AUTOMATIC SYNTHESIS OF CMOS DIGITAL/ANALOG CONVERTERS

by

Robert McKinstry Robinson Neff

Memorandum No. UCB/ERL M95/28

21 April 1995

AUTOMATIC SYNTHESIS OF CMOS DIGITAL/ANALOG CONVERTERS

by

Robert McKinstry Robinson Neff

Memorandum No. UCB/ERL M95/28

21 April 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Automatic Synthesis of CMOS Digital/ Analog Converters

by

Robert McKinstry Robinson Neff

Abstract

Automatic Synthesis of CMOS Digital/Analog Converters

by

Robert McKinstry Robinson Neff

Doctor of Philosophy in

Engineering -- Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Paul R. Gray, Chair

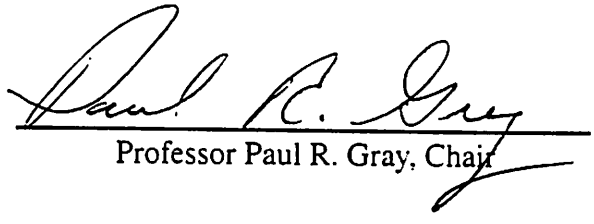
Synthesis of analog functional blocks in integrated circuits offers promise for improved designer productivity. By developing module generators for commonly used analog circuit elements, a synthesis methodology may be matched to a particular application, with approaches and algorithms determined by the particular needs of target circuit type. An analog circuit designer should be able to input design specifications and underlying technology information, and a synthesis methodology should determine circuit parameter values and dimensions, creating the required mask layouts. Slow, tedious design and redesign methods should be replaced by one in which the computer finds minimum cost designs which meet performance requirements. This work implements synthesis methods for a widely used analog block, the digital/analog converter (DAC).

In practice, there are a number of difficult problems in synthesis methodologies. Accurate performance prediction is required, including effects of parasitic elements, and a mix of device level and circuit block level analyses. A design sizing and selection process must be determined, and methods for circuit layout must be selected.

This thesis focuses on approaches best suited to the digital/analog converter synthesis problem. A mixed analysis/simulation model for DAC behavior is developed, including an explicit inclusion of parasitic capacitive and spacing effects. A design optimization approach is developed,

using a single level optimization, mixing device sizing and architecture parameters in one step. A mixed integer non-linear programming algorithm was adapted to the requirements of this optimization. Layout approaches used previously in digital datapath applications were adapted to DAC layout, producing dense layouts.

The module generation process was demonstrated through a high performance video DAC prototype, with 8-bit linearity and 100 MSample/s performance. The prototype met most performance specifications, and discrepancies between expected and observed performance were traced back to errors in the technology database input. Cell size was comparable to a manual design. The module synthesis process, including development of an initial design database, requires design time comparable to a manual approach, but subsequent reuse of the database has resulted in implementation times of a few days or even hours, thus demonstrating the ability of this combination of approaches to dramatically reduce the implementation time for high performance, digital/analog converter designs.



Professor Paul R. Gray, Chair

Table of Contents

CHAPTER 7

APPENDIX A

Acknowledgments

This work would not have been possible without the unending support, encouragement, and guidance of my research advisor, Dr. Paul R. Gray. His advice on this thesis work, as well as more general research questions, have helped me understand a proper role for university based research in Electrical Engineering, and how it can make an impact beyond the university. Dr. Alberto Sangiovanni-Vincentelli has also provided helpful insights and advice.

I have benefitted from encouragement from several generations of graduate students. Steve Lewis, Bosco Leung, and Joey Durenburg helped me through my Masters' work, have always been encouraging when I have seen them at conferences. C.K. Wang has been an especially good friend, helping me understand the value of the doctoral degree when we worked together, and encouraging me after I returned to UCB. He knew I would finish, even when I was not so sure. Greg Uehara, Cormac Conroy, Gani Jusuf, Ken Nishimura, Tim Hu, and Ed Liu, all members of a later generation, have helped give advice, share new ideas, and shown me how to succeed at Berkeley. I am indebted to today's generation of students, including Dave Cline, Thomas Cho, Edoardo Charbon, Eric Felt, Cynthia Keyes, and especially Henry Chang, who have helped with detailed discussions of common circuit, CAD, and design issues.

When I discussed returning to Berkeley with my managers at IBM Corp, I was surprised at how much they were willing to support my future plans. Fritz Weidmer, my immediate manager when I began my educational leave, and Ed Clausell, who took over when Fritz retired, have kept me well appraised of the changing situation there. Having the support of the educational leave of absence has been a comforting factor during my time in school.

My parents, Samuel and Ruth Neff, have shown me how to get to this point by their encouragement and their example, letting me find my goals, and helping me towards them. I'm sorry mom, this is not solar energy or windmills I am working on, but I find it a lot more interesting.

Most of all, I am grateful to my wife, Nancy Robinson Neff, for her support and devotion, throughout my years in this Ph. D. program. She has helped keep me focused on the overall goal,

and not distracted by setbacks along the way. As our lives continue together, I hope to support her goals with as much fortitude.

This research effort was funded by the Semiconductor Research Corporation under contract SRC-94-DC-324, and I am grateful for their support as well.

CHAPTER 1

Introduction

The rapid progression of design automation for digital integrated circuits (ICs) has enabled rapid synthesis of digital designs. Increasing IC areal densities and chip sizes have allowed greater functionality in digital ICs. Analog circuit design methodologies have not kept up with this pace. Although the circuit design software used by today's analog IC design engineers is more user friendly, and converges to solutions better than fifteen years ago, there has not been a parallel explosion of design capability for analog circuit designers. Today's analog circuit designers often use a combination of hand analysis and circuit simulation that was widely available in 1980. There has been some improvement in efficiency and capability since that time, thanks to the development of improved user interfaces for software tools, and new capabilities for behavioral, high level, and mixed signal simulation, but the task of circuit design is still reserved for experience circuit designers using manual design and layout.

The general goal of Analog Computer Aided Design (ACAD) is to reduce the manual design and layout time required for circuit design. Improving analytic tools is a first step, and creating in-house cell libraries from previous designs is a low tech way to speed circuit development through design re-use, but more aggressive techniques which result in design information reuse and automated design synthesis offer greater promise for reducing design time. These ACAD approaches include analog standard cell libraries and analog synthesis approaches such as circuit synthesis,

layout synthesis, and module generation, and hierarchical design synthesis. For now, some working definitions of these terms are needed. Analog standard cell approaches consist of a library of well characterized circuit blocks, which the designer assembles into a full analog sub-system. In general analog synthesis, an input specification is used to drive a process which creates elements for meeting the design specification. In circuit synthesis a set of circuit specifications are used to drive a synthesis process which sizes the circuit elements and devices. In layout synthesis an analog circuit netlist, including additional information about parasitics, matching, and performance constraints, creates an appropriate layout. Module generation will be used to describe the combination of analog circuit and layout synthesis, with tools and libraries adapted to specific analog modules such as opamps, filters, analog/digital, or digital/analog converters. Hierarchical design synthesis takes an analog design for a complex block or analog subsystem, and decomposes this to lower level sub-block specifications, which may then be either pulled from a cell library, or synthesized using circuit synthesis techniques. All of these techniques promise to increase the productivity of analog circuit designers, and in some cases eliminate the need for expert designers! These approaches will be explored in this chapter.

Design re-use is not new, of course. From the earliest days, the first task of an engineer facing an new design problem has been to look over old designs of his own (or other engineers') to find a circuit or topology which may be applied to his problem. Often this is an informal process, but in many companies libraries of previously designed analog blocks are cataloged for re-use in later designs. Where little or no modification to the existing circuit is required, this may result in a rapid design process, but for large changes to a circuit design, the previous design may just be a starting point for another manual design cycle, and design specific information is not efficiently re-used.

The notion of an analog standard cell library [SMIT89,LABE87,SERH85] came from the obvious success of digital designers' implementations of standard cell approaches to complex circuit problems. By abstracting circuits into logical functions, and predicting the digital performance of these blocks, naive digital designers were able to place and interconnect basic building blocks into complex digital functions, creating IC designs out of digital cells as they would have created digital printed circuit boards out of the TTL building blocks of the previous generation. The promise of an analog standard cell approach was that with the right set of predefined building blocks, a similar

transition from board level to IC level design could take place for analog circuit designers, and a small set of core cells could be reused across many application designs. The user base could relatively naive, and only need to assemble analog building blocks as they would have in a board level design.

In the Micro-linear implementation [LABE87], a large interdependent library of cells was created for this naive user base, but in practice the standard cell approach did not work for these users. The complexities of analog block interaction made design too difficult for the standard cell building block model. While this library failed at its original goal, this set of building blocks has provided a useful library of cells which are routinely reused by expert designers to this day.

In the IBM approach [SMIT89], the system depended upon application circuit designers to create the standard cells for the system, and supplied the tools and methodology to stitch these together in large mixed-signal ICs. These standard cells tended to be application specific, with bias circuit cells designed to match functional cells (such as amplifiers or oscillators). When a new design required circuits similar to those already in the library, designers often redesigned existing circuits, rather than using cells directly out of the library, due to small differences between the application specifications and existing cell specs. This methodology has been used to successfully design large, high performance mixed signal circuits[PHIL94], but has not shown as much promise for design reuse.

In both cases analog standard cells do not have the flexibility to adapt to specifications unforeseen by the original cell designers, even if small variations to the cell design could meet a user specification. Also, as process technology changed, the cell library had to be redesigned for the new process. These problems with analog standard cells point toward an analog synthesis approach.

Analog circuit synthesis is a process in which design specifications are used as an input which are used to select appropriate circuit topologies and size devices. This method promises to solve the limited applicability of standard cells by creating cells to meet new specifications as they are needed. Since the process technology is just an input to the synthesis process, changing technology

is no more complicated than changing the specification. In either case the synthesis a tool will create the best design for the input specification.

While an analog standard cell approach defines a library of cells, to be browsed and checked out when needed, analog synthesis defines a process for creating a design. This design process is a combination of a set of design and technology inputs, and a methodology which transforms these inputs to the final design specification. This is analogous to the analog circuit design process, in which a set of tools are used by an expert designer to create a circuit design from a set of design primitives such as transistors, resistors, and capacitors. To date there has been no successful synthesis methodology which can design circuits from first principles, so design specific knowledge must be incorporated into the methodology. We shall see that this is either through the encapsulation of knowledge in a circuit selection method, or through the definition of topologies and circuit simulation measurements to be used in the synthesis process. Because synthesis defines a process, the input design knowledge is broadly applied to a class of circuits, across technologies, instead of at the single solution point found in a cell library.

1.1 Analog Synthesis

This chapter continues with a look at the requirements for an analog synthesis methodology. In particular, it is necessary to identify the potential users of the analog synthesis, the user requirements, and the elements of analog synthesis methodologies. There has been ongoing work in different aspects of analog synthesis for more than 15 years, and some of these approaches will be reviewed in light of these user requirements. A distinction will be drawn between general analog synthesis and module generation, motivating the choice of module generation. In this thesis the synthesis of Digital-to-Analog Converters (DACs) is described, and the motivation and previous work for this class of circuits will be discussed. The chapter concludes with discussion of the contributions of this work, and the organization of the following chapters.

Before looking further into specific approaches for analog synthesis methodologies, it is important to take a step back to look at some more general questions. Is the tool developed for a specific application, or as a methodology which includes several interesting applications. Who is going to use an analog synthesis tool? What are the requirements for such a tool? What are the

inputs, both for a circuit topology, and an individual design? How is performance measured, and what performance is good enough (and what would be excellent)? We shall see that there are several synthesis performance factors which have limited the acceptance of tools developed to date.

The answer to the first question is that any analog synthesis method must define a process, and supply tools to support that process, rather than be limited to the solution of a single problem. This is particularly true for analog circuits, where there are many different problems which can be attacked with similar methods. An analog synthesis tool optimized for one problem, but which is difficult to apply to similar problems will quickly become obsolete, while one which allows ease of incorporation of new designs may succeed in the long run. This has an important implications for the methodology. The process of applying the methodology to a new analog synthesis problem must be a reasonable task for those who understand the circuit to be synthesized.

1.1.1 The Analog Synthesis Tool User

The next question is "Who is the user?" Should one develop a tool to improve the productivity of those already doing circuit design, and expect them to have a thorough, hands-on understanding of the tool? Should a relative novice to be able to implement a state of the art design with this methodology? Should a system designer, well versed in digital signal processing, be able to use this design methodology to create the analog glue circuits which bind his DSP core to the outside world? In this section these three user classes -- the system engineer, the circuit designer new to this application, and the circuit designer experienced with this application -- are considered.

The creation of turnkey analog synthesis tools for a system designer is a long term goal for analog CAD tools, but is unrealistic at this time. There are too many dependencies in the design process, including inaccurate models, chip level noise issues, and analog circuit interface issues which require some circuit expertise to understand. Also, there is a responsibility issue. Is the user responsible for verification of the design, and its ultimate design success, or should the tool be robust enough to ensure working designs for every input and condition. Historically CAD tool vendors have not been held responsible for the application of their tools, but have depended upon knowledgeable users which can understand the limits of the tools' application.

The second class of user is an important one.This user may not have created the circuit module, but may reuse it for a new application. It is important that he can understand the limitations of the analog synthesis tool.

The user who is already doing this analog circuit design is most important. He is the circuit designer who may implement new designs in the analog synthesis methodology. He has the understanding of the underlying circuit design which must be incorporated into the synthesis process. Having developed the design module, he will be the user best able to exploit the analog synthesis tool.

The key point is that the initial users of the analog synthesis methodology are engineers who are already doing circuit design, and will be able to improve their productivity through use of the analog synthesis tool. This has two implications. The first is that these are knowledgeable engineers, who can understand the inputs and results from the module generator. The second implication is that the engineers who are going to implement new designs are more likely to be familiar and comfortable with circuit simulation language for describing a design, rather than a high level programming language. If the tool uses a familiar input format it will be faster to incorporate new designs [OCHO94].

1.1.2 Performance Standards for Analog Synthesis

There are several metrics for measuring performance of a module generator. The most important are the tool's predictive accuracy, tool run time, and design entry time. The quality of the created designs is a more subjective measure, but is also important.

Adequate tool accuracy is the most important performance standard. When creating a module the user inputs a set of circuit performance specifications to be met. If the tool cannot accurately predict all circuit performance measures, the user may not have an assurance that the developed circuit will meet the specification. One method for circumventing this is to over-specify the design, but this will result in a lower quality result. Tool accuracy has been a particular problem for design tools which use first order device models for short channel MOS transistors.

Tool run time is probably an overused performance metric. There are several module generator tools which take only a few minutes to run, but in the time scale of a typical analog circuit design, speedy tools are not critical. A typical IC design will take days, if not weeks to specify, and weeks (if not months) to fabricate. Having the analog synthesis step take several hours of compute time is not significant in this time scale. In this work, the run time requirement is that the circuit synthesis step take no more than an overnight run on a modern workstation for the most complex circuits.

Tool run time is strongly impacted by the increase in computer capability over time. Computational problems which were considered overly long a few years ago are orders of magnitude faster today. The increasing speed of CPUs and size of workstation main memory have allowed the increase in simulation size from key subcircuits to full chip circuit simulations, running as overnight jobs on an engineering workstation. This gives today's analog CAD engineers an advantage over our predecessors, allowing consideration of methods of attack incomprehensible 15 years ago, and too costly only 5 years ago.

A second time metric, often overlooked, is the time required to implement a new module generator in the design methodology [OCHO94]. This time, which is usually much longer than the tool running time, dominates the man-hours spent creating the first implementation of the design. If the design entry process is much longer than the time required to do a full custom design, or is beyond the capabilities of the engineer who must understands the module, then the tool will not be used.

Quality of a completed module is difficult to measure. For automated analog layout programs, it is typical to see a comparison between a tool and a layout technician. For analog synthesis it is more difficult to make a comparison, because the tool must first size all circuits, and then create mask geometries. A high quality design should match the predicted circuit performance in exhaustive simulation and in fabricated circuits. It also should be comparable to custom layouts in circuit area for the same technology and specifications.

There are other implementation requirements as well. The final predicted design performance must meet the specifications. The technology related inputs should be easy to change. Since designs must work across a range of process and temperature conditions, there must be a way to include the effects of process and temperature variation on circuit performance.

1.2 Analog Synthesis Approaches.

There are four decisions to be made when specifying an analog synthesis methodology. The methodology must choose an estimation approach, a design selection approach, and a layout approach. The fourth decision is the whether to attack the problem in a flat or hierarchical manner. In this section these decisions will be described, and some analog synthesis implementations will be examined in this context. Fig. 1.1 illustrates a typical synthesis approach.

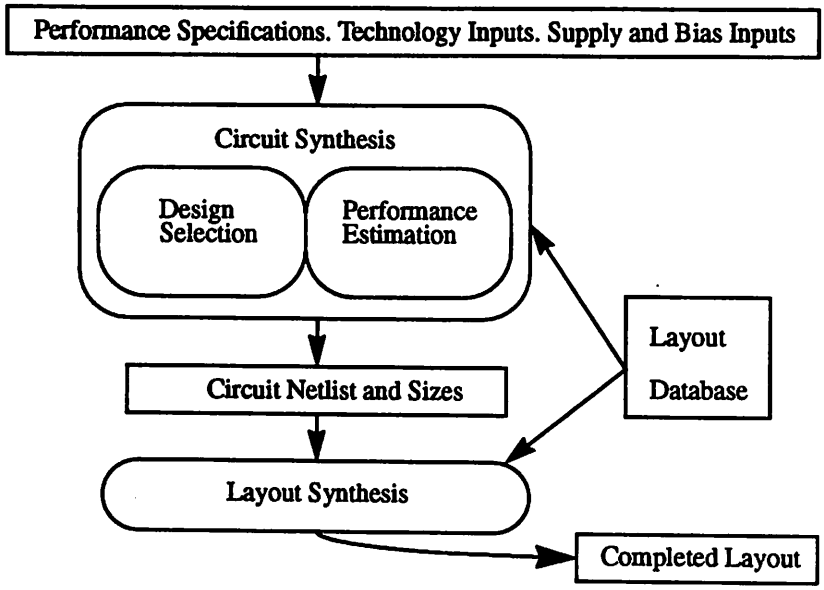


Figure 1.1 Typical Analog Synthesis Flow.

1.2.1 Design Estimation

The first decision is the design estimation approach. What methods should be used to estimate the performance of the circuit for a set of design inputs. In general, there is a trade-off between estimation accuracy and estimation computation time, though that does not hold universally. At one extreme, circuit simulation (SPICE) may be used to estimate performance results. If underlying device models are accurate, and care is taken to include parasitics, this may yield very accurate results for transient, DC, and AC measurements of device level circuits. A second full simulation approach is Asymptotic Waveform Estimation (AWE), which performs AC and DC analysis of the circuit at the device level. Behavioral simulation is the third full simulation approach, in which subcircuits are abstracted, and then their interaction is modelled with a behavioral simulation engine. This has been used for both transient and statistical modelling [LOUI94, LIU93, JUSU93].

Behavioral simulation approaches allow high level simulations which would be impossible with a full simulation approach. System level performance such as Sigma Delta Data Converter linearity can be modeled, which would take too long by full circuit simulation, and too complex for direct analysis. The accuracy is limited by the limitations on the underlying subcircuit models. A last approach is a purely analytic approach, in which design equations are derived which predict circuit performance for design inputs. This is an order of magnitude faster than the other simulation approaches, but accuracy of this approach depends on the accuracy of the modelling which supports it. Where this is used to replace linear behavioral models, there may be no loss of accuracy, but analytic equations using first order device models cannot accurately replace device level circuit models for reasonably sized devices.[1] Also, analysis of a complex circuit my be possible, but difficult compared to a simulation approach. A set of analytic equations predicting opamp behavior may take two weeks to code, while creating a circuit simulation to return the same information may take an afternoon [KOH90,OCHO93]. Fig 1.2 illustrates the application of these techniques graphically. On the horizontal axis is the degree of design abstraction, and on the y axis is the estimation accuracy. The shaded areas indicate where the methods may be applied. Note that analytic approaches become more accurate at higher levels of design abstraction.

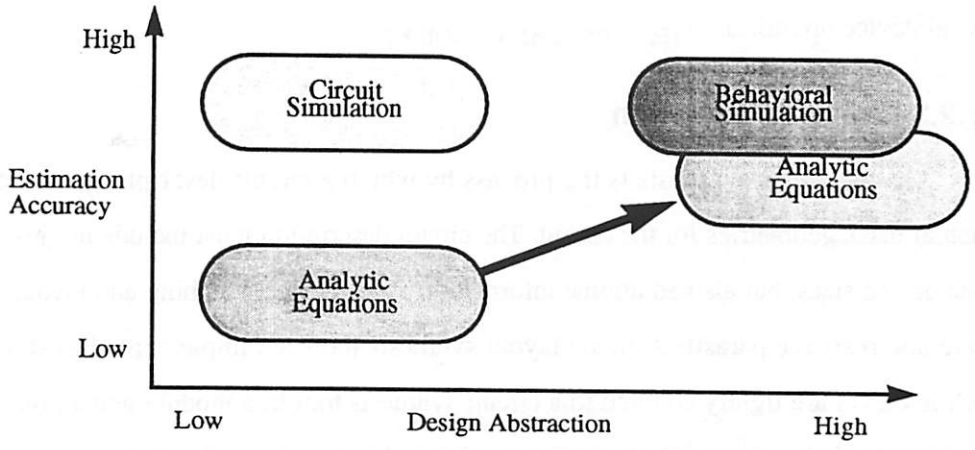


Figure 1.2 Application of Design Estimation Techniques

1. Kundert suggests that first order models are inaccurate for MOS device channel lengths below 10 μm. [KUND93]

1.2.2 Design Selection

The design selection process is the process by which the design method determines the final design. There are two main approaches. Optimization based approaches use some optimization algorithms to reach a final solution. Several different optimization algorithms have been used, ranging from steepest descent with a single lumped cost function to nonlinear constrained optimization to simulated annealing. Besides simple device sizing, there have also been efforts to incorporate topology selection within the optimization algorithm [MAUL92b,HARV92]. Several factors affect the choice of the optimization algorithm, including the design estimation time and the existence of local minima in the solution, In all cases the optimization algorithm is used as a "dumb" numerical technique which attempts to find an optimal solution.[BRAY81, NYE88, DEGR89, CHAN94, GIEL90, JUSU93, HOCE90, KOH89, MAUL93, OCHO94, NING92, SHYU88].

The second type of design selection is a knowledge based approach, in which a design method which mimics the steps an expert designer may take is used to find a good design solution. The idea is that the methodology can save a lot of time if it can march straight to the correct solution using the same logical reasoning a designer may choose, rather than iterating through an optimization algorithm many times. In practice these approaches do reach solutions quickly, but they do can not guarantee any optimality for the result. Also, these design recipes are unique to each topology, and must be re-implemented. as part of every new module[ALLE85, DEGR87, BERK88, HARJ88, MAKR92]. Some selection techniques use a combination of heuristic topology selection with low level device optimization [FOTO94, KOH89, ONOD90].

1.2.3 Layout Automation

Analog layout synthesis is the process by which a circuit description is used to create the actual mask geometries for the circuit. The circuit description must include not just the netlist and the device sizes, but also additional information about device matching and layout related capacitive and resistive parasitics. Some layout synthesis tools are implemented as stand-alone tools, while others are tightly coupled to a circuit synthesis tool in a module generator. When a tool is tightly coupled to synthesis, the netlist may be predetermined, so it may make sense to provide the layout tool with a circuit specific template.

The first important issue for layout synthesis is the means of communicating analog layout information to the tool from the circuit synthesis process. When the tools are loosely coupled, it is typical for the input to include netlists with device sizes, and some additional rules indicating critical device pairs and nets which should be matched. In a performance constraint driven approach [CHOU90,OHTS92,CHAR92], the sensitivities of layout effects on analog circuit performance are passed to the layout tools, and the layout process uses these sensitivities to limit the adverse affect of layout parasitics on performance. When layout is tightly coupled to the synthesis process and the layout is a deterministic process, it may be possible to predict layout parasitics without actually going to the layout step, and compensate for layout parasitics in the circuit synthesis process, or layouts may be generated as part of the synthesis process, for determination of parasitics from actual layouts during circuit synthesis [ONOD90].

Layout synthesis techniques typically use opamp circuits as their model for a typical analog layout. In general opamps have relatively few devices, but large device sizes, and consist of both individual transistors and transistors grouped into differential pairs and cascode configurations. Typically the synthesis tool will size and place these subcells, and then route with area or channel routers. The results tend to work (and look) best when the input circuit fits this opamp model of relatively large devices, with few interconnects.

Analog placement can be separated into two general classes. Procedural approaches use deterministic algorithms to place devices, optimizing device sizes in a template [KOH89, JUSU92,DEGR89], or successively splitting the circuit into smaller parts until all devices have been placed[BERK88, KAYA88, LIN91, CONW92, MEYE93, CHEN89, MOGA89, MEHR91, ONOD92] Routing techniques seek to limit parasitics through controlling parasitic resistance and capacitive coupling [HARA92, SMIT89, CHOU90]. Simulated annealing (SA) approaches use nonlinear SA optimization to place devices, incorporating matching and parasitic rules into the move set and cost function parts of the optimization. Of approaches that do not use templates, the SA approaches have produced the best looking automatically generated layouts [GARR88, COHN91, CHAR94]. Though the absolute value of circuit parasitics cannot be predicted using a SA placement approach, they may be bounded by using a constraints in the SA process [CHAR92]. These methods also have the longer run times, but this is not a significant problem when layout is a stand-alone process.

While general analog place and route tools are oriented toward opamps, there are some module specific tools written for specific structures. CADICS and ADORE [YAGH88,JUSU93] have template based layout tools written specifically for switched capacitor A/D converters and filters. Leme describes another tool for layout of capacitor arrays for CMOS A/D and D/A structures[LEME91]. A module specific tool may not have general application, but solve circuit specific layout and parasitic problems which are not well addressed by tools written with opamp style circuits in mind.

1.2.4 Design Complexity

As example analog circuits have become more complex, the design synthesis problems have become too difficult to attack as a single non-hierarchical problem [JUSU93, CHAN94, DEGR87]. The obvious direction is to cast the design problem into a hierarchical framework, and decompose the high level specifications into subcell specifications, develop sub-module generators and create the subcells to these specifications. Several analog synthesis approaches use a high degree of hierarchical decomposition, using simple circuit elements such as differential pairs and current sources and current mirrors [BERK88, HARJ88]. In a design with significant interaction between the subcells, the decomposition step is difficult. In a knowledge based framework, the top down decomposition again follows a designer recipe, with the limitations to preprogrammed topologies. In an optimization based framework, the process of creating subcell specifications is fraught with peril. To create an "optimal" set of subcell specifications requires knowing the "cost" of any set of subcell specifications. There are two approaches that have been used. In the first case, a subcell optimization is run every time the cost of a set of subcell specs is requested, resulting in full optimizations within optimization [JUSU93]. In the second approach, estimator functions for circuit cost [PADU87] or subcircuit flexibility [CHAN94] replace the subcell optimization, allowing a simple optimization at each step in the hierarchy, but since the estimator functions are not exact, the final solution is not optimal. In both approaches the final solution will meet specifications. The advantage of using a hierarchical approach is that it allows attack of complex problems. The disadvantage is that either simulation time is increased, or the solution not optimal, and natural places to break an architecture into a hierarchy must be found.

1.3 Module Generation vs. General Analog Synthesis.

Module Generation is the application of analog synthesis techniques to a particular class of circuits. It usually includes both the circuit synthesis and layout synthesis methodologies required for the selected module. When implementing a module generator there are usually special issues for the particular module which may not be adequately addressed in a general synthesis methodology. The choice of a particular type of analog circuit strongly affects the subsequent choices for design estimation, selection, and layout. Just as in other areas of analog circuit research, good choices for examples tend to yield research results which will solve real problems. There is a danger that tools developed for a particular analog module will not be useful beyond that application, but if the tools developed for module generation can be kept separate from the design specific information, then they may be applied to similar problems in other circuits.

To date, most general analog synthesis approaches have actually used opamp designs as examples, and this choice has strongly affected the synthesis tools developed. We shall see that these general synthesis approaches do not take the best approach for DAC module synthesis. For example, layout tools have been oriented toward opamp style designs consisting of a handful of large devices. In DAC modules, the design usually consists of a large number of relatively small devices, where matching and routing issues are significant for circuit performance and area. In developing module generation tools for DACs new problems must be solved which have not been addressed by other analog synthesis techniques.

1.4 Digital/Analog Converters (DACs)

Digital/Analog Converters (DACs) convert a signal from a digital representation to an analog signal proportional to the digital input. The output signal may be in the form of voltage, current, or charge. They are one of the most common analog interface circuits, making the connection from a digital system to the analog world around it. There are many application for DACs, ranging from low speed, high resolution audio applications to high speed, low resolution video, and including a broad band of general instrumentation and control applications in between. Figure 1.3 illustrates these applications, with resolution and sample rate as the axes. Of these applications, the low sample rate, high resolution architectures are dominated by oversampling architectures [ADI92, CRYS92,

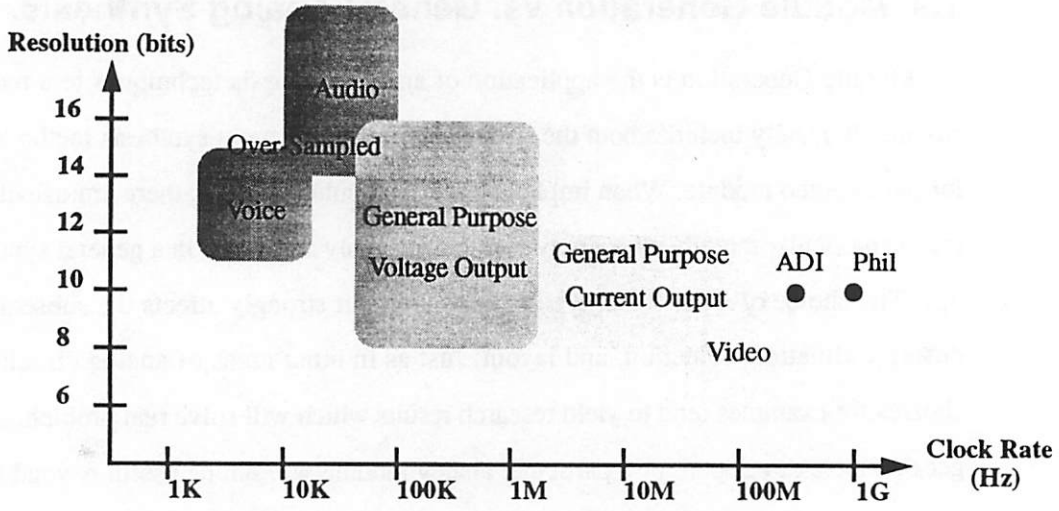


Figure 1.3 Monolithic DAC implementations. Shaded areas represent commercial products. Points are state of the art for high speed CMOS and bipolar. [ADI92, REYN94, VORE94] LERC91, KUP91], due to the inherent linearity and simplicity of the analog circuits in those architectures. Throughout the rest of the range nyquist rate DACs dominate. DACs which are designed to drive an external load typically output either voltage or current. A current output DAC is easily modified to a voltage output configuration through a resistive load. From the figure it is apparent that a current output DAC module generator, capable of resolution to 10 bits, and sample rates to 135 MS/s could have wide application.

1.5 Design Qualities of Nyquist Rate DACS.

There are several DAC implementations found in CMOS circuits [PELG90, YANG89, SHEN83, SCHO88, and many others]. Although these designs have quite different topologies, they have common properties which make their design and analysis processes different from other analog functional blocks, such as Opamps, and A/D Converters. First, they require two levels of analysis -- both a low level circuit analysis, predicting device level bias conditions and mismatch, and high level behavioral analysis of DAC statistical and global effects. Second, they tend to be constructed of a large number of identical subcircuits. Third, these subcircuits tend to be fairly simple, with a handful of devices per cell. Fourth, there is a close coupling between layout and performance. Finally, like A/D converters, they have key design variables which must take on integer, and usually power

of 2, values. These qualities have implications for all parts of the module generation process: design estimation, design selection, and layout.

For design estimation, this means that DC and Transient analyses which require circuit simulation may be done at the full DAC level, using only a few devices to simulate the entire DAC. On the other hand, the interaction of low level and high level circuit characteristics, with layout placement information, means that an accurate high level DAC estimation requires accurate prediction of low level circuit conditions, and prediction of the module layout. For example, when considering second order effects such as supply resistive drops, voltage coefficients, device mismatch, or finite output impedance of circuits, one must first find the bias conditions and subcell layout spacing before the magnitude of these effects may be determined. This implies a need for design estimation at both a circuit level and a DAC system level.

For design selection, the requirement for integer valued results complicates the choice of optimization or selection algorithm.

The layout problem for DACs also different from that seen in other analog applications. To obtain the best possible device matching, it is important to maintain identical geometries and identical spacing across the circuit[PELG89, SCHO88]. In order to keep circuit area down the DAC cell must be tightly packed, because wasted space is multiplied by the number of cells. There is a large amount of digital circuitry in a DAC, as well as output signals which may have a large swing and fast rise time. It is critical that these large signals not couple to sensitive bias signals through parasitic wiring capacitances in the layout. As noted above, cell layout area must be predicted as a function of design inputs to accurately estimate high level parameters.

1.6 Module generation for DACs.

In the preceding sections the requirements for usable module generation have been discussed, as well as the some important aspects of DAC generation. This leads to the question "For DAC module generation, what approaches can work?"

The method requires accurate estimation. At the circuit level, this means using SPICE circuit simulation for DC and transient analysis. (AWE is not suitable, because it does not incorporate

transient analysis.) Since there are high level layout and statistical effects, these are best computed using either behavioral or analytic approaches. The analysis for DAC performance is relatively easy, so a design equations approach is acceptable.

For design selection, a constrained, nonlinear optimization approach gives the possibility of reaching an optimal solution. The simulated annealing approach is not reasonable, since transient SPICE simulations must be done within the optimization loop. The optimization must find integer solutions, in the context of long design estimation step. None of the previous work in gradient based constrained optimization suggests a solution to this problem which will complete in a reasonable time.

For layout, previously described layout techniques just do not match the problem. Although the matching and parasitic capacitance issues have been dealt with in general analog placement algorithms, the stronger area minimization and predictable circuit area requirements are not met by these tools.

The last question is that of a hierarchical approach. There is a natural break up of device level and DAC level optimizations in these topologies[CHAN94], but using a hierarchical approach has its drawbacks if an optimum solution is desired. The method of optimization within optimization will give an optimum, but it is impractical for DAC synthesis, because the inner loop optimization is a SPICE based optimization which will explode the problem. If estimates of cost functions are used for the high level optimization, then errors in the cost functions may give non-optimal results. By avoiding hierarchy, and including all relevant design variables, an optimal solution may be found, but in this case design estimation must incorporate circuit and DAC analyses simultaneously.

1.7 Previous work

Two DAC synthesis tools have been described in the literature. Allen and Barton implemented DAC functions in a standard cell based silicon compiler. This approach compiled modules using procedural techniques [ALLE86]. ARDAC [CHAN94] synthesized current output CMOS D/A converters as a demonstration of a top-down constraint driven analog design methodology. Specifica-

tions were limited to important static specifications, and the synthesis method used a hierarchical approach which used flexibility functions. Analysis used DC SPICE simulations at the device level, and behavioral simulation at the DAC level. Layout used custom layout generators to create three subcircuits. Test chips were fabricated to verify performance. Dynamic performance was not included in the specifications, and some important design variables were determined a priori, rather than used in synthesis, limiting the optimality of the solution.

1.8 Thesis Contribution

This thesis describes DSYN, a module generator for the synthesis of CMOS Digital/Analog converters. In the course of this development there are several research contributions.

- Development of an accurate DAC performance estimation method, using a combination of circuit simulation and DAC architecture analysis.
- Implementation of an optimization algorithm for constrained, mixed integer optimization, adapted to this application.
- Application of cell stretching and tiling techniques, commonly seen in digital macro-cell layout synthesis, for area efficient, predictable DAC circuit layout synthesis.
- Incorporation of these steps together into a DAC module generator, considering important real world issues such as performance sensitivities to layout parasitics, device model inaccuracies, and process variation.
- Verification of this synthesis methodology through high performance prototypes. A fabricated test circuit has demonstrated synthesis to commercial specifications for video DAC applications. Other test circuits have been synthesized with higher resolution or different technology specifications.

1.9 Thesis Organization

Chapter 2 describes DAC architectures and implementations in general, and discusses the motivations which lead to the particular high speed current source CMOS DAC architecture chosen for this work.

Chapter 3 introduces DSYN, the DAC module generator program. It outlines the synthesis process, inputs and outputs required for the process, and the algorithms used for estimation, design selection, and layout in general terms. Verification issues for synthesis inputs and the synthesis process are considered.

In chapter 4 both analysis and design selection processes are covered, because these two processes are tightly linked. This chapter includes a review of important approaches, and illustrates how good choices for both analysis and optimization can lead to a circuit synthesis process which meets the requirements described in this chapter.

Chapter 5 discusses the layout process implemented for DSYN. A review of aspects of DAC design motivates the choice of a template based approach for these DACs.

Chapter 6 is a review of the results obtained with DSYN. Two prototype DACs have been built in a 1.2 micron technology, and the performance predicted from estimation and extracted spice simulations is compared with fabricated results. Designs implemented in other technologies demonstrate the reuse of the design across technology.

Chapter 7 is the concluding chapter for this thesis, summarizing results, and discussing the application of these to an industrial environment. The appendix describes the implementation details for the DAC optimization, simulation, and layout libraries.

The appendix summarizes the software tools and design libraries developed in the course of this work, and is users' manual for succeeding tool users.

CHAPTER 2

DAC Architectures for Synthesis

2.1 Introduction

This chapter is a discussion of the issues important for choosing a DAC architecture for module generation, and ultimately describes the choice made for DSYN. First the DAC designs are described, including their specifications, architectural choices, and implementation choices. A set of desirable properties for a module generator is discussed, and a DAC architecture and implementation choice is made based on those. That architecture is then described in some detail, including design inputs that may be used as variables during synthesis, performance estimates which may be used as constraints, and some information about typical layout styles for this architecture. The chapter closes with a brief summary.

2.2 D/A Converter Specifications and Design Inputs

2.2.1 Specifications

As with any analog circuit, there are two classes of D/A converter specifications, performance specifications and operating environment specifications. For DACs the performance specifications may be further divided into static and dynamic specifications. Static specifications define how accurately the output matches the input digital code under DC conditions, and describe the worst

case matching of output signal levels to an ideal DAC response. Dynamic specifications describe how the DAC responds to transitions between levels, and to digital switching at the chip level. Environmental specifications describe the conditions under which the circuit must operate, including power supply voltage range, circuit loading, temperature conditions, and compliance of the output signal.

The two most important static specifications are Integral Non-Linearity (INL) and Differential Non-Linearity (DNL). INL is the difference of any point on the transfer function from an ideal transfer function drawn between the maximum and minimum outputs. DNL is the error in the difference between adjacent DAC levels. Both are expressed in units of a Least Significant Bit (LSB). It turns out that the INL at any DAC level is the sum of DNL up to that level. These errors can be written algebraically as follows, where N is the number of levels, and A_{FS} is the full scale signal, and $A_o(i)$ is the DAC value for input i. Δa is defined as the ideal step between levels. DAC output levels are assumed to be unipolar, from 0 to A_{FS}. Then:

$$\Delta a = \frac{A_{FS}}{N} \tag{2.1}$$

$$DNL(0) = 0 \tag{2.2}$$

$$DNL(i) = \frac{A_o(i) - A_o(i-1)}{\Delta a} - 1 \tag{2.3}$$

$$INL(i) = \frac{A_o(i)}{\Delta a} - 1 \tag{2.4}$$

$$INL(i) = \sum_{1}^{i} DNL(i) \tag{2.5}$$

There are two other specifications based on the static transfer characteristic. Gain Error (GE) is the error in the full scale output relative to the ideal. Absolute accuracy, or Total Unadjusted Error (TUE) is the maximum deviation from the ideal transfer function for any code. GE and TUE may be expressed in LSB, or as a percent of full scale. Static transfer function errors are summarized in Fig. 2.4.

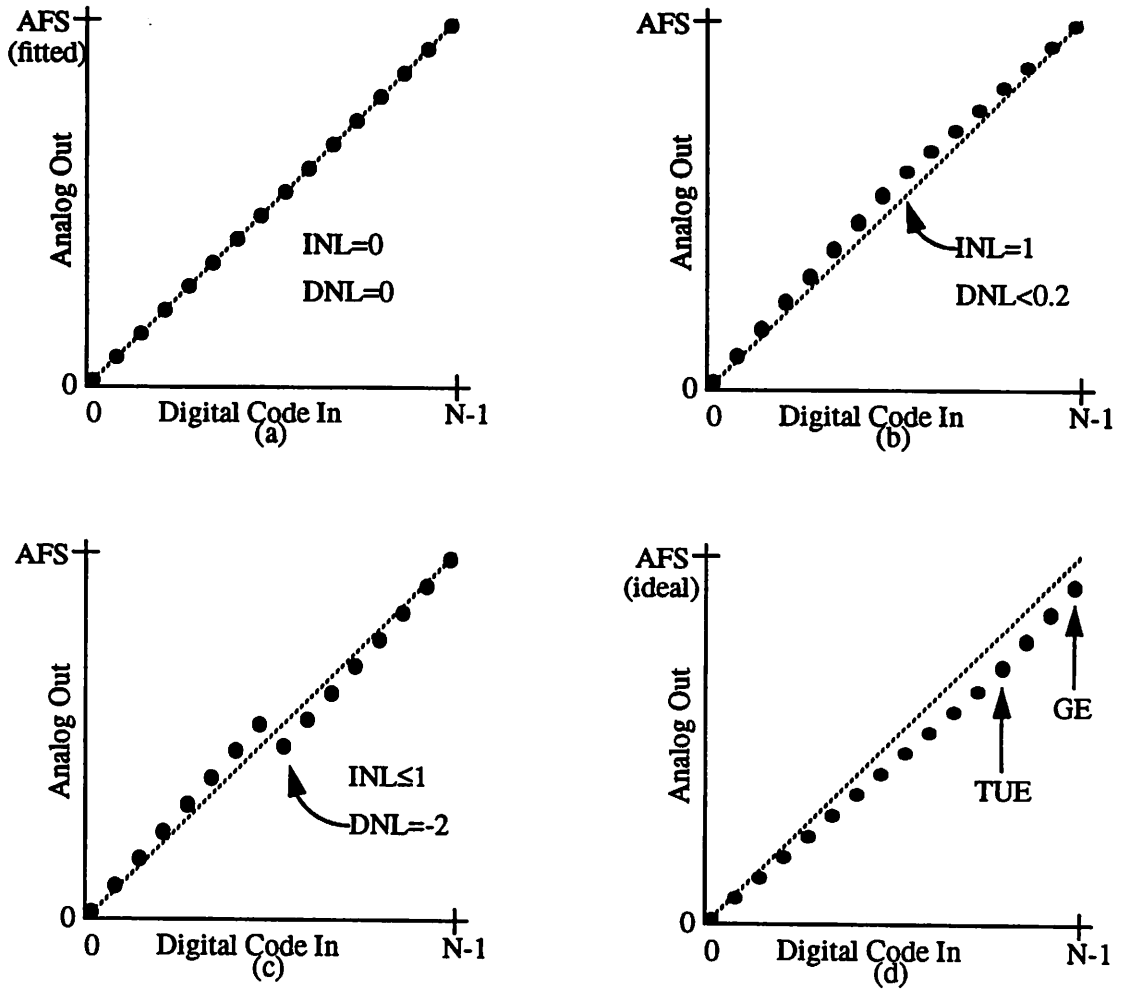


Figure 2.4 Static Errors in D/A Converters. a) Ideal DAC transfer function. b) Transfer function with systematic non-linearity. b) Transfer function with step non-linearity at a major transition. d) Transfer function with GE and TUE measured with respect to ideal transfer curve.

Dynamic specifications describe the performance of the circuit at speed. Delay Time (TD) is the time delay from a clock signal to the 50% point of the output signal transition. Settling Time (TS) is the time from that 50% point to settling to within some delta of the final value (the value of delta is usually 1/2 LSB, but this is not standardized). Switching time (TSWIT) is the time required to go from 10% to 90% of the transition. Glitch Energy (Glitch) is the worst case spurious energy produced for a 1 LSB transition. These specifications are measured using the worst case transition, so for the switching and settling specifications full scale transitions are considered worst, while for glitch energy the worst case is usually the transition at the major carry at half of full scale. These

specifications must be met for all load conditions. Fig. 2.5 summarizes these dynamic performance specifications.

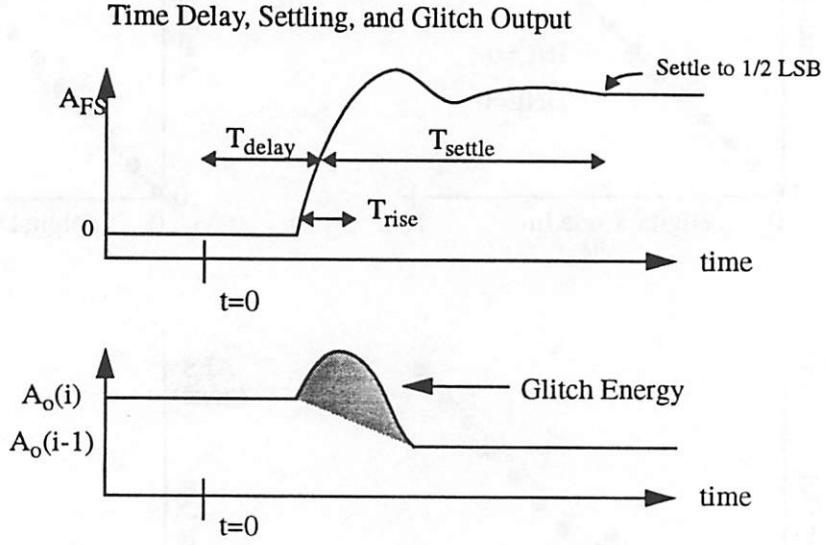


Figure 2.5 Dynamic Errors in D/A Converters. Time Delay, Switching Time, Settling Time, and Glitch Energy. Clock at t=0.

Other specifications cover the operating environment, including supply voltages, operating temperatures, and conditions for the output signal. For a voltage output DAC, the impedance of the load should be specified. For a current output DAC, there is a compliance range of voltages over which the output current linearity will be in spec. The nominal output range should be specified, and there may be specifications for any external bias signals as well. Table 2.1 lists a set of specifications for a current output DAC [ADI92]

2.2.2 Integrated Circuit Technology

The technology to be used by a designer is another important input. The IC processing technology must be characterized, and technology related information which is important to the design must be quantified. This includes nominal device models, design rules, interconnect capacitance parasitics, as well as important second order effects. For digital designs the second order process variation effects are usually considered, but for a DAC, the designer must also know the random

Table 2.1 Specifications for ADV 7120

Specifications	Value	Units	Comment
STATIC PERFORMANCE			
Resolution	8	bits	
INL	± 1	LSB max	
DNL	± 0.5	LSB max	Guaranteed Monotonic.
Gain Error	± 1	% of full scale	
DYNAMIC PERFORMANCE			
Glitch Energy	50	pV secs typ.	
Switching Time	3	ns max	
Settling Time	12	ns max	
Delay Time	20	ns typ	
OTHER SPECIFICATIONS			
f_{max}	80	MHz	Clock Rate
I_{fs} (Full Scale Current)	15	mA min	
	22	mA max	
R_{out} (Output Impedance)	100	kΩ typ	
C_{out} (Output Capacitance)	30	pF max	I_{OUT} = 0 mA
V_{OC} (Output Compliance)	-1	V min	
	+1.4	V max	
V_{AA} (Power Supply)	5	V nom	+/- 5%
PSRR	0.5	%/%	f = 1 kHz

mismatch of devices, and the discrepancies between the fitted device models and actual device I-V curves, particularly if the design uses short channel devices in the weak inversion region [TSIV94].

