 2. 4. 1. Common Source 197

A. 2. 4. 2. Operational Amplifier 197

A. 2. 5. Bandwidth 197

A. 2. 5. 1. Common Source 197

A. 2. 5. 2. Differential Pair 198

A. 2. 5. 3. Operational Amplifier 198

A. 2. 6. Phase Margin 199

A. 2. 6. 1. Common Source 199

A. 2. 6. 2. Differential Pair 199

A. 2. 6. 3. Differential Pair with Output Stage 200

A. 2. 6. 4. Operational Amplifier 200

B. Source Code 201

B. 1. Language, Format and Compilation 201

B. 2. Data Base 203

B. 3. Lookup and Interpolation 227

B. 3. 1. Interpolation of LUT 229

B. 3. 2. Differential Interpolation 235

B. 3. 3. Cluster Interpolation 238

B. 3. 4. Linear Regression 243

B. 3. 5. Composite Interpolation 248

B. 4. Transistor-Level Sizer 254

B. 5. Transistor-Level Characterization 279

B. 5. 1. Low-level Data Access 279

B. 5. 2. Process Parellelization 307

B. 5. 3. 4D Characterization Script 310

B. 5. 4. 5D Characterization Script 332

B. 6. Γ Circuit Compiler 352

C. Usage Model 403

C.1. Use the Tool as a Design Aid. 403

LIST OF TABLES

Table 1: Six Executable Binaries Compiled by the Compilation Script 32

Table 2: HIT Cell Types and their Cost in [B] 58

Table 3: Transistor-Level Modeled Parameters 63

Table 4: Circuit Grading and Eliminating w.r.t. a Spec 95

Table 5: Useful IDS (SS, 125 °C) resolutions 121

Table 6: Useful I_{DS} (SS, 125 °C) resolutions vs. access time 122

Table 7: Structure of Device+Corner+Parameter Detailed Report 125

Table 8: Test Machine's Hardware Specifications 127

Table 9: Comparison of Query Times per Model Type 129

Table 10: Run time of ΓCE and NGSPICE .op [msec] 131

Table 11: Breakdown of Noise Contribution per Transistor 136

Table 12: Γ and Virtuoso Final Performance Metrics 138

Table 13: Search Path Summary 138

LIST OF FIGURES

Figure 1: Moore's law demonstrated by x86 processors family [2] 1

Figure 2: Traditional Analog IC Design Process 2

Figure 3: High-level Dataflow Diagram of the Γ System 21

Figure 4: Offline Dataflow of Γ 24

Figure 5: Γ Online Toolset 27

Figure 6: Layer Model of the Γ System 29

Figure 7: Block Diagram of Executable Software Components 31

Figure 8: Implementation, Interpreted and Processed Language per Subsystem 34

Figure 9: Example Segment of a cTree Structure 36

Figure 10:Five dimentional . hypercube: a cell in I_{DS}, g_m and r_o LUT. 43

Figure 11: I_{DS} Parallel Characterization Circuit 45

Figure 12: Percentage of cells that fit linear representation for IDS (LCP) 49

Figure 13: 2D full interpolation using in-place compression loop. 52

Figure 14: Zoom in on Separation Area 53

Figure 15: Linear and Composite Interpolation vs SPICE 53

Figure 16: 4D Square Channel Characterization and Binning Mismatch 62

Figure 17: Circuit Compilation Flow Stages 64

Figure 18: Linear Model of N-type MOSFET 65

Figure 19: I_{Deq} is the intercept of the $I_{DS}(V_{GS},V_{DS})$ plane with the I_{DS} axis 66

Figure 20: Example Topology for Nodal Analysis 68

Figure 21: MNA matrix generated for the compiled topology in Figure 20 69

Figure 22: Rout Equivalent Circuit 76

Figure 23: Γ Circuit Evaluator Data Flow 87

Figure 24: PAT Entry Structure 91

Figure 25: PAT Seed and Breed Phases 92

Figure 26: PAT populating decreasing retention rate 93

Figure 27: Corner rules added to Algorithm 11 107

Figure 28: "Sizer" Screen Shot 109

Figure 29: Search Path Plot on top of Spec-Distance Function Surface (zoom-in below)......... 110

Figure 30: System Architecture from a Designer's Perspective. 112

Figure 31: First of Two-Staged Comparison Procedure Data Flow 113

Figure 32: Scatter Plot of Spectre vs SPICE Ids Values (TSMC 40nm NMOS in SS 125C) 114

Figure 33: Relative Error Histogram For Figure 32 115

Figure 34: Spectre vs SPICE Ids match, without W/L>10 Channels 116

Figure 35: Relative Error Histogram for Figure 34 117

Figure 36: 180nm NMOS and PMOS Ids comparison between Sepctre and SPICE 118

Figure 37: Relative Error Histogram for Figure 36 119

Figure 38: IDS resolution combinations memory and resulting maximal error 120

Figure 39: IDS resolution combinations access time and resulting maximal error. 122

Figure 40: Error Summary Report for TSMC 40nm and Resolution 5:5:3:8 124

Figure 41: Four Tested Topologies 125

Figure 42: Interpolation Time [nsec] vs Dimensionality 128

Figure 43: PAT Insertion Time per Size 132

Figure 44: Ratio between PAT Insertion Time and Size 132

Figure 45: PAT circuits that meet gain and BW 134

Figure 46: DC Gain vs input transistor length and width 135

Figure 47: Circuit 696 trades BW for less noise 137

Figure A-1: 40nm NCH SS I_{DS} LUT Queries to SPICE Matching Statistics 149

Figure A-2: 40nm NCH TT I_{DS} LUT Queries to SPICE Matching Statistics 150

Figure A-3: 40nm NCH FF I_{DS} LUT Queries to SPICE Matching Statistics 151

Figure A-4: 40nm NCH SS g_m LUT Queries to SPICE Matching Statistics 152

Figure A-5: 40nm NCH TT g_m LUT Queries to SPICE Matching Statistics 153

Figure A-6: 40nm NCH TT g_m LUT Queries to SPICE Matching Statistics 154

Figure A-7: 40nm NCH SS g_o LUT Queries to SPICE Matching Statistics 155

Figure A-8: 40nm NCH TT go LUT Queries to SPICE Matching Statistics 156

Figure A-9: 40nm NCH FF g_o LUT Queries to SPICE Matching Statistics 157

Figure A-10: 40nm NCH SS C_{GS} LUT Queries to SPICE Matching Statistics 158

Figure A-11: 40nm NCH TT C_{GS} LUT Queries to SPICE Matching Statistics 159

Figure A-12: 40nm NCH FF C_{GS} LUT Queries to SPICE Matching Statistics 160

Figure A-13: 40nm NCH SS C_{GD} LUT Queries to SPICE Matching Statistics 161

Figure A-14: 40nm NCH FF C_{GD} LUT Queries to SPICE Matching Statistics 162

Figure A-15: 40nm NCH TT C_{GD} LUT Queries to SPICE Matching Statistics 163

Figure A-16: 40nm NCH SS N_T LUT Queries to SPICE Matching Statistics 164

Figure A-17: 40nm NCH TT N_T LUT Queries to SPICE Matching Statistics 165

Figure A-18: 40nm NCH FF N_T LUT Queries to SPICE Matching Statistics 166

Figure A-19: 40nm NCH SS N_F LUT Queries to SPICE Matching Statistics 167

Figure A-20: 40nm NCH TT N_F LUT Queries to SPICE Matching Statistics 168

Figure A-21: 40nm NCH FF N_F LUT Queries to SPICE Matching Statistics 169

Figure A-22: 40nm PCH SS I_{DS} LUT Queries to SPICE Matching Statistics 170

Figure A-23: 40nm PCH TT I_{DS} LUT Queries to SPICE Matching Statistics 171

Figure A-24: 40nm PCH FF I_{DS} LUT Queries to SPICE Matching Statistics 172

Figure A-25: 40nm PCH SS g_m LUT Queries to SPICE Matching Statistics 173

Figure A-26: 40nm PCH TT g_m LUT Queries to SPICE Matching Statistics 174

Figure A-27: 40nm PCH FF g_m LUT Queries to SPICE Matching Statistics 175

Figure A-28: 40nm PCH SS g_o LUT Queries to SPICE Matching Statistics 176

Figure A-29: 40nm PCH TT g_o LUT Queries to SPICE Matching Statistics 177

Figure A-30: 40nm PCH FF g_o LUT Queries to SPICE Matching Statistics 178

Figure A-31: 40nm PCH SS C_{GS} LUT Queries to SPICE Matching Statistics 179

Figure A-32: 40nm PCH TT C_{GS} LUT Queries to SPICE Matching Statistics 180

Figure A-33: 40nm PCH FF C_{GS} LUT Queries to SPICE Matching Statistics 181

Figure A-34: 40nm PCH SS C_{GD} LUT Queries to SPICE Matching Statistics 182

Figure A-35: 40nm PCH TT C_{GD} LUT Queries to SPICE Matching Statistics 183

Figure A-36: 40nm PCH FF C_{GD} LUT Queries to SPICE Matching Statistics 184

Figure A-37: 40nm PCH SS N_T LUT Queries to SPICE Matching Statistics 185

Figure A-38: 40nm PCH TT N_T LUT Queries to SPICE Matching Statistics 186

Figure A-39: 40nm PCH FF N_T LUT Queries to SPICE Matching Statistics 187

Figure A-40: 40nm PCH SS N_F LUT Queries to SPICE Matching Statistics 188

Figure A-41: 40nm PCH SS N_F LUT Queries to SPICE Matching Statistics 189

Figure A-42: 40nm PCH FF N_F LUT Queries to SPICE Matching Statistics 190

Figure C-1: Screen Capture of the Γ-powered Website 403

Figure C-2: Zoom-in on leftmost part of the circuits’ table 404

Figure C-3: Output DC level shown on Mouse-Hover Event 405

Figure C-4: Heat-map of Thermal Noise vs transistor size 406

LIST OF ACRONYMS

RAMSpice – CSU version of NGSPICE-derived SPICE simulator with extensions implementing this research

cTree – Context Tree – a hierarchical database designed to provide access, save and load objects relevant to this research

Γ - Gamma – a circuit evaluating and sizing system

LUT – lookup table – an object made up of array and header that serves as a transistor-level model for an electrical property

LIT – linear interpolation table – an extension to LUT, designed to simplify interpolation at the expense of allocated memory.

HIT – hierarchical interpolation table – a tree-structure to replace the LUT, with heterogeneous cells

PAT – Pareto associative table – a vector of circuit records and a set of insertion and query methods, designed to keep the inventory and retrieve it according to Pareto dominance rules

ΓCC - Gamma Circuit Compiler – circuit netlist to C code generator

ΓCE - Gamma Circuit Evaluator – a per-topology compiled code that calculates a set of performance properties out of sizing parameters

1. Introduction

1.1. Electronic-Design Automation

An early observation made by Gordon E. Moore, a co-founder of Intel, is that the number of transistors on integrated circuits will grow exponentially in time[1](Figure 1). One immediate outcome of this proven growth is that manual design has become extremely difficult and rare.

Microprocessor Transistor Counts 1971-2011 & Moore's Law

Transistor count

Date of introduction

curve shows transistor count doubling every two years

Figure 1: Moore's law demonstrated by x86 processors family [2]

The electronic circuits segment of computer-aided design (CAD) was transferred from an accelerating improvement to necessity. Out of the two main signal domains, analog and

digital, digital designs enjoyed better success in automation[3]. The reasons for that include the fact that digital systems lend themselves better to hierarchical modularization and divide-and-conquer strategies. Since typical systems-on-chip (SoC) are mostly digital and pure analog chips tend to be much smaller than digital chips, acceleration efforts prefer digital designs for better return on investment (ROI). After decades of digital design automation, there are designs that start in highly abstract design languages and go through the flow of synthesis, placement, routing and layout in a relatively short computation time and minimal human intervention. The doubling of the number of transistors every 18 months, as is now the commonly accepted version of Moore's law, happens mainly for digital designs. Manufacturing technology presents more challenges than circuit complexity though[4]. The transistors themselves and interconnects are becoming more and more complex. This makes the design of the same circuit in different technologies an ongoing and increasing effort. For digital designs, challenges of transistor-level and component-level complexity met with "more of the same" approach to its toolset. For analog designs, the design process is stuck in trial-and-error.

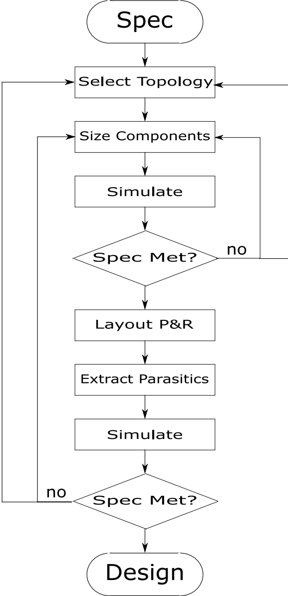


Figure 2: Traditional Analog IC Design Process

1.2. Analog Design Automation

Analog circuit design automation has been lagging significantly behind its digital counterpart. The main reason for this is the complexity of evaluating circuits against users' specifications and a large number of competing design goals in the optimization process. Since the 1980's, many software tools attempted to improve design productivity and even provide fully automated synthesis of analog circuits[5][6][7][8]. Most did not mature beyond academic prototypes. The few that made it into market and attracted customers[9] did not scale up to complex topologies and failed to migrate along each generation of silicon technology[10]. Yet, demand for lower power consumption, lower cost, and higher performance for analog circuits is increasing[11]. Designing analog circuits to meet design specifications has become an increasingly challenging task[17]. The need for practical solutions to improve design productivity is ever present in the semiconductor industry, where time-to-market project constraints push the analog design efforts closer to a bottleneck position.

The majority of the existing tools have the following characteristics in common:

1. They start with a topology plus performance constraints and attempt to generate production-ready circuit autonomously without any designer feedback in the process.
2. They use internal or en-wrapped simulations to evaluate solutions during the optimization process.
3. They run all tools on-premises, as opposed to remotely hosted, requiring customers to provide the hardware and sometime complementary software.

To circuit designers, the existing tools and approaches can be viewed as black-box approaches.

This type of black-boxed evaluation based optimization has several disadvantages:

1. For approaches using the SPICE engine for their performance evaluations, it typically takes long time to run to completion, breaking the natural flow of users' decision making process.
2. For approaches using high-level algebraic formula for their performance evaluations, the results are typically far from desired optima due to their inabilities of capturing circuit's high order effects with even complex algebraic formula.
3. Users often have to constrain the design problem well and be vigilant about the tasks they hand-over to the machine, because of the high risk of starting a long run that ends with results that are outside the constraint region. Without knowing how tools operate, this is extremely difficult to do.
4. The internal optimization process in the flow does not allow designer feedback. Although fully automated optimization flows are useful in solving high volume of designs with predictble outcomes (e.g. ASIC place & route), for more unpredictable analog designs designers often have better insights on design tradeoffs.

Existing attempts make a common mistake of putting the machine at the center of the flow by prioritizing functional features and failing to give users the control to run optimization steps and stages interactively and intuitively[12]. Incorporating designers' intent into the flow is crucial in analog circuit design process as the number of competing design goals are often too numerous for any algorithm to handle successfully. In most existing tools, users are kept out of

the loop once the optimization process has started and then are given the choice of accepting the final result or re-spin it, if those are not satisfactory.

Furthermore, web-based tools do not have the luxury of running processes of A-Z optimization flow in hours. Commonly accepted expectations from internet responsiveness[13] dictate much shorter response time, continuous controllability, and intuitive and dynamic visualization of design processes in progress. These challenging expectations can also be opportunities for a new class of analog design tools that put designers back at the helm of the design flow. Web applications for sizing transistors and designing analog circuits must focus on the least computationally-ambitious tasks and allow users to navigate between them and connect the flow. One aspect of design automation that can be a good fit for the web is mapping and visualizing solution spaces of topologies and specs to provide users with performance trade-off they need to make engineering decisions.

Stand-alone or tool-integrated SPICE simulations decouple the task of evaluating circuits from the optimization engine. The biggest benefit of using generic SPICE engines is that technology-specific parameters and models[14] can be integrated and modified with foundry characterized behavior with little tool vendor involvement. However, the silicon-accurate results provided by SPICE come at a cost. Evaluations of reasonably sized analog circuits include overheads, such as topology analysis, repeated calculations of transistor physical characteristics, and simulations of elaborate test harnesses, necessary to adapt circuit property measurements to one of the generalized simulation types: DC, AC and Trans. The amount of overhead can make the repeated evaluations required during analog circuit design and tradeoffs extremely inefficient.

This impact can be even worse for analog circuit design tools in the SaaS environment where a large number of circuits need to be evaluated during a typical HTTP transaction interval.

On-premises software is a common feature to most contemporary industrial EDA software. High demand in computational resources makes tool vendor focus on software alone, leaving the customer to provide their own hardware. The complexity and diversity of applications gave birth to the classic EDA support structure. Tools are coded by EDA vendors with a vast range of algorithms and configurations. The task of choosing between all the configurations is the responsibility of customers, who are left to determine best usage of input scripts, configuration files, and GUI's. Vendors have to employ an army of field applications engineers (FAE) to help customers determine the best configuration and keep the vendor itself up to date with the market needs. An FAE, like the software itself, is most often embedded on-premises with the user teams. This semi-automation structure is expensive and cumbersome. Effective SaaS systems attempt to implement crowd-sourced features, peer networking, and remote help forums to eliminate the need for human-based support structure. Direct communication channel can improve tool performance with automatically collected usage statistics. Certain results can be shared and reused between customers in a behind the scenes automated learning and archiving system. New features, conceived automatically from public-demand chats and surveys, are available instantly, without any user effort or need of IT support.

1.3. The Proposed Γ Analog Design System

This dissertation presents an alternative to the optimization-centered approach. The commercial failure of tools that offer a complete automation of the manual design process is contrasted by the relative success of tools that offer only SPICE-based evaluation and let the

designer drive the process. The proposed system is constructed to give designers the data they would normally gather from repeated trial-and-error SPICE runs and aid them in navigating the solution space towards an optimum.

The run-time penalty associated with SPICE analyses is normally accepted as a price paid for accuracy. However, it can be avoided by using alternative transistor-level modeling strategy[21] and circuit representation, both selected to accelerate circuit evaluation, with reasonable accuracy. Improving the responsiveness of circuit evaluating systems is a leading goal in providing a designer-centered tool. Instead of adapting an existing tool class to the Software-as-a-Service (SaaS) trend[29][30][31], the proposed system is designed from the beginning to benefit from the advantages presented by the web medium. Such advantages include sharing hardware resources between users, sharing calculation results anonymously, and streamlining deployment and support.

Due to the use of novel transistor modeling techniques, tightly integrated components for faster evaluation and better tradeoff options, and intuitive SaaS user interface, the proposed Γ analog design system is capable of producing sweep analyses in one and two dimensions in under 2 seconds and give the user graphic representation of the solution space and tradeoffs accociated with topologies and performance requirements. A reasonable accuracy of less than 10% circuit-performance deviation from SPICE is achievable with memory requirements in the magnitude of 100MB, for a current analog circuit manufacturing technology of 40nm. ….

2. Existing Research

2.1. Overview

With the tremendous progress made in SPICE simulation techniques[15][16], academic research in the field of analog design automation has focused mainly on automating the manual design flow[17]. This was done by employing several optimization and synthesis strategies on one hand, while leaving circuit evaluation to SPICE on the other. Although most automation work attempted to replace the designer's solution search[8], several algorithmic strategies that were explored in the past are adapted to be used in this work of creating designer aiding tool, rather than designer replacement tool. Design supporting tools has very little past publications to form comprehensive comparison. However, the architecture proposed here is inspired by incorporating desirable features among the existing systems reported so far, whereas overcoming some of their shortcomings.

2.2. Symbolic Analysis

SPICE simulation is performed by repeated numeric solving of the Modified Nodal Analysis matrix[18] (herein MNA see 5.3.1 for details). Symbolic analysis is an alternative to that approach, where the matrix entries are kept in their original algebraic terms representation and the solution to the matrix is performed once, to generate expressions for nodes of the circuits as functions of inputs, component sizes etc. Analytic models were demonstrated for circuit optimization by Gielen et al in their ISAAC/OPTIMAN system[7]. ISAAC was described as "symbolic simulator", in charge of calculating circuits' performance properties based on

algebraic representation of the optimized topology. OPTIMAN was an optimization engine based on simulated annealing (SA). The objective it received was a weighted cost function that represented the user's priority. That system used simplified equation-based transistor models and ran on a mainframe computer. MNA representation of circuits[18], was used extensively when transistor equation models could be simplified reliably to be able to find a deterministic solution to circuit constraints. Lately, topology analysis is limited to the initial steps of SPICE simulations and circuit-level equations play only internal role in it. Algebraic representation of circuit can be reused for an off-line compilation of design tool's topology templates. Doing this offline enables optimized compilation of topology equations and thus shortens response time compared to SPICE significantly. Offline MNA and compilation stages come at the expense of flexibility, because the system requires a compiled module per analyzed topology. However, this inflexibility is expected and tolerated in a system that offers a library of topologies that can be sized in a short schedule. On a circuit-level, there is no difference in accuracy between SPICE and a symbolic evaluation engine. However, when it comes to transistor level modeling, equation models trade accuracy for speed[19]. This shortcoming can be avoided by using alternative models that are faster than SPICE, but do not sacrifice too much accuracy.

2.3. Alternative Transistor Modeling

Equation-based transistor models incorporating underlying physics are used at the core of SPICE simulations[14]. Developed independently of foundries, they are fitted formulas of physical phenomena that have significant influence on transistor behavior. The number of significant phenomena grows with every technological generation and thus the number of equations and fitted coefficients that are supplied by foundries based on lab measurements. Two

advantages of equation models, accuracy and smooth waveforms make them well suited for simulation. However, the overhead associated with the long stack of equations, many not even needed for the bottom-line analysis, makes these models too time-consuming for evaluating large sets of circuits. Trying to accelerate the execution of these models by simplifying the equations made some commercial success[9]. However, speed advantages of simplified equation-based models were eventually negated by the inaccuracy stemmed from difficulties of capturing physical phenomena. As transistor scaling continues, those higher order physical phenomena became more important and made the models less accurate[10]. This weakness can be addressed by a modeling approach that's independent of the physics it is attempting to mimic: lookup tables (LUT). Yoon and Allen first suggested replacing equation models with lookup tables[20], with the goal of speeding up simulations. The result of their work was a mere 10% saving in run time, primarily because their method required run-time calculation of quadratic interpolation coefficients. This particularly expensive interpolation was chosen to ensure

1. Smoothness of curves to allow numeric derivation of I_{DS}
2. Better-fit to the small tables used back with memory resources typical to the early 1990's

However, when no simulation is required, smoothness is redundant and computational cost can be saved by using first-order ad hoc data location and interpolation techniques. Further, using contemporary hardware resources, larger tables can provide better resolution at low cost and the derivatives of I_{DS} can be precalculated and stored in separate tables[21]. Latest experiments with a LUT-based model showed that a 2MB table can generate a transistor's physical property (e.g. g_m) with <1% error compared to BSIM, in under ½ microsecond. The same hardware setup measured 28usec per BSIM query.

2.4. Multi Objective Optimization

Breadth-first analysis on a given circuit is not a new or uncommon concept. Sweep-analysis and scripted simulations[22] are already standard in every existing analog design environment[23] (ADE) offered by commercial vendors. However, an automated sweep is often a simple loop, lacking search objectives and often limited in dimensionality. Scripted search can potentially perform any test algorithm[24], although the overheads associated with running an interpreted algorithm that launches costly SPICE runs make it much less desirable for implementing a web experience.

Still, there are strategies and algorithms explored in previous research that can be useful for mapping circuit-size spaces and can be re-implemented on much faster software environment. Ant colony optimization (ACO)[25], for instance, is a popular approach for populating a multi-objective Pareto front. Other parallel algorithms, such as particle-swarm (PSO)[26] and genetic algorithm (GA)[27] have been researched thoroughly. However, efficient implementation with regard to users' expectations was largely overlooked. Some of these approaches could power offline preparation steps of web-tool database and some simpler ones (e.g. simulated annealing) could serve in online optimization.

2.5. Design-Supporting Analysis

Binkley et al. presented a "MOS design tool", that's a graphic calculator of key transistor performance properties[28]. Its novelty is not in promising an end to end automated design solution, but in providing immediate data to support design choices and calculations. The tool's engine was based on a simulation-level model, which is an accurate but slow batch of equations.

It did not offer automated search for optimal solution, but rather functioned as transistor-level behavior calculator to assist a design process. A similar tool, Sizer, was built at Colorado State University[21] in a web format to demonstrate the usage of a table based models. The tool was using gradient-based optimization algorithm to search for transistor length and width that meet a spec in a given operating point. This search required repetitive evaluation of the transistor's models, which in turn required faster evaluating model. Sizer's transistor-level models were, therefore, based on lookup tables, rather than physical equations.

2.6. Web-based Tools

Most analog design tools[29][30][31] found on the web are simply a web-adapted version of the on-premises software that's been part of the industry for more than three decades. One can find "integrated design environment" for digital[31] and analog[29][30] design disciplines, which are only different from their on-premises predecessors in one aspect: they include internet-collateral features that are easy to add and expected in every other creative website. Such features include user forums and blog outlets. The business model of online EDA tools is more web-oriented: subscription fees and/or 3rd party advertising revenue. Updates and feedback are also in the spirit of web-apps, such as the ones made famous by the Google Company. However they make no computational utilization of web-specific capabilities, mainly because they only attempt to mimic the expected functionality of on-premises software that was designed without those capabilities available to begin with. There are no centrally-shared calculations, for instance, because the depth-first analyses they are designed to do do not lend themselves to archiving and reusing results anonymously.

2.7. Hierarchical Design

A class of analog design generators assumes the existence, ability, efficiency, and accuracy of simulation engines, either commercial or freeware, and focuses instead on facilitating the design flow in a higher level. Such design tools offer flow management, user intent record, replay and reuse framework and generator-generators through high level abstraction of circuits. Often referred to as "Top-Down tools"[32] In this category there are tools like the historic interactive design tool for analog CMOS circuits (IDAC)[6], OASYS[33] and most recently the Berkeley Analog Generator (BAG)[34].

BAG is a Python-scripted framework designed for "closing the gap between designer and CAD communities" [34]. Both IDAC and BAG are knowledge-based, in the sense that they rely on packaged cell-level libraries, and predefined design procedures. BAG includes an automation framework for helping designers create those procedures and enrich their library with PyCells, a Python layout-generator standard. PyCells is similar to Cadence SKILL[35]. It does not include its own circuit-evaluation software and therefore its ability to search for optimal sizing is bound by the commercial simulator it calls.

For sign-off quality of post layout circuits, on-premises, large, SPICE-accurate tools are still needed. However, for a preliminary analysis of topologies' fitness to perform a specified analog performance, a fast web-based application that requires neither setup nor dedicated hardware is more fitting. Such application can rapidly produce trade-off graphs and heat-map visualization of sizing to performance dependence with few clicks of buttons and can potentially be accessed from any web client.

2.8. Commercial Attempts

Borrowing the concept of EDA from digital-domain toolset, both academic research and industry have prioritized creating synthesizers of sorts. Cadence provides an optimizer plug-in to their Virtuoso design environment, NeoCircuits[36], which serves as a top-level optimization loop on top of their simulators, Spectre[37] and UltraSim[38]. Its algorithmic infrastructure is based on GA and drives autonomously to a single solution that satisfies the spec. This architecture of using a simulator as a black-boxed evaluation engine in an optimization loop was offered since the DELIGHT[1] project. Success is to be determined by this tool's adoption outside research institutes[39]. Synopsys offers its own automation solution, Laker, which promises a "complete solution for analog, mixed-signal, and custom digital design and layout"[40]. Laker is geared mostly to layout and design-environment, rather than optimization of circuit sizing. The aforementioned Barcelona Design made an attempt to break away from SPICE-based evaluation, but failed to convert its tool to the next technology and beyond its narrow topology set. Its equations-based evaluation was not accurate enough for the next generation of technology and not fast enough for bigger circuits[10]. However, EDA industry did succeed in providing design environments, such as Cadence's ADE[41] and websites that mimic it. Those tools, while not automating design, show that a preferred design flow is to keep the engineer at the center. A lesson learned from the failure of commercial synthesizers and acceptance of environments is that customers want tools that provide them with accurate, useful and timely data needed to navigate toward a solution. This type of automation fits the web as software platform, because of its native features of interactivity and prompt data visualization.

2.9. Summary

The research presented in this dissertation is fundamentally different from the work reviewed in this chapter, in the sense that it is designed to enhance designers' solution search by giving them rapid feedback and data visualization of circuit performance trends and tradeoffs. It is also different in being developed primarily for a web environment, where rapid responses and controllability are not optional.

However, past automation strategies that were explored for full automation, and on-premises tools can be revisited and adapted for implementing the system described here. Symbolic analysis, LUT-based transistor modeling, and knowledge-based cell libraries can be used to implement a fast, responsive, and accurate circuit evaluation and optimization tool that is offered as an internet service.

Overcoming the constraints and using advantages of the internet and the cloud infrastructure can only be done efficiently when the system is designed from scratch to perform in these environments. The needs and technical analyses that were done for this research derived the required performance and features to meet the specific challenges posed by user expectations from an online tool. This is a major conceptual break from the trend of adapting on-premises software to be served from the cloud and designing new software as a service along the familiar lines of application categories.

3. Motivation for Proposed Architecture

One of the important aspects of designing an analog circuit is finding the right transistor sizes in a circuit topology that yield a required performance. A major challenge in that search is lack of information about the tradeoff trend among all competing requirements during the search process, given a circuit topology and manufacturing technology. A designer often choses to run many single or sweep simulations to understand how circuit performance parameters are trending relative to circuit physical parameters, and ultimately makes design decisions based on his/her own experience without necessarily gaining a more complete landscape for design tradeoff. This long process can be helped by providing a graphic mapping of a topology's performance Pareto front. A tool that can create graphical information for a designer about what a topology can and cannot do, what needs to be given-up by certain amount in the spec in order to find a solution and the expected costs of meeting the spec in terms of area and power. The tool needs to be accurate enough to provide a designer with circuits requiring few tweaks to achieve the design sign-off status.

Furthermore, requirements from web-based tool are driven by users' expectation of short response time[13]. Previous top-down technical analyses of system's requirements always fell short of reaching the bottom layer of non-SPICE based circuit evaluation with sufficient accuracy. Thus, keeping SPICE as the circuit evaluation tool of choice. Replacing SPICE with simplified transistor and circuit level equations gave systems the required speed boost, but with significant accuracy loss that could not scale to bigger circuits or advance technologies. One of

the goals of this research is replacing SPICE with an evaluation engine that is accurate enough to generate useful results and fast enough to serve as the backbone of a web service.

The following factors that slow down SPICE can be addressed to speed up evaluation:

1. SPICE's transistor-level models are physics-oriented fitted equations that go through many stages that are unnecessary for bottom-line operating point and performance parameters evaluation.
2. Topology analysis is done from scratch, with every invocation of SPICE. Stages like parsing and constructing pointer-linked structures take up time from the overall transaction schedule.
3. Analysis procedures, such as OP, DC and AC, are programmed in abstract fashion, employing run-time MNA and other algorithms that are circuit-independent. Since the SPICE analyses code is circuit-independent, it has to employ linked structures to construct and solve matrices according to the analyzed topology and given sizes, instead of directly calculating circuit-specific equations.
4. There are no built-in commands for calculating performance properties of interest. For example, calculating gain and bandwidth requires post-processing of generalized DC and AC analyses respectively.
5. SPICE's hardware requirements did not grow much since the time it was initially introduced to the PC niche. Its generational speedup is therefore mostly attributed to speed up in CPU performance. Very little has been done to leverage on growing RAM volumes or changing hardware distribution trends.

SPICE is analysis-oriented, as opposed to goal-oriented, which means it cannot be directed to find a solution according to the user's needs. Performing optimization with SPICE is therefore a task for an en-wrapping optimization software that is in charge of navigating the search, while SPICE itself remains a black-boxed evaluation program.

The research described here breaks away from the existing approaches of analog circuit design. The new system was constructed with circuit sizing in mind, with the goal to create an engine that is tailored to provide specified performance goals and decision-supporting graphics as fast as possible.

The main construction guidelines were:

1. Prefer data look-up over calculations – starting from a lookup table transistor modeling and continuing with archived Pareto containers, specifically designed for quick queries of pre-evaluated circuits.
2. Precompile topology-specific calculations – via generated C or tailored VM code, every analysis that can be done offline saves run time from online queries. The most time-consuming code optimization can be done offline to produce make circuit-evaluating routines as fast as possible.
3. Include performance property specific code in the generated C – instead of relying on the SPICE-standardized repertoire of OP, DC, AC and TRAN analyses, and calculate performance properties that can be matched to spec, directly from the transistor's parameters and operating point.

4. Support automated tuning of circuit parameters according to constraints and make evaluation procedures that were traditionally kept for SPICE wrappers and scripts, more efficient.
5. Pre-calculate topology-specific solutions – use circuit generators and retention guidelines to produce a general set of popular circuits offline that can be filtered as first response to spec and be used as baseline for optimization.
6. Keep optimization runs interactive and on-going until session expiration. Instead of building a complete automated flow, dynamically build up solution set in the database and let users participate in decision making based on best-so-far results. This way, even an atypical minutes-long online task remains interactive and avoids the risk of losing designers' intent in the process.
7. Make use of asynchronous web interface schemes (e.g. Ajax), leaving the user with a sense of using a search-engine, rather than a classic synthesizer. Similar to other web-based search applications used in popular sites, the quality of results should be developed dynamically over time with an animated picture of evolving solution. This way, users receive initial crude (yet useful) results and get closer to optimum results as the design progresses.
8. Curate optimized solutions for future queries in a background process – crowd-sourcing optimization results means that every spec is calculated only once for the entire user community.

Careful division of tasks between online and offline processes is the key for ensuring web usability. Contemporary hardware resources enable more of this separation than was available in the time when circuit simulators took the lead in evaluating circuits. This means that for cloud-

based analog EDA there are more opportunities to pre-calculate solutions and data leading to solutions and thus accelerate both automated and manual design tasks. This work presents a system that is architecturally designed to benefit from these opportunities. It enables a web application that provides users with a visual aid for searching an optimal solution according to dictated spec, topology and manufacturing technology. This web application is geared to present capabilities and tradeoff trends in navigational maps that are responsive and to allow progress and back-track convergence toward a solution.

4. System Architecture

4.1. The Γ System

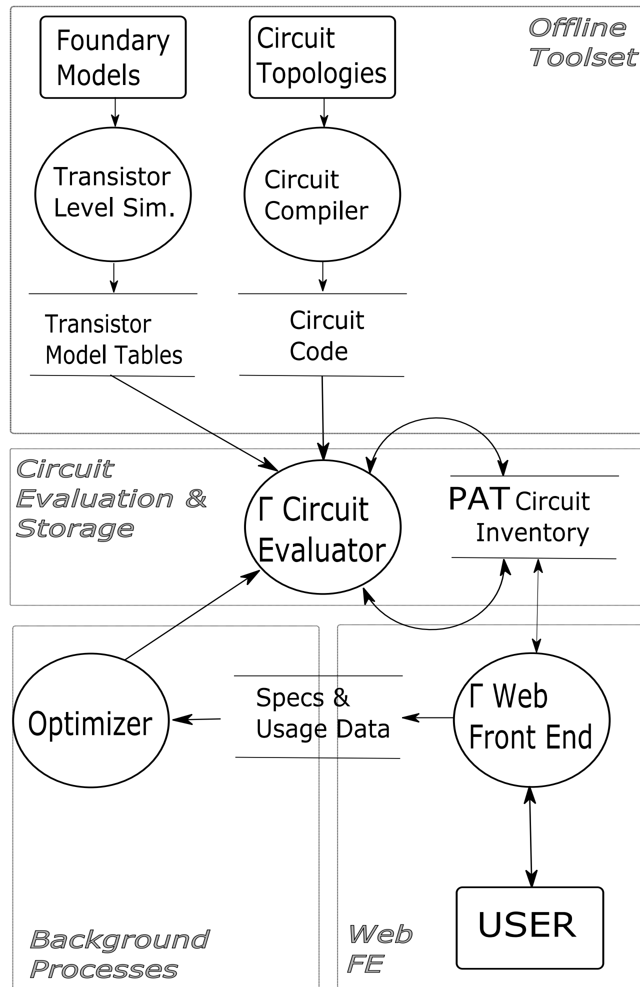


Figure 3: High-level Dataflow Diagram of the Γ System

The Γ[43] system[1] (Figure 3) consists of four main subsystems:

1 Offline toolset for transistor-level modeling, topology compilation and performance mapping. This subsystem consists of a transistor-level model extractor and a circuit-level compiler.
2 Online toolset for circuit evaluation. The circuit evaluator uses the transistor model tables and the compiled circuit code, created by the offline toolset, to generate an initial set of circuits that are stored in the PAT, a circuit inventory. The same evaluator is used in online circuit-evaluation tasks.
3 Online toolset for web front-end and session management. This is a set of web-browser run methods that translate the user's requirements and session controls to server commands and website graphics.
4 Background daemon for optimizing and managing the circuit inventory. This is a process that receives usage data from the web front-end and runs automatic searches that enrich the circuit inventory with circuits that may be of interest to the users' community.

[1] The term "Gamav" (Aramaic for "its Gamma") appears in an ancient Jewish law compilation, known as the "Babylonian Talmud", as an attribute of a lost and found object [42]. The term is understood to mean the length and the width of an item, as hinted by the shape of the Greek letter. Γ was borrowed to name this research, because of the main usage of the system: determining lengths and widths of MOSFET channels in a circuit.

The system operates in two modes: online and offline. The offline mode generates as much data as possible to alleviate the execution time bottleneck of online circuit evaluations. Online mode is used only for web transactions.

The two modes of operation share two of the main engines of Γ:

1. Γ Circuit Evaluator – calculating a circuit's performance out of sizing parameters, using precompiled code and lookup table based transistor-level models. This building block (see section 5.4.1 for details) is this research efficient alternative to the SPICE simulator.
2. Circuit Inventory Manager (PAT) – manages and stores pre-calculated sizing and performance figures of a set of circuits that are stored in a dedicated database that's designed for a set of candidate circuits relevant to given specifications (see section 5.5 for details).

4.2. Offline Toolset

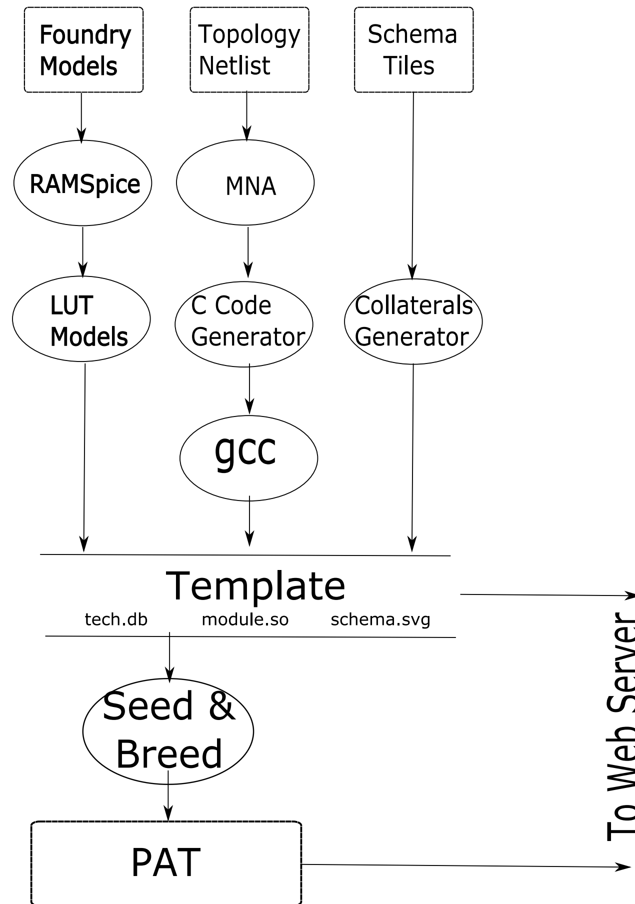


Figure 4: Offline Dataflow of Γ

Figure 4 shows three parallel processes that generate a database for online operations.

1. Characterizing a set of component-level models out of foundry models (see Section 5 for details)– this process starts with files released from a foundry to users of commercial tools, so the latter can configure their toolset to evaluate circuits that are to be manufactured by the former.

   a. The first stage of this tool translates the rich language of the commercial tool to flat listing of models, parameters, and values. This translation gets rid of function calls and other abstractions found in the foundry files and generates equivalent models that can be read by a generic SPICE.
   b. RAMSpice takes the simplified models and runs characterization loops that measure key transistor-level parameters (e.g. I_{DS} and g_m) in a range of channel sizes and operating points.
   c. The results of the RAMSpice runs are stored in binary arrays, the lookup-tables (LUT).
2. Preparing binary module (either shared-object of UNIX or DLL for MS Windows) specific to a circuit topology.
   a. Starting from a SPICE-like netlist, it uses a nodal analysis algorithm (MNA) to generate all the equations needed to establish operating point and performance properties.
   b. The tool generates a C code, containing all the functions needed to evaluate circuits, populate the PAT, and compose heat-maps from sweep analyses.
   c. The generated C code is sent to the Gnu Circuit Compiler (gcc) for final compilation.
3. Processing graphic and other information related to a topology. This tool prepares vector-graphics and other web-related data (see Section 5.6.1 for details) for visual representation of sized circuits. The input to this tool is an array of tiles that describe the schema of the analyzed circuit. The tool uses a library of predefined tiles for

components, interconnections, and terminals to translate that array to web-presentable scalable vector graphics format (SVG).

4. The final offline tool prepares an initial inventory of circuits (see section 5.5.1 for details) that can be queried later according to a users-provided spec. There are several stages to this process, starting with complete random generation of circuits and ending with performing random size variations on saved circuits. The tool uses the binary executable generated by the circuit compiler and the transistor-level LUT to evaluate circuits on random selection (seed) and alterations of stored ones (breed). The tool uses the domination principles of Pareto and other criteria to ensure that this inventory contains viable, useful and diverse set of circuit. Unlike previous three processes, this one cannot start before there are component-level models and topology-level code in place and therefore cannot be parallelized with the rest.

4.3. Online Toolset

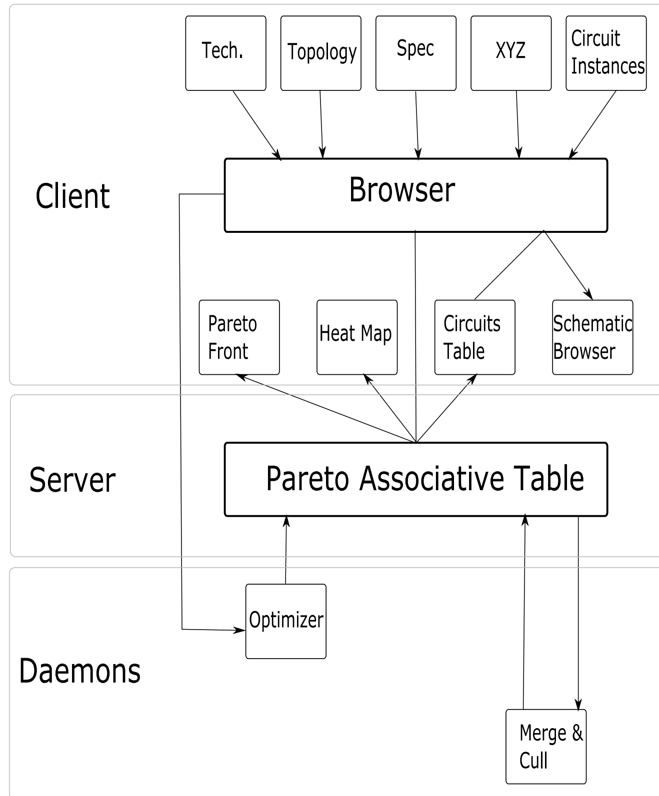


Figure 5: Γ Online Toolset

After logging in to the website, the designer can select a Technology and Topology for the solution search (Figure 5). A Spec form is filled out and the system produces plots interactively. The starting point for the search can be obtained by generating a Pareto front plot. The designer selects two properties from the specifications and the system presents their tradeoff in the form of a curve with markers that represent circuits stored in the Pareto Associative Table (PAT). The designer can add a circuit to the Circuit Table by clicking on one of the Circuit Instances. Each circuit in the Circuit Table can be used as a pivot for the next step in the search. The designer can now modify the pivot circuit by performing 2D parameter sweeps that present

on screen as Heat Maps, controlling the analyzed parameters by choosing the XYZ axes and clicking on modified circuits that step closer to optimum.

Each selected pivot circuit modifies a Schematic Browser with detailed information: length/widths of each transistor channel, node DC voltages, and bias values. Hidden from the designer, Merge, Cull and Optimizer daemons perform general data management tasks that keep the circuit repository in effective size, diverse, and relevant to designers' common queries.

The three software subsystems that participate in online activities are the client web browser, the Apache+Γ web server and background daemon processes. The client runs client-end JavaScript code that is dispatched from the Apache server. The web page is used to select technology and topology, type in a spec and then navigate in the solution space, stepping between intermediary circuits until a satisfactory one can be downloaded. Each step sends a request via the Apache server to Γ to produce a tradeoff graph or sweep two parameters to generate a heat map of an interesting performance property. Γ uses two resources in generating online data: Circuit Inventory (PAT) and Circuit Evaluator (ΓCE). The PAT is a vector of pre-evaluated circuits that is filtered according to a spec and can produce graphic representation of a tradeoff, given that spec. The evaluator uses precompiled code and a set of transistor-level models to calculate performance levels in X/Y raster.

4.4. Architecture Layer Model

At its core, Γ is structured to replace the traditional SPICE as circuit evaluation engine, with similar layer stack that's described in Figure 6.

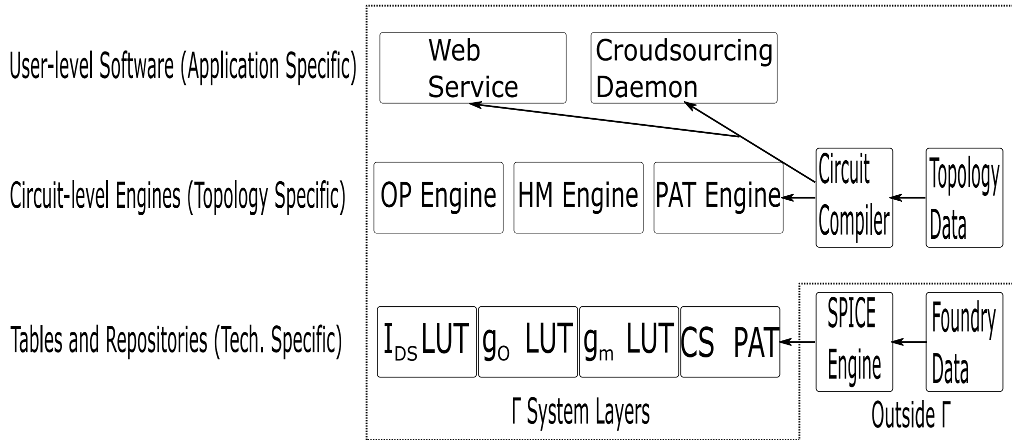


Figure 6: Layer Model of the Γ System

The bottom layer contains transistor-level models in the form of lookup tables (LUT), which recreate the functionality of simulation models, such as BSIM, in a much shorter computation time. These models contain minimal to no algebraic content, but instead capture every electrical parameter of the transistor in multidimensional array that spans available geometrical and operating point conditions a transistor may be in in a circuit. A LUT model contains its parameter's array, interpolation-assisting data and a set of interpolation functions for continuous-space lookup operations. In addition to the LUT, the bottom layer contains a set of sized circuits, stored with their component sizes, operating point, and performance properties to form samples of a topology-technology pair Pareto front (PAT). The data needed for the bottom layer is stored in a binary database that can be saved and loaded from files.

The middle layer contains a set of engines for evaluating circuits and performing various circuit search and data management operations. The engines are typically programmed in a meta-C language, which is used by a circuit compiler to generate C code that gets compiled specifically per circuit topology. The main engine in this layer is the operating-point (OP) engine. It receives a set of component sizes and reference levels and solves the DC voltage for every node of the circuit. This engine contains all the equations needed to calculate performance properties of the circuit and code for inserting sized circuits back to the PAT. Another engine performs higher level sweep analyses of the solution space in order to assemble a heat-map (HM). A set of smaller functions is compiled to an engine that populates the PAT in an initial sample that can be used as starting point for user queries.

The top layer is front-end software that uses the Γ engines to perform sweep evaluations, solution searches, graphic generation and data management to serve a user side application. This layer is coded in interpreted languages, such as Tcl and JavaScript. This layer is responsible for all web-related UX aspects on one hand and invoking the various engines on the other. Another top-layer subsystem performs data merging and filtering in the background. By merging data that comes from multiple users into a single repository, this process effectively crowdsources PAT samples and accelerates searches for popular specifications.

4.5. Software Components

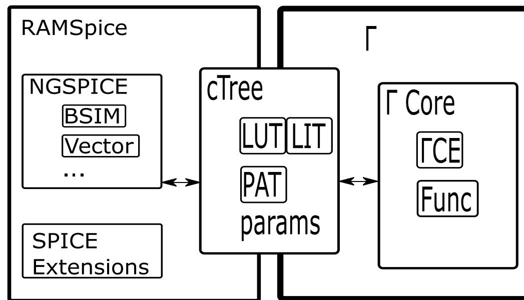


Figure 7: Block Diagram of Executable Software Components

Figure 7 shows the components of the RAMSpice and Γ executable software. Thick rectangles show components that are available as stand-alone, binary executables: RAMSpice and Γ. RAMSpice is the research version of NGSPICE. It includes all additional components and modifications needed for executing scripts and applications described here. When run by itself, Γ has no access to SPICE functionality inherited from NGSPICE and cannot use the Tcl commands that are specifically associated with the simulator. Γ contains all the circuit-evaluation features that are proposed as alternative to SPICE in web-based applications.

NGSPICE is provided to the public with a configurable make-file. The repository contains several modules that were added over the years by different contributors to the main SPICE simulator. Some of the modules downloaded with the repository were redundant to this research. They include:

1. X Windows support – no step, configuration or flavor of the system was intended to send graphics to a local screen or across a Virtual Network Computing (VNC) session.

The system is HTTP oriented and its clients are web browsers. Therefore, X-related code and compilation was dropped from the local repository.

2. Tcl support – the built-in Tcl support is partial, particular to general simulation uses and relies on data management and visualization package that receives too little community attention and support. An alternative, local Tcl support code had to be added and therefore most of the original one was discarded.

After organizing the local repository, a new compilation script was written that mimics the make file, but adds Tcl preprocessing stage and updates all compilation targets (Table 1):

Table 1: Six Executable Binaries Compiled by the Compilation Script

	Full RAMSpice	Γ
Silent	ramspice-silent	gamma-silent
Regular	ramspice	Gamma
Debug	ramspice-debug	gamma-debug

4.6. Language and Compilation

This research contains the NGSPICE open-source code repository, which started the first task of implementing a fast transistor-level model characterizer. Therefore, the C language made the most sense as primary implementation language. In the EDA world, C is often integrated with the Tool command Language (Tcl, or TCL) to provide it with an interactive front-end and scripting shell. Tcl is a Polish-Notation (PN) language, with roots in both the LISP and C-shell language families. Its primitive and relaxed grammar (or lack of, according to its inventor John Ousterhout [44]) makes it useful for some applications that were not intended by its creator:

It supports bottom-up programming tasks: implement primitive blocks in C, integrate next layer in Tcl, identify the critically-slow Tcl blocks, re-implement them in C and repeat the process. Building blocks, such as the LUT (see section 5.1.2.1 for details), were initially written as a Tcl command extensions. Once verified to be functioning correctly, they were added with upper layers in C.

It is very easy to implement Domain-Specific Language (DSL) compilers with a Tcl interpreter [45]. The interpreter is mainly a parser that reacts to the first token in a line as a command. Giving the interpreter some vocabulary procedures can transform it to a language processor and even full compiler to a language that simplifies domain-specific programming task. For example, drawing the schematic view of circuits was done via a concise DSL that was compiled to vector graphics (see section 5.6.1 for details). Even standardized languages used in this research were given local support with some useful extensions. For example, the SPICE netlist (.sp) was extended for topology constructs supported exclusively by Γ (.gsp see Γ Circuit Compiler block in Figure 8).

Tcl can be used as an alternative to the C compiler preprocessor (tclC see RAMSpice block in Figure 8). The standard preprocessor lacks programming features, such as loops, that prevent it from becoming a full code generator. Significant acceleration was achieved by converting run-time loops and branching to compilation-time. This would have been very hard and error-prone without a scripted meta-programming layer.

Another approach uses Tcl as the main language and embeds C command declarations as output of the script (*C see Γ Circuit Compiler and Γ Circuit Evaluator in Figure 8). Generated C,

of both language extensions, saves on coding and debugging time by providing the programmer with a grammar that is more flexible and more expressive than ANSI C.

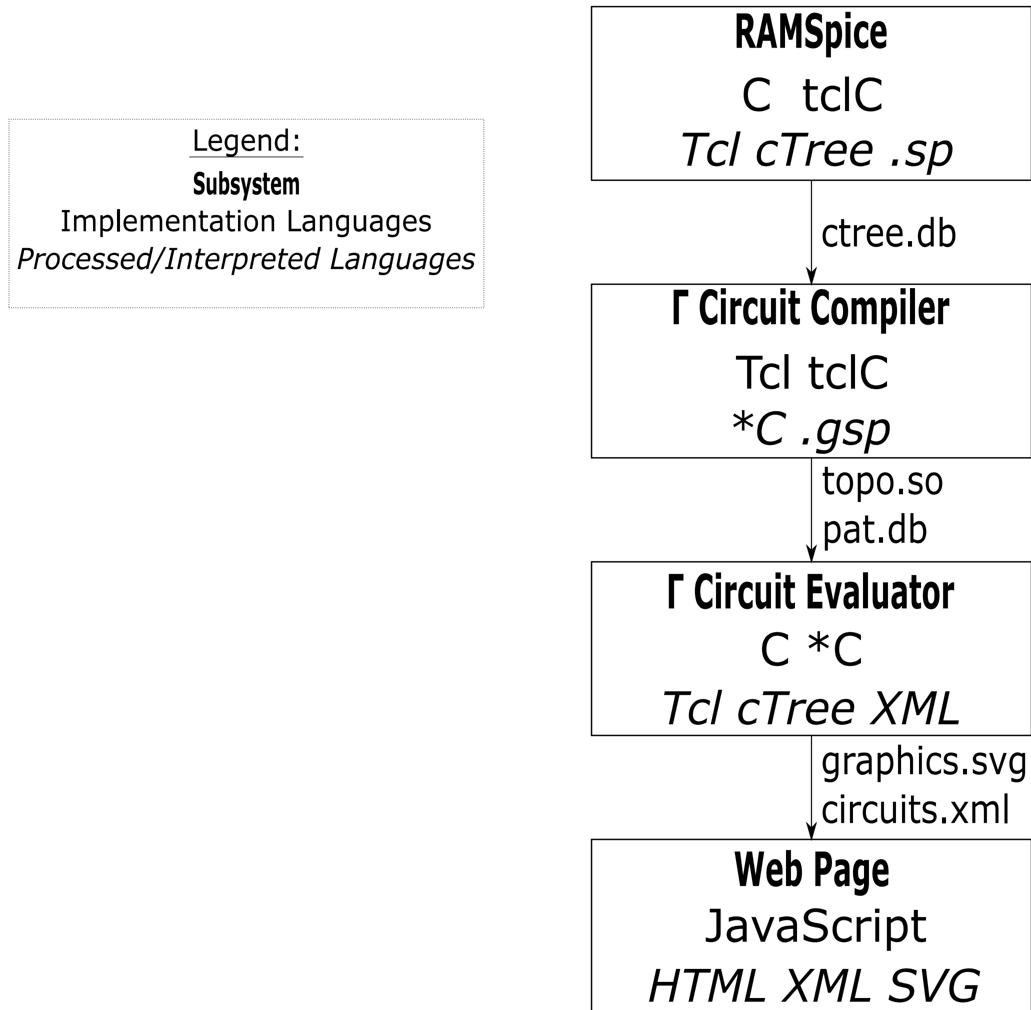


Figure 8: Implementation, Interpreted and Processed Language per Subsystem

4.7. Context-Tree Data Base

cTree (context tree) is the backbone database used in the implementation of RAMSpice and Γ. Both RAMSpice and Γ require a large collection of collateral data. Transistor-level LUT's, PAT's, parameters, and configurations need to be stored in a combined data model that can be saved in a single repository and loaded with every launch of a new process. The data is heterogeneous, composed of very different types and structures. It is also hierarchical in nature. The LUT set, for example, can be sorted by component type, speed and temperature corner, and modeled parameter. PAT's can be sorted according to topology and corner. The chosen data model is therefore a hierarchical tree, with support for several types at each storage node (Figure 9). Each hierarchy node and leaf in the tree is a "context". The children nodes of a context are "sub contexts", while parent node is a "super context". The uniform context hierarchy has the following advantages:

1. It allows programming functions that can be applied to any location in the tree, independently of the data's target purpose, a technique known as software module decoupling[46].
2. It allows the system to store multiple work spaces and be scaled up to multi-user support, without any significant change to the code itself. Each work space and user can be further scaled up to support multiple projects, tasks etc. by changing the hierarchy depth of the tree.
3. Tree structures are prevalent in design systems. Structured formal languages, schematic description formats, and XML can be mapped to tree structures. A general-

purpose tree support built into Tcl simplifies programming algorithms for tree-structured data and promotes a uniform coding style.

4. Joint and uniform central database simplifies saving and loading large and diverse data sets ("data marshalling") by using recursive functions.

1.

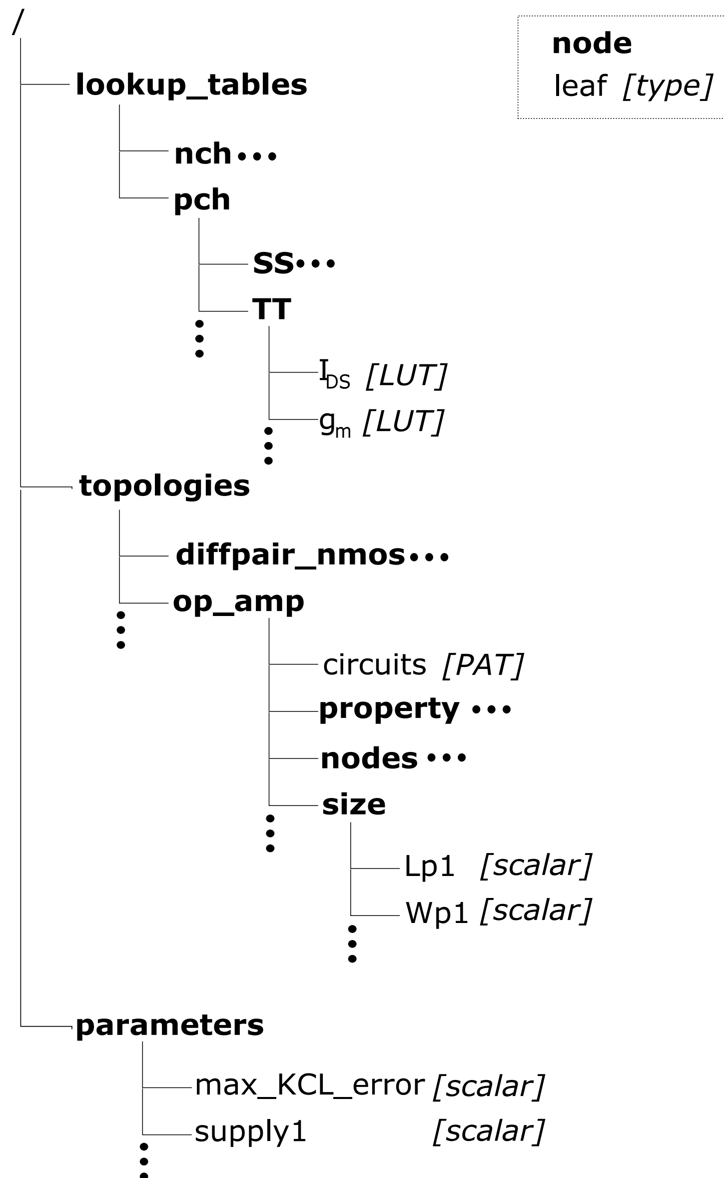


Figure 9: Example Segment of a cTree Structure

cTree contains the following components:

1. Tcl interface language - The context tree structure is homomorphic to a file system's directory structure, which makes the directory path notation useful as context notation in Tcl. Example: @ /parameters/Length = 1e-6. Script and user-side commands include automatic looping on sub-hierarchies, queries on context existence, and file load/save.
2. C API - A set of commands provides access to contexts and values from compiled C functions. The Γ circuit compiler (ΓCC) uses that API to automatically generate code that links local variables to memory locations in cTree.
3. Access commands to leaf data structures – LUT (see 5.1.2.1) and PAT (see 5.5) are two of the data structures that can be added as leaf contexts. The cTree interface language includes specific commands for creating and accessing the specific structures that are used by Γ to evaluate and manage sets of circuits.
4. Data marshalling – Any sub-tree can be saved to disk and loaded back under any context. cTree contains a hierarchy of functions that serialize the different types of contexts and pack them back in memory when loaded from disk.

4.8. Foundry Model Translation

Commercial tools that were developed from SPICE3 support superset formats for device modeling. These formats contain constructs that are not supported by the main distribution of NGSPICE, yet are essential for the understanding of the models' parameters. For example, the .model SPICE card can contain in its payload references to arithmetic functions and conditional multiplexers that make the models file much shorter than if every model and parameter were explicitly listed.

The other complication that the commercial-tool specific format presents is the usage of model bins. Bins model specific behaviors of a device per ranges of geometrical parameters. Two identically instantiated devices may get different model parameters, because one is longer and/or wider than the other. This scheme comes to compensate for the challenges of independently developed physical models (such as BSIM) in being fitted to advance manufacturing technologies. The fitted equations included with these models are better at predicting behavior of limited geometrical ranges than the entire available sizes of the device. Unfortunately, the original distribution of SPICE does not support this partitioning and thus bins have to be expressed as separate models.

To match SPICE simulation to a commercial tool's one, the following steps must be taken:

1. Internal references to external files and conditional sections must be followed by the interpreting script (flattening).

2. All function calls and variable references in the commercial model file must be evaluated and converted to explicit parameters and assigned values.
3. The hierarchy of device models and its geometrical bins must become separate models.
4. A collateral script must be generated, to enable netlist generators an automatic selection of the per-bin model upon instantiating a device.

The method for implementing this translator uses the Tcl interpreter as a domain-specific language (DSL) compiler. In such compiler, input files are first converted with a series of regular expressions into a Tcl script. The script is then executed with a library of the domain specific commands ("proc's") and a new file is generated. The Tcl DSL method has the advantage of creating compilers that are simple to code and debug. The function calls included in the original format are implemented by similar Tcl procedures and thus leave no room for introducing bugs. However, the Tcl interpreter is slow compared to dedicated binary executable and thus the ease of coding this compiler is offset by minutes-long execution. Tcl DSL is a better choice than compiled C converter, because the translation is done only once, when a new foundry manufacturing technology file is released to the commercial EDA industry and the run is done offline, without any efficiency implications on SPICE or Γ.

The output of this initial step is the following files:

1. Models file in the original flat SPICE .model format. The output file is a listing of models, parameters and values. There are no variables or other abstractions in that file. A single

component in the original technology files can become several components in the output file, due to binning.

2. Tcl array initializing script for downstream scripted netlist generators. Several offline and online procedures need this information to map a transistor's instance to its bin-specific model. This is done by following the lmin/lmax and wmin/wmax data stored in the provided array. Other information contained in this file is parameters of design rules that can be used for estimating a transistor's area.

5. Implementation and Applications of Γ

5.1. Transistor-level Modeling for Γ

5.1.1. Motivation

Γ's transistor-level modeling approach focuses on replacing equation-based transistor models with faster query tables. Equations are the most commonly used models. They represent a set of physical phenomena that have significant influence on transistor behavior of current generation. The equations are hard-coded, but key parameters are left to tech-files that are produced by foundries to be used in SPICE-class simulators. There are two improvements the proposed approach achieves:

1. Decouple the model from physics.
2. Trade hardware resources for faster parameter calculation.

For example: short channel effects on V_T that are covered by BSIM equations are only a step in the calculation of I_{DS}. Similarly, intermediary physical values, such as depletion layer width and effective channel length are not useful for the purpose of obtaining I-V operating points. Γ is only interested in the current through the channel, dynamic conductance and capacitance values for small-signal analysis and some secondary figures, such as noise and manufacturing statistics. All the physics-related steps can therefore be consolidated to a single model that converts geometry, voltages and process/temperature corner to the needed values.

To implement the reduced set of relationships, equations do not need to be fitted. This approach made sense back in the 1980's, when a typical machine's memory size was less than 1MB. Today, there is a lot of room to partition the model to small areas in the transistor geometry and operating space. Indeed, accuracy deficiencies in standardized models are already overcome by the industry by partitioning geometry to look-up tables, in the method mentioned above and commonly-known as "binning".

5.1.2. Implementation

RAMSpice is a collection of a SPICE simulator (NGSPICE), database structures and other extended simulation and circuit analysis features needed to generate transistor and circuit level layers for Γ. RAMSpice started from the NGSPICE open source SPICE simulator that is distributed under the BSD license and considered to be the current version of the original Berkeley SPICE dynasty. The simulator was added with C-implemented Tcl interface to a hierarchical database called a "Context Tree" (cTree). This locally-developed database is the infrastructure on which binary data structures are implemented. It allows storage and retrieval of data in a hierarchy notation that resembles file systems' directory structure. cTree performs all the data marshalling operations needed for saving and loading all supported data types to and from disk. NGSPICE's code itself was modified and added with extensions that accelerate copying data from its simulation result vectors to cTree and extend simulation capabilities to support features needed by this research. The main role of the extended simulator and added database is providing accelerated characterization tools to create a fast transistor-level model that is an alternative to the BSIM model SPICE uses for simulation. Once that goal was accomplished, Γ was built to capitalize on the acceleration of both the new transistor models and the benefits of

symbolic analysis of circuit topologies. Γ operations interface cTree and use a special structure to store circuit evaluation results in it. Higher-level functionality, such as offline preparation procedures and web front-end scripts are implemented in Tcl, using the low-level commands built into RAMSpice in general and Γ in particular.

5.1.2.1. Look-up Tables

The look-up-table (LUT) used in our approach was designed to meet the following goals:

1. Limitless dimensions, but typically 4-5 per table
2. Data-accelerated interpolation
3. Maximal size of GB's per device
4. Binary storage format

The LUT's are populated by sampling transistor geometries, process-temperature corner and operating points in a matrix of predefined values.

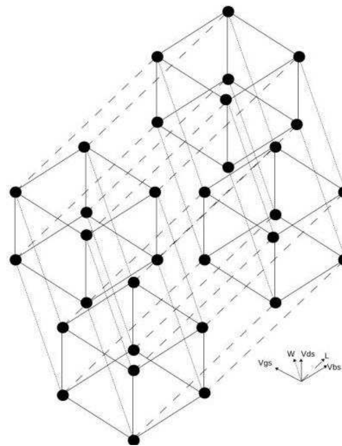


Figure 10: Five dimentional hypercube: a cell in I_{DS}, g_m and r_o LUT.

After populating the basic array, the LUT can be expanded further to allow quicker queries in exchange for more memory. The LUT is a multidimensional array, dividing the parameter space into cells (Figure 10). Some cells' values may be close enough to a special case that allows faster interpolation than Lagrange over a hypercube. The faster interpolations require additional parameters, which require an additional Linear-Interpolation-Table (LIT) that can reduce the evaluation time complexity from 2^N to N (number of dimensions) per cell.

The LUT raw entries define the corners of interpolation cells. Therefore, to support M_i intervals of parameter i, M_i+1 values need to be measured. The overall number of raw entries in the LUT is then:

$$|LUT| = \prod_{i=1}^{N}(M_i + 1)$$

In order to achieve adequate resolution, millions of parameter-combinations need to be scanned, which may take months to complete in a typical script-enwrapped SPICE tool. To overcome that, the NGSPICE source code was used to create an adapted version, RAMSpice, with fast access and manipulation of internally stored result vectors. RAMSpice is the single-executable platform that was developed to implement this model. It contains extensions to the Tcl interface already provided with NGSPICE. To accelerate the characterization process, two groups of inputs were implemented separately:

5.1.2.2. Geometry Parameters

The trivial sweep analysis loops scan each transistor sizing combination and produce characterization slice for each. However, nested loops that modify single-transistor proved to be

very slow. Instead of looping through the W and L measurements of the transistors, in a Tcl script, a test circuit was constructed to contain all the required transistors in parallel (Figure 11). This shifted the weight of sweeping through geometry from the Tcl interpreter to the SPICE circuit analysis engine.

5.1.2.3. Voltage Parameters

The harness voltages were scanned by using the .dc command to scan all the required value combinations. A much needed modification to NGSPICE was to enable more than 2 nested sweeps. Otherwise, sweeping V_{GS}, V_{DS} and V_{BS} together would require using a Tcl loop, which proved time-costly.

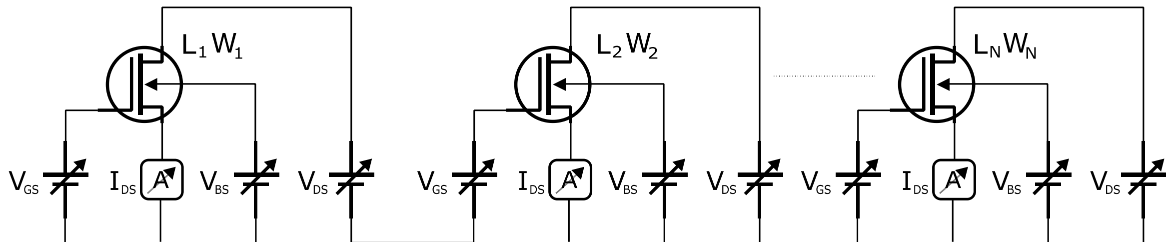


Figure 11: I_{DS} Parallel Characterization Circuit

5.1.2.4. Parallelization

Parallel execution of characterization loops saves time when there are enough computing resources to support all threads. The 8-thread machine dedicated to the characterization effort supported parallel execution of several corners simultaneously. The C command `fork()` was registered as a Tcl command and an infrastructure for split and merge was added as Tcl code. The main process generated the different characterization tasks and then forks to run each corner.

It then allocates the array for the LUT and waits in a polling loop until all forked processes are done. When the simulation results are ready, they are read from the /tmp directory and merged into the LUT's.

5.1.2.5. Reading and Post-Processing Simulation Results

On top of accelerating the input sweep, C code was added to allow fast extraction, on-the-fly processing of simulation vectors and forked-process parallelization (see B.B.5.1). The modifications reduced the characterization time from 5 hours for a single process-corner and low resolution (8 values per input) to under 30 minutes for 5 corners, high resolution (33-65 values per input) and 3 different modeled parameters.

All operating point parameters: I_{DS}, g_m and $r_{o,}$ are characterized using the same test circuit and DC sweep analysis. The difference between them is in the post-analysis of the simulation vectors.

Since:

$$I_{DS} = I_{DS}(V_{GS}, V_{DS}, V_{BS}, L, W)$$

$$g_m = {}^{1}\!/_{\varepsilon}\,[I_{DS}(V_{GS} + \varepsilon, V_{DS}, V_{BS}, L, W) - I_{DS}(V_{GS}, V_{DS}, V_{BS}, L, W)]$$

$$r_o = \frac{\varepsilon}{I_{DS}(V_{GS}, V_{DS} + \varepsilon, V_{BS}, L, W) - I_{DS}(V_{GS}, V_{DS}, V_{BS}, L, W)}$$

The three parameters are therefore extracted from 3 DC sweeps of I_{DS}, with appropriate shifts, and saving the internal SPICE vectors as binary data via added Tcl commands that perform the subtraction and division operations on the fly (see A.4.1).

5.1.3. Linear Interpolation Table

The optional linear-interpolation support was added as a post-characterization stage. This stage pre-allocates an array, similar in structure to the raw-samples one, but N+1 times larger, because it allocates N+1 entries per raw-data cell: one per slope approximation and one for the intercept value. The total volume of a LUT with linear interpolation table (**LIT**) comes to:

$$|LUT| = \prod_{i=1}^{N}(M_i + 1) + (N + 1)\prod_{i=1}^{N} M_i$$

Note that the added term for the linear-interpolation part is only a product of M_i's, without counting closing samples. To populate the linear-interpolation table, RAMSpice scans the raw-sample cells and performs simple regression loop (Alg. A). Each dimension is assigned an estimated slope "bucket", which is initialized to 0. There are two nested loops: external one scans the cell corners and internal one scans the dimensions. For each corner and each dimension, the sample value is either added to the dimension bucket or subtracted from it, depending on whether it is up or down that dimension, respectively. After the loop is complete, the buckets are normalized according to the number of corners and the physical size of the cell along the dimension they represent.

Algorithm 1: Fitting a hyper-cube's values to a hyper-plane slopes vector

```
Begin
    For each corner Cj in cell H
    Begin
        For each bucket Bi
        Begin
            If (Cj is on the top-face along dim i)
            Bi=Bi+Value(Cj)
            else
            Bi=Bi-Value(Cj)
        End
    End
    For each bucket Bi: Slopei=Bi/(2N-1)/(LengthOfDimi)
End
```

Finally, the intercept is calculated so that interpolating the center point of the cell yields the average of all corner values.

Each entry of the linear interpolation table is tested after calculation. The point of the test is to disqualify entries that fail to reproduce the original entries under predefined accuracy margins. An arbitrary 1% error margin was allowed, which can be changed to trade off accuracy for speed. Wider margin would disqualify fewer cells, which in turn would increase the percentage of fast-calculated queries. That percentage figure served also as a general health indication for the LUT. If the LUT to LIT conversion achieved fewer than 50% linear cells proportion, it was an indication that the resolution is insufficient to recreate the original model. A very high proportion (>95%) was an indication of over-sampling memory wasting. Failing cells

were marked by setting their intercept field of their entries to NaN (not-a-number). In case where the LIT option was skipped, be it for saving on memory or temporarily accelerating software debug, the interpolation still worked, albeit slower because fast-interpolation coefficients were not available.

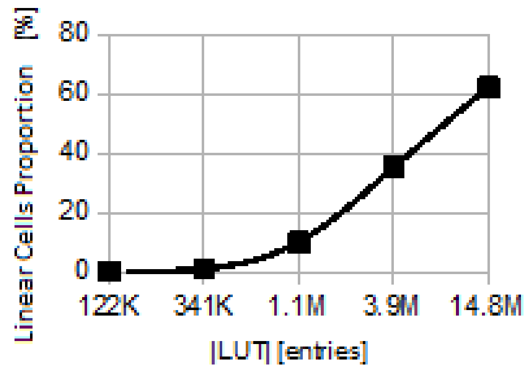


Figure 12: Percentage of cells that fit linear representation for IDS (LCP)

The Linear-Cells Proportion metric (**LCP**) depends on the resolution (Figure 12) but also on the modeled behavior. Some parameter can gain LCP by using a simple variable change. For example, all the first-order textbook equations for I_{DS} include a factor of W/L, which may give the impression that these two input parameters can be removed from the model and then multiply its calculated I_{DS} value . In fact, W and L affect so many other aspects of a transistor behavior that it is impossible to separate W/L and keep good accuracy. However, the quasi-linear scaling of I_{DS} by the W/L ratio suggests that if 1/(W/L) factor is applied to the LUT entries, the cells are left to deal with higher-order phenomena, which were negligible enough to be left out of textbooks and therefore require less resolution. Changing a LUT from I_{DS} to $I_{DS_size}=I_{DS}/(W/L)$ gave us an instant gain of ~20% in LCP, without changing resolution or compromising on accuracy. The same method was used to alleviate interpolation effort from g_m and g_o (a reverse

$R_{o,}$, which gets better LCP). Implementing that change of variables had to be done in C and executed before the LIT entries are calculated. This seems like re-introducing equations to the pure-LUT model, but the speed-up gain can justify this with the following example: Suppose we have a pure LUT of 5 dimensions and LCP of 40%. Linear interpolations cost 5 multiplications, while full interpolations cost 31 (2^N-1). The average cost of interpolation can be calculated using the following equations.

$$Cost_{avg} = (1 - LCP) \times (2^N - 1) + LCP \times N$$

With the chosen parameters discussed above, the average cost is 18. If we count the W/L factor as 2 multiplications (W and L are not constant) and the additional LCP to be 20%, the new average cost would be14.8 which results in a 17.8% saving on interpolation time.

Additional, compromise interpolation modes were considered, which are faster than the full mode, yet are less probable to fail than the strict linear criterion of fitting all 2^N corners close to a hyperplane in the $\mathbb{R}^{N+1}$ space. One option is to “promote” one parameter to full interpolation and keep slopes for all the rest. But the memory requirement for that is too high. Instead of N+1 coefficients for linear interpolation, we now have to allocate N-1 slopes plus intercept for each half of the cell, which is 2(N-1+1) or 2N. Promoting two parameters requires 4(N-2+1) and so on.

5.1.4. Interpolation Procedures

Every model query results in cell interpolation. First, the interpolation procedure determines which cell contains the data for the interpolation. This is done separately on each input parameter (=dimension). There can be two types of parameters: uniformly partitioned and non-uniformly partitioned. If a parameter is characterized with uniform partition, the search is a

simple normalizing of the input according to the base-level value and step (O(1)). Otherwise, a binary search retrieves the right interval in $O(\log(M_i))$ steps. Thus the maximal time-complexity for cell location is $O(\sum\log(M_i))$.

The next step is to determine if there is a LIT entry corresponding to the cell. A cell that failed linearization is flagged with a "not a number" (NaN) intercept. If LIT cell exists, the intercept is copied to the accumulator. The rest of the slope coefficients are simply multiplied-accumulated with the input parameters. The faster linear interpolation is therefore an inner-product between two N-long vectors plus an added intercept, which is O(N).

In case no LIT exists or that a particular cell failed linearization, the full interpolation must be performed. Lagrange-equivalent procedure was chosen that assigns w_i weights to each corner of the cell according to the input parameters relative location within the cell. The weights are simple multiplications of the relative distance of each parameter within the range defined in the cell along a dimension. That relative distance $r_i \in [0,1]$ is given as a byproduct of the cell location stage. A naïve approach to calculating w_i's from r_i's would be to populate a 2^N long buffer with all the products of r_i and $(1-r_i)$ combinations and then calculate an inner-product between those weights and the corner values. This gives $N2^N$ multiplications. The strategy was to populate the buffer with the 2^N corner values. Then, repetitively compress the buffer to half its size (Algorithm 2) using a weighting formula on each pair of corners: $C_{j/2}=(1-r_i)C_j+r_iC_{j+1}=C_j+r_i*(C_{j+1}-C_j)$. Notice that $C_{j/2}$ is free to serve as storage for the next compression and therefore it can be performed in-place. After N compressions, the buffer shrinks to a single entry, which is the interpolated value. The total number of multiplications is 2^N-1.

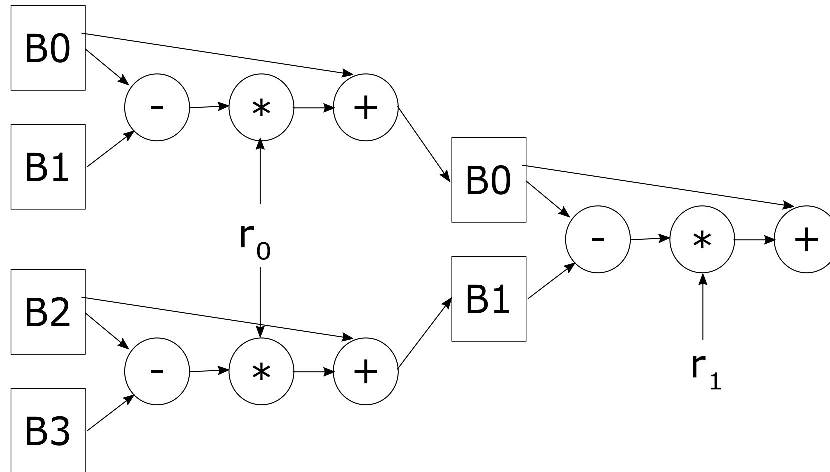


Figure 13: 2D full interpolation using in-place compression

Algorithm 2: Lagrange-equivalent compression loop

```
Begin
     For each corner Ci
     Begin
         Bufferi=Value(Ci)
     End
     Let buffer_limit=2N
     For each relative location ri
     Begin
         For (j=0 ; j<buffer_limit ; j+=2)
         Begin
             Bufferj/2=Bufferj+ri*(Bufferj+1-Bufferj)
         End
         buffer_limit=buffer_limit/2
     End
     Return Buffer0
End
```

5.1.5. Composite Interpolation

For evaluating a circuit's operating point, I_{DS} lookup needs to be more accurate than the rest of the parameters, because slight errors in I_{DS} can translate to significant shifts in V_{DS} and thus nodes' DC potentials. To achieve higher accuracy, a departure from the general-purpose lookup code was needed as an alternative to increasing the tables' resolution. This came in the form of composite lookup algorithm. This method looks up g_m and g_o of a transistor first and then uses their values as additional slope information of I_{DS} along V_{GS} and V_{DS} respectively.

The result is both faster look up, since entry location is done once for all three tables, and more accurate one, since it adds a 2nd order approximation to the original linear one. The interpolation on I_{DS} is replaced by an interpolation on its <u>intercept</u> with the $V_{GS}=V_{DS}=0$ axis. Instead of loading the interpolation buffer with I_{DS} values from the table entry, it is loaded with $I_{DEq,i}=I_{DS,i}-V_{GS,i}*g_{m,i}-V_{DS,i}*g_{o,i}$.

After interpolating I_{DEq} values, the interpolated g_m and g_o from the previous lookups are used together with the input V_{GS} and V_{DS} to go back and calculate I_{DS}, in a more accurate value: $I_{DS}=I_{DEq}+V_{GS}*g_m+V_{DS}*g_o$

Figure 14 shows a segment from the V_{DS}/I_{DS} curve as it is plotted from a SPICE simulation (black) and a Lagrange-interpolated lookup of the Γ model (red). The minuscule

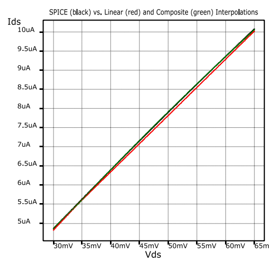


Figure 14: Interpolation vs SPICE

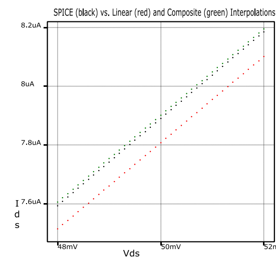


Figure 15: Zoom in on Separation Area

difference in I_{DS} between SPICE and the lookup line is of the 100nA magnitude. However, looking at the lateral error, it shows ~700uV maximal separation between the two. Figure 15 zooms-in on the 50mV region and shows how closer the composite interpolation lookup (green) gets to the SPICE curve (black), in comparison with the linear one (red). V_{DS} error drops from 700uV to under 50uV.

Other algorithms that were considered involved looking beyond the immediate hypercube to neighboring samples and performing higher order interpolations. Yoon and Allen[20] chose quadratic interpolation for their application to gain better accuracy and smoothness for lower resolution. This and other higher order algorithms get these two advantages in exchange for run time, which is one of the important resources. Calculating quadratic interpolation, for instance, requires putting together the Q matrix in run time, or otherwise storing a huge data base of all the possible Q matrices in advance. The choice of staying in first order and finding ways to accelerate those algorithms in exchange for memory and some accuracy was therefore motivated by the need to keep model response time as short as possible.

