1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file: Votre référence

Our file: Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Delay and Power Macro-Models for Optimizing ECL Circuits

by

David F. Skoll, B.Eng.

**A thesis submitted to the Faculty of Graduate Studies
and Research in partial fulfillment of the requirements
for the degree of**

Master of Engineering

Department of Electronics

**Carleton University
Ottawa, Ontario, Canada**

18 August 1994

© Copyright 1994 David F. Skoll

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-98587-9

Name DAVID F. SKOLL

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics Engineering

SUBJECT TERM

0544

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language
General 0679
Ancient 0289
Linguistics 0290
Modern 0291
Literature
General 0401
Classical 0294
Comparative 0295
Medieval 0297
Modern 0298
African 0316
American 0591
Asian 0305
Canadian (English) 0352
Canadian (French) 0355
English 0593
Germanic 0311
Latin American 0312
Middle Eastern 0315
Romance 0313
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion
General 0318
Biblical Studies 0321
Clergy 0319
History of 0320
Philosophy of 0322
Theology 0469

SOCIAL SCIENCES

American Studies 0323
Anthropology
Archaeology 0324
Cultural 0326
Physical 0327
Business Administration
General 0310
Accounting 0272
Banking 0770
Management 0454
Marketing 0338
Canadian Studies 0385
Economics
General 0501
Agricultural 0503
Commerce-Business 0505
Finance 0508
History 0509
Labor 0510
Theory 0511
Folklore 0358
Geography 0366
Gerontology 0351
History
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
General 0615
International Law and
Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
General 0626
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family
Studies 0628
Industrial and Labor
Relations 0629
Public and Social Welfare 0630
Social Structure and
Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture
General 0473
Agronomy 0285
Animal Culture and
Nutrition 0475
Animal Pathology 0476
Food Science and
Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology
General 0306
Anatomy 0287
Biostatistics 0308
Botany 0309
Cell 0379
Ecology 0329
Entomology 0353
Genetics 0369
Limnology 0793
Microbiology 0410
Molecular 0307
Neuroscience 0317
Oceanography 0416
Physiology 0433
Radiation 0821
Veterinary Science 0778
Zoology 0472
Biophysics
General 0786
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
General 0566
Audiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0564
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and
Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry
General 0485
Agricultural 0749
Analytical 0486
Biochemistry 0487
Inorganic 0488
Nuclear 0738
Organic 0490
Pharmaceutical 0491
Physical 0494
Polymer 0495
Radiation 0754
Mathematics 0405
Physics
General 0605
Acoustics 0986
Astronomy and
Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and
High Energy 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering
General 0537
Aerospace 0538
Agricultural 0539
Automotive 0540
Biomedical 0541
Chemical 0542
Civil 0543
Electronics and Electrical 0544
Heat and Thermodynamics 0348
Hydraulic 0545
Industrial 0546
Marine 0547
Materials Science 0794
Mechanical 0548
Metallurgy 0743
Mining 0551
Nuclear 0552
Packaging 0549
Petroleum 0765
Sanitary and Municipal 0554
System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Abstract

ECL circuits are used when high-performance digital processing is required in silicon. ECL circuits consume high power, and intense effort has been spent on trying to lower the power requirement while maintaining performance. This thesis describes timing and power macro-models based on circuit simulations, and explains how they can be used to optimize circuits. Software for automatically characterizing circuit blocks is described. This software uses an efficient numerical technique to simplify delay and power expressions while retaining accuracy. The macro-models can be used to optimize small or large circuits. Two circuits (an ECL inverter and a dual-modulus prescaler) are optimized. For the prescaler, circuit simulations predict a 22% reduction in power consumption due to optimization. Because macro-models are closed-form expressions, circuit optimization is very fast. Optimization times are hundreds of milliseconds for the inverter, and a few seconds for the prescaler. The simulation time spent generating the macro-models is consumed only once; careful modelling ensures that the same models can be used in multiple optimization runs or circuit contexts.

Acknowledgements

I would like to thank my supervisors, Prof. Martin C. Lefebvre and Prof. Miles A. Copeland, for their guidance, suggestions and support. Their advice has been instrumental in shaping this thesis.

I would like to thank the Natural Sciences and Engineering Research Council of Canada and the Department of Electronics, Carleton University, for their financial support. Northern Telecom Electronics provided fabrication facilities and device models used in this thesis. Some of the SUN SparcStations used for this work were lent to the Department of Electronics by the Canadian Microelectronics Corporation, which also provided the Cadence Edge design environment used to develop the test chip.

Prof. Calvin Plett, Neil Pickles, Tom Riley, John Long, and Luc Lussier contributed suggestions, assistance and technical know-how. Roger Shum provided inspiration for some of the macro-modelling software. Arthur Castonguay kept the computer systems running. Finally, thanks are due to my wife, Norine Hum, who put up with me and kept me going.

Table of Contents

Chapter 1: Introduction	1
1.1 Perspective	1
1.2 Objective	1
1.3 Contributions	1
1.4 Thesis Outline	2
Chapter 2: Related Work	4
2.1 Modification of Circuit Topology	4
2.1.1 Active Pull-Down Stages	5
2.2 Reduction of Supply Voltage and Voltage Swing	9
2.3 Circuit Optimization	11
2.3.1 Delay Expression Derivation	11
2.4 Summary	13
Chapter 3: Macro-Modelling	15
3.1 Introduction	15
3.2 Parameter Definition	15
3.3 Equation Generation	16
3.4 The Least-Squares Curve-Fitting Problem	16
3.4.1 Numerical Solution of the LS Problem	17
3.5 The Curve-Fitting Algorithm	18
3.5.1 Choice of Initial Basis Functions	18
3.5.2 Accuracy Specification	18
3.5.3 Importances of Basis Functions	19
3.6 Q-R Factorization Update	21
3.7 Additional Improvements to the Curve-Fitting Algorithm	21
3.8 Model Accuracy and Computation Time – Curve-Fitting	22
3.9 Model Accuracy and Computation Time – Optimization	23
3.10 Types of Optimization	23
3.11 Macro-Modelling Software Implementation	24
3.11.1 Parameter Definition and Pre-Processing	24
3.11.2 Circuit Simulation, Post-Processing and Curve-Fitting	24
3.12 Summary	25
Chapter 4: Inverter Macro-Model Development	26

4.1 Inverter Circuit for Modelling	26
4.2 Inverter Model Parameters.....	26
4.2.1 Parameter Ranges	27
4.2.2 Basis Functions.....	27
4.3 Inverter Simulation Results	28
4.4 Inverter Model Accuracy	30
4.5 Inverter Optimization.....	30
4.5.1 Current Minimization with Delay Constraint.....	31
4.5.2 Delay Minimization with Current Constraint.....	32
4.5.3 Design-Space Exploration.....	34
4.6 Model Generation CPU Time	37
4.7 Optimization CPU Time	38
4.8 Summary.....	39
 Chapter 5: Dual-Modulus Prescaler Circuit.....	40
5.1 Motivation.....	40
5.1.1 Typical Application of the Prescaler	40
5.2 Circuit Description.....	42
5.3 Circuit Operation	43
5.4 Circuit Diagram	45
5.4.1 Bias Voltage Generation	46
5.4.2 Four-Phase Clock Generation	48
5.4.3 4-to-1 MUX.....	49
5.4.4 AND Gate.....	50
5.4.5 ECL Input Buffer.....	50
5.4.6 Master-Slave D Flip-Flop.....	51
5.4.7 Johnson Counter	52
5.5 Test Chip.....	53
5.5.1 Layout Issues.....	53
5.6 Measurements	57
5.6.1 Scope Traces.....	58
5.7 Summary	60
 Chapter 6: Optimization of the Dual-Modulus Prescaler	61
6.1 Macro-Model Development.....	61
6.2 ECL AND Gate Models.....	63
6.2.1 AND Gate Delay Model.....	63
6.2.2 AND Gate Current Model	65
6.3 Emitter-Follower Models.....	66
6.3.1 Emitter-Follower Delay Model	67
6.3.2 Emitter-Follower Current Model.....	68
6.4 Remaining Models.....	70
6.5 Constraint Generation	72

6.5.1 Timing Path 1	73
6.5.2 Timing Path 2	74
6.5.3 Timing Path 3	75
6.5.4 MUX Phase Relationship	76
6.6 Objective Function.....	76
6.7 Prescaler Optimization.....	77
6.7.1 Optimization for 1GHz Operation.....	77
6.7.2 Multiple Optimization Runs - Frequency Specified.....	79
6.7.3 Multiple Optimization Runs - Total Current Specified.....	80
6.8 Non-Linear Optimization Issues.....	81
6.9 Summary	82
 Chapter 7: Conclusions and Future Work.....	83
7.1 Summary.....	83
7.2 Contributions	83
7.3 Future Work	84
7.3.1 Range and Domain Matching.....	84
7.3.2 Definition of Loading	84
7.3.3 Model Complexity.....	85
7.3.4 CAD Issues	85
 Glossary	86
 Appendix A: HPP Manual	88
 Appendix B: Sample HPP Output	95
B.1 GAMS Output	95
B.2 ANSI C Output.....	96
B.3 HPP Equation File Format	98
 Appendix C: Sample Curve-fitting Code.....	100
 References.....	104

List of Figures

Figure 1. Differential ECL Inverter	5
Figure 2. AC-Coupled Active Pull-Down Stage	6
Figure 3. Cross-coupled Active Pull-Down	7
Figure 4. P-channel JFET Realization	8
Figure 5. JFET Active Pull-Down Circuit	9
Figure 6. LSG D-Latch	10
Figure 7. ECL Inverter with Load Capacitances	26
Figure 8. Inverter Delay – Simulation Results	29
Figure 9. Inverter Macro-Model Error Distribution	30
Figure 10. Inverter Optimization - Delay Constraint = 150ps	31
Figure 11. Inverter Optimization – Delay Constraint = 100ps	32
Figure 12. Inverter Delay Minimization with Current Constraint	33
Figure 13. Inverter Design-Space Exploration – Simulated Delay	34
Figure 14. Inverter Design-Space Exploration – Predicted Delay	35
Figure 15. Inverter Design-Space Exploration – Error Distribution	36
Figure 16. Inverter Design-Space Exploration – Alternate View	37
Figure 17. Frequency Synthesizer with Programmable Divider	41
Figure 18. Dual-Modulus Prescaler - Block Diagram	42
Figure 19. 4-Phase Counter Timing Diagram	42
Figure 20. Dual-Modulus Prescaler Timing Diagram - 2.5 Mode	44
Figure 21. Dual Modulus Prescaler - Schematic Diagram	45
Figure 22. Bandgap Reference - Schematic Diagram	46
Figure 23. Bandgap Voltage vs. Temperature	47
Figure 24. ECL Latch - Schematic Diagram	48
Figure 25. 4-to-1 MUX - Schematic Diagram	49
Figure 26. AND Gate - Schematic Diagram	50
Figure 27. ECL Input Buffer - Schematic Diagram	51
Figure 28. Master-Slave D Flip-Flop	51
Figure 29. Phase-select Johnson Counter	52
Figure 30. AND Gate Layout	54

Figure 31. Bandgap Reference Generator Layout	55
Figure 32. MUX Layout	56
Figure 33. Timing Diagram with Latch after Phase-Select Counter	57
Figure 34. 1-GHz Input Signal, Divide-By-2.5 Mode	58
Figure 35. 2-GHz Input Signal, Divide-By-2.5 Mode	59
Figure 36. 200-MHz Input Signal Showing Modulus Switch	59
Figure 37. Dual-Modulus Prescaler Block Diagram showing Optimized Portion	61
Figure 38. ECL AND Gate Delay Model	64
Figure 39. ECL And Gate Current Model	65
Figure 40. Emitter-Follower Schematic.....	66
Figure 41. Emitter-Follower Delay Model	67
Figure 42. Emitter-Follower Current Model.....	68
Figure 43. Timing Path #1	73
Figure 44. Timing Path #2	74
Figure 45. Timing Path #3	75
Figure 46. Prescaler Simulation Waveforms	77

List of Tables

Table 1. Inverter Optimization Results	38
Table 2. BATMOS Bipolar Parameters.....	53
Table 3. Summary of Dual-Modulus Prescaler Macro-Models	70
Table 4. Interpretation of Model Parameters.....	71
Table 5. Subscripts Identifying Circuit Elements.....	72
Table 6. Resistor Values Chosen by GAMS for 1GHz Operation	78
Table 7. Supply Currents for Various Prescaler Frequencies.....	79
Table 8. Frequency Optimized for Various Supply Currents.....	80

Chapter 1: Introduction

1.1 Perspective

For digital circuits operating at gigahertz frequencies, bipolar Emitter-Coupled Logic (ECL) is often the logic family of choice. ECL circuits form the fastest digital logic family, with gate delays measured in tens of picoseconds. In BiCMOS processes, ECL circuitry can be used in high-speed sections of a circuit, while CMOS circuitry can be used in the low-speed sections.

Although ECL circuitry is fast, it consumes large amounts of power. This high power consumption is the major disadvantage of ECL, and many people have investigated ways to reduce the power consumption of ECL circuitry.

1.2 Objective

The objective of this thesis is to examine methods by which ECL power consumption can be reduced. Some general power minimization techniques will be examined, but the main objective is to present a timing macro-model for ECL circuits, and use the model to optimize an ECL circuit. Simulation results will be used to compare the original circuit design with the optimized design, and to evaluate the macro-model optimization process.

1.3 Contributions

This thesis will demonstrate several advantages of macro-modelling over semi-analytical or ad hoc simulation-and-optimization techniques:

1. Macro-modelling is based on circuit simulation, and is therefore very accurate. As improvements are made in simulation and device modelling, these improvements are immediately reflected in macro-model generation.

2. Macro-modelling relies on simulation of small circuit blocks, and is therefore not very sensitive to the size of the entire circuit being optimized. Careful modelling will yield models usable in many circuits without the need for re-simulation.
3. Optimization using macro-models is very quick, since closed-form expressions describing circuit behaviour are available.
4. As improvements are made in non-linear optimization techniques, these improvements become available immediately to the designer.

The thesis will describe a method for generating macro-models based on circuit simulation, and describe software designed to facilitate the modelling process. It will demonstrate modelling and optimization of a small circuit (an ECL inverter) and a larger circuit (a dual-modulus prescaler.)

1.4 Thesis Outline

Chapter 2 contains a discussion of related work in the areas of power reduction, circuit optimization and delay modelling.

Chapter 3 presents the theory behind macro-modelling. The issues in macro-model development are discussed, including parameter definition, equation generation, and control of model accuracy. The curve-fitting algorithm using fast Q-R refactorization is described. The impact of model accuracy on equation generation time and optimization time is discussed. Finally, there is a brief discussion of the macro-modelling software implementation.

Chapter 4 describes the macro-modelling of an ECL inverter. This shows the range of problems for which macro-modelling is applicable, and demonstrates the operation of the macro-modelling and optimization software.

Chapter 5 contains a description of the 2/2.5 Dual-Modulus Prescaler circuit. This circuit is an example of an ECL application, designed to operate at 1GHz. It contains two

Johnson counters, two flip-flops, two 4-to-1 multiplexers, and a two-input AND gate. These blocks provide a fairly wide design space for optimization purposes. The circuit was designed using 0.8 μ m-wide polysilicon emitter bipolar transistors in the Northern Telecom BATMOS BiCMOS process. The process also features polysilicon resistors, three levels of metallization, and high-performance CMOS transistors. Minimum-geometry bipolar transistors have a unity-gain frequency F_T of 11 GHz.[1]

Chapter 6 describes the macro-modelling and optimization of the dual-modulus prescaler. Simulation results of optimized versions of the circuit are compared with the original simulation results.

Finally, Chapter 7 presents some concluding thoughts, and recommendations for further work in this area.

Chapter 2: Related Work

This chapter describes related work in power minimization and modelling of ECL circuits. The approaches can be divided into three general areas: Modification of circuit topology (the use of novel circuits and/or devices); reduction of power supply voltage and voltage swing; and optimization of circuit parameters given a particular circuit.

2.1 Modification of Circuit Topology

The first area of power minimization research involves novel circuit topologies. These are usually designed to reduce the standby current of the circuit.

During the discussion of these techniques, it will be useful to refer to Figure 1 on page 5, which is the schematic diagram of the basic ECL differential inverter. In a variation of the circuit shown below, the emitter-followers are sometimes loaded by resistors terminated at V_T (which may or may not be the same as V_{EE}) rather than current sources. The current sources are typically realized by a transistor with an emitter resistor to V_{EE} and a reference voltage applied to the base. Finally, some ECL circuits are shown with V_{CC} connected to ground and V_{EE} as a negative voltage; this is done for noise-immunity reasons and does not otherwise affect circuit operation.

Figure 1. Differential ECL Inverter

2.1.1 Active Pull-Down Stages

One way to reduce the power consumption without adversely affecting the speed of the circuit is to replace the current sources I_{EF} with active pull-down stages, which supply increased current during switching transients, but minimal current while the circuit is quiescent. This reduces the average current, and hence power consumption, of the circuit.

Active pull-down stages may consist of AC-coupled devices [3, 4], cross-coupled devices [5], or novel devices using unusual design rules [6]. These techniques are now reviewed.

One type of AC-coupled active pull-down stage [4] is shown in Figure 2 on page 6. A similar stage would be used for OUTB.