2.3 DAC Architectures and Implementations

In DAC design, there are two orthogonal parts to the design. DAC architecture describes how circuit elements will be assembled to create the DAC output, but does not necessarily imply a particular circuit design. In this discussion, the DAC implementation describes the set of low level elements which will be assembled, but does not imply a particular architecture. Any combination

of architecture and implementation is possible, and it is difficult to find a combination that has not been tried. In some cases hybrid approaches which use a combination of circuit implementations for different parts of a DAC architecture are found.

2.3.1 Architectures

For nyquist rate DACs, the choice of DAC implementation affects all performance measures, while the architecture choice particularly effects the DNL and Glitch Energy performance. In the following sections each of the potential architectures will be discussed, with first order predictions of INL, DNL, and Glitch Energy performance using due to random element mismatch and timing mismatches [GRAY90]. The oversampled architecture will be discussed briefly, though it is difficult to compare within the same framework. The switched current source implementation will be used as an example to illustrate these architectures for an example 6 bit DAC implementation

2.3.1.1 Unit Element Switching Architecture

A Unit Element Switching architecture consists of a set of equal elements which are added to the output one at a time as the DAC input is increased. For each DAC transition corresponds to a particular unit element. Fig. 2.6 illustrates this architecture for a current source example, sometimes referred to as a "sea of current sources." Another common implementation is as a tapped resistor string.

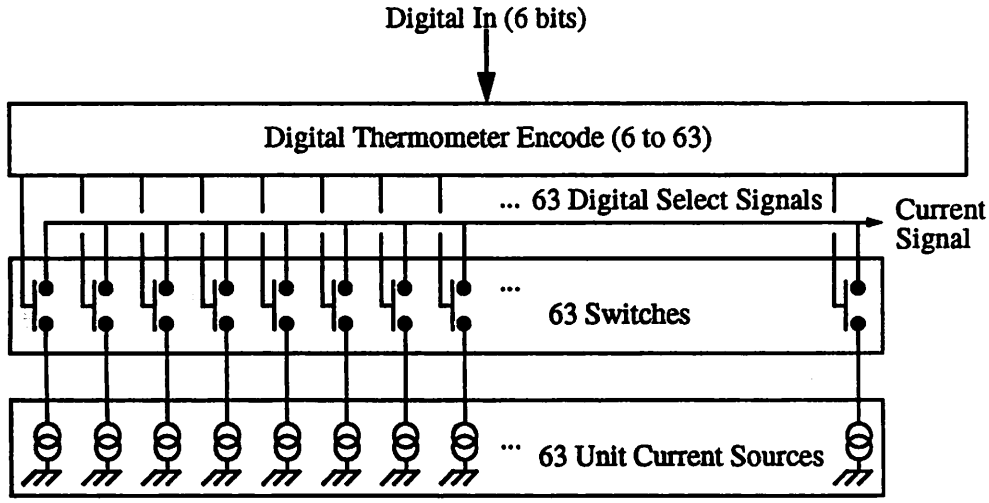


Figure 2.6 Unit Element DAC in a 6 bit switched current source implementation.

Before predicting INL, DNL, and Glitch Energy, a few terms must be defined:

- a_i = i^{th} analog element value. a_i is a random variable, mean Δa, variance $\sigma_{\Delta a}^2$.
- A_{FS} = Full Scale Value

$$A_{FS} = \sum_{i=0}^{N-1} a_i + \Delta a \tag{2.6}$$

- $A_o(n)$ = Output at n^{th} output level:

$$A_o(n) = \sum_{i=0}^{n} a_i \tag{2.7}$$

- Normalized output relative to AFS:

$$\frac{A_o(n)}{A_{FS}} = \frac{\sum_{i=0}^{n} a_i}{\sum_{i=0}^{N-1} a_i + \Delta} \tag{2.8}$$

- Ideal output ($a_i = \Delta a$, $\sigma_{\Delta a}=0$):

$$\frac{A_o(n)}{A_{FS}} = \frac{n \cdot \Delta a}{(N-1)+\Delta a} = \frac{n}{N} \tag{2.9}$$

Assume that a_i are independent, identically distributed (iid), then compute variance for INL and DNL at the worst case points. For INL, the worst point is at the midpoint, since by definition INL = 0 at the endpoints, and the function is symmetric. INL is found by computing the variance of the signal at the midpoint (n=N/2), subtracting from the zero error value, and converting units to LSBs. Assume the variation in a_i, δ_{ai} is a small compared to Δa and substitute $\Delta a + \delta_{ai}$ for a_i in the following:

$$\frac{A_o(N/2-1)}{A_{FS}} = \frac{\sum_{i=0}^{N/2-1} a_i}{\sum_{i=0}^{N/2-1} a_i + \sum_{i=N/2}^{N-1} a_i + \Delta a} \approx \frac{N/2}{N} + \frac{\delta A(N/2-1)}{A_{FS}} \tag{2.10}$$

Contributions to $\delta A(N/2-1)$ consist of slightly different terms for $i<N/2$ and $i>N/2$. Cross products in δa_i may be ignored at this point because they are small. The factor of 1/2 comes about

because for i<N/2, δa_i is partially cancelled by a denominator term, and for i>N/2, the δa_i error only contributes through the denominator, with a factor of 1/2.

For i<N/2, each a_i contributes:

$$\frac{A_0(N/2)_i(i<N/2)}{A_{FS}} = \frac{(N/2)\cdot\Delta a+\delta a_i}{N\cdot\Delta a+\delta a_i} \approx \frac{1}{2}+\frac{1}{2}\cdot\frac{1}{N}\cdot\left(\frac{\delta a_i}{\Delta a}\right) \tag{2.11}$$

For i>N/2, each a_i contributes:

$$\frac{A_0(N/2)_i(i>N/2)}{A_{FS}} = \frac{(N/2)\cdot\Delta a}{N\cdot\Delta a+\delta a_i} \approx \frac{1}{2}-\frac{1}{2}\cdot\frac{1}{N}\cdot\left(\frac{\delta a_i}{\Delta a}\right) \tag{2.12}$$

When these are summed over all i, and the error term is separated, the result is:

$$\frac{\delta A(N/2)}{A_{FS}} = \frac{1}{2N}\sum_{i=0}^{N/2-1}\left(\frac{\delta a_i}{\Delta a}\right)-\frac{1}{2N}\sum_{i=N/2}^{N-1}\left(\frac{\delta a_i}{\Delta a}\right) \tag{2.13}$$

Since these δa_i variables are iid, the variance of this is the weighted sum of the variances, and recall that INL is just this δA error. Since δA has mean 0, VAR(δA) can express:

$$VAR(INL(N/2)) = E\left[\left(\frac{\delta A(N/2)}{A_{FS}}\right)^2\right] = \frac{(\sigma_{A(N/2)})^2}{(A_{FS})^2} = \frac{1}{4N}\left(\frac{(\sigma_a)^2}{\Delta a^2}\right) \tag{2.14}$$

Usually the INL is expressed as the standard deviation, and converted to units of LSB, and the computation is assumed to be made at the midpoint. This gives:

$$\sigma_{INL} = \frac{\sqrt{N}}{2}\cdot\frac{\sigma_a}{\Delta a} \tag{2.15}$$

For a unit element DAC, DNL and Glitch are easy to compute. Because only one element switches at a time, the DNL variance at any step is the same as the variance for the unit element.

$$\sigma_{DNL} = \frac{\sigma_a}{\Delta a} \tag{2.16}$$

Fig. 2.7 shows typical INL and DNL plots for a unit element DAC.

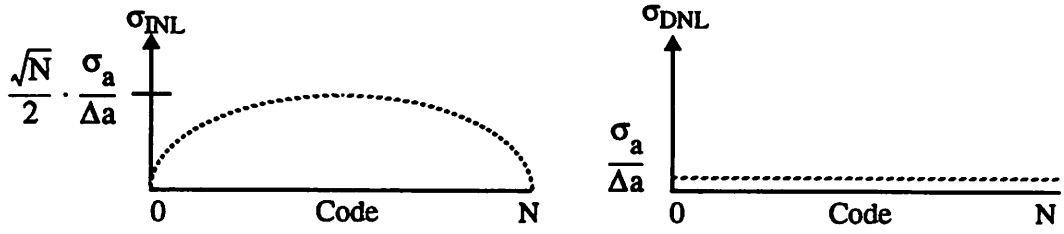


Figure 2.7 Qualitative INL and DNL curves for a unit element DAC.

In this discussion the model for Glitch Energy is that it is caused by charge injection and timing mismatches between elements turning off and other element turning on. There may be other contributions to glitch, dependent on the implementation. With these assumptions, the glitch energy for a unit element DAC is zero, because for any single level transition there is only one element switching.

$$\text{Glitch Energy} = 0 \tag{2.17}$$

To summarize, unit element architectures give excellent DNL and glitch energy performance, at the expense of a relatively complex digital encode circuit.

2.3.1.2 Binary Weighted Architecture

A Binary Weighted architecture uses binary weighted elements, and sums these according to the digital input. A 6-bit binary weighted current source architecture is illustrated in Fig. 2.8. The most significant bit controls a weight equal to half of full scale, the next bit half as much, down to the lsb, which controls one unit. Typical implementations include weighted current source and weighted capacitor arrays. In bipolar implementations thin film resistors in R-2R ladder structures are commonly used to create binary weighted currents.

To compute INL and DNL for this architecture, use the same assumptions about element random variation as in the unit element DAC case, but assume that the weighted elements consist of groups of identical unit devices. Then for INL, the computation is identical to the unit element case, with the worst case INL at the middle of the range.

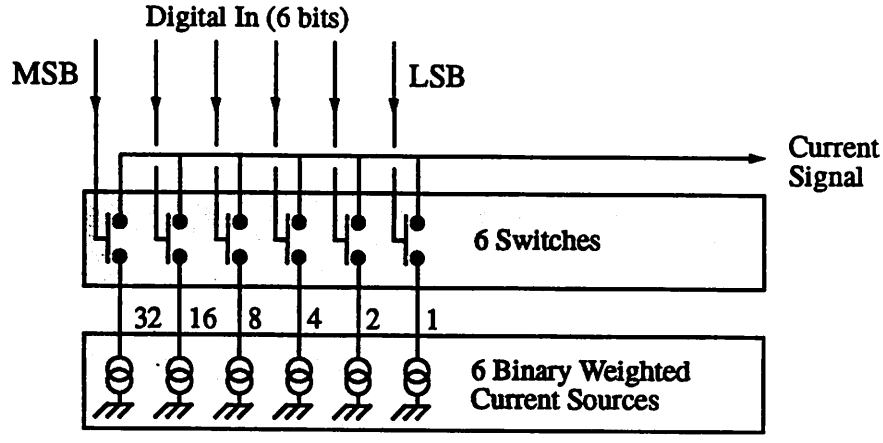


Figure 2.8 Binary Weighted DAC in a 6-bit switched current source implementation.

$$\sigma_{INL} = \frac{\sqrt{N}}{2} \cdot \frac{\sigma_a}{\Delta a} \tag{2.18}$$

For DNL, the situation is much worse. DNL occurs when one set weights is turning off, while another is turning on at a code transition. The worst case is at mid scale, going from code 011111 to 100000 in a 6-bit example. At that step the DNL is:

$$DNL(N/2) = A_0(N/2) - A_0(N/2-1) = \sum_{i=N/2}^{N-1} a_i - \sum_{i=0}^{N/2-1} a_i \tag{2.19}$$

This is similar to the INL expression, but without the denominator term, making DNL a factor of 2 worst than the INL expression. At the midpoint the standard deviation of DNL is:

$$\sigma_{DNL}(N/2) = \sqrt{N} \cdot \frac{\sigma_a}{\Delta a} \tag{2.20}$$

Other major carry code transitions also have high DNL, proportional to the number of elements being switched on and off. Fig. 2.7 shows typical standard deviations due to random mismatch for INL and DNL in a binary weighted DAC. Note that worst case DNL is twice the worst case INL.

A first order estimate of Glitch Energy can be computed as the number of elements switching, multiplied by the timing mismatch between elements switching off and elements switching on.

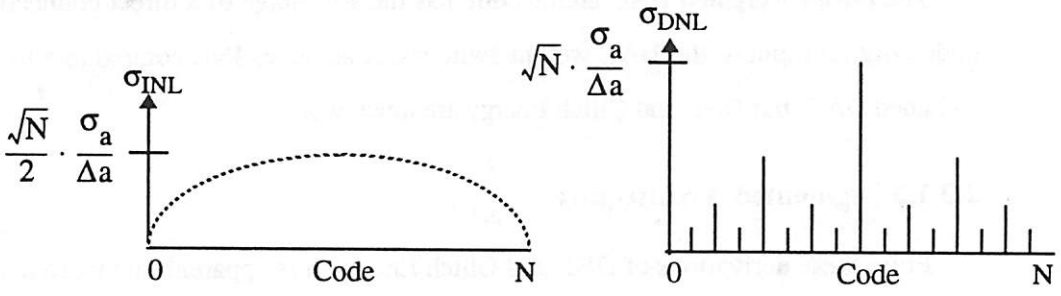


Figure 2.9 Qualitative INL and DNL curves for a binary weighted DAC.

The worst case glitch energy occurs at the major transition, when N/2 devices switch on, and N/2 - 1 switch off. Graphically this is shown in Fig. 2.7. Mathematically, if the timing difference is

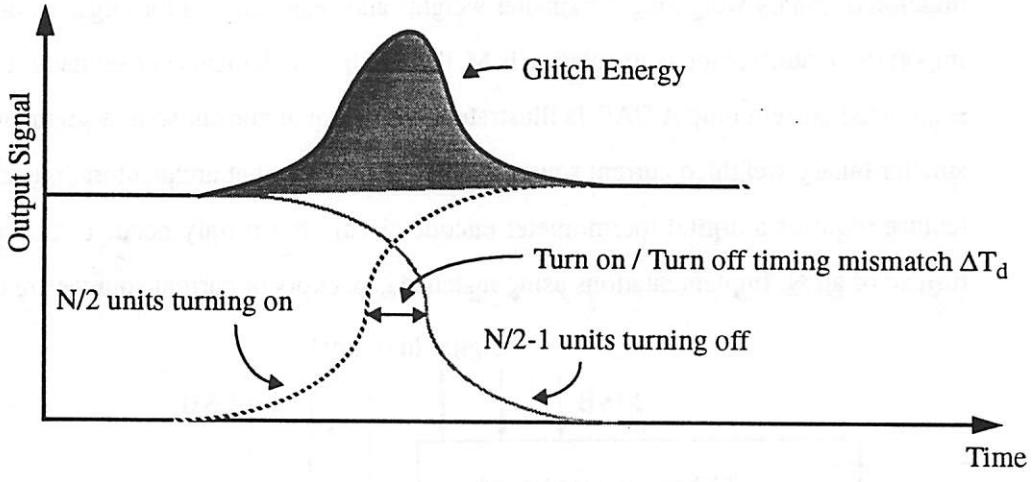


Figure 2.10 Timing mismatches lead to glitch energy. Solid line is output signal, and area of the shaded region is the glitch energy.

expressed as ΔT_d, and the number of elements switching is N_s, Glitch Energy is expressed as:

$$\Delta T_d = T_d(\text{rise}) - T_d(\text{fall}) \tag{2.21}$$

$$\text{Glitch Energy} = \Delta T_d \cdot N_s \tag{2.22}$$

For a binary weighted DAC, the worst case is at mid-range, with the result:

$$\text{Glitch Energy} = \Delta T_d \cdot (N/2 - 1) \tag{2.23}$$

The binary weighted DAC architecture has the advantage of a direct connection of a binary coded digital input to the DAC weight switches. It achieves INL comparable to a unit element switched DAC, but DNL and Glitch Energy are much worse.

2.3.1.3 Segmented Architecture

From these derivations of DNL and Glitch Energy, it is apparent that these non-idealities are caused by the large number of devices switched simultaneously at major transitions. When the maximum number of switched devices is reduced, then the DNL and Glitch errors will be reduced also. The segmented architecture is a compromise between unit element switched and the binary weighted architectures, which partially splits up the DAC into equally weighted segments. A combination of binary weighting for smaller weights and segmentation for larger weights is used. The important parameter for segmentation is M, the number of elements per segment. In Fig. 2.8 a 6 bit segmented current output DAC is illustrated, with 7 equal current source segments (M=8), and 3 smaller binary weighted current sources. Like the unit element architecture, the segmented architecture requires a digital thermometer encode circuit, but it only needs to encode N/M signals instead of all N. Implementations using matched capacitors or current sources are common.

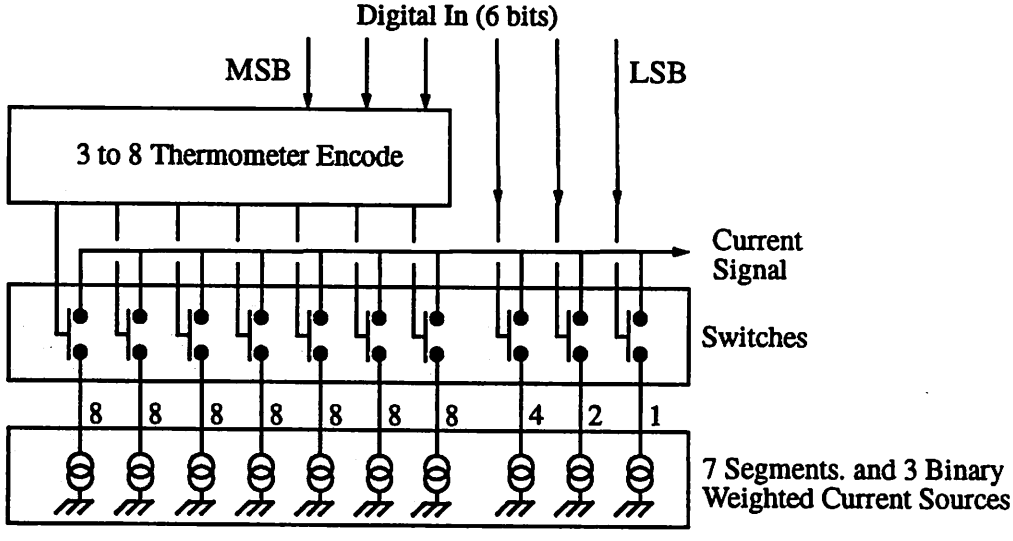


Figure 2.11 Segmented DAC in a 6-bit switched current source implementation with M=8.

Looking at the INL, DNL, and Glitch energy, we first see that the INL computation does not change as a function of segmentation. Once again, INL does not depend on the order that elements are chosen in, and only on the number of elements and their variance. The expression for σ_{INL} is the same as in Eq. 2.18.

For DNL and glitch the number of elements being switched in the worst case transition is important, and for a segmented architecture, this is M elements switched, instead of N/2 found in the binary weighted case. DNL and glitch energy performance may be computed by simple substitution of M for N/2 in Eq. 2.20 and Eq. 2.23. Qualitative plots of σ_{INL} and σ_{DNL} for a segmented architecture are in Fig. 2.7.

$$\sigma_{DNL}\,(\text{EachSegment}) = \sqrt{2M} \cdot \frac{\sigma_a}{\Delta a} \tag{2.24}$$

$$\text{Glitch Energy} = \Delta T_d \cdot (M - 1) \tag{2.25}$$

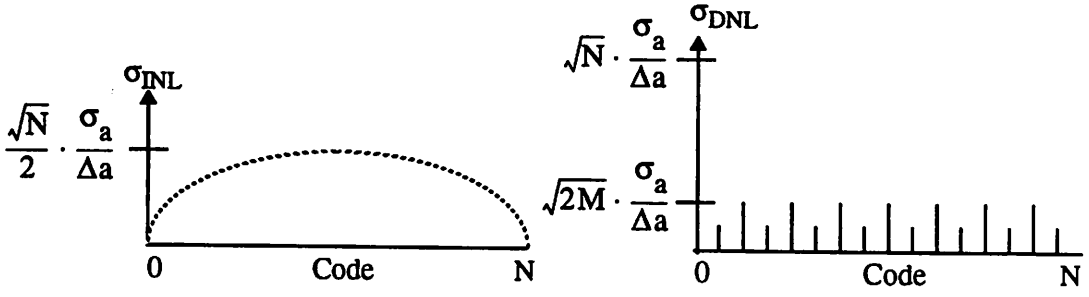


Figure 2.12 Qualitative INL and DNL curves for a segmented DAC, N/M=8.

It can be seen that DNL and Glitch Energy can be arbitrarily reduced by decreasing M, the size of the segments. When M=N/4, worst case INL and DNL due to random effects are the same. In low speed applications, where glitch energy is not critical, M = N/4 or N/8 is common. In high speed applications, where glitch energy is to be minimized, M=8, M=4, or unit element(M=1) architectures are seen [MIKI92, LETH87, SHEN85].

2.3.1.4 Interpolated Architecture

In both of the previous cases the worst case DNL occurs at the major transitions, where elements being turned off must match others being turned on. The interpolated architecture attacks

this problem by never turning off elements as the digital code is increased. In a current switched implementation this is done by starting with a set segments which cover the DAC range, and allowing a three way selection (to output, interpolator, or dump), based on the code in the MSBs of the input. The segment may be switched on, off, or sent to a separate sub-divider circuit. The sub-divider is controlled by the LSBs, and passes a portion of the next segment to be selected to the output. As the digital code is increased, the sub-divider increases the amount of a segment that is passed to the output, until a segment transition is reached, when the current segment is re-directed to the output, and the next segment is directed to the sub-divider. This architecture is illustrated in

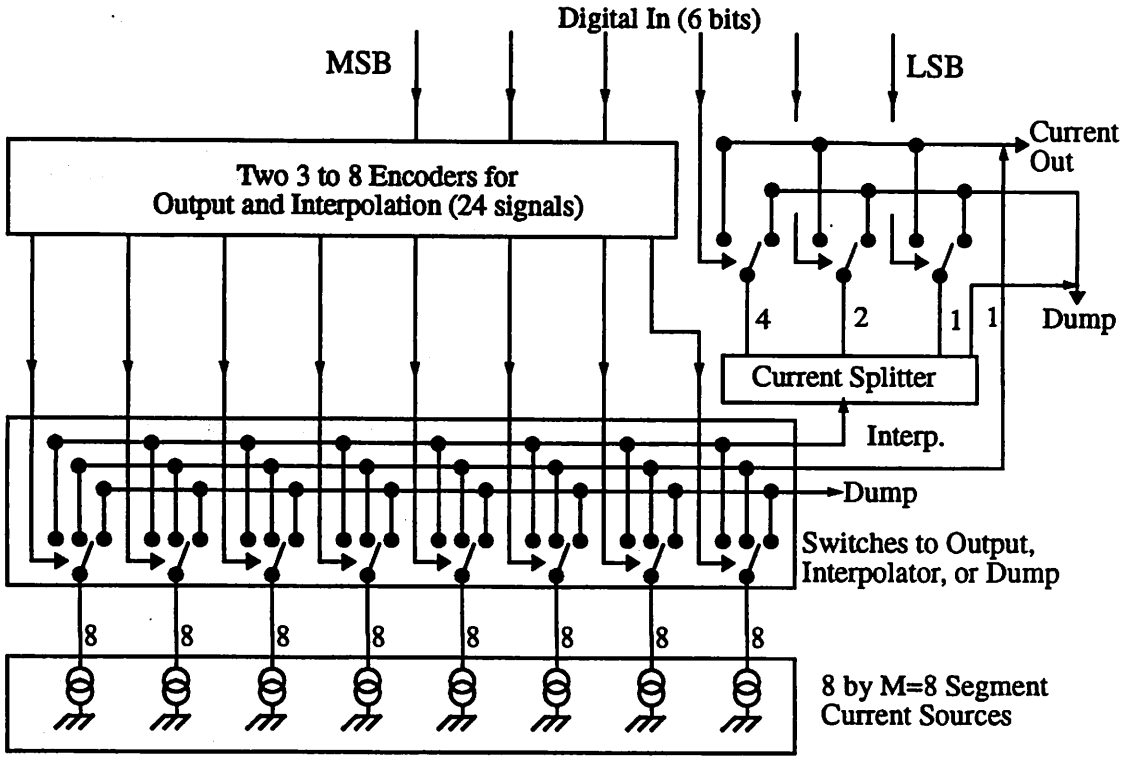


Figure 2.13 Segmented DAC in a 6-bit switched current source implementation with M=8.

Fig. 2.8[SCHO88]. Other implementations using resistor/capacitor structures to create segment interpolation have also been used. INL, DNL, and glitch energy performance for interpolated architectures are implementation dependent, so in this section a more qualitative discussion is given than in the previous two.

In the interpolated architecture, INL is dominated by mismatches in the main segments. If non-linearity in the interpolation function can be ignored, then the INL due to the main segments may be computed, for S segments, with variance σ_s^2, and M LSBs per segment:

$$\sigma_{INL} = \frac{\sqrt{S}}{2} \cdot \frac{\sigma_s}{M \cdot \Delta a} \tag{2.26}$$

DNL tends to be dominated by gain errors and device mismatches in the subdivider. Device mismatch may especially be a problem if the subdivider is implemented in a binary weighted fashion. If the subdivider is implemented as a unit element architecture, and gain errors are negligible, DNL variance should be the only the variance of one subdivider element.

The complicated switching scheme in an interpolated architecture makes Glitch Energy difficult to estimate without knowing the implementation.

Interpolated architectures are often found high dynamic range and control applications, where DNL is the most important specification, but absolute accuracy and linearity is not important.

2.3.1.5 Oversampled Architecture

Oversampled DAC architectures use time division instead of current or voltage division to create analog signals proportional to a digital input. This architecture uses digital filtering and signal processing to create a bit-stream which is passed to a 1-bit D/A. The signal from the D/A is analog low pass filtered, removing high frequency components of the modulated signal, and leaving a low frequency output. The advantage of this architecture is that a 1-bit D/A is inherently linear, and has zero INL and DNL. This architecture has been successfully used for a number of audio frequency designs, where a low frequency, high resolution baseband signal is required[KUP91, GROE89, LERC91]. Fig. 2.8 illustrates this technique.

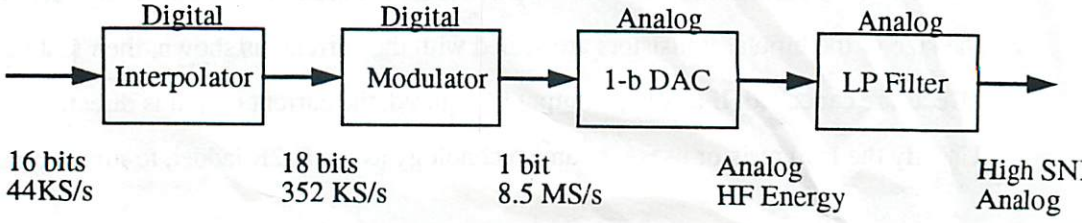


Figure 2.14 Block Diagram for an Oversampled DAC.

2.3.2 DAC Implementations

Creating a set of matched elements, and using them to create an output signal are the most important elements for any DAC implementation. There are some general rules to follow when building a set of matched devices in any circuit technology. In this section the implementation approaches and their compatibility with CMOS processing will be discussed. To start this discussion, a brief description of layout issues for matching is needed.

To create matched elements in an IC technology, there are a handful of layout rules that must be followed, due to random device mismatches, edge effects, process and temperature gradients, and orientation sensitive processing. Matched devices should be placed with identical orientation and dimensions. Large devices may be broken into sub-elements, and placed in a common centroid with other devices. When ratioed devices are needed, integer ratios are best, and devices should be built up of unit elements. When this cannot be done, ratios of perimeter to area should be matched for all elements. In many cases the space surrounding matched elements is filled with identical dummy elements to eliminate edge of the array effects. [McCR81, SHYU84, LAKS86, PELG89, NAKA91, BAST91] Early DAC implementations in bipolar and hybrid technologies depended upon precision, thin film resistors to implement matched devices, and these are still found in many products today. These resistors are typically configured in an R-2R ladder, which creates binary weighted currents if a constant voltage is applied across the ladder. The values of resistance required are only R and 2*R, so a small set of matched R valued resistors can create this ladder. Laser trimming may be used to improve the matching of the resistors to 12 bits or more. Equal weighted segment currents are created with single R valued resistors.

To create an output signal from an R-2R ladder, the output nodes of the ladder must be held at constant voltage, and the current switched to the DAC output dependent on the code. Fig. 2.15 shows a set of current sources with bipolar transistors used to hold the voltage across the ladder. If the size of the bipolar transistors are scaled with the current, as shown, then first order I_b and V_{be} effects are cancelled. If a voltage output is required, the current output is directed to a load resistor. Usually the load resistor uses the same technology as the R-2R ladder, to match process variation.

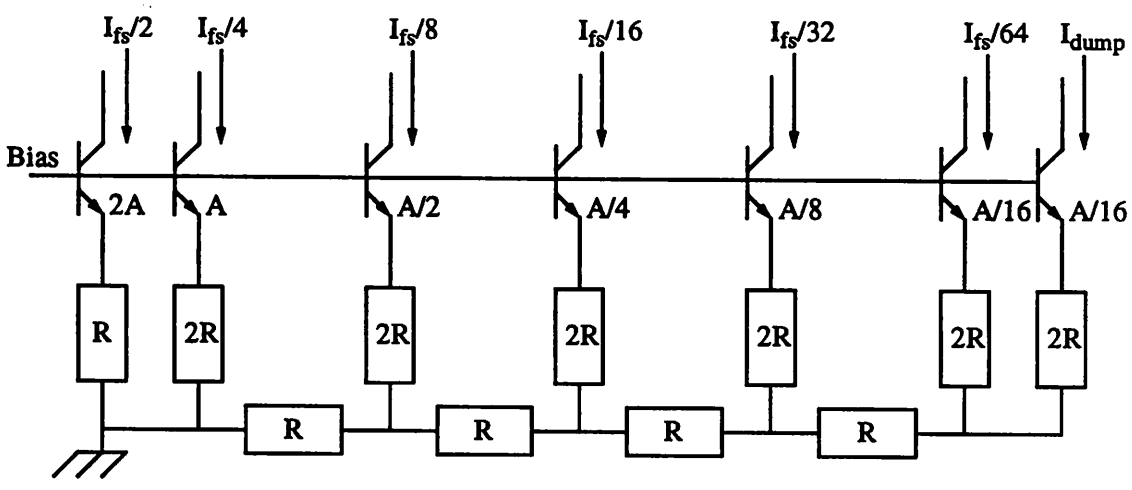


Figure 2.15 6-bit R-2R current source structure creates binary weighted currents.

Though this technology is not found in typical digital CMOS fabs, it is widely available in bipolar technologies, and some specialized CMOS analog processes [ADI92]. When available, MOS devices are ideal for the current steering function, because they have no gate current. The general R-2R structure has also seen use as a current divider using MOS switches instead of resistors. [BULT92]

In technologies which do not have precision resistors, a set of DAC elements may be created from matched current sources. Both bipolar and MOS current source arrays have been used this way. Fig. 2.15 illustrates a set of cascoded MOS current sources for a 6-bit DAC. All weighted

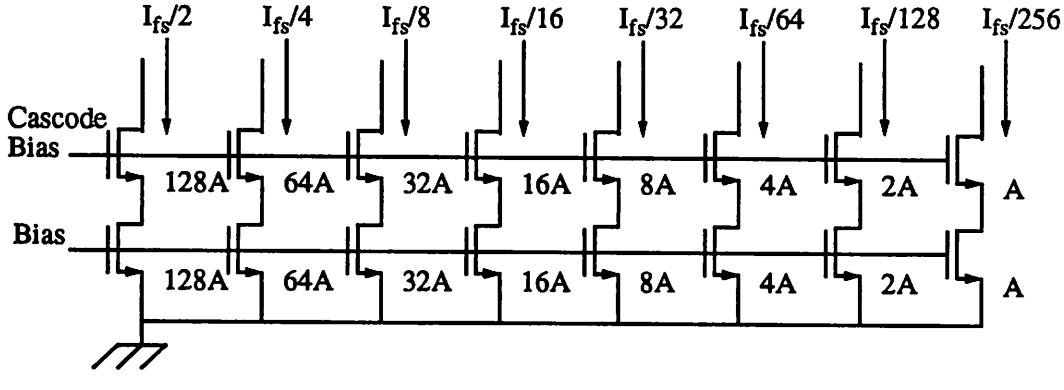


Figure 2.16 8-bit weighted current source structure using cascoded MOS transistors.

current sources are implemented using multiple unit current sources. Note that the MSB source is 128 times as large as the LSB source.

In some implementations the large ratio required to obtain binary weights is costly in circuit area, and may make matching difficult. A coarse/fine approach, making use of a current divider, may reduce the range of required element weights [SAUL84, SCHO88]. Fig. 2.15 implements another

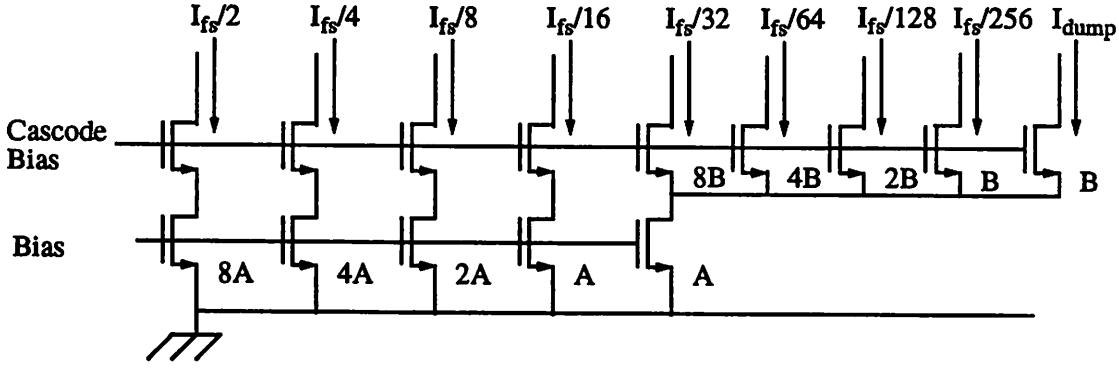


Figure 2.17 8-bit weighted current source structure using a current divider.

set of weights for an 8 bit DAC, using a current divider. In this circuit The 4 MSB sources are created using multiple unit current sources, while the 4 LSB sources subdivide a one MSB source. This reduces the number of main unit currents from 256 to 16.

Current output implementations have seen wide application across resolution and speed. When combined with interpolation or oversampling, these have been used for high resolution audio, and when highly segmented these are used for high speed CMOS and bipolar video and signal synthesis applications [REYN94, VORE94, SCHO88, SCHO91].

The next two implementations are commonly found only in MOS circuits, because they use MOS devices as switches, or take advantage of the infinite gate impedance property of MOS transistors

Tapped resistor structures use a long chain of resistors which is tapped by MOS switches. The ends of the resistor are tied to reference voltages, and by choosing which MOS switches to acti-

vate, the correct voltage is tied to the output. The circuit implementation often uses a diffusion identical to the MOS transistor source for the resistor, eliminating the explicit connection between the resistor and the pass gates. This may yield a very compact layout. Fig. 2.18 illustrates a tapped

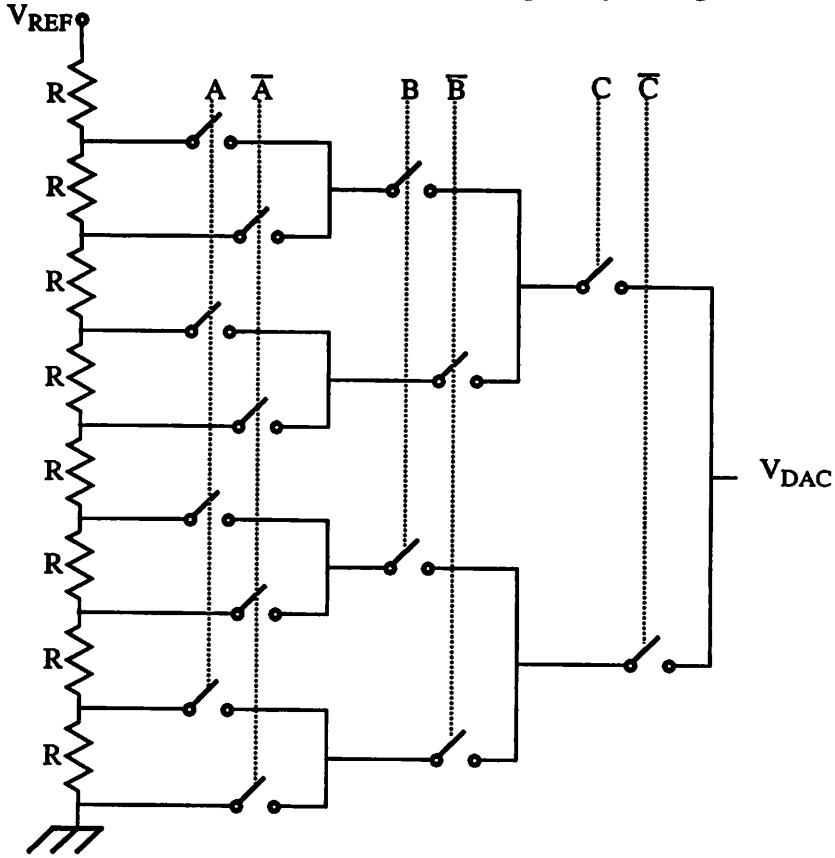


Figure 2.18 A 3-bit Resistor String DAC, using MOS switches. C is MSB.

resistor DAC, with a binary decoding using pass gates. These DACs tend to have a large series resistance due to the pass gate decoding of the output, and may be slow if driving a large capacitive load directly. Since this is a unit element architecture, DAC monotonicity is guaranteed. The diffused resistor technology is compatible with standard digital CMOS processing. Resistor string DACs are often integrated with digital controller ICs, and used as reference level generators for flash ADC structures. This architecture has been demonstrated in some high speed applications, requiring a fast output buffer to drive external loads [PELG90].

Capacitive DAC structures operate in the charge domain, sampling a reference voltage on weighted capacitor arrays and then replicating this charge at the output of an opamp. Fig. 2.19

illustrates a capacitive DAC. Their more common use is as part of successive approximation

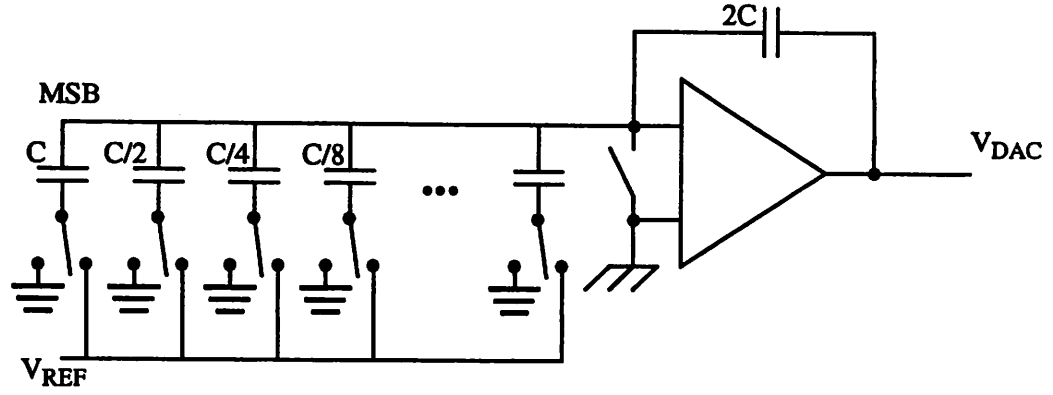


Figure 2.19 A Capacitive DAC samples a charge. The opamp and feedback capacitor convert the charge to a voltage output.

capacitive A/D converters, where the same capacitive array is used for both the sampling and the DAC function, and the only other circuit is a low offset comparator [GRAY94]. Capacitive DACs are also used in 1 bit DACs found in oversampled architectures [KUP91].

In medium accuracy DAC designs, nominal element matching without trimming or calibration may be accurate enough. When higher accuracy is required, a number of techniques have been applied to correct for element mismatches. Thin film resistors often are laser trimmed at production time to achieve accurate matching [NAYL83]. Laser trimming using fuses has been implemented in CMOS processes [DEWI93]. Autocalibration techniques, in which the circuit calibrates itself on power up, have also been implemented. Lee measured the errors for 6 MSB weights in a binary weighted array, stored the error digitally, and compensated for these errors with a correction DACs when the weights were used. All analog calibration techniques, using dynamic element matching or current copying, have been implemented to achieve 16 bit matching for bipolar and CMOS current source elements [SCHO86,GROE89].

Multistage implementations using combinations of these techniques are found in the literature. In particular, the combination of a capacitive DAC, using a resistor string for interpolation has been used for a calibrated high resolution converter [LEE84]. The opposite approach, with a coarse resistor string followed by a capacitor array has also been used [YANG89].

2.4 A DAC Architecture and Implementation for Synthesis

Now that the range of possible DAC architecture and implementation choices has been briefly covered, a combination of architecture and implementation must be chosen for DAC module generation. There are a number of issues to consider when making this choice. The DAC module should be compatible with standard CMOS processing, widely usable across speed, supply, resolution, and application, have a flexible layout aspect ratio, and suitable for some automatic layout method. It should also be scalable for either on chip or off chip loading. It is impossible to consider either the architecture or the implementation independently, but for this discussion the motivation for the implementation will be described first, and then the architectural choice.

It is best if the DAC module generator is applicable across all CMOS fabrication technologies. Resistor string and MOS current source implementations are compatible with standard CMOS digital process technology (with one polysilicon layer). Capacitor implementations are compatible with analog CMOS (double polysilicon) process technology, but are more difficult to build in a single poly technology. Thin film resistors are seldom seen integrated with CMOS, and were not seriously considered.

Capacitor and resistor/capacitor charge based DAC implementations are very common as elements in switched capacitor Analog/Digital Converters, where the charge output is a natural choice for the DAC, but in stand alone applications there are two drawbacks. The first is that creation of the analog charge output is a two phase process, and there is a clock phase on which the output is not valid. The second is that the design for a stand-alone DAC becomes two separate designs, one for the switched capacitor DAC, and a separate module generation for a buffer amplifier. At high speeds the buffer amplifier circuit is a difficult problem by itself.

Resistor string implementations are more amenable to a stand-alone application than capacitive designs, because they do not require two clock phases to create a DAC output. They still have relatively low driving capability, and require a buffer circuit in most applications. The fastest resistor string designs require both fast settling at the resistor string output and specialized output buffer [Pelgrom].

The current source implementation was chosen for this work. Current source DACs have seen a wide application from low to high speed and low to high resolution. In particular, the high volume video DAC application for computer displays is dominated by current output DACs driving 75 ohm transmission lines directly [ADI92]. While obviously applicable to current output applications, it is not difficult to convert a current output to a voltage through a simple resistor, or transimpedance buffer for voltage output applications. Furthermore, this technology is compatible with the least complicated CMOS processing.

Once the implementation choice was made, an architecture decision was required.

Oversampled architectures were not considered for analog synthesis, because the analog circuits are relatively simple, and most of the design is spent on the digital design of the upsampling DSP and the analog anti-alias filter which follows the one bit DAC. Design synthesis techniques have been applied to oversampled ADC configurations, in which a standard cell was used for the analog circuit, and digital circuits were synthesized to match the resolution and bandwidth to the specifications[MAR93]. A similar approach, with an analog standard cell and a synthesized DSP makes sense for oversampled DAC synthesis, meeting high resolution but relatively low frequency specifications.

Binary weighted architectures suffer from poor DNL and Glitch Energy at the major transitions, and point to the use of a segmented architecture to solve these problems.

Interpolated architectures do "guarantee" monotonicity, but at the cost of additional circuit complexity. A low DNL specification may be met either through interpolation, a unit element approach, or through a high degree of segmentation.

This leaves us with the segmented architecture, with the segment size design parameter M strongly affecting DNL and Glitch. As seen above there are actually two flavors of this architecture, depending on how critical the Glitch Energy performance is. In low speed applications, when the Glitch specifications are relatively easy to meet, the segmentation size M is relatively large, such as N/4 or N/8. These segmented architecture circuits and layouts are very similar to standard binary weighted architectures. The additional number of cell selection signals and thermometer

encoding logic is has minimal effect on the complexity, compared to a binary weighted DAC. In high speed designs, where M < 16 to meet glitch energy performance specifications, there may be 64 or more segments to be controlled. The thermometer encoding for the segment select signals becomes a significant problem, and the placement and routing problem for many segment cells is difficult. The best placement and routing strategy for a one level encoding of these signals is a single row placement, with digital encode cells aligned with single current sources and switches [SCHO88], but in this case the aspect ratio of the DAC layout cannot be controlled, and matching of devices across large distances is an important issue. The solution to this encoding and placement problem is a two level encoding, separating the encode into row and column encoding, with a local digital encode circuit at each cell[LETH87]. In this layout style the complexity at the cell level does not scale with the number of segments, so going to smaller values of M does not make the layout problem more difficult.

A segmented current source architecture which allows a high degree of segmentation in a 2 dimensional current cell layout was selected. This covers a range of medium resolution, medium to high speed applications, including the video DAC application.

2.5 D/A Converter Summary

After the preceding sections, it is helpful to summarize the specific DAC architecture and circuits chosen for this synthesis project. Besides describing the architecture, circuits, and layout style, this section will also list key design variables for both an architectural and circuit level description of the design.

A highly segmented current source architecture is selected for synthesis. The 2-dimensional layout style, with row/column encode circuits and local cell level encode logic is used. In each cell is a digital selection circuit, an optional clocked latch, an inverter, a matched current source and a current switch. A central analog I/O bus runs vertically through the array. analog signals and supplies are routed from the bottom center of the array to each row, and then across the rows. Digital row and column select signals are latched at the top and sides of the array, and buffered to drive the large loads across the array. Row signals are driven from both sides, to prevent digital routing across the analog bus in the center. All signals, analog and digital, and all supplies are routed hori-

zontally across the array, to prevent capacitive coupling between analog and digital lines. The lone exception is the column select lines, which must run vertically. This line is must be shielded from analog supplies and bias lines as it passes through the cell.

Bias cells are placed in each row, 1/4 and 3/4 of the way across the array, so that the average distance from the bias cells to the analog bus is the same as the average distance from current cells to the bus. (This reduces gain error caused by differences between bias and current cells). The bias cell has current carrying devices with the same orientation as the current cell, for matching, plus additional devices to generate bias voltages. Bias current is on a separate input from bias voltage, to eliminate bias current induced IR drops on the bias voltage lines. If the size of the bias circuitry is to be increased, this is done through repetitive placement of the cell.

The number of rows and columns (NROWS and NCOLS) in the array are dependent on the designer input. By adjusting the number of rows and columns the aspect ratio of the design may be modified. Rows and columns are turned on using a hierarchical symmetrical switching scheme, cancelling out first order and second order gradient effects. The disconnection between right and left side row signals permits a more sophisticated row select ordering, which centers each logical row to the middle of the array.