5.1.6. Hierarchical Interpolation Tree

The hierarchical approach was an intuitive solution to creating a compact look-up database that is also fast to interpolate. This class of solutions partitions the function domain into sections, which get separate analyses and recursive partitioning depending on the local level of complexity. The expectation from a physical function is to require more resolution in some sections than others to achieve the same level of interpolation accuracy. A database that saves space on the less-detailed domain areas and allows shorter interpolation time where certain

inputs can be neglected seems like the right approach in an application that demands growing amount of memory.

The effort of populating such a hierarchy can be high in this application, because the source (SPICE) performs significantly faster in uniform sweeps than in select parameters that need to be determined on the fly. However, once an exhaustive uniform sampling of the domain was performed, a Hierarchical Interpolation Tree (HIT) can be easily constructed in post-processing. A HIT is a binary tree, splitting with each node a selected dimension and domain value. A coordinate can locate a cell by descending down the nodes, each comparing a single dimension to a value and sending the search left or right.

The following types of HIT leaves were considered:

1. Full-cell leaves – storing all 2^N corners of each hypercube in every leaf
2. Partial-cell leaves – initially storing only lower-left corner per leaf.

The benefits of using full-cell leaves are:

1. Locating the leaves (=cells) is fast and corresponds to the depth of the tree, which reflects the complexity of the function per domain section.
2. Every leaf contains all the information needed for interpolation. There is no equivalent stage of fetching hypercube corners from an array because the interpolation buffer is stored as it is needed for the interpolation itself.
3. There can be more types of leaves, compressing information and accelerating interpolation according to opportunities presented by the local samples. Since none of

the samples in a leaf are shared with other leaves, any compression can be decided on and performed locally without destroying data in other cells.

The cell types that were coded are:

LUC – a copy of a LUT hypercube (2^N scalars).

LIC – a copy of LIT cells (N+1 scalars)

cLUC – 8b mapping of the LUC (Scalar for base-level, Scalar for scale, Byte x 2^N corners).

cLIC1-cLIC3 – a short LIC cells, featuring only one, two or three dominant dimensions in a linear interpolation (1 scalar for the intercept and 1 scalar per dimension).

The biggest drawback of using full cells is the data-bloat that starts with 2^N. There are several factors that mitigate this bloat:

1. As in the original intent of using hierarchy, some of the cells from the initial construction can be merged with others and thus eliminated. Once the tree is constructed, a recursive procedure finds adjacent cells that can reproduce their shared corners after merger and replaces their binary node with a single cell.
2. Linear regression and further compression of LIC and LUC cells reduces amount of data even more.

A test Ids LUT was selected to evaluate the memory/time performance of the full HIT option. The initial result was not satisfactory in terms of access time. Following a linked tree, pointer to pointer is time consuming, most likely because of cache misses. However, after coding

the entire tree in a bytecode script, the access time dropped to ~350nsec, which is a 4x factor compared to the original LUT. Memory-wise, the cost was high. Even after eliminating 60% of the cells, linearizing 7.4% and compressing 24%, the overall bloat was almost 7x. The partial HIT option, of keeping only the corner closest to the origin (lower-left in 2D terms) per cell, eliminated the bloat almost completely. However, it had some serious disadvantages:

1. Now that the cells are no longer self-contained, the interpolation procedure needs to fetch every corner separately and the corners are not arranged predictably as in the LUT. This 2^N fetches increased the overall time 5x, instead of reducing it. Even after careful coding of the fetching code and caching points along the tree-descending procedure, the access time was not significantly different than the original LUT one. There was no acceleration.
2. The opportunities for compression and linearization are not available now that every stored sample can be part of up to 2^N cells and therefore cannot be replaced with an 8b representation or by a vector of slopes.
3. Reducing the memory bloat to close to 1x was only going to serve the next phase, of eliminating cells by merger. This step proved too complicated. Merging pairs of cells in the full HIT was simple, because the eliminated points were also stored separately in every cell that still needed them. In contrast, after eliminating cells in the partial HIT, corners that were needed for other cells disappeared. This presented another complication of having to keep more than one value per leaf and adjusting the fetch procedure to deal with that change. The saving on memory therefore was not going to be as high as in the full HIT and the access time would inevitably increase again.

Table 2: HIT Cell Types and their Cost in [B]

	64b	Array	24b	HIT			
Dim	LIT	LUT	LIC	LUC	cLUC	cLIC0	cLIC1
2	24	32	12	16	12	4	8
3	32	64	16	28	16	4	8
4	40	128	16	52	24	4	8
5	48	256[2]	20	100	40	4	8

After converting the models to 4D by eliminating channel-width as a dimension presented the benefit of converting all of the remaining dimensions to be uniformly sampled, the acceleration benefits of the HIT became obsolete. The 4D LUT option accelerated the cell-locating phase and reduced memory volumes. The return on investment of HIT did not justify complicating the code further. However, the code for converting LUT to HIT and using it in interpolation is still present in the system and can be used if determined beneficial per-case.

5.1.7. Resolution Budgeting

This model size is bound by the memory resources of the host machine. After implementing its mechanism, the remaining challenge is balancing the resolution allocated to each parameter within the overall process capacity and accuracy requirements. The first

[2] LUT cell volume is counted in bytes, which is always 2^{DIM}*|scalar|. In case of 5D double scalars, each cell takes up 2^5*8=256B. However, unlike HIT cells, a LUT cell shares its entries with its neighbors. This makes the actual contribution of each cell to the total LUT volume as small as 8B, for inner cells.

indication of adequate resolution was the LCP. The bottom-line indicators are the standard-deviation of the relative error and the max-error of 99% of the samples. The latter indicator gives a better measure of the overall accuracy, because the error distribution is neither normal nor symmetrical.

The first tool used to manage resolution allocation was a table of bits-per-parameter. Uniformly-partitioned parameters are sampled in equidistant M_i+1 values, to form M_i intervals. To simplify initial budgeting, M_i legal values were limited to 2^{bi} , where b_i is the number of resolution-bits given to the i'th input parameter. Neglecting the closing sample for each input, the total volume of the LUT is either $2^{\Sigma bi}$ without LIT or (N+2) $2^{\Sigma bi}$ with LIT. The sum of b_i is therefore a representative of the memory allocated for that view and can be used as a guide for distributing resolution between inputs. Even when M_i was released from having to be powers of 2, the ease of using resolution-bits instead of M_i directly made us go back and translate resolution to effective resolution bits $b_i=\log_2(M_i)$ that could be taken off or added to a budget.

The resolution budgeting procedure was based on a bottom-up search. Low resolution characterization is fast (seconds to minutes), so it started with allocating 2 or 3 resolution bits uniformly to all input parameters and got a baseline read of the quality indicator. Then the procedure gradually added bits to the overall budget, testing their effect when allocated to each input parameter. Some parameters had higher effect on accuracy or linearity when the resolution was low or when their role was saturated, other parameters got the next resolution bits. Note that each added bit doubles the size of the LUT, so the procedure was time-bound by N times the final characterization. Budgeting the resolution of a view was typically finished in less than a day of manual, repetitive characterization runs.

After budgeting uniform resolution, the sampling values of non-uniform inputs (typically W and L) were fine-tuned. To better position samples, relative error per interval were measured and thus got an indication for where an interval should be split up and where a sample is redundant.

5.1.8. LUT Interpolation Implementation

Eight (8) was chosen as a maximal supported dimensionality, but any number could be configured into the compilation script. To improve coding efficiency, a code-generator was added to the compilation script. This code-generator extended the C preprocessor with Tcl-based commands, such as #Foreach and #For loops. That improvement enabled convertion of some of the run-time loops to compilation-time, unrolled ones without sacrificing code readability. The immediate benefit was in automatically generating 8 interpolation functions specifically for 8 table dimensions, each completely unrolled for both linear and full interpolation modes. In early tests of the average interpolation time (~70% linear and 30% full interpolations in a LUT+LIT 6D setup), it was discovered that unrolling saved about 10% of the CPU time, from 2.76µsec to under 2.5µsec. Unrolling is expected to have more significant effect on LUT's than LIT's, because of the nested loops needed by the full interpolation procedure.

Characterization scripts begin with converting foundry-provided technology files from commercial tools' format to NGSPICE .model format. The script flattens all the equations, functions and bins of the foundry format to simple model/parameter listing, one device per bin. These simplified transistors are then included in a test circuit netlist. RAMSpice uses the test circuits in a nested .dc sweeps. The code saves the vector results in a binary format, which is later copied to the LUT array. Some characteristics require post-processing. For instance, g_m

requires running I_{DS} loop once, changing V_{GS} by a small amount, re-running I_{DS} and calculating the derivative, based on the I_{DS} vectors and ΔV_{GS}. Vector to vector subtraction and factoring operations are also built into the RAMSpice C code to allow faster on-the-fly calculations of such views.

5.1.9. 4D Compact Model

Initial models of 6D were accurate and flexible, because they tracked 3 transistor voltages, 2 channel dimensions and the temperature. The typical size of such comprehensive model (in the GB's) required that the models be memory-resident, as in a waiting server process, or lose all run-time benefits by requiring lengthy upload time at the beginning of every circuit evaluation. To enable fast-uploading models, a series of changes were made to scale down model dimensionality:

1. Temperature became part of corner definition – e.g. SS125OC and TT25OC tables – this degeneration is acceptable as long as temperature-sensitivity analysis is not performed.
2. The width dimension was avoided by characterizing only square channels.

Square channels are a specific case where the width is equal to the length of the channel. The width to length ratio (W/L) can be referred to as number of squares in the channel. Square-channel characterization assumes the following:

1. Channel current (I_{DS}) , conductance (g_m, g_o) and parasitic capacitances scale linearly with the number of squares.
2. Linearity is kept accurately for a useful range of squares per channel.

The observed useful range of squares per channel is 10. A total W/L>10 is still possible by using the fingers structure, which is equivalent to connecting a number of transistors in parallel. Square-channel tables are more than a magnitude smaller than 5D ones, but they do come with an accuracy shortfall: since SPICE models come in the shape of binned BSIM parameters, the square-channel loop does not exercise all bins in the W and L matrix, only the ones containing W=L dimensions. This means that for some channels, SPICE will be using a model from a bin that was not part of the characterization loop and an error is thus introduced when Gamma and SPICE are matched. This problem is demonstrated in Figure 16. Each rectangle represents a geometry bin in the channel modeling space. Since only square channels are characterized for 4D, only the bins that are on the diagonal get characterized directly. However, more than one square per channel may be used by a circuit. The problem is highlighted by the checkered bin, which draws its SPICE characterization from a bin that is not exercised in 4D. Instead, the bold line segment provides the LUT with (probably inaccurate) parameters. This problem becomes more acute when more squares per channel are permitted in generating a SPICE netlist. Naturally, when only square channels are permitted in SPICE netlists, leaving fingers to carry all W/L, this problem goes away.

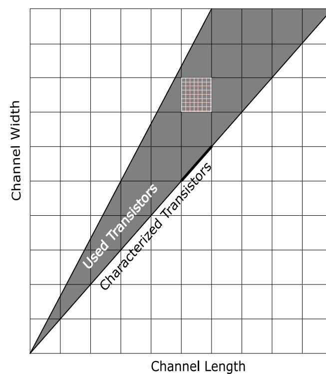


Figure 16: 4D Square Channel Characterization and Binning Mismatch

The final transistor-level models are produced from BSIM with the RAMSpice simulator that populates all the lookup tables in their final resolution (Table 3).

Table 3: Transistor-Level Modeled Parameters

Characterized Parameter	Table Resolution	Approximate Access Time
g_m	2.5MSamples	450nsec
r_o	2.5MSamples	450nsec
I_{DS}	2.5MSamples	1.1µsec (inc. g_m, r_o)
V_T	10KSamples	300nsec
V_A	10KSamples	300nsec
N_T	6.5KSamples	1.4µsec
N_f	6.5KSamples	1.4µsec
C_{GS}	6.5KSamples	1.4µsec
C_{GD}	6.5KSamples	1.4µsec
I_{DSmis}, V_{OS}	257Samples	120nsec

5.2. Circuit Compiler

Γ Circuit Compiler (ΓCC) is the offline tool for generating online Γ engines. It is scripted in Tcl. Topology analysis produces the KCL equations of circuit nodes and derived expression for the output as a small-signal function of all circuit inputs – signal inputs, power supplies and noise sources. However, this is the only resemblance between the proposed system and SPICE. The derived code in this system is precompiled either via an adapted virtual machine compiler for hardware-independence or as generated C code for hardware-efficiency (Figure 17). This too gives the evaluator advantage over SPICE, as the topology is loaded into the system in the form

of pre-processed batch of instructions (be it VM or machine code) that is optimized by a compiler to perform all the needed calculations, from operating point to global performance figures. SPICE is loaded with topologies as netlists, which are analyzed and inefficiently compiled in run time.

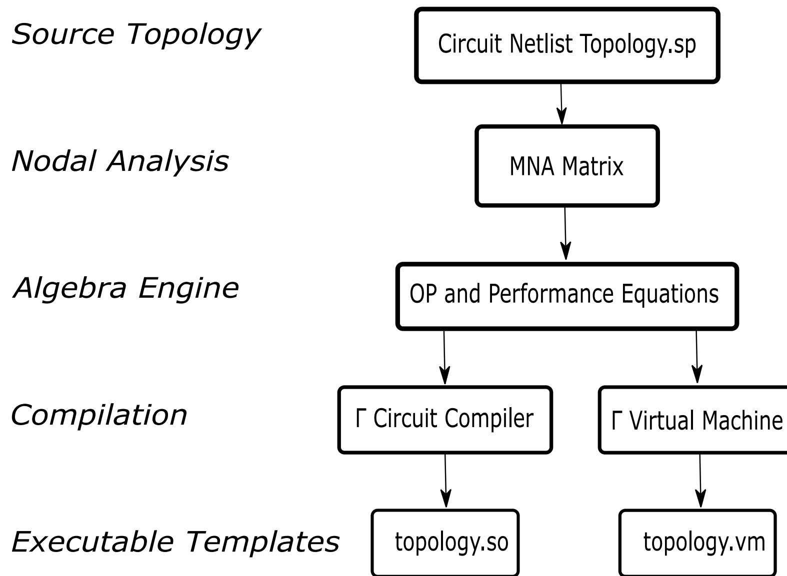


Figure 17: Circuit Compilation Flow Stages

5.3. Linear Model of Transistor

Γ's circuit analysis is based on the small-signal linear model. Each MOSFET is replaced with a small network of linear components that are sized to emulate the transistor behavior around an operating point.

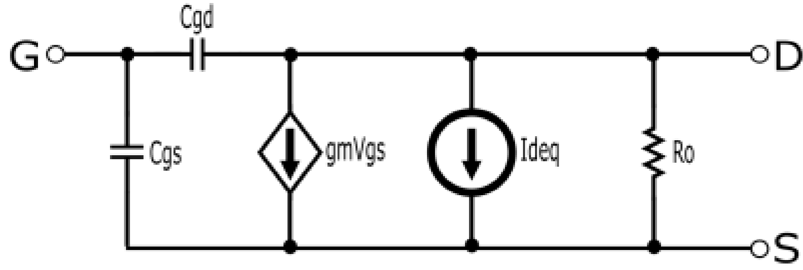


Figure 18: Linear Model of N-type MOSFET

Figure 18 shows the main components in a linear sub-circuit that replaces an N-type MOSFET, without body effect components. All of the components are sized based on the transistor-level set of models, expressed in LUT's. The model is used for establishing equilibrium operating point in the circuit-level analysis as well as calculating performance properties through subsequent equations. Some components require additional calculations after lookup:

$$g_o = {}^{1}\!/_{R_O}$$

$$I_{Deq} = I_{DS} - g_m V_{GS} - g_o V_{DS}$$

I_{Deq} is the size of the independent current source in the linear model. It models the intercept of the tangent plane of the $I_{DS}(V_{GS},V_{DS})$ surface at the $V_{GS},V_{DS}=0$ point, while g_m and g_o represent the slopes (Figure 19).

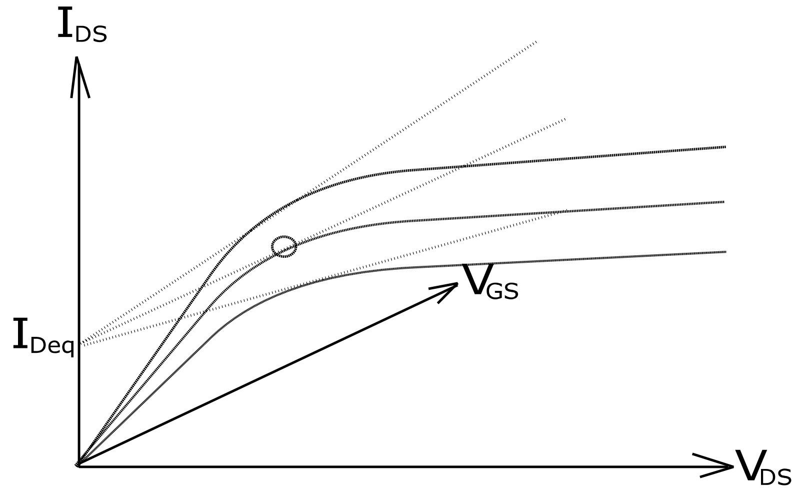


Figure 19: I_{Deq} is the intercept of the $I_{DS}(V_{GS},V_{DS})$ plane with the I_{DS} axis

The small-signal planar representation of I_{DS} is therefore:

$$I_{DS\,(small\ signal)}(V_{GS},V_{DS}) = I_{Deq} + g_m V_{GS} + g_o V_{DS}$$

Capacitive components are read directly from their LUT's. They have no role in calculating the operating point, so they are not included in the initial DC analysis. In AC analysis, their susceptance is included in the Laplace form: sC.

Noise sources (not shown) are considered as parallel current sources to I_{Deq}. Their added contribution is analyzed as statistically-independent between frequencies and therefore all calculation, from LUT query to total-noise, is done in the squared units per frequency: [A^2/Hz] and [V^2/Hz].

5.3.1. Nodal Analysis

Modified Nodal Analysis (MNA) is a matrix-based circuit solution algorithm. Its starting point is the Kirchhoff Current Law (KCL) equation: GV=I , where G is a matrix of admittance

values, V is the vector of voltages at the various nodes and I is a vector of independent current sources. The matrix used by MNA is an extension of the KCL matrix, including independent voltage sources. The method simplifies one challenge of the original KCL method: nodes that connect to others via voltage sources. The original method required detection of such nodes and special work-around of pre-merging and post-splitting the nodes. MNA matrix can therefore be larger than KCL's, but its generation and solution are more straight-forward.

The method of generating an MNA matrix for a given circuit is essentially identical to that of SPICE. The main differences are:

1. Γ's MNA matrix is generated off-line, as part of the circuit compilation phase, while SPICE generates the matrix during simulation.
2. The solution of the matrix in Γ generates an algebraic representation of the dependence of each node on input signals, components' admittance and modeled current sources (symbolic analysis), while SPICE solves the matrix numerically (matrix loading).
3. By the time Γ solves the circuit's nodes, there is no trace of the original matrix and in some cases the generated solution equations go through further analysis, so they do not resemble the MNA representation. SPICE relies on the matrix in every step of the simulation and has no equation to process.

Γ Circuit Compiler (ΓCC) contains a netlist to MNA matrix converter. The input is a SPICE-like netlist, given with variable size parameters and sizing constraints (see topology example in Figure 20).

Three MNA matrices are produced automatically and printed for inspection in HTML Figure 21:

1. DC operating point and small signal matrix
2. R_{OUT} test matrix
3. AC transfer function matrix

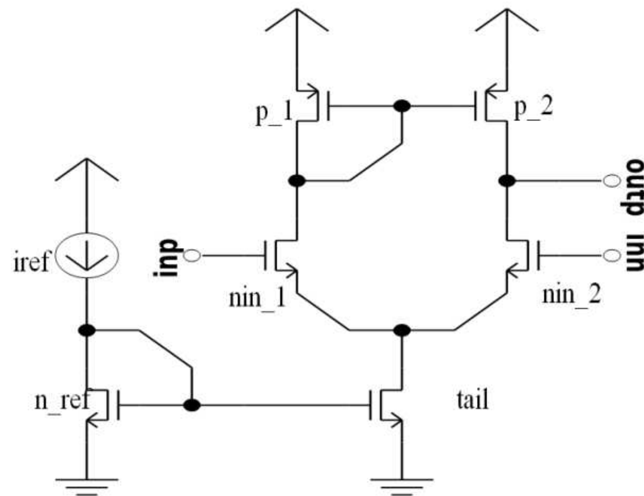


Figure 20: Example Topology for Nodal Analysis

The process of generating an MNA matrix relies on one atomic operation of connecting single admittance component between two nodes. Single admittance component can be either conductance (g components) or susceptance (sC components) for AC matrices. When an admittance is added between nodes *i* and *j*, it is added with addition operator to the $M_{i,i}$ and $M_{j,j}$ entries and with subtraction operator to $M_{i,j}$ and $M_{j,i}$. The I vector is listing the independent current sources in the circuit, where for each entry (node) the in-flowing current sources are added with positive sign and out-flowing ones in negative.

0	0	0	0	0	0	0	1	0	0	inn	0
0	0	0	0	0	0	0	0	0	1	inp	0
0	nin_1:gm	p_1:go +p_1:gm +nin_1:go	0	-nin_1:go -nin_1:gm	0	-p_1:go -p_1:gm	0	0	0	outm	-nin_1:Id -p_1:Id
nin_2:gm	0	p_2:gm	Gds_outp_ 0+p_2:go +nin_2:go	-nin_2:go -nin_2:gm	0	-p_2:go -p_2:gm	0	0	0	outp	-nin_2:Id -p_2:Id
-nin_2:gm	-nin_1:gm	-nin_1:go	-nin_2:go	nin_1:go +nin_1:gm +nin_2:go +nin_2:gm +n_tail:g o	n_tail:gm	0	0	0	0	tail	-tail:Id +nin_2:Id +nin_1:Id
0	0	0	0	0	n_ref:go +n_ref:gm	0	0	0	0	vbias	-n_ref:Id +iref
0	0	-p_1:go -p_1:gm -p_2:gm	-p_2:go	0	0	p_1:go +p_1:gm +p_2:go +p_2:gm	0	1	0	vdd	p_2:Id +p_1:Id -iref
1	0	0	0	0	0	0	0	0	0		inn
0	0	0	0	0	0	1	0	0	0		vdd
0	1	0	0	0	0	0	0	0	0		inp

Figure 21: MNA matrix generated for the compiled topology in Figure 20

5.3.2. Algebra Engine

Γ circuit-compiler solves matrices symbolically, i.e. they each produce an algebraic equation for the output and other nodes' voltage. The solution is achieved via Cramer's rule, which states that the solution to Ax=y can be achieved by calculating the determinant of A (|A|) and a series of determinants of modified A's, where column i is replaced with the vector y (A_i). Cramer's rule states that solution of $x_i=|A_i|/|A|$. This means that each node voltage function of circuit admittances, modeled current sources and input signals, can be obtained by first calculating a global entity $DET(g_{m's},g_{o's})=|MNA|$, its inverse value $TED(g_{m's},g_{o's})=1/DET$ and then obtain the formulas $V_i(g_{m's},g_{o's},I_{eq's},V_{in's},I_{in's})=TED*|MNA_i|$.

The majority of algebra operations are done in Polish Notation (PN). This allows efficient "divide and conquer" methods to be applied to large equations:

1. Partial derivatives – some performance properties' equations are generated by deriving the output's symbolic solution with respect to inputs. Since the MNA matrix itself contains only constants and admittance variables, the main determinant and its inverse TED are independent of input signals, bias, and current sources. This means that deriving the fraction $\partial(|A_{output}|/|A|)/\partial X$ can be done on the output's determinant only: $\partial(|A_{output}|/|A|)/\partial X=\partial(|A_{output}|\cdot TED)/\partial X=TED\cdot(\partial|A_{output}|/\partial X)$. The expression obtained from $|A_{output}|$ contains only additions, subtractions and multiplications, which have simple derivation rules (Algorithm 3).

Algorithm 3: Recursive Derivation of PN Expression

```
Derive(pn_expression,WRT) Begin
    If(pn_expression==WRT) return 1
    If (is_leaf(pn_expression)) return 0
    // This is a node. Get the operator, two arguments
    [operator A B]=pn_expression
    // and call the recursion
    dA=Derive(A,WRT)
    dB=Derive(B,WRT)
    If (operator==”*”) return [+ [* A dB] [* dA B]]
    If (operator==”+”) return [+ dA dB]
    If (operator==”-“) return [- dA dB]
End
```

2. Laplace to complex Fourier conversion – AC expressions obtained from the MNA(s) matrix are expressed on the Laplace domain s. To search for key points on the circuit spectrum, s has to be converted to iω and the expressions need to be split to real and imaginary parts. This is done by recursively applying the simple addition and multiplication rules of complex equations (Algorithm 4).

Algorithm 4: Recursive Conversion of PN Laplace Expression to Complex Fourier

```
S_to_iw(pn_expression) Begin
    If (pn_expression==”s”) return (0,ω)
    If (is_leaf(pn_expression)) return (pn_expression,0)
    // This is a node. Get the operator, two arguments
    [operator A B]=pn_expression
```

```
    // and call the recursion
    (Ar,Ai)=S_to_iw(A)
    (Br,Bi)=S_to_iw(B)
    If (operator==*)
        return ([- [* Ar Br] [* Ai Bi]] , [+ [* Ar Bi] [Ai Br]])
    If (operator==+) return ([+ Ar Br] , [+ Ai Bi])
    If (operator==-) return ([- Ar Br] , [- Ai Bi])
End
```

3. Expression simplification – Algebraic operations often leave a lot of redundant terms that can be pruned to prevent generating dead code. Simplification of PN equations is done with recursively applying trivial arithmetic rules (Algorithm 5). When an operation is commutative (+ or * operation) and the algorithm decides not to prune it, it is returned in lexicographical order, to increase the chance of equivalent expressions canceling each other in subtraction (e.g. (a+b*c+d)-(a+c*b+d) can be detected to be 0, although (a+b*c+d)-(d+a+b*c) is beyond the reach of single-node lexicographical sorting).

Algorithm 5: Recursive Simplification of PN Equation

```
Simplify(pn_expression) Begin
    If (is_leaf(pn_expression)) return pn_expression
    // This is a node. Get the operator, two arguments
    [operator A B]=pn_expression
    // and call the recursion
    As=Simplify(A)
    Bs= Simplify(B)
    // List of trivial cases:
```

```
    If (operator==”*” && (As==0 || Bs==0)) return 0
    If (operator==”*” && As==1) return Bs
    If (operator==”*” && Bs==1) return As
    If (operator==”+” && As==0) return Bs
    If (operator==”+” && Bs==0) return As
    If (operator==”-“ && Bs==0) return As
    If (operator==”-“ && Bs==As) return 0
    // return commutative expression in lexicographical
    // order to increase chance of catching the last case (X-X=0)
    If (operator!=”-“ && As>Bs) (As,Bs)=(Bs,As)
    // No trivial case caught, return the term as-is
    return [operator As Bs]
End
```

4. Back-conversion of PN expressions to infix notation – This algorithm (Algorithm 6) looks into the leading operators of the sub-terms, because infix notation relies on parentheses to regulate operator order. While encapsulating all sub-terms would be arithmetically correct, the generated code could be long to compile and hard to understand by human readers.

Algorithm 6: Recursively Converting PN to Infix Equations

```
PN_to_infix(pn_expression) Begin
    If (is_leaf(pn_expression)) return pn_expression
    [operator A B]=pn_expression
    // and call the recursion
    Aif= PN_to_infix(A)
    Bif= PN_to_infix(B)
```

```
    // catch cases that require parentheses
    If (operator==”*” && operator(Aif)==”+” && operator(Bif)==”+)
        return “(Aif)*(Bif)”
…
    // no need for parentheses, return the simple infix notation
    return Aif+operator+Bif
End
```

5.3.3. Generated Equations

5.3.3.1. Equilibrium Operating Point Voltages (OP)

The first stage in evaluating a circuit is a successive approximation of voltage nodes to the equilibrium point. In equilibrium, each node is KCL compliant: ΣI=0. Alternaively, the sum of in-flowing currents equals to the out-flowing ones. When nodes are not in equilibrium, the residual current indicates if the node’s voltage is below or above the equilibrium point and a step toward equilibrium can be calculated. The step is obtained by Millman’s theorem[48], which is equivalent to finding V(I) slope in the Newton-Raphson method used by SPICE:

$$\Delta V = \frac{\sum I_i}{\sum g_i}$$

is the sum of all admittance (=conductance in DC) that are connected to the node. This sum is already available in the diagonal entry of the MNA matrix at the row+column that correspond to the node. Since transistors are nonlinear in their behavior, those equations are put inside a loop for convergence, with a stepping factor that lets the loop converge without overshooting the 0-VDD rails.

5.3.3.2. Low Frequency Performance: Gain and Rejection Ratios

Based on partial derivatives of the output w.r.t. the inputs, common-mode and supply, the algebra engine can generate equations for the gain, common-mode rejection ratio and power-source rejection ration, respectively. In a single-source amplifier: $A_{DC}=\partial V_{out}/\partial V_{in}$, which translates to $A_{DC}=TED*\partial|MNA_{Vout}|/\partial V_{in}$ (since TED is independent of current and voltage sources, it can be regarded as constant w.r.t. this derivation). In a dual-source amplifier, V_{in+} and V_{in-} are first substituted with V_{in} and $-V_{in}$ respectively. The above derivative applies with a factor of ½, because V_{in} is effectively doubled when applied to both inputs simultaneously $A_{DC}=0.5*\partial V_{out}/\partial V_{in}= 0.5*TED*\partial|MNA_{Vout}|/\partial V_{in}$ with $V_{in+}=V_{in}$ & $V_{in-}=-V_{in}$. The common-mode rejection is calculated by substituting both V_{in+} and V_{in-} with V_{in}. The gain derivative is not halved, because V_{in} is the full input signal. The ratio is therefore: $CMRR=A_{DC}/(TED*\partial|MNA_{Vout}|/\partial V_{in})$ with $V_{in+}= V_{in-}=V_{in}$.

The power supply rejection is calculated the same way for any circuit. It is the partial derivative of the output node w.r.t. V_{DD} node: $PSRR= A_{DC}/(TED*\partial|MNA_{Vout}|/\partial V_{DD})$. Gains and rejection ratios are normally expressed in dB, which means they are to be converted to logarithmic scale before being registered in the circuit's performance vector.

Algorithm 7: Low-Frequency Performance in Pseudo-Code Equations

```
Begin
  A_DC=20*log10(0.5*TED*DER(DET(V_out),V_in))
  CMRR=A_DC-20*log10(TED*DER(DET(V_out),V_in))
  PSRR=A_DC-20*log10(TED*DER(DET(V_out),V_DD))
End
```

5.3.3.3. Output Resistance: Rout

R_{out} is extracted from an equation generated from a special MNA matrix that contains no voltage or current sources. The R_{out} method connects a dummy 1A current source to the output node, omits all other current sources and zeros all voltage sources. The matrix, with its already calculated admittances behaves like a linear resistor network with its output node's voltage solution in volts matching numerically to the output resistance in Ω's. Note that there is no re-calculation of operating point for this stage, so there are no illinearity effects stemming from the unrealistic current source. The transistors are already replaced with equivalent linear and ideal resistors, so there is no "danger" of a transistor railing or being in an uncharacterized operating point.

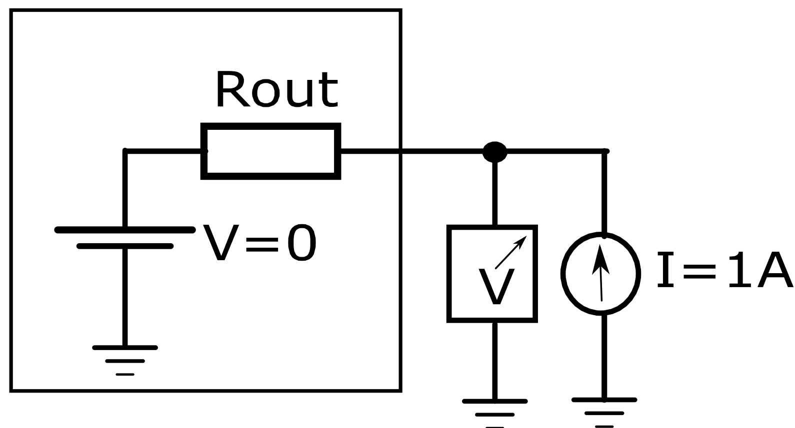


Figure 22: Rout Equivalent Circuit

Figure 22 shows the equivalent circuit. Once all transistors are replaced with linear resistors and their modeled current sources removed, the circuit behaves like a resistor network.

With voltage sources zeroed, the voltage at the output node is numerically the same as the equivalent output resistance.

5.3.3.4. AC: Bandwidth, Settling Time, and Phase Margin

Bandwidth, settling time and phase-margin are extracted from analysis of the transfer function, which is generated from the AC matrix. The Laplace representation of the output node goes through a special function s_to_iw (see Appendix A for implementation), which converts the Laplace function to real and imaginary parts of a Fourier function, depending on ω instead of s. When ω is 0 (DC point), the real part is equal to the operating point voltage and the imaginary part is zero. As ω grows, the real part decreases and the imaginary one increases. As ω grows further, the magnitude and phase of the output (now complex) changes and goes through points of interests:

1. 3dB drop frequency, or ½ of the square-magnitude of DC – a definition for bandwidth.
2. 0dB frequency –unit-gain frequency, where phase margin is calculated.

Algorithm 8: AC Performance in Pseudo-Code Equations

```
Begin
  A(ω)=sToiω(DER(DET(Vout(s)),Vin))/sToiω(DET(s))
  BW=ω when A(ω)=0.5*A(0)
  PM=180^0-tan^-1(imag(A(ω_0dB))/real(A(ω_0dB))) with A(ω_0dB)=1
  Cin=CGD,in*(gm,in/∑go+1)+CGS
 End
```

5.3.3.5. Noise Parameters and Corner Frequency

Thermal noise N_t is looked up as the noise floor and flicker noise N_f is looked up in the 1Hz frequency. The transistor-level spot noise in any arbitrary frequency is:

$$N_i(f) = N_{f,i}/f + N_{t,i}$$

Since noise in the transistor-level is modeled as added current source between a transistor's D and S terminals, the transmission factor of each transistor's noise to the combined circuit noise is calculated by deriving the output's DC equation with the transistor's I_{Deq} and dividing that number with A_{DC} (to infer input-equivalent):

$$Fnoise_i = \frac{1}{A_{DC}} \cdot \frac{\partial V_{OUT}}{\partial I_{Deq,i}}$$

Each transistor's looked-up flicker and thermal noise are multiplied by this factor (squared) and the combined contributions are summed up to a single input-inferred noise spectrum:

$$N_{f(1Hz)} = \sum_{i=1}^{N} Fnoise_i^{\,2} \cdot N_{f,i} \quad and \quad N_t = \sum_{i=1}^{N} Fnoise_i^{\,2} \cdot N_{t,i}$$

The evaluator is capable of producing the breakdown of noise contribution per transistor, although those are not logged in the performance vector, but can be re-evaluated on request.

Total noise is calculated from integrating the $N_f(f)=N_f/f$ function and N_t noise-floor in frequency, up to 2BW. Corner frequency is found from the relation $N_f/f_c=N_t$ to be $f_c= N_f/N_t$

5.3.3.6. Additional Performance Parameters

In addition to MNA-derived equations, other estimations are added to the parameters set:

1. V_{OS} = estimated with parametrized equations based on foundry statistics
2. Power – estimated as $\Sigma I_{DS} V_{DS}$ across all transistors.
3. Area – estimated based on gates' lengths and widths and additional parameters taken from the foundry design rules.

5.3.4. Internal Circuit Dependencies and Design-Feedback

The circuit evaluator is designed to process all sizing dimensions into performance metrics. However, for larger circuits, some of the input sizes cannot be left up to a random search, because they are meant to serve a specific performance target that is normally controlled manually.

For example: an op-amp has a bias voltage that is meant to keep the DC level of the output at $\frac{1}{2}V_{DD}$. Left to random search and filtering, this biasing voltage complicates the design optimization and adds a redundant dimension to an otherwise simple evaluation. To overcome this challenge, the evaluator code has the option to add a feedback stepping to the loop that converge V_{OUT} to $\frac{1}{2}V_{DD}$, during the OP convergence steps. When the circuit's KCL equations are close to be met (e.g. the maximal excessive current flowing into a node is under 0.1uA) this tuning code kicks in and steps the bias voltage up or down to get the output closer to the requirement. The OP loop is designed not to stop until both KCL and auto-tune conditions are met, unless the number of iterations so far suggests none-convergence. The designer need not

worry about this bias voltage being correctly set, the output being any other DC voltage (non-converging circuits will not register in PAT) and the circuit's design dimensionality stays in check.

5.3.5. Code Generator

The algebraic representation of the topology can be converted to programs using one of the two compilers:

1. ΓVM – "Gamma Virtual Machine" an interpreter tailored for the Γ system, with flexibility and hardware independence for cross platform execution.
2. ΓCC – "Gamma Circuit Compiler", uses generic GCC to produce .so (alternatively, .dll on Windows) file.

The virtual machine is a fast bytecode interpreter that mixes stack-machine and custom data structures, such as polynomials and rational functions. It has instructions for accessing Γ-specific data structures, such as LUT queries and PAT insertions. The main motivation behind that option is uncertainty about the target machine that runs the evaluator itself. It is possible that topology preparation may be done by a 3rd party organization, such as IP company, who would like to keep the internals of the circuit to itself, but still want the template produced to be available to run on a public Γ website. Using the virtual machine option, the evaluator is still running in reasonable speed and independently from the equipment that created it. ΓCC is the faster choice and the one recommended for united site operation and topology preparation model.

In the ΓVM flow, the equations go through standardized representation, where redundant operations are eliminated and some common subexpression elimination takes place.

The final stage of compilation creates a shared-object module from C code (or the equivalent ΓVM assembly). The API it provides Γ is common to all topologies. However, initial values and sizing/performance vectors are topology-dependent. The compiler prepares all the collateral database binaries that go with the compiled shared object. These two binaries, together with the topology netlist and schematic representation, form a topology/technology template that can be used for the next step – initial mapping of the Pareto front of the topology's performance envelope.

5.3.5.1. *c Templates

The *c command is an alternative to the Tcl-preprocessor extension. It is intended for parts of code where Tcl dominates over plain C and therefore plain C with Tcl preprocessing directives could become unreadable. Each of the functions in the final ΓCE shared object is defined by a *c template. Since the templates are essentially Tcl scripts (see B.B.6), they can call abstract algebraic operators, such as determinant and partial derivatives and thus use MNA and algebra engine resources. *c statements insert C line of code into the shared object's source. Unlike ordinary C, the lines of code can include references to cTree context, by using at (@) and a context. The intuitive context separator used everywhere else in this system is forward-slash (/). However, this separator is problematic in equations processed by the algebra engine, because it can be confused with the division operator. Therefore, for ΓCC applications only, a colon separator (:) is used instead of forward-slash.

*c is a Tcl procedure that accepts a single line of code. In accordance with Tcl grammar, using quoted lines allows integration of Tcl variables and embedded code in the *c calls.

Example:

```
foreach node $::independent_nodes {
    *c "@$node:V=[DETy $MNA $node]/[DET $MNA];"
}
```

This for-each loop generates equations for each node's voltage level. References to Tcl variables by dollar signs ($) and embedded determinants in square brackets are substituted by the Tcl interpreter before the call to *c. The cTree reference by at (@) is substituted later by the C code generator.

5.3.5.2. ΓCE Shared Object Structure

The generated shared object is a library of compiled Tcl commands. It comes with all the circuit-specific functions that are available both for C and Tcl calls. The shared object includes a batch of global variables that are used within its functions. Γ uses cTree as its main database. For cache considerations, all the cTree references in the *c templates are converted to a shadowing batch of variables. This means that before the first call to any ΓCE function from a Tcl script, all the cTree references need to be imported to the buffer. Similarly, before any reading of a ΓCE accessed variable in a Tcl script, all buffer variables need to be exported back to cTree.

The shared object contains the following functions:

1. `::C::import` – used to prime ΓCE with cTree values
2. `::C::export` – used before results can be read from cTree
3. `::C::op` – evaluate circuit's operating point and performance and send to PAT

4. `::C::random` – generate a set of random circuits and send them to `::C::op`
5. `::C::breed` – generate a set of random variations on existing PAT circuits and send them to `::C::op`
6. `Gamma_Init` (not exposed as Tcl command) – called automatically upon `load`, it links all pointers from the ΓCE buffer variables to their corresponding cTree contexts and registers the above functions in the Tcl interpreter.

5.3.5.3. C Compilation

The final stage of the circuit compiler is calling the C compiler in an external process (`exec`). The target binary is a shared-object (similar to DLL in PC Window systems). It is a compact binary file that is to be loaded during run-time to Γ before any evaluation begins. Loading of the shared object is done via the Tcl interpreter's `load` command.

5.3.5.4. Manual Code Edits

The evaluator is compiled automatically per topology. However, the compilation flow leaves room for user manual intervention to enable some obvious code improvements that are difficult to automate, such as:

1. Identifying static nodes that can be eliminated from the OP loop – supply and bias voltage nodes can be easily identified in the code. The stepping equation always equals 0 and the calculation can be omitted.
2. Separating nodes from independent sub-circuits to preliminary OP loops – the automatically-generated OP loop bundles all nodes and converges them together in a

single stepping batch. However, the OP loop's dimensionality can be reduced by partitioning the nodes to separate loops corresponding to independent sub circuits, such as bias diodes and input levels.

3. Merging symmetric nodes into a single OP stepping calculation – support for symmetry in a circuit exists in the form of shared sizing parameters. However, there is no automatic algorithm in place to detect sets of nodes that follow the same DC voltage. Manual inspection of the code can reveal such calculation redundancies and eliminate stepping equations and even transistor look-ups. For example: a differential pair's OP can be solved by converging on only one of the branches and doubling the current passed to the tail transistor. This simple improvement saves on node and two transistor lookups from the OP loop.

While manual optimization of the compiled topology code can accelerate the OP loop, they are not necessary to achieve full functionality of Γ. The automatically-generated code is ten times faster than SPICE simulation, before any manual intervention (see 6.4).

5.4. Γ Online Engines

5.4.1. Γ Circuit Evaluator

Circuit evaluation is the operation of calculating a vector of a circuit's performance properties from a vector of transistor sizes and other design parameters. This is where Γ replaces the traditional role reserved to SPICE in simulating circuits' performance aspects. Unlike SPICE, the evaluating code is not generalized, but rather generated per topology in an offline compilation (see 5.2). Topology analysis is done similarly to SPICE, using the Modified Nodal

Analysis (MNA). However, Γ's offline analysis is done symbolically, producing algebraic representation of the nodes' equations, rather than numerical values. The offline process ends with a C code that is specific to a given topology. The code contains the explicit formulas for calculating:

1. The distance and step to an operating point equilibrium
2. Constraint-derived dependencies between circuit parameters
3. Performance properties needed for comparing a circuit of that topology against the specifications or generating a pixel in a map. Those functions are derived from the MNA-generated equations using abstract templates in a meta-C language.
4. Classification of circuits according to viability and usefulness.
5. Other circuit-specific functions that generate circuits for mapping the technology's performance limits in a Pareto front.

The C code is sent to the GNU C Compiler (gcc) to compile into a shared object, a binary module that can be loaded into a running Γ process. The circuit-level module is used in both offline and online processes. In the offline process it is used to generate an initial set of circuits that can be used later for generating a first response to user's queries. The same module is loaded when Γ is called by an online transaction to generate a graph of heat map representing a sweep analysis around a pivot circuit.

The evaluator is not fully pre-coded. Instead, the system contains a compiler that generates code per topology. This gives Γ its unique efficiency in producing a rapid performance evaluation per each point in the sizing space. Just like SPICE, the evaluator has to access

transistor-level physical models. These are kept as look-up tables for even more rapid execution time.

Γ avoids algebraic analysis of topologies in real time by employing the circuit compiler. Therefore, every operation is accounted for and directed at the bottom-line goal of producing a vector of performance figures, which can be inserted to an archive of sized.

Figure 23 shows the dataflow in Γ's main engine ΓCE. The evaluator includes the following stages:

1. Operating-point loop – assigns voltages to all independent nodes, calculates transistors' behavior (I_{DS} and admittances) and goes back to calculate voltages out of those, until convergence. The loop's starting point may be an MNA-derived rough estimate of nodes' voltages or previous OP calculated for a similarly sized circuit.
2. User-defined dependences of design sizes are automatically tuned during this loop. Performance property calculations – from simple equations for DC properties to Newton-Raphson loop for pole/zero analysis, these equations populate the output vector. Some additional table lookups are needed here for specific transistor physical values, e.g. thermal noise.
3. PAT access – trying to add the circuit to the general inventory, in accordance with Pareto domination rules. The evaluator routine has several exit points.

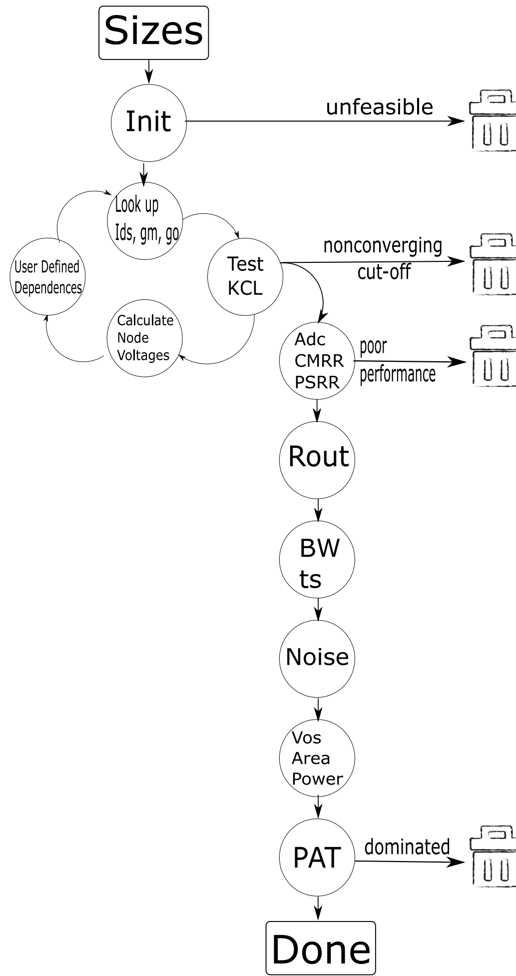


Figure 23: Γ Circuit Evaluator Data Flow

Since it is the system's gateway into the PAT, it includes conditions essential for protecting the database from accepting garbage circuits. Such circuits can be of any of the following categories:

1. Geometrically or electrically violating circuits – based on manufacturing technology rules and constraints provided with the topology. Such circuit should not be presented to the evaluator to begin with, but may be a result of random search and should be rejected before OP evaluation begins.

2. Nonconverging OP – if after so many iterations the OP phase did not reach satisfying KCL thresholds, it gives up and the routine abandons the rest of the calculations. This status is likely to be a symptom of a bad condition that can be discovered downstream.
3. Containing non-saturated or cut-off transistors – most filtered by condition 2, these are not expected to yield good performance and therefore discarded.
4. Extremely poor performance – e.g. negative dB (<1) gain at DC is not useful for amplifiers in any case and such circuit should not continue to be evaluated on other performance aspects.

Only if a circuit survives all early exit points it is introduced to the PAT, which in turn determines its position regarding the topology's Pareto front. The routine informs calling algorithm about the classification of the circuit it provided.

5.4.2. Random Circuit Engine

This is the simplest engine generated by the circuit compiler. Its role is to create the seed to the offline PAT populating process. The engine is compiled with the parameter ranges given in the topology netlist and uses the C built-in uniform distribution randomizer.

5.5. Circuits Pareto Front

Γ does not begin each specification query from scratch. To save time, a spec query can leverage on existing set of pre-evaluated circuits. Although this set of circuits cannot contain an instance for every possible spec, it spans the range of performance properties and provides a selection of approximations to optimum for each query. The set of circuits needs to be managed

to focus on instances that are useful, that are not redundant by using only partial capability of the technology and diverse enough to offer real alternatives to a solution.

The first requirement, usefulness, is simply enforced by determining the lower bounds of performance values. An amplifier with very low gain, for instance, should not take up storage space, because it will never apply to any user requirements.

Redundancy is avoided by complying with Pareto's rules of dominance. Pareto sets, as defined by the economical work of Vilfredo Pareto [43]; do not contain two items where one is better than the other in every aspect. In the case of circuits and performance, if a circuit A is introduced to the set and there is already a circuit B, which is better than A in every performance level and implementation cost, circuit A is considered dominated by B and thus rejected from the set. In case circuit A is better than B in every aspect, A dominates B and thus retained while B is deleted. If there are no circuits that dominate or being dominated by A, the set accepts A. The stream of introduced circuits develops the set by converging it to the limits of performance enabled by the given technology. If a circuit is far from that limit, a dominating circuit will eventually arrive and throw it out. The final set forms a front, where no significant improvement in one performance property can be achieved without sacrificing another.

Diversity is achieved by scanning the set and removing too-similar circuits. Certain level of similarity is permitted for some populating algorithms that use member circuits as basis for generating others. After the set reaches milestone sizes, it is culled to ensure diversity.

5.5.1. Circuit Inventory Container and Manager - PAT

Pareto fronts of various topologies are kept in the Pareto Associative Table (**PAT**). It is a vector structure, designed to keep the front up to date with every additional circuit. Every inserted circuit is checked against the previously accepted ones in the PAT main vector, to see if it is either dominated by any one of them or dominates some of them. Insertion can therefore result in one of the following

1. The circuit is dominated and therefore rejected
2. The circuit dominates other circuits, so it is accepted and the other circuits are deleted
3. No dominance found and the circuit is accepted

Other methods coded for that structure include:

1. Circuits from spec – preliminary selection of relevant subset of circuits and then extracting specific Pareto front for a given spec are the steps in associative retrieval of circuits from performance spec.
2. Automated culling of entries to make sure they are unique and diverse.

These methods were coded in C to ensure fast update and queries of the database.

The PAT plays a role in both offline and online operations. In the offline phase, it is pre-populated with a set of sized circuits that can be used as a starting point for charting tradeoff curves and jump-start optimization algorithms.

5.5.2. PAT Entry

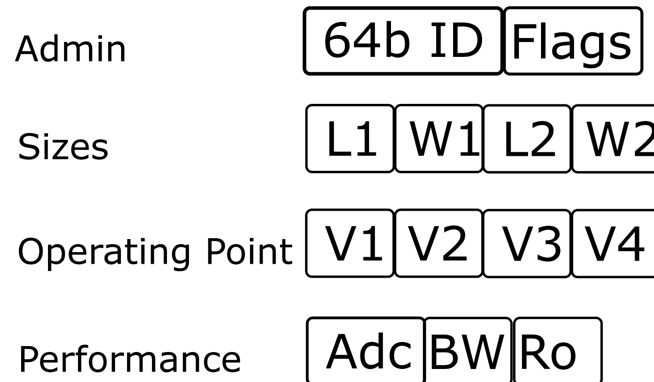


Figure 24: PAT Entry Structure

Each entry in the PAT (Figure 24), is made of 4 parts:

1. The administrative part contains a unique 64b identification number and scratch flags for marking dominated circuits.
2. The 2nd vector contains design choices in the circuit. Those are mostly geometric parameters, but can also be used for reference current, bias voltage and loads.
3. Operating point voltages are kept in the 3rd vector. Although they can be re-evaluated from the Design vector, they are kept to save on execution time in later operations.
4. The 4th vector contains all the performance properties associated with the circuit. This is the part that participates in Pareto front domination evaluation. To keep the code simple, properties that are considered negative (less is better) are stored with a negative sign. e.g. Gain = 40dB, BW = 100MHz, Area = -10(um)2

The PAT, as well as the LUT, is part of Γ's hierarchical database. This database is stored as raw binary sequence on file, which together with the shared-object output of the circuit compiler forms the topology template.

5.5.3. Offline PAT Populating

Once the topology code is created, Γ uses it to populate a PAT with a general Pareto front, which is made of a sample of the topology's sizing space. The goal of this stage is to find circuits that perform at the limits of the given topology, which makes them markers of the Pareto front of size vs. performance. This flow is has two stages (Figure 25):

At first stage – seeding, the PAT populating loop draws random sizers, based on specified min, max and distribution and creates random circuits. The circuits are then tested for viability. A viable circuit is one that has saturated transistors and minimal performance values. An amplifier that has negative dB DC gain, for instance, is rejected at this point. Viable circuits are then inserted into the PAT. As mentioned above, some circuits are rejected, some retained and some retained while knocking out others.

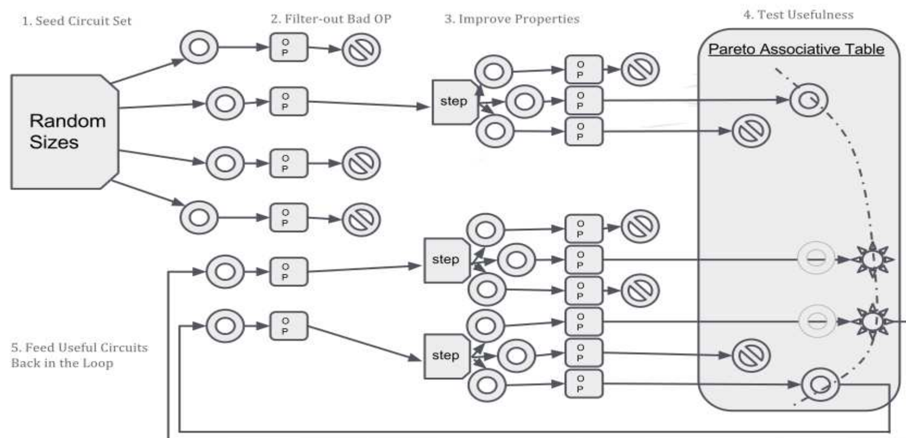


Figure 25: PAT Seed and Breed Phases

The retention rate of the PAT is monitored to make sure it drops in time (Figure 26), which is an indication that the front is saturated and thus the probability of randomly producing a circuit that is not dominated becomes smaller and smaller.

The second stage – Breeding, the PAT is populated with circuits that are random augmentations of circuits that are already retained in the front. The retention rate at this point climbs up, because there are more chances of finding a non-dominated and non-dominant circuit in the neighborhood of a member of the front than elsewhere in the size space. This stage also runs faster, because there are more chances of finding a viable circuit right next to a viable one. Further, circuits that are bred from existing ones are evaluated much faster, because their operating points are close to the parent circuits and thus take less iteration to converge.

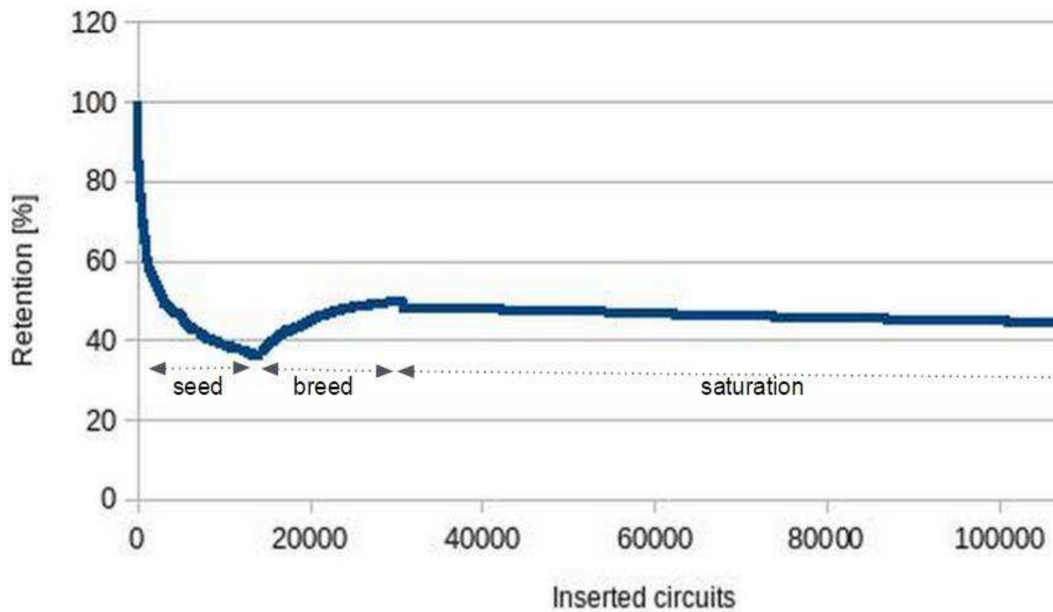


Figure 26: PAT populating decreasing retention rate

5.5.4. Extracting Pareto Fronts

Naturally, it is impossible to store all the sizing combinations and the performance attributes they produce. An opposite approach makes more sense: store only a subset of useful combinations of performance attributes and the circuit sizes that implement them. Usefulness of a circuit is established if there is no other circuit that can perform better in every defined aspect. This is the definition of non-dominated circuit and the set of those circuits forms a Pareto front. The common objectives of circuit optimization: area and power are also included in the definition of performance aspects. The PAT does not pre-assign circuit performance properties to potential "constraints" or "objectives", thus allowing users to switch between dual problems, e.g. "what minimal area should be expected from A_{DC}>=20dB?" Or "What is the maximal A_{DC} that can be expected for area<=10$(um)^2$?" The PAT does discriminate between "more is better" (e.g. BW) and "less is better" (e.g. Thermal Noise) properties.

The first step toward a solution is extracting archived solutions that are relevant to the spec. This is done by collapsing the generally calculated Pareto in the degenerated space defined by the spec. A spec may assign one of the following types of values to each one of the circuit's properties:

1. Unspecified – ignore this property for dominance consideration.
2. Best – consider this property value for dominance.
3. Inequality – use property value as-is, unless it is greater than the threshold in the spec, in which case take the threshold in its stead.

4. Equality – actual value for dominance consideration is the distance between property value and one in the spec. – only used in operating-point and constant sizer requirements.

When a spec is applied to a Pareto front, previously co-existing circuits can now be dominating/dominated, because ones advantage over the other may be assigned “unspecified”. Another possibility of elimination is that they both exceed an inequality, which means they are equal under the spec, leaving them with only disadvantages.

Table 4: Circuit Grading and Eliminating w.r.t. a Spec

	A_{DC}	CMRR	BW	Area	
Spec:	>30dB	ϕ	>2MHz	best	
Circ1	35dB	45dB	4MHz	$20um^2$	Dom by 2
Circ2	32dB	40dB	3MHz	$15um^2$	Met Spec
Circ3	25dB	50dB	2MHz	$5um^2$	Trade-off
Circ4	20dB	60dB	1.5MHz	$15um^2$	Dom by 3

Applying a spec to the PAT (Table 4), two circuits are eliminated:

1. Circuit 1 is better than 2 in general, although it pays for its improved gain, common-mode rejection and bandwidth in area. When the spec sets the requirement to 30dB and 2MHz, both circuits’ surplus gain and BW are ignored and circuit 2’s lesser area makes it dominant over 1.
2. Circuit 4 has better common-mode rejection compared to circuit 3, which is the justification for its inclusion in the PAT in general. However, this performance aspect is

outside of the spec. In other words: specifically irrelevant. All that is left are disadvantages compared to circuit 3, which cause circuit 4's elimination.

The specific-front is therefore a subset of the general front. It may contain circuits that do not meet the spec's inequality levels, but are useful for showing the user how loosening the requirements on one property can yield better results on another. In the example above, DC gain can be traded for area. It is useful, because a small sizing change to it may bring it up to the specified gain at a still smaller area of circuit 2. In optimization theory terms: it may not be a feasible solution, yet closer to the optimum than the found feasible one.

In case there is more than one "Best" value, which translates to multi-objective optimization, there may be more than one circuit that meets the thresholds. The algorithm for re-calculating a sub-front is $O(PN^2)$ P being the number of properties and N number of circuits, which can be slow for PAT's of N in the magnitude of 10^5 and P in the magnitude of 10^1. To overcome that, a loose fitness function is first applied to filter out all but $N=10^3$ circuits. This non-pure step actually assigns weights to the properties and thus contradicts the multi-objective concept behind Pareto. However, the circuits left by this filtration still form well populated fronts and the filter's target size can be modified if it is too aggressive for some specs.

After extracting a specific front, the PAT further filters the set according to the required graphic presentation. This is done by applying an additional artificial spec, which is made of "Best" entries for the axes properties and "unspecified" for everything else. The resulting set is a front tracing the tradeoff between the axes properties, with regards to the required performance figures.

5.5.5. Culling

Another PAT operation, designed to keep the circuit set diversified, removes circuits with duplicate performance according to a similarity factor S:

Algorithm 9: Culling Similar Circuits from the PAT

```
Pi,j – property i of circuit j
S – similarity factor (smaller -> more aggressive culling)
Begin
    // Init thresholds
    Foreach i
    Begin
        Pi,min = ∞
        Pi,max = -∞
        Foreach j
        Begin
            If (Pi,j<Pi,min) Pi,min=Pi,j
            If (Pi,j>Pi,max) Pi,max=Pi,j
        End
        Pi,th=( Pi,max- Pi,min)/S
    End
    // Detect and eliminate similarities
    For (i=0;i<|circuits|;i++)
    Begin
        Similar=1;
        For (k=i+1;k<|circuits|;k++)
```

```
            Begin
                Foreach j: If (abs(Pi,j-Pk,j)>Pj,th) Similar=0;
            End
            If (Similar) eliminate(i);
        End
    End
```

The expensive algorithm described in Algorithm 9 (time complexity: $O(PN^2)$) can be accelerated by sorting the circuits according to their properties in separate lists and focusing on suspect duplication. However, the motivation to do that is not high, because this algorithm runs in the more relaxed offline phase and if repeated in its aggressive setting (low S) often enough, has very few circuits to run on.

For the purpose of lab tests, there is no real need to limit the number of long-term archived circuits. However, it is necessary for community-wide deployment to make sure the PAT's do not grow unchecked. Space limitation is not only a matter of disk space, but primarily derived from required response time. PAT inflation can occur when a large number of users' queries fill up the PAT with circuits too quickly. The tool needs to avoid becoming "electro-mechanical" by having to swap large PAT listings between DRAM and hard drive.

Cache management LRU policy is useful for discarding no longer needed circuits in order to keep PAT in a reasonable size. Each circuit gets a time-stamp field, which gets refreshed when it becomes part of some user's specific front. Merging and culling operations essentially turn two circuits into one. When one of two circuits get eliminated, the latest time-stamp is assigned to the remaining one. After merging and culling, the daemon applies binary-search for an expiration date that only keeps the number of circuits it can fit in the allocated space.

5.5.6. PAT Size Requirements and Limitations

The pre-populated PAT is necessary to give an initial indication of topology performance limitations. Without a pre-populated PAT, online generation of Pareto fronts would take hours, which is far from a commonly-expected query time on a website. The lower-limit of a PAT size was observed to be in the 10K sized circuits, provided they are diverse enough to capture the performance surface evenly. The upper limit is determined by the number of circuits inserted to the PAT in a single transaction and storage limitations. At measured average insertion time of 300nsec per already-stored circuit (see. Experimental Results D.2.ii), a transaction of 103 of circuits to take place in ~10 seconds is in the magnitude of 1M circuits, which occupy 10 of disk space, a reasonable size per topology.

5.6. Web Applications

5.6.1. Data Visualization

Graphic visualization of data is the bottom line outcome of the Γ system. The main format chosen for final data output is the Scalable Vector Graphics (SVG). SVG has the following advantages:

1. It is an ASCII format, easy to be generated with simple Tcl procedures.
2. It is an instance of XML, which is both web-oriented and compatible with Tcl's Polish-Notation (PN).
3. Generated images are scalable without loss of quality.
4. It is a well-established format, having ample community support in freely available editing and converting software.

Visualization in Γ can be any of the following 4 categories:

1. Scatter plot – showing a field of markers along X/Y axes. The field is marked with a mesh and labels on the axes for orientation. The markers can be connected to show curves.
2. Histogram – column diagram used mainly for plotting the distribution function of errors.
3. Heat map – false-colored “heat” representation of a 2D functions
4. 3D wire mesh – isometric surface representation of a 2D function.

All applications of data visualization use the same code, consisting of few multi-purpose procedures. This makes the look and feel of charts uniform across online and offline flows and enables centralized upgrades and maintenance. Regardless of type of plot, each has to go through the following stages:

1. Calculate the range of the X/Y domains and the ratio between physical levels and pixels.
2. Determine appropriate pitch per axis for the background grid
3. Generate SVG code for the frame, grid, labels, titles and markers.

X/Y ranges are easily determined from the extreme values of the given data. There is an option to add a belt to the range by extending the range by 5-10% on each side. Determining the grid’s positions and pitch was initially done in a naïve procedure that divided each axis range to 10 equidistant stops and labeled them accordingly. This approach is simple to implement and requires minimal calculations. However, the grid lines fall in fractional positions that may contribute to user disorientation instead of suppressing it. For example, it X is ranging between

1.35 and 8.55, the labels generated in the simple method are 1.35, 2.15, 2.95… Automatic graph generators of the kind found in spreadsheet programs do a better job of finding a pitch and starting point that are more natural for humans to absorb in a glance. For the above example, a simple 1, 2, 3… grid serves a better orientation guide, even though it has fewer grid lines along the X axis. The data visualization procedures were updated with automatic grid finder that looks for appropriate units is in currency-denominations steps: 1's, 2's, 5's, 10's etc.

Histogram plot uses rectangles for columns instead of markers. Since histograms convey the statistical distribution of the data, optional statistical notation can be added. Average, standard-deviation multiplications and top percentiles can be marked either by drawing vertical lines or by changing the columns' fill colors.

3D surface plotting of a 2D function is done with isometric projection. This plotting mode is limited to presenting the path of slope optimization algorithms, because it is hard to discern X/Y positions from location on screen or function values from the surface height.

For most purposes, the heat-map is the preferred method for showing functions of z=f(x,y) form. The heat map itself is generated in a bit-map (BMP) format that is referenced from the SVG code. Bit maps are binary arrays of pixels' color values. The bitmap is generated from a list of (x,y,z) points using Algorithm 10.

Algorithm 10: Interpolate a heat map out of a data set in bitmap format

Begin

For all given $p_i=(x_i, y_i, z_i) \in P$

find x_{max}, x_{min}, y_{max}, y_{min}, z_{max}, z_{min}

```
Allocate float F[X_resolution,Y_resolution]
Define: X_i=floor((x_i-x_min)/(x_max-x_min)*X_resolution)
Define: Y_i=floor((y_i-y_min)/(y_max-y_min)*Y_resolution)
// Scan all pixels in the array
For (i=0;i<X_resolution*Y_resolution;i++)
Begin
    // Scan all given data points
    Sigma=0
    Weights=0
    For (j=0;j<|P|;j++)
    Begin
        Sigma=Sigma+z_j/((x_i-x_j)^2+(y_i-y_j)^2)
        Weights=Weights+1/((x_i-x_j)^2+(y_i-y_j)^2)
    End
    F[i%X_resolution,i\X_resolution]=Sigma/Weights
End
Allocate float TH[|colors|+1]
TH[0]=z_min
TH[|colors|]=z_max
// Find optimal thresholds with binary search
For (i=1;i<|colors|;i++)
Begin
    Target_num_of_pixels=i*X_resolution*Y_resolution/|colors|
    TH_min=z_min
    TH_max=z_max
    TH=(TH_min+TH_max)/2
```

```
    For (j=0;j<N;j++)
    Begin
        Num_of_pixels=0;
        For (k=0;k<X_resolution*Y_resolution;k++)
        Begin
           If (F[k%X_resolution,k\X_resolution]<TH) Num_of_pixels++
        End
        If (Num_of_pixels<Target_num_of_pixels)
            TH_min=TH
        Else
            TH_max=TH
        TH=(TH_min+TH_max)/2
    End
    TH_i=TH
End
// create the bitmap array
Allocate BM[X_resolution,Y_resolution]
For (x=0;x<X_resolution;x++)
Begin
    For (y=0;y<Y_resolution;y++)
    Begin
        C_max=|colors|
        C_min=0
        C=(C_max+C_min)/2
        For (i=0;i<log_2(|colors|);i++)
        Begin
```

```
                    If (F[x,y]<THc)
                        Cmin=C
                    Else
                        Cmax=C
                    C=(Cmax+Cmin)/2
                End
                // Color the pixel according to the threshold
                BM[x,y]=Palette[C]
            End
        End
    End
```

After interpolating all the pixels in the array, Algorithm 10 performs two binary searches: 1. per color, calculate its z levels, in a way that each color gets an equal number of pixels 2. per pixel, find the appropriate z level segment and assign it a color.

A faster, cleaner, and memory-efficient algorithm avoids pixel-level interpolation and produces scalable vector representation of the heat map. Algorithm 11 assumes that the set of given pixels P is aligned to a matrix with relatively small number of rows and columns. Typically, a sweep of parameters has 30-50 value in each dimension, which means that the input set has few thousands of samples. Therefore, the threshold allocation loop can be done on the input set of pixels, without spending time on interpolating each pixel in the final image. After allocating z thresholds, the algorithm assigns color per input pixel and groups together adjacent pixels that share the same color to polygon sets. In the final step, the algorithm walks around

each polygon set, using a short list of rules, and estimates the location of all the points that define the boundary of the polygon.

Algorithm 11: Scalable Vector Graphics heat map generator

```
// Allocate and calculate TH[] as in Algorithm 10, using P only
  // Color each p_i as in Algorithm 10
// Group together p_i's that have the same color
Define stepX[]={-1,0,1,0} // these numbers mean
Define stepY[]={0,1,0,-1} // west,north,east,south
For each polygon set PS_i
Begin
    Find (x,y)=(x_0,y_0) on the boundary of PS_i
    Boundary_i={}
    C_this=color(x_0,y_0)
    Dir=0 // start with west
    While "forever" do
    Begin
        C_other=color[x+stepX[Dir],y+stepY[Dir]]
        If (C_this!=C_other)
        Begin
            x_boundary=x+stepX[Dir]*(z-TH[C_this/other])/(z-z_other)
            y_boundary=y+stepY[Dir]*(z-TH[C_this/other])/(z-z_other)
            Dir=(Dir+1)%4 // rotate clockwise
            Append (x_boundary,y_boundary) to Boundary_i
        End
        Else
```

```
            Begin
                x=x+stepX[Dir] // walk
                y=y+stepY[Dir]
                Dir=(Dir-1)%4 // rotate back
            End
            If (x,y==x0,y0) break
        End
    End
```

Algorithm 11 assigns each shape a closed contour. However, it neglects special cases where the contours leave areas in the image uncovered. Using the walk-around loop as described above can create white triangular artifacts in the image. This can be overcome by using a set of rules described in Figure 27. The simple rotation corner in Algorithm 11 is performed only when there are two colors between the corner and its neighbors. In case there are three or four colors in the corner, an additional point is added to the boundary. Figure 27 shows three types of corners in the walk around a green polygon. The black dots show linear estimation of where two color regions should border. The white dots represent a special corner point that is added to avoid a white triangle artifact in the final image.

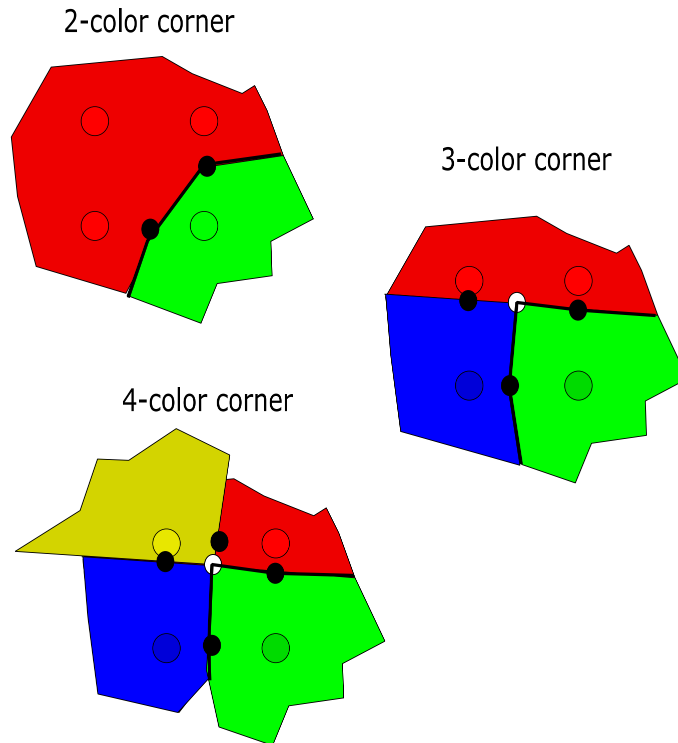


Figure 27: Corner rules added to Algorithm 11

The list of points $Boundary_i$ is printed out for each polygon in SVG format. Even though it is ASCII based, files of this representation of a heat-map are ~100 smaller than the binary bit-maps.

5.6.2. Single Transistor Sizer

The first optimization tool chosen to illustrate the performance of the proposed LUT approach is a transistor size calculator that can convert operating conditions and constraints into length and width metrics. It is a tool close in function to the one proposed by Binkley[28]. However, the approach proposed by Binkley is structurally intended for sizing single transistors. The "MOS Design Tool" was created to give designers information about a single transistor's

performance parameters, given design choices using the EKV model. The tool provides a straightforward "pass" or "fail" for a given spec for a transistor with the execution of a batch of equations from the EKV model. There is no attempt to understand designer's intent. Therefore, no search is performed to meet the spec, and/or minimize transistor size/power while meeting the spec. The usage model of this tool proposed by Binkley justifies its reliance on using the full-blown SPICE model for evaluation.

However, for the purpose of automatically executing a thorough search in the solution space, this model cannot be scaled up. The "Transistor Sizer" proposed in this paper evaluates 1000's of transistor configurations, varying in geometric and other design parameters (such as bias current and over-drive voltage), to meet a given spec while optimizing area and/or power. This is only the first step toward multi-transistor circuit optimization, in which the speed and accuracy advantages of LUT models over equation-based ones is bound to play a cardinal role.

The tool starts with a fill-in form (Figure 28) that lets the user chose top-level characteristics, such as technology and channel type, and then type in V_{DS} and I_D. The user sets constraints on other aspects of the transistor selection space, such as noise floor or offset. After the user hits the "submit" button, the tool scans transistor sizes and biasing parameters in the space permitted by the spec. It uses a gradient algorithm to find a feasible and size-optimal solution that meets all the specs, if such solution is possible. Since this tool is a research engine for verifying the correctness and usefulness of the LUT models, the tool also generates SPICE circuits and harness for comparing its findings with NGSPICE. It launches a simulation and fills up the 3 leftmost columns with results and error metrics.

Transistor Sizer TSMC018

Parameter	Input	Calculation	Source	Simulation	|Error|	Error[%]
device	nch					
process	ff					
temp		-40°C	(step 1)			
Operating Limits						
V_{DS}[mV]	0.4	V				
I_{DS}[mA]		22.333uA	(step 4)	21.739uA	594.165nA	2.733%
Primary Performance Targets						
g_M		140.912u℧	(step 6)	141.206u℧	-294.049n℧	-0.208%
r_O		83.688KΩ	(step 9)	84.616KΩ	-928.298Ω	-1.097%
Gain		21.432dB	(step 10)			
Optional Parameters						
V_{GS}[mV]	0.55	V				
V_{BS}[mV]	0	V				
Secondary Performance Targets						
Hz ">Noise_Floor		1.344pA/√Hz	(step 11)	1.547pA/√Hz	-203.053fA/√Hz	-13.120%
i_{Nf}*f		5.030nA	(step 12)	4.169nA	860.628pA	20.641%
f_C		3.741KHz	(step 13)			
V_a		-7.130V	(step 14)			
V_T		394.007mV	(step 15)			
Calculated Parameters						
L	1.8e-7	m				
W	2.3e-7	m				
Area		41.4mum^2	(step 16)			
σI_{DS}		1.900%	(step 17)			
σV_T		3.596%	(step 18)			
L_{min}	1.8e-7	m				

Submit

1 Calculating temp based on process
temp=process2temp(process)=-40°C

2 Calculating Ids_unit based on {Vgs Vds Vbs L W}
Ids_unit=Ids(Vgs Vds Vbs L W)=17.478uA

Figure 28: "Sizer" Screen Shot

The sizer produces a report with all the steps it needs to search and calculate the unconstrained parameters. The slope algorithm used to find optimum starts with a minimal length and width transistor. For each step, it scans an array of lengths and widths around the current transistor size and selects the one that gets the minimal distance from the spec. The selected length and width serve as the search direction followed by the optimization algorithm. The algorithm steps in the found direction and doubles the step size until there is no longer improvement in the distance to the spec. When the selected direction is exhausted, a new search is initiated and a new direction selected. The solution stops when the spec is met. The path is presented on an isometric projection of the spec-distance function (Figure 29).

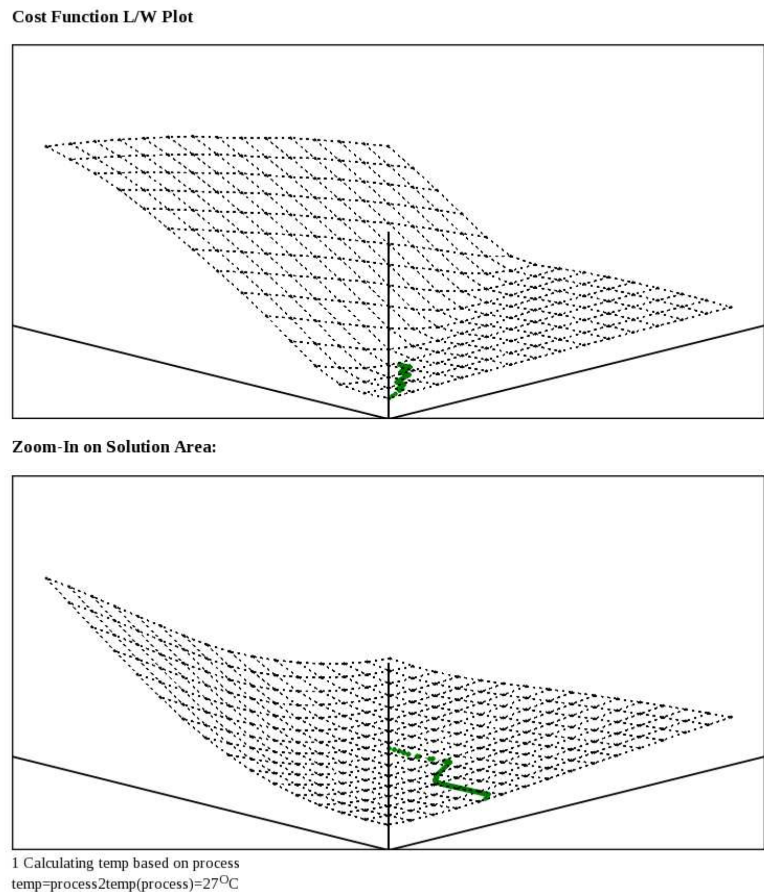


Figure 29: Search Path Plot on top of Spec-Distance Function Surface (zoom-in below)

The solution is posted in the form and a sign-off simulation is launched to verify the results' accuracy.

The sizer is implemented as a scripted prototype. Its structure and design lessons served as a starting point in constructing the next level in circuit evaluation software for automated synthesis.

5.6.3. FETScape

The front-end of Γ is a website, implemented in Ajax architecture to accelerate transactions and match responsiveness with on-premises tools. The site is organized in tabs. After the main Analysis tab, one can find trivial web services, such as forum and chat, report authoring utility and help.

A three-tier service model creates responsive interactivity (Figure 30)

1. User-Browser exchange – Γ stores all the navigational data it can in the browser's JavaScript workspace. This makes selecting circuits from the presented map, populating the table and updating hover-events on the schematic pane to appear instantly, well under the 0.1sec time limit.
2. Browser-Server queries – Generating maps as a reaction to spec changes and axes selection is performed as response to Ajax requests. The map generating script loads the PAT, applies the spec and generates the SVG and sometime bitmap graphics for the displayed map. All this is done in ~1sec time budget. Every Ajax request is repeated several times in 5sec intervals to update the graphics with PAT changes.

3. Background Optimization – outside users' direct control, an iterative daemon is looking for active sessions in Γ's work area. If it finds one, it looks for GUI-selected circuits in the session records as hints to the user's interests. It then uses optimization algorithm (random hill-climbing, line searches etc.) to improve on user-selected circuits. It inserts the improved ones back into a new copy of session-dedicated PAT. This new version of the central PAT is from now on the one used by cycle 2, which means that the user can expect to see new circuits appear on the map every 5-10sec, as long as the session is active.

The goal of creating curves and heat-maps in seconds was reached for the system. Further accelerating the website can be done by converting some of the asynchronous CGI transactions to a resident process server that is preloaded with all the necessary binaries: PAT, topology shared-object and transistor-level tables. This future acceleration will free some time from the server and distribute the bulk of the work to remote and more powerful machines.

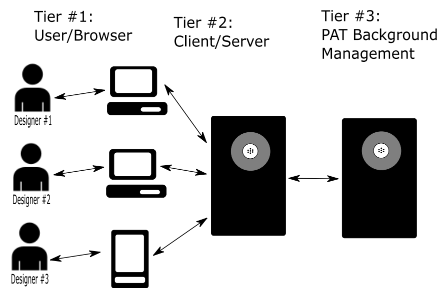


Figure 30: System Architecture from a Designer's Perspective.

6. Experimental Results

6.1. Methodology

Quality analysis of the constructed systems, Sizer and Γ, is composed of the following:

1. Accuracy of results, compared to commercially available tools (Spectre®).
2. Calculation time, compared to SPICE simulations.

Exhaustive comparison of transistors and circuits is not possible, due to the continuous and multi-dimensional nature of the sampling space. Even comparison of too large (>100's) sample size is difficult, because automated simulation loops in Spectre SKILL interpreter can take hours to finish and more often crash the tool completely before results can be post-processed to statistics. The workaround to this problem is a two-staged comparison, which serves to validate both NGSPICE and the systems derived from it. The first stage compares RAMSpice itself to Spectre (Figure 31). The two simulation tools are expected to be very close to each other, because they are inheriting the same simulation code and architecture and implement the same transistor models. Differences, if any occur, can only be attributed to different numerical methods (round-off), errors in converting the models parameter files, and differences between test circuits. Since significant differences are not expected, a smaller sample size is required for this stage, compared to the next one.

Figure 31: First of Two-Staged Comparison Procedure Data Flow

The next stage of comparing Sizer and Γ to SPICE is done within the single executable RAMSpice. This monolithic binary contains both SPICE simulator and the original systems presented here. Since this stage is expected to come up with some mismatch between the two tools compared, a relatively larger sample size is required to collect enough data. RAMSpice runs a Tcl script that can collect 1000's of data points in under an hour and construct all visualized collateral in a document that provides a "certificate of calibration".

6.2. Transistor-Level Results

6.2.1. Spectre to SPICE Comparison Stage

The main parameter to compare between Spectre and SPICE in transistor level is the channel current I_{DS} as a function of voltages, length, width and corner (Figure 32).

