1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

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.

Automatic Behavioral Design of Mixed Analog-Digital ASICs

Design Space Exploration

By

Silvana Diana Romagnino

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

Masters of Engineering

Department of Electronics Engineering

Faculty of Engineering

Carleton University

December 1990

© copyright

1990, Silvana Diana Romagnino

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

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.

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 a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-68899-8

Canada

The undersigned recommend to the Faculty of Graduate Studies and Research the acceptance of the thesis:

**"AUTOMATIC BEHAVIORAL DESIGN OF MIXED
ANALOG-DIGITAL ASICs**

Design Space Exploration"

submitted by Silvana D. Romagnino in partial fulfillment of the requirements for the degree of Master of Engineering.

Prof. J.P. Knight
Thesis Supervisor

Prof. J.S. Wight
Chairman,
Department of Electronics

Carleton University

December, 1990

Abstract

This thesis describes some system (behavioral) level automated design aids for mixed analog-digital Application Specific Integrated Circuits (ASICs). From an abstract specification of the system's behavior, the aids assist the system designer to select good initial design solutions. The methods proposed do this by:

- Suggesting where the boundaries between analog and digital sections should be drawn.**
- Choosing an implementation for each block in the system from component libraries which consist of standard cell libraries, silicon compilers, and performance estimates of what can be custom designed.**
- Estimating which blocks may be marginal when the detailed design is completed. These blocks are most likely to have their final specifications below initial estimates and cause the overall circuit to fail. Thus, they should be designed first.**

Acknowledgements

I would like to thank my supervisor, John Knight, for his guidance during my studies and for his suggestions to my work.

Calvin Plett also deserves credit for patiently sharing his knowledge and directing my studies on analog design issues.

The financial support of the Natural Sciences and Engineering Research Council of Canada and Bell-Northern Research is gratefully acknowledged.

Finally, I would like to thank my husband for his support, patience and typing.

Abbreviations

AA	anti-aliasing
ASIC	application specific integrated circuit
A/D	analog-to-digital (used when refering to converters)
AtoD	analog-to-digital (used when refering to boundaries)
BB	Blackboard
BBD	behavior block diagram
BW	bandwidth
BW _{IS}	bandwidth of interest of signal
BW _S	bandwidth of signal
CAD	Computer aided design
CAE	Computer aided engineering
CKB	component knowledge base
D _i	the signal delay through block i
D/A	digital-to-analog (used when refering to converters)
DtoA	digital-to-analog (used when refering to boundaries)
E{}	expectation of
FA	forced analog
f _s	sampling frequency
FT	Fourier Transform
FD	forced digital
h(t)	impulse response
H(Ω)	the fourier transform of h(t)

$H(z)$	the z-transform of $h(t)$
IC	Integrated circuit
LPF	low-pass filter
$L[\bullet]$	Linear system
M	the number of blocks in the system
MDAC	multiplying D/A converter
m_j	margin to unacceptability for attribute j
PDM	pulse density modulation
PSD	power spectral density
R_{ij}	reliability of estimate for attribute j of block i
RV	random variable
S&H	sample-and-hold
SC	switched-capacitor
S_j	score w.r.t. attribute j, e.g. S_{area} is the score w.r.t. area.
SNR	signal-to-noise-ratio
SSS	strict sense stationary
T_s	sampling period
TS	total score
U_i	Uncertainty measure for block i
VLSI	Very large scale integration
V_j	Value for attribute j
WSS	wide sense stationary
Ω	angular frequency

η **mean**

Glossary

- AtoD boundary:** analog-to-digital interface. (p.42)
- Attribute:** a system specification described using penalty-credit functions and used in the TS calculation. (p.23)
- ASIC:** integrated circuits designed for a specific application with good performance and fast turnaround. Semicustom design is implied. (p.1)
- Behavior Block Diagram (BBD):** a graph whose nodes are behaviorally described system blocks and whose arcs define the direction of information flow. (p.23)
- Branch Point:** a block (node) of the BBD that has greater than one parent (more than one input) and/or greater than one child (a fanout greater than one). (p.72)
- BW_{IS}:** the 'interesting' signal bandwidth. That is, the part of the signal bandwidth that contains pertinent signal information. (p.59)
- Conversion h/w:** the hardware required for the analog-digital or digital-analog interfaces. (p.42)
- DtoA boundary:** digital-to-analog interface. (p.42)
- Domain:** in reference to blocks of a BBD, domain means analog or digital.
- leaf:** a node of a directed graph for which all arrows enter it.
- 'more-digital':** a BBD, designated as A, is 'more-digital' than another BBD, designated as B, if the following holds: every digital block in B is also digital in A, and one or more analog blocks in B are digital in A. (p.99)
- Power:** in this thesis, power refers to the average power of a signal assuming a 1 ohm load resistor. (p.30)
- root:** a node of a directed graph for which all arrows leave it.

- Score:** unitless numerical values which provide a common currency between attributes (area, delay, ...) and reflect the relative importance of the attributes. (p.24)
- Section:** a grouping of BBD blocks so that the bordering blocks (*section* inputs and outputs) are of forced domain (FA or FD), and all other blocks can be implemented in either domain. (p. 66)
- Special Branch Point:** a branch point that has, within a *section*, both FA and FD ancestors and both FA and FD descendants, and at least one parent or child with FA and FD ancestors or FA and FD descendants, respectively. (p.72)
- Style:** specific implementation for a BBD block. (p.20)
- SumOf:** a symbolic representation of the different operations required to obtain the total value of different attributes (area, delay, SNR, ...). (p. 26)
- Time-shared boundaries:** time-shared A/D or D/A converters of boundaries. (p. 69)
- Translation:** partitioning of a block's specifications to its constituent subblocks so that the overall block meets its own specifications. (p.20)
- Uncertainty measure:** a measure used to rank the blocks of a BBD in order of urgency for detailed processing. (p.109)

Table of Contents

Abstract.....	iii
Acknowledgements.....	iv
Abbreviations.....	v
Glossary.....	viii
Table of Contents.....	x
List of Figures.....	xiii
1. Introduction.....	1
1.1. Motivation.....	1
1.2. Survey of the Thesis.....	3
2. Background.....	5
2.1 Review of Continuous-Time Analog, Discrete-Time Analog and Digital Design Tradeoffs.....	5
2.1.1 Switched-Capacitor vs Continuous Analog MOS Circuits.....	7
2.1.2 Switched-Capacitor Circuits vs Digital Circuits.....	8
2.1.3 Mos vs Bipolar.....	9
2.2 Review of Analog CAD/CAE.....	10
2.2.1 Focus for Analog CAD/CAE: Simple Designs.....	10
2.2.2 Status of Analog CAD/CAE.....	11
2.3 Chapter Summary.....	15
3. An IC Design Paradigm.....	16
3.1. The Blackboard Model.....	16
3.1.1. The Blackboard.....	16
3.1.2. The Problem Specific Knowledge.....	17
3.1.3. The Knowledge Application Strategy.....	17
3.2. The Blackboard Framework.....	17
3.3. An IC Design Paradigm.....	19

3.3.1. The IC Design Hierarchy: The Blackboard	19
3.3.2. The Knowledge Modules	20
3.3.3. The Knowledge Application Strategy	22
3.4. Chapter Summary	23
4. The System Level Inputs and Design Space	24
4.1 Input Data	24
4.1.1 System Level Performance Specifications	24
4.1.2 The Blocks of the BBD and Their Unique Specifications	39
4.1.3 Component Library	40
4.2 A Simple Example	48
4.3 Chapter Summary	57
5. The System Level of the Design Hierarchy	58
5.1 Algorithm and Assumptions for Finding Initial Solutions	58
5.2. The First Pass Through the BBD	61
5.2.1 The 'Interesting' Signal Bandwidth for Noise Estimates	62
5.2.2 Signal Level	64
5.2.3 Algorithm for First Pass	67
5.3. Initial Location of Analog-Digital Boundaries	69
5.3.1 Algorithm for Initial Boundary Placement	69
5.3.1.1 Determination of Initial Boundary Location for Section Type 4b.	76
5.4 Finding an Initial Solution	82
5.5 Changing Location of Analog-Digital Boundaries	87
5.5.1 Algorithm for Boundary Movement	88
5.5.1.1 Complexity Analysis of Algorithm	99
5.6 Review of Previous Sections and Enhancements	100
5.6.1 Check for Too Many Boundaries	100
5.6.2 Check for Too Many Digital Blocks	102
5.6.3 Check if Time-Sharing Data Conversions is Valid	106
5.6.4 Estimate if Accuracy is Attainable	111
5.6.5 When the Pruning of Boundary Locations Occurs	111

5.7	Specifying the Order for Refinement Down the Hierarchy: Rating Blocks by Their "Uncertainty" Measure.....	111
5.8	Selecting other Possible Initial Solutions for Refinement.....	115
5.9	Functions That Include Domain Conversions.....	116
5.10	Chapter Summary.....	117
6.	An Example.....	119
6.1.	A Tactile Communication Aid and Its Specifications.....	119
6.2.	A Simplified Example.....	124
7.	Conclusions and Future Work.....	132
7.1	Conclusions.....	132
7.1.1	Summary of Work.....	132
7.1.2	Benefits.....	135
7.2	Future Work.....	136
Appendix A.....		141
A.1	Simplified Switched-Capacitor Circuit Theory.....	141
Appendix B.....		142
B.1	Statistics of a Signal.....	142
B.2	Propagation of PSD Through a Linear System.....	145
Appendix C.....		151
C.1	Pseudo Code for the Algorithm of Section 5.3.....	151
C.2	Pseudo code for the Algorithm of Section 5.5.1.....	166
References.....		173

List of Figures

Figure 1.1. Typical Signal Processing System.....	3
Figure 2.1. Bandwidths of signals used in signal processing applications.....	6
Figure 2.2. Signal bandwidths that can be processed by present day technologies.....	6
Figure 3.1. The BB Framework.....	18
Figure 3.2 Examples of design hierarchy.....	20
Figure 4.1. Example Penalty-Credit function.....	25
Figure 4.2. Example Penalty-Credit function for die area.....	26
Figure 4.3. Delay Calculations.....	29
Figure 4.4. Chopper-stabilized amplifier circuit.....	33
Figure 4.5. Example 1.....	35
Figure 4.6. Example 2.....	36
Figure 4.7. Part of the component knowledge base.....	42
Figure 4.8. AtoD boundary.....	43
Figure 4.9. AtoD boundary with decimation.....	44
Figure 4.10. Implications of adjusting f_s and N	45
Figure 4.11. DtoA boundary.....	45
Figure 4.12. DtoA boundary with interpolation.....	45
Figure 4.13. Example of a PDM (an oversampled) signal.....	47
Figure 4.14. Conversion Boundaries and Hardware requirements.....	47
Figure 4.15. A hydro wattmeter.....	51
Figure 4.16. User Inputs.....	52
Figure 4.17. Sample wattmeter implementations.....	53
Figure 4.18. Four possible locations for boundaries.....	54
Figure 4.19. Total Score evaluation for two implementations.....	55
Figure 4.20. Reasoning behind design approach.....	56
Figure 5.1. Algorithm Flow Chart.....	59
Figure 5.3. Sample forward-visit and backward-visit lists.....	63

Figure 5.4. Calculation of BWs and BWis requires two passes through the BBD.....	66
Figure 5.5. Algorithm outline for determining BWS and BWIS.....	68
Figure 5.6. The section types.....	70
Figure 5.6. (continued). The section types.....	71
Figure 5.7. Sharing Boundaries.....	72
Figure 5.8. Sample section layout.....	73
Figure 5.9. Sample layout of a system with four sections	74
Figure 5.10. A system with four sections	76
Figure 5.11. Demonstration of the use of condition 2.....	78
Figure 5.12. Demonstration of the use of condition 4.....	79
Figure 5.13. Examples illustrating how branch point conditions are used to control movement.....	80
Figure 5.14. Exhaustive search considers all combinations of all viable styles.....	83
Figure 5.15. Algorithm for finding the highest score initial solution using exhaustive search with branch-and-bound.....	85
Figure 5.16. Example illustrating boundary movement.....	88
Figure 5.17. Example illustrating use of condition 2.....	90
Figure 5.18. Example illustrating use of condition 4.....	90
Figure 5.19. Examples showing boundary movement.....	91
Figure 5.20. Example of boundary movement within a section.....	93
Figure 5.21. Importance of proper boundary locations on BBD list of boundaries.....	97
Figure 5.22. Within a section, different boundary locations may mean a different number of boundaries.....	99
Figure 5.23. The routine FindInitialSoln.....	101
Figure 5.24. Illustration of 'more-digital' BBDs.....	102
Figure 5.25. The function IsMoreDigital.....	105
Figure 5.26. Examples of signals that share the same boundary.....	107
Figure 5.27. Different conversion sequences for 2, 3, and 4 signals.....	109
Figure 5.28. Examples for $R_{i,area}$	113
Figure 5.29. Example illustrating the use of the uncertainty measure.....	115

Figure 5.30. Use of 'domain-converting' functions.....	117
Figure 6.1. Block diagram of the tactile aid and its external interfaces.....	120
Figure 6.2. Some PCFs for the Tactile aid.....	122
Figure 6.3. Autocorrelation and power spectral density for speech.....	123
Figure 6.4. The generic BBD for the simplified system.....	125
Figure 6.5. Boundary movement.....	126
Figure 6.6. Selected boundary location for example.....	127
Figure 6.7. Possible solution to simplified system.....	128
Figure 6.8. Part of the component knowledge base.....	129
Figure 6.9. First order low-pass filter.....	130
Figure 7.1. Example of reconvergent fanout.....	136
Figure A.1. Switched-Capacitor Equivalent Resistance.....	141
Figure B.1 Linear System.....	147
Figure B.2. Sampled Signals.....	150

*To my father,
for all of his encouragement*

CHAPTER

1. Introduction

1.1. Motivation

This thesis deals with design aids for mixed analog-digital application specific integrated circuits (ASICs). The term ASIC has been used in several different contexts. In the broadest sense, an ASIC is an integrated circuit designed for a specific application. That is, it is not a general purpose circuit. The creation of these ASICs requires the combination of system technology and VLSI technology.

According to J. Prang, president of Valid Logic Systems, "[The] ASIC design process can be thought of as the fast food of the IC world. It gives the system designers a consistent product, with good performance, fast turnaround, and just enough have-it-your-way flexibility to maintain creativity" [MagN89]. This latter definition focuses on a subset of ASICs that have particular characteristics. The first requirement for fast turnaround drives the need for design aids. The second requirement is for good, and not necessarily optimal, performance. This lowering of expectations for performance comes as a direct consequence of the need for fast turnaround. Superior performance is achieved by letting the designer and the computer each do what they do best. For instance, designers, to date, are better at employing ad hoc design tricks, and computers are better at register allocation in a data path [MidJ88]. However, design time restrictions may require using the computer to perform some of the tasks at which the designer is better but slower, resulting in lower performance.

Finally, creativity must be possible. Creativity translates into exploring the design space. Design tools that consistently explore this space may allow for greater creativity than a system designer (discussed in section 7.2). System designers may quicken the exploration by providing useful information for pruning this design

space. Throughout this thesis, the term ASIC will refer to integrated circuits with the aforementioned requirements.

In the area of application specific integrated circuits, designers are mixing more and more analog and digital circuits on the same chip. Recent studies predict the number of IC designs incorporating analog circuit modules are expected to make up 40% of all ICs in 1991, and more than 60% by the year 2000 [WebS89,CarL89]. Many of the industry movers in the ASIC area promote the same view, for instance: "[NCR Corp. states that] about a third of its ASICs contain analog circuits, and that number is growing."; and "Sierra [states] that mixed signal application will capture half of all semicustom designs by 1994 ..." [RunS89].

This mixture of analog and digital circuitry on the same chip can be attributed to the increasing number of complete systems designed on one IC and the fact that most VLSI systems require analog circuits (such as reconstruction filters, digital-to-analog and analog-to-digital convertors, voltage comparators, automatic gain control circuits, etc). For these system-on-a-chip designs, the replacement of analog circuit applications by digital circuits has found its limitations. Some applications, like external analog interfaces, have to be performed in analog, while others are smaller and faster when performed in analog. This 'mixed-mode' design presents a new challenge for design-automation tools. Whereas, digital CAD/CAE is maturing, until recently, the lack of analog CAD/CAE has been quite apparent and has contributed to making the design of analog sections the bottleneck that limited fast turnaround in IC design. Consequently, more and more, design-automation tools geared towards analog and mixed analog-digital designs are being developed.

The goal of this thesis is to outline a synthesis package for mixed-mode ASICs focusing on the system (behavioral) level of the design process. At the system level, the ASIC is already partitioned into functional blocks described behaviorally and a synthesis tool must decide how to implement each block: which domain (digital or analog) and which structure. This thesis proposes some system level

automated design aids which suggest where the boundaries between digital, sampled-analog, continuous analog and oversampled analog should be drawn. This partitioning of the circuit is illustrated in figure 1.1 [AllP87]. The location of the boundaries between the analog and digital parts depend on the application, the performance and the area. As well, the aids suggest implementations for the system blocks based on an extensive library. The library includes specifications for standard cells, estimates of specifications for circuits which can be automatically designed by silicon compilers, or even estimates for circuits which could be hand designed. Once the blocks are selected, the aids can estimate which blocks are most likely to cause the overall design to fail because their final specifications failed to satisfy the initial estimates.

Figure 1.1. Typical Signal Processing System [AllP87].

1.2. Survey of the Thesis

In chapter 2, the tradeoffs between digital and analog circuits are presented. Then a short summary of the current status of analog CAD/CAE is given. The outline of a complete synthesis package, from behavioral description to layout, is introduced in chapter 3. The design paradigm works on the principle of a hierarchical representation of a design for which higher level specifications and constraints are propagated down the hierarchy to assist in the selection process. The importance of a good initial solution is emphasized. The input information for system level design is presented in chapter 4. Chapter 5 describes automated design aids to assist the system designer select good initial design solutions by performing the following functions:

- Suggesting sections of the circuit that could be advantageously implemented in analog, sampled-analog or over-sampled analog; thereby, calculating good positions in the circuit for signal conversions.
- Choosing implementations for the blocks either from standard cell libraries, silicon compilers, or performance estimates of what can be custom designed.
- Calculating an "uncertainty" for each block in meeting the overall circuit specifications. Marginal blocks might cause the overall design to fail if their final specifications were below the initial estimates. These blocks should have their lower level design and verification done first. That is, the rest of the system should not be completed until it is known if such marginal blocks can be successfully designed.

An example illustrating some of the ideas of chapters 4 and 5 is given in chapter 6. Finally, chapter 7 concludes by summarizing the contributions and limitations of the work presented, and discusses future work.

CHAPTER

2. Background

This chapter summarizes the tradeoffs between analog and digital circuits. Thus, it serves to show that mixed-mode designs are advantageous because they can take advantage of these tradeoffs. In several past applications, such as the field of signal processing in both audio and video frequency ranges, digital techniques have supplanted analog ones. This occurred because digital techniques offered easier designs; however, this came at a cost of increased area. Analog CAD systems are now challenged with reversing this decision.

Also, this chapter introduces the developing area of analog CAD and shows that mixed-mode synthesis is a valuable undertaking.

2.1 Review of Continuous-Time Analog, Discrete-Time Analog and Digital Design Tradeoffs

Circuits designed in any of the following domains: digital, continuous-time and discrete-time analog (such as, switched-capacitor), may exist on a single chip implemented in MOS or BiCMOS technology. In signal processing applications, signal bandwidth is probably the most important system consideration. Figures 2.1 and 2.2 show the bandwidths of different signals and the bandwidths processed by different technologies, respectively [AllP87]. Whereas, digital techniques are used almost exclusively below 100Hz and analog techniques above 10MHz, in the overlap region a tradeoff must be made between the high accuracy and flexibility of a digital approach and the low cost, power, and size of an analog approach. It is systems that deal with frequencies in this overlap region that make up mixed-mode applications and thus, are the focus of this thesis. The following sections will give a brief summary of the existing tradeoffs between the domains and the technologies.

Figure 2.1. Bandwidths of signals used in signal processing applications [reproduced from AllP87].

Figure 2.2. Signal bandwidths that can be processed by present day technologies [reproduced form AllP87].

2.1.1 Switched-Capacitor vs Continuous Analog MOS Circuits

Switched-capacitor (SC) circuits are sampled analog circuits. The basic functioning of SC circuits is described in appendix A. The building blocks (e.g. operational amplifiers, comparators, etc.) for SC circuits are essentially analog. However, certain circuits (e.g. filters, multipliers, programmable-gain amplifiers, rectifiers, etc.) in ICs are usually designed as SC circuits rather than continuous analog ones. This choice reflects the fact that continuous analog circuits are less accurate and larger in area than their SC counterparts. Given that resistors and capacitors have about a 10% absolute but only a 1% tracking error¹, the error of any RC time constant in a continuous analog circuit may be as large as 20%; whereas, for a SC circuit, the RC time constant ($\cong (T/C_R) C = C/C_R \cdot 1/f_c$) has an error of 1% or less resulting from the tracking error of C and C_R [GreR86]. In addition, the area needed for the simulated resistor (i.e. the capacitance C_R) is usually much smaller than that needed for the direct realization. For instance, in a typical example from [GreR86], for $R = 10\text{M}\Omega$ and $f = 100\text{kHz}$, the area occupied by R is approximately 1600 mils; whereas, the area occupied by $C_R = 1\text{pF}$ is approximately 4 mils!

Another disadvantage of continuous analog circuits over SC ones is that continuous analog circuits are designed with MOS resistors. MOS resistors are inherently nonlinear and therefore, have a limited operating range. Increased linearity comes at the expense of more complicated circuitry.

In contrast to the above advantages, when compared to continuous analog, SC circuits have the following disadvantages:

- SC circuits do not work at as high frequencies.

¹Tracking error is the error in the ratio of two capacitors or of two resistors on the same IC.

- Being a sampled data design, a SC circuit must deal with aliased noise.
- If the clock rates are not at least 20 times higher than the signal frequencies of interest, the time delay through the SC integrator can also give rise to a non negligible error of the form $((\sin x)/x)^{-1}$ in the magnitude of the frequency response, and to excess phase shift [MavJ83]. However, there are methods which easily compensate for this magnitude error. For instance, specifications for filters in the design can be altered so that the gain in the passband is varied to compensate for this error.
- The MOS switch in a SC circuit introduces clock feedthrough noise [GreR86].

2.1.2 Switched-Capacitor Circuits vs Digital Circuits

Switched-capacitor (SC) circuits have several advantages over digital circuits. In terms of area, the density of operations is greater because the basic operations of multiplication, addition, and delay are much simpler to perform than in digital. This simplicity reflects the fact that SC circuits deal with the 'real' signal; whereas, digital circuits deal with bit encoded data. Because the SC circuits are simpler, they have higher speed and lower DC power requirements than equivalent digital circuits.

However, compared to digital circuits, SC circuits have limited accuracy since capacitor ratios can only be made accurate and stable to about 0.1% of their nominal value. This corresponds to only 10 bit accuracy in digital terms. This limited accuracy shows up in such factors as filter center frequency.

Also, the dynamic range (maximum-signal-to-noise ratio) of SC circuits seldom exceeds 100dB (typically 70 to 90dB) because of noise originating in the operational amplifier, the switches and coupled from the clock and supply lines [GreR86].

In conclusion, the choice between SC and digital is not a clear one; both are useful. A specific task should be assigned to a SC functional block whenever the following are of prime importance [GreR86]:

- speed of operation,
- small chip area,
- small DC supply power,
- the required input (output) is inherently analog in nature.

Alternately, digital functional blocks should be assigned if:

- the input and output signals are already digital,
- flexibility for programming or multiplexing is critical,
- accuracy and/or dynamic range requirements are very stringent,
- the necessary components are already in the library,
- the necessary components can be time-shared.
- for high volume ICs, if fast manufacturing tests are necessary.²

2.1.3 Mos vs Bipolar

The behavior of an amplifier realized on MOS rather than on Bipolar is superior because it has a much higher input impedance and reduced cost, size and power requirements. However, it is inferior in terms of offset voltage, dynamic range, and noise (especially at low frequencies due to $1/f$ noise) [GreR86, AllP87].

²Digital manufacturing tests currently deal with testing for 'stuck-at' faults in combinational logic; whereas, to date, analog manufacturing tests form a subset of characterization tests. These tests verify that specifications are met.