Segment cells have current sources made up of an M individual 1 LSB current sources, wired in series. The current sources may be in a mirrored layout, or all devices should be placed with the same orientation. This has not caused a penalty in video DAC designs, because the LSB currents require larger than minimum sized devices, but in low current designs it would be advantageous to allow single device segments, and then some current subdivision technique for the LSB cells.

LSB cells, for binary weighted currents with weight less than M are placed in the top row. The number of LSB cells is limited by the number of columns, and allowing for common centroid placement, this limits M to the NCOLS/2. These cells are identical to segment cells, except that they require no special digital encoding circuits, and their current sources and switches are scaled to the appropriate cell weight. Their inclusion in the array permits accurate prediction of the mismatch errors between the LSB elements and the segment cells. In the implementations done for this work, the LSB cells have weight of 1 LSB only, simplifying cell design, but limiting the max-

imum value of M to the NCOLS. Cells are mirrored as they are placed, to improve device matching in LSB cells with only one current source.

A conceptual placement of this DAC layout is shown in Fig. 2.20a, emphasizing DAC function, and the placement used for layout is shown in Fig. 2.20b. Note that when the number of segment rows reduces to 1, this architecture degenerates to a single row DAC architecture, and the cell level decodes are no longer required. Additional subcell schematics for the DAC module generators are in appendices B and C.

This is an excellent architecture for high speed Video DAC applications, and prototype circuits developed with this module generator have exploited that. Some limitations to the cell library are disadvantageous for low speed and low current applications. Relatively simple changes to the existing architecture, such as allowing more than 1 LSB current per LSB section cell, or developing a set of current subdivision based cells for the LSB row, would alleviate these problems. These are viewed as potential improvements to the cell library, but were not required for this research.

There are two types of design variables for this DAC. Architectural variables describe the degree of segmentation for the cell, overall placement at the module level. These are usually integers. Circuit level variables describe the device sizing for analog and digital subcells, such as current sources, switches, bias devices, digital latches, inverters, logic, and buffers. Table 2.2 lists these variables, with a brief description of their use, and their type. When used in a DAC implementation, all of these must be rounded to an integer or power of 2 number.

In this chapter the choices for a DAC implementation have been briefly described, motivating the choice of a segmented current switched DAC for module generation. In the following chapters the implementation of this in an analog synthesis system using this DAC architecture will be described.

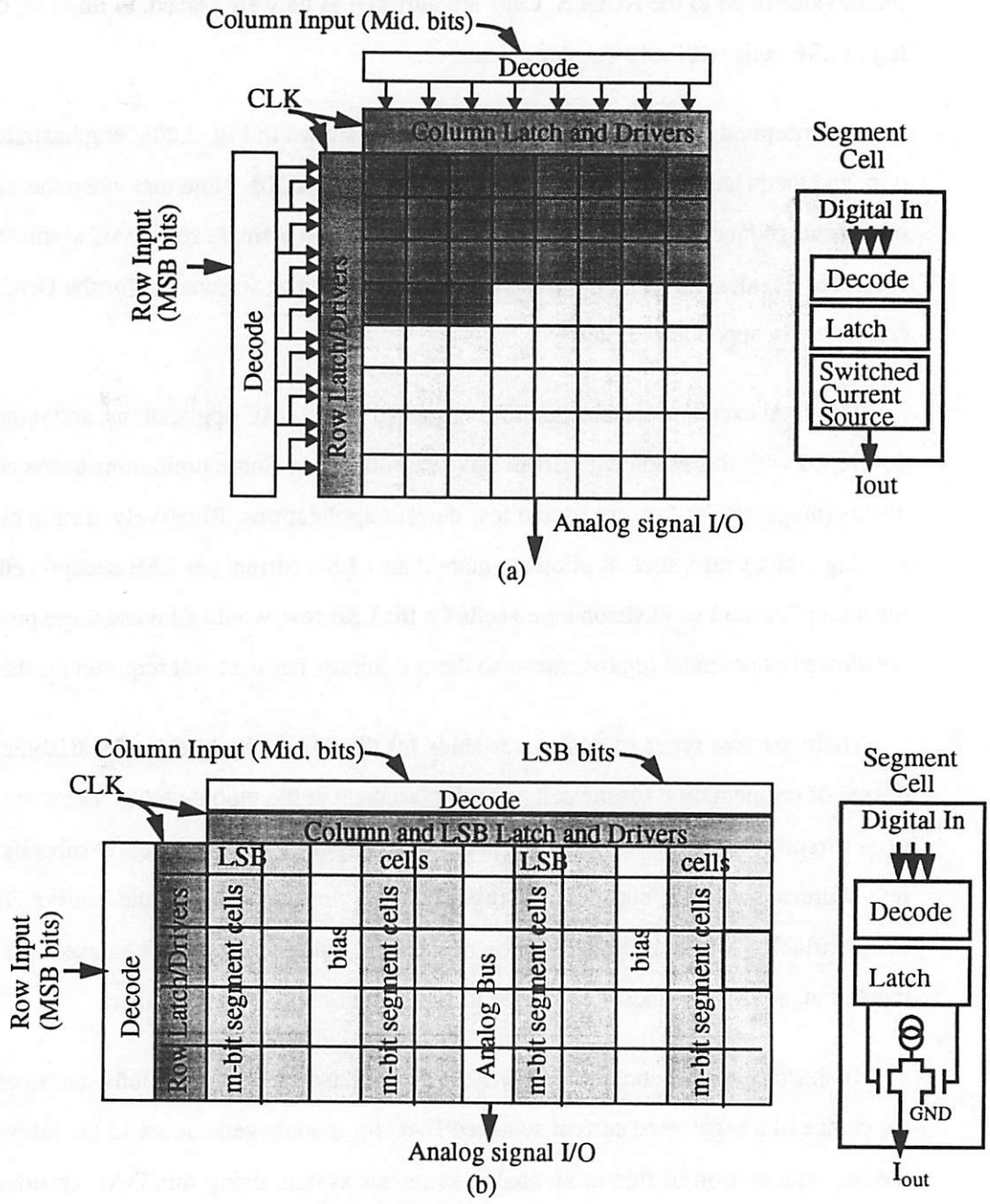


Figure 2.20 DAC Circuits for Module Generation: 2-D Current Switched Architecture. a) functional block diagram and b) module layout.

Table 2.2 Design Variables for DAC Module

Variable	Description	Type
Architecture Level Variables		
N	Number of Levels (# of LSB)	Power of 2
M	Number of LSB per Segment	Power of 2
Rows	Number of Rows	Power of 2
Cols	Number of Columns[a]	Power of 2
Bias	Number of Bias Columns	Integer
Cell Level Variables		
W1, L1	Size of current source main device	Layout[b]
WC, LC	Size of current source cascode devices	Layout
WS	Width of Current Switch	Layout
WBias1,2	Sizes for bias cell devices	Layout
WBUS	Width of power bus	Layout
WDig1-8	Sizes for digital buffers and decodes	Layout

a. N,M, Rows and Columns obey relationship N=M*Rows*Cols.
b. Layout dimensions must be rounded to match the layout grid. They are treated as an integer number of grid units.

CHAPTER 3

DSYN -- A Compiler for CMOS Current Switched Digital/Analog Converters

3.1 Introduction

This chapter introduces the DAC synthesis process for DSYN, a set of tools and libraries for design synthesis and layout of Digital/Analog Converters. Once a DAC module has been implemented within the DSYN framework, a finished layout may be produced from an input specification. DSYN consists of two important parts. It includes a set of generic simulation, optimization, and layout tools, appropriate for DAC module generation, but not circuit or design specific. For each DAC implementation the design specific estimation, design partitioning and layout inputs must be created. DSYN uses the synthesis issues described in chapter 1 to motivate its choices for design selection, analysis, and module layout. The DAC modules implemented to date have been the high-speed segmented current-output type described in chapter 2.

This chapter gives the outline of the synthesis process, from specifications and designer input through intermediate results to a final layout. The structure of the DSYN framework is the focus -- what exactly are the technology inputs, what are the tool inputs and outputs, and how is the design verified. Design specific and non-specific aspects of DSYN will be identified. At this level an interesting issue is the view that different users have of this set of tools. This chapter will consider DSYN from the point of view of a user synthesizing a DAC to a his specification, a designer

implementing a new DAC in the DSYN framework, and a technologist changing the process technology description. While the user, designer, and technologist may all be the same person, this is the nomenclature I will use for this chapter. The algorithms for design selection, analysis, and layout will be described briefly here, but the motivation for the use of these algorithms and an in depth description must wait until chapters 4 and 5.

3.2 Synthesis Process

Fig. 3.1 illustrates the inputs, outputs, and synthesis process for DSYN. To create a module to specifications, the user inputs the design specifications and application conditions. These may be used as design constraints, or as constants for use during design analysis. A second type of input is

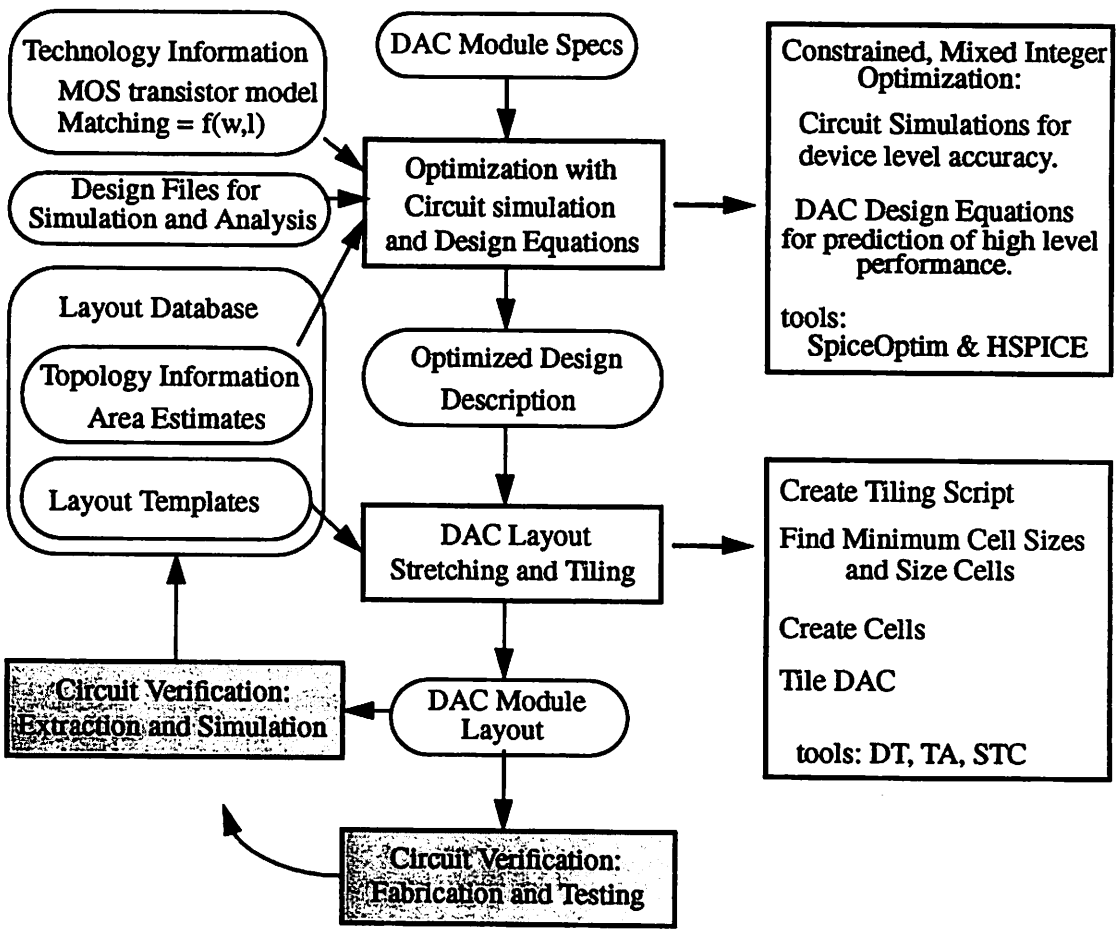


Figure 3.1 DAC Synthesis process in DSYN.

the technology description, in the form of device level models and layout parasitics such as wiring capacitance and resistance. Module specific inputs are also required. A cell library and a tiling algorithm are required for the layout step, and a set of simulation files which will predict circuit performance based on any design input is needed. The design input also specifies the design variables for the module, and may include some design specific constraints, such as bias conditions which must be maintained.

The DSYN synthesis process follows a standard two step analog synthesis process. In the first step the circuit architecture and device sizes are determined, and in the second step a layout is created. The intermediate result is the set of design inputs plus the selected values of design variables. This can be expanded into a full circuit netlist. The final output is the DAC layout, and a list of DAC performance estimates.

The key elements in the first step are design estimation and selection. For design estimation, a combination of low level circuit simulation and high level analytic equations are used to predict the performance of the DAC. As shall be shown in chapter 4, this allows accurate estimation from low level bias conditions, through random current cell mismatch, to DAC level static nonlinearity. This is implemented using the commercial circuit simulator HSPICE, allowing use of modern short channel device models, evaluation of design equations, use of re-runs for worst case design, and a flexible performance measurement processing, all within the same tool. The estimation step may take from a few seconds up to a minute or more, depending on the complexity of the circuit, and the need for transient simulations to estimate dynamic characteristics.

For design selection, an optimization approach was chosen, to allow a minimization of design cost, without having the solution space be restricted by a knowledge based framework. The optimization algorithm must perform a mixed integer, constrained optimization, minimizing either an area or power cost, while using a slow circuit simulation tool for design estimation. An optimization approach using a combination of a supporting hyperplane algorithm for optimization with a branch-and-bound mixed integer step met these requirements, and was implemented in the optimization tool SpiceOptim. The complete optimization takes from 100 to several hundred simulation runs, depending on the number of design variables, the degree of discreteness of these variables, and the number of active constraints. The optimization problem is simplified through the use of a

parametrized design, reducing the number of design variables, as in OPASYN [KOH89] and many other synthesis approaches. Optimization time typically takes a 6 hours compute time or less, dominated by the time spent running HSPICE simulations.

Once the optimization is complete, a list of all device sizes and cell structure variables is created, to drive the layout synthesis step. Many tools produce a netlist at this point, but in DSYN this is omitted, because the layout step inputs topology information through a cell library, and requires input of values for device sizes and tiling algorithms, rather than a single netlist with all device sizes. There is a common pre-processing step used in both simulation and layout which expands the design description from a set of design parameters to a full description of the DAC cell.

For the high speed CMOS current switched DAC implementation chosen for DSYN, the layout strategy was selected to create area efficient layouts, and allow accurate prediction of layout parasitics without requiring a complete layout step. Towards this end, the layout approach starts with a cell library, and a tiling algorithm, and completes the layout through a deterministic process of stretching the library cells and tiling them for the DAC layout. The first design specific input is the tiling file generator, which converts high level DAC information, such as number of rows, columns, and bias cells, to a tiling file, which is a simple list of the specific subcells to use at each point in a tiled array. By including some flexibility in this tiling file generator, the same program has been used for several different DAC implementations.

The cell library is defined with minimum sized devices, and locations in the cell where the cell may be stretched to resize the devices. By stretching the cell the device sizes can be changed to the sizes determined from optimization. Stretch operations may include just one device, or may cover the width of the library cell. The stretching tool, STC, first implements the required stretches, and then performs additional cell modifications to create a rectangular output. It also has a sizing mode, in which it returns the expected cell dimensions, but does not create the cell. The tiling program, TA, first runs the cell generator in sizing mode for every cell, finding minimum dimensions for every row and column in the array. Then it uses STC to create the subcells, and tiles them into a final layout. The total layout step requires less than 2 minutes run time on a workstation. The tools use MAGIC [SCOT85] as the cell database, and the final output is a MAGIC layout. The full description of the choices that let to this layout approach and its implementation is in chapter 5.

3.3 Accurate Performance Estimation and Verification

An important goal for DSYN is accurate performance prediction, but the uncertainties and inaccuracies of the synthesis inputs conspire against this. A definition for accurate performance prediction must be made first, and then a methods of attack for accurate performance estimation will be described. This will motivate the estimation philosophy used in DSYN.

Design verification is an important part of any design or CAD process. To verify the correctness of the performance predictions made in this design process, both design process inputs and the simulation/analysis process must be checked. An important technology input is the process model, including device models, verified before synthesis begins, and models for device mismatch, which can be initially estimated, and then verified through fabricated results. The simulation/analysis step is verified through a back-end verification, using full circuit extraction and simulation after synthesis, and testing of prototypes.

3.3.1 Performance Estimation Philosophy

What is a metric for accurate performance prediction? In many analog synthesis approaches a final comparison of predicted results to SPICE circuit simulation is used to indicate the accuracy of predictions, and therefore the general promise of the method for widespread use [KOH89, OCHO93, HARV92]. This may indicate the matching of analytic results to SPICE models, or SPICE models used in SPICE to SPICE models used in another simulator, but this does not include the misfit of SPICE models to real devices. As will be seen in the following section, model inaccuracies for short channel MOS devices used with typical analog bias conditions are significant. A better metric is to compare predicted results to actual fabricated devices[JUSU93,GIEL90,DEGR87], with the effects of model misfit included in the estimation process.

Two general approaches can be taken to correct the estimation process -- a back-end correction after design estimation, or a front-end correction to specific known errors in the inputs. In a back-end correction, the synthesis is run, the results measured through simulation or fabrication and test, and correction factors used to calibrate the predicted results to the actual results [KOH89, JUSU93]. After the correction factor is known, later uses of the tool should result in performance predictions which match fabricated results. There are problems with this approach. What form should the cor-

rection factor take? Additive? Multiplicative? How should one choose? Is it possible to guarantee that a correction factor found for one set of specifications apply to a different spec? No! A better approach is to locate the specific places where design inputs are inaccurate, and make corrections at that point. This front-end approach is more time consuming, because all significant sources of error must be accounted for, but with this estimation philosophy circuit performance can be accurately predicted across a range of design specifications, instead of corrected for each specific design.

DSYN adheres to the front-end philosophy, correcting for all errors in simulation and modelling where they occur, and avoiding the use of back-end calibration of the estimation process. Comparison to fabricated circuit performance is the ultimate test for this approach.

3.3.2 Device Model Verification

Verification of process models is done before circuit synthesis begins. In MOS device modelling, the most common models used, including SPICE level 1, 2, 3, BSIM 1, 2, 3, and the Metasoft BSIM 1 variant (level 28) are designed to fit well in strong inversion and in sub-threshold. These are the important regions of interest for digital circuit simulations. In analog circuit design two other parts of the device curves are emphasized: the weak inversion part, with Vgs a few hundred mV above threshold, and the onset of saturation, with Vds<Vgs, but Vds>Vdsat. At best, these regions are treated as transition regions in the device model. An effort is made than to line up the Id curves at those points in the models, with continuous derivatives. Many of these models have continuous first derivatives, but have discontinuous second derivatives at the onset of saturation, resulting in large changes in the sensitivities of small signal derivatives to bias at those points. In the context of a circuit simulation and optimization environment, this discrepancy between device models and actual device curves must be bounded. For this work a set of BSIM 2 models were optimized to measured device curves, and the discrepancies after optimization were noted. While the models fit the device currents with small absolute errors, there were large discrepancies in the small signal g_{ds} conductance. Fig. 3.2 plots measured g_{ds} for a 1.2 μm device, and the fitted model. The model is fit for all regions of operation, so it is not possible to further optimize the fit shown here without impacting the overall curve fit. After a man-month of model optimization and curve fitting, using BSIM, Level 28, and BSIM2 models, this was the best fit seen in this region! In the

lowest pair of curves the device is biased in weak inversion, with current density of 2 microamperes per micron of device width. Here the model fits poorly across the saturation region, and underestimates g_{ds} by a factor of 4. In the top pair of curves the same device is biased in strong inversion, (V_{gs}-V_t > 1volt), and a much better fit is seen in the saturation region. At the onset of saturation, near V_{ds}=1V, a kink in the model is seen, and in that region the model is again in error by a factor of 2. In the middle pair of curves a large error is again seen throughout the saturation region, but it is smaller than in the weak inversion fit.

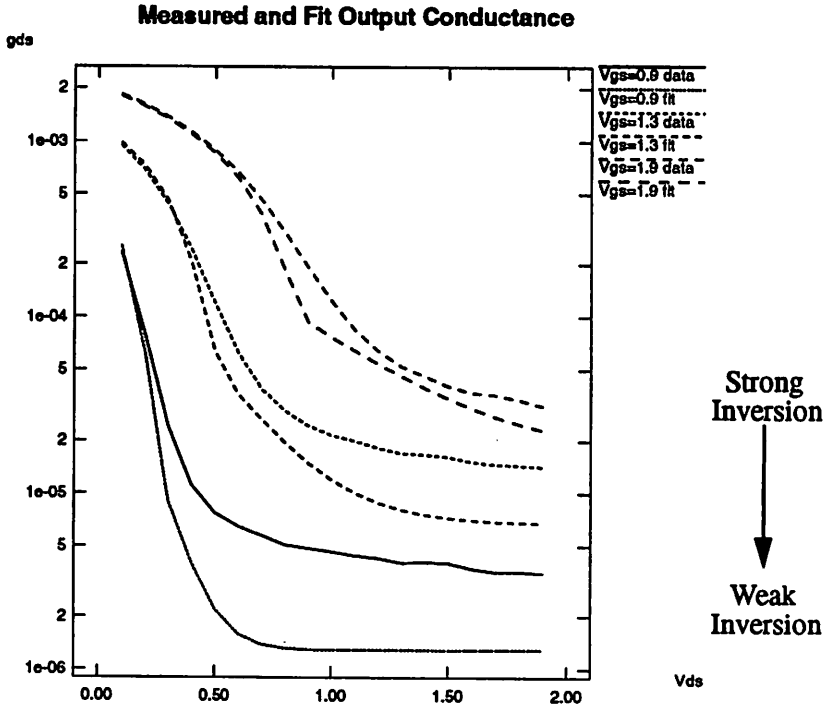


Figure 3.2 Measured data and model fits for g_{ds} of a 1.2μm channel length MOS device. Model g_{ds} is too small for low V_{gs} and V_{ds} near V_{gs}-V_t

If synthesis using circuit simulations does not include the effects of inaccurate device models, then the results of fabricated devices may not meet design specifications. Unlike typical analog MOS circuits, DAC current sources tend to be biased in strong inversion to reduce threshold voltage induced mismatch, so the modelling inaccuracies in weak inversion do not affect these designs. The errors at the onset of saturation are important, particularly if high swing cascode current source biasing is used, or a large voltage swing is seen at the current source output. In DSYN

these errors were compensated for through constraining the minimum V_{ds} for saturated devices to 200 mV above V_{dsat}, and by derating the output conductance of saturated devices by a factor of 2. Poor device models is one of the MOS analog circuit design issues which has made the field difficult for analog CAD, and interesting for circuit designers.

Estimates of element mismatch is an important technology input, but it is impossible to fit a mismatch model to a technology without test structures. Most work in this area gives device matching data for a single technology, and does not suggest a method for extrapolation to new technologies. There is an exception to this -- Pelgrom measured matching for several different technologies, and suggested trends in mismatch behavior as oxide thicknesses and line widths are reduced [PELG89]. DSYN uses the mismatch model suggested by Pelgrom, and conservative extrapolations from his data have been used for the mismatch parameters. The mathematical model commonly used for device mismatch as a function of spacing [PELG89, MICH92] corresponds exactly to a linear process gradient, so a linear gradient (with random direction) is assumed for these functions.

3.3.3 Design Synthesis Verification

Once the synthesis process has been run, a second verification step is a full extraction of the design, including all parasitic capacitances, and simulation. This step verifies the estimation of parasitic capacitances made during the optimization simulations, and overall correctness of estimates based on simplified circuits. While statistical variations cannot be simulated, dynamic specifications such as settling and glitch energy can be verified, and proper operation of digital buffer and decode circuits as well.

The third verification step is through fabrication of devices, and measurement to test of the designed circuits. This can verify the models and constants used for random mismatch and process gradients, and simulation models for chip and board level circuit behavior.

3.4 Views of the DSYN synthesis process

Depending on the user function, there are different views of any synthesis process. In this section the different views of the synthesis process will be described, depending on the purpose of the

user interaction. The DSYN tool will be described from the point of view a user, a designer, and a technologist.

When a DSYN user implements a DAC design for a new set of specifications, using an existing module architecture and technology, the process follows the path shown on the left side of Fig. 3.3. The user inputs the design specifications to the optimization/analysis step, and waits for the results. Depending on the complexity of the module this may take up to a few hours. The intermediate result is passed to layout, and within a few minutes the layout is created. This design should be extracted and simulated to verify the correctness of the design before fabrication. From this view the complexity of the design is not apparent, except as measured in optimization time. The choices for design variables, design partitioning, and estimation accuracy have already been made, and are not required for the user.

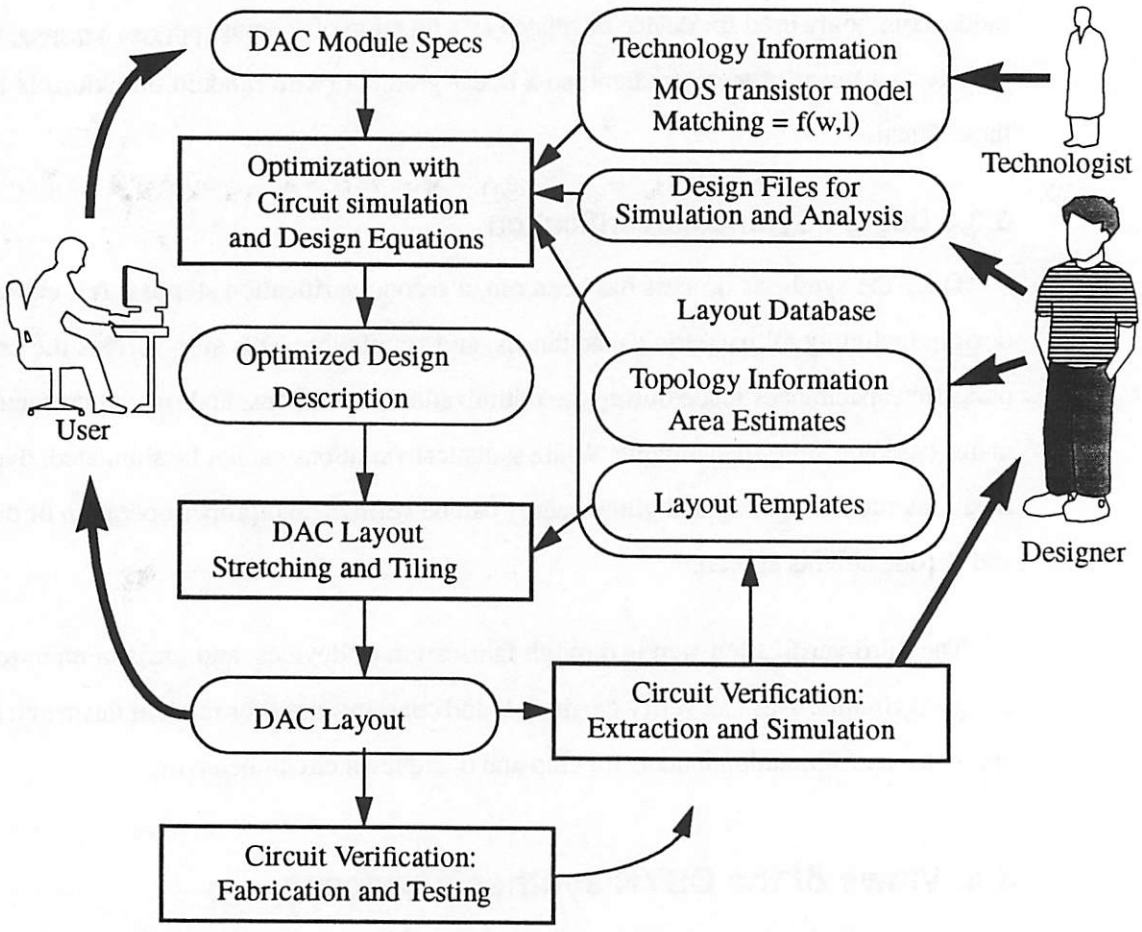


Figure 3.3 DSYN as seen by a user, designer, and technologist.

While creating a module using an existing design is fast, implementing a new design is not. Implementing a new DAC architecture in DSYN requires design partitioning, determining the set of simulations and equations for design estimation, creation of layout cell libraries and verification of all of this through circuit extraction and simulation. The right side of Fig. 3.3 illustrates the libraries a designer must create as part of this process, and the role of the designer in process verification. This process is the same magnitude of difficulty as the full custom design process. In both cases the designer must understand all important circuit issues. Fortunately, if a previous design exists, opportunities for re-use of derived equations and simulation files speeds the design implementation, just as in full custom design.

It is not clear whether design entry for DSYN is more or less difficult than the analog design process for a single specification. In both cases analysis and simulation must be specified for determination of the performance of the circuit as a function of input specifications. There are two distinct mind sets for this. In full custom design, the designer answers the question "what is the best design which meets my specifications," while in design entry for DSYN the designer must answer, "How can I simulate and compute the specifications from any set of design inputs." Unfortunately there are several factors which make design entry for DSYN more difficult than standard analog design. The design partitioning must be done explicitly, and all constraints must be computed explicitly. Since the optimization process is not smart, circuit simulations must be set up to converge, despite poor choices for design input. Most importantly, for design entry in DSYN any set of reasonable specifications must be allowed, so simplifying assumptions which may be applied for a particular specification cannot be used if a wide range of input specifications is expected. The main advantage the DSYN engineer has is that he does not need to choose the final values for his design variables and device sizes. Once the design has been entered into the simulator, the optimization process will find these for him.

The third view of the DSYN process is from the view of the an engineer making a technology shift. In a shift to a new technology, the technologist must enter nominal and worst case technology inputs for wiring widths and overlap capacitances, input device models for nominal, fast, and slow corners, and note model discrepancies, as shown in Fig. 3.3. Estimates of device mismatch as

a function of device area and spacing must be made. Not shown in the figure are any modifications to cell libraries required when layouts are scaled to the new groundrules.

3.5 Summary

This chapter has presented an overview of the DSYN digital/analog converter synthesis tools and libraries. The operation of the tools, and the interactions between the tools and various users have been described for both module generation from specifications and design entry for new architectures and technologies. In the following chapters the choices for optimization, estimation, and layout algorithms will be explored, and motivation for these particular implementations will be given.

CHAPTER 4

DAC Analysis and Optimization for Synthesis

4.1 Introduction

In this chapter the details of the DAC design analysis and selection process will be discussed, including a review of synthesis requirements, available approaches, and implementation details. This introduction will review issues raised in previous chapters, and then the analysis and selection methods will be discussed in turn. Since an optimization algorithm was used for design selection, a discussion of various optimization methods is included. The interaction between these methodology choices is important, and the complimentary operation of these algorithms is discussed. A key design choice is to avoid a strongly hierarchical approach in this problem, and the motivation and consequences of this will be touched on.

In chapter 1 the requirements for a circuit synthesis methodology were discussed, and the points which apply to design estimation and selection bear repeating here. For design selection, the methodology should fully explore the design space; it should not be limited by algorithms which may ignore better solutions, or limit the choice of free variables. Also, the design selection process must constrain the design to meet performance specifications. Performance estimation must be accurate, so that when designs are predicted to meet specifications they actually will. The use of high level fudge factors to correct for discrepancies between estimated and actual performance is

strongly discouraged, because it is usually not clear how discrepancies for one design specification should map to other design specifications. Finally, the analog synthesis process should not take excessive computer time, but speed is not critical. A few hours on a fast workstation is good enough here.

The preceding issues are for general synthesis. There are some specific issues which must be addressed for the current switched DAC architecture used in this work. Recall that some of the key design variables must be integer valued by the end of the synthesis process. If an optimization algorithm is used for design selection, this requires some integer programming to reach the final result, but it also has implications for estimation. Though these variables are typically considered integers, if gradients are to be computed using finite differences, then being able to compute performance for non-integer values of (eventually) integer variables will be helpful. If the circuit is being evaluated in an infeasible region, it is difficult to guarantee that derivatives for performance estimates will be reasonable. One solution is to choose an optimization algorithm which does not leave the feasible region [BRAY81,NYE88], and another is to choose an algorithm which is tolerant of these inaccurate derivatives of performance estimates when the estimate itself is clearly infeasible.

For this class of DACs, estimation of design performance requires a combination of analyses, at both circuit level and DAC architecture level. A circuit level analysis of DC performance is needed to predict bias margins, and bias dependent mismatch of devices. A transient analysis is required for determining settling, switching, and delay times. For analysis of static linearity (INL and DNL), the predicted mismatch of devices is combined with sensitivities to voltage drops and DAC architecture inputs. For glitch energy the results of a simple transient may be further analyzed to obtain glitch energy predictions, or a separate transient analysis may be used. Finally, the circuit is subject to process variation and layout related parasitics, and these must be incorporated into the synthesis process somehow. No one analysis method meets the combination of accuracy and speed requirements for DAC design estimation.

There is also a set of standard rules for optimization which improves the robustness of any optimization approach. As much as possible, the number of optimization variables should be minimized, since the optimization time is linear or worse with the number of optimization variables,

(especially if gradients must be computed through finite differences). The complexity of the objective and constraint functions should be minimized, since the more linear the system, the better and faster the convergence in optimization[1].

4.2 Design Estimation Review

Design estimation approaches were mentioned briefly in chapter one. Here these will be discussed in more depth, covering circuit simulation, analytic equations, and behavioral simulation approaches. A description of each approach and its application to DAC design estimation is included, indicating advantages and disadvantages for this problem.

4.2.1 Circuit Simulation (SPICE)

Design estimation using circuit simulation is the most familiar mode of low level computer aided circuit analysis for design engineers. The best models for MOS devices have been developed for these simulators, and accurate simulation results for DC, AC, and transient performance are easy to obtain. Disadvantages of this approach are that it is relatively slow, even for small circuits. In the DAC problem, this is the best method for low level circuit estimation, because accurate performance prediction requirements may be met. Slow simulation time is a fact of life for this approach, and though some steps can be made to improve this situation in a synthesis implementation[NYE88]. If this approach is used, then the optimization algorithm should try to minimize the number of simulation runs.

For high level DAC estimation, full simulation is a poor choice for determining static linearity. Two important effects are difficult to manage. Resistive drops in a multiple current source design are important, but to include this effect in a full simulation framework requires a inclusion of N/M current sources and N/M resistors between the current sources. If there are many segments, the complexity of the problem gets large, and simulation is slow. If implementations must be evaluated when the N/M ratio is non-integer, then the setup of the problem is unclear. For statistical mismatch a more insidious problem appears. Monte Carlo methods are typically applied to statistical

1. Specifically, a function $\sqrt{a^2 + b^2}$ as problematic when it was part of the objective function, with both a and b as design variables. Also, a choice is generally made to let M, the number of lsb elements per segment be an integer design variable, rather than using $B = \log_2(M)$ as the design variable, since the relationship of M to performance is much more linear than B.

analysis of circuits, requiring multiple reruns, and yielding a result which is itself a random variable. If the finite differences required to compute a gradient use these random variable results, then the gradients will be random variables, leading to convergence problems in optimization. A faster, completely deterministic approach is required for high level DAC estimation.

4.2.2 Behavioral Simulation

By abstracting the circuit into a set of abstracted elements, behavioral simulation can be used for high level DAC estimation [LIU93,CHAN94]. Some of the design complexity problems in full simulation approaches can be overcome, and complete transfer functions, including envelopes for INL and DNL curves may be produced. Methods which use a combination of design sensitivities and parameter variances avoid the used of monte carlo analysis, and directly predict mean and variance of all output codes, making this approach suitable for an optimization framework. There are some reasons not to use behavioral simulation. The first is the simulation setup problem when non-integer architecture variables are input, similar to that described for low level simulation. Second, behavioral simulation, though much faster than circuit simulation, may not be fast. Simulation of an example 5-bit DAC took 540 seconds on DEC 5000/125 workstation [LIU93]. While much better than the full simulation method, it is still orders of magnitude slower than analytic methods.

Behavioral simulation is not appropriate for low level DAC estimation, because of device modeling accuracy problems.

4.2.3 Analytic Equations

The analytic equation approach, using design specific derived equations, is a third estimation method. For high level DAC estimation, it can be used for determination of static linearity from device mismatch and resistive drop information. Rather than computing INL and DNL for the full curve, analytic approaches concentrate on finding an expression for INL and DNL at the worst point in the curve. In general this approach is the fastest, can deal with non-integer architecture variables, and is accurate, assuming the analytic models used are good. The problem with this approach in general is that analytic equations must be derived, a process which may be error prone for complex systems. When second and third order effects are important, it may be better to

depend on a simulation framework for inclusion of these effects, rather than to deal with them in an analytic mode. For this high level DAC estimation problem this is not the case. The important high level architecture inputs and effects which impact worst case non-linearity were modeled well with design equations.

Analytic equations were not seriously considered for low level DAC estimation, due to the device modeling problem.

4.3 Design Estimation for DACs

The DAC design estimation approach uses a combination of low level circuit simulation and high level analytic equation solving. The key elements to the approach are the use of circuit simulation for low level DAC performance estimation, combined with a set of derived analytic equations which first predict element mismatches from bias conditions, and then predict DAC static performance based on element mismatch. Equations similar to those derived in chapter 2 are used for estimating worst case INL, DNL, Gain Error and TUE contributions from device mismatch, and deterministic effects such as resistive drops are included. Chapter 6 has a full description of these design equations. Besides using analytic equations to solve for design performance, analytic relationships are also used to expand the design description from a set of input variables to a full description of all device sizes, circuit areas, and circuit parasitics at the beginning of the estimation step. Fig. 4.1 shows a block diagram of the approach.

As an illustration of the estimation process, consider the steps required to compute INL for a DAC. The initial device sizes and DAC architecture variables are input to the estimator. All device sizes can be computed from this input, as well as all predictions of layout parasitics, and DAC cell sizes. The random mismatch of devices is predicted, based on technology inputs and device areas, and mismatches due to process gradients are computed from the cell size and architecture information. A circuit simulation is run to determine the DC operating point of the circuit, and small signal conductances g_m and g_{ds} are extracted for the current source devices. The post-simulation analysis starts with an analysis of the matched current sources, giving the sensitivity of output current to Vt mismatch, and the current source output resistance. From these intermediate results the contributions to INL from random mismatch, process gradients, and resistive drops in bias lines may be

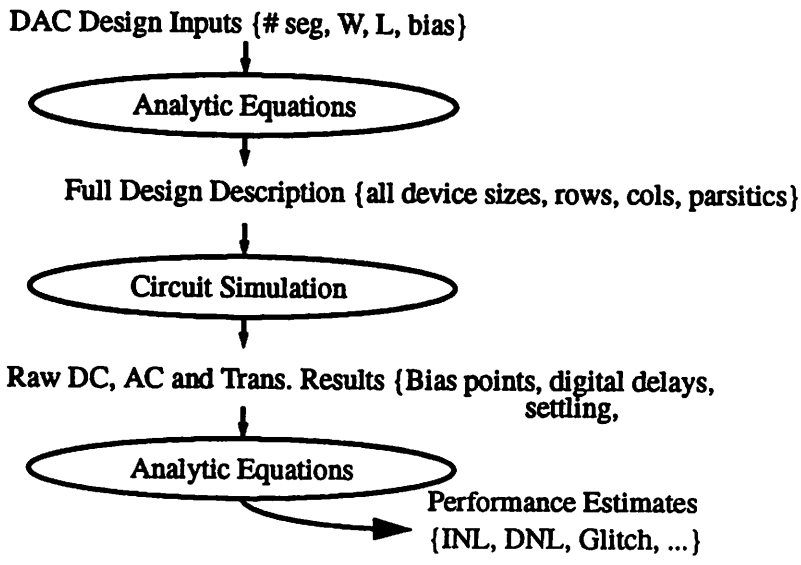


Figure 4.1 Estimation Process flow consists of analytic preprocessing, circuit simulation, and analytic postprocessing. Implementation is in an HSPICE job.

computed for the worst case point in the INL transfer curve. The statistical effects are multiplied to obtain 3-sigma bounds, and the absolute values of the contributions are added to obtain a total INL estimate.

There are a number of implementation considerations for analog circuit design estimation, driven by estimation accuracy and optimization algorithm requirements. Estimation inaccuracies due to layout parasitics and device model inaccuracies, IC technology process variation, as well as some optimization issues must be considered.

4.3.1 Inclusion of Parasitics

Inaccurate performance estimation due to circuit parasitics may be compensated for through either of two methods. In the first, parasitics are accurately predicted during the synthesis process, while in the second a performance margin is allocated to the layout, and a constraint based layout process prevents a layout induced performance degradation from exceeding the allocated margin. When the layout process is tightly coupled to design synthesis, or when it is inexpensive to implement the layout for every candidate set of design inputs, the actual layout parasitics may be included in design estimation [ONOD90]. When the process is loosely coupled the only reasonable option is a constraint driven analog place and route [CHAN92].

In DSYN, the layout process is a deterministic algorithm, so all circuit parasitics can be predicted from the design inputs, without creating a layout. The layout predictions include device source and drain capacitances, wiring parasitics, and also predictions of DAC subcell and module sizes. The DAC layout dimensions are useful as a performance objective to be minimized, and also for prediction of bus wiring resistance and device mismatch due to process gradients.

4.3.2 Device Model Inaccuracies

This estimation process was designed assuming that accurate circuit simulations could be performed for short-channel MOS devices using the most modern device models. As seen in the previous chapter (section 3.3.2), this is simply not true, especially for devices biased in the common regions of operation for analog MOS devices. The method described there to derate the simulated device output conductance was used to correct the simulation predictions. The magnitude of this correction is dependent on the expected bias conditions in the application, and was chosen conservatively. For devices expected to be biased in strong inversion a factor of 2 was used.

4.3.3 Process Variation

Process variation is a fact of life for IC designers. It is the responsibility of the designer to create a design which will maintain high yield despite variations in the process over time, and performance variation over device lifetime and temperature variation. Some designers use nominal device models, and maintain enough performance margin between simulation and specifications to remain confident of device yield in manufacturing. The required margin for this process is difficult to quantify, except through experience.[1] A second process is worst case design, in which the combination of threshold voltage, oxide thickness, channel width and length variation which results in the fastest and slowest possible circuit speeds is used to bracket the design. By simulating with these worst case model files, the designer is assured of good yield despite large process variations. More sophisticated but time consuming methods are available. The process variation may be modelled statistically, and monte carlo analysis run to obtain a distribution of circuit performance due to process variation [SPOT86]. The technique of design centering is used to modify circuits to improve yield, by optimizing to reduce the probability of poor performance. Besides process vari-

1. I found this description difficult to believe -- the engineer describing it said that the other methods were too pessimistic, and he had been successful at meeting specifications with this method. The foundry never produced devices as poor as the worst case process corner.)

ation affecting MOS device speed, the designer must consider other sources of design variability. Circuit speed is sensitive to temperature, and speed, and voltage margins may be sensitive to variations in supply voltage, externally supplied bias current, and integrated resistor values.

For analog circuits the definition of "worst case" is not as simple as for digital logic. For some design constraints, such as signal switching time, the worst case is related to circuit speed, corresponding to slow devices, high temperature, and low voltage. For worst case in performance related to matching the worst case is for fast devices, high vdd, but low bias currents. It is possible for worst case design specifications to occur at combinations other than the traditional fast and slow corners, but exhaustively simulating at all possible combinations of design variation increases the number of simulations by 2 for every additional source of variation. An important problem is that the worst case performance for one specification may move from one process corner to another as the circuit optimization progresses.

Two systematic methods appear in the literature. Dharchoudhury [DHAR92] estimates the worst case performance using a response surface technique, and optimizes predicted worst case performance. When a solution is found, the response surface is updated, and the optimization repeated. This appears to cost a factor of 7 times more circuit simulations[1], due to the repeated optimizations and the cost of computing the response surface. Mukherjee [MUKH94] finds the worst case corner at each step of a simulated annealing optimization, and uses that corner to find the next point in the optimization The search for the new corner is expensive, and is repeated many times. An optimization[2] which was initially 10 minutes became 900 minutes when this method was applied. In both cases the eventual solution meets performance requirements across these variations, at the expense of greater simulation time.

The DSYN framework does not enforce a methodology for designing with process variations. In the DAC optimization implementations a limited worst case design approach has been used. The circuit designer identifies process corners which will result in worst case performance, and simulations at these worst case corners are run at every step of the optimization. Performance must meet constraints for all these specified corners. It is the designers responsibility to include all lim-

1. This problem was an opamp optimization with 7 independent process variables, 5 design variables, and 2 constraints.
2. This problem was an opamp optimization with 6 process and 6 design variables.

iting cases. Process, supply, and bias variations are included in the implementation. Temperature variation was not included in the demonstrations used for this work, but could easily have been incorporated within the same framework. Three corners were used for DC analysis, and the slow corner was used in transient analysis. The simulation time penalty for inclusion of these additional DC simulation runs was approximately 10%. Use of the more systematic methods of optimization with process variation would have resulted in excessively long optimization times.

4.3.4 Estimation in Optimization

As described earlier in this chapter, the use of an optimization approach for design selection places requirements on the estimation algorithm: evaluations must make sense for real valued inputs, and the process must be tolerant of poorly chosen design inputs, at least returning the information that a result does not meet constraints for poor inputs. Variables which are typically integer valued may be evaluated with non-integer inputs by designing the circuit simulations to use unit elements, and letting high level analysis multiply element level results by appropriate factors. Non-convergence problems can be avoided by taking care in the setup of circuits for simulation, or through the use of indicator functions when the circuit simulation does not function as expected. DSYN uses an optimization algorithm which does not require accurate performance sensitivities for infeasible circuit inputs, so a simple indicator function for no-convergence is enough.

4.3.5 Design Estimation Implementation

The design estimation process is implemented within the framework of HSPICE, a commercial version of the SPICE circuit simulator [META93]. HSPICE contains many extensions to the original SPICE implementation, including better convergence algorithms, incorporation of user parameters with algebraic function evaluations, library support, and special measurement statements. The HSPICE library structure was used extensively to allow loading and unloading of process models and simulation setups. The entire design estimation step, from design parameter input to analytic post-simulation processing was implemented in one simulation job. This included the expansion of the parametrized design input to a full design description, multiple simulations for worst case analysis, and post-processing to convert DC and transient simulations to performance measures such as INL, DNL, and glitch energy.

Other implementations of optimization with a circuit simulator have tended to use a tighter coupling of optimization and simulation, and this has given some speed up to the optimization process [NYE88, SHYU88, OCHO94, META92]. In the loosely coupled case, the circuit simulator program must be loaded, the circuit read in, and the simulation set-up run every iteration of the optimization process. In a tightly coupled system, these set-up steps are only done once, but if circuit simulation time is large compared to the set-up time, then this actual simulation time will dominate the run time in both cases. A second advantage of tightly integrated simulation and optimization programs is that performance sensitivities to design inputs may be efficiently computed from one simulation, without requiring finite difference computations across multiple runs [NYE88, SHYU88]. This can greatly reduce the number of circuit simulations required[1].