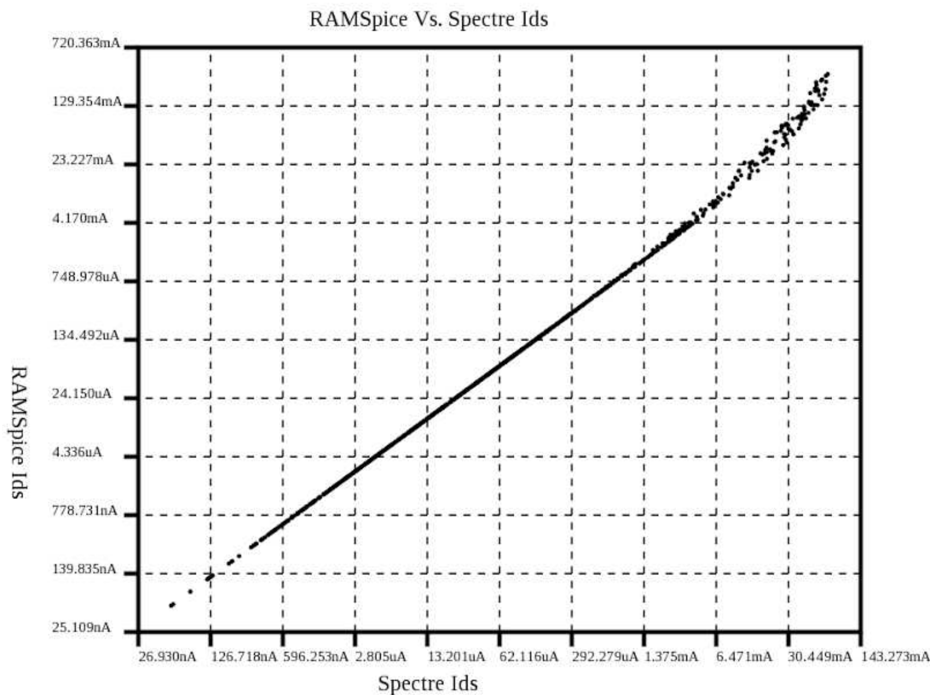


Figure 32: Scatter Plot of Spectre vs SPICE Ids Values (TSMC 40nm NMOS in SS 125C)

The plot seems to follow the Y=X line for currents up to 1mA, where some mismatch begins do develop. This phenomenon was initially attributed to code differences between the two tools, which may result in accumulated round-off errors. However, when the relative-error histogram was plotted, a different picture begun to appear (Figure 33).

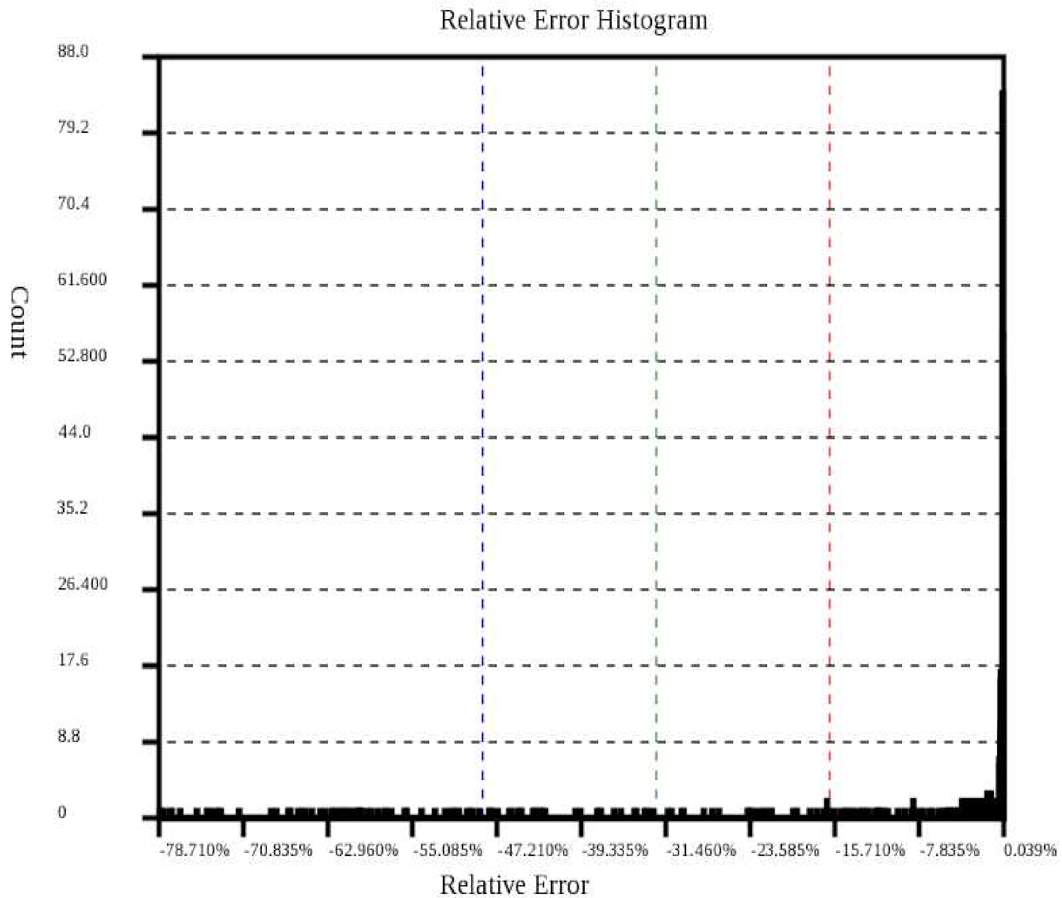


Figure 33: Relative Error Histogram For Figure 32

Figure 33 shows that the majority of samples do fall on the X=Y line (see thick column on the right). However, the errors can go up to 75% and more when they are not prt of this line. Some transistor geometries are understood completely differently by the two tools.

Inspecting the error occurrence in different geometries revealed that some model bins are more immune to these errors than others. Specifically, wider transistors display more discrepancy between SPICE and Spectre than narrower ones. After eliminating W/L>10 transistors from the comprison, the tools show perfect match (Figure 34).

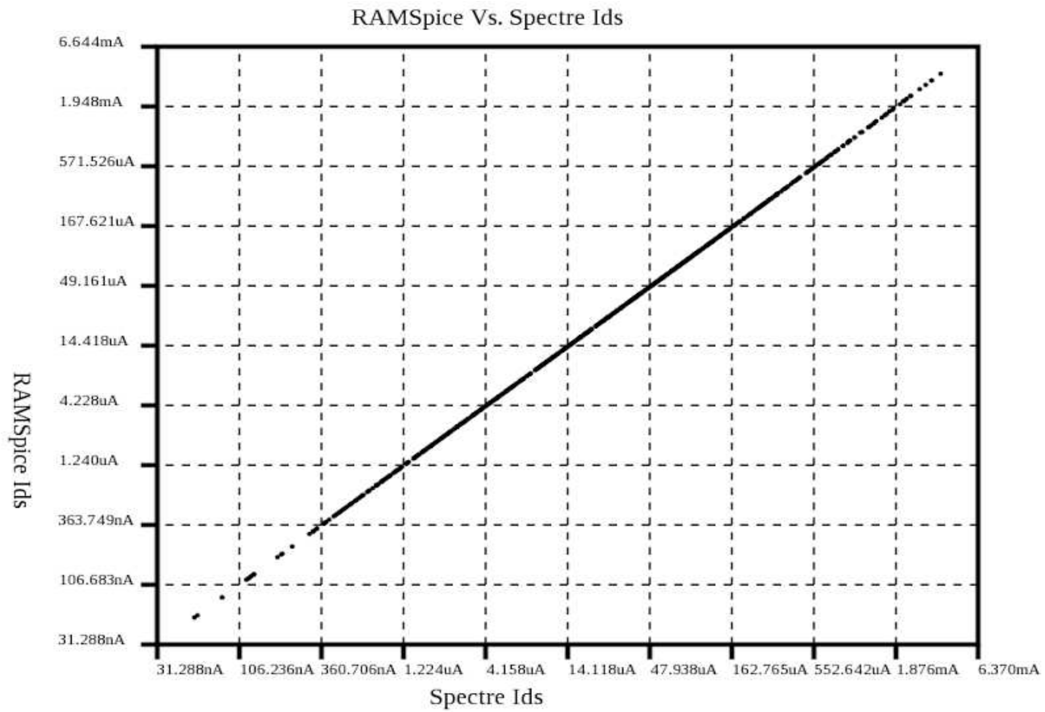


Figure 34: Spectre vs SPICE Ids match, without W/L>10 Channels

The fuzzy top of Figure 32 is now gone and the histogram reveals that the errors are negligible (Figure 35).

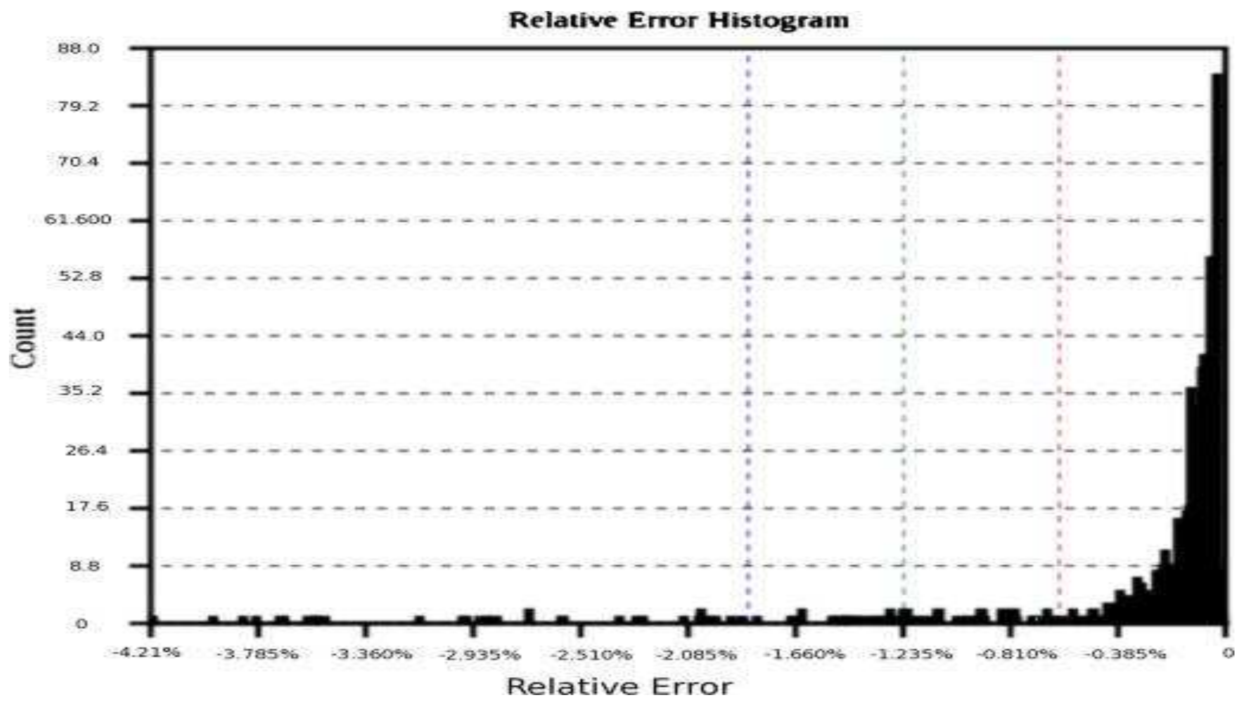


Figure 35: Relative Error Histogram for Figure 34

Since Spectre is a complete black-boxed tool, due to its commercially-protected source code, there are not many opportunities for understanding in depth the contribution of transistor widths to SPICE/Spectre mismatch. However, wide channel transistors of more than 10 squares are never used in an actual design and they pose greater problems in comparing 4D models in the second stage. Thus, the finding that RAMSpice and Spectre match as long as W/L does not exceed 10 was considered satisfactory for this research. Transistors of wider channels can still be implemented as parallel connection between two or more narrower channels, known as “fingers”. That option is available by both tools in a parameter known as “m”, the multiplier.

All operating-point parameters are derived from I_{DS}. For example, g_m is extracted from running I_{DS} characterization loop with V_{GS} stepped up ε above the original values and then using the original I_{DS} to calculate the derivative of I_{DS} w.r.t. V_{GS}. Therefore, matching I_{DS} between the SPICE and Spectre is enough to prove that the two models are understood the same by the two. Repeating the same analysis for TSMC 180nm yields the same quality of results (Figure 36).

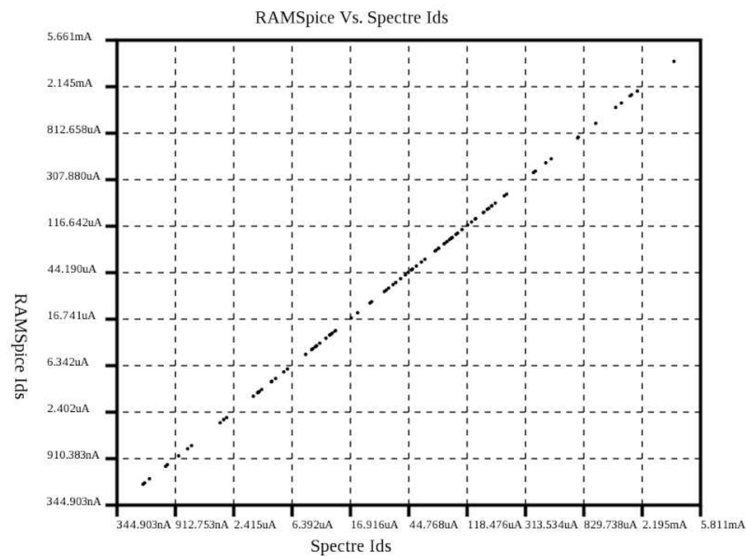


Figure 36: 180nm NMOS and PMOS Ids comparison between Sepctre and SPICE

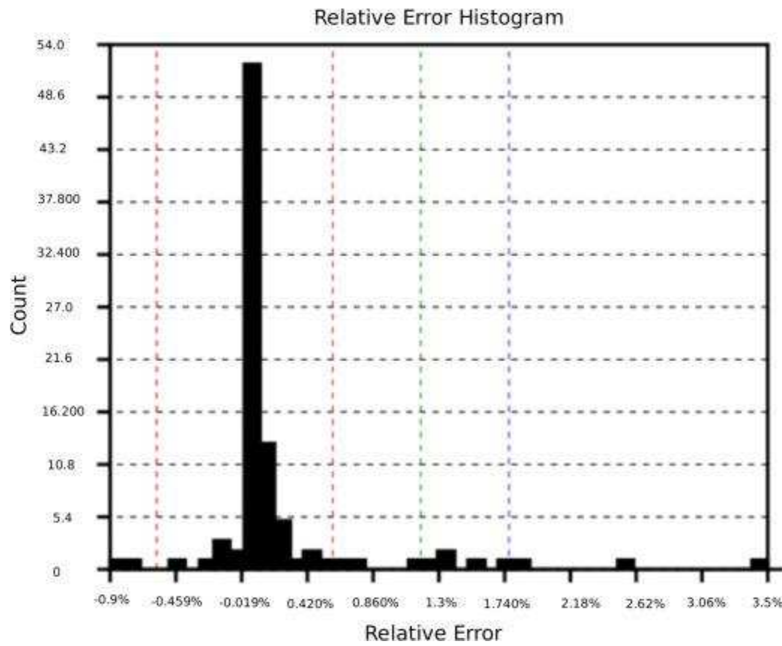


Figure 37: Relative Error Histogram for Figure 36

Figure 37 lumps together all the 180nm relative errors in a histogram. There is no control of channel type or length, except for limiting W/L≤10. The two tools match for 180nm as well as 40nm.

6.2.2. Budgeted Resolution

To optimize the resolution per view and corner it is not enough to find a resolution that satisfies our required accuracy and access time. Given the first-order interpolation methods discussed in 5.1.35.1.5, the source of mismatches between Γ's LUT models and the ones used by SPICE is the curvature of the modelled functions. The mismatches are therefore minimized when the sample resolution grows and that translates directly to higher memory demand. Furthe more, the same overall number of samples can be used in different combinations of per-parameter resolution. Some parameters return more accuracy per invested resolution, so it is important to

chart the best use of memory when a sample size is invested to gain accuracy and speed. The trade-off can be analyzed using Pareto optimization. Nested loops scanned logarithmic scale of resolutions in √2 steps, for each view and corner. The resulting combinations were plotted in the error/memory field (Error: Reference source not found), where the X-axis shows the budgeted memory and the Y-axis shows the maximal error. Any point that was dominated by another i.e. required more memory yet resulted in greater error than the dominating one, was removed from the set. After removing the dominated points, some other points were ruled out by proving low return on investment compared to others using convex hull algorithm. The final set represents the Pareto front, which is used to show the limits of memory/accuracy trade-off and the optimal distribution of resolution between the parameters, per trade-off point. The Pareto front is a set of useful resolution combinations. The same analysis is repeated for the memory/access-time trade-off. Since the internal partitioning of the behavior space is hidden from the querying code, each device, view and corner LUT is free to be constructed in a different resolution. Choosing the minimal memory size per LUT that satisfies both accuracy and access-time requirement is therefore dependent on this preliminary analysis.

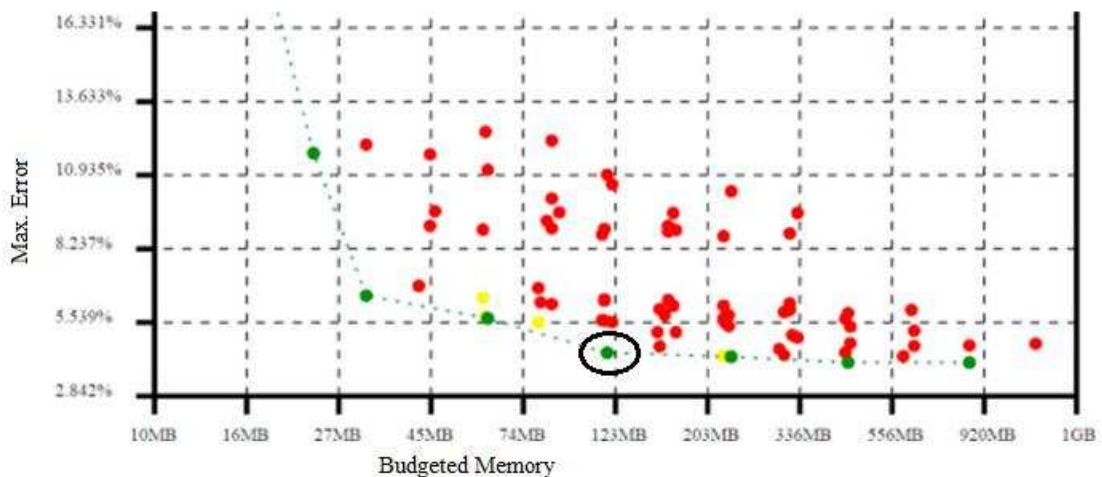


Figure 38: IDS resolution combinations memory and resulting maximal error

The elimination process for I_{DS} yielded a Pareto front comprised of the following useful resolution combinations (Table 5):

Table 5: Useful IDS (SS, 125 °C) resolutions

	V_{GS}	V_{DS}	V_{BS}	Memory[MB]	Max Error[%]
	9	9	9	13	28.600
	9	17	9	24	11.737
	12	17	9	32	6.520
	12	33	9	61	5.686
	23	33	9	117	4.427
	23	65	9	231	4.276
	23	65	17	437	4.075
	23	65	33	848	4.069

Since other parameters are derivatives of I_{DS}, the choice for the latter's sample resolution takes the highest priority. The advantage of keeping all parameters at the same budgeted resolution is that hypercube location code can be shared between them and composite interpolation is enabled.

The same analysis was done for access time vs accuracy trade-off (Error: Reference source not found) and the selected resolutions (Table 6).

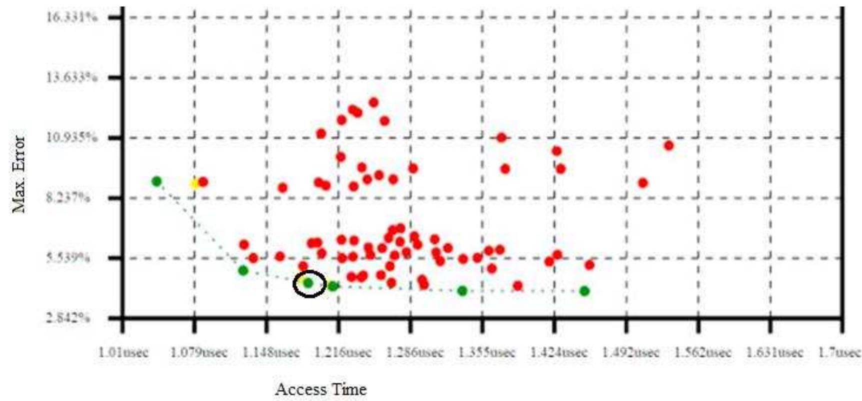


Figure 39: IDS resolution combinations access time and resulting maximal error.

Table 6: Useful I_{DS} (SS, 125 °C) resolutions vs. access time

	V_{GS}	V_{DS}	V_{BS}	**Time** [µs]	**Top error** [%]
	33	17	9	1.042	8.989
	33	65	9	1.125	4.989
	23	33	9	1.187	4.427
	23	65	9	1.211	4.276
	23	65	17	1.335	4.075
	23	65	33	1.452	4.069

The voltage resolution chosen for the follow-up analysis is therefore 5 bit for VGS and VDS and 3 bit for VBS. As the curves show, return on investment drops significantly for higher voltage resolutions. To maximize accuracy in the budgeted memory size (10MB per parameter LUT), the length resolution was chosen to be 8 bit. 5:5:3:8 translates to 8MB of single precision array.

6.2.3. Transistor Level LUT to SPICE Matching Regression

The results given here are PDF files generated automatically by the Γ regression flow. The regression suite is a collection of all the tools used to generate the LUT models: SPICE technology library generator and characterization loops. It contains an additional random sampling and statistical analysis of LUT vs. SPICE tool. The statistical results are plotted and summarized in a hierarchy of HTML pages that are published online. For a given manufacturing technology and resolution, the system populates all the models required for solving circuits containing N and P channel MOSFET's (nch and pch). Following LUT generation, a random sample of voltages and geometries is generated in a RAMSpice script. Γ's results are recorded together with results taken directly from a SPICE simulation. The models that are compared to SPICE represent a selection of parameters from 3 performance domains:

1. DC - I_{DS}, g_m, g_o
2. AC – C_{GS}, C_{GD}
3. Noise – thermal spot and flicker at 1Hz.

The sample size selected for graphing the results is 500. However, different sample sizes can be selected for getting more confident statistical results. The regression suite generates a hierarchy of HTML files that are published online and then converted automatically to PDF. The accuracy portal is a simple table showing analyzed technologies and resolutions. Clicking on one of the technology+resolution links leads to a summary page, showing a matrix of devices, corners, parameters error values in different percentiles and relative [%] or absolute values (Figure 40). The passing thresholds can be configured by using a configuration file, but by default a 10% diversion between Γ's interpolation and SPICE results is permitted to 99% of the

samples. The pass/fail status of each parameter is presented in green/red font coloring, respectively. Figure 40 shows a summary done for the TSMC 40nm technology, sampled at 5b resolution at V_{GS}, 5b at V_{DS}, 3b at V_{BS} and 8b at channel length (abbreviated to 5:5:3:8). This is a 4D LUT, which means higher order width effects are neglected. The two failing g_o corners can be waived because of the relatively low maximal absolute error in order of 10^{-7} Ʊ, compared to typical g_o values in the order of 10^{-5}Ʊ.

Accuracy Results for Tech: tsmc040 Resolution: 5:5:3:8

		nch						pch					
		tt		ff		ss		tt		ff		ss	
		delta	rel	delta	rel	delta	rel	delta	rel	delta	rel	delta	rel
Ids	50	63.535nA	0.123%	58.529nA	0.207%	68.927nA	0.269%	6.342nA	0.178%	9.671nA	0.246%	6.671nA	0.076%
	75	84.345nA	0.382%	103.17nA	0.441%	82.327nA	0.559%	15.962nA	0.284%	27.098nA	0.356%	9.561nA	0.174%
	90	186.364nA	0.661%	223.379nA	0.819%	184.672nA	1.093%	40.127nA	0.542%	98.378nA	1.019%	15.288nA	0.278%
	95	299.664nA	0.79%	290.469nA	1.520%	380.672nA	1.472%	50.362nA	0.974%	134.188nA	2.164%	23.468nA	0.338%
	99	420.764nA	1.137%	467.87nA	4.430%	576.172nA	1.817%	66.762nA	2.758%	255.511nA	3.906%	61.068nA	0.683%
Cgd	50	20.538aF	0.845%	21.3aF	0.893%	8.017aF	0.793%	9.465aF	1.258%	21.101aF	1.205%	15.258aF	1.249%
	75	20.814aF	1.095%	21.554aF	1.115%	8.216aF	1.032%	9.709aF	1.640%	21.331aF	1.470%	15.462aF	1.611%
	90	20.988aF	1.793%	21.725aF	2.399%	8.363aF	1.855%	9.86aF	3.404%	21.496aF	3.317%	15.631aF	3.497%
	95	29.431aF	3.859%	26.122aF	4.702%	27.291aF	4.258%	9.91aF	4.123%	21.557aF	4.226%	57.438aF	4.321%
	99	594.031aF	5.317%	549.782aF	5.525%	181.991aF	5.264%	388.075aF	4.616%	674.717aF	4.797%	376.198aF	4.539%
Cgs	50	23.573aF	0.221%	15.702aF	0.241%	29.153aF	0.184%	11.117aF	0.279%	16.335aF	0.254%	18.453aF	0.241%
	75	26.563aF	0.254%	18.99aF	0.266%	31.206aF	0.215%	13.021aF	0.311%	18.286aF	0.282%	20.472aF	0.28%
	90	36.696aF	0.811%	24.477aF	0.949%	41.206aF	0.655%	13.948aF	0.777%	19.701aF	0.757%	21.114aF	0.686%
	95	116.696aF	1.040%	74.477aF	1.341%	131.206aF	0.925%	69.232aF	0.966%	84.023aF	1.103%	116.769aF	0.875%
	99	366.696aF	1.327%	284.477aF	1.568%	501.206aF	1.115%	309.232aF	1.059%	384.023aF	1.205%	311.769aF	0.947%
gm	50	748.412nƱ	0.77%	1.079uƱ	0.801%	517.401nƱ	0.748%	67.795nƱ	0.341%	111.399nƱ	0.391%	55.86nƱ	0.308%
	75	1.046uƱ	1.363%	1.683uƱ	1.495%	663.898nƱ	1.147%	148.604nƱ	0.766%	252.299nƱ	0.938%	100.86nƱ	0.586%
	90	1.425uƱ	1.677%	2.055uƱ	2.143%	961.101nƱ	1.488%	404.795nƱ	1.422%	568.5nƱ	1.530%	243.66nƱ	1.193%
	95	1.705uƱ	1.976%	2.326uƱ	2.710%	1.161uƱ	1.732%	594.795nƱ	1.758%	806.199nƱ	1.836%	436.639nƱ	1.602%
	99	2.067uƱ	2.344%	3.042uƱ	3.559%	1.785uƱ	2.373%	850.895nƱ	2.331%	1.078uƱ	2.542%	761.46nƱ	2.588%
go	50	43.856nƱ	0.807%	62.035nƱ	0.882%	34.352nƱ	0.845%	10.745nƱ	0.936%	11.772nƱ	0.872%	10.924nƱ	0.892%
	75	86.483nƱ	1.396%	129.475nƱ	1.542%	65.782nƱ	1.416%	24.534nƱ	1.577%	27.527nƱ	1.502%	21.509nƱ	1.592%
	90	212.263nƱ	2.599%	252.095nƱ	2.581%	146.562nƱ	2.549%	48.535nƱ	2.961%	60.407nƱ	2.540%	38.939nƱ	2.580%
	95	291.846nƱ	3.790%	383.464nƱ	3.594%	245.342nƱ	3.850%	70.545nƱ	4.570%	107.147nƱ	4.579%	53.979nƱ	4.056%
	99	591.736nƱ	6.575%	536.735nƱ	6.698%	544.027nƱ	7.602%	235.255nƱ	12.382%	316.067nƱ	12.131%	166.049nƱ	5.536%
Flicker_Noise_at_1Hz	50	407.51mV^2/Hz	3.878%	673.395mV^2/Hz	3.657%	214.663mV^2/Hz	3.209%	1.702V^2/Hz	2.930%	3.516V^2/Hz	2.434%	941.923mV^2/Hz	2.343%
	75	1.208V^2/Hz	5.728%	1.438V^2/Hz	5.804%	634.59mV^2/Hz	4.932%	1.907V^2/Hz	4.543%	3.730V^2/Hz	4.400%	1.059V^2/Hz	3.561%
	90	2.107V^2/Hz	7.227%	2.598V^2/Hz	7.154%	1.173V^2/Hz	6.382%	3.187V^2/Hz	5.709%	7.925V^2/Hz	6.661%	1.785V^2/Hz	4.667%
	95	2.685V^2/Hz	8.319%	3.519V^2/Hz	7.747%	1.679V^2/Hz	7.666%	7.105V^2/Hz	6.526%	13.728V^2/Hz	7.522%	3.343V^2/Hz	5.379%
	99	3.949V^2/Hz	9.685%	4.887V^2/Hz	9.940%	2.777V^2/Hz	9.162%	16.951V^2/Hz	8.364%	30.497V^2/Hz	8.662%	7.917V^2/Hz	6.516%
Thermal_Noise_Floor	50	25.084mV^2/Hz	1.123%	23.115mV^2/Hz	1.067%	21.32mV^2/Hz	1.171%	4.46mV^2/Hz	1.277%	5.382mV^2/Hz	1.268%	4.132mV^2/Hz	0.897%
	75	35.644mV^2/Hz	2.082%	37.734mV^2/Hz	1.907%	28.226mV^2/Hz	2.014%	6.672mV^2/Hz	1.891%	8.016mV^2/Hz	2.150%	6.413mV^2/Hz	1.379%
	90	51.275mV^2/Hz	3.108%	49.665mV^2/Hz	3.034%	46.236mV^2/Hz	2.690%	9.603mV^2/Hz	2.601%	13.665mV^2/Hz	2.901%	12.185mV^2/Hz	1.895%
	95	86.185mV^2/Hz	3.693%	70.174mV^2/Hz	3.789%	90.346mV^2/Hz	4.033%	16.554mV^2/Hz	3.004%	16.942mV^2/Hz	3.343%	20.915mV^2/Hz	2.763%
	99	179.145mV^2/Hz	6.201%	168.594mV^2/Hz	5.444%	174.556mV^2/Hz	7.343%	25.62mV^2/Hz	3.682%	27.311mV^2/Hz	3.807%	35.435mV^2/Hz	4.276%

Figure 40: Error Summary Report for TSMC 40nm and Resolution 5:5:3:8

Each device, corner and parameter rubric links to a detailed report. The report has four quadrants presenting different analyses of the results (Table 7).

Table 7: Structure of Device+Corner+Parameter Detailed Report

X/Y Scatter Plot (expected Y=X straight line)	Listing of Statistical Metrics
Histogram of Errors in Original Units	Histogram of Relative Errors [%]

6.3. Circuit Level Results

6.3.1. Tested Circuits

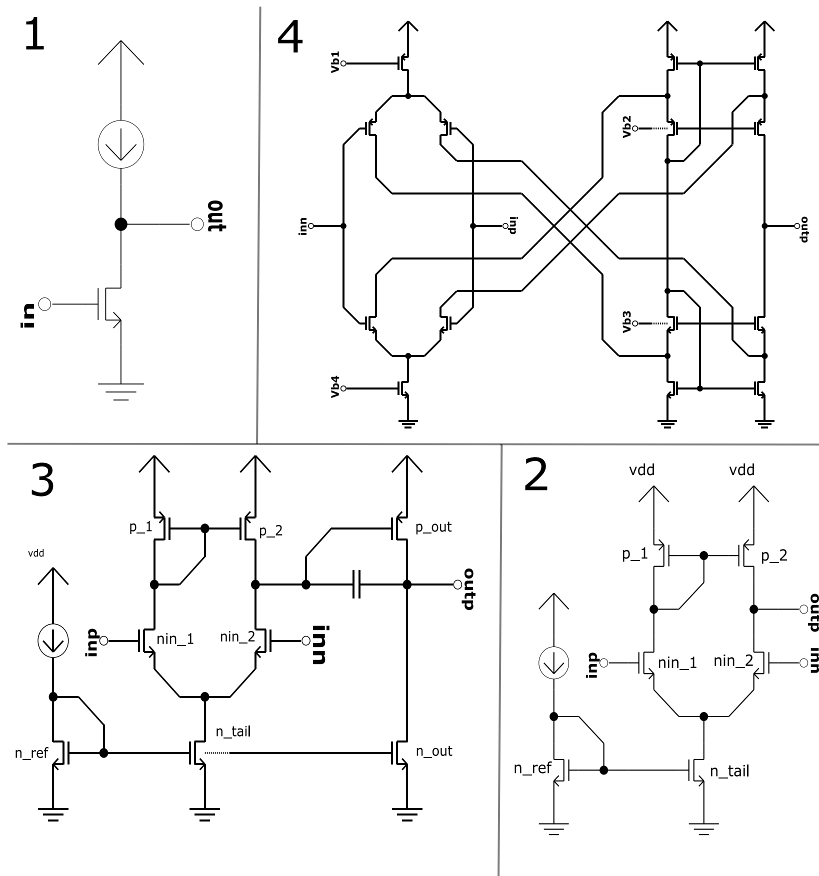


Figure 41: Four Tested Topologies

Figure 41 shows the four topologies tested for accuracy:

1 Common-source with ideal current source.

2 Differential pair

3 Differential pair with output stage

4 Folded Cascode operational amplifier

6.4. Run Time Analysis

6.4.1. Transistor Level

6.4.1.1. Methodology

Improving model access time was the main motivation for transistor-level phase of the research. The existing BSIM provided batch of equations is considered the standard for accuracy and the upper bar for run-time. Measuring run-time of a single C function was done using a statistical method used in lab time-interval instruments of averaging large number N of low-resolution T_0 pulse counts. In a relatively steady time interval, we can expect up to two values for each count: x and x+1, counted P and Q times respectively: N=P+Q. The sample size and standard deviation determine the uncertainty of the final result[48]. The worst case is when P=Q or Q/N=1/2: In the Tcl-provided library, the time interval command Tcl_GetTime provides counts in resolution of 1μsec. A sample size of 10000 gives worst case σ=1000nsec/2*100=5nsec and 3σ=15nsec.

Hardware specifications for the test machine used to compare SPICE and Γ run times are given in Table 8.

Table 8: Test Machine's Hardware Specifications

Resource	Value
CPU	AMD Phenom(tm) II X4 955
#Cores	4
Freq.	2.5 GHz
RAM	12 GB
OS	CentOS 6

6.4.1.2. Results

The BSIM batch of equations was instrumented first to get a base level for comparison. The BSIM3v32 and BSIM4v5 took an average of 28μsec and 50μsec to complete a query respectively (Table 9).

The first interpolation method that was tested was a full Lagrange interpolation of 6D entries. The initial coding of the interpolation used pure C, leaving all conditions and loops to run time. This method yielded results in 8μsec per query. This was not a sufficient improvement, considering that a BSIM query yields all the operational parameters of a transistor, while a LUT query returns only one. To achieve an operating point, at least three parameters: I_{DS}, g_m, and g_o have to be looked up in each iteration, which makes the initial query time improvement negligible.

The next improvement was to change entry locating phase of the query from linear to binary search. This change reduced the 6D average query time to 6.1μsec.

The biggest improvement was achieved by converting all the conditions and loops to compilation time. The Tcl preprocessor generated a C code that had a trace execution of all the

Lagrange interpolation in a hypercube, without any branching and in-place. This reduced the query time to 1.8µsec. A LIT version of the same 6D table was queried in 1.0µsec.

Each query is composed of two stages: entry locating and interpolation. The entry locating stage had 3 types:

1. Linear search – used only for initial prototype
2. Binary search – used for most tables (~600nsec)
3. Direct conversion of coordinates – used for uniformly sampled tables (~150nsec)

The interpolation stage run time is more influenced by the array's dimensionality (Figure 42), since its Lagrange algorithm is of $O(2^N)$ time complexity. A linear LIT interpolation is of O(N) time complexity.

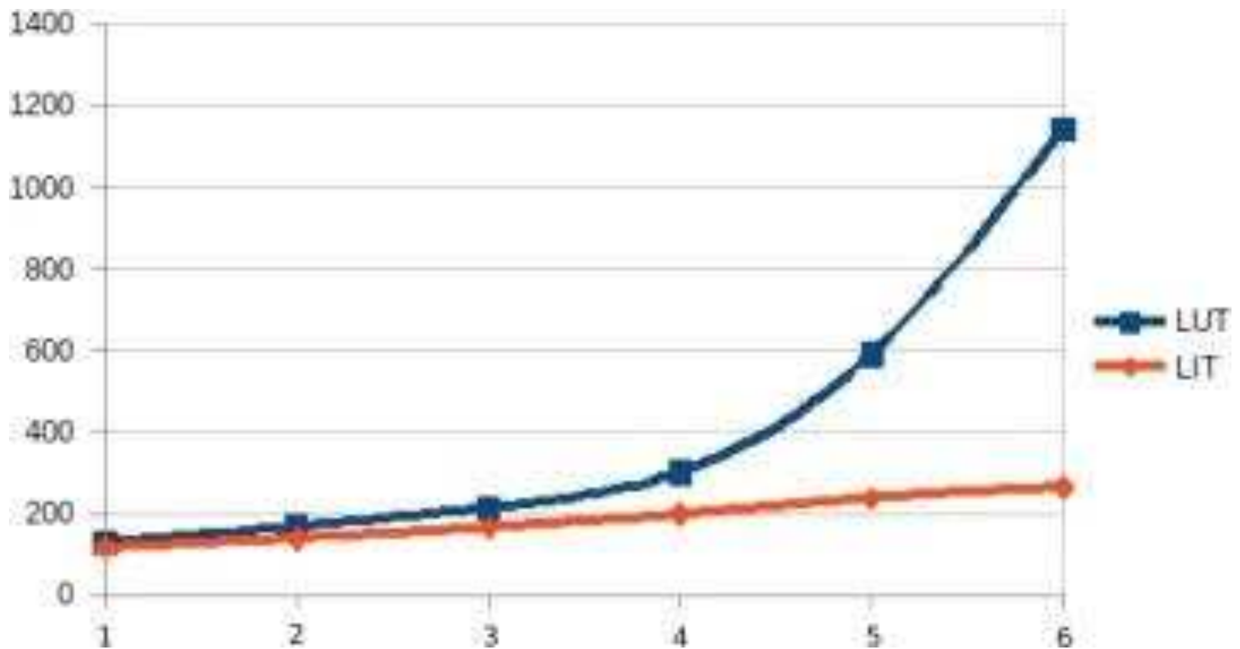


Figure 42: Interpolation Time [nsec] vs Dimensionality

The final LUT version used in the Γ circuit-level evaluations are 4D, neglecting wide channel effects, linking temperature with corners, using uniformly sampled array, and interpolating in the full Lagrange method. This reduced the query time to 450nsec per parameter when queried separately. In a composite interpolation (5.1.5) 3 parameters were query with an additional I_{DS} accuracy improving stage were completed in 1.1μsec, which translates to 370nsec per parameter.

Table 9: Comparison of Query Times per Model Type

Model Type	Description	Query Time
BSIM4v5	Model used in 40nm SPICE runs	50μsec
BSIM3v32	Model used in 180nm SPICE runs	28μsec
6D Full	Initial prototype of LUT	8μsec
6D Full, binary	Improved entry locating	6.1μsec
6D Full, trace	Removed branching by using Tcl preprocessor	1.8μsec
6D Linear, trace	Used Tcl preprocessor and LIT	1.0μsec
4D Full, uniform	Uniformly sampled 4D array	450nsec
4D composite	3 OP parameters queried together	1.1μsec

6.4.2. Circuit Level

6.4.2.1. Offline Activities

The first offline task regarding a topology is compilation. This requires nodal-analysis, algebraic manipulation and generation of performance-property equations. Typical compilation time ranges from a few tens of seconds to a few hundreds of seconds depending on circuit complexity.

The next offline task is to populate an initial PAT with ~10^5 circuits that span the performance front of the topology. Typically, populating PAT takes between a minute to tens of minutes depending on the population size.

Once a PAT is initially populated in the server's data area, the topology can be mapped and further resolution can be added to performance corners of interest. Overall offline activities per topology, including preparation of netlist and schematic representation, can be finished in about an hour and result in about 100MB of data per topology+technology pair.

6.4.2.2. Circuit Evaluation Time

One of the key innovations of the proposed design system is rapid evaluation of circuit performance without sacrifice in accuracy. Table 10 illustrates the run time difference between Γ and SPICE on determining the operating point for the folded Cascode amplifier circuit collected over 1000 circuit evaluations. The "Min" and "Max" numbers show the shortest and longest evaluation times respectively. "Med" is the median evaluation time. It is clear that Γ is significantly faster than SPICE while maintaining the SPICE level accuracy as it is demonstrated by the results in **Error! Reference source not found.**.

SPICE does not have built-in commands for evaluating all listed performance metrics for a given circuit, but rather has a number of simulation commands that can lead to scripted or manual post-processing. Therefore, the .op command was chosen for comparison, as it has the closest parallel in Γ, the Circuit Evaluator. ΓCE includes operating-point loop, but also calculations for all the performance parameters. It completes all the steps needed to evaluate a circuit in 1/10 of the typical time SPICE takes to just analyze the operating point.

Table 10: Run time of ΓCE and NGSPICE .op [msec]

Tool	Min	Max	Med	Average	σ
SPICE	23.8	387.8	52.8	108	104
Γ	4.5	14	4.9	5.7	1.8

Both execution times for SPICE and Γ were measured on the server-side machine, without any network effects. In the online SaaS environment, the actual response time may be affected by network traffic. However, the significant performance advantage by Γ will allow Γ's server to handle many client requests to improve the overall response time for remote users.

6.4.2.3. PAT Insertion Time

The time complexity for inserting a point into the PAT is O(N), with N being the number of circuits already in the PAT. This is because every inserted point can potentially be compared with all stored ones before it is retained. However, every insertion has a probability to be discarded as dominated before it visits all the stored points. Therefore a cone-shaped scatter-plot of insertion time vs. size is expected. Figure 43 shows the insertion time as a function of PAT population size. The bimodal distribution of the PAT insertion time can be attributed to two classes of operations: one without requiring prolonged evaluations for eligibility, and the other involving a series of computations for eligibility, including operations to delete other dominated entries in the PAT.

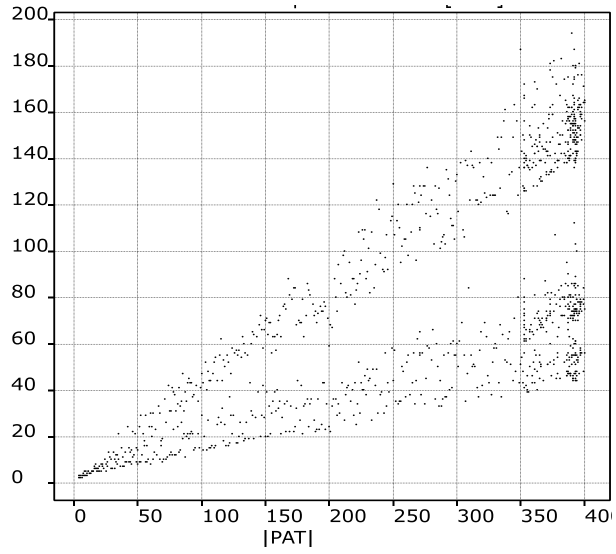


Figure 43: PAT Insertion Time per Size

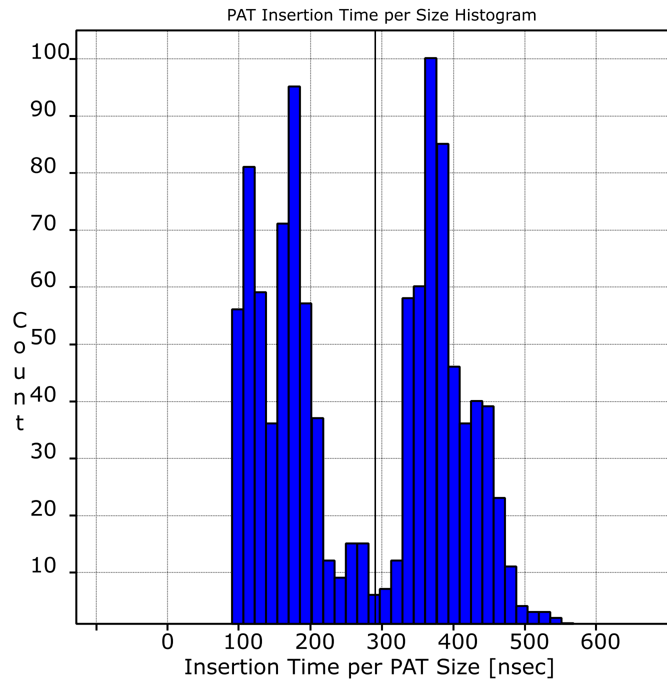


Figure 44: Ratio between PAT Insertion Time and Size

Figure 44 shows the histogram of PAT insertion time, after it is normalized per number of points retained in the PAT prior to the insertion. This gives the distribution of the slope per

point in Figure 44. The average insertion time is therefore ~300nsec (marked by wide vertical line in Figure 44) per pre-stored circuit.

6.5. Γ Website Usage Example

To demonstrate the tool's usage, a flow that may be used by designers to optimize the circuit size (Figure 41.4) for a given specification is shown here. The primary specifications for the circuit are:

1 Technology: 40nm

2 DC Gain >60dB

3 Bandwidth > 5MHz

4 Total Noise $<100\mu V^2$

5 Main optimization objective: Area

Secondary Specifications:

6. CMRR>30dB
7. PSRR>50dB
8. $PM>60^O$
9. Corner Frequency<30KHz

6.5.1. Select Topology and Enter Specification

After login, the specified technology and topology are selected from the drop menus. A default A_{DC}/BW front appears and the thresholds are typed in. To allow trade-off, a slightly relaxed specification values are specified: 57dB and 4.5MHz for gain and BW

respectivley, while the noise parameters are waived for the moment, so we can start the search from a close enough circuit that can be improved to meet the whole specification. The updated perato front from Γ, shows existing circuits already in the PAT that meet the relaxed specification (Figure 45).

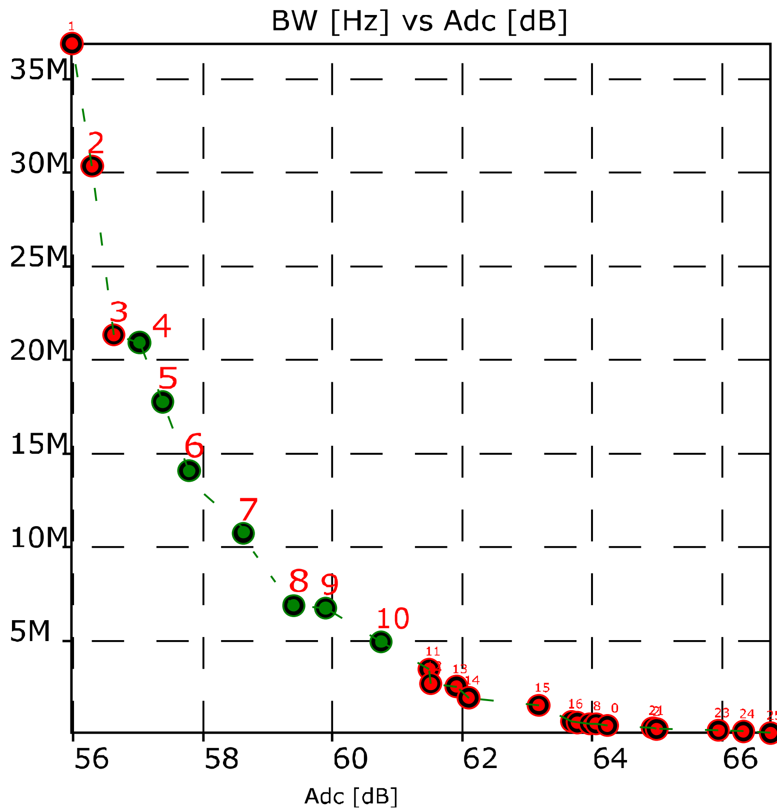


Figure 45: PAT circuits that meet gain and BW

Circuits 9 and 10 of that front are selected. Each of them violates the original specification. Circuit 10 meets gain requirement, but fails on BW and vice-versa for Circuit 9. Both circuits violate the noise requirement.

However, circuit 9 has significant BW slack (6.7MHz), which can be traded more easily for gain and noise and therefore it is chosen to be the pivot circuit for the next search step.

6.5.2. Pick an Initial Circuit

Circuit 9 has the following performance metrics:

DC Gain=59.8dB

BW=6.7MHz

Total Noise=186μV^2

6.5.3. Search for Gain-Valid Circuit

Increasing the gain of Circuit 9 can be done by upsizing the input transistors. The search for better gain is done by plotting a tradeoff map of the gain with respect to changes in Ln and Wn of the input transistors (Figure 46).

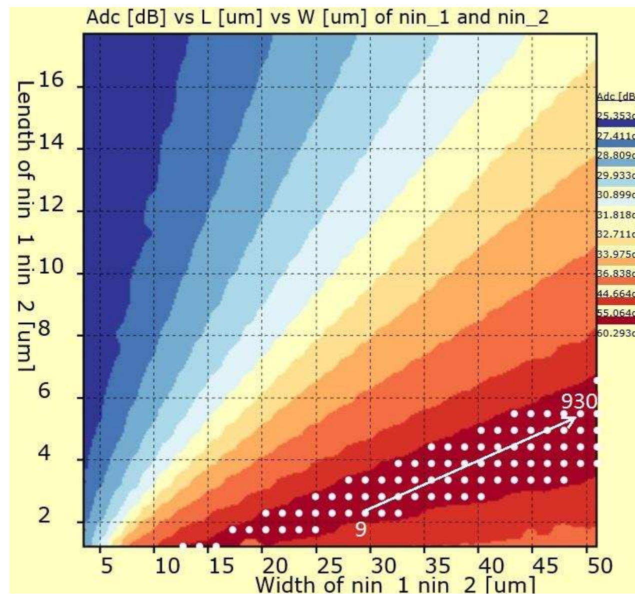


Figure 46: DC Gain vs input transistor length and width

To increase the gain and reduce noise, while keeping the BW, the next circuit is selected from the white dots[3], in the center of the deepest red area and as wide channel as possible (see marker 930 in Figure 46). The BW only drops to 6.3MHz and the total noise drops to 177μV^2.

6.5.4. Reduce Noise to Meet Spec

The next step is to reduce the noise to the specified level of 100μV^2. Another tradeoff map centered on Circuit 930 is generated using the current-mirror p-channel transistors' sizes to trade BW with noise (Figure 47). The initial circuit showed a relative noise contribution of ~40% from the two p-channel transistors (Table 11), which makes them good candidates for noise reduction.

Table 11: Breakdown of Noise Contribution per Transistor

Transistors	% Noise Contribution (each)
p_1, p_2	19.7%
nin_1, nin_2	15.1%
n_tail, n_ref	2.2%
n_out	1.1%
p_out	24.9%

[3] The original heat maps' green markers were accented in white for print quality.

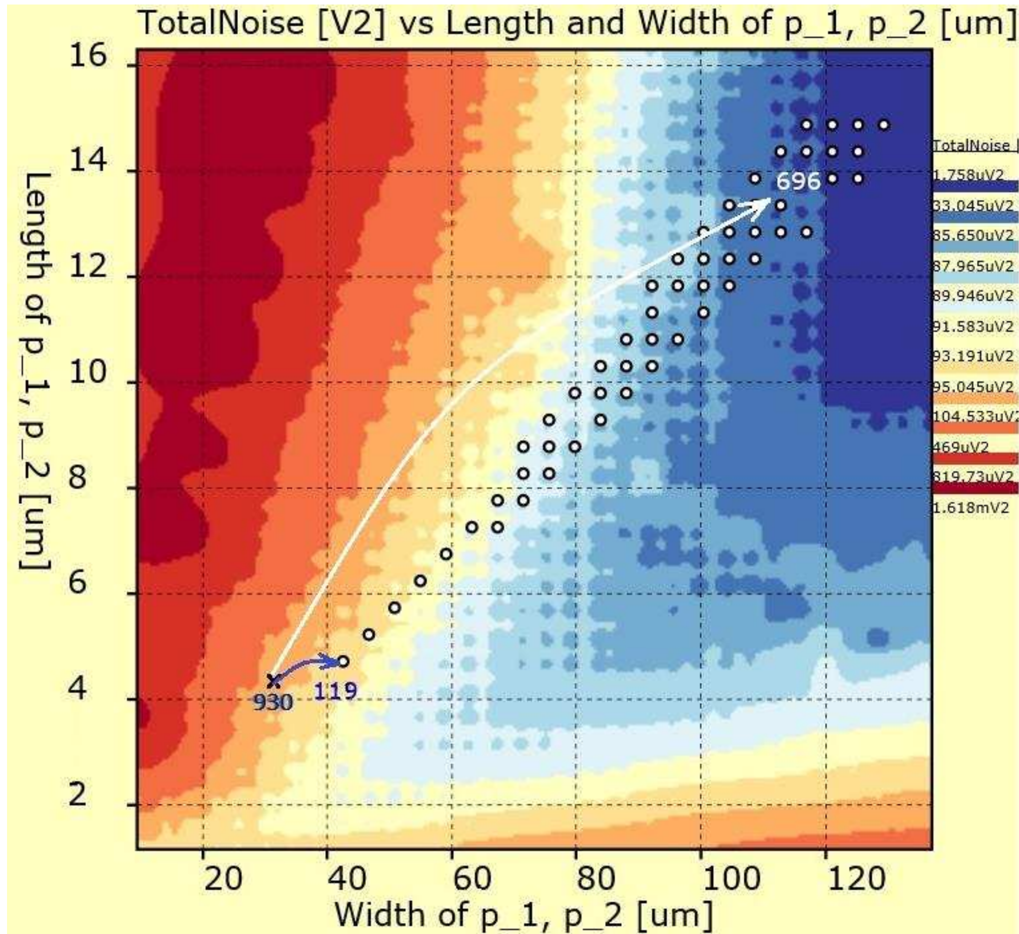


Figure 47: Circuit 696 trades BW for less noise

Circuit from the "cooler" parts of the heat map, represented by Circuit 696 in Figure 47, lose all the BW slack and meets the noise requirements at 86μV^2 . However, they have significant penalty on total area. The noise slack of 14μV^2 means that the circuit is over designed and a better solution can be more area-optimal. Circuit 119 (in blue, Figure 47) is a better solution, since it is in the color region corresponding to the required noise and closer to the origin, which means it has smaller area. Circuit 119 is the final circuit candidate, with 94.9 μV^2 total noise. The final circuit (Circuit 119) was simulated on Cadence® Virtuoso® and the following results were collected and compared with Γ's (Table 12).

Table 12: Γ and Virtuoso Final Performance Metrics

Property	Γ	Virtuoso	Rel. Err.
DC Gain	60.2dB	60.8dB	1%
BW	5.1MHz	5.5MHz	7.8%
Total Noise	94.9μV^2	104μV^2	9.4%
Corner Freq.	25KHz	27KHz	8%
PM	83°	81°	3%
CMRR	39.8dB	40dB	2.3%
PSRR	59.1dB	59.8dB	8%
Est. Area	8100$(\mu m)^2$		

Through a set of steps (Table 13), the results here show that Γ provides designers with a powerful, and yet simple, process to converge to an optimal design solution for a given specification. Further fine tuning may be necessary after verifying Γ's final results with SPICE.

Table 13: Search Path Summary

Circuit #	Gain	BW	Noise
9	59.8dB	6.7MHz	186μV^2
930	60.4dB	6.3Mhz	177μV^2
696	60.2dB	5MHz	86μV^2
119	60.2dB	5.1MHz	94.9μV^2

7. Conclusions

An analog design system, Γ, is presented in this dissertation. Using the novel modeling techniques, Γ combines the speed of symbolic evaluation of circuits in consideration and the accuracy of SPICE-class circuit level simulations. Instead of using the trial-and-error approach in many of the existing analog circuit design flows, Γ provides designers with extensive features for rapid and reliable design tradeoffs among numerous design candidates through Pareto front analysis and design tradeoff maps during the design cycle. These capabilities will ultimately help a design to converge faster and improve the quality of the final design.

The novel features incorporated in Γ enable the design system to achieve desired performance and accuracy. The LUT approach to modeling transistor-level behavior accelerates parameter lookup in DC analysis by a factor of 30-50, compared to BSIM calculation during SPICE simulations. Interpolated LUT DC parameters are typically within 1% and AC+noise parameters are typically within 10% of their BSIM calculation, with a reasonable memory investment of ~100MB. On circuit level, the system produces a full circuit evaluation in typically 1/10 of the time it takes a traditional SPICE simulator to finish just the operating point part of the evaluation. Follow-up evaluations of small size changes in 1D and 2D sweep analyses are even shorter, with a typical run time of 1/50 of the SPICE .op analysis. Errors of circuit-level analysis are under 5% for low-frequency performance metrics and 10% for AC and noise. …

Γ's novel features in speed and accuracy also make it ideal for providing the SaaS features through its web functions. Breadth-first analysis strategy through archived, as well as online, Pareto candidates of viable circuits is well suited for SaaS applications, because the entire

design process can be broken into a series of short queries and much of the computation effort can be archived in a shared database that serves all present and future users and sessions. The SaaS frontend is capable of providing queries with an average latency of 10sec for Apache-based CGI calls and only 1-2sec, when the system serves HTTP calls. The difference between the two modes stems from the fact that the latter retains binary data (.os, LUT, and PAT) in RAM while the former has to perform initialization tasks on each query.… Enabling of SaaS features for Γ is a significant step towards providing faster and updated design tool delivery at very low cost. With increasing recognition of the power of cloud computing, Γ certainly moves the delivery, maintenance, quality, and usability of future analog design tools in the right direction.

Γ currently makes no attempt to reduce the amount of slacks among any performance parameters in the solutions, because the process of reducing slacks of any performance parameters often results in worsening of other performance parameters in often tightly constrained designs. Without knowing designer's intent, it is difficult for Γ to reach correct decisions during the search process. Rather, Γ collects and stores all relevant data and presents them to designers to facilitate useful designer manual interventions to incrementally capture design intent during the design process. This is illustrated by the design example in Section 6.5. Allowing designer manual interventions during the design process is one of the key features that distinguish Γ from the previous attempts in developing analog circuit design tools. It provides a better path for faster design convergence.

While some of the performance parameters implemented in Γ's circuit evaluator are natural for static, small-signal analysis (e.g. CMRR, BW and other parameters can be estimated from that analysis, such as settling time from pole/zero analysis), other parameters cannot be

inferred directly from component dimensions and operating point. Such parameters, such as total harmonic distortion, can only be derived from classic simulation techniques. While the existing Γ features are capable of meeting the majority of design needs for analog circuits, further development of Γ may include alleviate these limitations, for example, by using piece-wise linear LUTs with predictive distortion models for output distortion calculation. Furthermore, additional analyses and visualization features may include:

1. Combined objective through cost function – multi-objective analysis keeps the data open for follow up application-specific choice. The final choice can be illustrated by using cost-function weights. Visualization of circuit choice per weight assignment can be implemented on the browser side using JavaScript.
2. Load support – re-evaluate circuits in real time with a given capacitive load for more realistic performance values.
3. Integration of support circuits – allow better integration of sizing algorithms to include support circuits, such as bias circuits, to achieve better global results.
4. Multi-topology search – instead of limiting the search to the selected topology template, Γ perform search across different topology templates to find better solutions.

8. References

[1] Mack, Chris. "The Multiple Lives of Moore's Law." Spectrum, IEEE 52.4 (2015): 31-31.

[2] WikiCommons, user wgsimon

[3] Graeb, Helmut. "ITRS 2011 analog EDA challenges and approaches." Proceedings of the Conference on Design, Automation and Test in Europe. EDA Consortium, 2012.

[4] Annema, Anne-Johan, et al. "Analog circuits in ultra-deep-submicron CMOS." Solid-State Circuits, IEEE Journal of 40.1 (2005): 132-143.

[5] William Nye, David C. Riley, Alberto Sangiovanni-Vincentelli, Andre L. Tits., "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on (Volume:7 , Issue: 4), Pages: 501 – 519, April 4, 1988

[6] Degrauwe, Marc GR, et al. "IDAC: An interactive design tool for analog CMOS circuits." Solid-State Circuits, IEEE Journal of 22.6 (1987): 1106-1116.

[7] Georges G.E. Gielen, Herman C.C. Walscharts, Willy M. Sansen , "Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing" Solid-State Circuits , Pages: 252 – 255, June 1990

[8] Rutenbar, Rob. "Analog design automation: Where are we? Where are we going?." Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993. IEEE, 1993.

[9] Barcelona Design Unveils Revolutionary Analog Circuit Solution, prnewswire.com/prn/11690X15251303, April 8, 2002

[10] Costello's analog automation pioneer, Barcelona, to fold, EETIMES doc_id: 1217996, 3/4/2005.

[11] Allan, Alan, et al. "2001 technology roadmap for semiconductors." Computer 35.1 (2002): 42-53.

[12] Carl Martin Allwood, Tomas Kalen, “Usability in CAD – a Psychological Perspective”, International Journal of Human Factors in Manufacturing Volume 4, Issue 2, pages 145–165, 1994

[13] Jakob Nielsen, “Usability Engineering”, Morgan Kaufmann, San Francisco, Chapter 5.5, 1993

[14] Chenming Hu, “BSIM Model for Circuit Design Using Advanced Technologies”, VLSI Circuits, 2001. Digest of Technical Papers. 2001 Symposium on, pages 5-10

[15] L. Nagel, “SPICE2: A computer program to simulate semiconductor circuits,” Electronics esearch Lab., Univ. Calif., Berkeley, Memo UCB/ERL M520, May 1975.

[16] Ngspice Users Manual Version 26 (Describes ngspice-26 release version) Paolo Nenzi, Holger Vogt January 11, 2014

[17] Gielen, Georges GE, and Rob Rutenbar. "Computer-aided design of analog and mixed-signal integrated circuits." Proceedings of the IEEE 88.12 (2000): 1825-1854.

[18] Ho, Chung-Wen, Albert E. Ruehli, and Pierce A. Brennan. "The modified nodal approach to network analysis." Circuits and Systems, IEEE Transactions on22.6 (1975): 504-509.

[19] Borchers, Carsten, Lars Hedrich, and Erich Barke. "Equation-based behavioral model generation for nonlinear analog circuits." Proceedings of the 33rd annual Design Automation Conference. ACM, 1996.

[20] Kwang S. Yoon, Phillip E. Allen “An adjustable accuracy model for VLSI analog circuits using lookup tables” Analog Integrated Circuits and Signal Processing Volume 1, Issue 1 , Pages: 45-63, 1991

[21] Yishai Statter, Tom Chen, “A Novel High-Throughput Method for Table Look-Up Based Analog Design Automation”, Integration, the VLSI Journal (2016), pp. 168-181

[22] Á. B. Tadej Tuma, "Circuit Simulation with SPICE OPUS," in NUTMEG scripting language, Springer, 2009, pp. 177-255.

[23] Guide, Litho Physical Analyzer User. "Cadence." San Jose, CA, USA (2004).

[24] Kim, Jang Dae, et al. "Algorithmic reactive testbench for analog designs." U.S. Patent No. 7,853,908. 14 Dec. 2010.

[25] Benhala Bachir , Ahaitouf Ali , Mechaqrane Abdellah, “Multiobjective Optimization of an Operational Amplifier by the Ant Colony Optimisation Algorithm” Scientific & Academic Publishing, Electrical and Electronic Engineering 2012, Vol 2, Pages 230-235

[26] R.A. Vural, , T. Yildirim, “Analog circuit sizing via swarm intelligence”, AEU - International Journal of Electronics and Communications, Pages 732-740, September 2012

[27] John R. Koza, Riccardo Poli “Ch. 5, GENETIC PROGRAMMING”, “Search Methodologies Introductory Tutorials in Optimization and Decision Support Techniques”, Pages 127-164 Springer 2005.

[28] David M Binkley “Tradeoffs and Optimization in Analog CMOS Design”, University of North Carolina , Pages 25-26, 2008

[29] CircuitLab, online circuit editing and simulating. https://www.circuitlab.com/docs/the-basics/

[30] PartSim, an online simulator http://www.partsim.com/

[31] Logic Lab, online tool for simple logic simulations. http://www.neuroproductions.be/logic-lab/

[32] S. Donnay et al., “Using top–down CAD tools for mixed analog/digital ASICs: A practical design case,” Kluwer Int. J. Analog Integrated Circuits Signal Processing (Special Issue on Modeling and Simulation of Mixed Analog–Digital Systems), vol. 10, pp. 101–117,

[33] June–July 1996.R. Harjani, R. Rutenbar, and L. R. Carley, “OASYS: A framework for analog circuit synthesis,” IEEE Trans. Computer-Aided Design,vol. 8, pp. 1247–1265, Dec. 1989

[34] J. Crossley, A. Puggelli, H.-P. Le et al., “BAG: A designer-oriented integrated framework for the development of AMS circuit generators,” IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 74–81, 2013.

[35] Scheible, Juergen, and Jens Lienig. "Automation of Analog IC Layout–Challenges and Solutions." Proceedings of the 2015 Symposium on International Symposium on Physical Design. ACM, 2015.

[36] Cadence, "Virtuoso NeoCircuits Datasheet," 2 2007. [Online]. Available: http: // www. cadence. Com /products /custom_ic.

[37] Spectre, Cadence Virtuoso. "Cadence home page."

[38] Simulator, Cadence Virtuoso UltraSim Full Chip. "Datasheet." 1-4.

[39] Business Wire, "Business Wire," 5 February 2004. [Online]..

[40] Laker3 Custom Design Tools. “Synopsys Solvenet”

[41] Wan, Bin, and XinGang Wang. "Overview of commercially-available analog/RF simulation engines and design environment." Solid-State and Integrated Circuit Technology (ICSICT), 2014 12th IEEE International Conference on. IEEE, 2014.

[42] Babylonian Talmud, Damages, Baba Metzi'a, 28a, תלמוד בבלי, נזיקין, בבא מציעא דף כ"ח, עמוד א',

[43] Statter, Yishai, and Tom Chen. "Γ (Gamma): A SaaS-enabled fast and accurate analog design System." Integration, the VLSI Journal 55 (2016): 67-84.

[44] Pareto, Vilfredo. *The mind and society*. Рипол Классик, 1935.

[45] Ousterhout, John K. Tcl: An embeddable command language. University of California, Berkeley, Computer Science Division, 1989.

[46] J. K. Ousterhout. Scripting: Higher level programming for the 21st century. IEEE Computer , March 1998.

[47] Parker, John. "A comparison of design methodologies." *ACM SIGSOFT Software Engineering Notes* 3.4 (1978): 12-19.

[48] Millman, Jacob. "A useful network theorem." *Proceedings of the IRE* 28.9 (1940): 413-417.

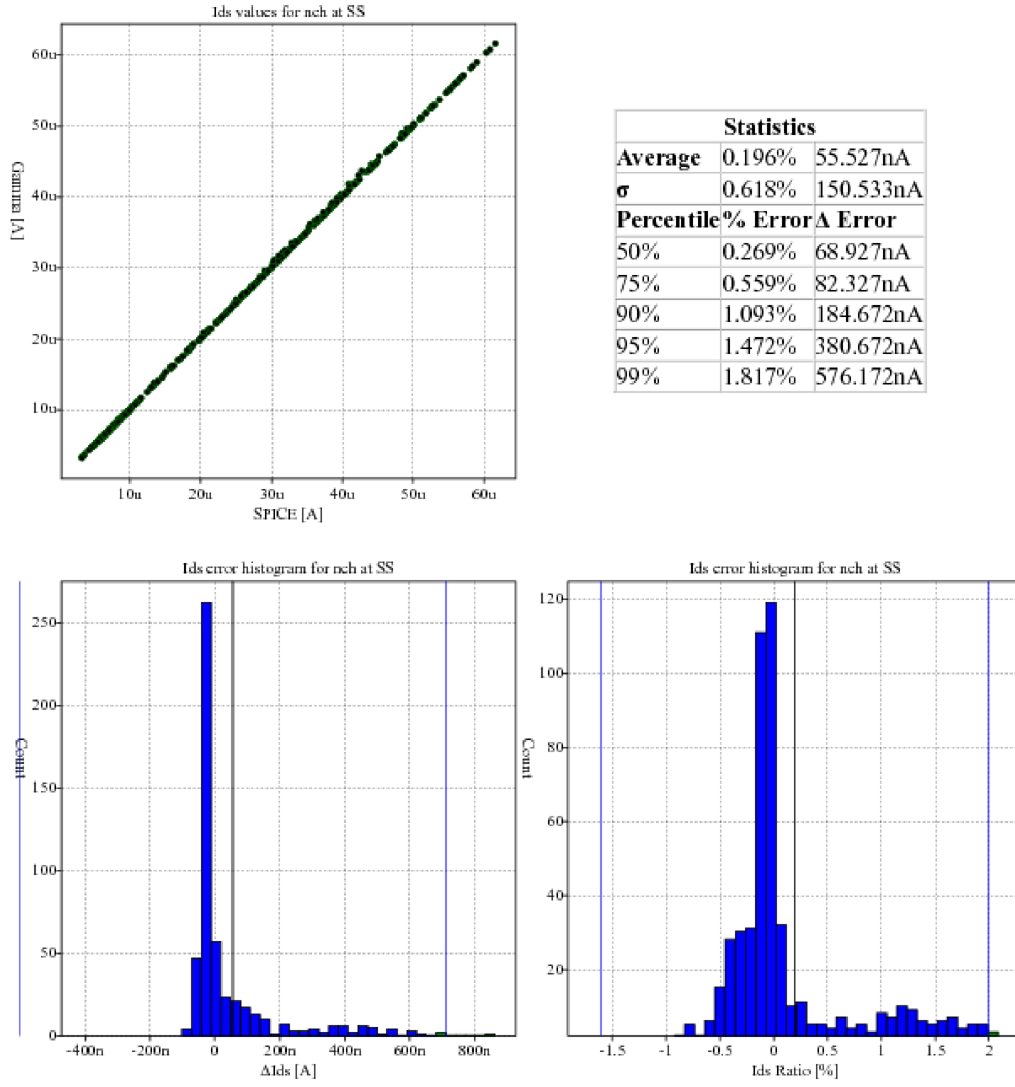
[49] Sorden, James L. "A New Generation in Frequency and Time Measurements." Hewlett Packard J 25 (1975) page 15.

9. Appendices

A. Γ/SPICE Matching Results

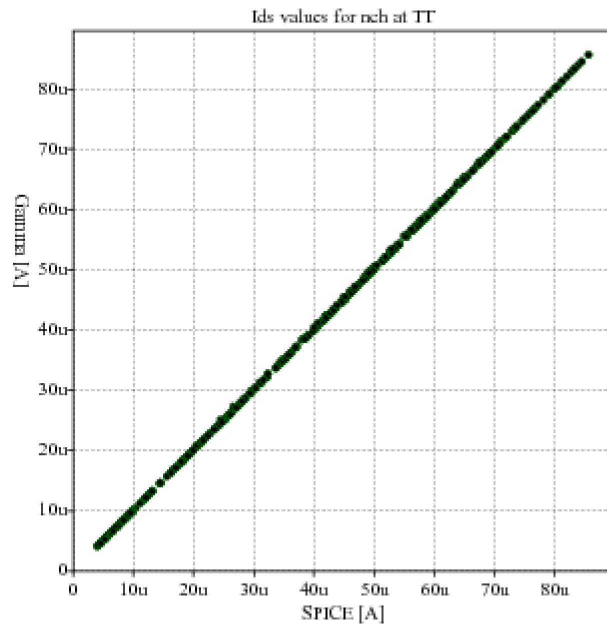
This appendix contains experimental data comparing results from LUT and SPICE. The purpose of the comparison is to determine the accuracy of the LUT results. The comparison data is organized into two sections: comparison of single transistor instances and comparison of circuit template instances. Each figure listed in this appendix is intended to capture how well the results match. The degree of matching is shown in a table in each figure, where the average error and its standard deviationare given for two types of errors: relative, on the left column and nominal, on the right. Since the distribution of errors is not normal, the table lists maximal errors for key precentiles of the sample population: 50%, 75%, 90%, 95% and 99%.