Figure 2. AC-Coupled Active Pull-Down Stage

During the quiescent state, the DC current I_o in the output stage is given approximately by

$$I_o = \frac{V_{OUT} - V_{D1} - V_{BEQ4}}{R_E} \quad (1)$$

I_o is dependent upon V_{OUT} , but is higher when $V_{OUT} = V_{OH}$, making it easier to turn on Q_4 when OUT must be pulled low. (R_{BB} is a large resistor which maintains a small bias current through D_1 .)

During a low-going transient on the output, node A goes from low to high, coupling a rising transient to the base of Q_4 through C_x . Because the speedup capacitor C_E prevents the voltage at the emitter of Q_4 from changing quickly, V_{BEQ4} increases, and Q_4 is turned on quickly. This draws a large current from the output, pulling OUT low quickly. When OUT goes below V_{OL} , Q_3 turns on briefly, and charges C_E through D_1 and the base-emitter junction of Q_4 . This turns off Q_4 , and the pull-down action is thus self-terminating.

When the output goes high, node A goes low, and a negative-going transient is coupled to the base of Q_4 , turning it off. The output is then quickly charged to V_{OH} by Q_3 . Because

Q_4 is still off, there is a slight overshoot in the output due to lower V_{BEQ3} because of the reduced I_o . When Q_4 turns on again, V_{OUT} returns to the nominal V_{OH} level.

One problem with AC-coupled active pull-down stages is the requirement of two capacitors per pull-down stage. C_x can be quite small, and is easily realized with a reverse-biased junction capacitor. However, C_E should have about the same capacitance as the load capacitance for optimum performance, and this can be fairly high.

Alternate AC-coupled active pull-down circuits which do not require C_E exist [3, 4], but they require either substantially more components, or Schottky-clamped devices, which are not available in most BiCMOS technologies.

Another active pull-down scheme [5] uses cross-coupled pull-down devices, eliminating the need for coupling capacitors.

Figure 3. Cross-coupled Active Pull-Down

Resistors R_1 and R_2 set the steady-state current levels in the emitter-follower branches. Capacitors C_1 and C_2 are speed-up capacitors. These capacitors may be omitted, but the pull-down devices provide larger transient current capability if the capacitors are included.

R_1 , R_2 and V_T are chosen so that when OUT is high and OUTB is low, Q_5 is biased at "cut-in" (barely on with very little current.) On the other hand, the high voltage at OUT keeps Q_6 on, which draws steady-state current in the OUTB branch. When the circuit state changes, the voltage at OUTB rises, and quickly turns on Q_5 . This draws a large current from OUT, quickly pulling it down. In addition, this turns off Q_6 and helps speed up the rising transient at the OUTB node, since Q_6 no longer tries to pull OUTB down.

In the cross-coupled circuit, only one emitter-follower branch has current flowing through it in the steady-state – the high branch only carries the base current for the opposite pull-down device, which is very small. In conventional ECL circuits, both branches have steady-state current.

One final active pull-down scheme [6] uses an unusual approach of violating the normal design rules of a bipolar process. In this scheme, the emitter stripe of an npn transistor is elongated to cut the base region in half, making a p-channel JFET.

Figure 4. P-channel JFET Realization

The p-channel JFET is used as a load device on the emitter-followers, and acts as a voltage-controlled current-source.

Figure 5. JFET Active Pull-Down Circuit

When the output is high, node “X” is one V_{BE} drop higher than node “OUT”. If V_{EF} is low enough, the JFET operates in the saturation region, and the steady-state current is almost independent of the output state. During a falling transient, the voltage at node “X” falls before the output voltage can follow. V_{GS} of the JFET decreases correspondingly, and the drain current of the JFET increases, pulling down the output faster. During a rising transient, “X” increases faster than the output, increasing V_{GS} and reducing the drain current through the JFET.

This technique has limited application, because large loads require large JFETs, which place a heavy capacitive load on node “X”. More complicated circuits [6] can reduce this problem. More importantly, however, most bipolar and BiCMOS processes are not optimized for the production of JFETs, and the JFET characteristics may be poor or poorly-controlled.

2.2 Reduction of Supply Voltage and Voltage Swing

One simple way to reduce the power consumption of ECL circuits is to reduce the supply voltage [7] while keeping the current levels constant. This does not affect circuit speed,

In the latch shown above, Q_5 and Q_6 function both as switching elements and current sources. The enable inputs, EN and ENB, are driven by special voltage-compensated buffers to ensure that the high level at the inputs tracks variations in V_{EE} . This ensures that the current level in Q_5 or Q_6 during the on state is independent of V_{EE} , and that Q_5 and Q_6 will never be driven into deep saturation.

2.3 Circuit Optimization

Once a circuit topology is picked, its power consumption can be reduced through the careful choice of component values and current levels. This type of optimization is the main thrust of the thesis, and the topic of much previous work [10, 11, 12].

2.3.1 Delay Expression Derivation

One approach to optimizing circuit performance is to derive an expression for propagation delay whose terms involve circuit parameters, and then use normal numerical optimization techniques to pick circuit parameters. Chor, Brunnschweiler and Ashburn [9] assume that the propagation delay of an ECL gate can be expressed approximately as a linear combination of various RC time constants and transit times in the circuit, as proposed in [13, 14]. The propagation delay equation is:

$$\tau_d = \sum_i \sum_j K_{ij} R_i C_j + \sum_m K_m \tau_m \quad (2)$$

where K_{ij} and K_m are weighting factors of the various RC terms and transit-time terms, respectively. From Eq. 2, we see that:

$$K_{ij} = \frac{\partial^2 \tau_d}{\partial R_i \partial C_j} \quad (3)$$

$$K_m = \frac{\partial \tau_d}{\partial \tau_m} \quad (4)$$

By varying the R , C and τ parameters slightly, the derivatives can be approximated and the K factors determined from circuit simulation.

Once the delay expression is known, the various resistances and capacitances can be related to actual resistances and capacitances in the circuit, or to parasitic resistances and capacitances in the transistor model. This information can be used to select current levels and resistance values for optimum performance.

Fang [10] uses a similar approach, but made the insightful observation that "it is seen in the literature that different people obtain different expressions under different assumptions..." He then performed a "linearity study" by varying circuit and device parameters over a wide range, and checking that the propagation delay varied linearly for each parameter. This analysis is flawed, however – to truly settle the question of linearity, all possible combinations of parameters must be varied and the linearity examined.

Tien [11] also obtains a linear delay expression for the ECL gate, but uses an "analytical" approach based on the small-signal transistor models, rather than sensitivity analysis and circuit simulation.

It would be quite remarkable if the behaviour of a non-linear circuit could be described in terms of linear RC time constants. Recognizing the limitations of this approach, Hsieh, Chin and Chuang [12] developed an optimization procedure which uses circuit simulation as its key element. The idea is to determine the optimal current partition and emitter sizes for an ECL inverter. The authors consider the three parameters, η , J_{CS} and J_{EF} , where η is defined as the power in the current-switch divided by the power in the emitter-follower circuit, and J_{CS} and J_{EF} are the current densities in the current-switch and emitter-follower transistors, respectively. With those parameters, the time delay can be expressed as:

$$t_d = f(\eta, J_{CS}, J_{EF}) \quad (5)$$

In [12], Eq. 5 is approximated by a quadratic surface for simplicity:

$$f(x_1, x_2, x_3) \equiv f(x_1^0, x_2^0, x_3^0) + \sum_{i=1}^3 \frac{\partial f}{\partial x_i} (x_i - x_i^0) + \frac{1}{2!} \sum_{i=1}^3 \frac{\partial^2 f}{\partial x_i^2} (x_i - x_i^0)^2 \text{ with } |x_i - x_i^0| < 1 \quad (6)$$

Eq. 6 is a close approximation to the true surface for points close to the nominal point at which the derivatives are first computed. The accuracy falls off further from the nominal point.

The procedure used by Hsieh et al. is as follows:

- 1. Select a nominal point on the surface*
- 2. Approximate the partial derivatives in Eq. 6 with several circuit simulations, perturbing the parameters around the nominal point.*
- 3. Find the optimal point on the approximate surface.*
- 4. If the optimal point from (3) is not sufficiently close to the nominal point, set the nominal point to the optimal point, and go to step (2).*

This algorithm repeatedly computes approximate surfaces to the real design surface, ultimately converging (it is hoped) to the optimal point on the real design surface. When the approximate surfaces converge to the optimal surface, the partial derivatives found in step (2) will be small, and the optimal point from (3) will be close to the previous guess, causing termination in step (4).

This procedure is very accurate, since circuit simulations are done at each step. However, although the procedure will converge to a local minimum, there is no guarantee that it will converge to the global minimum. Also, several circuit simulations are required to compute the partial derivatives at each iteration. For a small circuit, this is not a problem, but for large circuits, the simulation time could be excessive, especially if a large number of circuit parameters are involved.

2.4 Summary

Many of the topology-modification techniques described in Section 2.1 can be used to good effect. However, the degree of improvement offered by the techniques is quite technology-dependent, especially for esoteric techniques like the JFET pull-down.

Reduction of supply voltage and voltage swing is a good technique for reducing power in ECL circuits. However, at very low voltages, novel circuits are needed to reduce the number of series elements between the rails.

Finally, the work described in Section 2.3 invariably concentrated on simple circuits, such as inverters or buffers. There is no clear way to extend the work to larger circuits. Even the simulation method described in [12] cannot easily be extended to large circuits.

Chapter 3: Macro-Modelling

3.1 Introduction

A macro-model is a “black box” view of a circuit. The circuit is modeled as a functional block which relates independent variables to dependent variables. The dependent variables of interest may be propagation delay τ_D , power consumption P or output transition time τ_{TO} . The independent variables may be transistor sizes, current levels and load capacitances. The generic macro-model consists of the set $MM = \{ \{V_i, C_i\}, \{F_j\} \}$, where the set $\{V_i, C_i\}$ contains all the independent variables and their constraints (typically, the independent variables are constrained to the range over which the circuit was simulated). The set $\{F_j\}$ is the set of equations describing the output variables in terms of the input variables. These equations are derived by curve-fitting basis functions to simulation data.

The macro-modelling procedure is essentially that described in [21]. However, the curve-fitting algorithm has been modified to use a Q-R refactorization procedure rather than recomputing a Q-R decomposition from scratch at each step. This significantly speeds up the modelling process.

3.2 Parameter Definition

A crucial factor in the success of a macro-model is the correct choice of independent and dependent variables to model. Enough independent variables must be chosen to provide accuracy over a wide range of conditions, and dependent variables must be chosen so that macro-models can cascade correctly.

For example, consider an ECL inverter as shown in Figure 1 on page 5. The two dependent variables of most interest are the propagation delay τ_D and the power consumption P . The obvious independent variables are the current levels I_{CS} and I_{EF} , and the transistor

sizes S_{CS} and S_{EF} . (These sizes may be emitter areas, or it may be necessary to specify the emitter length and width separately.) The loading will also have an effect on the propagation delay, and the load C_L must be included in the model. In addition, the transition time of the input could affect the delay, and must be included. Finally, to cascade models correctly, the output transition time and input capacitance of the inverter must be measured and modeled. (In fact, the loading of a differential pair may require a more complicated model than a simple load capacitance.)

3.3 Equation Generation

The equations $\{F_j\}$ in the macro-model are generated by curve-fitting basis functions to simulation data. The basis functions which the macro-modelling software uses are products of powers of the independent variables. The exponents must be chosen experimentally (and are fixed.) However, the curve-fitting software aids in this choice by eliminating terms until the accuracy drops below a specified point. Thus, it is acceptable to choose many possible powers, and allow the software to eliminate those powers which do not correspond closely to the shape of the simulation surface. Because the exponents in the basis functions are fixed, linear curve-fitting can be used.

The curve-fitting software automatically generates HSPICE input files, varying independent variables over their ranges. Since each variable is swept individually, several variables can result in a large number of simulation points. The software executes HSPICE, reads the results, performs the curve-fitting, and outputs the macro-models in a number of programming languages. The equations can then be used by non-linear optimization software to optimize the circuit.

3.4 The Least-Squares Curve-Fitting Problem

Given a set of n basis functions $\{f_n(\hat{x})\}$, a set of $m > n$ independent parameter vectors $\{\hat{x}_m\}$, and a vector of m observed dependent variables $\hat{b}$, the least-squares curve-fitting problem (LS problem) consists of finding a vector $\hat{c}$ of n coefficients such that

$$\rho_{LS}^2 = \|A\hat{c} - \hat{b}\|_2^2 \quad (7)$$

is minimized, where $A \in \mathfrak{R}^{m \times n}$ consists of the basis functions $\{f_n(\hat{x})\}$ (arranged in columns) evaluated at each data point in $\{\hat{x}_m\}$ (arranged in rows.) The solution which minimizes ρ_{LS}^2 is called $\hat{c}_{LS}$.

In the context of macro-modelling, each vector in $\{\hat{x}_m\}$ is one set of values of circuit parameters. For example, in an ECL inverter, we may be interested in varying the current in the differential pair, the current in the emitter followers, and the capacitive load. Each $\hat{x}_i$ would be a point in the design space described by those parameters.

Choosing the basis functions is a major issue in the LS problem. Ideally, the basis functions should approximate the real behaviour of the observed variable. They should also be fairly "well-behaved" so that they can be considered reliable for interpolating between observed points. Choosing the number of basis functions is another major issue; the algorithm presented in section 3.5 addresses this.

3.4.1 Numerical Solution of the LS Problem

The LS problem is attacked by converting it into an equivalent problem by premultiplying A and $\hat{b}$ by an orthogonal matrix $Q \in \mathfrak{R}^{m \times m}$ such that:

$$Q^T A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix} \quad (8)$$

is upper triangular. If

$$Q^T \hat{b} = \begin{bmatrix} \hat{e} \\ \hat{f} \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix} \quad (9)$$

then

$$\|A\hat{c} - \hat{b}\|_2^2 = \|Q^T A\hat{c} - Q^T \hat{b}\|_2^2 = \|R_1 \hat{c} - \hat{e}\|_2^2 + \|\hat{f}\|_2^2 \quad (10)$$

for any $\hat{c} \in \mathcal{R}^n$. If $\text{rank}(A) = \text{rank}(R_1) = n$, then $\hat{c}_{LS}$ is defined by the solution to $R_1 \hat{c}_{LS} = \hat{e}$.

The full-rank LS problem can be solved once we determine Eq. 8, which is called the *Q-R factorization* of A . The Q-R factorization is performed using Householder transformations with column pivoting, as described in [15] and [16].

3.5 The Curve-Fitting Algorithm

The curve-fitting algorithm uses the Q-R factorization as a subroutine to perform curve-fitting. It eliminates basis functions one at a time until the accuracy of the curve-fit drops below a specified threshold.

3.5.1 Choice of Initial Basis Functions

For this implementation, the basis functions are products of small powers of the independent parameters. The basis function powers are user-specified; they typically range from about -2 to 2, and may include square-roots and reciprocal square-roots. For each independent parameter, the user specifies a list of powers to use. For example, if the independent parameters are I_{DP} and I_{EF} , and the user specifies $\{-2, -1, 0, 0.5, 1\}$ as the list of powers for I_{DP} and $\{-1, 0, 1\}$ for I_{EF} , then there are 15 basis functions, consisting of the various combinations from $I_{DP}^{-2}I_{EF}^{-1}$ through $I_{DP}I_{EF}$.

The basis functions are reasonably well-behaved - they are smooth (except for the reciprocals at singular points) and do not oscillate if the powers are low.

3.5.2 Accuracy Specification

The most obvious measure of accuracy of a curve-fit is the value ρ_{LS}^2 . However, this is not suitable for our purposes, since it measures the average deviation of the curve-fit approximation from the simulated results. It says nothing about the distribution of the errors.

The accuracy measure we use is specified by two parameters: *acc* and *tol*. The "tolerance" of a data point is a measure of the deviation from the simulated value. For example, if a data point has a tolerance of 0.1, then the curve-fit approximation is within 10 percent of the simulated value at that data point.

Given a specified *tol*, *acc* is the fraction of all data points whose tolerance is less than or equal to *tol*. Thus, if *tol* is 0.1 and *acc* is 0.9, then we are sure that for 90 percent of the data points, the curve-fit approximation is within 10 percent of the simulated value. Knowledge of process variation assists in the selection of *acc* and *tol*.

3.5.3 Importances of Basis Functions

For a circuit in which several design parameters are varied, the number of basis functions can be quite large. If there are q parameters, and each parameter p_i has e_i exponents in its exponent list, then the number of basis functions is $n = \prod_{i=1}^q e_i$. For three parameters with six exponents each, this works out to 216. It is desirable to reduce the number of terms to the minimum required to attain the specified accuracy. This speeds up the optimization procedure, and is likely to provide a better-behaved curve-fit approximation - i.e., one that does not suffer from wild swings between measured points.

At a given data point $\hat{x}_i$, the *contribution* of each basis function $f_j(\hat{x}_i)$ is defined as

$$\eta_{ij} = \frac{|c_j f_j(\hat{x}_i)|}{\sum_{k=1}^n |c_k f_k(\hat{x}_i)|} \quad (11)$$

where c_j is the j 'th component of $\hat{c}_{LS}$. Thus, the contribution of a basis function at a given data point is a measure of its weight in the curve-fit approximation at that data point.

Given a specified contribution level η_{spec} , we define the *importance* of a basis function $f_j(\hat{x})$ with respect to η_{spec} as one of the following:

$$\lambda_j = \sum_{i=1}^m \delta_{ij}, \text{ where } \delta_{ij} = \begin{cases} 1 & \text{if } \eta_{ij} \geq \eta_{spec} \\ 0 & \text{otherwise.} \end{cases} \quad (12)$$

$$\lambda_j = \frac{1}{m} \sum_{i=1}^m \eta_{ij} \quad (13)$$

$$\lambda_j = \max_{i=1}^m \eta_{ij} \quad (14)$$

Equation 12 is used to eliminate terms using the *importance* method. Equation 13 is called the *average contribution* method, and Equation 14 is the *maximum contribution* method.