DSYN originally used the SPICE3 circuit simulator developed at U.C. Berkeley, linked at compute time to an optimization algorithm and some algebraic C code for high level analysis. This approach had the advantage of loading the simulation program once, but did not take advantage of other possible speed-ups. SPICE3 was dropped when it became apparent that the convergence problems with the simulator made optimizations unreliable. It had the additional disadvantage that design specific information was in two places: a circuit simulation file, and C code, linked into the optimization program. Switching to HSPICE solved the convergence problem, and the use of algebraic constructs in the HSPICE deck kept all circuit implementation specific data in the circuit simulation files. Though the algebraic manipulation available within HSPICE is not as powerful as found in a programming language, it is a familiar environment for the circuit designer who must enter the design implementation.

Even with fast modern computer workstations, this design estimation process is not instantaneous for the DAC circuits implemented by DSYN. The first pass simulations were run on a DEC-Station 5000/133, and required 45 seconds run time per estimation. These simulations included 3 DC simulations, one transient simulation for dynamic behavior, plus algebraic pre and post-processing. 8 months later the second pass DAC simulations were run on a DEC Alpha, a machine roughly twice as fast as the DECStation, but the simulation requirements had grown to include

1. Sensitivities are available in HSPICE, but only for DC simulations. Since transient simulations were critical for this work there was no way to use HSPICE and avoid multiple runs computing finite differences.

more complicated DC simulations plus an additional transient for glitch energy, resulting in run times of 36 seconds. The additional available computer power was quickly applied to the estimation step. These simulation times should be kept in mind in the next section, when total optimization time becomes an important limitation to the kinds of optimization algorithms which can be applied.

4.3.6 Example: Estimation of Integral Non-linearity (INL)

To illustrate the design estimation process, an example showing how INL is determined due to random Vt mismatch and finite output conductance in a current source DAC follows. In the full implementation other factors, such as process gradients, resistive bias drops, and device current factor mismatch are also included.

The procedure follows that outlined in Fig. 4.1. A set of design parameters is input, and these are used to compute all device sizes and bias currents for the analog DAC circuits. The random variation in Vt is predicted from device sizes. The circuits are simulated using HSPICE to determine DC bias conditions, for the cascoded current sources used in this design, and small signal gm and gds for the current source devices M1 and M2. Analytic expressions are used to convert the outputs of simulation to INL predictions. An implicit assumption is that operating point variations across the DAC array are small, so all DAC current sources have the same bias point, with the same small signal parameters.

4.3.6.1 INL due to Output Resistance

For a perfect current source, the output impedance is infinite, but in realistic designs the output impedance is finite, and the change in DAC output impedance as a function of code creates a non-linearity. To compute INL due to finite output impedance of the current sources, the output conductance is of a single current source is computed first. After a DC simulation, the bias point of the devices in the current source are known. For the cascode current source in Fig. 4.2, the output conductance g_{out} of the current source is computed, based on the small signal g_{ds} and g_m conductances of M1 and M2:

$$g_{out} = \frac{g_{ds1} \cdot g_{ds2}}{g_{m2} + g_{ds1} + g_{ds2}} \approx \frac{g_{ds1} \cdot g_{ds2}}{g_{m2}} \tag{4.1}$$

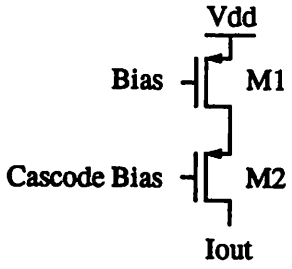


Figure 4.2 Cascode Current Source, using PMOS devices.

The total DAC output conductance is just the conductance of an individual current source multiplied by the number of elements connected to the output, so it is signal dependent:.

$$g_{DAC}(n) = n \cdot g_{out} \tag{4.2}$$

When the DAC is used to drive a resistive load, the output resistance causes an absolute error in the transfer function, related to the relative difference between the signal current going to the load and the signal leaked into the DAC through the finite output conductance. The voltage output and voltage error may be written:

$$V_{DAC}(n) = n \cdot I_{unit} \cdot R_{load} \cdot \frac{1}{1 + R_{load} \cdot g_{out} \cdot n} \tag{4.3}$$

$$V_{DAC}(n) \approx n \cdot I_{unit} \cdot R_{load} \cdot (1 - R_{load} \cdot g_{out} \cdot n) \tag{4.4}$$

$$\text{VoltageError}(n) = n^2 \cdot I_{unit} \cdot (R_{load})^2 \cdot g_{out} \tag{4.5}$$

This error results in a gain error and in bow non-linearity. The worst case non-linearity is at the midpoint, n=N/2, and the gain error must be subtracted out before INL error is computed. Resulting gain error and INL are:

$$\text{Absolute Gain Error} = \text{Error(N)} = N^2 \cdot I_{unit} \cdot (R_{load})^2 \cdot g_{out} = V_{FS} \cdot R_{load} \cdot N \cdot g_{out}$$

$$\text{Gain Error (in lsb)} = N^2 \cdot R_{load} \cdot g_{out} \tag{4.6}$$

$$\text{Midpoint INL (in lsb)} = \text{Error(N/2)} - (\text{Gain Error} / 2) = \frac{N^2}{4} \cdot R_{load} \cdot g_{out} \tag{4.7}$$

4.3.6.2 Computing INL due to Threshold Voltage Mismatch

Random variation in device threshold voltage (V_t) results in a random variation in current cell current, which leads to variation in INL. The model for random variation is the same as Pelgrom's [PELG89]. Given the technology dependent mismatch constant S_{Vt} and device area, the variance in V_t can be computed for every device:

$$\sigma_{Vt}^2 = \frac{(S_{Vt})^2}{(W \cdot L)} \tag{4.8}$$

After a DC simulation is run, the small signal conductances are known, and the linear relationship between V_t and output current can be used to compute σ_I. Contributions from both transistors in the current source are considered:

$$\sigma_I^2 = \left(\sigma_{Vt}(M1) \cdot \left(\frac{g_{m1} \cdot g_{ds2}}{g_{m1} + g_{ds2}}\right)\right)^2 + \left(\sigma_{Vt}(M2) \cdot \left(\frac{g_{m1} \cdot g_{m2}}{g_{m1} + g_{m2}}\right)\right)^2 \tag{4.9}$$

The σ_I / I_{unit} ratio is substituted into Eq. 2.15 to obtain the σ_{INL} from this effect. The magnitude of all individual INL contributions are added to find the worst case total INL. A three-sigma bound is used to ensure a 99% yield.

4.4 Design Selection by Optimization

For design selection, optimization approaches are chosen to allow a full search of the design space. The optimization algorithm must minimize an objective, while meeting all performance constraints. The algorithm must converge to integer results, and be tolerant of poor performance estimation for infeasible results. This is a mixed-integer non-linear programming (MINLP) problem. The optimization must converge to a solution in a few hours running time, using a performance estimation step which requires on the order of 20 to 90 seconds of CPU time per estimation. This constrained non-linear programming (NLP) optimization problem is set up:

Minimize the objective: $f(\mathbf{x})$

Subject to constraints: $\mathbf{g(x)} \leq \mathbf{b}$

where x is the vector of design variables, g(x) is the vector of constraint functions, and b is the vector of bounds on the constraints.

Gradient based approaches have been used for circuit optimizations in several previous works. In these approaches the gradients of the objective and constraint functions are found at the local point, a direction for a move is chosen, and the algorithm moves to the next point. The simplest algorithms may converge to a local minima instead of the global minima, but many include heuristics to avoid local minima.

One option for constrained optimization is to build a weighted cost function which incorporates the individual performance constraints and the design objective into one function, and then use the optimization to minimize the single cost function [BRAY81,JUSU93, KOH90, META93]. Though the optimization algorithm may minimize the cost function, there is no guarantee that all constraints will be met when the function is minimized. A dynamic re-weighting is required if the problem is to be forced to meet all constraints. Since the constraint and objective functions are combined into one measure, the optimization algorithms may over-design a constraint to minimize the global cost function, instead of minimizing the desired objective function.

Simulated Annealing is commonly used in problems in which there are many local minima which prevent a direct path to a global optimum. It optimizes a single cost function, and gradually converges to a final solution. The weighted cost function requires re-weighting to force the algorithm to move toward solutions which meet all constraints[GIEL90, OCHO94]. More importantly, simulated annealing is a slowly converging algorithm, requiring hours to converge when the cost function is quickly computed. In this implementation, with a slow simulation step, the application of simulated annealing is wholly inappropriate.

It is better to use an optimization algorithm which meets constraints directly while optimizing the objective function. General constrained optimization packages such as MINOS [MURT87] and NPSOL [GILL86] can do this, and these work well with analytic objective and constraint functions [MAUL93]. When used with circuit simulation these packages have two problems. The algorithms are written to deal with complicated multi-variate optimizations, with perhaps hundreds of variables and constraints, but mostly linear constraints, or non-linear constraints that are quickly com-

puted. In circuit simulation most constraint functions are non-linear, and require a relatively long time to evaluate, but there are relatively few variables and constraints. There is a poor match between the capabilities of these packages and the needs for this problem. A second issue is the problem of poorly defined function values for infeasible regions. These packages require reasonable gradients everywhere in order to work -- a difficult requirement for a simulation based analysis. In the course of this work optimizations using MINOS and circuit simulations have been run, but these optimizations have not reliably converged. By comparison, using MINOS with the set of analytic equations developed for OPASYN has resulted in a robust constrained optimization.

The feasible directions algorithm is a particular constrained optimization algorithm which has been applied successfully to circuit simulations [BRAY81, NYE88]. It requires an initial point in the feasible region, and then does not allow moves out of the feasible region. In DELIGHT [NYE88] a three step approach for simulation based optimization is implemented, with the first two steps forcing moves toward and into the feasible region, and the last step an optimization only in the feasible region.

While feasible directions does solve the constrained optimization problem for circuit simulation, it does not solve the mixed integer problem. The easiest way to obtain an integer result is to round the final point in the optimization to the nearest mixed-integer solution. There is no guarantee that this will be the optimal point, or even a feasible point, but if a simple weighted cost function is used then meeting constraints may not have been guaranteed anyway. In this case the solution is not qualitatively worse than the original real-valued solution (!). This is the method used in the optimization algorithm built into the HSPICE circuit simulator, but because a feasible solution cannot be assured it was not considered in here. If the MINLP problem is to be truly solved, there are two types of algorithms in the literature.

A direct approach to the solution of the MINLP problem is taken with the branch-and-bound algorithm [GUPT80]. The branch-and-bound algorithm runs multiple constrained optimizations, adding additional bounding constraints which force the solution to the optimal set of integer design variables which meet the specification. In the course of solving the MINLP problem, it must find the solution to many NLP sub-problems. This has been reported previously in an analytic constrained optimization, in which the integer variables were used to make an architecture selection in

an opamp circuit [MAUL93]. This algorithm works well when the underlying optimization takes only a few seconds to a few minutes. When applied on top of simulation based circuit optimizations which already require an hour or more of compute time, the problem explodes, and is not solvable in the few hours time allowed for DSYN.

The second approach to the MINLP problem creates an approximation to the constraint functions with a set of linear constraints, and uses this approximation to specify a mixed integer linear programming (MILP) sub-problem. This approach is taken in the Outer Approximation[DURA84] and Generalized Bender's Decomposition[GEOF72] solution methods. The MILP problem is relatively easy to solve, using the branch and bound algorithm and an LP solver. Once a solution to the MILP problem is found, it is checked for feasibility with the non-linear constraints. If the MILP solution is infeasible, then the algorithm specifies a method for creating additional linear constraints from the infeasible solution. For example, Duran specifies solving a non-linear programming (NLP) problem to create outer bounds for the constraint space. This approach is faster than the direct approach because it separates the search for an integer solution from the evaluation of non-linear constraint and objective functions. It also places additional convexity requirements on the constraint functions. Fig. 4.3 shows the general approach.

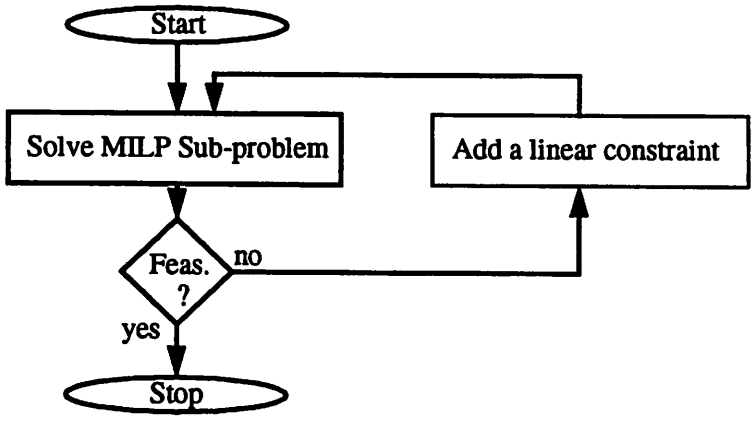


Figure 4.3 Mixed Integer Nonlinear Programming problem is solved with a Mixed Integer Linear Programming problem and a means for creating linear constraints.

The algorithm used in DSYN takes this second approach, separating the search for the mixed integer solution from the evaluation of non-linear constraint functions. The algorithm is a hybrid

of the Outer Approximation[DURA84] and Supporting Hyperplane[LUEN84] methods. The problem is separated into an MILP sub-problem and a method for obtaining outer approximations to the non-linear constraints, and the Supporting Hyperplane method is used to obtain those approximations.

4.4.1 Supporting Hyperplane Algorithm for Optimization

This section describes the implementation of the particular cutting plane algorithm used in DSYN -- the supporting hyperplane algorithm [LUEN84]. It includes a general description of the algorithm, some application specific issues for DSYN, and some general properties of the algorithm The key modification to the algorithm described by Luenberger is the solving of the sub-problem as an MILP problem instead of an LP problem.

The supporting hyperplane method takes the form:

minimize the objective: $\mathbf{c}^T\mathbf{x}$

subject to $\mathbf{g(x)} \leq \mathbf{b}$

where x has dimension n, and g(x) has dimension p, the g_i's are continuously differentiable, and the constraint region S defined by the inequalities is convex. A feasible point y must be known, such that $g_i(y) \leq b_i$ for all i.

Start with an initial space P containing S, such that $\mathbf{c}^T\mathbf{x}$ is bounded below on S. Then

- Step 1: Determine w = x to minimize the objective function over P. If w is in S, then stop. Otherwise continue.
- Step 2: Find the point u on the line joining y and w that lies on the boundary of S. Let i be an index for which $g_i(u) = b_i$, and define the half space $H = \{x: \nabla g_i(u)(x - u) \leq b_i\}$. Update P by intersecting with H. Return to Step 1.

The process is illustrated in Fig. 4.4. First the optimal point w_1 is found in the space P. Since this is not in the feasible space S, the algorithm retreats back towards the feasible point y until it finds the boundary of S at u_1. Taking the gradient of $g_i(u_1)$ creates a linear constraint which bounds the half space H_1, as described above. On the second iteration the optimal point in H_1 is found at w_2, from which the feasible point u_2 is located, and the gradient of the active constraint

$g_i(u_2)$ is defines an additional half-plane H_2. The next point w_3 is the optimal point which lies in both H_1 and H_2. This process continues with the addition of more linear constraints until the sub-problem solution w_i is a feasible point.

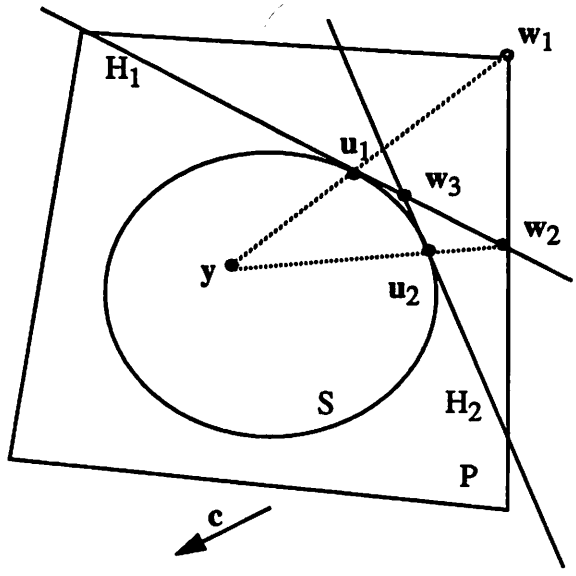


Figure 4.4 Supporting Hyperplane algorithm creates bounds on feasible region.

4.4.2 Algorithm Implementation and Discussion

This section begins by describing the optimization algorithm implementation in detail, and then continues with a discussion of some practical optimization issues related to the implementation. These include a prediction of the complexity and running time of the algorithm, the relaxation of accuracy requirements for simulations giving infeasible results, the speed-up of convergence through the use of the MILP sub-problem solver, the search for the required initial feasible point, the use of a linear approximation for the objective function, and the optimality of the final solution.

4.4.2.1 Implementation

The algorithm is implemented using the simulation/analysis method described above to estimate all objective and constraint functions. In this implementation the sensitivities of these functions to design variables are not computed explicitly, but finite differences are used to obtain these sensitivities for gradient computations. This increases the optimization time, but simplifies the construction of the design estimator, saving design implementation time.

The algorithm is given an initial feasible point y. In this implementation the first step is to approximate the nonlinear objective function with a linear function, found by computing the gradient at the initial feasible point.

$$\mathbf{c} = \nabla \mathbf{f}(\mathbf{y}) \tag{4.10}$$

Next the sub-problem must be solved. This problem is an MILP problem, easily solved by the branch-and-bound algorithm implemented with MINOS [MURT87] used as a linear programming solver. On the timescale of the total optimization, the MINOS optimizations take zero time, so even though the branch and bound algorithm is not efficient, and may require hundreds of calls to the MINOS package, the total job time is not significantly changed by the time spent solving the MILP sub-problem.

The solution to the subproblem is the design point $\mathbf{w}_1$, which is tested for feasibility by the estimator. If it is infeasible, then the point $\mathbf{u}_1$, at the boundary of S, is found by conducting a binary search along the line between $\mathbf{w}_1$ and y, The linesearch runs a performance estimation at each point in the search, and stops when the change in all design variables is less than a user defined tolerance. The result of the linesearch is actually two points, $\mathbf{u}_{feas}$ and $\mathbf{u}_{inf}$, on opposite sides of the boundary to the feasible region S, and the knowledge of the particular $g_i(\mathbf{x})$ constraint function which becomes infeasible across the boundary. The new linear constraint must exclude $\mathbf{u}_{inf}$, but not $\mathbf{u}_{feas}$. If the bound on the constraint function is g_{i0}, then, using the linear term of the taylor expansion the new constraint may be written:

$$\{\mathbf{x}: \nabla g_i(\mathbf{u}_{inf})^T \mathbf{x} \le \nabla g_i(\mathbf{u}_{inf})^T \mathbf{u}_{inf} + g_{i0} - g_i(\mathbf{u}_{inf})\} \tag{4.11}$$

If the last two terms on the right side of Eq. 4.11 are left out, then the new constraint will not exclude $\mathbf{u}_{inf}$, and the algorithm is prone to entering infinite loops, returning to $\mathbf{u}_{inf}$ on every iteration. Note that the right hand side is a constraint, so this may be rewritten as a simple linear constraint of the form:

$$\{\mathbf{x}: \mathbf{c}^T \mathbf{x} \le \mathbf{b}\} \tag{4.12}$$

The new linear constraint is added to the sub-problem, and the process is repeated. When the w_i is feasible, then this final w_i is returned as the solution for this linear objective function.

The linear objective function used in the sub-problem is an approximation to the objective, made at the initial feasible point. If the gradient of the objective changes across the feasible region, then w_i may not be the optimal solution. In this implementation the heuristic is used to overcome this. The linear objective function is re-computed at the solution point, w_i, and the optimization restarted, using the existing linear constraints, and w_i as the new feasible point. This process of restarting the optimization with a new feasible point continues until the result of the optimization is the same as the initial point.

4.4.2.2 Algorithm Running Time and Simulation Requirements

A prediction of algorithm running time and design simulation requirements can be made by tracing the algorithm through one iteration. Assume that the simulation has already been running for some time, so a feasible point y is known, a linearized objective function is known, and some of the linear half-plane constraints have been defined. In tracking algorithm time a good assumption is that simulation time is the dominant factor, so counting the number of simulations gives a good measure of the total optimization time. The algorithm starts by solving the linear sub-problem for point w_i, using a branch and bound algorithm to solve the MILP problem. Since the simulation is not required, this may be considered a zero time step. This w_i is tested for feasibility, and if it is infeasible, a linesearch proceeds along the path between w_i and y. A simulation is required at each step in the linesearch, to check feasibility. There is assumed to be only one crossing point, so a simple binary search is used. The binary search is stopped when the difference in design variables is less than some ζ, and the number of simulations required for the binary search is found:

define maxdelt = MAX(y_j - w_j) over all j.

$$\text{Number of simulations per linesearch} = \log_2\left(\frac{\text{maxdelt}}{\zeta}\right)$$

Typically this is between 7 and 10 simulations. In these simulations the results are checked for only for feasibility, so accurate values for estimated results are not required in the infeasible region. At the end of this search the point u_{inf} has been determined, and each variable is perturbed

in turn to compute the gradient at u_{inf} by finite differences. This does require accurate simulation results, but since it is the edge of the feasible region, this is a reasonable requirement. If there are N design variables, then this requires N simulations. A new linear constraint is created, and the algorithm repeats. This requires a total of $\log_2(\frac{\text{maxdelt}}{\zeta}) + N$ simulations per iteration.

The requirement of integer results actually reduces the number of iterations, by limiting the number of possible feasible points. In this work the largest problems have required about 20 iterations, for a total optimization time (assuming 1 minute per simulation) of 6-7 hours. When this algorithm is used to solve problems with real valued design variables many more iterations may be required, and other algorithms such as feasible directions may be a better choice.

One other property of this algorithm related to running time is that the creation of a set of linear constraints creates a memory for the algorithm. If the optimization is stopped for some reason, but the linear constraints are saved, then restarting the algorithm with these linear constraints as a starting point saves optimization time.

4.4.2.3 Initial Feasible Point

In practice it is not difficult to supply an initial feasible point for the optimization algorithm. The feasible point does not need to have integer variables, or a minimized objective. If an automated method is desired, a constrained optimization algorithm may be applied, with no objective specified. This has been implemented for DSYN. If the algorithm does not converge to a feasible point, a designer may proceed using design specific heuristics to find a feasible point.

4.4.2.4 Approximating the Objective Function

In this implementation the objective function is evaluated through calls to the circuit simulation step. This is not a necessity, since the objective function is a purely analytic function of the design inputs, but the DSYN architecture puts all design information, for both the objective and constraint functions, in the same HSPICE based framework, to unify design entry. To avoid slow objective evaluations, the objective function is replaced with a linear approximation, found by evaluating its gradient at the best feasible point found so far in the optimization. This approximation has worked when the direction of the gradient does not change much across the design space.

When this was not true, as in a case with an objective function of the form $f_{obj} = c \cdot \sqrt{x^2 + y^2}$, the algorithm will ping-pong between feasible solutions, but not find the optimal. If a nonlinear objective function is required there are two possible solutions. A higher order approximation to the objective function may be taken, but if all second order terms are to be included in the approximation, then N^2 function evaluations are required, instead of just N. A second approach is to input the analytic objective function into the optimization program, so that no approximations are required. Though the sub-problem optimization is made more complicated with a nonlinear objective function, this is still a no-cost step compared to function evaluations with simulation. In the DAC optimizations implemented for this work the strategy used was to avoid non-linear objective functions, so that linear approximations to objective functions proved satisfactory.

4.4.2.5 Optimality

Luenberger and Duran both discuss the requirements for optimal solutions with this algorithm [LUEN84,DURA84]. The key requirement is that the linearized constraints must not over-constrain the solution space, and this will be true if the constraint functions are convex. Because the linear approximations are only computed in the feasible space, this requirement may be relaxed slightly, to requiring a convex space in the region where the functions are feasible. When the space is not convex, linear constraints made early in the optimization may prevent finding the optimal solution.

4.4.3 Optimization Implementation

All simulation based optimization algorithms are implemented in a single program, SpiceOptim, with the optimization algorithm chosen by user input. The program operates as an optimization manager, controlling user file I/O, the simulation interface, and the optimization algorithm interface. All algorithms use HSPICE simulation runs to compute objective and constraint functions, so the program is design independent. The user inputs a lists of design variables, design constraints, design constants, and any optimization algorithm parameters. The program creates a list of parameters for each HSPICE run, executes the HSPICE run, and reads results of.MEASURE statements directly from the HSPICE list output. The optimization algorithms supported are:

- a general constrained optimization using MINOS
- a combination of branch and bound integer programming and MINOS
- general constrained optimization without an objective, used to find a feasible point

- supporting hyperplane algorithm without integer programming
- supporting hyperplane algorithm with integer programming
- supporting hyperplane algorithm using a non-linear approximation to the objective function

For all algorithms it is possible to read in information from previous runs, for restarting an optimization upon an unexpected halt. Fig. 4.5 illustrates the inputs and capability of SpiceOptim.

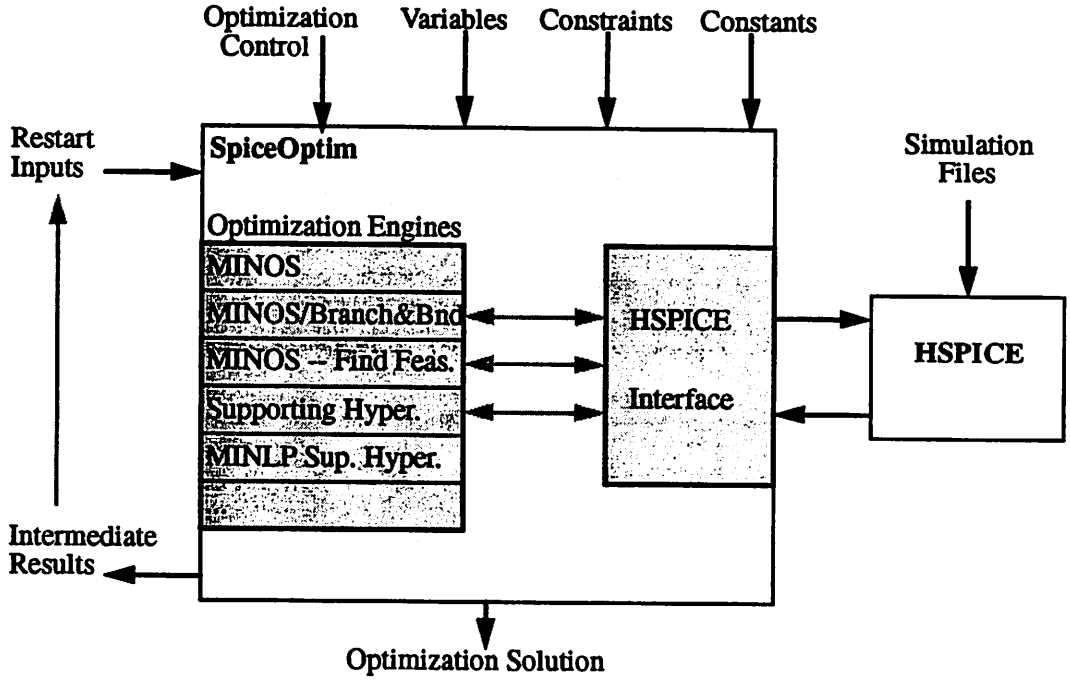


Figure 4.5 SpiceOptim program connects optimization algorithms to HSPICE simulation.

A shell script, optScript, has been written which runs the most commonly used combination of runs -- an optimization to find a feasible point, a supporting hyperplane run with integer programming, and, as a heuristic to check convergence, a second supporting hyperplane optimization starting from the previous solution. Fig. 4.6 illustrates the optimization flow for optScript.

4.5 Hierarchy in Estimation and Selection

The approach taken in this work is to avoid hierarchy if possible while searching for a design solution, because the imposition of a design hierarchy at a low design level imposes limitations on

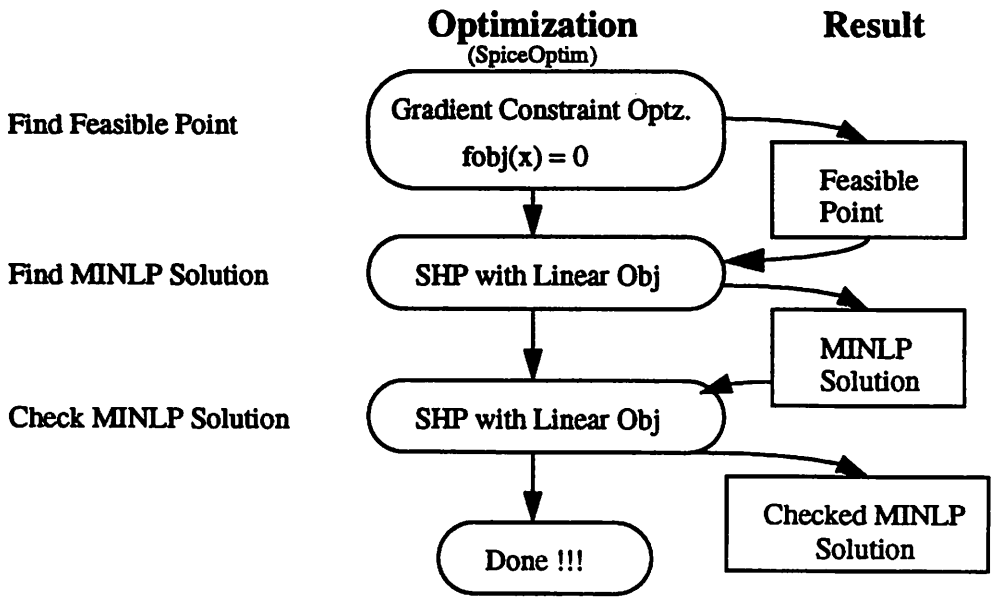


Figure 4.6 Optimization flow implemented in optScript.

design choices. The emphasis has been to include both high level DAC architecture and low level device sizes in the same optimization, so that the two areas for design choices may be optimized simultaneously. In the first circuits chosen for this work the complete current output DAC design selection step was done as a single optimization. This was possible because those particular circuits were relatively simple, and the number of independent design variables was kept to 7 through the use of a highly parametrized model for digital buffer sizing. In the second set of prototypes the use of a more complicated bias and current source cell, and the addition of more digital buffers in the DAC layout resulted in additional design variables. It no longer was practical, or even sensible, to consider a single optimization. Instead of creating a design hierarchy, the choice was made to split the optimization into separate digital and analog optimizations. In the "analog" optimization, the key DAC performance objectives are used as constraints, and 13 independent design variables have an impact here. The "digital" optimization, with 6 independent design variables, is for the digital row and column buffer circuits, and local decodes, which operate on the clock phase before the data is actually transferred to the output. The constraints for this optimization are determined from the requirement that valid data arrive in time for the cell level latch. These optimizations are cascaded, since the "digital" simulation may only be run after the loading due to the optimized "analog" circuits is known. More detail of these implementations is found in chapter 6.

4.6 Summary

In this chapter the design estimation and selection problems for DAC synthesis have been reviewed, and the approaches taken in DSYN have been described.

The estimation approach uses a combination of circuit simulation and analytic equations to obtain accurate predictions of circuit and DAC level performance. The entire process is implemented in a single HSPICE simulation, using built in features for parametric analysis and performance measurement. Estimation includes provisions for inaccurate device models and worst case analysis for process variation. The estimation step usually takes less than 1 minute on the fastest workstation available.

The design selection step uses a combination of integer programming with the supporting hyperplane algorithm. This method successfully minimizes the objective while meeting constraints, subject to the vagaries of optimization using circuit simulation. Typical optimizations take up to 6 hours of compute time. The optimization problem includes all DAC design variables affecting analog performance in a single optimization.

CHAPTER 5

Layout Synthesis for DACS

5.1 Introduction

Layout synthesis is the second important step in module synthesis. Automated layout synthesis is used to create the desired layout from the outputs of the circuit synthesis step. For DSYN there are two goals for this layout synthesis -- a compact circuit area, and method for inclusion of accurate layout parasitics in circuit synthesis. The circuit area should be comparable to custom DAC layouts, or else the results will be seen as irrelevant by potential users of these tools. Minimizing DAC area also improves static linearity, because it reduces the absolute mismatch of devices caused by process gradients. As seen in chapter 3, the circuit synthesis process used in DSYN requires an accurate estimate of layout parasitics for inclusion in circuit optimization, so the layout process must provide this.

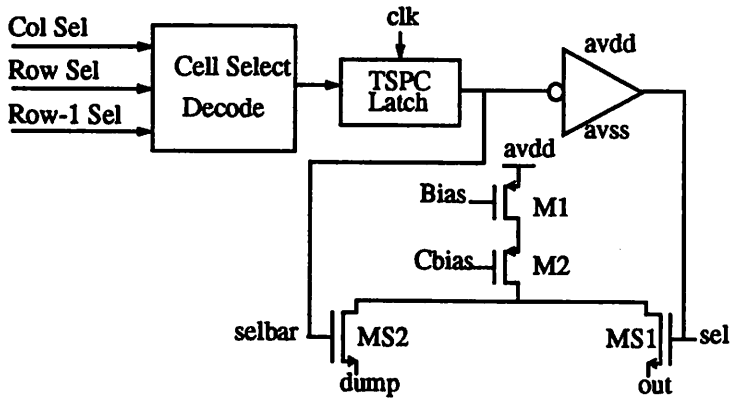
Most previous analog layout synthesis tools have been developed for opamp and comparator circuits. These circuits have several design qualities which have influenced the approaches taken for analog layout synthesis. Opamps and comparators are primarily differential circuits, with circuit performance affected by matching of pairs of devices, or device parasitics in the differential structures. There are several heterogeneous sub-sections to the circuit, such as current mirrors, differential pairs, or current sources. The devices tend to be large, so circuit area is dominated by

device area, rather than wiring. Wiring parasitics are important, but wiring complexity is usually not considered an important problem. In the solution to analog layout synthesis, the emphasis has been on device placement, considering matching issues and merging of MOS source and drain diffusions to minimize area and parasitics. Routing algorithms which enforce differential matching, capacitive, and resistive constraints are employed.

In DAC circuits a different set of design qualities are present, which limits the application of existing analog layout synthesis approaches. Consider both the generic DAC issues, and then the specific issues for the segmented, current switched architecture.

For all DAC layouts, there is a set of identical devices which must be well matched, including matched nominal resistance, capacitance, or device dimensions, and matched parasitics. Devices are placed in an array configuration, and regular placement in the array is needed for cancellation of process gradient effects, and to maximize matching. DAC circuits are less complicated than opamps, with typically only two functional blocks -- a matched element and a means for connecting it to the output. For example, current switched implementations require current sources and switches, resistor strings require matched resistors and analog switches, and capacitive structures require capacitors and switches.

Because of the regular, cellular nature of current-switched DACs, consisting of many similar segments, most manual DAC layouts are implemented with tightly packed abutting cells so as to minimize the interconnection area. For a current switched DAC implemented in DSYN, consider the layout at the segment current source cell level and the DAC module level. The DAC implementation described in section 6.2 is used an example here. At the cell level, the key elements are a digital decode circuit, an optional digital latch, a digital inverter, a current steering switch, and the matched current source for the segment. This is a mix of analog and digital circuit functions, and there are more devices devoted to the digital functions than to analog ones. All digital functions have minimum device lengths, but design specified widths, while analog current source devices have both design specified widths and lengths. When the latches are included, there are 22 digital devices, and only 4 analog devices, although the analog devices may consist of multiple unit element devices. Fig. 5.1 shows a simplified schematic for this DAC cell, including a list of the number of transistors in each module. DAC performance is influenced by parasitic coupling



Circuit	# of Devices
Decode	8
Latch	12
Inverter	2
I-src	2
Switch	2

Figure 5.1 Simplified schematic for a switched Current Source DAC segment cell.

from digital signals to bias and output, and by coupling from analog output to bias. Resistive drops, particularly in supply and bias lines may be important for static linearity. To minimize layout area, the key problem at the cell level is digital functions, with relatively small devices, and complex wiring. If this section is done inefficiently then the layout may be too large.

At the DAC module level the layout consists of many identical cells which are to be placed in a regular structure. If inter-cell wiring can be accomplished across the cells, rather than around them, and cells can be abutted, then this elimination of routing channels results in a significant area savings.

This chapter continues with a review of both analog and digital circuit synthesis methods applicable to this problem, a description of the synthesis approach taken for DSYN, and a discussion of the advantages and disadvantages of that approach.

5.2 Layout Synthesis Approaches

In chapter 1 included a general review of analog layout synthesis, and there it was indicated that the best candidates for general analog layout synthesis used a simulated annealing algorithm for placement, and area routing [COHN91,CHAR94]. In this section these approaches will be considered, but also some approaches previously used on digital circuits will be considered for application to DAC layout synthesis.

5.2.1 General Analog Place and Route

This analog layout synthesis approach starts with a circuit netlist and some analog layout rules. In KOAN/ANAGRAM [COHN91] these include device matching and parasitic minimization rules, used to drive the placement selection part of the Simulated Annealing algorithm. The UCB tools use this approach, and include performance driven constraint of parasitics [CHAR94]. The placement step takes advantage of opportunities to merge diffusions to reduce layout areas. Analog routing tools also match and limit parasitics, Analog routing typically limits resistive parasitics by using metal wiring layers only. The programs may take several hours to run, and the layout is not deterministic. When used with synthesis, the synthesis algorithm must allocate some performance degradation to the layout process.

This approach works well with opamp and comparator circuits, with the characteristics described in the introduction to this chapter. The use of layout rules for device matching and performance driven parasitic constraints yields circuits which meet performance objectives when the layout is extracted and simulated. When layout area is dominated by MOS device sizes the area penalty for automated synthesis is not significant.

Unfortunately these approaches do not fit the DAC synthesis problem, particularly for the implementation used in DSYN. For digital decodes, latches, and analog switch circuits these layout synthesis algorithms work poorly, because the device sizes in these circuits may be a small part of the whole, and circuit area is a strong function of the ability to solve interconnect problems. This is the wrong problem for these solutions. A manual design may also take advantage of polysilicon wiring for interconnect, an opportunity missed by the automated programs. The fundamental differences between these circuit types make these tools a poor choice for DAC cell layout synthesis.[1]

There are other less important implementation dependent disadvantages. At present there is no good way to automatically route across a cell with these tools, or ensure routing between cells with abutment, but this is more an implementation issue than an algorithmic one. Also, the slow layout

1. Automatic synthesis of digital logic circuits is not easy by itself. Most standard cell libraries employ manually designed cells, or an automated technology shrink of a manually designed cell. There is some recent work in layout synthesis for logic cells [CHEN89].

step without predictable parasitics is not compatible with DSYN as it stands, but the ability to incorporate performance constraints in layout synthesis is another effective technique for synthesis to specifications.

5.2.2 Layout Synthesis for Digital Circuit Modules

Though the outputs of DAC circuits are analog signals, the large number of digital signals in a DAC, and the obvious similarity of high speed DAC layouts to digital ROM and RAM modules suggests looking toward digital module generators for solutions to these layout problems.

Tiling of subcells is a commonly used technique for module generation of ROM, FSM, and RAM blocks [NEFF87]. In these layouts subcells are placed with connections made by abutment. Though the complexity of the module scales with the number of rows or columns in the structure, the subcells are the same for all module specs. Depending on the module specification, the number of rows or columns, the address space, or the ROM contents may be set by the choice of cell tiled at each location. The individual tiles include ROM or RAM cells, row drivers, column drivers, sense amplifiers, or output latches and buffers. These are usually manually constructed cells. Reducing the area of individual cells with tight manual layouts has a large payoff in reduced module area and parasitics.

DAC layouts may be viewed in a similar way. The DAC module may be constructed with an array dominated by DAC segment cells. The complexity of the DAC segment cell does not scale with DAC size, so the same cell structure may be used for 6-bit to 14-bit module designs. Also, layout methods which result in compact DAC segment cells will give big payoffs at the module level.

Device or cell stretching is a second device customization technique found in digital module synthesis and custom device generation. In the CADENCE tool set parametrized cells (pcells) use automated stretch operations to correctly size transistors from a template [CADE94]. Modification of subcells by stretching across a user defined plane allows customization of more complicated cells. Stretching has been used at the module level to customize datapath cells, creating more cell space for routing when a datapath compiler required it [TSUJ94], and in standard cell libraries to size buffers on the fly. Using stretching operations to modify manually designed cells mixes the

qualities of fixed template automated methods with manual design. As in manual design, compact layouts may be created by exploiting the polysilicon and diffusion as interconnect layers. The stretching creates a customized cell with desired device or cell dimensions. Because this technique is a fixed template approach, variations in element sizes may result in white space at the cell level. Algorithms for shape optimization have been implemented for template based synthesis [KOH89], but for simple stretching operations in a multi-device cell, some white space will accumulate.

In DSYN, layout synthesis is implemented by using cell stretching to customize template library cells, and tiling the subcells to create a DAC module. The approach starts with a library of cells. The correct library cells are chosen, based on the module architecture, and customized by stretching according to input device dimensions. The customized cells are then tiled to create the DAC module. Interconnects between cells are made by abutment. This method was found to meet the two most important requirements for DSYN. Module area was comparable to manual layouts, and circuit parasitics could be predicted a priori, and these predictions used within the design synthesis optimization process.

The following sections describes the synthesis implementation in detail, and discusses advantages, disadvantages, and synthesis results for CMOS current switched DAC modules.

5.3 DAC Layout Synthesis with Cell Stretching and Tiling

Fig. 5.2 outlines the layout synthesis process, including layout inputs, process, and outputs. This section will cover these layout inputs, the algorithm used for layout, and the programs created for that implementation.

5.3.1 Inputs

There are two user inputs for layout synthesis, a cell library and the optimization results from circuit synthesis. The library consists of subcells for every part of the DAC module, including DAC segments, row and column drivers, bias generation, analog buses, LSB cells, and spacer cells needed to fill out the tiled array. The cell library includes specifications for cell stretching, indicating where a cell should be stretched for each input device dimension. The stretch annotation includes an indication of the default dimension, so when a minimum device size is input no stretch

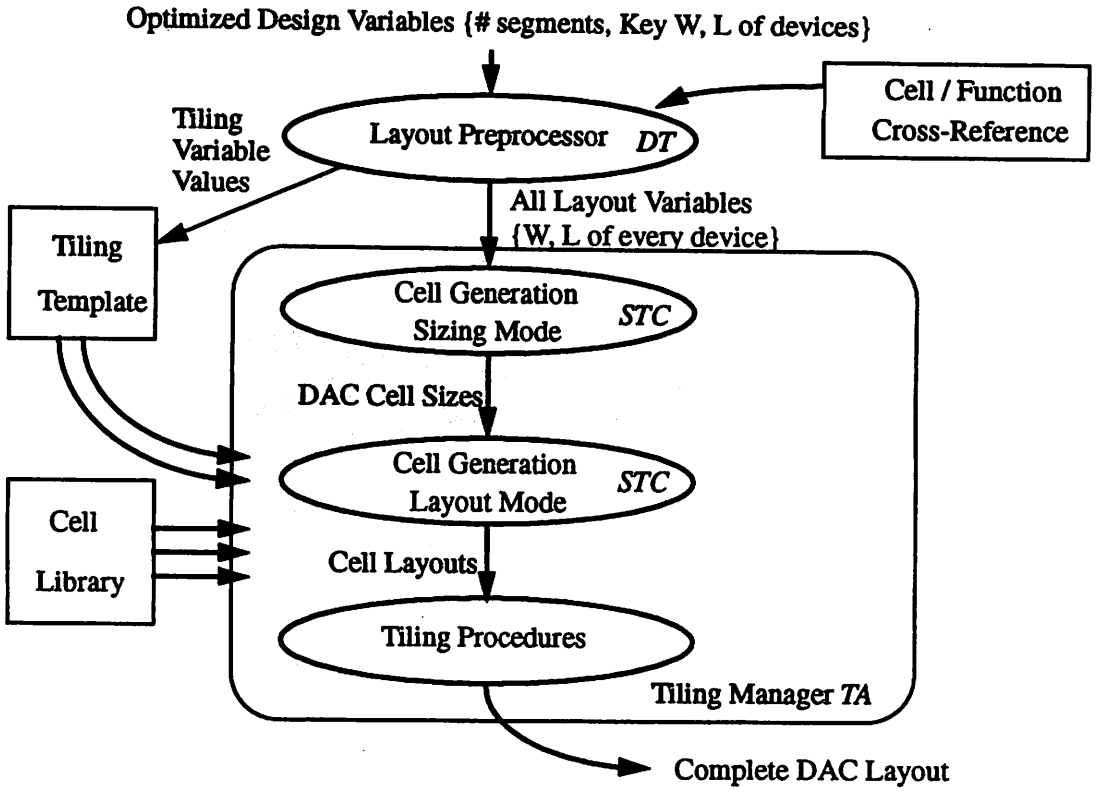


Figure 5.2 DSYN layout synthesis process. For key elements are the cell library, the Layout Preprocessor (*DT*), the Stretching program (*STC*), and the Tiling Manager (*TA*).

is made. It also includes special horizontal and vertical stretching locations, which are used to further stretch the cell to the desired height or width. These height and width stretches are used to rectangularize all cells, and force all cells in a row to the same height. The cell designer is responsible for making certain that interconnections between cells will be made by abutment, and abutment or stretching operations will not result in design rule violations. Along with the library, the user must provide a cross-reference file for the tiling program, which associates library cell names with module layout cell types.

The design input from the DAC circuit synthesis step may be separated into two parts. The device and element sizes are used to control the cell stretching operations, resulting in desired device dimensions. The high level DAC parameters, such as number of rows, columns, and lsb per segment, are used to determine the number and types of cells used when the DAC is tiled.

5.3.2 Algorithm

From these user inputs a stretching and tiling algorithm creates the module layout. The first step is the creation of a tiling template for the DAC module. Using the cross reference input and the module architecture design inputs, a DAC architecture-specific program creates an assignment of library cells for every location in the DAC module array. The second step is an initial cell sizing, finding the minimum cell size for the user inputs. The maximum cell height in a row is used to determine the row height, and likewise for columns. Then all cells are assigned height and width dimensions to match their location in the array, and cells are created with these sizes. The DAC module is tiled from the set of created cells, and all connections between cells are made by abutment. A final step is to locate I/O terminals, and assign terminal names. In the algorithm there are two additional cell listing steps, used to limit the cell sizing and creation tasks to only the number of unique cells, rather than the total number of cells in the module.

5.3.3 Layout Synthesis Implementation

Layout synthesis is implemented with three programs. The tiling template is created by the *DT* (Dac Template) program. Cell stretching and sizing is done by the *STC* (STretch Cell) program, and row/column sizing, and cell placement of the tiled array is done with *TA* (Tile Array). The programs require a cell library implemented using the MAGIC layout editor [SCOT85], and create command scripts for MAGIC to implement all layout stretching and placement operations.

DT is a module specific template generator, which was developed for these DAC modules. It organizes the placement of cells into the correct number of rows and columns for the specified DAC design. The cross-reference file is used to map library cells to locations in the DAC module. The program is flexible enough that architecture modifications, such as elimination of row drivers or column drivers may be accommodated through the specification of empty cells in the cross-reference file.

STC operates in either of two modes. It takes design inputs and a specified library cell, and creates the stretched version of the cell. In sizing mode, it does not create the cell, but predicts cell size for this design input. In cell creation mode it generates the stretched version of the cell. The program allows both vertical and horizontal stretches, and keeps track of the interactions between

the two, and the shape of the cell after stretches have been implemented. It implements a final set of horizontal and vertical stretches to shape the cell into a rectangle.

TA organizes the tiling process, calling STC to do cell sizing, determining the row and column sizes, and then placing the cells. The cell listing steps described in the algorithm are used to minimize the number of calls to STC. Connections are made by cell abutment.

