AUTOMATED DESIGN OF SIGNAL ACQUISITION MODULES

by

Monte Frank Mar

Memorandum No. UCB/ERL M93/21

3 March 1993

AUTOMATED DESIGN OF SIGNAL ACQUISITION MODULES

by

Monte Frank Mar

Memorandum No. UCB/ERL M93/21

3 March 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Automated Design of Signal Acquisition Modules

by

Monte Frank Mar

Doctor of Philosophy in Engineering-Electrical
Engineering and Computer Science

University of California Berkeley

Professor Robert W. Brodersen, Chair

Signal acquisition modules implement the functions of capturing a signal and providing a digitized representation of the signal. They usually include an A/D converter and filters to prevent anti-aliasing after sampling. This dissertation discusses methods for automating the design process from a high-level specification to layout using oversampling A/D converters. Using common levels of abstraction, a design framework was established allowing centralization of design information and development of a unified design methodology for these systems. The method of hierarchical design estimation was developed to guide design development. At the lowest level, hierarchical design estimation relies on the use of architecture templates to capture design information for specific implementations of processors and to increase accuracy of design estimates. In the initial search of the design space, performance estimates are made and propagated in a top down fashion through the various levels of the hierarchy. At a high level, promising design candidates can be identified and pursued. Accurate simulation of the mixed signal system provides verification of the design. Models for 1/f noise were incorporated with existing difference equation methods to provide improved simulation capabilities for the analog oversampling modulators. These models were able to predict the performance of several fabricated modulator variations to within 3 dB of measured results. To verify the functionality of the system, several chip designs were implemented. The chips covered applications including data acquisition, linear phase filtering, and signal acquisition for speech recognition. Analysis using the design framework reduced design time significantly while still providing high performance, fully functional chips.

Robert W. Brodersen

Chairman of the Committee

Table of Contents

CHAPTER 7 Design and Implementation of the Analog Circuits

CHAPTER 8 Design Examples

CHAPTER 9 Conclusion

List of Figures

List of Tables

Acknowledgments

A project of this magnitude required the assistance and cooperation of many people. I wish to thank my advisor Professor Bob Brodersen for creating a research group with tremendous resources and an atmosphere of interaction and collaboration. There are not many groups like this where I could have performed this type of research. I thank him for his advice, for his willingness to point me in the right direction, and for editing this dissertation. I also thank Professor Jan Rabaey for his input during my preparation for the qualifying exam. I thank Professors Paul Gray, Alberto Sangiovanni-Vincentelli, and Stuart Dreyfus for their input during my qualifying exam. I thank Professor Don Pederson for his help in selecting classes and for helping to make my time as a teaching assistant enjoyable.

I thank Max Hauser for starting this project and laying the basic groundwork that I was able to build on. I thank Brian Brandt of Texas Instruments for loaning me test devices to verify my test setup. I thank Joe Wann for making the 1/f noise measurements on test devices. Gordon Jacobs, Bill Baringer, Greg Uehara, and Cormac Conroy provided helpful discussions and advice along the way.

Within our research group, there are so many people who made the Ph. D. process a little easier. They have my gratitude. Phil Schrupp and Sue Mellers provided help in designing boards and setting up lab spaces for testing. Brian Richards provided support in debugging programs and settling various problems. Tom Boot always answered my many administrative questions and helped me to get my papers out on time. Peggye Brown and Carole Frank also helped me with administrative matters. Kirk Thege and Kevin Zimmerman were always fast and helpful when solving computer system problems, and did a great job of keeping our systems stable. I thank Gautam Doshi, Shankar Narayanaswamy, Robert Yu, Lars Thon, Anton Stölzle, Andy Burstein, and Sam Sheng for being terrific office mates. I thank Anantha Chandrakasan for encouraging me as I submitted papers and for helping me to develop my presentations.

I had the opportunity to collaborate with several researchers on this project, and I thank them for deepening my understanding of the oversampling techniques. Professor Avideh Zakhor, Soren Hein, and Mei-Tjng Huang worked with me on the zoomer implementation. Peter Xiao introduced me to superconducting applications of oversampling converters. Professor Rajeev Jain and Soei Shin Hang of UCLA provided DECGEN package. Weijie Yun showed me how to use an oversampling converter as an accelerometer. Randy Allmon of DEC helped me to develop my documentation and tutorials.

I thank my brother and sisters for the phone calls and for the entertainment and the help in relaxing when I was away from Berkeley. I thank my Thursday Night Study Group for listening to my problems and offering support. I also thank my great-aunt Pearl Mah for providing dinners and commentary during the 1991 Baseball American League Championship Series.

Finally, I thank my mother and father, Brian and Hazel Mar, for helping me to come to Berkeley and for their constant support in my studies. I thank them for their assistance in getting a new computer when I killed my old one. This computer became the heart of my chip testing setup. My father is a professor and my many visits to his office as I was growing up inspired me to get a doctorate degree in engineering.

CHAPTER 1

Introduction

Mixed signal systems use both analog and digital signals to perform signal processing tasks. These types of systems are found in many areas including biomedical instrumentation, communication systems, and industrial control. The monolithic implementation of these systems has been hampered by the conflicting demands of analog and digital circuit design.

In the past, mixed signal designs have been separated into discrete digital and analog portions. The use of commodity parts encouraged this type of solution for mixed signal design. The boundary between analog and digital was sharply defined allowing each portion to be designed independently. Translating these designs directly to CMOS is not always possible since several different technologies and design styles may have been used in the discrete components. In addition, redesigning the entire interface is complicated since simple architectural changes can have a strong effect on the implementation. Investigating the trade-offs requires extensive amounts of design knowledge in both the analog and digital domains.

In CAD of digital integrated circuits, these problems have been addressed by abstracting the design process and allowing designers to focus on the algorithm level. Designers have the ability to write high-level descriptions of algorithms and have them compiled to mask descriptions for integrated circuits [1], [2]. These application specific integrated circuits (ASICs) can be developed with short design cycles and with high probability of success. While these digital systems can cover a wide range of applications, it is still difficult to integrate A/D and D/A interfaces due to the lack of integrated design tools for mixed signal systems.

Analog CAD tools typically focus on lower levels of abstraction than digital tools, concentrating on transistor-level optimization of known circuit topologies. The different levels of abstraction and the different types of knowledge required for each design style make it difficult to develop tools that deal with mixed signal ASIC systems. In addition, there has been a recent trend of moving many signal processing functions into the digital domain. Digital signal processing offers advantages in precision and dramatically decreases the development cycle time. A/D and D/A converters and RF analog circuitry could soon become the only analog circuitry required in many IC applications. Techniques to raise the level of abstraction for the design of A/D and D/A converters should be developed.

In this project, oversampling A/D converters were used to illustrate unified design automation techniques for a class of mixed signal systems. Instead of separating the analog and digital systems, high-level design exploration is used to examine system alternatives. Cell and architecture libraries, estimation tools, and accurate high-level models provide users with the design information required to quickly evaluate design candidates. This integrated approach allows encapsulation of information, freeing designers to examine system interactions rather than being distracted by the plethora of low-level design issues.

1.1 Previous Approaches to Mixed Signal Design Systems.

Transforming behavioral specifications to circuit layout requires many steps, and these steps differ between analog and digital circuit design. Many research efforts have focused on different aspects of the problem and now these results can be integrated into a coherent design system. Mixed signal system design is much more than a circuit design problem and more tools are focusing on higher levels of abstraction. This section outlines some of the recent advances in analog and digital design tools that address portions of the design process for implementing mixed signal ASICs.

1.1.1 Analog CAD Tools.

In analog CAD, researchers have found that techniques similar to those used in digital CAD systems are not that successful since the design space is very large and since there is a lack of general purpose analog functional blocks. It is therefore difficult to map behavioral descriptions of an analog function to a set of general purpose circuit blocks. Present design systems rely on translating specifications and mapping them to circuits generated by module generators.

Generation of analog circuits must be done carefully. While digital circuits can be adapted to different process technologies with relatively minor changes, analog circuits are more sensitive to parasitics and process variations. A variety of routing techniques must be used and cells usually have to be redesigned for each process. Early efforts at analog module generators provided template design styles for opamps, comparators, and references [3], [4]. An example of this type of tool is Opasyn which makes use of an optimization engine that could search the parameter space of a circuit configuration to find the best device sizes for a given design specification [5]. To speed up the search, models were developed to predict the circuit performance based on device sizes. The designer had to partition the high-level design based on the system specification and then translate system parameters to parameters that the module generators worked with. In addition to these module generators, generalized analog place and route tools have been developed [6].

Since module generators provide help at the lowest level of the design process, higher level compilers were needed to implement functions like A/D and D/A converters, buffers, amplifiers, and switched capacitor filters. One approach to solving this problem was used in the IDAC (Interactive Design of Analog Circuits) system [3]. Each function was described by a template. Device sizing algorithms were provided for basic block design. Custom software modules were developed to oversee the integration of the basic blocks into more complex functions. Designers could specify parameters for the template and the software modules would generate the macrocell. Another approach was to build compilers dedicated to generating a single functional block such as ADORE for switched-capacitor filters [7], and various A/D converter generators [8], [9], [10].

The basic approach for design using analog module generator tools can be described using the example of a voiceband PCM telephone codec. The system is decomposed into known signal processing functions based on design experience. In this case, the design is partitioned into transmit and receive filters and A/D and D/A converters. Different circuit topologies and techniques are then evaluated to decide which circuit configuration to use. For the A/D converter, the designer would have to choose between successive approximation, flash, and oversampling A/D converters. Specifications are translated to parameters for basic building blocks and the module generators are then used to implement the building blocks with analog place and route tools to complete the layout phase [11].

1.1.2 Digital CAD tools.

Analog CAD tools focused on translating specifications to known designs. In contrast, digital CAD tools for custom IC signal processor design focus on higher levels of abstraction. These CAD tools help designers to meet performance goals through analysis and transformation of algorithms to exploit parallelism. The emphasis is on high-level synthesis at the algorithm level of abstraction, transforming behavioral descriptions to hardware structure. Many systems have been designed to perform this task and a variety of techniques have been developed [12].

In comparing analog and digital CAD methods, we find that both approaches attempt to fit the problem to a promising architecture style and that architecture selection is critical. Digital systems have the advantage that effective architectures can be synthesized based on algorithmic properties and less attention needs to be given to circuit design. Analog functions must be adapted to known architectures and circuits must be highly optimized for the given application. Digital CAD tools work at the algorithmic level where transformations provide many degrees of freedom for choosing good architectures, while analog CAD tools work at the architecture and circuit levels with the designer being responsible for the architecture selection.

1.1.3 System Design: Methodologies and Design Tools

The system definition for mixed signal systems can be partitioned in a variety of ways. Analog circuits tend to be smaller and can consume less power than digital circuits. Digital circuits tend to be more flexible, easier to design, and are immune to component drifts and cross-talk effects. The trend in recent years has been to try to move more of the signal processing functions to the digital domain, giving rise to vastly different system architectures that can provide the same functionality.

This trade-off can be examined in the design of superheterodyne receivers. The basic receiver block diagram is pictured in Figure 1.1. While previous receivers have been implemented using analog circuits, the emergence of new A/D converter techniques has allowed more of the receiver to be implemented digitally. For digital cellular phones, digital chip sets are available to handle all the functions except for the RF front end [13]. Bandpass Δ–Σ systems promise increased performance with simpler systems [14]. Studying the design trade-offs for these systems is difficult since system complexity is high, and trade-offs must be balanced with respect to algorithms, architectures, and circuits. To properly study these trade-offs, an overall system design methodology is needed so that as the design proceeds, different options are identified and investigated.

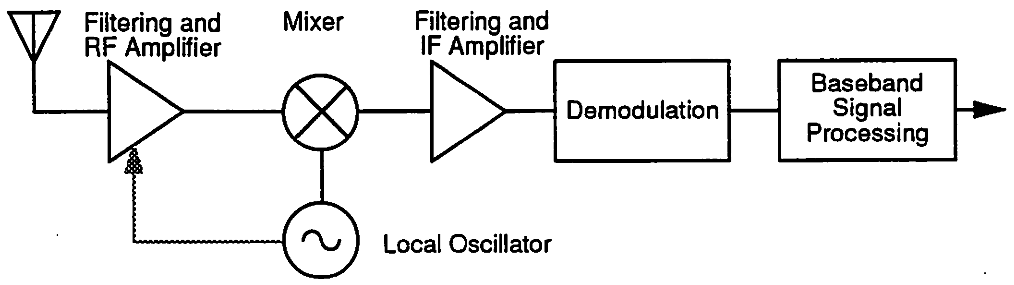

Filtering and RF Amplifier
Mixer
Filtering and IF Amplifier
Demodulation
Baseband Signal Processing
Local Oscillator

Figure 1.1 Block diagram of a superheterodyne receiver.

The Electronic Systems Design Methodology (MCSE) methodology [15] is an example of a design methodology to aid in the design and analysis of complex real-time systems. The methodology is based on the development of a system by creating 3 views, the functional, behavioral, and hardware views. The functional view specifies the types of functions or methods that must be performed by the system while the behavioral view gives a description of what each function should do. The hardware view describes what hardware should be used in the implementation. The 3 views are created in a process that is characterized as being globally top-down for design and bottom-up for implementation with the design process being split into 2 phases, specification and functional design. The implementation process starts by developing hardware and software implementations for the lowest level descriptions in the hardware view and then assembling these components in a bottom-up fashion.

Methodologies guide the design process but frameworks are needed to combine design tools and methodologies into a design system. The YODA system was an effort at this type of design system [16]. It made use of feasibility studies using high-level models to obtain an estimated performance index for implementations of digital filters. The results of the studies help the designer to select a promising design plan. An additional advice tool was available to advise designers on design choices. While this system provided strong analysis tools to aid in the top-down design phase, it did not provide help for implementation and development of architectures and circuits.

1.2 Design System Overview

These ideas from system design methodologies and analysis tools can be combined with the circuit design CAD tools to form an integrated design system for mixed signal systems. We have limited the scope of the problem to signal acquisition modules, which implement the tasks of anti-alias filtering, sampling, quantization, and signal conditioning. A typical block diagram for a module is shown in Figure 1.2. The goal of the framework is to integrate design and analysis tools for

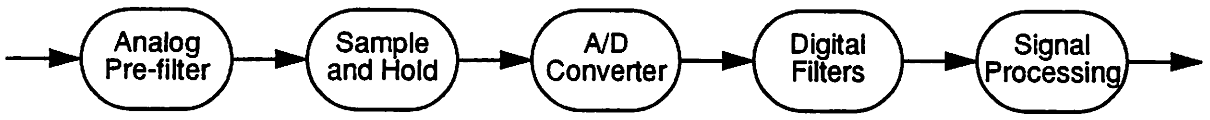


Figure 1.2 Block diagram for a typical signal acquisition module.

each of these functions in order to facilitate the processes of specification, optimization, and layout. The design problem can be thought of as combining several heterogeneous IC processor modules to provide the behavior of signal sampling and coding. While automation of the design process is attractive, automation at the expense of the ability to explore design trade-offs for optimization is not desirable.

Several problems must be addressed in a design framework. A higher level of abstraction is needed for the analog circuits and a basic design methodology must be developed to help manage the complexity of the design process. Additionally, methods are needed to help encapsulate the extensive design knowledge needed for analysis and design.

1.2.1 Levels of Abstraction

In order to overcome the differences in design styles for analog and digital CAD tools, we propose a design methodology based on 3 levels of abstraction shown in Figure 1.3 along with examples of concepts that would be dealt with at each level of abstraction. By defining shared levels of abstraction, a common framework can be developed allowing both analog and digital CAD tools to work together in parallel. These divisions partition the framework so that new tools can be developed and easily integrated.

The levels of abstraction also provide distinct breaks in the design process where exploration can occur. Rather than pursuing a depth-first design strategy, a breadth-first search is encouraged. At each level, estimation models can be used to provide designers with the information necessary to guide design decisions. The goal of exploration is not to locate the best design at each level, but to study the design trends so that promising design candidates can be identified. Only details at the current level of abstraction need to be considered, making the design of these complex systems easier to manage. As the design progresses to lower levels of abstraction, candidates will be discarded as designers decide on which candidates to focus implementation and optimization efforts.