With all of those definitions in place, we now describe the algorithm for reducing the number of basis functions given a specified accuracy:

Algorithm 1: Reduce-Basis-Functions

Input: a specified tolerance tol_{spec} and accuracy acc_{spec} ; a set of n linearly-independent basis functions; a set of $m \geq n$ data points.

Output: A set of $q \leq n$ basis functions; a vector of q coefficients for the basis functions.

run curve-fit with all basis functions and calculate acc with respect to tol_{spec} ;

while ($acc \geq acc_{spec}$) do

n = number of basis functions remaining;

$\eta_{spec} = 1/n$;

 for $j = 1$ to n do

 calculate λ_j with respect to η_{spec} according to one of Eq. 12, 13 or 14;

 endfor

 delete the i 'th basis function where $\lambda_i = \min(\lambda_j)$, for $j = 1$ to n ; (DELETION)

 run curve-fit and calculate new acc ;

 while ($acc < acc_{specified}$) do

 restore the previously-deleted basis function;

 if (all basis functions have been tried) then

 return results of most recent curve-fit where $acc \geq acc_{specified}$;

 else

 delete the basis function with the next larger λ_i ;

 endif

 re-run curve-fit and calculate new acc ;

 endwhile

endwhile

This algorithm is the same algorithm as described by Shum and Lefebvre [21]. However, the curve-fit subroutine has been modified so that after a DELETION step, the Q-R factorization can be computed incrementally from the previous Q-R factorization, rather than starting from scratch.

3.6 Q-R Factorization Update

To calculate the Q-R factorization of $A \in \mathbb{R}^{m \times n}$ takes time $O(mn^2)$ using Householder decomposition with column pivoting. However, to update the Q-R factorization after a column of A has been deleted takes time $O(n^2)$. Since we always have $m > n$ for this LS problem, the time savings are significant.

Suppose that $\bar{A} = [\bar{a}_1, \dots, \bar{a}_{k-1}, \bar{a}_{k+1}, \dots, \bar{a}_n] \in \mathbb{R}^{m \times (n-1)}$ which is just A with its k 'th column deleted. If Q and R are the Q-R factorization of A , then

$$Q^T \bar{A} = \begin{bmatrix} R_{11} & R_{12} \\ 0 & \bar{w}^T \\ 0 & R_{33} \end{bmatrix} = H \quad (15)$$

is upper Hessenberg. The unwanted subdiagonal elements can be zeroed with a sequence of Givens rotations [15, 16] $J_{n-1}^T \dots J_k^T H = R_1$ and R_1 is upper-triangular. If we accumulate $Q_1 = Q J_k \dots J_{n-1}$, then $\bar{A} = Q_1 R_1$ is the Q-R factorization of $\bar{A}$.

3.7 Additional Improvements to the Curve-Fitting Algorithm

In addition to the Q-R factorization update, we can improve performance by noting that the matrix Q need not be computed explicitly. Instead, we only need to keep track of its effect on $\hat{b}$ according to Eq. 9. The loops in the Q-R factorization and Q-R update procedures which would normally accumulate Q instead accumulate its effects on $\hat{b}$. This does not change the asymptotic running times, but does significantly lower the hidden constants associated with the big- O notation.

However, there is a price to pay for this optimization. There exists a procedure to update the Q-R factorization after a column is added to A in time $O(mn)$. [15] This would be useful whenever a column which has been deleted must be restored. However, the procedure requires the matrix Q . In the current implementation, the Q-R factorization is simply recomputed whenever a column is restored. Although it takes time $O(mn^2)$, in practice the matrices are much smaller than their original size by the time it becomes necessary to restore deleted columns. Alternatively, we could keep copies of the Q-R factorization prior to deleting a column, and revert to them when a column is restored. This would take time $O(mn)$, but would consume $O(mn)$ additional memory. Copying the matrices prior to deleting columns would also increase the asymptotic time of the DELETION step from $O(n^2)$ to $O(mn)$.

3.8 Model Accuracy and Computation Time – Curve-Fitting

In general, the more accurate the model equations are, the more terms they have. Reducing tol towards 0 or increasing acc towards 1 increases the number of terms in the final equation. Because fewer DELETION steps are performed, the curve-fitting process is generally quicker for small values of tol and large values of acc . However, this is not always the case: if the equation has many terms left when the accuracy drops below the acceptable level, the term-elimination algorithm may perform many full Q-R factorizations as it searches fruitlessly for a term to discard. The full factorization is significantly slower than the Q-R refactorization after DELETION.

In summary, the computation time of the curve-fitting process is a complex function of the number of data points, the number of basis functions, the value of tol , and the value of acc . Experimental evidence shows that the time spent curve-fitting is usually less than (and often far less than) the circuit simulation time.

3.9 Model Accuracy and Computation Time – Optimization

The optimization procedures evaluate the model functions in closed-form. Thus, the computation time is directly proportional to the number of terms in the model function. The function evaluation is very quick; thus, only for very large circuits should function evaluation time be a concern. In general, it is better to have as accurate models as possible than to worry about computation time during optimization.

3.10 Types of Optimization

Once macro-models of circuit blocks are available, various types of optimization can be performed easily.

In the first type of optimization, timing constraints on the circuit are given, and power consumption should be minimized while respecting the constraints. This is probably the most common form of optimization. To perform this optimization, the timing equations are included in constraints, and the power consumption is specified as the objective function for the optimization software.

In the inverse of the power optimization, we are given a maximum allowable power consumption, and should optimize the circuit to be as fast as possible while obeying the power constraints.

Finally, we may wish to perform a design-space exploration, which consists of multiple optimization runs. For example, we may wish to plot the minimum propagation delay of a circuit for various values of the load capacitances, along with the power required to achieve the delay. This gives an overall picture of the circuit behaviour, and allows power-delay trade-offs to be visualized easily. Alternatively, several curves could be plotted showing the power required to meet various delay specifications at various loads.

3.11 Macro-Modelling Software Implementation

The macro-modelling algorithm described above was implemented in C on a Sun Sparc10 workstation. The code is about 7600 lines long. The macro-modeler, named HPP, was implemented as a pre- and post-processor for the HSPICE circuit simulator. [24]

The macro-modelling consists of three phases:

1. Parameter definition and pre-processing
2. Circuit Simulation
3. Post-processing and curve-fitting

3.11.1 Parameter Definition and Pre-Processing

HPP reads a control file from disk. This file specifies the independent parameters, and the values for each parameter at which the circuit is to be simulated. It then writes out an HSPICE input file with all combinations of parameter values.

3.11.2 Circuit Simulation, Post-Processing and Curve-Fitting

Once HPP has created an HSPICE file, it invokes HSPICE, unless data exists from a previous run. This eases file maintenance, and reduces the chance of unnecessary simulations.

After HSPICE completes, HPP reads the results. It checks each parameter against bounds specified in the HPP control file, and flags those values which are out-of-bounds. This helps the designer to detect problems in the circuit simulation, such as measure statements which did not trigger properly, or a combination of parameter values which causes circuit operation to fail. The bounds checking has proven to be an invaluable sanity check.

HPP then executes curve-fit commands as specified in the control file, and writes the results out in a number of formats. Additional commands in the control file can be used to write data out in MATLAB format.

One useful post-processing feature is the ability to scale data. HPP can scale all variables by independently-specified quantities before generating models. Thus, parameters can be expressed in natural units such as picofarads, milliamperes and kilo-ohms. Scaling compresses the range of input and output variables, which is very beneficial for non-linear optimization software. It can also be used for generating simulation models for simulators which expect parameters to be in units other than the HSPICE defaults of farads, seconds, ohms, etc.

3.12 Summary

The choice of parameters, parameter ranges and basis functions is critical to the success of macro-modelling. The procedure used to develop macro-models is essentially that described in [21]. One major improvement is the use of Q-R refactorization during term elimination, rather than the computation of a new Q-R decomposition at every step. This speeds up the macro-modelling by an order of magnitude.

In addition, software was developed to ease the generation of macro-models, provide checks that the simulation proceeded successfully, and help interface the macro-models with popular computer languages and non-linear solvers.

Chapter 4: Inverter Macro-Model Development

This chapter demonstrates the development of a macro-model for an ECL inverter. It describes the process in detail, so that the development of models for the blocks in the dual-modulus prescaler can be understood easily.

4.1 Inverter Circuit for Modelling

The inverter circuit is shown in Figure 7 below. It is the same circuit as the one in Figure 1 on page 5, except the load capacitances are shown explicitly.

Figure 7. ECL Inverter with Load Capacitances

4.2 Inverter Model Parameters

For the inverter circuit, the following independent parameters were identified:

1. The load capacitance C_L .
2. The current in the emitter-followers I_{EF} .

3. The current in the differential pair I_{CS} .
4. The input transition time τ_{TIN} .
5. The sizes of Q_1 and Q_2 .
6. The sizes of Q_3 and Q_4 .

For a fixed voltage swing, R_L depends on I_{CS} , and thus is not an independent parameter.

Because the transistor models which were used are available only for discrete transistor sizes, the transistor sizes were kept constant at minimum size. The alternative would have been to generate separate models for each combination of allowed sizes.

The dependent parameters of interest are:

1. The propagation delay τ_D .
2. The output transition time τ_{TOUT} .
3. The total current consumption I_{TOTAL} .

4.2.1 Parameter Ranges

For the inverter macro-model, the following ranges of independent parameters were used. Note that the steps need not be (and in some cases, were not) linearly spaced.

1. C_L was swept from 0.01pF to 1.01pF in 11 steps.
2. I_{EF} was swept from 0.2mA to 2.5mA in 12 steps.
3. I_{CS} was swept from 0.05mA to 2.5mA in 14 steps.
4. τ_{TIN} was swept from 80ps to 280ps in 8 steps.

As a result of these sweeps, there were $11 \times 12 \times 14 \times 8 = 14784$ simulation points.

4.2.2 Basis Functions

The basis functions consist of products of powers of the independent parameters, as described in section 3.5.1 on page 18. For τ_D and τ_{TOUT} , the exponents were:

1. C_L : {0, 0.5, 1, 2}
2. I_{EF} : {-1, -0.5, 0, 0.5, 1, 2}
3. I_{CS} : {-1, -0.5, 0, 0.5, 1, 2}
4. τ_{TIN} : {0, 1}

These choices yield $4 \times 6 \times 6 \times 2 = 288$ initial basis functions. For I_{TOTAL} , the exponents were:

1. C_L : {0, 1}
2. I_{EF} : {0, 1}
3. I_{CS} : {0, 1}
4. τ_{TIN} : {0, 1}

which yield $2 \times 2 \times 2 \times 2 = 16$ initial basis functions.

4.3 Inverter Simulation Results

Simulation results revealed that the inverter delay was markedly dependent upon the load capacitance and current levels, but not very dependent upon the input transition time. This is because the input differential pair has a very high gain, and input voltage differences of only a few V_T are sufficient to swing the current completely to one side or the other. Figure 8 on page 29 shows some of the simulation results.

(a) Delay vs. C_L and t_{TIN} ($I_{EF} = 0.9\text{mA}$; $I_{CS} = 0.7\text{mA}$)

(b) Delay vs. I_{EF} and t_{TIN} ($C_L = 0.31\text{pF}$; $I_{CS} = 0.7\text{mA}$)

(c) Delay vs. I_{CS} and t_{TIN} ($C_L = 0.31\text{pF}$; $I_{EF} = 0.9\text{mA}$)

(d) Delay vs. I_{CS} and C_L ($t_{TIN} = 128\text{ps}$; $I_{EF} = 0.9\text{mA}$)

(e) Delay vs. I_{CS} and I_{EF} ($C_L = 0.31\text{pF}$; $t_{TIN} = 128\text{ps}$)

(f) Delay vs. I_{EF} and C_L ($t_{TIN} = 128\text{ps}$; $I_{CS} = 0.7\text{mA}$)

Figure 8. Inverter Delay – Simulation Results

4.4 Inverter Model Accuracy

The inverter delay model had 103 terms, with a maximum error of -10.9ps, and a maximum relative error of 4.99 percent.

Figure 9. Inverter Macro-Model Error Distribution

Figure 9 shows the distribution of errors between the macro-model and the simulated results over the entire simulation space. As Figure 9(b) shows, all points were within 5 percent of the simulated value.

4.5 Inverter Optimization

Several types of optimizations were run. In the first set, the delay was constrained, and current consumption was optimized for various values of load capacitance. In the second set, the total current was constrained, and delay was optimized for various C_L . Finally, a design-space exploration was conducted as follows: For several values of C_L and several target delays, total current was minimized. This produced a set of curves useful for design-space exploration.

For all of the inverter optimizations, the input transition time was set at 100ps.

4.5.1 Current Minimization with Delay Constraint

Two sets of optimizations were performed. In the first, the delay was constrained to 150ps, and the load capacitance was swept from 0.01pF to 1.0pF in steps of 0.01pF. The results are shown in Figure 10 below.

Figure 10. Inverter Optimization - Delay Constraint = 150ps

In Figure 10, the delay was constrained to 150ps. A non-linear optimization was run for each step of C_L , trying to minimize I_{TOTAL} . The optimizer was unable to find feasible solutions for $C_L > 0.72\text{pF}$, indicating that the delay was impossible to achieve with this large a load. The top graph of Figure 10 shows the HSPICE simulated delay, using the load capacitance and current levels suggested by the non-linear solver. At low load capacitances, the model was pessimistic and over-estimated the actual delay. However, the predicted delay of 150ps was within 8.7ps of the simulated delay for the entire range of C_L , for a maximum relative error of 6.2%. This is slightly worse than the 5% error over the initial simulation space. However, because of the large number of steps, the error did not increase significantly between simulation points. The bottom graph of Figure 10 shows the

currents selected by the non-linear optimizer. Note that for about half the range of C_L , I_{EF} was set to the lower limit of 0.2mA. This suggests that extending the domain of the model to lower values of I_{EF} may have improved the optimization results.

Figure 11. Inverter Optimization – Delay Constraint = 100ps

Figure 11 contains results of similar optimizations with a target delay of 100ps. For this constraint, the problem was infeasible for loads larger than 0.28pF. The error between the predicted and simulated delays was small – at most 1.4ps, or 1.4%.

4.5.2 Delay Minimization with Current Constraint

For this set of optimizations, the total current was fixed, and the propagation delay was minimized using the non-linear solver. In essence, this set of optimizations found the optimal current split between I_{CS} and I_{EF} , similar to [12]. However, each optimization was very quick, because analytical delay and current expressions were used.

Figure 12. Inverter Delay Minimization with Current Constraint

In Figure 12(a), the total current was constrained to 1.25mA. The top graph shows the predicted and simulated delays, which are virtually identical. The bottom graph shows the values of I_{CS} and I_{EF} selected by the optimizer. As the load capacitance rises, more current is shifted into the emitter-followers. However, at a certain point, it becomes more productive to shift the current back into the current switch. Note that there are two emitter followers, so an increase in the emitter-follower current has twice the effect of an increase in the differential-pair current on the total current.

In Figure 12(b), the total current was constrained to 2.5mA. As the load capacitance rises, more current is shifted to the emitter-followers (except at very low loads, at which the current split has a very small effect on the delay. Even slight model inaccuracies can cause large differences in the current split.) Again, as the load is increased, there comes a point where it no longer pays to shift more current into the emitter-followers, and current is shifted back into the differential pair. With the larger available current, however, this point is reached at a higher load capacitance than in Figure 12(a).

A common rule of thumb is to use the same values for I_{CS} and I_{EF} . The results shown in Figure 12 suggest that it may be better to use more current in the differential pair than the emitter-followers.

4.5.3 Design-Space Exploration

The final set of optimizations provided an overall view of the inverter behaviour. For several values of C_L , multiple optimizations were performed for many specified delay constraints. At each optimization, the total current was minimized.

Figure 13. Inverter Design-Space Exploration – Simulated Delay

Figure 13 (above) shows the design-space exploration for various values of the simulated delay. At low load capacitances, smaller delays are possible than at high load capacitances. As the capacitance increases, more current is required for a given delay. Figure 13 sums up the optimal delay-power trade-offs for various loads.

Figure 14. Inverter Design-Space Exploration – Predicted Delay

Figure 14 is similar to Figure 13, except the x-axis reflects the delay predicted by the delay model rather than the simulated delay measured by HSPICE. It is almost identical to Figure 13, confirming good agreement between predicted and simulated delays.

Figure 15. Inverter Design-Space Exploration – Error Distribution

Figure 15 shows the distribution of relative errors between the predicted and simulated delays for all points in the design-space exploration. The maximum relative error was about 6 percent.

Figure 16. Inverter Design-Space Exploration – Alternate View

Figure 16 shows the same information as Figure 14, except that the x-axis is C_L , and the multiple curves represent various values of τ_D . This shows at a glance what values of load capacitance are feasible for a given propagation delay.

4.6 Model Generation CPU Time

The generation of the inverter models was time-consuming. The 14784 HSPICE simulations took about 25 hours of CPU time, or about 6 seconds per point. In retrospect, it appears that fewer simulation points would have yielded reasonably accurate models, since the graphs in Figure 8 on page 29 are fairly smooth. The generation of the full 16-

term current model and the 288-term delay model took 4324 CPU seconds, or 1.2 hours. The generation of the 103-term 05/95 model took about a week – many Q-R decompositions were required, and the term-elimination procedure had to backtrack many times. For circuits with many basis functions and simulation points, it may not be worth trying to reduce the length of the delay expression.