A. 1. Transistor Level

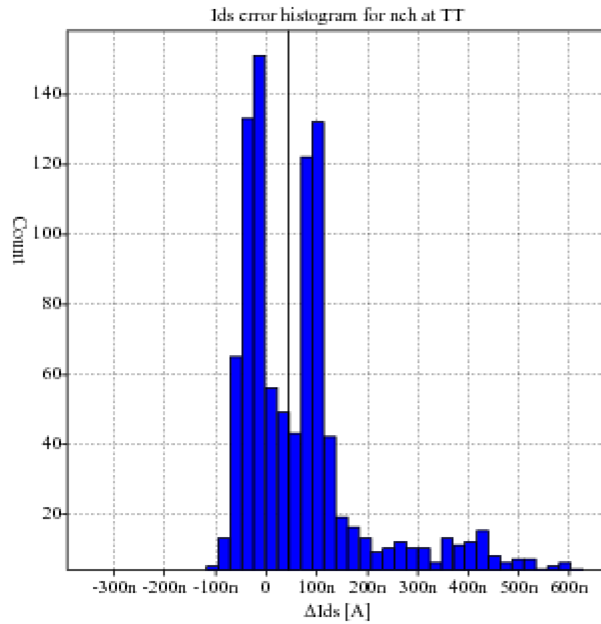


Statistics		
Average	0.196%	55.527nA
σ	0.618%	150.533nA
Percentile	% Error	Δ Error
50%	0.269%	68.927nA
75%	0.559%	82.327nA
90%	1.093%	184.672nA
95%	1.472%	380.672nA
99%	1.817%	576.172nA

Figure A-1: 40nm NCH SS I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	0.02%	43.135nA
σ	0.394%	125.08nA
Percentile	**% Error**	**Δ Error**
50%	0.123%	63.535nA
75%	0.382%	84.345nA
90%	0.661%	186.364nA
95%	0.79%	299.664nA
99%	1.137%	420.764nA



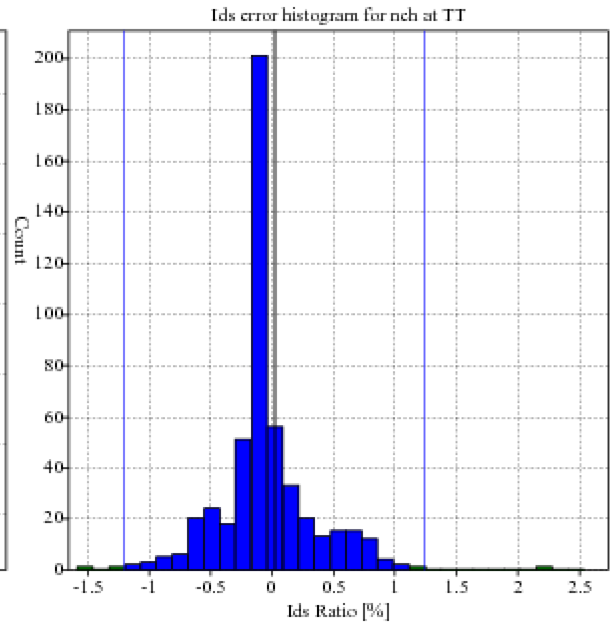
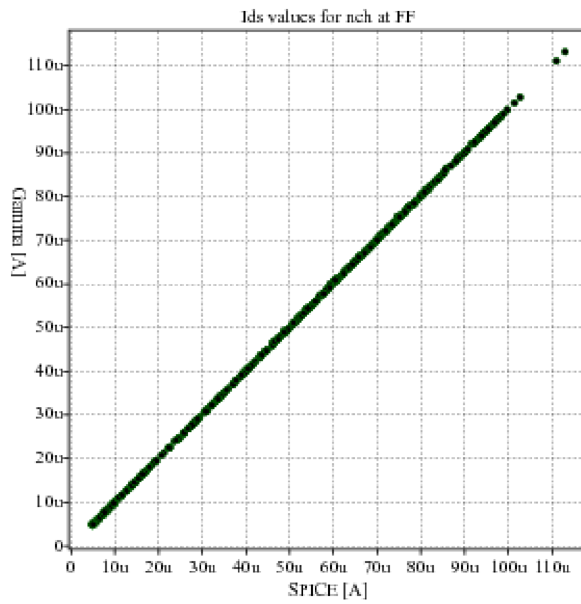
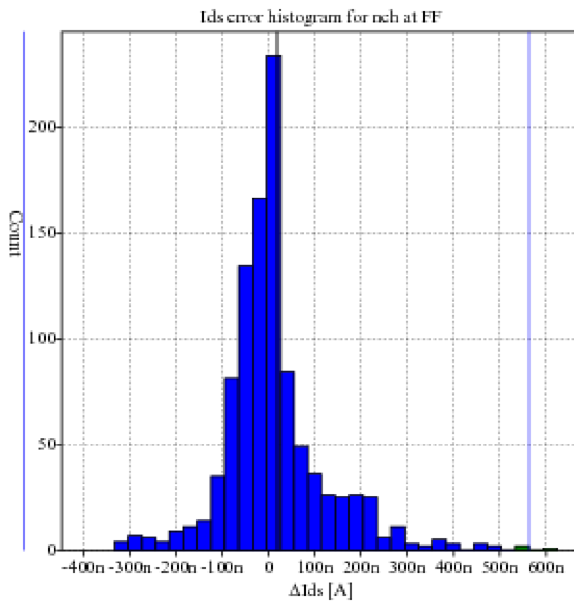


Figure A-2: 40nm NCH TT I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.198%	18.029nA
σ	0.85%	133.686nA
Percentile	**% Error**	**Δ Error**
50%	0.207%	58.529nA
75%	0.441%	103.17nA
90%	0.819%	223.379nA
95%	1.520%	290.469nA
99%	4.430%	467.87nA



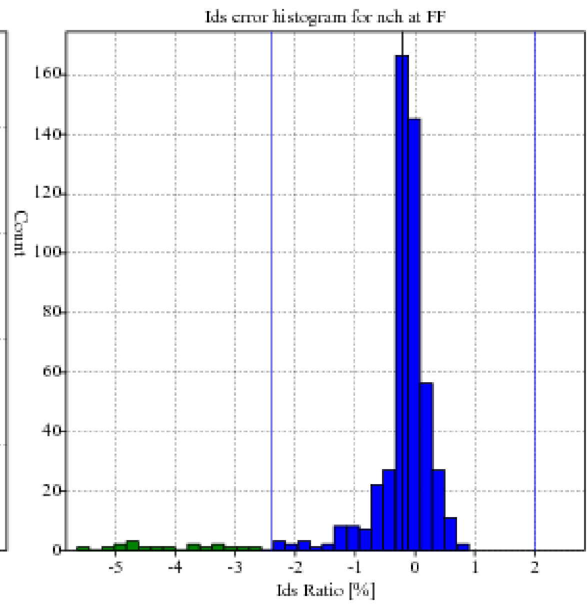
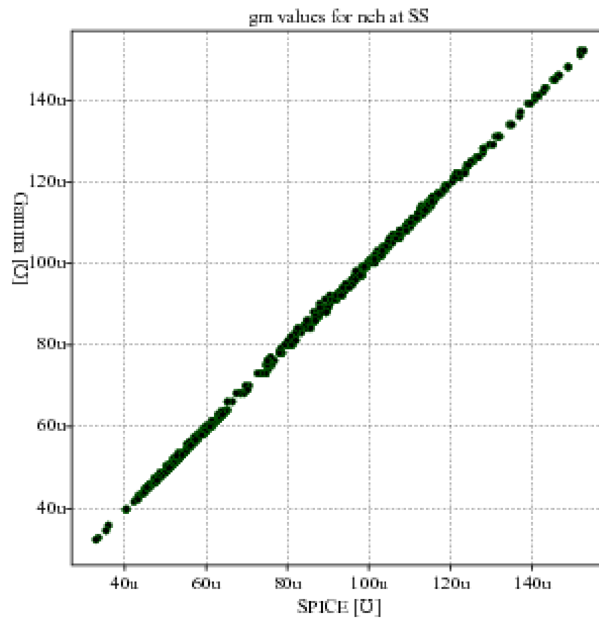
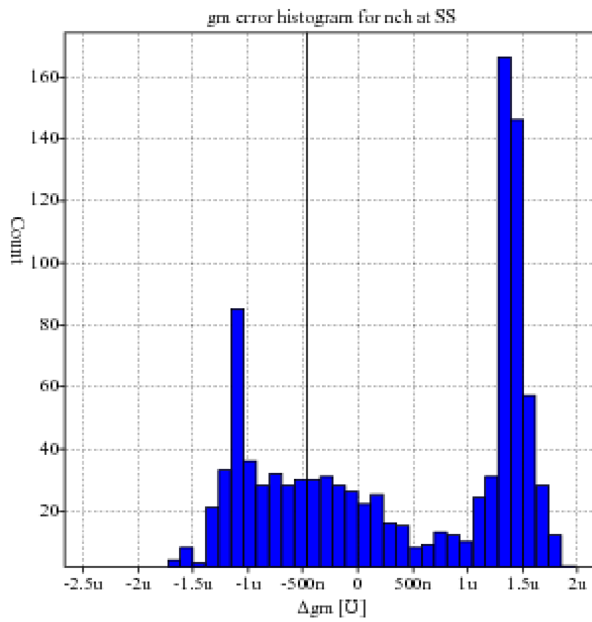


Figure A-3: 40nm NCH FF I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.694%	-459.101n℧
σ	0.954%	642.243n℧
Percentile	**% Error**	**Δ Error**
50%	0.748%	517.401n℧
75%	1.147%	663.898n℧
90%	1.488%	961.101n℧
95%	1.732%	1.161u℧
99%	2.373%	1.785u℧



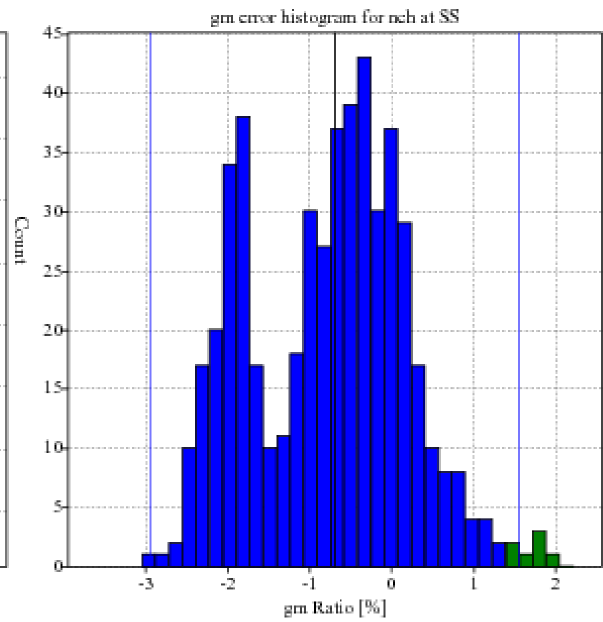
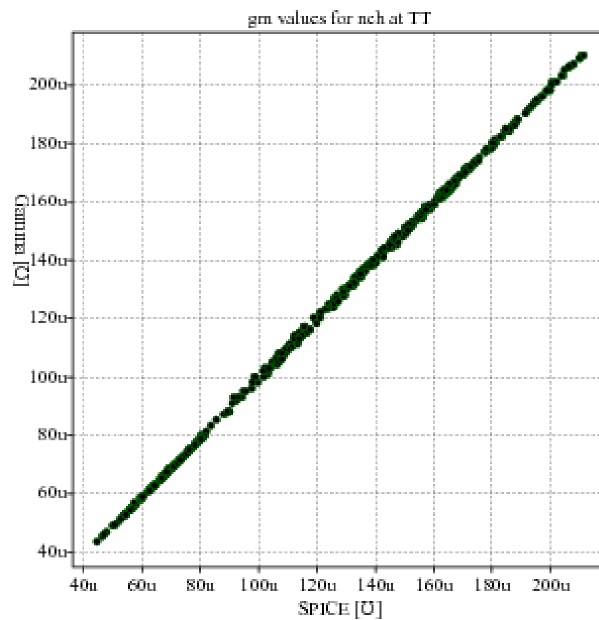
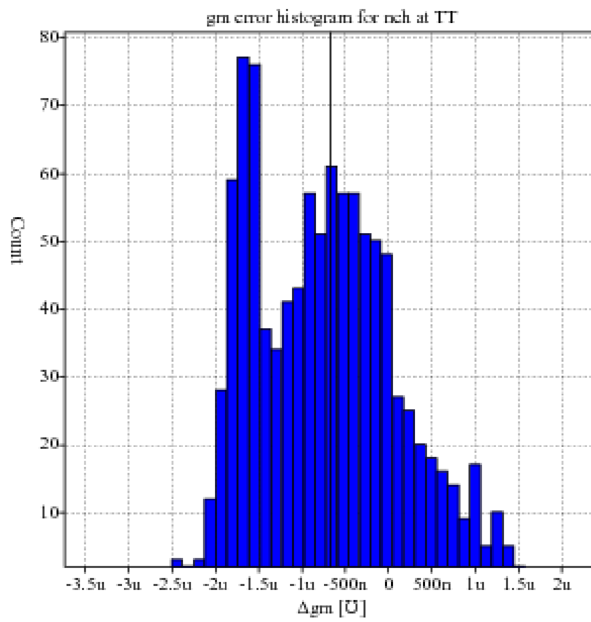


Figure A-4: 40nm NCH SS g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.758%	-660.587n℧
σ	1.065%	905.159n℧
Percentile	**% Error**	**Δ Error**
50%	0.77%	748.412n℧
75%	1.363%	1.046u℧
90%	1.677%	1.425u℧
95%	1.976%	1.705u℧
99%	2.344%	2.067u℧



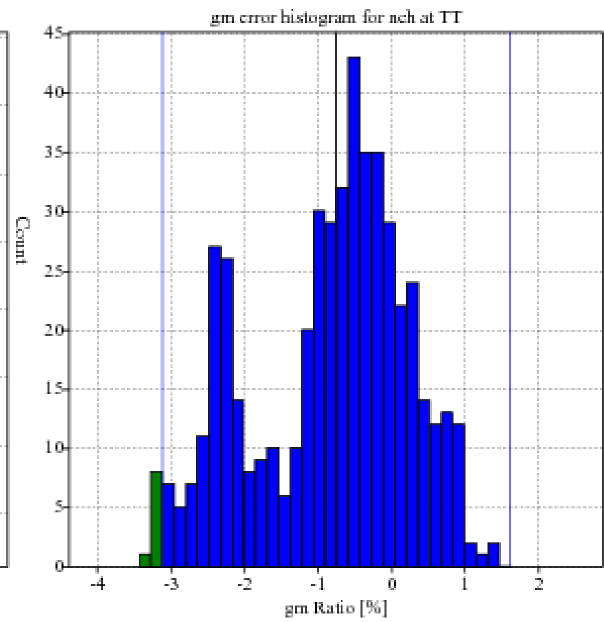
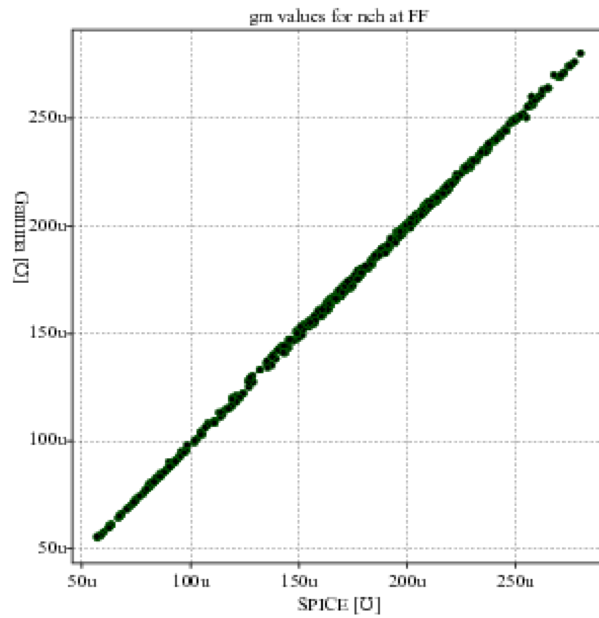
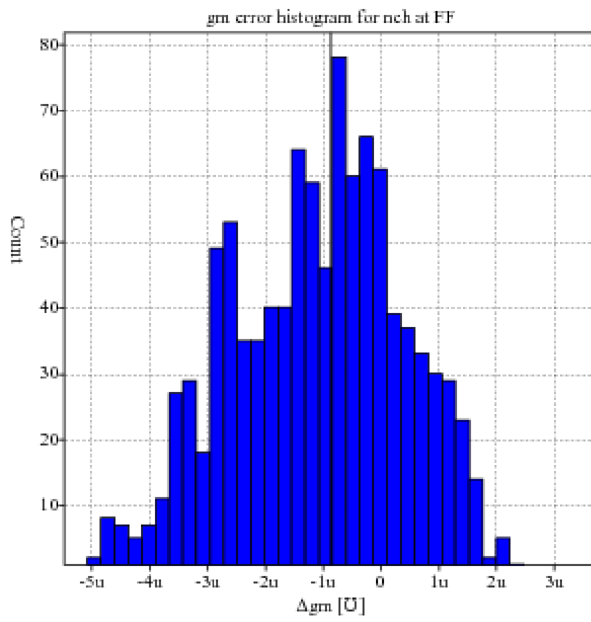


Figure A-5: 40nm NCH TT g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.792%	-874.437n℧
σ	1.304%	1.352u℧
Percentile	**% Error**	**Δ Error**
50%	0.801%	1.079u℧
75%	1.495%	1.683u℧
90%	2.143%	2.055u℧
95%	2.710%	2.326u℧
99%	3.559%	3.042u℧



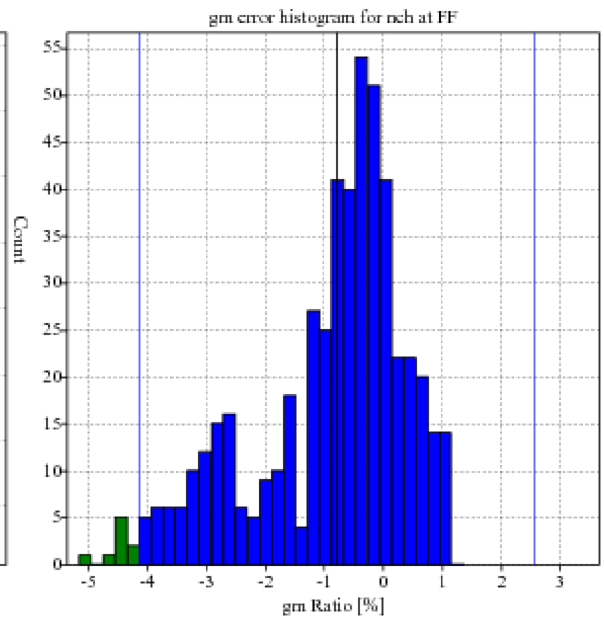
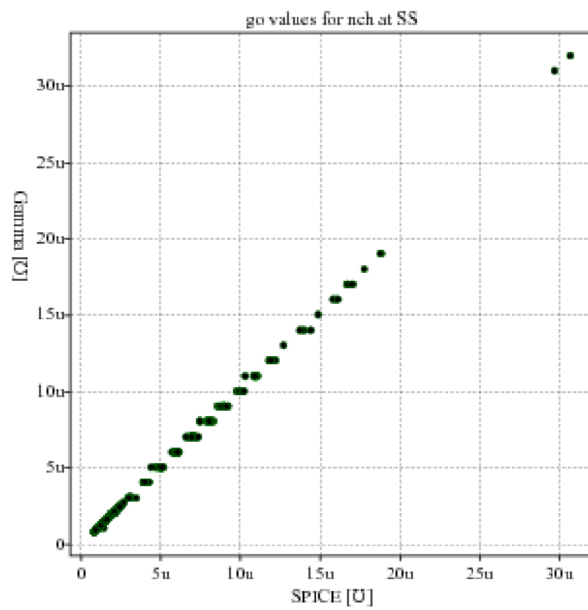
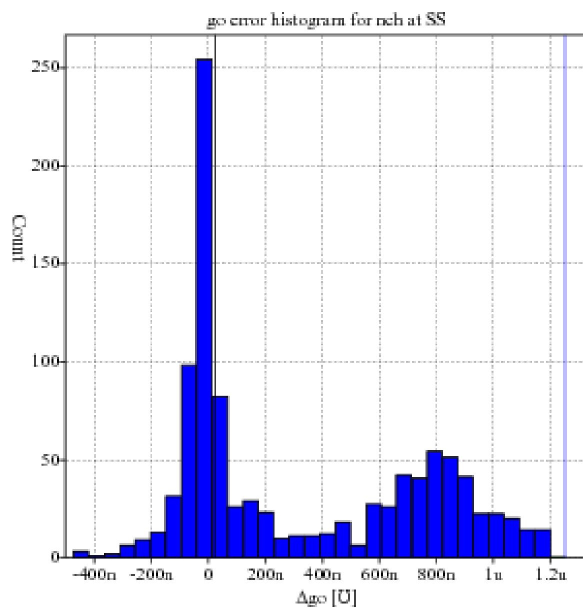


Figure A-6: 40nm NCH TT g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	0.213%	23.902n℧
σ	2.288%	132.963n℧
Percentile	**% Error**	**Δ Error**
50%	0.845%	34.352n℧
75%	1.416%	65.782n℧
90%	2.549%	146.562n℧
95%	3.850%	245.342n℧
99%	7.602%	544.027n℧



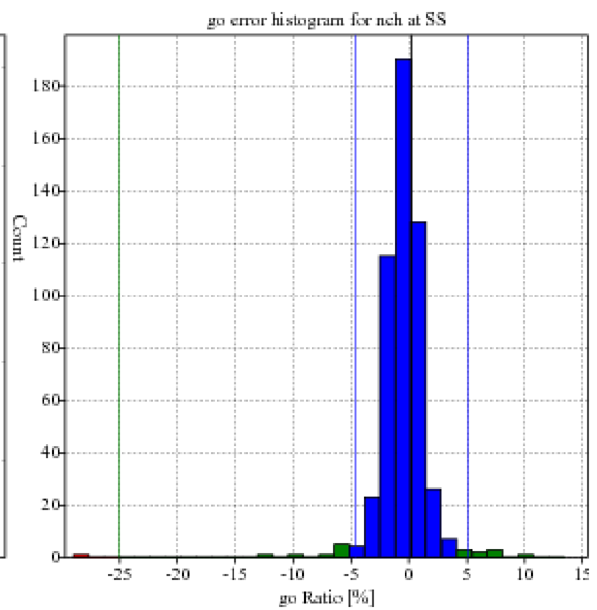
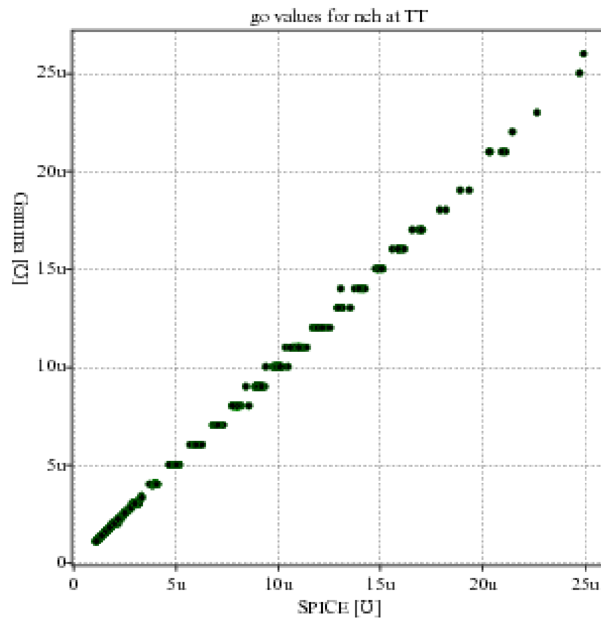
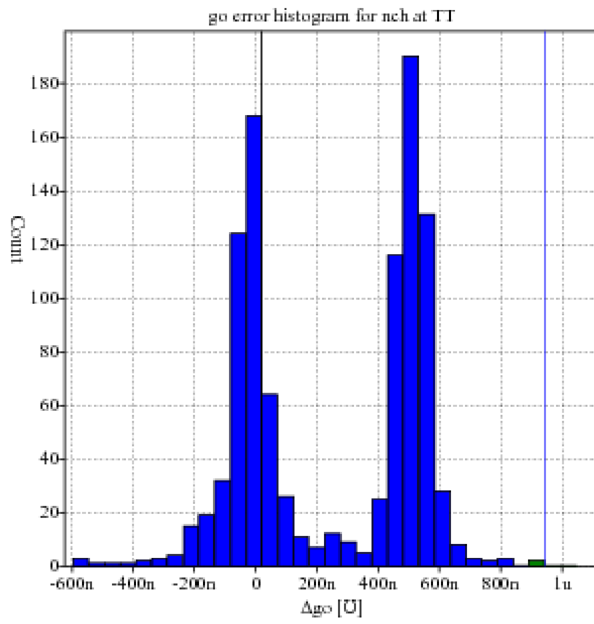


Figure A-7: 40nm NCH SS g_o LUT Queries to SPICE Matching Statistics



Statistics		
Average	0.197%	18.063n℧
σ	1.746%	145.992n℧
Percentile	**% Error**	**Δ Error**
50%	0.807%	43.856n℧
75%	1.396%	86.483n℧
90%	2.599%	212.263n℧
95%	3.790%	291.846n℧
99%	6.575%	591.736n℧



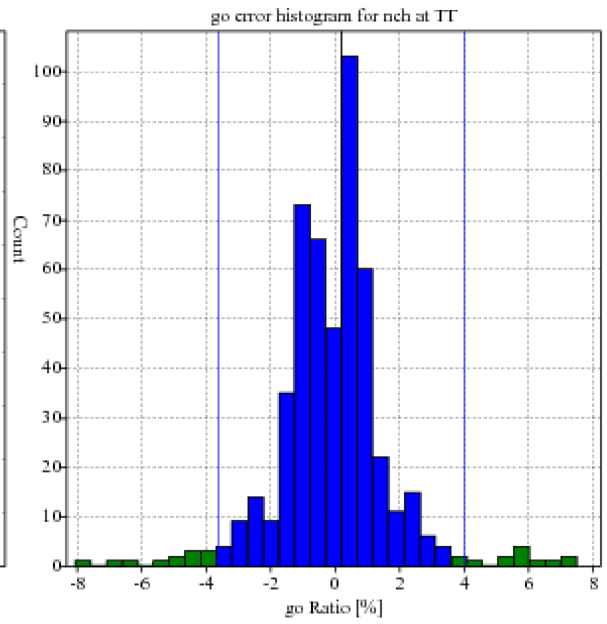
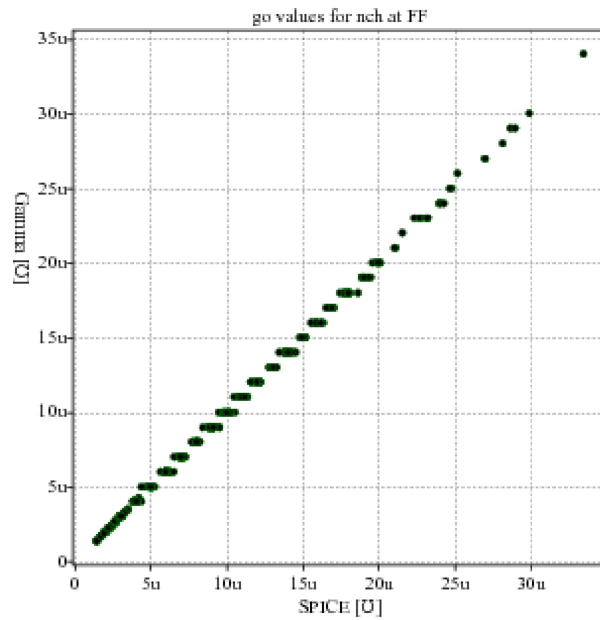
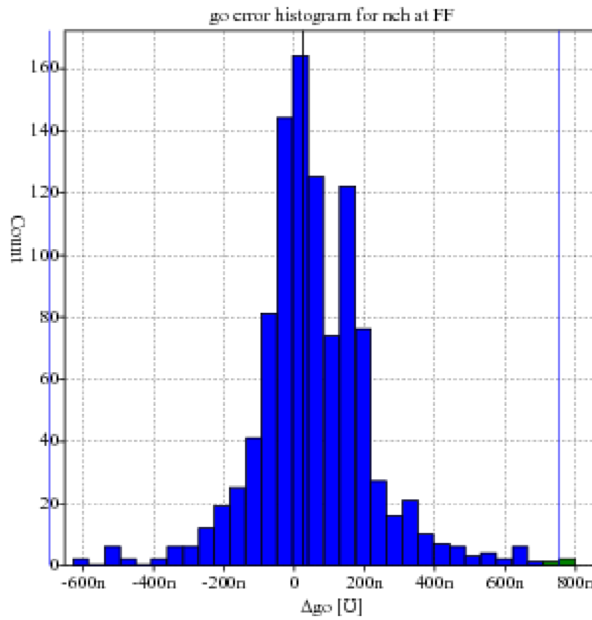


Figure A-8: 40nm NCH TT go LUT Queries to SPICE Matching Statistics



Statistics		
Average	0.329%	28.435nƱ
σ	1.820%	161.152nƱ
Percentile	**% Error**	**Δ Error**
50%	0.882%	62.035nƱ
75%	1.542%	129.475nƱ
90%	2.581%	252.095nƱ
95%	3.594%	383.464nƱ
99%	6.698%	536.735nƱ



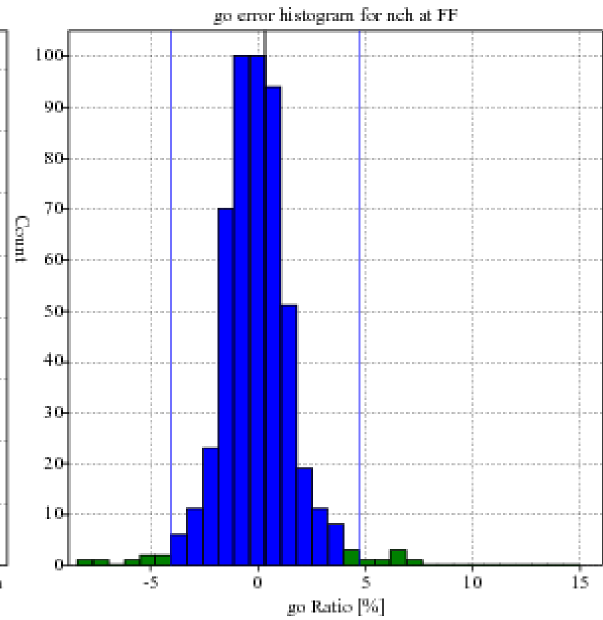
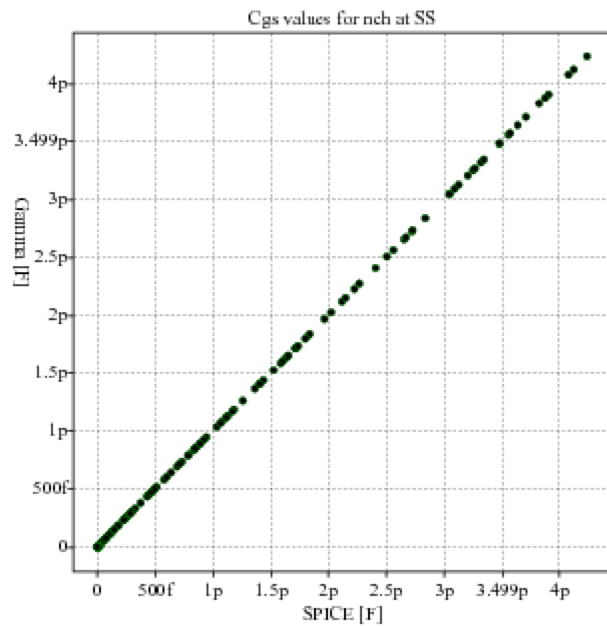
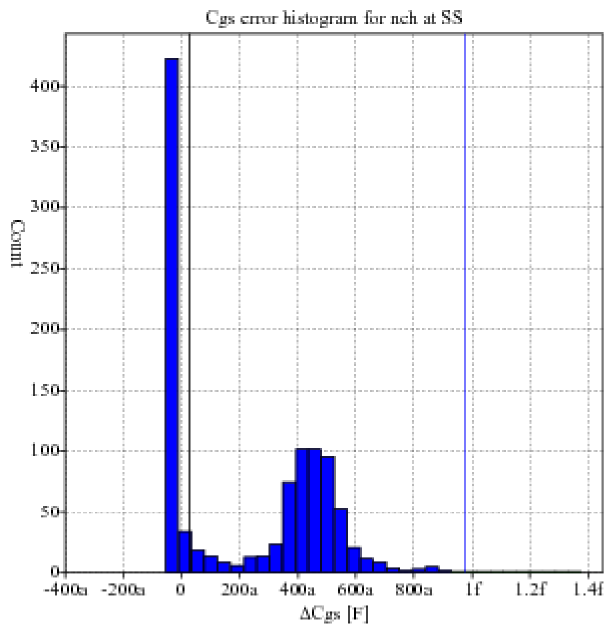


Figure A-9: 40nm NCH FF g_o LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.178%	28.793aF
σ	0.357%	111.071aF
Percentile	**% Error**	**Δ Error**
50%	0.184%	29.153aF
75%	0.215%	31.206aF
90%	0.655%	41.206aF
95%	0.925%	131.206aF
99%	1.115%	501.206aF



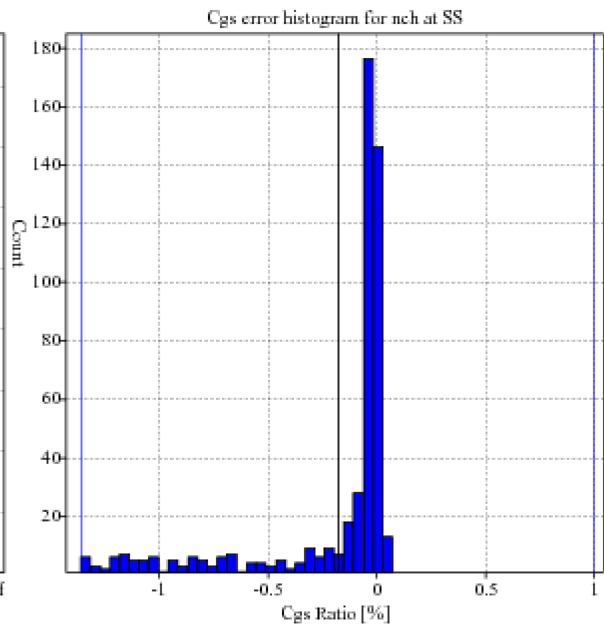
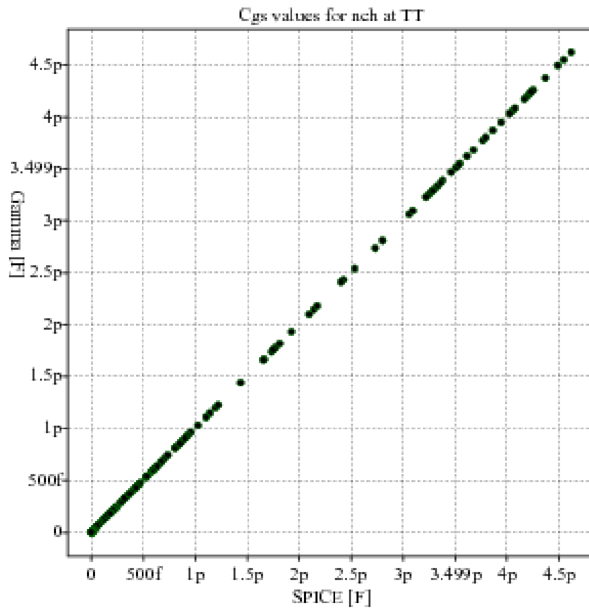
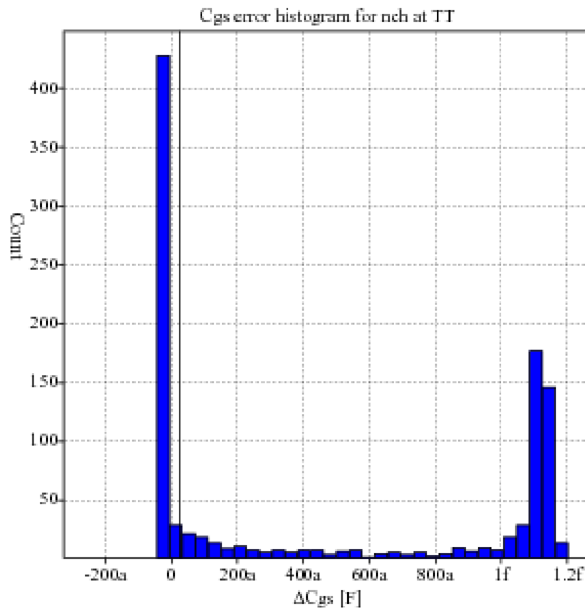


Figure A-10: 40nm NCH SS C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.218%	23.303aF
σ	0.429%	89.62aF
Percentile	% Error	Δ Error
50%	0.221%	23.573aF
75%	0.254%	26.563aF
90%	0.811%	36.696aF
95%	1.040%	116.696aF
99%	1.327%	366.696aF



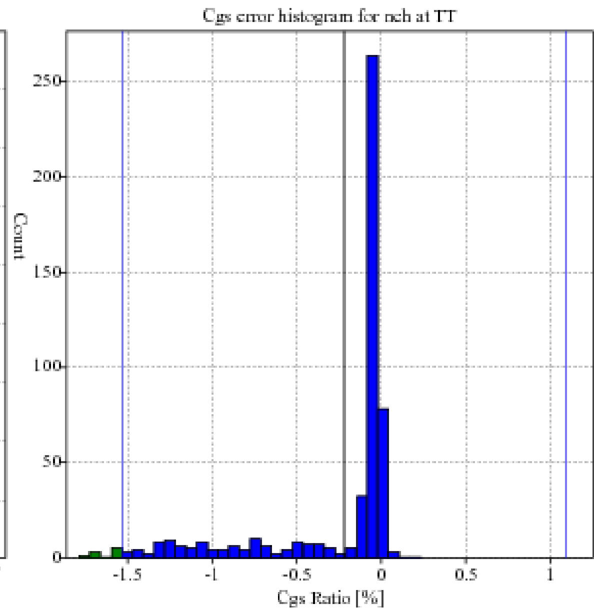
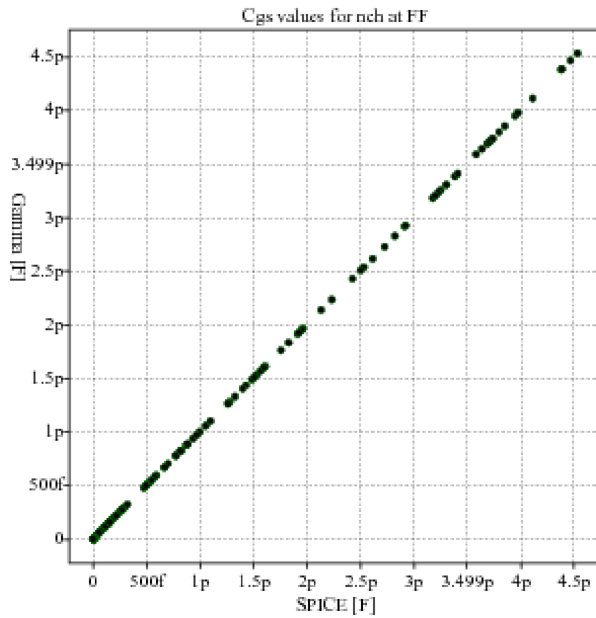
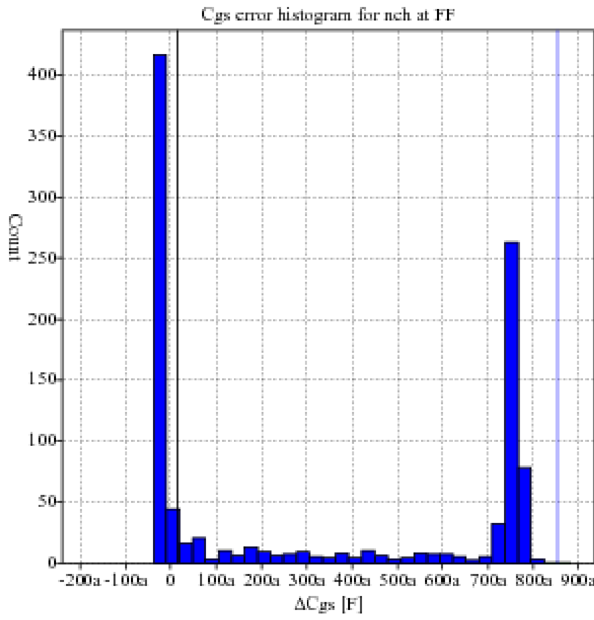


Figure A-11: 40nm NCH TT C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.238%	15.522aF
σ	0.505%	65.294aF
Percentile	**% Error**	**Δ Error**
50%	0.241%	15.702aF
75%	0.266%	18.99aF
90%	0.949%	24.477aF
95%	1.341%	74.477aF
99%	1.568%	284.477aF



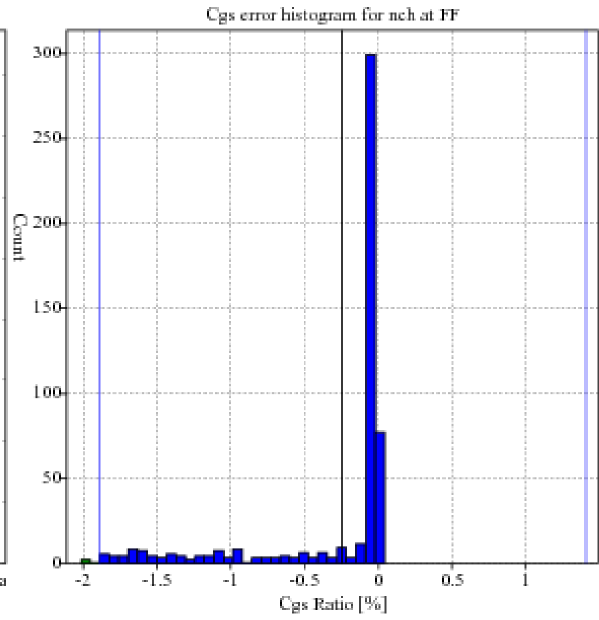
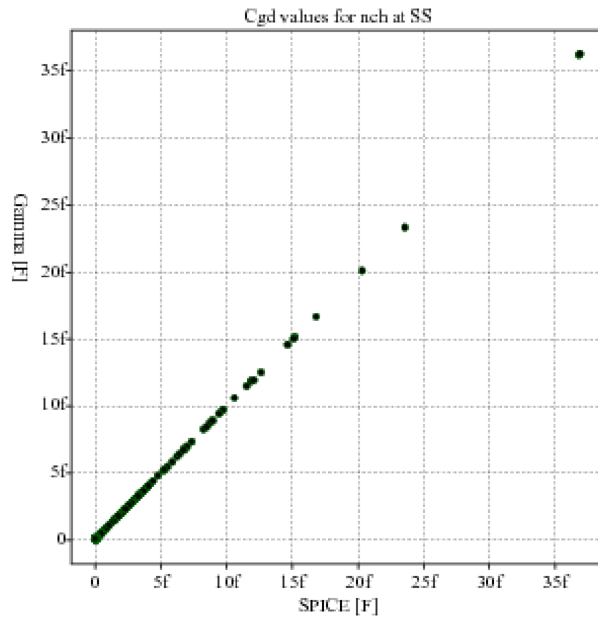
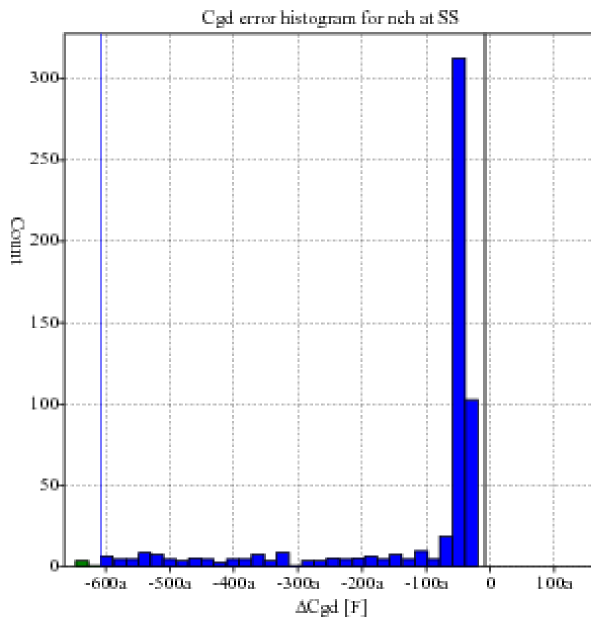


Figure A-12: 40nm NCH FF C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.159%	-8.408aF
σ	1.533%	44.483aF
Percentile	**% Error**	**Δ Error**
50%	0.793%	8.017aF
75%	1.032%	8.216aF
90%	1.855%	8.363aF
95%	4.258%	27.291aF
99%	5.264%	181.991aF



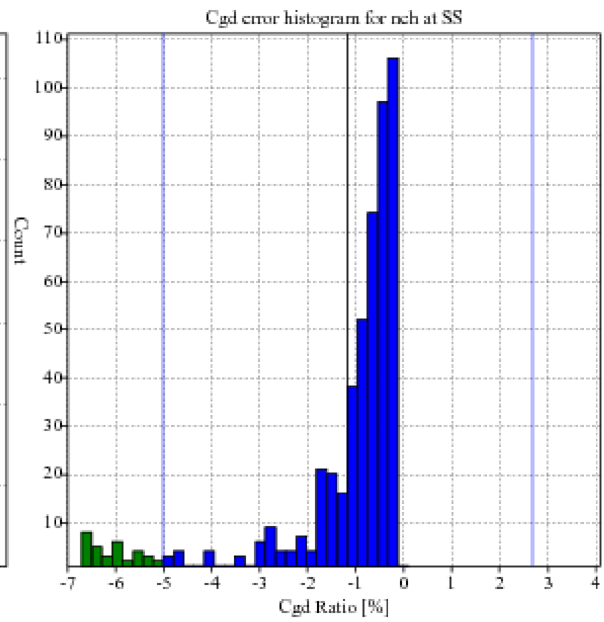
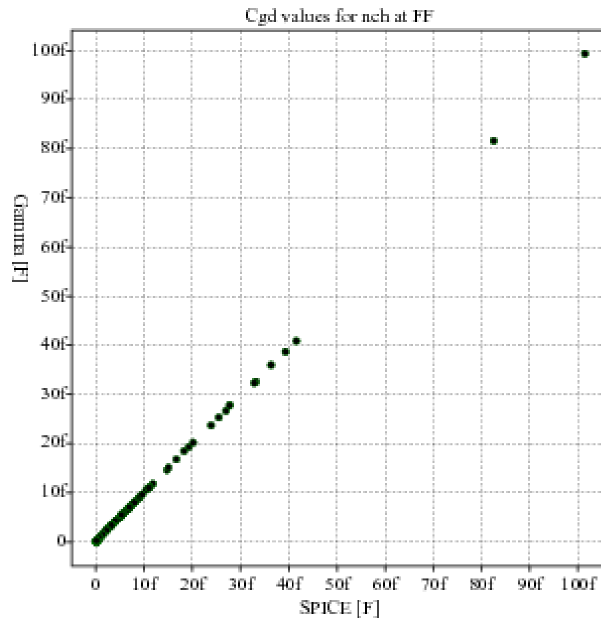
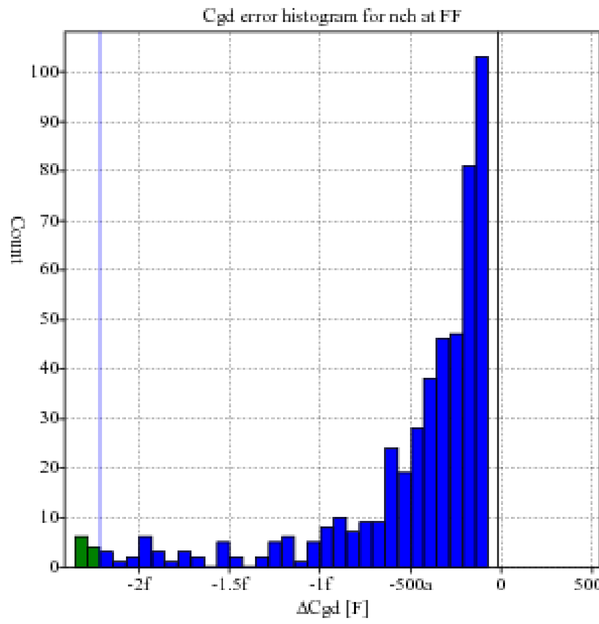


Figure A-13: 40nm NCH SS C_{GD} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.263%	-21.817aF
σ	1.671%	138.482aF
Percentile	**% Error**	**Δ Error**
50%	0.893%	21.3aF
75%	1.115%	21.554aF
90%	2.399%	21.725aF
95%	4.702%	26.122aF
99%	5.525%	549.782aF



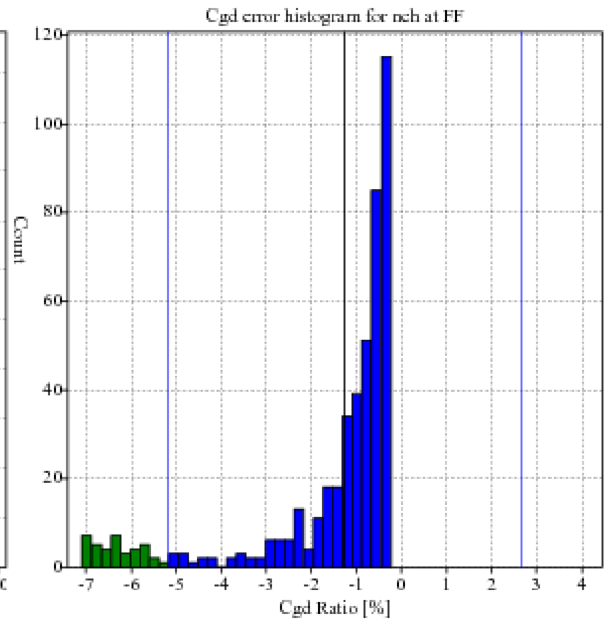
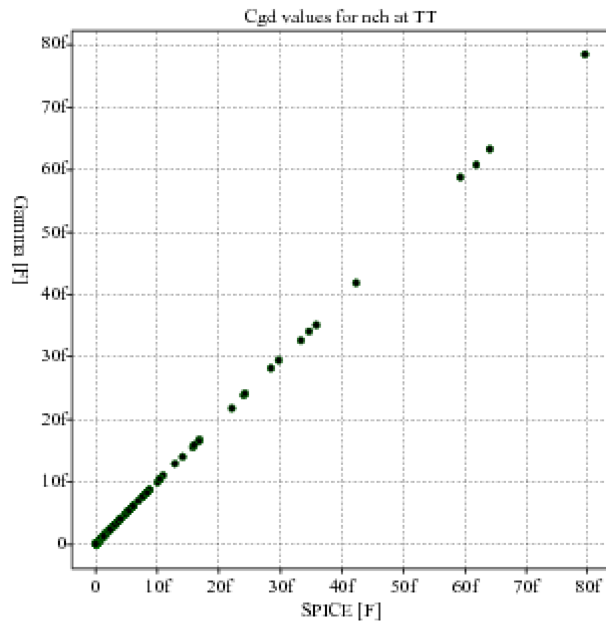
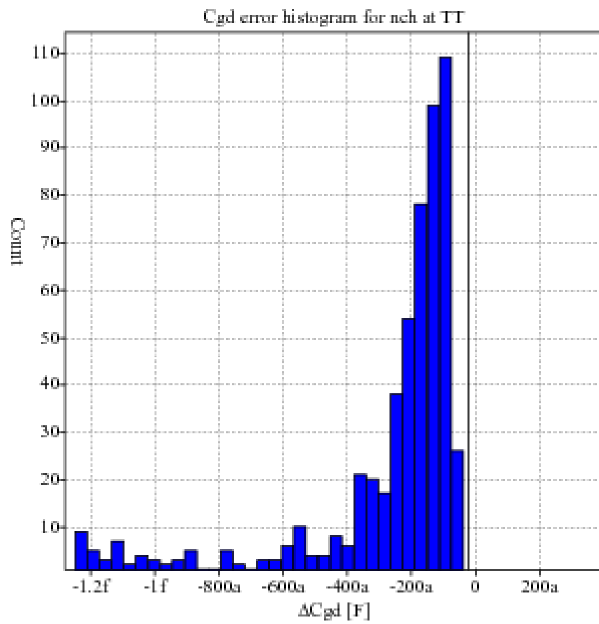


Figure A-14: 40nm NCH FF C_{GD} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.222%	-21.068aF
σ	1.467%	109.264aF
Percentile	**% Error**	**Δ Error**
50%	0.845%	20.538aF
75%	1.095%	20.814aF
90%	1.793%	20.988aF
95%	3.859%	29.431aF
99%	5.317%	594.031aF



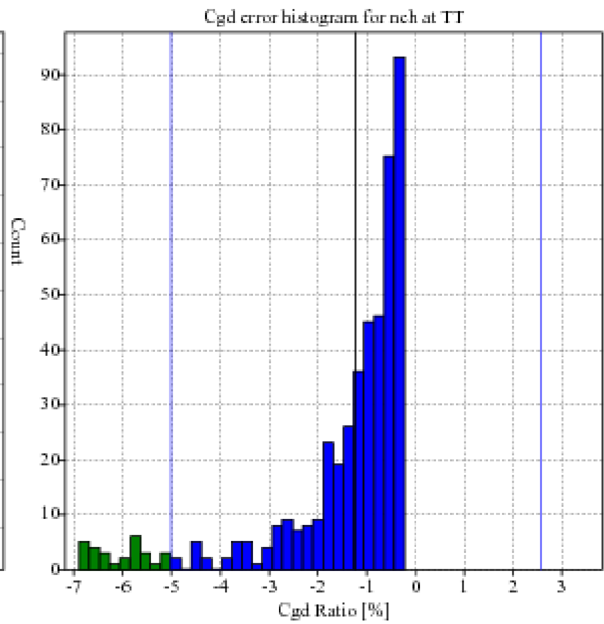
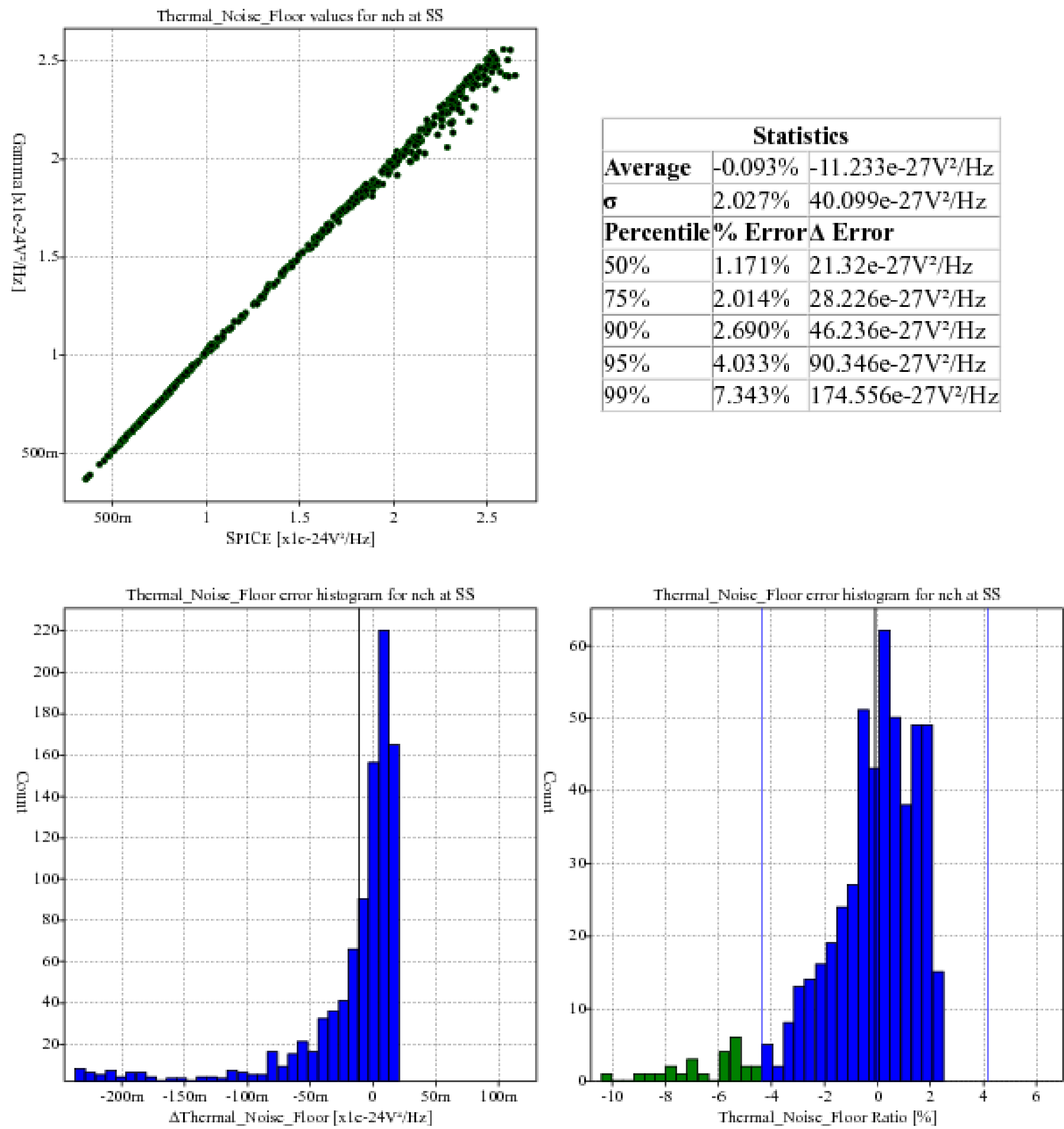
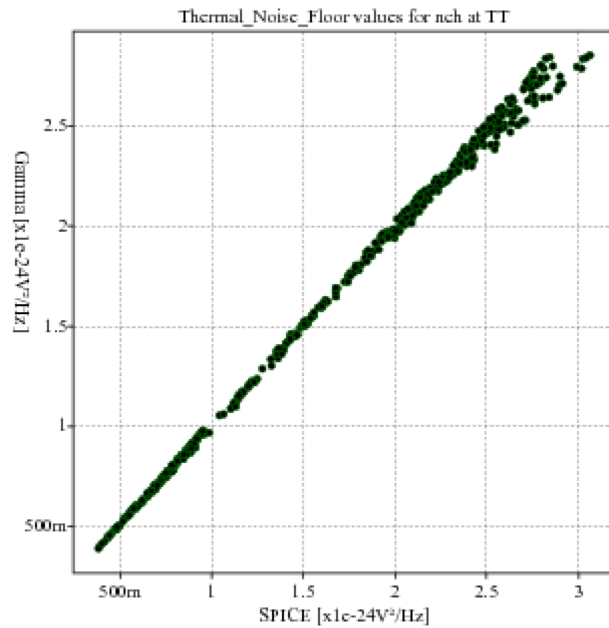


Figure A-15: 40nm NCH TT C_{GD} LUT Queries to SPICE Matching Statistics

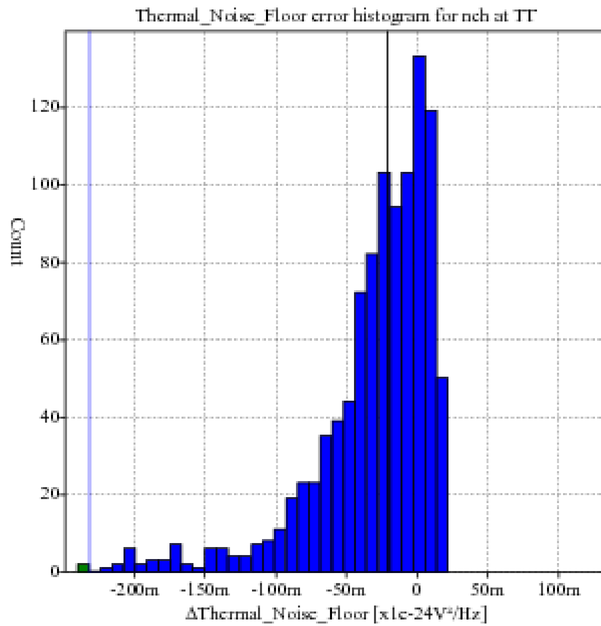


Statistics		
Average	-0.093%	-11.233e-27V²/Hz
σ	2.027%	40.099e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	1.171%	21.32e-27V²/Hz
75%	2.014%	28.226e-27V²/Hz
90%	2.690%	46.236e-27V²/Hz
95%	4.033%	90.346e-27V²/Hz
99%	7.343%	174.556e-27V²/Hz

Figure A-16: 40nm NCH SS N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.606%	-20.644e-27V²/Hz
σ	1.977%	44.31e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	1.123%	25.084e-27V²/Hz
75%	2.082%	35.644e-27V²/Hz
90%	3.108%	51.275e-27V²/Hz
95%	3.693%	86.185e-27V²/Hz
99%	6.201%	179.145e-27V²/Hz



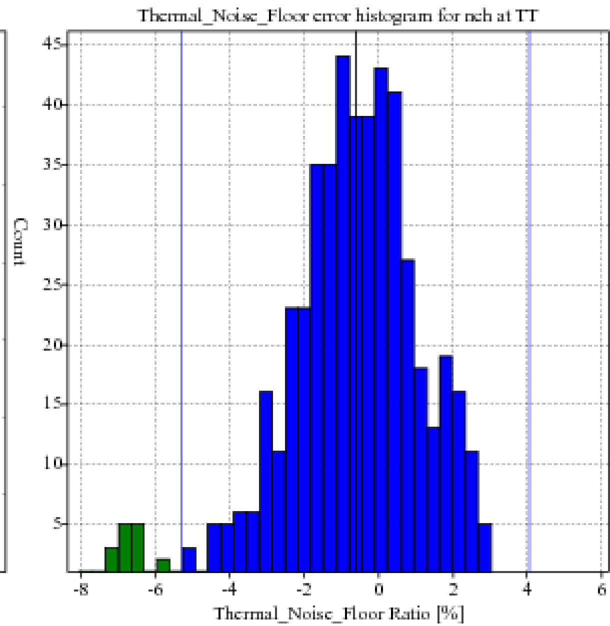
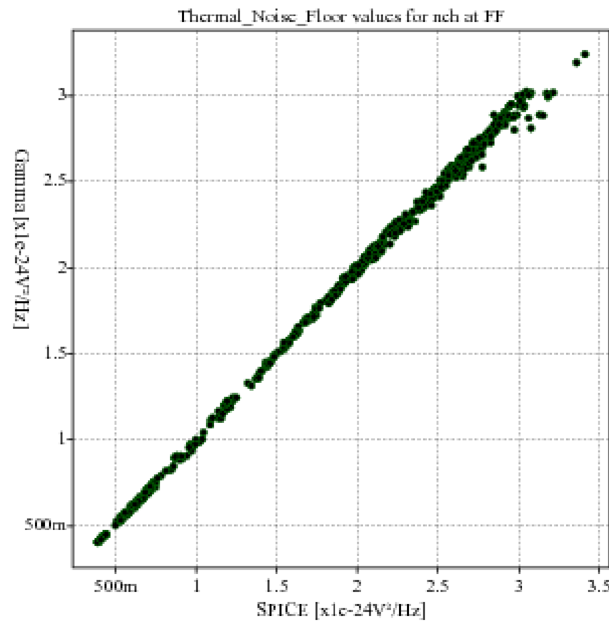
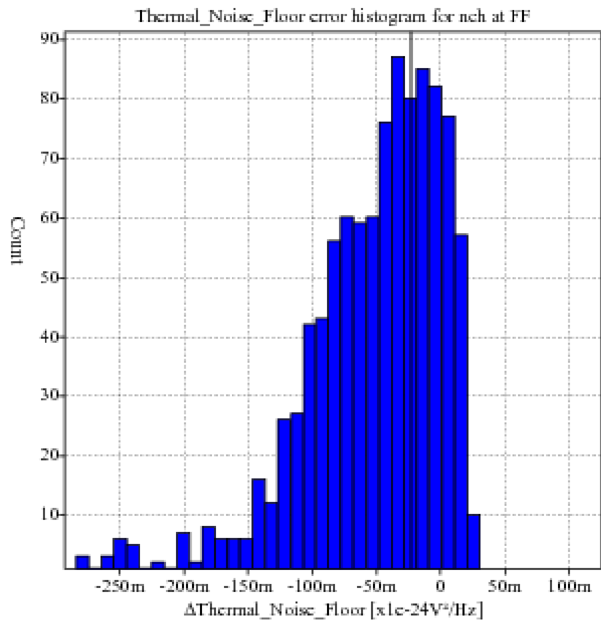


Figure A-17: 40nm NCH TT N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.784%	-23.495e-27V²/Hz
σ	1.877%	41.916e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	1.067%	23.115e-27V²/Hz
75%	1.907%	37.734e-27V²/Hz
90%	3.034%	49.665e-27V²/Hz
95%	3.789%	70.174e-27V²/Hz
99%	5.444%	168.594e-27V²/Hz



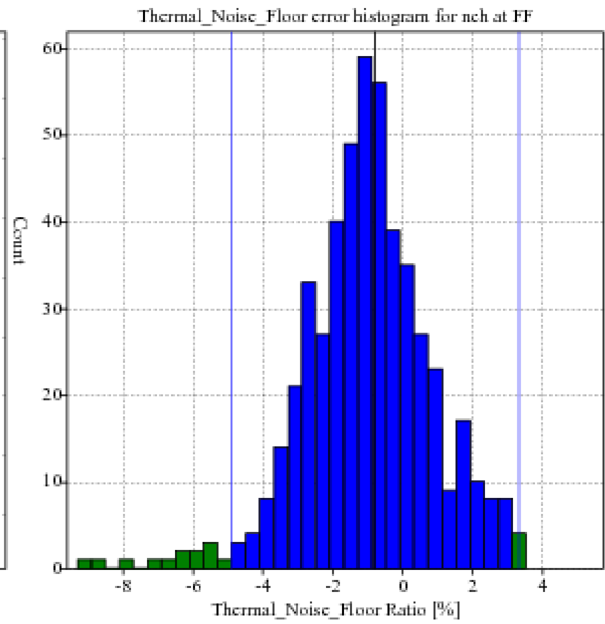
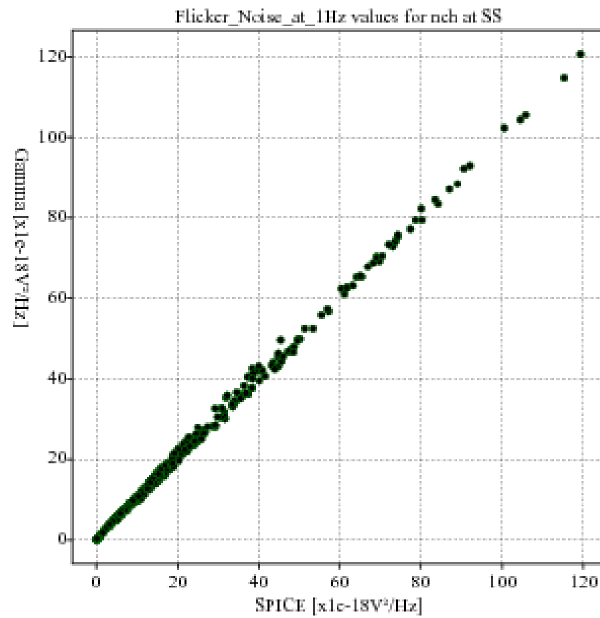
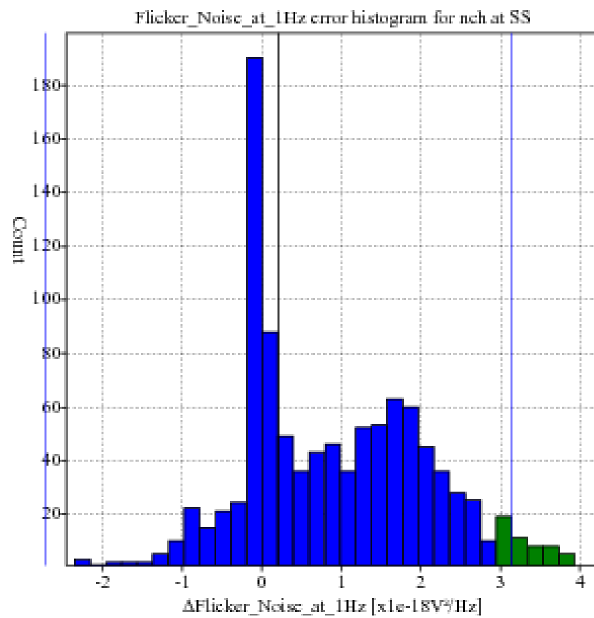


Figure A-18: 40nm NCH FF N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	3.1%	214.79zV^2/Hz
σ	4.116%	736.429zV^2/Hz
Percentile	**% Error**	**Δ Error**
50%	3.209%	214.663zV^2/Hz
75%	4.932%	634.59zV^2/Hz
90%	6.382%	1.173aV^2/Hz
95%	7.666%	1.679aV^2/Hz
99%	9.162%	2.777aV^2/Hz



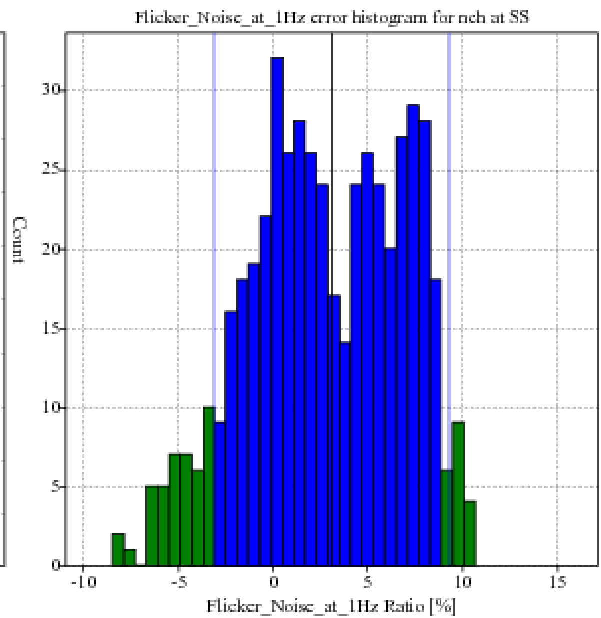
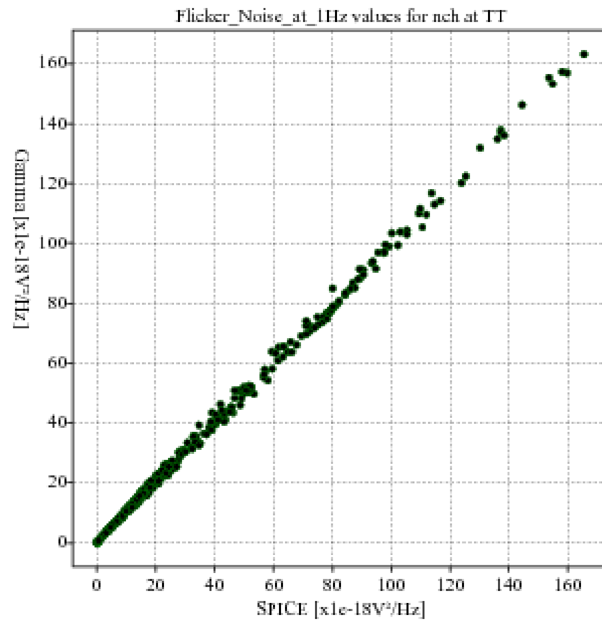
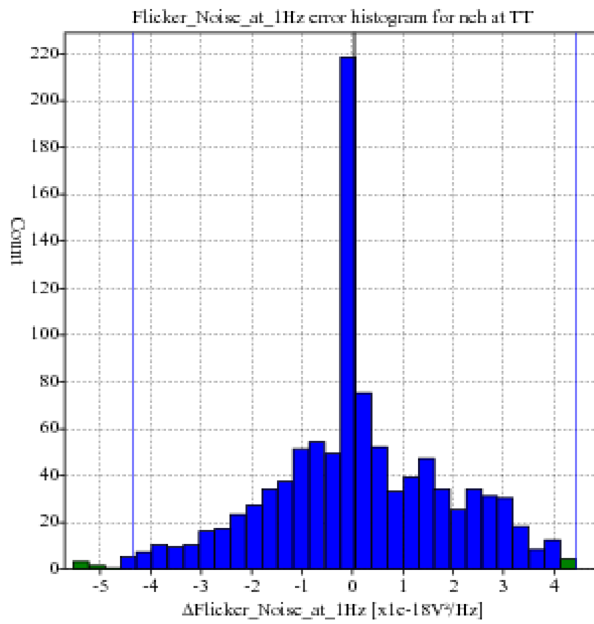


Figure A-19: 40nm NCH SS N_F LUT Queries to SPICE Matching Statistics



Statistics		
Average	2.587%	44.41zV²/Hz
σ	4.641%	1.223aV²/Hz
Percentile	**% Error**	**Δ Error**
50%	3.878%	407.51zV²/Hz
75%	5.728%	1.208aV²/Hz
90%	7.227%	2.107aV²/Hz
95%	8.319%	2.685aV²/Hz
99%	9.685%	3.949aV²/Hz



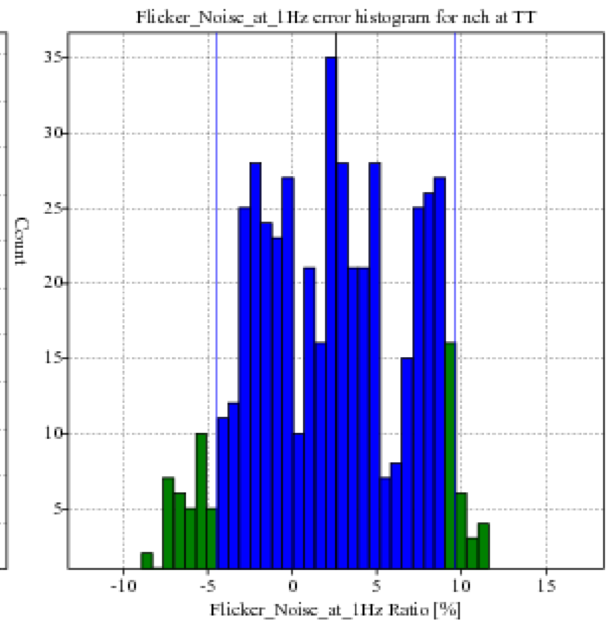
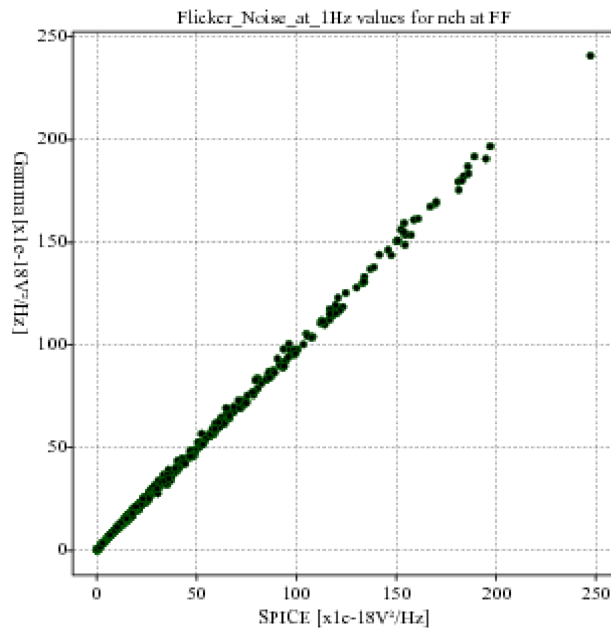
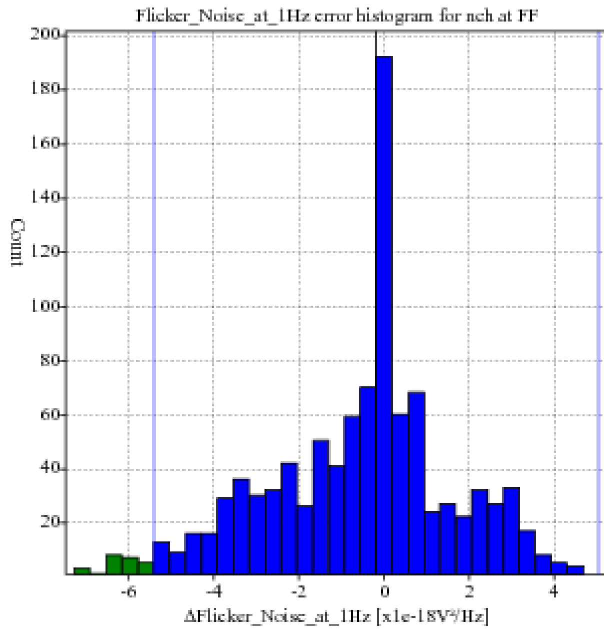


Figure A-20: 40nm NCH TT N_F LUT Queries to SPICE Matching Statistics



Statistics		
Average	1.977%	-194.225zV²/Hz
σ	4.542%	1.542aV²/Hz
Percentile	**% Error**	**Δ Error**
50%	3.657%	673.395zV²/Hz
75%	5.804%	1.438aV²/Hz
90%	7.154%	2.598aV²/Hz
95%	7.747%	3.519aV²/Hz
99%	9.940%	4.887aV²/Hz



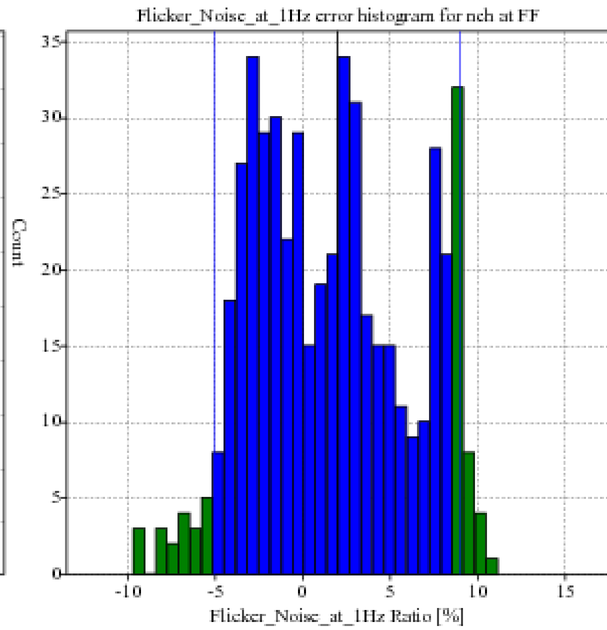
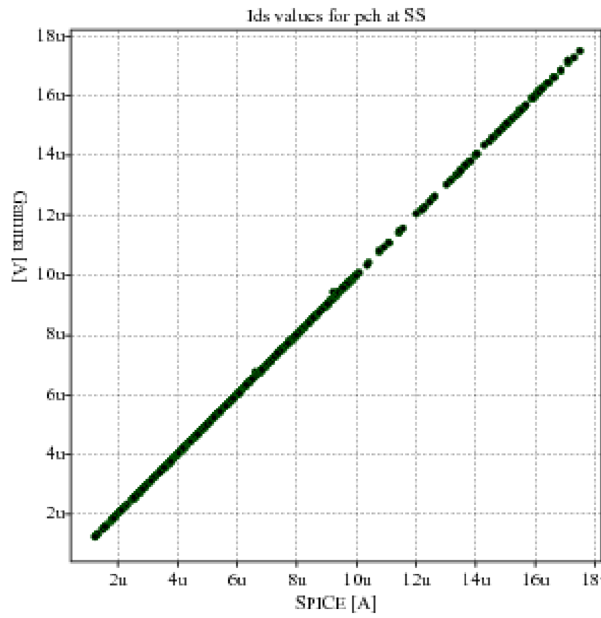
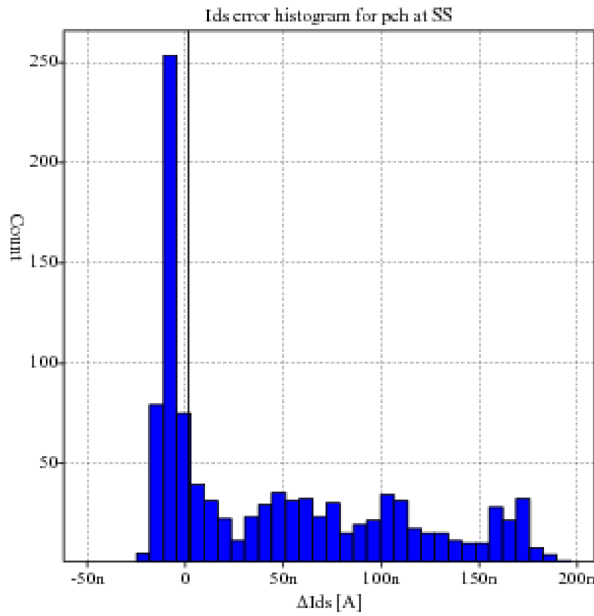


Figure A-21: 40nm NCH FF N_F LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.018%	1.431nA
σ	0.229%	16.576nA
Percentile	**% Error**	**Δ Error**
50%	0.076%	6.671nA
75%	0.174%	9.561nA
90%	0.278%	15.288nA
95%	0.338%	23.468nA
99%	0.683%	61.068nA



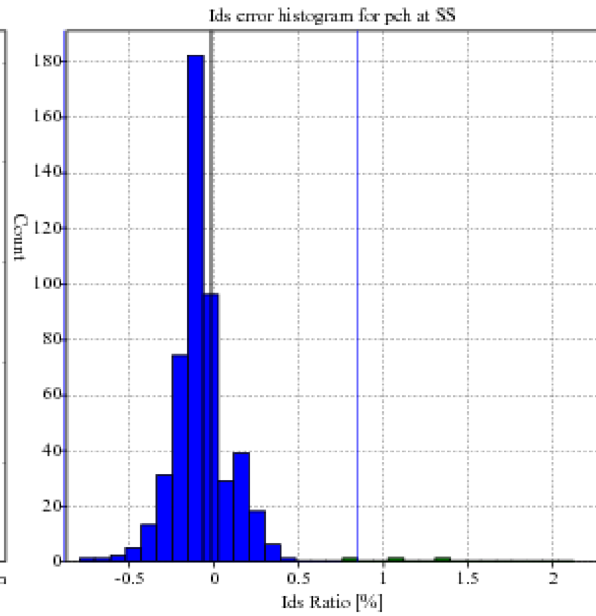
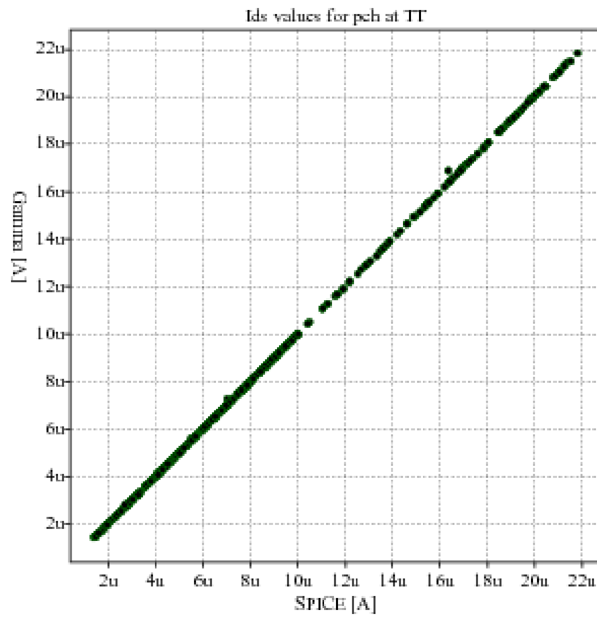
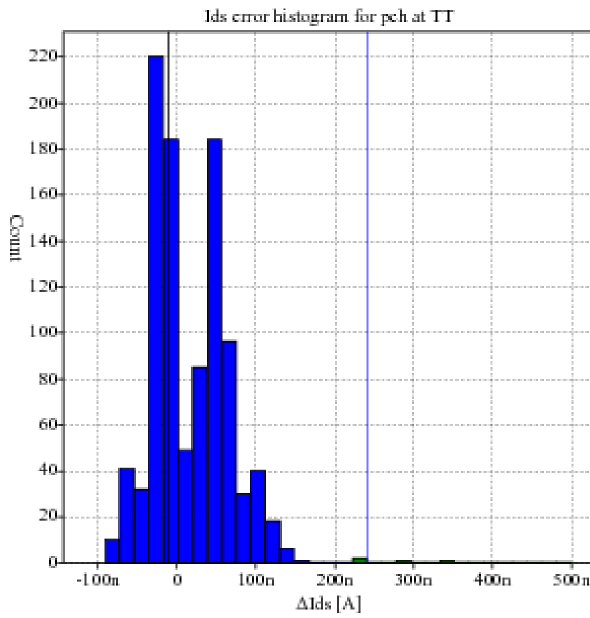


Figure A-22: 40nm PCH SS I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.236%	-10.262nA
σ	0.516%	33.513nA
Percentile	**% Error**	**Δ Error**
50%	0.178%	6.342nA
75%	0.284%	15.962nA
90%	0.542%	40.127nA
95%	0.974%	50.362nA
99%	2.758%	66.762nA



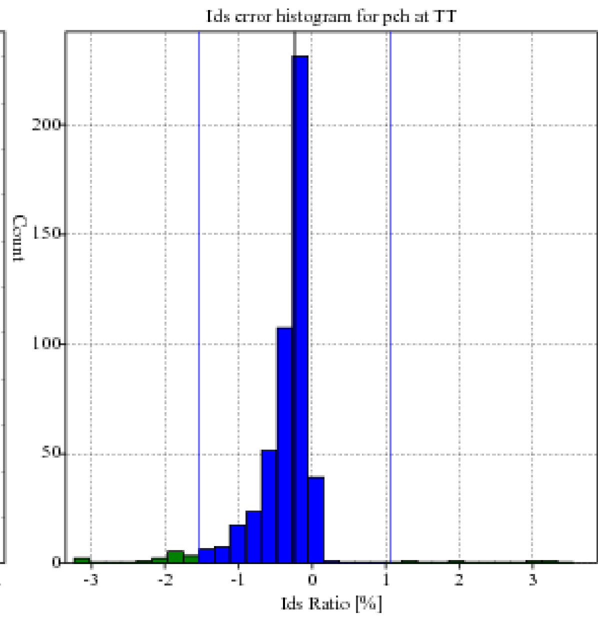
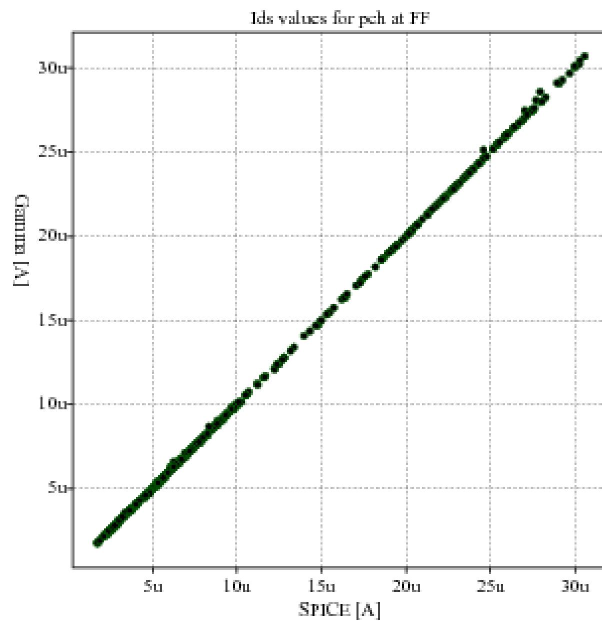
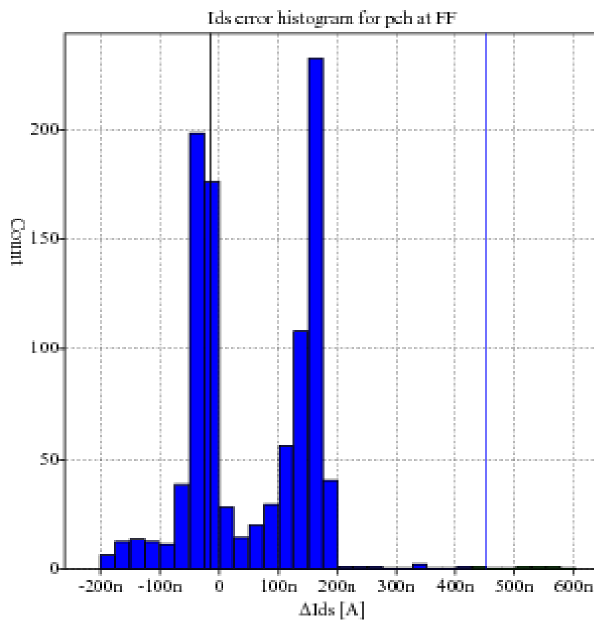


Figure A-23: 40nm PCH TT I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.319%	-13.971nA
σ	0.884%	66.612nA
Percentile	**% Error**	**Δ Error**
50%	0.246%	9.671nA
75%	0.356%	27.098nA
90%	1.019%	98.378nA
95%	2.164%	134.188nA
99%	3.906%	255.511nA



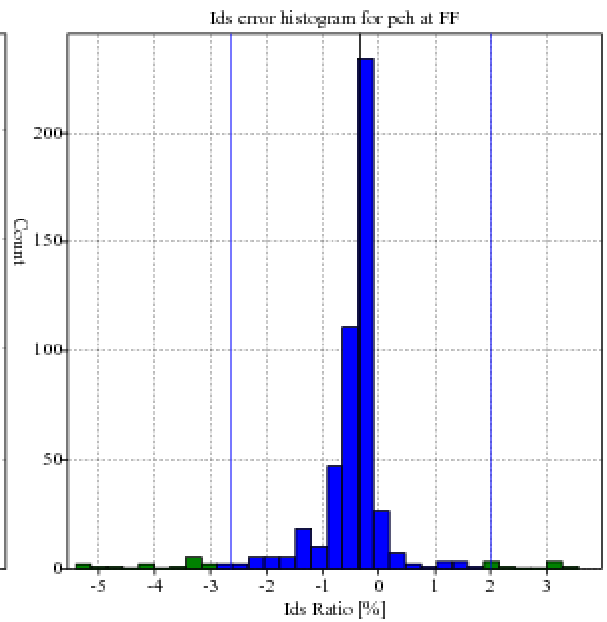
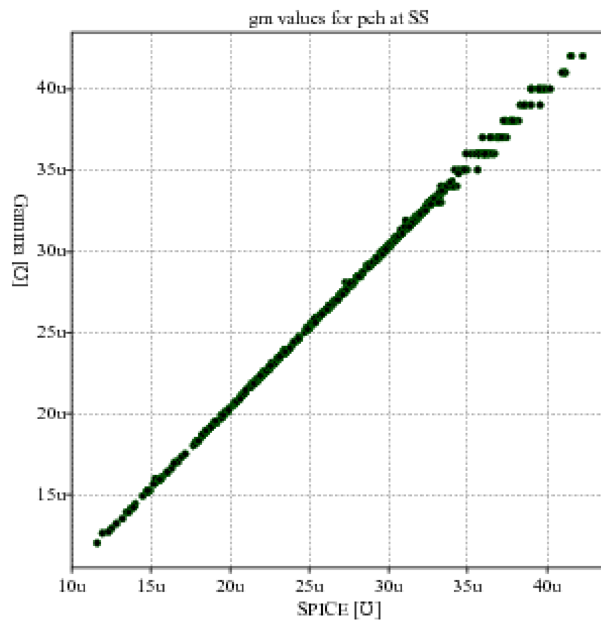
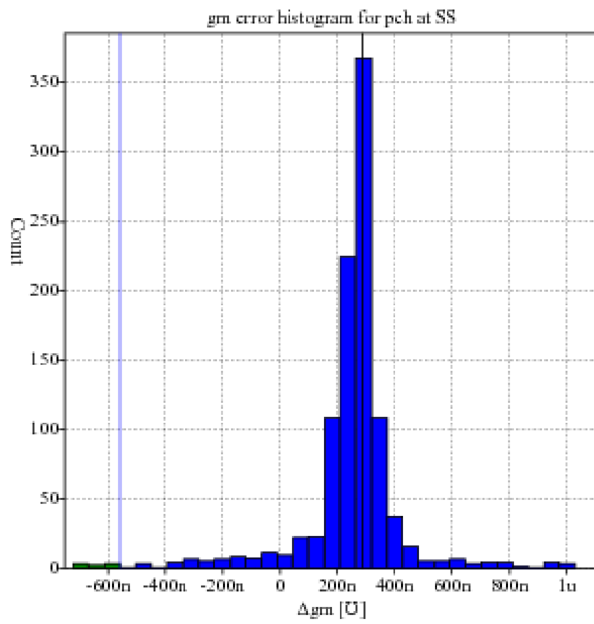


Figure A-24: 40nm PCH FF I_{DS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	1.159%	289.939n℧
σ	0.731%	178.874n℧
Percentile	**% Error**	**Δ Error**
50%	0.308%	55.86n℧
75%	0.586%	100.86n℧
90%	1.193%	243.66n℧
95%	1.602%	436.639n℧
99%	2.588%	761.46n℧



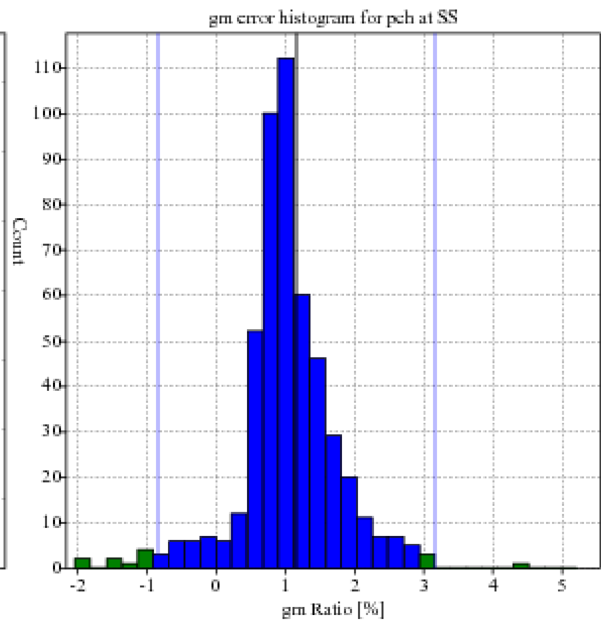
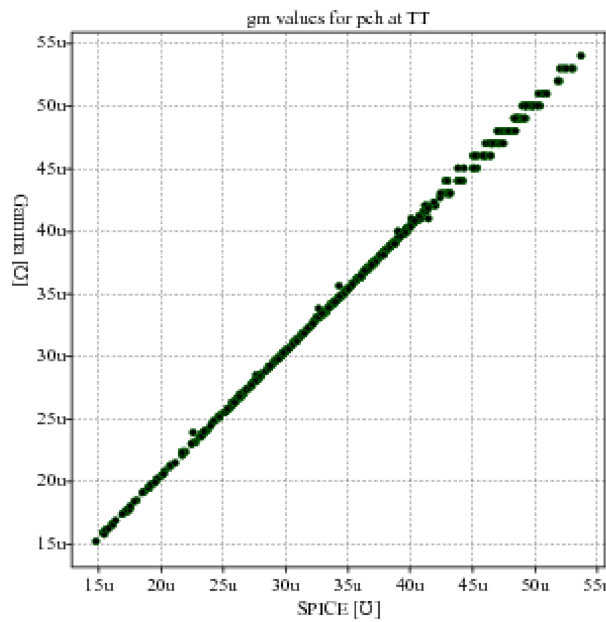
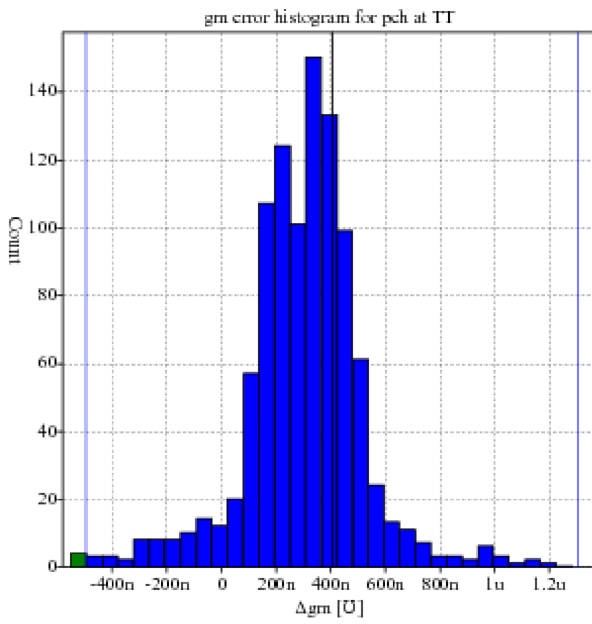