As for speed, referring back to figures 2.1 and 2.2, even though the upper limit of MOS-LSI digital logic is the fastest moving boundary, it is unlikely that MOS-LSI circuits will achieve faster speeds than bipolar technology.

2.2 Review of Analog CAD/CAE

This section describes analog CAD/CAE. The problems and the present status of analog CAD are presented with a brief mention of analog CAE.

2.2.1 Focus for Analog CAD/CAE: Simple Designs

As stated in the introduction, while digital CAD/CAE is quite mature, analog CAD/CAE is still in its infancy. This condition is a result of the differences between digital and analog design. These differences show why digital design is amenable to CAD where analog design is not. The differences are listed below [AllP89, CarL88]:

- In continuous analog design, signals have a continuum of values for amplitude and time; in discrete-time (switched-capacitor) analog design, signals have a continuum of values for amplitude; in digital design, signals have only one of two amplitude states and are also discrete in time.
- Analog components have a continuum of values, whereas, digital components have fixed values.
- Even though analog circuits tend to have fewer transistors than digital circuits, there is typically much more design time per transistor in analog. Thus, the time required to design an analog circuit is 4-10 times greater than the time required to design a digital circuit. This extra design time is attributed to the fact that analog circuits exploit the full spectrum of capabilities exhibited by individual devices and the fact that the macroscopic properties of analog circuits are affected by many interacting electrical constraints (such as gain, frequency response, stability, etc.).

- Modeling can be simplified for digital designs; whereas, precise modeling is required for analog design.
- Because of the many interacting variables in analog design, the simplified models used are never completely adequate, whereas, accurate though simple models are used in digital design. So digital designs produce successful circuits the first time, while analog designs usually require several, (2-3), iterations for success.
- Analog circuits are made of irregular blocks that have been customized. On the other hand, digital circuits are made from regular standard blocks.
- Fabrication process constraints appear in far greater detail and far earlier during the analog design process than in the digital design process. This occurs because analog device characteristics are more dependent on parameters extracted or predicted for the process.
- The transfer of the system level design to the circuit level and then to the layout level is much easier for digital designs than it is for analog designs.

The above comparisons demonstrate that the difficulty in designing analog circuits is due to the larger number of mutually interacting parameters from potentially different levels of the analog hierarchy. Consequently, when applying CAD techniques to analog IC designs, the premier approach is to consider simple analog designs. Here, a simple analog design is a design using a unique grouping/interconnection of ordinary non-interacting analog functions; in contrast, a complex design is one which pushes the limits of precision and performance [AHP89].

2.2.2 Status of Analog CAD/CAE

Many digital design methodologies exist, such as, macro cells, standard cells, gate arrays, parameterized blocks or module generators, fully custom, silicon compilers, etc. Analog design

methodologies have followed similar development paths. Below, several analog design methodologies are discussed.

Analog Arrays

Analog arrays are pre-processed ICs containing unconnected preplaced components (including resistors, capacitors, transistors, digital gates, operational amplifiers, switched-capacitor filter blocks, etc.). Such arrays are completed by defining interconnections on one or more mask layers (metal levels). Some of the corporations that use this methodology are Micro Linear, Linear Technology, and Silicon Systems [AllP89].

Analog Standard Cells

Analog standard cells are predesigned components (including operational amplifiers, comparators, switch arrays, oscillators, capacitor arrays, bandgap reference, peak detectors, A/D and D/A converters, delay cells, automatic gain control) with different fixed parameters. Standard cells reside in a software database. They may be designed to be of fixed height, so circuits can be laid out in rows with routing between rows. Or cells may be of arbitrary size so that specialized placement and routing are required. Some of the corporations that use analog standard cells are Silicon Systems, Harris Semiconductor, IMP, NCR, and Sierra Semiconductor [AllP89].

Analog Standard Tiles [MezP87]

Analog standard tiles consist of pre-designed components, similar to those found in analog arrays; however, these tiles reside in a software database. Each tile contains a few devices (e.g. resistors, transistors, capacitors) and each is of the same dimensions (or multiple of) so that it joins easily to other tiles. These tiles are interconnected using two metal layers. Tiles have less long-term costs than other standard cells or arrays, though greater front-end costs. Here long-term costs are those costs dealing with performance penalty and area; and short term costs are those dealing with design difficulty and prototyping.

Specifically, since arrays satisfy the 'general' design, devices are not always in the right place and certainly do not occur in optimum proportions; therefore, consuming more area than tiles. For the same reason, arrays suffer performance penalties since devices meet 'general' performance specifications. However, tiles have much higher wafer prototyping costs than arrays.

Tiles also require less area than general-purpose standard cells and exhibit a much higher performance level than these fixed-parameter standard cells. However, even though the prototyping costs are comparable, tiles are more difficult to design than standard cells.

Consequently, tiles should be used for applications requiring high performance at reasonably high production volumes.

Analog Parameterized Cells (Module Generators)

Analog parameterized cells are partially pre-designed circuits; what is left to be specified are the values of a few of the performance parameters. However, the ranges of these values is limited. Usually, the circuit topologies are fixed and transistor sizes are varied. This methodology is most successful when used for designs exhibiting regular structure, such as, switched-capacitor filters. These circuits also reside in a software database and include the same variety of components as analog standard cells. Examples of analog parameterized cells are switched-capacitor filters from IMP, and the SRC AIDE2 module generator system which uses standard cells [AllP89].

Analog Silicon Compilers

Analog silicon compilers automatically design analog circuits. Given a high level specification, a compiler generates a circuit schematic and a layout. IDAC and ILAC by Centre Suisse d'Electronique et de Microtechnique (CSEM), ACACIA: OASYS and ANAGRAM by Carnegie Mellon University (CMU) and OPASYN by University of California are all examples of silicon compilers [CarL89, HarR87, DeGM87, DeGM89].

Though quite different in philosophy, all three work with a knowledge base containing several circuit topologies for each specific circuit (e.g. an operational amplifier), as well as, models of the components making up the topology and a set of analytical equations describing the behavior of the circuit. These equations express circuit performance in terms of designable parameters, such as, transistor size, threshold voltages, etc [CarL88]. Given input specifications, the compiler must select a topology and assign the best possible values to the components. The output is a sized schematic diagram ready to layout. Each compiler is coupled with a layout package. A comparison of the three compilers is reported in [DeGM89]. Clearly, many distinct circuits can be produced since each topology represents an infinite number of designs corresponding to different component sizing. Examples of the types of circuits designed using silicon compilers are operational transconductance amplifiers, operational amplifiers, voltage references, current references, A/D converters, and comparators.

It is noteworthy to state that creating design knowledge is both difficult and very time consuming. For example, it took almost 2 years to acquire the knowledge for designing two topologies of amplifiers for OASYS [HarR89]. Thus designing synthesis packages is only economical for frequently used analog blocks.

Other areas under study

A lot of research is being conducted in the analog environment in the areas of modeling (defining device, macro and behavior models), mixed simulation (both mixed analog-digital and multilevel), layout (sensitivity driven or rule driven) and testing. [IsmM90]

2.3 Chapter Summary

Some of the many tradeoffs existing between analog and digital circuits have been listed. Mixed-signal designs can take advantage of these tradeoffs. However, fast turnaround designs are only feasible with automation. This leads to new challenges in analog and mixed-signal CAD/CAE. The current status of analog CAD was summarized. This trend shows that mixed-signal synthesis is a valuable undertaking.

CHAPTER

3. An IC Design Paradigm

This chapter briefly describes a problem solving model for the complete IC design problem from specifications to layout. A problem solving model is a scheme for organizing reasoning steps and problem specific knowledge to construct a solution to a problem. The problem solving model referred to as the Blackboard model [NiiH86] is introduced and used here. Much of the detailed description of the model's components and their behavior is taken from the "Expert System Paradigm for Design" under development by F.D. Brewer and D.D. Gajski [BreF86], which is an application of the Blackboard model.

After the complete solution process is covered, the specific phase in the solution process from the specification (behavioral) level to the next lower level is selected for further study. The specification level implementation of the ideas introduced in this chapter is described in the following chapters.

3.1. The Blackboard Model

The Blackboard (BB) model prescribes the organization of the knowledge and of the data needed to solve the problem. As well, it provides a strategy for applying that knowledge. The BB, the domain (problem specific) knowledge and knowledge application strategy are described in this section.

3.1.1. The Blackboard

The BB or solution space, holds all the data. The BB is hierarchically organized with information at each level representing partial solutions. A unique vocabulary describes the information at each level. The hierarchy serves to partition the problem (here, the IC design task) into subproblems (loosely coupled subtasks). This partitioning makes a great deal of difference to the clarity of the

approach, the speed with which solutions are found, the resources required, and even the ability to solve the problem.

3.1.2. The Problem Specific Knowledge

The domain knowledge is partitioned into separate and independent modules of knowledge which perform subtasks. The modules transform information at one level of the hierarchy into information on the same or other higher or lower levels.

3.1.3. The Knowledge Application Strategy

The knowledge application strategy applies *opportunistic* reasoning within the overall organization of the solution space and task specific knowledge. That is, pieces of knowledge are applied at the most "opportune" time, i.e. no apriori established order exists. Specifically, which knowledge module to apply is determined dynamically, one step at a time, resulting in the incremental generation of partial solutions, and is based on the solution state and on the existence of knowledge modules capable of improving the current state of the solution. The knowledge modules do not communicate directly; rather, the globally accessible information on the BB serves as a medium of interaction.

3.2. The Blackboard Framework

The Blackboard (BB) framework, figure 3.1, bridges the gap between a model and a working system by extending the model to include a detailed description of the model's components and their behavior. For instance, the framework specifies that knowledge modules are algorithmic procedures, sets of rules, or both procedures and rules. How a piece of knowledge is represented is an engineering decision. As well, it specifies that the properties of the BB objects are represented by, say, attribute value pairs and that the relationships between objects are represented by named links: "part-of" or "in-support-of" links for objects on different levels, and "next-to" or "follows" links between objects on the same level. The framework

describes the control for the knowledge application strategy as a set of control modules that monitor the BB changes and decide what action to take next: either the selection of which partial solution to pursue or which knowledge module to apply, or the invocation of a knowledge module to a BB object. At every stage, the reasoning may be data driven (top-down) or goal driven (bottom-up). Also, both top-down and bottom-up reasoning can be used with the hypothesize-and-test¹ strategy. What control mechanisms are needed depends on the complexity of the application task.

Figure 3.1. The BB Framework.

Obviously, different applications are implemented with different combinations of knowledge representations, reasoning schemes, and

¹In the hypothesize-and-test strategy a knowledge source generates a hypothesis which is validated by other knowledge sources using top-down and bottom-up reasoning which assigns credibility to the hypothesis.

control mechanisms. The BB framework for IC design is described next.

3.3. An IC Design Paradigm

3.3.1. The IC Design Hierarchy: The Blackboard

The IC design hierarchy recasts the design process into smaller tasks. From top to bottom (i.e. from higher levels of abstraction to more concrete levels), the hierarchy proceeds from behavioral to structural to physical description. Also, within each description, there are different levels of abstraction. Two examples, one analog [HarR89] and one digital are given in figure 3.2.

As is true in many complex problems, hierarchy aids in the design process by dealing with smaller design tasks for which specific knowledge is available. Hierarchy is also advantageous in that it promotes the reuse of sub-blocks in different contexts and requires that the specific information for designing a sub-block exists only once. The drawback of hierarchical decomposition, especially apparent in analog designs, is the inability to allow design tricks across many levels of the hierarchy. When the design of one block depends on the internal details of another block, analog design experts use design tricks. In these cases, a higher performance design is created at a higher level, than would otherwise result from dividing the design into subtasks and interconnecting the sub-blocks that result. To allow for design tricks, the higher level design encompassing both blocks would have to exist as a pre-designed library cell. However, this would only be appropriate for very common functions. A more interesting proposal suggests that for a design activity related to block X to modify the internal details of the design for an unrelated block Y, the modifications should be recast as input specifications for block Y [HarR89].

Figure 3.2 Examples of design hierarchy:

- (a) A/D convertor showing some of the levels and the blocks within the levels;
 (b) digital filter showing the various levels.

3.3.2. The Knowledge Modules

Some of the knowledge modules required for IC design are the already existing CAD and CAE tools. Some knowledge modules work at one level of the hierarchy, while others take information at one

level and transform it into information at another level. For example, a gate level simulator works at one level; whereas a module generator takes the specifications (parameter values) at one level and transforms this into a list of standard cells and their interconnects at another level.

On the other hand, many of the knowledge modules must still be developed. Of particular interest is the knowledge stored in the component database, as well as the component database itself.

The component database contains the information required for all available blocks. This information is used for *style* selection and *translation* [BreF86, HarR89]. A *style* is a specific implementation for a block. Different *styles* imply different tradeoffs. For example, a multiplier may be implemented using the Booth, shift-add, or iterative algorithms; a current mirror may be designed as a cascode or a simple type. *Translation* is the partitioning of higher level performance specifications (such as area and delay) to the constituent sub-blocks, so that the overall block meets its own performance specification. This partitioning process is unique to each style. Therefore, with every style a plan or strategy (i.e. an ordered set of design steps) which performs translation is stored. These plans include simplified models to guide the tradeoffs being made during constraint partitioning.

As well as a plan for *translation*, and for constraint propagation down the hierarchy, a plan for constraint propagation up the hierarchy must be stored [BreF86]. This upward propagation of constraints is required to update earlier estimates for performance measures. That is, at higher levels of the hierarchy estimates are made on component performance. Then as the design process moves to lower levels, the actual design performance figures should be propagated back up to the higher level to update earlier estimates. In this way, the estimates become more accurate and allow better repartitioning of constraints when failure occurs.

Failures occur when the performance figures of a component do not satisfy the input specifications (estimates) from higher levels. Knowledge modules are required for failure handling. In the expert-system design paradigm [BreF86], successive failures cause redistribution/repartitioning of constraints, change of *style*, and finally, failure to higher levels with a report specifying the reason for failure. As an example, consider a failed circuit consisting of a multiplier and an adder. Redistribution of constraints may try assigning more area to the multiplier leaving less for the adder. Change in multiplier *style* may consider another multiplication algorithm. Failure to a higher level leads to considering constraint repartitioning and change of *styles* at the higher level. Strategies for constraint repartitioning and new *style* selection are stored in the component database.

3.3.3. The Knowledge Application Strategy

The solution space is made up of partial solutions, and it is up to the controller(s) to decide the next partial solution to proceed with. Performance estimates evaluate partial designs and this evaluation allows the controller to proceed with the most promising initial designs. A few sequences of events are given below:

sequence 1:

- Select a style for a given function and performance specifications
- Translate performance specifications and specify order-of-attack for subtasks.

sequence 2:

- Simulate a partial solution
- If simulation results satisfy estimates, update estimates
- Else, proceed with failure handling

The translation step of sequence 1 includes the specification of an order-of-attack for the subtasks. That is, a method is needed that decides the order in which the several subtasks are to be refined down the hierarchy. Experts know which components are the critical ones in the system and design these first. In the same way, every style should have a plan-of-attack which orders the subtasks in decreasing probability of failure. The subtasks most likely to fail should be refined first. In this way, if a prospective solution is going to fail, it will fail sooner rather than later, saving valuable time. In chapter 5, an "uncertainty" measure is introduced for generation of this order.

3.4. Chapter Summary

This chapter has described a problem solving framework for IC design. The salient features of this framework are the hierarchically represented data, the knowledge modules that perform *translation* and *style* selection, the component database knowledge module, and the opportunistic reasoning (good plans-of-attack) used in the knowledge application strategy.

The remainder of this thesis proposes design aids that work at the system level of the hierarchy. The aids perform the required *translation* of system input/output and constraint specifications to block specifications, and select the *style* for each functional block. This process is an extension of Brewer's and Gajski's expert-system digital design aids to include mixed-signal design. Thus, the *style* selection implies a domain selection, analog or digital, and the addition of conversion hardware at the analog-digital boundaries.

As well, one design aid specifies a plan-of-attack for the order in which subblocks (subtasks) are to have their lower level designs completed and verified.

CHAPTER

4. The System Level Inputs and Design Space

In this chapter the system level inputs are described. As well, a small example is given to demonstrate finding a good initial solution and the factorial nature of the design space.

4.1 Input Data

The system level inputs can be grouped into user defined and synthesis system supplied inputs. The user inputs consist of

- a behavioral block diagram (BBD),
- specifications on system inputs and outputs, and
- specifications on system constraints.

The BBD is a graph whose nodes are behaviorally described system blocks and whose arcs define the direction of information flow, see for example figure 4.16a.

The synthesis system input is the component library.

4.1.1 System Level Performance Specifications

System level specifications include specifications on the environment (such as temperature range and radiation levels), on the inputs and outputs, and on the global constraints. In signal processing, input/output specifications include dynamic range (DR), bandwidth (BW), power spectral density (PSD), and probability density function. Global constraints include area, latency (the delay between an output and its input), throughput (the delay between successive outputs), noise, inaccuracies, and power consumption. The desired system performance is a function of the tradeoffs required to achieve that performance level. Tradeoffs exist between the global constraints (attributes).

The method for describing attributes using penalty-credit functions introduced in [ZhuX88] is used here. The penalty-credit functions are piecewise linear curves as shown in the example in figure 4.1.

Figure 4.1. Example Penalty-Credit function [ZhuX88]

A credit is a score above zero. A penalty is one below zero. V_j is the value of some attribute, thus if $j = \text{area}$, then V_{area} is the number of μm^2 used. If $j = \text{power}$, V_{power} is the number of μWatts used. Some of the other attributes a designer might want to constrain are SNR, bandwidth, propagation delay, and/or dynamic range.

The user must specify a penalty-credit (score) function for each input attribute. The scores (S_j 's) are unitless numerical values which provide a common currency which reflect the relative importance of each attribute. In this way, for instance, $S(V_{\text{area}})$ can be compared directly to $S(V_{\text{power}})$. The scores deal with relative values, i.e. values greater or less than some reference value whose score is 0. In this way, attribute values of different magnitudes are translated to a comparable scale. The piecewise curve is an extension of the simpler concept of normalized values $(V_j - V_{j\text{ref}})/V_{j\text{ref}}$ where $V_{j\text{ref}}$ is some reference or required value within the acceptable range. The example of figure 4.2 serves to show how piecewise curves provide more precise information about the criteria for selection.

Figure 4.2. Example Penalty-Credit function for die area, assuming package sizes available are 4 mm² and 6 mm².

The area values between 0 and V_3 , and between V_3 and V_5 represent two die areas. Generally, as more area is used the score decreases (cost increases). As the area gets too close to the limit of the pad frames (between V_2 and V_3 and between V_4 and V_5), an increase in area means a proportionally greater decrease in score than in the sections where the area is well within the die area limits (between V_1 and V_2 and between V_3 and V_4). There is a big jump in cost when the area is increased so that the larger die must be used (at V_3). In this specification, requiring more than one die is unacceptable.

The example demonstrates the increased flexibility and better precision with which specifications may be given using penalty-credit functions.

The designer supplied attribute penalty-credit curves represent the attribute value for the whole system. The attribute values of the constituent blocks of the system must be combined appropriately to determine the score for the whole system. That is, given M blocks in the system, the score for attribute j is given by:

$$S(V_j) = S\left(\sum_{i=1}^M (V_{ji})\right)$$

where $V_{j,i}$ is the value of attribute j for block i

Note that SumOf rather than Σ is used here. This is done purposely, since SumOf does not necessarily represent an algebraic sum. Rather, SumOf is a symbolic representation of the different operations required for obtaining the resulting values, V_j , for each attribute. For instance, for area and power, SumOf represents simple algebraic addition; whereas, for delay and noise, it is a more complex function. Because of the factorial nature of the search for good solutions, which will become apparent in the following sections, the function SumOf must be simple to conserve computer time, so approximations must often be used. For instance, the mean and variance through a nonlinear operation may be calculated using first order approximations. This is usually inadequate for large signals; therefore, the computer must warn the designer when it suggests circuits derived from such calculations. The user then decides if further analysis is required. The use of SumOf for the area, power, signal delay and SNR attributes follows:

1) Area:

$$\sum_{i=1}^M \text{SumOf}(V_{\text{area}}) = \sum_{i=1}^M V_{\text{area},i} = V_{\text{area},1} + V_{\text{area},2} + \dots + V_{\text{area},M}$$

2) Power

$$\sum_{i=1}^M \text{SumOf}(V_{\text{power}}) = \sum_{i=1}^M V_{\text{power},i}$$

3) Signal Delay (Latency)

At every stage define $V_{\text{delay},i}$ as the worst-case signal delay measured between the system input(s) and the output of block i . Then, $V_{\text{delay},i}$ is defined recursively as:

$$V_{\text{delay},i} = V_{\text{delay},i-1} + D_i \quad (\text{figure 4.3a})$$

where D_i is the signal delay through block i .

Then it follows that the worst case signal delay from system input(s) to system output(s) is given by $V_{\text{delay},M}$, that is

$$\sum_{i=1}^M (V_{\text{delay},i}) = V_{\text{delay},M} \quad (\text{figure 4.3b})$$

For blocks with greater than one input, $V_{\text{delay},i-1}$ is defined as follows:

$$V_{\text{delay},i-1} = \left[\text{Max}_{\substack{k \text{ is a} \\ \text{parent of } i}} \{ V_{\text{delay},k} \} \right] \quad (\text{figure 4.3c})$$

That is, the parent with the longest delay is considered.

D_i represents the signal delay through block i . If block i performs a filtering function, then the delay (group delay) may be approximated. In particular, for low-pass filters of order greater than two, a useful approximation for delay is the risetime $t_r \approx 1/(2 \text{ BW})$, where t_r is the 10% to 90% rise time in seconds and BW is the -3 dB bandwidth in Hz [StrF77]. For nonfiltering functions (memoryless systems¹), D_i is the circuit delay for continuous-time blocks and $1/f_s = T_s$, the time between samples, for discrete-time (sampled data) blocks. Note that for sampled data blocks, the circuit delay must be less than $1/f_s$. For digital blocks, if two or more

¹A memoryless system is one for which the output at any instant of time can be expressed as a function of the input at the same instant of time.

successive components have a combined delay of less than $1/f_s$, then a delay of $1/f_s$ is added for the first block and 0 delay is added for the other blocks. An example of such 'grouped' blocks is shown in figure 4.3d. For switched-capacitor circuits, there may be several stages of sampling so that the delay may be several clock periods ($N \cdot 1/f_s$).

Figure 4.3. Delay Calculations.

- (a) $V_{\text{delay},i}$; (b) $V_{\text{delay},M}$;
 (c) $V_{\text{delay},i}$ when block i has more than one parent;
 (d) $V_{\text{delay},i}$ for grouped digital blocks.

To summarize, D_i is defined as follows:

- $D_i =$
- for filtering functions
rise-time approximation.
 - for nonfiltering functions,

circuit delay,	for continuous-time analog blocks
$N \cdot 1/f_s,$	for discrete-time analog blocks
$1/f_s,$	for digital blocks
0,	for grouped digital blocks.

4) Signal-to-Noise Ratio (SNR)

The SNR of a system is the SNR measured at the output of the system, that is

$$\sum_{i=1}^M \text{SNR}_i = \text{SNR}_M.$$

where SNR_i is the SNR measured at the output of block i , and M is the last block in the system (leaf of the BBD).

Next the calculation of SNR is considered.

In signal processing applications, SNR may be used as a performance measure. SNR is the ratio of average signal power over average noise power. Thus, for voltage signals (similar results hold for current signals) SNR is given by the following:

$$\text{SNR} = \frac{\text{signal power}}{\text{noise power}} = \frac{V_{\text{rms}}^2(\text{signal})}{V_{\text{rms}}^2(\text{noise})}$$

where V_{rms} is the root mean square (rms) voltage and is calculated at any point in the circuit as follows:

- for signals with a discrete frequency spectrum,

$$V_{rms}^2 = \sum_{i=1}^N V_i^2$$

where N = number of signal frequency components f_i .

V_i = The amplitude of the component of the signal at frequency f_i .

- for signals with a continuous frequency spectrum from f_1 to f_2 and for which a power spectral density (PSD) is defined,

$$V_{rms}^2 = \int_{f_1}^{f_2} \text{PSD} df$$

where: PSD = a measure of the localization of power on the frequency axis specified in units of Volt²/Hz.

Notice that only the average power of the signal V_{rms}^2 assuming a 1 Ω load resistance is required since both the signal and the noise power are referenced to the same resistance. From this point onwards, power will be referred to a 1 Ω load resistance.