4.7 Optimization CPU Time

For the design-space exploration (Section 4.5.3), 1000 optimization runs were performed, consuming a total of 424.5 CPU seconds. Thus, each optimization took (on average) 424.5 ms. Feasible optimizations took much less (about 200ms) on average; those runs which had no feasible solution took more time.

For the current-limited case, the optimizations took 110ms on average for the 1.25mA case, and 123ms on average for the 2mA case. For the delay-limited case, the optimizations took 770ms on average for the 100ps case, and 717ms on average for the 150ps case. These times are summarized in Table 1.

Table 1. Inverter Optimization Results

Optimization	Number of Runs	Number Optimal	Number Infeasible	Number which Exceeded Iteration Limit	Average Time per Run
Design-Space Exploration	1000	814	167	19	425ms
Current Limited to 1.25mA	101	101	0	0	110ms
Current Limited to 2mA	101	101	0	0	123ms
Delay Limited to 100ps	101	28	69	4	770ms
Delay Limited to 150ps	101	72	23	6	717ms

In Table 1, column 2 shows the number of optimization runs performed for each type of optimization. Column 3 shows the number of runs which resulted in optimal solutions, and column 4 shows the number of runs for which the solver reported infeasibility. Column 5 shows those runs for which the solver did not report infeasibility, but exceeded a built-in iteration limit. Finally, column 6 shows the average time per optimization run.

This shows that once macro-models are available, optimizations are very quick, with CPU times measured in hundreds of milliseconds.

4.8 Summary

In this chapter, an ECL differential inverter was macro-modelled and optimized in various ways. The optimizations were very quick. The versatility of macro-models was demonstrated by the variety of optimizations performed, and the design-space exploration curves.

Chapter 5: Dual-Modulus Prescaler Circuit

5.1 Motivation

To prove the optimization algorithms developed for this thesis, a real-world ECL circuit is required. In addition to being a useful circuit, the prescaler is too large to optimize with brute-force circuit simulations. This chapter describes the prescaler circuit; Chapter 6 describes the optimization of the prescaler.

5.1.1 Typical Application of the Prescaler

A typical application of the dual-modulus prescaler is in a frequency synthesizer. A frequency synthesizer can be realized with a VCO in a feedback loop with a phase detector and a programmable divider, as shown in Figure 17. This circuit is capable of synthesizing integer multiples of a reference frequency F_{ref} , with the multiples depending on the range of ratios available in the programmable divider. If the programmable divider is switched between two moduli n and $n+k$, and the bit stream controlling the divider ratio is generated by a sigma-delta modulator, the circuit can synthesize frequencies between n and $n+k$ times F_{ref} , with very fine resolution (the resolution of the sigma-delta modulator.) This allows closely-spaced, high-frequency signals to be generated without requiring a high reference frequency. This type of frequency synthesizer is called a “fractional-N” synthesizer, even though the programmable divider may be capable of division only by integers. Fractional synthesis is achieved through the rapid switching of division ratios.

Figure 17. Frequency Synthesizer with Programmable Divider

The circuit used to prove the optimization procedures is a dual-modulus prescaler capable of dividing by 2 or 2.5. The division by 2.5 is realized directly, making the prescaler somewhat more complicated than a normal integer-only divider. However, the fractional division ratio allows closer spacing of the output frequencies, because the difference between the two ratios is 0.5 rather than the normal 1. This fractional difference of 0.5 is different from the normal meaning of fractional-N synthesis described previously.

5.2 Circuit Description

Figure 18. Dual-Modulus Prescaler - Block Diagram

Figure 18 is a block diagram of the dual-modulus prescaler. The 4-phase Johnson counter divides the incoming frequency by 2, and generates four outputs, 90 degrees apart, as shown in Figure 19 below.

Figure 19. 4-Phase Counter Timing Diagram

The four signals generated by the counter are called *cos*, *sin*, *cosb* and *sinb*. Note that these names are simply for convenience in tracking the phase relationships; the signals themselves are square waves. These four signals are the key to the operation of the divider.

The two MUXes each select one of the four outputs from the 4-phase counter, and are connected so that the output of the bottom MUX is always 90 degrees behind the top MUX. For example, if the top MUX is selecting the *cos* signal, then the bottom MUX will be selecting the *sin* signal, and so on.

The Johnson counter on the left is clocked on the rising edge of the output of the top MUX, gated with the modulus control input. The count is transferred from the Johnson counter to the MUXes through the MSFF on the falling edge of the output of the bottom MUX. The circuit is arranged so that each time the count presented to the MUX select inputs advances, the MUX selects the next 4-phase signal in the sequence *cos*, *sin*, *cosb*, *sinb*.

The circuit was designed to operate at up to 1GHz with a capacitively-coupled, 250mV peak-to-peak sinusoidal input signal.

5.3 Circuit Operation

When the modulus control is low, the Johnson counter is not clocked, and its count remains constant. Thus, the select inputs to the MUXes remain the same, and one of the signals from the 4-phase counter (*sin*, *cos*, *sinb* or *cosb*) will be passed through to the output. In this mode, the circuit operates as a simple divide-by-two counter.

When the modulus control is high, the Johnson counter is clocked on the rising edge of F_{ow} . The timing diagram for this mode is shown in Figure 20 on page 44.

Figure 20. Dual-Modulus Prescaler Timing Diagram - 2.5 Mode

The circuit operation is as follows: Initially, the count presented to the select inputs of the MUXes is such that the top MUX (JCK, F_{out}) is selecting cos , and the bottom MUX ($FFCK$) is selecting sin . At (1), the Johnson counter is clocked and its count advances. However, the count is not fed through to the MUXes until $FFCK$ falls, at (2). At this point, JCK and F_{out} begin following sin , and $FFCK$ follows $cosb$. This causes the glitch (3) to appear in $FFCK$. However, because JCK is stable, the Johnson counter output is stable too, and the glitch does not affect circuit operation. As long as the loop containing the lower MUX is fast enough for $FFCK$ to recover by the next rising edge of $cosb$ at (4), the circuit will work. The circuit continues in this manner, cycling through the four phases to produce a divide-by-2.5 output.

Figure 21. Dual Modulus Prescaler - Schematic Diagram

The circuit in Figure 21 follows the block diagram in Figure 18 very closely. In the schematic, however, both wires carrying a differential signal are shown. Also, emitter-follower blocks are used to shift voltage levels where needed, and additional blocks (the input amplifier, the ECL input level shifter, and the bandgap voltage reference generator) are shown.

5.4.1 Bias Voltage Generation

Each of the circuit blocks in the divider uses a current-source composed of a transistor with an emitter resistor to V_{EE} . The base of the current-source transistor is connected to a bias voltage. This bias voltage should be kept constant over a wide temperature range to ensure almost constant current in the current sources. The bias voltage is generated by a bandgap voltage source [25], shown in Figure 22.

Figure 22. Bandgap Reference - Schematic Diagram

at "6", then the divider works once again, and the timing diagram resembles that of Figure 20 on page 44.

Since the loop involving the phase-select counter has approximately constant delay, the updated phase count moves towards the forbidden region from the left as the clock frequency is increased. When the clock frequency increases still further, the updated phase count leaves the forbidden region, and the prescaler works again. Finally, at a very high frequency, the updated count arrives too late, and the prescaler fails.

The new version of the circuit uses an edge-sensitive flip-flop to eliminate this problem – the prescaler always works in the same mode, regardless of the frequency. Layouts of the new circuit with both original and optimized component values have been created, but there was insufficient time to have the new circuits fabricated for this thesis.

5.6.1 Scope Traces

Photographs of scope traces were scanned, and are presented below.

Figure 34. 1-GHz Input Signal, Divide-By-2.5 Mode

5.4.2 Four-Phase Clock Generation

The four-phase clocks *cos*, *sin*, *cosb* and *sinb* are generated from the input clock by differential ECL latches I8 and I9. The latch circuit diagram is shown in Figure 24. All transistors are minimum-size.

Figure 24. ECL Latch - Schematic Diagram

When *EN* is high and *ENB* is low, current flows through the left-hand differential pair and *D* and *DB* are transferred to *Q* and *QB*. When *EN* is low and *ENB* is high, current flows through the right-hand differential pair, and the state of *Q* and *QB* is latched.

The latch is designed to operate at a bandgap voltage of about 1.3V. The resistor values are chosen to give a voltage swing of about 300mV. The nominal high and low voltages for *D*, *DB*, *Q* and *QB* are 4.15V and 3.85V. For *EN* and *ENB*, the nominal high and low voltages are 3.3V and 3.0V.

5.4.3 4-to-1 MUX

The 4-to-1 MUX selects one of four differential inputs and passes it to the differential output. The MUX schematic is shown in Figure 25.

Figure 25. 4-to-1 MUX - Schematic Diagram

The nominal high/low voltage levels for the inputs ($I0/I0B$, etc.) and outputs Q and QB are 4.15V/3.85V; for $S1/S1B$, the levels are 3.3/3.0V; and for $S0/S0B$, the levels are 2.45/2.15V. The MUX has three stages of series gating, and therefore will not work at a supply voltage of 3.3V. It is the only circuit in the divider with this property; thus, only the MUX would need to be redesigned for a low-voltage version of the divider.

5.4.4 AND Gate

The modulus input gates *JCK* through an AND gate, shown in Figure 26. This gate is implemented with the usual series-gating topology.

Figure 26. AND Gate - Schematic Diagram

5.4.5 ECL Input Buffer

The modulus control input is driven with ECL levels, which are nominally $V_{CC}-0.9V$ and $V_{CC}-1.8V$. An input buffer shifts these levels to the 3.3/3.0V levels required by the AND gate. The ECL input buffer is shown in Figure 27. Note that no on-chip voltage reference is provided; either the input buffer must be driven with differential signals or an off-chip voltage reference of $V_{CC}-1.35V$ must be supplied.

Figure 32. MUX Layout

5.4.7 Johnson Counter

The final element in the prescaler is the Johnson counter which cycles the phase selected by the MUXes. This consists of two master-slave flip-flops connected in a loop containing one inversion. Because there are only two flip-flops in the counter, it is guaranteed to contain a valid Johnson count at all times, and no reset provisions are necessary.

Figure 29. Phase-select Johnson Counter

The flip-flops in the phase-select Johnson counter are similar to those shown in Figure 28, except the clock phases are reversed – the Johnson counter outputs change on the rising edge of the clock.

5.5 Test Chip

A test chip incorporating the dual-modulus prescaler and ECL output drivers [25] was fabricated by Northern Telecom Electronics in the NT BATMOS technology, a 0.8-micron BiCMOS technology. Important bipolar parameters are summarized in Table 2. [2]

Table 2. BATMOS Bipolar Parameters

Minimum Emitter Size	0.8 μm x 4.0 μm
β	100
f_T	11 GHz
R_E	30 Ω
R_B	350 Ω
C_{eb}	19 fF
C_{cb}	24 fF
C_{cs}	45 fF
BV_{ceo}	8.0 V
V_{Early}	60 V

The test chip used a level-sensitive latch after the phase-select counter, rather than a flip-flop as shown in Figure 21 on page 45. This caused anomalous behaviour at certain frequencies, as will be explained in Section 5.6. The latch was replaced with a flip-flop to eliminate this behaviour in later versions of the circuit.

5.5.1 Layout Issues

For all of the ECL digital gates, minimum-size npn transistors were used. Good matching is required between the current-source emitter resistors and the load resistors on the collectors of the differential pair in order to assure stable logic levels. The current-source resistors were 900 Ω , and the load resistors were 600 Ω . They were implemented using 300 Ω resistor tiles to ensure good matching. In addition, the orientation of all of the tiles was kept the same. The critical resistors for the bandgap reference generator were also implemented as tiles.

Each cell had the V_{CC} rail running in metal at the top of the cell, with V_{EE} in metal in the middle of the cell. Inputs and outputs ran horizontally in topmetal, and V_{BIAS} from the bandgap generator ran horizontally in metal.

Figure 30. AND Gate Layout

Inputs and outputs ran across the cell, and were accessible from either side. The V_{EE} rail contained a row of substrate contacts to keep the substrate at the lowest potential. Figures 31 and 32 show the layout of the bandgap voltage reference and the MUX, respectively. The AND gate, MUX and bandgap circuit are representative of the layout style used for all the cells.

Figure 31. Bandgap Reference Generator Layout

Figure 32. MUX Layout

5.6 Measurements

The test chip performed correctly with a 250mV peak-to-peak input from DC to 1500MHz, and from 1700 to 2100MHz. The 200MHz “dead zone” was caused by a race condition involving the phase-select counter and the latch. In fact, the high-frequency zone depended upon the late arrival of the updated phase-select value.

Figure 33. Timing Diagram with Latch after Phase-Select Counter

Figure 33 shows the timing diagram of the test chip. At “1”, the output is following cos , and $FFCK$ is following sin . At “1”, the phase-select counter is incremented. At “2”, the latch is enabled, and the updated phase-select count is fed to the MUXes. A short time later at “3”, the output follows sin and $FFCK$ follows $cosb$. At “4”, the latch is enabled again, but since the phase-select counter is stable, no change occurs. At “5”, the latch is disabled in preparation for the next phase change.

However, if the phase-select counter takes too long to increment, incorrect behaviour occurs. Suppose the output is following $cosb$, and $FFCK$ is following $sinb$. If the updated phase count is received within the forbidden region, the output glitches. However, if the updated phase count is delayed until after the forbidden region, but before $cosb$ rises again

at "6", then the divider works once again, and the timing diagram resembles that of Figure 20 on page 44.

Since the loop involving the phase-select counter has approximately constant delay, the updated phase count moves towards the forbidden region from the left as the clock frequency is increased. When the clock frequency increases still further, the updated phase count leaves the forbidden region, and the prescaler works again. Finally, at a very high frequency, the updated count arrives too late, and the prescaler fails.

The new version of the circuit uses an edge-sensitive flip-flop to eliminate this problem – the prescaler always works in the same mode, regardless of the frequency. Layouts of the new circuit with both original and optimized component values have been created, but there was insufficient time to have the new circuits fabricated for this thesis.

5.6.1 Scope Traces

Photographs of scope traces were scanned, and are presented below.

Figure 34. 1-GHz Input Signal, Divide-By-2.5 Mode

Figure 35. 2-GHz Input Signal, Divide-By-2.5 Mode

Figure 36. 200-MHz Input Signal Showing Modulus Switch

Figure 34 shows the output of the divide-by-2.5 mode for a 1-GHz input signal. The output swing is approximately 700mV. Figure 35 shows the output of the divide-by-2.5 mode for a 2-GHz input signal. The output swing has dropped to about 500mV, and drops further to about 400mV for divide-by-2 mode at this frequency. The output appears sinusoidal, since higher-order harmonics are filtered out by the test equipment – the oscilloscope has a 1-GHz bandwidth. Finally, Figure 36 shows a modulus change for a 200-MHz input, and a 10-MHz rate of modulus change. The top trace is the clock output, and the bottom trace is the modulus control input.

One potential problem with the prescaler is the presence of subharmonics at one-quarter of the output frequency. Because of slightly different loading conditions on the sin, cos, sinb and cosb signals, they may not be exactly 90 degrees apart. This causes the subharmonics, which could cause problems in a frequency synthesizer. Careful layout can minimize these effects.

5.7 Summary

A divide-by-2/2.5 dual-modulus prescaler was described. The circuit was fabricated by Northern Telecom Electronics, and functioned correctly. An optimized layout was generated, but there was insufficient time for fabrication and testing of the optimized version.

Chapter 6: Optimization of the Dual-Modulus Prescaler

6.1 Macro-Model Development

The dual-modulus prescaler circuit (Figure 21 on page 45) was examined, and four basic building-blocks were identified:

1. 2-input ECL AND gate
2. Emitter-follower
3. Master-slave flip-flop
4. 4-to-1 multiplexer

Figure 37. Dual-Modulus Prescaler Block Diagram showing Optimized Portion

Figure 37 shows the portion of the circuit which was optimized. Only the core of the prescaler was considered; the input amplifiers, output drivers, and four-phase clock generator were not considered, because they can be optimized separately – their delays do not depend on delay paths within the core circuit.

As Chapter 4 revealed, modelling a circuit for a variety of loads and input transition times is very time-consuming. To reduce the amount of time needed to generate the models and perform the optimizations, the four basic blocks described above were subdivided into nine blocks:

1. Two-input ECL AND gate
2. Emitter-follower feeding two flip-flops (on FFCK line)
3. Emitter-follower feeding Johnson counter (on JCK line)
4. Emitter-follower feeding two MUXes (on SEL1 line)
5. Two emitter-followers feeding two MUXes (on SEL0 line)
6. Johnson counter
7. Master-slave flip-flop
8. Bottom multiplexer (driving FFCK)
9. Top multiplexer (driving AND gate)

For these nine blocks, the loading conditions and input transition times could be estimated quite accurately, and were omitted from the model equations. Instead, when the models were generated, the blocks were simulated with actual nominal loading conditions and input transition times. Thus, the models generated were appropriate for the specific conditions in which the blocks were used. While these models were not general, they were much faster to generate, and more accurate, than if all parameters were allowed to vary.

It was found through simulation that the current level in a differential pair did not significantly affect the loading seen by a driving emitter-follower over the range of currents tested. Thus, the model simulation was done using nominal values for the current in the load differential-pair.