Figure A-25: 40nm PCH SS g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	1.291%	404.895nƱ
σ	0.796%	231.032nƱ
Percentile	**% Error**	**Δ Error**
50%	0.341%	67.795nƱ
75%	0.766%	148.604nƱ
90%	1.422%	404.795nƱ
95%	1.758%	594.795nƱ
99%	2.331%	850.895nƱ



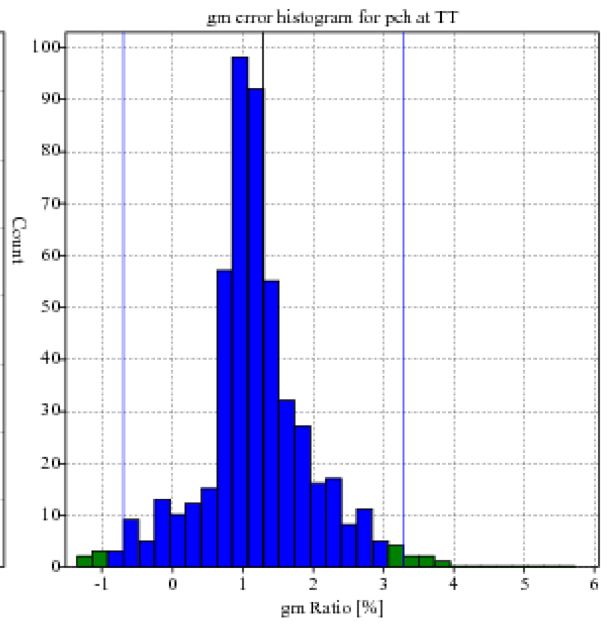
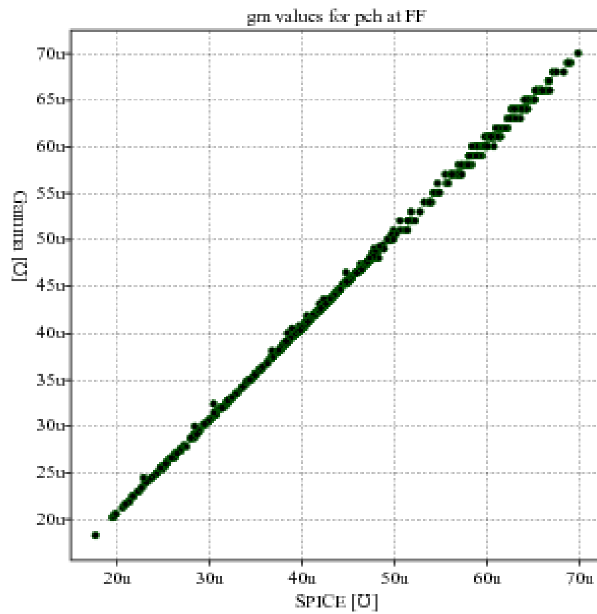
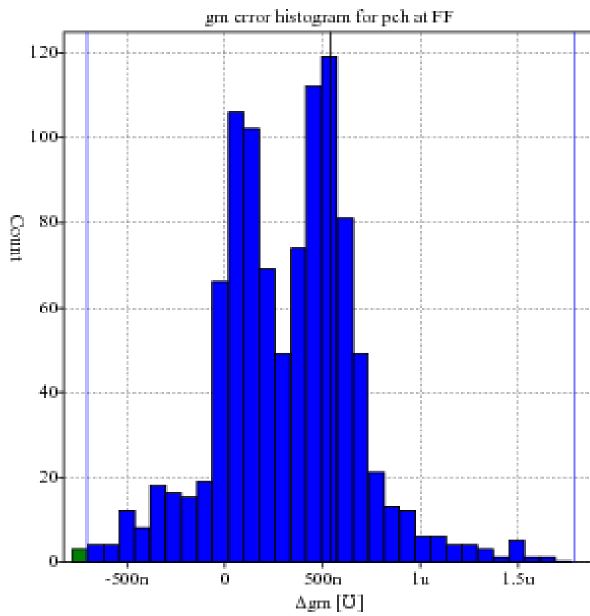


Figure A-26: 40nm PCH TT g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	1.392%	543nƱ
σ	0.917%	325.423nƱ
Percentile	**% Error**	**Δ Error**
50%	0.391%	111.399nƱ
75%	0.938%	252.299nƱ
90%	1.530%	568.5nƱ
95%	1.836%	806.199nƱ
99%	2.542%	1.078uƱ



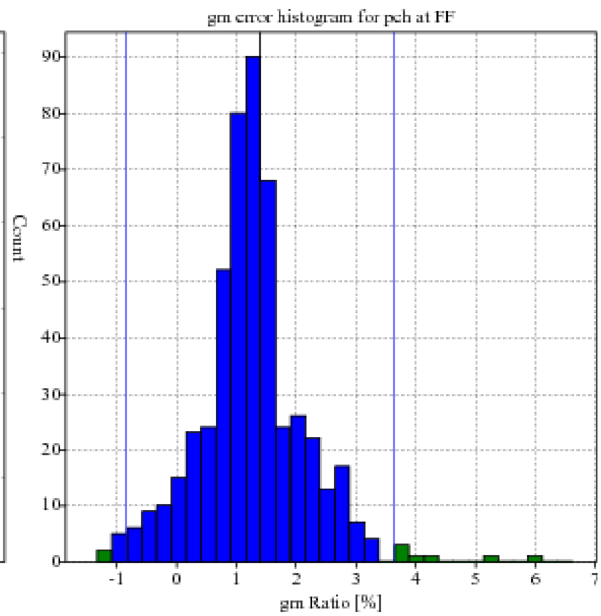
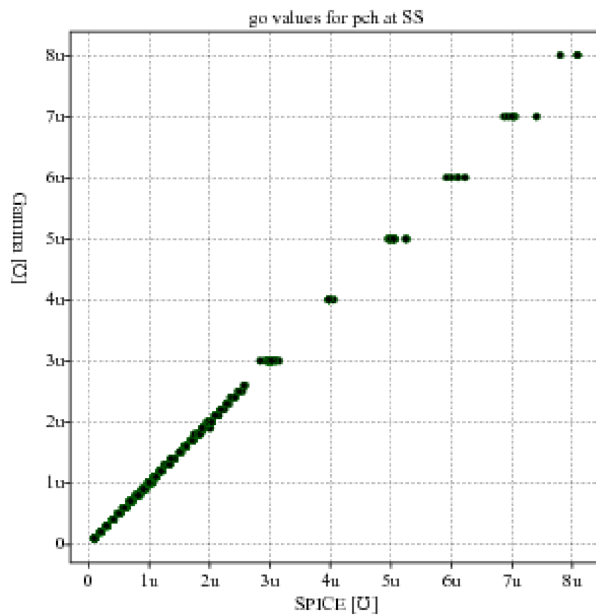
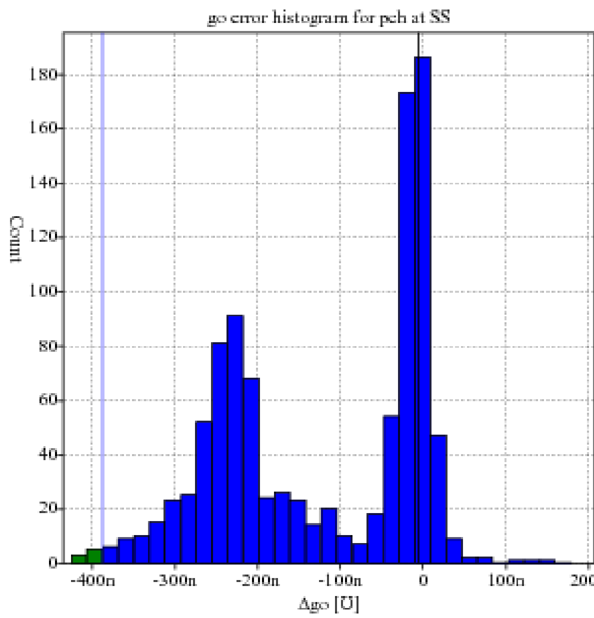


Figure A-27: 40nm PCH FF g_m LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.195%	-4.459n℧
σ	1.756%	37.323n℧
Percentile	**% Error**	**Δ Error**
50%	0.892%	10.924n℧
75%	1.592%	21.509n℧
90%	2.580%	38.939n℧
95%	4.056%	53.979n℧
99%	5.536%	166.049n℧



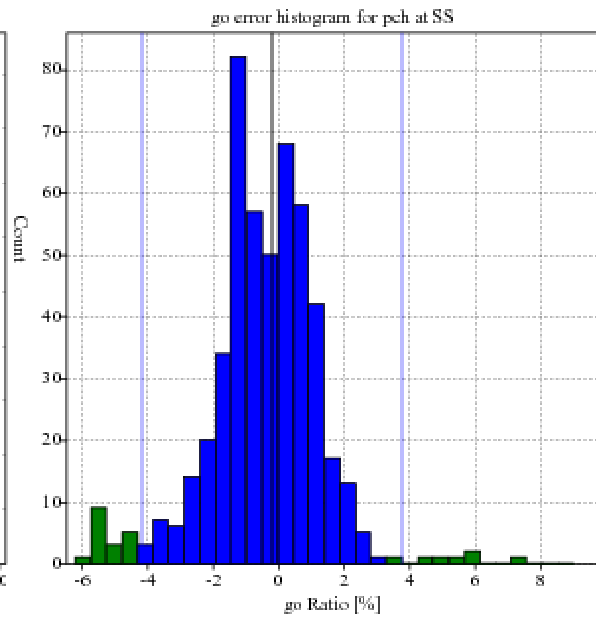
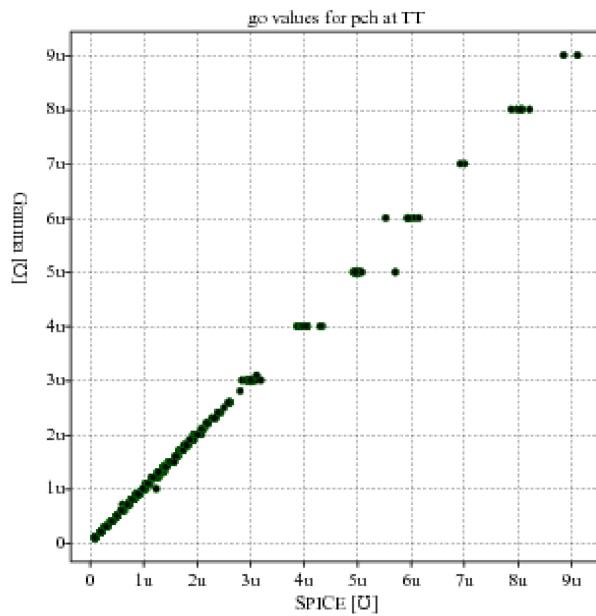
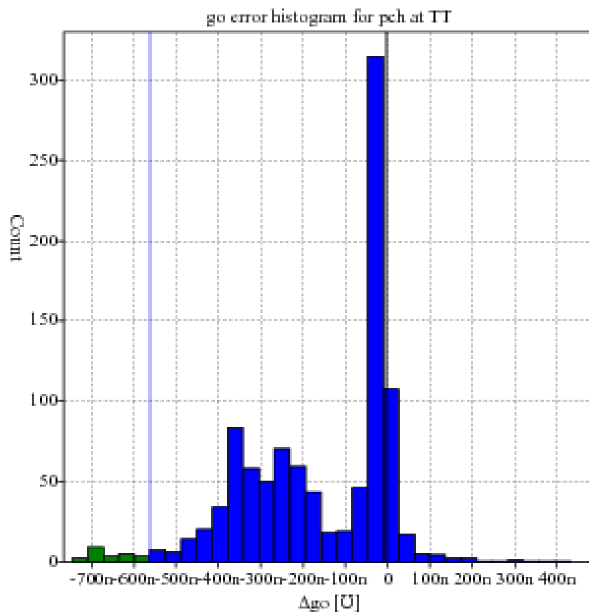


Figure A-28: 40nm PCH SS g_o LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.042%	-3.084n℧
σ	2.443%	54.605n℧
Percentile	**% Error**	**Δ Error**
50%	0.936%	10.745n℧
75%	1.577%	24.534n℧
90%	2.961%	48.535n℧
95%	4.570%	70.545n℧
99%	12.382%	235.255n℧



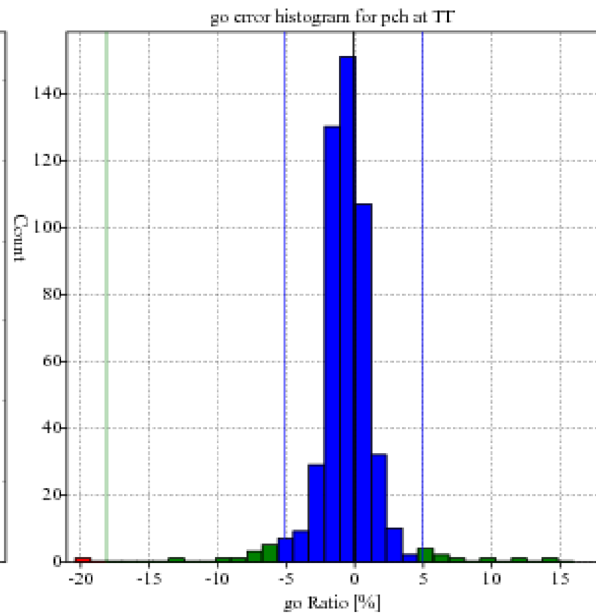
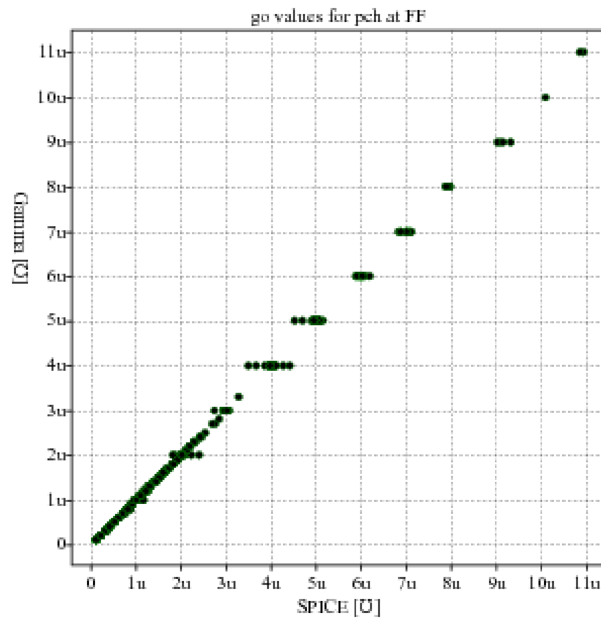
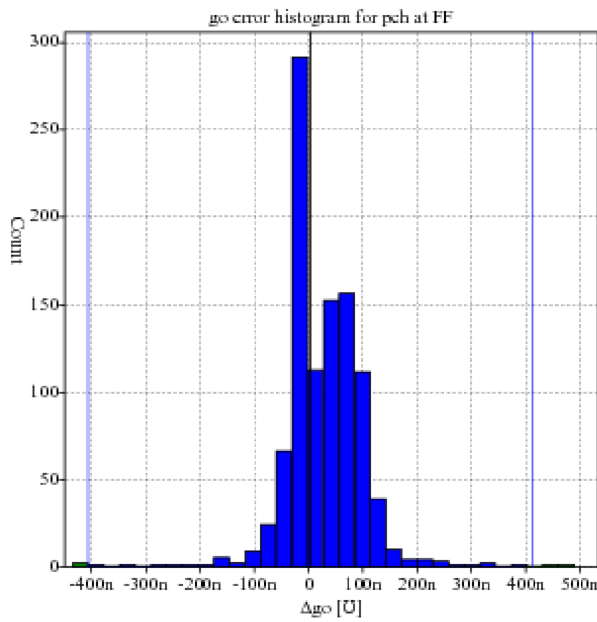


Figure A-29: 40nm PCH TT g_o LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.012%	2.657n℧
σ	2.566%	63.163n℧
Percentile	**% Error**	**Δ Error**
50%	0.872%	11.772n℧
75%	1.502%	27.527n℧
90%	2.540%	60.407n℧
95%	4.579%	107.147n℧
99%	12.131%	316.067n℧



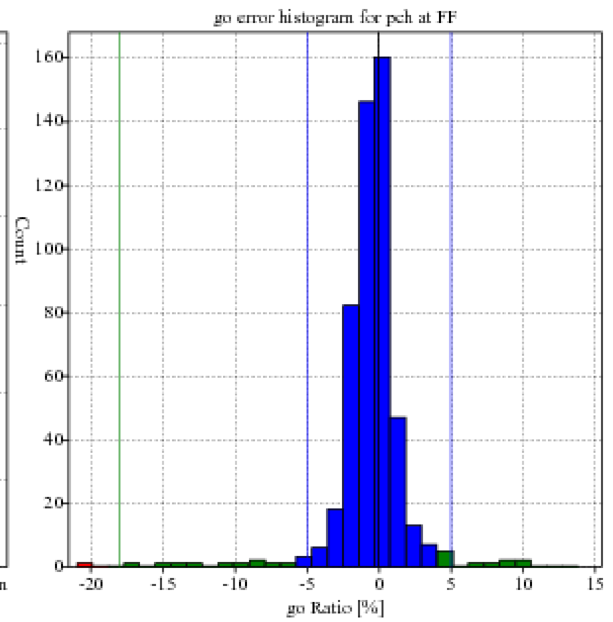
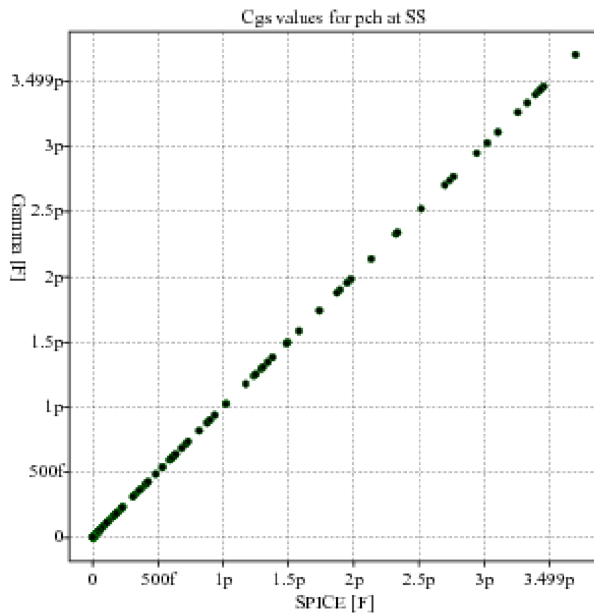
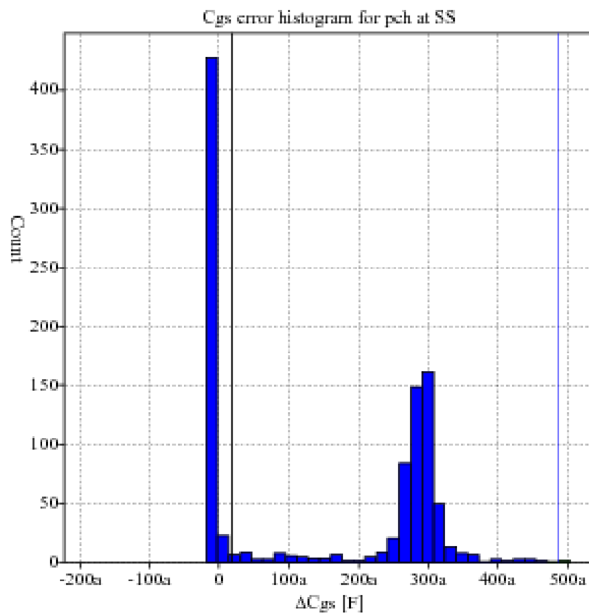


Figure A-30: 40nm PCH FF g_o LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.231%	18.23aF
σ	0.372%	66.699aF
Percentile	**% Error**	**Δ Error**
50%	0.241%	18.453aF
75%	0.28%	20.472aF
90%	0.686%	21.114aF
95%	0.875%	116.769aF
99%	0.947%	311.769aF



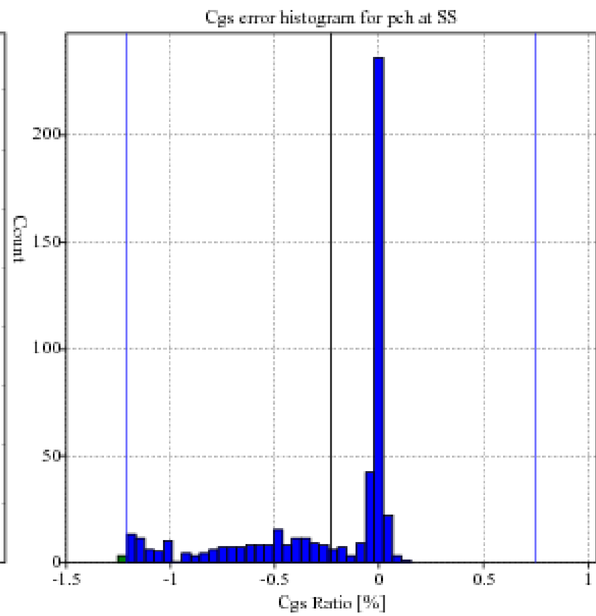
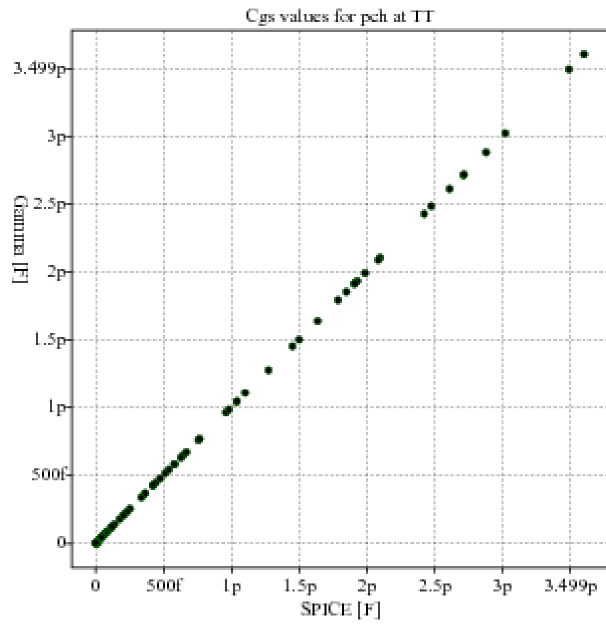
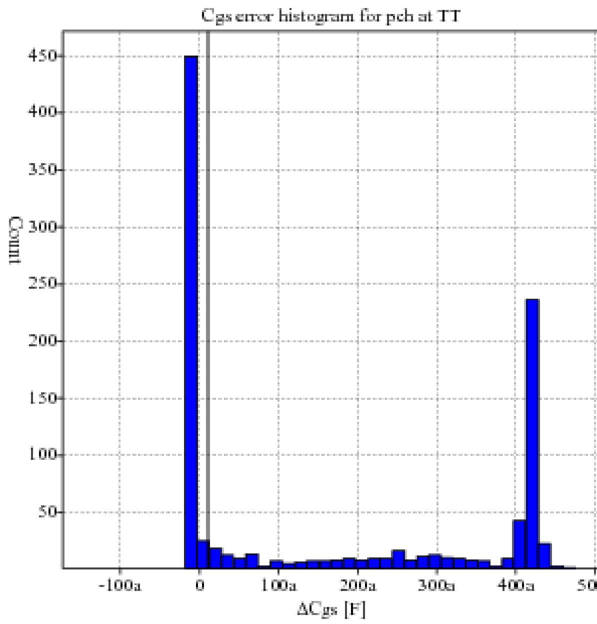


Figure A-31: 40nm PCH SS C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.274%	10.767aF
σ	0.418%	49.449aF
Percentile	**% Error**	**Δ Error**
50%	0.279%	11.117aF
75%	0.311%	13.021aF
90%	0.777%	13.948aF
95%	0.966%	69.232aF
99%	1.059%	309.232aF



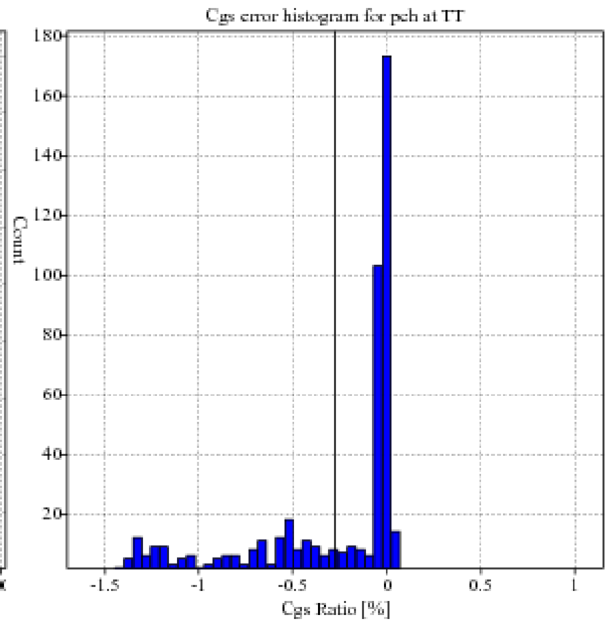
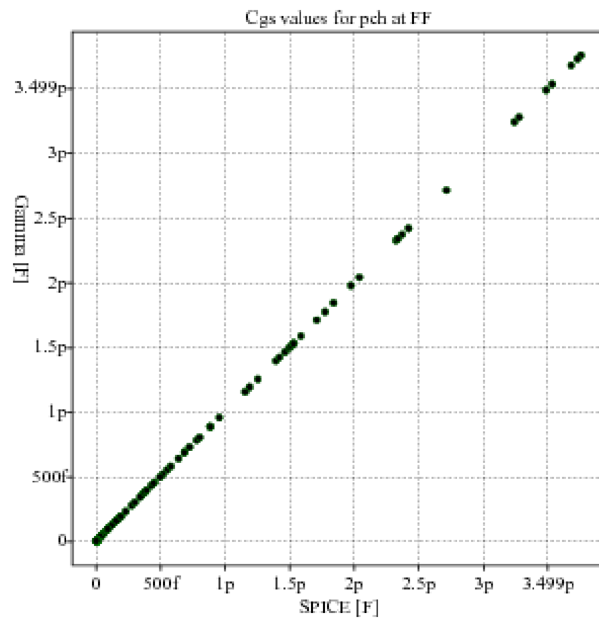
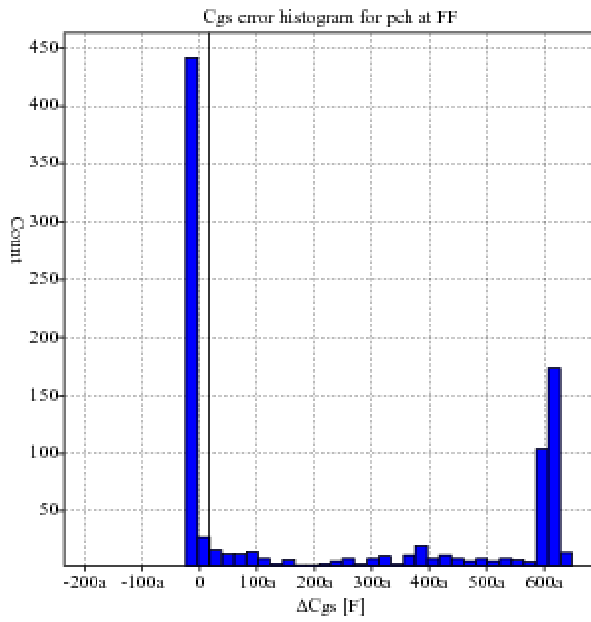


Figure A-32: 40nm PCH TT C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-0.25%	15.976aF
σ	0.432%	68.294aF
Percentile	**% Error**	**Δ Error**
50%	0.254%	16.335aF
75%	0.282%	18.286aF
90%	0.757%	19.701aF
95%	1.103%	84.023aF
99%	1.205%	384.023aF



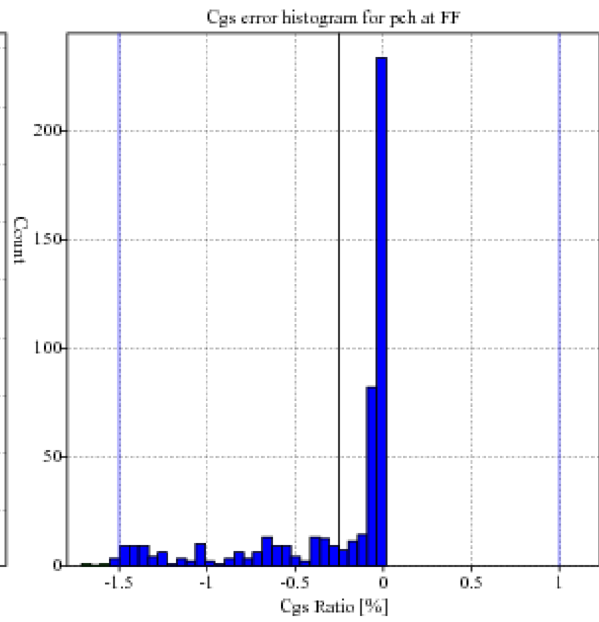
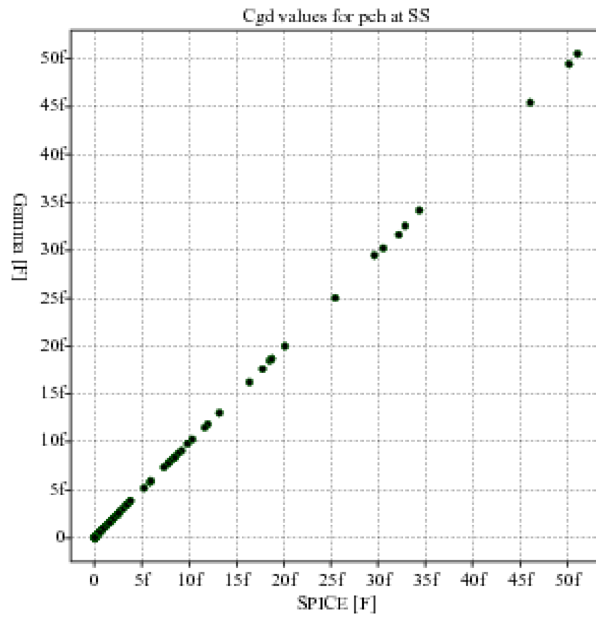
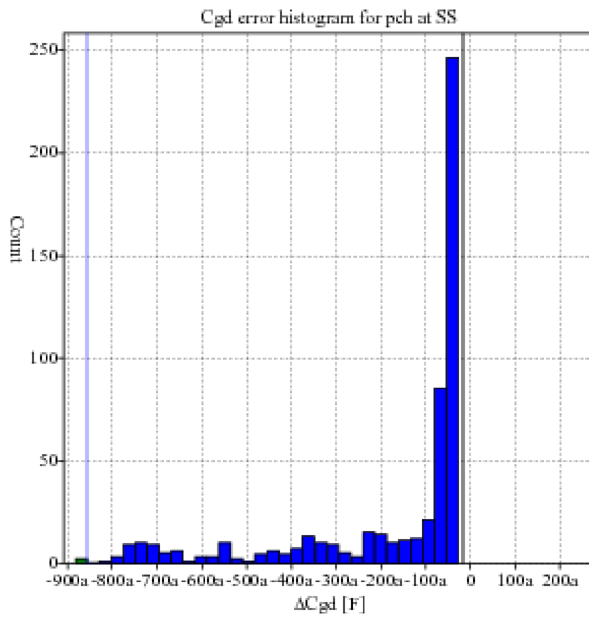


Figure A-33: 40nm PCH FF C_{GS} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.786%	-15.701aF
σ	1.821%	77.182aF
Percentile	**% Error**	**Δ Error**
50%	1.249%	15.258aF
75%	1.611%	15.462aF
90%	3.497%	15.631aF
95%	4.321%	57.438aF
99%	4.539%	376.198aF



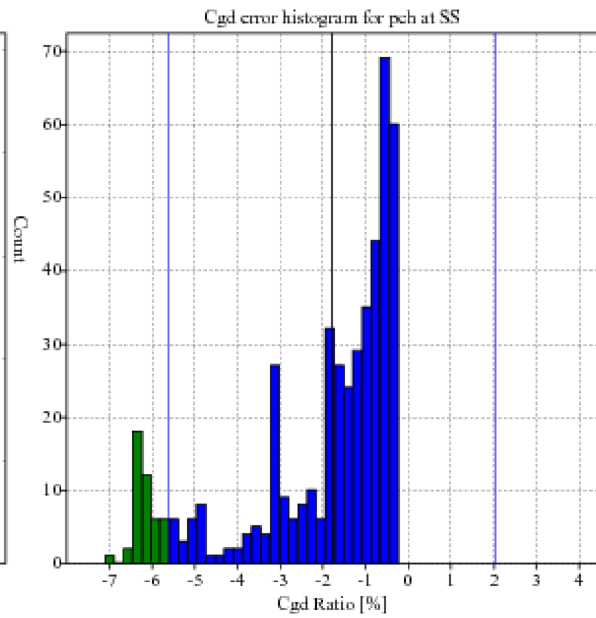
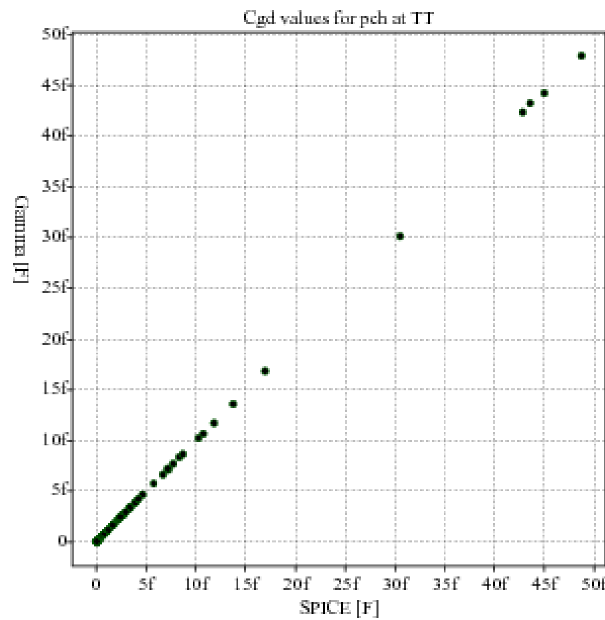
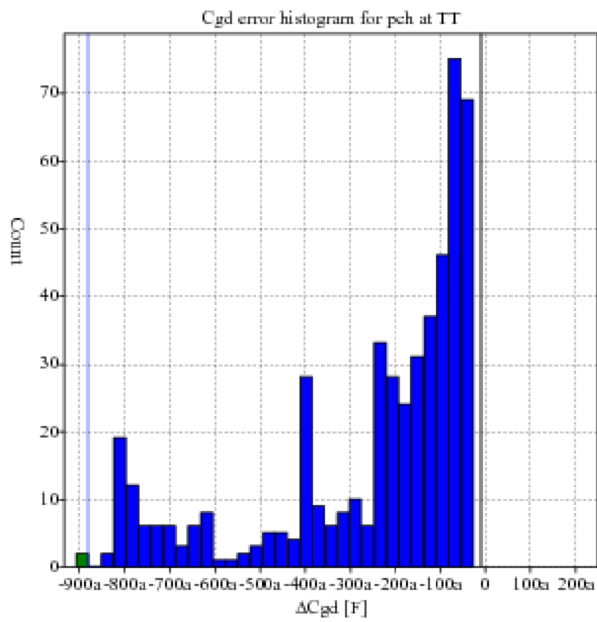


Figure A-34: 40nm PCH SS C_{GD} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.751%	-9.924aF
σ	1.811%	65.793aF
Percentile	**% Error**	**Δ Error**
50%	1.258%	9.465aF
75%	1.640%	9.709aF
90%	3.404%	9.86aF
95%	4.123%	9.91aF
99%	4.616%	388.075aF



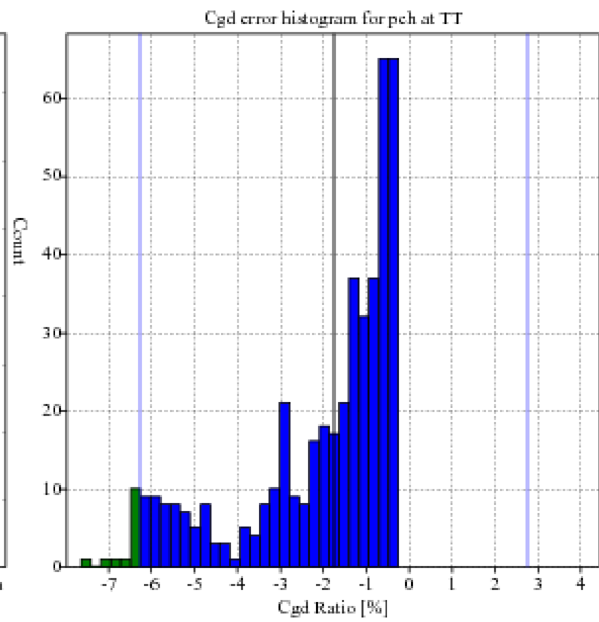
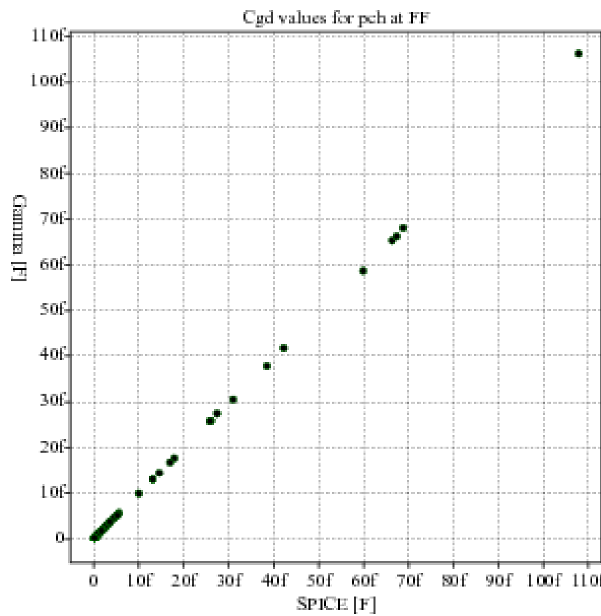
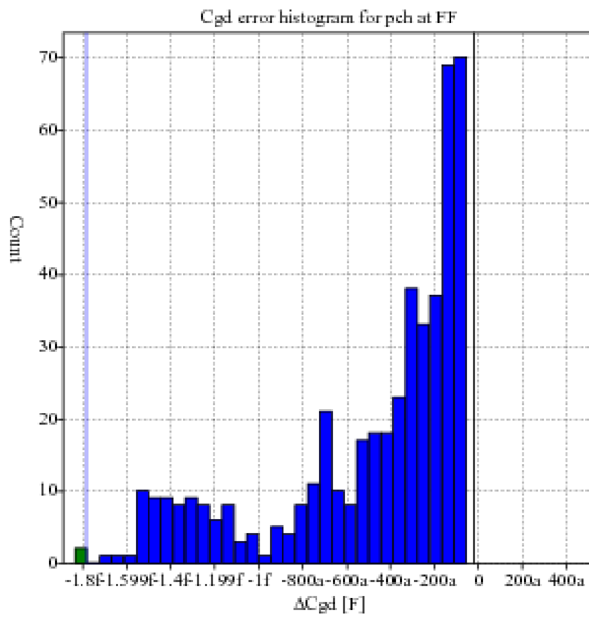


Figure A-35: 40nm PCH TT C_{GD} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-1.573%	-21.582aF
σ	1.760%	141.557aF
Percentile	**% Error**	**Δ Error**
50%	1.205%	21.101aF
75%	1.470%	21.331aF
90%	3.317%	21.496aF
95%	4.226%	21.557aF
99%	4.797%	674.717aF



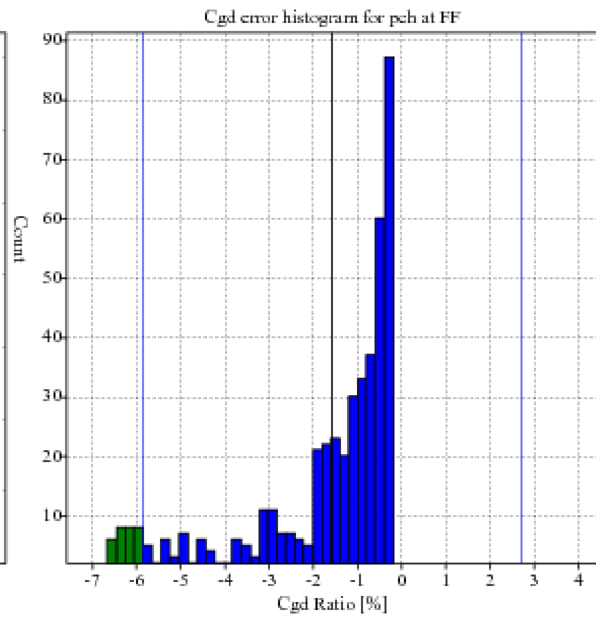
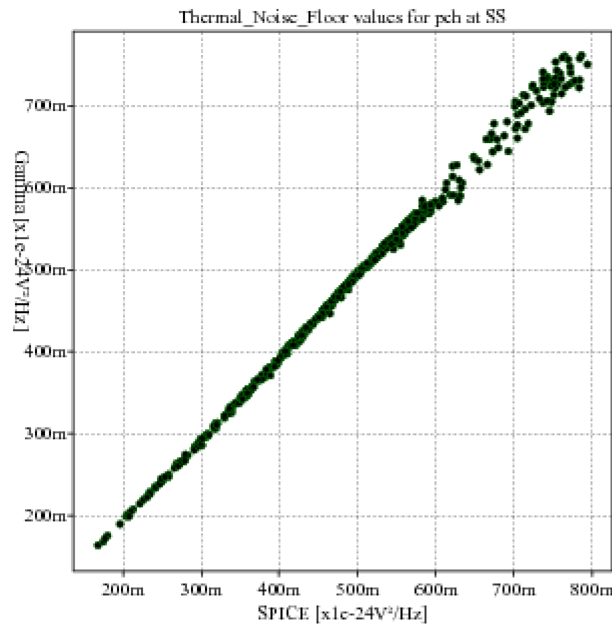
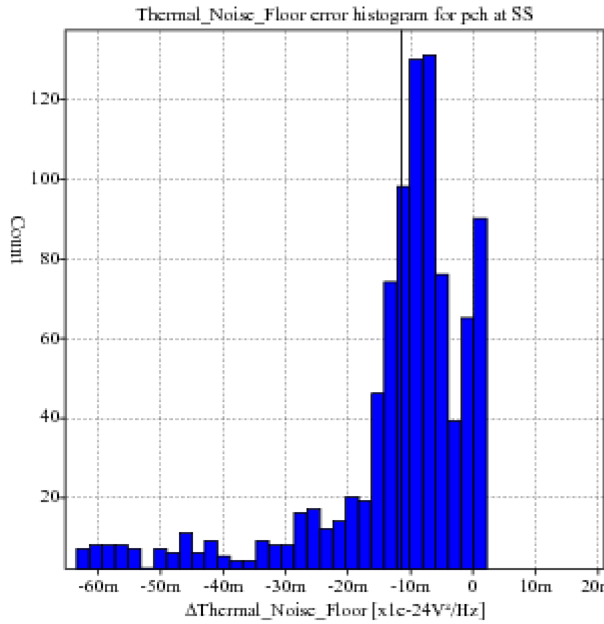


Figure A-36: 40nm PCH FF C_{GD} LUT Queries to SPICE Matching Statistics



Statistics		
Average	-2.354%	-11.43e-27V²/Hz
σ	1.331%	9.176e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	0.897%	4.132e-27V²/Hz
75%	1.379%	6.413e-27V²/Hz
90%	1.895%	12.185e-27V²/Hz
95%	2.763%	20.915e-27V²/Hz
99%	4.276%	35.435e-27V²/Hz



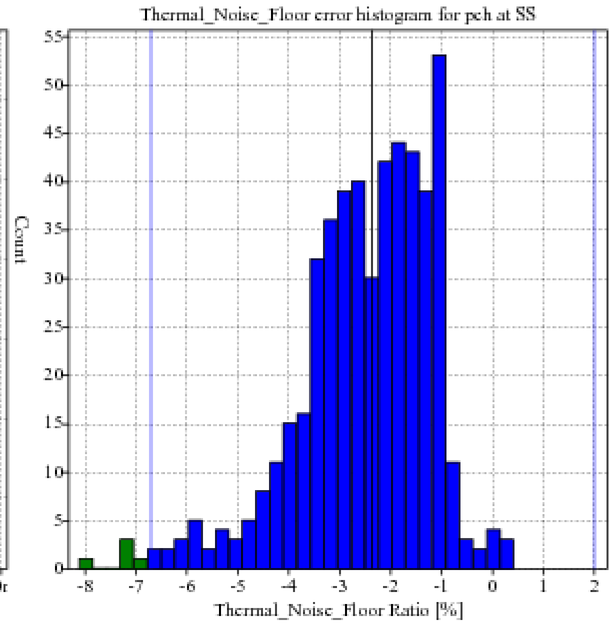
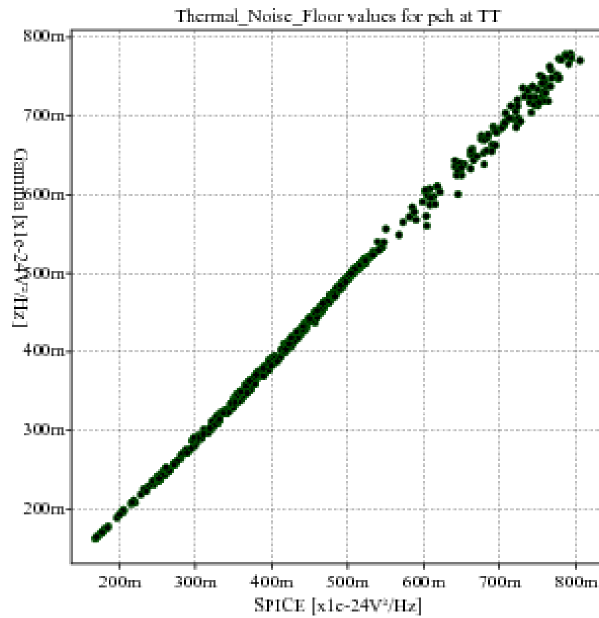
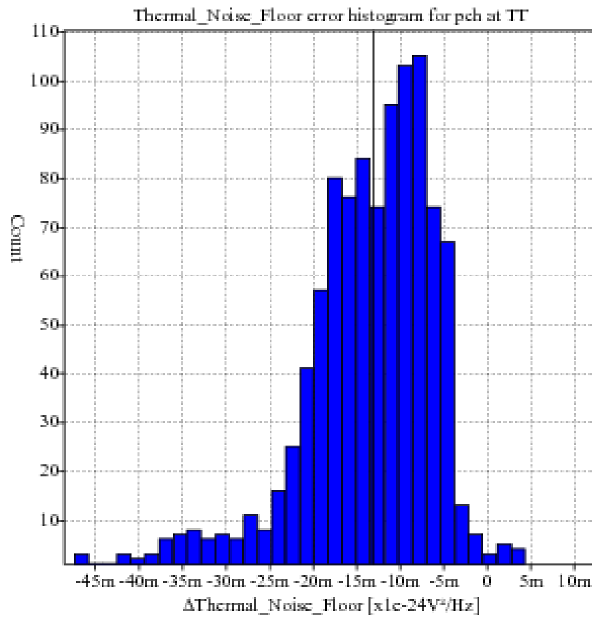


Figure A-37: 40nm PCH SS N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	-3.001%	-13.007e-27V²/Hz
σ	1.605%	7.374e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	1.277%	4.46e-27V²/Hz
75%	1.891%	6.672e-27V²/Hz
90%	2.601%	9.603e-27V²/Hz
95%	3.004%	16.554e-27V²/Hz
99%	3.682%	25.62e-27V²/Hz



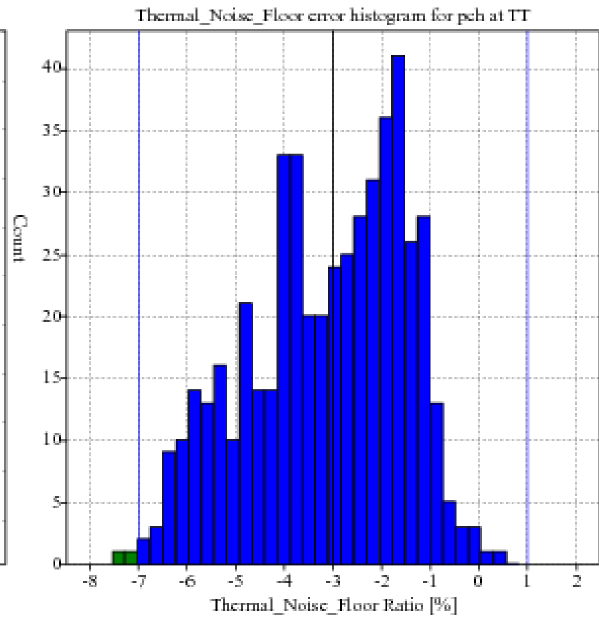
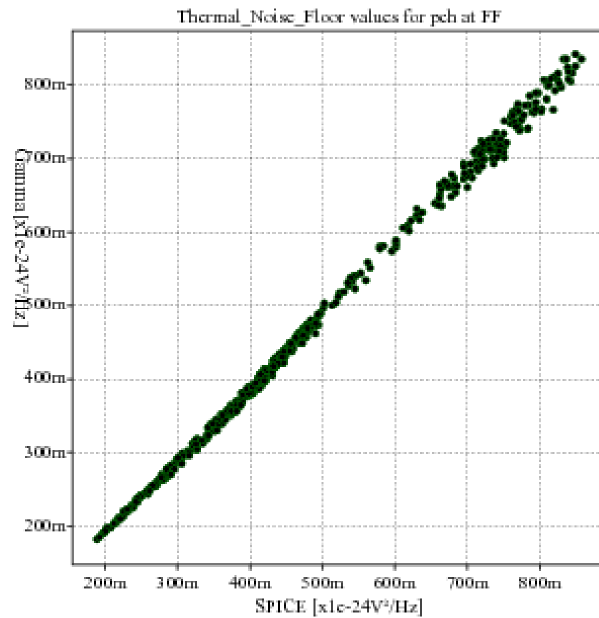
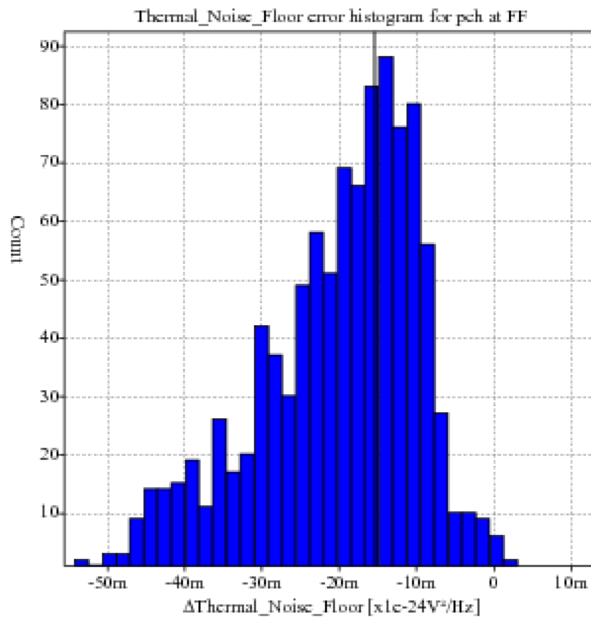


Figure A-38: 40nm PCH TT N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	-3.524%	-15.418e-27V²/Hz
σ	1.744%	8.302e-27V²/Hz
Percentile	**% Error**	**Δ Error**
50%	1.268%	5.382e-27V²/Hz
75%	2.150%	8.016e-27V²/Hz
90%	2.901%	13.665e-27V²/Hz
95%	3.343%	16.942e-27V²/Hz
99%	3.807%	27.311e-27V²/Hz



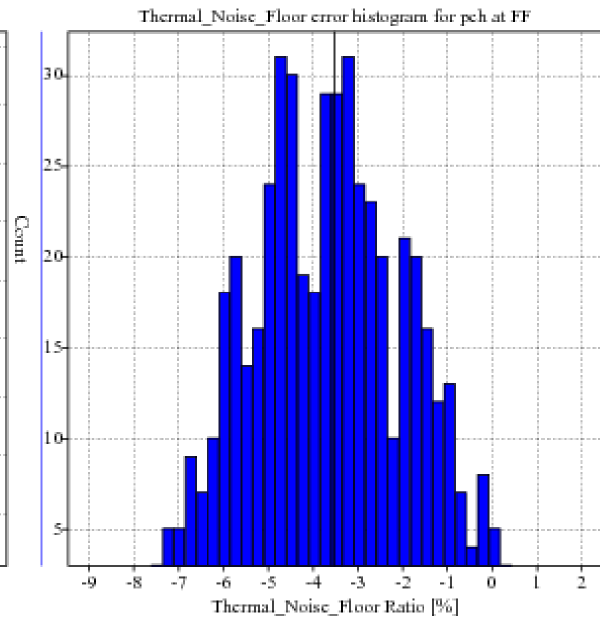
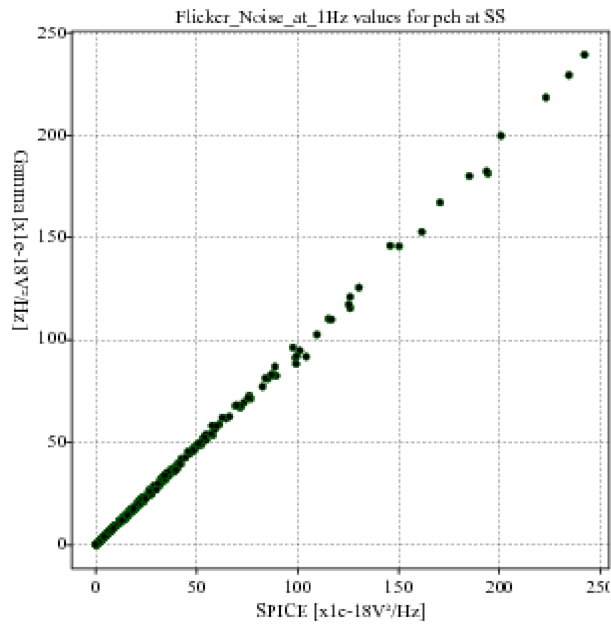
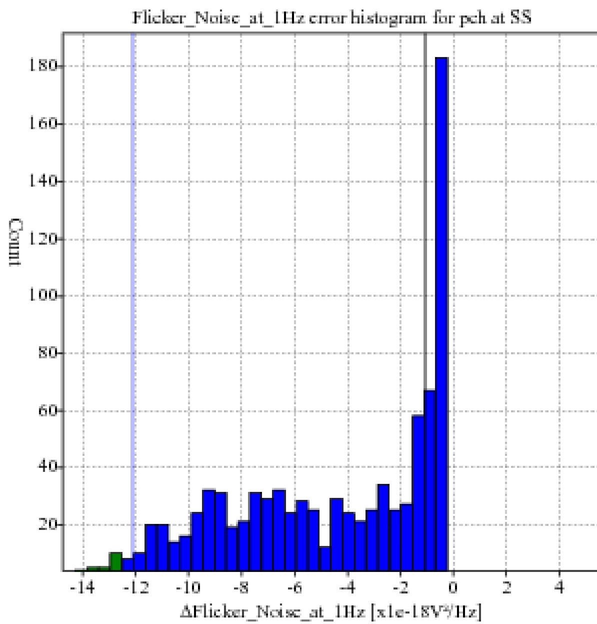


Figure A-39: 40nm PCH FF N_T LUT Queries to SPICE Matching Statistics



Statistics		
Average	-6.203%	-1.061aV²/Hz
σ	2.954%	1.826aV²/Hz
Percentile	**% Error**	**Δ Error**
50%	2.343%	941.923zV²/Hz
75%	3.561%	1.059aV²/Hz
90%	4.667%	1.785aV²/Hz
95%	5.379%	3.343aV²/Hz
99%	6.516%	7.917aV²/Hz



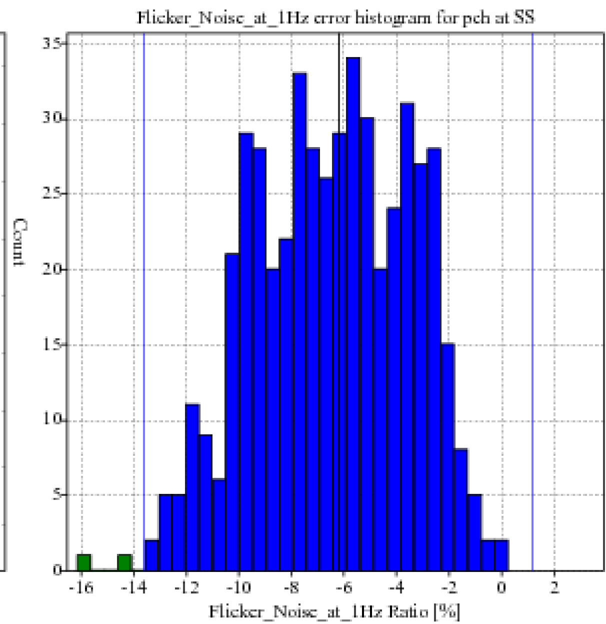
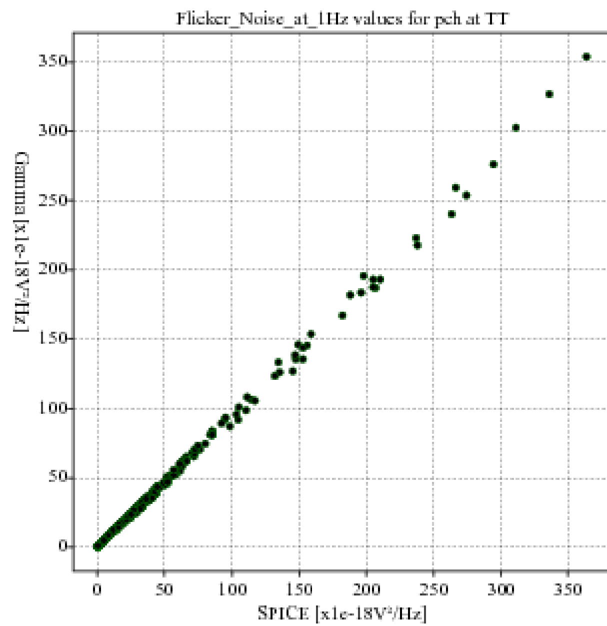
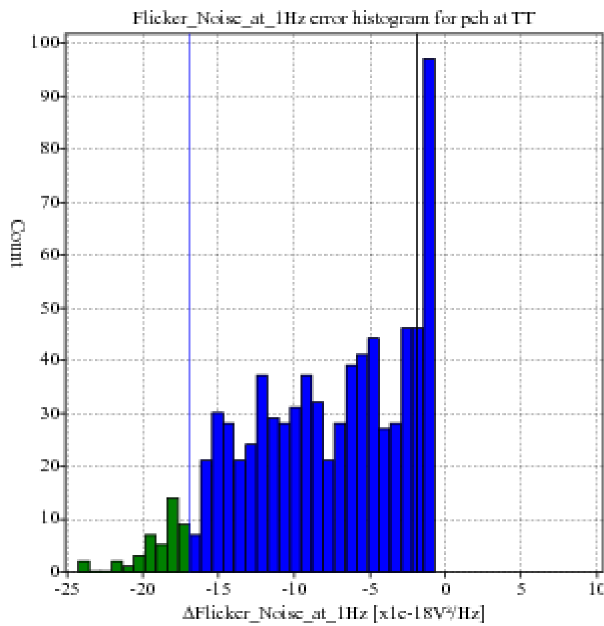


Figure A-40: 40nm PCH SS N_F LUT Queries to SPICE Matching Statistics



Statistics		
Average	-7.489%	-1.911aV²/Hz
σ	3.693%	3.446aV²/Hz
Percentile	**% Error**	**Δ Error**
50%	2.930%	1.702aV²/Hz
75%	4.543%	1.907aV²/Hz
90%	5.709%	3.187aV²/Hz
95%	6.526%	7.105aV²/Hz
99%	8.364%	16.951aV²/Hz



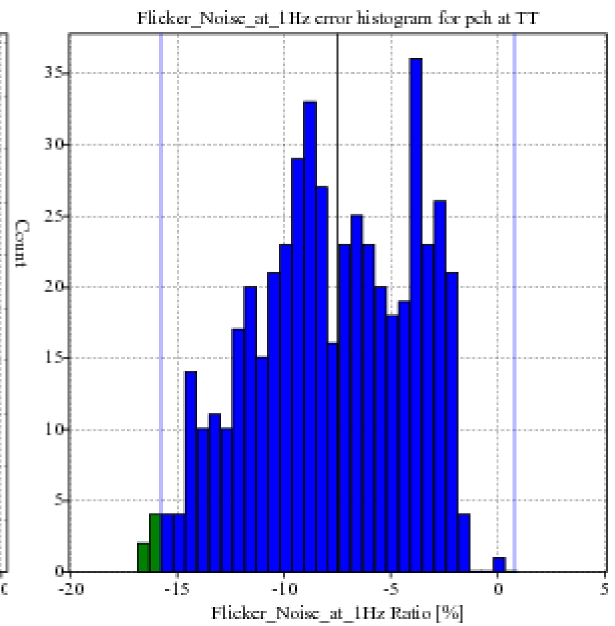
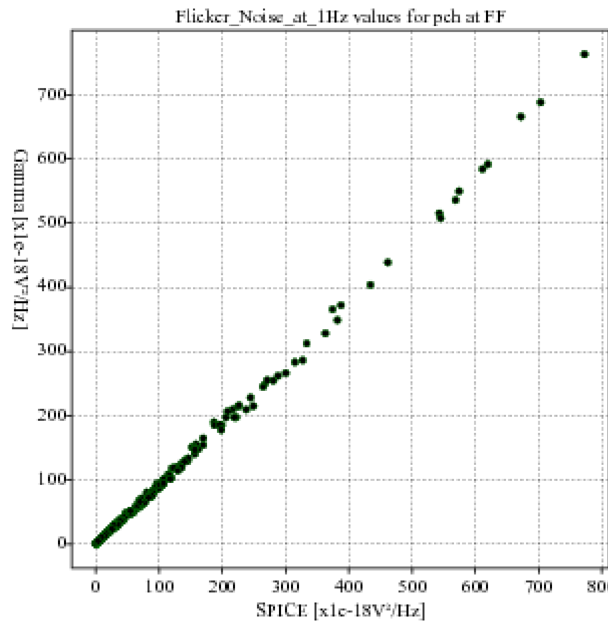
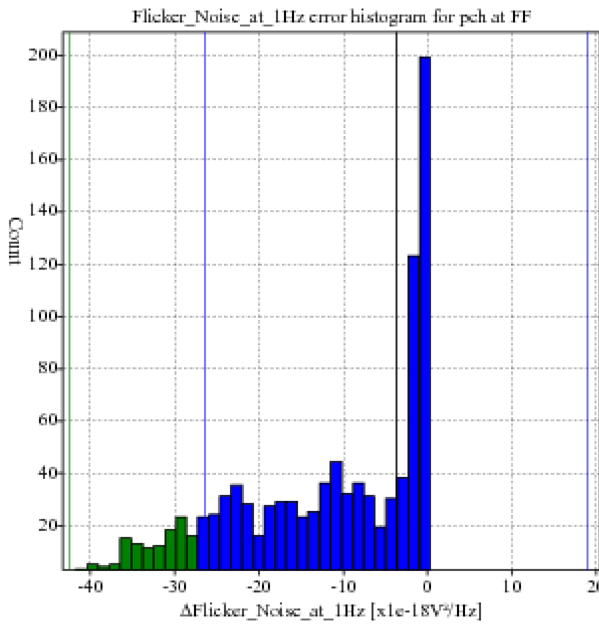


Figure A-41: 40nm PCH SS N_F LUT Queries to SPICE Matching Statistics



Statistics		
Average	-8.818%	-3.731aV²/Hz
σ	3.780%	6.859aV²/Hz
Percentile	**% Error**	**Δ Error**
50%	2.434%	3.516aV²/Hz
75%	4.400%	3.73aV²/Hz
90%	6.661%	7.925aV²/Hz
95%	7.522%	13.728aV²/Hz
99%	8.662%	30.497aV²/Hz



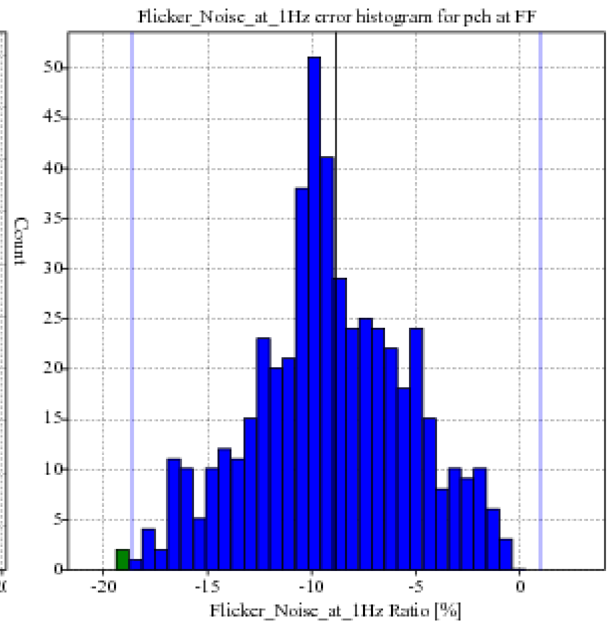


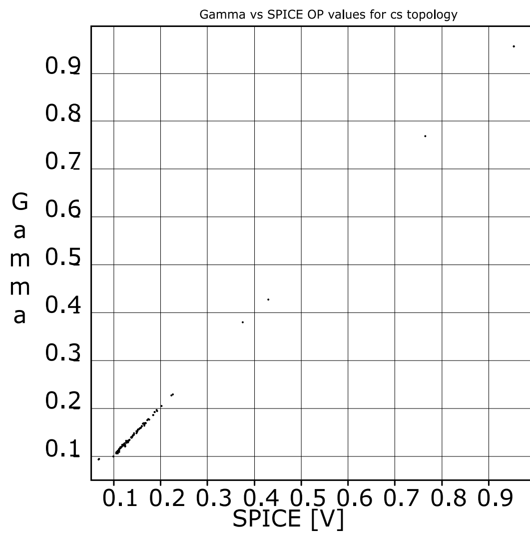
Figure A-42: 40nm PCH FF N_F LUT Queries to SPICE Matching Statistics

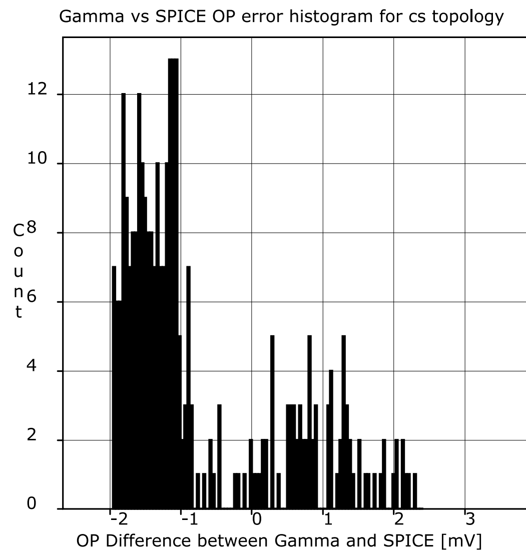
A.2. Circuit Level

The four topologies were tested in TSMC 40nm, in SS corner at 125°C. Each figure shows

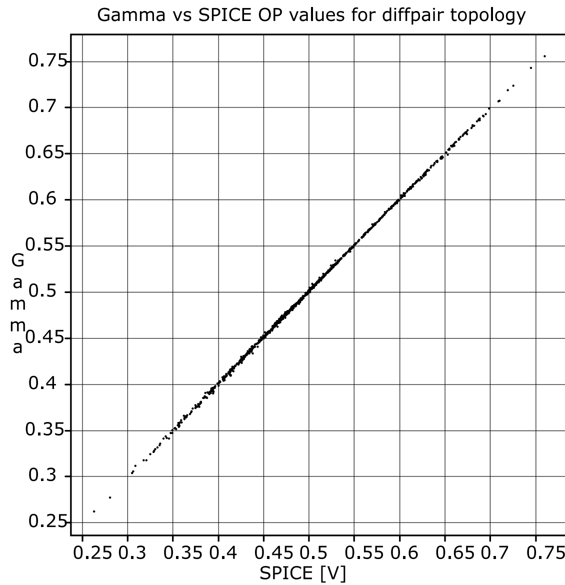
A.2.1. Operating Point

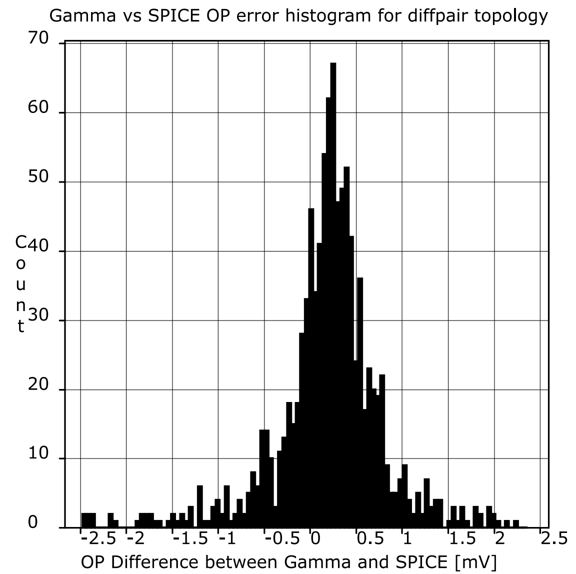
A.2.1.1. Common Source



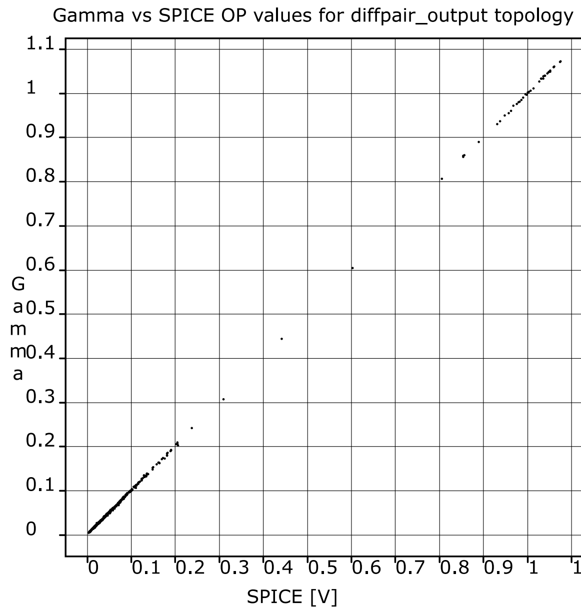


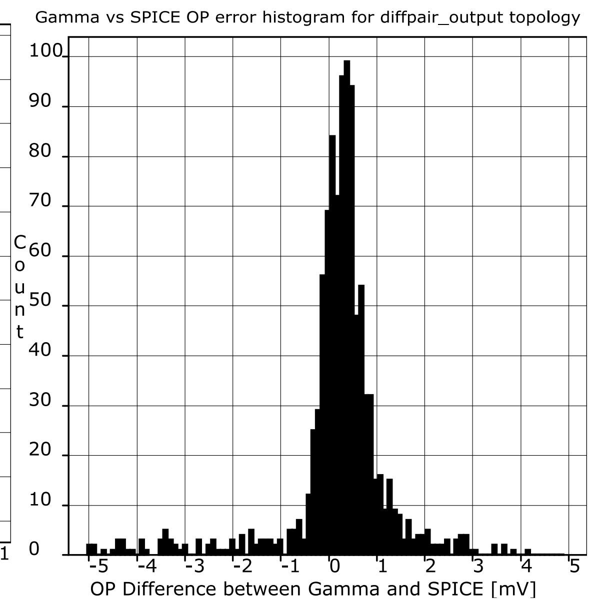
A.2.1.2. Differential Pair



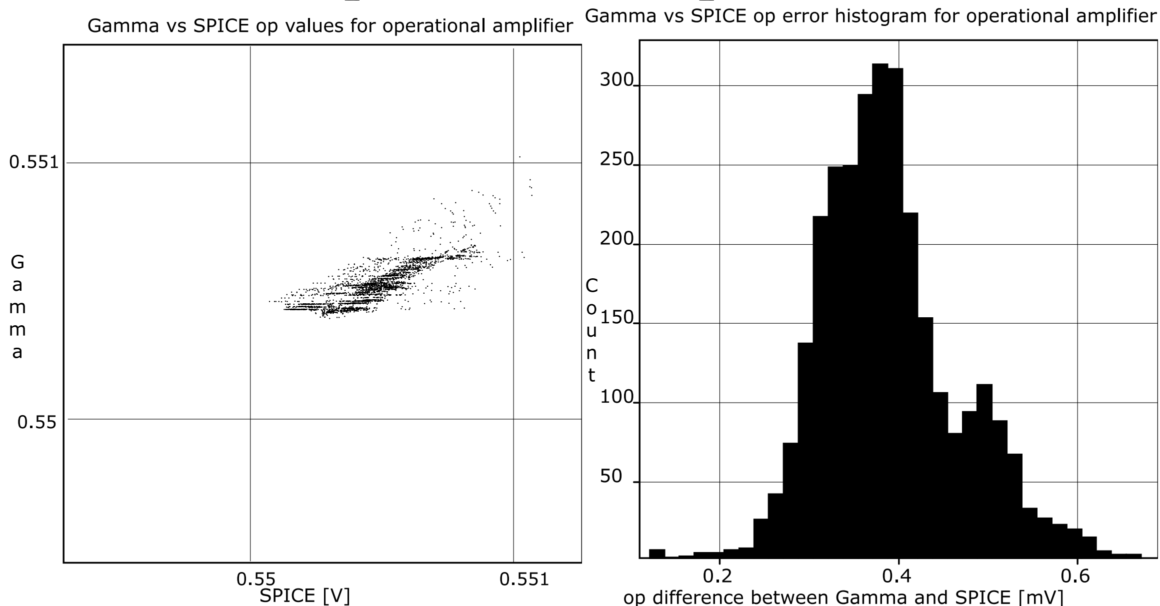


A.2.1.3. Differential Pair with Output Stage





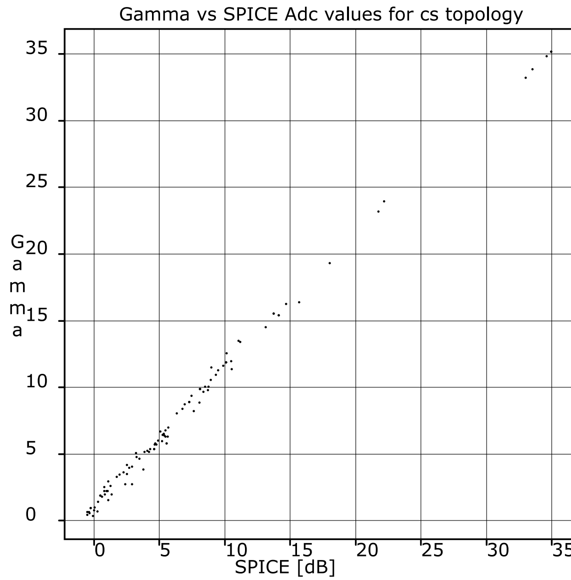
A.2.1.4. Operational Amplifier

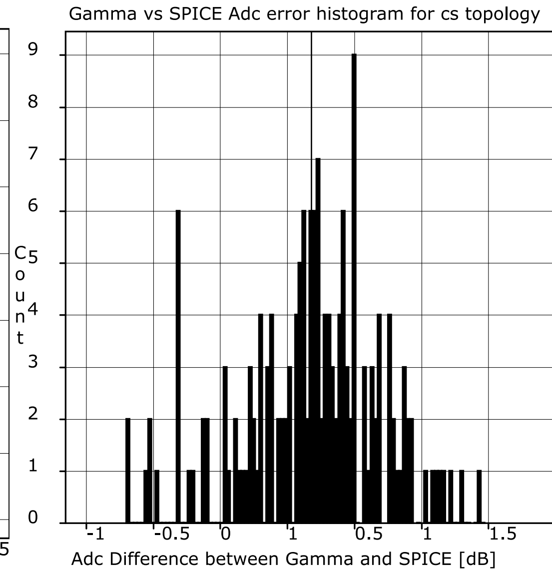


Note: The output DC point of the operational amplifier does not span a large range of values between 0 and V_{DD}, like other circuits do under different geometries. It remains close to 550mV, regardless of its assigned transistor sizes. This is due to its netlist constraint and OP feedback that tunes the output stage's bias voltages to control the output to be always at mid-rail DC level.

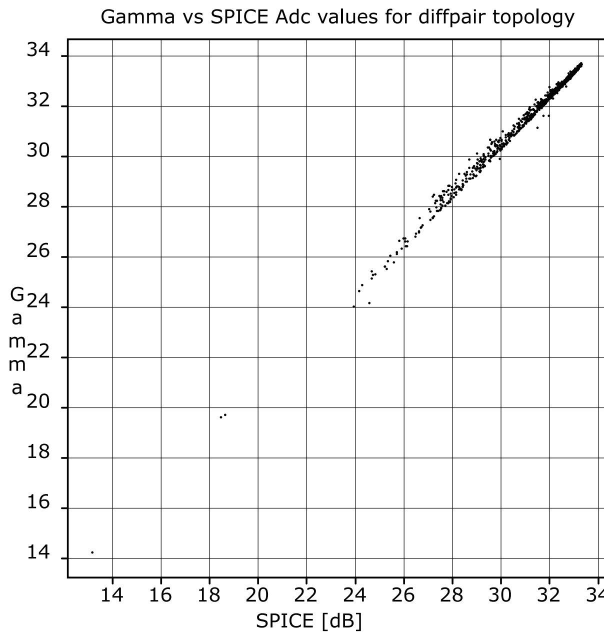
A.2.2. Low Frequency Gain

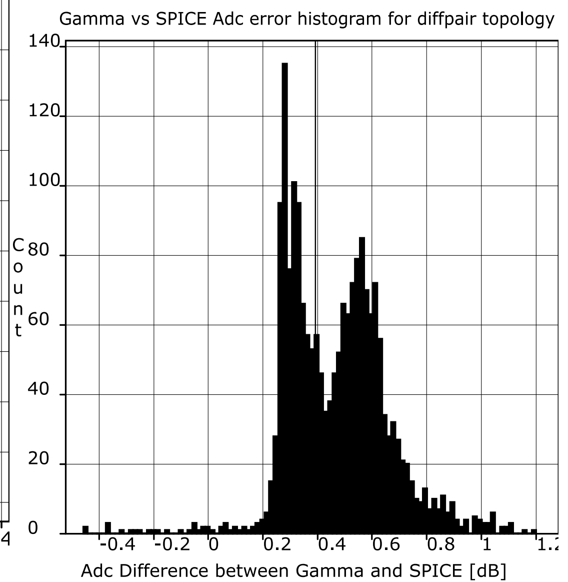
A.2.2.1. Common Source



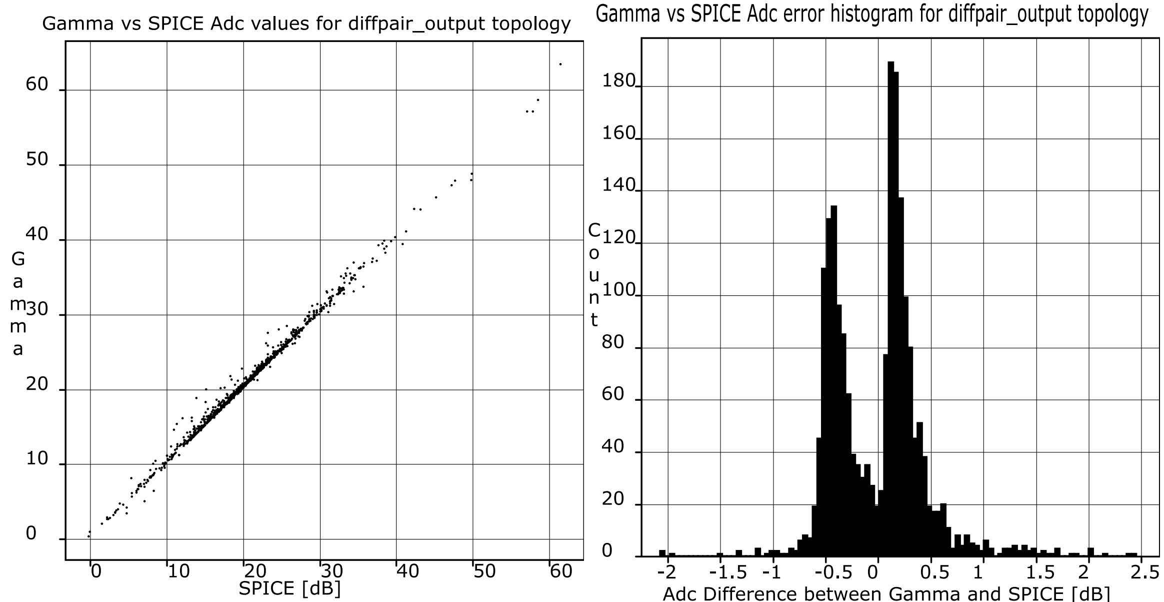


A.2.2.2. Differential Pair

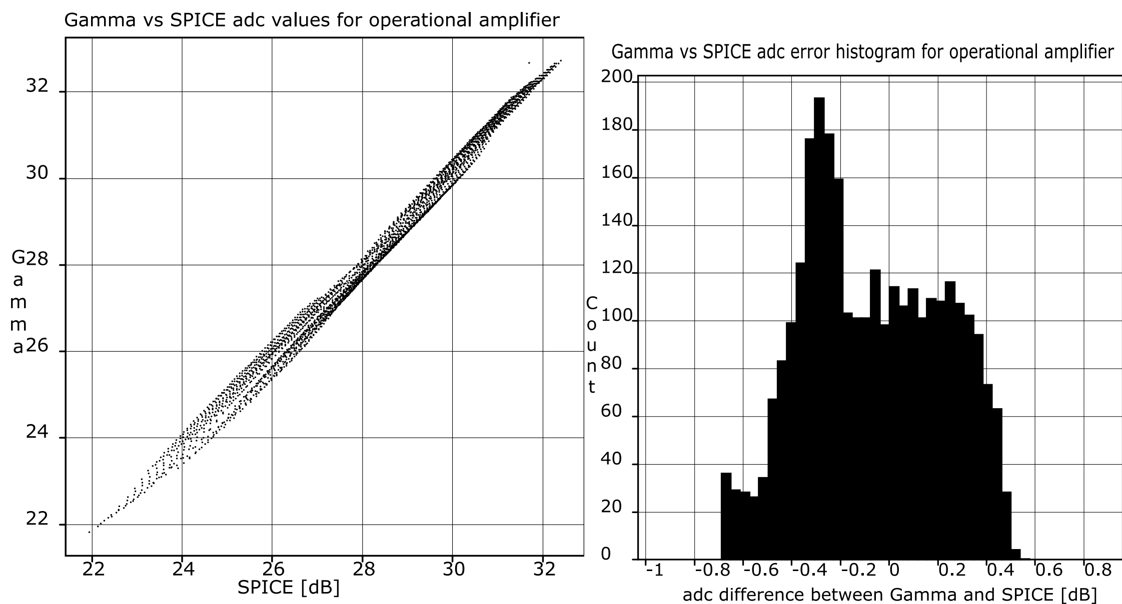




A.2.2.3. Differential Pair with Output Stage



A.2.2.4. Operational Amplifier

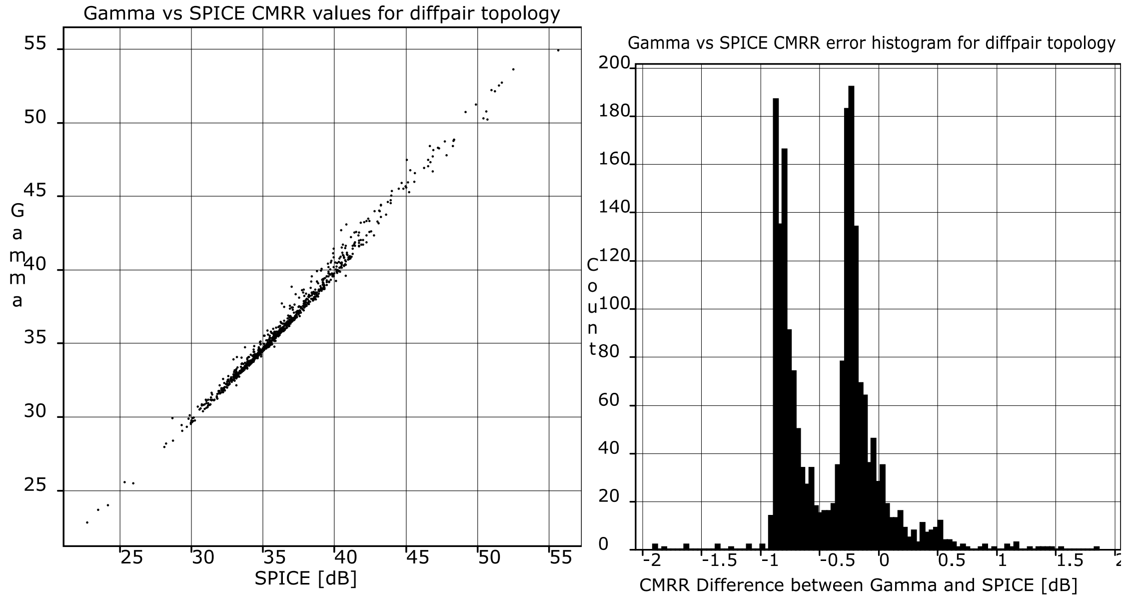


A.2.3. Common Mode Rejection Ratio

A.2.3.1. Common Source

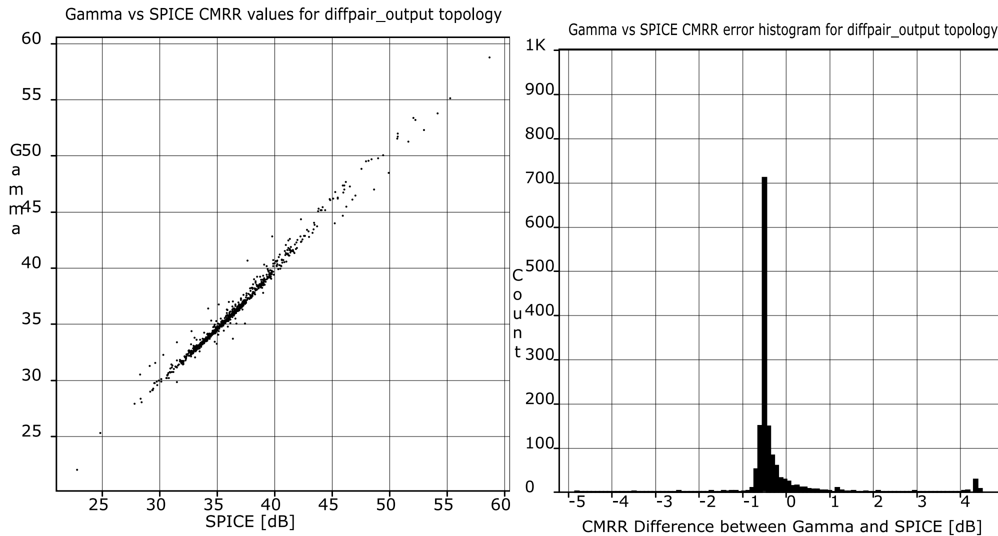
Common-Source amplifiers have no "common-mode", because they have only one input.