The determination of signal power for stochastic processes, for random variables and for deterministic time varying and constant signals is discussed in appendix B.

The following discussion demonstrates that power spectral density and not simply average power must be propagated through the system. To evaluate different design alternatives, attribute values must be compared. In this thesis, consideration of different design alternatives is performed using an exhaustive search of all viable styles for each block in the system. As is described in more detail in chapter 5, the branch-and-bound technique is used and it requires

that all attributes are evaluated every time a block is selected and added to a partial design solution. If signal-to-noise ratio is used as an attribute, then it must be calculated using the frequency band containing pertinent signal information ("frequency band of interest" in section 5.2) and not over the whole spectrum. This will prevent denoting a partial solution as unacceptable or performing badly because of a low SNR over the whole spectrum when most of the noise gets filtered out at later stages. To illustrate this, consider the following example.

In figure 4.4 a chopper-stabilized amplifier is shown [GreR86]. Here $1/f$ noise from amplifier A1 is eliminated. The modulator M1 is used to shift the signal so that it is centered at f_c before going through amplifier A1 (see (d)). Then the combined modulated signal and noise are modulated again resulting in (e). The frequency range used in the calculation of SNR is shown. Notice that if, say, for (d) SNR is calculated over the whole spectrum, then the $1/f$ noise would be included in the calculation; consequently, the partial design would be wrongfully concluded as bad since it includes the $1/f$ noise which is filtered out by the LPF as seen in (f).

Clearly, if SNR is to be used as a criteria, then it is not sufficient to propagate average power through the system; the power spectral density is needed. Nonstationary stochastic processes have a time varying PSD. This limits the use of SNR as an attribute to random variables and stationary stochastic processes. In practical applications, nonstationary processes are usually dealt with by modelling them as stationary ones, or they undergo specific analysis to understand the amount of variation in PSD. This type of analysis is process dependent and is not dealt with in this thesis.

Figure 4.4. Chopper-stabilized amplifier circuit.

a) block diagram; b) input spectrum; c) noise spectrum; d) spectrum at node A; e) spectrum at node B; f) spectrum at node C [GrcR86].

The propagation of PSD through memoryless nonlinear systems and through linear systems will be considered next. Linear systems, denoted by $y(t) = L[x(t)]$ are defined by the principle of superposition, which combines the additivity and scaling properties:

$$L[a_1 x_1(t) + a_2 x_2(t)] = a_1 L[x_1(t)] + a_2 L[x_2(t)]$$

where a_i is a constant which may be a random variable over an ensemble and x_i is a stochastic process. As an example, most filters are linear systems. The output of a linear system is a convolution of the input $x(t)$ with the impulse response $h(t) = L[\delta(t)]$ of the system.

$$y(t) = L[x(t)] = x(t) * h(t) = \int_{-\infty}^{\infty} x(t-\alpha) h(\alpha) d\alpha$$

where $\delta(\cdot)$ = Dirac delta function

$$= \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases} \quad \text{such that} \quad \int_{-\infty}^{\infty} \delta(t) dt = 1$$

The determination of $\text{PSD}_y(\Omega)$ is discussed in appendix B where it is shown that

$$\text{PSD}_y(\Omega) = \text{PSD}_x(\Omega) |H(\Omega)|^2$$

where $H(\Omega)$ is the fourier transform of $h(t)$.

In appendix B, inputs consisting of signal only and of signal and noise are considered. For the signal and noise case, results are given for uncorrelated noise. The following two examples illustrate the propagation of PSD.

$$H_1(\Omega) = \frac{V(c)(\Omega)}{V(A)(\Omega)} = \frac{R_o/R_i}{1 + \Omega/\Omega_o}, \text{ where } \Omega_o = \frac{1}{R_o C_o}$$

$$H_2(\Omega) = \frac{V(c)(\Omega)}{V(B)(\Omega)} = 1 + \frac{R_o/R_i}{1 + \Omega/\Omega_o}$$

Figure 4.5. Example 1

In example 1, figure 4.5, the linear system is an analog low-pass filter using an operational amplifier. The input $x(t)$ has both signal and noise content and it goes through the system transfer function $H_1(\Omega)$. As well, the linear system itself contributes more noise to the signal. Here $n_2(t)$ is input referred noise reflecting the system's thermal and 1/f noise. This noise sees a different transfer function, $H_2(\Omega)$, to the output. Since the input signal and noise and the amplifier input referred noise are all independent² and therefore uncorrelated, the output PSDs are calculated as follows:

$$\text{PSD}_{\text{Signal}}(\Omega) = \text{PSD}_s(\Omega) |H_1(\Omega)|^2$$

$$\text{PSD}_{\text{Noise}}(\Omega) = \text{PSD}_{n1}(\Omega) |H_1(\Omega)|^2 + \text{PSD}_{n2}(\Omega) |H_2(\Omega)|^2$$

²Two signals are independent if their joint density function is the product of their densities.

$$H_1(z) = \frac{V_{(C)}(z)}{V_{(A)}(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

$$H_2(z) = \frac{V_{(C)}(z)}{V_{(B)}(z)} = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

$$\eta_{N2} = 0$$

$$\sigma_{N2}^2 = 5 \frac{2^{-2B}}{12}, \text{ where } B \text{ is the number of bits in the datapath}$$

$$R_{N2N2}[n] = \sigma_{N2}^2 \delta[n]$$

$$PSD_{N2} = \sigma_{N2}^2$$

Figure 4.6. Example 2

In example 2, figure 4.6, the linear system is a digital IIR filter. The linear system adds its own noise. In this case, the noise source is quantization noise. For this example the models and assumptions used to obtain a description for $N_2[n]$ can be found in [OppA89]. The output PSDs are given by:

$$\text{PSD}_{\text{Signal}}(\Omega) = \text{PSD}_S(\Omega) = \text{PSD}_S(\Omega) |H_1(e^{j\Omega T})|^2$$

$$\text{PSD}_{\text{Noise}}(\Omega) = \text{PSD}_N(\Omega) = \text{PSD}_{N_1}(\Omega) |H_1(e^{j\Omega T})|^2 + \sigma_{N_2}^2 |H_2(e^{j\Omega T})|^2$$

The sampling function is also considered in appendix B where it is shown that if $x[n]$ equals the time-samples of a continuous time process $x_c(t)$ sampled at equally spaced intervals T , so that

$$x[n] = x_c(nT),$$

then

$$\text{PSD}(\Omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \text{PSD}_c(\Omega + \frac{2n\pi}{T}).$$

A memoryless nonlinear system, denoted by $y(t) = g[x(t)]$, is one for which the output $y(t)$ at any instant of time can be expressed as a function of the input $x(t)$ at the same instant of time. Nonlinear systems with any arbitrary input are difficult to deal with. Usually, system analysis is undertaken for special interesting functions (squarer, limiter,...) and inputs (Gaussian or Poisson processes, white noise, sinusoidal signal,...). An example best illustrates this.

Consider a square law detector: $y(t) = a x^2(t)$ taken from [DavW87]. Let $x(t)$ be made of signal and noise so that $y = a (s(t) + n(t))^2$. If $s(t)$ and $n(t)$ are independent processes ($E\{sn\} = E\{s\} E\{n\}$) with zero means ($E\{s\} = E\{n\} = 0$), then:

$$\begin{aligned} R_y(t_1, t_2) &= a^2 E\{(s_1 + n_1)^2 (s_2 + n_2)^2\} \\ &= a^2 [E\{s_1^2 s_2^2\} + 4 E\{s_1 s_2\} E\{n_1 n_2\} \\ &\quad + E\{s_1^2\} E\{n_2^2\} + E\{s_2^2\} E\{n_1^2\} + E\{n_1^2 n_2^2\}] \end{aligned}$$

Furthermore, if s and n are wide sense stationary,

$$R_y(\tau) = a^2 [R_{s2}(\tau) + 4 R_s(\tau)R_n(\tau) + 2 \sigma_s^2 \sigma_n^2 + R_{n2}(\tau)]$$

where $\sigma_s^2 = E\{s^2\} = R_s(0)$, the variance.

Then, the output signal and noise autocorrelation function are given by:

$$R_{\text{Signal}}(\tau) = a^2 R_{s2}(\tau)$$

$$R_{\text{Noise}}(\tau) = 4a^2 R_s(\tau) R_n(\tau) + 2a^2 \sigma_s^2 \sigma_n^2 + a^2 R_{n2}(\tau).$$

Whereas in the linear case only $R_s(\tau)$ and $R_n(\tau)$ are required, here $R_{s2}(\tau)$ and $R_{n2}(\tau)$ are also needed. Clearly, for different systems other statistics will be required. These statistics may be worked out apriori for specific interesting cases. For instance, for the square law detector, if the noise is a gaussian process, then:

$$R_{n2}(\tau) = R_n^2(\tau) + \sigma_n^4.$$

This result can be stored in the component data base. However, dealing with any type of noise source becomes computationally prohibitive.

In the general case of any input process, the density functions will have to be propagated through the BBD. Then a statistic such as $R_{n2}(\tau)$ is calculated using

$$R_{n2}(\tau) = R_{n2}(t_1, t_2) \Big|_{\tau=t_1-t_2}$$

The propagation of the second order noise density function through the system may be done using the direct method or the transform method [DavW87]. The direct method is used if g is differentiable. However, for problems of any complexity beyond the square law

detector, analytical difficulties often arise in the evaluation of various integrals involved. In many interesting cases, the transform method of analysis may avoid such difficulties. Even if these methods can be used, the calculations involved are too numerous for the exhaustive search algorithm. Rather, the results for several special input types for signal and noise, such as a pure sinusoidal signal with white noise, can be predetermined and stored in the component data base. Then if one of these cases holds, the statistics will be readily available. If none hold, then the user must supply a description of the output signal and noise processes. This may require simulation and the use of mean square estimation of a random variable y in terms of a well characterized random variable x , or the use of spectral estimation [PapA84].

The analysis of noise in non-linear systems with memory is even more difficult and is not within the scope of this thesis

4.1.2 The Blocks of the BBD and Their Unique Specifications

The nodes of the BBD perform high level functions, such as, filtering, amplification, multiplication, integration, etc. Each different function is described behaviorally. A tool such as the AT&T IOSpec Analyzer [LeeE89], would be used to input the design data for each function. The IOSpec Analyzer provides "Specification forms" for the IC designer to fill out. The values of all entries are audited for sanity checks. This design information is captured only once and is reused throughout the whole design process to drive different applications. For instance, the data is used to generate command files that describe various options and parameters for execution control of CAD tools. Ongoing work on this project includes support for a hierarchical I/O spec environment.

Different types of specifications and their treatment are considered next.

- Function-specific:** Define the function and are specified by the user. These include such inputs as filter cutoff frequency and averaging period.
- Derived:** Block I/O specifications, such as dynamic range and bandwidth, that are derived by propagating the system level I/O specifications down the BBD.
- Undetermined:** Block constraints, such as noise and area, whose values cannot be known because they are affected by the choice of *style*. The default specification is the 'workable' range of values; i.e. values that do not readily lead to unacceptable solutions. These specifications are determined by considering what the system attribute values are when styles for all the ancestor blocks of the BBD are selected; and calculating what the current block may contribute without resulting in any unacceptable attribute values. This process is illustrated in section 5.4. The users can further limit this range of values if their experience so dictates. The smaller the range, the fewer the styles considered.
- Inherited** Certain design decisions limit the style choices at lower levels of the design by forcing compatible component selections. These include decisions on technology, domain, clocking schemes, layout technology (such as, gate array or standard cell), etc. Such decisions are not partitioned, rather they are inherited.

4.1.3 Component Library

The synthesis system input consists of the component library. This contains information such as specifications of standard cells, curves of the specifications believed procurable from silicon compilers, and

estimates of what experienced designers feel are possible for hand-crafted circuits. All blocks of the BBD must map to a component in this library.

Figure 4.7 illustrates a section of the component knowledge base. The low-pass filter function is considered. The leaves of the tree structure represent all the different styles available. For each style, information on achievable specifications are given. For each specification, this information is given as a list of achievable ranges in the form $[(v_1, v_2) (v_3, v_4) \dots]$. The values $v_1, v_2, \dots$ may be known or estimated values, or may be determined from curves which produce accurate or estimated values. The ancestor specifications are formed by obtaining the union of the respective children specifications. This has been shown in figure 4.7 for the area specification.

In this way, the number of styles that are scanned may be reduced. That is, if the required specifications are not satisfied at a given node, it is not required to look below that node. As well, the component knowledge base is divided into different domains before looking at different styles. This makes it easy to quickly determine if a function can be implemented in either domain. Also, it allows quick rejection of all styles in a given domain. This is required in style selection during an exhaustive search when the domain is already determined and no longer left as a design choice.

Figure 4.7. Part of the component knowledge base.

Since the methods that follow in chapter 5 attempt to find optimum locations for analog-to-digital (AtoD) and digital-to-analog (DtoA) boundaries, it is of interest to describe some of the information stored in the component library about these blocks. In this thesis, an AtoD boundary refers to the hardware required to perform the data conversion from the analog domain to the digital domain. This hardware includes an A/D converter, and generally consists of the following blocks:

Figure 4.8. AtoD boundary

In figure 4.8, f_s denotes the sampling frequency and N the number of bits. The anti-aliasing filter is required only if the continuous or sampled analog signal is not already sufficiently bandlimited. If the input is a continuous analog signal that is not sufficiently bandlimited the anti-aliasing filter may be:

- a continuous analog low-pass filter (LPF)
- a simple continuous analog low-pass filter followed by a precise switched-capacitor low-pass filter.
- a simple continuous analog low-pass filter followed by oversampling (sample-and-hold, S&H, at a frequency several times greater than Nyquist) and then followed by a precise digital low-pass filter and downsampling.

The last alternative is illustrated below [OppA89]:

Figure 4.9. AtoD boundary with decimation.

Sometimes a sample-and-hold circuit in which a capacitor charges for say $t_s/2$ can act as part of the anti-aliasing filter.

Not only will the component library select between these alternatives, but also it must be able to return interesting choices for sampling rate, f_s , and precision, N . Increasing f_s has the advantage of producing less aliased noise. As well, if the boundary is in the critical delay path, it means a faster throughput and less delay between system input and output. However, taking fewer samples per second has the advantage that the A/D converter and subsequent digital blocks may be slower which may allow more serial implementations and therefore, require less area. The larger N is, the less the quantization noise; however, the larger the A/D for partly or completely parallel implementations, and the greater the conversion time for partly or completely serial implementations. These tradeoffs are summarized in figure 4.10. The selected value for f_s is used to select between the different styles of A/D converters: flash, successive approximation, dual slope, etc.

	Implications	Advantage	Disadvantage
Greater f_s	faster conversion time faster subsequent digital blocks less aliased noise	less delay less delay less noise	more area more area
Larger N	larger A/D <parallel> longer conversion time A/D <serial> less quantization noise	 less noise	more area more delay

Figure 4.10. Implications of adjusting f_s and N

Similarly, a DtoA boundary, as referred to in this thesis, includes a D/A converter, and in general consists of the following constituent blocks:

Figure 4.11. DtoA boundary.

For a continuous analog output, the reconstruction filter may be one of the following:

- an analog low-pass filter (LPF)
- a precise switch capacitor low-pass followed by a simple analog low-pass filter
- a filter in the digital domain using interpolation (increasing the sampling rate) as shown below [OppA89]

Figure 4.12. DtoA boundary with interpolation.

If a sampled analog output is desired, then a reconstruction filter is not needed.

In summary, the component library function that returns viable styles for AtoD and DtoA boundaries must decide on how the filtering functions will be performed, on the style and precision of the converter to be used and on the sampling rates to be tried.

So far, only analog and digital domain signals have been considered. That is, *time domain signals* for which data information are contained in time relationships of the signal variations and not in amplitude variations have not been mentioned. *Time domain signals* have a high and low logic level region and the information is contained in either the period between transitions, the pulse widths, or in the frequency of transitions. An example of the latter, is pulse density modulation (PDM) and is illustrated in figure 4.13. Transition from analog to PDM occurs via an oversampled A/D. The work that follows emphasizes analog-to-digital and digital-to-analog domain transitions; however, it may easily be extended to include transitions from and to *time domain signals*. The goal of the methods in chapter 5 are to limit expensive boundary conversion hardware. Thus conversions such as those from sampled analog to continuous analog and vice versa, which may require a filter are not considered to be boundaries. The filter is simply part of the blocks' design and is decided on by the component library.

Transitions between the different domains discussed are summarized in figure 4.14. Those transitions which require expensive hardware and for which their numbers should be limited are noted. These transitions are singled out as boundaries.

Figure 4.13. Example of a PDM (an oversampled) signal.

(a) The continuous analog signal
 (b) The PDM signal where '1' = 1
 '0' = 0

Signal Domain		Considered a Boundary in Chapter 5	Hardware
S1	S2		
CA	SA	No	AA filter ¹ , S&H
	D	Yes	AA filter ¹ , S&H, A/D converter
	PDM	Yes	AA filter ¹ , S&H, Oversampled A/D
SA	CA	No	Reconstruction filter ²
	D	Yes	A/D converter
	PDM	Yes	
D	CA	Yes	D/A converter, Reconstruction filter ²
	SA	Yes	D/A converter
PDM	CA	No	LPF
	D	No	Counter

where CA = continuous analog, SA = sampled analog, D = digital, PDM = pulse density modulation (oversampled analog).

¹ required only if S1 is not sufficiently bandlimited.

² required only if subsequent blocks pass high frequencies.

Figure 4.14. Conversion Boundaries and Hardware requirements.

4.2 A Simple Example

In this section a simple example is considered. It illustrates the issues involved in finding a solution. Chapter 5 will describe several methods for tackling these issues.

The "mixed" system considered is a wattmeter used to measure average power at 60 Hz and is shown in figure 4.15. The user defined input BBD and input specifications are shown in figure 4.16. This figure shows a long integration to calculate watt hours. This must be implemented in digital due to the long storage time required. The designer specified analog inputs, an analog output proportional to average power, and a digital output showing total energy (watt-hours) for six months. From these constraints, only the multiplier and averager may be implemented in either domain; thus, at this point the domain of these blocks and the location of the boundaries is not determined. Figure 4.17 shows several sample implementations which use different domain implementations for different blocks and thus, require different analog-to-digital (AtoD) and digital-to-analog (DtoA) boundaries. Figure 4.18 shows the four possible combinations of boundary locations. As well, it demonstrates the number of possible solutions when all combinations of all the different viable styles for each block are considered. For this small example, there are 8180 different solutions in total. These solutions do not include all the variations of sharing and not sharing boundaries of the same type nor the special cases. In this thesis, shared boundaries means sharing only the converters that make up the boundaries; i.e. the sample-and-hold circuit and anti-aliasing filter are not shared, rather, they must be replicated. Special cases are those for which a block performs a function as well as a domain conversion, such as, the multiplying digital-to-analog converter (MDAC) shown in figure 4.17b of the sample implementations. Clearly, performing the lower level detailed designs of all these solutions is not feasible since operations such as simulation and verification performed at the lower levels are

computer intensive. The number of solutions considered can be reduced by limiting the number of styles for each block considered and the number of different locations considered for the boundaries. Of the remaining solutions, the advantage of the different signal modes must be weighed. Using the scores for all the N attributes, described in the previous section 4.1.1, a measure of performance for individual designs is determined using the following:

$$\text{Total Score} = \text{TS} = \sum_{j=1}^N S(V_j) w_j$$

The Total Score, TS, function combines the scores (numerical values representing the penalty or credit) of each attribute value of a design with the relative importance (designated by a weight w_j) of each attribute. The TS is used to evaluate and compare designs and thus, to suggest good design solutions. Of the many solutions, two have been selected to illustrate the use of the TS function. They are shown in figure 4.19. Note that different weights result in the selection of different alternatives.

The use of weights in the total score (TS) function may seem unnecessary. Their effect can simply be included in the individual score functions. However, exploring the design space in terms of changing the TS function would mean redefining and entering all the changed penalty-credit functions. Therefore, it is suggested that each score function be used to describe the effect of changes in a specific attribute with respect to itself. For example, the area score would define what changing area, relative to the reference or required area, means in terms of penalty or credit. This can be done for each attribute without considering what the score values mean with respect to other attributes. Then the weights can be used to define this relative importance. That reduces exploring the design space to simply varying the weights. It is not assumed that defining score functions is easy. It certainly requires some expertise. However, intuitively, functions dealing with the effects of changing

one attribute at a time are easier to deal with than functions considering similar changes in many attributes.

As mentioned above, defining good score functions is not easy. The same can be said about setting weights. Designers know what the penalty-credit curves should look like and understand the relative importance of different attributes and therefore, understand which weights should be greater than others. However, designers find it difficult to set actual score and weight values. To learn about setting these values, it may be helpful to ask a lot of expert designers to set these values for past designs. However, the values are very application specific and therefore, would have to be modified for new designs. Alternatively, the values may be set, a solution obtained, and then based on this resulting solution, the values may be modified. This process can proceed until a satisfactory solution is obtained. In this way, 'good' values are obtained iteratively. For example, in the wattmeter example of section 4.2, assume that a solution was obtained for which the area required was just a little over that of the smaller die so that the larger die must be used. Then, if the weight for the area attribute was slightly increased, the resulting solution may sufficiently reduce the area so that the smaller die is used and a better TS is obtained. Note that changing values leads to exploring a different part of the solution space. Further study of the topic of weight and score setting is not included as part of this thesis.

(a)

The wattmeter displays average power

Average Power = Average over T_1 of $I(t) * V(t)$, $T_1 \sim 1$ second

(b)

Figure 4.15. A hydro wattmeter.

(a) the wattmeter system showing the external inputs

(b) the IC block diagram and outputs. This is what would initially be specified by the system designer.

(a)

(b)

Figure 4.16. User Inputs

- (a) The input BBD. This is the circuit after the first pass (Sec chapter 5). The blocks (domains) that are forced to be analog or forced to be digital have been specified.
- (b) Input specifications. These are the detailed scores (see section 4.1.1) which tell the system to trade-off area vs error vs power dissipation as it finalizes the design.

Figure 4.17. Sample wattmeter implementations.

Assume that the component knowledge base returns for each of B_1 (*) and B_2 (averager)
 3 viable analog styles
 5 viable digital styles
 for B_3 (integrator)
 5 viable digital styles
 and for each boundary
 4 viable styles

Four locations for the AtoD and DtoA boundaries are shown below. The number of possible solutions is shown beneath each sketch, based on the number of styles for B_1 , B_2 , and B_3 assumed above.

Total = 8180 !!

Sample calculation: For (a), B_1 has 3 viable styles since its domain is analog. Similarly, B_2 has 3 styles. B_3 is digital with 5 styles. The one AtoD boundary can have 4 styles (types of AtoD).

AtoDs occurring more than once are time-shared (c and d). Similarly, for DtoAs.

Figure 4.18. Four possible locations for boundaries and for each, a calculation for the possible number of solutions.

This is one of the 180 solutions from (a) of figure 4.18:

Block Bi	Area mm ²	Power mW	Errorout Full Scale
B1	.1	30	1
B2	.05	15	1.02
B3	1	55	1.04
B4	1.6	.05	1.04
Total	2.75	100	1.04
Score	0.5	0.3	-2.5

if all w_j 's = 1,
 then $TS = \sum_j V_j = -1.7$ (best choice)
 if $w_{err} = 1$, $w_{power} = 0.5$, $w_{area} = 0.3$,
 then $TS = \sum_j V_j = -2.2$

This is one of the 2000 solutions from (c) in figure 4.18.

Block Bi	Area mm ²	Power mW	Errorout Full Scale
B1	1.0	55	.05
B2	.66	.1	.05
B3	.9	.05	.05
B4	1.6	.05	.05
B5	.36	.1	.05
Total	4.52	55	.05
Score	-7	1	3.5

if all w_j 's = 1,
 then $TS = \sum_j V_j = -2.5$
 if $w_{err} = 1$, $w_{power} = .5$, $w_{area} = .3$,
 then $TS = \sum_j V_j = 1.9$ (best choice for low
 error but larger area)

Figure 4.19. Total Score evaluation for two implementations. The "Score" can be found from the "Total" value at the bottom of each column using the curves in figure 4.16b.