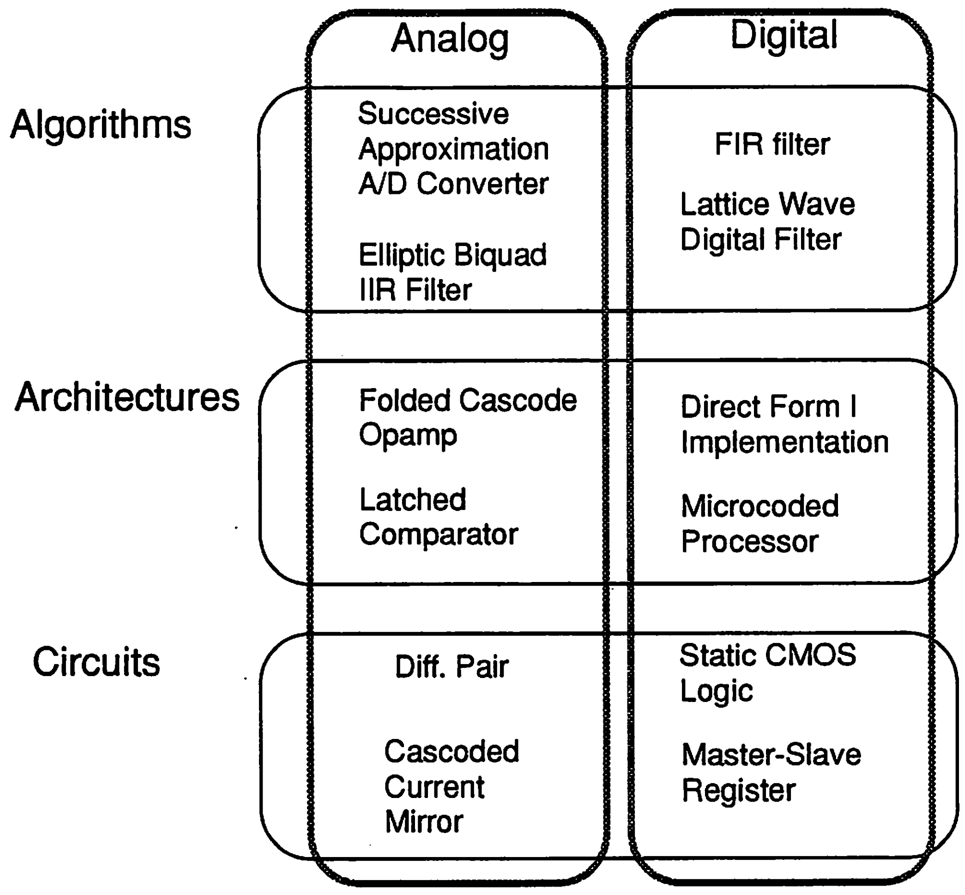


Figure 1.3 Levels of abstraction for design.

Following the top-down design phase, a bottom-up implementation is pursued which involves layout and verification. Rather than performing these time-consuming tasks every time a new architecture is desired, it is possible to develop parameterized architecture templates and to use architecture synthesis tools. Extensive design knowledge is needed to fully implement a VLSI signal processor, so capturing this information in reusable templates has several advantages. It speeds up the design process and encapsulates design information, shielding designers from the detailed implementation information. In addition, it can greatly aid the accuracy of high-level estimators since many details about the implementation are now known. The circuit level of abstraction is then only dealt with by those who implement new architecture templates or synthesis schemes.

The use of architecture templates alters the flow of the design process. When a favorable design candidate has been identified at the architecture level, a layout can be generated using automated silicon assembly tools. Some degree of architecture exploration is encouraged, but it is limited to the architecture styles supported in the design system. This will not be a serious limitation since new architecture templates are easily developed and provisions are made for easy addition of new templates into the design system.

1.2.2 Functional Compilers and Estimation

Functional compilers are used to map high-level functions to a collection of library templates [17]. The mapping is achieved either by a specialized module generator or by combining the results of several module generators. Compilers provide short development times but changes to the architecture can be difficult since a compiler hides many implementation details. The functional compiler approach has been previously applied to this problem with early approaches making use of an oversampling modulator and a fixed choice of filters [18], [19]. Only the oversampling ratio could be varied to change the resolution and speed. A later compiler incorporated area estimation techniques in an effort to optimize the design [20]. Other approaches focus on silicon compilation of modulators via arbitrary netlists [21] and compilation of filters onto fixed processor architectures [22], [23]. Functional compilers are limited since they are only able to cover a fixed design space.

The functional compiler approach can be modified to provide more design exploration and design space coverage. An overall design manager could be used to control a library of functional compilers. Rather than comparing the results of several compilers, the compilers should be written to provide feedback information for the designer without the need for layout generation. This allows several design paths to be investigated simultaneously at a high level of abstraction. The initial feedback information from the compilers can be rough estimates since they can be refined as more design information is acquired. This approach will be called hierarchical design estimation and is a critical component in the design framework.

Using hierarchical design estimation, designers are able to compare the results of different modules combinations at a high level of abstraction in an effort to meet their design specification. This approach involves the designer and helps them to find beneficial design trends. It also provides solutions that can cover a broad range of applications, not just deviations from a specific design. While functional compilers seek to simplify the design by automating the design process and hiding many details, hierarchical design exploration seeks to simplify the design by limiting the amount of design information presented to the designer. Only small amounts of information are necessary to make design decisions at a given level of abstraction.

Combining the levels of abstraction, the information about A/D interface design, and hierarchical design estimation leads to the design framework used in this project. The design tasks are partitioned into high-level, detailed-design, simulation, and layout generation tasks as shown in Figure 1.4. In high-level design, estimators are used to guide the choice of A/D conversion or fil-

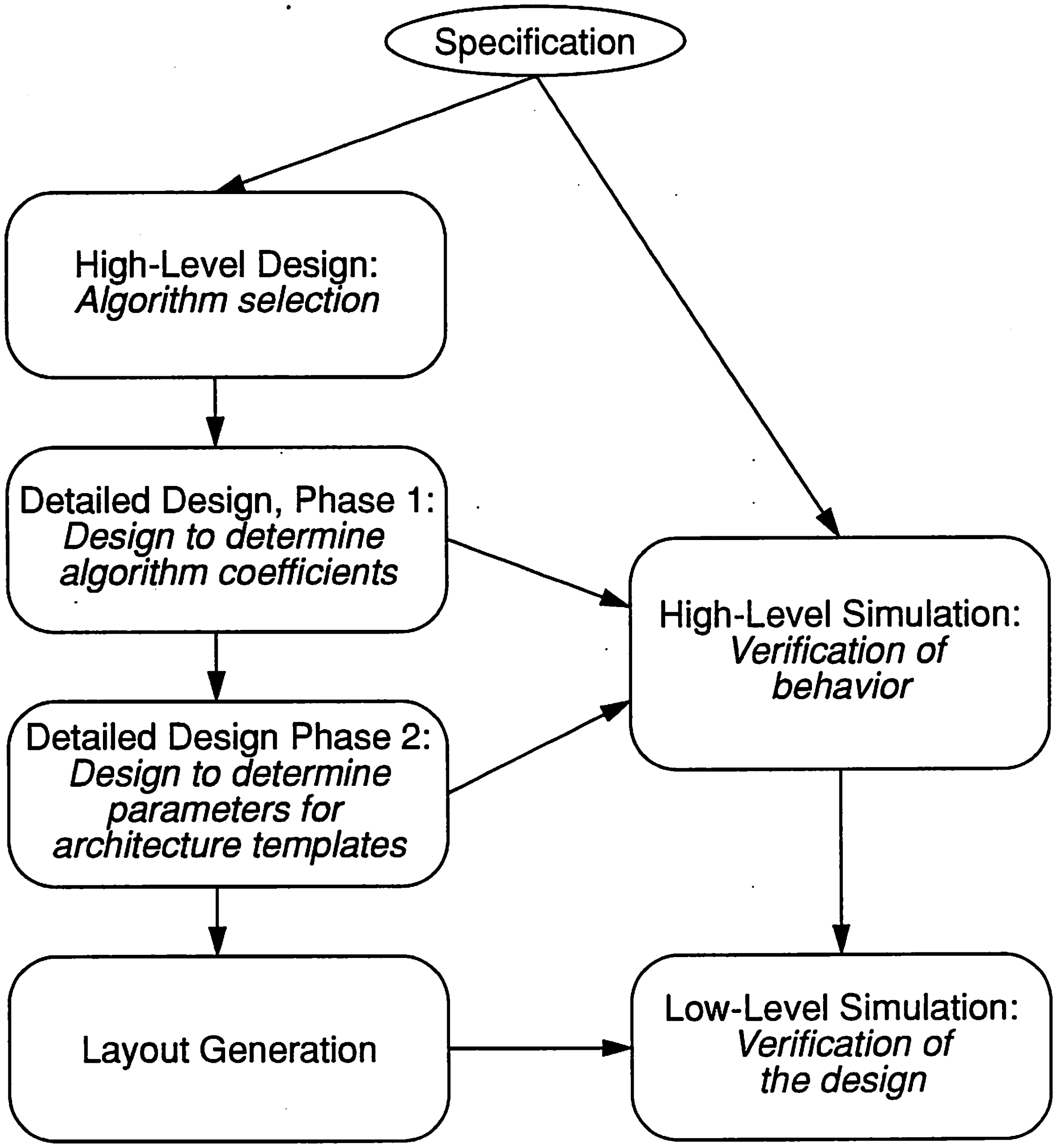

Specification
High-Level Design: *Algorithm selection*
Detailed Design, Phase 1: *Design to determine algorithm coefficients*
Detailed Design Phase 2: *Design to determine parameters for architecture templates*
Layout Generation
High-Level Simulation: *Verification of behavior*
Low-Level Simulation: *Verification of the design*

Figure 1.4 Flow for the design framework.

tering algorithm. Design candidates are defined and studied and designers can vary the interface specification to investigate the effects and study design feasibility. High-level design deals primarily with the algorithm level of abstraction. In the detailed design phase, the algorithms are implemented, and design estimates can be updated. At this point, there is enough information for a high-level simulation and architecture templates can be chosen for the components in the design candidate. Based on these choices, final estimates can be made for the design and decisions can be made on whether to proceed with layout generation.

1.3 Summary

This chapter presented a brief introduction to the methodology of the mixed signal system design. Recent trends have shown that more signal processing functions are being moved to the digital domain. This places a premium on the design of A/D and D/A converters. A survey of digital and analog CAD tools showed that analog tools focus on the circuit and architecture levels of abstraction, while digital tools provide support from the algorithm level. By defining common levels of abstraction and using architecture templates, a design methodology for attacking mixed signal design was proposed. A brief discussion was given showing how the methodology could be applied to the design of signal acquisition modules.

The following chapters will cover various aspects of the design system and describe the design concepts in depth. Chapter 2 presents an introduction to oversampling A/D converters which includes multirate digital filter design. Chapter 3 will present an overview of the framework for design including high-level analysis. Chapter 4 discusses simulation techniques. Chapters 5, 6, and 7 document the design tools for transforming algorithms to analog and digital circuits. Chapter 8 presents some example circuits designed with the system. Chapter 9 presents conclusions and directions for future work, including how this approach can be extended to more general mixed signal problems.

CHAPTER 2

Oversampling A/D Conversion

2.1 Introduction

Oversampling A/D converters are attractive for IC implementation since they can be realized in standard CMOS and have been used to demonstrate greater than 16 bit resolution without the need for trimming or precision analog circuitry. The term oversampling A/D converter is usually applied to the combination of a noise shaping coder or modulator and a set of digital filters as shown in Figure 2.1. This chapter will provide an overview of the A/D converter and describe the

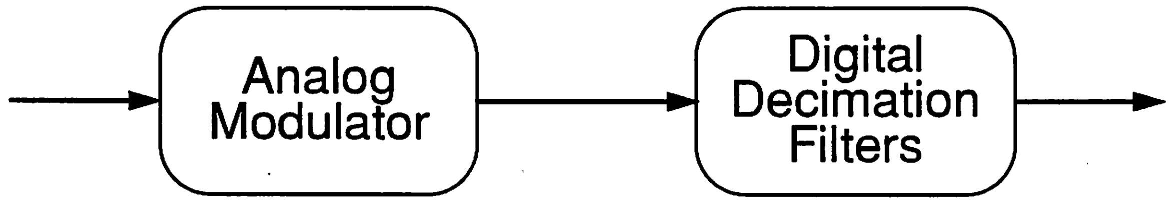


Figure 2.1 Block diagram and functions of an oversampling A/D Converter.

concepts used in the modulator and the digital filters.

2.2 Modulators for Oversampling A/D Conversion

Oversampling modulators are related to Delta Modulators, and were initially proposed as a means of overcoming some of the problems encountered in Delta Modulation [24]. They are referred to as modulators since they were first used to encode telemetry information in a bit stream [25]. Inose and Yasuda, the researchers who developed the method, named it Δ–Σ modulation.

Later researchers have also used the name Σ–Δ modulation which has caused some confusion. In this dissertation, the terms modulator and noise shaping coder will be used interchangeably to describe these circuits. This section will provide an overview of oversampling, quantization noise, noise shaping, and the different types of modulators currently in use.

2.2.1 Quantization Noise and Oversampling

The process of signal quantization consists of sampling a signal and then assigning each sample a digitized representation. The quantization error is defined to be the difference between the actual analog value and the digitized representation that is assigned to the sample. Uniform quantizers use the same step size for each digital level assigned to a value. In a linear 8 bit A/D converter, for example, there are 256 levels equally spaced across the full scale voltage range.

Quantization can be modelled using an additive error signal e(t) to simulate the actual quantization noise. If the quantizer has a resolution larger than 2 bits and if the input signal is active, the error signal e(t) will tend to a uniform distribution on the interval defined by the quantizer step size. Consecutive samples from e(t) appear to be statistically independent and the quantization noise can be modelled using white noise. Assuming that the errors are uniformly distributed and that the step size is Δ, it can be shown that the quantization error has power $\Delta^2/12$. A simple model for an A/D converter can be implemented by adding white noise of with power $\Delta^2/12$ to the original signal.

By using the additive white noise model, the effects of oversampling can be studied. If the signal is oversampled by a factor of 2, the total quantization error still has power $\Delta^2/12$, but it is spread over a larger frequency region. Figure 2.2 illustrates this using power spectral densities s(f). An ideal low pass filter can be used to limit the signal bandwidth to fs, eliminating half of the noise power and increasing the signal power to noise power ratio by 2. Since the signal is now bandlimited, it can be resampled at half the rate providing data at the original desired rate. When using higher oversampling ratios, we find that each octave of oversampling provides a gain of 3 dB in SNR when perfect digital filtering is used.

For sinusoids, it can be shown that if a full scale signal is quantized using an N bit quantizer, the SNR will be given by Equation 2.1.

$$\mathrm{SNR} = (6.02N + 1.76)\,\mathrm{dB} \qquad (2.1)$$

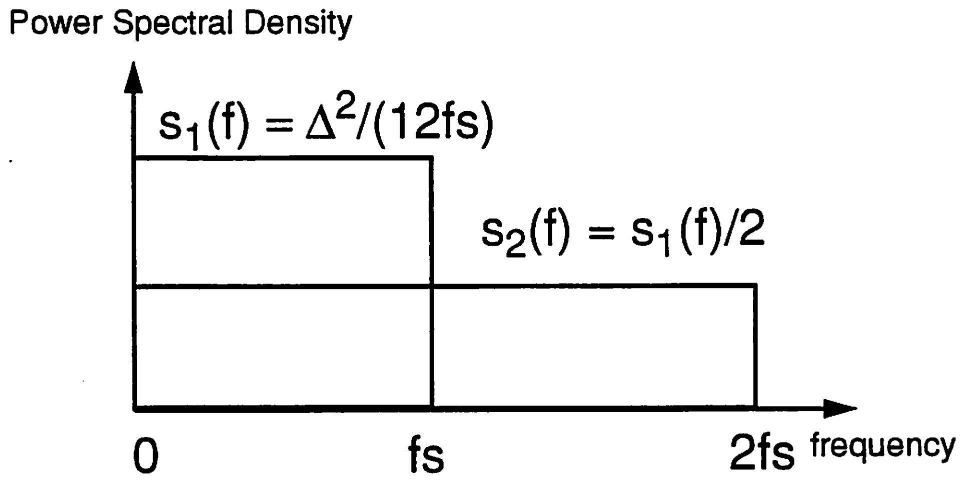


Figure 2.2 Power spectral density of the quantization noise for a signal sampled at fs and 2fs.

Thus the 3 dB gain per octave of oversampling translates into 1/2 of a bit in effective resolution for an A/D converter. This illustrates the fundamental trade-off of resolution for speed in A/D converters. Rather than using better A/D converters which can require costly trim steps, designers can use oversampling and digital decimation filtering which adds costs in on-chip digital filters. The gains through oversampling are limited, since a 2 bit gain in converter performance requires a signal to be oversampled by a factor of $2^4 = 16$. A better solution is to gain more resolution per octave of oversampling by using noise shaping along with oversampling.

2.2.2 Noise Shaping and the Δ–Σ Modulator Family

Noise shaping is achieved by pushing quantization noise away from a particular region of interest and can be implemented using the Δ–Σ modulator shown in Figure 2.3. The circuit con-

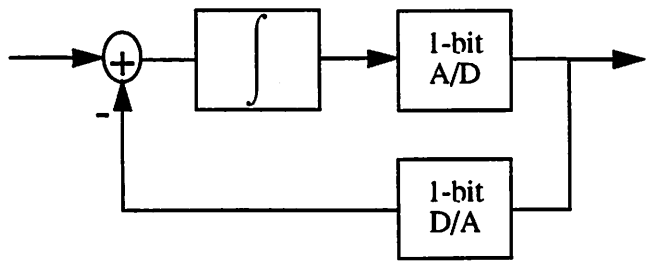


Figure 2.3 Block diagram of a first order Δ–Σ modulator.

sists of an integrator in a feedback loop along with an A/D and a D/A converter. In the figure, 1-bit A/D and D/A converters are used, but noise shaping will occur regardless of the number of bits in the converters. To examine how noise is shaped, the A/D converter can be replaced with an additive white noise source to provide the signal flow graph shown in Figure 2.4.