A.2.3.2. Differential Pair



An output stage has no effect on performance, CMRR-wise.

A.2.3.3. Operational Amplifier

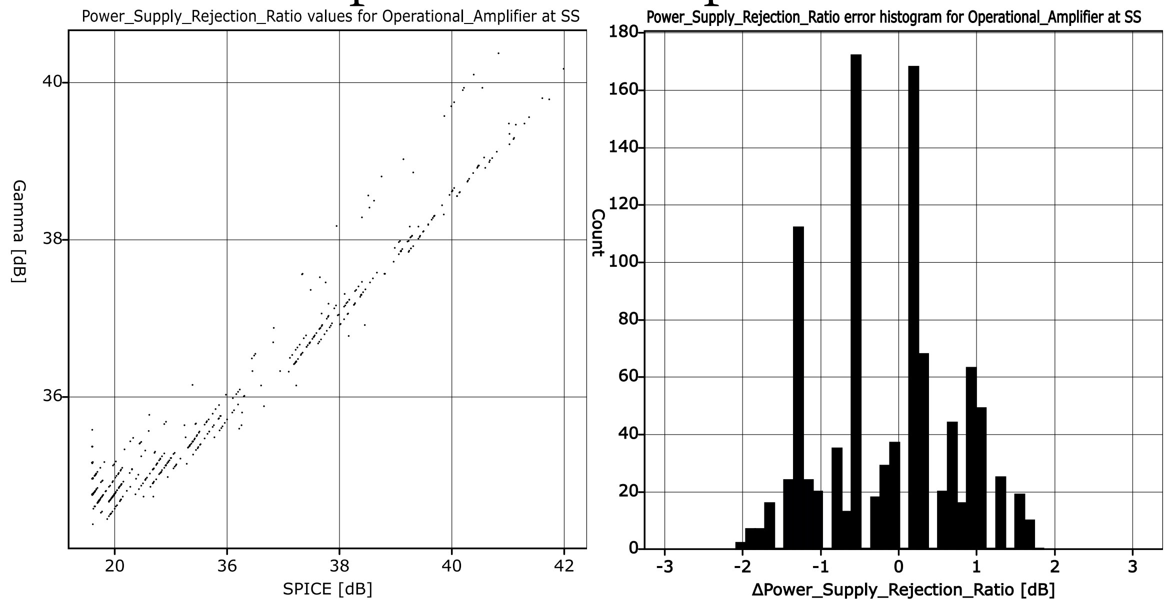


A.2.4. Power Supply Rejection Ratio

A.2.4.1. Common Source

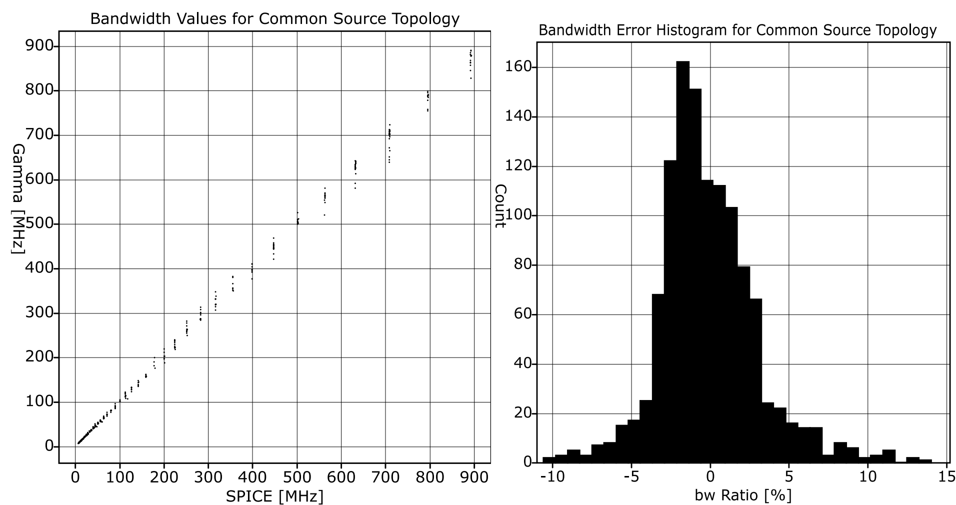
Common-Source amplifiers in this test are powered with an ideal current source.

A.2.4.2. Operational Amplifier

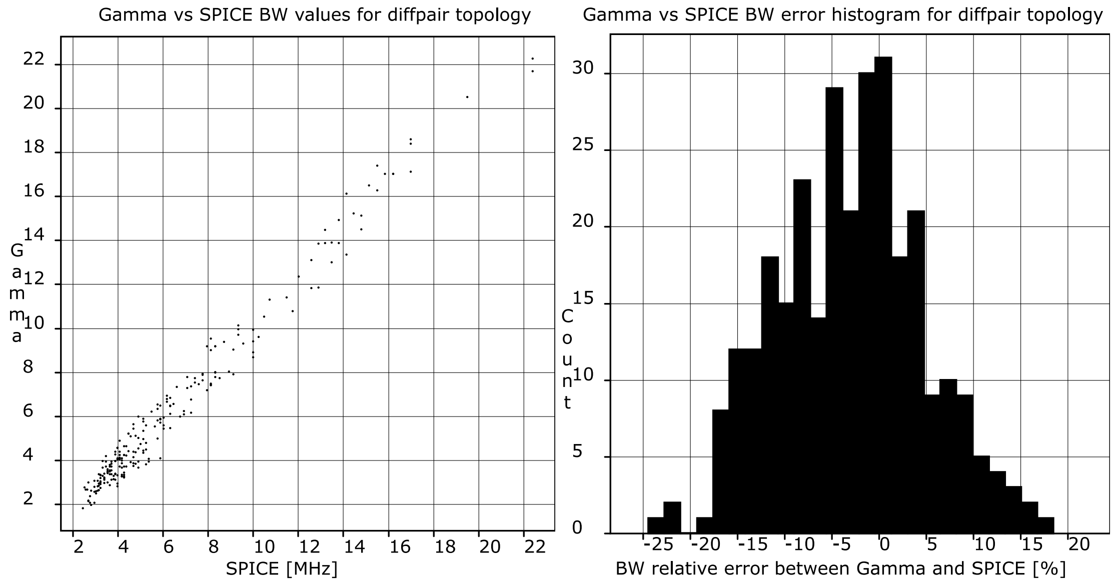


A.2.5. Bandwidth

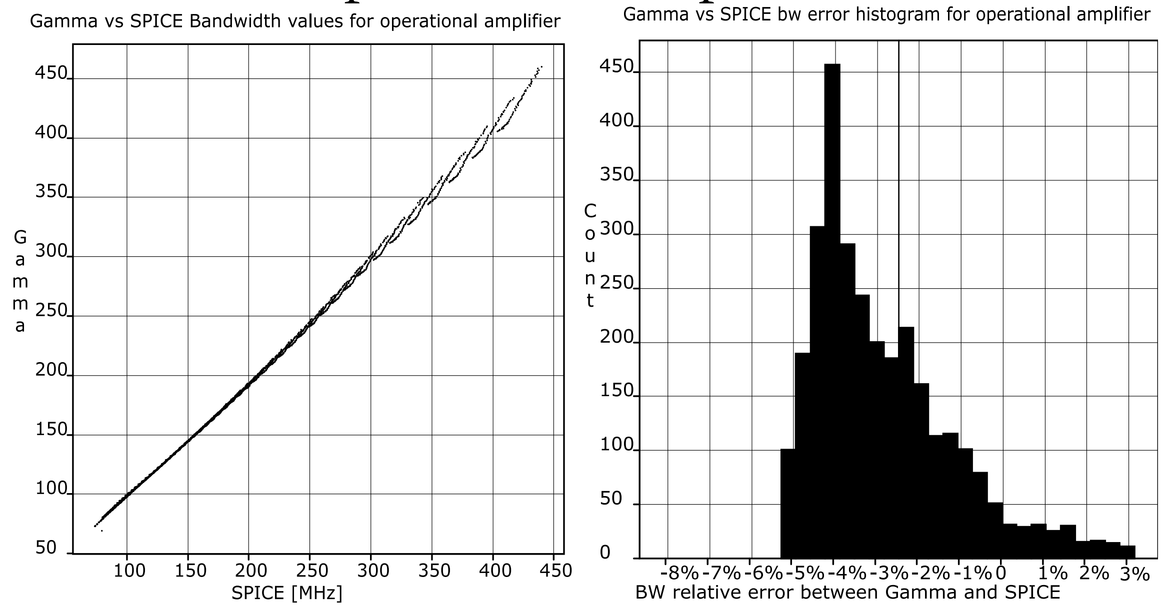
A.2.5.1. Common Source



A. 2. 5. 2. Differential Pair

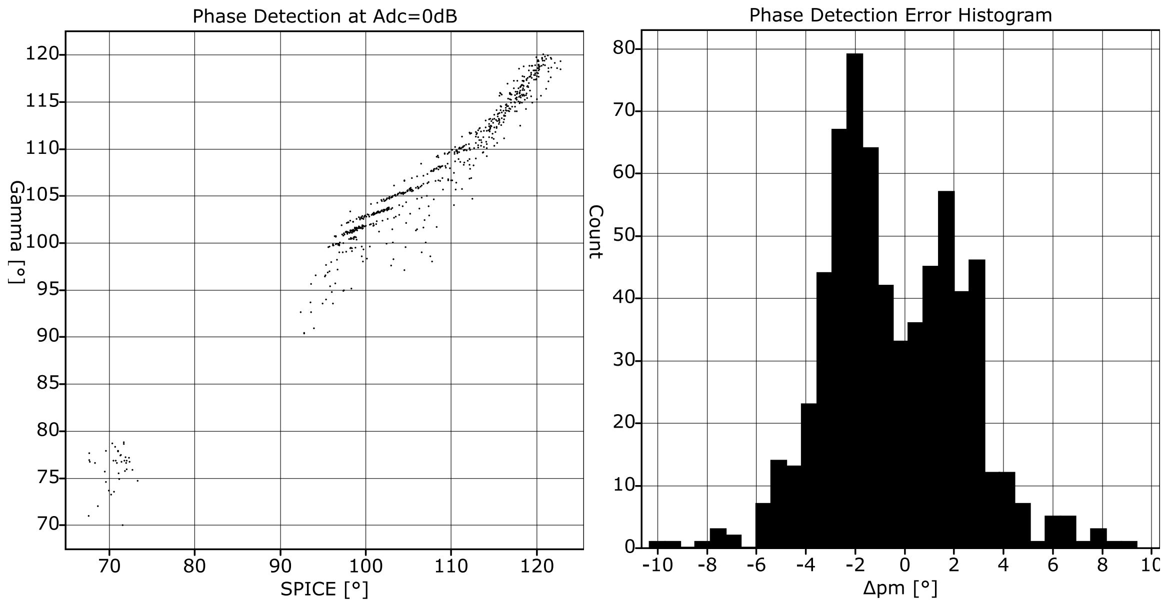


A. 2. 5. 3. Operational Amplifier

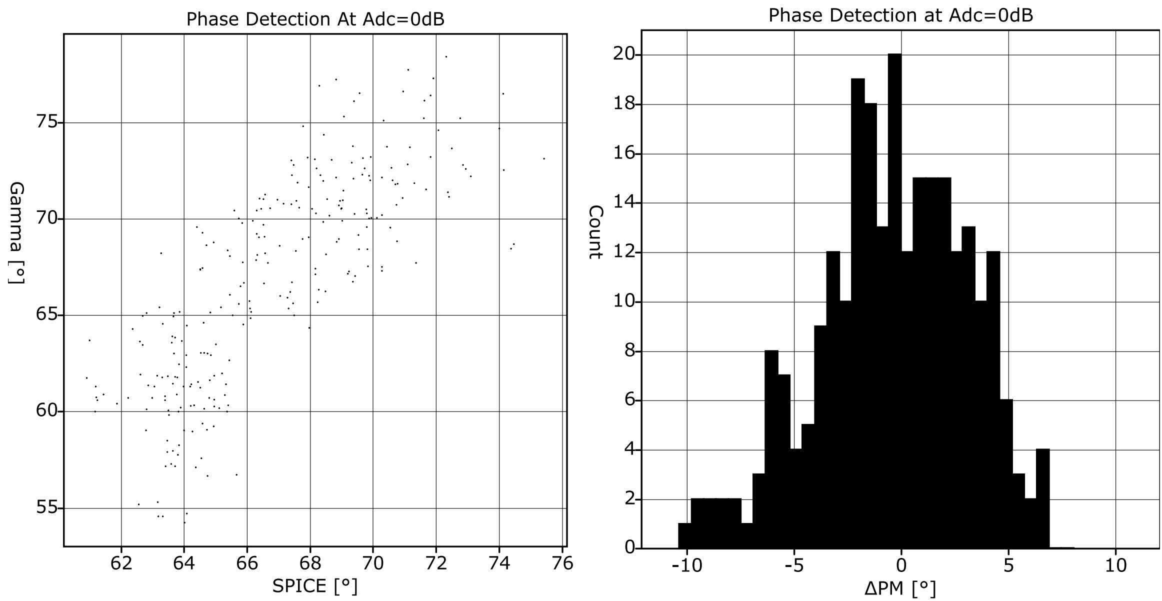


A. 2. 6. Phase Margin

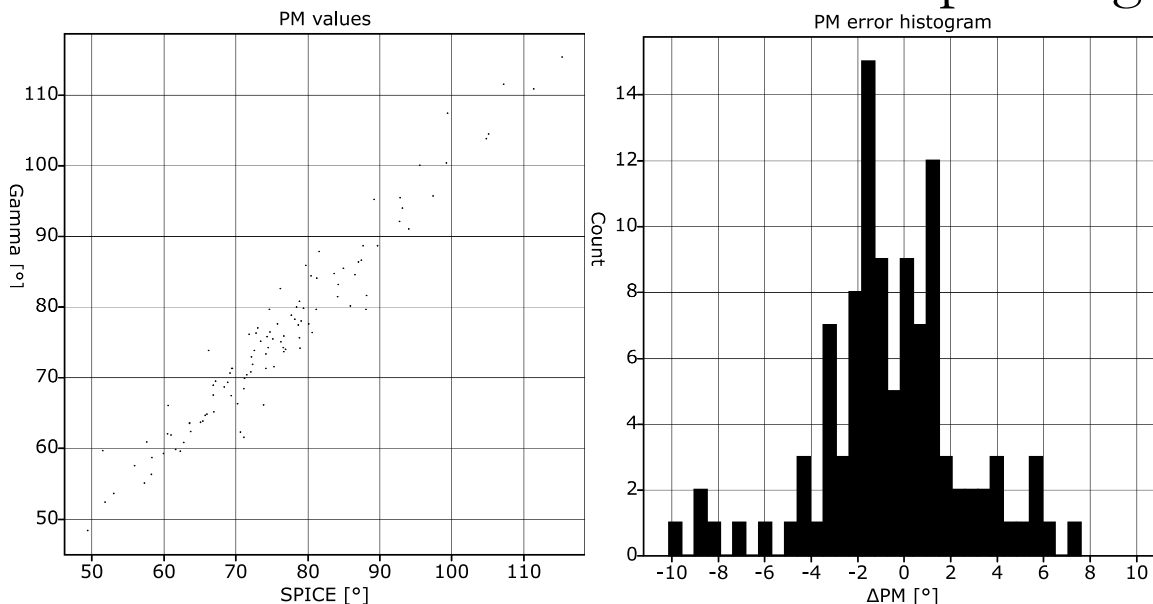
A. 2. 6. 1. Common Source



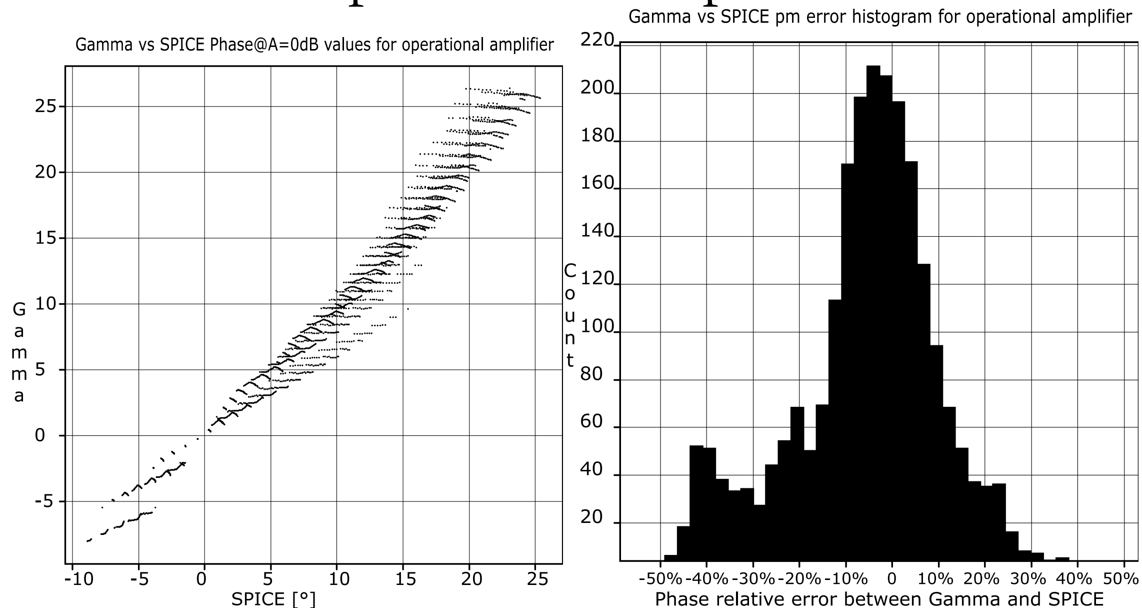
A. 2. 6. 2. Differential Pair



A. 2. 6. 3. Differential Pair with Output Stage



A. 2. 6. 4. Operational Amplifier



B. Source Code

B. 1. Language, Format and Compilation

NGSPICE is the contemporary heir of SPICE2, a C-language rewrite of the original Fortran SPICE. Therefore, the RAMSpice tool is predominantly implemented in C, with its interactive interpreter Tcl. The bulk of the code is still the inherited open-source NGSPICE. However, mining for information inside the original simulator's code reveals the different accumulated layers that were added over the 3+ decades that passed since SPICE2 became public domain. One aspect of the system that seems to have been reinvented with each contribution is messaging (print-outs). Almost every module uses the familiar "info", "warning" and "error" classification, but the mechanism and look-and-feel are unique to each. The first refactoring effort was replacing the messaging commands to a preprocessor-like statements of the following format:

#Info: "a printf pattern using field such as %s, %d etc." string_in integer_in

Trivially, #Warning:, #Error: and #Fatal: are supported the same way. There are debug-mode messages (verbose) that are intuitively named #Dinfo:, #Dwarning:, #Derror:

Messages printed out of Tcl scripts have similar notation, except that they do not need patterns and do not start with the preprocessor leading character '#'. All messages, Tcl and C, trigger the same Tcl code, thus ensure that chronological order of issued messages is kept in log files, unlike the multitude of print mechanisms that raced each other to reach the screen before. The Tcl-consolidated messaging system makes printing a bit "expensive" computation-time-wise. However, the compilation script makes up for this by compiling three different flavors for each

executable:

1. Regular – with all #Info: #Warning: and #Error: messages
2. Debug – with regular plus all #Dinfo:, #Dwarning: and #Derror: messages
3. Silent – quickest execution, with no printed messages

The compilation script is therefore responsible for implementing the new Tcl-preprocessor directives. This infrastructure converts the new preprocessor directives to C first, in a “make”-like mechanism, and then sends the flat C code to the gcc compiler.

Support for original preprocessor directives (herein the “Tcl-preprocessor”) opened the door for additional improvements:

1. In-code support for Tcl variable notation: $DIM – the current function’s dimentionality
2. Embedded Tcl code to manipulate Tcl variables and loops: #tcl set DIM 5
3. For, for-each and while loops to automate code generation and achieve higher abstraction and reusability, without C++ and its templates: #For: {set i 0} {$i<$DIM} {incr i} {…}

The Tcl-preprocessor enabled writing more time-efficient and abstract code for the same coding effort. Examples include:

1. Compilation-time loops and unrolled interpolation sequences that compile to traces, rather than branching loops.
2. Multiple nesting of old NGSPICE analyses that were previously limited to 2 levels
3. Shorthand coding of repetitive interfaces and operations that follow the same lists of variables

The compilation script is therefore an essential part of the source code. It does not use the original Make file distributed with NGSPICE, but modeled after it.

Some of the code downloaded with NGSPICE was discarded for redundancy. For example, since RAMSpice and Γ are meant to be used in a web-service setup, maintaining the X libraries and all the data visualization code that came with it did not make sense. The BLT database that was required by the TclSpice module of NGSPICE is limited to dealing with simulation vectors, became obsolete with cTree and therefore was bypassed in the original code.

B. 2. Data Base

The cTree database was built into the Tcl interpreter's command table. The top-level command is listed below as tcl_ctree and registered in the interpreter as '@', for easy identification in the Tcl code. Each cTree command in Tcl expects the '@' bullet, followed by a context string, a sub-command and optional arguments.

Example: @ /parameters/height = 3.4 assigns 3.4 to context 'parameters', sub-context 'height'. The context is parameters/height, the sub-command is '=' and the argument is 3.4.

Example: @ /parameters foreach_child c {

puts "$c=[@ parameters/$c]"

}

prints all sub-contexts (= children) of parameters and their values.

Context notation borrows from UNIX file-system notation. The root of the tree is forward-slash '/'. A context path is accessed via forward-slash delimited list. Similar to working-

directory in UNIX (pwd), there is a working-context, which can be set by the cd sub-command and referenced by relative context.

Example: @ /parameters cd changes the working context to parameters

Example: @ height = 3.4 sets /parameters/height to 3.4

Example: @ ../changed_parameter = 1 sets a context next to parameters to 1

Although intuitive and familiar, the forward-slash separator is problematic when used within ΓCC templates, because it gets confused by expression analysis with division operator. Therefore, both forward-slash and colon ':' are supported as separators. The former is used in most Tcl scripts, while the latter is mandated in ΓCC templates.

The structure of contexts (tree nodes) is:

```
struct context {
    context **children; // array of sub-contexts
    context *parent;    // pointer to the super-context
    ordinal num_of_children;
    ordinal max_num_of_children; // Current allocation size of the children array
    ordinal sibling_order;  // 0 - first-born, 1 - second child, etc.
    CTYPE value_type;    // Enumeration of context type
    so_union value;     // value, if fits in 64b field
    char *name;
};
```

The '@' bullet triggers the following top-level command:

static int

```
tcl_ctree (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[])

{

    if (argc<2) {

        #Error: "(ctree) got no context"

        return TCL_ERROR;

    }

    Tcl_ResetResult(interp);

    context *c=Context;

    float *array_entry;

    if ((argv[1][0]=='/')||(argv[1][0]==':')) {

        c=ctree;

    }

    // "exists" and "create" require an exception to the rule other commands require,
that the context given as 1st arg must be valid.

    if (argc==3) {

      if
((strcmp(argv[2],"exists")==0)||(strcmp(argv[2],"?")==0)||(strcmp(argv[2],"exists?")==0)) {

            tcl_append_int(interp,resolve_context(argv[1],&c,&array_entry));

            return TCL_OK;

        }

      if
((strcmp(argv[2],"!")==0)||(strcmp(argv[2],"create")==0)||(strcmp(argv[2],"exists!")==0)) {

            create_context(argv[1]);

            return TCL_OK;

        }

    }

    if (!(resolve_context(argv[1],&c,&array_entry))) {

        create_context(argv[1]);

        resolve_context(argv[1],&c,&array_entry);

    }

    if (argc==2) {

        // simple value return
```

```
if (c->value_type==ctype_POLY) {
    tcl_append_float(interp,calc_POLY(c->value.v));
    return TCL_OK;
}
if (c->value_type==ctype_void) {
    return TCL_OK;
}
if (c->value_type==ctype_string) {
    Tcl_AppendElement(interp,(char *)c->value.v);
    return TCL_OK;
}
if (c->value_type==ctype_real) {
    tcl_append_float(interp,c->value.s);
    return TCL_OK;
}
if (c->value_type==ctype_integer) {
    tcl_append_int(interp,c->value.o);
    return TCL_OK;
}
if (c->value_type==ctype_LUT) {
    if (array_entry==NULL) {
        #Error: "(ctree) invalid array access %s" argv[1]
        return TCL_ERROR;
    }
    tcl_append_float(interp,*array_entry);
    return TCL_OK;
}
if (c->value_type==ctype_PAT) {
    PAT *p=(PAT *)c->value.v;
    tcl_append_int(interp,p->content->num_of);
```

```
                return TCL_OK;
            }
            #Error: "(ctree) ccontext has unrecognized value_type. (%d)" c->value_type
            return TCL_ERROR;
        }
        // identify context’s type
        if (strcmp(argv[2],"type")==0) {
            tcl_append_int(interp,c->value_type);
            return TCL_OK;
        }
    /****
       POLY support
    ****/
        // return a polynomial expression string
        if (strcmp(argv[2],"expression")==0) {
            if (c->value_type!=ctype_POLY) {
                #Error: "(ctree) The expression command is to be used with a polynomial
only."
                return TCL_ERROR;
            }
            if (argc!=3) {
                #Error: "(ctree) The expression command takes no arguments"
                return TCL_ERROR;
            }
            if (c->value_type!=ctype_POLY) {
                #Error: "(ctree) The expression command is to be used with a polynomial
only."
                return TCL_ERROR;
            }
            POLY *p=(POLY *)c->value.v;
            Tcl_AppendElement(interp,p->expression);
```

```
    return TCL_OK;
}
// set a denominator for a rational function
if (strcmp(argv[2],"denom")==0) {
    if (c->value_type!=ctype_POLY) {
        #Error: "(ctree) The denom command is to be used with a polynomial only."
        return TCL_ERROR;
    }
    if (argc<4) {
        #Error: "(ctree) The denom command requires a polynomial"
        return TCL_ERROR;
    }
    POLY *nom=get_POLY(argv[1]);
    nom->denom=get_POLY(argv[3]);
    return TCL_OK;
}
// calculate a polynomial in a derivative mode
if (strcmp(argv[2],"derive")==0) {
    if (c->value_type!=ctype_POLY) {
        #Error: "(ctree) The derive command is to be used with a polynomial only."
        return TCL_ERROR;
    }
    if (argc<4) {
        #Error: "(ctree) The derive command requires a by-variable"
        return TCL_ERROR;
    }
    context *by=Context;
    float *array_entry;
    if ((argv[3][0]=='/')||(argv[3][0]==':')) {
        by=ctree;
```

```
            }
            float *array_context;
            if (!resolve_context(argv[3],&by,&array_context)) {
                #Warning: "(ctree) The derive command was given a non-existent context %s"
argv[3]
                tcl_append_float(interp,0);
                return TCL_OK;
            }
            void *by_var=&(by->value.s);
            tcl_append_float(interp,derive_POLY(c->value.v,by_var));
            return TCL_OK;
        }
        // Apply Newton-Raphson to a polynomial
        if (strcmp(argv[2],"root")==0) {
            if (c->value_type!=ctype_POLY) {
                #Error: "(ctree) The root command is to be used with a polynomial only."
                return TCL_ERROR;
            }
            if (argc<4) {
                #Error: "(ctree) The root command requires a by-variable"
                return TCL_ERROR;
            }
            float init=0;
            if (argc==5) init=strtod(argv[4],NULL);
            context *by=Context;
            float *array_entry;
            if ((argv[3][0]=='/')||(argv[3][0]==':')) {
                by=ctree;
            }
            float *array_context;
            if (!resolve_context(argv[3],&by,&array_context)) {
```

```
                #Warning: "(ctree) The root command was given a non-existent context %s"
argv[3]

                tcl_append_float(interp,0);

                return TCL_OK;

            }

            void *by_var=&(by->value.s);

            tcl_append_float(interp,root_POLY(c->value.v,by_var,init));

            return TCL_OK;

        }

        // Implicit derivative (derive one var w.r.t another, within the polynomial)

        if (strcmp(argv[2],"imp_derive")==0) {

            if (c->value_type!=ctype_POLY) {

                #Error: "(ctree) The root command is to be used with a polynomial only."

                return TCL_ERROR;

            }

            if (argc<5) {

                #Error: "(ctree) The implicit derivative command requires two by-variables"

                return TCL_ERROR;

            }

            float init=0;

            if (argc==6) init=strtod(argv[5],NULL);

            float *array_context;

            context *by=Context;

            if ((argv[3][0]=='/')||(argv[3][0]==':')) {

                by=ctree;

            }

            if (!resolve_context(argv[3],&by,&array_context)) {

                #Warning: "(ctree) The root command was given a non-existent context %s"
argv[3]

                tcl_append_float(interp,0);

                return TCL_OK;
```

```
            }
            void *by_var=&(by->value.s);
            context *root=Context;
            if ((argv[4][0]=='/')||(argv[4][0]==':')) {
                root=ctree;
            }
            if (!resolve_context(argv[4],&root,&array_context)) {
                #Warning: "(ctree) The root command was given a non-existent context %s"
argv[4]
                tcl_append_float(interp,0);
                return TCL_OK;
            }
            void *root_var=&(root->value.s);
            tcl_append_float(interp,imp_derive_POLY(c->value.v,by_var,root_var,init));
            return TCL_OK;
        }
    /****
       PAT support
    ****/
        if (strcmp(argv[2],"PAT")==0) {
            if (c->value_type!=ctype_PAT) {
                #Error: "(ctree) The PAT command is to be used with a pareto-associative
table only. Use double parentheses to declare one: @ PAT((size1,size2|prop1,prop2)) !"
                return TCL_ERROR;
            }
            if (argc<4) {
                #Error: "(ctree) The PAT command requires a sub-command: size, index,
delete, foreach"
                return TCL_ERROR;
            }
            PAT *p=(PAT *)c->value.v;
            if (strcmp(argv[3],"size")==0) {
```

```
          tcl_append_int(interp,p->content->num_of);
          return TCL_OK;
      }
      // List the performance properties
      if (strcmp(argv[3],"properties")==0) {
          ordinal i;
          for (i=0;i<p->properties->num_of;i++) Tcl_AppendElement(interp,p-
>properties->content[i]);
          return TCL_OK;
      }
      // List the sizing parameters
      if (strcmp(argv[3],"sizes")==0) {
          ordinal i;
          for (i=0;i<p->sizes->num_of;i++) Tcl_AppendElement(interp,p->sizes-
>content[i]);
          return TCL_OK;
      }
      // List margins for similarity algorithm
      if (strcmp(argv[3],"margins")==0) {
          ordinal i;
          for (i=0;i<p->margins->num_of;i++) tcl_append_float(interp,p->margins-
>content[i]);
          return TCL_OK;
      }
      // access entry at index
      if (strcmp(argv[3],"index")==0) {
          if (argc!=5) {
              #Error: "(ctree) The PAT index sub-command requires an index"
              return TCL_ERROR;
          }
          ordinal i,j;
          j=atoi(argv[4]);
```

```
        for (i=0;i<p->sizes->num_of;i++)
           tcl_append_float(interp,p->content->content[j]->sizes->content[i]);
        for (i=0;i<p->properties->num_of;i++) {
            float value=\
      p->factors->content[i]*p->content->content[j]->properties->content[i];
            tcl_append_float(interp,value);
        }
        return TCL_OK;
    }
  // the culling operation
  if (strcmp(argv[3],"unique")==0) {
        if (argc!=5) {
            #Error: "(ctree) The PAT unique sub-command requires a factor"
            return TCL_ERROR;
        }
        pat_unique(p,atof(argv[4]));
        return TCL_OK;
    }
    // Read the identification tag at certain index
    if (strcmp(argv[3],"id")==0) {
        if (argc!=5) {
            #Error: "(ctree) The PAT id sub-command requires an index"
            return TCL_ERROR;
        }
        tcl_append_int(interp,p->content->content[atoi(argv[4])]->id);
        return TCL_OK;
    }
    // Find the index where certain id is stored
    if (strcmp(argv[3],"id2index")==0) {
        if (argc!=5) {
```

```
        #Error: "(ctree) The PAT id sub-command requires an index"
        return TCL_ERROR;
    }
    int id=atoi(argv[4]);
    int i;
    for (i=0;i<p->content->num_of;i++)
        if (p->content->content[i]->id==id) break;
    if (i==p->content->num_of) tcl_append_int(interp,-1);
    else tcl_append_int(interp,i);
    return TCL_OK;
}
// Generate graph entries from PAT
if (strcmp(argv[3],"graph")==0) {
    if (argc!=7) {
        #Error: "(ctree) The PAT graph sub-command requires an output file, x
and y axes"
        return TCL_ERROR;
    }
    int i,x=-1,y=-1;
    for (i=0;i<p->properties->num_of;i++)
        if (strcmp(p->properties->content[i],argv[5])==0) x=i;
    for (i=0;i<p->properties->num_of;i++)
        if (strcmp(p->properties->content[i],argv[6])==0) y=i;
    if (x==-1) {
         #Error: "No such property %s in PAT %s" argv[5] c->name;
         return TCL_ERROR;
    }
    if (y==-1) {
        #Error: "No such property %s in PAT %s" argv[6] c->name;
        return TCL_ERROR;
    }
```

```
        FILE *O=fopen(argv[4],"w");
        fprintf(O,"%s,%s\n",argv[5],argv[6]);
        pat_graph(O,p,x,y);
        fclose(O);
        return TCL_OK;
    }
    // Delete entry according to index.
    if (strcmp(argv[3],"delete")==0) {
        if (argc!=5) {
            #Error: "(ctree) The PAT delete sub-command requires an index"
            return TCL_ERROR;
        }
        ordinal j;
        j=atoi(argv[4]);
        delete_entry_vector_pointer_PAT_entry(p->content,j);
        return TCL_OK;
    }
    // Stars empty the PAT from all entries,
    // except for the ones holding a global record on at least one property.
    if (strcmp(argv[3],"stars")==0) {
        if (argc!=4) {
            #Error: "(ctree) The PAT stars sub-command requires no more arguments"
            return TCL_ERROR;
        }
        pat_stars(p);
        return TCL_OK;
    }
    // Unrecognized sub-command error
    #Error: "(ctree) Unrecognized PAT sub-command %s. It requires a sub-command:
size, index, delete, foreach" argv[3]
    return TCL_ERROR;
```

```
        }
        // Apply a spec and calculate specific-front.
        if (strcmp(argv[2],">>>")==0) {
            if (c->value_type!=ctype_PAT) {
                #Error: "(ctree) The >>> operator is to be used with a pareto-associative
table only. Use double parentheses to declare one: @ PAT((size1,size2|prop1,prop2)) !"
                return TCL_ERROR;
            }
            if (argc!=4) {
                #Error: "(ctree) The >>> operator requires a list of properties."
                return TCL_ERROR;
            }
            ordinal i;
            PAT *p=(PAT *)c->value.v;
            // instead of applying spec, clear all the flags from all >>> operations
            if (strcmp(argv[3],"reset")==0) {
                for (i=0;i<p->content->num_of;i++) p->content->content[i]->flags=0;
                return TCL_OK;
            }
            // instead of applying spec, clear all the flags from previous >>>
            if (strcmp(argv[3],"undo")==0) {
                for (i=0;i<p->content->num_of;i++) p->content->content[i]->flags>>=1;
                return TCL_OK;
            }
            int ARGC;
            char **ARGV;
            Tcl_SplitList(interp,argv[3],&ARGC,&ARGV);
            vector_float *properties=new_vector_float();
            for (i=0;i<ARGC;i++) add_entry_vector_float(properties,atof(ARGV[i]));
            free(ARGV);
            for (i=0;i<p->content->num_of;i++) p->content->content[i]->flags<<=1;
```

```
            pat_front(p,properties);
            for (i=0;i<p->content->num_of;i++)
                if (!(p->content->content[i]->flags)) tcl_append_int(interp,i);
            return TCL_OK;
        }
        // insert a new entry to the PAT. Retention depends on Pareto dominance rules.
        if (strcmp(argv[2],"<<<")==0) {
            if (c->value_type!=ctype_PAT) {
                #Error: "(ctree) The <<< operator is to be used with a pareto-associative
table only. Use double parentheses to declare one: @ PAT((size1,size2|prop1,prop2)) !"
                return TCL_ERROR;
            }
            if (argc!=5) {
                #Error: "(ctree) The <<< operator requires a list of sizes and a list of
properties."
                return TCL_ERROR;
            }
            int ARGC;
            char **ARGV;
            Tcl_SplitList(interp,argv[3],&ARGC,&ARGV);
            vector_float *sizes=new_vector_float();
            int i;
            for (i=0;i<ARGC;i++) add_entry_vector_float(sizes,atof(ARGV[i]));
            free(ARGV);
            Tcl_SplitList(interp,argv[4],&ARGC,&ARGV);
            vector_float *properties=new_vector_float();
            for (i=0;i<ARGC;i++) add_entry_vector_float(properties,atof(ARGV[i]));
            free(ARGV);
            tcl_append_int(interp,add_pat_entry((PAT *)c->value.v,sizes,properties));
            return TCL_OK;
        }
```

```
// Assignment operator
if (strcmp(argv[2],"=")==0) {
    #Dinfo: "%s gets assignemnt" argv[1]
    if (argc==4) {
        if (c->value_type==ctype_POLY) {
            POLY *p=new_POLY();
            p->expression=strdup(argv[3]);
            link_POLY(p);
            c->value.v=p;
            return TCL_OK;
        }
        if (c->value_type==ctype_LUT) {
            if (array_entry==NULL) {
                #Error: "(ctree) invalid array access %s" argv[1]
                return TCL_ERROR;
            }
            *array_entry=atof(argv[3]);
            return TCL_OK;
        }
        if (strcmp(c->name,"POLY")==0) {
            POLY *p=new_POLY();
            p->expression=strdup(argv[3]);
            link_POLY(p);
            c->value.v=p;
            c->value_type=ctype_POLY;
            return TCL_OK;
        }
        c->value.s=atof(argv[3]);
        #Dinfo: "ASSIGNMENT %x=%s %g" &(c->value.s) argv[3] c->value.s
        c->value_type=ctype_real;
```

```
            return TCL_OK;
        }
        if (argc<5) {
            #Error: "(ctree) usage: @ <context> = [<type>] <value>"
            return TCL_ERROR;
        }
        // Setting type explicitly
        if (strcmp(argv[3],"real")==0) {
            c->value.s=atof(argv[4]);
            c->value_type=ctype_real;
            return TCL_OK;
        }
        if (strcmp(argv[3],"integer")==0) {
            c->value.o=atol(argv[4]);
            c->value_type=ctype_integer;
            return TCL_OK;
        }
        if (strcmp(argv[3],"string")==0) {
            c->value.v=strdup(argv[4]);
            c->value_type=ctype_string;
            return TCL_OK;
        }
        return(copy_ctree_structure(interp,argv[1],argv[3],argv));
    }
    // Array support commands
    if (strcmp(argv[2],"is_array")==0) {
        if (argc!=3) {
            #Error: "(ctree) usage: @ <context> is_array"
            return TCL_ERROR;
        }
```

```
        if (c->value_type==ctype_LUT) {
            tcl_append_int(interp,1);
        } else {
            tcl_append_int(interp,0);
        }
        return TCL_OK;
    }
    if (strcmp(argv[2],"delete")==0) {
        if (argc!=3) {
            #Error: "(ctree) usage: @ <context> delete"
            return TCL_ERROR;
        }
        context *d=c->parent;
        int i=0,j=0;
        for (i=0;i<d->num_of_children;i++) {
            if (d->children[i]==c) j++;
            if (j>=d->num_of_children) break;
            d->children[i]=d->children[j];
            j++;
        }
        d->num_of_children--;
        delete_context(c);
        Context=Ctree;
        return TCL_OK;
    }
    // Context navigation
    if (strcmp(argv[2],"cd")==0) {
        if (argc!=3) {
            #Error: "(ctree) usage: @ <context> cd"
            return TCL_ERROR;
```

```
        }
        Context=c;
        return TCL_OK;
    }
    if (strcmp(argv[2],"list")==0) {
        if (argc!=3) {
            #Error: "(ctree) usage: @ <context> list"
            return TCL_ERROR;
        }
        int i;
        for (i=0;i<c->num_of_children;i++) Tcl_AppendElement(interp,c->children[i]-
>name);
        return TCL_OK;
    }
    if (strcmp(argv[2],"foreach_child")==0) {
        if (argc!=5) {
            #Error: "(ctree) usage: @ <context> foreach_child <iterator> <code>"
            return TCL_ERROR;
        }
        int i;
        char buf[1024*1024];
        for (i=0;i<c->num_of_children;i++)  {
            sprintf(buf,"set %s %s",argv[3],c->children[i]->name);
            Tcl_Eval(interp,buf);
            Tcl_Eval(interp,argv[4]);
        }
        return TCL_OK;
    }
    if (strcmp(argv[2],"path")==0) {
        if (argc!=3) {
            #Error: "(ctree) usage: @ . path"
```

```
        return TCL_ERROR;
    }
    context_print_path(interp,c);
    return TCL_OK;
}
// Disk interface
if (strcmp(argv[2],"save")==0) {
    if (argc!=4) {
        #Error: "(ctree) usage: @ <array context> save <filename>"
        return TCL_ERROR;
    }
    FILE *O=fopen(argv[3],"w");
    context_save(c,O);
    fclose(O);
    return TCL_OK;
}
if (strcmp(argv[2],"load")==0) {
    if (argc!=4) {
        #Error: "(ctree) usage: @ <array context> load <filename>"
        return TCL_ERROR;
    }
    open_to_read(argv[3]);
    context_load(c,0);
    done_reading();
    return TCL_OK;
}
// calc=lookup
if (strcmp(argv[2],"calc")==0) {
    if (c->value_type!=ctype_LUT) {
        #Error: "(ctree) @ calc requires an array context"
```

```
                return TCL_ERROR;
            }
            LUT *a=(LUT *)c->value.v;
            if (argc-3!=a->dim) {
                #Error: "(ctree) Array %s has %d dimentions. cannot interpolate with %d coordinates." a->name a->dim argc-3
                return TCL_ERROR;
            }
            int i;
            for (i=0;i<a->dim;i++) global_coord[i]=atof(argv[i+3]);
            tcl_append_float(interp,a->interpolate(a,global_coord));
            return TCL_OK;
        }
        #Error: "(ctree) unsupported command %s" argv[2]
        return TCL_ERROR;
    }
```

The top-level tcl_ctree uses a low-level context creation/navigation command: resolve_context

```
    int resolve_context(char *i_key,context **i_context,float **array_entry) {
        context *temp_context=Context;
        if ((i_key[0]=='/')||(i_key[0]==':')) temp_context=Ctree;
        char context_name_buffer[1024];
        int j=0,i=0;
        #Dinfo: "Started Resolving Context %s from %x" i_key temp_context
        while (i_key[i]) {
            while (((i_key[i]=='/')||(i_key[i]==':'))&&(i_key[i])) i++;
            if (i_key[i]==0) break;
            while ((i_key[i]!='/')&&(i_key[i]!=':')&&(i_key[i]))
```

```
        context_name_buffer[j++]=i_key[i++];
    context_name_buffer[j]=0;
    #Dinfo: "Resolving SubContext %s from %x" context_name_buffer temp_context
    j=0;
    // Support for relative context
    if (strcmp(context_name_buffer,"..")==0) {
        if (temp_context->parent==NULL)  {
            #Derror: "(resolve_context) No such context: %s, failed at %s" i_key
context_name_buffer
            return 0;
        }
        temp_context=temp_context->parent;
        continue;
    }
    if (strcmp(context_name_buffer,".")==0) continue;
    int k=0;
    // Array notation supported here
    while ((context_name_buffer[k])&&(context_name_buffer[k]!='(')) k++;
    if (context_name_buffer[k]=='(') {
        context_name_buffer[k]=0;
        int l;
        context *next_context=NULL;
        for (l=0;l<temp_context->num_of_children;l++) {
            if (strcmp(context_name_buffer,temp_context->children[l]->name)==0) {
                next_context=temp_context->children[l];
                break;
            }
        }
        if (!next_context) {
            #Derror: "(resolve_context) No such context: %s, failed at %s" i_key
context_name_buffer
```

```
        return 0;
    }
    temp_context=next_context;
    if (temp_context->value_type!=ctype_LUT) {
        #Error: "(resolve_context) No such array: %s" temp_context->name
        return 0;
    }
    LUT *a=(LUT *)temp_context->value.v;
    if (a==NULL) {
        #Error: "(resolve_context) No such array: %s" temp_context->name
        return 0;
    }
    char *argv[256];
    int argc=1;
    argv[0]=&(context_name_buffer[k+1]);
    for (l=k+1;context_name_buffer[l]!=')';l++)
         if (context_name_buffer[l]==',')
              argv[argc++]=&(context_name_buffer[l+1]);
    for (l=k+1;context_name_buffer[l]!=')';l++)
        if (context_name_buffer[l]==',') context_name_buffer[l]=0;
    context_name_buffer[l]=0;
    if (argc!=a->dim) {
        #Error: "(resolve_context) Array: %s has %d dimension, but accessed
with %d indices" temp_context->name a->dim argc
        return 0;
    }
    ordinal offset=1;
    ordinal index=0;
    for (l=0;l<a->dim;l++) {
        index+=atoi(argv[l])*offset;
        offset*=a->size[l];
```

```
                }
                *array_entry=&(a->content[index]);
                continue;
            }
            context *next_context=NULL;
            for (k=0;k<temp_context->num_of_children;k++) {
                if (strcmp(context_name_buffer,temp_context->children[k]->name)==0) {
                    next_context=temp_context->children[k];
                    break;
                }
            }
            if (!next_context) {
                #Derror: "(resolve_context) No such context: %s, failed at %s!" i_key
context_name_buffer
                return 0;
            }
            temp_context=next_context;
        }
        #Dinfo: "Resolved context %s -> %x (%x,%g)" i_key temp_context &(temp_context-
>value.s) temp_context->value.s
        *i_context=temp_context;
        return 1;
    }
```

B. 3. Lookup and Interpolation

The LUT is an array with some header information:

```
typedef struct {

    char *name; // Identifying string, no longer necessary

    ordinal dim;    // the number of dimensions or the length of size[]

    ordinal size[$::MAXDIM]; // an array of integers, the range of indices per dimension.

    float *legend[$::MAXDIM]; // an array of arrays of the real values of the coordinates

    float physical_factor[$::MAXDIM]; // factors to convert values to indices

    float hypercube_volume;  // In case the LUT is uniformly sampled, the single volume
saves a lot of calculation time.

    ordinal sizer[$::MAXDIM]; // pre-calculation of the value in index-offset per
dimension

    float *content; // The array payload. This is where everything is stored

    ordinal neighbors[1<<$::MAXDIM]; // Array of index-offset per hypercube corner

    linear_interpolation_table *LIT; // Optional LIT

    hit_node *hit;                  // Optional HIT

    unsigned char *hit_bytecode;     // HIT interpreter language buffer
```

```
    ordinal hit_bytecode_size;

    float (*interpolate)(void *a,float *coord); // interpolation functions

    float (*gamma_interpolate)(void *a);

    float (*gamma_gradient)(void *a);

} LUT;
```

The optional LIT has the following structure:

```
struct linear_interpolation_table{
    ordinal dim;    // the number of dimensions or the length of size[]
    ordinal *size; // an array of integers, the number of bits per dimension.
    ordinal *index_weight; // an array of integers, the index-weight of each dimension.
    ordinal volume;
    float *baseline;
    float *inv_step;
    float *content; // The array payload
} ;
```

The optional HIT is a connected tree. Each node has the following structure:

```
typedef union {
   float s;
   ordinal o;
   void *p;
} hit_node;
```

B. 3. 1. Interpolation of LUT

The simple lookup and interpolation function supports the following options:

1 Uniform vs. non or partially-uniform sampling

LUT alone vs. LUT+LIT

When the hypercube_volume field is 0, the interpolation function treats the array as non-uniform and uses the legend arrays in a binary-search to find the looked-up entry. If hypercube_volume is not 0, entry location is simpler and done by linear scaling of the coordinates directly into integer indices.

Notice that the function definition is nested in a Tcl-preprocessor loop. The listing below results in 100's of ANSI C lines of code, because each dimension from 1 to MAXDIM (currently: 8) gets its own function. The motivation for this coding style is the compilation-time interpolation loop that performs all arithmetic operations in the buffer, without branching or conditioning in real time.

```
#For: {set DIM 1} {$DIM<$::MAXDIM} {incr DIM} {
    float lut_interpolation_$DIM(void *i_a,float *coord) {
        #tcl set num_of_corners [expr 1<<$DIM]
        LUT *a=(LUT *)i_a;
        Tcl_Time start_time,end_time;
        Tcl_GetTime(&start_time);
        linear_interpolation_table *L=a->LIT;
        ordinal i,j,end;
        // Find the hyper-cube
```

```
float retval=0;
ordinal index=0;
ordinal lit_index=0;
ordinal sizer=1;
ordinal lit_sizer=1;
int I;
float hcv=a->hypercube_volume;
if (hcv==0) {
    hcv=1;
    // Non-uniform sampling, do a binary search
    #For: {set i 0} {$i<$DIM} {incr i} {
        end=a->size[$i]-1;
        // Separate ascending from descending legends
        if (a->legend[$i][end]>a->legend[$i][0]) {
            // Ascending legend
            if (coord[$i]<=a->legend[$i][0]) {
                key[$i]=0;
                #Dwarning: "Undershoot %g<%g" coord[$i] a->legend[$i][0]
            } else if (coord[$i]>=a->legend[$i][end]) {
                key[$i]=end-1;
                #Dwarning: "Overshoot %g>%g" coord[$i] a->legend[$i][end]
            } else {
                ordinal pre=0;
                ordinal post=end;
                while (post-pre>1) {
                    ordinal mid=(post+pre)/2;
                    if (coord[$i]>=a->legend[$i][mid]) {
                        pre=mid;
                    } else {
                        post=mid;
```

```
                }
            }
            key[$i]=pre;
        }
    } else {
        // Descending legend
        if (coord[$i]>=a->legend[$i][0]) {
            key[$i]=0;
            #Dwarning: "Undershoot %g>%g" coord[$i] a->legend[$i][0]
        } else if (coord[$i]<=a->legend[$i][end]) {
            key[$i]=end-1;
            #Dwarning: "Overshoot %g<%g" coord[$i] a->legend[$i][end]
        } else {
            ordinal pre=0;
            ordinal post=end;
            while (post-pre>1) {
                ordinal mid=(post+pre)/2;
                if (coord[$i]<=a->legend[$i][mid]) {
                    pre=mid;
                } else {
                    post=mid;
                }
            }
            key[$i]=pre;
        }
    }
    hcv*=(a->legend[$i][key[$i]+1]-a->legend[$i][key[$i]]);
    index+=key[$i]*sizer;
    lit_index+=key[$i]*lit_sizer;
    sizer*=a->size[$i];
```

```
            lit_sizer*=(a->size[$i]-1);
        }
    } else {
        // Uniform sampling
        float i_f;
        #For: {set i 0} {$i<$DIM} {incr i} {
            i_f=(coord[$i]-a->legend[$i][0])*a->physical_factor[$i];
            int key${i}=(int)i_f;
            if (key${i}<0) key${i}=0;
            if (key${i}>=a->size[$i]-1) key${i}=a->size[$i]-2;
            index+=key${i}*a->sizer[$i];
            lit_index+=key${i}*lit_sizer;
            sizer*=a->size[$i];
            lit_sizer*=(a->size[$i]-1);
        }
    }
    if (L) {
        // LIT exists! Try the faster, simpler linear interpolation
        float *slopes=&(L->content[lit_index*($DIM+1)]);
        float intercept=slopes[$DIM];
        if (intercept!=0) {
            #Dinfo: "Linear entry exists"
            retval=intercept;
            #For: {set i 0} {$i<$DIM} {incr i} {
                retval+=slopes[$i]*coord[$i];
            }
            goto interpolation_time_$DIM;
        }
    }
    #Dinfo: "Full  "
```

```
            // Full interpolation, refer back to the original array
            float *hypercube=&(a->content[index]);
            #For: {set corner 0} {$corner<$num_of_corners} {incr corner} {
                double interpolation_buffer$corner=hypercube[a->neighbors[$corner]];
                #Dinfo: "Corner $corner=%g" interpolation_buffer$corner
            }
            // Lagrange interpolation in Tcl-preprocessor code
            float w1,w2;
            #tcl set weighing_dim 0
            #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr
$breadth/2]} {
                #Dinfo: "Dim $weighing_dim: key=%d %x" key${weighing_dim} coord
                w1=coord[$weighing_dim]-a->legend[$weighing_dim][key${weighing_dim}];
                w2=a->legend[$weighing_dim][key${weighing_dim}+1]-coord[$weighing_dim];
                #tcl set j 0
                #Dinfo: "Dim $weighing_dim: key=%d %g (%g,%g) (%g,%g)" key${weighing_dim}
coord[$weighing_dim] a->legend[$weighing_dim][key${weighing_dim}] a-
>legend[$weighing_dim][key${weighing_dim}+1] w1 w2
                #For: {set i 0} {$i<$breadth} {incr i 2} {
                    #tcl set k [expr $i+1]

interpolation_buffer$j=interpolation_buffer$k*w1+interpolation_buffer$i*w2;
                    #Dinfo: "interpolation_buffer$j=%g" interpolation_buffer$j
                    #tcl incr j
                }
                #tcl incr weighing_dim
            }
            // The result is bloated with all the dimensions' sizes
            // normalized here with the hypercube volume
            retval=interpolation_buffer0/hcv;
            // The exit label is used to measure the average interpolation time
        interpolation_time_$DIM:
```

```
    Tcl_GetTime(&end_time);
    get_Tcl_timer+=end_time.sec*1e6+end_time.usec-\
                   start_time.sec*1e6-start_time.usec;
    get_Tcl_counter++;
    return(retval);
}}
```

B. 3. 2. Differential Interpolation

For algorithms that need the slope, rather than the value, of a looked-up entry, the following function makes a slight modification to the interpolation sequence, in order to achieve both value and derivative per dimension (gradient) in the looked-up coordinates. Notice that the Tcl-preprocessor manipulates the function's argument list as well as its dimensionality.

```
#For: {set DIM 1} {$DIM<$::MAXDIM} {incr DIM} {
    float gamma_gcc_interpolateg_$DIM(
    void *i_a
    #For: {set j 0} {$j<$DIM} {incr j} {
        ,float c$j
    }
    #For: {set j 0} {$j<$DIM} {incr j} {
        ,float *p$j
    }
    ) {
        #Dinfo: "Gamma machine's gradient function for ${DIM}D LUT is %x" i_a
        #For: {set i 0} {$i<$DIM} {incr i} {
            #Dinfo: "Input $i is %g" c$i
        }
        #For: {set i 0} {$i<$DIM} {incr i} {
            #Dinfo: "Derivative $i goes back to (%x)" p$i
        }
        #tcl set num_of_corners [expr 1<<$DIM]
        LUT *a=(LUT *)i_a;
        Tcl_Time start_time,end_time;
        Tcl_GetTime(&start_time);
        ordinal i,j,end;
        // Find the hyper-cube
```

```
            float retval=0;
            ordinal index=0;
            float i_f;
            #For: {set i 0} {$i<$DIM} {incr i} {
                i_f=(c$i-a->legend[$i][0])*a->physical_factor[$i];
                #Dinfo: "coord$i=%g base=%g factor=%g Key=%g" c$i a->legend[$i][0] a-
>physical_factor[$i] i_f
                int key${i}=(int)i_f;
                if (key${i}<0) key${i}=0;
                if (key${i}>=a->size[$i]-1) key${i}=a->size[$i]-2;
                index+=key${i}*a->sizer[$i];
                #Dinfo: "key$i=%d/%d %d index=%ld" key${i} a->size[$i]  a->sizer[$i] index
            }
            // Full interpolation, refer back to the original array
            float *hypercube=&(a->content[index]);
            #For: {set corner 0} {$corner<$num_of_corners} {incr corner} {
                double interpolation_buffer$corner=hypercube[a->neighbors[$corner]];
                #Dinfo: "Corner $corner=%g" interpolation_buffer$corner
            }
            float w1,w2;
            #tcl set weighing_dim 0
            #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr
$breadth/2]} {
                #Dinfo: "Dim $weighing_dim: key=%d" key${weighing_dim}
                w1=c$weighing_dim-a->legend[$weighing_dim][key${weighing_dim}];
                w2=a->legend[$weighing_dim][key${weighing_dim}+1]-c$weighing_dim;
                #tcl set j 0
                #Dinfo: "Dim $weighing_dim: key=%d %g (%g,%g) (%g,%g)" key${weighing_dim}
c$weighing_dim a->legend[$weighing_dim][key${weighing_dim}] a-
>legend[$weighing_dim][key${weighing_dim}+1] w1 w2
                #For: {set i 0} {$i<$breadth} {incr i 2} {
                    #tcl set k [expr $i+1]
```

```
interpolation_buffer$j=interpolation_buffer$k*w1+interpolation_buffer$i*w2;
                        float gradient_buffer${weighing_dim}_$j=interpolation_buffer$i-
interpolation_buffer$k;
                        #For: {set l 0} {$l<$weighing_dim} {incr l} {

gradient_buffer${l}_$j=gradient_buffer${l}_$k*w1+gradient_buffer${l}_$i*w2;
                            #Dinfo: "grad_buffer${l}_$j=%g" gradient_buffer${l}_$j
                        }
                        #Dinfo: "Interpolation Buffer $j=%g" interpolation_buffer$j
                        #tcl incr j
                    }
                    #tcl incr weighing_dim
                }
                #Dinfo: "Interpolation is DONE"
                #For: {set i 0} {$i<$DIM} {incr i} {
                    #Dinfo: "Sending derivative back to p$i (%x)" p$i
                    if (p$i) *p$i=gradient_buffer${i}_0/a->hypercube_volume;
                    #Dinfo: "Derivative %d: %g" $i *p$i
                }
                #Dinfo: "Result: %g" interpolation_buffer0/a->hypercube_volume
                interpolation_time_$DIM:
                Tcl_GetTime(&end_time);
                get_Tcl_timer+=end_time.sec*1e6+end_time.usec-start_time.sec*1e6-
start_time.usec;
                get_Tcl_counter++;
                #Dinfo: "DONE: Gamma machine's gradient function for ${DIM}D"
                return(interpolation_buffer0/a->hypercube_volume);
            }
        }
```

B. 3. 3. Cluster Interpolation

Some algorithms require repetitive lookups on the same transistor and parameter with a very small coordinate change. For instance, random search of steepest slope samples around a center point until it finds the best direction to continue the search to optimum. This means that the same interpolation buffer gets loaded again and again, because the small change in coordinates hardly ever leaves the original hypercube and even if it did, a small extrapolation does not justify reloading of a new buffer. Cluster interpolation simply allocates a shadow interpolation buffer that accepts a copy of the initial one. On each of the following calls to that function, the entry-locating routine and buffer loading are replaced by copying of the shadow buffer to the interpolation buffer.

The cluster is defined by the following structure:

```
typedef struct {
    float lower_margin[$::MAXDIM]; // left boundary of hypercube
    float upper_margin[$::MAXDIM]; // right boundary of hypercube
    float hyper_volume;  // the total volume of the hypercube
    float *interpolation_buffer; // The interpolation beffer (corner values)
} cluster;
```

And the function that uses it:

```
 #For: {set DIM 1} {$DIM<$::MAXDIM} {incr DIM} {
    float lut_cluster_interpolation_$DIM(LUT *a,float *coord,cluster **i_cluster) {
        #tcl set num_of_corners [expr 1<<$DIM]
        float retval=0;
        if (*i_cluster) {
            // This is not the first interpolation, use the cluster buffer
```

```
                    #For: {set corner 0} {$corner<$num_of_corners} {incr corner} {

                        double interpolation_buffer$corner=(*i_cluster)-
>interpolation_buffer[$corner];

                    }

                    #tcl set weighing_dim 0

                    #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr
$breadth/2]} {

                        #tcl set j 0

                        #For: {set i 0} {$i<$breadth} {incr i 2} {

                            #tcl set k [expr $i+1]


interpolation_buffer$j=interpolation_buffer$k*(coord[$weighing_dim]-(*i_cluster)-
>lower_margin[$weighing_dim])+interpolation_buffer$i*((*i_cluster)-
>upper_margin[$weighing_dim]-coord[$weighing_dim]);

                            #tcl incr j

                        }

                        #tcl incr weighing_dim

                    }

                    retval=interpolation_buffer0/(*i_cluster)->hyper_volume;

                    return(retval);

                }

                Tcl_Time start_time,end_time;

                Tcl_GetTime(&start_time);

                linear_interpolation_table *L=a->LIT;

                ordinal i,j,end;

                // Find the hyper-cube

                ordinal index=0;

                ordinal lit_index=0;

                ordinal sizer=1;

                ordinal lit_sizer=1;

                ordinal key[$DIM];

                *i_cluster=(cluster *)malloc(sizeof(cluster));

                (*i_cluster)->interpolation_buffer=\
```

```
        (float *)malloc(sizeof(float)*$num_of_corners);
int I;
#For: {set i 0} {$i<$DIM} {incr i} {
    end=a->size[$i]-1;
    if (a->legend[$i][end]>a->legend[$i][0]) {
        if (coord[$i]<=a->legend[$i][0]) {
            key[$i]=0;
            #Dwarning: "Undershoot %g<%g" coord[$i] a->legend[$i][0]
        } else if (coord[$i]>=a->legend[$i][end]) {
            key[$i]=end-1;
            #Dwarning: "Overshoot %g>%g" coord[$i] a->legend[$i][end]
        } else {
            ordinal pre=0;
            ordinal post=end;
            while (post-pre>1) {
                ordinal mid=(post+pre)/2;
                if (coord[$i]>=a->legend[$i][mid]) {
                    pre=mid;
                } else {
                    post=mid;
                }
            }
            key[$i]=pre;
        }
    } else {
        if (coord[$i]>=a->legend[$i][0]) {
            key[$i]=0;
            // #Warning: "Undershoot %g>%g" coord[$i] a->legend[$i][0]
        } else if (coord[$i]<=a->legend[$i][end]) {
            key[$i]=end-1;
            // #Warning: "Overshoot %g<%g" coord[$i] a->legend[$i][end]
```

```
        } else {
            ordinal pre=0;
            ordinal post=end;
            while (post-pre>1) {
                ordinal mid=(post+pre)/2;
                if (coord[$i]<=a->legend[$i][mid]) {
                    pre=mid;
                } else {
                    post=mid;
                }
            }
            key[$i]=pre;
        }
    }
    index+=key[$i]*sizer;
    lit_index+=key[$i]*lit_sizer;
    sizer*=a->size[$i];
    lit_sizer*=(a->size[$i]-1);
    (*i_cluster)->lower_margin[$i]=a->legend[$i][key[$i]];
    (*i_cluster)->upper_margin[$i]=a->legend[$i][key[$i]+1];
}
#Dinfo: "Full  "
// Full interpolation, refer back to the original array
#For: {set corner 0} {$corner<$num_of_corners} {incr corner} {
    double interpolation_buffer$corner=a->content[index+a->neighbors[$corner]];
    (*i_cluster)->interpolation_buffer[$corner]=interpolation_buffer$corner;
}
#tcl set weighing_dim 0
float weight;
float hyper_volume=1;
```

```
                #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr $breadth/2]}
{
                    #tcl set j 0
                    weight=(coord[$weighing_dim]-a->legend[$weighing_dim][key[$weighing_dim]]);
                    hyper_volume*=(a->legend[$weighing_dim][key[$weighing_dim]+1]-a-
>legend[$weighing_dim][key[$weighing_dim]]);
                    #For: {set i 0} {$i<$breadth} {incr i 2} {
                        #tcl set k [expr $i+1]
                        interpolation_buffer$j=(interpolation_buffer$k-
interpolation_buffer$i)*weight+interpolation_buffer$i;
                        #tcl incr j
                    }
                    #tcl incr weighing_dim
                }
                (*i_cluster)->hyper_volume=hyper_volume;
                retval=interpolation_buffer0/hyper_volume;
                interpolation_time_$DIM:
                Tcl_GetTime(&end_time);
                get_Tcl_timer+=end_time.sec*1e6+end_time.usec-start_time.sec*1e6-
start_time.usec;
                get_Tcl_counter++;
                return(retval);
            }
        }
```

B. 3. 4. Linear Regression

To populate a LIT from a LUT, a special function performs linear regression on each LUT cell and registers the slopes+intercept in a LIT cell. A regression in this case is very simple, because a hypercube has only two coordinate values per dimension. Therefore, the regression is simply adding up the corners' values to 2×DIM buckets, subtracting low coordinates from high coordinates and normalizing the slopes. The DIM+1 constants (slopes+intercept) are then used to recreate the original corner values. If one of the interpolated values is off by more than 1% from the original one, the cell is marked as invalid, so the interpolation function knows to use the full LUT interpolation instead. The percentage of LIT cells that are valid (LCP) is returned.

Allocating a new LIT is done by the following top-level function:

```
void new_linear_interpolation_table(LUT *a) {
    linear_interpolation_table *L;
    L=(linear_interpolation_table *)malloc(sizeof(linear_interpolation_table));
    a->LIT=L;
    L->dim=a->dim;
    L->size=(ordinal *)malloc(sizeof(ordinal)*L->dim);
    L->index_weight=(ordinal *)malloc(sizeof(ordinal)*L->dim);
    L->baseline=(float *)malloc(sizeof(scalar)*L->dim);
    L->inv_step=(float *)malloc(sizeof(scalar)*L->dim);
    ordinal i;
    L->volume=1;
    for (i=0;i<a->dim;i++) {
        L->size[i]=-1;
        ordinal tmp=a->size[i];
        while (tmp) {
```

```
        L->size[i]++;
        tmp/=2;
    }
    L->index_weight[i]=L->volume;
    L->volume*=(a->size[i]-1);
    L->baseline[i]=a->legend[i][0];
    L->inv_step[i]=1/(a->legend[i][1]-a->legend[i][0]);
}
ordinal num_of_corners=1<<a->dim;
ordinal offset;
for (offset=0;offset<num_of_corners;offset++) {
    ordinal sizer=1;
    ordinal index=0;
    // scratch 'tmp' to roll the offset bits without destroying offset itself:
    ordinal tmp_offset=offset;
    for (i=0;i<a->dim;i++) {
        index+=sizer*(tmp_offset&1);
        sizer*=a->size[i];
        // next bit
        tmp_offset>>=1;
    }
    a->neighbors[offset]=index;
}
L->content=(float *)malloc(sizeof(scalar)*(1+a->dim)*L->volume);
ordinal pass=0;
float error_rms=0;
float progress_index=1;
#Dinfo: "LIT allocated (%ld entries). Calculating slopes" L->volume
for (i=0;i<L->volume;i++) {
    if ((10.0*i/L->volume)>progress_index) {
```

```
                    #Dinfo: "Linear Cells' Precentage=%d%%   %d%% left   potential=%d%%"
100*pass/i 100-100*i/L->volume 100*pass/L->volume+100-100*i/L->volume

                    progress_index+=1;

                }

                float error=generate_lit(a,L,i);

                if(error<1.0) pass++;

                error_rms+=error*error;

            }

            error_rms/=L->volume;

            error_rms=sqrt(error_rms);

            #Dinfo: "%ld fit linear out of %ld entries (LCP: %d%%  RMS error=%e)" pass L-
>volume 100*pass/L->volume error_rms

        }

        The regression per cell is done in this low-level function:

        float generate_lit(LUT *a,linear_interpolation_table *L,ordinal i_index) {

            int i;

            // Find the hyper-cube

            ordinal key[$::MAXDIM];

            float retval=0;

            ordinal index=i_index*(a->dim+1);

            float *slopes=&(L->content[index]);

            ordinal tmp_index=i_index;

            for (i=a->dim-1;i>=0;i--) {

                key[i]=tmp_index/L->index_weight[i];

                #Dinfo: "key(%d)=%d" i key[i]

                tmp_index%=L->index_weight[i];

            }

            // First, check lower-left-closest corner for possible pre-calculated slopes

            ordinal sizer=1;

            ordinal array_index=0;

            for (i=0;i<a->dim;i++) {

                array_index+=sizer*(key[i]);
```

```
    sizer*=a->size[i];
}
float constant=0;
float midpoint[$::MAXDIM];
ordinal l=1;
for (i=0;i<a->dim;i++) {
    slopes[i]=0;
    l*=2;
    midpoint[i]=(a->legend[i][key[i]]+a->legend[i][key[i]+1])/2;
}
l/=2;
ordinal corner=0;
ordinal num_of_corners;
num_of_corners=1<<a->dim;
for (corner=0;corner<num_of_corners;corner++) {
    ordinal corner_index=array_index+a->neighbors[corner];
    float corner_value=a->content[corner_index];
    // constant is pre-loaded with the average between all points
    constant+=corner_value;
    ordinal tmp_corner=corner;
    for (i=0;i<a->dim;i++) {
        if (tmp_corner&1) {
            slopes[i]+=corner_value;
        } else {
            slopes[i]-=corner_value;
        }
        // next bit
        tmp_corner>>=1;
    }
}
// Normalizing constant to the average
```

```
constant/=l;
constant/=2;
for (i=0;i<a->dim;i++) {
    // Normalizing each slope
    slopes[i]/=l;
    slopes[i]=slopes[i]/(a->legend[i][key[i]+1]-a->legend[i][key[i]]);
    #Dinfo: "slope(%d)=%g" i slopes[i]
    // subtracting the regression from the average
    constant-=slopes[i]*midpoint[i];
}
slopes[a->dim]=constant;
#Dinfo: "intercept=%g" constant
////////////////////////////////////// Testing:
float max_error=0;
for (corner=0;corner<num_of_corners;corner++) {
    ordinal index=0;
    // scratch 'tmp' to roll the corner bits without destroying corner itself:
    ordinal tmp_corner=corner;
    float interpolated_value=slopes[a->dim];
    tmp_corner=corner;
    for (i=0;i<a->dim;i++) {
        interpolated_value+=slopes[i]*a->legend[i][key[i]+(tmp_corner&1)];
        tmp_corner>>=1;
    }
    float original_value=a->content[array_index+a->neighbors[corner]];
    float error=100*fabs((original_value-interpolated_value)/original_value);
    if (fabs((original_value-interpolated_value))<1e-12) error=0.1;
    if (error>max_error) max_error=error;
}
#Dinfo: "max_error=%g" max_error
if (max_error<1.0) return max_error;
```

```
        // Staying with full interpolation
        so_union sob;
        sob.o=$::MAXDIM;
        sob.o&=NAN_UNMASK;
        sob.o|=NAN_VALUE;
        slopes[a->dim]=sob.s;
        slopes[a->dim]=0;
        return 100.0;
    }
```

B. 3. 5. Composite Interpolation

An exception to the above interpolation functions, which are generalized and contain no reference to the modeled electrical parameter, the following interpolation function is designed especially for a transistor's g_m, g_o, and I_{DS}. The three are looked-up in a single function that shares entry-locating between the three and uses g_m and g_o as additional slope information to achieve more accurate I_{DS}.

```
    #For: {set DIM 4} {$DIM<5} {incr DIM} {
        void composite_gamma_gcc_interpolate_$DIM(void *i_a_ids,void *i_a_gm,void
*i_a_ro,float *gm, float *go, float *Ids
        #For: {set j 0} {$j<$DIM} {incr j} {
            ,float c$j
        }
        , float c4
        ) {
            #Dinfo: "Gamma machine's interpolation function for ${DIM}D %x %x %x" i_a_ids
i_a_gm i_a_ro
            #tcl set num_of_corners [expr 1<<$DIM]
            LUT *a=(LUT *)i_a_gm;
            #Dinfo: "DIM=%d" a->dim
```

```
Tcl_Time start_time,end_time;

Tcl_GetTime(&start_time);

ordinal i,j,end;

float L=c3;

float W=c4;

float Gamma=W/L;

#Dinfo: "c0=%g c1=%g c2=%g c3=%g W=%g L=%g Gamma=%g" c0 c1 c2 c3 W L Gamma

// Find the hyper-cube

float retval=0;

ordinal index=0;

float i_f;

#For: {set i 0} {$i<$DIM} {incr i} {

    int key${i};

    i_f=(c$i-a->legend[$i][0])*a->physical_factor[$i];

    if (a->physical_factor[$i]>0) {

        key${i}=(int)i_f;

    } else {

        key${i}=-((int)(-i_f));

    }

    #Dinfo: "coord$i=%g base=%g factor=%g Key=%g" c$i a->legend[$i][0] a-
>physical_factor[$i] i_f

    if (key${i}<0) key${i}=0;

    if (key${i}>=a->size[$i]-1) key${i}=a->size[$i]-2;

    index+=key${i}*a->sizer[$i];

    #Dinfo: "key$i=%d/%d %d index=%ld" key${i} a->size[$i]  a->sizer[$i] index

}

// Full interpolation, refer back to the original array

float *gm_hypercube=&(a->content[index]);

#For: {set corner 0} {$corner<$num_of_corners} {incr corner} {

    float interpolation_buffer$corner=gm_hypercube[a->neighbors[$corner]];

    #Dinfo: "Gm Corner $corner=%g" interpolation_buffer$corner
```

```
                }
                float w1,w2;
                #tcl set weighing_dim 0
                #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr $breadth/2]}
{
                    #Dinfo: "Dim $weighing_dim: key=%d" key${weighing_dim}
                    w1=c$weighing_dim-a->legend[$weighing_dim][key${weighing_dim}];
                    w2=a->legend[$weighing_dim][key${weighing_dim}+1]-c$weighing_dim;
                    #tcl set j 0
                    #Dinfo: "Dim $weighing_dim: key=%d %g (%g,%g) (%g,%g)" key${weighing_dim}
c$weighing_dim a->legend[$weighing_dim][key${weighing_dim}] a-
>legend[$weighing_dim][key${weighing_dim}+1] w1 w2
                    #For: {set i 0} {$i<$breadth} {incr i 2} {
                        #tcl set k [expr $i+1]

interpolation_buffer$j=interpolation_buffer$k*w1+interpolation_buffer$i*w2;
                        #Dinfo: "interpolation_buffer$j=%g" interpolation_buffer$j
                        #tcl incr j
                    }
                    #tcl incr weighing_dim
                }
                #Dinfo: "gm=%g/%g=%g (*%g/%g=%g)" interpolation_buffer0 a->hypercube_volume
interpolation_buffer0/a->hypercube_volume W L interpolation_buffer0/a->hypercube_volume*Gamma
                *gm=interpolation_buffer0/a->hypercube_volume;
                a=i_a_ro;
                #Dinfo: "DIM=%d" a->dim
                float *ro_hypercube=&(a->content[index]);
                #For: {set corner 0} {$corner<$num_of_corners} {incr corner} {
                    interpolation_buffer$corner=ro_hypercube[a->neighbors[$corner]];
                    #Dinfo: "Ro Corner $corner=%g" interpolation_buffer$corner
                }
                #tcl set weighing_dim 0
                #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr $breadth/2]}
{
```

```
                    #Dinfo: "Dim $weighing_dim: key=%d" key${weighing_dim}

                    w1=c$weighing_dim-a->legend[$weighing_dim][key${weighing_dim}];

                    w2=a->legend[$weighing_dim][key${weighing_dim}+1]-c$weighing_dim;

                    #tcl set j 0

                    #Dinfo: "Dim $weighing_dim: key=%d %g (%g,%g) (%g,%g)" key${weighing_dim}
c$weighing_dim a->legend[$weighing_dim][key${weighing_dim}] a-
>legend[$weighing_dim][key${weighing_dim}+1] w1 w2

                    #For: {set i 0} {$i<$breadth} {incr i 2} {

                        #tcl set k [expr $i+1]


interpolation_buffer$j=interpolation_buffer$k*w1+interpolation_buffer$i*w2;

                        #Dinfo: "interpolation_buffer$j=%g" interpolation_buffer$j

                        #tcl incr j

                    }

                    #tcl incr weighing_dim

                }

                #Dinfo: "go=%g/%g=%g (*%g/%g=%g)" interpolation_buffer0 interpolation_buffer0
a->hypercube_volume 1/(interpolation_buffer0/a->hypercube_volume) W L
1/(interpolation_buffer0/a->hypercube_volume)*Gamma

                *go=1/(interpolation_buffer0/a->hypercube_volume);

                // This is where the composite interpolation takes place

                // 1. The Ids values from the LUT get Vgs*gm and Vds*go subtracted before
insertion to interpolation buffer

                a=i_a_ids;

                #Dinfo: "DIM=%d" a->dim

                float *hypercube=&(a->content[index]);

                float cornerVgs,cornerVds,corner_ids;

                #For: {set corner 0} {$corner<$num_of_corners} {incr corner} {

                    #tcl set VgsIndex [expr $corner%2]

                    #tcl set VdsIndex [expr $corner%4/2]

                    cornerVgs=a->legend[0][key0+$VgsIndex];

                    cornerVds=a->legend[1][key1+$VdsIndex];

                    corner_ids=hypercube[a->neighbors[$corner]];

                    #Dinfo: "Ids($corner)=%g" corner_ids
```

```
                    interpolation_buffer$corner=hypercube[a->neighbors[$corner]]-
cornerVgs*gm_hypercube[a->neighbors[$corner]]-cornerVds/ro_hypercube[a->neighbors[$corner]];

                    #Dinfo: "Ideq Corner Vgs=%g (real %g) Vds=%g (real %g) $corner=%g"
cornerVgs c0 cornerVds c1 interpolation_buffer$corner

                }

                #tcl set weighing_dim 0

                #For: {set breadth $num_of_corners} {$breadth>1} {set breadth [expr $breadth/2]}
{

                    #Dinfo: "Dim $weighing_dim: key=%d" key${weighing_dim}

                    w1=c$weighing_dim-a->legend[$weighing_dim][key${weighing_dim}];

                    w2=a->legend[$weighing_dim][key${weighing_dim}+1]-c$weighing_dim;

                    #tcl set j 0

                    #Dinfo: "Dim $weighing_dim: key=%d %g (%g,%g) (%g,%g)" key${weighing_dim}
c$weighing_dim a->legend[$weighing_dim][key${weighing_dim}] a-
>legend[$weighing_dim][key${weighing_dim}+1] w1 w2

                    #For: {set i 0} {$i<$breadth} {incr i 2} {

                        #tcl set k [expr $i+1]


interpolation_buffer$j=interpolation_buffer$k*w1+interpolation_buffer$i*w2;

                        #Dinfo: "interpolation_buffer$j=%g" interpolation_buffer$j

                        #tcl incr j

                    }

                    #tcl incr weighing_dim

                }

                // 2. The final value is added with the Vgs*gm and Vds*go values calculated
from previous interpolations

                *Ids=(interpolation_buffer0/a->hypercube_volume+(*gm)*c0+(*go)*c1);

                #Dinfo: "Ids=%g/%g=%g (%g)" interpolation_buffer0 a->hypercube_volume
interpolation_buffer0/a->hypercube_volume+(*gm)*c0+(*go)*c1 (interpolation_buffer0/a-
>hypercube_volume+(*gm)*c0+(*go)*c1)*Gamma

                *Ids*=Gamma;

                *gm*=Gamma;

                *go*=Gamma;

                Tcl_GetTime(&end_time);

                get_Tcl_timer+=end_time.sec*1e6+end_time.usec\
```

```
                    -start_time.sec*1e6-start_time.usec;
        get_Tcl_counter++;
    }
}
```

B. 4. Transistor-Level Sizer

Before Γ, an optimization application of the LUT models was created for a single transistor sizing, the "Sizer". The code is reliant on the Tcl-preprocessor to track a list of properties, size parameters and operating point. The sizer uses a feasibility evaluating function, a search function and a Tcl-implemented web server for generating the website.