The STC and TA programs are generic, and could be used for any array type structure.

Fig. 5.3 illustrates a simple stretching example. This is an inverter cell, with stretches annotated for the NMOS and PMOS devices, as well as stretch locations for horizontal (HORIZ) and vertical (VERT) stretching. The annotations include the default dimensions of the devices affected by the stretches. In these MAGIC cell layouts the annotations are simple labels, and the CUT_ prefix is used to indicate their use in STC. The numeric suffix indicates the default dimension. Note

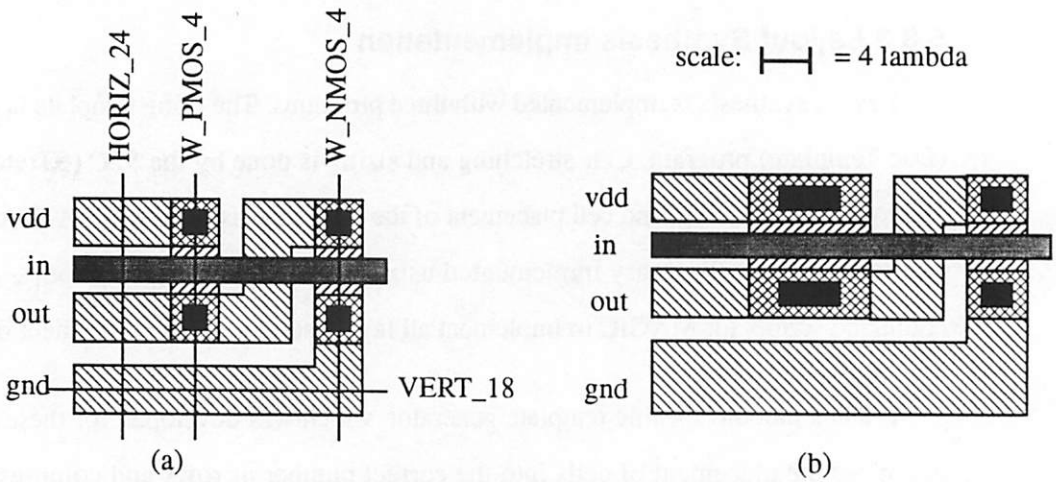


Figure 5.3 Simple cell stretch example. a) is the unstretched library cell. b) is after stretching with inputs W_NMOS=5, W_PMOS=10, and VERT=20.

that the default dimensions for the NMOS and PMOS devices are 4, the default height is 18, and default width is 24 in the figure. (All dimensions in this example are in lambda.) Fig. 5.3a is the library cell. Fig. 5.3b is the cell after stretching, with inputs as indicated. The devices have been stretched to the correct value, and the cell further stretched to obtain the correct overall height. of 20 lambda

Fig. 5.3 is a more complicated example, used for a DAC segment, including a digital decode, an inverter, switches, and current source devices. Note that all of the device widths may be modified, and current source device lengths may be modified also. The figure shows the library cell and the stretched, rectangularized version. Analog layout techniques, such as shielding of signal lines from digital switching lines on the left side of this cell, are done through hand layout of the library cells.

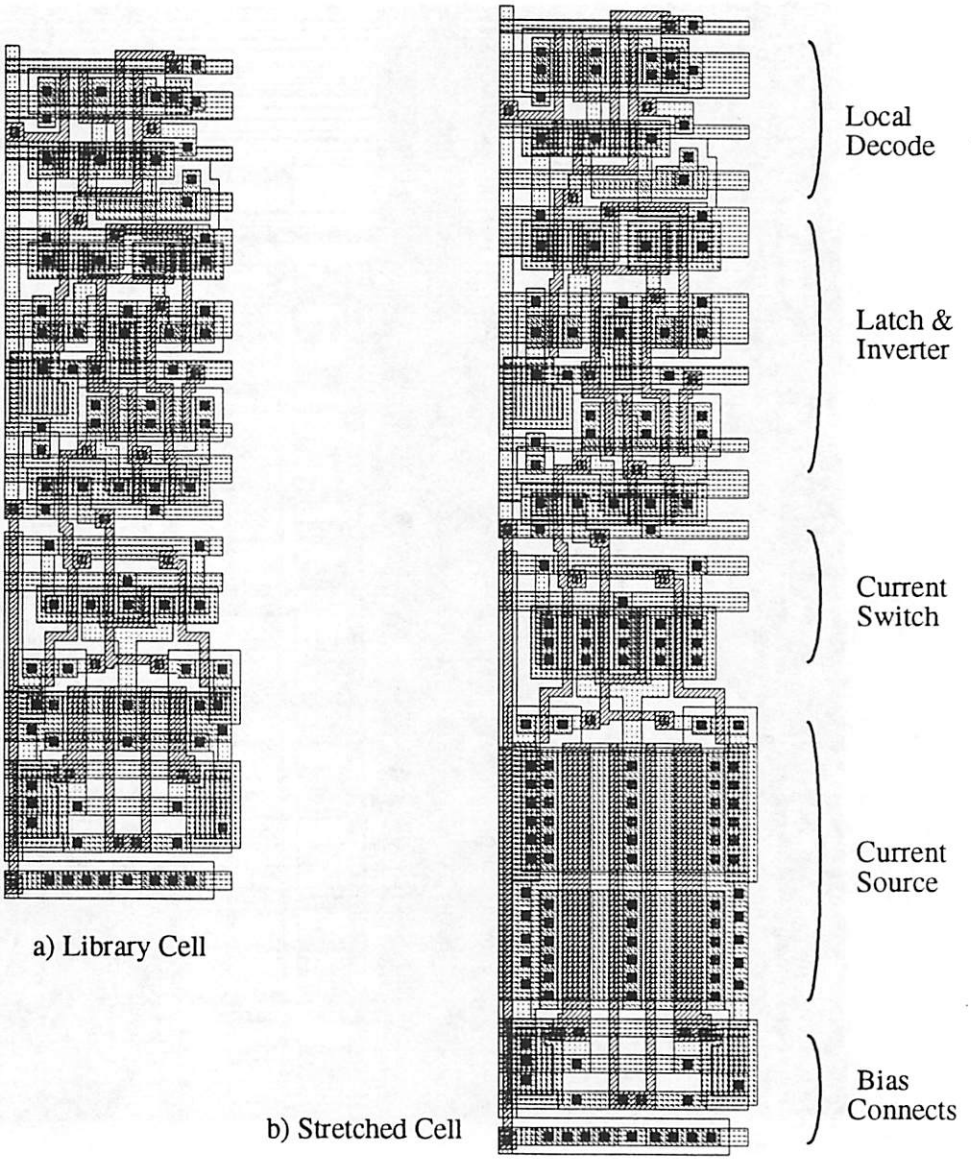


Figure 5.4 Examples of the library cell and its implementation for a DAC segment

Fig. 5.5 is an example of an 8-bit DAC module created from design specifications. The full design is described in chapter 6, but here it is used as an example of a tiled DAC module. The final design has 4 lsbs per segment, with 4 rows of 16 segments each. The second row is used for the 3 lsb segments. The boundary cells on the left, right, and top are used for row and column latch/ driver cells. Analog I/O uses the center cells, and bias generation cells are placed at 1/4 and 3/4 of the way across the rows.

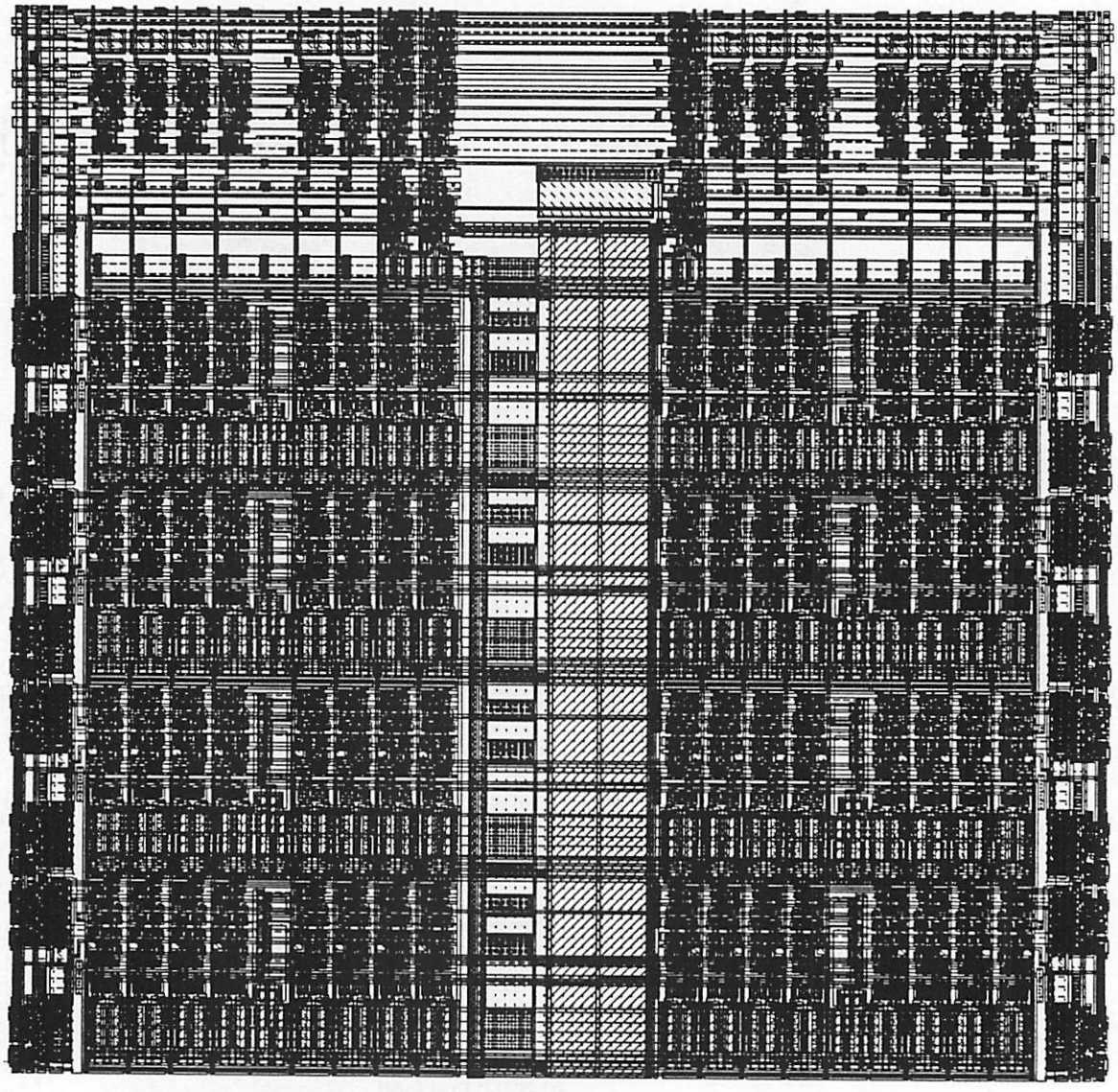


Figure 5.5 Tiled DAC module for an 8-bit example.

5.4 Layout Synthesis Conclusions

5.4.1 Disadvantages

The most important disadvantage for this layout synthesis method is the requirement of a manually designed cell library. When migrating to a new technology, with different design rules, this requires the creation of new cells, or at least a modification of existing cells. This disadvantage is alleviated by two factors. If scalable layout design rules are used [SCOT85], then the same library may be used across multiple technologies. This method has been used for digital cell libraries scaled from 3 mm to 0.8 mm line widths, but recent technology shrinks have tended to reduce channel lengths, but not interconnect pitches, so this seems less likely in the future. It may make sense to redesign cells to take advantage of additional interconnect layers as they appear in new technologies below 0.8 μm channel length. A second factor is that the required library cells are not complex, and there are only five library cells which actually contain transistors in the current switched DAC module. (Five others contain interconnect only.) Modifying the cell library for a new technology is a one time job which takes a few days time.

A second disadvantage relates to the use of stretched and tiled cell library for layout. This can result in white space in the layout for two reasons, due to white space in the cells, and empty cells in the module.

The use of a stretched library cell is equivalent to a fixed cell template, with no shape optimization. It is well known that if a fixed cell template is used, then some choices for device sizes may result in white space in the cell. In this implementation cell white space has not been significant because of the alignment of all device width stretches, and the range of variations of current source channel lengths. The most common cell in the DAC module is the segment current source cell. For the two 5 volt designs described in chapter 6, the amount of required circuit and interconnect area as a percentage of total cell area may be used as a measure of layout efficiency, with 100% for a cell with no white space. This is a measure of the difference between the module size obtained using this CAD approach and the possible full custom module size, given identical circuit topologies. For these designs, that percentage is 92% and 94%.

For some DAC implementations the use of sparsely filled cells is a more important loss of circuit area. In particular, the 100-Msample/s 8-bit DAC design in Fig. 5.5 has 16 columns, but only 3 lsb cells. In the lsb row of the array, there are only 3 cells with transistors in them, out of 21 across the array. This row results in most of the white space in this DAC implementation. The module layout efficiency is estimated at of 81% due mainly to these empty cells and additional white space in the column driver row. (In contrast, the 10-bit example matches the number of columns with the number of LSB cells, resulting in 100% space efficiency.) In general, the trade-off of a simpler overall DAC implementation, with deterministic layout of lsb elements, was made at the expense of some wasted circuit area. The layout densities for these circuits was acceptable for a CAD solution.

5.4.2 Advantages

This layout synthesis approach satisfied all DSYN requirements, including the quality and compactness of the results, and the predictability of layout related parasitics.

Though not as compact as manual designs, the layouts generated by DSYN using stretching and tiling are much more compact than those created by typical analog synthesis tools, especially for the digital decodes, buffers, and latches required in these cells. Use of a cell library results in compact cell layouts through the use of all possible interconnect layers and compact module tiling due to cell terminal placement for abutment connections. DAC specific analog layout issues, such as device placement for matching, parasitic capacitive coupling effects, and parasitic resistive effects, may be considered when creating the library cells.

The layout process is a predictable algorithmic process. When the cell library is known, it is possible to predict all parasitics a priori from design variables. This is the approach taken with DSYN for inclusion of layout parasitics within the optimization process.

The layout synthesis step takes a few minutes of compute time on a workstation. This is not fast enough for inclusion with circuit extraction in a circuit optimization, but this relatively short layout time does help when debugging stretch annotations in the cell library, and checking the stretched cells for connections through abutment.

5.4.3 Summary

The advantages found for this approach led to its use for DAC layout synthesis. It meets the requirements for the design synthesis approach used in DSYN, and results in compact layouts of both analog current sources and switches, and the digital circuits used in the DAC module. The complete layout process typically requires 3 minutes to run on a DECstation 5000/133. The synthesis methods used here have been used for digital circuits, but their application to analog DAC modules is a new.

CHAPTER 6

DAC Module Synthesis Implementation and Results

6.1 Introduction

In this chapter the implementation of the current sourced DAC module generator is described, and the results are discussed. It starts with description of a high speed current output DAC architecture implemented here, with a description of important design parameters and mapping of those parameters to layout geometries and module size. This is followed by a discussion of the technology inputs, sources of non-ideal DAC behavior, and steps taken in a DAC implementation to mitigate some of these effects. Then the estimation process for this DAC implementation is described, determining performance from design inputs. The design synthesis process is used to size the DAC for two specifications. After these test devices were fabricated, measured results could be compared to specifications and estimated performance. The sources of performance differences are discussed, for future incorporation of corrections to the DSYN process. A discussion of those circuit and performance effects that could not be included in the synthesis process is undertaken. Finally, some conclusions are drawn from the synthesis, fabrication, and test process.

When writing this chapter, it was difficult to find a good place to start. Here the synthesis process is described in a linear manner, start to finish, but in practice the process is more organic. There is a high degree of interaction between the parametrization of the design, the choice of sim-

ulation/analysis methods, the layout implementation, and the optimization process, and it is difficult to consider each these issues individually.

6.2 A Parametrized Current Switched DAC Module

The video DAC application was a target for this work, and a current switched DAC architecture capable of meeting those specifications was chosen. In this application current is sourced from the DAC, and used to drive a transmission line directly. The current source architecture has PMOS current sources for this reason. In this section the overall module design and module parameters will be discussed. Then each circuit will be described, with its design parametrization. Free and dependent variables will be labeled in each section. The complete design has 20 free variables. This is an elaboration of the DAC architecture description at the end of chapter 2.

6.2.1 DAC Module Circuits

The DAC module is tiled from 12 stretched cells, of which 6 contain important circuits, while the remainder are used for routing. The key subcells are located and listed in Fig. 6.1. They are the switched segment current source cell, the switched lsb source cell, the bias cell, an analog bus cell, row latch and driver cell, and column latch and driver cell. All have may be stretched to change key device sizes, except the analog bus cell, where stretching changes the width of the analog buses. Both digital and analog circuits can be modified in this module.

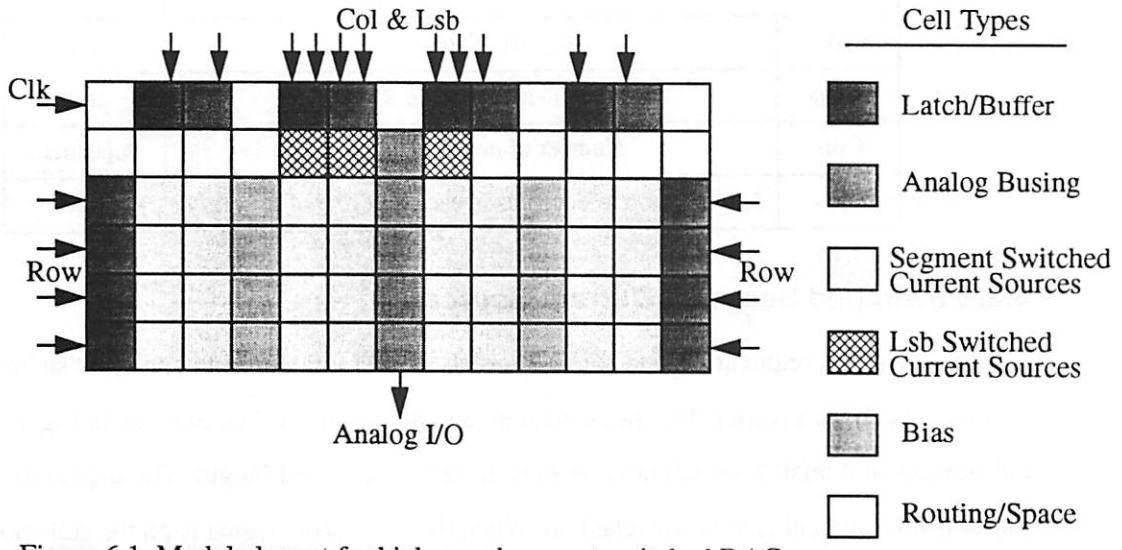


Figure 6.1 Module layout for high speed, current switched DAC.

The signal flow through the module passes digital data in at the top and side boundaries, latching and buffering row and column select signals at the edge of the array. Row and column select signals are driven across the module. Each individual cell determines if it should be enabled, and sends the current signal to either an output or dump line. The analog bus in the center of the array collects the output and passes the current signal out through the bottom of the array.

Data latches are located at the row/column inputs, and after before the switches in the current source cells. True Single Phase Clock (TSPC) [YUAN89] latches are used throughout, with latching on the rising edge of the clock. Latches are required to align the data, preventing large glitches in the output. In a lower speed design the latches may be omitted.

An additional binary to thermometer encoding is required before row and column signals are input to the module.

At the module level there are 5 parameters, of which 4 are independent (or 'free'). These are listed in table 6.1. N, M, Rows and Cols must obey the relationship N = M*Rows*Cols. The right hand column is used to specify the type of variable when it is used in optimization. Classifications are free, dependent, previously defined, and fixed by the user.

Table 6.1 DAC Parameters at Module Level

Variable	Description	Type
N	Resolution --Number of DAC levels.	free
M	Segment Size, in lsb	free
Rows	Number of module Rows	free
Cols	Number of module Columns	dependent
Bias	Number of bias elements in each half row of the array.	free

6.2.2 Switched Segment Current Source

The switched segment current source consists of a M unit current source, its switch, a logical decode, a latch, an inverter. The block diagram and transistor level circuits are in Fig. 6.2. The digital devices and analog switch devices have minimum channel length. The digital decode determines if this element is to be switched on. When the Next_Row signal is on the cell should always be selected. Otherwise the cell is selected when both the Row and Col signals are on. The TSPC

latch The current steering switch devices use two gates and a shared diffusion per switch device to minimize output capacitance. The current source is a cascode current source, made up of M individual sources, placed in a mirrored arrangement to null current direction mismatch effects. Separate analog and digital power supplies are used. Analog Vdd is routed in a wide bus over the current sources, minimizing resistance in that bus.

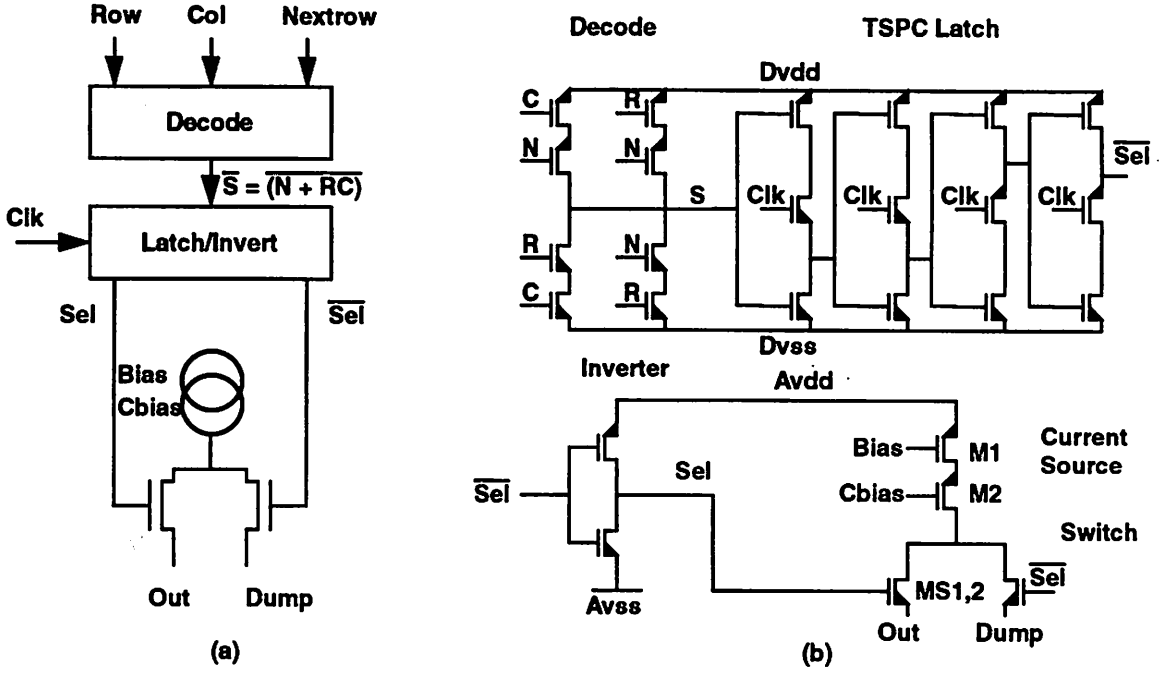


Figure 6.2 Segment Switched Current Source. a) Block diagram b)circuits.

User input is used to set all device widths, and lengths for the current source devices. There are 16 design variables for this circuit, listed in table 6.2.

Table 6.2 Variables for Switched Segment Cell

Variable	Description	Type
M	Number of sources per segment	Defined
W1	W of M1	Free
L1	L of M1	Free
W2	W of M2	Free
L2	L of M2	Free

Table 6.2 Variables for Switched Segment Cell

Variable	Description	Type
WS1	W of MS1 and MS2	Free
WNINV	W of NMOS in inverter	Dependent
WPINV	W of PMOS in inverter	Free
WP_NLAT	W of PMOS in N section of latch (first two legs)	Free
WN_NLAT	W of NMOS is N section of latch (first two legs)	Dependent
WP_PLAT1	W of PMOS in first P section of latch (third leg)	Free
WN_PLAT1	W of NMOS in first P section of latch (third leg)	Dependent
WP_PLAT2	W of PMOS in second P section of latch (fourth leg)	Dependent
WN_PLAT2	W of NMOS in second P section of latch (fourth leg)	Dependent
WPDECODE	W of all PMOS in decode	Free
WNDECODE	W of all NMOS in decode	Free

6.2.3 Switched LSB Current Source

The switched lsb current source cell is functionally similar to the segment current source cell, except that it has a single input enable line, does not require the logic decode, and has only a single current source. The block diagram and schematic are in Fig. 6.2. The current source is a single cascode of two devices, with the same dimensions and bias voltages as the segment source. In each cell there is only one source, so it cannot be mirrored, but the lsb current source cells are mirrored and grouped so that for all even codes there is an equal number of right-to-left and left-to-right current sources connected to the output.

The same user inputs which set the segment cell control sizing in the lsb cell. The digital devices and current source switches are scaled down due to reduced currents and device sizes seen in this cell. There are no cell specific independent variables.

6.2.4 Bias for Current Sources

Bias cells create a high swing cascode bias [LETH87]. The circuit is in Fig. 6.2. The MB1 and MB2 devices match the M1 and M2 current source devices. M5, M7, and M9 create a second bias current which drives M13 at a high current density to create the cascode bias voltage. The cell has

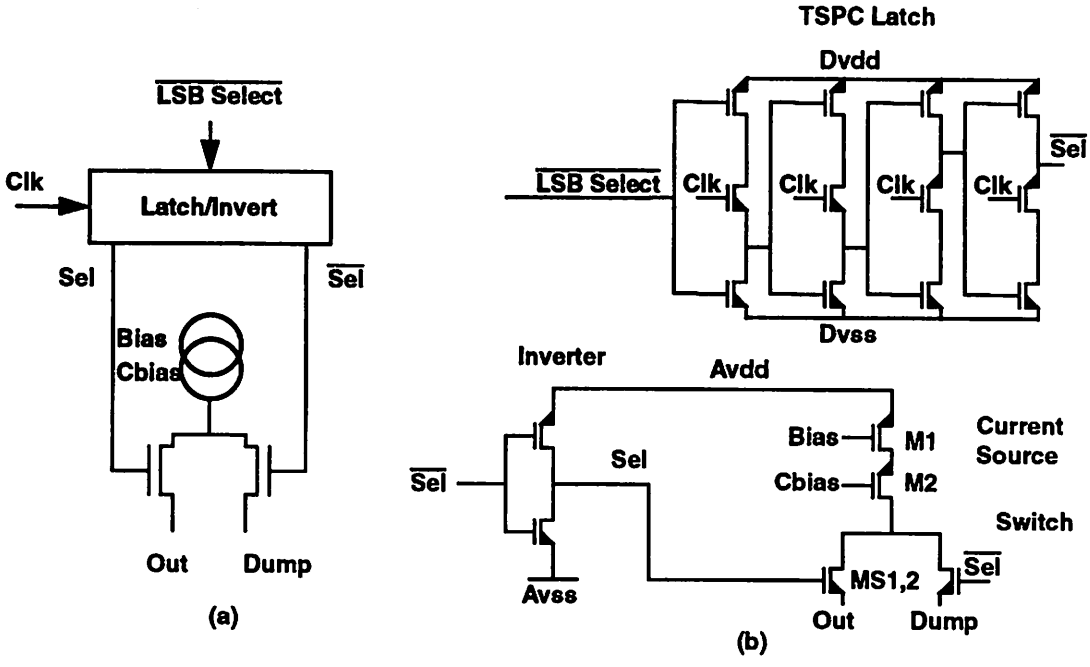


Figure 6.3 LSB Switched Current Source. a) Block diagram b) circuits.

half the number of current sources as the current segment cell, to make room for the additional bias circuitry. Bias circuit design parameters are summarized in table 6.3.

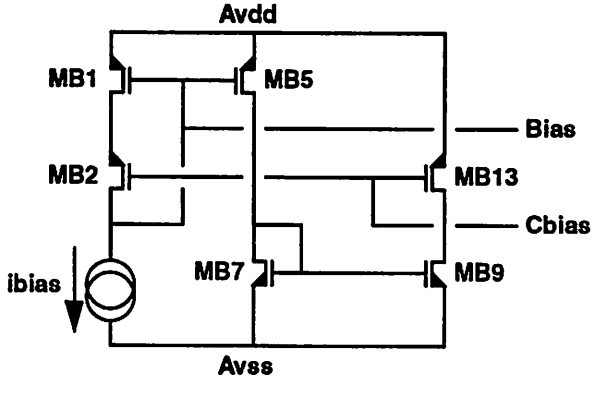


Figure 6.4 DAC bias cell circuit.

Bias cells are placed in the center of each half row, to first order match resistive drops seen by the bias to resistive drops seen by the current segments. One of the design inputs sets the number of bias cells per half row. If the bias must be settled quickly, one optimizer option is to increase the number of bias cells, reducing the impedance on the bias lines.

Table 6.3 Bias Cell Variables

Variable	Description	Type
M	Number of sources per segment	Defined
W1	W of MB1 and MB5	Defined
L1	L of MB1, MB5, and MB13	Defined
W2	W of MB2	Defined
L2	L of MB2	Defined
W7	W of MB7 and MB9	Free
L7	L of MB7 and MB9	Fixed
W13	W of MB13	Free

6.2.5 Analog Bus

The analog signal bus cell is used to connect supplies and bias to the current sources, and pass the output current out of the cell. It consists of routing on metal1 and metal2, and polysilicon shielding geometries to reduce substrate coupling to the analog output. Stretches are defined in the analog bus cell to match the current source segment, so that interconnects will align. Also, bus widths are controllable. The minimum bus widths for bias current, output, and supply are set by electromigration rules. In some low voltage designs buses must be widened further to reduce non-linearity due to resistive drops. Parameters are summarized in table 6.4.

Table 6.4 Analog Bus Variables

Variable	Description	Type
WVDD	Width of Vdd supply	Free
WVSS	Width of Vss	Dependent on Electromigration rules.
WIBIAS	Width of Current Bias Input	
WOUT	Width of Current Outputs	

6.2.6 Row and Column Latch/Buffer Circuits

Row and column latch/buffer circuits are used to latch the input data and drive row, column, and lsb select digital signals. The latch and buffer circuits are identical for all of these, except for a signal inversion in the lsb select path. The layouts differ, to better pitch match the DAC module.

The basic circuit is illustrated in Fig. 6.2. The row cell consists of a latch/buffer circuits for the Row and Next_Row signals. The column cell has latch/buffer circuits for Col and LSB signals.

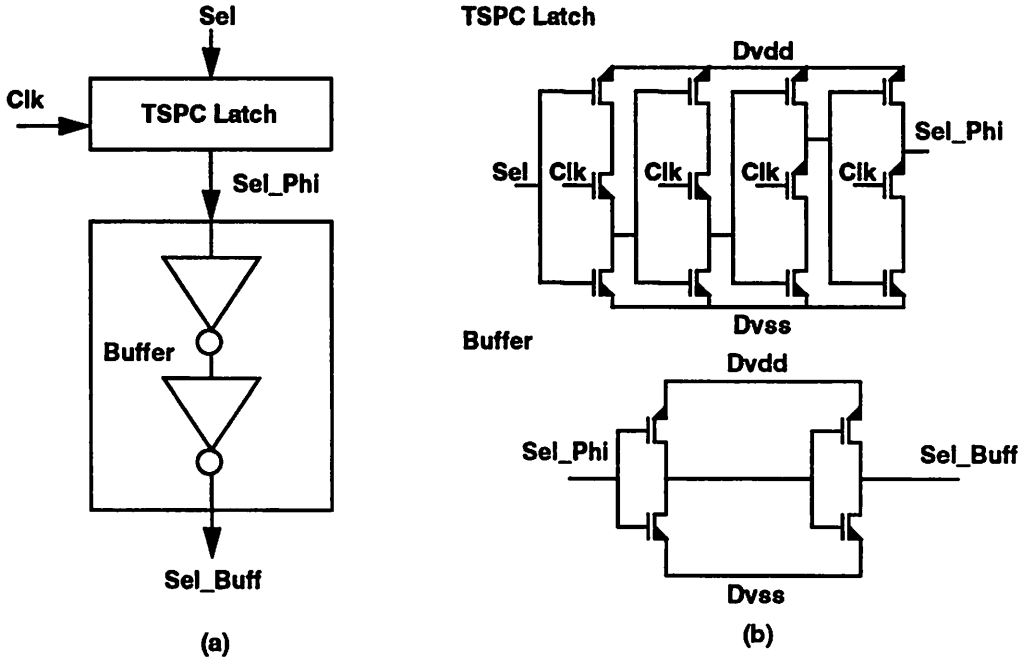


Figure 6.5 Latch and Buffer circuit topology used to drive row and column signals. a) block diagram. b) circuits.

For these circuits the TSPC latch circuit is not parametrized, but the inverters in the buffer are. Table 6.5 lists the parameters for the row and column versions of this cell.

Table 6.5 Variables for Row and Column buffers.

Variable	Description	Type
WC1	Scales column buffer first stage.	Free
WC2	Scales column buffer second stage.	Free
WR	Scales all row buffer sizes	Free

6.2.7 Other Layout Cells

There are a half dozen other layout cells in the DAC module, used for routing and spacing of the DAC design. Most have simple stretches used to obtain correct cell size and to pitch match routing to other cells.

6.3 Inputs for DSYN

A synthesis tool depends on accurate inputs for circuit performance estimation. Typical inputs for any circuit design problem include technology inputs and design related inputs. Technology inputs include device models, models for all electrical elements, and predicted lot-to-lot process variation. Design related inputs include power supply voltages, bias currents, and output loading. The inputs must specify both nominal inputs and the range of variation for all of these. For this analog circuit design, this basic set of inputs must be expanded upon, to include estimates of device mismatch due to random variation, and random process gradients. In these circuits relatively high currents may be present, so electromigration rules must be included. Device thermal and 1/f noise were not important in these applications, so accuracy of these factors was not measured in this DAC work, though in low power amplifier, filter, buffer circuits noise models would be an important input.

Accurate design estimation is an important goal for this work. According to the estimation philosophy described in section 3.3.1, any errors in the design inputs should be corrected before they are used in calculations. For example, corrections to MOS device output conductance (g_{ds}) measurements should be made before g_{ds} is used to compute current source R_{out} and DAC INL. In this section known errors and limitations to the input will be highlighted, so that front-end corrections can be made.

This section discusses each of these synthesis inputs in turn, noting the importance of the input, the accuracy of the available data, and any special considerations when used in synthesis.

6.3.1 Nominal Process

The process model consists of two parts: the MOS device models and electrical characteristics for circuit interconnects. Before considering second order effects, the nominal modeling data should be evaluated. For this work a 1.2 µm foundry process from ORBIT Semiconductor was used. Test devices were measured for fitting to device models, and other process specific information was obtained from the technology description[ORBI92].

6.3.1.1 Nominal MOSFET Models

The nominal device models for a process are fitted from measured device performance curves. The definition of a good model is application specific. Most MOS device models are adequate for digital circuit performance, but none of the widely available models meet all the requirements for analog circuit applications. Tsividis offers a list of 6 specific tests, and no commonly available model passes all of these[TSIV94][1]. In specific applications other model deficiencies may also result in erroneous results[2]. As seen in section 3.4, the model fit for g_{ds} in weak inversion and near the linear/saturation region boundary is a weak point of the models available in the HSPICE simulator, and this is an important part of the curve for accurate prediction of DAC static performance. A derating of g_{ds} of all devices by a factor of 2 is used at all times, compensating for the worst case misfit seen for devices in strong inversion.

6.3.1.2 Nominal Parasitics and Electromigration Rules

Nominal process parasitics and minimum wiring widths were provided by the foundry. These included wiring layer capacitances, sheet resistance, and maximum current densities for metal 1 and metal 2 interconnects.

6.3.2 Process Variation

Process variation is the lot-to-lot change in process characteristics over time. For the technology used in this work worst case variations to device model inputs were specified, and these were used to create fast and slow test files. Fast and slow modifications for the NMOS transistor are illustrated in table 6.6. The PMOS device used identical magnitude variations, with appropriate signs. The fast NMOS / fast PMOS, and slow NMOS / slow PMOS cases were used in design estimation to find worst case design points.

For other circuit parasitics range of variation of resistances and capacitances was given. For this DAC circuit the nominal capacitance values were used, and worst case resistances. Electromigration rules are expressed for a worst case lifetime, so no additional spec biasing was used.

---

1. BSIM 3 models in development at U.C. Berkeley have passed most of these, but have not passed the weak inversion output conductance problem. [HUI94]

2. Gate capacitance in moderate inversion is an important, poorly modelled, effect in some low power gain block applications [CHIE94].

Table 6.6 Nominal, Fast, and Slow Variations for an NMOS Transistor in Orbit 1.2 μm technology [ORBI92].

Technology Input	Nominal	Fast Bias	Slow Bias	Units
t_{ox}	225	-15	+15	Angstrom
V_t	850	-200	+200	mV
W_{eff}	W - Wd	+0.25	-0.25	μm
L_{eff}	L - 2*Ld	-0.15	+0.15	μm

6.3.3 Temperature Variation

All semiconductor products are specified for an operating temperature range, and designs are built to function across a range of junction temperatures. Device models include temperature effects, and circuits are simulated at the extremes of the temperature range. Worst case design techniques identical to those used for process variation are used to ensure that the design will function in all applications. In this research temperature effects were not included explicitly. The approach for synthesis including operating temperature range is a worst case analysis approach, like that as described for process effects, so leaving out this aspect of the problem does not compromise the overall applicability of the results.

6.3.4 Statistical Matching Effects

Device mismatch is an important factor for DAC performance, causing static non-linearity. Device mismatch is typically separated into two components, one the mismatch found in adjacent devices, dependent on the geometry of the devices, and the second mismatch a function of the distance between the devices. In amplifier circuits, the distance function is usually insignificant, but in this case, with matched devices separated by distances up to 1000 μm, both effects are significant.

6.3.4.1 Random mismatch

Several studies of random mismatch behavior have been done, looking at causes and models for capacitor and MOS device mismatches [SHYU84,LAKS86,PELG89]. The two earlier studies emphasized edge effects causing variations in device width or length, and formulated models based on these assumed causes of device mismatch. Data was then fit to these models. Unfortunately, the test data used did not have a wide enough range of device aspect ratio to adequately test

these assumptions. Also the test data taken in those works was for single technologies, so it is difficult to re-apply that data to a newer technology. In Pelgrom's work, much more data was taken, with a larger range of device aspect ratios and in several MOS technologies. The results from the large aspect ratio devices refuted the mismatch due to edge effects model, and instead suggested random variation in mobility (β) and threshold voltage (V_t) across a device's area. This gave an expression for variance of V_t and β, inversely proportional to device area ($W \cdot L$):

$$(\sigma_\beta)^2 = \frac{(A_\beta)^2}{W \cdot L} \tag{6.1}$$

$$(\sigma_{V_t})^2 = \frac{(A_{V_t})^2}{W \cdot L} \tag{6.2}$$

With A_{V_t} and A_β, being constants specific to the technology. These technology specific constants were plotted as a function of oxide thickness (t_{ox}). When this was done no trend in A_β was obvious, but A_{V_t} was inversely proportional to t_{ox}. This is consistent with the a physical model of β variation due to random mobility variation in the device, and V_t variation due to random trapped charge at the oxide interface.

Pelgrom measured devices in technologies with tox ranging from 25nm to 100nm. Most of the data is from processes in use at Phillips, although he includes some data points from other sources. In this DAC synthesis application, the technology uses a 22.5 nm process from Orbit Semiconductor, so a conservative extrapolation to a foundry and technology outside Pelgrom's data is necessary for this work. The expressions used for A_β and A_{V_t} used in DSYN are:

$$A_{V_t} = 0.66 * t_{ox} \tag{6.3}$$

$$A_\beta = 0.02 \times 10^{-6} \tag{6.4}$$

Where t_{ox} is in meters, A_{V_t} is in volts*meters, and A_β is in change*meters.

After observing results from fabricated devices corrections to the estimates of A_{V_t} and A_β may be made. In an industrial setting better data is usually available for determining these constants before synthesis is done.

6.3.4.2 Mismatch due to Device Spacing

In all of these matching papers, device mismatch as a function of distance has also been considered. For all effects, a good model has been:

$$\sigma^2(d) = S^2 d^2 \tag{6.5}$$

where d is the distance between devices, and S is a technology dependent constant. This is the model for two devices placed along one dimension; a sigma space (σ_{space}) analysis may be done to see find an appropriate model for multiple devices in 2 dimensions [MICH92], in which all pairs of devices meet this matching model. The solution to the analysis is dependent on the $\sigma^2(d)$ function. When $\sigma^2(d) \propto d^2$, the solution corresponds to a linear gradient across the space, and the stochastic nature of the problem is expressed through the direction and slope of the linear gradient.

In DSYN a linear gradient model is used, with an application dependent worst case direction assumed, and magnitude set by S for the technology. It will be seen that the assumption of linearity allows cancellation of most of the DAC nonlinearity caused by this effect. Again, limited data is available, so values of S_{V_t} and S_β were taken from a 50 nm process characterized by Pelgrom.

$S_{V_t} = 4.0$ V/m

$S_\beta = 2\ m^{-1}$

6.3.5 Design Inputs

Design inputs can be expressed either as constants, used as performance inputs or constraints, or may express a domain of operating conditions for proper device function. Analog integrated circuits are usually specified for operation across a range of power supply, bias, and loading conditions. For this DAC implementation the range of possible bias and supply conditions was included as part of the worst case analysis of the circuit, and the external loading was simulated at the maximum specified lead inductance and load capacitance.

6.4 DAC Implementation Techniques

Before getting to the actual optimization of the current switched DAC design, there are some low cost implementation steps which can be used to reduce the effects of the non-idealities described previously. This includes current source layout for matching and current source switch ordering to minimize process gradient and resistive drop effects. In this section the general approaches are outlined, and then the circuit implementations for this work are described.

6.4.1 Layout for Matching

Several rules for layout of matched MOS devices should be obeyed in integrated circuit implementations [PELG89,NAKA91]. To reduce random mismatch, the important rules are:

1. Matched devices should have the same dimensions.
2. Matched devices should have the same orientation.
3. Matched devices should have current flow in the same direction.
4. Metal coverage of the device, especially M1, should be identical.
5. Layout adjacent to matched devices should be identical.

In this implementation, these rules were generally followed in the library cell layouts for the segment current source, lsb current source, and bias cell. Rules 1 and 2 were followed throughout, with all segment current sources implemented as multiples of lsb weighted current sources. Devices were mirrored to reduce rule 3 mismatch. Rule 4 was followed except in the cascode devices of lsb sources. Rule 5 was not strictly followed, and this has been identified as a source of error in the test circuits.

6.4.2 Cell Switch Ordering for Improved Linearity

Process gradients and resistive drops result in graded errors across large arrays of devices. In this implementation the power supply is distributed through a bus in the center of the array, resulting in graded errors in the vertical direction, and symmetrical errors in the horizontal direction (Fig. 6.1 illustrated this layout.) The turn on ordering of the cells may greatly effect the DAC INL. If the devices are turned on in a sequential, left to right ordering, INL error is accumulated across half of a row before it is cancelled by the negative error in the second half of the row. This INL accumulation results in a significant error. Nakamura introduced a technique of hierarchical sym-

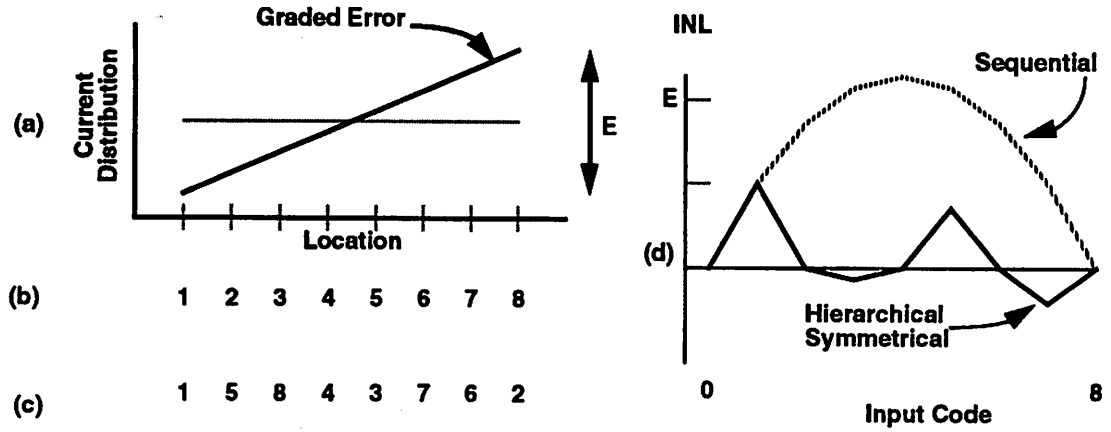


Figure 6.6 Switch ordering cancellation of gradients. a) Graded error b)Sequential switch ordering. c) Hierarchical Symmetrical switch ordering. d)INL for these methods.[NAKA91]

metrical switching which cancels the error accumulation due to graded and symmetrical errors across a row, through a no cost cell ordering scheme [NAKA91]. Fig. 6.6 illustrates this concept for an example with a linear gradient. The maximum INL for these switching schemes is:

$$\text{Sequential Switching: } INL = \frac{E}{8} \cdot \frac{N^2}{N-1} \tag{6.6}$$

$$\text{Hierarchical Symmetrical Switching: } INL = E/2 \tag{6.7}$$

Note that the INL with sequential switching is proportional to the number of elements (N) times the total error (E). For the suggested switching scheme the INL is only proportional to E.

When the graded error E is due to a process gradient effect, which is related to output signal through a linear relationship, then an expression for E is:

$$E = N1 \cdot K \cdot S_{effect} \cdot N2 \cdot CellSize \tag{6.8}$$

Where N1 is the number of lsb elements per switched unit, and N2 is the number of units in the row. N2 * CellSize is the total distance across the switched devices. S_{effect} is the slope of the graded error source. For this circuit an important effect is V_t mismatch of the main device, which is related to the output current through the g_m of that device:

$$E = N1 \cdot g_m(M1) \cdot S_{Vt} \cdot N2 \cdot CellSize \tag{6.9}$$

An important limitation is that E sets a lower limit for INL, regardless of the switching scheme applied.

This switching scheme is applied to both the rows and columns in the DAC design. The DAC has M lsb/segment, R cells per Row, and C cells per column. Each row is turned completely on before the next row is used. Typically the DAC aspect ratio is near 1, so assume DacSize = N2 * CellSize is the same for both rows and columns. Then

$$E(col) = M \cdot g_m(M1) \cdot S_{Vt} \cdot DacSize \tag{6.10}$$

$$E(row) = R \cdot M \cdot g_m(M1) \cdot S_{Vt} \cdot DacSize = R \cdot E(col) \tag{6.11}$$

With these assumptions, the INL contribution when the rows are switched, E(row), is most significant. In a low voltage, high current design, the E(row) factor due to threshold voltage gradients prevented the design from meeting INL specifications, and further reduction in gradient induced INL was required.

6.4.3 Row Splitting

A row splitting scheme was used to eliminate the E(row) effect. The array was split in half horizontally, and each logical row consisted of a left and right half-row. These half-rows were chosen so that all logical rows have a common centroid layout, cancelling linear gradient mismatches between rows. Fig. 6.7 illustrates this technique.