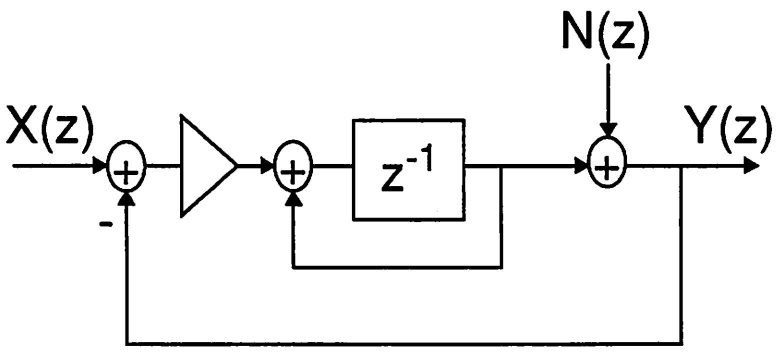


Figure 2.4 Linearized model of the first order Δ–Σ modulator.

Early methods for analyzing oversampling modulators were developed around the additive white noise model. While the model is definitely not accurate for 1 bit quantizers, it has been applied with some success to the design of delta modulators [24], [26]. Early extensions of this method to Δ–Σ modulators were shown to be somewhat promising [27], [28] with circuit results having some degree of matching. The approach taken by Candy will be reproduced here to illustrate the benefits of noise shaping [27].

The linearized flowgraph of Figure 2.4 can be analyzed using linear system theory. The transfer function from input to output is given in Equation 2.2

$$\frac{Y(z)}{X(z)} = z^{-1} \tag{2.2}$$

The transfer function from the noise source to the output is given in Equation 2.3.

$$\frac{Y(z)}{N(z)} = 1 - z^{-1} \tag{2.3}$$

Equation 2.2 is a delay, which causes no signal distortion while Equation 2.3 is a high pass transfer function. The quantization noise has been pushed away from the low frequency baseband region. It must be stressed that this analysis is based on assumptions that are not fully valid. In fact the quantization noise of the 1 bit A/D converter has been shown to be far from white [29]. The results provided by this method provide some insight, but actual coder behavior is marked by nonlinear dynamics which may be better characterized using chaos theory [30].

These noise shaping results can be verified in simulation. The FFT of the output of a modulator with a sinusoidal input is shown in Figure 2.5. The input signal passes through the modulator

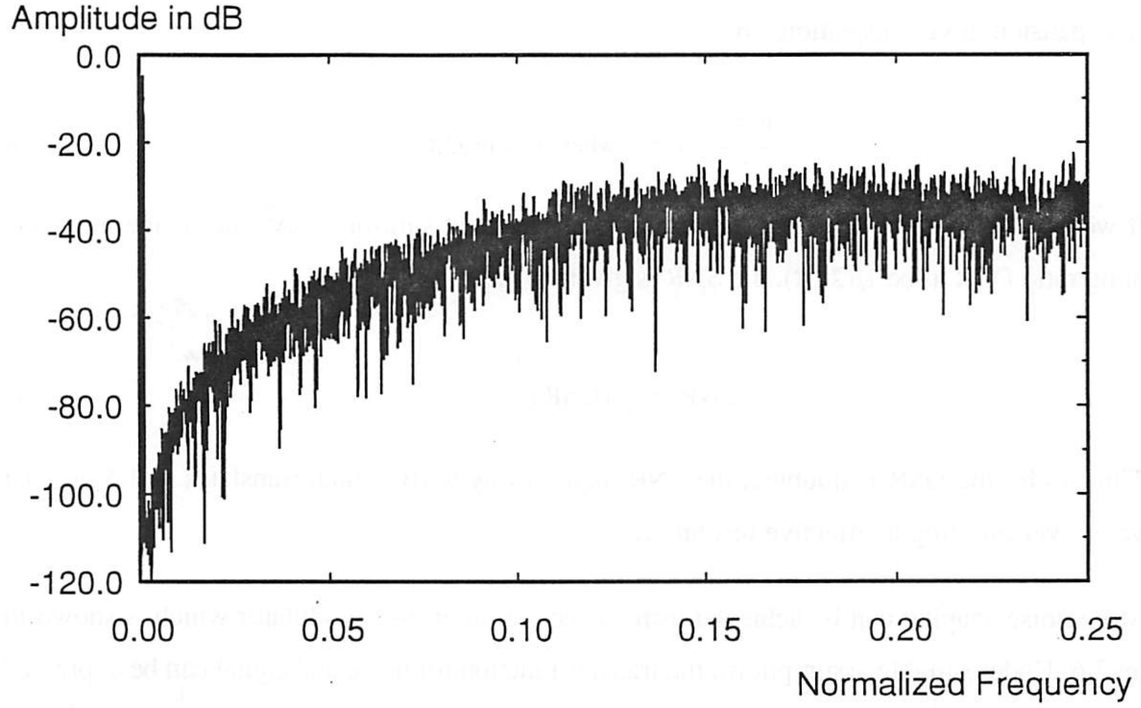


Figure 2.5 FFT output of a second-order Δ–Σ modulator with a sinusoidal input.

with no distortion or attenuation but the quantization noise is shaped with a high pass response. If the high pass noise was filtered out, the original sinusoid could be recovered with high resolution since the low quantization noise region would be preserved.

The gain in SNR per octave of oversampling can be calculated by using Equation 2.3. It is assumed that the 1-bit A/D converter contributes additive white noise with power $e_o^2 = \Delta^2/12$ and the sampling period is τ. The power spectral density of the noise at the output of the modulator can be derived from Equation 2.3.

$$\mathrm{PSD} = 2\tau e_o^2(2-2\cos\omega) \tag{2.4}$$

To find the total in band noise, the power spectral density must be integrated from 0 to ω_0, the highest frequency of interest. After performing the integration, the in-band noise power N can be expressed by Equation 2.5.

$$N = \frac{2e_o^2}{\pi}(\omega_o - \sin\omega_o) \tag{2.5}$$

This can be simplified if the sine function is approximated by the first 3 terms of a Taylor's Series expansion, giving Equation 2.6.

$$N \approx \frac{e_o^2\pi^2}{3}(2f_o\tau)^3 \text{ where } f_o = \omega_o/2\pi \tag{2.6}$$

If we assume that the power of the largest possible input sinusoid is Δ^2 and define the oversampling ratio OSR to be $1/(2f_0\tau)$, the SNR is given by Equation 2.7.

$$SNR = \frac{6}{\pi}(OSR)^{\frac{3}{2}} \tag{2.7}$$

Thus each time OSR is doubled, the SNR increases by 9 dB, which translates to 1.5 bits per octave of oversampling in effective resolution.

More noise shaping can be achieved using a second order Δ–Σ modulator which is shown in Figure 2.6. Under suitable assumptions, the transfer function for noise and signal can be expressed

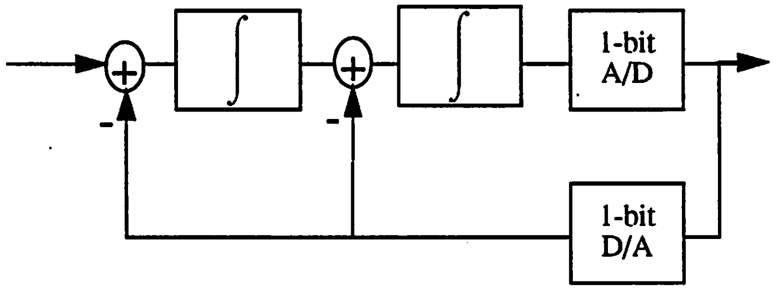


Figure 2.6 Block diagram of a second order Δ–Σ modulator.

as in Equation 2.8.

$$Y(z) = E(z)(1-z^{-1})^2 + X(z)z^{-1} \tag{2.8}$$

The SNR value can be calculated using the method described previously giving Equation 2.9. For each octave of oversampling, a 15 dB gain in SNR is achieved.

$$SNR = \frac{\sqrt{60}}{\pi^2}(OSR)^{\frac{5}{2}} \tag{2.9}$$

These calculations estimate the possible resolution that could be achieved if a perfect brick wall filter were used to eliminate the out of band noise. While the equations are not accurate for design work, they do illustrate the major advantages of oversampling A/D converters. A small set of modulators can be coupled with different digital filter combinations to provide signal acquisition at a wide range of resolutions and sampling rates. Recent results have shown that implementations are possible from 20 bits at low sampling rates [31] up to 12 bits at 2 MHz [32]. Another advantage is in the small amount of precision analog circuitry required. The modulators are simple, requiring a few opamps, capacitors, and comparators. Since little precision analog circuitry is required, these units can be implemented on-chip with digital signal processing systems.

The additive white noise model ignores the nonlinear behavior of the loop. First and second order Δ–Σ modulators exhibit tones that are created by limit cycles in the loop. However, first and second order Δ–Σ modulators are inherently stable. Higher order modulators based on cascades of integrators, like the one shown in Figure 2.7, have been studied [33], but stability problems make

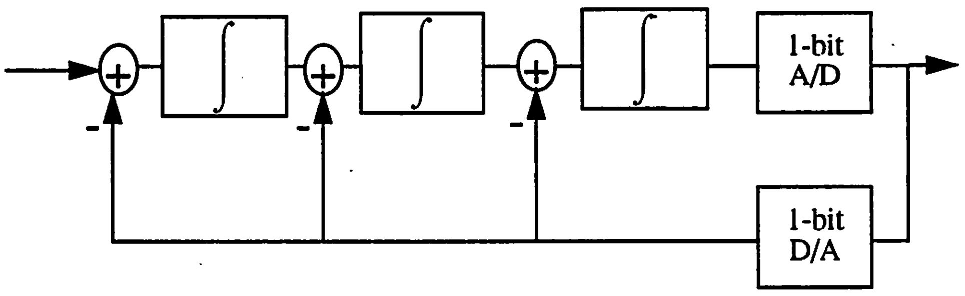


Figure 2.7 A third order Δ–Σ modulator using triple integration.

them less attractive than the first and second order modulators. Theoretical methods can be used to explain the limit cycle behavior.

2.2.3 Theoretical Analysis Methods for Modulators

The additive white noise model does not provide insight into the limit cycle behavior of the modulators, but it does help to predict how much resolution can be achieved. It is important to characterize the limit cycle behavior since it can severely limit usefulness. The structure and size of these limit cycles can be analyzed by studying the nonlinear dynamics of the modulators.

Theoretical analysis has led to better understanding of the nonlinear dynamics of the modulator loop. Early analyses focused on the quantization noise at the output of the modulator in an ideal first order modulator with a DC input [27]. For highly oversampled signals, the input varies slowly over a single sampling period and a DC input is a good approximation. It was found that noise peaks were present in the output spectrum whenever the DC input was a rational fraction of the quantizer stepsize [34], which gives rise to a fixed bitstream pattern at the output. The pattern in the bitstream is periodic so tones can be created in the baseband. Models and equations were developed to allow calculation of approximate SNR values given an input DC bias point and a signal value. The solution to these equations provides information on only a single data point in a given waveform. For a variety of inputs, complete solutions require too much time for calculation limiting usefulness for circuit design. However, these methods can be used to help determine good bias points that do not create baseband tones.

Researchers have shown that the spectrum for the quantization noise of an ideal first order modulator can be calculated for sinusoidal inputs [29] and that the additive white noise assumption does not adequately predict modulator behavior [35], [36]. The power spectral density of the quantization noise has been shown to be discrete in nature rather than white.

It has been shown that if the initial condition of the modulator is known, the limit cycle pattern can be predicted. This analysis problem can also be inverted so that given a limit cycle pattern, the correct input can be identified within an error bound. In theory, this method provides the highest possible resolution for an A/D converter. One algorithm for optimal decoding has been studied for implementation as an integrated circuit [37].

Other theoretical approaches have been applied to stability analysis of higher order loops. One method modelled the quantizer as additive white noise with a linear gain that depends on the input [38]. Using this approach, it was noted that the loop gain of a modulator can decrease with larger inputs. If the loop gain decreases too much, the excess tones caused by the quantizer can cause the loop to go unstable. In addition to predicting this conditional stability, this method provides some guidelines on how to design modulators for stable operation given the signal swing. Another approach used root locus techniques to show that limiters in the feedback loop can add stability to higher order systems [39].

For the most part, these theoretical analyses can provide good design approximations when circuits are limited by quantization noise phenomenon as in digital implementations which have been used to verify these concepts. However, analog circuits contain thermal noise and imperfections that are difficult to include in theoretical analyses, so simulation remains the easiest way to determine the parameters necessary for design.

2.3 Higher Order Noise Shaping Modulators

Δ–Σ modulators are not the only solutions for achieving noise shaping. Researchers have developed other topologies that have better limit cycle properties and higher order noise shaping functions. These topologies can be divided into multistage noise shaping modulators, modulators with higher order loop filters, and modulators that use multibit A/D and D/A converters.

2.3.1 The Multistage Noise-Shaping Modulators

This topology is obtained by cascading low order Δ–Σ modulators to obtain the transfer function of a high order noise shaping function. The trademark name MASH has been applied to one of these topologies by researchers at NTT [40]. The basic concept used in the multistage modulator is to take the quantization error from a modulator and then feed this into another modulator which encodes the noise in a bitstream. Subsequent processing and merging of the bitstreams gives rise to a noise shaping function that is of higher order than any of the component modulators. Detailed comparisons and discussions of the various multistage modulators and the theory of operation have been previously presented [41], [42], [43] and will not be repeated here. The primary advantage of these modulators is unconditional stability even for high order implementations and better randomization of the quantization noise leading to fewer limit cycles. Matching limitations between the individual modulators can limit overall resolution.

Theoretical results have been extended to these modulators [43]. For modulators with an order of 3 or greater, the quantization noise has been shown to be nearly white regardless of the input. This means that the white noise assumption has more value for analysis of higher order multistage structures and can be used as an effective design tool.

2.3.2 Modulators with Higher Order Loop Filters

These modulators make use of multiple feedback and feedforward paths within the loop filter to create a noise shaping transfer function as shown in Figure 2.8. The noise and signal transfer functions can be designed arbitrarily by moving the poles and zeroes in the loop filter. A design method based on classical filter design principles and the additive white noise assumption has been used to determine the gain factors [44]. The major drawback with this topology is the fact that it is

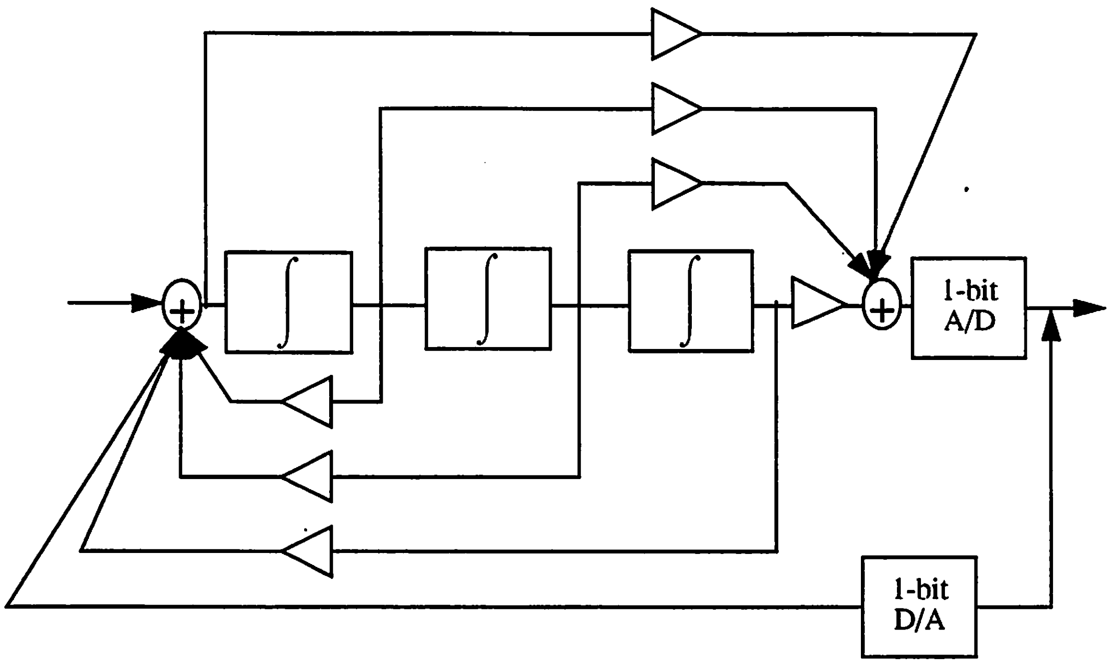


Figure 2.8 Block diagram of a modulator with a higher order loop filter.

conditionally stable [38] since pole placement using the white noise assumption does not guarantee stability. Nonlinear dynamics dictate the behavior of the actual loop so rules of thumb were developed for designing stable systems. If the loop does go unstable, reset circuitry for the integrators can be used to bring the loop to a stable state, correcting the problem.