In the following sections, the design approach taken generates good initial solutions using estimates and heuristics. The idea is to arrive at these good initial solutions relatively quickly and eliminate solutions that are unable to meet specifications. As shown in figure 4.20, the heuristic solution(s) are near the global maximum and thus, reduce the total amount of fine tuning (design iteration) required to arrive at the global maximum. Clearly, figure 4.20 is not a precise representation of the search (TS) space; however, it is simply used to convey the idea behind the design approach. The x-axis is a simplification of the multi-dimensional space of the independent variables, all the attribute scores and the weight settings. Since the penalty-credit functions are piecewise linear, the TS space would be piecewise smooth.

Consequently, the relatively little computing used to obtain the good initial solutions saves a large amount of computing time at the lower levels of the design process. Note, however, that the approach is based on the assumption that expert designers have a "good feel" for the design space and can incorporate design strategies and heuristics in the component knowledge base that can arrive at a reasonably good solution [HarR89].

Figure 4.20. Reasoning behind design approach [HarR89].

4.3 Chapter Summary

This chapter described the system level inputs. These include both system-supplied and user-supplied inputs. A method for propagating the user-supplied input/output specifications of dynamic range and bandwidth is given. The penalty-credit functions were described and used for specifying user-supplied attributes, such as, area and delay. The 'SumOf' function was introduced for the calculation of total system values for the attributes. The specifics of the 'SumOf' function were given for area, power dissipation, signal delay and SNR.

The component knowledge base, the system-supplied input, was described, focussing on AtoD and DtoA boundaries.

As well, this chapter describes a small example. This example demonstrates the factorial nature of the design space and thus, the importance of reducing it. Removing one set of boundary locations removes many possible solutions. The total score function was introduced as a measure of performance for individual designs.

CHAPTER

5. The System Level of the Design Hierarchy

Starting with a block diagram at the system level, this chapter proposes a method that suggests good locations for the boundaries between digital and analog sections. The locations of these boundaries are determined by first minimizing the number of boundaries, and then finding sections of the system that could be advantageously implemented in analog. As well, a method for selecting the implementation of the blocks from standard cell libraries, silicon compilers or performance estimates of what can be custom designed is discussed. Finally, a method is introduced for ordering the system blocks in decreasing likelihood of causing the overall design to fail because their final specifications fail to satisfy initial estimates.

5.1 Algorithm and Assumptions for Finding Initial Solutions

The flow diagram describing the algorithm for finding good initial choices for block implementations is given in figure 5.1. The following sections describe the different parts of the algorithm in more detail.

Basically, the algorithm begins by deciding on the domain of implementation (continuous-time or discrete-time analog, or digital) for each block. This is accomplished by deciding on the location of the conversion hardware: the hardware required at the interfaces between analog and digital sections. Then an exhaustive search is done with all viable *styles* of each block considered. The branch-and-bound technique is used to limit, on average, the search. Several possible solutions are found for each different location for the conversion hardware. From these, the user selects one for refinement down the design hierarchy (the default choice is the

Figure 5.1. Algorithm Flow Chart

Figure 5.1 (continued). Algorithm Flow Chart

highest total score initial solution). Once the choice is made, an algorithm is run to decide the order in which the different system blocks should have their lower level design attempted.

The following assumptions are made in this thesis:

- the behavior block diagram (BBD) has no feedback, i.e. it is an acyclic graph, and
- the BBD has no reconvergent fanout.

The extensions of the methods to include these behaviors is left as future work.

5.2. The First Pass Through the BBD

The first pass through the BBD determines the derived function specifications for each block. Then it determines the blocks that must be implemented in analog or must be implemented in digital. The order in which blocks are visited must be set for both a forward search from inputs (roots) to outputs (leafs) and a backward search from outputs to inputs. In the forward search, a component is considered only after all its inputs are visited, that is, only once all its ancestor components have been processed. Whereas, in the backward search, a component is considered only after all its outputs are visited, that is, only once all its descendents have been processed. An outline for the algorithm used to set up the order of visiting, is given in figure 5.2. A modified depth-first search algorithm is used. A list of roots and leafs is available and the algorithm proceeds, for the forward-visit list, as follows: starting at a root, proceed using depth first search, adding each visited component on the forward-visit list. Proceed until a component is reached for which all ancestors have not been visited. Then, start with another root. An example is given in figure 5.3.

5.2.1 The 'Interesting' Signal Bandwidth for Noise Estimates

In the first pass, the derived function specifications are determined with all sources of error introduced by components ignored. The determination of signal bandwidth (BW) and signal level are considered here. For BW, the following distinction is made between the actual signal bandwidth and the 'interesting' signal bandwidth:

BW_S = the actual signal BW including all the signal information, both pertinent signal information and noise.

BW_{IS} = the 'interesting' signal BW, that is, the part of BW_S , that contains pertinent signal information.

Create forward-visit list	Create backward-visit list
ForwardList () BEGIN Initialize all nodes to not visited; For each root in nodes MakeFlist (root); END; MakeFlist (parent) BEGIN Add <i>parent</i> to forward-visit list; Set <i>parent</i> status to visited; For each child of <i>parent</i> If all parents visited Then MakeFlist (child); END;	BackwardList () BEGIN Initialize all nodes to not visited; For each leaf in nodes MakeBList (leaf); END; MakeBList (child) BEGIN Add <i>child</i> to backward-visit list; Set <i>child</i> status to visited; For each parent of <i>child</i> If all children visited Then MakeBList (parent); END;

Figure 5.2: Algorithms for creating ordered lists for forward and backward BBD passes.

Given:

List of roots: A, G.

List of leafs: E, F, H.

Resulting lists:

Forward-visit list:

A, B, G, C, D, E, F, H.

Backward-visit list:

E, F, D, H, C, B, A, G.

Figure 5.3. Sample forward-visit and backward-visit lists.

BW_{IS} is required for the evaluation of certain performance criteria for partly finished designs. For instance, if SNR is used as a performance measure, and $BW_{S,Out}$ is 0-50 kHz for a function block and is followed by a filter with a 4 kHz cutoff, then $BW_{IS,Out}$ is 0-4 kHz. This $BW_{IS,Out}$ is used in the evaluation of SNR (i.e., in determining signal and noise powers). In this way, a design is not erroneously considered bad or unacceptable because noise within the 4-50 kHz range, which is later filtered out, is used in the calculations.

The forward pass through the BBD propagates the BWs from system inputs to system outputs, requiring the component knowledge base (CKB) to determine $BW_{S,Out}$ after each block. The backward pass from system outputs to system inputs sets $BW_{IS,Out}$ equal to the union of the $BW_{IS,In}$ of its children, and requires the CKB to derive $BW_{IS,In}$ from $BW_{IS,Out}$. The example in figure 5.4 illustrates the above. Whereas, $BW_{IS,Out}$ is used for performance evaluations (calculation of TS); $BW_{IS,In}$ is useful in the design process because it focusses on the frequencies of interest, and therefore better directs the selection of the design alternatives.

5.2.2 Signal Level

Signal level or dynamic range (DR) must also be determined. One quick estimate of determining minimum and maximum signal level values requires taking the minimum and maximum input values and propagating these minimum and maximum values through the system. For instance, let A be a subset of the real numbers $\mathcal{R}$, such that:

$$A = [a_1, a_2] = \{t \mid a_1 \leq t \leq a_2; a_1, a_2 \in \mathcal{R}\}$$

A is called a closed real interval. Here A defines the values a signal can take, where a_1 is the minimum and a_2 is the maximum value. From interval analysis [AleG83], operations of $+$, $-$, $*$, and $/$ on intervals $A=[a_1, a_2]$ and $B=[b_1, b_2]$ may be calculated explicitly as:

$$A+B = [a_1+b_1, a_2+b_2]$$

$$A-B = [a_1-b_2, a_2-b_1]$$

$$A*B = [\min\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}, \max\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}]$$

$$A/B = [\min\{a_1/b_1, a_1/b_2, a_2/b_1, a_2/b_2\}, \max\{a_1/b_1, a_1/b_2, a_2/b_1, a_2/b_2\}]$$

where $\emptyset \notin B$ in the case of division.

As shown above, interval analysis considers all combinations of the two intervals' minimums and maximums. For rational functions and polynomials, interval analysis results in a range which includes the range of the true function. Thus, interval analysis is not exact, rather, it defines a bound. As well, interval evaluation of a function f is dependent on the choice of expression for f . Consider the following examples:

Example 1.

$$f(x;a) = ax^2$$

$$\text{where } X = [-2,4], A=[1,2]$$

Interval analysis leads to

$$f(X;A) = [1,2] [-2,4] [-2,4] = [-16,32]$$

whereas the true function range = $[4,32]$, which is a subset of $[-16,32]$

Example 2.

$$f(x;a) = a - \frac{x}{1+x}, \quad x \neq -1$$

where $X = [-0.5,1]$, $A = [2,3]$

Interval evaluation leads to

$$[2,3] - \frac{[-0.5,1]}{1 + [-0.5,1]} = [0,4]$$

whereas the true function range = $[1.5,4]$, which is a subset of $[0,4]$.

The true function range is obtained by forcing each occurrence of x to take on the same value throughout the expression, and considering all values for x and a within X and A , respectively. Interval evaluation is exact for a function $f(x_1, x_2, \dots, x_n; a_1, a_2, \dots, a_m)$ if each x_i and a_j occurs only once in the expression, where here x_i^2 means x_i occurs twice.

Much more work has been done with interval analysis [AleG83]; however, it does not always produce the exact function range and so is not considered further. Rather, when interval analysis is not exact, the function's extrema are calculated. Although, this method is time consuming, it is performed once per function (i.e. it does not depend on the particular *style*) and therefore, the time is justified. Note that for monotonic unary operations such as $\ln x$, e^x , x^n for $x > 0$, etc., interval analysis is exact.

Signals may take on values from several intervals. For instance, for division, B may be defined on two intervals $[b_1, b_2]$ where $b_1, b_2 < 0$ and $[b_3, b_4]$ where $b_3, b_4 > 0$. In this way, the signal takes on both positive and negative values. For such cases, all combinations of minimum and maximum from both intervals of B are considered with the minimum and maximum of interval A .

When solving for dynamic range, some very large integers, several orders of magnitude larger than the inputs, may result. This is usually because of a real pole in the algorithm. Not considering what to do for inputs which give this pole is an error on the part of the algorithm designer. In such cases, the program would query the user, asking if the user wants to define a new output dynamic range and, if so, asks what to do about values outside this range. Options on the treatment of such values may be provided by the program. Options may include turning on an error indicator, switching algorithms, or simply ignoring those values. Accuracy requirements must be met by this new output range and not by values outside it.

Figure 5.4. Calculation of BWs and BW_{is} requires two passes through the BBD.

The backward pass follows the forward pass.

Calculations of noise and signal power are performed on BW_{is,out}.

Once all the functional specifications are available, the CKB decides for each block whether it can be implemented in one or both of the analog and digital domains. This is done by comparing the library specifications of various types of analog and digital circuits to the block specifications. Some of the reasons which lead to must-be-digital components are the need for high accuracy, long-term storage, extensive programmability, or testability. On the other hand, low area, high speeds, and analog system inputs/outputs are some of the reasons leading to must-be-analog components. Refer to chapter 2 for more detail.

5.2.3 Algorithm for First Pass

The knowledge for determining the derived function specifications and the domains of implementation for each block is stored in the CKB. Using input data and block specifications, the CKB determines the output data, output dynamic range, BW_S , and BW_{IS} for each block. Also, by comparing library specifications of various types of analog and digital circuits to block specifications, the CKB determines each block's allowed domains of implementation. The development of the CKB and its associated knowledge is beyond the scope of this thesis; an object-oriented database seems naturally suited and therefore should be considered. Refer to section 4.1.3 figure 4.7 for an illustration of the CKB. An outline of the algorithm for the first pass is given in figure 5.5

```

/*
* CKB = component knowledge base
*
* Forward Pass: determination of  $BW_S$ 
* Using list of blocks for forward search
*
*/

For each block in forward-visit list
  BEGIN
    For each input
      Set  $BW_{S,In}$  to  $BW_{S,Out}$  of its parent;

      Request CKB to determine  $BW_{S,Out}$ ;
    END;

/*
* Backward Pass: determination of  $BW_{IS}$ 
* Using list of blocks for backward search
*
*/
For each block in backward-visit list
  BEGIN
    If leaf of BBD Then
       $BW_{IS,Out} = BW_{S,Out}$ 
    Else
       $BW_{IS,Out} = \text{Union } [BW_{IS,In} \text{ of children}]$ ;

    Request CKB to determine  $BW_{IS,In}$ ;
    Request CKB to determine possible domains of implementation;
  END;

```

Figure 5.5. Algorithm outline for determining BW_S and BW_{IS}

5.3. Initial Location of Analog-Digital Boundaries

Once the must-be-analog and must-be-digital components are known, the locations of the analog-digital boundaries are selected. The number of boundaries required is determined by the following heuristic: "As few boundaries as possible". This heuristic is based on the fact that analog-to-digital and digital-to-analog conversion hardware is very expensive¹ and consequently, it is seldom advantageous to add extra conversions above the required ones. Also extra, not needed, conversions means more signal degradation. The use of this heuristic is important in reducing the large design space since it excludes (prunes out) many different boundary locations and therefore, many more possible solutions. (Refer back to the example of section 4.2 for an indication of the large size of the design space).

Since most circuits have sections between the required analog and digital blocks which could go either way, the placement of the boundaries must be determined. At this stage, the heuristic "as much analog as possible" is used. Experience shows that if accuracy requirements are satisfied, analog circuits are usually faster and take up less area than their digital alternatives.

5.3.1 Algorithm for Initial Boundary Placement

To determine the location of the boundaries, the BBD is first divided into *sections*. Each *section* is bordered by blocks that have a forced domain implementation; thus, these blocks are the *sections* inputs and outputs. All other *section* blocks may be implemented in either domain. Four *section* types are considered:

¹ Conversion of complex signals which requires several bits of resolution for A/D or D/A converters apply; comparators, or 1 bit conversions, do not apply and are not treated.

Direction of signal flow
A boundary
Direction of boundary movement

branch points are blocks (nodes) with greater than one parent and/or greater than one child.

Figure 5.6. The *section* types.

Type (1): *section* inputs and outputs all analog or all digital;

Type (2): *section* inputs FD and *section* outputs FA;

Type (3): *section* inputs FA and *section* outputs FD.

Figure 5.6. (continued). The *section* types.

Type (4): *section* inputs FA and FD and/or *section* outputs FA and FD . The arrows show the direction the boundaries will be moved to try to optimize the circuit. At the start it is assumed that the maximum amount of analog circuitry will be the best (smallest and fastest). As optimization proceeds, the boundaries are then moved to encompass more digital circuitry.

- 1) *sections* with no boundaries required, i.e. all inputs and outputs are forced analog (FA) or all inputs and outputs are forced digital (FD).
- 2) *sections* with all inputs FA and all outputs FD.
- 3) *sections* with all inputs FD and all outputs FA.
- 4) *sections* with both FA and FD inputs and/or with both FA and FD outputs.

These different *section* types are illustrated in figure 5.6. Figure 5.6 also indicates the initial location of boundaries and the type of boundary: AtoD or DtoA. Examples of each *section* type are also included. Clearly, all initial boundary locations were chosen to maximize the number of analog components.

Figure 5.7. Sharing Boundaries

- (a) the AtoD boundary is shared, i.e. the A/D converter is time-shared.
 (b) the AtoD boundary is not shared allowing slower conversion times.

Several assumptions are made about the boundaries. First, it is assumed that all AtoD boundaries and all DtoA boundaries of a *section* are time-shared boundaries, that is, the A/D or D/A converter is time-shared between two or more inputs. However, the sample-and-hold circuitry and filters are replicated. Even with this overhead, a shared boundary counts as one boundary. Therefore, referring to figure 5.7, if data-conversion times are not achievable with shared hardware (5.7a), then only locations where the hardware is not shared (5.7b) will be considered. This reasoning is in accordance with the heuristic that a minimum number of

boundaries should be considered. Only if no solution is found, would additional boundaries be considered (not covered in this thesis).

Second, it is assumed that boundaries from different *sections* are not shared. The success of sharing such boundaries depends not only on achievable data conversion times, but also on the success of routing sensitive analog lines. To see this, first consider a *section* with two data conversions. For such a *section*, a possible layout that would separate noisy digital signals from sensitive analog ones would be as shown in figure 5.8.

Figure 5.8. Sample *section* layout

Next consider the system in figure 5.9. Here three sections time-share the same boundary. Let A_i and D_i represent the different analog and digital *sections*. Let the signal flow be such that $A_1 \rightarrow D_1 \rightarrow A_2 \rightarrow D_2 \rightarrow A_3 \rightarrow D_3$. The long analog lead from *section* A_3 to the shared AtoD may cause problems. As well, the signal may be degraded by the undesired coupling of digital signals. This degradation in the signal is difficult to quantize because it is highly sensitive to the circuit layout. Guidelines are used to minimize crosstalk [WarG88]. On the other hand, sharing between two *sections*, here A_1 and A_2 , should not cause any problems if layed out and routed properly. Future work should consider such alternatives. The decision to not share boundaries between *sections* has been taken at this stage in the tools development until further study of the subject is undertaken.

Figure 5.9. Sample layout of a system with four *sections* . The long analog lead from A₃ to the shared AtoD may cause problems.

Lastly, it is assumed that every data path from any *section* input to any *section* output has at most two data conversions. Even though extra conversions may sometimes be realized by sharing boundaries and thus, not increasing the amount of hardware, the author has not seen any such designs to date. These extra conversions may increase signal degradation.

Notice that in figure 5.6, *section* types (1), (2), and (3) allow boundary movement so that all blocks, except inputs and outputs, will be considered to be implemented in the analog and in the digital domain. However, for *section* type (4) only certain cases, such as the example for type 4a, allow the same movement. For other cases, such as the example for type 4b, DtoA boundaries may not be moved from FD blocks all the way to the FA inputs and outputs without causing greater than 2 conversions in a given data path. For these cases, a second set of initial boundary locations is considered; this will exercise all AtoD and DtoA boundary locations that limit 2 conversions per data path within one *section*. These particular cases

all exhibit the same quality: they all possess a 'special' branch point. A branch point is a block (node) with greater than one parent and/or greater than one child. A 'special' branch point has the following properties (within a *section*):

- i) It is a branch point of the graph.
- ii) The branch point has both FA and FD ancestors and both FA and FD descendants.
- iii) At least one parent or at least one child of the branch point has FA and FD ancestors or FA and FD descendants, respectively.

An example of a system with four *sections* is shown in figure 5.10. Figure 5.10b and 5.10c show the two systems with the initial boundary locations to be considered.

It becomes clear from this example that only if a BBD contains special branch points, will it produce more than one system, each with different initial boundary locations. In fact, if there are i *sections* with a special branch point, then there will be 2^i BBDs with different initial boundary locations. In most applications there are few *sections* and only some of these have special boundaries so that only a few BBDs will be generated.

The set of initial locations for the boundaries is described next. For *section* types 2, 3, and 4a, AtoD boundaries are placed above FD outputs and DtoA boundaries are placed below FD inputs, where applicable. For *section* type 4b, the location of boundaries is such that the AtoD and DtoA boundaries are moved as far away from the FA inputs and FA outputs, respectively, without causing any *section* path with greater than two boundaries. The determination of the locations for *section* type 4b is considered next.

Figure 5.10. A system with four *sections*.

Section 2 has a special branchpoint and therefore, produces two systems with initial boundary locations.

5.3.1.1 Determination of Initial Boundary Location for Section Type 4b.

For *section* type 4b, the initial placement of boundaries is as follows:

- AtoD boundaries are moved as far away as possible (downwards) from the FA inputs, and

- DtoA boundaries are moved as far away as possible (upwards) from the FA outputs,

without causing greater than two boundaries in any *section* path. This is achieved by initially placing AtoD boundaries below FA inputs and DtoA boundaries above FA outputs. Then, the AtoD boundaries are moved down just above the nearest branch point and DtoA boundaries are moved up just below the nearest branch point. Then, movement across the branch point only occurs if the move does not cause greater than two boundaries in any *section* path. If movement is allowed the boundaries are moved once again further from the FA input or output to the next branch point. The process repeats until movement across all branch points is not allowed. This then defines the initial placement of boundaries.

Movement across a branch point must not cause greater than two boundaries in any one *section* path. Therefore, movement is valid only whenever all of the following conditions hold within a *section* :

- 1) the branch point is not of a forced domain. Such a point must be a *section* input or output and boundaries only move within a *section* .
- 2) all branch point parents with FA only ancestors are AtoD boundaries. See figure 5.11 for an example.
- 3) all branch point children with FA only descendents are DtoA boundaries. (Complement of condition 2)
- 4) all branch point *parents* with FA and FD ancestors satisfy the following:

If (all children of branch point are not DtoA boundaries) Then

parents must be AtoD boundaries.

See figure 5.12 for an example

- 5) all branch point *children* with FA and FD descendents satisfy the following:

If (all parents of branch point are not AtoD boundaries) Then

children must be DtoA boundaries.

(Complement of condition 4)

Conditions (2) and (3) ensure that all boundaries are properly situated before proceeding. Whereas, conditions (4) and (5) prevent the occurrence of more than two boundaries in any one *section* path. Figure 5.13 gives a few examples illustrating these conditions. The condition that is active at points of interest is indicated.

Figure 5.11. Demonstration of the use of condition 2

Figure 5.12. Demonstration of the use of condition 4

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

F : condition does not hold
T : condition holds

Figure 5.13. Examples illustrating how branch point conditions are used to control movement.

Results of conditions are taken when the appropriate conversions are situated at the tip of the arrow (above or below branch point).

If all conditions hold, the boundary movement across the branch point consists of the following:

```
For every branch point parent Do
  If (branch point parent is an AtoD boundary) Then
    remove it
  Else
    insert a DtoA boundary between branch point and
    branch point parent;

For every branch point child Do
  If (branch point child is a DtoA boundary) Then
    remove it
  Else
    insert an AtoD boundary between branch point and
    branch point child;
```

Whenever a boundary is created from moving across a branch point, it is then again pushed further from the FA input or output to the next branch point. Movement across this branch point is also attempted. This process repeats itself until no further movement is valid.

The pseudo code which generates the different initial boundary locations is found in Appendix C.

5.4 Finding an Initial Solution

Once all the function specifications are obtained and the locations of the conversion hardware and the domains of the blocks are determined, a search for the highest total score solution, consisting of the best combination of block *styles*, begins. This search is exhaustive; however, the exponential nature of this search is greatly reduced if there is a tight area constraint which disallows most digital block implementations, or a high accuracy requirement which disallows most analog implementations. Also, the branch-and-bound technique [HilF74] as well as other methods of pruning are used to limit the number of combinations attempted.

The exhaustive search successively considers the blocks of the behavior block diagram (BBD) in the order given by the forward-visit list obtained in section 5.2. Figure 5.14 shows how for every viable *style* of a block, all other viable *styles* of every other block are considered. A different set of parent-child pointers are required for the search tree and for the BBD parent-child links. The search tree links are used to generate different solutions; whereas, the BBD links are used to obtain a child's input data information from its parent(s) output data information.

The different solutions are compared using the total score function described in section 4.2. The branch-and-bound technique directs the search by always progressing with the highest score partial solution. This is illustrated by the outline of the algorithm given in figure 5.15 line 8. In addition to using the highest TS of the partial solutions of the current search as a bound, the TS from previous solutions is used (line 9). That is, the best TS resulting from previous exhaustive style searches for other boundary locations serves as a bound in the current search. Though in the worst case branch-and-bound is exhaustive, on average it is much better.

(a)

Each path from root to leaf represents a different partial solution

(b)

Figure 5.14. Exhaustive search considers all combinations of all viable *styles*.

(a) BBD; (b) exhaustive search tree

Note how the search tree sometimes does not follow parent-child edges in the original BBD. This is why the pointer from C5 to B3 is needed.

Exhaustive search always proceeds with highest TS. The step numbers describe the progression of the algorithm, i.e.

Step 1 Calculate TS for A1 and A2

Step 2 Calculate TS for partial solutions (A1,B1), (A1,B3), (A1,B7). (50>41).

Step 3 Calculate TS for partial solutions (A1,B3,G3), (A1,B3,G4). (48>44>41>35)

etc ...

(c)

Figure 5.14 (continued):

(c) progression of exhaustive search algorithm.

ExhaustiveSearchWithBranchAndBound ()**BEGIN**

/*

let $c[1]$, $c[2]$, ..., $c[i]$, ..., $c[N]$ index the nodes of the forward-visit list.