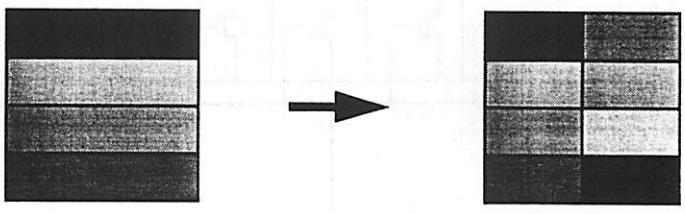


Figure 6.7 Row splitting for common centroid row layout. Each shading represents a logical row in a 4 row DAC design. a) The original design, with each logical row corresponding to a physical row. b) Rows split, with the common centroid of each row at the center of the DAC.

In this DAC implementation the current source array is already split into two halves to allow routing of the bias, signal, and supply through the center of the array in a vertical bus (See Fig. 6.1). The row select signals are routed in from either side. This design was initially selected to reduce resistive drop effects from the analog bus to the extremes of the array. In this implementation the addition of row splitting is free, requiring only a different ordering of row select signals between the two sides of the array.

6.4.4 Current Carrying Bias Lines

Resistive drops may be an important effect in current source DAC designs with large output currents. Current flows in the power supply buses, the output signal lines, and the bias input current line. It is impossible to avoid current in the power supply buses, and the main design technique is to let the synthesis process size the width of these buses. Parasitic resistance in the output path does not impact linearity significantly, because it is orders of magnitude smaller than the output impedance of the DAC. If the signal line used to set current source gate bias carries current, then there will be current source mismatch across the array.

This source of mismatch can be avoided. Fig. 6.8 illustrates the technique used to eliminate this effect on non-linearity in this DAC. The bias current is run in a separate signal line from the bias voltage, and no bias voltage signal sees any DC current. Drops in the bias current line are not seen by the DAC current source cells, so these do not affect linearity.

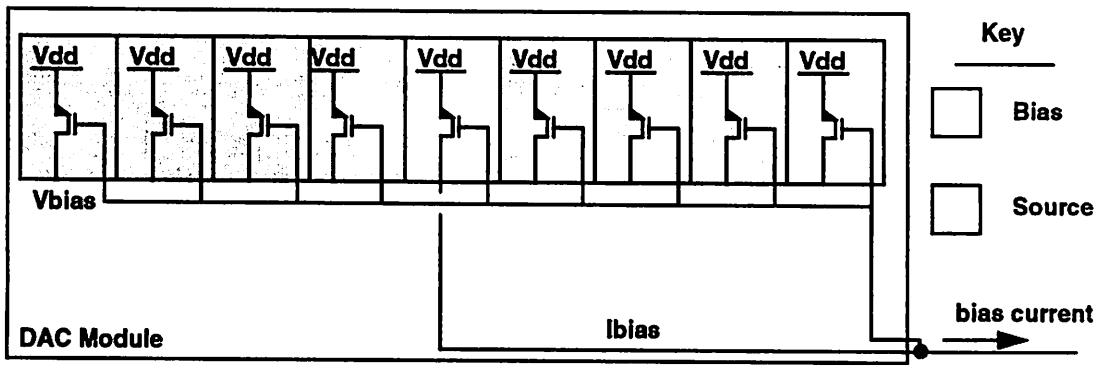


Figure 6.8 Bias Voltage signal carries no current, and sees zero resistive drop.

6.5 Design Estimation for a Current Switched DAC

The design estimation process follows that described in chapter 4. Using the design inputs, a full description of all element, bus, and cell sizes is created. Succeeding simulation and analysis steps are used to predict circuit performance. In this section the specific design steps and simulations for a current output DAC are described. This is broken into the initial problem setup phase, simulation/analysis for static performance prediction, and simulation/analysis for dynamic performance prediction.

6.5.1 Problem Setup

In the problem setup phase, the user input is combined with technology and layout specific data to determine physical dimensions for the DAC cell. The DAC design is a parametrized design [KOH89], so all design dimensions are computed from a smaller set of design inputs. Additionally, circuit elements which are a function of layout parasitics are sized, and device mismatch is predicted. The same setup step is used to produce the necessary inputs for layout synthesis.

This is done through the following steps.

1 Compute all device sizes from parametrized input.
2 Compute device random mismatch from device sizes.
3 Compute cell and DAC size from device sizes.
4 Compute process gradient mismatch from cell and DAC sizes.
5 Compute bus sizes from inputs and electromigration rules.
6 Compute parasitic resistance and overlap capacitance in signal buses.
7 Compute device diffusion capacitances from device widths.

At this point the design is fully described.

6.5.2 Simulation/Analysis for Static Performance

When the design description is complete, DC circuit analysis is used to find the operating point of the circuit, and then further analysis based on the operating point obtains circuit performance measures. The circuit operating point is first checked for design feasibility. Then static linearity of the DAC is predicted based on deterministic and random effects. In the implementation INL, DNL, Gain Error, and Total Unadjusted Error were all included. Computing the last two is usually rela-

tively trivial once INL has been determined. INL and DNL are considered the most important, and will be discussed in depth here.

6.5.2.1 Design Feasibility

In circuit optimization a set of design inputs may result in a circuit operating point which violates some of the assumptions made by the circuit designer. Most importantly in analog MOS design, many devices should operate in the saturation region with some margin. In design optimization this is specified as a constraint, that Vds - Vdsat > MARGIN. There are additional bias margins that ensure that the input current source has adequate headroom. Most of these voltage margin constraints can be seen by considering the DAC bias circuit. The circuit is in Fig. 6.9, and table 6.7 lists the applicable constraints, with an explanation of each.

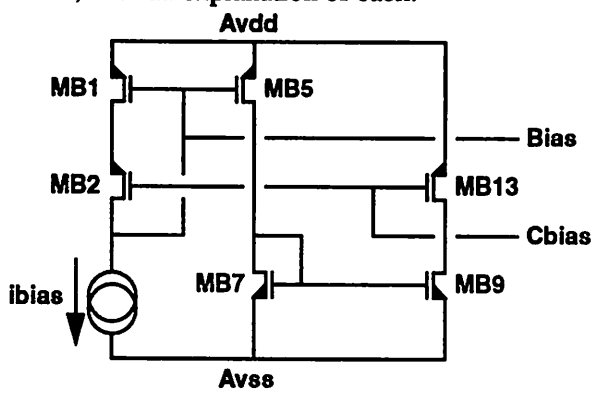


Figure 6.9 DAC bias circuit

Table 6.7 Design Feasibility Constraints

Constraint	Explanation	Typical Values (Volts)
vdsmar1	Min. Vds - Vdsat for MB1	0.2
vdsmar2b	Min. Vds - Vdsat for MB2	0.2
vdsmar5	Min. Vds - Vdsat for MB5	0.2
vdsmar9	Min. Vds - Vdsat for MB9	0.2
vdsat7	Min. Vdsat for M7	0.4
vinmarg	Min. Ibias input voltage	1.0

6.5.2.2 Deterministic Effects on Static Performance

There are two important deterministic effects which affect static performance, Finite output resistance of the DAC current sources, and resistive drops across the array.

Output Resistance

The current source output resistance causes a second order nonlinearity as more current sources are connected to the output, creating INL. The ratio of load resistance to output resistance determines the magnitude of this error. In section 4.3.6, the gain error computation is made, based on small signal parameters extracted from the operating point and module level parameters. The maximum INL occurs at the midpoint of the range. That result is repeated here:

$$\text{Gain Error (in lsb)} = N^2 \cdot R_{load} \cdot g_{out} \tag{6.12}$$

$$\text{INL (in lsb)} = \text{Gain Error} / 4 \tag{6.13}$$

Resistive Voltage Drops

Resistive supply voltage drops result in second order gradients in effective bias voltage throughout the array. The supply is routed from the bottom center of the current source array, and resistive drops result in a decreasing V_{dd} - V_{bias} going up the center column and out the rows. This is non-linear, because current in the supply decreases the further one travels from the supply input. Fig. 6.10 plots the shape of supply drops for a row supplied at one end. It is assumed that supply drops are small, so all current sources see the same sensitivity to supply drops through the g_m of device M1.

If all wiring resistances and current sources are equal, then a unit voltage drop is defined by:

$$V_{unit} = I_{src} R_{wire} \tag{6.14}$$

and the voltage drop at the i^{th} current source in a row of R segments can be expressed:

$$Vdrop_i = V_{unit} \sum_{j=1}^{i} (R - j) = V_{unit} (R \cdot i - \frac{i(i+1)}{2}) \tag{6.15}$$

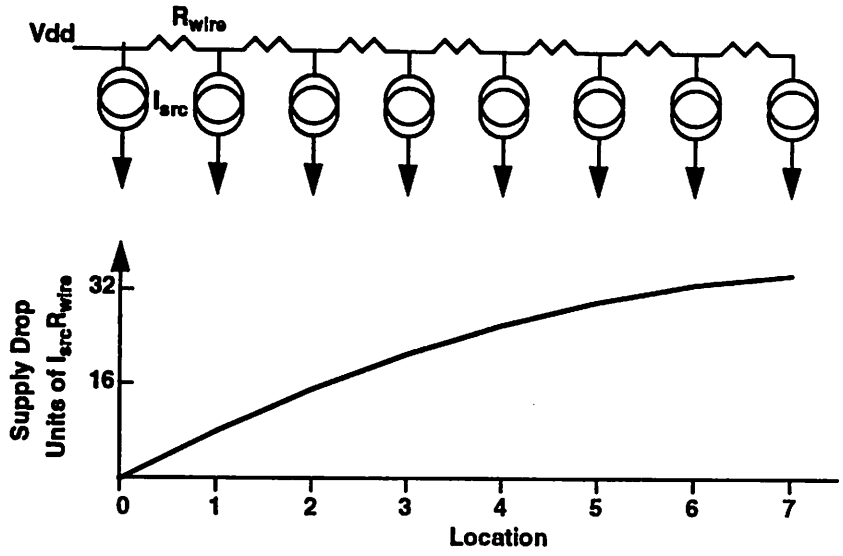


Figure 6.10 Current source arrays cause a non-linear supply voltage drop. All resistors and current sources are identical in this plot.

The current source error is linearly proportional to this voltage drop through g_m(M1). To scale this to units of lsb, divide by i_{lsb}. Miki has analyzed the non-linearity observed when this current source non-linearity is switched with a symmetrical switching scheme[MIKI86]:

$$INL = \frac{N_{Rows}^2 V_{unit} \cdot g_m(M1)}{I_{lsb} \cdot 36\sqrt{3}} \tag{6.16}$$

This is an optimistic limit, because in computing it Miki assumed a continuous function for the voltage drops, and infinitesimal switched elements. A more pessimistic analysis includes the discrete nature of the switched elements. If Row splitting were not used, the worst case INL would be due to the difference between the average current and the total worst case row current:

$$INL = (V_{worst} - V_{avg}) \cdot M \cdot N_{Cols} \cdot \frac{g_m(M1)}{i_{lsb}} \tag{6.17}$$

$$INL = V_{unit}\left(\frac{2N_{Rows}^2 - 3N_{Rows} + 1}{6}\right) \cdot M \cdot N_{Cols} \cdot \frac{g_m(M1)}{I_{lsb}} \tag{6.18}$$

where $V_{unit} = R_{wire} \cdot N_{Cols} \cdot M \cdot I_{lsb}$

When row splitting is used the effect is to approach the infinitesimal addition of contributors, as in Miki's derivation. There are now two contributions, due to the discrete addition of individual segments, and the mismatch between the average of the outer elements and the total average. These are derived as a segment effect and a row effect, with the following results:

$$\text{INL} = \text{Row Effect} + \text{Segment Effect} \tag{6.19}$$

$$\text{INL} = V_{unit}\left(\frac{N_{Rows}^2 - 3N_{Rows} + 2}{12}\right) \cdot 2 \cdot N_{Cols} \cdot M \cdot \frac{g_m(M1)}{I_{lsb}}$$

$$+V_{unit}\left(\frac{2N_{Rows}^2 - 3N_{Rows} + 1}{6}\right) \cdot M \cdot \frac{g_m(M1)}{I_{lsb}} \tag{6.20}$$

Comparing Eq. 6.18 with Eq. 6.20, the use of row splitting results in a factor of 2 improvement in INL, even in this case where the non-ideality is a quadratic effect.

In this implementation INL, DNL, and gain error effects are computed based on this resistive drops from the central analog bus to the edge of the array, and the voltage drops along the vertical supply bus. A pessimistic estimate is taken, assuming no cancellation between row and column effects. Additional drops due to current drained by bias circuits was also included in the computation.

6.5.2.3 Stochastic Effects on Static Performance

There are three sources of nonlinearity due to stochastic effects. Random mismatch of devices contributes to both INL and DNL. Gradients cause mismatch in current segments, and between the main DAC array and the lsb elements, again resulting in INL and DNL.

Random Device Mismatch

When analyzing DAC architectures in section 2.3.1.3, expressions for INL and DNL as a function of unit element mismatch was derived. Those results are repeated here:

$$\sigma_{INL} = \frac{\sqrt{N}}{2} \cdot \frac{\sigma_a}{\Delta a} \tag{6.21}$$

$$\sigma_{DNL}(\text{EachSegment}) = \sqrt{2M} \cdot \frac{\sigma_a}{\Delta a} \tag{6.22}$$

In this case Δa is the value for a unit current, I_{fs}/N, and σ_a is the variance of that current. Section 4.3.6.2 has the computation for σ_i due to threshold voltage mismatch. The other source of random mismatch is variation in device mobility, and it is computed with the same approach. In both cases the device variation is found as a function of device area, and then small signal parameters are used to convert this to an element current variance. These two effects are assumed independent, and the total INL variance is found from the sum of the variances.

For DNL this is the variance for an individual DNL sample, but there are actually N/M code transitions which have this variance. To obtain a better than 99% yield for a design with 64 such transitions, a 4.5-sigma design is needed for this random variable, instead of the usual 3-sigma.

Gradients

For this design linear process gradients were assumed, with a worst case direction, and slope assumed to be the 3-sigma slope. The worst case direction and slope is the one that maximizes the mismatch of the two most extremely placed matched elements in the DAC. Recall that the device mismatch standard deviations are proportional to distance, as seen in equation 6.5.

For INL computations, two contributions were identified. Within each row, the end devices have the maximum gradient mismatch. Between rows the maximum row deviation results in a nonlinearity. In either case, the unit current element variance at the maximum distance is computed:

$$\sigma_i^2(d) = \left(S_\beta^2 + \left(\frac{g_m}{i_d} S_{Vt}\right)^2\right) d^2 \tag{6.23}$$

Where σ_i^2 is the relative variance of a unit element current source, and g_m and i_d are for the main current source device M1. d is the maximum distance between matched devices, or groups of devices. If common centroid row switching is not used, the distances of interest are the DAC width (w) and height (h), the INL contributions are:

$$\text{INL due to a horizontal gradient across a row: } \sigma_{INL} = \frac{M}{2}\sigma_i(w) \tag{6.24}$$

$$\text{INL due to a vertical gradient across the DAC: } \sigma_{INL} = \frac{M \cdot cols}{2}\sigma_i(h) \tag{6.25}$$

The factor of 1/2 comes from the symmetrical switching. When common centroid row switching is used, the effect of vertical gradients is greatly reduced, with the cols factor removed, but the distance the diagonal across the array:

$$\text{INL due to a vertical gradient across the DAC: } \sigma_{INL} = \frac{M}{2}\sigma_i\sqrt{h^2 + w^2} \tag{6.26}$$

For DNL the worst case gradient is longest distance from the centroid of the lsb elements to any segment:

$$\sigma_{DNL} = M \cdot \sigma_i(d_{max}) \tag{6.27}$$

6.5.2.4 Addition of Stochastic and Deterministic Effects

Total estimates for static nonlinearity are found by combining stochastic and deterministic effects. All stochastic effects are assumed to be independent, so the variances of these estimates are added. This design uses a three sigma error estimate. Individual deterministic effects are combined additively, and added to the 3 sigma stochastic nonlinearity estimate. This gives a pessimistic estimate for the combined performance.

6.5.3 Simulation/Analysis for Dynamic Effects

Direct simulation is used for dynamic performance estimation In a first pass all dynamic performance was estimated from the simulation of a worst case rising and falling output, using additional analysis to compute glitch energy. This saved simulation time, but the glitch energy analysis required very simplifying assumptions. In the second iteration a separate simulation with glitch energy specific transients was run. An additional requirement at high data throughput rates was a simulation to ensure data integrity through the DAC.

6.5.3.1 Output Settling

The output settling simulation is used to estimate all dynamic performance specifications associated with a full scale transient. These are the clock delay time, the settling time, and the DAC

switching time. The important issues are identification of the worst case transients, and the reliable measurement of output settling.

An abstract view of a DAC undergoing a full scale transition is shown in Fig. 6.11. There are three primary effects affecting settling behavior. The differential switching waveforms at current switch determine when the current makes the transition, and the magnitude of any charge injection due to either capacitive coupling or a dead time when both switches are off. The output capacitive, resistive, and inductive loading determines the settling waveform. The coupling from the output to the bias and the settling time constant on the bias may result in a slow settling tail. The entire picture is further complicated by parasitic coupling between signals and the usual MOS parasitic capacitances. Not shown in this figure are the latch and inverter circuits which create sel and $\overline{sel}$.

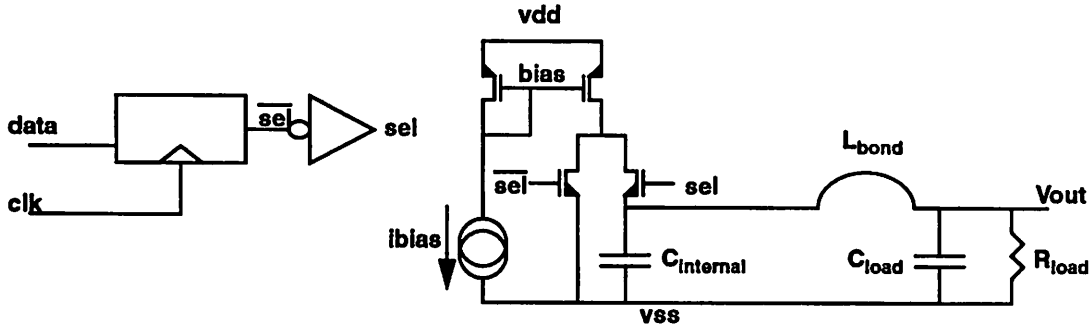


Figure 6.11 Abstract view of a switched DAC, including bias, switched current source, and external loading. Clock transitions trigger the cell to switch.

This is a good abstract view for the worst case rising waveform, when the signal transitions from zero to full scale (FS). When the signal settles at the maximum value slow settling tails from the bias have maximum impact, and in this case the ratio of signal swing to settled value is maximized as well.

Less obvious is the worst case falling waveform. In particular, the settling from FS to 0 maximizes capacitive coupling effects, but disconnects the DAC cell from the output, and does not include any bias settling tails. A small transition from FS to 0.9 FS maximizes the sensitivity to bias settling, but reduces the amount of charge injection from signal switching. A compromise transition is used, from FS to 0.5 FS. Typical output waveforms for these transitions are shown in Fig. 6.12.

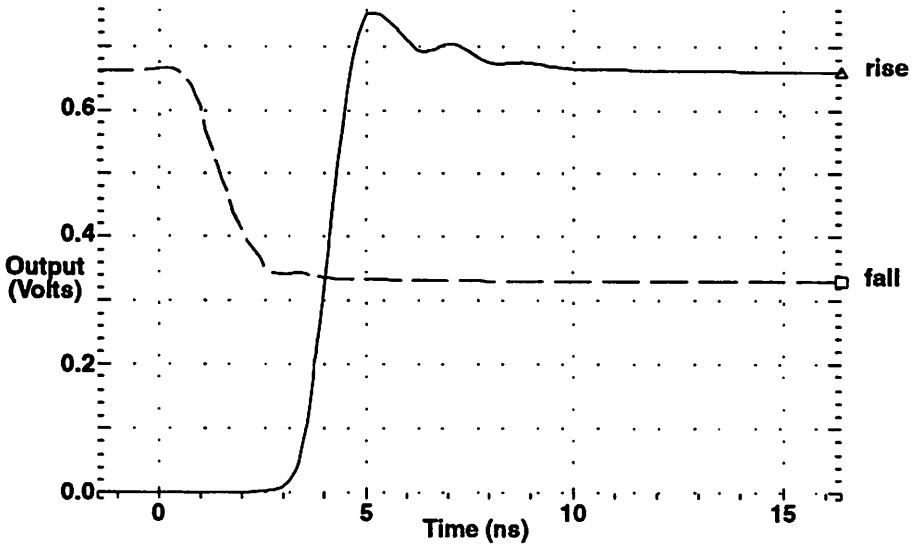


Figure 6.12 Rise and Fall transitions simulated for dynamic performance estimation.

Measuring the waveform for rise time and delay time is easily done with HSPICE '.MEASURE' statements, but techniques to accurately and efficiently measure the settling waveform is worth mention here. The settling time is specified as the time from the 50% point in the signal transitions to the settling within some delta (δ) of the final value. The most direct ways to determine the final value are error prone. Sampling the output at the end of simulation is gives erroneous results if the signal is not completely settled at the end time. The fix for this requires very long simulations, and wastes CPU time. Computing the final value a priori from design inputs may be in error if there are built in offsets in the bias and current source circuits. The method used successfully in this work was to create a replica circuit which was not switched, and compare the settling output to the replica signal. Settling simulations were run to 1.5 times the constrained settling time. If the output did not settle to within δ by the end of the simulation, then the observed final error was used to constrain the optimization solutions to those with complete settling.

6.5.3.2 Glitch Energy

Glitch energy is seen when switches turning off are cancelled by other switches turning on. In this DAC architecture glitches occur when a segment is turned on while all lsb elements are turned off. In section 2.3.1 a first order calculation of glitch energy was made, based on the difference in turn on and turn off delay, and the number of elements being simultaneously switched. This simplified analysis ignores any circuit mismatches between lsb and segment circuits, and charge injec-

tion in the current switch operation. A simulation approach was undertaken to include these effects.

In the glitch energy simulation, the implicit assumption is that the output from the switched elements does not change the output voltage significantly, and output current into a low impedance can be measured instead. A clock signal is used to switch on an M-bit segment current, while M lsb currents are switched off. The output currents are added, compared to a constant M-bit signal, and the difference integrated to find the glitch energy. The opposite transition is simulated simultaneously. The glitch energy is normalized to units of (settling time) * (lsb current).

6.5.3.3 Digital Signal Integrity

This DAC module must pass data from the edges of the array through a set of latches, buffers, a logic decode, and another latch to reach the current source switches. At the high clock rates envisioned for this module this is not a trivial task. If the digital circuits are incorrectly sized, the digital signals may not arrive in time at the cell latches, or the digital delays in the buffer and decode circuits may affect the analog circuit performance. In this module performance constraints have been used to ensure the digital signal integrity. This is done through the specification of setup time requirements for all latch circuits, and minimum signal level requirements latched signals in the dynamic TSPC latch circuits. Fig. 6.13 illustrates some of the points in the design where the signal integrity must be verified At the latch in the segment cell, the select signal (S) must arrive ahead of

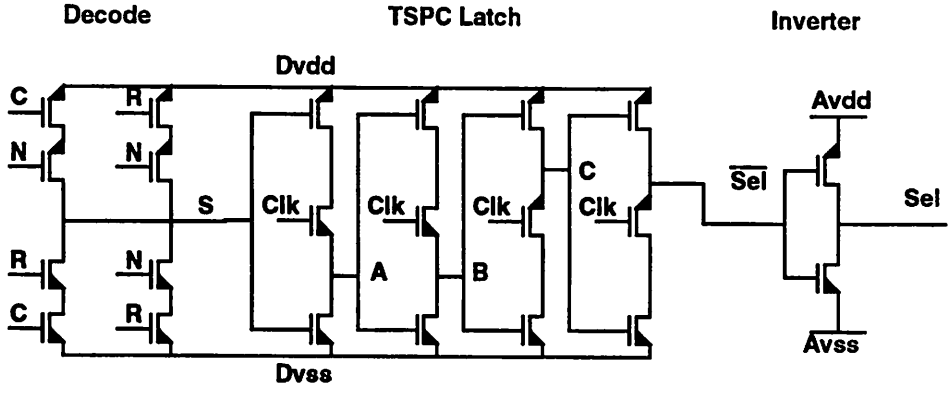


Figure 6.13 Digital signal integrity in the segment cell. Examples of locations where signal may be compromised. R, N, and C are driven by row and column buffers.

the Clk signal falling edge with some timing margin, typically 1ns. The levels of the dynamic nodes A, B, C, and Sel must be checked to make certain they go to full logic levels. The performance constraint used is that they must be within a voltage margin (typically 300mV) of the full CMOS high and low levels. This is simulated with both logic transitions, and the row and column latches are also checked for full swing signals.

6.6 DAC Synthesis Limitations

The module synthesis process described in this chapter does not solve the complete DAC design problem, because it is by nature limited to module level design. In a mixed-signal chip implementation there are additional chip level issues of importance for DAC performance. These non-idealities are the bonding wire lead inductance, and noisy digital supply and signal coupling to the analog circuits through capacitive parasitics and the chip substrate. They result in an environment in which the entire chip substrate may be ringing with respect to board ground, and this, in combination with digital switching noise results in a large noise floor seen at the analog output. Even when the analog circuits are shielded from sources of digital noise on chip, the process of taking the analog signal off chip results in a large noise signal component. These issues are particularly significant in the case of single-ended output video DACs, which do not have the inherent common mode rejection of differential outputs.

There are a number of common design techniques used to minimize these effects. Multiple bonding wires are used for supplies, digital output drivers are designed for minimum switching current, supplies on chip are carefully separated to minimize direct coupling to sensitive supplies, and digital circuits may be completely disconnected from the substrate in an attempt to reduce noise in the substrate. When this is not enough, specialized digital cell libraries can be used which have low signal swings and constant current drain, reducing coupled noise and eliminating di/dt induced supply noise. Additional resistive and capacitive structures can be added on chip to create a low impedance, low Q on chip supply, at the cost of some supply headroom. This is an area of continuing research in mixed-signal circuit design, and these issues must be solved if the combination of high SNR circuits and digital signal processing is to be integrated on the same substrate.

All of these techniques are chip level design techniques, which cannot be designed in at the DAC module level. At the module level the designer is much more limited, to creating circuits which are relatively insensitive to supply noise, and which do not create significant additional parasitic coupling.

In this DAC implementation these issues were addressed through the use of good design practices at the module design and chip levels, but they were not explicitly included in the module optimization process. For example, care was taken to avoid overlaps between digital signals and supplies and the analog output and bias signals. Polysilicon layer was used to shield the output from the substrate, and the substrate was not connected directly to any of the noisy supplies. An RC network was connected to the current bias input pin, filtering noise from the external current source connection.

Designing with these issues in mind is one of the areas where analog circuit design looks more like an art than a science. The processes involved are impossible to simulate or quantify efficiently or completely. This makes inclusion of these effects in a synthesis framework difficult.

6.7 Design Example 1: 8-bit, 100-MS/s Video DAC

One of the target applications for this module generator is the video DAC application. The goal of this example was to demonstrate the synthesis of a DAC for a widely applicable specification in the Personal Computer (PC) arena. Today's PCs typically offer resolutions to 1280 x 1024, with screen refresh rates of 72 MHz, and color depth of 256 levels. When the video blanking intervals are included, this translates into a DAC specification of 8-bit resolution at 135 MS/s. Video systems typically use 75 Ω impedance levels, with a standard voltage range [ADI92]. When this DAC is used to drive a doubly terminated 75 Ω line directly the full scale DAC current is a nominal 17.6 mA. In this section the synthesis setup, optimization, and results from devices fabricated in a 1.2 μm process will be summarized.

6.7.1 Synthesis Setup

The DAC module generation problem is separated into a set of constant inputs and constraints, based on the DAC input specifications. These are in addition to the 1.2µm technology inputs discussed earlier in the chapter. They are:

Table 6.8 DAC Synthesis Design Constants from input specifications and technology data.

Constant	Value	Units	Comment
Resolution	8	bits	
Clock Rate	135	MS/s	
Power Supply	5.0	Volts	+/- 0.25
Full Scale Current	17.6	mA	+/- 3%
Nominal Load	37.5	ohms	
Cload	20	pF typical	typical
Bond Wire Inductance	5	nH	typical
Compliance Range	+1	Volts max	
Clock Rise/Fall Time	2	ns	at module input

Other design specifications are integrated into the list of optimization constraints:

Table 6.9 DAC Synthesis Design Constraints.

Constraint	Limit	Units	Comment
INL	≤ 1	lsb	3 σ
DNL	≤ 0.5	lsb	
Gain Error	≤ 2	lsb	
Total Error	≤ 4	lsb	
R_{out}	>10	kΩ	
PSRR	< 0.01	%/%	at 1 kHz
Delay Time	< 5	ns	
Rise/Fall Time	< 3	ns	
Settling Time	< 13	ns	
Glitch Energy	< 1	lsb * T_{settle}	33 pVs
Vds margin	> 0.2	Volts	M1, M2, M5, M9

Table 6.9 DAC Synthesis Design Constraints.

Constraint	Limit	Units	Comment
Vdsat margin	>0.4	Volts	M7
4 other constraints to force feasible solutions			
6 Digital Latch Setup Time < 1ns constraints			
12 Digital Signal Level Integrity constraints			
W1 < W2 constraint[a]			

a. This constraint was added because it removed a gross non-linearity from the objective function, and all observed optimizations met this condition anyway.

The 20 design variables described in section 6.2 are used in the MINLP optimization which meets constraints and minimizes circuit area. All variables are expressed as integers -- architecture variables must be integer or power of 2, and layout dimension variables must be placed on a lambda = 0.6 μm grid.

This optimization problem, with 20 variables, minute long estimation steps, is too large to solve in a reasonable time on a modern engineering workstation. Fortunately it is relatively easy to separate the simulation into two, largely independent parts, and it makes intuitive sense to have separate optimizations for each part.[1] The problem is split at the cell latch, into "analog" and "digital" optimizations. The digital optimization sizes the digital row and column buffers, and the logic decode, making certain that digital signal integrity constraints are met. The analog optimization starts when the select signal is clocked out of the cell level latch, and initiates a switched DAC output. It includes the AC and DC analog simulations required for static linearity, output resistance, and PSRR. The two simulations are summarized in table 6.10.

Table 6.10 Optimization split into digital and analog optimizations.

	Digital Optimization	Analog Optimization
Variables	Digital Buffer, decode, and latch device sizes	All analog device sizes. Digital inverter sizes. DAC architecture vars.

1. The size of the problem goes up (first order) with the product of the number of variables and constraints, so splitting into two equal parts should reduce the problem by into two problems, of approximately 1/4 the original size

Table 6.10 Optimization split into digital and analog optimizations.

	Digital Optimization	Analog Optimization
Constraints	Signal integrity constraints	Static and Dynamic DAC performance constraints. Bias constraints
Objective	Digital Circuit Area	DAC Circuit Area
Typical Run Time	1 hour	5 hours

In this process there are significant dependencies from the analog optimization to the digital one. In particular, the analog optimization determines the loading on the final signal nodes in the digital problem. In the other direction there are some second order effects, due to small percentage changes in segment cell size as the digital circuits are sized, but this is a relatively insignificant factor. In practice they are run sequentially -- the analog optimization is run first, and the solution is used to set the loading in the succeeding digital optimization.

6.7.2 Synthesis Process

Once the design has been described, the synthesis process can proceed, using the optimization algorithm described in chapter 4 with the constraints and DAC specific simulations and analysis described previously in this chapter. In practice this process is an iterative one, with initial optimization results requiring a full circuit extraction and verification simulations. Once the module is verified, it can be further integrated with other chip elements. In this section the actual process resulting in this completed design is described.

Initially, the circuit optimization was run without the completion of the DAC cell library, using estimates of circuit parasitics based on proposed device source and drain merging, and estimates of subcell sizes. The 135 MS/s spec was used, and it was found that the optimal value for M with this spec was 4. A square aspect ratio was desired, and for the proposed cell layout this resulted in a DAC module with 4 rows of 16 DAC segment cells. The layout cell library was implemented for an M=4 segment, and further optimizations using parasitics and cell sizes based on actual layout were run. Other parts of the chip design were implemented in this timeframe, including the external digital row and column encoding circuits.

When the external circuits were implemented, initial simulations from the extracted layouts indicated that the 4 to 16 column select encoder was not fast enough for the 7.4 ns clock period, especially when using the slow process file. Two iterations of the design were done in an attempt to speed up the circuit, but these did not solve the problem. Since the purpose of this research is not to develop fast custom digital circuits, but to demonstrate module generation techniques, the DAC specification was relaxed to 100MS/s, a speed which these circuits could meet.

When the 100MS/s spec was defined, the circuits were optimized again, this time forcing M=4 to utilize the existing cell library. Simulation results were passed to the layout programs, for automated DAC module layout generation. Then circuit parasitics were extracted from layout for full simulation. In this pass the extracted circuits did not pass digital data through the cell level latches as quickly as predicted in circuit estimation. The discrepancy was found in a 12 fF parasitic wiring capacitance which had not been included in optimization. The optimization was re-run, this time with better simulation accuracy and a better understanding of the signal integrity constraints.

A final optimization problem was solved at the end of the design process, when it was observed that the optimizations were behaving poorly with a non-linear objective function. After the adding the last constraint in table 6.9, which removed a corresponding nonlinearity from the objective function, optimizations converged more quickly, because the linear approximation to the objective used in the optimization algorithm was more accurate.

Before the final release of the chip to the fab, additional chip level design techniques were used to reduce the supply ringing problems. The final implementation used multiple bond wires for all supplies, and separate digital, analog, and substrate supplies, in an attempt to reduce digital noise in the signal output.

6.7.3 Results

This DAC module was fabricated in the Orbit Semiconductor 1.2 μm process, using single poly and double metal layers. The chip has 4 mm^2 active area available, of which this DAC module uses 0.71 mm^2. It is packaged in a 44 pin LCCC. Fig. 6.14 is the die photograph for this part. 12 parts were fabricated, and all were functional.

Static performance was measured using a computer controlled setup, in which the computer sets the DAC input, and measured the output through a 5 digit voltmeter. In this measurement the total measurement time for all codes may be significant, so care must be taken to minimize the effects of temperature variation in external components during the measurement process. A temperature compensated bias current was implemented on the test board, and a butterfly sampling pattern, similar to symmetrical switching patterns, was used to time average the current flowing into the external load resistor, reducing its temperature variation over the course of the measurement. Full scale and midscale readings were taken at the beginning and the end of the test cycle to

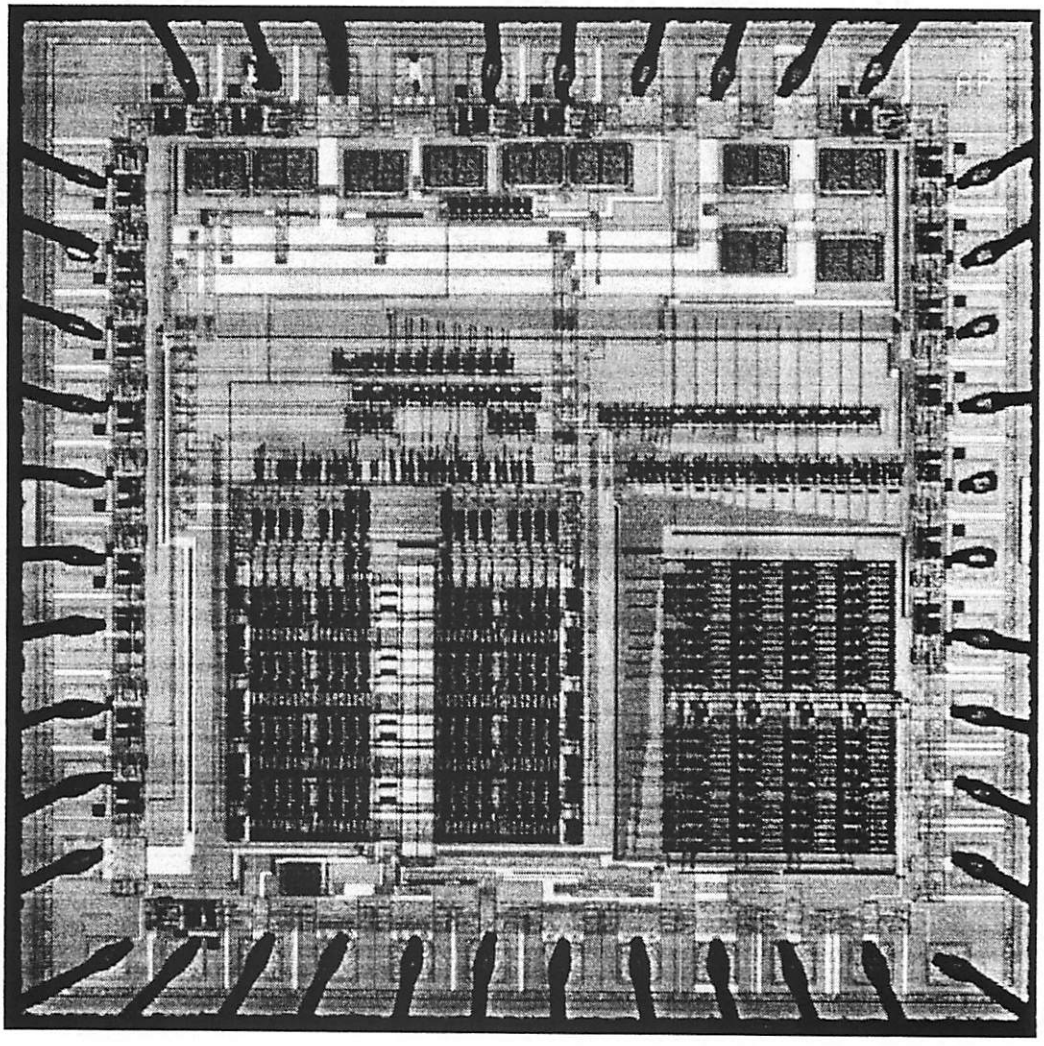


Figure 6.14 Die Photo for DAC test chip. 8-bit part in lower left, 10-bit part in lower right.

check for measurement drift. Observed drift was less than 0.05 lsb for these tests. A plot with all INL curves is in Fig. 6.15. Worst case INL and DNL plots of the 12 samples are in Fig. 6.16.

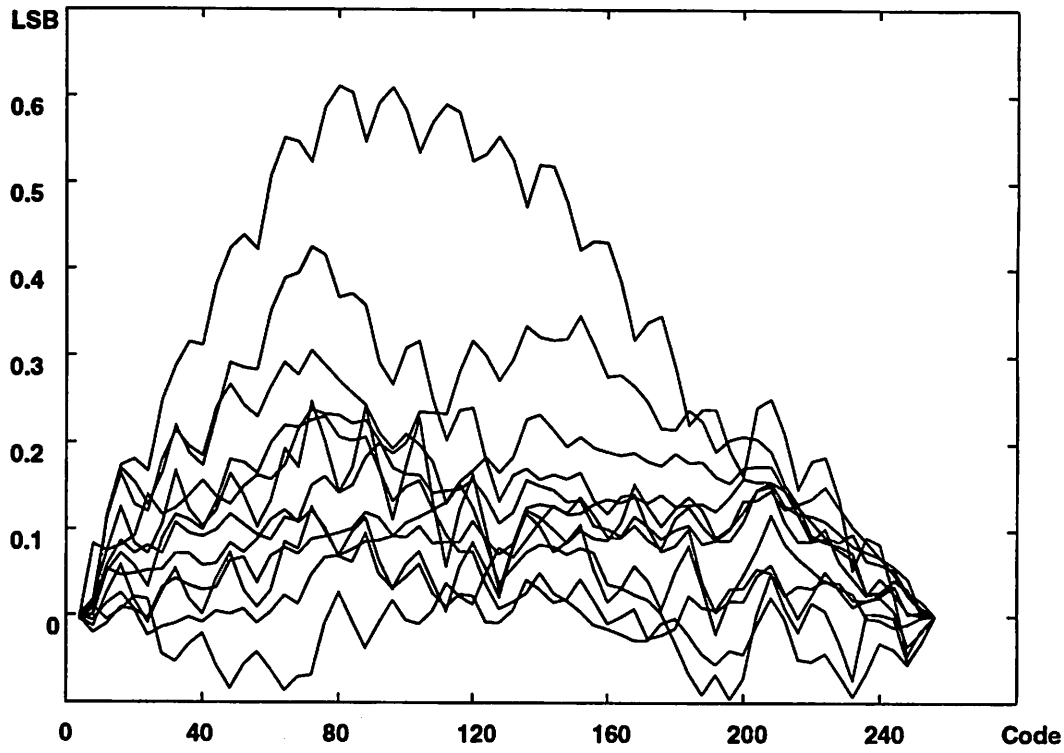


Figure 6.15 INL plots for all test chips.

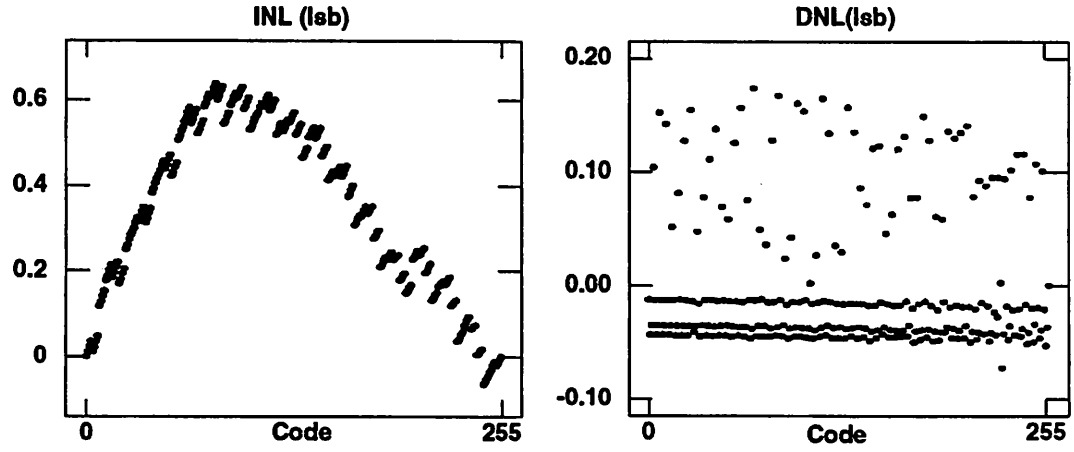


Figure 6.16 INL and DNL plots for the 8-bit DAC. (Worst of 12 parts).

All dynamic performance specifications used in video applications occur at a transition between two levels, so integrated on this chip is a data toggle circuit which will alternate between two 8 bit inputs, controlled by the DAC clock. This allows static data inputs for high speed tests, reducing on chip noise. For a full scale transition, 0 and 255 are applied to the inputs, while for glitch energy the desired code transition is applied. The DAC also has a digital test output which is at $F_{clk}/2$, and is useful as a trigger when the output difference is small. The test setup was designed for the DAC output to terminate on board to the 75 ohm line impedance, and drive a 75 ohm coax to a Tektronix DSA 602 digitizing oscilloscope. At the DSA, the an external termination is used, and the DSA is connected in a high impedance mode. When this setup was used the impedance mismatch at the DSA resulted in reflections in the signal. To get a cleaner signal the system was converted to a 50 Ω impedance, using the internal 50 Ω termination in the DSA. This gave better impedance matching in the measurement setup. As a result of this change in the impedance levels, the dynamic switching and settling performance should be improved by a factor of 2/3, so measurements taken were derated by a compensating factor of 1.5.

Measurement of full scale transitions were read off the DSA. For settling time the settling to within 1 lsb of the final value could not be measured, because of digital noise coupling, and settling to within 2% of the final value was measured instead. The switching waveform is shown in Fig. 6.17a. To observe glitch energy, two waveforms were digitized, one with the glitch, and one with DC input at the same value. The difference waveform subtracts out the coherent digital noise, and the glitch waveform can be observed in Fig. 6.17b.

Table 6.11 summarizes the performance of the synthesis process and the performance of this test chip. It lists specification inputs for synthesis, the estimated performance, and then the measured results for this small sample. There is good agreement between these predictions and measured results. The static linearity data has been investigated more thoroughly, indentifying the sources of nonlinearity seen in the data with deterministic and stochastic nonidealities. It was found that measurable INL was caused by adjacent layout specific mismatch, between segments at the edge of the array and those in the main body, and between lsb elements and segment elements. Process gradients were twice the value predicted from the literature search. There appears to be more variation in the first level metal sheet resistance than indicated by the technology description.

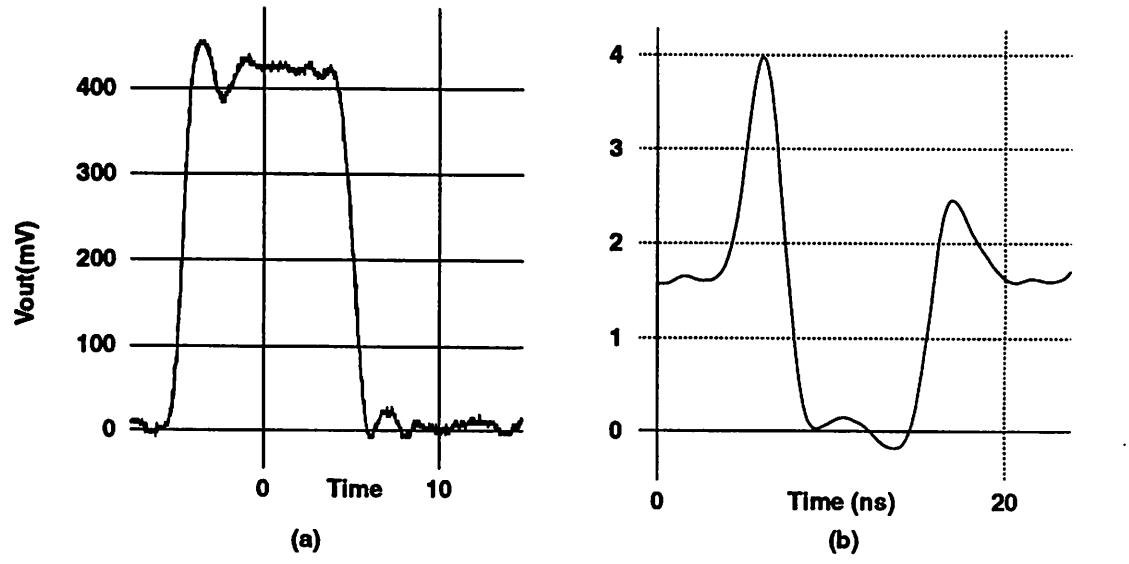


Figure 6.17 Typical switching and glitch waveforms for 8-bit DAC. Measured driving 50-Ω doubly terminated transmission line to the DSA.

Details of this analysis may be found in the appendix to this chapter. When the data was taken and results analyzed, some errors were discovered in the analysis, and a factor had been left out of the gain error computation. In the estimated results column of the table, the value in parenthesis is the original erroneous estimate.