The modulator can be designed so that the noise transfer function has a notch rather than high-pass response. This modification gives rise to the bandpass Σ–Δ modulators that can be used in superheterodyne receivers [45], [14].

2.3.3 Modulators with Multi-Bit A/D and D/A converters

One of the major benefits of noise-shaping coders is the tolerance to imperfections in analog circuitry. The use of 1-bit A/D and D/A converters provides inherent linearity allowing high resolution conversion limited primarily by opamp non-idealities or excess noise in the system. By using multi-bit A/D and D/A converters in the coder, better resolution can be obtained with lower order modulators or at lower oversampling ratios, but these types of converters are limited by linearity in the A/D and D/A circuits.

Simulation provides the easiest method of assessing how nonlinearities will affect loop performance [46]. These methods have shown that mismatch in step sizes can lead to tones that can be in the baseband. Methods have been proposed to correct nonlinearities in the circuits [47], [48].

2.4 A/D Converters and Filters

The previous discussion has shown that oversampling modulators are complex systems that can be used to achieve high resolution A/D conversion. However, the resolution can only be achieved through the use of digital filtering. Figure 2.9 shows a summary of the functions per-

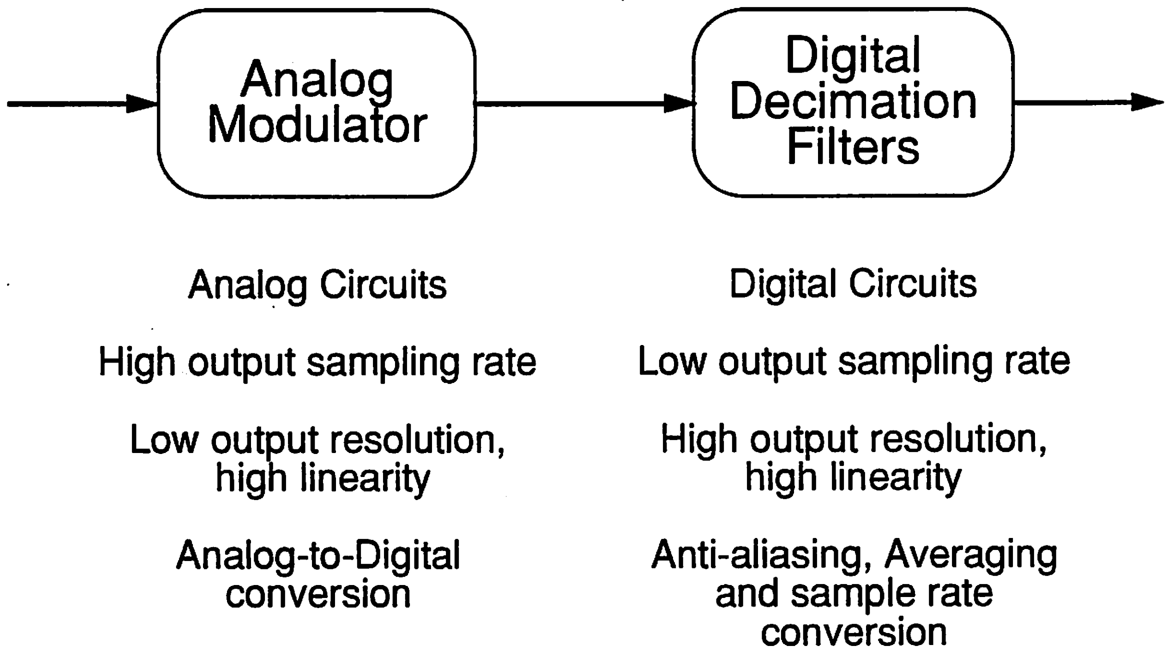

Analog Modulator
Digital Decimation Filters
Analog Circuits
Digital Circuits
High output sampling rate
Low output sampling rate
Low output resolution, high linearity
High output resolution, high linearity
Analog-to-Digital conversion
Anti-aliasing, Averaging and sample rate conversion

Figure 2.9 Summary of functions for the modulator and filters.

formed by the modulator and by the digital filters. Since oversampling is used, the initial anti-alias filter, which isn't shown in the figure, is allowed a very large transition band. This reduces the order of the filter so that 1 or 2 pole RC filters suffice with the bulk of the anti-alias filtering performed in the digital domain where circuits can take advantage of scaled geometries.

Besides providing anti-aliasing, the digital filters average a number of samples over time to implement the speed for resolution trade-off. Since the signal was originally highly oversampled, provisions must be made to reduce the sampling rate to a lower rate. These functions are implemented through the use of multirate digital filters, which will be discussed in the remainder of the chapter.

2.5 Multirate Filter Design

A key issue in oversampling converters is the design of efficient multirate digital filters. Efficiency is complex function of user-defined constraints on area, speed, resolution, and power consumption. While multirate digital signal processing techniques have been studied extensively [49], there is no general method for optimizing these types of systems when mixing a variety of techniques. Some optimization methods have been developed under assumptions such as fixed filter type or fixed architectures [50], [51], [52]. This section discusses several approaches to multirate filter design to provide background for the tools used in the design system.

Throughout this chapter, references are made to multirate and decimation filtering. Decimation is the process of lowering the sample rate. It originally referred to retaining 1 out of 10 samples from a signal, but now is used to describe lowering the sample rate of a signal. Decimation is closely related to interpolation, which is the process of inserting samples to increase the sampling rate. While most of the examples in this chapter are developed with decimation in mind, the results can also be applied to interpolation.

Multirate digital signal processing is based on sampling concepts that were originally applied to implementations of narrow band digital filters which require a steep rolloff between passband and stopband in a small transition width. High order filters are needed to achieve the desired frequency response which results in increased sensitivity to coefficient truncation and long wordlengths. Instead of implementing narrowband filters with a single filtering stage, a better solution is to use several filter stages operating at different sampling rates.

2.5.1 A Multirate Filtering Example Using FIR Filters

As an example, consider the system specified in Figure 2.10. The signal bandwidth of interest is 2 kHz, but the input sampling rate is 200 kHz. There can be other signals outside of the band of interest, so a digital filter must be used to limit the bandwidth of the signal. One way to implement such a filter would be to decimate by a factor of 40, apply a steep filter, then interpolate by 40 to get back to the 200 kHz sampling rate. If we are only interested in the data in the low frequency

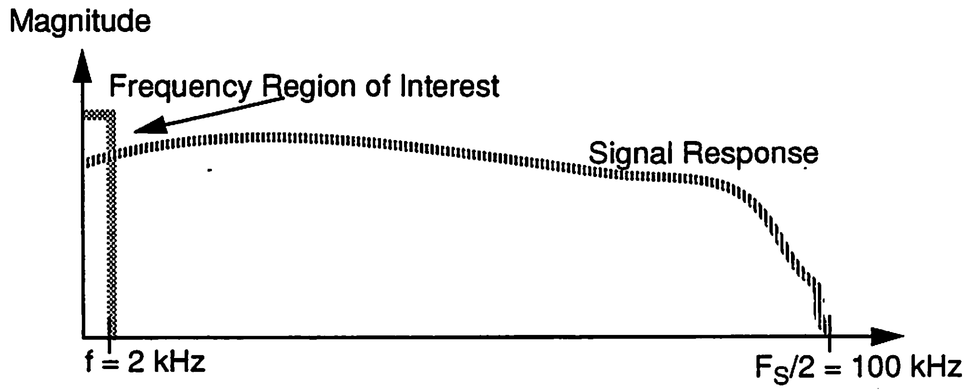


Figure 2.10 Frequency specification for a narrow-band filter.

region, interpolation is not necessary, and the output can be taken at the lower sampling rate of 5 kHz. A block diagram of this approach is shown in Figure 2.11.

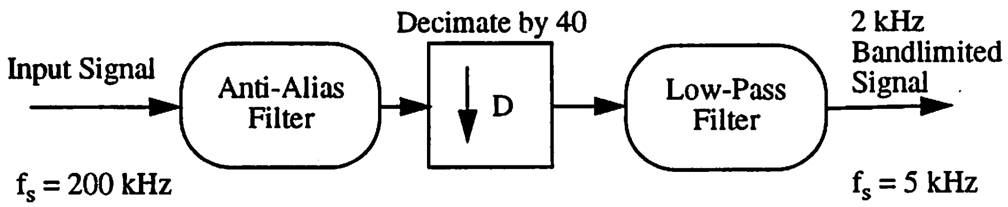


Figure 2.11 Block diagram of a system to recover 200 Hz bandlimited data.

When decimating, simple resampling is not sufficient. Resampling at a lower sampling rate can be achieved by multiplying the current input stream with the Discrete Fourier Series representation of a pulse train that samples at the lower rate. If s[n] is the input signal sampled with period T, and we choose to decimate by a factor D, this can be expressed in the time domain as follows:

$$s'[n] = s[n]\frac{1}{D}\sum_{k=0}^{D} e^{\frac{j2\pi kn}{D}} \qquad (2.10)$$

In the frequency domain, this represents a convolution. The Fourier transform of Equation 2.10 is given in Equation 2.11, where ω is defined in terms of the higher sampling rate.

$$S'(e^{j\omega}) = \frac{1}{D}\sum_{k=0}^{D} S\left(e^{j\frac{(\omega-2\pi k)}{D}}\right) \text{ where } \omega = 2\pi/T \qquad (2.11)$$

The aliasing effect expressed by Equation 2.11 is illustrated in Figure 2.12 for the case where

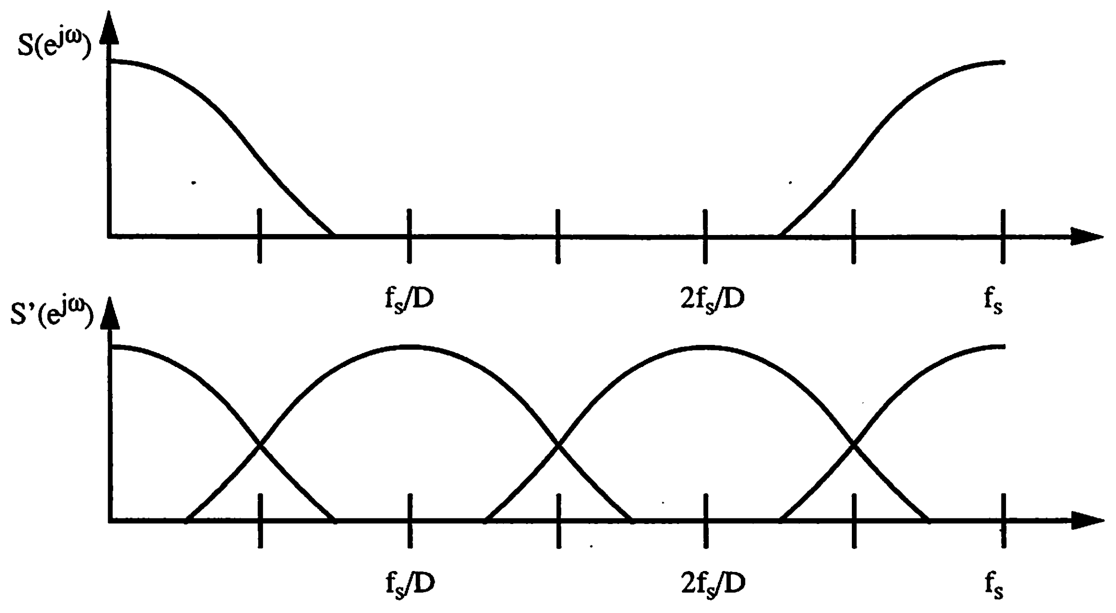


Figure 2.12 Frequency domain interpretation of resampling a signal that isn't bandlimited.

D = 3. Clearly, the signal must be bandlimited before decimation. Specifications are usually given to define how much the aliased signals should be attenuated.

Suppose the specification called for 40 dB of anti-aliasing for frequencies above 3 kHz up to 100 kHz. If we choose to limit the implementation to one that uses equiripple FIR filters, we can quickly estimate the complexity of the problem by estimating the length of the required filter using a well known design equation [53]. A single FIR filter would require a filter of length 328. A more effective solution is to use 2 filters, one that decimates by 10 and one that decimates by 4. To avoid aliasing, the filters must have specifications as shown in Table 2-1. Using the same design equa-

TABLE 2-1 Summary of filter requirements

Filter	Dec. Ratio	Passband Edge	Stopband Edge	Number of taps required
Single Filter Decimation Stage	40	2 kHz	3 kHz	328
First Filter in 2 stage	10	2 kHz	18 kHz	24
Second Filter in 2 stage	4	2 kHz	3 kHz	39

tion, we find that the first filter requires 24 taps and the second filter requires 39. There are fewer total coefficients needed and the second filter operates at a much lower rate, so less computation is needed. These results can be analyzed to find the number of operations required to implement the filters.

In calculating the total number of operations, the figures must be adjusted since FIR filters have advantages when used in multirate applications. Each output is calculated as a linear combination of past input values. Since there are no recursive operations, only the output values at the lower sampling rate must be calculated. Normally, for every input sample supplied to an FIR filter with L taps, L state variables must be updated with a multiply-accumulate operation and one output sample is generated. This is illustrated in Figure 2.13 for the case where L = 4. When decimat-

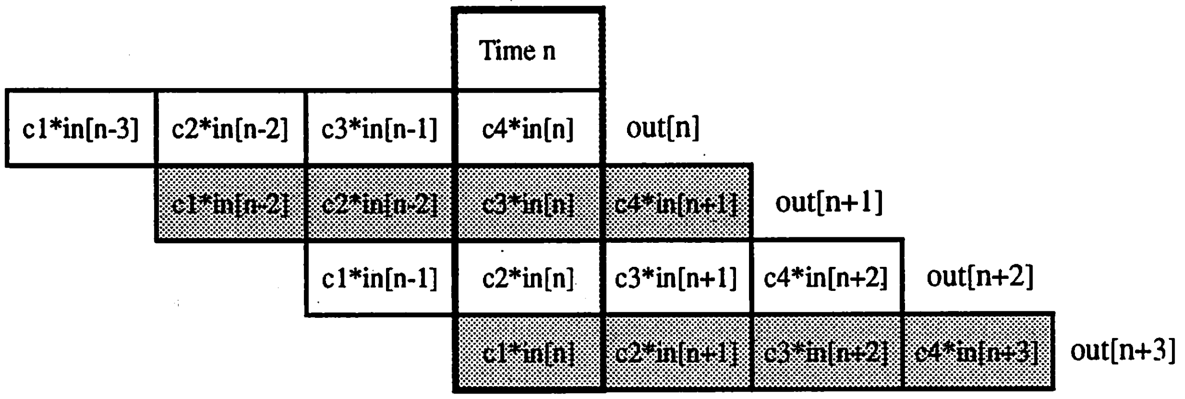


Figure 2.13 Calculations in an FIR filter with L = 4.

ing by a factor D, only L/D state variables must be updated since only these L/D state variables contribute to output samples that won't be discarded. In Figure 2.13, only the unshaded multiply-accumulate operations need to be calculated, since out[n+1] and out[n+3] will later be discarded. This means that the computation rate can be decreased by a factor of D when decimation is incorporated within the FIR filter.

Considering the 2 examples again, we can calculate the number of multiply-accumulate operations needed per sampling period to implement both options. If decimation isn't incorporated in the first example, then 328x40 = 13120 multiply accumulate operations are required per output sample generated and 39 intermediate output samples are discarded. If decimation is incorporated in the filter, the first example requires 328 multiply-accumulate operations per output sample, showing that quite a gain can be achieved. In the 2-stage example with decimation incorporated in both filters, the first filter must be run 4 times and the second filter once per output sample generated. This means that 4x24 + 39 = 135 multiply accumulate operations are required, about 1/3 of the total for the first example. The second example can be easier to implement, since it requires fewer operations per output sample but this depends on the hardware implementation.

2.5.2 Other Solutions to Multirate Filtering Problems

In the last section, 2 solutions to a decimation filtering problem were examined and other combinations could provide even better results. Crochiere and Rabiner developed design charts and rules to guide the design of these types of multirate filters when only cascaded equiripple FIR are used [49]. These results have shown that computations tend to be minimized when the decimation ratios of all stages are similar. Another approach proposed a branch-and-bound optimization scheme locate the optimal structure based on cost functions that are applicable to certain VLSI realizations [52].

FIR filters do have many desirable properties that make them easy to work with but they are computationally inefficient for narrowband filtering. IIR filters can require fewer operations to implement a given magnitude specification, but they cannot have a linear phase response over all frequencies of interest. Additionally, it is more difficult to gain computational advantages when altering the sample rate with IIR filters. Classical IIR filters include biquadratic and lattice wave digital implementations of Butterworth, Chebyshev, and Elliptical magnitude approximations.