For each delay model, an accuracy of 1.0 with a tolerance of 0.05 was specified. This means that at all points in the simulation space, the model equations must be within 5 percent of the simulation results. The same accuracy and tolerance constraints were imposed on the power models.

6.2 ECL AND Gate Models

The AND gate schematic is shown in Figure 26 on page 50. The two delays are from A and AB to Y and YB, and from B and BB to Y and YB. For the purpose of optimizing the prescaler, only the delay from B and BB was considered. The signals A and AB are connected to the modulus control of the prescaler, and are not inside a loop. Hence, this delay does not play a role in the maximum operating frequency of the prescaler. (It does affect the time required for a change in modulus, however.) In addition to the delay, the total current consumed by the AND gate was modeled.

6.2.1 AND Gate Delay Model

For the AND gate, the independent parameters were R_{CS} and R_{EF} . The models were based directly on the current-source resistances (rather than using HSPICE current source statements) for higher accuracy. The load resistors R_L were adjusted according to R_{CS} so that the output voltage swing stayed approximately constant at about 300mV.

The model is valid for values of R_{CS} and R_{EF} from 400Ω to $5k\Omega$. These parameters were varied in 47 steps each, for a total of 2209 simulation points.)

Figure 38. ECL AND Gate Delay Model

Figure 38 shows the delay model for the AND gate, which had 27 terms. Diagram (a) shows the simulated τ_D vs. R_{CS} and R_{EF} . Diagram (b) shows the error between the model and the simulation results, which ranges from -5.7 ps to 2.5 ps. Diagram (c) shows the percentage error between the model and the simulation results. Finally, diagram (d) is a histogram showing the distribution of errors. Note that all model values are within 5 percent of the simulated result.

6.2.2 AND Gate Current Model

The current model was generated over the same ranges of R_{CS} and R_{EF} as the delay model. The total current was defined as the current through the V_{CC} power source as measured by HSPICE. The results are shown in Figure 39 below.

Figure 39. ECL AND Gate Current Model

The final model for I_{TOTAL} had 4 terms, as follows to six significant digits:

$$I_{TOTAL,AND} = -\frac{59938.3}{R_{EF,AND}^2} + \frac{1056.26}{R_{EF,AND}} + \frac{441.273}{R_{CS,AND}} + 0.0681715 \quad (16)$$

(In Eq. 16, the resistances are in Ohms, and the current is in milliamperes.) As expected, I_{TOTAL} had a strong reciprocal dependence on R_{CS} and R_{EF} . The remaining terms are sec-

ond-order effects required to achieve the desired accuracy. Note that the current macro-model is within about 3 percent of the simulated results, but the errors are distributed more uniformly than in the delay macro-model. This is probably because of the small number of terms in the current macro-model.

6.3 Emitter-Follower Models

The emitter-follower schematic is shown in Figure 40.

Figure 40. Emitter-Follower Schematic

The only independent parameter in the model is R_{EF} . The dependent parameters of interest are the total supply current I_{TOTAL} , and the propagation delay τ_D . Because the input and output are differential, and the circuit paths are identical, the delay is the same for IN falling as for IN rising. (The delay is defined as the time between IN and INB crossing, and OUT and $OUTB$ crossing.)

This section describes the models created for the emitter-follower feeding the master-slave flip-flop; similar models were created for the three other emitter-followers.

6.3.1 Emitter-Follower Delay Model

For the delay model, R_{EF} was swept from $1\text{K}\Omega$ to $40\text{K}\Omega$ in 125 steps. The results are shown in Figure 41.

(a) Simulated τ_D (ps) vs. R_{EF}

(b) Model error (ps) vs. R_{EF}

(c) Percentage error vs. R_{EF}

(d) Percentage error distribution

Figure 41. Emitter-Follower Delay Model

Diagram (a) shows the simulated results of delay vs. R_{EF} . Diagram (b) shows the error between the model and the simulated results as a function of R_{EF} . The error ranges from about -7 to 3 ps. Diagram (c) shows the percentage error as a function of R_{EF} , and diagram (d) shows the distribution of the errors. Note that the model is within 5 percent of the simulation at all points.

The final delay model consisted of three terms:

$$\tau_{D,EFFF} = \frac{179238}{R_{EF}} - \frac{2815.53}{\sqrt{R_{EF}}} + 0.662047 \times \sqrt{R_{EF}} \quad (17)$$

In Equation 17, the delay is in picoseconds, and R_{EF} is in Ohms.

6.3.2 Emitter-Follower Current Model

The emitter-follower current model was generated over the same simulation space as the delay model. The results are shown in Figure 42 below.

Figure 42. Emitter-Follower Current Model

Diagram (a) shows the simulated total current as a function of the emitter resistor. Diagram (b) shows the difference between the model and the simulated value as a function of

R_{EF} . Diagram (c) shows the percentage error as a function of R_{EF} and Diagram (d) shows the error distribution. The model is within about 1.6 percent of the simulated results for all values of R_{EF} used.

The seemingly large spike in the error at $R_{EF}=6k\Omega$ was due to numerical problems with HSPICE. The magnitude of the spike is about $6\mu A$, which is very small (about 0.6 percent of the total current at that point.)

The current model had 3 terms:

$$I_{TOTAL, EFFF} = -\frac{379243}{R_{EF}^2} + \frac{6129.57}{R_{EF}} + 0.0545080 \quad (18)$$

In Equation 18, R_{EF} is in Ohms, and I_{TOTAL} is in milliamperes.

6.4 Remaining Models

The remaining circuits were modeled using similar procedures. Table 3 contains a summary of all the models generated.

Table 3. Summary of Dual-Modulus Prescaler Macro-Models

Circuit	Parameter	Number of Terms
AND Gate	I	4
	T_{HI}	27
	T_{LO}	18
Emitter Follower feeding Flip-Flop	I	3
	T_{HI}	3
	T_{LO}	5
Emitter Follower feeding Johnson Counter	I	3
	T_{HI}	3
	T_{LO}	4
Emitter Follower feeding MUXes	I	3
	T_{HI}	5
	T_{LO}	6
2 Emitter Followers feeding MUXes	I	3
	T_{HI}	5
	T_{LO}	6
Johnson Counter	I	4
	T_{QOH}	18
	T_{QOL}	81
	T_{QIH}	18
	T_{QIL}	18
Master-Slave Flip-Flop	I	4
	T_{CKLQH}	19
	T_{CKLQL}	19
	T_{SETH}	26
	T_{SETL}	26
Bottom MUX	I	4
	T_{SOHQH}	24
	T_{SOLQH}	24
	T_{SIHQH}	16
	T_{SILQL}	28

Table 3. Summary of Dual-Modulus Prescaler Macro-Models

Circuit	Parameter	Number of Terms
Top MUX	I	4
	T_{SOHQH}	25
	T_{SOLQH}	17
	T_{SIHQH}	33
	T_{SILQH}	21

Table 4 contains definitions of the quantities in column 2 of Table 3.

Table 4. Interpretation of Model Parameters

Circuit Block	Quantity	Meaning
All	I	Total supply current consumed by circuit
	T_{HI}	Propagation delay for rising input
	T_{LO}	Propagation delay for falling input
Johnson Counter	T_{Q0H}	Delay between rising edge of JCK and rising edge of $Q0$
	T_{Q0L}	Delay between rising edge of JCK and falling edge of $Q0$
	T_{Q1H}	Delay between rising edge of JCK and rising edge of $Q1$
	T_{Q1L}	Delay between rising edge of JCK and falling edge of $Q1$
Master-Slave Flip-Flop	T_{CKLQH}	Delay between falling edge of $FFCK$ and rising edge of Q
	T_{CKLQL}	Delay between falling edge of $FFCK$ and falling edge of Q
	T_{SETH}	Setup time for D high – delay between $FFCK$ rising and rising edge master latch Q signal
	T_{SETL}	Setup time for D low – delay between $FFCK$ rising and falling edge master latch Q signal
4-to-1 MUX	T_{SOHQH}	Delay between $S0$ going high and Q going high
	T_{SOLQH}	Delay between $S0$ going low and Q going high
	T_{SIHQH}	Delay between $S1$ going high and Q going low
	T_{SILQH}	Delay between $S1$ going low and Q going low

Note that because of symmetry, $T_{SOHQH} = T_{SOHQL}$, $T_{SIHQH} = T_{SIHQH}$ and so on. The same is true for T_{HI} and T_{LO} , but because of slight numerical differences in the HSPICE runs, the models were slightly different. Also, because all models for a particular circuit were generated with a single set of transient analyses, the circuit may not have relaxed

completely between the end of one measurement and the start of the next. These differences were very small, however.

6.5 Constraint Generation

The circuit (Figure 21 on page 45) was examined, and 16 timing constraints were found. In the timing constraints which follow, the quantities are those described in Table 4. In addition, the following subscripts are used to identify circuit elements. (The circuit elements are also shown in Figure 21.)

Table 5. Subscripts Identifying Circuit Elements

Subscript	Circuit Element
<i>EFFF</i>	Emitter-follower feeding MSFF
<i>EFMX2</i>	Two emitter-followers on the SEL0 line feeding the MUXes
<i>EFMX</i>	Emitter-follower on the SEL1 line feeding the MUXes
<i>EFJC</i>	Emitter-follower feeding Johnson Counter
<i>MT</i>	Top MUX
<i>MB</i>	Bottom MUX
<i>JC</i>	Phase-select Johnson Counter
<i>FF</i>	Master-Slave Flip-Flop
<i>AND</i>	ECL AND-Gate

6.5.1 Timing Path 1

Figure 43. Timing Path #1

$$T_{LO,EFFF} + T_{CKLQH,FF} + T_{HI,EFMX2} + T_{SOHQH,MT} \leq \frac{PERIOD}{2} \quad (19)$$

$$T_{LO,EFFF} + T_{CKLQL,FF} + T_{LO,EFMX2} + T_{SOLQH,MT} \leq \frac{PERIOD}{2} \quad (20)$$

$$T_{LO,EFFF} + T_{CKLQH,FF} + T_{HI,EFMX} + T_{S1HQL,MT} \leq \frac{PERIOD}{2} \quad (21)$$

$$T_{LO,EFFF} + T_{CKLQL,FF} + T_{LO,EFMX2} + T_{S1LQL,MT} \leq \frac{PERIOD}{2} \quad (22)$$

Equations 19 through 22 specify that the maximum allowable delay along Path 1 is half the clock period. Referring to the timing diagram in Figure 20 on page 44, we see that for the divide-by-2.5 mode, the time delay between the falling edge of *FFCK* and the output of the top MUX changing to reflect the new phase must be less than or equal to one-half the input clock period. If this delay is any longer, then the top MUX will not change phase in time and a glitch will occur. Again referring to Figure 20, suppose that *CKOUT* is following *cos*. When *FFCK* falls, *CKOUT* must switch from following *cos* to following *sin* before the next rising edge of *cos*, which occurs one-half a clock period later.

6.5.2 Timing Path 2

Figure 44. Timing Path #2

$$T_{LO, EFFF} + T_{CKLQH, FF} + T_{HI, EFMX2} + T_{SOHQH, MB} \leq PERIOD \quad (23)$$

$$T_{LO, EFFF} + T_{CKLQL, FF} + T_{LO, EFMX2} + T_{SOLQH, MB} \leq PERIOD \quad (24)$$

$$T_{LO, EFFF} + T_{CKLQH, FF} + T_{HI, EFMX} + T_{S1HQL, MB} \leq PERIOD \quad (25)$$

$$T_{LO, EFFF} + T_{CKLQL, FF} + T_{LO, EFMX2} + T_{S1LQL, MB} \leq PERIOD \quad (26)$$

Equations 19 through 22 specify that the maximum delay around Path 2 can be at most one clock period. Referring again to the timing diagram on page 44, suppose that the bottom MUX (driving *FFCK*) is following *sin*. Then, after the falling edge of *FFCK*, the bottom mux must follow *cosb* before the next rising edge of *sin*, which occurs one clock period later.

6.5.3 Timing Path 3

Figure 45. Timing Path #3

$$T_{HI,AND} + T_{HI,EFJC} + T_{Q0H,JC} \leq \frac{3PERIOD}{2} - T_{SETH,FF} \quad (27)$$

$$T_{HI,AND} + T_{HI,EFJC} + T_{Q0L,JC} \leq \frac{3PERIOD}{2} - T_{SETL,FF} \quad (28)$$

$$T_{HI,AND} + T_{HI,EFJC} + T_{Q1H,JC} \leq \frac{3PERIOD}{2} - T_{SETH,FF} \quad (29)$$

$$T_{HI,AND} + T_{HI,EFJC} + T_{Q1L,JC} \leq \frac{3PERIOD}{2} - T_{SETL,FF} \quad (30)$$

Equations 27 through 30 specify the maximum delay for a rising edge of *CKOUT* along Path 3. Referring to the diagram on page 44, we see that a rising edge of *CKOUT* must change the state of the phase-select Johnson Counter before the next falling edge of *FFCK*, which occurs three half-periods later. In fact, the Johnson counter must change state sufficiently early to allow an adequate setup time for the new state to propagate through the flip-flop. This accounts for the subtraction of the appropriate setup times from the right-hand sides of Equations 27 through 30.

Earlier versions of the circuit used a simple latch instead of a master-slave flip-flop. This altered the circuit operation to require the new phase-select state to be available on the ris-

ing edge of $FFCK$, allowing only one half-period for the Johnson counter to settle. This proved to be the limiting path, and the flip-flop was used to relax the constraint on the speed of the Johnson counter.

6.5.4 MUX Phase Relationship

The preceding constraints implicitly assume that the delays through the two MUXes are approximately equal. This assumption is derived from the explicit assumption that the phases of the two MUX outputs are always 90 degrees apart.

Constraining the two MUX delays to be exactly equal would result in an infeasible problem. Thus, the MUXes are constrained to have approximately equal delays as follows:

$$T_{SOLQH,MB} \geq \beta \times T_{SOLQH,MT} \text{ and } T_{SOLQH,MT} \geq \beta \times T_{SOLQH,MB} \quad (31)$$

$$T_{SOHQH,MB} \geq \beta \times T_{SOHQH,MT} \text{ and } T_{SOHQH,MT} \geq \beta \times T_{SOHQH,MB} \quad (32)$$

$$T_{S1LQL,MB} \geq \beta \times T_{S1LQL,MT} \text{ and } T_{S1LQL,MT} \geq \beta \times T_{S1LQL,MB} \quad (33)$$

$$T_{S1HQL,MB} \geq \beta \times T_{S1HQL,MT} \text{ and } T_{S1HQL,MT} \geq \beta \times T_{S1HQL,MB} \quad (34)$$

Here, β is some parameter smaller than, but close to, 1. Typically, it is around 0.99, meaning that $0.99 \leq \frac{T_{MT}}{T_{MB}} \leq \frac{1}{0.99}$ where T_{MT} and T_{MB} are corresponding delays for the top and bottom MUX respectively.

6.6 Objective Function

The objective function is the total current in the circuit, which should be minimized. Therefore, we should minimize:

$$I_{TOTAL} = I_{AND} + I_{EFF} + I_{EFJC} + I_{EFMX} + I_{EFMX2} + I_{JC} + 2I_{FF} + I_{MB} + I_{MT} \quad (35)$$

subject to the constraints in Equations 19 through 34.

6.7 Prescaler Optimization

The circuit optimization was performed with the GAMS [22] software using the MINOS5 non-linear solver. The GAMS input file consisted of 1684 lines. About 240 lines were used to declare the various parameters and constraints, 77 lines were used to run the solve and output the results, and 1367 lines contained the macro-model equations.

6.7.1 Optimization for 1GHz Operation

To optimize the circuit for a clock frequency of 1GHz with $\beta=0.99$ took 10.6 CPU seconds, with an additional 1.3 seconds of GAMS compilation overhead.

Figure 46. Prescaler Simulation Waveforms

Figure 46 shows simulation results of the original prescaler and the prescaler optimized for 1GHz operation. The top panel of Figure 46 shows the input clock – a 1GHz, 200mV peak-to-peak sine wave. This is capacitively coupled to the input amplifier, and then fed to the 4-phase Johnson counter.

The second panel shows the output waveforms for the non-optimized prescaler. These are the waveforms as they appear at the ECL pad driver outputs. The third panel shows the outputs for the optimized divider. They are phase-shifted from the original divider because of slightly different delays, and different start-up times during the starting transient.

Finally, the fourth panel shows the supply current consumed by the optimized blocks (Figure 37). For the original prescaler, the total supply current was 22mA, and for the optimized prescaler, it was 17mA, for a 22.7% reduction in power consumption.

Table 6. Resistor Values Chosen by GAMS for 1GHz Operation

Resistance	Optimized Value (Ohms)	Original Value (Ohms)
$R_{CS,AND}$	5000	900
$R_{EF,AND}$	5000	900
$R_{CS,JC}$	5000	900
$R_{EF,JC}$	5000	900
$R_{CS,FF}$	567.0	900
$R_{EFF,FF}$	1503	900
$R_{CS,MB}$	502.4	900
$R_{EF,MB}$	4793	900
$R_{CS,MT}$	400.0	900
$R_{EF,MT}$	400.0	900
$R_{EF,EFF,FF}$	4764	7000
$R_{EF,EF,JC}$	10000	7000
$R_{EF,EF,MX}$	10000	7000
$R_{EF,EF,MX2}$	5178	7000

Table 6 shows the resistor values selected by GAMS for the 1GHz optimization, with the original values for comparison. As the table shows, the optimization shifts more power

into some elements (the MUX differential-pairs and the flip-flop) and away from other elements, for a net reduction in power consumption.