CKB = component knowledge base.

*/

1 Request CKB for viable *styles* of $c[1]$.2 Create root nodes for these *styles*.

3 For each root

BEGIN

4 Calculate attribute values, SumOf V_{ji} , pruning out unacceptable solutions, i.e. solutions for which one or more attributes take on values in the unacceptable value range: $S(V_j) \rightarrow -\infty$;

5 Calculate TS;

END;

6 Until a solution(s) is found Do

BEGIN

7 Find partial solution with highest TS

/* partial solution has $c[i]$ as last block processed */

8 For highest TS partial solution

BEGIN

9 If (TS is lower than the best TS from previous solutions)
Then suggest not proceeding to designer.10 Request CKB for viable *styles* of $c[i+1]$;11 Create nodes for these *styles* and add as *children* of partial solution;12 For each *child*

BEGIN

13 Calculate attribute values pruning out unacceptable solutions;

14 Calculate new TS;

END;

END;

END;

END;

Figure 5.15. Algorithm for finding the highest score initial solution using exhaustive search with branch-and-bound.

Line 4 and 13 of the algorithm need further explanation. The calculation of attribute values uses the SumOf function described in section 4.1.1 except that the indices are changed, that is, the following is used:

$$S(V_j) = S\left(\sum_{i=1}^k V_{ji}\right)$$

where

$k < M$,

k = number of components selected so far for the partial solution under consideration

M = total number of components in the BBD.

If at any point $S(V_j)$ is unacceptable, for example area exceeds the maximum allowed, the corresponding branch is pruned.

The algorithm requires the component knowledge base (CKB) to select and return viable *styles* for each block of the forward-visit list (line 1 and 10). The different *styles* present various tradeoffs between specifications. The component knowledge base uses the block specifications to select appropriate *styles*. Clearly, the fewer the *styles* returned, the lesser the number of solution combinations. Each selection algorithm in the CKB must at least perform structural discrimination [HarR89], that is, it must only return viable *styles*. For instance, if differential outputs are required in an operational amplifier, a design *style* with only one output is not selected. Or, for NMOS current mirrors, cascode design *style* is not feasible if the output voltage compliance is less than one threshold voltage [HarR89].

Selection algorithms should be able to bound the range of values for different attributes. For instance, using the dynamic range of the input, the algorithm should be able to decide on the range for the number of bits, B , of a digital component. With no extra knowledge, the largest range would set the lowest value for B to that which just

would make the solution acceptable, and the highest value for B to that for which anything higher does not have a significant effect on output dynamic range. As well, it might increase a word width in increments of 20% rather than a single bit at a time. The limits and increment, if applicable, of all attribute ranges show up in each associated blocks specification form (refer to 4.1.2). The designer may override any of these values. With the ranges specified, the program then selects all *styles* for which all attribute values come within the specified bounds. The ramification bush used in [ZhuX88] may be used for this selection process. Several commercial packages such as Analog Devices Component Selection disk [DevA89], when supplied with the attribute ranges, already perform this type of *style* selection.

Future work could focus on further pruning of the search tree to enhance the performance of the algorithm. One idea would require the following addition to the algorithm:

Given the current highest score partial solution consisting of m selected components, consider all other partial solutions for which more than m components have been selected. Then, compare the score for each attribute value resulting from selecting the first m components for each of these partial solutions. If all $S(V_j)$ for the current highest score partial solution are worse than all $S(V_j)$ of another partial solution, then prune it. It can never get better than the other partial solution.

Of course, before implementing this, it must be verified that the time required to make this check outweighs the time gained from the number of branches pruned.

5.5 Changing Location of Analog-Digital Boundaries

The previous sections described how the initial locations of the boundaries are determined (section 5.3), and how the exhaustive search of system blocks is performed to get an initial solution (section 5.4). Now, the movement of the boundaries is considered.

For each alternate location of boundaries an exhaustive search over all blocks styles is performed.

The other boundary locations that are considered do not result in a need for additional boundaries. That is, for instance, if block B4 in figure 5.16, which is currently analog, is to be implemented in digital, then block B5 must also be implemented in digital. Respecting this *dependency* of B5 on B4 will not increase the number of boundaries, and will result in exploring fewer design possibilities. In the example shown, the AtoD boundary may move up until just before block B1, and the DtoA boundary may move down until just before B9. All combinations of these boundary locations are considered.

Figure 5.16. Example illustrating boundary movement.

5.5.1 Algorithm for Boundary Movement

Basically, boundary movement leads to a system with a greater number of digital blocks. Thus all AtoD boundaries move up the BBD, either away from FD outputs or towards FA inputs; and all DtoA boundaries move down the BBD, either away from FD inputs or towards FA outputs. Specifically, if the child of a DtoA boundary is not of a forced domain and not a branch point, then the DtoA

boundary swaps location with the child so that the child is now to be implemented in the digital domain. Similarly, if the parent of an AtoD boundary is not of a forced domain nor is it a branch point, the AtoD boundary swaps location with the parent so that the parent is now to be implemented in the digital domain. If a boundary encounters a forced domain block, it simply has reached its final position. If a boundary encounters a branch point, then it may or may not move. Branch points are a little more complex and movement across them is discussed next.

If a boundary encounters a branch point, it may move only if the following conditions hold (within a *section*):

- 1) the branch point is not of a forced domain
- 2) all branch point parents with FD only ancestors are DtoA boundaries. See figure 5.17 for an example.
- 3) all branch point children with FD only descendents are AtoD boundaries. (Complement of condition 2)
- 4) all branch point *parents* with FA and FD ancestors satisfy the following:

If (all children of branch point are not AtoD boundaries) Then

the *parents* must be DtoA boundaries.

See figure 5.18 for an example

- 5) all branch point *children* with FA and FD descendents satisfy the following:

If (all parents of branch point are not DtoA boundaries) Then

the *children* must be AtoD boundaries.

(Complement of condition 4)

Figure 5.17. Example illustrating use of condition 2

Figure 5.18. Example illustrating use of condition 4

Figure 5.19. Examples showing boundary movement.
Note that in (c) the movement cannot expand past either branch point
or more than two boundaries would be needed

Conditions (2) and (3) make sure that all boundaries are properly situated before proceeding. Figure 5.19a,b gives two examples that illustrate this. Condition (4) and (5) prevent the occurrence of more than two boundaries in any one *section* path. Figure 5.19b,c give examples of this, indicating at each branch point which condition does not hold and thus prevents movement.

If all conditions hold, the boundary movement across a branch point consists of the following:

```

For every branch point parent Do
  If (branch point parent is a DtoA boundary) Then
    remove it
  Else
    insert an AtoD boundary between branch point and
    branch point parent;

For every branch point child Do
  If (branch point child is an AtoD boundary) Then
    remove it
  Else
    insert a DtoA boundary between branch point and
    branch point child;

```

The movement of all boundaries within a *section* is as follows. Basically, a boundary moves forward until it meets a forced-domain block (FA or FD) or until it encounters a branch point for which all conditions do not hold. Then it returns to its start location and another boundary advances by one and the first boundary advances again. The process repeats until both boundaries may not move forward. Then they are both returned to their start location, and a third boundary advances. This process continues until all boundaries may not move forward. Then, no other desired combination of boundary locations exists. Figure 5.20 gives an example.

For (a) through (c), the BBD ListOfBoundaries is

For (d) through (g), the BBD ListOfBoundaries is

Figure 5.20. Example of boundary movement within a *section*.

Illustrating use of StartLocList and of RemoveList.

Each picture shows another step in the boundary movement. Thus, boundary 1 moves downward, (b) and (c), until it hits a branch point. There it splits into boundaries 3 and 4, (e) and (f).

For (h) through (j), the BBD ListOfBoundaries is

For (k) through (n), the BBD ListOfBoundaries is

Figure 5.20: (continued). Example of boundary movement within a section. Illustrating use of StartLocList and of RemoveList.

Figure 5.20 (continued): Example of boundary movement within a *section*. Illustrating use of StartLocList and of RemoveList.

From figure 5.20 it is clear that all possible boundary locations with a maximum of two boundaries per any *section* path are explored. In order to exercise all boundary locations, the behavioral block diagram (BBD) list of boundary pointers includes two lists for each boundary: the 'StartLoc' list which holds a list of start or initial locations for each boundary; and a 'Remove' list which points to all boundaries that must be deleted when the current boundary is returned to its start location. The use of these lists is best illustrated by going through the example. First, bndy1 moves forward. At the branch point b6, it creates a new boundary bndy3 (figure 5.20d). bndy3 has a start location set to its current position; bndy4 has its start location set to that of bndy1. Now bndy3 moves forward until it encounters the FA input. At this point it can no longer move forward and thus, returns to its start location and bndy4 advances by one. Once again bndy3 moves forward. In figure 5.20h both bndy3 and bndy4 may not move forward so they return to their initial location and bndy2 advances by one. However, the move of bndy4 to its initial location removes bndy3. The process continues until figure 5.20p where no boundary may move forward.

The algorithm that performs the above description is found in Appendix C. The interesting points to note are listed below:

- Insert new boundaries in the same location on the BBD boundary list as the ones that they are replacing.
- When going past a branch point, only one new boundary holds the original start locations. Other new boundaries have their start locations set to their current location. The one boundary that holds the original start location must be placed after all other new boundaries in the BBD boundary list. For instance in figure 5.20d, bndy4 has its start location set to the original start location of bndy1, and bndy3 has its start location set to its current location. Thus bndy3 must be allowed to advance first since when bndy4 returns to its start location, it removes bndy3.

These points ensure that all location combinations are explored and that they are explored only once. The first point is needed for proper operation of the Remove and StartLoc lists. That is, if the first point is adhered to, the algorithm explores all possibilities before repeating any. Figure 5.21 shows the same example as figure 5.20; however, in figure 5.21 the newly created boundaries 3 and 4 are not inserted in the correct location on the BBD boundary list. Consequently, the boundaries are not advanced in the same order as those of figure 5.20. This leads to the exploration of only some possibilities. Note that the number of possibilities explored would depend on the location selected on the boundary list for the newly created boundaries. The second point deals with created boundaries. It ensures that if boundary A moves and creates boundary B, C and D, and if boundary C has boundary A's start location, then boundary B and D move first. Otherwise, if C proceeds before B and D, then it will always remove B and D when it returns to its start location and thus prevents B and D from advancing.

BBD ListOfBoundaries
bndy1->bndy2->•

boundary 1 advances
before boundary 2.

Suppose that BBD ListOfBoundaries is
bndy2->bndy3->bndy4->•
instead of
bndy3->bndy4->bndy2->•

Thus, the newly created boundaries, 3 and 4, have not been inserted in the same location on the ListOfBoundaries as their originator, boundary 1.

The Remove and StartLoc list are the same as in fig. 5.20d.

The successive boundary movements advance bndy2 before bndy3 and bndy4, as follows:

Figure 5.21. Importance of proper boundary locations on BBD list of boundaries.

Example 5.20 repeated, however, here the newly created boundaries are not placed in the boundary list in the same location as the boundary they are replacing (their originator).

Remove and StartLoc lists as in fig. 5.20 p. Back to the initial boundary locations without generating the boundary locations of fig. 5.20 g,h,i,j,n,p.

Figure 5.21 (continued).

Notice that the algorithm results in exploring only some possibilities because the newly created boundaries were placed arbitrarily on the boundary list. Clearly, the number of possibilities generated becomes dependent on where the newly created boundaries are placed.

Lastly, the list of all the initial solutions is ordered in increasing number of boundaries and decreasing number of analog blocks, with the former criterion taking precedence. The occurrence of different numbers of boundaries is best explained by considering an example. Consider the *section* of figure 5.22. Different boundary locations in this case means a different number of boundaries. In accordance with the heuristic "as few boundaries as possible", the design of the system consisting of the *section* on the right would be considered first. If the design of this system succeeds, then the design of the system consisting of the *section* on the left is attempted if the user desires it. If the design of the system with the right *section* fails, the program returns the reason for failure, and based on this, the user decides whether or not to proceed with the design of the system with the left *section*. For instance, if failure was due to unacceptable area, then the user may decide to attempt the design of the system

with the left *section* if the user felt that the five additional analog components would reduce area sufficiently to offset the increase due to the additional boundary.

Figure 5.22. Within a *section*, different boundary locations may mean a different number of boundaries.

5.5.1.1 Complexity Analysis of Algorithm

Creating a new location for the boundaries usually requires very little time, since it translates into simply swapping the location of a boundary with its parent or child. Most of the time is spent verifying if a move across a branch point is valid. Here again, however, most of the required information may be acquired once during a forward and backward pass of the BBD. In fact, it may be acquired at the same time that the *sections* are defined. This information includes, for every node, the type(s) (FA and/or FD) of the forced ancestors and descendents that it has. The branch point conditions then simply look at the branch point parents or children for this information.

Therefore, searching for forced ancestor and descendent types for every new location of boundaries is not required, and is replaced by a one-time initial search. Other information simply requires checking if branch point parents or children are boundaries.

5.6 Review of Previous Sections and Enhancements

The routine FindInitialSolns, in figure 5.23, reviews the work discussed so far in this chapter. As well, lines 5 to 8 add new enhancements to further reduce the solution space.

This routine ties in everything mentioned so far. First, in line (1) a list BBDs specifying different initial boundary locations is generated (refer to section 5.3). Recall that each BBD has not more than two boundaries per any *section* path and that each maximizes the number of analog blocks. In line (3), a list of all allowed boundary locations for each BBD is generated (refer to *section* 5.5). This list is in increasing order of the number of boundaries and, for each number of boundaries, in decreasing order of the number of analog blocks. In line (9), an exhaustive search is performed on all BBDs. Several other lines, (5) to (8), introduce new ideas that have not yet been mentioned. These ideas serve to further reduce the number of BBDs considered. Specifically, they prune out different boundary locations before the exhaustive *style* search. Therefore, pruning out many possible solutions. (Refer back to the example of section 4.2 for an indication of the large size of the design space and of the resulting design space reduction achieved when pruning out boundary locations).

5.6.1 Check for Too Many Boundaries

Line (5) of fig. 5.23 checks if the particular BBD has a greater number of boundaries than any previous BBD solution. If so, it is reported to the user. As well, the user is reminded that all the following BBDs will all have the same or a greater number of boundaries. The user then decides to exit the program, to discard the current BBD but proceed with the others (especially if the user wants to see the other BBDs before discarding them), or to simply proceed.

FindInitialSoln()**BEGIN**

- 1 Create ListOfInitBdryLocs, a list of BBDs each specifying one of the initial boundary locations to be considered. [Section 5.3]
- 2 For each member of the ListOfInitBdryLocs Do
- 3 Create all the different combinations of the allowed locations of all boundaries. Add these to the ListOfDiffBdryLoc in order of increasing number of boundaries and decreasing number of analog blocks, with the former criterion taking precedence. [Section 5.5]
- 4 For each BBD member of the ListOfDiffBdryLoc Do
BEGIN
- 5 If initial solution with fewer boundaries was found
Inform user who, in turn, decides whether to proceed or not.
- 6 If (IsMoreDigital (BBD))
Inform user who, in turn, decides whether to proceed or not.
- 7 If boundary hardware specification, in terms of time sharing data conversions, is not achievable
Continue; /* reject this BBD */
- 8 If accuracy is not attainable
Continue; /* reject this BBD */
- 9 Do Exhaustive Search using branch-and-bound technique to find N best solutions for the given combination of boundary locations, where $N = \min\{M_1, M_2\}$ and M1 is specified by the user, M2 refers to the last solution such that its TS is no worse than x% less than the best solution TS for this search and previous searches (for other boundary locations); x specified by the user. [Section 5.4].
END;
- 10 If no solution found
- 11 Try a larger number of boundaries /*Not considered in thesis */
END;

Figure 5.23. The routine FindInitialSoln.

5.6.2 Check for Too Many Digital Blocks

In line (6) of fig. 5.23 an attempt is made to delete BBDs with too many digital blocks. Specifically, a check is made to determine if the current BBD is 'more-digital' than any other previous BBD that failed to produce a solution because of area. The current BBD is 'more-digital' than a failed BBD if every block implemented digitally in the failed BBD is also designated as digital in the current BBD. Since the BBDs are not the same, the current BBD must have at least one more digital block. Clearly, 'more-digital' does not simply mean a greater number of digital blocks. An example of a 'more-digital' and a not 'more-digital' BBD are shown in figure 5.24.

Figure 5.24. Illustration of 'more-digital' BBDs. If (a) is a failed BBD, then (b) is 'more-digital' and (c) is not, even though numerically it contains more digital blocks.

In figure 5.24, suppose that the reason for failure for figure 5.24a was excessive area. The system in figure 5.24b contains all the same digital nodes as A but in addition has moved the boundary so that B2 is now also digital. Clearly, system (b) will be larger than (a) (unless, possibly, B3 is bit serial). Therefore, the program would recommend to the designer that system (b), the 'more-digital' system, be rejected without further checking. In the same way, any other systems

derived from (a) by changing analog nodes to digital should be rejected.

However, the program cannot readily make the same recommendation for the not 'more-digital' system in figure 5.24c since even though B10, B11, and B12 are now implemented in digital, B3 and B4 have also changed and are implemented in analog. B3 and B4, when digitally implemented, may have taken up most of the chip area.

The algorithm for determining if a BBD is 'more-digital' than a failed BBD follows. This algorithm requires the use of compare lists and failure lists. The compare lists for figure 5.20 are shown below:

WhatToCompareLists

These lists are created when forward movement of boundaries across branch points occurs. A DtoA boundary is compared to all DtoA boundaries it creates; and an AtoD boundary is compared to all AtoD boundaries it creates. Note that bndy1 creates bndy3, 4 and 5 and that the alternate solutions either include bndy1, or bndy3 and 4, or bndy3 and 5. It is sufficient to compare bndy1 to bndy3 or 4, or to bndy3 or 5, respectively. For this reason, a DtoA boundary is compared only to all DtoA boundaries (and no AtoD boundaries) it creates. Similarly for AtoD boundaries.

If a DtoA boundary creates N DtoA boundaries, then the compare list containing that DtoA boundary is replicated N-1 times and a different created DtoA boundary is added to the end of each list. Similarly, for AtoD boundaries. For instance, in the following example

The associated compare lists are

WhatToCompareLists

The failure list simply is a list that holds the boundary locations of failed BBDs. If the failure list for figure 5.20 is

FailureList

then the boundary locations (g) and (k) to (p) inclusive would all be considered 'more-digital'. The IsMoreDigital routine is found in figure 5.25.

Function IsMoreDigital (BBD_ptr) : BOOLEAN;

/*For lines 8 and 12, a search for the boundary locations in the BBD is not needed if the block IDs are not selected arbitrarily. Here it is assumed that a block's parent IDs are always less than its ID, and that a block's children IDs are always greater than its ID. Then, the location of a boundary relative to another boundary simply requires checking parent or child IDs. */

BEGIN

1 IsMoreDigital = TRUE;

2 For each boundary of BBD Do

3 For each compare list containing the boundary Do

4 For each failure list Do

5 For each boundary on failure list Do

6 If (boundary on compare list) Then

7 If (boundary is DtoA)

8 If (child ID of BBD boundary < child ID of failure list boundary)

/*location of BBD boundary is above the location of the failure list boundary*/

BEGIN

9 IsMoreDigital = FALSE;

10 return;

END

11 Else /* boundary is AtoD */

12 If (child ID of BBD boundary > child ID of failure list boundary)

/*location of BBD boundary is below the location of the failure list boundary*/

BEGIN

13 IsMoreDigital = FALSE;

14 return;

END

END.

Figure 5.25. The function IsMoreDigital.

5.6.3 Check if Time-Sharing Data Conversions is Valid

Line (7) of figure 5.23 is a quick check to verify that the boundary conversion times are achievable. Recall that it is assumed that the A/D and D/A converters of AtoD and DtoA boundaries, respectively, are time-shared within a *section*; however, sample-and-hold circuitry and filters are not, rather, they are replicated. The information on the bandwidth of the signals sharing a boundary, the required output dynamic ranges (i.e. minimum number of bits), and the fastest conversion times for A/D and D/A converters is used to quickly reject unreasonable designs.

Using the minimum signal sampling frequency, i.e. twice the maximum signal frequency (Nyquist), for each signal, the maximum conversion period can be determined. The examples of figure 5.26 are used to illustrate this point. Let $f_{i,max}$ be the maximum signal frequency for the input i . Then $T_{i,s} = 1/2 * f_{i,max}$ is the maximum sampling period for signal i . Thus, signal i must be sampled and converted at least once every $T_{i,s}$ seconds. The examples of figure 5.26 show sets of signals that share the same boundary, and for each set, show the calculation of the slowest (maximum period) conversion time. The corresponding sequence of conversions that allows this maximum conversion period is also given. Example (2) and (3) illustrate that the same number of signals may result in different conversion sequences producing the maximum conversion period. Clearly, this is because the signals are of different bandwidths and so require a different set of sampling periods. This conversion period is passed on to the CKB. If the period is achievable, the program proceeds with the current BBD; otherwise, the current BBD is discarded.

The general case for 2, 3, and 4 signals is shown in figure 5.27. For these, the calculation of the maximum conversion period is shown.

Ex.1 N=4 signals

Signal	Sampling Period	
	required	actual
S1	10	10
S2	10	10
S3	40	20
S4	200	20

Sequence of conversions:

$$\text{Maximum conversion period} = \min \{10/3, 10/3, 40/6, 200/6\} = 10/3$$

Ex.2 N=3 signals

Signal	Sampling Period	
	required	actual
S1	10	8
S2	16	16
S3	17	16

Sequence of conversions:

$$\text{Maximum conversion period} = \min \{10/2, 16/4, 17/4\} = 4$$

Ex.3 N=3 signals

Signal	Sampling Period	
	required	actual
S1	10	10
S2	12	10
S3	14	10

Sequence of conversions:

$$\text{Maximum conversion period} = \min \{10/3, 12/3, 14/3\} = 10/3$$

Figure 5.26. Examples of signals that share the same boundary. For each sequence the maximum conversion period is calculated.

Ex.4 N=8 signals

Signal	Sampling Period	
	required	actual
S1	10	10
S2	12	10
S3	21	20
S4	22	20
S5	40	40
S6	60	40
S7	100	40
S8	400	40

Sequence of conversions:

$$\text{Maximum conversion period} = \min \{10/4, 21/8, 40/16\} = 2.5$$

Figure 5.26. (continued). Examples of signals that share the same boundary. For each sequence the maximum conversion period is calculated.

Assume $T_{s1} < T_{s2} < T_{s3} < T_{s4}$, where T_{si} is the minimum required sampling period for signal i .

Max Conversion time
CT

For two signals,

For three signals,

OR

$CT_{3,MAX} = \max\{CT_{31}, CT_{32}\}$

For four signals,

OR

OR

OR

$CT_{4,MAX} = \max\{CT_{41}, CT_{42}, CT_{43}, CT_{44}\}$

Figure 5.27. Different conversion sequences for 2, 3, and 4 signals.

The general derivation of the sequence representing the maximum conversion period for any number of signals is not covered here. Figure 5.27 derives it for up to 4 signals. However, the determination of this sequence is only required if the following two quick checks are inconclusive. Assume $T_{S1} < T_{S2} < \dots < T_{SN}$, then the first check decides if the sequence below is feasible for S1.

If this cannot be satisfied because of S1, then any other sequence would be rejected. This sequence corresponds to a conversion (sampling) period of $T_{S1}/2$.

The second check decides if the sequence below is feasible.

If this satisfies S1, then any other sequence is acceptable. This sequence corresponds to a conversion period of T_{S1}/N .

To sum up, the following steps are used to determine if boundary hardware conversion times are achievable:

```

If ( $\text{Min}_i\{T_{i,s}\}/2$  is not achievable) Then
  discard BBD
Else If ( $\text{Min}_i\{T_{i,s}\}/N$  is achievable) Then
  consider BBD
Else
  BEGIN
  Find the sequence with the greatest conversion time;
  If (this conversion time is achievable)
    consider BBD
  Else
    discard BBD;
  END

```

Note: In the previous examples, it was assumed that $T_{S1} = \text{Min}_i\{T_{i,s}\}$

5.6.4 Estimate if Accuracy is Attainable.

Lastly, if the BBD is still considered, line (8) of figure 5.23 makes an optimistic estimate of the attainable accuracy and compares it to the required accuracy. If this estimate fails to satisfy the requirement, then the BBD is rejected. As with line 7, line 8 tosses out obvious misfits. An optimistic estimate of accuracy is made by assuming that digital blocks contribute no error, and that the most accurate analog blocks are used. Here, SumOf for accuracy is used just as it is in the exhaustive search. However, no other attribute is considered and the calculation of SumOf is only used to discover if accuracy is unacceptable. Because of the BBD ordering in the ListOfPossSolns, the check on accuracy tends to eliminate BBDs in the early stages; whereas, the check for too many digital blocks eliminates BBDs later on.