Table 6.11 Measured Performance of 8 bit video DAC

Name	Specified	Estimated	Measured	Units
INL / DNL	< 1.0	0.53 (0.59)	0.64 max 0.31 avg	lsb
DNL	< 0.5	0.16 (.06)	0.18 max 0.10 typical	lsb
Gain Error	< 2	3.6 (1.5)	3	lsb
Total Error	< 4	3.6 (1.9)	3	lsb
Tsettle (rise / fall)	< 13.0	7.4 / 4.43	5.5 / 5.25	ns
Tswit (rise / fall)	< 5.0	1.6 / 1.0	1.8 / 1.1	ns
Tdelay	< 5.0	4.75	NA[a]	ns
Glitch (rise / fall)	< 1.0	0.77 / 0.62	0.74 / 0.46	lsb*Ts
Rout	> 10	10.5	> 12.5	kΩ
Clock Rate	> 100	101	225 max	MS/s

a. Not measurable.

In this design the dominant cause of nonlinearity is finite output resistance of the current sources. In the typical designs, this is half of the total INL. All other sources of error tend to look like a voltage mismatch at the gate of current source devices, and the high current densities used to minimize design area tends to reduce the effects of voltage mismatch. Fortunately this limited the effects of the unexpected mismatch and gradient magnitude errors in this design. 11 of 12 devices had less than the maximum estimated INL and DNL errors, even in the presence of these unmodelled nonidealities.

6.8 Design Example 2: 10-bit Instrumentation DAC

A second test DAC was built, using some of the remaining space on the die. A 10 bit specification, with lower full scale current and no dynamic specifications was developed. That is summarized in table 6.12 and table 6.9. The elimination of the dynamic performance specs meant that the latches and buffers could be removed from the layout, reducing cell size. A single optimization, similar to the "analog" optimization for the 8-bit DAC, was run, but without the transient simulations. Simulations ran quickly for this problem, with results in less than an hour. The optimizations predicted a minimum sized design for M=32, but with this module layout and available cell library M was limited to $M \leq Cols$, and Cols = 16 was chosen for aspect ratio reasons. The layout is in the lower right corner of: Fig. 6.14.

Table 6.12 10-bit DAC Design Constants.

Constant	Value	Units	Comment
Resolution	10	bits	
Power Supply	5.0	Volts	+/- 0.25
Full Scale Current	4.0	mA	+/- 3%
Nominal Load	256	ohms	
Compliance Range	+1.1	Volts max	

Table 6.13 10-bit DAC Design Constraints.

Constraint	Limit	Units	Comment
INL	≤ 2	lsb	3σ

Table 6.13 10-bit DAC Design Constraints.

Constraint	Limit	Units	Comment
DNL	≤ 0.5	lsb	
Gain Error	≤ 4	lsb	
Total Error	≤ 8	lsb	
R_{out}	>20	kΩ	
Vds margin	> 0.1	Volts	M1, M2, M5, M9
Vdsat margin	>0.2	Volts	M7

The INL and DNL specifications were not met when this DAC was fabricated. As in the first design, some INL and DNL causing effects were not predicted ahead of time, but in this case the design optimization process tended to reduce the current density in the current sources, reducing circuit area, but increasing sensitivity to effective gate voltage mismatch. In a second pass these sources of mismatches may be better quantified, and the design re-optimized to meet specifications. The INL and DNL data is summarized in table 6.14.

Table 6.14 Measured Performance of static 10-bit DAC

Name	Specified	Estimated	Measured	Units
INL	2.0	1.87	2.1	lsb
DNL	0.5	0.16	1.6	lsb

This performance sensitivity to an unmodelled effect points to a problem with this methodology, which is a general synthesis problem as well. In analog design there is often some second or third order effect which is known to exist, but is difficult to analyze or simulate. How does one develop a module generator which is robust in the presence of poorly modelled or unknown process and fabrication effects? Often the approach taken is to overspecify the problem, to force the result to tighter constraints than original specification. The problem is that if the source of error is not part of the performance estimation process, then the optimized design may meet the new, tighter, specification, yet remain very sensitive to the unmodelled effect, and still not meet specs in fabrication. An approach which these DAC results suggest is to model these unknown effects as an estimation input, in this case as a user defined maximum gate offset voltage input, and force the optimization to meet design requirements despite gate voltage errors of that magnitude, in addition

to the currently estimated nonlinearities. In this technology it appears that this general error voltage constant would be on the order of 12 mV, based on mismatches found related to unmatched adjacent device layouts. For each module generator, and perhaps each technology, these unclassified sources of errors may have to be re-calibrated.

6.9 Module Generator Development and Synthesis Time

As discussed in chapter 1, implementation design time is an important metric for any analog design, so it must be a metric used when discussing the application of this methodology. It is difficult to strictly classify the time spent in this research as module design time, tool development, or algorithm development time, because all three occurred simultaneously for much of this research. As long as the optimization process finds an optimized, constrained solution in a few hours, it is not the limiting factor in minimizing design time. The most time consuming parts of the module design are the design analysis and input time for optimization, the cell library development time, and the technology analysis and entry time. In this section the time required to develop a new module generator, and to reapply an existing tool to different specifications or technology is discussed.

6.9.1 Module Synthesis Development Time

Module synthesis development time is the dominant design time for a new module generator implementation. This includes the typical circuit design tasks of developing the circuit topologies for the application, and finding simulation and analysis methods to estimate design performance, and additional synthesis related tasks of specifying design variables, automating design estimation, and integrating estimates of parasitics into design estimation. The synthesis process will find optimized design variable values, a time savings, but this is offset by the need to explicitly include all design analysis in the estimation framework, and develop additional tests to force the design to maintain bias margins. As a result, initial module synthesis development time is the same order of magnitude as manual design. Designer time savings occur in subsequent design cycles, when the same design is re-implemented with new specifications or technology.

6.9.2 Layout Cell Library Development Time

A new cell library may be required when a module design is implemented, or a new technology is used[1]. It is advantageous to adapt pre-existing cell designs whenever possible. Layout of a cell library for a new module design may take 2 weeks, including the layout of important module cells, and verification that cell stretching and tiling results in properly connected cells. When an existing cell library can be adapted to a new design, the cell library design time shrinks to hours or a few days at most.

6.9.3 Technology Analysis and Input Time

Migration to a new technology requires the input of new technology constants, and verification or fitting of MOS device models. For DACs estimates of device mismatch constants are needed. Ideally this is a zero time step, requiring just the simple substitution of one well documented technology description file for another, and using scalable design rules to reuse the layout library in the new technology.

6.9.4 Example Implementation Times

In the 8-bit DAC example described in this chapter, the basic form of the DAC was inherited from an earlier prototype, but most of the design implementation was completely new, including new low level circuits, new estimation methods for settling and glitch, new layout cells and parasitic computations, and new design parametrizing. From starting with design specifications to tape-out of the finished test chip took 6 months, but this time was not exclusively devoted to this 8-bit DAC module. Subtracting out time spent implementing layout automation algorithms, implementing the 10-bit DAC, chip level design and implementation of digital circuits, analysis of chip level noise and coupling issues, leaves an estimated 3 months for module generation.

The 10-bit design example re-used the static linearity analysis from the 8-bit design, and modified the cell library, removing latch and buffer circuits. It used the same technology as the 8-bit design. Implementation time for this DAC module was about 2 weeks, with initial optimizations

1. When a scalable layout style is used, such as the scalable CMOS layout rules supported by MOSIS in technologies from 0.8 μm to 3 μm, then new cell libraries are not required for a technology shrink, but may be advantageous when the technology shrink includes a new metal routing layer.

done first, and then required library cells developed for a design with M=16 unit current sources per segment. Back end verification was also done in this time.

A third example design took the existing video DAC specification, and migrated it from 1.2 μm design rules to a 0.8 μm technology. Both technologies could be described with scalable CMOS design rules, so the cell library could be re-used. The majority of the implementation time was spent sorting out the new technology inputs, estimating errors in the device fits, process variation, and mismatch constants. This was concurrent with completion of module synthesis libraries, so this design time was spread out due to instability in the underlying synthesis libraries. Once the libraries were stable, the design was completed in 1 week, with most of that time spent waiting for optimization and back-end verification simulations to complete.

A last example was a re-specification of an existing module generator for a 7-bit, 5MS/s application. In that case the same module libraries and technology files could be re-used, and no changes were required. The design synthesis was completed over the course of a day, with DAC specifications determined in a brief meeting in the morning, the optimization running over the course of the afternoon, and a finished layout by evening.

6.10 Chapter Summary

In this chapter the application of the DSYN tools for design and layout synthesis to a specific DAC module architecture have been described, including the specific circuits, the complexity of the optimizations, the implementation of these circuits in a 1.2 μm technology, and the fabricated results.

The most important conclusion is that the process works. The 8-bit part met all static and dynamic specifications on the first pass, and over 90% of the test chips fell within the predicted worst case specifications. It is unclear if the outlier is due to a random deviation, a defect, or poor process data. The 10-bit DAC did not meet specifications on this pass, but corrections to estimated process gradient inputs and elimination of DAC layout mismatches would fix these problems on a second pass.

The difficulties in quantizing all possible sources of non-idealities point to a larger problem in performance estimation, due to poorly modelled or "unsimulatable" inputs. These results suggest that these are best considered as a nonideality added to process model inputs, rather than as an overconstraint on the final design.

Design time for this process has demonstrated that synthesis is not a time saver for an initial design, but is a significant one when a module is to be re-implemented with a new specification, in a new technology, and even when the circuits are slightly changed, but the most of the design input analysis can be reused.

6A Appendix: Estimated and Actual Causes of Nonlinearity

The estimated and actual INL and DNL for the 8-bit DAC are investigated here. The causes and their respective contributions to overall non-linearity are listed. Three sources of measurable INL and DNL have been found which were not included in the estimates used in DAC synthesis. The worst case process gradients seen are much worse than predicted in the literature, and there is evidence that there is a wide variability in metal 1 sheet resistance as well, though this is more difficult to separate from random errors.

6A.1 Nonlinearity Contributors.

Table 6A.1 lists causes of nonlinearity, and their predicted maximum effect on INL and DNL in synthesis. All errors are expressed in lsb at 8 bits, for the specific 8-bit design implemented in section 6.7. These errors are of two types -- stochastic and deterministic. The DAC transfer function available from 12 test chips can be analyzed to see if the causes of circuit non-idealities can be tracked to the observed DAC INL and DNL. In this work it is best to concentrate on the 63 DAC segments, and assume the lsb current sources perform with zero mismatch relative to the nearest DAC segment. After the INL is measured, as seen in Fig. 6.15, the first step is to take an average of all INL curves, seen here in Fig. 6A.1

Table 6A.1 Summary of predicted static linearity - 8 bit DAC units in lsb

Cause	Effect on INL	Effect on DNL
Finite Rout	.24	0
Resistive Drops	.06	0.02
Random Mismatch	0.19	.07
Mismatch Gradients	0.04	.07
totals	.53	.16

6A.2 Deterministic Effects

This average INL is dominated by two components -- a quadratic bow non-linearity due to Rout, with average magnitude 0.15 lsb, and an S shaped sawtooth due to resistive drops between the rows of the DAC, with maximum predicted amplitude of 0.04 lsb on this plot[1]. There is

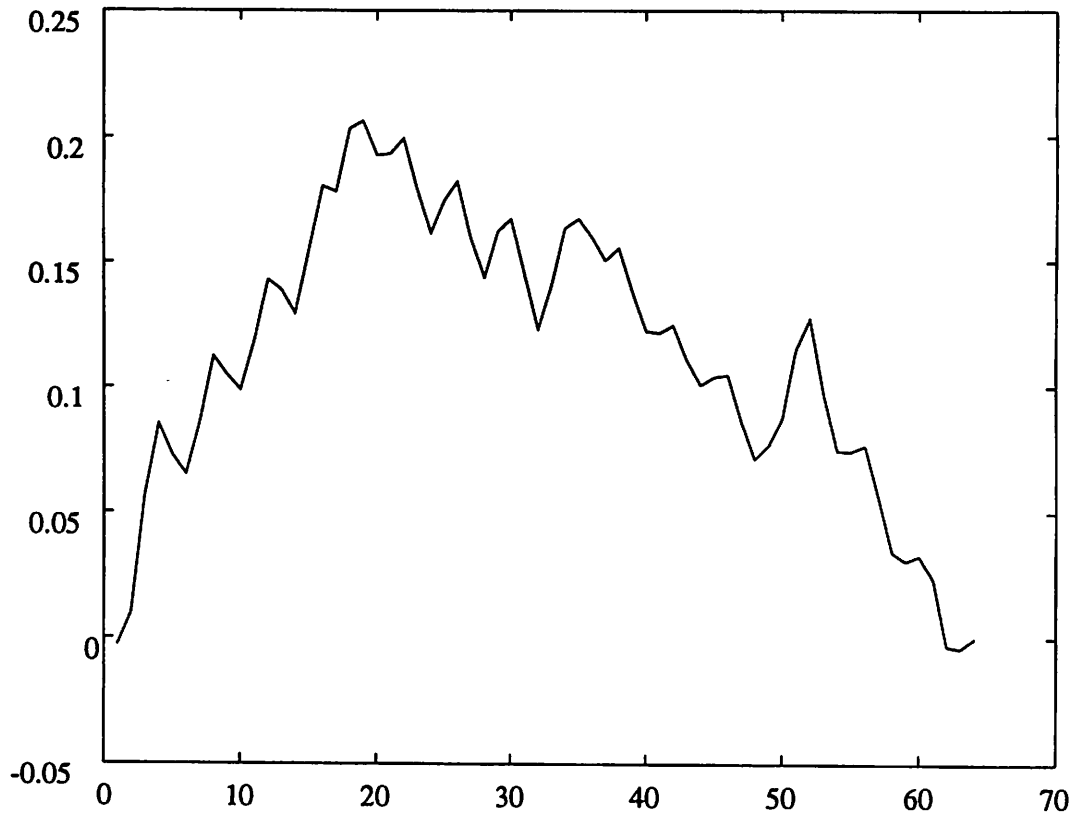


Figure 6A.1 Average 8-bit DAC INL.

another effect, which is best seen by taking the 64 average segment currents, and plotting them as a function of their column location on the DAC. This is done in Fig. 6A.2. Ideally this should be 4 parallel lines, due to resistive drops between rows, and this effect is seen, though it is complicated by the left-to-right variation due to the combination of switch ordering and output resistance. The unexpected effect is the variation at the ends of the array, in columns 1 and 16, of +0.025 lsb on average. The best explanation for this is a device mismatch, due to different adjacent circuit layouts between the edges of the array and the main body of current source segments. This average edge effect is magnified by the symmetrical switching method, which chooses columns 1 and 16 consecutively, resulting in sawtooth nonlinearity of magnitude 0.05 lsb, and period of 1/4 full scale. A similar layout induced mismatch was seen between the segments and the lsb current sources, with a resulting mismatch of 0.02 lsb between the average lsb current source and the average segment.

1. This effect increases to 0.06 lsb when the lsb sources are included in the analysis, as in table 6A.1.

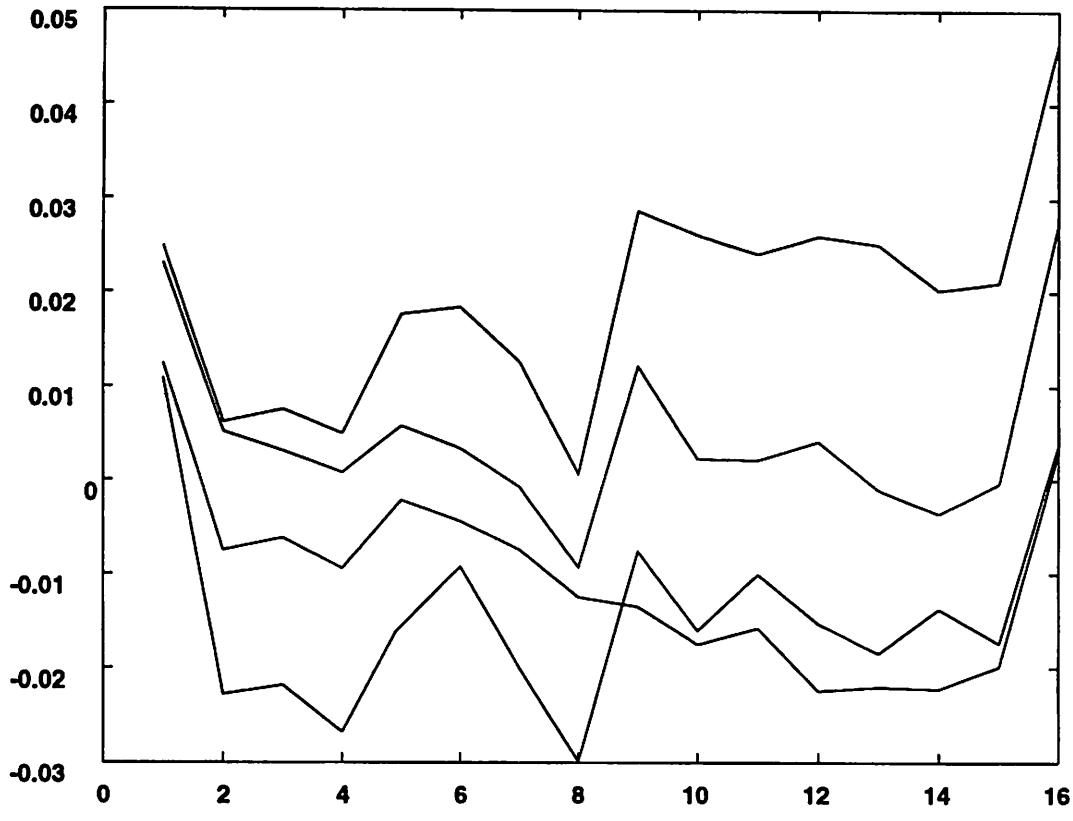


Figure 6A.2 Average Segment Current vs. Column Location.

6A.3 Stochastic Effects

The stochastic effects can be divided into two parts -- random mismatch, and random gradients. The analysis approach taken was to look at plots of segment current source values, with the average subtracted out, and look for the magnitude of any gradient seen. Fig. 6A.3 shows some representative plots from 4 of the test DACs. The estimate of random variation of current sources for this design was a current source standard deviation of 0.01 lsb. The observed variation certainly has σ less than 0.02 lsb. It may be as low as 0.01 lsb -- it is difficult to separate measurement noise from actual mismatches in the data taken here. A conservative designer may want to use twice the random mismatch factor in the future. The predicted gradient across the array should give a 3σ mismatch of 0.08 lsb, but judging from this data, a better 3σ limit would be 0.14 lsb, or roughly twice the gradient mismatch initially estimated.

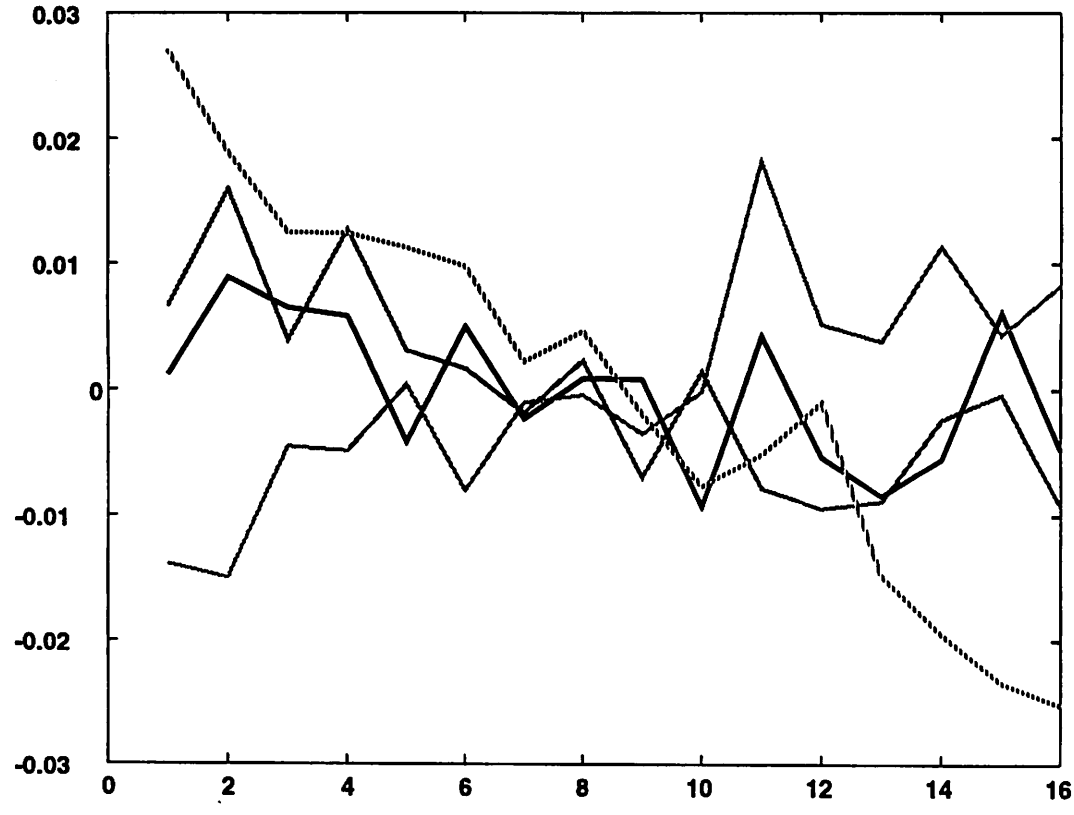


Figure 6A.3 Observed current source values across columns of the DAC, for 4 different test devices. A combination of random gradients and device mismatch are seen.

These additions and corrections to the nonlinearity inputs can be used to recompute estimates of INL and DNL. Table 6A.1 is the same as table 6A.1, with the addition of a column for corrected and new causes of nonlinearity. With these additions the estimates bracket the observed INL and DNL. [1]

1. The INL seen in the worst case transfer function is equal to this worst case estimate, but the combination of effects seen in that part does not correspond to this model exactly. In particular, the assumption that gradients are linear, and perfectly cancel should be re-visited. A slight curvature in the threshold voltage gradient would explain the additional INL contributions in that chip.

Table 6A.2 Summary of predicted and observed static linearity. Units in lsb. Shaded cells indicate corrections.

Cause	Effect on INL	Effect on DNL
Finite Rout	.24	0
Resistive Drops	.06	.02
End Segment Layout Mismatch	.05	.025
Segment/LSB Layout Mismatch	.02	.02
Random Mismatch	.19	.07
Mismatch Gradients	.07	.14
totals (original total)	.63 (.53)	.28 (0.16)
Observed Max	.64	0.21
Avg	.30	.10

CHAPTER 7

Conclusion

7.1 Summary of Research Results

This research has demonstrated the computer aided synthesis of high performance CMOS digital/analog converters. The combination of several factors made this possible.

- A DAC circuit estimation process was developed which emphasized accurate performance prediction from circuit, topological, and technology inputs.
- The use of back-end performance calibration factors was avoided, in favor of front end corrections to modelling, process or matching parameter errors.
- A combination of circuit analysis and full circuit simulation was found to be a flexible framework for all design estimation needs. Multiple simulations were used to simultaneously find static and dynamic behavior at worst case design corners.
- A new Mixed Integer Non-linear Programming algorithm was implemented, for efficient optimization with this estimation method. This algorithm uses a cutting plane method to create linear constraints for a fast mixed integer, linear programming subproblem, and avoiding a direct combination of a mixed integer solution with nonlinear

constraint functions. The optimization algorithm completes in several hours, even with function evaluation times on the order of a minute.

- The layout synthesis approach is matched to typical DAC layout styles, using a combination of tiling and stretching algorithms to complete the DAC module. The completed module has > 80% of the density of an ideal manual design.

- Tight coupling of the deterministic layout process with circuit performance estimation allows accurate parasitic prediction in the estimation phase. Parasitic capacitances as small as 12 fF have been important in design simulations, so this coupling is necessary.

- Implementation of a new DAC module design takes several months, a period comparable to manual design. Re-targeting to a new specification, new technology, or even a modified architecture is very fast, with several examples requiring less than 1 week.

- The DAC synthesis process has been demonstrated with 8-bit video DAC and 10-bit resolution prototypes. These test designs have validated the synthesis process.

7.2 Barriers to Acceptance of Analog Synthesis CAD tools

In the course of this work the question of the application of analog CAD techniques to real world situations has come up many times. It is worthwhile to look at where innovations in CAD for analog circuit designers are moving forward rapidly, and at the barriers which prevent the current widespread use of analog synthesis CAD tools

7.2.1 Where is analog CAD successful today?

Analog CAD tools are used everywhere analog circuit design occurs. The typical circuit designer uses a set of sparse, problem specific tools in an ad-hoc way, developing circuit and system simulations with a combination of circuit simulation (SPICE), behavioral simulation (SPICE or Ptolemy), specialized simulation tools (SWITCAP), or self-written programs. Circuit layout methods use a similar set of sparse tools, aiding the user in an essentially manual methodology. Improvements continue all the time with these tools. The capability for circuit simulation is larger

every year, the inclusion true mixed signal simulation and behavioral simulation with analog subcircuits increases the users' capability. Optimization linked to circuit design is now used to quickly size individual circuits under user control. Graphical interfaces promise to make circuit design as we know it more efficient and less error prone. Layout automation to simplify element creation, automate routing, and automate digital cell layout is proceeding.

All of these advances maintain the existing philosophy of giving the user greater power to simulate and understand design options, and speed design entry, but do not fundamentally alter the way the engineer specifies the design inputs, and expects the tools to accurately predict performance.

Circuit synthesis tools take a fundamentally different approach, seeking to create designs automatically to meet user specifications. While synthesis tools have gained wide acceptance in digital circuits, they have still seen little use by analog circuit designers.

7.2.2 Limitations to today's synthesis tools

The most important limitation to widespread acceptance of analog synthesis tools is the limited capabilities of the tools themselves. There limitations include barriers to the input of new designs in a synthesis framework, the incomplete design solution which many current synthesis approaches provide, and the incomplete technology database available to synthesis.

When a new circuit architecture is to be synthesized, someone, usually an analog circuit designer, must input the elements of the design required for synthesis. This is a time consuming task for any real design. At minimum, the synthesis designer must specify all design constraints, and the methods for performance estimation. He must make some choices for parametrization of the design. An unfamiliar input format will make this task more difficult. As a result of these factors, the design input process requires a user with considerable design expertise, but may take a considerable amount of time. The design input process may be error prone, especially when analytic methods which cannot be checked through simulation are used. Though some tools have been developed with these issues in mind, it is difficult to see how this problem can be completely avoided when real, complex module synthesis is done.

Recent synthesis approaches have been oversold, advertising a complete synthesis solution while solving simplified examples[1]. In many cases synthesis methods are demonstrated using simple opamp circuits, demonstrating AC and DC performance with nominal models, inputs, and circuit loads. Real designs typically use more complicated circuit topologies, usually require transient analysis, and must be tested over a full range of temperature, process, input range and loading. These additional requirements make circuit design significantly more difficult. In practice obtaining a topology which meets performance specs under nominal conditions may be done rapidly, but consideration of all process corners increases design complexity by an order of magnitude. No body has demonstrated a complete worst case design, using an optimization approach with interesting circuits.[2]

As seen in the error sources described in the previous chapter, it is difficult to comprehensively describe the technology and interactions which affect CMOS circuit performance. Device models may be incomplete, lacking noise and matching information, and may not even fit the devices in important regions of operation. Layout dependent and substrate noise effects are not fully understood at this time. These uncertainties are difficult to incorporate efficiently into synthesis. Ideally, the design must meet specifications, despite these nonidealities, yet not be wastefully overdesigned. The lack of a fully qualified technology database prevents the reliable prediction of circuit performance in the first pass of a design.

Layout synthesis is a bright spot here. While typical analog layout synthesis tools were not the best choices for the DAC synthesis application, the methods developed and currently in research show promise for stand-alone layout synthesis from annotated netlist inputs [CHAR92,COHN91

7.2.3 Analog design culture works against synthesis acceptance

Besides limitations to existing analog synthesis tools, there are elements of the current analog circuit design culture which work against acceptance of analog synthesis.

One of the first rules seen in the practice of analog circuit design is do not change a known good design. Potential improvements to a design are never worth the perceived risk of change.

1. The one exception is IDAC, which was geared toward industrial application, including interesting circuit blocks and worst case analysis, but required long design input times. [DEGR89]
2. This work has some worst case design aspects, but it is not done systematically enough to meet this criterion.

Though some tweaking may make sense to a designer, if the current part and layout meets specifications, do not change it. If the specification changes, try to change as little as possible. If a standard cell meets the circuit requirements, use it. Designers tend to re-use familiar circuit architectures, and specialize them for key system analog blocks. This culture is not receptive to the design style suggested by analog module generator tools, in which designs are implemented to specifications in every case, with reuse of design knowledge through the module generator, but not a reuse of the known good design.

A second cultural issue is an engineering management one. There are never enough analog circuit designers, and time to market is critical to company profits. If the payoff of an analog synthesis methodology is uncertain, in terms of actual synthesis capability and module generator design reuse, then the extra time spent by circuit designers on first implementations in a module generation methodology are difficult to justify. The inclination is to continue to do designs the old way, and only accept incremental changes to the tools, which will moderately increase analog designer efficiency.

7.2.4 How can this change?

To gain wide acceptance of analog synthesis tools, improvements must occur in existing synthesis tools and methodologies, and industry must be willing to take some risks with these new tools.

Development of synthesis tools must continue, and the limitations identified earlier in this section must be addressed. We must get beyond nominal design of opamp circuits, and address manufacturability, worst case design, and realistic design problems. Testcases must be used which complete the design process through to fabrication, because these testcases are the ones which can convince industrial customers that these synthesis approaches meet their promise.

Industry must be willing to devote resources and analog designers to the problem. It is unrealistic to expect those who specialize in circuit synthesis methods to simultaneously provide all the design expertise required for high performance circuit synthesis implementations. Unfortunately, industry has gone down this road in the past, with well publicized, but unsuccessful efforts aimed at allowing design of complex circuits by relatively naive designers [DEGR89, LABE87]. Since these

have not panned out, analog synthesis does not have the track record that digital circuit synthesis boasts, and there is not the same level of industry trust. Consumers of these tools must keep reasonable expectations: the tools may never help naive designers with complicated circuits, but they do provide a path toward rapid high performance circuit synthesis when used by educated users.

7.3 Future Directions

In the course of this research several areas of need for analog synthesis have been identified.

- A complete technology database for an analog circuit technology has never been strictly defined. Obviously the typical foundry data is not enough for DAC design, as it does not include device matching constants, though this may be changing[MICH92]. What other information should be included? Can all possible variations in devices and process be described? Can layout dependent effects be included? Is it possible to complete such a technology description before designers are ready to move on to the next technology?

- This research has considered module generation as a stand-alone process, but in a system design situation it is only part of a larger system optimization and partitioning process. In that application high level estimates of inverse cost functions (or "flexibility functions") are needed for each sub-circuit. If the synthesis process can be run in an automated way with a variety of design inputs, is it possible to efficiently automate an estimation process for module cost functions?

- In this module generation implementation a limited worst case design approach was used, with the user determining a subset of design corners for design estimation. A full corner simulation includes 2^C simulations, where C is the number of corner variables[1], but is obviously excessive, while an ad hoc method such as the one used here may miss a significant corner. Methods which ensure that an optimized design meets constraints under all conditions is required, without forcing exhaustive simulation.

1. C includes process and temperature corners, and also input specification corners, such as resistive loading ranges and common mode input range.

- When these parts were fabricated, "non-simulatable" effects had an important impact on the total performance. These included noise coupling through the substrate from digital circuits, and layout dependent device mismatch. Is there a good way to abstract circuit, technology, and device non-idealities for inclusion in simulation, without knowing the exact cause of these effects? Can the problem be described so that the effects from non-simulatables will be minimized in the optimization process?

References

[ALLE85] P. E. Allen and E. R. Macaluso, "AIDE2: An Automated Analog IC Design System." in *Proc. IEEE Custom Integrated Circuits Conference*, May 1985, pp. 498-501. [Module Compilers (procedural) and Standard Cell Construction.]

[ALLE86] P. E. Allen and P.R. Barton, " A Silicon Compiler for Successive Approximation A/D and D/A Converters," *Proc. IEEE Custom Integrated Circuits Conference*, 1986, pp. 552-555.

[ADI92] "Data Converter Reference Manual, Vol. I," Analog Devices Inc, 1992.

[BAST91] C.A.A. Bastiaansen, D. Groeneveld, H. Schouwenaars, and H. Termeer, "A 10-b 40-MHz 0.8μm CMOS Current-Output D/A Converter." *IEEE Journal of Solid State Circuits*, SC-26(7), July 1991, pp. 917-921. [Demonstrates 10-bit matching of current sources using a large binary weighted array of large devices, closely spaced in a 2d layout. ERB (effective Resolution Bandwidth) = 5 MHz.]

[BERK88] E. Berkcan, M. d'Abreu, and W. Laughton, "Analog Compilation based on Successive Decompositions," in *Proc. ACM/IEEE Design Automation Conference*, 1988, pp. 369-375. [An_Com, a top down approach to analog design automation. Successive decomposition allows working down from a high level without exhaustive simulation. Required behavioral or macromodel circuit representations.]

[BRAY81] R.K. Brayton, G.D. Hachtel, and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated Circuit Design," Proceedings of the IEEE, 69(10), October 1981, pp. 1334-1364. [Summarizes optimization for nominal and statistical design]

[BULT92] K. Bult, and G.J.G.M. Geelen, "An inherently Linear and Compact MOST-only Curent Division Technique." *IEEE Journal of Solid State Circuits*, SC-27(12), Dec. 1992, 1730-5. [An R-2R structure of MOS transistors creates a linear current division.]

[BURI85] M.R. Buric and T.G. Matheson, "Silicon Compilation Environments," *Proc. IEEE Custom Integrated Circuits Conference*, May 1985, pp. 208-212. [silicon compilation for digital from functional to geometric.]

[CADE94] *Design Framework II User Manual*, Cadence Design Systems, 1994.

[CHAN92] H. Chang, A. Sangiovanni-Vincentelli, et al. "A top-down, constraint-driven design methodology for analog integrated circuits," *Proc. IEEE Custom Integrated Circuits Conference*, May 1992, pp. 8.4.1-6.

[CHAN94] H. Chang et al, "Top-Down, Constrait-Driven Design Methodology Based Generation of n-bit Interpolative Current Source D/A Converters," *Proc. IEEE Custom Integrated Circuits Conference*, May 1994, pp. 15.5.1-4.

[CHAR92] E. Charbon, E. Malavasi, and A. Sangiovanni-Vincentelli, "A Constraint Driven Placement Methodology for Analog Integrated Circuits," *Proc. IEEE Custom Integrated Circuits Conference*, May 1992, pp. 1711-1714.

[CHEN89] C.C. Chen and S.L. Chow, "The Layout Synthesizer: An automatic Netlist-to-Layout System," *Proc. ACM/IEEE Design Automation Conference*, 1989, pp. 232-238. [Digital cell builder]

[CHIE94] G. Chien, private communication. [In a low power ADC application, gate capacitance is poorly modelled by HSPICE with the level 28 model, especially for Vdsat < 100mV.]

[CHOU90] U. Choudhury and A. Sangiovanni-Vincentelli, "Use of Performance Sensitivities in Routing of Analog Circuits," *Proc. International Symposium on Circuits and Systems*, May 1990, pp. 348-351.

[COHN90] J.M. Cohn, D.J. Garrod, R.A. Rutenbar, and L.R. Carley, "New Algorithms for Placement and Routing of Custom Analog Cells in ACACIA," *Proc. IEEE Custom Integrated Circuits Conference*, May 1990, pp. 27.6.1-5. [SA algorithms applied to placement, with designer input rules for matching, parasitics. Allows more free-form layouts, esp. device merging.]

[COHN91] J.M. Cohn, D.J. Garrod, R.A. Rutenbar, and L.R. Carley, "Techniques for simultaneous placement and routing of custom analog cells in KOAN/ANAGRAM II," *Proc. International Conference on Computer-Aided Design*, Nov. 1991, pp. 394-397.

[CONW92] J.D. Conway, "An Automatic Layout Generator for Analog Circuits," *Proc. European Conf. on Design Automation*, March 1992, pp. 513-19. [Exhaustive slicing structure approach, with size optimization.

[CREM89] A. Cremonesi, F. Maloberti, and G. Polito, "A 100-MHz CMOS DAC for Video-Graphic Systems," *IEEE Journal of Solid State Circuits*, SC-24(3), June 1989, pp. 635-639. [Glitch energy reduced by equalizing on/off delays. Requires digital to differential conversion, using an externally supplied bias voltage (!)]

[CRYS92] "Analog/Digital Conversion ICs Data Book, Volume I," Crystal Semiconductor, 1992.

[DEGR87] M.C. Degrauwe, et al.,"IDAC: An interactive design tool for Analog CMOS circuits," *IEEE Journal of Solid State Circuits*, 22(6), December 1987, pp. 1106-16.[Knowledge based approach creates a formal description which creates device sizes from specs. Hiearchy allows subsystem design.]

[DEGR89] M.C. Degrauwe, et al. "Towards an Analog System Design Environment," *IEEE Journal of Solid State Circuits*, 24(3), June 1989, pp. 659-671. [IDAC 3 + ILAC. IDAC includes synthesis, optimization, and analyzers per architecture.]

[DEWI93] M. de Wit, Tan, K.-S. Tan, , R.K. Hester, "A low-power 12-b analog-to-digital converter with on-chip precision trimming." *IEEE Journal of Solid State Circuits*, April 1993, vol.28, (no.4): pp. 455-61.[Trim DAC section using CMOS compatible fuses.]

[DHAR92] A. Dharchoudhury, and S.M. Kang, "An Integrated Approach to Realistic Worst-Case Design Optimization of MOS Analog Circuits," *Proc. ACM/IEEE Design Automation Conference*, 1992, pp. 704-709. [Finds worst case design file per every design constraint.]

[DURA86] M.A. Duran and I.E. Grossmann, "An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs. *Mathematical Programming*, 36:307-339, 1986. [Solves MINLP problem using a combination of a relaxed MILP master program, and a NLP sub-problem which creates cuts eliminating infeasible solutions.]

[FELT94] E. Felt, A. Narayan, adn A. Sangiovanni-Vincentelli, "Measurement and Modeling of MOS Transistor Current Mismatch in Analog ICs, " *Proc. International Conference on Computer-Aided Design*,1994, pp. 272-277. [Measures device mismatches, finding large process gradient effects.]

[FOTO94] B. Fotouhi, "Optimization of Chopper Amplifiers for Speed and Gain," *IEEE Journal of Solid State Circuits*, 29(7), July, 1994, pp. 823-828. [Demonstrates a circuit optimization without CAD approach.]

[FOUR91] J.M. Fournier and P. Senn, "A 130-MHz 8-b CMOS Video DAC for HDTV Applications," *IEEE Journal of Solid State Circuits*, SC-26(7), July, 1991, pp. 1073-7. [For row-col lookup, describes a row decode source of glitch, and deglitches locally.]

[GADE91] G. Gad-El-Karim and R. S. Gyurcsik, "Use of Performance Sensitivities in Analog Cell layout," *Proc. International Symposium on Circuits and Systems*, 1991, pp. 2008-2011. [Demonstrates the generation of performance sensitivies to layout placement and parasitics.]

[GARR88] D.J. Garrod, R.A. Rutenbar, and L.R. Carley, "Automatic Layout of Custom Analog Cells in ANAGRAM," *Proc. International Conference on Computer-Aided Design*, Nov. 1988, pp. 544-547. [SA based placement and routing to minimize crosstalk.]

[GEOF72] A.M. Geoffrion, "Generalized Benders Decomposition," *Journal of Optimization Theory and Applications*, 10(4): 237-260, 1972. [Describes the Benders decomposition MILP algorithm. With each iteration through an ILP sub-problem, either find the solution, or add a violated inequality to the subproblem.]

[GIEL89] G.G.E. Gielen, H.C.C Walscharts, and W.M.C. Sansen, "ISAAC: A Symbolic Simulator for Analog Integrated Circuits," *IEEE Journal of Solid State Circuits*, 24(6), Dec. 1989, pp. 1587-1597.

[GIEL90] G.G.E. Gielen, H.C.C. Walscharts, and W.M.C. Sansen, "Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing,"*IEEE Journal of Solid State Circuits*, 25(3), June, 1990, pp. 707-713.

[GILL86] P. Gill, W. Murray, M. Saunders, and M. Wright, " User's guide for NPSOL vers. 4.0," Technical Reports SOL 86-2, Stanford University, Jan. 1986.

[GRAY90] P.R. Gray, EECS 290Y Lecture Notes, University of California at Berkeley, Spring, 1990. [Notes on D/A and A/D conversion for analog CMOS subsystems.]

[GRAY94] P.R. Gray and R.R. Neff, "Analog-Digital Conversion Techniques for Telecommunications Applications," in *Design of Analog-Digital VLSI Circuits for Telecommunications and Signal Processing*, ed. J.E. Franca and Y. Tsividis, Prentice Hall, 1994, pp. 289-316.

[GROE89] D. W. J. Groeneveld, J. Schouwenaars, J. Termeer, and C. Bastiaansen, "A Self-Calibration Technique for Monolithic High-Resolution D/A Converters," *IEEE Journal of Solid State Circuits*, SC-24(6), December 1989, pp. 1517-1522. [Introduces dynamic current copier based matching. Audio application.]

[GUPT80] O.K. Gupta, "Branch and Bound Experiments in Nonlinear Integer Programming," PhD. thesis, Purdue University, Oct. 1980.[Describes implementation of branch and bound algorithm for MINLP optimization.]

[HARA92] I. Harada, H. Kitazawa, and T. Kaneko, " A Layout System for Mixed A/D Standard Cell LSIs," *IEICE Trans. on Electronics*, Vol. E75-C(3), March 1992, pp. 322-332. [Covers issues for a mixed Analog/Digital Std. Cell Environment, particularly shielding of analog wires.]

[HARJ87] R. Harjani, R. A. Rutenbar, and L. R. Carley, "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis," *Proc. ACM/IEEE Design Automation Conference*, June, 1987. [Knowledge based - mimics designer methods using design equations.]

[HARJ88] R. Harjani, R. A. Rutenbar, and L. R. Carley, "Analog Circuit synthesis for Performance in OASYS," *Proc. International Conference on Computer-Aided Design*, November, 1988, pp. 492-495. [Some comparators in OASYS]

[HARV92] J.P. Harvey, M.I. Elmasry, and B. Leung, "STAIC: An Interactive Framework for Synthesizing CMOS and BiCMOS Analog Circuits," *IEEE Trans. on Computer-Aided Design*, 11(11), November 1992, pp. 1402-17. ["Successive Solution Refinement" and separate layout tool. Nice comparison of other tools.]

[HOCE90] D.E. Hocevar, et al., "A Usable Circuit Optimizer for Designers," *Proc. International Conference on Computer-Aided Design*, Nov. 1990, pp. 290-293. [Flexible Spice/Optimization environment. Identifies user needs.]

[HONG90] S.K. Hong, and P. E. Allen, "Performance Driven Analog Layout Compiler," *Proc. International Symposium on Circuits and Systems*, May 1990, pp. 835-838. [Difficult paper. Results do not look impressive, and no verification, even with spice, of results.]

[HORT92] N.C. Horta, J. Vital, J.E. Franca, "Automatic multi-level macromodel generation for data conversion systems employing binary-weighted capacitor-arrays." *Proc. International Symposium on Circuits and Systems*, May 1992, pp. 2561-4. [Behavioral Simulation with Automatic macromodel generation.]

[HUAN94] J.H. Huang, Z. Liu, M.C. Jeng, K. Hui, M. Chan, P. Ko, and C. Hu, "BSIM3 Manual," Department of EECS, U.C. Berkeley, March 7, 1994. [BSIM 3 model and extraction manual.]

[HUI94] K. Hui, private communication.

[JENG90] M-C. Jeng, "Design and Modeling fo Deep-submicrometer MOSFETs," UCB/ERL Memo M90/90, U.C. Berkeley, October, 1990. Ph. D. Thesis. [Describes BSIM2 MOSFET model and methods for fitting the model.]

[JUSU90] G. Jusuf, P.R. Gray, and A.Sangiovanni-Vincentelli, "CADICS - Cyclic Analog-to-Digital Converter Synthesis." in *Proc. International Conference on Computer-Aided Design*, Nov. 1990, pp. 286-289.

[JUSU93] G. Jusuf, "Automatic Synthesis of CMOS Algorithmic Analog-to-Digital Converter," University of California at Berkeley, May 1993, Ph.D. Thesis. [Sythesis and layout of A/D converters. Synthesis uses hierarchical optimization with behavioural and equation based analysis. Layout uses hierarchical slicing structures like OPASYN].

[KAYA88] M. Kayal, S. Piquet, M. Declercq and B. Hochet, "SALIM: A Layout Generation Tool for Analog ICs." *Proc. IEEE Custom Integrated Circuits Conference*, May 1988, pp. 7.5.1-7.5.4. [Bottom up grouping of devices into modules, and slicing structure layout. Attempts to obey routing rules. No notion of routing constraints.]

[KOH89] H.Y. Koh, "Design Synthesis of Monolithic Operational Amplifiers," University of California at Berkeley, May 1989, Ph. D. Thesis.

[KOH90] H.Y. Koh, C.H. Sequin, and P.R. Gray, "OPASYN: A Compiler for CMOS Operational Amplifiers," *IEEE Trans. on Computer-Aided Design*, 9(2): 113-125, Feb. 1990. [Circuit synthesis via equation based analysis with optimization. Layout with a template and cell size optimizations.]

[KUND93] K.S. Kundert, I.H. Clifford, "Achieving Accurate Results with a Circuit Simulator," in *IEE Colloquium on 'SPICE: Surviving Problems in Circuit Evaluation'*, June, 1993, p. 4/1-5. [Discusses SPICE simulation and model accuracy problems.]

[KUP91] B.M.J. Kup, E. C. Dijkmans, P.J.A. Naus and J. Sneep, "A Bit-Stream Digital-to-Analog Converter with 18-b Resolution," *IEEE Journal of Solid State Circuits*, SC-26(12), December 1991, pp. 1757-1763. [Oversampled D/A Converter, in BiCMOS, including digital filtering and 1-b D/A. Analog chip is just stereo bitstream D/As, no filtering. Philips]

[LABE87] C.A. Laber, C.F. Rahim, S.F. Dreyer, G.T. Uehara et al., "Design considerations for a high-performance 3- μm CMOS analog standard-cell library," *IEEE Journal of Solid State Circuits*, 22(2), April, 1987, pp. 181-9. [Describes a library of standard cells. Successfully reused by designers, but did not allow customer use as digital blocks would. Many still in use today.]

[LAKS86] K.R. Lakshmikumar, R. A. Hadawy, and M.A. Copeland, "Characterization and Modeling of Mismatch in MOS Transistors for Precision Analog Design," *IEEE Journal of Solid State Circuits*, SC-21(6), Dec. 1986, pp. 1057-66. [Creates a model for mismatch and fits limited data to that model, for a 3 μm technology. Derivation of contributions of mismatch to DAC INL.]

[LEE84] H.S. Lee, D.A. Hodges, and P. R. Gray, "A Self-Calibrated 15 Bit CMOS A/D Converter, *IEEE Journal of Solid State Circuits*, SC-19(6), December 1984, pp. 813-819. [Calibrates a 2-step Capacitor-Resistor structure with an error memory and a calibration DAC.]

[LERC91] R.G. Lerch et al., "A Monolithic ΣΔ A/D and D/A converter with Filter for Broad-Band Speech Coding," *IEEE Journal of Solid State Circuits*, SC-26(12), December 1991, pp. 1920-7. [Codec application of oversampled A/D and D/A. All filters and analog circuits in 31mm^2 monolithic part.]