Martinez and Liu proposed one IIR structure that takes advantage of sample rate alteration to lower the computation rate [54]. This structure is similar to the one proposed by Bellanger [55]. The numerator of an IIR transfer function can be implemented as an FIR filter that only needs to compute samples at the output rate. If the denominator is implemented with delays that are multiples of the decimation ratio, then it also can operate at the lower sampling rate since no intermediate states need to be calculated. This is illustrated in for a decimate by 2 case in Figure 2.14 where the computation rate can be cut in half.

This idea was extended by Ansari and Liu to polyphase IIR filters [56]. Each branch of a polyphase filter approximates an all-pass function. The all-pass functions are restricted so that they use delays that are multiples of the decimation or interpolation ratio. These functions can be implemented with fewer multiplies, so an extra computational savings can be achieved above that given in the previous scheme. Since the branches are all-pass, fewer degrees of freedom are available, limiting the range of magnitude responses that can be implemented. The filter design method for

Desired Transfer Function for a decimate by 2 filter

$$H(z) = \frac{a_1 + a_2 z^{-1}}{1 + bz^{-2}} = (a_1 + a_2 z^{-1})\left(\frac{1}{1 + bz^{-2}}\right)$$

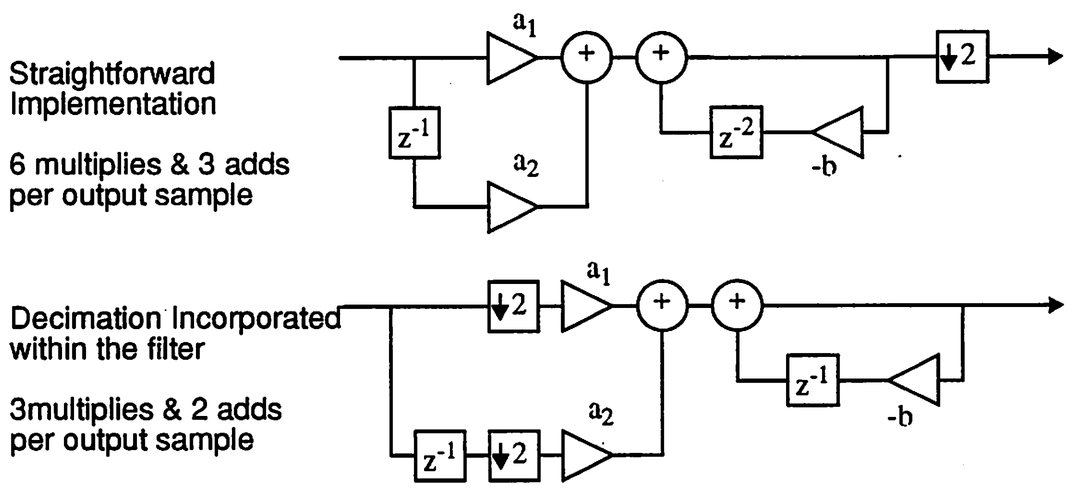

Straightforward Implementation
6 multiplies & 3 adds per output sample
a_1
+
+
z^{-1}
a_2
z^{-2}
-b
↓2
Decimation Incorporated within the filter
3multiplies & 2 adds per output sample
↓2
a_1
+
+
z^{-1}
-b
a_2
z^{-1}
↓2

Figure 2.14 Incorporating decimation in an IIR filter structure.

these polyphase filters leads to implementations that are close to N-th band filters. A half-band filter is a special case of an N-th band filter and has the property that the magnitude response is symmetric when mirrored about the point $f_S/4$. If a half-band filter has frequency response H, it must satisfy Equation 2.12.

$$|H(e^{j\omega})|^2 = 1 - |H(e^{j\omega})|^2 \tag{2.12}$$

The property that governs the frequency response for N-th band filters is given in Equation 2.13 for filters having N branches and decimating or interpolating by N [57].

$$\sum_{n=0}^{N-1} \left| H\left(e^{j\left(\frac{\omega + 2\pi n}{N}\right)}\right) \right|^2 = 1 \tag{2.13}$$

The -3 dB frequency occurs near fs/N. N-th band filters can be implemented using FIR filters as well [58], but the decrease in computation is not as dramatic than for polyphase IIR Nth-band filters which have been shown to be the most computationally efficient filters for sample rate alteration. These polyphase filters are good for small changes in sample rates. They can maintain good passband and stopband ripple, but there is too much computation involved for large sampling rate changes.

While previous methods have shown that using many stages can be beneficial in lowering computation rates, this is not a good guideline for VLSI implementation of filters. Using additional filtering stages may lower the average number of operations, but can greatly increase the chip area. Rather than using extra stages of large complexity and large area, another approach has been to use very simple FIR filters for large sampling rate changes.

This approach partitions the sample rate change into 2 tasks: sample rate change and bandshaping [50]. Some filters are assigned the task of sample rate change, while a few are used for bandshaping. The sample rate change filters focus mostly on anti-alias requirements and not much attention is given to the passband providing more design freedom and lower computation requirements. After the sample rate change, bandshaping filters are used to correct for droop caused by the sample rate change filters and to give the correct bandshaping according to the specification. These filters can be difficult to design since they have an arbitrary magnitude response in the passband.

Simple comb and integrator filters were used for sample rate changes. These filters are based on cascades of filters with rectangular windows and are sometimes called Comb-Integrator Cascade (CIC) filters. In continuous time, the frequency response of a rectangular window is described by the sinc function, sinc(x) = sin(x)/x, giving rise to the name sinc filters. The transfer function for a first order sinc filter is given in Equation 2.14 for the case where the decimation ratio is N and N is odd.

$$H(e^{j\omega}) = \frac{1-e^{-j\omega N}}{1-e^{-j\omega}} = \sum_{n=0}^{N-1} e^{-j\omega n} = \frac{\sin(\frac{N}{2}\omega)}{\sin(\omega)} e^{-j\omega(\frac{N-1}{2})} \tag{2.14}$$

These filters can be implemented using an accumulate and dump FIR filter or using integrators and differentiators. Candy used filters with these types of magnitude responses as optimal decoders for Δ–Σ modulators [59]. In practice the structure based on integrators and differentiators is used since the implementation can be very efficient in area.

2.6 Summary

In this chapter, the basic concepts involved in oversampling A/D converters were presented. It was shown that the noise shaping ability of the modulator determines the gain in resolution per octave of oversampling and that several different modulator types are available. In order to achieve the full conversion, digital filtering and sample rate alteration are needed. Several filtering techniques were introduced and basic strategies for decimation filtering were presented. Figure 2.15

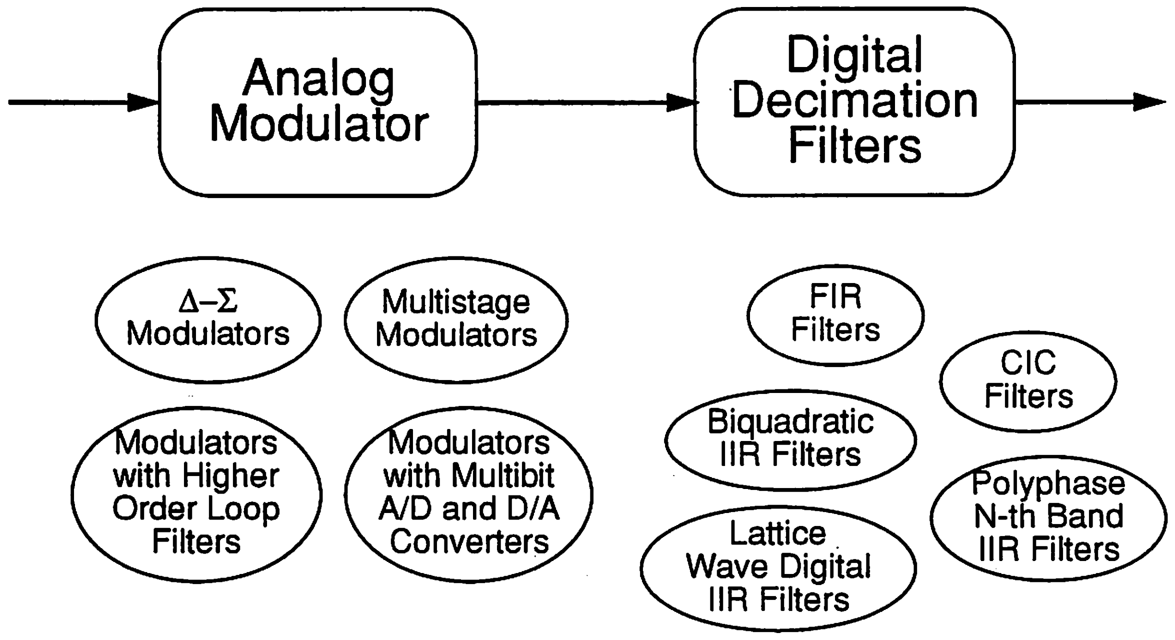

Analog Modulator
Digital Decimation Filters
Δ-Σ Modulators
Multistage Modulators
Modulators with Higher Order Loop Filters
Modulators with Multibit A/D and D/A Converters
FIR Filters
CIC Filters
Biquadratic IIR Filters
Polyphase N-th Band IIR Filters
Lattice Wave Digital IIR Filters

Figure 2.15 Modules for implementing an oversampling A/D converter.

illustrates the concepts introduced in this chapter.

In order to implement a converter, the basic idea is to select from the blocks shown in Figure 2.15 and to combine them in a way that implements the desired function. There is a large amount of flexibility, since resolution requirements can be met by selecting higher order modulators or increasing the decimation ratio. Since most of the filtering is performed in the digital domain, the full implementation can be integrated with other digital signal processing circuits in standard CMOS.

While oversampling converters provide flexibility, a major problem is that widely varying solutions can often be comparable in performance. For example, varying implementations of oversampling A/D converters for encoding speech have been reported in the literature using different algorithms and architectures [60], [61], [62]. While the modulator design was basically the same, different approaches were taken for the digital filters but performance results are not that different. This indicates that no single optimal solution exists.

Optimization techniques can help to direct the designer towards a good solution, but overall optimization of this type of system is not easy. The design is too complex for a single optimization algorithm since a variable number of complex system components are involved. Since interactions between parts of a system are hard to predict, it is possible that two very different solutions could both give good results. Our design methodology attempts to explore these options through hierarchical design estimation. High-level design strategies need to focus on the performance of the modulator and the design of digital multirate filters.

CHAPTER 3

Framework Overview

In this chapter, a complete overview of the design system is presented. The details of high-level design are discussed and a prototype tool for high-level design is presented. An overview of the CAD framework for design and the design database are also presented.

3.1 Hierarchical Design Estimation

3.1.1 Introduction

Hierarchical design estimation tries to provide the designer with only a minimal amount of information needed to make critical design decisions at a given level of design abstraction. However, it can be difficult to decide what type of information to present to the designer. Past filter design algorithms have used the number of operations and memory storage as design metrics. While these numbers can be helpful for comparing software implementations of filters, they do not serve as adequate metrics for IC design since comparisons in the different design phases can be broad and application specific. For example, a designer may want to compare an FIR filter against biquad and wave digital implementations of digital filters. Comparing the number of operations is not sufficient because it can be strongly dependent on the implementation. A custom FIR filter may only be slightly larger in area than the others if a full multiplier is used. Additionally, the wave digital implementation may be favored since it has better limit-cycle properties and can be implemented in small area with shift-add arithmetic. These issues make comparison difficult, so only the most basic metrics should be used.

Many aspects of oversampling A/D converter design are quite application dependent. For example, some designers insist that a constant group delay response in the passband is necessary for audio applications while other designers will allow group delay to vary with an unclearly specified amount of tolerable variation. Another basic choice is in the analog modulator design, with different modulator topologies providing advantages with respect to stability, ease of use, and sensitivity to process variations.

These issues require human designers to be involved and make a fully automated solution difficult. Studies have shown that the main problems in decision making are due to large numbers of variables. Saaty has shown that hierarchical approach can be used to limit the number of variables in cases where multiple criteria must be considered [63]. His approach showed improved results for decision making, even for problems where metrics for comparison were difficult to quantify. Thus narrowing the number of variables and using hierarchy allows the designers to critically examine the trade-offs and to focus on the real factors affecting the decision process. In addition, once critical factors are identified, the designer can alter the specification to insure that they are not overly conservative. Better designs can be achieved when suboptimal solutions to smaller problems are balanced to give a good overall solution.

This approach is applicable to design systems where the design decisions are not easily characterized into rules that would allow development of automated tools. It has also been used as a strategy for developing design methodologies for the design of complex systems [64]. As design experience is gained and as the methodology matures, it may be possible to develop design critics that can fully automate the design process.

Area, power, speed, and resolution were chosen as the basic metrics that would enable good design decisions. Economic issues force designers to balance these quantities, so it seems appropriate to use them as metrics for all levels of the design hierarchy.

3.1.2 Estimation Strategies

Within the framework, the architecture level is the lowest level at which estimation occurs. Since architecture templates are used, estimation of area and speed can be performed based on information from the layout generation tools and cell libraries. Estimating the resolution for the analog modulators can be difficult, but simulation methods discussed in Chapter 4 were shown to provide adequate information. The simulation models can be used with regression techniques to generate a mathematical model for the behavior of the modulator as a function of high-level design parameters. Estimation of power consumption is a difficult problem that is currently being studied [65]. Power estimation tools were not used in this project, but the framework should support these tools when they become available.

Estimation at the algorithm level can be heavily influenced by the choice of implementation. Architecture and circuit features can alter the actual performance numbers and skew the estimation process. To avoid these problems, a hierarchical approach is used. At the different levels of hierarchy, estimators are designed to produce data that can drive lower level estimators. Full design estimates can be generated by applying the estimators in a top-down fashion. A common set of parameters was defined to simplify comparisons between numbers based on different design styles.

This hierarchical approach can be illustrated in the estimation of area for the implementations of an FIR filter. At the algorithm level, the length of the filter can be estimated from frequency domain specifications using a set of empirical formulas [53]. This is propagated to the architecture level estimators. In addition to the length of the filter, the sampling rate and the number of non-zero bits needed to code the coefficients are needed for estimation. The sampling rate is available from the initial specification while the non-zero code bits can be estimated using a basic rule of thumb [66]. The area estimate is found by choosing an architecture, and applying the architecture estimator to the input data.

3.2 Design System Overview

3.2.1 Objectives for the Design System

The basic objective for the design system is to automate the process of converting specifications into layout for oversample and decimate A/D converters. The design system must handle the complexity and breadth of the design tasks from implementation to layout and chip testing. Each of the processor modules requires different types of expertise to achieve a successful design. In addition, the interactions between these modules must be accounted for, since beneficial interactions between modules can be exploited to reduce design costs. These requirements led to these goals that we have set for the design system:

1. To provide a means for designers to focus on one aspect of the design problem, such as modulator design, while relying on library modules to allow full implementation of a complete A/D interface.
2. To provide designers with the tools to study design trade-offs between widely varying implementations.
3. To provide a modular environment for designers to study oversampling A/D and D/A conversion and multirate digital signal processing which allows easy integration of new architectures and design techniques.
4. To investigate the possibility of portable module generators for oversampling modulators.
5. To simplify and speed up design times for a class of mixed signal VLSI systems.

3.2.2 CAD Framework

A CAD framework was defined to support the analysis and design of these systems. The basic flow for the design process is shown in Figure 3.1. Within the CAD framework, tools are needed for hierarchical design estimation and for implementing designs. Methods were established to allow efficient communication between design tools and to provide designers with access to the relevant design information. The design process was not totally automated since designers are forced to make decisions about implementation choices based on information from the estimation process. If the choices for system components, algorithms, and architectures have been made, much of the design process can be automated.

The approach to the CAD framework was partitioned between design tasks and estimation tasks. Most design programs are not interactive and can be coordinated by specifying a set of sequential design tasks. This can easily be implemented using a set of design programs and some shell scripts or a design manager program. The long execution times of design programs makes them less favorable candidates for interactive tools, so designers are accustomed to waiting for the results. However, when designers are performing estimation, they would rather have an interactive tool to work with. Immediate feedback allows designers to use “what if” analyses to quickly isolate effective design techniques. Rather than implementing a fully interactive design system, the decision was made to use a traditional design manager for design tasks and to use a spreadsheet for the estimation tasks. This provides a solution that is easy to implement and provides much flexibility and modularity. Thus the CAD framework is divided into estimation tools implemented using a spreadsheet and the detailed design tools implemented using as programs coordinated by a design manager.