5.6.5 When the Pruning of Boundary Locations Occurs

The methods start with "as much analog as possible". The test for accuracy is used to eliminate (prune out) BBDs in the early stages. This test should pass when the number of analog and digital blocks occur in 'reasonable' amounts. Then as the boundaries are moved to incorporate more digital blocks, the 'more-digital' test and the TS bound of previous solutions is used to eliminate alternatives with too many digital blocks.

In the worst-case, the desired design is bit-serial so that the TS bound has little effect.

5.7 Specifying the Order for Refinement Down the Hierarchy: Rating Blocks by Their "Uncertainty" Measure

Once a given choice of *style* for each block and an associated partitioning of the system level constraints amongst the blocks is accomplished, each block must be considered in turn for translation

of its constraints amongst its constituent subblocks. Refinement of blocks, as described in section 3.3, goes all the way down the hierarchy. The order in which the blocks are refined has a direct effect on the computing time required to find a solution. Design experts will always design components with high risks of failure before designing components with very little likelihood of failure. A high risk component, for instance, would be a noisy component, for which output noise is roughly estimated, in a system with a stringent noise specification. The designer would first attempt to work on the design of this block until the designer is satisfied with its ability to sufficiently limit its noise contribution. That is, only when the performance specifications at the lower levels of design satisfy the initial estimates would the designer proceed with the other blocks. This avoids wasting time designing most of the other blocks of a system which may never function.

This time-conscious procedure is adapted in the automatic design process. An uncertainty measure for each block, U_i for block i , is used to rank the blocks in order of urgency for processing. The following criterion are used in the calculation of U_i :

- (1) The reliability of the estimate of the attribute value j for block i , R_{ij} , contained in the CKB. Some examples for the area attribute are given in figure 5.28.
- (2) The percentage of change in the system attribute value which would yield an unacceptable score. This measure, m_j , represents the margin to unacceptability for attribute j . It is calculated as follows:

$$m_j = \frac{|V_{j,\text{unacceptable}} - V_j|}{V_j}$$

where

$V_{j,\text{unacceptable}}$ = the unacceptable system value for attribute j

$V_j =$ the estimated system value for attribute j

The larger m_j , the less probable it is that the attribute value becomes unacceptable¹. Alternatively, the larger $1/m_j$, the more likely it is that the attribute value becomes unacceptable.

- (3) The contribution of block i to the system attribute value j , C_{ij} . The calculation of C_{ij} is different for each attribute. For instance, for area $C_{i,area} = V_{i,area}/V_{area}$, and for SNR $C_{i,SNR} = V_{i,NF}/V_{NF}$ where noise factor (NF) is the ratio SNR_{in}/SNR_{out} .

$R_{i,area}$	Significance
0	Area is known exactly. Component i may be a predesigned library cell or a parameterized memory cell, where only the number of bits and words are parameterizable.
0.4	Area is fairly accurate. Component i may consist of unconnected standard cells.
0.8	Area is a rough estimate. Component i may be a synthesized analog block designed from one of several sized schematics.

Figure 5.28. Examples for $R_{i,area}$

¹An attribute value becomes unacceptable when refinement of components results in improved estimates that fail to meet specs, for which failure handling (section 3.3.2) cannot resolve the inconsistencies.

Combining the above criteria results in the following uncertainty measure for block i , considering all its N attributes:

$$U_i = \sum_{j=1}^N U_{ij} = \sum_{j=1}^N \frac{R_{ij} * C_{ij}}{m_j}$$

The higher U_i the more likely the final simulations will find the block unsatisfactory. Therefore, the block with the highest U_i is refined first unless the designer overrides the decision. That is, at every stage, the program states its selection and asks if the designer is in agreement; if the designer disagrees, the designer must specify an alternate choice. Once the order-of-attack for the individual blocks is specified, refinement down the hierarchy proceeds as described in chapter 3.

Note that the values for V_{ij} and R_{ij} are not static; rather, they change during the design process. At any given time, the constituent subblocks of the system blocks are refined to different levels of the hierarchy. When simulation or other verification is performed at lower levels, estimates on attribute values are improved. These improvements must result in a change of values for V_{ij} and R_{ij} . In turn, these changes require that the uncertainty measure is recalculated.

The use of the uncertainty measure is illustrated by considering the wattmeter example of section 4.2. Figure 5.29 demonstrates the calculation of U_i for each block for one solution. In this example, the analog multiplier, block B_1 , is by far the most critical block.

Block B _i	Area mm ²	R _{i,area}	PD mW	R _{i,pd}	Error out, % of FS	$\frac{\text{Err out}}{\text{Err in}}$	R _{i,error}	U _i
B ₁	.1	.4	30	.3	1	10	.6	.85 ⁺
B ₂	.05	.7	15	.3	1.02	1.02	.6	.24
B ₃	1.0	0	55	.2	1.04	1.02	.5	.11
B ₄	1.6	0	.05	0	1.04	1	0	0
Total: Score:	V _{area} = 2.75 S(V _{area}) = .5		V _{pd} = 100 S(V _{pd}) = .3		V _{err} = 1.04 S(V _{err}) = -2.5			

$$V_{\text{area,unacc}} = 5 \quad V_{\text{pd,unacc}} = 300 \quad V_{\text{err,unacc}} = 2$$

$$+ U = .4 * \frac{.1}{2.75} * \frac{2.75}{5-2.75} + .3 * \frac{30}{100} * \frac{100}{300-100} + .6 * \frac{10}{10.4} * \frac{1.04}{2-1.04} = 0.85$$

Figure 5.29. Example illustrating the use of the uncertainty measure.

5.8 Selecting other Possible Initial Solutions for Refinement

Once a solution has been designed to lower levels, the actual total score (TS) will most probably differ from the initial estimates of TS. In fact, the actual TS may actually be worse than the TS for other initial solutions. Whether or not this is true, the designer may choose to design other initial solutions.

If the designer is unhappy with solution A, then the designer should only consider alternate solutions that are significantly different. That is, the designer considers alternate solutions that involve interesting tradeoffs among several blocks. Otherwise, if two initial solutions A and B are very similar, then if the estimated TS of A is better than that of B, but the actual TS of A is worse than its estimated value, then it is very likely that the actual TS of B will be worse than its estimate. Therefore, the initial solution B should not be pursued. In fact, small changes should be left to the fine tuning or optimization programs which work on complete solutions.

In this work, it is up to the designer to decide on interesting alternative initial solutions. Future work may endeavor to automate the comparison of initial solutions to decide on good alternatives.

5.9 Functions That Include Domain Conversions

There exist certain functions, such as a multiplying D/A converter (MDAC), that perform a function as well as a conversion. In the case of a MDAC, if one input is analog, the other is digital, and the desired output is analog, then a MDAC would be considered since it takes up less area than a D/A converter and an analog multiplier.

To include such circuits, the program may simply be extended to perform some preprocessing before a new combination of boundary locations is considered. That is, before an exhaustive search for *style* allocation of blocks is performed. For each boundary, the preprocessing would consider functions performed before it as well as those performed after it. If any combination leads to a 'domain-converting' function, then this system would be considered as an alternative.

For time-shared boundaries, the new combination must work in all locations; otherwise, it is rejected. If kept, the new domain-converting function is shared as was the shared boundary it replaces. For instance, example 1 of figure 5.30 has one shared DtoA boundary and two analog multipliers and is replaced by a system with one shared sampled-analog MDAC² [GreR86]. In example 2, however, the shared DtoA boundaries cannot be combined to produce shared domain-converting functions; the alternative is worse and therefore, it is not considered.

²This advocates the sharing of switched-capacitor circuits. Though this makes sense theoretically, the author has not to date seen any circuits put this to practice. This sharing may cause unacceptable noise and crosstalk performance. Analysis of sharing sampled-analog circuits is beyond the scope of this thesis.

Figure 5.30. Use of 'domain-converting' functions.

EX 1: The MDAC can be time-shared in two boundaries.

EX.2: The MDAC cannot be shared since the '+' function is incompatible.

5.10 Chapter Summary

This chapter introduces a heuristic for determining good locations for signal conversion in a system. The minimum number of conversions required is determined and the heuristic specifies that only solutions with this given number of conversions be considered. An algorithm for obtaining all these possible solutions is described.

For each solution, an exhaustive search over all block styles is performed. This search uses the total score function in a branch-and-bound technique.

Other methods for further pruning the design space are given. These include the following:

- verifying that a BBD is 'more-digital' than a failed BBD;
- determining the maximum period for time-sharing converters and comparing it to library component specifications to see if it is acceptable; and
- determining an optimistic estimate on system accuracy, and verifying that it is acceptable.

As well, this chapter introduces an uncertainty measure which decides the order of refinement of system blocks based on their likelihood of causing the overall system to fail because their final specifications fail to satisfy initial estimates.

CHAPTER

6. An Example

An example is used to demonstrate the use of the methods described in this thesis. First, the example is used to show the usefulness of specifying constraints with the penalty credit function. Next, a simplified version of the same example serves to show how only a sensible number of boundaries is considered and to illustrate how constraints are propagated. Since the wattmeter example, introduced in chapter 4, was used in chapter 5 to illustrate the use of the TS function and the uncertainty measure, this will not be done here.

6.1. A Tactile Communication Aid and Its Specifications

The tactile hearing aid discussed here was designed by Tim Rahrer [RahT90]. This tactile aid is used by profoundly deaf persons. It includes vibrators attached to the deaf person's arm. The amplitudes of vibration of these vibrators deliver information about speech sounds. The tactile aid is used together with lip reading or some other aid and therefore, it operates in real time. The tactile aid is based on principal components analysis. Principal components analysis considers the energy at different frequencies for speech sounds. (For more information on principal component analysis refer to [RahT90]).

The basic functioning of this tactile aid is shown in figure 6.1. Speech is received by a microphone and then goes through a preamplifier. The output of the preamplifier is the input to the system being considered. The principal components are generated by passing the input through a spectrum analyzer: a twelve-channel filter bank. Each output from a band pass filter (BPF) goes through an energy detection circuit and then through a log function. Next, the resulting outputs are fed into four weighted-sum circuits generating four principal components. Each principal component corresponds to the

amplitude of vibration of one of the four vibrators attached to a user's arm.

Figure 6.1. Block diagram of the tactile aid and its external interfaces.

It consists of vibrators attached to the deaf persons arm. The amplitudes of vibration of these vibrators deliver information about the speech sounds.

The requirements of this system are specified using penalty-credit functions (PCF), some of which are shown in figure 6.2. The first PCF specifies area. Similar to the wattmeter example, different sections of the curve represent the different die areas that are available. The drop in score at 14 mm^2 is much larger than the drop at 9 or 12 mm^2 ; this is because a circuit larger than 14 mm^2 requires two chips.

The PCF for delay shows that if the delay between the input and output is greater than 30 ms, then the aid is useless. 30 ms is the speech frame length over which most speech sounds remain stable. However, some sounds in speech often change (40dB) in less than 3 ms. The PCF however does not show any marked increase in score beyond 10 ms, since this approaches the skin's limit in time resolution. Note that the aid works together with lip reading or some other aid and must allow the user to function normally in a conversation, therefore it must operate in real time.

The PCF specifying the time between battery recharges shows that any increase in the time between battery recharges beyond 8 hours is not worth striving for at the expense of any other criteria. Also, it is unacceptable to require recharges more frequently than every two hours.

Finally, the PCF for accuracy has not only a lower bound but also a higher bound on amplitude resolution. The ability of the skin to detect a change in amplitude beyond 48 dB is the limiting factor.

The input specifications are also required. The input dynamic range is 40 dB with $V_{\text{peak}} = \pm 5$ Volts. The speech information is contained in the bandwidth from 0-4kHz; however, the input is not bandlimited. The general shape of the autocorrelation function and PSD estimate for speech are shown in figure 6.3a [RabL78]. The

long-time PSD for continuous speech is shown in figure 6.3b. The piecewise-constant curve for men is used as the input for this example.

⁺ Power assuming a battery capacity of 800 mAh and given that the vibrators dissipate 150 mW.

⁺ Equivalent bit accuracy (total error < 1/2 lsb)

Figure 6.2. Some PCFs for the Tactile aid.

Autocorrelation function

Power spectral density estimate for speech

(a)

Long-time power density spectrum for continuous speech

(b)

Figure 6.3. Autocorrelation and power spectral density for speech
[RabL78]

Assume that the input signal power from 0 to 4KHz is 0.7mW. Then the relative power of 1, 0dB in figure 6.3c, is equivalent to 0.5mW. Negligible noise input is assumed within the band of interest. Also, note that though the power spectral density of speech changes with time, the designer has chosen to use a simplified model of speech (stationary) since reasonable estimates have been shown to follow from such a simplification [RabL78].

6.2. A Simplified Example

To illustrate the propagation of a few attributes and of the input BW, the simplified system shown in figure 6.4 is used. Figure 6.5a to 6.5f shows all the locations of the boundaries that the methods consider. This system has only one section and it has no special branch points; thus, only one initial boundary location exists, figure 6.5a. Note that when more than two AtoD (DtoA) boundaries are shown, they are assumed to be one time-shared unit.

The locations of the boundaries selected for this example is shown in figure 6.6. A sample implementation of this initial locations for the boundaries is shown in figure 6.7. Here the CKB realizes that an anti-aliasing (AA) filter is required before proceeding to the sampled analog filter bank. The anti-aliasing filter selected consists of a simple analog low-pass filter (LPF) followed by two more precise switched-capacitor (SC) filters. If the filter bank was implemented in digital, then sharing of the band-pass filters (BPF) would have been considered. Note that the multiplications and the shared DtoA boundary are replaced by a shared sampled-analog MDAC.

Not only are the decisions of what to place at the interfaces between different domains flexible, but each block may be implemented in several styles. For instance, the BPF are specified as 6th order IIR filters by the designer. Figure 6.8 (repeated from figure 4.7) is a look into part of the component knowledge base (CKB) where the low-pass filter function is defined. The designer selects the appropriate approximation (Butterworth, Chebyshev, elliptic, ...) for

the filter function. Then for a switched-capacitor implementation, the CKB returns styles that correspond to the structures (ladder, cascaded biquad, ...) that satisfy specifications. For each of these, different circuit 'extras' that reduce error may or may not be included. As well, for each, the sampling frequency may be varied. All these choices constitute the different styles available. If, however, a digital filter bank was used, then besides different structures and different levels of parallelism, the number of bits can be varied.

Next, signal delay (latency) and SNR are considered. In figure 6.7, the signal delay through the system is calculated. The signal delay through the filters is approximated by $1/BW$ ($1/F_c$) [StrF77]. The delay through the SC nonfiltering functions is one clock period.

Figure 6.4. The generic BBD for the simplified system.

Figure 6.5. Boundary movement.

(a) initial location for boundaries;

(b) to (f) movement of boundaries.

Figure 6.6. Selected boundary location for example.

As for SNR, consider the output of the analog anti-aliasing filter, a first order LPF. The CKB would contain all the information found in figure 6.9a. In this example, the only sources of error considered are thermal and flicker ($1/f$) noise. The thermal noise from the resistors depends on the resistor values and the input referred noise depends on the operational amplifier chosen. The values of the components are partly determined by the user specifications on cutoff frequency $f_0=500\text{kHz}$ and gain $= 1$. The different styles that would be considered include different styles of operational amplifiers, and different values for the resistances and capacitance. The different styles of operational amplifier make tradeoffs between global constraints on such things as the amounts of the different static and dynamic errors. Some operational amplifier specifications, such as, slew rate and bandwidth must simply satisfy specific block specifications. The different resistor and capacitor values result in tradeoffs between area and noise.

Figure 6.7. Possible solution to simplified system.

Figure 6.8. Part of the component knowledge base.
Repeated from figure 4.7 for convenience.

$$H_1(s) = \frac{V_o(s)}{V_{in}(s)} = \frac{V_o(s)}{V_{Ri}(s)} = \frac{-R_0/R_i}{1 + C_0 R_0 s} \Rightarrow |H_1(\Omega)|^2 = \frac{R_0^2/R_i^2}{1 + C_0^2 R_0^2 \Omega^2}$$

$$H_2(s) = \frac{V_o(s)}{V_{Ro}(s)} = \frac{-1}{1 + C_0 R_0 s} \Rightarrow |H_2(\Omega)|^2 = \frac{1}{1 + C_0^2 R_0^2 \Omega^2}$$

$$H_3(s) = \frac{V_o(s)}{V_n(s)} = 1 + \frac{R_0/R_i}{1 + C_0 R_0 s} \Rightarrow |H_3(\Omega)|^2 = 1 + \frac{2R_0/R_i + R_0^2/R_i^2}{1 + C_0^2 R_0^2 \Omega^2}$$

at the output

$$\text{PSD}_{\text{Signal}}(\Omega) = \text{PSD}_s(\Omega) |H_1(\Omega)|^2 \quad (\text{Volts}^2/\text{Hz})$$

$$\begin{aligned} \text{PSD}_{\text{Noise}}(\Omega) &= \text{PSD}_{V_{Ri}}(\Omega) |H_1(\Omega)|^2 + \text{PSD}_{V_{Ro}}(\Omega) |H_2(\Omega)|^2 + \text{PSD}_{V_n}(\Omega) |H_3(\Omega)|^2 \\ &= 4kTR_i |H_1(\Omega)|^2 + 4kTR_0 |H_2(\Omega)|^2 + \text{PSD}_{V_n}(\Omega) |H_3(\Omega)|^2 \end{aligned}$$

where $4kT = 1.66 \times 10^{-20} \text{ VC}$

(a)

$$\Omega_0 = 1/C_0 R_0 = 2\pi(500\text{KHz}), \quad R_0/R_i = 1$$

$$\text{PSD}_{V_n}(f) = (10\text{nV})^2/\text{Hz} (1 + 1000/f), \text{ for } f > 60\text{Hz}$$

(b)

Figure 6.9. First order low-pass filter.

(a) Specification in CKB for generic LPF_i.

(b) Values require for specific implementation.

The program receives $\text{PSD}_{\text{signal}}$ and $\text{PSD}_{\text{noise}}$ for the output from the CKB, and performs an integration from 0 to 4kHz to measure the

signal and noise power. Note that the integration need not be very precise since it is used for comparison only. Though, it should be optimistic so as not to calculate unacceptable SNR when SNR is acceptable.

For $R_o = R_i = 100\text{k}\Omega$, $C_o = 3.183\text{pF}$. Then the noise power and signal power calculated up to 4kHz are given by:

$$\text{Noise Power} = \frac{1}{2\pi} \int_0^{2\pi(4\text{k})} \text{PSD}_{\text{Noise}}(\Omega) d\Omega \cong 2.49 \times 10^{-10} \text{ watts}$$

$$\begin{aligned} \text{Signal Power} &= \frac{1}{2\pi} \int_0^{2\pi(4\text{k})} \text{PSD}_{\text{Signal}}(\Omega) d\Omega \\ &= \int_0^{4\text{k}} \text{PSD}_S(f) |H_1(f)|^2 df \\ &= \sum_{i=1}^9 C_i \int_{f_{i1}}^{f_{i2}} \frac{(500\text{k})^2}{(500\text{k})^2 + f^2} df \\ &= 500\text{k} \sum_i C_i \tan^{-1} \left[\frac{f}{500\text{k}} \right] \bigg|_{f_{i1}}^{f_{i2}} \\ &= 0.7 \text{ mW} \end{aligned}$$

where C_i is the value for the piecewise constant curve for men in figure 6.3b of $\text{PSD}_{\text{Signal}}$ in the frequency range from f_{i1} to f_{i2} , and f_{11} to f_{92} covers the frequency range from 0 to 4kHz. Therefore, SNR is calculated to be:

$$10 \log(0.7\text{mW} / 2.49 \times 10^{-10}\text{W}) \cong 64.5 \text{ dB}.$$

The signal power was calculated for illustrative purposes. The CKB would not actually calculate it because $\text{BW}_{\text{IS}} = (0\text{-}4\text{kHz})$ and $4\text{kHz} \ll 500\text{kHz}$. Realizing this, the CKB would 'know' that the signal power is only affected by the passband gain, which in this case is unity.

CHAPTER

7. Conclusions and Future Work

7.1 Conclusions

7.1.1 Summary of Work

Though the digital synthesis problem is well defined, the analog synthesis problem is not. Little previous thought has been given to the problems of mixed-mode behavioral synthesis. The system described in this thesis is a first step in a mixed-signal synthesis system, with an emphasis on mixed. To the author's knowledge, this is the first attempt in synthesis to deal with mixed-signals at the behavior level. Because mixed-signal synthesis is a new area, this thesis briefly describes the overall design process from behavioral level to lower levels. As well, it describes many of the issues involved, laying down a path for future work to follow.

Some of the problems in mixed-signal synthesis identified in this thesis were the following:

- Selection of boundary locations between the analog and digital sections.
- Noise tracking.
- Design space reduction.
- Description of the complete synthesis problem, from behavior to layout, and of how mixed-signal behavioral synthesis fits in.

Problems for which partial or complete solutions were introduced in this thesis are discussed next.

Boundary locations

A heuristic for determining good locations for signal conversion in a system is given. The minimum number of conversions required is

determined and the heuristic specifies that only solutions with this given number of conversions be considered. The grounds for this heuristic is that for most applications it is seldom advantageous to add extra conversions above this minimum number. For the rare cases where a greater number of conversions is required, the methods must be extended.

An algorithm for obtaining all the possible solutions with this minimum number of conversions is given. The algorithm does not consider sharing of boundaries between sections.

Pruning the design space by rejecting certain boundary locations

Any pruning of this type greatly reduces the design space because every BBD pruned out eliminates a complete exhaustive search. Some pruning methods introduced are listed below:

'More-digital':

The order in which the different locations of boundaries is considered proceeds from more analog to more digital systems. It is assumed that when an analog circuit meets the specifications, it is usually better (less area). Thus, proceeding in this manner should reduce the design space explored by pruning out the 'more-digital' circuits. However, when delay is not very important, analog designs are not always better. At the extreme, digital bit serial designs may be used. For such cases, the synthesis package will not prune 'more-digital' designs and so, will explore all possibilities (worst case).

Maximum period for time-sharing converters:

The formula for the maximum period for conversion in time-shared AtoD or DtoA converters was calculated for up to four signals. With knowledge of the bandwidths of the signals sharing the converter, and of the fastest converters in the library, bad time-shared boundary locations are quickly rejected.

System accuracy:

Because some possible solutions include many analog blocks, a method for quickly rejecting possible solutions with too many analog blocks was introduced. The accuracy was calculated considering perfect digital blocks and the best case analog blocks in the library, while ignoring all other attributes. If the optimistic accuracy value that results is not acceptable, the corresponding BBD is rejected.

Pruning the design space by reducing the number of styles

The use of branch-and-bound using the total score function in the exhaustive search limits the number of different styles considered.

Noise calculations --the 'interesting' signal bandwidth, BW_{IS}

The 'interesting' signal bandwidth was introduced for calculating noise power in partial design solutions. This was required so as not to reject partial solutions by considering noise which is later filtered out. A method for calculating BW_{IS} was given.

Use of the Blackboard model and Brewer and Gajski's expert-system design paradigm

The ideas of *translation* and *style* selection were adopted from Brewer and Gajski's expert-system design paradigm. In this thesis, the steps of *style* and *translation* at the behavioral level were performed using the exhaustive search algorithm with the total score functions. The total score functions were based on penalty-credit functions.

These mechanisms of *translation* and *style* selection allow for the transfer of information in the Blackboard framework from the behavioral level to the next lower level. The Blackboard model also describes control modules that perform opportunistic reasoning. That is, the control modules can decide what to do next. In this

System accuracy:

Because some possible solutions include many analog blocks, a method for quickly rejecting possible solutions with too many analog blocks was introduced. The accuracy was calculated considering perfect digital blocks and the best case analog blocks in the library, while ignoring all other attributes. If the optimistic accuracy value that results is not acceptable, the corresponding BBD is rejected.