"feasibility" is the distance to spec.

```
float feasibility(
#Foreach: input $::sizer_inputs {
    float $input,
}
#Foreach: parameter $::sizer_parameters {
    float *$parameter,
    LUT *${parameter}_LUT,
}
float *Gain, float *Area, float *fc, float tolerance
) {
    float point_Area=W*L*1e12;
    #tcl set i 0
    #Foreach: input $::sizer_inputs {
        global_coord[$i]=${input};
        #Dinfo: "$i) $input=%g" $input
        #tcl incr i
    }
    #Foreach: parameter $::sizer_parameters {
        float point_${parameter}=${parameter}_LUT->interpolate(${parameter}_LUT,global_coord);
    }
```

```
        #Foreach: parameter {Ids gm go} {
            point_${parameter}*=W/L;
        }
        point_Nth*=sqrt(point_gm);
        point_Nflicker*=point_gm/sqrt(W*L);
        point_sigmaVt*=(100/point_Vt);
        float point_Gain=20*log_ten(point_gm/point_go);
        float point_fc=(point_Nflicker*point_Nflicker)/(point_Nth*point_Nth);
        float distance=0;
        // Ids gets a special "tightening"
        if (!isnan(*Ids)) {
            float point_distance=(point_Ids/(*Ids))-1.0;
            #Dinfo: "Distance from Ids is %g/%g=%g tolerance=%g" point_Ids (*Ids)
point_distance tolerance
            point_distance*=point_distance;
            if (point_distance<=tolerance*tolerance) point_distance=0;
            distance+=point_distance;
        }
        #Foreach: parameter [concat $::sizer_parameters_no_ids Gain Area fc] {
            if (!isnan(*$parameter)) {
                float point_distance=(point_$parameter/(*$parameter))-1.0;
                #Dinfo: "Distance from $parameter is %g/%g=%g tolerance=%g"
point_$parameter (*$parameter) point_distance tolerance
                point_distance*=point_distance;
                if (point_distance<=tolerance*tolerance) point_distance=0;
                distance+=point_distance;
            }
        }
        return distance;
    }
```

The search function is "sizer". It finds the steepest slope for minimizing the feasibility metric and follows it until the feasibility begins to rise, doubling its stride every step. When the descent stops, sizer searches for a new direction. The search stops when the distance is zero.

```
float sizer(
#Foreach: input $::sizer_inputs {
    float *$input, float min_$input, float max_$input,
}
#Foreach: parameter $::sizer_parameters {
    float *$parameter,
    LUT *${parameter}_LUT,
}
float *Gain, float *Area, float *fc, float tolerance
) {
    #Foreach: input $::sizer_inputs {
        float point_${input}=min_$input;
        if (!isnan(*$input)) point_${input}=*$input;
        #tcl set step $::sizer_steps($input)
        float step_$input=$step;
        float chosen_$input=point_${input};
    }
    float point_distance=feasibility(
    #Foreach: input $::sizer_inputs {
        point_${input},
    }
    #Foreach: parameter $::sizer_parameters {
        $parameter,${parameter}_LUT,
    }
    Gain,Area,fc,tolerance
    );
```

```
float minimal_distance=point_distance;
#Foreach: input $::sizer_inputs {
    int index_$input;
    int start_$input=-1;
    int stop_$input=2;
    if (!isnan(*$input)) {
        start_$input=0;
        stop_$input=1;
    }
}
int continue_searching=1;
ordinal watchdog=0;
FILE *TRAIL=fopen("/tmp/trail.tcl","w+");
float trail_L=point_L;
float trail_W=point_W;
fprintf(TRAIL,"set data \{\n    %g %g\n",point_L,point_W);
#Foreach: search_dims [list $::sizer_voltage_inputs $::sizer_inputs] {
    continue_searching=1;
    while ((point_distance>0)&&(continue_searching)&&(watchdog++<10000000)) {
        if (((trail_L!=point_L)||(trail_W!=point_W))) {
            fprintf(TRAIL,"    %g %g\n",point_L,point_W);
            trail_L=point_L;
            trail_W=point_W;
        }
        continue_searching=0;
        minimal_distance=point_distance;
        #Foreach: input $::sizer_inputs {
            float chosen_step_$input=0;
        }
        #Foreach: input $search_dims {
            for (index_$input=start_$input;\
```

```
         index_$input<stop_$input;\
         index_$input++)
{
    #Foreach: input $::sizer_inputs {
        float try_$input=point_$input;
        if ((try_$input+step_$input*index_$input>max_$input)||\
           (try_$input+step_$input*index_$input<min_$input)) \
           continue;
        try_$input+=step_$input*index_$input;
    }
    #Foreach: input $::sizer_inputs {
        #Dinfo: "Trying $input step=%g" step_$input
    }
    float try_distance=feasibility(
    #Foreach: input1 $::sizer_inputs {
        try_${input1},
    }
    #Foreach: parameter1 $::sizer_parameters {
        $parameter1,${parameter1}_LUT,
    }
    Gain,Area,fc,tolerance
    );
    if (try_distance<minimal_distance) {
        #Foreach: input $::sizer_inputs {
            chosen_step_$input=step_$input*index_$input;
        }
        continue_searching=1;
        minimal_distance=try_distance;
    }
}
if (continue_searching) {
```

```
#Foreach: input $::sizer_inputs {
    #Dinfo: "Chosen $input step=%g" chosen_step_$input
}
while (1) {
    #Foreach: input $::sizer_inputs {
        float try_$input=point_$input;
        if ((try_$input+chosen_step_$input>max_$input)||\
           (try_$input+chosen_step_$input<min_$input)) \
           break;
    }
    #Foreach: input $::sizer_inputs {
        try_$input+=chosen_step_$input;
    }
    float try_distance=feasibility(
    #Foreach: input1 $::sizer_inputs {
        try_${input1},
    }
    #Foreach: parameter1 $::sizer_parameters {
        $parameter1,${parameter1}_LUT,
    }
    Gain,Area,fc,tolerance
    );
    if (try_distance>point_distance) break;
    #Foreach: input $::sizer_inputs {
        point_$input=try_$input;
    }
    #Foreach: input $::sizer_inputs {
        chosen_step_$input*=2;
    }
    point_distance=try_distance;
    #Dinfo: "Distance=%g" point_distance
```

```
                }
            }
        }
    }
    fprintf(TRAIL,"    %g %g\n",point_L,point_W);
    fprintf(TRAIL,"\}",point_L,point_W,point_distance);
    fclose(TRAIL);
    #Foreach: input $::sizer_inputs {
        *$input=point_${input};
    }
    #tcl set i 0
    #Foreach: input $::sizer_inputs {
        global_coord[$i]=point_${input};
        #tcl incr i
    }
    #Foreach: parameter $::sizer_parameters {
        #tcl set dim $::sizer_parameter_dim($parameter)
        #tcl set start_dim [expr 4-$dim]
        *${parameter}=${parameter}_LUT-
>interpolate(${parameter}_LUT,&(global_coord[$start_dim]));
    }
    // Parameters scaled by W/L get corrected here
    #Foreach: parameter {Ids gm go} {
        *${parameter}*=point_W/point_L;
    }
    // Special parameters calculations
    *Nth*=sqrt(*gm);
    *Area=point_W*point_L*1e12;
    *Nflicker*=*gm/sqrt(point_W*point_L);
    *sigmaVt*=(100/(*Vt));
    *Gain=20*log_ten(*gm/(*go));
```

```
    *fc=((*Nflicker)*(*Nflicker))/((*Nth)*(*Nth));
    return(point_distance);
}
```

The web-server uses Tcl socket to implement the website. It is a stand-alone server, independent of Apache or other web infrastructure.

```
# \
exec $RAMSPICE/ramspice $0 $argv
get_opts

foreach arg [lrange $argv 2 end] {
    if {[regexp {^\-(\S+)$} $arg -> found_key]} {
        set key $found_key
        set $key {}
        continue
    }
    set $key $arg
}

default opt(tech) tsmc040
default ::randomized {}
set tech $opt(tech)
source $::env(RAMSPICE)/Etc/Tech_DB/$::opt(tech)/binning_$::opt(tech).tcl
set ::mho &#x2127
set ::ohm &#x2126
default opt(port) 1024
set ::port $opt(port)
while {[catch {set socket [socket -server server $::port]}]} {
    incr ::port
```

```
    }
    proc server {chan addr port} {
        fconfigure $chan -buffering line ;# NOT -blocking 0 (see below!)
        while {[gets $chan line]>=0} {
            puts $line
            if {[catch $line res]} {
    Error: $res
    Error: $::errorInfo
                continue
            }
            puts $chan $res
            flush $chan
            break
        }
        close $chan
    }
    proc respond {code body {head ""}} {
        return "HTTP/1.0 $code ???\nContent-Type: text/html; charset=ISO-8859-1\nConnection: 
close\n$head\n$body"
    }
    set ::solution {}
    proc Log: {args} {
        set text $args
        if {[llength $text]==1} {
            set text [lindex $text 0]
        }
        regsub -all {\$:*} $text {} text
        regsub -all {\[\s*LUT\s+(\S+)\s+([^\]]+)\]} $text {\1(\2)} text
        append ::solution "$text<br>\n"
    }
    set ::open_fields {}
```

```
proc GET {args} {
    foreach var $::reset_list {
        set ::$var {}
    }
    array unset ::original_value
    set error {}
    set ::tolerance_analysis 0
    if {[regexp {\?(\S+)\s} $args -> assignment_list]} {
        set ::open_fields {}
        set ::randomized {}
        foreach assignment [split $assignment_list &] {
            set field [lindex [split $assignment =] 0]
            set value [lindex [split $assignment =] 1]
            de_http field
            de_http value
            set ::original_value($field) $value
            if {$value!={}} {
                if {[regexp {^\s*([xyz])\s*(\S*)} $value -> axis goal]} {
                    set ::tolerance_analysis 1
                    set ::tolerance($axis) $field
                    set value $goal
                }
                if {$value=="#"} {
Info: Random value for $field: $::const($field,min) $::const($field,max)
                    set value "rand()*($::const($field,max)-
$::const($field,min))+$::const($field,min)"
                    lappend ::randomized $field
                }
                if {![catch {set expr_value [uplevel \#0 "expr $value"]}]} {
                    set value $expr_value
                    if {$::const($field,factor)!="-"} {
```

```
                                append value "*($::const($field,factor))"
                            }
                            if {$::const($field,min)!={}} {
                                set min $::const($field,min)
                                if {![catch {set expr_min [uplevel \#0 "expr $min"]} msg]} {
                                    set min $expr_min
                                } else {
        Error: $msg
                                }
                                if {$value<$min} {
                                    append error "$field was assigned a value outside
predefined limits: $value<$min<br>"
                                    continue
                                }
                            }
                            if {$::const($field,max)!={}} {
                                set max $::const($field,max)
                                if {![catch {set expr_max [uplevel \#0 "expr $max"]} msg]} {
                                    set max $expr_max
                                } else {
        Error: $msg
                                }
                                if {$value>$max} {
                                    append error "$field was assigned a value outside
predefined limits: $value>$max<br>"
                                    continue
                                }
                            }
                        }
        Info: Setting $field to $value
                } else {
                        lappend ::open_fields $field
```

```
                }
                set ::$field $value
            }
            set ::solution {}
            set ::step_index 0
            set ::taboo_list {}
            pre_calculate
            if {[catch calculate msg]} {
                append error [concat $msg <br> $::errorInfo]
            }
            post_calculate
        }
        if {$error!={}} {
    Error: $error
        }
        set I [open /tmp/tmp.html r]
        append post_solution [read $I]
        close $I
        append post_solution $::solution
        return [respond 200 "<html><body>[subst $::HTML]<font
color=\"red\">$error</font>$post_solution</body></html>"]
    }
    proc post_calculate {} {}
    proc de_http {varname} {
        upvar $varname var
        while {[regexp {^(.*)%([0-9A-Fa-f][0-9A-Fa-f])(.*)$} $var -> pre code post]} {
            set var $pre
            append var [format "%c" 0x$code]
            append var $post
        }
    }
```

```
set unknown {
    if {[string match *: [lindex $args 0]]} {
Info: ignoring unknown command $args
        return
    }
}
append unknown [info body unknown]
proc unknown args $unknown
set ::fields {}
set ::form_fields {}
set ::target_fields {}
set ::source_fileds {}
set ::Lmin 180e-9
proc form_sep {title} {
    lappend ::form_fields [list @sep $title]
}
proc form_field {field display factor type default min max unit} {
    lappend ::fields $field
    lappend ::form_fields $field
    set ::$field $default
    if {$type=="text"} {
        set ::const($field,type) "type=\"$type\""
    } else {
        set ::const($field,type) "list=\"$type\""
    }
    if {$min!={} && $max!={}} {
        set min [uplevel \#0 "expr $min"]
        set max [uplevel \#0 "expr $max"]
        set ::const($field,min) $min
        set ::const($field,max) $max
    } else {
```

```
            set ::const($field,min) $min
            set ::const($field,max) $max
        }
        if {$::const($field,min)>$::const($field,max)} {
            set tmp $::const($field,min)
            set ::const($field,min) $::const($field,max)
            set ::const($field,max) $tmp
        }
        set ::const($field,unit) $unit
        if {$display=="-"} {
            set display $field
        }
        set ::const($field,display) $display
        set ::const($field,factor) $factor
    }
    proc min {a b} {
        if {$a<$b} {
            return $a
        }
        return $b
    }

    proc gen_form {} {
        set retval "<table border=\"1\"><tr><td align=\"center\"><b>Parameter</b></td><td
align=\"center\"><b>Input</b></td><td align=\"center\"><b>Calculation</b></td><td
align=\"center\"><b>Source</b></td><td align=\"center\"><b>Simulation</b></td><td
align=\"center\"><b>|Error|</b></td><td align=\"center\"><b>Error\[%\]</b></td></tr>"
        foreach field $::form_fields {
            if {[lindex $field 0]=="@sep"} {
                append retval "<tr><td colspan=\"7\"><h3>[lindex $field 1]</h3></td></tr>"
                continue
            }
```

```
            set value [set ::$field]
            if {$value!={}} {
                if {[catch {set value [uplevel \#0 "expr $value"]}]} {
                    set value [set ::$field]
                }
            }
            set width [expr 80-[string length $field]]
            set sim_value ""
            set abs_err ""
            set rel_err ""
            if {[info exists ::simulated($field)]} {
                set sim_value [eng $::simulated($field) $::const($field,unit)]
        Info: $field value=$value sim=$::simulated($field)
                if {[catch {set abs_err [eng [expr $value-$::simulated($field)]
$::const($field,unit)]}]} {
                    set abs_err 0
                }
                if {[catch {set rel_err [eng [expr ($value/$::simulated($field)-
1)*100] %]}]} {
                    set rel_err 0
                }
            }
            set hint [eng $::const($field,min) $::const($field,unit)]
            append hint "   -   "
            append hint [eng $::const($field,max) $::const($field,unit)]
            set field_token "<div title=\"$hint\">$::const($field,display)</div>"
            if {![info exists ::step_lookup($field)]} {
                set step_reference (defaulted)
            } else {
                set step_reference "(step $::step_lookup($field))"
            }
            set display_value {}
```

```
            if {[info exists ::original_value($field)]} {
                set display_value $::original_value($field)
            }
            set input_color black
            if {$::Distance>=$::Tolerance} {
                set input_color red
            }
            append retval "<tr><td>$field_token</td><td><input $::const($field,type)
name=\"$field\" value=\"$display_value\" width=\"$width\" style=\"color:
$input_color;\"></td><td>[eng $value
$::const($field,unit)]</td><td>$step_reference</td><td>$sim_value</td><td>$abs_err</td><td>$re
l_err</td></tr>"
        }
        append retval "</table>"
        return $retval
    }
    proc LUT {name corner args} {
        if {![@ /look_up_tables/$::device/$name/$corner exists]} {
    Warning: look up in $name:  $args => does not exist
            return -1
        }
        set retval [uplevel \#0 "@ /look_up_tables/$::device/$name/$corner calc $args"]
    #  Info: @ /look_up_tables/$::device/$name/$corner calc $args => $retval
        if {[string match *n* $retval]} return -1
        if {[string match *N* $retval]} return -1
    #  Info: look up in $name ($::device):  $args => $retval
        return $retval
    }
    ############ Solver
    set ::link_index 0
    set ::reset_list {}
    proc link_bwd {var dep code} {
        set ::calc($::link_index,var) $var
```

```
    set ::calc($::link_index,code) [regsub -all {\$:*} $code {$::}]
    set ::calc($::link_index,dir) bwd
    set ::calc($::link_index,dep) $dep
    default ::calc_list($var)
    lappend ::calc_list($var) $::link_index
    incr ::link_index
}
proc link {var code args} {
    default ::$var
    if {[lsearch $::fields $var]==-1} {
        lappend ::reset_list $var
    }
    if {[lsearch $::target_fields $var]==-1} {
        lappend ::target_fields $var
    }
    set ::calc($::link_index,var) $var
    set ::calc($::link_index,code) [regsub -all {\$:*} $code {$::}]
    set ::calc($::link_index,dir) fwd
    regsub -all {\$([A-Za-z_0-9]+)} $code {`$\1`} var_list
    set varlist $var
    foreach section [split $var_list `] {
        if {[regexp {\$([A-Za-z_0-9]+)} $section -> varname]} {
            if {[lsearch $varlist $varname]==-1} {
                lappend varlist $varname
            }
        }
    }
    set ::calc($::link_index,dep) [lrange $varlist 1 end]
    default ::calc_list($var)
    lappend ::calc_list($var) $::link_index
    incr ::link_index
```

```
        for {set i 1} {$i<[llength $varlist]} {incr i} {
            set pre [expr $i-1]
            set post [expr $i+1]
            link_bwd [lindex $varlist $i] [concat [lrange $varlist 0 $pre] [lrange $varlist
$post end]] $code
        }
        foreach {flag value} $args {
            switch $flag {
                "-unit" {
                    set ::const($var,unit) $value
                }
                "-min" {
                    set ::const($var,min) $value
                }
                "-max" {
                    set ::const($var,max) $value
                }
            }
        }
    }
    proc untaboo {var} {
        if {[set i [lsearch $::taboo_list $var]]!=-1} {
            set ::taboo_list [lreplace $::taboo_list $i $i]
        }
    }
    proc . {} {
        return [string repeat . [info level]]
    }
    proc calc_var {i} {
        if {$::device=="pch"} {
            foreach var {::Vgs ::Vds ::Vbs} {
```

```
            set $var [expr -[set $var]]
        }
    }
    set val [subst $::calc($i,code)]
    if {[catch "expr $val" msg]} {
Log: $msg
        if {$::device=="pch"} {
            foreach var {::Vgs ::Vds ::Vbs} {
                set $var [expr -[set $var]]
            }
        }
        return $val
    }
    set retval [eval "expr $val"]
    set name $::calc($i,var)
    if {![info exists ::track($name)]} {
        set ::track($name) $retval
    } else {
        set change [expr abs($retval/$::track($name)-1)*100]
        if {$change>$::max_change} {
            set ::max_change $change
        }
    }
    if {$::device=="pch"} {
        foreach var {::Vgs ::Vds ::Vbs} {
            set $var [expr -[set $var]]
        }
    }
    return $retval
}
#/**
```

```
# * Recursively tries to calculate each variable it's given
# * By default, all form-fields are calculated
# * @param  $varlist - a list of variables to calculate
# */
proc calculate {{varlist {}}} {
# By default, calculate every field
    set ::max_change 0
    set vars_to_be_calculated $varlist
    if {$varlist=={}} {
        set vars_to_be_calculated $::fields
    }
# A signal to the upper level in the recursion that this variable list cannot be
completed
# and another expression may be needed to calculate the variable
    set backtrack_from_this_calculation 0
# Scan all variables
    foreach var $vars_to_be_calculated {
        # No need to re-calculate a variable that has assigned value
        if {[set ::$var]!={}} continue
        # Taboo-list is used to prevent infinite loops on dependence cycles
        lappend ::taboo_list $var
        for {set i 0} {$i<$::link_index} {incr i} {
            if {$::calc($i,var)!="$var"} continue
            if {$::calc($i,dir)!="fwd"} continue
            # If I'm here, I found a direct expression
    # First make sure there are no dependece cycles in this expression
    set backtrack 0
    foreach dep_var $::calc($i,dep) {
        if {[lsearch $::taboo_list $dep_var]!=-1 && [set ::$dep_var]=={}} {
            set backtrack 1
        break
```

```
            }
        }
        if {$backtrack} continue
        # Now decend into each variable and calculate it recursively
        if {[calculate $::calc($i,dep)]} continue
        # If I'm here, the variables this one depends on are calculated and the chosen
expression is ready to be calculated
                # Log the calculation step for the HTML page
                incr ::step_index
      Log: $::step_index Calculating $var based on $::calc($i,dep)
                # Calculate the variable !
                set ::$var [calc_var $i]
                #remove it from the taboo list. Cycles containing this variables are broken
anyway because it is assigned a value
                untaboo $var
                # Log the result
                set unit {}
                if {[info exists ::const($var,unit)]} {
                    set unit $::const($var,unit)
                }
      Log: $var=$::calc($i,code)=[eng [set ::$var] $unit]
                set ::step_lookup($var) $::step_index
                break
            }
            # Go to the next var if done
            if {[set ::$var]!={}} continue
            set backtrack_from_this_calculation 1
            set ::$var {}
            untaboo $var
        }
        if {$varlist!={}} {
            return $backtrack_from_this_calculation
```

```
            }

        # Now for the more expensive and expansive solver: the randomized sample, deflating solution-space search

            set independent_vars {}

            foreach var $::source_fields {

                set val [set ::$var]

                if {[set ::$var]=={}} {

                    lappend independent_vars $var

                    set centre($var) [expr "($::const($var,min)+$::const($var,max))/2"]

                }

            }

            if {$independent_vars=={}} return

            set dependent_vars {}

            foreach var $::target_fields {

                if {[set ::$var]!={}} {

                    set target_value($var) [set ::$var]

                    lappend dependent_vars $var

                }

                set ::$var {}

            }

        # Do this N times: from a sample of M coordinate combinations pick the one with the minimal squared-error and centre the next solution space arround it.

        # Each space is the size of the original divided by the iteration number: 1 1/2 1/3 .... 1/N

        # To make sure the error is only descending, only reset the error variable at the top level and then update the centroid only if smaller error figure found.

            set error {}

            for {set iteration 1} {$iteration<=2} {incr iteration} {

                set ::max_change 0

                # Find span for each variable. Make sure to clip the solution space according to original min/max figures

                foreach var $independent_vars {

                    set span [expr "($::const($var,max)-$::const($var,min))/$iteration"]
```

```
                    set min($var) [expr $centre($var)-$span/2]
                    set max($var) [expr $centre($var)+$span/2]
                    # Clip min and max to top-level feasible boundaries
                    if {$min($var)<$::const($var,min)} {
                        set min($var) $::const($var,min)
                    }
                    if {$max($var)>$::const($var,max)} {
                        set max($var) $::const($var,max)
                    }
                }
                # Withing the shrinking solution space, draw M samples and calculate square-
error for each
                for {set sample_index 0} {$sample_index<64} {incr sample_index} {
                    # Get a random value per independent variable
                    foreach var $independent_vars {
                        set ::$var [expr $min($var)+rand()*($max($var)-$min($var))]
                    }
                    # Run calculation "forward" as above
                    calculate $dependent_vars
                    # Calculate square-error
                    set local_error 0
                    foreach var $dependent_vars {
                        set local_error [expr $local_error+pow([set ::$var]-
$target_value($var),2)]
                        #      set local_error [expr ($local_error)*$::L*$::W]
                    }
                    # A new minimizer? If so, keep it as the next centre for the next iteration
                    if {$error=={} || $local_error<$error} {
                        set error $local_error
                        foreach var $independent_vars {
                            set centre($var) [set ::$var]
                        }
```

```
                }
                # Reset the fields so the next call to calculate doesn't skip them
            foreach var $::target_fields {
                set ::$var {}
            }
        }
#      Info: error=$error
#       if {$::max_change<0.01 && $i>3} break
}
# Searched variables still show in the report, although I need to find a creative way
to make it convincing.
set ::solution {}
set ::step_index 0
foreach var $independent_vars {
    set ::$var $centre($var)
    incr ::step_index
    Log:
    Log: $::step_index $var was automatically searched to fit given $dependent_vars
    set unit {}
    if {[info exists ::const($var,unit)]} {
        set unit $::const($var,unit)
    }
    Log: $var=[eng [set ::$var] $unit]
    set ::step_lookup($var) $::step_index
}
# The rest of the report is populated with this final calculation, which shouldn't end
with a search (all independents were searched)
                calculate $::target_fields
            }
##### Import circuit-specific fields and rules
        source $::env(RAMSPICE)/Etc/utils/sizer_functionality.tcl
        foreach field $::fields {
```

```
            if {[lsearch $::target_fields $field]==-1} {
                lappend ::source_fields $field
            }
        }
####################
Info: [.] Ready! Log into: $::env(HOSTNAME):$::port
        vwait forever
```

B. 5. Transistor-Level Characterization

B. 5. 1. Low-level Data Access

"Characterization" here means the operation of reconstructing a model out of simulation results. Two type of transistors are modeled: p-channel and n-channel MOSFET, in various manufacturing technologies and physical properties. The characterization loops use the built-in .dc SPICE command, with a modification that allows it to run more than 2 nesting levels:

```
int
DCTsetParm(CKTcircuit *ckt, JOB *anal, int which, IFvalue *value)
{
    TRCV *job = (TRCV *) anal;
    NG_IGNORE(ckt);
    switch(which) {

// Notice usage of Tcl-preprocessor loop and variables on original NGSPICE code to
enhance its functionality to ANALYSIS_NESTING_DEPTH-deep nesting levels

#For: {set i 0} {$i<$::ANALYSIS_NESTING_DEPTH} {incr i} {
            #tcl set j [expr $i+1]
        case DCT_START$j:
            job->TRCVvStart[$i] = value->rValue;
            job->TRCVnestLevel = MAX($i, job->TRCVnestLevel);
            job->TRCVset[$i] = TRUE;
            break;

        case DCT_STOP$j:
            job->TRCVvStop[$i] = value->rValue;
            job->TRCVnestLevel = MAX($i, job->TRCVnestLevel);
            job->TRCVset[$i] = TRUE;
            break;
```

```
        case DCT_STEP$j:
            job->TRCVvStep[$i] = value->rValue;
            job->TRCVnestLevel = MAX($i, job->TRCVnestLevel);
            job->TRCVset[$i] = TRUE;
            break;

        case DCT_NAME$j:
            job->TRCVvName[$i] = value->uValue;
            job->TRCVnestLevel = MAX($i, job->TRCVnestLevel);
            job->TRCVset[$i] = TRUE;
            break;

        case DCT_TYPE$j:
            job->TRCVvType[$i] = value->iValue;
            job->TRCVnestLevel = MAX($i, job->TRCVnestLevel);
            job->TRCVset[$i] = TRUE;
            break;
        }

    default:
        return(E_BADPARM);
    }
    return(OK);
}


static IFparm DCTparms[] = {
#For: {set i 1} {$i<=$::ANALYSIS_NESTING_DEPTH} {incr i} {
        { "start$i",DCT_START$i, IF_SET|IF_REAL,     "starting voltage/current"},
        { "stop$i",DCT_STOP$i,  IF_SET|IF_REAL,     "ending voltage/current" },
```

```
            { "step$i",DCT_STEP$i,  IF_SET|IF_REAL,     "voltage/current step" },
            { "name$i",DCT_NAME$i,  IF_SET|IF_INSTANCE, "name of source to step" },
            { "type$i",DCT_TYPE$i,  IF_SET|IF_INTEGER,  "type of source to step" },
        }
    }
```

A DC sweep for characterization is done on 3 dimensions: V_{GS}, V_{DS} and V_{BS}. The geometrical dimensions L and W are not part of the sweep, but implemented as a big circuit that contains all combinations of L and W. The results are stored internally in NGSPICE vectors and need to be extracted and organized for re-packaging in the LUT. The following Tcl commands provide interface to that data and all the operations needed to process them:

1. get_vectors - Get all vector names
2. get_spice_data - Get data from vector:
    a. Entry by index
    b. all - entries as a Tcl list
    c. length – size of vector in entries
    d. end – the last entry
    e. trigger index t> t< tx – where signal crosses threshold up, down or either
    f. statistics – average, stdev
    g. Special case: RAMSpice-specific global variables that expose last simulation results
3. set_spice_var – set a value to a RAMSpice global variable
4. save_characterization_slice – a set of commands that extract a set of vectors (slice) and copy them as binary data to file:
    a. Save slice as is

b. Mark slice as baseline

c. Save difference with last baseline slice

d. Save delta vectors generated from entries in this slice

e. Generate delta vectos and save differences to last baseline slice (b and c combined).

5. load_characterization_slice – loads a saved slice into a LUT.

The minimal set of operations: get_vectors and get_spice_data, subcommands all and length, would be enough to perform all characterization operations, since they can be used for any other operation in the context of a Tcl script. However, leaving all post-processing and re-packagind of data to Tcl scripts is extremely inefficient and may prolong execution from few minutes to many hours. The choice to use a disk-file as medium for slices came to simplify combining data from parallel processes.

```
static int
get_vectors (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[])
{
    if (argc!=1) {
        #Error: "get_vectors requires no arguments"
        return TCL_ERROR;
    }
    Tcl_ResetResult(interp);
```

```
    if (plot_cur==NULL) {
        #Error: "No vectors exist"
        return TCL_ERROR;
    }
    struct plot *any_plot=plot_list;
    while (any_plot) {
        struct dvec *d=any_plot->pl_dvecs;
        while (d) {
            Tcl_AppendElement(interp,d->v_name);
            d = d->v_next;
        }
        any_plot=any_plot->pl_next;
    }
    return TCL_OK;
}

static int
get_spice_data (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[])
{
    if (argc!=3) {
        #Error: "get_vector_data requires vector name and index/command"
        return TCL_ERROR;
    }
    Tcl_ResetResult(interp);
    if (plot_cur==NULL) {
        #Error: "No vectors exist"
        return TCL_ERROR;
    }
    char *vector_name=argv[1];
    struct dvec *d=plot_cur->pl_dvecs;
    struct dvec *V=NULL;
```

```
while (d) {
    if(strcmp(vector_name,d->v_name)==0) {
        V=d;
        break;
    }
    d = d->v_next;
}
// Try to get the vector in other plots
if (V==NULL) {
    struct plot *any_plot=plot_list;
    while (any_plot) {
        d=any_plot->pl_dvecs;
        while (d) {
            if(strcmp(vector_name,d->v_name)==0) {
                V=d;
                break;
            }
            d = d->v_next;
        }
        if (V) break;
        any_plot=any_plot->pl_next;
    }
}
// Last chitce: maybe the user wants a global variable
#Foreach: global_var $::global_c_variables {
    if (strcmp(vector_name,"$global_var")==0) {
        tcl_append_float(interp,$global_var);
        return TCL_OK;
    }
}
if (V==NULL) {
```

```
    #Error: "get_vector_data did not get a valid vector name '%s'" vector_name
    return TCL_ERROR;
}
char *command=argv[2];
if (strcmp(command,"length")==0) {
    tcl_append_int(interp,V->v_length);
    return TCL_OK;
}
if (strcmp(command,"end")==0) {
    tcl_append_float(interp,V->v_realdata[V->v_length-1]);
    return TCL_OK;
}
if (command[0]=='>') {
    int i;
    int res=-1;
    float th=atof(&(command[1]));
    for (i=0;i<V->v_length;i++) {
        if (V->v_realdata[i]>th) {
            res=i;
            break;
        }
    }
    tcl_append_int(interp,res);
    return TCL_OK;
}
if (command[0]=='<') {
    int i;
    int res=-1;
    float th=atof(&(command[1]));
    for (i=0;i<V->v_length;i++) {
        if (V->v_realdata[i]<th) {
```

```
                res=i;
                break;
            }
        }
        tcl_append_int(interp,res);
        return TCL_OK;
    }
    if (command[0]=='x') {
        int i;
        int res=-1;
        float th=atof(&(command[1]));
        for (i=0;i<V->v_length;i++) {
            if (((V->v_realdata[i]<th)&&(V->v_realdata[0]>th))||((V-
>v_realdata[i]>th)&&(V->v_realdata[0]<th))) {
                res=i;
                break;
            }
        }
        tcl_append_int(interp,res);
        return TCL_OK;
    }
    if (command[0]=='t') {
        struct dvec *T=NULL;
        d=plot_cur->pl_dvecs;
        while (d) {
            if(strcmp("time",d->v_name)==0) {
                T=d;
                break;
            }
            d = d->v_next;
        }
```

```
if (T==NULL) {
    #Error: "get_vector_data cannot process operator t, no time vector found."
    return TCL_ERROR;
}
if (command[1]=='>') {
    int i;
    int res=-1;
    float th=atof(&(command[2]));
    for (i=0;i<V->v_length;i++) {
        if (V->v_realdata[i]>th) {
            res=i;
            break;
        }
    }
    if ((res==-1)||(res>=T->v_length)) {
        tcl_append_float(interp,-1);
        return TCL_OK;
    }
    tcl_append_float(interp,T->v_realdata[res]);
    return TCL_OK;
}
if (command[1]=='<') {
    int i;
    int res=-1;
    float th=atof(&(command[2]));
    for (i=0;i<V->v_length;i++) {
        if (V->v_realdata[i]<th) {
            res=i;
            break;
        }
    }
```

```
                if ((res==-1)||(res>=T->v_length)) {
                    tcl_append_float(interp,-1);
                    return TCL_OK;
                }
                tcl_append_float(interp,T->v_realdata[res]);
                return TCL_OK;
            }
            if (command[1]=='x') {
                int i;
                int res=-1;
                float th=atof(&(command[2]));
                for (i=0;i<V->v_length;i++) {
                    if (((V->v_realdata[i]<th)&&(V->v_realdata[0]>th))||((V-
>v_realdata[i]>th)&&(V->v_realdata[0]<th))) {
                        res=i;
                        break;
                    }
                }
                if (res==-1) {
                    tcl_append_float(interp,-1);
                    return TCL_OK;
                }
                if (res>=T->v_length) {
                    tcl_append_float(interp,-2);
                    return TCL_OK;
                }
                tcl_append_float(interp,T->v_realdata[res]);
                return TCL_OK;
            }
        }
        if (strcmp(command,"type")==0) {
```

```
            if (V->v_realdata) Tcl_AppendElement(interp,"real");
            else if (V->v_compdata) Tcl_AppendElement(interp,"complex");
            return TCL_OK;
        }
        if (strcmp(command,"average")==0) {
            float average=0;
            int i;
            for (i=0;i<V->v_length;i++) average+=V->v_realdata[i];
            tcl_append_float(interp,average/V->v_length);
            return TCL_OK;
        }
        if (strcmp(command,"stddev")==0) {
            float average=0;
            int i;
            for (i=0;i<V->v_length;i++) average+=V->v_realdata[i];
            average/=V->v_length;
            float stddev=0;
            for (i=0;i<V->v_length;i++) stddev+=(V->v_realdata[i]-average)*(V-
>v_realdata[i]-average);
            stddev/=V->v_length;
            tcl_append_float(interp,sqrt(stddev));
            return TCL_OK;
        }
        if (strcmp(command,"all")==0) {
            int i;
            for (i=0;i<V->v_length;i++) tcl_append_float(interp,V->v_realdata[i]);
            return TCL_OK;
        }
        int index =atoi(command);
        if (index>=V->v_length) {
            #Error: "get_vector_data vector %s has only %d entries" vector_name,V->v_length
```

```
        return TCL_ERROR;
    }
    if (V->v_realdata) {
        tcl_append_float(interp,V->v_realdata[index]);
    } else if (V->v_compdata) {
        tcl_append_float(interp,V->v_compdata[index].cx_real);
        tcl_append_float(interp,V->v_compdata[index].cx_imag);
    }
    return TCL_OK;
}

static int
set_spice_var (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[])
{
    if ((argc!=3)&&(argc!=2)) {
        #Error: "usage: %s <var> [<val>]" argv[0]
        return TCL_ERROR;
    }
    if (argc==2) {
        #Foreach: global_var $::global_c_variables {
            if (strcmp(argv[1],"$global_var")==0) {
                tcl_append_float(interp,$global_var);
                return TCL_OK;
            }
        }
        #Error: "No such spice variable %s" argv[1]
        return TCL_ERROR;
    }
    #Foreach: global_var $::global_c_variables {
        if (strcmp(argv[1],"$global_var")==0) {
            $global_var=atof(argv[2]);
```

```
                return TCL_OK;
            }
        }
        #Error: "No such spice variable %s" argv[1]
        return TCL_ERROR;
    }
    static int
    baseline_characterization_slice (ClientData clientData,Tcl_Interp *interp,int argc,char
*argv[]) {
        if (argc!=1) {
            #Error: "usage: %s " argv[0]
            return TCL_ERROR;
        }
        save_slice_base=plot_cur;
        return TCL_OK;
    }
    static int
    save_characterization_slice (ClientData clientData,Tcl_Interp *interp,int argc,char
*argv[]) {
        if ((argc!=7)&&(argc!=6)) {
            #Error: "usage: %s <file name> <array sizes> <per vector dim>  <slice dim>
<post indices> [<factor>]" argv[0]
            return TCL_ERROR;
        }
        ordinal i;
        float factor=1;
        int factor_mode=0;
        if (argc==7) {
            if (argv[6][0]=='/') {
                factor=atof(&(argv[6][1]));
                factor_mode=1;
            } else {
```

```
        factor=atof(argv[6]);
    }
}
FILE *O=fopen(argv[1],"w");
int dim;
char **sizes_string;
Tcl_SplitList(interp,argv[2],&dim,&sizes_string);
ordinal size[MAXDIM];
for (i=0;i<dim;i++) size[i]=atoi(sizes_string[i]);
free(sizes_string);

ordinal offset[MAXDIM];
offset[0]=1;
for (i=1;i<dim;i++) offset[i]=offset[i-1]*size[i-1];

ordinal vector_dim=atoi(argv[3]);
ordinal slice_dim=atoi(argv[4]);
int starting_index=0;
if (slice_dim==1) starting_index=1;
int postfix_dim;
char **postfix_indices_string;
Tcl_SplitList(interp,argv[5],&postfix_dim,&postfix_indices_string);
ordinal postfix_indices[MAXDIM];
for (i=0;i<postfix_dim;i++) postfix_indices[i]=atoi(postfix_indices_string[i]);
free(postfix_indices_string);
ordinal initial_offset=0;
for (i=0;i<postfix_dim;i++)
initial_offset+=postfix_indices[i]*offset[i+vector_dim+slice_dim];
struct dvec *d=plot_cur->pl_dvecs;
save_slice_base=plot_cur;
#Info: "Saving vectors for slice %s in %s" argv[5],argv[1]
```

```
            while (d) {
                if (d->v_name[0]!='V') {
                    d = d->v_next;
                    continue;
                }
                char vector_id[128];
                for (i=2;d->v_name[i]!=')';i++) {
                    if (d->v_name[i]=='_') {
                        vector_id[i-2]=' ';
                    } else {
                        vector_id[i-2]=d->v_name[i];
                    }
                }
                vector_id[i-2]=0;
                int tmp_dim;
                char **vector_index_string;
                Tcl_SplitList(interp,vector_id,&tmp_dim,&vector_index_string);
                if (slice_dim!=tmp_dim) {
                    d = d->v_next;
                    continue;
                }
                ordinal final_offset=initial_offset;
                for (i=0;i<slice_dim;i++) final_offset+=(atoi(vector_index_string[i])-
starting_index)*offset[i+vector_dim];
                free(vector_index_string);
                write_ordinal(O,final_offset);
                write_ordinal(O,d->v_length);
                float w;
                if (factor_mode) {
                    for (i=0;i<d->v_length;i++) {
                        w=factor/d->v_realdata[i];
```

```
                write_float(O,w);
            }
        } else {
            for (i=0;i<d->v_length;i++) {
                w=d->v_realdata[i]*factor;
                write_float(O,w);
                //      #Info: "%s) Ids(%d)=%g" d->v_name i w
            }
        }
        /*
        for (i=0;i<d->v_length;i++) {
            if (isnan(d->v_realdata[i])) {
                #Error: "%s(%d)=%g" d->v_name i d->v_realdata[i]
            } else {
                #Info: "%s(%d)=%g" d->v_name i d->v_realdata[i]*factor
            }
        }
        */
        d = d->v_next;
    }
    fclose(O);
    return TCL_OK;
}
static int
save_characterization_slice_differential (ClientData clientData,Tcl_Interp *interp,int
argc,char *argv[]) {
    if ((argc!=7)&&(argc!=6)) {
        #Error: "usage: %s <file name> <array sizes> <per vector dim>  <slice dim>
<post indices> [<factor>]" argv[0]
        return TCL_ERROR;
    }
    if (save_slice_base==NULL) {
```

```
            #Error: "(%s) cannot save a differential slice before a baseline slice was saved" argv[0]
            return TCL_ERROR;
        }
        ordinal i;
        float factor=1;
        int factor_mode=0;
        if (argc==7) {
            if (argv[6][0]=='/') {
                factor=atof(&(argv[6][1]));
                factor_mode=1;
            } else {
                factor=atof(argv[6]);
            }
        }
        FILE *O=fopen(argv[1],"w");
        int dim;
        char **sizes_string;
        Tcl_SplitList(interp,argv[2],&dim,&sizes_string);
        ordinal size[MAXDIM];
        for (i=0;i<dim;i++) size[i]=atoi(sizes_string[i]);
        free(sizes_string);

        ordinal offset[MAXDIM];
        offset[0]=1;
        for (i=1;i<dim;i++) offset[i]=offset[i-1]*size[i-1];

        ordinal vector_dim=atoi(argv[3]);
        ordinal slice_dim=atoi(argv[4]);
        int starting_index=0;
        if (slice_dim==1) starting_index=1;
```

```
            int postfix_dim;
            char **postfix_indices_string;
            Tcl_SplitList(interp,argv[5],&postfix_dim,&postfix_indices_string);
            ordinal postfix_indices[MAXDIM];
            for (i=0;i<postfix_dim;i++) postfix_indices[i]=atoi(postfix_indices_string[i]);
            free(postfix_indices_string);
            ordinal initial_offset=0;
            for (i=0;i<postfix_dim;i++)
initial_offset+=postfix_indices[i]*offset[i+vector_dim+slice_dim];
            struct dvec *d=plot_cur->pl_dvecs;
            #Info: "Saving vectors for slice %s (differential) in %s" argv[5],argv[1]
            while (d) {
                if (d->v_name[0]!='V') {
                    d = d->v_next;
                    continue;
                }
                struct dvec *b=save_slice_base->pl_dvecs;
                while (b) {
                    if (strcmp(b->v_name,d->v_name)==0) break;
                    b=b->v_next;
                }
                if (b==NULL) {
                    #Error: "(%s) vector %s has no baseline" argv[0],d->v_name
                    return TCL_ERROR;
                }
                if (d->v_length!=b->v_length) {
                    #Error: "(%s) vector %s is of length %d, while its baseline has length %d"
argv[0],d->v_name,d->v_length,b->v_length
                    return TCL_ERROR;
                }
                char vector_id[128];
                for (i=2;d->v_name[i]!=')';i++) {
```

```
            if (d->v_name[i]=='_') {
                vector_id[i-2]=' ';
            } else {
                vector_id[i-2]=d->v_name[i];
            }
        }
        vector_id[i-2]=0;
        int tmp_dim;
        char **vector_index_string;
        Tcl_SplitList(interp,vector_id,&tmp_dim,&vector_index_string);
        if (slice_dim!=tmp_dim) {
            d = d->v_next;
            continue;
        }
        ordinal final_offset=initial_offset;
        for (i=0;i<slice_dim;i++) final_offset+=(atoi(vector_index_string[i])-
starting_index)*offset[i+vector_dim];
        free(vector_index_string);
        write_ordinal(O,final_offset);
        write_ordinal(O,d->v_length);
        /*        for (i=0;i<d->v_length;i++) {
            printf("(%g-%g)*%g\n",d->v_realdata[i],b->v_realdata[i],factor);
            fflush(stdout);
        }*/
        if (factor_mode) {
            for (i=0;i<d->v_length;i++) write_float(O,factor/(d->v_realdata[i]-b-
>v_realdata[i]));
        } else {
            for (i=0;i<d->v_length;i++) write_float(O,(d->v_realdata[i]-b-
>v_realdata[i])*factor);
        }
        d = d->v_next;
```

```
        }
        fclose(O);
        return TCL_OK;
    }
    static int
    save_characterization_slice_delta (ClientData clientData,Tcl_Interp *interp,int
argc,char *argv[]) {
        if ((argc!=9)&&(argc!=8)) {
            #Error: "usage: %s <file name> <array sizes> <per vector dim>  <slice dim>
<post indices> <vector order list> <first offsets> [<factor>]" argv[0]
            return TCL_ERROR;
        }
        ordinal i;
        float factor=1;
        int factor_mode=0;
        if (argc==9) {
            if (argv[8][0]=='/') {
                factor=atof(&(argv[8][1]));
                factor_mode=1;
            } else {
                factor=atof(argv[8]);
            }
        }
        FILE *O=fopen(argv[1],"w");
        int dim;
        char **sizes_string;
        Tcl_SplitList(interp,argv[2],&dim,&sizes_string);
        ordinal size[MAXDIM];
        for (i=0;i<dim;i++) size[i]=atoi(sizes_string[i]);
        free(sizes_string);

        ordinal offset[MAXDIM];
```

```
            offset[0]=1;

            for (i=1;i<dim;i++) offset[i]=offset[i-1]*size[i-1];


            ordinal vector_dim=atoi(argv[3]);

            ordinal slice_dim=atoi(argv[4]);

            int starting_index=0;

            if (slice_dim==1) starting_index=1;

            int postfix_dim;

            char **postfix_indices_string;

            Tcl_SplitList(interp,argv[5],&postfix_dim,&postfix_indices_string);

            ordinal postfix_indices[MAXDIM];

            for (i=0;i<postfix_dim;i++) postfix_indices[i]=atoi(postfix_indices_string[i]);

            free(postfix_indices_string);

            ordinal initial_offset=0;

            for (i=0;i<postfix_dim;i++)
initial_offset+=postfix_indices[i]*offset[i+vector_dim+slice_dim];

            int first_offset_argc;

            char **first_offset_argv;

            Tcl_SplitList(interp,argv[7],&first_offset_argc,&first_offset_argv);

            float *previous_offset=(float *)malloc(sizeof(scalar)*first_offset_argc);

            for (i=0;i<first_offset_argc;i++) previous_offset[i]=atof(first_offset_argv[i]);

            free(first_offset_argv);

            int vector_order_argc;

            char **vector_order_argv;

            Tcl_SplitList(interp,argv[6],&vector_order_argc,&vector_order_argv);

            #Info: "Saving vectors for slice %s (delta) in %s" argv[5],argv[1]

            ordinal vector_index;

            for (vector_index=0;vector_index<vector_order_argc;vector_index++) {

                struct dvec *d=plot_cur->pl_dvecs;

                while ((d)&&(strcmp(d->v_name,vector_order_argv[vector_index])))  d = d->v_next;

                if (!(d)) {
```

```
                    #Error: "(%s) listed vector %s was not simulated" argv[0]
vector_order_argv[vector_index]
                    return TCL_ERROR;
                }
                char vector_id[128];
                for (i=2;d->v_name[i]!=')';i++) {
                    if (d->v_name[i]=='_') {
                        vector_id[i-2]=' ';
                    } else {
                        vector_id[i-2]=d->v_name[i];
                    }
                }
                vector_id[i-2]=0;
                int tmp_dim;
                char **vector_index_string;
                Tcl_SplitList(interp,vector_id,&tmp_dim,&vector_index_string);
                if (slice_dim!=tmp_dim) continue;
                ordinal final_offset=initial_offset;
                for (i=0;i<slice_dim;i++) final_offset+=(atoi(vector_index_string[i])-
starting_index)*offset[i+vector_dim];
                free(vector_index_string);
                write_ordinal(O,final_offset);
                write_ordinal(O,d->v_length);
                if (factor_mode) {
                    for (i=0;i<d->v_length;i++) {
                        write_float(O,factor/(d->v_realdata[i]-previous_offset[i]));
                        previous_offset[i]=d->v_realdata[i];
                    }
                } else {
                    for (i=0;i<d->v_length;i++) {
                        write_float(O,(d->v_realdata[i]-previous_offset[i])*factor);
                        previous_offset[i]=d->v_realdata[i];
```

```
            }
        }
    }
    free(previous_offset);
    fclose(O);
    return TCL_OK;
}
static int
save_characterization_slice_delta_differential (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[]) {
    if ((argc!=10)&&(argc!=9)) {
        #Error: "usage: %s <file name> <array sizes> <per vector dim>  <slice dim> <post indices> <vector order list> <first offsets> <basline first_offsets> [<factor>]" argv[0]
        return TCL_ERROR;
    }
    if (save_slice_base==NULL) {
        #Error: "(%s) cannot save a delta+differential slice before a baseline slice was saved" argv[0]
        return TCL_ERROR;
    }
    ordinal i;
    float factor=1;
    int factor_mode=0;
    if (argc==10) {
        if (argv[9][0]=='/') {
            factor=atof(&(argv[9][1]));
            factor_mode=1;
        } else {
            factor=atof(argv[9]);
        }
    }
    FILE *O=fopen(argv[1],"w");
```

```
            int dim;
            char **sizes_string;
            Tcl_SplitList(interp,argv[2],&dim,&sizes_string);
            ordinal size[MAXDIM];
            for (i=0;i<dim;i++) size[i]=atoi(sizes_string[i]);
            free(sizes_string);
            ordinal offset[MAXDIM];
            offset[0]=1;
            for (i=1;i<dim;i++) offset[i]=offset[i-1]*size[i-1];
            ordinal vector_dim=atoi(argv[3]);
            ordinal slice_dim=atoi(argv[4]);
            int starting_index=0;
            if (slice_dim==1) starting_index=1;
            int postfix_dim;
            char **postfix_indices_string;
            Tcl_SplitList(interp,argv[5],&postfix_dim,&postfix_indices_string);
            ordinal postfix_indices[MAXDIM];
            for (i=0;i<postfix_dim;i++) postfix_indices[i]=atoi(postfix_indices_string[i]);
            free(postfix_indices_string);
            ordinal initial_offset=0;
            for (i=0;i<postfix_dim;i++)
initial_offset+=postfix_indices[i]*offset[i+vector_dim+slice_dim];
            int first_offset_argc;
            char **first_offset_argv;
            Tcl_SplitList(interp,argv[7],&first_offset_argc,&first_offset_argv);
            float *previous_offset=(float *)malloc(sizeof(scalar)*first_offset_argc);
            for (i=0;i<first_offset_argc;i++) previous_offset[i]=atof(first_offset_argv[i]);
            int baseline_first_offset_argc;
            char **baseline_first_offset_argv;

Tcl_SplitList(interp,argv[8],&baseline_first_offset_argc,&baseline_first_offset_argv);
```

```
            float *baseline_previous_offset=(float
*)malloc(sizeof(scalar)*baseline_first_offset_argc);

            for (i=0;i<baseline_first_offset_argc;i++)
baseline_previous_offset[i]=atof(baseline_first_offset_argv[i]);

            free(first_offset_argv);

            free(baseline_first_offset_argv);

            int vector_order_argc;

            char **vector_order_argv;

            Tcl_SplitList(interp,argv[6],&vector_order_argc,&vector_order_argv);

            #Info: "Saving vectors for slice %s (delta and differential) in %s" argv[5],argv[1]

            ordinal vector_index;

            for (vector_index=0;vector_index<vector_order_argc;vector_index++) {

                struct dvec *d=plot_cur->pl_dvecs;

                while ((d)&&(strcmp(d->v_name,vector_order_argv[vector_index])))  d = d->v_next;

                if (!(d)) {

                    #Error: "(%s) listed vector %s was not simulated" argv[0]
vector_order_argv[vector_index]

                    return TCL_ERROR;

                }

                struct dvec *b=save_slice_base->pl_dvecs;

                while ((b)&&(strcmp(b->v_name,vector_order_argv[vector_index])))  b = b->v_next;

                if (!(b)) {

                    #Error: "(%s) listed vector %s was not simulated in the saved baseline"
argv[0] vector_order_argv[vector_index]

                    return TCL_ERROR;

                }

                char vector_id[128];

                for (i=2;d->v_name[i]!=')';i++) {

                    if (d->v_name[i]=='_') {

                        vector_id[i-2]=' ';

                    } else {

                        vector_id[i-2]=d->v_name[i];

                    }
```

```
                }
                vector_id[i-2]=0;
                int tmp_dim;
                char **vector_index_string;
                Tcl_SplitList(interp,vector_id,&tmp_dim,&vector_index_string);
                if (slice_dim!=tmp_dim) continue;
                ordinal final_offset=initial_offset;
                for (i=0;i<slice_dim;i++) final_offset+=(atoi(vector_index_string[i])-
starting_index)*offset[i+vector_dim];
                free(vector_index_string);
                write_ordinal(0,final_offset);
                write_ordinal(0,d->v_length);
                if (factor_mode) {
                    for (i=0;i<d->v_length;i++) {
                        #Dinfo: "%s   %g/((%g-%g)-(%g-%g))=%g" d->v_name factor d->v_realdata[i]
previous_offset[i] b->v_realdata[i] baseline_previous_offset[i] factor/((d->v_realdata[i]-
previous_offset[i])-(b->v_realdata[i]-baseline_previous_offset[i]))
                        write_float(0,factor/((d->v_realdata[i]-previous_offset[i])-(b-
>v_realdata[i]-baseline_previous_offset[i])));
                        previous_offset[i]=d->v_realdata[i];
                        baseline_previous_offset[i]=b->v_realdata[i];
                    }
                } else {
                    for (i=0;i<d->v_length;i++) {
                        #Dinfo: "%s   ((%g-%g)-(%g-%g))*%g=%g" d->v_name d->v_realdata[i]
previous_offset[i] b->v_realdata[i] baseline_previous_offset[i] factor factor*((d-
>v_realdata[i]-previous_offset[i])-(b->v_realdata[i]-baseline_previous_offset[i]))
                        write_float(0,((d->v_realdata[i]-previous_offset[i])-(b->v_realdata[i]-
baseline_previous_offset[i]))*factor);
                        previous_offset[i]=d->v_realdata[i];
                        baseline_previous_offset[i]=b->v_realdata[i];
                    }
                }
            }
```

```
        free(previous_offset);
        free(baseline_previous_offset);
        fclose(O);
        return TCL_OK;
    }
    static int
    load_characterization_slice (ClientData clientData,Tcl_Interp *interp,int argc,char
*argv[]) {
        if (argc!=3) {
            #Error: "usage: %s <array name> <file name>" argv[0]
            return TCL_ERROR;
        }
        LUT *a=get_LUT(argv[1]);
        if (!a) {
            #Error: "(%s) array %s must be initialized before slices can be loaded"
argv[0],argv[1]
            return TCL_ERROR;
        }
        open_to_read(argv[2]);
        #Info: "Slice loader from file %s" argv[2]
        while (more_to_read()) {
            ordinal i;
            ordinal offset=read_ordinal();
            ordinal length=read_ordinal();
            for (i=0;i<length; i++) {
                get_float(&(a->content[offset+i]));
                #Dinfo: "%ld+%ld  =  %g" offset i a->content[offset+i]
            }
        }
        done_reading();
        return TCL_OK;
    }
```

B. 5. 2. Process Parellelization

The characterization script is written in Tcl. Its loops are either regular interpreter repetitions or parallelized processes that merge data at the end. A forked process has a cloned version of the memory heap of the original one. Therefore, the main process starts allocating memory fot LUT's and populate them with the saved slices only after slices were generated by forked processes. A forked process is "self aware" that it is forked, in to support special behavior for side-threads, such as modifying message leading token to indicate the pid of its origin. To support parallelism, the following commands were added:

1. Tcl command for fork – provides functionality of the C command fork in a Tcl script.
2. fork_task – process the payload code in a forked process
3. wait_for_forked – pause the main process until all forked processes are finished

```
static int
tcl_fork (ClientData clientData,Tcl_Interp *interp,int argc,char *argv[])
{
    if (argc!=1) {
        #Error: "fork requires no arguments"
        return TCL_ERROR;
    }
    Tcl_ResetResult(interp);
    int PID=getpid();
    int child_pid=fork();
    if (getpid()!=PID) this_process_forked=1;
    tcl_append_int(interp,child_pid);
    return TCL_OK;
}
```

```
proc fork_task {group_var task {limit {}}} {
    upvar $group_var group
    if {$limit=={}} {
        set limit $::fork_limit
    }
    if {![info exists group]} {
        set group {}
    }
    while {1} {
        sleep 10000
        set num_of_active [llength [glob -nocomplain /tmp/forked_processes/[pid]-*-
running]]
        if {$limit>$num_of_active} break
    }
    set parent_pid [pid]
    set p [fork]
    if {$p!=0} {
        if {$p!=$parent_pid} {
            Info: Spawned $p
            lappend group $p
            return
        }
    }
    set ::ParentProcess [open /tmp/${group_var}-${parent_pid}-[pid].tcl w]
    exec touch /tmp/forked_processes/${parent_pid}-[pid]-running
    if {[catch {uplevel $task} msg]} {
        Error: $msg "$::errorInfo"
    }
    close $::ParentProcess
```

```
    Info: Done
    exec touch /tmp/forked_processes/${parent_pid}-[pid]
    file delete /tmp/forked_processes/${parent_pid}-[pid]-running
    exit
}

proc wait_for_forked {forked_processes_var} {
    upvar $forked_processes_var forked_processes
    set wait 1
    while {$wait} {
        sleep 10000
        set wait 0
        set proc_list {}
        set fork_signal_list [glob -nocomplain /tmp/forked_processes/[pid]-*]
        foreach fork_signal $fork_signal_list {
            if {[regexp {\-([0-9]+)$} $fork_signal -> proc_num]} {
                lappend proc_list $proc_num
            }
        }
        foreach process $forked_processes  {
            if {$process==[pid]} continue
            if {[lsearch $proc_list $process]==-1} {
                set wait 1
                break
            }
        }
    }
    Info: All forked processes from [pid] finished for $forked_processes_var
    foreach file [glob -nocomplain /tmp/${forked_processes_var}-[pid]-*.tcl] {
        source $file
        #   file delete $file
```

```
    }
}
```

B. 5. 3. 4D Characterization Script

The 4D version of the characterization script omits the width characterization, leaving it to post factoring of results by W/L. This characterization flavor is adequate for channel W/L<10 and higher ratios obtained by using the M parameter (multiplier). The L sweep is also midified to uniform sampling, which saves on interpolation time by avoiding the L-segment binary search.

```
textbox {
    #######################################################
    #######################################################
    ##                                                   ##
    ##         Starting characterization processes       ##
    ##                                                   ##
```

```
    ########################################################
    ########################################################
}
source $::env(RAMSPICE)/Etc/tests/geo_values.tcl
set ::geo_stepping 20
foreach dim {l w} {
    set values {}
    foreach {key value} [array get ::bin n,*,$dim*] {
        if {[lsearch $values $value]!=-1} continue
        lappend values $value
    }
    set values [lsort -real $values]
    set ${dim}_values {}
    for {set i 0} {$i<[llength $values]-1} {incr i} {
        set this [lindex $values $i]
        set next [lindex $values [expr $i+1]]
        if {$i<[llength $values]-2} {
            for {set j 0} {$j<$::geo_stepping} {incr j} {
                lappend ${dim}_values [expr $this+($next-$this)*$j/$::geo_stepping]
            }
        } else {
            for {set j 0} {$j<=$::geo_stepping} {incr j} {
                lappend ${dim}_values [expr $this+($next-$this)*$j/($::geo_stepping+1)]
            }
        }
    }
}
set ::epsilon 1e-2
foreach type [split $device :] {
    set l_values {}
    set w_values {}
```

```
            set p [string index $type 0]
            set lmin [set ::global_${p}lmin]
            set lmax [set ::global_${p}lmax]
            set wmin [set ::global_${p}wmin]
            set wmax [set ::global_${p}wmax]
            for {set l $lmin} {$l<=$lmax} {set l [expr $l+($lmax-$lmin)/pow(2,$l_rez)]} {
                lappend l_values $l
            }
        Info: l_values=$l_values
            set max_supply $topv
            if {[regexp {^p} $type]} {
                set max_supply [expr -$topv]
            }
            set vt_db_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_vt.db
            set va_db_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_va.db
            set min_vt_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_min_vt.tcl
            set ids_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_ids.db
            set gm_file $::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_gm.db
            set ro_file $::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_ro.db
            set vth_mis_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_vth_mis.db
            set ids_mis_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_ids_mis.db
            if {[file exists $vt_db_file]&&[file exists $min_vt_file]&&[file exists
$va_db_file]} {
                source $min_vt_file
        Info: LUT exists for type=$type $vt_db_file $min_vt_file $va_db_file
            } else {
                textbox    "Characterizing Vt and Va for $type"
                constrain "
```

```
Vgs  0          $max_supply          $vgs_rez
Vds  0          $max_supply          $vds_rez
Vbs  [expr -$max_supply/2]       0            $vbs_rez
L    [set ::global_${p}lmin] [set ::global_${p}lmax] $::l_rez
"
set ::fork_limit 6
set ::mid_vgs [expr ($topv+0.5)/2]
set ::min_Ids 1e-12
set ::scalar_Ids_multiplier 3.9e-11
set ::limit_Ids_multiplier 1e-1
set ::minVgs [expr 0.5*$max_supply]
set ::minVds [expr 0.1*$max_supply]
set ::maxVds $max_supply
set ::maxVgs [expr 0.8*$max_supply]
set ::minVt  $max_supply
################################
set ::low_vgs [expr $max_supply-$::epsilon]
set ::high_vgs $max_supply
# Clean droppings
foreach droppings [glob -nocomplain /tmp/*characterization_task*] {
    file delete $droppings
}
foreach droppings [glob -nocomplain /tmp/{ids,gm,ro}.*] {
    file delete $droppings
}

set short_views {Vt Va}
set views {Ids gm ro}
foreach ::corner $::corner_list {
    set ::temp $::corner_to_temp($::corner)
    fork_task char_vt_task {
```

```
    set ::minVt $max_supply
    textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
    ######### Template netlist
    netlist ".include $::env(RAMSPICE)/Etc/Tech_DB/${tech}/${tech}.sp"
    netlist {
        ** the N-transistor

    * name D G S B model L W
}
set i3 1
foreach L $l_values {
    set rtest 1e-12
    set W $L
    if {$W<$wmin} {
        set W [expr $L*$wmin/$lmin]
        set rtest [expr 1e-12*$lmin/$wmin]
    }
    netlist ".temp $::temp"
    # mosfet {name type D G S B L W Lmin}
    mosfet mn_${i3} $type D G ${i3} B $L $W
    netlist "r_${i3} ${i3} 0 $rtest"
    incr i3
}
netlist "
    Vds D 0 dc $max_supply ac 0
    Vgs G 0 dc $max_supply ac 0
    Vbs B 0 dc 0 ac 0
    .end
    "
update_netlist Vt $::corner $::temp
```

```
            ######### Initialize database
Info: Measuring Vt(W,L) $::low_vgs $::high_vgs
            ::spice::dc vgs [expr $max_supply/2-$epsilon] [expr $max_supply/2] $epsilon
            set i2 1
            set i3 0
            foreach L $l_values {
                set Ids_low  [get_spice_data V(${i2}) 0]
                set Ids_high [get_spice_data V(${i2}) 1]
                if {[catch {set slope [expr ($Ids_high-$Ids_low)/$epsilon]} msg]} {
Info: Ids_high=$Ids_high Ids_low=$Ids_low epsilon=$epsilon
Error: $msg
                    exit
                }
                set Vt [expr $max_supply/2-$Ids_high/$slope]
                if {[regexp {^p} $type]} {
                    set Vt [expr -$Vt]
                }
                ^ @ look_up_tables/$type/Vt/${::corner}($i3) = $Vt
                if {$i3>3} {
                    ^ if "abs($Vt)<abs(\$::minVt)" "set ::minVt $Vt"
                }
                incr i2
                incr i3
            }

Info: Measuring Va(W,L)
            ::spice::dc vds [expr $max_supply-$epsilon] $max_supply $epsilon
            set i2 1
            set i3 0
            foreach L $l_values {
                set Ids_low  [get_spice_data V(${i2}) 0]
```

```
                set Ids_high [get_spice_data V(${i2}) 1]
                if {[catch {set slope [expr ($Ids_high-$Ids_low)/$epsilon]} msg]} {
Info: Ids_high=$Ids_high Ids_low=$Ids_low epsilon=$epsilon
Error: $msg
                    exit
                }
                set Va [expr $max_supply-$Ids_high/$slope]
                ^ @ look_up_tables/$type/Va/${::corner}($i3) = $Va
                incr i3
                incr i2
            }
        }
    }
    foreach ::corner $::corner_list {
        foreach array $short_views {
            @ /look_up_tables/$type/$array/${::corner}([llength $l_values]) !
        }
        foreach array $short_views {
            set i3 0
            foreach L $l_values {
                LUT_set_legend /look_up_tables/$type/$array/${::corner} 0 $i3 $L
                incr i3
            }
        }
    }
    wait_for_forked char_vt_task
Info: Saving Arrays
    @ /look_up_tables/$type/Vt save $vt_db_file
    @ /look_up_tables/$type/Va save $va_db_file
    set minVt 0
Info: minVt=$minVt
```

```
    set MVT [open $min_vt_file w]
    puts $MVT [list set minVt [set minVt]]
    close $MVT
    # Cleanup
    foreach char_file [glob -nocomplain /tmp/char_vt_task*] {
        file delete $char_file
    }
}
if {![file exists $ids_file]||![file exists $gm_file]||![file exists $ro_file]} {
    if {[regexp {^p} $type]} {
        set minVt [expr -$minVt]
    }
    textbox    "Characterizing Ids, gm and ro for $type"
    constrain "
        Vgs  $minVt         $max_supply           $::vgs_rez
        Vds  0              $max_supply           $::vds_rez
        Vbs  [expr -$max_supply/2]          0     $::vbs_rez
        L    [set ::global_${p}lmin] [set ::global_${p}lmax] $::l_rez
        "
    set ::fork_limit 8
    set ::mid_vgs 0.8
    set ::min_Ids 1e-12
    ################################
    # Clean droppings
    foreach droppings [glob -nocomplain /tmp/*characterization_task*] {
            file delete $droppings
        }
        foreach droppings [glob -nocomplain /tmp/{Ids,gm,ro}.*] {
            file delete $droppings
        }
        set views {Ids gm ro}
```

```
set ohmic_factor 1e12
    foreach ::corner $::corner_list {
        set ::temp $::corner_to_temp($::corner)
        fork_task char_vig_task {

            Info: "Corner [string toupper $::corner], Temperature=$::temp degC"
            netlist ".include $::env(RAMSPICE)/Etc/Tech_DB/${tech}/${tech}.sp"
            netlist {
                ** the N-transistor
                * name D G S B model L W
            }
            set i3 1
            set i4 0
            foreach L $l_values {
        set rtest 1e-12
        set W $L
        if {$W<$wmin} {
            set W [expr $L*$wmin/$lmin]
        set rtest [expr 1e-12*$lmin/$wmin]
        }
                netlist ".temp $::temp"
                # mosfet {name type D G S B L W Lmin}
                mosfet mn_${i3} $type D G ${i3} B $L $W
                netlist "r_${i3} ${i3} 0 $rtest"
                incr i3
            }
            netlist "Vds D 0 dc 0 ac 0"
            netlist "Vgs G 0 dc 0 ac 0"
            netlist "Vbs B 0 dc 0 ac 0"
            netlist ".end"
            update_netlist VIG $::corner $::temp
```

```
                        set index_range {}
                        foreach var {Vgs Vds Vbs} {
                            lappend index_range $::constraints($var,index_range)
                        }
                        lappend index_range [llength $l_values]
                        Info:  Measuring Ids(Vgs,Vds,Vbs) gm(Vgs,Vds,Vbs) and ro(Vgs,Vds,Vbs)
                        Info:  Vgs ($::constraints(Vgs,minval),$::constraints(Vgs,maxval))
step=$::constraints(Vgs,step)
                        Info:  Vds ($::constraints(Vds,minval),$::constraints(Vds,maxval))
step=$::constraints(Vds,step)
                        Info:  Vbs ($::constraints(Vbs,minval),$::constraints(Vbs,maxval))
step=$::constraints(Vbs,step)
                        Info: simulation started ([clock format [clock seconds]])
                        ::spice::dc vgs $::constraints(Vgs,minval) $::constraints(Vgs,maxval)
$::constraints(Vgs,step) vds $::constraints(Vds,minval) $::constraints(Vds,maxval)
$::constraints(Vds,step) vbs $::constraints(Vbs,minval) $::constraints(Vbs,maxval)
$::constraints(Vbs,step)
                        Info: done Ids running. Saving results. ([clock format [clock seconds]])
                        save_characterization_slice /tmp/Ids.$::corner $index_range 3 1 0
$ohmic_factor
                        ::spice::dc vgs [expr $::constraints(Vgs,minval)+$::epsilon] [expr
$::constraints(Vgs,maxval)+$::epsilon] $::constraints(Vgs,step) vds $::constraints(Vds,minval)
$::constraints(Vds,maxval) $::constraints(Vds,step) vbs $::constraints(Vbs,minval)
$::constraints(Vbs,maxval) $::constraints(Vbs,step)
                        Info: done gm running. Saving results. ([clock format [clock seconds]])
                        save_characterization_slice_differential /tmp/gm.$::corner $index_range
3 1 0 [expr $ohmic_factor/$::epsilon]
                        ::spice::dc vgs $::constraints(Vgs,minval) $::constraints(Vgs,maxval)
$::constraints(Vgs,step) vds [expr $::constraints(Vds,minval)+$::epsilon] [expr
$::constraints(Vds,maxval)+$::epsilon] $::constraints(Vds,step) vbs $::constraints(Vbs,minval)
$::constraints(Vbs,maxval) $::constraints(Vbs,step)
                        Info: done ro running. Saving results. ([clock format [clock seconds]])
                        save_characterization_slice_differential /tmp/ro.$::corner $index_range
3 1 0 /[expr $::epsilon/$ohmic_factor]
                        Info: is done saving results. ([clock format [clock seconds]])
                        Info: Done ([clock format [clock seconds]])
                    }
                }
```

```
foreach ::corner $::corner_list {
    set ::temp $::corner_to_temp($::corner)
    ######### Characterizing loops
    set index_range {}
    foreach var {Vgs Vds Vbs} {
        lappend index_range $::constraints($var,index_range)
    }
    lappend index_range [llength $l_values]
    foreach array $views {
        @ look_up_tables/$type/$array/${::corner}([join $index_range ,]) !
    }
    foreach array $views {
        foreach_in_range Vgs i0 {
            LUT_set_legend /look_up_tables/$type/$array/$::corner 0 $i0 $Vgs
        }
        foreach_in_range Vds i1 {
            LUT_set_legend /look_up_tables/$type/$array/$::corner 1 $i1 $Vds
        }
        foreach_in_range Vbs i2 {
            LUT_set_legend /look_up_tables/$type/$array/$::corner 2 $i2 $Vbs
        }
        set i3 0
        foreach L $l_values {
            LUT_set_legend /look_up_tables/$type/$array/$::corner 3 $i3 $L
            incr i3
        }
    }
}
wait_for_forked char_vig_task
Info: Loading Saved Slices
foreach ::corner $::corner_list {
```

```
                    load_characterization_slice /look_up_tables/$type/Ids/$::corner
/tmp/Ids.$::corner
                    load_characterization_slice /look_up_tables/$type/gm/$::corner
/tmp/gm.$::corner
                    load_characterization_slice /look_up_tables/$type/ro/$::corner
/tmp/ro.$::corner
                }
                Info: Saving Array
                @ /look_up_tables/$type/Ids save $ids_file
                @ /look_up_tables/$type/gm save $gm_file
                @ /look_up_tables/$type/ro save $ro_file
                # Cleanup
                foreach char_file [glob -nocomplain /tmp/char_vig_task*] {
                    file delete $char_file
                }
#       Info: Ids=[expr [@ /look_up_tables/$type/Ids/ss calc 1.8 1.8 0 180e-9]*220/180]
            }
            if {![file exists $vth_mis_file]||![file exists $ids_mis_file]} {
                textbox    "Characterizing Ids and Vt mismatch for $type"
                set ::fork_limit 6
                @ /struct/dist/type = string
                @ /struct/dist/arg1 = string
                @ /struct/dist/arg2 = string
                set scaling 12
                for {set i 1} {[info exists bin(n,$i,lmin)]} {incr i} {
                    @ /simulation_config/mc/nch_mc_$i = string {
                        set lef [expr $BSIM_l - 20e-9]
                        set wef $BSIM_w
                        set toxn 4.08e-9
                        set geo_fac [expr 1/sqrt($lef*$wef*1e12)]
                        set vthmis [expr $scaling*3.635e-3*$geo_fac]
                        set dlmis [expr $scaling*4.58e-3*$geo_fac*$lef]
```

```
            set dwmis [expr $scaling*3.73e-3*$geo_fac*$wef]
            set toxmis [expr $scaling*1.01e-3*$geo_fac*$geo_fac*$toxn]
        }
        @ /simulation_config/mc/nch_mc_$i/l = /struct/dist gauss * {{$dlmis}}
        @ /simulation_config/mc/nch_mc_$i/w = /struct/dist gauss * {{$dwmis}}
        @ /simulation_config/mc/nch_mc_$i/vth0 = /struct/dist gauss * {{$vthmis}}
        @ /simulation_config/mc/nch_mc_$i/tox = /struct/dist gauss * {{$toxmis}}
    }
    # Clean droppings
    foreach droppings [glob -nocomplain /tmp/*characterization_task*] {
        file delete $droppings
    }
    foreach droppings [glob -nocomplain /tmp/char_{mis,size,ids,gm,ro}*.*] {
        file delete $droppings
    }
    set views {Ids_mis Vth_mis}
    foreach ::corner $::corner_list {
        set ::temp $::corner_to_temp($::corner)
        fork_task char_mis_task {
            textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
            ######### Template netlist
            netlist ".include $::env(RAMSPICE)/Etc/Tech_DB/${tech}/${tech}.sp"
            netlist {
                ** the N-transistor
                * name D G S B model L W
            }
            netlist ".temp $::temp"
            set vars_of_interest {}
            set i3 1
            foreach L $l_values {
        set rtest 1e-12
```

```
set W $L
if {$W<$wmin} {
    set W [expr $L*$wmin/$lmin]
set rtest [expr 1e-12*$lmin/$wmin]
}
    # mosfet {name type D G S B L W Lmin}
        mosfet mn_${i3} $type D G ${i3} 0 $L $W
        netlist "r_${i3} ${i3} 0 $rtest"
        lappend vars_of_interest mn_${i3}
        lappend vars_of_interest Vth
        lappend vars_of_interest mn_${i3}
        lappend vars_of_interest Ids
        incr i3
    }
    netlist "
    Vds D 0 dc $max_supply ac 0
    Vgs G 0 dc [expr $max_supply/2] ac 0
    .end
    "
    set O [open ~/temp/tempMis$::corner_$::temp.sn w]
    puts $O $template_netlist
    close $O
    update_netlist Mis $::corner $::temp


    Info:  Measuring mis(W,L)
    ######### Characterizing loops
    Info: simulation started ([clock format [clock seconds]])
    set result [monte_carlo_${::bsim_version} 200 /simulation_config/mc
$vars_of_interest ::spice::op]
                Info: result=[join $result \n]
    Info: done Mismatch running. Saving results. ([clock format [clock
seconds]])
```

```
                    Info: Done ([clock format [clock seconds]])
                    set i 0
                    set i3 0
                    foreach L $l_values {
                        set Vth [lindex $result $i]
                        incr i
                        set Sigma [lindex $result $i]
                        ^ @ /look_up_tables/$type/Vth_mis/${::corner}($i3) = [expr
$Sigma/$Vth]
                        incr i
                        set Ids [lindex $result $i]
                        incr i
                        set Sigma [lindex $result $i]
                        ^ @ /look_up_tables/$type/Ids_mis/${::corner}($i3) = [expr
$Sigma/$Ids]
                        incr i
                        incr i3
                    }
                }
            }
            foreach ::corner $::corner_list {
                set ::temp $::corner_to_temp($::corner)
                ######### Initialize database
                foreach array $views {
                    Info: New Array: look_up_tables/$type/$array/${::corner}([llength
$l_values])
                    @ look_up_tables/$type/$array/${::corner}([llength $l_values]) !
                }
                foreach array $views {
                    set i3 0
                    foreach L $l_values {
                        LUT_set_legend /look_up_tables/$type/$array/$::corner 0 $i3 $L
```

```
                    incr i3
                }
            }
        }
        wait_for_forked char_mis_task
        foreach ::corner $::corner_list {
            set ::temp $::corner_to_temp($::corner)
            foreach array $views {
                generate_lut  /look_up_tables/$type/$array/$::corner
            }
        }
        @ /look_up_tables/$type/Vth_mis save $vth_mis_file
        @ /look_up_tables/$type/Ids_mis save $ids_mis_file
        # Cleanup
        foreach char_file [glob -nocomplain /tmp/char_mis_task*] {
            file delete $char_file
        }
    }
    set noise_complete 1
    foreach ::corner $::corner_list {
        set thermal_noise_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_thermal_noise.db
        set flicker_noise_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_flicker_noise.db
    set cgs_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_cgs.db
    set cds_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_cds.db
        if {[file exists $thermal_noise_file]} continue
        set noise_complete 0
        break
    }
    if {!$noise_complete} {
```

```
            #       textbox    "Characterizing Noise for $type Vgs=($minVt,$max_supply) 
Vds=(0,$max_supply)"
            textbox    "Characterizing Noise for $type "
    #        constrain "
    #        Vgs  $minVt           $max_supply     7
    #        Vds  [expr $max_supply/20] $max_supply     4
    #        Vbs  [expr -$max_supply/3] 0           1
    #        "
            constrain "
            Vgs  $minVt          $max_supply     2
            Vds  [expr $max_supply/20]  $max_supply     2
            Vbs  [expr -$max_supply/3]  0           1
            "
            set ::fork_limit 1
            foreach old_task_file [glob -nocomplain /tmp/*.tcl] {
                file delete $old_task_file
            }
            set views {cgs cds flicker_const thermal_noise}
            set total_array_volume 1
            proc noise_cont {coord} {
                @ look_up_tables/$type/flicker_const/${::corner}($coord) = 0
                return -code continue
            }
            set index_range {}
            foreach var {Vgs Vds Vbs} {
                lappend index_range $::constraints($var,index_range)
            }
            lappend index_range [llength $l_values]
            set_spice_var Captured_Quick_Noise 1
            foreach ::corner $::corner_list {
                set thermal_noise_file 
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_thermal_noise.db
```

```
                    set flicker_noise_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_flicker_noise.db

                    set cgs_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_cgs.db

                    set cds_file
$::env(RAMSPICE)/Etc/Tech_DB/${tech}/4d/${::rez}/${tech}_${type}_${::corner}_cds.db

                    if {[file exists $thermal_noise_file]} continue

                    set ::temp $::corner_to_temp($::corner)

                    foreach array $views {

                        @ look_up_tables/$type/$array/${::corner}([join $index_range ,]) !

                    }

                    foreach array $views {

                        foreach_in_range Vgs i0 {

                            LUT_set_legend /look_up_tables/$type/$array/$::corner 0 $i0 $Vgs

                    Info: 0/$i0 $Vgs=[LUT_get_legend /look_up_tables/$type/$array/$::corner 0
$i0]

                        }

                        foreach_in_range Vds i1 {

                            LUT_set_legend /look_up_tables/$type/$array/$::corner 1 $i1 $Vds

                    Info: 1/$i1 $Vds=[LUT_get_legend /look_up_tables/$type/$array/$::corner 1
$i1]

                        }

                        foreach_in_range Vbs i2 {

                            LUT_set_legend /look_up_tables/$type/$array/$::corner 2 $i2 $Vbs

                    Info: 2/$i2 $Vbs=[LUT_get_legend /look_up_tables/$type/$array/$::corner 2
$i2]

                        }

                        set i3 0

                        foreach L $l_values {

                            LUT_set_legend /look_up_tables/$type/$array/$::corner 3 $i3 $L

                    Info: 3/$i3 $L=[LUT_get_legend /look_up_tables/$type/$array/$::corner 3 $i3]

                            incr i3

                        }

                    }
```

```
textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
for {set section 1} {[info exists bin(n,$section,lmin)]} {incr section} {}
ETA $section
for {set section 1} {[info exists bin(n,$section,lmin)]} {incr section} {
    Info: Analyzing bin $section [clock format [clock seconds]]
    set dummy_flicker 4.3e-12
    set dummy_thermal 5e-11
    set dummy_count_flicker 1
    set dummy_count_thermal 1
    netlist ".include $::env(RAMSPICE)/Etc/Tech_DB/${tech}/${tech}.sp"
    netlist ".temp $temp"
    netlist {
        ** the N-transistor
        * name D G S B model L W
        rtest 3 0 1e-12
        Vgs G 0 dc 0 ac 0
        Vds D 0 dc 0 ac 0
        Vbs B 0 dc 0 ac 0
    }
    # mosfet {name type D G S B L W Lmin}
    mosfet mn_0_0 $type D G 3 B $bin(n,$section,lmin) $bin(n,$section,wmin)
$section
    netlist {
        .end
    }
    update_netlist Noise $::corner $::temp
    set i3 0
    foreach L $l_values {
set W [expr $L*$wmin/$lmin]
        set prefix [string index $type 0]
        if {$section!=[find_mosfet_bin $prefix $L $W]} {
```

```
                        incr i3
                        continue
                    }
                    ::spice::alter mn_0_0 w = $W
                    ::spice::alter mn_0_0 l = $L
                    foreach_in_range Vgs i0 {
                        ::spice::alter vgs = $Vgs
                        foreach_in_range Vds i1 {
                            ::spice::alter vds = $Vds
                            foreach_in_range Vbs i2 {
                                ::spice::alter vbs = $Vbs
                                ::spice::noise v(3) vgs lin 2 1 2
                @ look_up_tables/$type/cgs/${::corner}($i0,$i1,$i2,$i3) =
[get_spice_data Captured_Cgs end]
                @ look_up_tables/$type/cds/${::corner}($i0,$i1,$i2,$i3) =
[get_spice_data Captured_Cgd end]
                                set thermal_noise [get_spice_data
Captured_Thermal_Noise end]
                                if {[string match *nan* $thermal_noise]} {
                                    set thermal_noise 0
                                }
                                set Ssi [get_spice_data Captured_Ssi end]
                                set Captured_EffFreq [get_spice_data Captured_EffFreq
end]
                                if {[string match *nan* $Captured_EffFreq]} {
                                    noise_cont $i0,$i1,$i2,$i3
                                }
                                set flicker_noise [expr $Ssi*$Captured_EffFreq]
                                if {![regexp {^[0-9\.\-\+e]+$} $flicker_noise]} {
                                    Info: flicker_noise=$flicker_noise
                                    set flicker_noise 0
                                }
                                ::spice::op
```

```
set Ids1 [get_spice_data V(3) 0]
::spice::alter vgs = [expr $Vgs+$::epsilon]
::spice::op
set Ids2 [get_spice_data V(3) 0]
set gm [expr ($Ids2-$Ids1)*1e12/$::epsilon]
if {$gm==0} {
    @
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3) =  [expr
$dummy_flicker/$dummy_count_flicker]
    @
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3) = [expr
$dummy_thermal/$dummy_count_thermal]
} else {
    if {[catch {@
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3) =  [expr
sqrt($flicker_noise*$W*$L/($gm*$gm))]}]} {
        @
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3) = [expr
$dummy_flicker/$dummy_count_flicker]
    } else {
        set dummy_flicker [expr
$dummy_flicker+sqrt($flicker_noise*$W*$L/($gm*$gm))]
        incr dummy_count_flicker
    }
    if {[catch {@
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3) = [expr
sqrt($thermal_noise/$gm)]}]} {
        @
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3) = [expr
$dummy_thermal/$dummy_count_thermal]
    } else {
        set dummy_thermal [expr
$dummy_thermal+sqrt($thermal_noise/$gm)]
        incr dummy_count_thermal
    }
}
#report_vars Vgs Vds Vbs W L
```

```
                                        #Info: thermal_noise=[@
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3)]

                                        #Info: flicker_noise=[@
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3)]

                                        ::spice::destroy all

                                        ::spice::alter vgs = $Vgs

                                    }

                                    ETA

                                }

                            }

                            incr i3

                        }

                    }

                    @ /look_up_tables/$type/cds/$::corner save ${cds_file}

                    @ /look_up_tables/$type/cgs/$::corner save ${cgs_file}

                    @ /look_up_tables/$type/thermal_noise/$::corner save ${thermal_noise_file}

                    @ /look_up_tables/$type/flicker_const/$::corner save ${flicker_noise_file}

                }

            }

        }

        exit
```