[LETH87] L. Letham, B.K. Ahuja, et al., "A High-Performance CMOS 70-MHz Palette/DAC," *IEEE Journal of Solid State Circuits*, SC-22(6), December 1987, pp. 1041-47. [Thorough Video-DAC paper, detailed circuits. 8 bits, 2-D segmented architecture.]

[LEME91] C. Leme, A. Yufera, N. Hora, J.E. Franca, et al, "Flexible Silicon Compilation of Charge Redistribution Data Conversion Systems," Midwest Conference, May, 1991.

[LIN91] Z. Lin, "DAVE: An Automatic Mixed Analog/Digital IC Layout Compiler," *Proc. IEEE Custom Integrated Circuits Conference*, May, 1991, pp. 5.4.1-4. [Netlist to Layout. Device symmetry, shape, matching and signal decoupling.

[LIU93] E.W.Y. Liu, "Analog Behavioral Simulation and Modeling," University of California at Berkeley, ERL Memo M93/38, May 1993. Ph. D. Thesis. [Behavioural modelling of statistical and dynamic effects.]

[LIU92] E. Liu, G. Gielen, H. Change, A. Sangiovanni-Vincentelli, and P.R. Gray,"Behavioral Modeling and Simulation of Data Converters," *Proc. International Symposium on Circuits and Systems*, May, 1992. [Estimates Static Linearity using a Behavioral Modelling tool and statistical inputs.]

[LOUI94] W.A. Louis III, B.E. Boser, E. Liu, and B. Wooley, "MIDAS-UCB User Manual," Center for Integrated Systems, Stanford University, Version 2.1-UCB, June, 1994.

[LUEN84] D.G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley Publishing Co., 2nd Ed., 1984.

[MAKR92] C.A. Makris and C. Toumazou, "Qualitative reasoning in Analog IC Design Automation," *Proc. IEEE Custom Integrated Circuits Conference*, May 1992, 8.3.1-4. [Circuit correction based on qualitative reasoning to correct errors in equation based approach. Creates numerical adjustments to errors.

[MAR93] M.F. Mar, "Automated Design of Signal Acquisition Modules," Electronics Research Lab, U. C. Berkeley, Memo UCB/ERL M93/21, March, 1993 (Ph. D. Dissertaion). [Uses hierarchical digital synthesis for filters for oversampled A/D converters.]

[MAUL92] P.C. Maulik, M.J. Flynn, D.J. Allstot, and L. R. Carley, ""Rapid redesign of analog standard cells using constrained optimization techniques." *Proc. IEEE Custom Integrated Circuits Conference*, May, 1992, pp. 8.1.1-3.

[MAUL92b] P.C. Maulik, L. R. Carly, and R. A. Rutenbar, "A Mixed-Integer Nonlinear Programming Approach to Analog Circuit Synthesis." *Proc. ACM/IEEE Design Automation Conference*, June, 1992, pp. 698-703. [Mixed integer optimization allows some circuit selection, as well as device sizing, by attaching integer values to selected circuit topology.]

[MAUL92c] P.C. Maulik, "Formulations for Optimization-Based Synthesis of Analog Cells," Carnegie Mellon University, Research Report CMUCAD-92-50, October 1992.

[MAUL93] P.C Maulik, L.R. Carley, and D.J. Allstot, "Sizing of cell-level analog circuits using constrained optimization techniques," *IEEE Journal of Solid State Circuits*, 28(3), March, 1993, pp. 233-41. [Separates analytic approach into circuit level and device level, and uses BSIM for device level. Optimization uses constrained optimization (NPSOL). KCL is one of the constraints.]

[McCR81] J.L. McCreary, "Matching Properties, and Voltage and Temperature Dependence of MOS Capacitors," *IEEE Journal of Solid State Circuits*, SC-16(6), Dec. 1981, pp. 608-616.

[MICH92] C. Michael, and M. Ismail, "Statistical Modeling of Device Mismatch for Analog MOS Integrated Circuits," *IEEE Journal of Solid State Circuits*, SC-27(2), February 1992, pp. 154-66. [Fits models to devices, and finds statistical model for underlying device parameters.]

[MIKI86] T. Miki, Y. Nakamura, et al., "An 80-MHz 8-bit CMOS D/A Converter," *IEEE Journal of Solid State Circuits*, SC-21(8), Dec. 1986, pp. 983-988. [Improvements to SHEN83. Static logic decode (not pass gate), symetrical switching to overcome process and IR drops.]

[MIKI92] T. Miki, Y. Nakamura, et al., "A 10it 50MS/s CMOS D/A Converter with 2.7V Power Supply," in *Proc. International VLSI Circuits Symposium*, 1992, pp. 92-93.

[MEHR91] S.W. Mehranfar, "A Technology-Independent Approach to Custom Analog Cell Generation," *IEEE Journal of Solid State Circuits*, 26(3), March 1991, pp. 386-392. [Another schematic to layout tool, preserving qualitative user inputs.]

[MERC94] D. Mercer, A 16-b D/A Converter with Increased Spurious Free Dynamic Range," *IEEE Journal of Solid State Circuits*, 29(10), October 1994, pp. 1180-5. [Decreases DAC glitch energy through equalizing rise and fall delay, not through decreasing segment size.]

[META92] "HSPICE User's Manual, version H92," Meta-Software, Inc, Campbell, CA.

[MEYE93] V. Meyer zu Bexten et al. "ALSYN: Flexible Rule-Based Layout Synthesis for Analog ICs," *IEEE Journal of Solid State Circuits*, 28(3), March 1993, pp. 261-268. [Hierarchical netlist to layout synthesis. Layouts do not have analog "look". No explicit constraints on routing parasitics.]

[MOGA89] M. Mogaki et al, "LADIES: An Automatic Layout system for Analog LSIs." *Proc. International Conference on Computer-Aided Design*, 1989, pp. 450-453. [Another netlist to layout system. Attempts to minimize area, but no obvious link to circuit performance.]

[MUKH94] T. Mukherjee, L.R. Carley, and R.A. Rutenbar, "Synthesis of Manufacturable Analog Circuits," *Proc. International Conference on Computer-Aided Design*, November, 1994, pp. 586-93. [Uses Infinite Programming algorithm to find worst case process corner in the inner loop of the ASTSX/OBLX formulation. Optimization time explodes 90x, but solution does remain within spec across design conditions and variations.]

[MURT87] B.A. Murtagh and M.A. Saunders, "MINOS 5.1 User's Guide," Technical Report SOL 83-20R, Dept. of Operations Research, Stanford University, December 1983, rev. Jan. 87.

[NAKA91] Y. Nakamura, T. Miki, et al., "A 10-b 70-MS/s CMOS D/A Converter," *IEEE Journal of Solid State Circuits*, SC-26(4), April 1991, pp. 637-642. [Improvements to MIKI86. Better switch ordering algorithm, consideration of device asymmetry in cell design.]

[NAYL83] J. R. Naylor, "A Complete High-Speed Voltage Output 16-Bit Monolithic DAC," *IEEE Journal of Solid State Circuits*, SC-18(6), December 1983, pp. 729-35. [Segmented Switched Current design. 16-b settling requires attention to thermal heating effects.]

[NEFF87] R. R. Neff, "Design of a custom IIR decimation/anti-alias filter for a multi-channel sigma-delta coder using LAGER," M.S. Report, U.C. Berkeley, 1987.

[NING91] Z. Ning, T. Mouthaan, and H. Wallinga, "SEAS: A simulated Evolution Approach for Analog Circuit Synthesis," *Proc. IEEE Custom Integrated Circuits Conference*, May, 1991, pp. 5.1.1-4. [SA optimization around a circuit description which includes both topology and analytic circuit models allows selection of designs as well as device dimensions.]

[NING92] Z. Ning, M. Kole, T. Mouthaan, and H. Wallinga, "Analog Circuit Design Automation for Performance," *Proc. IEEE Custom Integrated Circuits Conference*, May, 1992, pp. 8.2.1-4. [Improved version of NING91. Better description of synthesis algorithm.]

[NYE88] W. Nye et al, "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits," *IEEE Trans. on Computer-Aided Design*, 7(4), April 1988. [Optimization framework for SPICE, using feasible directions for Constrained optimization.]

[OCHO93] E.S. Ochotta, R. A. Rutenbar, and L.R. Carley, "ASTRX/OBLX: Tools for Rapic Synthesis of High Performance Analog Circuits", Report #93-62, Carnegie Mellon University, October 1993. [SA optimization with AWE simulations and design equations for trans. Many design vars allowed. Circuit Synthesis.]

[OCHO94] E.S. Ochotta, R. A. Rutenbar, and L.R. Carley, "Analog Circuit Synthesis for Large, Realistic Cells: Designing a Pipelined A/D Converter with ASTRX/OBLX," *Proc. IEEE Custom Integrated Circuits Conference*, May 1994, pp. 15.4.1-3. [Same as above. Claim of ADC design stretches opamp and comparator optimization to full ADC !!]

[OHTS92] T. Ohtsuka, H. Kunieda, and M. Kaneko, "LIBRA: Automatic Performance-Driven Layout for Analog LSIs," *IEICE Trans. on Electronics*, Vol. E75-C(3). March 1992, pp. 312-321. [Does performance driven layout, in particular, considers wiring resistance effect on offset.]

[ONOD90] H. Onodera, H. Kanbara, and K. Tamaru, "Operational-Amplifier Compilation with Performance Optimization." *IEEE Journal of Solid State Circuits*, SC-25(2):466-474, 1990. [Procedural approach to design selection, followed by tight coupling of optimization with extracted layouts. Spice used, but only AC and DC. Procedural layout techniques result in quick results.]

[ONOD92] H. Onodera and K. Tamaru, "Analog Circuit Placement -- Branch and Bound Placement with Shape Optimization," *Proc. IEEE Custom Integrated Circuits Conference*, May, 1992, pp. 11.5.1-6. [Just placement, using branch and bound to do placement without template. No routing. Shape optimization after placement.]

[ORBI92] "Foresight User's Manual," Orbit Semiconductor Corp, Rev. 1.5, July 1992.

[PADU87] S.L. Padula and J. Sobieszczanski-Sobieski, "A Computer Simulator for Development of Engineering System Design Methodologies," Report NASA TM-89109, NASA, Washington, DC, Feb. 1987. [Discussion of issues for hierarchial optimization. Use of sub-level estimators.]

[PELG89] M.J.M. Pelgrom, A. Duinmaijer, and A. Welbers, "Matching Properties of MOS Transistors," *IEEE Journal of Solid State Circuits*, SC-24(5), October 1989, pp. 1433-40. [Best paper for Matching properties of MOS devices. Verifies matching models with odd sized devices, and looks across process to allow interpolation and perhaps extrapolation to other processes. Data strongly suggests the var(mismatch) proportional to 1/(Device Area) model for mismatch.]

[PELG90] M.J.M. Pelgrom, "A 10-b 50-MHz CMOS D/A Converter with 75-Ω Buffer," *IEEE Journal of Solid State Circuits*, SC-25(6), Dec. 1990, pp. 1347-1352. [Uses a 2-d lookup to tap a resistor string. Voltage output passed to follower driving 75 Ω load. Power dominated by Buffer. Output current used to advantage in buffer design.]

[PHIL94] R.A Philpott, R. A. Kertis, R.A. Richetta, T.J. Schmerbeck, and D.J. Schulte, "A 65 MHz, Mixed-Signal, Magnetic Recording Channel DSP Using PRML," *IEEE Journal of Solid State Circuits*, SC-29(3), March 1994, pp. 177-184. [High performance disc drive chip in an analog standard cell technology.]

[REYN94] D. Reynolds, "A 320MHz CMOS Triple 8b DAC with On-Chip PLL and Hardware Cursor," *International Solid State Circuits Conf. Dig. of Tech Papers*, WP 3.2, Feb. 1994, pp. 50-51. [Current speed record holder in CMOS. No details on DAC. Uses parallel data paths to get data to the DACs in time.]

[RIJM89] J. Rijmenants, j. B. Litsios, T.R. Schwarz, M.G.R. Degrauwe, "ILAC: An Automated Layout Tool for Analog CMOS Circuits", *IEEE Journal of Solid State Circuits*, 24(2), April 1989, pp. 417-425. [Analog netlist to layout. Places and routes generated blocks, subject to usual suite of matching and parasitic constraints, annotated in the netlist. Placement is SA. Opamp and some hierarchical examples.]

[SAUL80] P. H. Saul, P.J. Ward and A.J. Fryers,"An 8-bit, 5 ns Monolithid D/A Converter Subsystem," *IEEE Journal of Solid State Circuits*, SC-15(6), Dec. 1980, pp. 1033-9. [High speed bipolar, binary weighted.]

[SAUL84] P.H. Saul and J. S. Urquhart, "Techniques and Technology for High-Speed D-A Conversion," *IEEE Journal of Solid State Circuits*, SC-19(1), Feb. 1984, pp. 62-68. [8, 10, and 12 bit DACs described in ecl process, with no trimming. 12 bit requires segmentation.]

[SCHO79] J.A. Schoeff, "An inherently monotonic 12B DAC," in *International Solid State Circuits Conf. Dig. of Tech Papers*, pp. 178-9, 1979. [Describes used of segmentation to improve linearity DNL.]

[SCHO86] H.J. Schouwenaars, E.C. Dijkmans, B.M.J. Kup, and E. Van Tuijl, "A Monolithic Dual 16-bit D/A Converter," *IEEE Journal of Solid State Circuits*, SC-21(3), June 1986, pp. 424-429. [Dynamic matching by switching between elements, averages out mismatch. Bipolar.]

[SCHO88] H.J. Schouwenaars, D. W. Groeneveld, and H. Termeer, "A Low-Power Stereo 16-bit CMOS D/A Converter for Digital Audio," *IEEE Journal of Solid State Circuits*, SC-23(6), December 1988, pp. 1290-7. [CMOS, Segmented coarse current sources are interpolated in a binary weighted divider to obtain 16 bit performance.]

[SCHO91] H.J. Schouwenaars, D. W. Groeneveld, C. Bastiaansen, and H. Termeer, "An Oversampled Multibit CMOS D/A Converter for Digital Audio with 115-dB Dynamic Range," *IEEE Journal of Solid State Circuits*, SC-26(12), Dec. 1991, pp. 1775-1780. [Combines oversampling with current source calibration to get high dynamic range.]

[SCOT85] W.S. Scott, G. Hamachi, J. Ousterhout, and R.N. Mayo, "1985 VLSI Tools: More Works by the Original Artists," Report UCB/CSD 85/225, Feb. 1985. CS Division, UC Berkely. [MAGIC]

[SERH85] G.I. Serhan, "Automated Design of Analog LSI," *Proc. IEEE Custom Integrated Circuits Conference*, May 1985, pp. 79-82. [Micro-linear bipolar gate array product. Attempts to create building block on bipolar masterslice. Design system allows user input of high level blocks and interconnects.]

[SHEN83] V. Shen and D.A. Hodges, "A 60ns Glitch-Free NMOS DAC," in *International Solid State Circuits Conf. Dig. of Tech Papers*, Feb. 1983, pp. 188-9. [Introduces highly segmented 2-d array architecture, for low glitch and high speed. Decoding is slow and NMOS specific, but 2 step decoding is introduced.]

[SHEN83b] V. Shen, "High Speed Digital-To-Analog Conversion," University of California at Berkeley ERL Memo M83/70, Nov. 1983. Ph. D. Thesis.

[SHEU87] B.J. Sheu, D.L. Sharfetter, P.K. Ko, and M-C. Jeng, "BSIM: Berkeley Short-Channel IGFET Model for MOS Transistors," *IEEE Journal of Solid State Circuits*, SC-22(4), August, 1987. [Description of BSIM 1 formulation.]

[SHYU84] J-B. Shyu, G.C. Temes, and F. Krummenacher, "Random Error Effects in Matched MOS Capacitors and Current Sources," *IEEE Journal of Solid State Circuits*, SC-19(6), Dec. 1984. [Considers random error effects as a mask edge effect, and fits data to this model. Data does not have enough W and L variation to test this model.]

[SHYU88] J. Shyu and A. Sangiovanni-Vincentelli, "ECSTASY: A New Environment for IC Design Optimization," *Proc. International Conference on Computer-Aided Design*, Nov. 1988, pp. 484-487. [Spice3 version of DELIGHT. Includes random search, and superlinear convergence. SPICE3 convergence is a real downer.]

[SMIT89] L.D. Smith, H.R. Farmer, M. Kunesh, M.A. Massetti, et al.,"A CMOS-based analog standard cell product family," *IEEE Journal of Solid State Circuits*, 24(3), April 1989, pp. 370-9. [IBM Analog Standard Cell effort. Describes technology, but really this is a place and route for custom blocks, incorporated into an environment for mixed signal design. Depends on application designers to fill out the standard cell library.]

[SPOT86] J.P. Spoto, W.T. Coston, and C.P. Hernandez, "Statistical Integrated Circuit Design and Characterization," *IEEE Trans. on Computer-Aided Design*, 5(1), January, 1986, pp. 90-103. [Sttistical design, including Monte Carlo and approximate statistical methods. Links process with device and circuit design.]

[SWIN90] K. Swings, G. Gielen, and W. Sansen, "An Intelligent Analog IC Design System Based on Manipulation of Design Equations" *Proc. IEEE Custom Integrated Circuits Conference*, May 1990, pp. 8.6.1-4. [User interaction stressed in reaching a good design, based on equation representation approach.]

[SWIN91] K. Swings, S. Donnay, and W. Sansen, "HECTOR, A Hierarchical Topology-construction program for analog circuits based on a declarative approach to circuit modeling," *Proc. IEEE Custom Integrated Circuits Conference*, May 1991, pp. 5.3.1-4.

[TOUM92] L. L. Toumelin, et al., "A 5-V CMOS Line Controller with 16-b Audio Converters,"*IEEE Journal of Solid State Circuits*, SC-27(3), March 1992, pp. 332-341. [Includes oversampled 16 bit D/A, with 2-b D/A conversion.]

[TSIV94] Y. P. Tsividis and K. Suyama, " MOSFET Modeling for Analog Circuit CAD: Problems and Prospects," *IEEE Journal of Solid State Circuits*, SC-29(3), March 1994, pp. 210-216. [Details problems with many commonly used MOSFET models, when applied in analog circuits.]

[TSUJ94] Y. Tsujihashi, et al., "A High-Density Data-Path Generator with Stretchable Cells," *IEEE Journal of Solid State Circuits*, 29(1), January 1994, pp. 1-8. [Demonstrates stretching and tiling of subcells to create digital datapath layouts.]

[VORE94] P. Vorenkamp, J. Verdaasdonk, R. van de Plassche, D. Scheffer, "A 1 Gs/s, 10-b Digital-to-Analog Converter," *International Solid State Circuits Conf. Dig. of Tech Papers*, WP-3.3, Feb. 1994, pp. 52-53. [Careful bipolar design matches delays and impedances. All current sources are identical size, including LSBs, which are passed through 50Ω R-2R to get cell sizes.]

[YAGH88] H. Yaghutiel, "Automatic Synthesis and Layout of Switched-Capacitor Filters." University of California at Berkeley, ERL M88/56, August 1988, Ph. D. Thesis. [Synthesis and layout of SC filters, building custom switch banks and capacitors, with std. cell opamps, and std. layout template.]

[YANG89] J.W. Yang, and K.W. Martin, "High-Resolution Low-Power CMOS D/A Converter," *IEEE Journal of Solid State Circuits*, SC-24(5), October 1989, pp. 1458-61. [2 step resistor string to binary weighted capacitor interpolator creates a clocked output through an opamp. 15-b, 100KHz.]

[YUAN89] J. Yuan and C. Svensson, "High-Speed CMOS Circuit Technique," *IEEE Journal of Solid State Circuits*, SC-24(1), Feb, 1989, pp. 62-70. [Description of True Single Phased Clocked (TSPC) latch and logic design.]

APPENDIX A

DSYN User's Manual

A.1 Introduction

This appendix serves as a user's manual for the DSYN release, including a description of tools, dependencies for tool compilation and use, and a description of the design libraries in the distribution, developed for the designs described in chapter 6.

A.2 DSYN Distribution Overview

The DSYN distribution tar includes 5 main components: Programs for design optimization, programs for layout, library files for optimization and layout of DAC designs, and additional libraries required for compilation of the design optimization program, and documentation. The directory structure is shown in Fig. A.4.

A.2.1 DSYN Programs and Compilation Requirements.

DSYN programs are implemented using C++, UNIX shell scripts, and AWK scripts. The layout programs **STC**, **DT**, and **TA** require no external libraries, and have compiled using cfront and g++. **SpiceOptim** has several dependencies for compilation. It requires the LEDA package, for definitions of C++ data types. The non-linear optimization package MINOS is used for linear and non-linear constrained optimization, and a C++ front end, named optz, is used to call MINOS. In

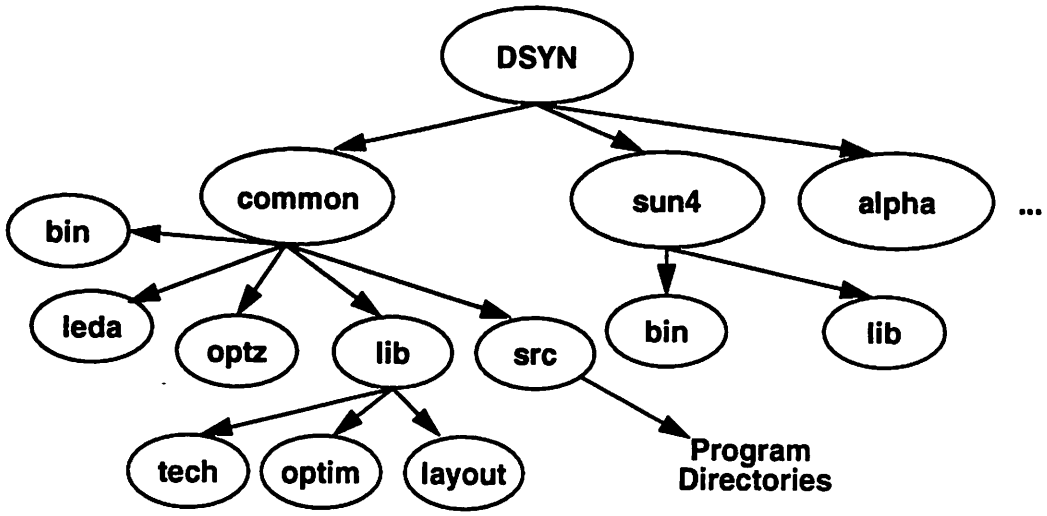


Figure A.4 DSYN Directory Structure

this tape release the current version of optz is distributed, without required MINOS source code, and only include files and compiled objects are included for LEDA, due to restrictions on the distribution of that package.

LEDA is developed at the Max Plank Institute, and is available via anonymous ftp from ftp.-mpi-sb.mpg.de: /pub/LEDA. The current version is 2.6.3.

MINOS is developed by the Systems Optimization Laboratory in the Department of Operations Research at Stanford University. The MINOS 5.1 User's Guide (Report number SOL 83-20R) is available from Department of Operations Research - SOL, Stanford University, Stanford, CA 94305. The version used here is 5.1. A license for the program and the source code should be obtained from the Office of Technology Licensing at Stanford.

The current optz version is 2.2.1. It has been developed as a C++ front end for MINOS by Eric Felt, Brian Lee, and Henry Chang at Berkeley.

The current versions of the compilers used are version 3.0.1 of the AT&T C++ front end (CC), and version 2.6.3 of the GNU g++ compiler, developed through the Free Software Foundation.

The programs have been compiled using g++ on DecStation (mips), Sun Sparc, and Dec Alpha platforms. The CC compiler was used with an earlier version of LEDA and optz for the mips architecture. See distribution information at the end of this chapter for availability of compiled programs for specific workstations.

A.2.2 DSYN Environment requirements

DSYN requires help from integrated circuit simulation and layout programs, as well as standard UNIX utilities. The shell scripts make extensive use of the UNIX nawk utility program. The optimization program calls the commercial HSPICE circuit simulator, from Meta-Software, Inc, (408)369-5400. The layout program directs the layout using the MAGIC layout editor, developed at U.C. Berkeley. It is available via ftp: contact "magic@decwrl.dec.com" for information, or it is available on magnetic tape from the EECS/ERL Industrial Liaison Program, Cory Hall, University of California at Berkeley, Berkeley, CA 94702. These three programs must be in the execution path for the programs to run correctly. If nawk is not available, but awk is, then a symbolic link may be set up to execute awk in place of nawk: "ln -s /usr/bin/awk nawk"

A summary of requirements is in table A.1

Table A.1 DSYN Dependencies

Dependency	Version	Source	Comments
LEDA	3.1.2	Max Plank Institue ftp.mpi-sb.mpg.de: /pub/leda	ftp (free)
MINOS	5.1	Stanford University Office of Technology Licensing (415)723-0651	$
optz	2.2.1	Berkeley	Included
GNU g++	2.6.3	Free Software Foundation	ftp (free)
hspice	most	Meta-Software, Inc.	$$$
magic	6	ftp: magic@decwrl.dec.com Tape: ILP at U.C. Berkeley	Free $

A.3 Optimization Tools

For design optimization the **spiceOptim** program is used. The **optScript** shell script is used to manage calls to the optimizer, and other shell scripts are used to save design information from various runs. The **layoutCmd** script converts results from optimization to a script with will run the layout process

A.3.1 spiceOptim Design Optimization Program

Design optimizations are run by **spiceOptim**. This program manages calls to the HSPICE circuit simulator, and uses the optimization algorithm selected by user input. There are a large number of input options, determining the names of file input sources and the optimization algorithm, which are listed through the -h command. The command line is so long for **spiceOptim** that typically it is called indirectly through the **optScript** command Usage is:

---> spiceOptim <-h -c spiceConFile -o spiceObjFile -p [printlevel]> -d optz -v optz_var -l optz_con -s spec -f dumpfile

Program options and input files, and their formats are as follows:

-c con_root

sets the root name for constraint simulation runs. To find the constraint values for a set of design variables, the program will generate a file "con_root.param", and then attempt to execute "hspice con_root.sp >con_root.lis" It expects to find the output of MEASURE statements in the hspice list output corresponding to all design constraints. Should the expected name appear multiple times, it takes the value for the first measurement. For normal operation, the con_root.sp file must exist, and include the "con_root.param" file (with the HSPICE .include statement).

-o obj_root

Sets the root name for objective function simulation runs. This is an identical setup to the constraint runs, except the program looks for a measurement of "objective" in the list output for use as the objective function in optimization.

-p #

Sets the output print level. Default is just to print output when constraints are not met, and information about the progress of the optimization. It is easy to get thousands of lines of output. The number input is a 5 bit binary number which sets switches for selecting debug output from different parts of program operation.

-h

Outputs the help message

-d filename

Sets the optimization defaults file name. These are design inputs which are constants for this optimization, such as nominal power supply, circuit loading, and number of DAC levels. Format is:

varname value

where value is given in a standard floating point format. These inputs are passed directly to the ".param" files for computing constraints and objectives. Comments in this and other input files are indicated with * or # characters in the first position in each line, and blank lines are ignored.

-v filename

Sets the file name for the optimization variable definition file. This is the set of design variables for the optimization. Format is:

varname starting_value min_value max_value scale_factor

Extra fields are ignored, and comments are specified as before. When an integer optimization is done, these are forced to integer numbers of the scale factor. For example, if a capacitor may vary from 2pf to 20pf, and must be an integer number of pF, the input line would be:

c1 5e-12 2e-12 20e-12 1e-12

-l filename

Sets the file name for optimization constraints (or limits). Format is:

constraintname value type scale_factor

Where type is *upper* or *lower*. Value and scale_factor is in standard floating format. It is important to set the scaled value to within an order of magnitude or so of unity. For example, a constraint on maximum settling time, may be:

tsmax 20e-9 upper 1e-9

-s filename

Sets optimization spec filename. The spec file is a set of inputs to the optimization program, determining some constants used by that program. Comments are as above, with format the same as the defaults file:

varname value

Different spec inputs may be used depending on the type of optimization being run. Important options are shown in table A.2.The used by field describes the part of the program which uses the option, either the MINOS optimizer, the optz MINOS handler, or the CUTTER cutting plane optimization implementation. For information about the MINOS options see the MINOS manual.

Table A.2 Optimization Spec file inputs

Spec Name	Explanation	Used By:
do_file_io	Uses files for MINOS fortran I/O	optz
print_level	Sets amount of printed output	MINOS
summary_file	Sets fortran unit for summary file	MINOS
summary_frequency	Sets frequency of summary output	MINOS
backup_basis_file	Sets fortran unit for backup of basis file	MINOS
new_basis_file	Sets fortran unit for basis file	MINOS
punch_file	Sets unit for punch output file	MINOS
dump_file	Sets unit for dump output file	MINOS
insert_file	Sets unit for loading insert file (punch output	MINOS
load_file	Sets unit for load file (from dump output)	MINOS
iterations_limit	Limits total number of iterations	MINOS / Cutter
major_iterations	limits minos major iterations	MINOS

Table A.2 Optimization Spec file inputs

Spec Name	Explanation	Used By:
minor_iterations	limits minos minor iterations	MINOS
function_precision	Expected precision of function evaluations	MINOS / Cutter
difference_interval	Difference used for finite differences	MINOS
row_tolerance	Tolerance used to see if constraints are met	MINOS
feasibility_tolerance	Tolerance used for determining accuracy of vars	MINOS
optimality_tolerance	Tolerance for objective function	MINOS / Cutter
major_damping_parameter	Damps steps taken in MINOS	MINOS
minor_damping_parameter	Damps steps taken in MINOS	MINOS
gradient_step	Design var differences used for finite difference gradients	Cutter
line_tolerance	Tolerance for reaching the end of the linesearch, with linesearch stopped when difference in scaled design variables less than line_tolerance.	Cutter
cut_overconstrain	Constraints over-specified by this amount. Speeds convergence, but loss of optimality. In normalized units applied to all constraints.	Cutter
optz_int_tol	Tolerance on integers in MILP step.	Cutter
optz_int_func_tol	Tolerance on meeting constraints in MILP step	Cutter

-O filename

Sets name for variable output file. This summarizes the state of the design variables at the end of the optimization. The format is:

varname finalval min max scale startingval

Other parts of the output file summarize constraint and objective results. Format is such that this output file can be used to start a subsequent optimization with this final point, using this output as a variable input file.

-f filename

Sets dump output file.

-S filename

Sets linear constraint save file name. spiceOptim saves the current value of the linear constraints on each call to the MILP solver when using the cutting plane algorithm. This file may be read in with the -L option to restore a set of constraints, saving optimization time if a job needs to be restarted. File format is the number of constraints on the first line, followed by lines with the format:

numVars constraintnum a1 a2 a3 ... b type

where type is 1 or -1 for upper and lower constraints. constraintnum is the index of the nonlinear constraint which forced this linear constraint. If **x** is the vector of design variables, and (a1 a2 ...) forms the vector **a**, these lines form linear constraints of the form $\mathbf{ax} < b$, or $\mathbf{ax} > b$, depending on type.

-L filename

Sets the name for loading the linear constraint file. (See -S above for creating that file.)

-m mode

Sets the optimization algorithm number (or mode). Options are listed in table A.3In practice options 2, 5, and 7 have been used in when spiceOptim is called by **optScript**.

SpiceOptim returns completion codes as follows:

0 -- optimal result found

1 -- feasible result found

2 -- infeasible result

Table A.3 Optimization options for spiceOptim.

Mode	Type of Optimization
0	Standard MINOS NLP
1	Branch and Bound MINLP with MINOS NLP Subproblems
2	Find Feasible (Run MINOS, but no objective, quit at first feasible point.)
3	Supporting Hyperplane, linear objective
4	Supporting Hyperplane, approximated non-linear objective.
5	MINLP implementation using Supporting Hyperplane to create constraints.
6	Supporting Hyperplane Algorithm, with NL IP. (Non Linear objective, with Integer Programming)
7	Supporting Hyperplane Algorithm, with nonlinear approximated objective, and integer programming.

A.3.2 OptScript shell script

With the large number of options available for spiceOptim, but a similar recipe used in most cases, the **optScript** shell script was written to simplify calling and managing optimizations. **optScript** is called from the command line:

---> optScript consRoot objRoot RunName

In which consRoot is passed in as the rootname of the spice file which computes constraints, objRoot is the rootname for objective functions, both passed directly to spiceOptim, and RunName is the root for all files to be used for this set of optimizations.

The script expects first looks for a set of optimization input files with the names RunName.OPTZ, RunName.OPTZ_SPEC, ..., but if these are not found will copy OPTZ* files found in the current directory to RunName.OPTZ*. Summary output from optimizations is passed to RunName.list. The script will run three optimizations, setting mode to 2, 5, and 7, to first find a feasible point, then an optimal point using a linear approximation for the objective, and finally a second optimal point using a nonlinear approximation to the objective. Optimization outputs are sent to RunName.feas_vars, RunName.cut_ip_vars, and RunName.final_vars. If one of the output files already exists, the corresponding simulation is skipped, and **optScript** proceeds to the next

optimization. Linear constraint files are saved for the second two optimizations to RunName.CUT_IP_VARS and RunName.CUT_AP_VARS. If these files already exist they are loaded with the optimization. After the optimizations are run the **spiceTime** script is used to obtain a summary of the simulation and computer time for each step in the run.

A.3.3 Other Optimization scripts

Three other scripts are provided for job management. **optSave** is used to save the current output files, **optUnsave** retrieves a set of previously saved run files, and **optDelete** removes files created by optSave. **spiceTime** is used to compute total simulation time for an optimization. Usage for these is:

---> optSave RunName Version_number

Creates a subdirectory named RunName of the current directory, and saves all run related files with the Version_number appended.

---> optUnsave RunName version_number

Looks in the RunName subdirectory, and copies back to the current directory files of this version, with the version_number stripped off the filename.

---> optDelete RunName version_number

Deletes the files associated with this version from the RunName sub-directory.

spiceTime optz_output_file

Scans the output file for summary results from HSPICE jobs, and filters these to extract real, user, and system compute time. Calls spiceTime.awk or spiceTime.${machine}.awk to filter the file. Special machine dependent files may be needed due to differences in the HSPICE summary format between platforms.

A.3.4 layoutCmd shell script

When the optimization is completed, the **layoutCmd** script is used to convert optimization results into a command which will drive the layout process. The process is two step. First an hspice run is done which replicates the read-in and design computation step done during normal design

estimation, to find the actual device and module dimensions computed from input design variables during simulation. The information is post-processed for inclusion in a command script which can execute the **DT** and **TA** layout programs. Usage:

---> layoutCmd layoutName

where layoutName is the name of the top level cell in the eventual layout. This command is run in the same directory as the design optimization, after optimization is complete. It assumes that the .param files exist containing the parameter values from the end of design optimization.

layoutCmd requires an HSPICE run named Result.sp, which runs the first part of design estimation to compute all device sizes, and then has measure statements for all outputs which are to be passed to layout. These identified by the prefix res_, e.g.: res_w1 is computed for the dimension of w1. Units must be compatible with the layout step -- typically integers and integer numbers of lambda.

A.4 Layout Tools

There are three programs used for layout: **DT** (DAC Template), **TA** (Tile Array), and **STC** (STretch Cell). **DT** is a design specific file which creates a template for the DAC layout. It lists the location of every subcell in a tiled array, including the name of the subcell, and the library cell referenced for that cell. **TA** and **STC** are not specific to a module architecture. **STC** is used to stretch dimensions in a cell, and rectangularize the stretched cell. **STC** also has a sizing mode in which sizes of cells are computed, but stretches are not implemented. **TA** uses sizing information from **STC** to locate all cells in the template created by **DT**, and then uses **STC** to implement the cells, and implements the completed array. None of the layout programs are case sensitive to parameter names, but are case sensitive to cell names.

Besides requiring MAGIC for the implementation of stretching and tiling, these programs require a cell library directory. That directory consists of library cells (in .mag format) for all required cells, and a cross reference file attaching functional block names known by **DT** to actual cell names. The DSYNLAYOUTLIB environment variable must be set to point to this library.

A.4.1 DT (Dac Template)

DT is a small program used to create the template file for a design. It is design specific, and has hard coded functional block names and expected parameter names. It uses the cross reference file XREF, found in the library directory, to associate the library cell implementation to the functional block name. If a DAC (or other module) requires a different cell organization, a new version of **DT** would be required. Some flexibility is built into the program -- number of rows and columns, and number of bias cells and lsb cells are set through user input, and types of cells may be left out by specifying a cell implementation with zero area. In the examples the "nodesign" cell has zero area, and is left out after processing by **TA**.

The **DT** command has the form:

---> DT [-h] [-l] newcellname [-d] <parm=val> ...

where -d turns on debug output, -h prints the help message, and -l produces a list of expected cell names (useful for starting a cross reference file, with some information about syntax of the cross reference file.) The newcellname is used as a root name for all created cells, as well as the name for the top level cell. The template is output to standard output. The parameter list follows. The only parameters used in the current implementation of **DT** are: rows, cols, m, and nbias.

The cross reference file is can substitute parameters into the layout list, letting a single library be used for multiple sets of design inputs. For example, if the cell function is cross-referenced:

bias dac${m}bias

then when the value of m = 2, it the program will use dac2bias as the library layout for a bias cell.

A.4.2 TA (Tile Array)

TA is used to size array elements to equalize differing elements across rows and columns, implement the cells, and then tile the module. It reads in a template, identifies all unique cells, sizes the cells, removes zero area cells, creates, and tiles the remainder. It is heavily dependent on **STC** for implementing of stretching operations. Usage is:

---> TA newcellname templatename [-d] parm=val ...

where newcellname is the name for the top level module, templatename is the name of the template file, -d turns on debug output, and the parameter list includes all parameters required by the stretching program. If the program is run without options a help message is given.

A.4.3 STC (Stretch Cell)

STC performs stretching and sizing operations at the cell level. Usage is:

---> STC newcellname templatename [-d] parm=val ...

where newcellname is the implemented cell name, templatename is the name of the library template file, -d turns on debug output, and the parameter list follows. If no parameters are given a usage message is output. Two special parameters result in sizing information being output. If VERT=-1 or HORIZ=-1 is passed in, then the program does not create the cell, and returns the appropriate minimum cell dimension (given other inputs) to standard output. If VERT or HORIZ is passed with a positive value, then the program expands the cell size to the input VERT or HORIZ value. The program looks in the directory pointed to by $DSYNLAYOUTLIB to find the template file.

The template file is a MAGIC file with labels used to indicate where stretches can occur. These labels have the format CUT_NAME_N_M. The underscores are part of the naming convention. CUT is the prefix indicating the usage of this label by STC. NAME must match a parameter name in the input list, or this label is ignored, N is the nominal value for this parameter, given no stretch, and M is a multiplier for this stretch. M is an optional parameter, defaulting to 1. For example, if the label is CUT_W1_4, and the parameter W1 is input with value 6, then the stretch associated with this label is 6-4=2. As a second example, if a cut covers two parallel devices, then the label may be CUT_W1_4_2, and for parameter input W1=10, the stretch is 10/2 - 4 = 1.

Each cut stretches mask layers either to the right or up. The program detects the direction associated with the cut by the orientation of the label. If the label defines a vertical line, or a box with height > width, then the stretch is to the right. Otherwise the stretch is up. Also, there are two types of stretches. Stretches defined by boxes are only applied to the circuits within the box. Stretches defined by lines are applied to all of the cell above the plane (or partial plane) defined by the label. VERT and HORIZ labeled planes are used to define the edges of the layout -- places where the cell

should be stretched to rectangularize the cell. The parameter information for HORIZ and VERT cuts is used to indicate default cell height and width before stretching. If the library cell is not rectangular multiple labels may be used to define cell edges. The program keeps track of the effects of stretches on other parts of the layout, so that a final set of stretches can be applied to rectangularize the cell, and information about the final size is known.

A.5 Design Libraries

The DSYN distribution includes design libraries for several optimizations and two layout libraries, as well as a library of technology files developed for the Orbit Semiconductor 1.2μm process.

A.5.1 Optimization

The optimization libraries consist of several directories, each containing information specific to a particular design. Each contains hspice runs for computing constraint and objective functions, OPTZ* files for starting an optimization, and some explanation of the design in a readme file. There is also a Results.sp file used to obtain layout parameters from design variable inputs. In each case they are set up to run an optimization out of the box, but the user can chang the application specifications in the OPTZ_CON and OPTZ files as needed. The recommended procedure is to copy the entire directory to a new location, and then run the optimizations in the new location. Optimization libraries are:

DACdynP3: Video DAC application based optimization. Includes static linearity, DC bias margins, and transient performance. Current sources are implemented with PMOS devices, to allow a resistive load tied to ground, and a signal range from 0 to V_{fs}.

DACstat: Static DAC optimization, including only the DC parts of DACdynP3, and set up for much lower current levels.

MirrorN: Nmos high swing cascode current mirror.

tiny: Simple two transistor mirror optimization.

A.5.2 Layout

There are two layout libraries provided, which match the DACstat and DACdynP3 optimization libraries above. The TERMINALS, VARS, and README files in each directory provides information about the use of these layout libraries.

scmos.p: A DAC design with no latches, and no separation of digital and analog supplies in the DAC module. This is meant for lower speed, static designs. For use with DACstat optimization.

scmos.p.l: A DAC design with latches in the cells, row and column drivers, and latches at the edge of the cell. This has been used to implement the high speed design described in this dissertation. For use with DACdynP3 optimization.

A.5.3 Technology

A technology database has been developed for the Orbit Semiconductor 1.2 μm process. This is located in the directory tech/ORBIT.lib, and includes model files, a library file which uses the model files to get nominal, fast, and slow models, a technology constants file which includes overlap capacitance, resistance, and other technology information, and a LINK file used to create symbolic links from this directory to the directory using these files.

A.6 Example Design

It is helpful to see how the tools work with a couple of design examples.

A.6.1 Tiny Current Mirror example

To see a simple optimization example, copy the contents of the DSYN/lib/optimization/tiny directory to a new location, and from there run the LINK script in the ORBIT.lib technology directory. Then to execute a tiny current mirror optimization use the command:

```
---> optScript tinyRun tinyObj test
```

This will create a test.list output file, which summarizes the results found during execution. For more extensive output, spiceOptim output is routed to a file in the /tmp directory. If the job

aborts, it often means that hspice job aborted, and this can be tested by executing a stand-alone hspice job:

---> hspice tinyRun.sp.

Also, if a constraint output is not found in the hspice output the job will quit.

There is no layout directory associated with this optimization.

This tiny job should run in under 20 minutes.

A.6.2 DAC example

For a more realistic example, consider one of the DAC example directories, DACdynP3.

Again, copy the contents of the optimization lib directory DACdynP3 to a new location, and link in the technology file. Use the -r option to include the DigBuff subdirectory in the copy. This optimization is started:

---> optScript dacRun dacObj test

After this result is completed (several hours), a second optimization is needed to optimize digital circuits. cd to the DigBuff directory, and source the LINK script which symbolically links results from the first run to this directory. Then execute the second optimization in this directory:

---> optScript digRun digObj test

If you wish results may be saved using the optSave command. The second optimization results must be passed back up to the top directory:

---> ln -s digRun.param ../.

Next the results from optimization are used to create the command script. In the directory which dacRun was run in, execute:

---> layoutCmd MyCellName

This creates an executable script that runs **DT** and **TA** with the final design values.

To run the layout part, make certain that the DSYNLAYOUTLIB is set to point to the scmos.p.l:

---> set DSYNLAYOUTLIB = ~DSYN/lib/layout/scmos.p.1

then create a subdirectory for the layout, and move the layout script to that subdirectory. Change directories to the layout directory, and execute the script:

---> mkdir layout ; mv MyCellName.scr layout ; cd layout ; MyCellName.scr

The layout job typically takes a few minutes, depending mainly on the network implementation. Most of the job time is taken by the time needed to load MAGIC for stretch operations on each cell implementation, and if MAGIC is loaded across the network, then it is somewhat slower. When done, you can execute MAGIC to browse the layout.

---> magic -dX11 MyCellName

For information on connectivity, see the TERMINALS file in the layout library.

A.7 Common Problems and Solutions

A.7.1 Finding an Initial Feasible Point

A common problem is that the optimizer will not find a feasible point in the first optimization. It is easier to manually find a feasible (if very costly) solution, which may take a designer a handful of runs, rather than letting the optimizer go for some time without finding a solution. Remember that the first solution only needs to be feasible, and the optimizer is better at improving an already feasible solution than at finding the initial feasible point. In practice, the easiest way manually search for a feasible set of design variables is to change the starting point for the design variables, execute **optScript**, and observe it a feasible point is found immediately in the first call to spiceOptim. If not feasible, halt the run, observe the outputs, including a list of violated constraints, and modify the design inputs.

A.7.2 Optimization stops when no improvement is seen

During the execution of the Supporting Hyperplane optimization, each additional linear constraint should eliminate the previous infeasible solution to the MILP step (see chapter 4 for the algorithm). If the same infeasible point is returned on consecutive calls to the MILP algorithm, then the algorithm will go in an infinite loop, returning the same constraint and same next point on each additional pass. This situation is detected, and the program will halt with an error message.

Theoretically this should be impossible, but since the feasibility tolerance (optz_int_func_tol) is non-zero in the Branch and Bound algorithm, it is possible that slightly infeasible results may be returned from the MILP step, causing this situation. The optimization should be restarted, with two possible solutions. The optz_int_func_tol value may be reduced, which will increase optimization time for Branch and Bound, or the cut_overconstrain value may be increased, to force the solution inside the linear constraints, speeding solution, but reducing optimality. It is possible to restart with an over-constrained optimization, find a result, and then go back and try reducing the cut_overconstrain input later in a subsequent re-run.

A.8 Finding the TAR

The compressed tar file for this distribution, including a postscript copy of this chapter, is currently retrievable by anonymous ftp at haiku.eecs.berkeley.edu, in the file pub/neff/DSYN/DSYN.-tar.Z. Because of the potential difficulty compiling the spiceOptim code, a copy of the spiceOptim program may also maintained there, in spiceOptim.$machine, where machine is sun4, alpha, or mips, for those architectures. If this site is unavailable the location of the distribution may be found by fingering: finger neff@eecs.berkeley.edu. The usual uncompress and tar commands can be used on this file. If you have questions the author can be reached at neff@eecs.berkeley.edu.

A.9 Compiling the Code

Makefiles are in each C++ source directory. This distribution is not streamlined, so you must go to each of the three subdirectories under the src hierarchy and set the machine and C++ compiler. If starting from scratch, first obtain and compile the LEDA source, then obtain the fortran MINOS source and place in the optz sub-directory, compile MINOS, compile optz, and then compile spiceOptim. If you can use the existing LEDA object code, then skip the first step and start with optz. Once the executables are made, make certain the DSYN/bin directory and perhaps DSYN/machine/bin directories are both in the user's path.

A.10 Disclaimer

This software is offered on an as-is basis. The user assumes all risk for the functionality of designs obtained with it. The designs in these libraries were made without a thorough patent search, so it is also the responsibility of the user to determine the existence of applicable patent infringements, if any, should these libraries be incorporated in a commercial design.

A.11 Acknowledgments

The optimization program described here would not have been possible without the contributions of MINOS and LEDA from Stanford and Max Plank Institute, as well as the development of the optimization interface done at Berkeley by Brian Lee, Henry Chang, and Eric Felt. Henry and Eric have always been helpful when I have had difficulties with my code. Funding for this research is provided through the Semiconductor Research Corporation, grant DC-94-324.