Pruning the design space by reducing the number of styles

The use of branch-and-bound using the total score function in the exhaustive search limits the number of different styles considered.

Noise calculations --the 'interesting' signal bandwidth, BW_{IS}

The 'interesting' signal bandwidth was introduced for calculating noise power in partial design solutions. This was required so as not to reject partial solutions by considering noise which is later filtered out. A method for calculating BW_{IS} was given.

Use of the Blackboard model and Brewer and Gajski's expert-system design paradigm

The ideas of *translation* and *style* selection were adopted from Brewer and Gajski's expert-system design paradigm. In this thesis, the steps of *style* and *translation* at the behavioral level were performed using the exhaustive search algorithm with the total score functions. The total score functions were based on penalty-credit functions.

These mechanisms of *translation* and *style* selection allow for the transfer of information in the Blackboard framework from the behavioral level to the next lower level. The Blackboard model also describes control modules that perform opportunistic reasoning. That is, the control modules can decide what to do next. In this

thesis, an uncertainty measure was introduced for deciding the order-of-attack for the next lower level.

Uncertainty measure

A novel method for determining the order of refinement of blocks down the hierarchy was introduced. A good order-of-attack saves a lot of computing time by requiring, on average, only a relatively few blocks to be designed before a bad solution is rejected. This method creates an order using the outcome of an uncertainty measure which includes different factors considered by expert designers. Although the way of measuring the different factors follows naturally, the way of combining them is not obvious. One method was described and used in this thesis.

7.1.2 Benefits

Long before a complete automated system is available, the methods discussed in this thesis could be used to decrease the large amount of design time spent on choosing good initial system implementations. The methods for making crude estimates on system attribute values (SumOf) can be used immediately.

Since the design space is always explored, the methods allow for innovative design choices. That is, whereas designers may do a lot of their work based on previous work, the automatic package is not biased and will always explore all reasonable avenues. A system designer, restricted in time, cannot always keep up with all the new styles created by silicon designers. It may take several designs before a system designer learns about a new style and its interesting tradeoffs. In contrast, the automated design system considers new styles immediately (as soon as they are added to the CKB).

As well, the automated system improves the interface between the silicon and system designers by automating it and therefore, standardizing it.

In addition, once the CKB is in place, the synthesis package may be used to determine which new styles or functions should be designed or programmed. That is, before investing a vast amount of time to create a silicon compiler or to custom design a new style for a particular function, the synthesis tool can help decide if the effort is warranted. The performance estimates of what experts think can be achieved for such designs can be input into the CKB. Then, statistics can be obtained on how often the particular circuits are selected and with what specifications. Based on these statistics a decision on whether or not to design or program may be made.

7.2 Future Work

The mixed-signal synthesis system described in this thesis is still in its initial stages. Several issues that make up some of the future work required for this project are briefly described next.

BBDs exhibiting reconvergent fanout:

Figure 7.1. Example of reconvergent fanout.

In figure 7.1, the signals S_1 and S_2 are a function of the signal S . The correlation between the signals S_1 and S_2 is a function of the operations performed and the delay incurred in the two paths from B_1 to B_5 . The degree of correlation affects the noise and signal powers and thus, the SNR. Sometimes reconvergent fanout is used to

reduce noise. Whatever the case, a method for estimating the correlation between the signals requires consideration.

BBDs exhibiting feedback:

The stability of a system with feedback must be ensured. Calculation of phase margin may be used. This requires the propagation of phase information through the system. The phase response of a system would be stored in the CKB. Given the input phase, the system phase response, the output phase for a linear system is readily calculated ($\arg[Y(e^{j\omega})] = \arg[X(e^{j\omega})] + \arg[H(e^{j\omega})]$). Calculations for estimating the output phase for different nonlinear systems with special common input types may be stored in the CKB.

As well, the correlation between the input and the fed back signal must be determined. Or, if a transfer function can be obtained for the complete feedback loop, then it may be used.

Time-sharing components

At the behavior level, not many, if any, blocks perform the same function; however, for those that do, time-sharing of one block may be considered. Sharing should be considered for digital blocks. Although in theory SC circuits can be shared, the author has yet to discover a system where this has been put into practice. Work has already been done for time-sharing digital components, for instance, refer to the HAL program developed by [PauP88].

At lower levels of the design hierarchy, the number of components, making up the constituent subblocks of the higher level behavior blocks, are much more numerous. Many more components will perform the same simpler functions. These components may be time-shared. For example, if two high level functions are described using RTL descriptions, the logic and registers may be shared between the two functions. A method to estimate and reflect this time-sharing in the V_j 's must be worked out.

Time-sharing boundaries between sections

This is similar to time-sharing components. However, as mentioned in section 5.5.1, the methods used must be extended to include the effects of routing long analog lines.

Distributed costs

Some costs, such as those incurred from including clocks, power supplies, offset elimination circuits, control logic and interconnect, are shared between some or all of the blocks. Therefore, these costs should be distributed among the respective blocks.

A simple solution would be to add the cost of a shared component to the first block requiring it, and then to add no other costs to the other blocks that share it. Though this method is simple, it is not very efficient. It causes the branch-and-bound technique to be much less effective by making partial designs appear much worse than they actually are.

Rather, incremental costs should be estimated. For instance, consider the control unit. The style of the control logic must be selected. The control may be centralized, distributed or a compromise between the two methodologies. As well, it may consist of one or several levels of decoding. Once this selection is made, then when a style for a block is selected, a method for estimating its contribution (extra memory and gates) to the control logic should be used.

In particular, interconnect area is difficult to deal with. However, since interconnect contributes significantly to the total chip area, it must be considered. The difficulty in dealing with interconnect area comes from its dependency on many factors, such as: selected style for control, placement of blocks, number of analog and digital sections, etc.

Considering a greater number of boundaries

The consideration of more than the minimum number of boundaries, especially when no solution is found, is another area of future study.

Since the benefit of the extra boundary must outweigh the cost of including it, it is unlikely that more than a few locations for the extra boundary are worthwhile attempting. A method for determining these locations would extend the existing algorithm. As a first step, blocks that are not of a forced domain but that perform significantly better in one domain should be so labelled. Then, the extra boundary can work with these blocks so that they are implemented in their favored domain. As well, some local maxima searches, where only the change in TS due to a few changed components is calculated, may be sufficient.

Additional pruning of design alternatives

The number of design alternatives may be reduced by further limiting either the number of boundary locations considered or the number of styles selected for each block, or both. For instance, if the range of values for a block specification can be further limited, then fewer styles are usually considered. The work done by [KumY90] estimates the required accuracy (number of bits) for each block in a digital system. This work may be extended to mixed systems. As well, any previously acquired application specific knowledge for different applications can be stored. Then when a new design is undertaken, the relevant stored information can be used either to select a specific block style or reduce the number of block styles considered for one or more blocks. Or it can be used to reduce the number of boundary locations considered.

Using other search techniques besides branch-and-bound

The branch-and-bound technique compares the TS of partial solutions together. Therefore, a partial solution with only a few blocks, say 2, is optimistically compared to another with many more

blocks, say 10. This is a very optimistic algorithm, since partial solutions with little prospect for success are still considered. Consequently, the bounding will tend to prune partial solutions only at the lower levels, where many blocks have already been selected. Better methods of pruning should be investigated. For example, some look-ahead techniques can be incorporated. The algorithm may consider looking for 'jumps' and comparing slopes in the penalty-credit functions. Or, some other algorithm, such as simulated annealing, could be considered.

Note that the alternate methods may only achieve local maximums, whereas, branch-and-bound obtains the global maximum. However, they may run significantly faster.

Other

This list of future work is certainly not complete. It does not include a lot of work that is not specific to mixed-signal designs, such as, the development of the CKB and of the control module for the Blackboard framework.

As mentioned above, much of the future issues are not new problems that have come about because of mixed-signal systems. Rather, they still exist in fully digital system synthesis. For example, the estimation of interconnect area.

Appendix A

A.1 Simplified Switched-Capacitor Circuit Theory

Switched-capacitor (SC) circuits are sampled analog circuits. Basically, a SC circuit replaces each resistor in a circuit by a combination of a capacitor and a few switches as shown in the figure below.

Figure A.1. Switched-Capacitor Equivalent Resistance

The SC circuit mimics the current limiting effect of a resistor. When ϕ_2 is high, C_R is discharged. When ϕ_1 is high, C_R is charged to the voltage $(V_{out} - V_{in})$. This causes a charge $q = C_R (V_{out} - V_{in})$ to flow through the branch. This cycle is repeated every T seconds so that the average current through the branch is given by $i_{ave} = q/T = C_R/T (V_{out} - V_{in})$. Comparing this to the current through the resistor $i = 1/R (V_{out} - V_{in})$ leads to the relation $R = T/C_R$. Actually, this relation holds if the periodic opening and closing of the switches by the clocks occurs at a rate much faster than that of the variations of the terminal voltages V_{in} and V_{out} . For such cases, the time sampling effects can be ignored in a first order analysis [MavJ83]. Refer to [GreR86] for a detailed description of the circuit functioning.

Appendix B

B.1 Statistics of a Signal

The determination of signal power is considered for time varying signals, whether they are stochastic processes or deterministic signals, as well as for static signals, whether they are random variables or constants.

i) Stochastic Process $x(t)$

For a given stochastic variable $x(t)$, the mean, autocorrelation and power are defined as:

$$\text{mean of } x = \eta_x(t) = E\{x(t)\} = \int_{-\infty}^{\infty} x f(x,t) dx$$

$$\text{autocorrelation of } x = R_{xx}(t_1, t_2) = E\{x(t_1) x^*(t_2)\}$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x_1 x_2^* f(x_1, x_2; t_1, t_2) dx_1 dx_2$$

$$\text{power of } x = R(t_1, t_2) \Big|_{t_1=t_2=t}$$

where $f(x;t)$ = first order density function of $x(t)$

$f(x_1, x_2; t_1, t_2)$ = second order density function of $x(t)$

In particular, wide sense stationary (WSS) stochastic processes will be considered. WSS processes have a constant mean and an autocorrelation that depends only on the time difference $\tau = t_1 - t_2$ between samples of $x(t)$. That is,

$$\text{mean of } x = \eta_x(t) = \eta_x$$

autocorrelation of $x = R(\tau) = E \{x(t+\tau) x^*(t)\}$

power of $x = R(0)$

As well, for WSS processes, the PSD is simply the Fourier transform of the autocorrelation function (this is not so for nonstationary signals). Thus, for WSS processes:

PSD of $x = \text{PSD}_x(\Omega) = \text{Fourier Transform } [R(\tau)]$

$$= \int_{-\infty}^{\infty} R(\tau) e^{-j\Omega\tau} d\tau, \quad \Omega = 2\pi f$$

and

$R(\tau) = \text{Inverse Fourier Transform } [\text{PSD}_x(\Omega)]$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \text{PSD}_x(\Omega) e^{j\Omega\tau} d\Omega$$

Since $R(0) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \text{PSD}_x(\Omega) d\Omega = \int_{-\infty}^{\infty} \text{PSD}_x(f) df$, $R(0)$ is equal to the total signal power.

The version of the preceding results for discrete-time processes is self evident. If $x[n]$ is a discrete time process, then

$$\eta[n] = E \{x[n]\} = \int_{-\infty}^{\infty} x f(x, n) dx$$

$$R[n_1, n_2] = E \{x[n_1] x^*[n_2]\}$$

$$\text{PSD}_x(\Omega) = \sum_{m=-\infty}^{\infty} R[m] e^{-jm\Omega T}, \text{ periodic of period } \frac{2\pi}{T}$$

and

$$R[m] = \frac{1}{2\pi/T} \int_{-\pi/T}^{\pi/T} \text{PSD}(\Omega) e^{jm\Omega T} d\Omega$$

ii) Deterministic Signal $x(t)$

For a deterministic signal, which can be considered as an extreme example of a stochastic process,

$$\eta_x(t) = x(t)$$

$$R(t_1, t_2) = x(t_1) x(t_2)$$

$$\text{power} = x^2(t)$$

and if the process is WSS, then the deterministic signal is a constant, $x(t) = x$. For deterministic static signals (constants),

$$R(\tau) = x(t) x(t+\tau) = x^2$$

$$R(0) = x^2$$

$$\text{PSD}(\Omega) = \text{FT}[x^2] = 2\pi x^2 \delta(\Omega)$$

Clearly, the power is DC and no AC component exists.

iii) Random Variable x

For a random variable x , the mean and variance of x are defined as follows:

$$\eta_x = E\{x\} = \int_{-\infty}^{\infty} x f(x) dx$$

$$\text{Var}(x) = \sigma_x^2 = E \{(x - \eta_x)^2\} = \int_{-\infty}^{\infty} (x - \eta_x)^2 f(x) dx$$

where $f(x)$ = density function of x

The power

$$E \{x^2\} = \sigma_x^2 + \eta_x^2$$

consists solely of DC power as was the case for static deterministic signals. As an example, consider the random variable voltage offset error. Assume an error with uniform distribution between +10 and -10 mV. Then the DC power is given by:

$$\int_{-10}^{+10} x^2 \frac{1}{20} dx = 33.3 \times 10^{-6} \text{ Volts}^2 = 33.3 \mu\text{Watts}$$

B.2 Propagation of PSD Through a Linear System

Linear systems denoted by

$$y(t) = L[x(t)]$$

are defined by the principle of superposition, which combines both the additivity and scaling properties:

$$L[a_1 x_1(t) + a_2 x_2(t)] = a_1 L[x_1(t)] + a_2 L[x_2(t)]$$

where a_i is a constant which may be a random variable over the ensemble and x_i is a stochastic process. The output of a linear system is a convolution of the input $x(t)$ with the impulse response $h(t) = L[\delta(t)]$ of the system, where $\delta(t)$ is the Dirac delta function.

$$y(t) = L[x(t)] = x(t) * h(t) = \int_{-\infty}^{\infty} x(t-\alpha) h(\alpha) d\alpha$$

If the input of a linear system is WSS, then its output is also WSS. Then for a WSS input, the mean, autocorrelation and spectrum are obtained using the following:

$$\eta_y = L[\eta_x] = \eta_x * h(t) = \eta_x H(0)$$

where: $H(\Omega)$ = system transfer function
= Fourier Transform $[h(t)]$

$$R_{yy}(\tau) = L_1[L_2[R_{xx}(t_1-t_2)]] \Big|_{\tau=(t_1-t_2)}$$

$$= R_{xx}(\tau) * \rho(\tau)$$

where L_1 acts on t_1 , L_2 acts on t_2 , and $\rho(\tau) = h(t) * h^*(-t)$

$$PSD(\Omega) = PSD_x(\Omega) |H(\Omega)|^2$$

The PSD is not only useful as a measure of the localization of power on the frequency axis, but it also simplifies the analysis of linear systems by reducing convolution to multiplication.

Similar results are obtained for discrete systems where the delta response $h[n] = L\{\delta[n]\}$ specifies the system. The system function, $H(z)$, is the z-transform of the delta response:

$$H(z) = \sum_{n=-\infty}^{\infty} h[n] z^{-n}$$

The system output is a discrete convolution of $x[n]$ with $h[n]$.

$$y[n] = \sum_{k=-\infty}^{\infty} x[n-k] h[k] = x[n] * h[n]$$

From this it follows that the mean, autocorrelation and PSD are

$$\eta_y = \eta_x H(1)$$

$$R_{yy}[m] = R_{xx}[m] * \rho[m]$$

$$\text{where } \rho[m] = h[m] * h^*[-m]$$

$$PSD_y(\Omega) = PSD_x(\Omega) |H(e^{j\Omega T})|^2$$

Next, a system with an input consisting of both signal and noise information is considered: $x(t) = s(t) + n(t)$ (see figure B.1). The output, $y(t)$, of the system is WSS only if both $s(t)$ and $n(t)$ are WSS and are jointly WSS (the cross-correlation between $s(t)$ and $n(t)$ depends only on $\tau = t_2 - t_1$). For this case the autocorrelation is given by:

$$R_{yy}(\tau) = R_{s's'}(\tau) + R_{n'n'}(\tau) + R_{s'n'}(\tau) + R_{n's'}(\tau) \quad (1)$$

$$\text{where } R_{n's'}(\tau) = R_{s'n'}^*(-\tau)$$

Figure B.1 Linear System

For the calculation of SNR, the signal and noise are considered separately. $R_{s's'}(\tau)$ is used to calculate the signal power and all other terms of equation (1) are used to calculate the noise power. The

cross-correlation between the signal and the noise contributes to error.

$$\text{PSD}_{\text{Signal}}(\Omega) = \text{FT} \{R_{s's'}(\tau)\}$$

$$\begin{aligned} \text{PSD}_{\text{Noise}}(\Omega) &= \text{FT} \{R_{n'n'}(\tau) + R_{s'n'}(\tau) + R_{n's'}(\tau)\} \\ &= \text{FT} \{R_{n'n'}(\tau)\} + \text{FT} \{R_{s'n'}(\tau)\} + \text{FT} \{R_{n's'}(\tau)\} \end{aligned}$$

Cross-correlation is a measure of the degree of correlation between the signal and the noise. In many cases, the signals are uncorrelated, $R_{xy} = \eta_x \eta_y$, or orthogonal, $R_{xy} = 0$ (uncorrelated and $\eta_x=0$ or $\eta_y=0$); or they are perfectly correlated and in phase, $R_{xy} = \eta_x \eta_y = 1$, or out of phase $R_{xy} = \eta_x \eta_y = -1$. Physically, uncorrelated signals are on average 90° out of phase, and perfectly correlated signals are 0° in or 180° out of phase. At this stage in the synthesis system's development, the designer must supply the function $R_{xy}(\tau)$ for signals that are partly correlated.

For uncorrelated signals, as is the usual relation between noise and the derived signal, the SNR is calculated as follows:

$$R_{yy}(\tau) = R_{s's'}(\tau) + R_{n'n'}(\tau) + 2 \eta_x \eta_y$$

$$\begin{aligned} \text{PSD}_{\text{Signal}}(\Omega) &= \text{FT} \{R_{s's'}(\tau)\} \\ &= \text{PSD}_{ss}(\Omega) |H(\Omega)|^2 \end{aligned}$$

$$\begin{aligned} \text{PSD}_{\text{Noise}}(\Omega) &= \text{FT} \{R_{n'n'}(\tau)\} + \text{FT} \{2 \eta_x \eta_y\} \\ &= \text{PSD}_{nn}(\Omega) |H(\Omega)|^2 + 2 \eta_x \eta_y \delta(\Omega) \end{aligned}$$

$$\text{SNR} = \frac{\frac{1}{\pi} \int_0^{\Omega_1} \text{PSD}_{\text{Signal}}(\Omega) d\Omega}{\frac{1}{\pi} \int_0^{\Omega_1} \text{PSD}_{\text{Noise}}(\Omega) d\Omega}$$

For orthogonal signals,

$$\text{PSD}_{\text{Signal}}(\Omega) = \text{PSD}_{ss}(\Omega) |H(\Omega)|^2$$

$$\text{PSD}_{\text{Noise}}(\Omega) = \text{PSD}_{nn}(\Omega) |H(\Omega)|^2$$

The sampling function

The sampling function is of particular interest in mixed-signal systems and is considered next. Let $x[n]$ equal the time-samples of a continuous time process $x_c(t)$ sampled at equally spaced intervals T , so that

$$x[n] = x_c(nT).$$

Then the statistics of $x[n]$ can be determined from the statistics of $x_c(t)$, in particular:

$$\eta[n] = \eta_c(nT)$$

$$R[n_1, n_2] = R_c(n_1T, n_2T)$$

Now if $x_c(t)$ is WSS, then so is $x[n]$. Then for $m = n_1 - n_2$,

$$R[m] = R_c(mT)$$

$$\text{PSD}(\Omega) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \text{PSD}_c\left(\Omega + \frac{2n\pi}{T}\right)$$

The signal information is contained in the term $n=0$; all other terms contribute to aliased noise. Generally, sampling is done at a high enough frequency so that only the neighboring terms (say, $n=-2, -1, 1, 2$) contribute any significant noise.

The terms of PSD_c simply add, because the aliased terms of $X_c(\Omega)$ are uncorrelated. Clearly, from figure B.2, the frequency components

being added of $X_c(\Omega)$ are different or, if the same frequency ($\pm f_2$) they are out of phase; thus, indeed uncorrelated.

Figure B.2. Sampled Signals

$X_c(\Omega)$ and $X(\Omega)$ represent a continuous time signal and the sampled signal. The frequency components from f_1 to f_2 are added with the aliased components from $-f_3$ to $-f_2$, respectively. Also, the frequency components from $-f_1$ to $-f_2$ are added with the aliased components from f_3 to f_2 respectively.

Appendix C

C.1 Pseudo Code for the Algorithm of Section 5.3

Before proceeding with this code, refer to section 5.3. All comments relate to the description of the algorithm in that section.

GetListOfPossibleSolnsWithInitBdryLocs()

/* Let GBBD = generic BBD, i.e. BBD with no boundaries */

**/* Create a List of Possible Solutions with initial Boundary Locations:
ListofPS_InitBdryLocs. */**

BEGIN

Set all GBBD nodes to not visited;

/* Call DownwardDFS to create a List of Sections */

NewSectionID = 1;

DownwardDFS (root_ptr, NewSectionID);

**/* Create a List of possible Solutions by determining the initial
locations of boundaries in each section. */**

FirstSection = True;

For each section of the GBBD ListOfSections Do

BEGIN

If (FirstSection) Then

BEGIN

Add copy of GBBD to ListOfPS_InitBdryLocs;

FirstSection = False;

END;

If [(numDigitalInputs = 0) AND (numAnalogOutputs = 0)] Then

**/* Section type 2: All inputs of this section are FA; all
outputs are FD. The initial locations of the boundaries, all of
which are AtoD, are just above the FD outputs. */**

For each BBD on the ListOfPS_InitBdryLocs Do

BEGIN

Increment BBD's numBoundaries;

AddBdry(AtoD, ABOVE, sectionOutputList);

END;

```

Else If [(numAnalogInputs = 0) AND (numDigitalOutputs = 0)]
Then
  /* Section type 3: All inputs of this section are FD; all
  outputs are FA. The initial locations of the boundaries, all of
  which are DtoA, are just below the FD inputs. */

  For each BBD of the ListOfPS_InitBdryLocs Do
    BEGIN
      Increment BBD's numBoundaries;
      AddBdry(DtoA, BELOW, sectionInputList);
    END;

Else If [(numAnalogInputs > 0) AND (numDigitalOutputs > 0)] OR
[(numAnalogInputs > 0) AND (numDigitalOutputs > 0)] Then
  /* Section type 4: Section inputs are FA and FD and /or
  section outputs are FA and FD. */

  For each BBD on the ListOfPS_InitBdryLocs Do
    BEGIN
      If (SpecialBrPt(Section_ptr)) Then
        BEGIN

          /* Make a duplicate set of BBDs since a special branch
          point means two different sets of initial boundary
          locations. */
          Make a copy of BBD;

          To copy Do
            BEGIN
              /* Section type 4b: This set of initial boundary
              locations places boundaries as far away from FA
              inputs and outputs without causing greater than 2
              conversions per segment path. AddBdry calls
              FindFirstBrPt to place AtoD boundaries above and
              DtoA boundaries below the first branch point
              encountered; and then calls Move to propagate the
              boundaries further checking that proper conditions
              hold for movement across branch points. */

              AddBdry(AtoD, ABOVE_BrPt, sectionInputList);
              AddBdry(DtoA, BELOW_BrPt, sectionOutputList);
            END;
          END;
        END;
      END;
    END;
  END;

```

For each boundary of BBD_listOfBoundaries
 Move(boundary_ptr);

Add 2 to BBD's numBoundaries;
 Add BBD to ListOfPS_InitBdryLocs;
 END;

END;

To BBD Do

BEGIN

/* Section type 4a: This set of initial boundary
 locations places AtoD boundaries above FD outputs and
 DtoA boundaries below FD inputs. */

AddBdry(AtoD, ABOVE, sectionOutputList);
 AddBdry(DtoA, BELOW, sectionInputList);

If [(numDigitalInputs = 0) OR (numDigitalOutputs = 0)]
 Then

Increment BBD's numBoundaries

Else

Add 2 to BBD's numBoundaries;

END;

Else

/* Section type 1: All inputs and outputs of the section
 are FA or they are all FD, thus no boundaries are
 required */
 Continue;

END; /* each section */

END. /* program */

DownwardDFS (node_ptr, sectionID)

/* DownwardDFS proceeds down through the BBD visiting each node and determining if it is an input or an output of a section. The procedure SetForced is called to set section inputs and outputs. The local variable sectionID indicates the current section. The global variable NewSectionID is used to create new section IDs. */

BEGIN

IsNewSection = False;

If (node not visited) Then

BEGIN

Set node to visited;

If (node domain is forced) Then

If (node is an BBD input) Then

SetForced(INPUT, sectionID, node_ptr)

Else

BEGIN

IsNewSection = True;

SetForced(OUTPUT, sectionID, node_ptr)

END;

If (node has > 1 parent) Then

For each parent Do

UpwardDFS(parent_ptr, sectionID);

For each child Do

IF (IsNewSection) Then

BEGIN

NewSectionID = NewSectionID + 1;

SetForced(INPUT, NewSectionID, node_ptr);

/* if node has a child, then it cannot be a BBD output */

Downward DFS(child_ptr, NewSectionID);

END

Else

DownwardDFS (child_ptr, sectionID);

END

END;

UpwardDFS (node_ptr, sectionID)

/* UpwardDFS proceeds up through the BBD visiting each node and determining if it is an input or an output of a section. The procedure SetForced is called to set section inputs and outputs. The local variable sectionID indicates the current section. The global variable NewSectionID is used to create new section IDs. */

BEGIN

IsNewSection = False;

If (node not visited) Then

BEGIN

Set node to visited;

If (node domain is forced) Then

BEGIN

IsNewSection = True;

SetForced(OUTPUT, sectionID, node_ptr);

END;

For each parent Do

If (IsNewSection) Then

BEGIN

NewSectionID = NewSectionID + 1;

SetForced(OUTPUT, NewSectionID, node_ptr);

UpwardDFS(parent_ptr, NewSectionID);

END

Else

UpwardDFS(parent_ptr, sectionID);

If (node has > 1 child) Then

For each child Do

DownwardDFS(child_ptr, sectionID);

END

END;

SetForced (InOrOut, sectionID, node_ptr)

/ SetForced creates a list of sections for the GBBD. Each call to SetForced specifies node_ptr as an input or an output of the section identified by sectionID. If the section sectionID is not already on the list of sections, a new section node is created and added to the list. */*
BEGIN

 Search for section sectionID on ListOfSections.