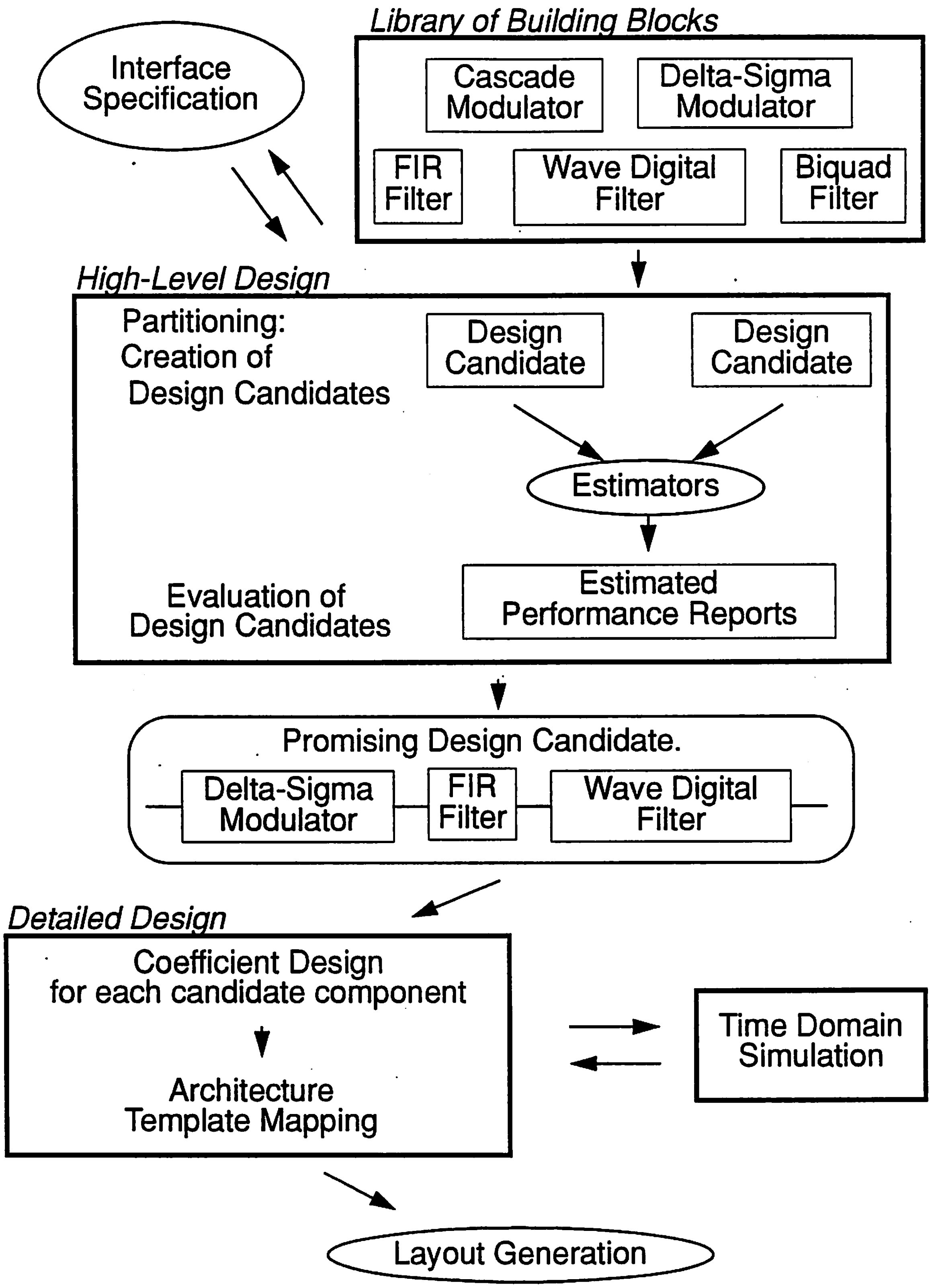


Figure 3.1 Basic design flow for the design system.

3.2.3 Status of the Design System

A complete set of detailed design tools was integrated into a CAD framework. A simple design manager was created and verified. Various modules can be specified and connected using a hierarchical design language. The design manager can then be used to direct the fully automated process of creating the structure descriptions used by the layout generators. After annotating these descriptions, the layout generators then create the mask descriptions for fabrication.

For the estimation tools, only a minimal set of equations was integrated into a spreadsheet. Separate tools for modulator selection and digital decimation filter design were created. These tools provide estimation of algorithmic parameters. The architectural level estimators were implemented along with the detailed design tools, so the full hierarchical design estimation concept was not implemented. Work on this phase has begun, but was not completed. Future work can focus on refining these tools and providing a strong link to the detailed design tools.

In the following sections, a brief overview is given of some basic spreadsheet tools and also for the detailed design tools for implementing a design. In addition, information about the database for the detailed design tools is given along with a description of adding programs into the CAD framework. In Section 3.8, the use of the design system is discussed.

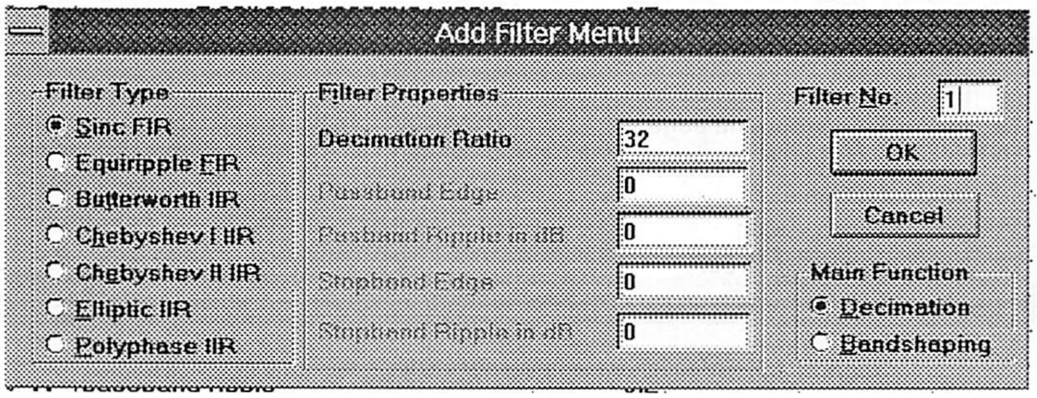
3.3 Spreadsheet-Based Design Tools

Since the high-level estimates can drive the architecture estimators, a large amount of design exploration can be performed using only estimators. Estimation techniques are not fully accurate so the designer needs to examine many designs. In addition, this search must be well documented to allow comparison of these high-level estimates to those obtained later in the design process when more information is available. The high-level design tools needed to aid the designer should be characterized by fast response time and an interactive interface. These tools try to capture the early calculations that most designers make when initially studying the design.

The algorithm and architecture estimators can be characterized by a set of mathematical equations which are usually simple in complexity and can often be implemented using a programmable calculator. In this project, a simple high-level tool for design exploration was implemented using a spreadsheet. Speadsheets allow the design equations to be stored as parameterized macros and the calculations can be performed interactively resulting in immediate updates of the design information after a specification change. The macro mechanisms allow new estimation routines or techniques to be quickly developed, tested, and made available to designers. Some spreadsheets allow customization of user interfaces so that a custom application can be developed without the need for custom code and provide summaries of data obtained during a session.

The basic design formulas for designing cascades of digital decimation filters were coded into a spreadsheet design tool using Microsoft Excel. A custom user interface was designed to simplify data entry. The frequency domain specifications for the overall filter are first entered into the spreadsheet. Users then have the option of adding different types of digital filters.

When a filter is added to the cascade, the minimum information required is the filter type and the decimation ratio. The menu for adding a filter is shown in Figure 3.2. If the filter is meant to


Add Filter Menu

Filter Type: [✓ Sinc FIR] Equiripple FIR, Butterworth IIR, Chebyshev I IIR, Chebyshev II IIR, Elliptic IIR, Polyphase IIR

Filter Properties:

Decimation Ratio	32
Passband Edge	0
Passband Ripple in dB	0
Stopband Edge	0
Stopband Ripple in dB	0

Filter No. 1

OK

Cancel

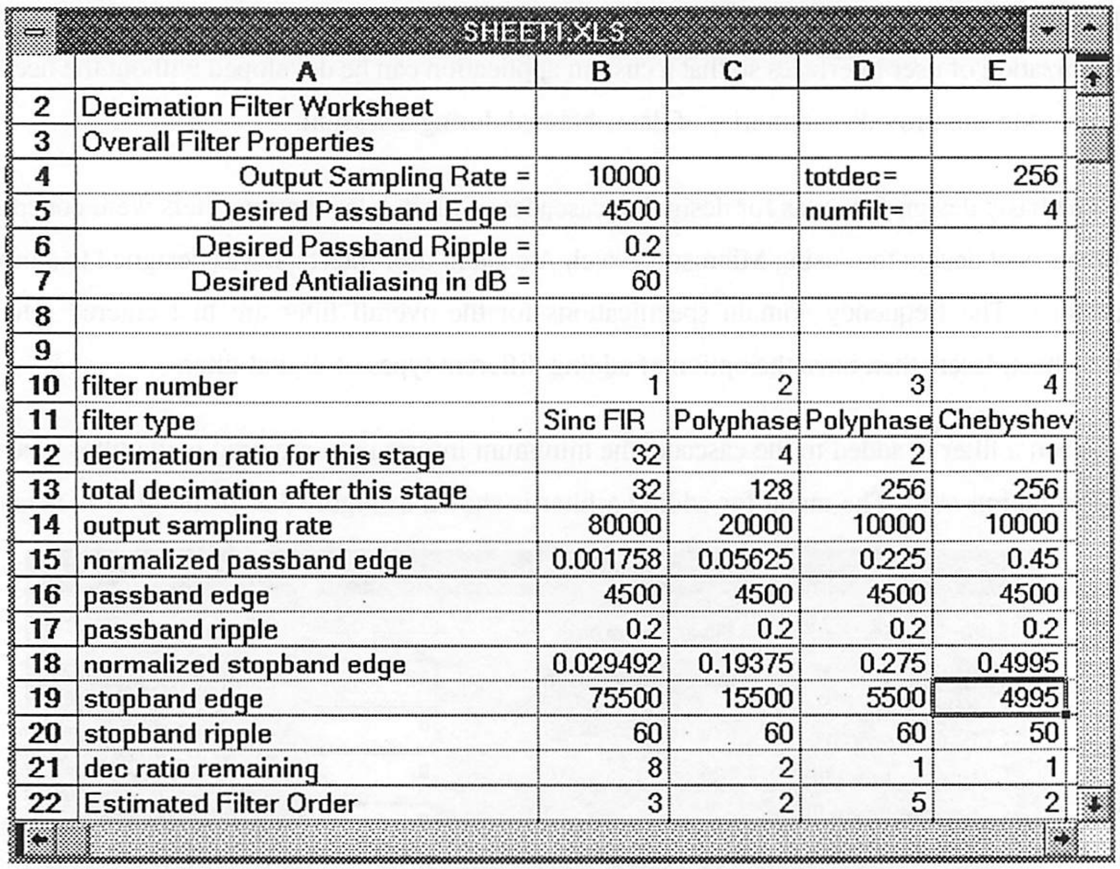
Main Function: [✓ Decimation] Bandshaping

Figure 3.2 The custom add filter menu generated in Excel.

have a bandshaping response, additional information can be entered. Otherwise, the spreadsheet is programmed to automatically calculate the proper band edge requirements. Figure 3.3 shows an Excel worksheet which displays the results from a design session. In this worksheet, the user can alter the decimation ratio for each filter stage to see the effect on the overall filter implementation and the default macros will continue to update the estimates for the required filter order. A second set of macros can be added for the estimation of die area required based on an architectural choice and additional macros for estimation of power and resolution can be easily developed.

3.4 Framework for Detailed Design

The CAD framework for design is based on the design flow shown previously in Figure 3.1. The tools used in the framework can be grouped into tools for coefficient design, architecture map-



	A	B	C	D	E
2	Decimation Filter Worksheet				
3	Overall Filter Properties				
4	Output Sampling Rate =	10000		totdec=	256
5	Desired Passband Edge =	4500		numfilt=	4
6	Desired Passband Ripple =	0.2			
7	Desired Antialiasing in dB =	60			
8					
9					
10	filter number	1	2	3	4
11	filter type	Sinc FIR	Polyphase	Polyphase	Chebyshev
12	decimation ratio for this stage	32	4	2	1
13	total decimation after this stage	32	128	256	256
14	output sampling rate	80000	20000	10000	10000
15	normalized passband edge	0.001758	0.05625	0.225	0.45
16	passband edge	4500	4500	4500	4500
17	passband ripple	0.2	0.2	0.2	0.2
18	normalized stopband edge	0.029492	0.19375	0.275	0.4995
19	stopband edge	75500	15500	5500	4995
20	stopband ripple	60	60	60	50
21	dec ratio remaining	8	2	1	1
22	Estimated Filter Order	3	2	5	2

Figure 3.3 Spreadsheet template for studying multirate digital filters.

ping, layout generation, and simulation. A simple design manager has been implemented to oversee the design tasks.

The framework is based on 3 clear breaks in the design process, one at the end of high-level design, one at the end of coefficient design, and one before layout generation. These breaks partition the detailed design task into well defined design tasks that can be automated. At the breaks in the design process, the human designer makes decisions on which directions to take. The design tasks between breaks are well defined, but the implementation of the tools to automate the task is not overly restricted. Any variety of techniques could be used to implement a design task as long as design information is read and written according to a fixed database policy. This provides flexibility and allows many existing design programs to be easily ported into the framework. The breaks in the design process also encourage designers to investigate how the design is progressing and allows them to find problems before investing excessive design time at the lowest levels.

The tools for coefficient design, architecture mapping, and simulation are discussed in the following chapters. Layout generation is performed using the Lager Silicon Assembly System [67].

3.5 The Detailed Design Database

The high-level design exploration data is stored in spreadsheet templates. Once detailed design begins, a well-defined database storage policy is needed is needed to dictate how design modules interact. The database serves a key function since a seamless interface between programs will allow higher levels of design automation. Designers are then free to focus on the critical design decisions rather than worrying about how to change data formats for the next design program.

For this application, the amount of design data that needs to be stored during design exploration is small. A simple database could be implemented using ASCII files to store data. The information consists of a set of properties, a frequency domain specification, and a set of coefficients. Issues like access time and relational querying are not a concern. Organization of the data and design management are more important considerations.

3.5.1 The OCT Database

The design system uses the OCT database [68] for storing information. OCT was originally developed for VLSI CAD applications and is an integral part of the CAD framework at Berkeley. OCT is an object-oriented database with well defined procedures for storing and retrieving data. In addition, many design tools have been created for OCT, and new design management tools are being developed. It is possible to use the design manager VOV [69] with the programs created in this project.

Within OCT, a policy must be defined to dictate how data is stored. OCT contains several objects that can be combined in different ways to create representations of a design called views. The rules governing the attachment of objects are called a policy, and each type of view has a policy.

The basic data structures in the database are bags, properties, terminals, nets, and instances. A bag is a construct that can contain OCT objects while a property is an object that stores some data. Properties can have values that are of the type real, integer, or string. Terminals and nets are used for describing connectivity. Instances are use to represent other OCT views within a given view.

The information that must be stored for a design can be broken down into categories that correspond to the levels of abstractions. The transition from one level of design abstraction to the next creates additional design information that needs to be stored. While separate OCT views could have been used at each stage, having all the information in a single view allows easier access and less clutter. The majority of information for the design system is stored using a policy briefly described in the next section.

3.5.2 OCT Component View Policy

The OCT component policy was defined for this project and is based on an earlier view defined for data storage in FIRGEN, a tool for automated VLSI implementation of FIR filters [70]. A simplified version of the policy is shown in Figure 3.4. OCT generic policy is supported, which

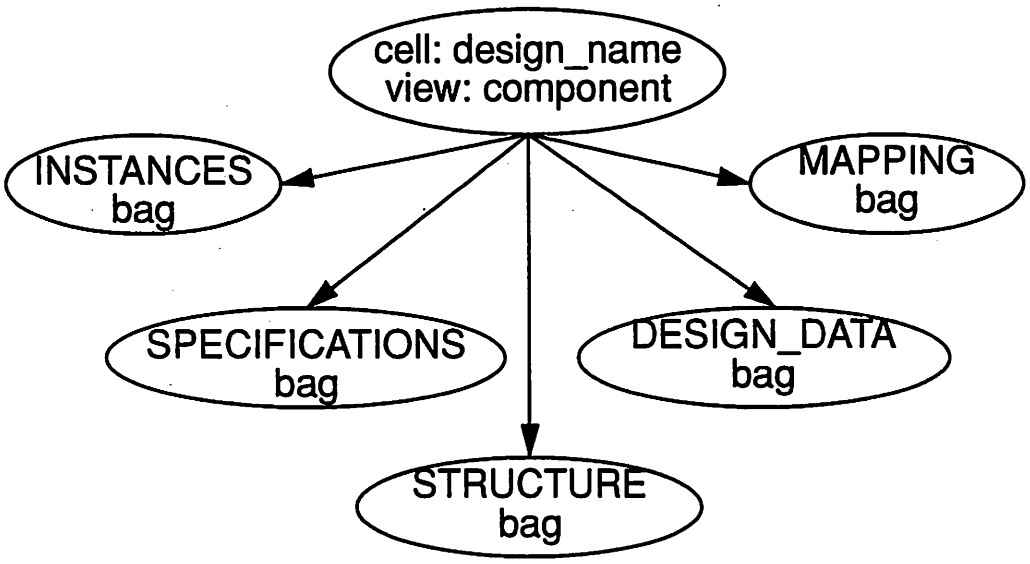


Figure 3.4 Top level of the policy for the component view.

allows instances and provides for interconnection of views using terminals and nets. Each of the bags in the view is assigned a policy depending on the data stored in it.