6.7.2 Multiple Optimization Runs - Frequency Specified

As a further exercise of the optimization procedure, the prescaler was optimized for target frequencies ranging from 200MHz to 1GHz, in steps of 100MHz. Table 7 shows the resulting current consumption. The optimizations took a total of 59.4 CPU seconds, for an average of 6.6 seconds per optimization.

Table 7. Supply Currents for Various Prescaler Frequencies

Nominal Input Frequency (MHz)	Simulated Maximum Input Frequency (MHz)	Supply Current Estimated by GAMS (mA)	Supply Current Simulated by HSPICE (mA)
200	650	7.44	7.15
300	650	7.52	7.25
400	650	7.52	7.25
500	700	7.67	7.4
600	790	8.19	7.95
700	820	8.46	8.3
800	950	10.27	10.2
900	990	11.75	11.55
1000	1060	17.15	17

The second column of Table 7 shows the simulated frequency at which the prescaler stopped working. For the lower frequencies, the simulated frequency is much higher than the nominal input frequency for which the circuit was optimized. This is because the range of resistances over which the circuit blocks were modeled precluded operation below about 650MHz. There was simply no way to slow the circuits down sufficiently given the domain of validity of the circuit models. The payoff for reducing frequency increases at higher frequencies. For example, reducing the nominal input frequency from 1000 to 900MHz yields a large power savings. This is to be expected, as increasing the power yields diminishing speed gains as the frequency increases.

Generally, the optimization was conservative – it over-estimated the supply current required, and under-estimated the maximum frequency at which the circuit would operate. This is because the timing constraints generated in Section 6.5 were conservative – they did not allow timing on one path to be relaxed in anticipation of a delay on another path. (For example, the delay along Timing Path 3 can be relaxed somewhat, because it assumes that the delay from the bottom MUX to the Flip-Flop clock input is zero.)

6.7.3 Multiple Optimization Runs - Total Current Specified

A second set of optimizations was performed, with the total current limited to values from 10mA to 20mA in steps of 2mA. For these optimizations, the upper bound on I_{TOTAL} was fixed, and the objective was to minimize *PERIOD*. The results are summarized in Table 8.

Table 8. Frequency Optimized for Various Supply Currents

Specified Maximum Current (mA)	Current Predicted by GAMS (mA)	Actual Current Consumed (mA)	Predicted Maximum Operating Frequency (MHz)	Simulated Maximum Operating Frequency (MHz)
10	10	10	840	960
12	12	11.8	924	960
14	14	13.8	957	990
16	16	15.7	981	1020
18	18	17.7	1006	1050
20	19.9	19.5	1010	1030

The first column in Table 8 is the specified upper bound on I_{TOTAL} . The second column is the level value of I_{TOTAL} returned by the solver after the optimization. Note that for upper bounds of 20mA and higher, the solver always returned the same answer (19.9mA and a predicted frequency of 1010 MHz.) This shows that the solver was unable to increase the circuit speed beyond this point, no matter how much current was supplied.

Column 3 contains the actual supply current as simulated by HSPICE. Column 4 contains the operating frequency predicted by the non-linear solver, and Column 5 contains the actual maximum operating frequency as simulated by HSPICE. The predicted and simu-

lated frequencies match to a worst-case error of 12.5%. Note also that in the last row of the table, the predicted frequency increases slightly from the previous row, but the actual maximum frequency decreases because of high-current effects. In the high-current region, the models are slightly inaccurate. Nevertheless, for all the rows, the circuit did operate properly at the frequency predicted by the solver.

The original, un-optimized divider consumed 22mA of current, and operated up to 1110 MHz. The fastest divider after power optimization consumed 17.7mA and ran at 1050 MHz, for a reduction in power of 19.5% at the cost of a reduction in speed of 5.4%.

6.8 Non-Linear Optimization Issues

In a non-linear optimization problem, one can not generally be certain that the solver has found a globally-optimum solution. The best that solvers can do in the general non-linear optimization is report a locally-optimum solution. It is up to the designer to assist the solvers by providing sensible starting points and formulating the problem in as tractable a manner as possible.

Model parameters can be restricted so that power and delay are monotonic functions of the independent parameters. This is usually possible over normal operating conditions, and can help assure that a global optimum is attainable.

Another mechanism for improving the tractability of the problems is scaling. The scaling facility of HPP was used to scale the independent and dependent parameters to numbers between around 0.1 and 100. The non-linear solvers perform better if the dynamic range of the variables is small.

For the prescaler optimizations, the initial starting point was the original divider, which should have provided a good starting point for the optimizer.

For multiple optimization runs, the optimal solution from one run was used as the starting point for the next run. Each set of optimizations was performed in two directions – in the

first direction, the constraint was set at its tightest bound and progressively relaxed, while in the second direction, it was set at its loosest bound and progressively tightened.

Finally, all optimizations were performed with two non-linear solvers – once with MINOS5 and once with CONOPT.

The results for the two solvers and the two directions were generally in close agreement. However, the direction in which the constraint was initially tight and then relaxed usually provided better solutions than the reverse direction. Also, MINOS5 solver was less dependent upon the starting point than CONOPT, although MINOS5 was significantly slower than CONOPT. (The timing results given in this thesis are those obtained with MINOS5.)

6.9 Summary

The dual-modulus prescaler was optimized in several ways using macro-models. The results predicted by the models were in close agreement with HSPICE simulations. The optimizations took a few seconds per run, setting the values of 14 independent parameters (Table 6), and obeying 21 constraint equations involving the independent parameters and 35 derived parameters (Table 3).

Non-linear optimization is always a delicate procedure. To increase the chances of obtaining an optimal solution, the optimizations were performed with a variety of starting points, and with two different non-linear solvers.

Chapter 7: Conclusions and Future Work

7.1 Summary

This thesis described the generation of power and delay models based on circuit simulation of small circuit blocks. The macro-models are general, reusable and cascable. Software to efficiently generate the models was developed. An ECL inverter was modeled and optimized to demonstrate the accuracy of the models. The inverter optimization was very fast, since the macro-models consist of closed-form algebraic expressions. The results agree well with circuit simulation.

A dual-modulus prescaler was partitioned into small blocks, which were modeled. The prescaler was optimized, yielding significant power savings at the target operating frequency. Again, optimization and modeling results agreed well with circuit simulation.

7.2 Contributions

The macro-modelling software created for this thesis allows for efficient creation of accurate circuit models. The accuracy of the models can be traded off against the number of terms in the model expressions. All modelling parameters are easily specified by the user in an control file, without any need to modify the software. The software uses an efficient *Q-R* refactorization technique to simplify model equations.

The modelling and optimization of the dual-modulus prescaler demonstrated the practicality of mathematical optimization of large circuits. The prescaler is much larger than the simple inverters or flip-flops described in previous work on circuit optimization.

7.3 Future Work

Several issues must be addressed to generalize macro-modelling and make it practical for very large circuits.

7.3.1 Range and Domain Matching

In order to cascade models, the ranges of output variables for one block must be completely contained within the domain of input variables for the next block. For example, if a particular circuit block can have an output transition time ranging from 50 to 250ps, then the macro-model for any block which it feeds must be valid for input transition times over the same range.

If it is impossible to match the range and domain, then the models must be artificially restricted to be compatible with one another. This introduces the possibility of a non-optimal solution.

7.3.2 Definition of Loading

For CMOS circuits, it is convenient to consider the input loading of a circuit as a capacitance. This gives accurate results, and input capacitance of digital CMOS gates can be calculated quite easily.

For ECL circuits, whose input elements are a BJT differential pair, it is not possible to characterize the circuit loading as capacitive. Instead, models must be developed with the loading characterized by the current in the differential pair, the size of the transistors, and the fanout. If all input stages in a circuit are similar, then loading can be modelled during simulation as fanout, yielding very accurate results. However, when gates can feed a variety of circuits with different sizes and different current levels in the input stages, the number of variables needed to characterize the loading becomes impractical. Thus, characterizing the loading of ECL circuits is an important problem to solve. In this thesis, the problem was avoided by simulating circuits with their nominal loading conditions.

7.3.3 Model Complexity

When many input parameters are needed to characterize a circuit block, macro-model development can be slow or impossible. For example, if a model is characterized with 5 variables, each of which is simulated at 10 points within its domain, the number of simulation points required is 10^5 . The size of the matrix for Q-R decomposition becomes impractical for any reasonable number of basis functions. Even 4 variables stress the practicality of model development, as was demonstrated in Section 4.6 on page 37.

Sacks et al. [26] propose a statistical technique for dramatically reducing the number of simulation points required to generate accurate models. They model the deterministic circuit simulation as a stochastic process, providing a statistical basis for choosing the inputs for efficient prediction. Their paper describes a problem with six independent variables, which they modeled fairly accurately with only 32 simulations.

7.3.4 CAP Issues

Finally, in order to realize the full potential of macro-modelling, layout generation must change. Currently, most ASICs are created using either a library of standard cells, or by customizing a few layers on a gate-array. In either case, the circuit designer has a limited choice of transistor sizes and resistor values. There is little point in optimizing a circuit by carefully picking sizes and resistances if the resulting circuit cannot be implemented because of library limitations. Thus, cell generators are required which can synthesize cells on demand. Each cell in a standard-cell library should be replaced with a cell generator, which can synthesize any of a family of cells, allowing the designer precise control over component values.

Note that cell libraries or gate arrays which allow a selection of various versions of a gate may be used. However, this effectively turns the optimization problem into a mixed integer-nonlinear programming problem. In principle, this simply implies the use of a different mathematical optimization package. In practice, mixed integer-nonlinear problems are much harder to solve than continuous nonlinear problems.

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

Glossary

bandgap voltage reference: A temperature-compensated voltage reference circuit.

basis function: One of a set of functions which are fit to observed data.

BiCMOS: An LSI technology which includes both CMOS and bipolar devices on the same substrate.

CONOPT: A non-linear optimizer which accepts GAMS-formulated problems.

contribution: A metric used to determine which basis functions to drop when simplifying macro-model expressions. See also: importance.

ECL: Emitter-Coupled Logic. A logic family whose basic element is the BJT differential pair.

GAMS: General Algebraic Modeling System. A language for formulating linear and non-linear optimization problems. Several solvers have been integrated with the GAMS language.

importance: A metric used to determine which basis functions to drop when simplifying macro-model expressions. See also: contribution.

Johnson counter: A digital counter formed by a ring of latches or flip-flops, with one inversion around the ring.

load: The loading seen by a driver. Often modeled as a simple capacitance, but should be modelled more accurately as a differential

	input pair with transistor sizes and collector currents as parameters.
macro-model:	A set of parameters and equations which model the behaviour of a circuit.
MINOS:	Modular In-core Non-linear Optimization System. A non-linear optimization program which accepts GAMS-formulated problems.
objective function:	The expression to be optimized during an optimization run.
orthogonal matrix:	A matrix whose inverse is its transpose.
prescaler:	The first stage of a programmable divider.
propagation delay:	In the differential ECL context, the time between an input signal pair crossing and the output signal pair crossing.
Q-R factorization:	The decomposition of a matrix A into the product of two matrices QR , where Q is orthogonal and R is upper-triangular.
size (transistor):	May be specified as the emitter area, or two separate parameters specifying the emitter length and emitter width.
transition time:	For this thesis, the time required for a differential signal to sweep from the 10% mark to the 90% mark in its high - low range.
VCO:	Voltage-controlled oscillator. An oscillator whose output frequency depends on a control voltage.

Appendix A: HPP Manual

HPP(1)

HPP(1)

NAME

hpp - a pre- and post-processor for HSPICE

SYNOPSIS

hpp [-p] [-m] [-h] [-c] *control_file* [*hspice_deck*]

DESCRIPTION

Hpp reads the *control_file* and produces an HSPICE input deck based on the control file. If *hspice_deck* is not supplied, then *Hpp* simply pre-processes the control file to standard output. If *hspice_deck* is supplied, then it is generated and HSPICE is run. The output of the HSPICE .MEASURE commands are post-processed according to instructions in *control_file*.

OPTIONS

- p Do not create a new *hspice_deck*.
- m Do not execute any *matgrab* or *matunique* commands.
- h Do not run HSPICE.
- c Do not execute any *curvefit* commands.

CONTROL FILE COMMANDS

The control file is a free-format text file. Lines beginning with '#' are treated as comments and are ignored. Commands in the control file can span any number of lines, and are terminated with a semicolon. *Hpp* is not case-sensitive.

There are three types of commands: Preprocessing, analysis and postprocessing commands. The preprocessing commands generate the input HSPICE deck, and the postprocessing commands examine the '.mt0', '.ma0', or '.ms0' file produced by .MEASURE commands. *Hpp* determines the proper file suffix based on the type of analysis requested in the *Hpp* input deck.

The main purpose of *Hpp* is to create HSPICE decks with several parameters varying independently. *Hpp* will create an HSPICE deck with all possible combinations of parameter values.

DATA SETS

A data set is simply a set of parameters and the lists of values over which the parameters are to be varied. You can create any number of data sets with *Hpp*, but only one data set is activated per run. This allows you to use different data sets for different purposes, and switch between them with simple modifications to the control file.

NUMBERS

Hpp accepts numbers in standard HSPICE input format. That is, you can use standard floating-point representations of numbers, followed by optional HSPICE multiplicative suffixes. The suffixes recognized are:

f	1e-15
p	1e-12
n	1e-09
u	1e-06
m	1e-03
k	1e+03
x or meg	1e+06
g	1e+09

PREPROCESSING COMMANDS

param dataset pname val1 [val2...];

The *param* command introduces a parameter and the list of values over which it can range. *dataset* indicates which data set the parameter belongs to, and *pname* is the name of the parameter.

HPP(1)

HPP(1)

For example, consider the following lines:

```
param set1 ibias 1mA 2mA 3mA;
param set1 vtrim 4v 4.1v 4.2v;
```

They create a data set called 'set1' which contains two parameters, 'ibias' and 'vtrim'. The data set will contain all 9 combinations of the values of ibias and vtrim.

In addition to supplying simple numbers as data points, you can supply ranges, which have the form *start:end:npoints*. *Start* and *end* specify the starting and ending values, and *npoints* specifies the number of points. *Npoints* can be omitted, in which case it defaults to 2. Range and point values can be mixed as desired. For example:

```
param set1 cl 1pf:10pf:10 12pt.52pf:11;
```

creates a parameter 'cl' in data set 'set1' which contains the 10 values 1pf, 2pf, ..., 10pf followed by the 11 values 12pf, 16pf, ..., 52pf.

If you place an 'L' or 'I' after *npoints*, the range is logarithmic rather than arithmetic. For example:

```
param set1 cl 1:1000:4L;
```

is the same as:

```
param set1 cl 1 10 100 1000;
```

If you use logarithmic ranges, both *start* and *end* must be positive.

hspice *hspice_text* ;

The **hspice** command simply copies text verbatim from the control file into the HSPICE input deck. It copies text until it encounters a semicolon. To ensure a final newline in the copied text, you should place the semicolon on a line by itself.

The **hspice** command is used to introduce stimuli or .MEASURE commands in the HSPICE deck. Here's an example:

```
hspice
.measure tran tdelay trig v(1)=2.5 rise=1
+ targ v(2)=2.5 rise=1
;
```

netlist *filename* ;

The file *filename* is copied to the HSPICE input deck, after being passed through a substitution filter described next. This command is typically used to copy the circuit topology from a reference file to the input deck.

substitute *from to* ;

This modifies the behaviour of the netlist command. If the HSPICE source file contains the text *from* enclosed in square brackets, the netlist command substitutes the text *to*.

For example, suppose the netlist file contained:

```
Q1 4 3 2 1 [NPNMODEL]
```

and that you had executed the command:

```
substitute NPNMODEL NN54111X;
```

Then, the output HSPICE deck would contain:

```
Q1 4 3 2 1 NN54111X
```

ANALYSIS COMMANDS

tran *dataset tran_text* ;

Writes out an HSPICE .TRAN command. Note that you must *manually* add the text "SWEEP DATA = *dataset*" as appropriate for the data set you supply. The tran command causes the data

HPP(1)

HPP(1)

set to be written to the HSPICE input file.

For example:

```
tran set1 0.01ns 5ns SWEEP DATA=set1;
```

writes the command ".TRAN 0.01ns 5ns SWEEP DATA=set1" into the HSPICE input deck. It also writes out all the data values in set1.

dc dataset *dc_text* ;

Similar to the tran command, but writes out a .DC analysis.

ac dataset *ac_text* ;

Similar to the tran command, but writes out an .AC analysis.

hspcommand *command* ;

Executes *command* when an HSPICE simulation should be run. By default, *command* is set to "exec hspice %s > %s 2>&1".

The string *command* can contain "C" printf() format characters. The first "%s" is substituted with the HSPICE input deck filename, and the second "%s" is substituted with the HSPICE log file, usually *filename.spo*.

If you have results from a previous HSPICE run and simply wish to post-process them, you can use the command:

```
hspcommand exec true;
```

to avoid re-running HSPICE.

POSTPROCESSING COMMANDS

bounds *pname* *low* *high* [file=*fname*];

This command verifies that the parameter *pname* is within the specified bounds. If not, it writes a message to standard error.

If the file=*fname* attribute is used, then data points which violate bounds are written to the specified file. If any bounds command uses this attribute, then all bounds commands in the control file write violations to *fname*. If more than one command specifies *fname*, only the last one in the control file is used. The bounds violation file is written in MATLAB format. Four arrays are created: 'lbnd' and 'ubnd' hold the lower and upper bounds respectively, and 'violations' holds the data points which violated the bounds. The array 'names' holds the names of the columns.