If (the section does not exist) **Then**

 Create a new section node and add to ListOfSections;

If (InOrOut = INPUT) **Then**

BEGIN

 Add node_ptr to sections input list;

If (node domain is Analog) **Then**

 increment section's numAnalogInputs

Else

 increment section's numDigitalInputs;

END

Else

BEGIN */* InOrOut = OUTPUT */*

 Add node_ptr to section's output list;

If (node domain is Analog)

 increment section's numAnalogOutputs

Else

 increment section's numDigitalOutputs;

END

END;

Function SpecialBrPt (section_ptr) : Boolean;

/*

SpecialBrPt returns true if the BBD section contains at least one branch point with both FA and FD ancestors and both FA and FD descendents; and with either at least one parent with FA and FD ancestors or at least one child with FA and FD descendents.

***/**

BEGIN

SpecialBrPt = False;

For each branchPt of section Do

BEGIN

FA_anc = False;

FD_anc = False;

FA_desc = False;

FD_desc = False;

parent_FAandFDanc = False;

child_FAandFDdesc = False;

For each parent of branchPt Do

If (parent forced) Then

If (parent FA) Then

FA_anc = True

Else

FD_anc = True

Else

BEGIN

ancType = Anc(parent_ptr);

If (ancType = FAandFD) Then

BEGIN

FA_anc = True;

FD_anc = True;

parent_FAandFDanc = True;

END;

Else If (ancType = FA) Then

FA_anc = True

Else

FD_anc = True;

```

    If (parent_FAandFDanc = True) Then
        leave; /* for each parent
    END /*Else

If (FA_anc and FD_anc) Then
    For each child of branchPt Do
        If (child forced) Then
            If (child FA) Then
                FA_desc = True
            Else
                FD_desc = True;
        Else
            BEGIN
                descType = Desc(child_ptr);
                If (descType = FAandFD) Then
                    BEGIN
                        FA_desc = True;
                        FD_desc = True;
                        child_FAandFDdesc = True;
                    END
                Else If (descType = FA) Then
                    FA_desc = True
                Else
                    FD_desc = True;

                If [FA_desc AND FD_desc AND (parent_FAandFDanc OR
                    child_FAandFDdesc)] Then
                    leave; /* for each child */

            END; /* else */

If [FA_anc AND FD_anc AND FA_desc AND FD_desc AND
    (parent_FAandFDanc or child_FAandFDdesc)] Then
    BEGIN
        SpecialBrPt = True;
        leave; /* For each BrPt */
    END; /*for each branch point

END;

```

Function Anc (node_ptr): ForcedType;

If node_ptr has FA and FD ancestors (in the *section*), return FAandFD;

If node_ptr has only FA ancestors, return FA;

If node_ptr has only FD ancestors, return FD;

END;

Function Desc (node_ptr): ForcedType;

If node_ptr has FA and FD descendants (in the *section*), return FAandFD;

If node_ptr has only FA descendants, return FA;

If node_ptr has only FD descendants, return FD;

END;

AddBdry (type, AboveOrBelow, ptrtoListOfNodes);

/*AddBdry calls **InsertBdry** for section types 2, 3, 4a to actually insert the boundary into the BBD. **AddBdry** calls **FindFirstBrPt** for section type 4b to place AtoD boundaries above and DtoA boundaries below the first branch point encountered. ***/**

BEGIN

 DtoA_SpecSheet_ptr = AtoD_SpecSheet_ptr = nil;

If (type = DtoA) **Then**

If (AboveOrBelow = BELOW) **Then**

BEGIN

For each node of ptrtoListOfNodes **Do**

If (node FD) **Then**

InsertBdry(AsChild, node_ptr, DtoA_bndry);

END

Else **/* AboveOr Below = BELOW_BrPt** ***/**

BEGIN

For each node of ptrtoListOfNodes **Do**

If (node FA) **Then**

FindFirstBrPt(DtoA_bndry, node_ptr);

END

Else **/* type = AtoD** ***/**

If (AboveOrBelow = ABOVE) **Then**

BEGIN

For each node of ptrtoListOfNodes **Do**

If (node FD) **Then**

InsertBdry(AsParent, node_ptr, AtoD_bndry);

END

Else **/* AboveOr Below = ABOVE_BrPt** ***/**

BEGIN

For each node of ptrtoListOfNodes **Do**

If (node FA) **Then**

FindFirstBrPt(AtoD_bndry, node_ptr);

END

END;

FindFirstBrPt (fn, node_ptr);

**/* FindFirstBrPt locates, within the section, the nearest branch point
to node_ptr. (Only used for section type 4b) */**

BEGIN

If (fn = DtoA_bndry) Then

For each parent of node_ptr Do

If (parent is a branchpt OR forced) Then

InsertBdry(AsParent, node_ptr, DtoA_bndry)

Else

FindFirstBrPt(fn, parent_ptr)

Else /* fn = AtoD_bndry */

For each child of node_ptr Do

If (child is a branchpt OR forced) Then

InsertBdry(AsChild, node_ptr, AtoD_bndry)

Else

FindFirstBrPt(fn, child_ptr);

END;

InsertBdry (AsChildOrParent, OfNode_ptr, fn):

/* InsertBdry actually inserts a boundary. For each section, there is only one specification form for each boundary type since the boundary is shared. The spec sheet holds a list of the shared boundaries it specifies. */

BEGIN

 Create a new BBD node of function fn and pointed to by
 newNode_ptr;

 If (AsChildOrParent = AsChild) Then

 InsertAsChild(newNode_ptr, OfNode_ptr)

 Else

 InsertAsParent(newNode_ptr, OfNode_ptr);

 If (fn = DtoA_bndry) Then

 BEGIN

 If (DtoA_SpecSheet_ptr = nil) Then

 Create a new Spec Sheet pointed to by DtoA_SpecSheet_ptr;

 Set BBD node specSheet_ptr to DtoA_SpecSheet_ptr;

 Add node_ptr to list of shared nodes in DtoA_SpecSheet;

 END;

 Else /*fn = AtoD_Bndry)

*/

 BEGIN

 If (AtoD_SpecSheet_ptr = nil) Then

 Create a new Spec Sheet pointed to by AtoD_SpecSheet_ptr;

 Set BBD node specSheet_ptr to AtoD_SpecSheet_ptr;

 Add node_ptr to list of shared nodes in AtoD_SpecSheet;

 END;

END;

Move (bndry_ptr)

/* Make more analog but never create greater than 2 boundaries in any one section path. (Only used for section type 4b) */

BEGIN

If (bndry_type = AtoD_bndry) Then

BranchPt = parent of bndry

Else

BranchPt = child of bndry;

GoAhead = True;

all_DtoA = AllChildrenDtoA(BranchPt);

all_AtoD = AllParentsAtoD(BranchPt);

/* Test all branch point conditions. If they all hold, GoAhead is TRUE. */

If (BranchPt FD) Then

/*condition 1*/

GoAhead = False;

Else

BEGIN

For each parent of BranchPt Do

If (parent not a bndry) Then

If (parent has only FA Anc within section) Then

/* condition 2 */

BEGIN

GoAhead = False;

leave; /* for */

END;

Else If (parent has FD and FA Anc within section) Then

If NOT(all_DtoA) Then

/* condition 4 */

BEGIN

GoAhead = False;

leave; /* for */

END;

If (GoAhead) Then

For each child of BranchPt Do

If (child not a bndry) Then

If (child has only FA Desc within section) Then

/* condition 3*/


```

        BEGIN
        GoAhead = False;
        leave; /* for */
        END;
    Else If (child has FD and FA Desc within section) Then
        If NOT (all_AtoD) Then
            /* condition 5 */
            BEGIN
            GoAhead = False;
            leave;
            END;
    If (GoAhead) Then
        /* If all branch point conditions hold, then move across
        branch point */
        BEGIN
        For each parent of BranchPt Do
            If (parent is AtoD) Then
                Remove bndry from BBD and from BBD bndryList;
            Else If (parent is FD) Then
                BEGIN
                InsertBdry (AsChild, parent_ptr, DtoA_bndry);
                Add bndry to front of BBD bndryList;
                /* do not consider again */

                END;
            Else /* parent not forced and not bndry */
                For each parent of parent Do
                    BEGIN
                    FindFirstBrPt(DtoA, parent_ptr, bndry_ptr);
                    Add bndry_ptr to end of BBD bndryList;
                    /* consider again */

                    END;
        For each child of BranchPt Do
            If (child is DtoA) Then
                Remove bndry from BBD and from BBD bndryList;
            Else If (parent is FA) Then
                BEGIN
                InsertBdry(AsParent, child_ptr, AtoD_bndry);
                Add bndry to front of BBD bndryList;
                /* do not consider again */

```

```

END;
Else                               /* child not forced and not bndry */
  For each child of child Do
    BEGIN
      FindFirstBrPt(AtoD, child_ptr, bndry_ptr);
      Add bndry_ptr to end of BBD bndryList;
                                     /* consider again */
    END;
  END;
Else                               /* GoAhead is False */
  BEGIN
    For each child of BranchPt Do
      If (child is a boundary) Then
        Change location in BBD bndryList to front of list;
                                     /* do not consider again */
      For each parent of BranchPt Do
        If (parent is a boundary) Then
          Change location in BBD bndryList to front of list;
                                     /* do not consider again */
        END;
      END;
    END;
  END;
END;

```

C.2 Pseudo code for the Algorithm of Section 5.5.1

Before proceeding with this code, refer to section 5.5.1. All comments relate to the description of the algorithm in that section.

GetListOfPossSolns()

/* Create a list of all possible solutions with a minimum number of conversions and no more than two conversions per any section path. */

BEGIN

For each BBD on the ListOfPS_InitBdryLocs Do

BEGIN

Copy BBD to ListOfPossSolns and to currentSoln;

While more combinations of boundary locations exist Do

BEGIN

NewLocation(currentSoln);

Copy currentSoln to ListOfPossSolns;

END;

END;

/* As described in section 5.5.1, possible solutions with different number of boundaries may arise. */

Order ListOfPossSolns in increasing number of boundaries; and for a given number of boundaries order list in decreasing number of analog blocks;

END.

NewLocation (BBD_ptr)

/* NewLocation considers each boundary in turn until one boundary is allowed to advance. Then, it calls **MoveBdry** to advance the boundary and returns. For the boundaries that were not allowed to advance, **MoveBdry** is called to return the boundary to its start location. ***/**

BEGIN

For each boundary on BBD ListOfBoundaries Do

BEGIN

Set nextBdry to next boundary on BBD ListOfBoundaries;

If [(boundary is AtoD AND parent is FA) OR (boundary is DtoA AND child is FD)] Then

/* Move boundary to its start location. */

MoveBdry(BACKWARD, boundry_ptr)

Else If [(boundary is AtoD AND parent is a branch point) OR (boundary is DtoA AND child is a branch point)] Then

BEGIN

If **NOT(AllBoundaryConditionsHold(boundary_ptr))**

MoveBdry(BACKWARD, boundary_ptr)

Else

BEGIN

/* Advance boundary one location. */

MoveBdry(FORWARD, boundary_ptr);

leave; **/* For */**

END

END

Else

BEGIN

MoveBdry(FORWARD, boundary_ptr);

leave;

END

END /*For */

END.

Function **AllBoundaryConditionsHold** (boundary_ptr) : BOOLEAN;
 BEGIN

 AllBoundaryConditionsHold = TRUE;

 If (boundary type is AtoD)

 BranchPt = parent of boundary

 Else

 BranchPt = parent of boundary;

 allAtoD = **AllChildrenAtoD**(BranchPt);

 allDtoA = **AllParentsDtoA**(BranchPt);

 /* Test all branch point conditions. If they all hold,
 AllBoundaryConditionsHold is TRUE.

*/

 If (BranchPt FA)

 /* condition 1 */

 AllBoundaryConditionsHold = FALSE

 Else

 BEGIN

 For each parent of BranchPt Do

 If (parent not a boundary)

 If (parent has only FD ancestors within section)

 /* condition 2 */

 BEGIN

 AllBoundaryConditionsHold = FALSE;

 leave;

 END

 Else If (parent has FA and FD ancestors within section)

 If NOT (allAtoD)

 /* condition4 */

 BEGIN

 AllBoundaryConditionsHold = FALSE;

 leave;

 END;

 If (AllBoundaryConditionsHold)

 For each child of BranchPt Do

 If (child not a boundary)

 If (child has only FD descendents within section)

 /* condition 3 */

 BEGIN

 AllBoundaryConditionsHold = FALSE;

 leave;

 END

Else If (child has FD and FA descendents within section)

 If NOT (allDtoA)

 /* condition 5 */

 BEGIN

 AllBoundaryConditionsHold = FALSE;

 leave;

 END

 END

END

MoveBdry (direction, bndry_ptr)

BEGIN

 If (direction = FORWARD)

 /* Advance boundary by one position increasing the number of digital blocks by one. */

 BEGIN

 If (bndry type is AtoD)

 If (parent is not a branch point)

MoveUp(bndry_ptr)

 Else

MoveAcrossBranchPt(parent_ptr)

 Else /* bndry type is DtoA */

 If (child is not a branch point)

MoveDown(bndry_ptr)

 Else

MoveAcrossBranchPt(child_ptr)

 END

 Else /* direction = BACKWARD */

 /* Return boundary to its start location. */

MoveToStartLocation(bndry_ptr);

END.

MoveUp (Bndry_ptr)

BEGIN

Swap location of bndry and bndry parent;

END.

MoveDown (Bndry_ptr)

BEGIN

Swap location of bndry and bndry child;

END.

MoveAcrossBranchPt(BBD_ptr, BranchPt_ptr)

BEGIN

IsFirst = TRUE;

tempBBDBndryList = nil;

tempRemoveList = nil;

tempStartLocList = nil;

For each parent of BranchPt Do

If (parent is a DtoA boundary)

BEGIN

/* Use of RemoveList and StartLocList discussed in section 5.5.1. */

To tempRemoveList add parent RemoveList;

To tempStartLocList add parent StartLocList;

Remove parent from BBD and from BBD ListOfBoundaries;

END

Else

BEGIN

Create a new AtoD boundary: newBdry, and insert as child of parent;

Set newBdry RemoveList to nil and StartLocList to current position;

Add newBdry_ptr to front of tempBBDBdryList ;

/*

First boundary will contain the Remove list and therefore, it is not added to the list; however, all other boundaries are.

*/

If NOT (IsFirst)

Add newBdry to tempRemoveList;

IsFirst = FALSE;

END

```

For each child of BranchPt Do
  If (child is an AtoD boundary)
    BEGIN
      To tempRemoveList add child RemoveList;
      To tempStartLocList add child StartLocList;
      Remove child from BBD and from BBD ListOfBoundaries;
    END
  Else
    BEGIN
      Create a new DtoA boundary: newBdry, and insert as parent
      of child;
      Set newBdry RemoveList to nil and StartLocList to current
      position;
      Add newBdry_ptr to front of tempBBDBdryList ;

      If NOT (IsFirst)
        Add newBdry to tempRemoveList;

      IsFirst = FALSE;
    END

  To last element of tempBBDBdryList Do
    BEGIN
      Add tempRemoveList to its RemoveList;
      Replace its StartLocList by tempStartLocList;
    END

  Insert tempBBDBdryList before nextBdry;
  Set nextBdry to first element of tempBBDBdryList;
END.

MoveToStartLocation(bndry_ptr);
BEGIN
  For each boundary specified on bndry StartLocList Do
    Add boundary to BBD and BBD ListOfBoundaries;

  For each boundary specified on bndry RemoveList Do
    Remove boundary from BBD and BBD ListOfBoundaries;

  Remove bndry from BBD and BBD ListOfBoundaries;
END.

```


GetBoundaryID (bndryStartLoc, id)

/* Whenever a boundary is selected, **GetBoundaryID** is called to determine its id. **GetBoundaryID** ensures that whenever a boundary is recreated, it is given its original id. This allows the comparison of boundary locations on different BBDs. Here, a **CreateLocationList** is used. It holds the created location for a particular id.

For instance, for the example of figure 5.20 the **CreateLocationList** would initially be as follows:

```
CreateLocationList--> id#1: parent: B3  --> id#2: parent: B8 --> .
                        child : B4          child: B9
```

After trying all the different boundary locations, the **CreateLocationList** would be as follows:

```
CreateLocationList--> id#1: parent: B3  --> id#2: parent: B8 -->
                        child : B4          child: B9

--> id#3: parent: B2  --> id#4: parent: B6 --> id#5: parent: B10
      child : B6          child: B7          child: B11      */
```

BEGIN

If (bndryStartLoc matches any location on the **CreateLocationList**)
 return that id

Else

BEGIN

 Create a new id;

 Add new id and bndryStartLoc to **CreateLocationList**;

 return new id;

END

END

References

- [AleG83] Gotz Alefeld and Jurgen Herzberger. *Introduction to Interval Analysis*, New York: Academic Press, 1983, Chpt. 1 to 3.
- [AllP87] Phillip E. Allen and Douglas R. Holberg. *CMOS Analog Circuit Design*, New York: Holt, Rienhart, and Winston, Inc., 1987, Chpt 1.
- [AllP89] Phillip E. Allen. "CAD for Analog VLSI," Talk given at Carleton University, Ottawa, April 24, 1989.
- [BreF86] F.D. Brewer and D.D. Gajski. "Expert-System Paradigm for Design," *Proc. 1986 ACM/IEEE 23rd Design Automation Conference (DAC)*, June-July 1986, pp. 62-68.
- [CarL88] L.R. Carley and Rob A. Rutenbar. "How to Automate Analog IC Designs," *IEEE Spectrum*, August 1988, pp. 26-30.
- [CarL89] L.R. Carley, et al. "ACACIA: The CMU Analog Design System," *Proc. 1989 IEEE Custom Integrated Circuits Conference (CICC)*, May 1989, pp. 4.3.1-4.3.5.
- [DavW87] Wilbur B. Davenport, Jr. and William L. Root. *An Introduction to the Theory of Random Signals and Noise*, New York: IEEE Press, 1987, Chpt. 12 and 13.
- [DeGM87] M.G.R. DeGrauwe, et al. "IDAC: An Interactive Design Tool for Analog CMOS Circuits," *IEEE Journal of Solid State Circuits*, Vol. SC-22, No. 6, Dec. 1987, pp. 1106-1115.
- [DevA89] Analog Devices. *Component Selection Disk*, Lamb Data Systems, 1989, (M1302-5-6/89).
- [GreR86] R. Gregorian and G.C.Temes. *Analog MOS Integrated Circuits For Signal Processing*, New York: John Wiley & Sons, Inc. 1986, Chpt. 1 and 5, pp502-503.
- [HarR89] Ramesh Harjani. "OASYS: A Framework for Analog Circuit Synthesis," Research Report No. CMUCAD-89-24, Department of Electrical and Computer Engineering, Carnegie Mellon University, March 1989.

- [HilF74] Frederick S. Hillier and Gerald J. Lieberman. *Operations Research*, Holden-Day, Inc., San Francisco, 1974, pp.699-704.
- [IsmM90] M. Ismail and J. Franca, et al, *Introduction to Analog VLSI Design Automation*, Kluwer Academic Publishers, Boston, Mass., 1990.
- [KumY90] Yatish Kumar. "Data Path Optimization in Mixed Digital/Analog Behavioral Synthesis," forth year project, Carleton University, April 1990.
- [LeeE89] E. Szeto Lee, H.T. Chang, et al. "The Input/Output Specifications Analyzer for IC Designs," *Proc. 1989 IEEE Custom Integrated Circuits Conference (CICC)*, May 1989, pp. 19.1.1-19.1.4.
- [MagN89] Nanci J. Magoun. (publisher). "ASICs defined," *ASIC Technology and News*, Vol. 1, No.1, (May 1989), p. 5.
- [MavJ83] J. Mavor, M.A. Jack and P.B. Denyer. *Introduction to MOS LSI Design*, London: Addison-Wesley Publishers Limited, 1983, chapter 8.
- [MezP87] P.J. Meza and W. Gross. "'Standard Tiles': A New Design Method for High Performance Analog ICs," *Proc. 1987 ACM/IEEE Design Automation Conference (DAC)*, June 1987, pp. 42-49.
- [MidJ88] Jenny Midwinter "Improving Interconnect and Register Allocation for the Behavioral Synthesis of Digital Circuits," Masters. Thesis, Carleton University, 1988.
- [NiiH86] H. Penny Nii. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine*, Vol. 7, No. 2, Summer 1986, pp. 38-53.
- [OppA89] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*, New Jersey: Prentice Hall, Inc., 1989, pp. 351-356.
- [PapA84] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, Inc., 1984.

- [PauP88] Pierre G. Paulin, "High-Level Synthesis of Digital Circuits Using Global Scheduling and Binding Algorithms," PhD. Thesis, Carleton University, January 1988.
- [RabL78] L.R. Rabiner and R.W. Schafer. *Digital Processing of Speech Signals*, New Jersey: Prentice Hall, Inc., 1978, Chpt.5.
- [RahT90] Tim Rahrer. "A Digital Signal Processing-Based Hearing Prosthesis and Implementation of Principal Components Analysis for a Tactile Aid," Masters Thesis, Carleton University, October 1990.
- [RunS89] Stan Runyon. "Mixed-Signal Design: Surmounting the Barrier," *High Performance Systems*, Vol X, No. 5, May 1989, pp. 26-31.
- [StrF77] Ferrel G. Stremler. *Introduction to Communication Systems*, Addison-Wesley Publishing Company, Massachusetts, 1977, chpt3.
- [WarG88] George H. Warren and Charles Hugo. "Noise, Crosstalk and Distortion in Mixed Analog/Digital Integrated Circuits," *Proc. 1988 IEEE Custom Integrated Circuits Conference*, 1988, pp. 12.1.1-12.1.4
- [WebS89] Samuel Weber. "Analog Gets Automated," *Electronics*, Vol. 62, Nov. 1989, pp. 62-63.
- [ZhuX88] XI-An Zhu and Melvin A. Breuer. "A Knowledge-Based System for Selecting Test Methodologies," *IEEE Design and Test of Computers*, Vol. 5, No. 5, Oct. 1988, pp. 41-59.

3 of/de 3

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

END

03·03·92

FIN