The frequency domain specification is stored in the SPECIFICATIONS bag using the format depicted in Figure 3.5. The specification is used to describe the frequency domain response of the component and is given in terms of behavior in a frequency band. Each band is characterized by at least 2 points, which specify the band edges. At each frequency point, the desired magnitude response and allowable deviation, both given in decibels, are stored as real properties. An optional

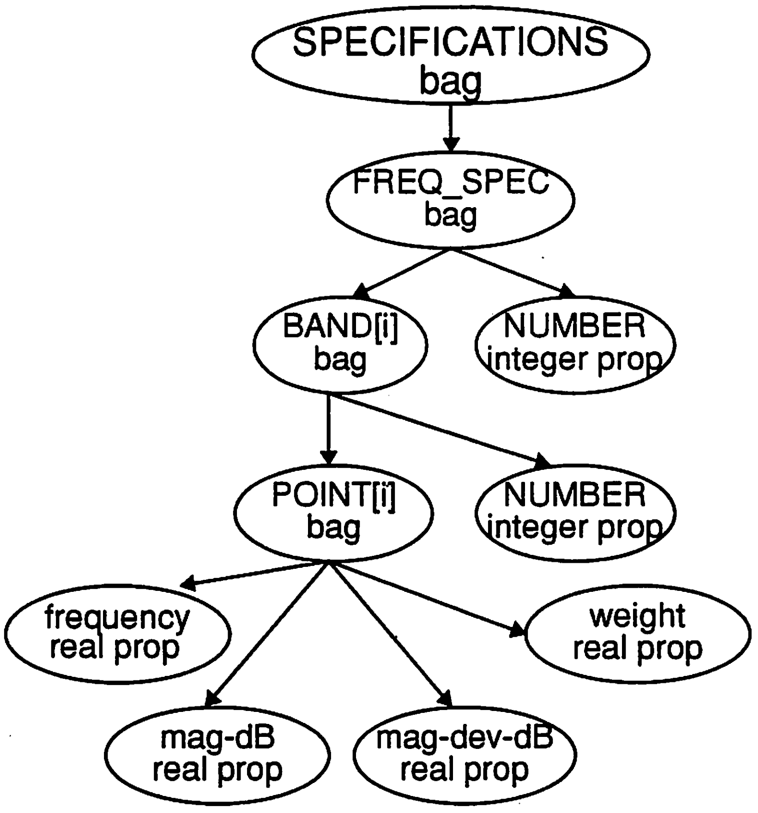


Figure 3.5 Policy for storing the specification for the frequency response of a component.

WEIGHT property is also stored to indicate the weighting that is needed in some design programs. The weighting usually indicates the relative effort given to satisfying the frequency specification. The NUMBER property is used to indicate the number of items in each bag. BAND and POINT bags are stored with indexes to allow separate bands to be retrieved with a single call.

The STRUCTURE bag holds the major properties needed to start the design for a given component. In addition, other properties that will be defined in later stages of the design process are also kept in this bag. Table 3-1 shows a list of some of the properties used in the design system. Required properties are denoted with an asterisk. These required properties form a minimal set suitable for running nearly all of the coefficient design programs of the first detailed design phase. While it may appear that most of the properties refer to digital filter design, the set can be expanded to handle other tasks. Coefficient design tools read the information in the STRUCTURE bag and then write results to the DESIGN_DATA bag.

The DESIGN_DATA bag holds the coefficient values determined during the design process.

TABLE 3-1 Basic properties defined for the STRUCTURE bag.

PROPERTY NAME	TYPE	DESCRIPTION
SAMPLING_FREQ*	REAL	Input sampling rate for the block.
DEC_RATIO*	INT	Decimation ratio implemented in the block.
ORDER*	INT	ORDER of the filter, which is determined by FIL-TER_TYPE.
FILTER_CLASS*	STRING	Values are: HIGHPASS, BANDPASS, BANDSTOP, OTHER
FILTER_STRUCTURE*	STRING	Values are: LATTICE_WDF, BIQUAD_CAS-CADE, LINEAR_ARRAY, POLYPHASE_LWDF, and POLYPHASE_BIQUADS
MAG_APPROX*	STRING	Values are: BUTTERWORTH, CHEBYSHEVI, CHEBYSHEVII, ELLIPTICAL, EQUIRIPPLE
INTERP_RATIO	INT	Interpolation ratio implemented in the block
PTOLEMY_MODEL	STRING	Name of the model to use for this block in simula-tion.
LIBRARY_CELL	STRING	Gives the name of the library cell to use for direct mapping.
INPUT_WORDLENGTH	INT	Input wordlength expected by the block.
OUTPUT_WORDLENGTH	INT	Output wordlength desired from the block.
INTERNAL_WORDLENGTH	INT	Wordlength used for internal calculations.
COEF_WORDLENGTH	INT	Wordlength for coefficient coding.
CODING_METHOD	STRING	Describes method for coding coefficients. Values are: TRUNCATION, ROUNDING, CSD.
COEF_CODE_FORMAT	STRING	Either UNSIGNED_BINARY, SIGNED_BINARY, or CSD.
PHASE_APPROX	STRING	Only LIN_PHASE allowed. Assumed to be DONT_-CARE if not assigned a value.
ARCH_FILE	STRING	Specifies the architecture description file.

Coefficients for all filter structures are stored using a linear array structure. The basic structure of the DESIGN_DATA bag is shown in Figure 3.6. Ordering of array is maintained using the bracketed index notation. Coefficients for different filter types are stored according to a policy defined for each FILTER_STRUCTURE type. In the CODED_COEFS bag, the CODE property is used to store a string representation of the actual coding. The floating point value of the coded coefficient is also saved since it is used in simulation.

The MAPPING bag holds information that will be needed for generating structure descriptions. Two properties should be stored in this bag by the architecture mapping programs. The PAR_FILE property gives a full file system path pointer to a set of parameter values that will be

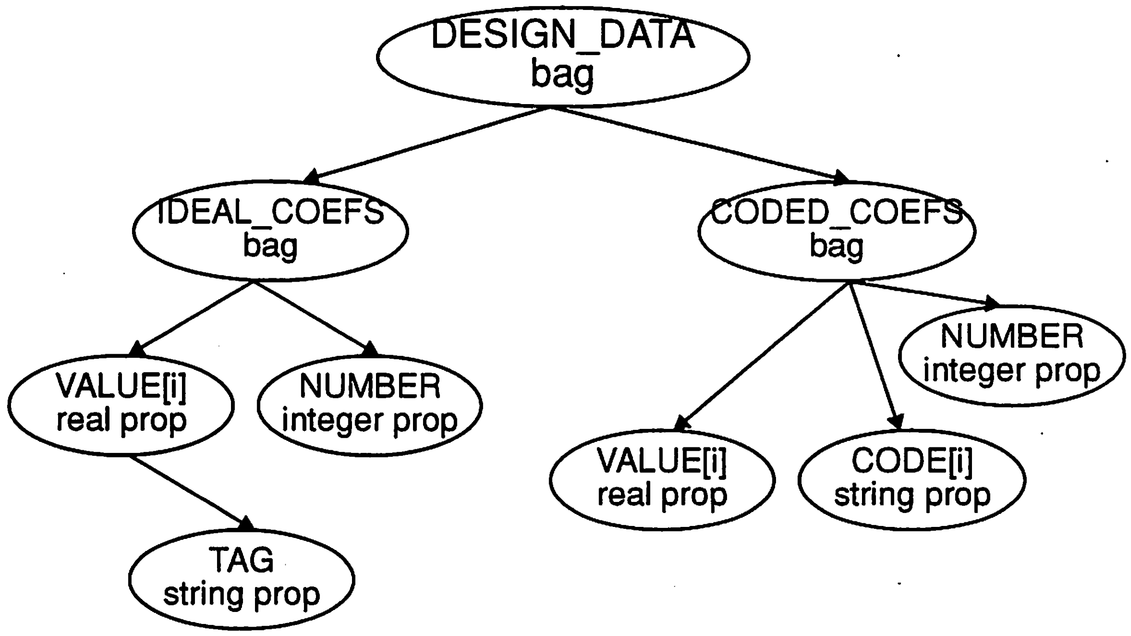


Figure 3.6 Policy for storage of coefficients in the DESIGN_DATA bag.

required for a structure description. The SMV_NAME property gives a file path pointer to an OCT view created from the structure description created by the architecture mapping program. The architecture mapping programs can also use this area to store other temporary information. The MAPPING bag also contains an ESTIMATES bag in which the architecture mapping programs can write various estimates as properties.

Several additional bags have been added to the top level cell in the database. These are called MAKE[i] and are used by the design manager. A set of COMMAND[i] string properties are stored in the MAKE bags. These properties store a command line used to invoke a particular design tool. Each COMMAND property contains a TIMESTAMP property which is used by the design manager to store timestamps for this step of the design process. As design tasks are completed, the design manager can update the timestamp so that on the next run, the design task is not repeated. The value of TIMESTAMP is a long integer value from the **time()** function in C.

In conjunction with the design manager, a design documentation tool was developed. A signal acquisition module is comprised of several components. The design manager attempts to run the coefficient design tools on each instance in the full design before attempting architecture mapping. As each design is completed, a documentation window can be called to allow the designer to note details of the design process. This information is stored in a separate database where it can be retrieved and examined throughout the design process. This information can be helpful since the design process is iterative in nature.

3.6 Adding Programs to the Design Framework

Details on design programs that were integrated into the design framework can be found in the following chapters and in Appendix B. Other programs can be integrated into the design system by providing routines for the program to read and write the component view database. This section discusses programming style and various routines developed to aid interface design. It is the task of the programmer to create interfaces so that design data is correctly read from or written to the database. Adherence to the policy guarantees that all programs will be able to communicate and share data.

3.6.1 Consistency in Programming

In all the programs, an effort was made to use consistent programming techniques. The first step in the process was the use of common exit codes for all the programs. The basic exit codes are listed in Table 3-2. Program exits are handled using the errtrap routines provided in the OCT pro-

TABLE 3-2 Exit codes for programs used in the design system framework.

Number	Defined Symbol	Condition
0		Normal program exit, no exceptions detected.
1		Generic code for an exception.
2	BAD_CLA	Bad command line argument passed to the program.
3	BAD_OCT_VIEW	An error was encountered processing the OCT database.
4	BAD_OCT_CALL	An error was encountered from calling an OCT function.
5	BAD_INPUT	Input data from a text file was not valid.
6	BAD_DATABASE	Similar to BAD_OCT_VIEW.
7	BAD_SYS_CALL	An error was detected at the return of a system() call.
8	BAD_INT_DATA	An error or inconsistency was detected within internal data structures.
10 and above .		These exit codes are reserved for the program. Symbols should be defined in the program header file along with definitions of error conditions.

gramming libraries. The exit call from this library provides the exit code and allows the programmer to supply a short message to guide the user in fixing the problem. The symbol names for the exit codes are defined in a common header file, dblib.h, which will be discussed later in this section. In addition to exit codes, all programs supply a -L option which specifies that extra program information will be written to log files.

The OCTTOOLS distribution provides several programming libraries in addition to the errtrap library. Utilities are available for searching paths, processing command line arguments, and for shortcuts in accessing the database. Use of these routines simplifies the programming task and gives more uniformity to the programs. A utility is available for generating makefiles to generate dependencies and process all the OCT related libraries. In addition to the OCT libraries, a set of routines were created in the dblib library supplied with the design system. These routines provide support for storing filter coefficients and processing other component view data structures. The header file dblib.h was created with function prototypes for the dblib routines and the standard exit codes are also defined in this file. A summary of the dblib routines is provided in Appendix D, along with the file dblib.h.

3.6.2 Database Interfaces for Coefficient Design Programs

Almost all filter design programs take a frequency specification as input. Various filter design algorithms are discussed in Chapter 5. Based on this information, coefficients are created and then displayed as output. The database interface will thus consist of a routine to read the frequency domain specification from the FREQUENCY_SPEC bag and a second routine to format the coefficients so they can be written to the DESIGN_DATA bag.

The frequency specification is available through a call to the function **fspec2array()** on the main facet of the component view. The specification is returned in a C structure. For writing the coefficients to the IDEAL_COEFS bag, a call to the routines **farray2dd()**, **tags2dd()**, or **codes2dd()** is all that is needed. Prior to the call, the coefficients must be written to an array in a format that is defined by the FILTER_STRUCTURE property. Details on the specifications for the array and for the frequency specification are found in Appendix D.

All coefficient design programs should be able to perform a design with only these properties stored in the STRUCTURE bag: SAMPLING_FREQ, DEC_RATIO, INTERP_RATIO, ORDER, FILTER_CLASS, FILTER_STRUCTURE, and MAG_APPROX. The definitions of these properties may have different meanings depending on the design program. Forcing design programs to use only these parameters simplifies the user interface. When changing design programs, designers will immediately know which parameters to specify without having to look in the documentation.

3.6.3 Database Interfaces for Coefficient Quantization Programs

These programs must read the coefficients from the IDEAL_COEFS bag along with some filter properties, perform quantization, then write the corresponding codes and their floating point equivalents to the CODED_COEFS bag. Methods used in these programs are discussed in Chapter 5. The ideal coefficients can be obtained from the component view by using the function **farray2dd()** which returns an array of type double with the coefficients formatted according to the FILTER_STRUCTURE property, as explained in Appendix D. When retrieving the coefficients from the database, programmers may want to assign a tag or alternate name to the coefficient so that the coefficients can be identified when they are processed by other programs. The **tags2dd()** routine allows programmers to do this.

After quantization, 2 sets of coefficients should be produced, one with the coded coefficients and one with the equivalent floating point values. The coded coefficients are stored as strings since a single numeric value is not sufficient, especially in the case of Canonical Sign Digit (CDS) coding where -1, 0, and 1 are used as digits. The second array is needed so that numerical simulations can be performed on using the values for the quantized coefficients. The coefficients need to be formatted into arrays and then written to the CODE_COEFS bag using the functions **codes2dd()** and **farray2dd()**.

After quantization, additional properties can be added to the STRUCTURE bag if they were not added previously. Properties like COEF_WORDLENGTH, CODING_METHOD, and COEF_CODE_FORMAT all should be defined and added.

3.6.4 Interfaces for Architecture Mapping Programs

The goal of the architecture mapping programs is to convert the quantized coefficients into the information necessary for layout. This varies according to the technique used for architecture mapping. Some examples of mapping techniques are described in Chapter 6. In some cases, it may require generation of the netlist description for a VLSI processor while in other cases, it may only require generation of parameters for a parameterizable architecture template.

No fixed policy was defined for how architecture mapping programs should access the database. The only major requirement is that the MAPPING bag be used for storing intermediate results. At the end of the process, the last program invoked on the component view should write the properties SMV_NAME, which gives the name of the database representation of the structure description, and PAR_FILE, which specifies the name of the parameter file to use with this structure description. The database for the structure description is generated by processing a LAGER structure description language (SDL) file [67]. SDL files can be used to store netlist information about a given architecture implementation.

If any estimates are created by these programs, they should be written to the ESTIMATES bag contained in the MAPPING bag. The basic structure of the ESTIMATES bag is documented in Appendix D. The design system program for documentation will look in this bag to summarize estimates for the design report.

3.7 Layout Generation

Prior to layout generation, a program called writeSDL can be run to create a skeleton SDL file that describes the entire design. The program attempts to write the SDL file using the SMV_NAME and PAR_FILE properties from the MAPPING bag. It collects all the parameter names and applies a transformation to insure uniqueness for all parameter names. No effort was made in the program to attempt the connection of nets between the modules in the design. This problem is not easily solved since it involves a one-to-many mapping. Extra information is needed to connect the various testing features, clock lines, and other signals between components. After the writeSDL program has been invoked, the generated SDL file must be annotated prior to layout generation.

Layout generation is performed using the Lager Silicon Assembly System. The Lager System consists of a set of cell libraries and tools for tiling (TimLager), place and route (Flint), datapath compilation (dpp), and standard cell compilation. Lager assembles cells according to a netlist description given in the SDL format which provides constructs that allow parameterization of designs.

3.8 Using the Design System

3.8.1 Basic Design Steps

The design system was created to aid design space exploration and system optimization. It is assumed that system users have a basic knowledge of multirate digital signal processing and A/D converters. A solid understanding of these concepts will allow designers to created and modify specifications and to take advantage of various signal processing options to optimize the design. Detailed knowledge about algorithms, architectures, and circuit design are not necessary, but will provide extra help in optimizing designs. The basic design task is to create the layout for an analog modulator and a set of digital filters that will meet resolution and frequency domain specifications. Appendix C provides a detailed design trace of an example using the design system. Tutorials are also available as part of the design system distribution.