You can use this command to help check the sanity of HSPICE .MEASURE files. If, for example, you are measuring tdelay as shown earlier, and know that it should be between 0.1ns and 1ns, you can use:

```
bounds tdelay 0.1ns 1ns ;
```

in the control file.

If an error is flagged, then in all probability the circuit did not operate as expected, and the waveform file should be reviewed.

match [*param1* *loval1* *highval1* [*param2* *loval2* *highval2*...]] ;

This command acts as a filter on the HSPICE output. It restricts subsequent commands which read the HSPICE measure file to consider only those entries where the specified *parameters* lie between the specified *lowvals* and *highvals*. You should ensure that at least one line from the measure file will match the filter.

For example, suppose you simulate a circuit over a wide range for three parameters, A, B and C. However, suppose you are interested in obtaining a graph of output variable Z for nominal values of B, and for values of C between C_lo and C_hi. You can filter this data out as follows:

```
match B nominal_B nominal_B C C_lo C_hi;
```


HPP(1)

HPP(1)

This causes subsequent **matgrab** or **curvefit** commands to use only the points at which **B** is nominal. **B** and **C** lies between **C_lo** and **C_hi**. To clear the match filter, simply use:

```
match;
```

matgrab *matfile valarray pname1 [pname2...]*;

This command reads the HSPICE measurement file and creates a MATLAB command file called *matfile*, which should have the suffix ".m". The file, when run under MATLAB, will create the array *valarray*.

Valarray will contain all of the parameters *pname1* to *pnameN*. The columns of *valarray* correspond to the parameters, and the rows correspond to the rows in the HSPICE measurement file which were passed through the match filter, if any. All of the named parameters must exist in the measurement file. Consider this example:

```
matgrab file.m vals ibias vtrim tdelay;
```

That example produces the file "file.m". When it is executed by MATLAB, the array "vals" will be set to a three-column matrix. Column 1 will contain the ibias values, column 2 will contain the vtrim values, and column 3 will contain the tdelay values.

matunique *matfile valarray pname1* ;

This is similar to the **matgrab** command, except that only the unique values are grabbed from the measurement file, and only one parameter can be specified. Typically, this is used to generate a column vector containing all the values over which a parameter ranges. For example:

```
matunique test.m vals a;
```

creates a column vector called "vals" which contains all the values over which a ranges.

matlab *matfile text* ;

This command writes arbitrary text to the matlab file *matfile*. To include a semicolon in the text to write, precede it with a backslash. Note that this is the ONLY command which allows you to escape a semicolon in this way.

For example, suppose you have a variable *z* which is a function of two parameters, *x* and *y*. The following control file will read the HSPICE output and generate MATLAB code to create a mesh plot of *z* versus *x* and *y*:

```
matgrab file.m arr x y z;
matunique file.m x x;
matunique file.m y y;

matlab file.m
z = makemesh(x, y, arr);
mesh(x, y, z)\;
;
```

THE CURVEFIT COMMAND

The most complicated command used by Hpp is the **curvefit** command. This command has the following syntax:

```
curvefit [acc=accuracy] [toler=tolerance] [reduce=reduce_method] [contrib=contrib_method] [importance=importance]
[matfile=matfilename] [mateq=matfilename] [matfun=funcname]
[forteq=fortfilename] [fortfun=funcname] [gamseq=gamsfilename] [gamsfun=funcname] [ceq=cfilename]
[cfun=funcname] [csearch=yes|no] [llspeq=lispfilename] [llspfun=funcname] [llsplambda=yes|no] [sensitivity=yes|no]
[solver=solver] [maxterms=max] [mintterms=min] [scale=scale_datasets] equation_file
data_set dependent_var ;
```

This command produces a least-squares curve fit of the *dependent_var* against variables defined in *data_set*. The output is written to the file *equation_file*.

HPP(1)

HPP(1)

The parameters with the equals signs are called attributes, and have the format **keyword=value**. There should be no spaces within an attribute name or value, or on either side of the equals signs. The attributes are all optional, and have the following meanings:

acc=accuracy

This parameter specifies the desired accuracy of the solution. Hpp will try to find a solution in which the fraction of points within a specified tolerance of the simulated values meets or exceeds *accuracy*. *Accuracy* should be specified as a number between 0.1 and 1.0.

toler=tolerance

This parameter specifies the tolerance used when computing the accuracy of a solution. It should be specified as a number greater than or equal to 0. For example, if *accuracy* is set to 0.9, and *tolerance* is set to 0.05, then Hpp will attempt to find a solution in which 90 percent of the calculated points are within 5 percent of the simulated values.

reduce=reduce_method

Not yet implemented.

contrib=contrib_method

If *contrib_method* is 0, then terms are discarded by discarding the "least important" terms. If it is 1, then terms are discarded by discarding the term with the smallest "average contribution." If it is 2, then terms are discarded by discarding the term with the smallest "maximum contribution."

importance=importance

The "importance" factor for determining whether or not a term is "important" at a particular data point. This parameter is a floating-point number, and defaults to $1/(\text{number_of_terms})$ where *number_of_terms* is the number of terms in the non-reduced curve-fit expression.

matfile=matfilename

Writes a matrix to the specified matlab file. The matrix contains all the independent variables, the simulated dependent variable, and the dependent variable predicted by the final curve-fitting expression.

mateq=matfilename

Writes the resulting curve-fit equation to the specified matlab file. The equation will be written in MATLAB "FUNCTION" syntax, suitable for evaluation in a MATLAB program.

matfun=funcname

If the *mateq* attribute is used, this specifies the name of the MATLAB function generated. If omitted, it defaults to the name of the dependent variable.

forteq=fortfilename

Writes the resulting curve-fit equation to the specified FORTRAN file. The equation will be written in proper FORTRAN syntax for embedding in a FORTRAN program.

fortfun=funcname

If the *forteq* attribute is used, this specifies the name of the FORTRAN function generated. If omitted, it defaults to the name of the dependent variable.

lispeq=lispfilename

Writes the resulting curve-fit equation to the specified Common Lisp file. The equation will be written in proper Common Lisp syntax for embedding in a Common Lisp program.

lispfun=funcname

If the *lispeq* attribute is used, this specifies the name of the Common Lisp function generated. If omitted, it defaults to the name of the dependent variable.

lisplambda=yes/no

If the *lispeq* attribute is used, the *lisplambda* attribute controls the generated Lisp code. If it is no (default), then the Lisp code contains a "defun" special form. If it is yes, then the Lisp code contains a "lambda" definition of the function, bound to the global variable named by *lispfun*. Also, the range-checking code is modified so the function returns NIL if any parameters are out of

HPP(1)

HPP(1)

range.

sensitivity=yes/no

If this attribute is set to yes, then after Hpp completes the curve-fitting, it runs a sensitivity analysis on the curve-fitted expression for each independent variable at each data point. It then prints the RMS and maximum sensitivities for each variable. For problems with many independent variables or many data points, this is quite time-consuming; thus, the default value for sensitivity is no.

gamseq=gamsfilename

Writes the resulting curve-fit equation to the specified GAMS file. The equation will be written in proper GAMS syntax for embedding in a GAMS input deck. Note that numbers with exponents outside the range -18 to 18 are written with reduced precision; otherwise, GAMS issues an "exponent too large" error.

gamseq=funcname

If the gamseq attribute is used, this specifies the name of the GAMS equation generated. If omitted, it defaults to the name of the dependent variable.

ceq=cfilename

Writes the resulting curve-fit equation to the specified ANSI C file. The equation will be written in proper ANSI C syntax for embedding in a C program.

cfun=funcname

If the ceq attribute is used, this specifies the name of the C function generated. If omitted, it defaults to the name of the dependent variable.

csearch=yes/no

If the ceq attribute is used, specifying yes for the csearch attribute causes a C function to be written which searches for an initial point for the non-linear optimization process. This function iterates over the HSPICE data points, and a user-defined function should specify whether or not a data point is acceptable as an initial point, and better than the current choice. See the C code produced for details.

solver=solver

Selects which solver to use. Possible values for *solver* are:

imsl Uses the IMSL routine D2QRR to do the curve-fitting.

lup Uses a built-in LUP decomposition method. Not recommended, as this method is numerically unstable.

fastqr Uses a built-in Q-R decomposition method. This is the same method used by IMSL, but the built-in one may be preferable, because the Q-R decomposition information is retained. This means that deleting a term simply requires adjustment to the existing Q-R decomposition rather than a re-computation from scratch.

maxterms=max

Specifies the maximum number of terms to keep in the reduced curve-fit expression. Currently NOT IMPLEMENTED.

minterms=min

Specifies the minimum number of terms to keep in the reduced curve-fit expression. Currently NOT IMPLEMENTED.

scale=scale_dataset

Specifies a data set used to scale the data. The corresponding variables are scaled by the first corresponding value in the data set, if there is one.

For example, suppose you are curve-fitting the independent variable X to equations involving dependent variables A, B and C. Then:

```
param sc X 1000;
```

HPP(1)

HPP(1)

```

param sc A 10;
param sc C 1m;
param sc FOO 100;
curvefit scale=sc
      ccq=out.c
      out.eq exp X;

```

will do the curve-fitting against exponent values in the data set *exp*. In the C output file, X will be scaled by 1000, A by 10, and C by 0.001. B will not be scaled, because it did not appear in the data set *sc*. The extra parameter FOO is not used for this curve-fit run.

Note that *scale* does *not* affect *out.eq*. It only affects the C, FORTRAN, GAMS, LISP or MATLAB function files, and files produced with the *matfile* attribute.

Scaling is useful for nonlinear solvers which like all of their variables to lie in similar ranges. For example, you could scale currents in milliamps by 1000; times in picoseconds by $1.0e12$; and large resistances in ohms by 0.001 to have all variables "near" 1 for the solver.

Equation_file is the name of the file to which the curve-fit information will be written.

Data_set is a data set containing the exponents of various independent variables. For example, consider the following data set:

```

param exp x -1 -0.5 1;
param exp y 0 1 2;

```

Using that data set will cause the unreduced curve-fitting equation to be a 9-term polynomial, consisting of all the combinations of (x^{-1} , $x^{-0.5}$, x^1) with (y^0 , y^1 , y^2). The parameters x and y must appear in the HSPICE measure file.

Dependent_var specifies the dependent variable used in the curve-fitting. it must appear in the HSPICE measure file.

NUMERICAL DIFFICULTIES

Hpp will detect certain numerical difficulties and report them in the curve-fit output files. The following difficulties are detected:

Domain error occurred when generating basis

This error generally occurs if you have fractional exponents in your basis functions, and negative values for the independent variables. If your independent variables take on negative values, you can only use integer-valued exponents.

Range error occurred when generating basis

This error occurs if you have large values of independent variables and exponent values.

Large RMS error between predicted and simulated values

This generally means that you don't have enough basis functions to closely fit the data, or the basis functions are of the wrong form, and do not match the general form of the data well. It could also happen if some of your HSPICE .MEASURE commands do not trigger correctly, and return meaningless results. Use the bounds command to check for this. The RMS error message is the most subjective, and is issued if the RMS error exceeds half of the maximum simulated value. Thus, it may be issued inappropriately, or may not be issued when it should. You should always examine the match between the simulated and curve-fitted data by hand before accepting the curve-fit results.

AUTHOR

David F. Skoll

COPYRIGHT

Hpp is Copyright 1993, 1994 by David F. Skoll. It may not be distributed without the permission of David F. Skoll.

Appendix B: Sample HPP Output

This Appendix contains HPP output for the ECL AND-gate delay model.

B.1 GAMS Output

```
*-----  
*  
* Equation for THiand  
* Accuracy=1, tolerance=0.05, npoints=2209  
* Origterms=81, finalterms=27  
*  
* HSPICE measure file: inp.mt0  
* HSPICE title: 'hspice netlist for and2'  
* Term Deletion based on importance  
*  
* RMS error=0.74%596  
* Max error=-5.84104  
* Max relative error=-0.0488717  
*  
* Scaling factor(s) follow:  
* THiand -- 1e+12  
*  
* Produced automatically by HPP  
* Copyright 1994 by David F. Skoll  
*  
*-----
```

VARIABLES

```
RCSand  
REFand  
THiandV
```

```
;
```

```
RCSand.LO = 400;  
RCSand.UP = 5000;  
REFand.LO = 400;  
REFand.UP = 5000;
```

EQUATIONS

```
THiand
```

```
;
```

```
THiand.. THiandV =E=  
  (0.00461929176510*((RCSand)**(1.50000))) +  
  (-4.86845892127e-05*RCSand*RCSand) +  
  (-54565953.3006*(1.0/(RCSand*RCSand))*sqrt(REFand)) +  
  (9193693.75249*((RCSand)**(-1.50000))*sqrt(REFand)) +
```

```
( -583217.584642*(1.0/RCSand)*sqrt(REFand)) +
( 16742.6010736*(1.0/sqrt(RCSand))*sqrt(REFand)) +
( -176.550357885*sqrt(REFand)) +
(-0.00680277190321*RCSand*sqrt(REFand)) +
(0.000143551625007*((RCSand)**(1.50000))*sqrt(REFand)) +
( 3046236.75388*(1.0/(RCSand*RCSand))*REFand) +
( -547618.772196*((RCSand)**(-1.50000))*REFand) +
( 38027.7132447*(1.0/RCSand)*REFand) +
( -1263.97946439*(1.0/sqrt(RCSand))*REFand) +
( 19.1783883754*REFand) +
(-0.0988026266640*sqrt(RCSand)*REFand) +
( -33680.6895346*(1.0/(RCSand*RCSand))*((REFand)**(1.50000))) +
( 6563.03406031*((RCSand)**(-1.50000))*((REFand)**(1.50000))) +
( -490.773294971*(1.0/RCSand)*((REFand)**(1.50000))) +
( 17.5633428176*(1.0/sqrt(RCSand))*((REFand)**(1.50000))) +
(-0.292216299814*((REFand)**(1.50000))) +
(0.00188245195961*sqrt(RCSand)*((REFand)**(1.50000))) +
(-2.99014478727e-06*RCSand*((REFand)**(1.50000))) +
( -5.28221992885*((RCSand)**(-1.50000))*REFand*REFand) +
( 0.669738524380*(1.0/RCSand)*REFand*REFand) +
(-0.0291676746022*(1.0/sqrt(RCSand))*REFand*REFand) +
(0.000422519427659*REFand*REFand) +
(-1.91778755419e-08*RCSand*REFand*REFand)
;
```

B.2 ANSI C Output

```
#ifndef STAND_ALONE_MODULE
#include <stdio.h>
#include <math.h>
#endif /* STAND_ALONE_MODULE */
/*-----
*
* Equation for thi
* Accuracy=1, tolerance=0.05, npoints=2209
* Origterms=81, finalterms=27
*
* HSPICE measure file: inp.mt0
* HSPICE title: 'hspice netlist for and2'
* Term Deletion based on importance
*
* RMS error=0.748596
* Max error=-5.84104
* Max relative error=-0.0488717
*
* Scaling factor(s) follow:
* thi -- 1e+12
*
* Produced automatically by HPP
* Copyright 1994 by David F. Skoll
```

```

*
*-----*/

double
thi(double RCSand, double REFand)
{
    /* Range over which thi is valid */
    static const double RCSAND_MIN = 400.000000;
    static const double RCSAND_MAX = 5000.000000;
    static const double REFAND_MIN = 400.000000;
    static const double REFAND_MAX = 5000.000000;

    /* Make sure parameters are within range. */
    if (RCSand < RCSAND_MIN)
        fprintf(stderr, "thi: RCSand(%g) lower than RCSAND_MIN(%g)\n",
            RCSand, RCSAND_MIN);
    else if (RCSand > RCSAND_MAX)
        fprintf(stderr, "thi: RCSand(%g) higher than RCSAND_
MAX(%g)\n",
            RCSand, RCSAND_MAX);

    if (REFand < REFAND_MIN)
        fprintf(stderr, "thi: REFand(%g) lower than REFAND_MIN(%g)\n",
            REFand, REFAND_MIN);
    else if (REFand > REFAND_MAX)
        fprintf(stderr, "thi: REFand(%g) higher than REFAND_
MAX(%g)\n",
            REFand, REFAND_MAX);

    /* Actual equation follows */
    return
        (0.00461929176510*pow(RCSand, 1.50000)) +
        (-4.86845892127e-05*RCSand*RCSand) +
        (-54565953.3006*(1.0/(RCSand*RCSand))*sqrt(REFand)) +
        (9193693.75249*pow(RCSand, -1.50000)*sqrt(REFand)) +
        (-583217.584642*(1.0/RCSand)*sqrt(REFand)) +
        (16742.6010736*(1.0/sqrt(RCSand))*sqrt(REFand)) +
        (-176.550357885*sqrt(REFand)) +
        (-0.00680277190321*RCSand*sqrt(REFand)) +
        (0.000143551625007*pow(RCSand, 1.50000)*sqrt(REFand)) +
        (3046236.75388*(1.0/(RCSand*RCSand))*REFand) +
        (-547618.772196*pow(RCSand, -1.50000)*REFand) +
        (38027.7132447*(1.0/RCSand)*REFand) +
        (-1263.97946439*(1.0/sqrt(RCSand))*REFand) +
        (19.1783883754*REFand) +
        (-0.0988026266640*sqrt(RCSand)*REFand) +
        (-33680.6895346*(1.0/(RCSand*RCSand))*pow(REFand, 1.50000)) +
        (6563.03406031*pow(RCSand, -1.50000)*pow(REFand, 1.50000)) +
        (-490.773294971*(1.0/RCSand)*pow(REFand, 1.50000)) +
        (17.5633428176*(1.0/sqrt(RCSand))*pow(REFand, 1.50000)) +

```

```

(-0.292216295814*pow(REFand, 1.50000)) +
(0.00198245195961*sqrt(RCSand)*pow(REFand, 1.50000)) +
(-2.99014478727e-06*RCSand*pow(REFand, 1.50000)) +
( -5.28221992885*pow(RCSand, -1.50000)*REFand*REFand) +
( 0.669738524380*(1.0/RCSand)*REFand*REFand) +
(-0.0291676746022*(1.0/sqrt(RCSand))*REFand*REFand) +
(0.000422519427659*REFand*REFand) +
(-1.91778755419e-08*RCSand*REFand*REFand)
;
}

```

B.3 HPP Equation File Format

----- Equation for thi -----

Curve fitting parameters:

```

Specified-Accuracy = 1
Specified-Tolerance = 0.05
Specified-Importance = 0.0123457
Actual-Accuracy = 1
Initial-Number-Of-Terms = 81
Final-Number-Of-Terms = 27
Number-Of-Datapoints = 2209
Solver = Fast-QR-Decomposition
Term-deletion-method = importance

```

Domain of validity:

```

RCSand_min = 400 RCSand_max = 5000
REFand_min = 400 REFand_max = 5000

```

Range of thi:

```

thi_min = 1.1328e-10 thi_max = 2.60696e-10

```

Sensitivity NOT computed.