B. 5. 4. 5D Characterization Script

The 5D version of the characterization script DC-sweeps on operating point voltages and tracks L and W via multi-transistor circuits.

```
textbox {
    #######################################################
    #######################################################
    ##                                                   ##
    ##         Starting characterization processes       ##
    ##                                                   ##
    #######################################################
    #######################################################
}
source $::env(RAMSPICE)/tests/geo_values.tcl
set ::epsilon 1e-2
foreach type [split $device :] {
    set max_supply $topv
    if {[regexp {^p} $type]} {
        set max_supply [expr -$topv]
    }
    set vt_db_file $::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_vt.db
    set va_db_file $::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_va.db
    set min_vt_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_min_vt.tcl
    set ids_file $::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_ids.db
    set gm_file $::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_gm.db
    set go_file $::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_go.db
    set vth_mis_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_vth_mis.db
    set ids_mis_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_ids_mis.db
```

```
            if {[file exists $vt_db_file]&&[file exists $min_vt_file]&&[file exists
$va_db_file]} {

                source $min_vt_file

            } else {

                textbox    "Characterizing Vt and Va for $type"

                constrain "

                Vgs  0          $max_supply          $vgs_rez

                Vds  0          $max_supply          $vds_rez

                Vbs  [expr -$max_supply/2]       0            $vbs_rez

                "

                set ::fork_limit 6

                set ::mid_vgs [expr ($topv+0.5)/2]

                set ::min_Ids 1e-12

                set ::scalar_Ids_multiplier 3.9e-11

                set ::limit_Ids_multiplier 1e-1

                set ::minVgs [expr 0.5*$max_supply]

                set ::minVds [expr 0.1*$max_supply]

                set ::maxVds $max_supply

                set ::maxVgs [expr 0.8*$max_supply]

                set ::minVt  $max_supply

                ################################

                set ::low_vgs [expr $max_supply-$::epsilon]

                set ::high_vgs $max_supply

                # Clean droppings

                foreach droppings [glob -nocomplain /tmp/*characterization_task*] {

                    file delete $droppings

                }

                foreach droppings [glob -nocomplain /tmp/{ids,gm,go}.*] {

                    file delete $droppings

                }
```

```
set short_views {Vt Va}
set views {Ids gm go}
foreach ::corner $::corner_list {
    set ::temp $::corner_to_temp($::corner)
    fork_task char_vt_task {
        set ::minVt $max_supply
        textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
        ######### Template netlist
        netlist ".include $::env(RAMSPICE)/Tech_DB/${tech}/${tech}.sp"
        netlist {
            ** the N-transistor
            * name D G S B model L W
        }
        set i3 0
        foreach L $l_values {
            set i4 0
            foreach W $w_values {
                netlist ".temp $::temp"
                # mosfet {name type D G S B L W Lmin}
                mosfet mn_${i3}_${i4} $type 3 1 ${i3}_${i4} 4 $L $W
                netlist "r_${i3}_${i4} ${i3}_${i4} 0 1e-12"
                incr i4
            }
            incr i3
        }
        netlist "
        Vds 3 0 dc $max_supply ac 0
        Vgs 1 0 dc $max_supply ac 0
        Vbs 4 0 dc 0 ac 0
        .end
        "
```

```
update_netlist

######### Initialize database
Info: Measuring Vt(W,L) $::low_vgs $::high_vgs
::spice::dc vgs [expr $max_supply/2-$epsilon] [expr $max_supply/2]
$epsilon
set i2 0
foreach L $l_values {
    set i3 0
    foreach W $w_values {
        set Ids_low  [::spice::get_spice_data V(${i2}_${i3}) 0]
        set Ids_high [::spice::get_spice_data V(${i2}_${i3}) 1]
set slope [expr ($Ids_high-$Ids_low)/$epsilon]
        set Vt [expr $max_supply/2-$Ids_high/$slope]
        if {[regexp {^p} $type]} {
            set Vt [expr -$Vt]
        }
        report_vars corner temp Vt L W
        ^ @ look_up_tables/$type/Vt/${::corner}($i2,$i3) = $Vt
if {$W/$L<=10.0} {
            ^ if "abs($Vt)<abs(\$::minVt)" "set ::minVt $Vt"
}
        incr i3
    }
    incr i2
}

Info: Measuring Va(W,L)
::spice::dc vds [expr $max_supply-$epsilon] $max_supply $epsilon
set i2 0
foreach L $l_values {
```

```
                set i3 0
                foreach W $w_values {
                    set Ids_low  [::spice::get_spice_data V(${i2}_${i3}) 0]
                    set Ids_high [::spice::get_spice_data V(${i2}_${i3}) 1]
        set slope [expr ($Ids_high-$Ids_low)/$epsilon]
                    set Va [expr $max_supply-$Ids_high/$slope]
                    report_vars corner temp Va L W
                    ^ @ look_up_tables/$type/Va/${::corner}($i2,$i3) = $Va
                    incr i3
                }
                incr i2
            }
        }
    }
    foreach ::corner $::corner_list {
        set ::temp $::corner_to_temp($::corner)
        set index_range {}
        foreach var {l w} {
            lappend index_range [llength [set ${var}_values]]
        }
        foreach array $short_views {
            @ /look_up_tables/$type/$array/${::corner}([join $index_range ,]) !
        }
        foreach array $short_views {
            set i3 0
            foreach L $l_values {
                ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 0
$i3 $L
                incr i3
            }
            set i4 0
```

```
                foreach W $w_values {
                    ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 1
$i4 $W
                    incr i4
                }
            }
        }
        wait_for_forked char_vt_task
        @ /look_up_tables/$type/Vt save $vt_db_file
        @ /look_up_tables/$type/Va save $va_db_file

        Info: minVt=$minVt
        set MVT [open $min_vt_file w]
        puts $MVT [list set minVt [set minVt]]
        close $MVT
        # Cleanup
        foreach char_file [glob -nocomplain /tmp/char_vt_task*] {
            file delete $char_file
        }
    }
    if {![file exists $ids_file]||![file exists $gm_file]||![file exists $go_file]} {
        textbox    "Characterizing Ids, gm and go for $type"
        constrain "
        Vgs  $minVt         $max_supply            $::vgs_rez
        Vds  0              $max_supply            $::vds_rez
        Vbs  [expr -$max_supply/3]          0      $::vbs_rez
        "
        set ::fork_limit 8
        set ::mid_vgs 0.8
        set ::min_Ids 1e-12
        ################################
```

```
# Clean droppings
foreach droppings [glob -nocomplain /tmp/*characterization_task*] {
    file delete $droppings
}
foreach droppings [glob -nocomplain /tmp/{Ids,gm,go}.*] {
    file delete $droppings
}

set views {Ids gm go}
foreach ::corner $::corner_list {
    set ::temp $::corner_to_temp($::corner)
    fork_task char_vig_task {

        Info: "Corner [string toupper $::corner], Temperature=$::temp degC"
        netlist ".include $::env(RAMSPICE)/Tech_DB/${tech}/${tech}.sp"
        netlist {
            ** the N-transistor
            * name D G S B model L W
        }
        set i3 0
        foreach L $l_values {
            set i4 0
            foreach W $w_values {
                netlist ".temp $::temp"
                # mosfet {name type D G S B L W Lmin}
                mosfet mn_${i3}_${i4} $type 3 1 ${i3}_${i4} 4 $L $W
                netlist "r_${i3}_${i4} ${i3}_${i4} 0 1e-12"
                incr i4
            }
            incr i3
        }
```

```
netlist "Vds 3 0 dc 0 ac 0"
netlist "Vgs 1 0 dc 0 ac 0"
netlist "Vbs 4 0 dc 0 ac 0"
netlist ".end"
update_netlist
set index_range {}
foreach var {Vgs Vds Vbs} {
    lappend index_range $::constraints($var,index_range)
}
lappend index_range [llength $l_values]
lappend index_range [llength $w_values]
Info:  Measuring Ids(Vgs,Vds,Vbs) gm(Vgs,Vds,Vbs) and go(Vgs,Vds,Vbs)
Info:  Vgs ($::constraints(Vgs,minval),$::constraints(Vgs,maxval)) step=$::constraints(Vgs,step)
Info:  Vds ($::constraints(Vds,minval),$::constraints(Vds,maxval)) step=$::constraints(Vds,step)
Info:  Vbs ($::constraints(Vbs,minval),$::constraints(Vbs,maxval)) step=$::constraints(Vbs,step)
Info: simulation started ([clock format [clock seconds]])
::spice::dc vgs $::constraints(Vgs,minval) $::constraints(Vgs,maxval) $::constraints(Vgs,step) vds $::constraints(Vds,minval) $::constraints(Vds,maxval) $::constraints(Vds,step) vbs $::constraints(Vbs,minval) $::constraints(Vbs,maxval) $::constraints(Vbs,step)
Info: done Ids running. Saving results. ([clock format [clock seconds]])
save_characterization_slice /tmp/Ids.$::corner $index_range 3 2 0 1e12
::spice::dc vgs [expr $::constraints(Vgs,minval)+$::epsilon] [expr $::constraints(Vgs,maxval)+$::epsilon] $::constraints(Vgs,step) vds $::constraints(Vds,minval) $::constraints(Vds,maxval) $::constraints(Vds,step) vbs $::constraints(Vbs,minval) $::constraints(Vbs,maxval) $::constraints(Vbs,step)
Info: done gm running. Saving results. ([clock format [clock seconds]])
save_characterization_slice_differential /tmp/gm.$::corner $index_range 3 2 0 [expr 1e12/$::epsilon]
::spice::dc vgs $::constraints(Vgs,minval) $::constraints(Vgs,maxval) $::constraints(Vgs,step) vds [expr $::constraints(Vds,minval)+$::epsilon] [expr $::constraints(Vds,maxval)+$::epsilon] $::constraints(Vds,step) vbs $::constraints(Vbs,minval) $::constraints(Vbs,maxval) $::constraints(Vbs,step)
Info: done go running. Saving results. ([clock format [clock seconds]])
```

```
                        save_characterization_slice_differential /tmp/go.$::corner $index_range
3 2 0 [expr 1e12/$::epsilon]

                        Info: is done saving results. ([clock format [clock seconds]])

                        Info: Done ([clock format [clock seconds]])

                    }

                }

                foreach ::corner $::corner_list {

                    set ::temp $::corner_to_temp($::corner)

                    ######### Characterizing loops

                    set index_range {}

                    foreach var {Vgs Vds Vbs} {

                        lappend index_range $::constraints($var,index_range)

                    }

                    lappend index_range [llength $l_values]

                    lappend index_range [llength $w_values]

                    foreach array $views {

                        @ look_up_tables/$type/$array/${::corner}([join $index_range ,]) !

                    }

                    foreach array $views {

                        foreach_in_range Vgs i0 {

                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 0
$i0 $Vgs

                        }

                        foreach_in_range Vds i1 {

                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 1
$i1 $Vds

                        }

                        foreach_in_range Vbs i2 {

                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 2
$i2 $Vbs

                        }

                        set i3 0

                        foreach L $l_values {
```

```
                        ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 3
$i3 $L
                        incr i3
                    }
                    set i4 0
                    foreach W $w_values {
                        ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 4
$i4 $W
                        incr i4
                    }
                }
            }
            wait_for_forked char_vig_task

            foreach ::corner $::corner_list {
                load_characterization_slice /look_up_tables/$type/Ids/$::corner
/tmp/Ids.$::corner
                load_characterization_slice /look_up_tables/$type/gm/$::corner
/tmp/gm.$::corner
                load_characterization_slice /look_up_tables/$type/go/$::corner
/tmp/go.$::corner
                normalize_ids /look_up_tables/$type/Ids/$::corner
                normalize_ids /look_up_tables/$type/gm/$::corner
                normalize_ids /look_up_tables/$type/go/$::corner
            }

            @ /look_up_tables/$type/Ids save $ids_file
            @ /look_up_tables/$type/gm save $gm_file
            @ /look_up_tables/$type/go save $go_file
            # Cleanup
            foreach char_file [glob -nocomplain /tmp/char_vig_task*] {
                file delete $char_file
            }
```

```
}
if {![file exists $vth_mis_file]||![file exists $ids_mis_file]} {
    textbox    "Characterizing Ids and Vt mismatch for $type"
    set ::fork_limit 6
    @ /struct/dist/type = string
    @ /struct/dist/arg1 = string
    @ /struct/dist/arg2 = string
    set scaling 12
    for {set i 1} {[info exists bin(n,$i,lmin)]} {incr i} {
        @ /simulation_config/mc/nch_mc_$i = string {
            set lef [expr $BSIM_l - 20e-9]
            set wef $BSIM_w
            set toxn 4.08e-9
            set geo_fac [expr 1/sqrt($lef*$wef*1e12)]
            set vthmis [expr $scaling*3.635e-3*$geo_fac]
            set dlmis [expr $scaling*4.58e-3*$geo_fac*$lef]
            set dwmis [expr $scaling*3.73e-3*$geo_fac*$wef]
            set toxmis [expr $scaling*1.01e-3*$geo_fac*$geo_fac*$toxn]
        }
        @ /simulation_config/mc/nch_mc_$i/l = /struct/dist gauss * {{$dlmis}}
        @ /simulation_config/mc/nch_mc_$i/w = /struct/dist gauss * {{$dwmis}}
        @ /simulation_config/mc/nch_mc_$i/vth0 = /struct/dist gauss * {{$vthmis}}
        @ /simulation_config/mc/nch_mc_$i/tox = /struct/dist gauss * {{$toxmis}}
    }
    # Clean droppings
    foreach droppings [glob -nocomplain /tmp/*characterization_task*] {
        file delete $droppings
    }
    foreach droppings [glob -nocomplain /tmp/char_{mis,size,ids,gm,ro}*.*] {
        file delete $droppings
    }
```

```
set views {Ids_mis Vth_mis}
foreach ::corner $::corner_list {
    set ::temp $::corner_to_temp($::corner)
    fork_task char_mis_task {
        textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
        ######### Template netlist
        netlist ".include $::env(RAMSPICE)/Tech_DB/${tech}/${tech}.sp"
        netlist {
            ** the N-transistor
            * name D G S B model L W
        }
        netlist ".temp $::temp"
        set vars_of_interest {}
        set i3 0
        foreach L $l_values {
            set i4 0
            foreach W $w_values {
                # mosfet {name type D G S B L W Lmin}
                mosfet mn_${i3}_${i4} $type 3 1 ${i3}_${i4} 0 $L $W
                netlist "r_${i3}_${i4} ${i3}_${i4} 0 1e-12"
                lappend vars_of_interest mn_${i3}_${i4}
                lappend vars_of_interest Vth
                lappend vars_of_interest mn_${i3}_${i4}
                lappend vars_of_interest Ids
                incr i4
            }
            incr i3
        }
        netlist "
        Vds 3 0 dc $max_supply ac 0
        Vgs 1 0 dc [expr $max_supply/2] ac 0
```

```
.end
"
set O [open test.sn w]
puts $O $template_netlist
close $O
update_netlist

Info:  Measuring mis(W,L)
######### Characterizing loops
Info: simulation started ([clock format [clock seconds]])
set result [monte_carlo_${::bsim_version} 200 /simulation_config/mc
$vars_of_interest ::spice::op]
Info: done Mismatch running. Saving results. ([clock format [clock
seconds]])
Info: Done ([clock format [clock seconds]])
set i 0
set i3 0
foreach L $l_values {
    set i4 0
    foreach W $w_values {
        set Vth [lindex $result $i]
        incr i
        set Sigma [lindex $result $i]
        ^ @ /look_up_tables/$type/Vth_mis/${::corner}($i3,$i4) = [expr
$Sigma/$Vth]
        incr i
        set Ids [lindex $result $i]
        incr i
        set Sigma [lindex $result $i]
        ^ @ /look_up_tables/$type/Ids_mis/${::corner}($i3,$i4) = [expr
$Sigma/$Ids]
        incr i
        incr i4
```

```
                        }
                        incr i3
                    }
                }
            }
            foreach ::corner $::corner_list {
                set ::temp $::corner_to_temp($::corner)
                ######### Initialize database
                set index_range {}
                foreach var {l w} {
                    lappend index_range [llength [set ${var}_values]]
                }
                foreach array $views {
                    Info: New Array: look_up_tables/$type/$array/${::corner}([join
$index_range ,])
                    @ look_up_tables/$type/$array/${::corner}([join $index_range ,]) !
                }
                foreach array $views {
                    set i3 0
                    foreach L $l_values {
                        ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 0
$i3 $L
                        incr i3
                    }
                    set i4 0
                    foreach W $w_values {
                        ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 1
$i4 $W
                        incr i4
                    }
                }
            }
```

```
                wait_for_forked char_mis_task
                foreach ::corner $::corner_list {
                    set ::temp $::corner_to_temp($::corner)
                    foreach array $views {
                        generate_lut  /look_up_tables/$type/$array/$::corner
                    }
                }
                @ /look_up_tables/$type/Vth_mis save $vth_mis_file
                @ /look_up_tables/$type/Ids_mis save $ids_mis_file
                # Cleanup
                foreach char_file [glob -nocomplain /tmp/char_mis_task*] {
                    file delete $char_file
                }
            }
            set noise_complete 1
            foreach ::corner $::corner_list {
                set thermal_noise_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_${::corner}_thermal_noise.db
                set flicker_noise_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_${::corner}_flicker_noise.db
                if {[file exists $thermal_noise_file]} continue
                set noise_complete 0
                break
            }
            if {!$noise_complete} {
        #        textbox    "Characterizing Noise for $type Vgs=($minVt,$max_supply)
Vds=(0,$max_supply)"
                textbox    "Characterizing Noise for $type "
                constrain "
                Vgs  $minVt         $max_supply     7
                Vds  [expr $max_supply/20]  $max_supply     4
                Vbs  [expr -$max_supply/3]  0           1
```

```
        "
        set ::fork_limit 1
        foreach old_task_file [glob -nocomplain /tmp/*.tcl] {
            file delete $old_task_file
        }
        set views {flicker_const thermal_noise}
        set total_array_volume 1
        proc noise_cont {coord} {
            @ look_up_tables/$type/flicker_const/${::corner}($coord) = 0
            return -code continue
        }
        set index_range {}
        foreach var {Vgs Vds Vbs} {
            lappend index_range $::constraints($var,index_range)
        }
        lappend index_range [llength $l_values]
        lappend index_range [llength $w_values]
        set_spice_var Captured_Quick_Noise 1
        foreach ::corner $::corner_list {
            set thermal_noise_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_${::corner}_thermal_noise.db
            set flicker_noise_file
$::env(RAMSPICE)/Tech_DB/${tech}/${::rez}/${tech}_${type}_${::corner}_flicker_noise.db
            if {[file exists $thermal_noise_file]} continue
            set ::temp $::corner_to_temp($::corner)
            foreach array $views {
                @ look_up_tables/$type/$array/${::corner}([join $index_range ,]) !
            }
            foreach array $views {
                foreach_in_range Vgs i0 {
                    ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 0
$i0 $Vgs
```

```
                        }
                        foreach_in_range Vds i1 {
                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 1
$i1 $Vds
                        }
                        foreach_in_range Vbs i2 {
                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 2
$i2 $Vbs
                        }
                        set i3 0
                        foreach L $l_values {
                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 3
$i3 $L
                            incr i3
                        }
                        set i4 0
                        foreach W $w_values {
                            ::spice::array_set_legend /look_up_tables/$type/$array/$::corner 4
$i4 $W
                            incr i4
                        }
                    }
                    textbox "Corner [string toupper $::corner], Temperature=$::temp degC"
                    for {set section 1} {[info exists bin(n,$section,lmin)]} {incr section} {}
                    ETA $section
                    for {set section 1} {[info exists bin(n,$section,lmin)]} {incr section} {
                        Info: Analyzing bin $section [clock format [clock seconds]]
                set dummy_flicker 4.3e-12
                set dummy_thermal 5e-11
                set dummy_count_flicker 1
                set dummy_count_thermal 1
                        netlist ".include $::env(RAMSPICE)/Tech_DB/${tech}/${tech}.sp"
                        netlist ".temp $temp"
```

```
netlist {
    ** the N-transistor
    * name D G S B model L W
    rtest 3 0 1e-12
    Vgs 2 0 dc 0 ac 0
    Vds 4 0 dc 0 ac 0
    Vbs 1 0 dc 0 ac 0
}
# mosfet {name type D G S B L W Lmin}
mosfet mn_0_0 $type 4 2 3 1 $bin(n,$section,lmin) $bin(n,$section,wmin)
$section
netlist {
    .end
}
update_netlist
set i3 0
foreach L $l_values {
    set i4 0
    foreach W $w_values {
        set prefix [string index $type 0]
        if {$section!=[find_mosfet_bin $prefix $L $W]} {
incr i4
continue
}
        ::spice::alter mn_0_0 w = $W
        ::spice::alter mn_0_0 l = $L
        foreach_in_range Vgs i0 {
            ::spice::alter vgs = $Vgs
            foreach_in_range Vds i1 {
                ::spice::alter vds = $Vds
                foreach_in_range Vbs i2 {
```

```
::spice::alter vbs = $Vbs
::spice::noise v(3) vgs lin 2 1 2
set thermal_noise [::spice::get_spice_data
Captured_Thermal_Noise end]
if {[string match *nan* $thermal_noise]} {
    set thermal_noise 0
}
set Ssi [::spice::get_spice_data Captured_Ssi end]
set Captured_EffFreq [::spice::get_spice_data
Captured_EffFreq end]
if {[string match *nan* $Captured_EffFreq]} {
    noise_cont $i0,$i1,$i2,$i3
}
set flicker_noise [expr $Ssi*$Captured_EffFreq]
if {![regexp {^[0-9\.\-\+e]+$} $flicker_noise]} {
Info: flicker_noise=$flicker_noise
    set flicker_noise 0
}
::spice::op
set Ids1 [::spice::get_spice_data V(3) 0]
::spice::alter vgs = [expr $Vgs+$::epsilon]
::spice::op
set Ids2 [::spice::get_spice_data V(3) 0]
set gm [expr ($Ids2-$Ids1)*1e12/$::epsilon]
if {$gm==0} {
@
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3,$i4) =  [expr
$dummy_flicker/$dummy_count_flicker]
@
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3,$i4) = [expr
$dummy_thermal/$dummy_count_thermal]
} else {
if {[catch {@
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3,$i4) =  [expr
sqrt($flicker_noise*$W*$L/($gm*$gm))]}]} {
```

```
                                @
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3,$i4) = [expr
$dummy_flicker/$dummy_count_flicker]

                            } else {

                                set dummy_flicker [expr
$dummy_flicker+sqrt($flicker_noise*$W*$L/($gm*$gm))]

                                incr dummy_count_flicker

                            }

                            if {[catch {@
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3,$i4) = [expr
sqrt($thermal_noise/$gm)]}]} {

                                @
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3,$i4) = [expr
$dummy_thermal/$dummy_count_thermal]

                            } else {

                                set dummy_thermal [expr $dummy_thermal+sqrt($thermal_noise/$gm)]

                                incr dummy_count_thermal

                            }

                            }

                            #report_vars Vgs Vds Vbs W L

                            #Info: thermal_noise=[@
look_up_tables/$type/thermal_noise/${::corner}($i0,$i1,$i2,$i3,$i4)]

                            #Info: flicker_noise=[@
look_up_tables/$type/flicker_const/${::corner}($i0,$i1,$i2,$i3,$i4)]

                                        ::spice::destroy all

                                        ::spice::alter vgs = $Vgs

                                    }

                                    ETA

                                }

                            }

                            incr i4

                        }

                        incr i3

                    }

                }
```

```
                @ /look_up_tables/$type/thermal_noise/$::corner save ${thermal_noise_file}
                @ /look_up_tables/$type/flicker_const/$::corner save ${flicker_noise_file}
            }
        }
    }
    Exit
```

B. 6. Γ Circuit Compiler

```
# \
    exec $RAMSPICE/ramspice $0 $argv
source $::env(RAMSPICE)/Gamma/GammaCC.tcl
get_opts
default ::opt(interpolation) LUT
default ::opt(process) ss
default ::opt(device) nch
default ::opt(tech) tsmc040
default ::opt(topv) 1.1
default ::opt(l) 360e-9
default ::opt(w) 360e-9
default ::opt(vgs) 1.0
default ::opt(vbs) 0.0
default ::opt(r) 50
default ::opt(ref) 100e-6
default ::opt(op_limit) 2
default ::opt(step_limit) 1000
default ::opt(step_count) 10
default ::opt(np) 1
```

```
default ::opt(mode) dc

default ::opt(rez) 5:5:3:6


set ::opt(mode) [string tolower $::opt(mode)]

default EPS0 8.85418e-12

default ::opt(epsrox) 3.9

default ::opt(source) $::env(RAMSPICE)/Etc/Tech_DB/$::opt(tech)/4d/$::opt(rez)/

source $::env(RAMSPICE)/Sizer/simplify.tcl

source $::env(RAMSPICE)/Sizer/matrices.tcl

source $::env(RAMSPICE)/Sizer/derivatives.tcl

source $::env(RAMSPICE)/Sizer/polynomials.tcl

source $::env(RAMSPICE)/Etc/Tech_DB/$::opt(tech)/binning_$::opt(tech).tcl

proc derive_by_polish {by expr} {

    if {[llength $expr]==1} {

        set expr [lindex $expr 0]

    }

    if {[llength $expr]==1} {

        if {$expr==$by} {

            return 1

        }

        return 0

    }

    lassign $expr op X Y

    switch $op {

        + {
```

```
            return [list $op [derive_by_polish $by $X] [derive_by_polish $by $Y]]
        }
        - {
            return [list $op [derive_by_polish $by $X] [derive_by_polish $by $Y]]
        }
        * {
            return [list + [list $op $X [derive_by_polish $by $Y]] [list $op [derive_by_polish
$by $X] $Y]]
        }
    }
}
proc derive_expression {by expr} {
    return [DERIVE $by $expr]
}
foreach dev {nch pch} dtox {2.7e-10 3.91e-10} toxe {2.47e-9 2.71e-9} {
    set toxp [expr $toxe-$dtox]
    @ /look_up_tables/$dev/cox = [expr $::opt(epsrox)*$EPS0/$toxp]
    @ /look_up_tables/$dev !
    foreach param {ids gm ro} {
        @ /look_up_tables/$dev load $::opt(source)/$::opt(tech)_${dev}_${param}.db
    }
    @ /look_up_tables/$dev/thermal_noise/ !
    @ /look_up_tables/$dev/flicker_noise/ !
    @ /look_up_tables/$dev/thermal_noise/ load
$::opt(source)/$::opt(tech)_${dev}_ss_thermal_noise.db
```

```
    @ /look_up_tables/$dev/flicker_noise/ load
$::opt(source)/$::opt(tech)_${dev}_ss_flicker_noise.db

    foreach cap {cgg cgd cgs cgb cdd cdg cdb cds csd csg css csb cbd cbg cbs cbb} {

        @ /look_up_tables/$dev/$cap/ load $::opt(source)/$::opt(tech)_${dev}_ss_$cap.db

    }

}


set ::all_transistors {}

set ::all_nodes {}

#array set ::vdc {0 0}


proc add_idc {name m p value} {

    set ::idc($m,$p) $value

    add_node $m $p

}

set ::dependent_nodes(0) 1

proc add_vdc {name m p value} {

    set init_value $value

    while {[regexp {^(.*)@([a-zA-Z0-9_:]+)(.*)$} $init_value -> pre c post]} {

        set init_value $pre

        append init_value [@ $c]

        append init_value $post

    }

    if {[@ $init_value ?]} {

        set init_value [@ $init_value]
```

```
    }
    if {[catch {expr $value}]} {
        set ::vdc($m,$p) @$value
    } else {
        set ::vdc($m,$p) $value
    }
    add_node $m $p
    if {[info exists ::dependent_nodes($m)]} {
Dinfo: $p depends on $m ($value)
        set ::dependent_nodes($p) {}
        $p:Next=>$m+$value
        $p:V=>$p:Next
        @ $p:Next = [expr [@ $m:Next]+$init_value]
        @ $p:V = [expr [@ $m:Next]+$init_value]
    } elseif {[info exists ::dependent_nodes($p)]} {
Dinfo: $m depends on $p ($value)
        set ::dependent_nodes($m) {}
        $m:Next=>$p+$value
        $m:V=>$m:Next
        @ $m:Next = [expr [@ $p:Next]-$init_value]
        @ $m:V = [expr [@ $p:Next]-$init_value]
    }
}
array set ::all_resistors {}
set ::all_nodes {}
```

```
proc add_resistor {name m p value} {
    add_node $m $p
    foreach node [list $m $p] {
        if {![info exists ::mna_equations($node)]} {
            set ::mna_equations($node) ""
        }
    }
    if {![info exists ::all_resistors($m,$p)]} {
        set ::all_resistors($m,$p) $value
    } else {
        set ::all_resistors($m,$p) [expr
$value*$::all_resistors($m,$p)/($value+$::all_resistors($m,$p))]
    }
}
array set ::mna_mapping {}
proc mna_map {node} {
    if {[info exists ::mna_mapping($node)]} {
        return [mna_map $::mna_mapping($node)]
    }
    return $node
}
default ::opt(eps) 1e-4
set ::epsilon $::opt(eps)
proc Vdiff {v1 v2} {
    if {$v2==0} {
```

```
            return $v1
        }
        if {$v2=="0:V"} {
            return @$v1
        }
        if {$v2=="\{0:V\}"} {
            return @$v1
        }
        if {$v1==0} {
            return "(-@$v2)"
        }
        if {$v1=="0:V"} {
            return "(-@$v2)"
        }
        if {$v1=="\{0:V\}"} {
            return "(-@$v2)"
        }
        if {$v1==$v2} {
            return 0
        }
        return "(@$v1-@$v2)"
    }
    proc add_mna {i j element} {
        if {$i=="0"} return
        if {$j=="0"} return
```

```
    set i [lsearch $::independent_nodes $i]

    set j [lsearch $::independent_nodes $j]

    default ::MNA($i,$j)

    default ::MNA(dim) 0

    if {$::MNA(dim)<=$i} {

        set ::MNA(dim) $i

        incr ::MNA(dim)

    }

    if {$::MNA(dim)<=$j} {

        set ::MNA(dim) $j

        incr ::MNA(dim)

    }

    append ::MNA($i,$j) $element

    regsub {^\++} $::MNA($i,$j)  {} ::MNA($i,$j)

    regsub {^\+\-} $::MNA($i,$j)  {-} ::MNA($i,$j)

    regsub {^\-\+} $::MNA($i,$j)  {-} ::MNA($i,$j)

}

proc tmp_sort {t1 t2} {

    regsub tmp $t1 {} i1

    regsub tmp $t2 {} i2

    if {int($i1)>int($i2)} {

        return 1

    }

    return 0

}
```

```
proc add_node {args} {
    foreach m $args {
        ladd ::all_nodes $m
        if {![@ $m:V ?]} {
            @ $m:V = 0
        }
    }
}
set ::circuit_components {}
proc add_transistor {name d g s b type args} {
    lappend ::all_transistors $name
    add_node $d $g $s $b
    set ::transistors($name,connectivity) trivial
    if {$d==$g} {
        set ::transistors($name,connectivity) diode
    }
    foreach field {d g s b type} {
        set ::transistors($name,$field) [set $field]
    }
    foreach param $args {
        lassign [split $param =] field value
        set ::transistors($name,$field) $value
    }
    set class [lindex [split $name _] 0]
    foreach field {L W} {
```

```
        if {![info exists ::transistors($name,$field)]} {
            set ::transistors($name,$field) size:$field$class
            @ size:$field$class = 3.6e-8
        } elseif {[regexp {^\((.*)\)$} $::transistors($name,$field) -> guide]} {
            set ::transistors($name,$field) size:$field$class
            @ size:$field$class = $guide
        } else {

        }
    }
}
default ::opt(iref) 50e-6
source $::env(RAMSPICE)/Etc/Topologies/$::opt(topology).gsp
@ param/unique = 0

foreach {p unit formula step_factor} {
    Adc     dB 20*log10(abs(@)) 1e-16
    CMRR     dB 20*log10(abs(@)) 1e-13
    PSRR     dB 20*log10(abs(@)) -1e-11
    Rout     Ohm @ -1e-19
    BW     Hz @ 7e-23
    PM     deg @ 1
    Cin     F @ -1e-15
    ts     sec @ -1e-6
    Nt     V^2/Hz @ -1e-9
```

```
    Nf    V^2/Hz @ -1e-14
    TotalNoise V @ -1e-14
    fc     Hz @ -1e-17
    Vos   V @ -1e-6
    Area m^2 @ -1e-12
    Power W @ -1e-7
} {
    @ /property/$p = 0
    @ /property/$p/unit = string $unit
    @ /property/$p/formula = string $formula
    @ /property/$p/step_factor = $step_factor
    if {$step_factor<0} {
        lappend pareto_properties -$p
    } else {
        lappend pareto_properties $p
    }
    @ /size foreach_child s {
        @ /property/$p/$s = 0
    }
}
.compile_circuit
if {[file exists $::env(RAMSPICE)/Etc/Templates/$::opt(topology)/models_$::opt(tech).db]} {
    exit
}
# Prepare some defaults in the skeleton db file
```

```
set pareto_properties {}
set pareto_sizes {}
@ size foreach_child s {
    @ size/$s = 1e-6
    lappend pareto_sizes $s
}
@ param foreach_child p {
    if {[regexp {^i[^n]} $p] } {
        @ param/$p = 10e-6
    } elseif {[string match r* $p]} {
        @ param/$p = 1e9
    } else {
        @ param/$p = [expr $::opt(topv)/2]
    }
}
foreach node $::all_nodes {
    @ $node/V = [expr $::opt(topv)/2]
}
@ vdd:V = $::opt(topv)
@ param/vdd = $::opt(topv)
@ p1 = 0
@ p2 = 0
@ op_iterations = 10

@ /pareto(([join $pareto_sizes ,]|[join $pareto_properties ,])) !
```

```
@ / save $::env(RAMSPICE)/Etc/Templates/$::opt(topology)/models_$::opt(tech).db

exit


        # \

            exec $RAMSPICE/ramspice $0 $argv


        ################################################

        #

        #            Topology compiler

        #

        ################################################

        proc .param {name = value {min {}} {max {}}} {

            if {[catch {set evaluated_value [expr $value]}]} {

                @ param/$name = $value

            } else {

                @ param/$name = $evaluated_value

            }

            if {$min!={}} {

                @ param/$name/min = $min

            }

            if {$max!={}} {

                @ param/$name/max = $max

            }

        }

        proc .size {name = value {min {}} {max {}} {step {}} {dependence {}}} {
```

```
    if {[catch {set evaluated_value [expr $value]}]} {
        @ size/$name = $value
    } else {
        @ size/$name = $evaluated_value
    }
    foreach field {min max step dependence} {
        if {[catch {set evaluated_value [expr [set $field]]}]} {
            set ::sizing_code($name,$field) [set $field]
        } else {
            set ::sizing_code($name,$field) @size:$name:$field
            @ size:$name:$field = $evaluated_value
        }
    }
    default ::sizers_list {}
    lappend ::sizers_list $name


}
proc .property {name args} {
    default ::DERMODE first
Info: PROPERTY "$name=$args"
    set original_der_mode $::DERMODE
    set ::DERMODE first
    array set opt {}
    set current_switch arguments
    foreach arg $args {
```

```
        if {[regexp {^\-(\S+)$} $arg -> switch]} {
            set current_switch $switch
            default opt($current_switch) {}
            continue
        }
        lappend opt($current_switch) $arg
    }
    default opt(to_display) @
    default opt(from_display) @
    default opt(unit) {}
    default opt(more) better
    default opt(denom) {}
    @ property/$name/denom = string $opt(denom)
    if {![info exists opt(expression)]} {
Error: property requires a -expression switch
        exit
    }
    # set opt(expression) [flat_expression $opt(expression)]
    foreach field [array names opt] {
        set ::properties($name,$field) $opt($field)
    }
    @ property/$name = 0
    # property:$name=>$opt(expression)
    switch $opt(more) {
        better {@ property/$name/op = string +}
```

```
            worse {@ property/$name/op = string -}

            default {

        Error: A property $name can be either '-more better' or '-more worse'.

                exit

            }

        }

        set ::DERMODE $original_der_mode

    }

    proc .dependence {args} {

        regsub -all {\s} $args {} expr

        lassign [split $expr =] net dep

        set ::DESIGN_DEPENDENCES($net) $dep

    }

    proc .spec {name op value} {

        if {![info exists ::properties($name,expression)]} {

    Error: $name is not a defined property you can use in a spec. Start with: .property
$name -expression <expression>

            exit

        }

    Info: Examining  $name $op $value w.r.t. previous specifications

        set useful_op 1

        foreach previous_entry [array names ::specification $name,*] {

            set previous_op [lindex [split $previous_entry ,] 1]


            switch $op {
```

```
                    "=" {
                        switch $previous_op {
                            "=" {
                                if {$::specification($previous_entry)!=$value} {
        Error: Conflicting specification! $name cannot be both $value and
$::specification($previous_entry)
                                    exit
                                }
                            }
                            "<" {
                                if {$::specification($previous_entry)<$value} {
                                    Error: Conflicting specification! $name cannot be both
$value and less than $::specification($previous_entry)
                                    exit
                                }
                                array unset ::specification $previous_entry
                            }
                            ">" {
                                if {$::specification($previous_entry)>$value} {
        Error: Conflicting specification! $name cannot be both $value and greater than
$::specification($previous_entry)
                                    exit
                                }
                                array unset ::specification $previous_entry
                            }
                        }
```

```
                    }
                    "<" {
                        switch $previous_op {
                            "=" {
                                if {$::specification($previous_entry)>$value} {
                                    Error: Conflicting specification! $name cannot be less than
$value and equal to $::specification($previous_entry)
                                    exit
                                }
                                set useful_op 0
                            }
                            "<" {
                                if {$::specification($previous_entry)<$value} {
                                    set useful_op 0
                                } else {
                                    array unset ::specification $previous_entry
                                }
                            }
                            ">" {
                                if {$::specification($previous_entry)>$value} {
    Error: Conflicting specification! $name cannot be less than $value and greater than
$::specification($previous_entry)
                                    exit
                                }
                            }
                        }
```

```
                    }
                    ">" {
                        switch $previous_op {
                            "=" {
                                if {$::specification($previous_entry)<$value} {
                                    Error: Conflicting specification! $name cannot be greater
than $value and equal to $::specification($previous_entry)
                                    exit
                                }
                                set useful_op 0
                            }
                            ">" {
                                if {$::specification($previous_entry)>$value} {
                                    set useful_op 0
                                } else {
                                    array unset ::specification $previous_entry
                                }
                            }
                            "<" {
                                if {$::specification($previous_entry)<$value} {
        Error: Conflicting specification! $name cannot be greater than $value and less than
$::specification($previous_entry)
                                    exit
                                }
                            }
                        }
```

```
            }
        }
    }
    if {$useful_op} {
        set ::specification($name,$op) $value
    } else {
        Warning: $name $op $value is not a useful spec
    }
}
proc s2iW {expr varReal varImag {top 1}} {
    upvar $varReal Real
    upvar $varImag Imag
    if {$top} {
        while {[regexp {[\-\+][\-\+]} $expr]} {
            regsub -all {\-\+} $expr {-} expr
            regsub -all {\+\-} $expr {-} expr
            regsub -all {\-\-} $expr {+} expr
            regsub -all {\++} $expr {+} expr
        }
        set expr [polish $expr]
        while {[llength $expr]==1} {
            set expr [lindex $expr 0]
        }
    }
    if {[llength $expr]==1} {
```

```
    if {$expr=="@s"} {
        set Real 0
        set Imag W
        return
    }
    set Real $expr
    set Imag 0
    return
}
set op [lindex $expr 0]
set L [lindex $expr 1]
set R [lindex $expr 2]
s2iW $L Lr Li 0
s2iW $R Rr Ri 0
switch $op {
    - {
        set Real [simple- $Lr $Rr]
        set Imag [simple- $Li $Ri]
    }
    + {
        set Real [simple+ $Lr $Rr]
        set Imag [simple+ $Li $Ri]
    }
    * {
        set Real [simple- [simple* $Lr $Rr] [simple* $Li $Ri]]
```

```
            set Imag [simple+ [simple* $Lr $Ri] [simple* $Li $Rr]]
        }
        default {
            Error: Operator $op not supported!
            exit
        }
    }
    while {[regexp {[\-\+][\-\+]} $Real]} {
        regsub -all {\-\+} $Real {-} Real
        regsub -all {\+\-} $Real {-} Real
        regsub -all {\-\-} $Real {+} Real
        regsub -all {\++} $Real {+} Real
    }
    while {[regexp {[\-\+][\-\+]} $Imag]} {
        regsub -all {\-\+} $Imag {-} Imag
        regsub -all {\+\-} $Imag {-} Imag
        regsub -all {\-\-} $Imag {+} Imag
        regsub -all {\++} $Imag {+} Imag
    }
    return
}
proc .prep_mna {mode} {
    array unset ::MNA
    set idc_orig [array get ::idc]
    set vdc_orig [array get ::vdc]
```

```
        set ::MNAy {}
        @ s = 0
        set ::independent_nodes {}
        foreach node [lsort $::all_nodes] {
            skip {$node==0}
            lappend ::independent_nodes $node
        }
        set dim [llength $::independent_nodes]
        array set ::MNA [list dim $dim]
        # Add resistors
        foreach res_nodes [array names ::all_resistors] {
            lassign [split $res_nodes ,] m p
            foreach node [list $m $p ] {
                if {[info exists ::vdc($node)]} {
                    set V($node) $::vdc($node)
                } else {
                    set V($node) $node:V
                }
            }
            if {[catch {set G [expr 1.0/$::all_resistors($res_nodes)]}]} {
                set ::G_equations(${m}_${p}) 1.0/@$::all_resistors($res_nodes)
                set G Gds_${m}_${p}
                set ::sensitivity(Gds_${m}_${p},$::all_resistors($res_nodes)) -
1.0/($::all_resistors($res_nodes)*$::all_resistors($res_nodes))
            }
```

```
    Info: Adding Resistor! m=$m p=$p
    add_mna $m $m $G
    add_mna $p $p $G
    add_mna $p $m -$G
    add_mna $m $p -$G
}
array unset ::Ids_equations

#Add transistors
foreach name $::all_transistors {
    foreach field {type L W d g s b} {
        set $field $::transistors($name,$field)
    }
    foreach node_name {d g s b} {
        set node [set $node_name]
        if {[info exists ::vdc($node)]} {
            set V$node_name $::vdc($node)
        } else {
            set V$node_name $node:V
            #               @ $node:V = [expr $::opt(topv)]
            if {![@ $node:V ?]} {
                @ $node:V = $::opt(topv)
            }
            if {![@ $node:Next ?]} {
                @ $node:Next = $::opt(topv)
```

```
                }
            }
        }
#        add_mna_entry $s $d "Gds_$name"
        set ::transistors($name,Vgs) [Vdiff $Vg $Vs]
        @ Gds_$name = 1
        add_mna $s $s "+@$name:go"
        add_mna $s $d "-@$name:go"
        add_mna $d $s "-@$name:go"
        add_mna $d $d "+@$name:go"
        default ::idc($d,$s)
        append ::idc($d,$s) "-@$name:Ideq"
        add_mna $s $s "+@$name:gm"
        add_mna $s $g "-@$name:gm"
        add_mna $d $s "-@$name:gm"
        add_mna $d $g "+@$name:gm"
        if {$type=="nch"} {
            set ::Ids_equations($name)
"gamma_gcc_interpolate_4(`@:look_up_tables:$type:Ids:$::opt(process):LUT,[Vdiff $Vg
$Vs],[Vdiff $Vd $Vs],[Vdiff $Vb $Vs],@$L)*@$W/@$L-@$name:gm*[Vdiff $Vg $Vs]-@$name:go*[Vdiff
$Vd $Vs]"
        } else {
            set ::Ids_equations($name) "-
gamma_gcc_interpolate_4(`@:look_up_tables:$type:Ids:$::opt(process):LUT,[Vdiff $Vg $Vs],[Vdiff
$Vd $Vs],[Vdiff $Vb $Vs],@$L)*@$W/@$L-@$name:gm*[Vdiff $Vg $Vs]-@$name:go*[Vdiff $Vd $Vs]"
        }
```

```
                set ::Nt_equations($name)
"gamma_gcc_interpolateg_4(`@:look_up_tables:$type:thermal_noise:ss:LUT,[Vdiff $Vg $Vs],[Vdiff
$Vd $Vs],[Vdiff $Vb
$Vs],@$L,`@$name:dNt_dvgs,`@$name:dNt_dvds,`@$name:dNt_dvbs,`@$name:dNt_dl)"

                set ::Nf_equations($name)
"gamma_gcc_interpolateg_4(`@:look_up_tables:$type:flicker_noise:ss:LUT,[Vdiff $Vg $Vs],[Vdiff
$Vd $Vs],[Vdiff $Vb
$Vs],@$L,`@$name:dNf_dvgs,`@$name:dNf_dvds,`@$name:dNf_dvbs,`@$name:dNf_dl)"

                set ::gm_equations($name)
"gamma_gcc_interpolateg_4(`@:look_up_tables:$type:gm:$::opt(process):LUT,[Vdiff $Vg
$Vs],[Vdiff $Vd $Vs],[Vdiff $Vb
$Vs],@$L,`@$name:dgm_dvgs,`@$name:dgm_dvds,`@$name:dgm_dvbs,`@$name:dgm_dl)*@$W/@$L"

                set ::go_equations($name)
"@$W/@$L/gamma_gcc_interpolateg_4(`@:look_up_tables:$type:ro:$::opt(process):LUT,[Vdiff $Vg
$Vs],[Vdiff $Vd $Vs],[Vdiff $Vb
$Vs],@$L,`@$name:dro_dvgs,`@$name:dro_dvds,`@$name:dro_dvbs,`@$name:dro_dl)"

                @ $name:Ideq = 0

                @ $name:go = 1

                @ $name:gm = 1

                set ::sensitivity($name:gm,$W) @$name:gm:$W

                @ $name:gm:$W = 0

                $name:gm:$W=>@$name:gm/@$W

                set ::sensitivity($name:gm,$L) @$name:gm:$L

                @ $name:gm:$L = 0

                $name:gm:$L=>(@$W*@$name:dgm_dl-2*@$name:gm)/@$L

                @ $name:go:$W = 0

                set ::sensitivity($name:go,$W) @$name:go:$W

                $name:go:$W=>@$name:go/@$W

                set ::sensitivity($name:go,$L) @$name:go:$L
```

```
        @ $name:go:$L = 0
        $name:go:$L=>(@$name:dro_dl*@$L/@$W-@$L/@$name:go)*@$name:go*@$name:go
        foreach pin {d g s b} {
            set ac($pin) [set $pin]
            if {$ac($pin)=="0"} {
            set ac($pin) vdd
            }
        }
        if {$mode=="ac" || $mode=="noise"} {
            foreach from {g d s b} {
                foreach to {d g s b} {
                    add_mna $ac($from)  $ac($to) "+@$name:c$from$to*@s"
                    @ $name:c$from$to = 0
                }
            }
        }
    }
    if {$mode!="zout"} {
        foreach idc_pair [array names ::idc] {
            lassign [split $idc_pair ,] m p
            foreach node [list $m $p] sign {+ -} {
                skip {$node==0}
                set index [lsearch $::independent_nodes $node]
                default ::MNA($index)
                set element $::idc($idc_pair)
```

```
            if {![regexp {^[0-9\-\+]} $element]} {
                set element "@$element"
            }
            regsub -all {\-\-} "$sign$element" {+} entry
            regsub -all {\-\+} $entry {-} entry
            regsub {^\++} $entry {+} entry
            append ::MNA($index) "$entry"
            regsub {^\++} $::MNA($index)  {} ::MNA($index)
        }
    }
} else {
    set index [lsearch $::independent_nodes $::output_net]
    default ::MNA($index)
    append ::MNA($index) +1
    regsub {^\++} $::MNA($index)  {} ::MNA($index)
}
set i $dim
set all_vdc [array names ::vdc]
foreach vdc_pair $all_vdc {
    lassign [split $vdc_pair ,] m p
    default ::MNA($m,$p)
    default ::MNA($p,$m)
    foreach node [list $m $p] entry {-1 1} {
        if {$node!="0"} {
            set j [lsearch $::independent_nodes $node]
```

```
                set ::MNA($j,$i) $entry
                set ::MNA($i,$j) $entry
            }
        }
        if {$mode=="zout"} {
            set ::MNA($i) 0
        } elseif {![regexp {^[0-9\-\+]} $::vdc($vdc_pair)]} {
            set ::MNA($i) @$::vdc($vdc_pair)
        } else {
            set ::MNA($i) $::vdc($vdc_pair)
        }
        incr i
    }
    for {set i 0} {$i<$dim} {incr i} {
        if {![info exists ::MNA($i)]} {
            lappend ::MNAy 0
        } else {
            lappend ::MNAy $::MNA($i)
        }
    }
    foreach vdc_pair $all_vdc {
        if {$mode=="zout"} {
            lappend ::MNAy 0
        } else {
            lappend ::MNAy $::vdc($vdc_pair)
```

```
    }
}
set dim [llength $::MNAy]
set ::MNA(dim) $dim
foreach entry [array names ::MNA] {
    set expr $::MNA($entry)
    regsub -all {\+\-} $expr {-} expr
    regsub -all {\-\+} $expr {-} expr
    regsub -all {\-\-} $expr {+} expr
    regsub -all {\+}   $expr {+} expr
    set ::MNA($entry) $expr
}
set old_y $::MNAy
set ::MNAy {}
foreach expr $old_y {
    regsub -all {\+\-} $expr {-} expr
    regsub -all {\-\+} $expr {-} expr
    regsub -all {\-\-} $expr {+} expr
    regsub -all {\+}   $expr {+} expr
    lappend ::MNAy $expr
}
if {$::C::target=="OP"} {
    if {$mode=="dc"} {
        set ::HTML [open $::env(RAMSPICE)/tmp/$::opt(topology)_MNA.html w]
        puts $::HTML <html>
```

```
puts $::HTML <head>

puts $::HTML {<style type="text/css">

    .matrix {

        position: relative;

    }

    .matrix:before, .matrix:after {

        content: "";

        position: absolute;

        top: 0;

        border: 1px solid #000;

        width: 6px;

        height: 100%;

    }

    .matrix:before {

        left: -6px;

        border-right: 0;

    }

    .matrix:after {

        right: -6px;

        border-left: 0;

    }

}

puts $::HTML </style>

puts $::HTML </head>

puts $::HTML <body>
```

```
}
puts $::HTML "<table class=\"matrix\" border=\"1\">"
for {set i 0} {$i<$dim} {incr i} {
    puts $::HTML <tr>
    puts $::HTML <td>
    puts $::HTML <b>
    puts $::HTML [lindex $::independent_nodes $i]
    puts $::HTML </b>
    puts $::HTML </td>
    for {set j 0} {$j<$dim} {incr j} {
        puts $::HTML <td>
        if {[info exists ::MNA($i,$j)]} {
            set td $::MNA($i,$j)
            regsub -all @ $td {} td
            regsub -all {:([a-zA-Z]+)} $td {<sub>\1</sub>} td
            puts $::HTML $td
        } else {
            puts $::HTML 0
        }
        puts $::HTML </td>
    }
    puts $::HTML <td>
    set td [lindex $::MNAy $i]
    regsub -all @ $td {} td
    regsub -all param: $td {} td
```

```
        regsub -all size: $td {} td
        regsub -all {:([a-zA-Z]+)} $td {<sub>\1</sub>} td
        puts $::HTML $td
        puts $::HTML </td>
        puts $::HTML </tr>
    }
    puts $::HTML </table>
    puts $::HTML <h2>
    DET ::MNA
set td $::det_calc_result
    regsub -all @ $td {} td
    regsub -all param: $td {} td
    regsub -all size: $td {} td
    regsub -all {:([a-zA-Z]+)} $td {<sub>\1</sub>} td
    puts $::HTML "DET=$td<br>"
    if {[set index_out [lsearch $::independent_nodes outp]]!=-1} {
        DET ::MNA ::MNAy $index_out
    } elseif {[set index_out [lsearch $::independent_nodes out]]!=-1} {
        DET ::MNA ::MNAy $index_out
}
set td $::det_calc_result
    regsub -all @ $td {} td
    regsub -all param: $td {} td
    regsub -all size: $td {} td
    regsub -all {:([a-zA-Z]+)} $td {<sub>\1</sub>} td
```

```
            puts $::HTML "V<sub>OUT</sub>=$td<br>"
            puts $::HTML </h2>
            if {$mode=="ac"} {
                puts $::HTML </body></html>
                close $::HTML
            }
        }
        array unset ::vdc
        array set ::vdc $vdc_orig
        array unset ::idc
        array set ::idc $idc_orig
        for {set i 0} {$i<$::MNA(dim)} {incr i} {
            set all_zeroes 1
            for {set j 0} {$j<$::MNA(dim)} {incr j} {
                skip {![info exists ::MNA($i,$j)]}
                skip {$::MNA($i,$j)==0}
                set all_zeroes 0
                break
            }
            skip {$all_zeroes==0}
            Error: Node [lindex $::independent_nodes $i] is dangling. Add path to ground or
a voltage source.
            exit
        }
    }
```

```
#proc .circuit {name} {
    #    set ::opt(topology) $name
#}
proc .compile_circuit {args} {
    if {[file exists $::env(RAMSPICE)/Etc/Templates/$::opt(topology)/bypass.ignore.c]}
{
        file copy -force
$::env(RAMSPICE)/Etc/Templates/$::opt(topology)/bypass.ignore.c /tmp/gamma_source.ignore.c
        gcc $::opt(topology) 0
        return
    }
    get_opts outp {} out {} outn {} in {} inn {} inp {} vdd {} name {}
    default ::opt(debug) 0
    set ::debug_mode $::opt(debug)
    foreach possible_ports {out outp outn inn inp in vdd} {
        skip {$opt($possible_ports)!={}}
        if {[@ param:$possible_ports ?]} {
            set opt($possible_ports) @param:$possible_ports
        }
        skip {$opt($possible_ports)!={}}
        if {[@ $possible_ports:V ?]} {
            set opt($possible_ports) @$possible_ports:V
        }
    }
    if {$opt(name)=={}} {
        set opt(name) $::opt(topology)
```

```
            }
            if {$opt(outp)=={} && $opt(outn)=={} && $opt(out)=={}} {
                Error: Cannot compile a circuit without a defined output. Add -out <net> to
the .compile_circuit command
                                        exit
                                    }
                                    if {$opt(inp)=={} && $opt(inn)=={} && $opt(in)=={}} {
        Error: Cannot compile a circuit without a defined input. Add -in <net> to
the .compile_circuit command
                exit
            }
            ######## Abstract circuit properties
            if {($opt(outp)!={} || $opt(out)!={}) && $opt(outn)=={}} {
                if {$opt(outp)!={} && $opt(out)!={} && $opt(out)!=$opt(outp)} {
                    Error: Conflicting definitions of output net: $opt(out)!=$opt(outp)
                    exit
                }
                if {$opt(outp)!={}} {
                    set ::output_net $opt(outp)
                } else {
                    set ::output_net $opt(out)
                }
                regsub {@} $::output_net {} ::output_net
                regsub {:.*} $::output_net {} ::output_net
                if {$opt(inp)!={} && $opt(inn)!={} && $opt(in)!={}} {
                    Error: Conflicting definitions of input net.
```

```
                exit
            }
            if {$opt(inp)!={} && $opt(inn)!={}} {
                set expr 0.5*([DERIVE $opt(inp) $::output_net]-[DERIVE $opt(inn)
$::output_net])
            } elseif {$opt(in)!={}} {
                set expr [DERIVE $opt(in) $::output_net]
            }
            Info: Adc expr=$expr ($::output_net)
            .property Adc -expression $expr -to_display 20*log10(@) -from_display
pow(10,@/20) -unit dB
            if {$opt(inp)!={} && $opt(inn)!={}} {
                .property CMRR -expression
derive($::output_net,$opt(inp))+derive($::output_net,$opt(inn)) -to_display 20*log10(@) -
from_display pow(10,@/20) -unit dB
            } elseif {$opt(in)!={}} {
                .property CMRR -expression 0 -to_display 20*log10(@) -from_display
pow(10,@/20) -unit dB
            }
            if {![@ property/PSRR ?]} {
                if {$opt(vdd)=={}} {
                    Error: Cannot define PSRR code for the circuit, because it has no -vdd
defined.
                        exit
                    }
                    .property PSRR -expression derive($::output_net,$opt(vdd)) -
to_display 20*log10(@) -from_display pow(10,@/20) -unit dB
```

```
                        }
                    }
                    foreach p {Adc CMRR PSRR Rout BW ts Nt Nf fc Vos Area Power} {
                        @ property/$p = 0
                    }
                    regsub {:V} $::output_net {} output_expr
                    .prep_mna dc
                    set dim $::MNA(dim)
                    @ op_iterations = $::opt(op_limit)
                    foreach metaC_file [glob -nocomplain $::env(RAMSPICE)/Gamma/metaC/*.tcl]
{
                regsub {\.tcl$} [file tail $metaC_file] {} target_name
                if {[file exists
$::env(RAMSPICE)/Gamma/metaC/$::opt(topology)/$target_name.tcl]} {
                    set metaC_file
$::env(RAMSPICE)/Gamma/metaC/$::opt(topology)/$target_name.tcl
                }
                Info: Compiling $target_name from [file dirname $metaC_file]
                code_target $target_name
                source $metaC_file
            }
            gcc $opt(name)
        }

        ##################################################
        #
```

```
#            gcc Interface
#
################################################

namespace eval C {
    variable O stdout
    array set code {}
    variable target OP
    variable code_template {
        #include <tcl.h>
        #include <stdio.h>
        #include <stdlib.h>
        #include "ctree.h"
        #include "look_up_table.h"
        // Some global pointers to keep reference of the contexts this object
manipulates
        GLOBAL_POINTERS_GO_HERE
        GLOBAL_VARIABLES_GO_HERE
        // The compiled function
        float vos(float isize) {
            VOS_FORMULA
        }
        static int tcl_gamma_import_cmd(ClientData clientData,Tcl_Interp *interp, int
objc, Tcl_Obj *const objv[]) {
            LOCAL_BUFFER_INIT_GOES_HERE
            return TCL_OK;
```

```
}

static int tcl_gamma_export_cmd(ClientData clientData,Tcl_Interp *interp, int objc, Tcl_Obj *const objv[]) {

    LOCAL_BUFFER_RETURN_GOES_HERE

    return TCL_OK;

}

static int tcl_gamma_op_cmd(ClientData clientData,Tcl_Interp *interp, int objc, Tcl_Obj *const objv[]) {

    //            Tcl_ResetResult(interp);

    OP_CODE_GOES_HERE

    return TCL_OK;

}

//      static int tcl_gamma_grad_cmd(ClientData clientData,Tcl_Interp *interp, int objc, Tcl_Obj *const objv[]) {

    //            Tcl_ResetResult(interp);

    //           GRAD_CODE_GOES_HERE

    //          return TCL_OK;

//    }

static int tcl_gamma_random_cmd(ClientData clientData,Tcl_Interp *interp, int objc, Tcl_Obj *const objv[]) {

    RANDOM_CODE_GOES_HERE

    return TCL_OK;

}

static int tcl_gamma_breed_cmd(ClientData clientData,Tcl_Interp *interp, int objc, Tcl_Obj *const objv[]) {

    BREED_CODE_GOES_HERE

    return TCL_OK;
```

```
}

static int tcl_gamma_random_breed_cmd(ClientData clientData,Tcl_Interp *interp,
int objc, Tcl_Obj *const objv[]) {

    RANDOM_BREED_CODE_GOES_HERE

    return TCL_OK;

}

static int tcl_gamma_random_breed_single_cmd(ClientData clientData,Tcl_Interp
*interp, int objc, Tcl_Obj *const objv[]) {

    RANDOM_BREED_SINGLE_CODE_GOES_HERE

    return TCL_OK;

}

// Initializing cTree references and registering the tcl_gamma_op_cmd command
as ::C::@name

int Gamma_Init(Tcl_Interp *interp) {

    if (Tcl_InitStubs(interp, TCL_VERSION, 0) == NULL) {

        return TCL_ERROR;

    }

    float *array_entry;

    context *c;

    GLOBAL_POINTER_INIT_GO_HERE

    Tcl_CreateObjCommand(interp, "::C::random", tcl_gamma_random_cmd, NULL,
NULL);

    Tcl_CreateObjCommand(interp, "::C::random_breed",
tcl_gamma_random_breed_cmd, NULL, NULL);

    Tcl_CreateObjCommand(interp, "::C::random_breed_single",
tcl_gamma_random_breed_single_cmd, NULL, NULL);

    Tcl_CreateObjCommand(interp, "::C::breed", tcl_gamma_breed_cmd, NULL, NULL);
```

```
                    // Tcl_CreateObjCommand(interp, "::C::grad", tcl_gamma_grad_cmd, NULL,
NULL);

                    Tcl_CreateObjCommand(interp, "::C::op", tcl_gamma_op_cmd, NULL, NULL);

                    Tcl_CreateObjCommand(interp, "::C::import", tcl_gamma_import_cmd, NULL,
NULL);

                    Tcl_CreateObjCommand(interp, "::C::export", tcl_gamma_export_cmd, NULL,
NULL);

                    return TCL_OK;

                }

            }

        }

        proc ::C::count_braces {line minvar maxvar} {

            upvar $minvar min

            upvar $maxvar max

            set max 0

            set min 0

            set count 0

            for {set i 0} {$i < [string length $line]} {incr i} {

                set c [string index $line $i]

                if {$c=="\\"} {

                    incr i

                    continue

                }

                if {$c=="\{"} {

                    incr count

                }
```

```
        if {$c=="\}"} {
            incr count -1
        }
        if {$count<$min} {
            set min $count
        }
        if {$count>$max} {
            set max $count
        }
    }
    return $count
}

proc ::C::print_line {line} {
    regsub -all {^\s*} $line {} line
    if {[string length $line]==0} return
    if {[regexp {^\S+:\s*$} $line]} {
        puts $::C::O $line
        return
    }
    set increment [::C::count_braces $line min max]
    if {$min<0} {
        incr ::rank $min
        puts -nonewline $::C::O [string repeat "    " $::rank]
        incr ::rank [expr -$min]
```

```
        } else {
            puts -nonewline $::C::O [string repeat "    " $::rank]
        }
        incr ::rank $increment
        puts $::C::O $line
    }
    proc ::C::tcl_preprocessor {c_code} {
        set bracket_rank 0
        set ::rank 0
        set lines [split $c_code \n]
        for {set i 0} {$i<[llength $lines]} {incr i ; set line [lindex $lines $i]} {
            set line [lindex $lines $i]
            if {[regexp {^\s*\/\/} $line]} {
                ::C::print_line $line
                continue
            }
            # Sometimes you can find Tcl code inside C. Don't pre-substitute variables in
those.
            if {[regexp {^\s*Tcl_Eval} $line]} {
                ::C::print_line $line
                continue
            }
            # Identify a beginning of Tcl preprocessed block
            if {[regexp {^\s*\#tcl\s+(.*)$} $line -> tcl_command] || [regexp {^\s*\#([A-
Z]\S+:.*)$} $line -> tcl_command]} {
                set bracket_rank [regexp -all {\{} $line]
```

```
        incr bracket_rank -[regexp -all {\}} $line]
        set template_body $tcl_command
        # one-liners can be precessed here
        while {($bracket_rank!=0)&&($i<[llength $lines])} {
            incr i
            set line [lindex $lines $i]
            append template_body \n
            append template_body $line
            incr bracket_rank [regexp -all {\{} $line]
            incr bracket_rank -[regexp -all {\}} $line]
        }
        uplevel #0 $template_body
        continue
    }
    # Substitute Tcl variables everywhere else.
    while {[regexp {\$\{(:*[a-zA-Z0-9_]+)\}} $line -> varname] } {
        if {[uplevel "info exists $varname"] && ![uplevel "array exists $varname"]}
{
            regsub -all "\\\$\\\{$varname\\\}" $line [uplevel "set $varname"] line
        } else {
            regsub -all "\\\$\\\{$varname\\\}" $line "\$`$varname" line
        }
    }
    while {[regexp {\$(:*[a-zA-Z0-9_]+)} $line -> varname] } {
        if {[uplevel "info exists $varname"] && ![uplevel "array exists $varname"]}
{
```

```
                regsub -all "\\\$$varname" $line [uplevel "set $varname"] line
            } else {
                regsub -all "\\\$$varname" $line "\$`$varname" line
            }
        }
        regsub -all {\$`} $line {\$} line
        # print simplified and substituted lines here
        ::C::print_line $line
    }
}
```

```
proc gcc {name {preprocess 1}} {
    if {$preprocess} {
        regsub -all @name $::C::code_template $name body
        set global_pointers {}
        set global_variables {}
        set global_pointer_init {}
        set local_buffer_init_goes_here {}
        set local_buffer_return_goes_here {}
        set used_var_names {}
```

```
set used_pointer_names {}
foreach target [array names ::C::code] {
    Info: Post processing $target
    set code $::C::code($target)
    while {[regexp {@+([A-Za-z0-9_:]+)} $code -> context_string]} {
        if {[info exists pointer_names($context_string)]} {
            #           regsub "&&@+$context_string" $code
`$var_names($context_string) code
            regsub "&@+$context_string" $code $pointer_names($context_string)
code
            regsub "@+$context_string" $code $var_names($context_string) code
            continue
        }
        Info: converting $context_string
        if {[regexp {^[0-9]} $context_string]} {
            regsub -all {[^a-zA-Z_0-9]} CONST_$context_string _ var_name
        } else {
            regsub -all {[^a-zA-Z_0-9]} $context_string _ var_name
        }
        regsub -all {[^a-zA-Z_0-9]} P$context_string _ pointer_name
        if {[lsearch $used_var_names $var_name]!=-1} {
            set i 0
            while {[lsearch $used_var_names $var_name$i]!=-1} {
                incr i
            }
            set var_name $var_name$i
```

```
                            }
                            lappend used_var_names $var_name
                            if {[lsearch $used_pointer_names $pointer_name]!=-1} {
                                set i 0
                                while {[lsearch $used_pointer_names $pointer_name$i]!=-1} {
                                    incr i
                                }
                                set pointer_name $pointer_name$i
                            }
                            lappend used_pointer_names $pointer_name
                            append global_pointers "float *$pointer_name;\n"
                            append global_variables "float $var_name;\n"
                            if {[regexp {(.*):PAT} $context_string -> base]} {
                                append global_pointer_init "$pointer_name=(float
*)get_PAT(\"$base\");\n"
                                regsub "&@+$context_string" $code $pointer_name code
                            } elseif {[regexp {(.*):LUT} $context_string -> base]} {
                                append global_pointer_init "$pointer_name=(float
*)get_LUT(\"$base\");\n"
                                regsub "&@+$context_string" $code $pointer_name code
                            } else {
                                #            append global_pointer_init
"resolve_context(\"$context_string\",`c,`array_entry);\n"
                                append global_pointer_init
"c=create_context(\"$context_string\");\n"
                                append global_pointer_init "$pointer_name=(float *)(`c-
>value.s);\n"
```

```
                append local_buffer_init_goes_here "$var_name=*$pointer_name;\n"

                append local_buffer_return_goes_here "*$pointer_name=$var_name;\n"

                regsub "&@+$context_string" $code $pointer_name code

                regsub "@+$context_string" $code $var_name code

            }

            set pointer_names($context_string) $pointer_name

            set var_names($context_string) $var_name

            incr i

        }

        regsub -all \& $code ` code

        regsub ${target}_CODE_GOES_HERE $body $code body

        Info: converted $context_string

    }

Info: Post Processing is Done

    regsub VOS_FORMULA $body $::VOS_FORMULA body

    regsub GLOBAL_POINTERS_GO_HERE $body $global_pointers body

    regsub GLOBAL_VARIABLES_GO_HERE $body $global_variables body

    regsub GLOBAL_POINTER_INIT_GO_HERE $body $global_pointer_init body

    regsub LOCAL_BUFFER_INIT_GOES_HERE $body $local_buffer_init_goes_here body

    regsub LOCAL_BUFFER_RETURN_GOES_HERE $body $local_buffer_return_goes_here body

    regsub -all `_ $body {P_} body

    regsub -all ` $body {\&} body

    regsub -all {\&look_up} $body {Plook_up} body

set ::C::O [open /tmp/gamma_pre_processed.ignore.c w]

puts $::C::O $body
```

```
        close $::C::O
            set ::C::O [open /tmp/gamma_source.ignore.c w]
            ::C::tcl_preprocessor $body
            close $::C::O
        }
        set find_lib_stub [glob -nocomplain /usr/*/libtclstub*]
                    if {$find_lib_stub=={}} {
    Error: This system has no tclstub library and therefore cannot compile code on the fly.
            exit
        }

        ######## Compilation
        default ::binary [ginfo binary]
        default ::target [ginfo target]
        set build_path /tmp/${::binary}_build/preprocessed-${::target}
        Info: Launching GCC

        uplevel "exec gcc -O3 [glob /tmp/${::binary}_build/object_files-[ginfo target]/*.o]
-fPIC -shared -DUSE_TCL_STUBS -I$build_path -I$build_path/Gamma/Data  -I$build_path/Gamma/LUT
-I$build_path/ngspice/root/maths/poly -I$build_path/ngspice/root/frontend -
I$build_path/ngspice/root/spicelib/devices -I$build_path/ngspice/root/xspice/icm/analog -
I/usr/include /tmp/gamma_source.ignore.c -L[file dirname [lindex $find_lib_stub 0]] -
ltclstub[info tclversion]  -o /tmp/libGamma.so |& tee  $::env(RAMSPICE)/compilation.log"
        if {[file exists /tmp/libGamma.so]} {
            Info: Shared Object was created for Gamma on [clock format [file mtime
/tmp/libGamma.so]]
            if {![file exists $::env(RAMSPICE)/Etc/Templates]} {
```

```
                file mkdir $::env(RAMSPICE)/Etc/Templates
            }
            if {![file exists $::env(RAMSPICE)/Etc/Templates/$name]} {
                file mkdir $::env(RAMSPICE)/Etc/Templates/$name
            }
            file copy -force /tmp/libGamma.so $::env(RAMSPICE)/Etc/Templates/$name
            file copy -force /tmp/gamma_source.ignore.c
$::env(RAMSPICE)/Etc/Templates/$name/
        }
        #    load $::env(RAMSPICE)/Etc/Templates/$name/libGamma.so
    }
    proc code_target {name} {
        set ::C::target $name
    }
    proc *c {args} {
        Info: CCC $args
        set body $args
        if {[llength $body]!=1} {
            append ::C::code($::C::target) "$body;\n"
        } else {
            append ::C::code($::C::target) [lindex $body 0]
            append ::C::code($::C::target) "\n"
        }
    }
    return
```

C. Usage Model

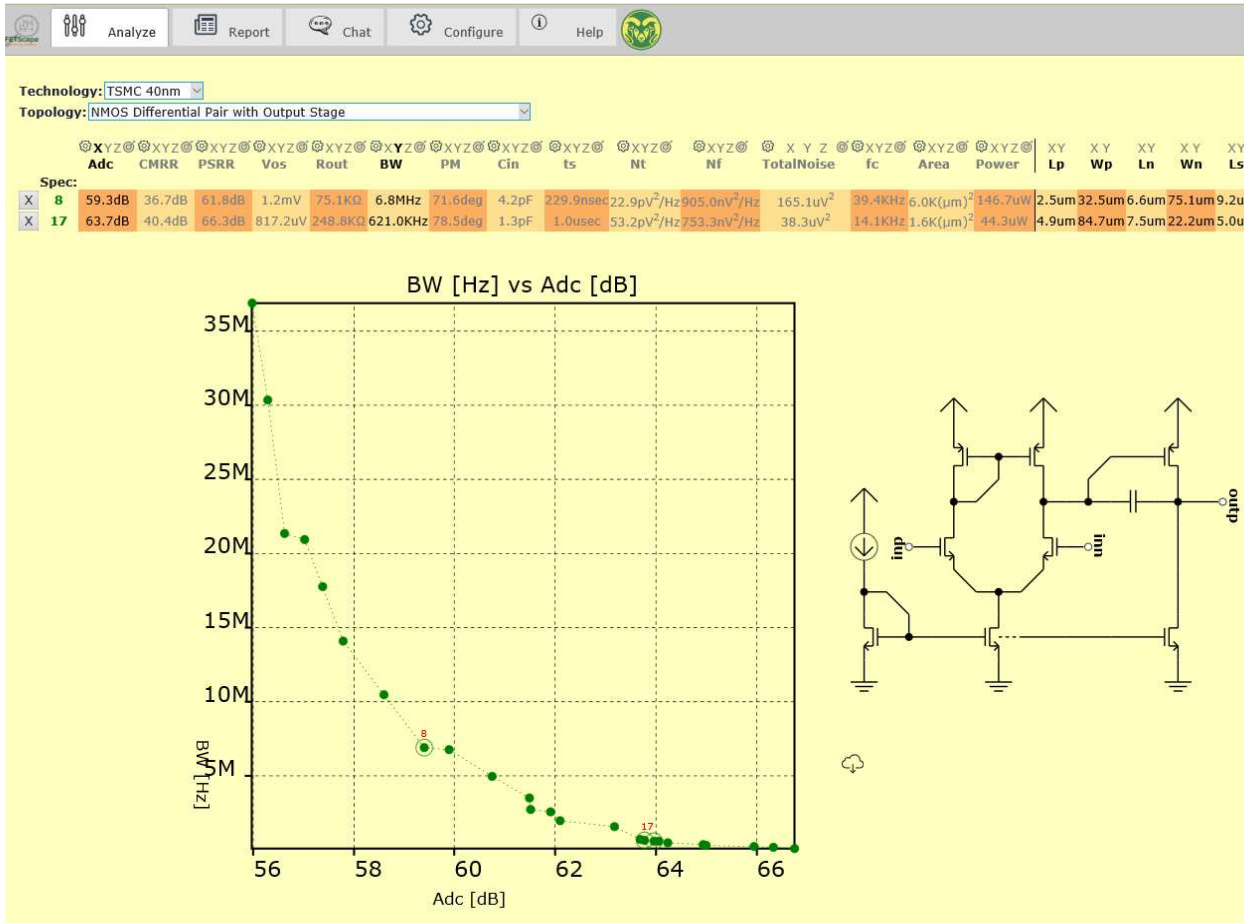


Figure C-1: Screen Capture of the Γ-powered Website

C.1. Use the Tool as a Design Aid.

The designer navigates the main page (Figure C-1) in the following order:

1. After login, select a target manufacturing technology and a topology to investigate.
2. A default front appears, with no specific requirements.

3. The designer can now enter a spec by clicking on the gear icon (Figure C-2) of each relevant performance property and filling in a pop-up entry line.

Changing the graph panel's axes designation is done by toggling the XYZ markers from gray to black. When the Z axis is selected it is shown in form of "heat" false color (Figure C-4**).** Otherwise, a Pareto front is shown (Figure C-1) A map – either 2D graph showing the corresponding Pareto front or a 2D heat-map showing performance dependence on selected parameters. Each marker corresponds to a circuit in the PAT. Intuitively, red ones show circuits that failed the spec and green ones that met. The markers are sensitive to mouse hover and click events. The designer can toggle each circuit in and out of the table pane by clicking on the marker. Chosen circuits are encircled and numbered to match with their line at the table.

Circuits table (Figure C-2) – Each line in the table corresponds to a circuit the designer found interesting and selected from the graph. Circuits can be removed by clicking on the X button. One of the table circuits can become the pivot, on which heat-maps are centered and the schematic updates, by clicking on the magnifying glass on the right.

Select as Map Axis

Pop-up spec window

Mark as Relevant to Optimization

		XYZ Adc	XYZ CMRR
	Spec:	>20dB	
X	1	1.090dB	-5.926dB
X	14	7.277dB	-8.544dB
X	15	8.882dB	-8.882dB
X	18	20.275dB	-53.542dB

Figure C-2: Zoom-in on leftmost part of the circuits' table

Schematic (Figure C-3) - showing the topology and used to display data on the pivot circuit by hovering the mouse above nets and components. Hover events can display DC values and component sizes.

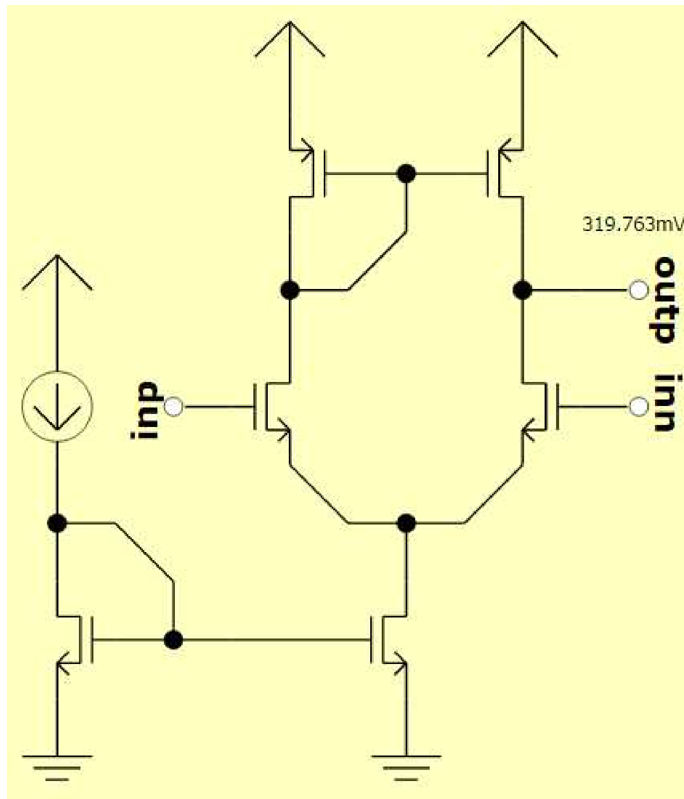


Figure C-3: Output DC level shown on Mouse-Hover Event

A search via heat-map produces variations on the pivot circuit. The field of the heat-map is dotted with red and green cursors (Figure C-4), for spec-failing and meeting circuits, respectively. A designer may want to switch focus to one of the other circuits by clicking on its

marker. This adds a new line to the circuits' table, without eliminating the previous pivot circuit.

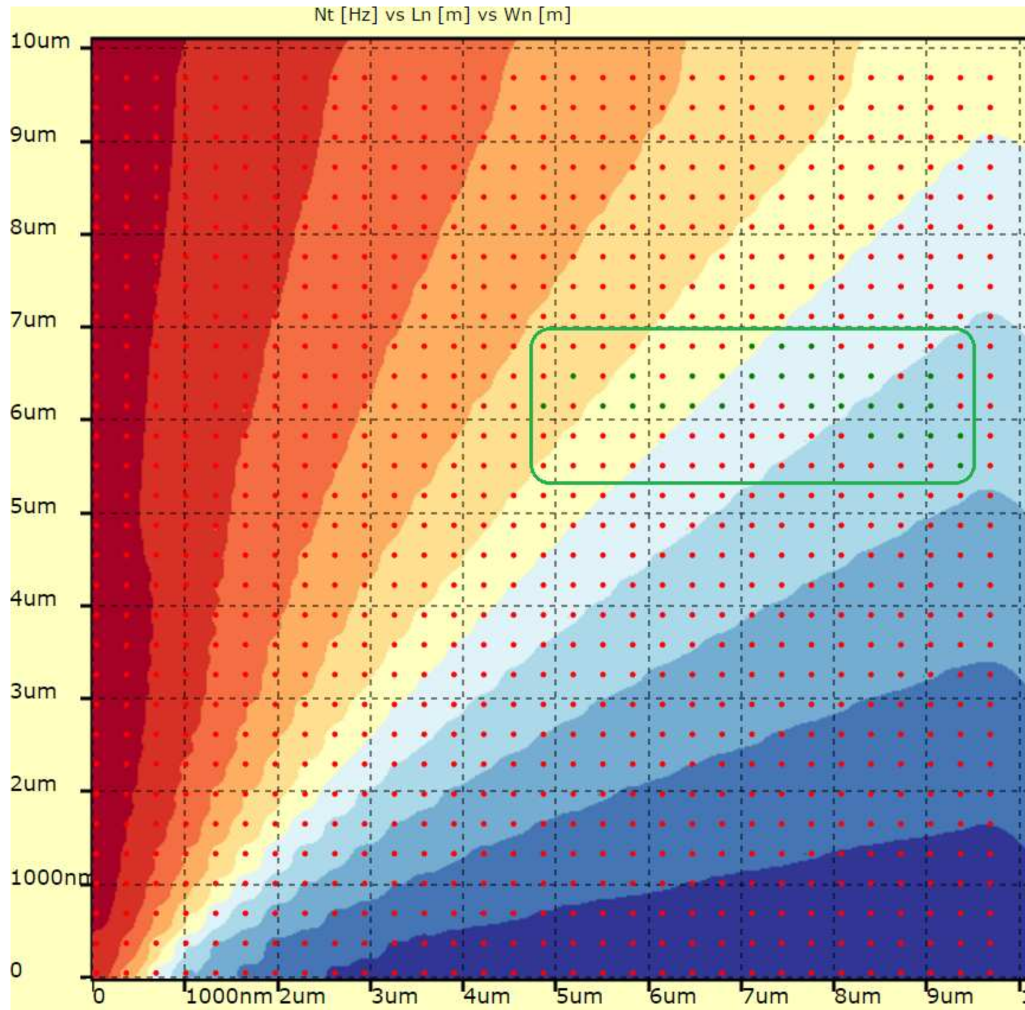


Figure C-4: Heat-map of Thermal Noise vs transistor size

Out of the circuits that were added to the table, any next circuit can be selected as pivot and navigation can continue via further heat-mapping performance vs. selected sizes.

Download - After all navigation through the solution space yields a circuit that meets the spec, a designer clicks on the cloud icon on the lower-left corner of the schematic pane. A "save as" pop-up appears and the system generates a ready to simulate SPICE netlist that can be further tweaked by external tools in a sign-off accuracy. The website generates the netlist, complete with

the transistor models and stimulus needed to run a simulation via NGSPICE and its commercial equivalent. The exact geometry bin, channel dimensions and finger multiplier are included with the netlist file.