***** A lot of numbers deleted here to save space *****

Integrated Average Contributions:

```

RCSand ^ 2 = 0.002111674 Normalized = 1
RCSand ^ 1 = 0.002410333 Normalized = 1.141433
RCSand ^ 1.5 = 0.003630524 Normalized = 1.719264
RCSand ^ -2 = 0.003991378 Normalized = 1.890149
REFand ^ 2 = 0.004923242 Normalized = 2.33144
RCSand ^ -1.5 = 0.01951551 Normalized = 9.241727
REFand ^ 0.5 = 0.01999182 Normalized = 9.467285
RCSand ^ 0.5 = 0.03456581 Normalized = 16.36891
REFand ^ 1.5 = 0.04901783 Normalized = 23.21278
RCSand ^ -1 = 0.05587225 Normalized = 26.45875
REFand ^ 1 = 0.08115053 Normalized = 38.42948

```


RCSand ^ -0.5 = 0.08563477 Normalized = 40.55303

Sum-of-squared-errors = 1.237916e-21

RMS-Error = 7.485963e-13

Maximum-Error = -5.84104e-12

Maximum-Relative-Error = -0.0488717

The HPP Equation File is a summary of the curve-fitting information, and aids the designer in determining the accuracy of the fit. The "Integrated Average Contributions" section attempts to measure the importance of each term in the curve-fit. It does this by summing the contributions (Equation 11 on page 19) of all factors involving a particular term over the entire simulation space. The numbers may guide the choice of basis functions, but should be treated with some skepticism, since they attempt to measure an ill-defined "closeness of fit."

Appendix C: Sample Curve-fitting Code

This appendix shows the C source code for the `ExecuteCurveFit` function, which performs the steps described in Algorithm 1 on page 20.

```

/*****
/*
/* ExecuteCurveFit
/*
/* This function does the actual curve-fitting
/*
/*
/*****/
static Matrix *ExecuteCurveFit(CurveFit *cf)
{
    Matrix *ans = NULL;
    Matrix *bestans = NULL; /* Best answer so far */
    MatState *bestbasisstate = NULL;
    Matrix *predicted = NULL;
    Matrix *abspred = NULL;

    /* Matrices for the fast QR decomposition algorithm */
    Matrix *R, *b;
    int *ipvt;

    REAL bestaccuracy = -1.0; /* Best accuracy so far */
    int bestcols = cf->Basis->cols;
    REAL modelval, simulated;
    REAL coeff;
    REAL temp;
    REAL accuracy; /* Actual accuracy of curve fitting */
    Heap *h; /* For keeping track of importances */
    HeapItem hi; /* Items to put on heap */
    int npoints; /* Number of points within tolerance */
    int i, j;
    int havebackedup; /* Did we get here by backing up? */

    h = MakeHeap();
    R = NULL;
    b = NULL;
    ipvt = NULL;
    while(1) {

        /* Calculate using initial basis */
        if (ans) DestroyMatrix(ans);
        if (cf->solver == SOLVE_IMSL) {
            ans = IMSLLeastSquaresApprox(cf->Basis, cf->Dep, NULL);
        } else if (cf->solver == SOLVE_LUP) {
            ans = LUPLeastSquaresApprox(cf->Basis, cf->Dep, NULL, NULL, NULL,
                                      NULL, NULL);
        }
    }
}

```

```

    } else {

/* This is the fast QR decomposition routine.  If Q, R and b exist,
   use them.  Otherwise, create fresh Q, R and b matrices */
    if (!R) {
        R = CopyMatrix(cf->Basis);
        MakeHistory(R);
        b = CopyMatrix(cf->Dep);
        ipvt = my_malloc(sizeof(int) * R->cols);
        QRDecompose(R, NULL, ipvt, NULL, NULL, b, 0);
    }

/* Over here, we have the R, b, and ipvt matrices set up.  Call QRSolve */
    ans = QRSolve(NULL, R, NULL, NULL, b);
    PermuteRows(ans, ipvt);
}

/* Calculate the accuracy */
StartTiming(tCheckAcc);
if (predicted) DestroyMatrix(predicted);
predicted = MultiplyMatrices(cf->Basis, ans, NULL);
/* Count how many points are within tolerance */
npoints = 0;
for (i=0; i<predicted->rows; i++) {
    simulated = VELEM(cf->Dep, i);
    modelval = VELEM(predicted, i);
    if (fabs(modelval-simulated) <= fabs(simulated*cf->toler)) npoints++;
}
accuracy = (REAL) npoints / (REAL) predicted->rows;
printf("Trms=%d acc=%1.4g RMSe=%g\n", cf->Basis-
>cols, accuracy, sqrt(CurveFitErr(cf->Dep, predicted)/cf->Dep->rows));
fflush(stdout);
StopTiming(tCheckAcc);

/* If it's better than the best answer so far, keep it */
if ( (cf->Basis->cols == bestcols && accuracy > bestaccuracy) ||
    (cf->Basis->cols < bestcols && accuracy >= cf->accuracy) ) {
    if (bestans) DestroyMatrix(bestans);
    if (bestbasisstate) free(bestbasisstate);
    bestans = CopyMatrix(ans);
    bestbasisstate = SaveMatState(cf->Basis);
    bestcols = cf->Basis->cols;
    bestaccuracy = accuracy;
}

/* If we are not within tolerance, then:
   - If we cannot back up, return the answer.
   - If we can back up, do so.  If heap is not empty,
     try deleting a different term.  Otherwise, return the answer.
*/
havebackedup = 0;
if (!cf->reduce || accuracy < cf->accuracy) {
    if (h->nelem == 0 ||
        ! cf->reduce ||
        ! DeletionsExistP(cf->Basis)) { /* Can't/shouldn't back up */

```

```

DestroyMatrix(ans);
DestroyMatrix(predicted);
if(h) DestroyHeap(h, NULL);
if (abspred) DestroyMatrix(abspred);
RestoreMatState(cf->Basis, bestbasisstate);
free(bestbasisstate);
/* Set the accuracy to actual answer */
cf->finalacc = bestaccuracy;
if (R) DestroyMatrix(R);
if (b) DestroyMatrix(b);
if (ipvt) free(ipvt);
return bestans;
} else { /* We are not within tolerance, but we can back up */
if (R) DestroyMatrix(R);
if (b) DestroyMatrix(b);
if (ipvt) free(ipvt);
R = NULL;
b = NULL;
ipvt = NULL;
UndoLastDelete(cf->Basis);
havebackedup = 1;
}
}

/* Over here, we are within desired accuracy. Try deleting a term */

/* If we have backed up, a full heap exists. Otherwise, we should
create a new heap */
if (!havebackedup) {
DestroyHeap(h, NULL);
h = MakeHeap();

StartTiming(tPickTerm);
/* Calculate the importance of each term */
if (abspred) DestroyMatrix(abspred);
abspred = MultiplyMatricesAbs(cf->Basis, ans, NULL);

/* If we're using the "contribution" method, do it.
Otherwise, use the "importance" method. */
if (cf->usecontrib) {
for (i=0; i<cf->Basis->cols; i++) {
hi.cost = 0.0;
hi.d.idata = i;
coeff = VELEM(ans, i);
for (j=0; j<cf->Basis->rows; j++) {
temp = fabs(ELEM(cf->Basis, j, i)*coeff) /
fabs(VELEM(abspred, j));
switch(cf->usecontrib) {
case 1:
hi.cost += temp;
break;
case 2:
if (temp > hi.cost) hi.cost = temp;
break;
default:

```

```

        Eprint("Bad usecontrib value: %d\n", cf->usecontrib);
        exit(1);
    }
    }
    AddToHeap(h, &hi);
}
} else {
    for (i=0; i<cf->Basis->cols; i++) {
        npoints = 0;
        coeff = VELEM(ans, i);
        for (j=0; j<cf->Basis->rows; j++) {
            if (fabs(ELEM(cf->Basis, j, i)*coeff) >
                fabs(VELEM(abspred, j)*cf->importance)) npoints++;
        }
        /* Put it on heap */
        hi.cost = npoints;
        hi.d.idata = i;
        AddToHeap(h, &hi);
    }
}
StopTiming(tPickTerm);
}

/* Pull the cheapest item off the heap, delete it from basis, and iterate */
hi = ExtractMin(h);

/* If we're using the fast Q-R algorithm, adjust the QR factorization */
if (R) QRPDeleteColumn(NULL, R, b, ipvt, hi.d.idata);
DeleteColumn(cf->Basis, hi.d.idata);
}
}

```

References

- [1] R. Hadaway et al., "BiCMOS Technology for Telecommunications", *Proc. IEEE 1993 Bipolar Circuits and Technology Meeting*, pp. 159-166.
- [2] Hsu Ho and Jim Seary, *Primer on CMC 0.8-Micron BATMOS, A Version of NTE BATMOS*, Canadian Microelectronics Corporation, May 1992.
- [3] Kai-Yap Toh et al., "Sub-15 ps Gate Delay with New AC-Coupled Active Pull-Down ECL Circuit", *Proc. IEEE 1991 Bipolar Circuits and Technology Meeting*, pp. 136-138.
- [4] Kai-Yap Toh, Chin-Te Chuang, Tze-Chiang Chen and James D. Warnock, "A 23-ps/2.1mW ECL Gate with an AC-Coupled Active Pull-Down Emitter Follower Stage", *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 5, October 1989, pp. 1301-1305.
- [5] C. T. Chuang, B. Wu and C. J. Anderson, "High-speed Low-power Cross-coupled Active-Pull-Down ECL Circuit", *Proc. IEEE 1993 Bipolar Circuits and Technology Meeting*, pp. 228-231.
- [6] Hyun J. Shin, P. F. Lu and C. T. Chuang, "A High-Speed Low-Power JFET Pull-Down ECL Circuit", *Proc. IEEE 1990 Bipolar Circuits and Technology Meeting*, pp. 136-139.
- [7] Roland Ranfft and Hans-Martin Rein, "High-Speed Bipolar Logic Circuits with Low Power Consumption for LSI – A Comparison", *IEEE Journal of Solid-State Circuits*, Vol. 17, No. 4, August 1982, pp. 703-712.
- [8] H. Sato et al., "A Voltage Compensated Series-Gate Bipolar Circuit Operating at Sub-2V", *Proc. IEEE 1993 Bipolar Circuits and Technology Meeting*, pp. 232-235.

- [9] Eng-Fong Chor, Arthur Brunnschweiler and Peter Ashburn, "A Propagation-Delay Expression and its Application to the Optimization of Polysilicon Emitter ECL Processes", *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 1, February 1988, pp. 251-259.
- [10] Wen Fang, "Accurate Analytical Delay Expressions for ECL and CML Circuits and Their Applications to Optimizing High-Speed Bipolar Circuits", *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 2, April 1990, pp. 572-583.
- [11] P. K. Tien, "Propagation Delay in High-Speed Silicon Bipolar and GaAs HBT Digital Circuits", *International Journal of High Speed Electronics*, Vol. 1, No. 1, 1990, pp. 101-124.
- [12] Hsueh Y. Hsieh, Kenneth Chin and Ching-Te Chuang, "Power Partition and Emitter Size Optimization for Bipolar ECL Circuit", *IEEE Journal of Solid-State Circuits*, Vol. 28, No. 5, May 1993, pp. 548-552.
- [13] D.D. Tang and Paul M. Solomon, "Bipolar Transistor Design for Optimized Power-Delay Logic Circuits", *IEEE Journal of Solid-State Circuits*, Vol. 14, No. 4, August 1979, pp. 679-684.
- [14] R. Ranfft and H.-M. Rein, "A simple optimisation procedure for bipolar subnanosecond ICs with low power dissipation", *Microelectronics Journal*, Vol. 13, No. 4, 1982, pp. 23-28.
- [15] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1983.
- [16] MATLAB Toolbox Source Code, *qrdelete.m*, The Math Works, Inc., 1992.
- [17] *User's Manual - FORTRAN Subroutines for Numerical Applications*, IMSL Inc., 1991.

- [18] Hyun J. Shin, P. F. Lu and C.T. Chuang, "A High-Speed Low-Power JFET Pull-Down ECL Circuit", *Proc. IEEE 1990 Bipolar Circuits and Technology Meeting*, pp. 136-139.
- [19] David F. Skoll, "An Improved Curve-Fitting Algorithm for Cell Characterization", Carleton University, August, 1993.
- [20] Leon S. Lasdon and Allan D. Waren, *LSGRG User's Guide*, 1992.
- [21] Roger S. Shum and Martin C. Lefebvre, "An Empirical Timing Characterization Method and its Application to CMOS Logic Circuits", *Microelectronics Journal*, Vol. 24, 1993, pp. 477-484.
- [22] Anthony Brooke, David Kendrick and Alexander Meeraus, *GAMS: A User's Guide*, Release 2.25, The Scientific Press, 1988, 1992.
- [23] Stuart Smith and Leon Lasdon, "Solving Large Sparse Nonlinear Programs Using GRG", *ORSA Journal on Computing*, Vol.4, No. 1, Winter 1992, pp. 2-15.
- [24] Meta-Software, "HSPICE User's Manual", 1992.
- [25] Neil S. Pickles and Martin C. Lefebvre, "ECL I/O Buffers for BiCMOS Integrated Systems: A Tutorial Overview", *Proc. Sixth Annual IEEE International ASIC Conference and Exhibit*, 1993, pp. 436-443.
- [26] Jerome Sacks et al., "Design and Analysis of Computer Experiments", *Statistical Science*, Vol. 4, No. 4, 1989, pp. 409-435.

END

3 1 - 0 5 - 9 5

FIN

HPP(1)

HPP(1)

set to be written to the HSPICE input file.

For example:

```
tran set1 0.01ns 5ns SWEEP DATA=set1;
```

writes the command ".TRAN 0.01ns 5ns SWEEP DATA=set1" into the HSPICE input deck. It also writes out all the data values in set1.

dc dataset *dc_text* ;

Similar to the tran command, but writes out a .DC analysis.

ac dataset *ac_text* ;

Similar to the tran command, but writes out an .AC analysis.

hspcommand *command* ;

Executes *command* when an HSPICE simulation should be run. By default, *command* is set to "exec hspice %s > %s 2>&1".

The string *command* can contain "C" printf() format characters. The first "%s" is substituted with the HSPICE input deck filename, and the second "%s" is substituted with the HSPICE log file, usually *filename.spo*.

If you have results from a previous HSPICE run and simply wish to post-process them, you can use the command:

```
hspcommand exec true;
```

to avoid re-running HSPICE.

POSTPROCESSING COMMANDS

bounds *pname* *low* *high* [file=*fname*];

This command verifies that the parameter *pname* is within the specified bounds. If not, it writes a message to standard error.

If the file=*fname* attribute is used, then data points which violate bounds are written to the specified file. If any bounds command uses this attribute, then all bounds commands in the control file write violations to *fname*. If more than one command specifies *fname*, only the last one in the control file is used. The bounds violation file is written in MATLAB format. Four arrays are created: 'lbnd' and 'ubnd' hold the lower and upper bounds respectively, and 'violations' holds the data points which violated the bounds. The array 'names' holds the names of the columns.

You can use this command to help check the sanity of HSPICE .MEASURE files. If, for example, you are measuring tdelay as shown earlier, and know that it should be between 0.1ns and 1ns, you can use:

```
bounds tdelay 0.1ns 1ns ;
```

in the control file.

If an error is flagged, then in all probability the circuit did not operate as expected, and the waveform file should be reviewed.

match [*param1* *loval1* *highval1* [*param2* *loval2* *highval2*...]] ;

This command acts as a filter on the HSPICE output. It restricts subsequent commands which read the HSPICE measure file to consider only those entries where the specified *parameters* lie between the specified *lowvals* and *highvals*. You should ensure that at least one line from the measure file will match the filter.

For example, suppose you simulate a circuit over a wide range for three parameters, A, B and C. However, suppose you are interested in obtaining a graph of output variable Z for nominal values of B, and for values of C between C_lo and C_hi. You can filter this data out as follows:

```
match B nominal_B nominal_B C C_lo C_hi;
```