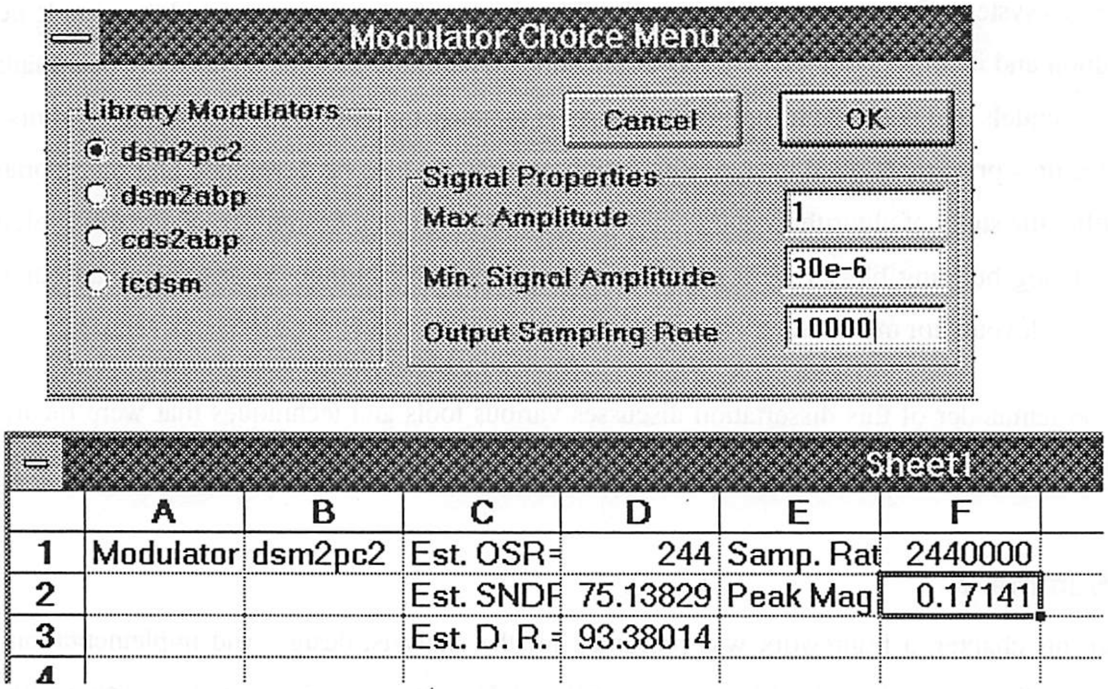
The first step in the process is to create a specification for the desired application. A complete specification will provide information on:

1. Desired output sampling rate.
2. Desired frequency domain response, including magnitude response, phase response, and anti-aliasing requirements.
3. Peak signal magnitude and the desired magnitude for minimum signal resolution.

The information about signal magnitudes can be translated to the desired dynamic range and peak signal to noise plus distortion (SNDR) values. These 2 pieces of information basically set the number of bits for the A/D converter and are needed to choose the modulator.

All of the modulators in the architecture library are characterized using simulation models and measured data from fabricated chips. Data was collected and regression techniques were used to develop quadratic models for predicting peak SNDR and dynamic range for a given oversampling ratio. Each of these models has been incorporated into a spreadsheet tool for estimating modulator performance. The main menu is shown in Figure 3.7 along with basic results. The user enters the information from the specification and chooses a modulator from the list. The results are written to a spreadsheet, giving information about the necessary oversampling ratio to achieve the desired dynamic range, the estimated peak SNDR and dynamic range, and the peak magnitude of the signal for the peak SNDR value. Once a modulator has been selected, the designer can use the digital decimation filter design aids presented earlier and pictured in Figure 3.3. The spreadsheet tools are meant to be used so that several design candidates are enumerated and compared. The power of the design system lies in the ability to compare several widely varying solutions to identify the best overall solution.

The spreadsheet tool for decimation filter design provides estimates of the order of the filter and the passband and stopband edge. This information must be translated by hand to the .adi for-



Library Modulators
[✓ dsm2pc2] dsm2abp cds2abp fcdsm

Signal Properties

Max. Amplitude	1
Min. Signal Amplitude	30e-6
Output Sampling Rate	10000

	A	B	C	D	E	F
1	Modulator	dsm2pc2	Est. OSR=	244	Samp. Rat	2440000
2			Est. SNDF	75.13829	Peak Mag	0.17141
3			Est. D. R.=	93.38014		
4						

Figure 3.7 Modulator estimation tool and results implemented in Microsoft Excel.

mat supported by the detailed design tools. The .adi extension is short for Analog to Digital Interface. When the input files are complete, the detailed design tools are invoked.

In the detailed design phase, designers can make adjustments to the specification. Results from coefficient design can be compared to the estimates. These results may affect the design so that another candidate becomes more favorable. Magnitude and phase responses can be displayed after coefficient design and quantization. Simulation models can be generated and simulation used to verify the specification. In the last step of the design, architecture mapping is applied. Based on the design information, the architecture estimators can be invoked providing further information about design candidates prior to layout. After architecture mapping, a final structure description is created and LAGER is used to generate the layout.

3.8.2 Other Modes For Design System Use

The last section illustrated a typical design session for a user that relies only on modules supported in the design system, but the design system does support researchers in developing new algorithms and architectures. The optimization tools are based on information related to currently existing design tools so the optimization search process is geared towards finding a solution that can be implemented. However, these tools can also be used to point out things that are lacking in the design system. Once an architecture or algorithm is identified, a designer could then implement a solution and incorporate it in the CAD framework. In addition, designers can write new mathematical models and code them into the spreadsheet to study the effects of different algorithms or architectures prior to detailed design. Simulation models can also be developed and incorporated to further the study of algorithms and architectures prior to detailed design. At the circuit level, the basic analog building blocks were included with the tool distribution, so designers can quickly build new layouts for modulators.

The remainder of this dissertation discusses various tools and techniques that were incorporated into the design system starting with simulation methods in the next chapter.

3.9 Summary

In this chapter, a framework was presented for the analysis, design, and implementation of analog to digital interfaces based on oversampling A/D converters. High-level analysis methods were incorporated along with modularity and reuse of structures providing a framework that allows rapid design space exploration with expert advice supplied by estimation tools.. Instead of relying on black box compilers, the designer becomes actively involved in finding and exploiting beneficial design trends. As later examples will show, designers from many fields can take advantage of this open framework to prototype new ideas without having to invest design effort in creating the surrounding components or tools.

CHAPTER 4

Simulation Methods

The ability to produce high-level simulation models for complex systems allows designers to study design trade-offs. Low-level circuit models often require large amounts of time for simulation. High-level models speed up the simulation by only modelling effects that have an impact on the final system. A key factor in model development is establishing which factors are critical in each application. While libraries can be used to store some information, the ability to formulate models frees designers from reliance on libraries allowing them to study circuit configurations.

4.1 Simulation Methods

Simulation provides the easiest method for studying the design of noise shaping coders. Most current design methods are based on simulation models which are used to verify circuit requirements and tolerances. Simulation models are used to generate a set of data points that are then analyzed by windowing methods or the Minimum Sinusoidal Error method [71] to determine signal to noise ratios. This section presents a brief overview of methods that have been applied to the simulation of modulators for oversampling A/D conversion.

4.1.1 Circuit-Level Simulation

Circuit-level simulation uses mathematical models to replace each of the devices in the circuit. Circuit descriptions are created based on layout information or schematics and used as input for a transistor level simulator. This approach is very time consuming when applied to oversampling converters since typical circuits use oversampling ratios of greater than 30. To create a single output then requires more than 30 clock cycles, and each clock cycle may take hundreds of time step iterations. Even with fast computers, a single simulation run can take weeks of CPU time. Simulations on extracted layouts have been performed using HSPICE on a Sun Sparcstation 1. It required 18 days of CPU time to generate 6200 output samples from a second order $\Delta-\Sigma$ modulator. Since the desired decimation ratio was 128, this was not enough data to provide conclusive results on dynamic range. It also turns out that numerical rounding errors in the calculations can limit the simulation accuracy [72]. This approach should be avoided for signal-to-noise ratio analysis, but it is valuable for layout verification when used for a few clock cycles.

4.1.2 Difference Equations and Behavioral Simulation

Since most oversampling modulators are designed using sampled data techniques, these systems can be described using difference equations. This approach works well for digital noise shaping coders which are often used in oversampling D/A converters. The equations correctly model the non-linear behavior in the feedback loops but non-idealities affect the behavior for analog circuits and modifications must be made to the difference equations. For switched-capacitor implementations, this is done by isolating non-idealities and deriving equations that capture the behavior. Hauser was one of the first to use this approach for oversampling A/D converters [73] and he showed that opamp gain and capacitor nonlinearities can severely limit modulator performance. Boser took this one step further by tabulating extensive results for second order $\Delta-\Sigma$ modulators [74]. This approach has also been applied to other topologies by various researchers [40], [75], [42], [41], [44]. While these simulations are very fast, they can lack the accuracy that circuit models can provide.

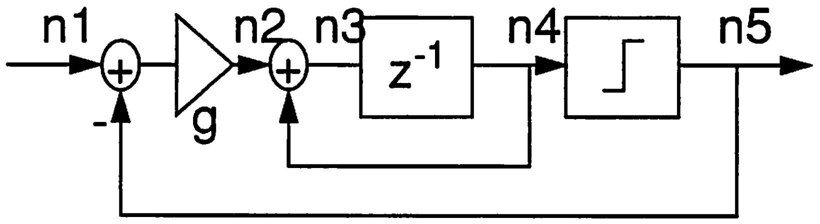
Instead of using difference equations, circuits can also be characterized by simpler models. Instead of simulating a full opamp transient every cycle, a few cycles can be use to characterize a set of look-up tables. Given the input and the output values for previous cycles, interpolation methods based on table data can be used to estimate the actual circuit response [76]. This method is related to difference equation models since it is basically a discrete time simulation with tables providing the network information rather than equations. Simulation speed is still reasonably fast, but the method has the potential for better accuracy since the tables are based on actual circuits. Since the tables are only valid for a single circuit, they must be recompiled for each new circuit configuration and process technology.

An approach based on behavioral modeling has been presented by Wolff [72]. Rather than using actual circuit data, behavioral models are derived based on knowledge of the types of circuits that could be used. The circuits are partitioned into piecewise linear and continuous time sections. Equations are developed for each model section and these equations describe the behavior given the model state. For continuous time sections, pole-zero models are used to determine continuous time response. This method is similar to table-based methods, but has the advantage that the models are more general and can be developed without assuming a circuit configuration. The flexibility of behavioral modelling can provide better results since table-based methods rely on interpolation of a few data sets.

4.2 Developing Accurate High-Level Simulation Models

Accurate simulation models of the analog modulators are needed to predict performance based only on process information allowing designers to simulate actual designs and get an indication of performance without having to go through fabrication. The difference equation model was chosen because it is fast and applies to a variety of circuit implementations. For a first-order modulator, the basic equations and the system diagram are shown in Figure 4.1. Finite gain and capacitor non-

Flowgraph



C code implementation

```
n2 = g*(n1 - n5);
n3 = accumulate(n4,n2)
if (n4 > 0) n5 = refval;
else n5 = -refval
n4 = n3;
```

Figure 4.1 Block diagram and C code fragment for modelling a first-order Δ–Σ modulator.

linearity can be accounted for by analyzing the difference equations for a switched-capacitor integrator.

4.2.1 Switched Capacitor Integrator Analysis.

A switched-capacitor integrator can be analyzed using charge conservation. The effects of nonlinearity and finite gain have been investigated using a combined time-domain and frequency-domain analysis to study the effects of distortion in switched-capacitor filters [77]. These methods have been modified [73], [78], and are presented in detail here.

Consider the circuit shown in Figure 4.2. When $\phi 1$ is asserted, the input is sampled and a

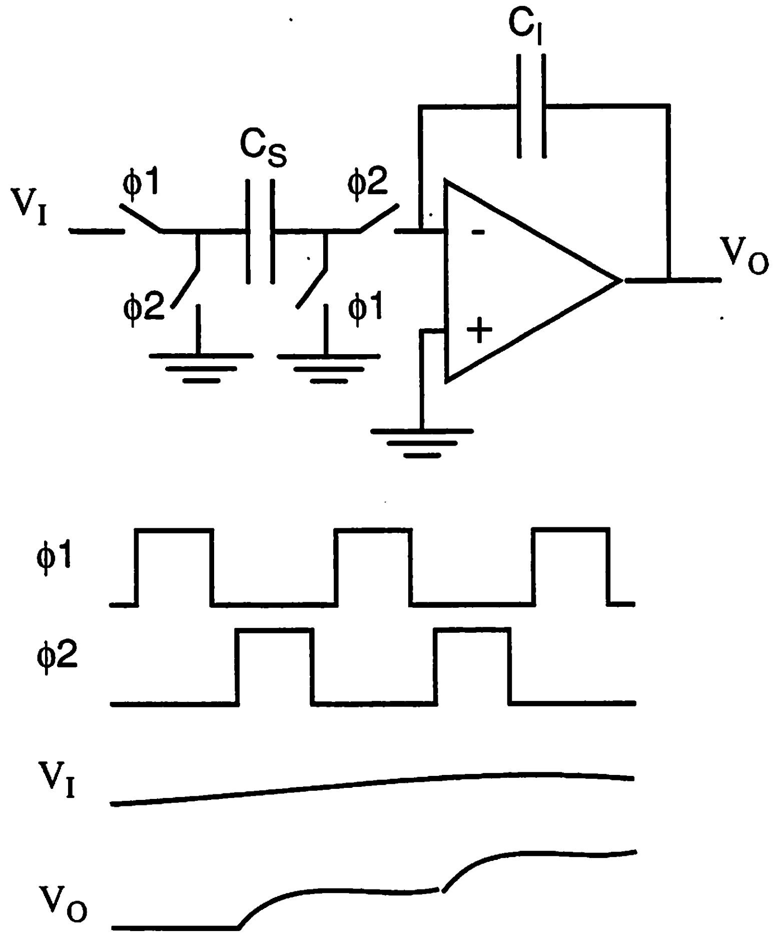


Figure 4.2 Switched capacitor integrator and waveforms.

charge is placed on C_S. During $\phi 2$, the charge is dumped on to the integrating capacitor. Ideally, this is modelled using the following equation.

$$C_I \times V_O(nT) = C_I \times V_O(nT-T) + C_S \times V_I(nT-T) \tag{4.1}$$

By using a discrete time representation, Equation 4.1 can be rewritten as a difference equation.

$$V_O[n] = V_O[n-1] + \frac{C_S}{C_I} V_I[n-1] \tag{4.2}$$

This equation is used to replace the function **accumulate()** in the C code shown in Figure 4.1.

4.2.1.1 Modelling the effects of finite opamp gain

To model finite gain in the opamp, the equations must be modified. A residual voltage will remain on the sampling capacitor C_S at the end (falling edge) of $\phi 2$. If the open loop gain of the opamp is A_{OL}, the magnitude of the residual voltage is V_O/A_{OL}. Incorporating this change in the charge conservation equation gives:

$$C_I \times V_O(nT) \times \left(1 + \frac{1}{A_{OL}}\right) = C_I \times V_O(nT-T) \times \left(1 + \frac{1}{A_{OL}}\right) + C_S \times \left[V_I(nT-T) - \frac{V_O(nT)}{A_{OL}}\right] \tag{4.3}$$

If a discrete time representation is used, solving for $V_O[n]$ gives:

$$V_O[n] = \frac{A_{OL}+1}{A_{OL} + \frac{C_S}{C_I} + 1} \times V_O[n-1] + \frac{C_S}{C_I} \times \frac{A_{OL}}{A_{OL} + \frac{C_S}{C_I} + 1} \times V_I[n-1] \tag{4.4}$$

4.2.1.2 Modelling the effects of nonlinearities

Nonlinearity can be modelled in a similar fashion. The three major sources of nonlinearity in a switched capacitor integrator are signal-dependent charge injection from the switches, nonlinearity in the capacitors, and nonlinearity in the opamp gain function. To illustrate techniques for modelling nonlinearity, capacitor nonlinearity will be considered and can be modelled by using a Taylor Series Expansion about a nominal operating point.

$$C(v) = C_0 \times (1 + \alpha_1 v + \alpha_2 v^2 + \ldots) \tag{4.5}$$

For simplicity, assume only the first term of the Taylor Series is retained and that the nominal operating point is v=0. The charge conservation equation can now be written:

$$C_{I0}(1 + \alpha_1 V_O[n]) V_O[n] = C_{I0}(1 + \alpha_1 V_O[n-1]) V_O[n-1] + C_{S0}(1 + \alpha_1 V_I[n-1]) V_I[n-1] \tag{4.6}$$

Solving for $V_O[n]$ gives:

$$V_O[n] + \alpha_1 (V_O[n])^2 = V_O[n-1] + \alpha_1 (V_O[n-1])^2 + \frac{C_S}{C_I}(V_I[n-1] + \alpha_1 (V_I[n-1])^2) \tag{4.7}$$

The difference equation is no longer linear and could be solved by iteratively choosing a value for $V_O[n]$ until the solution converges. Since the system is oversampled, efforts must be made to simplify the evaluation so that compute times are not excessive. The initial choice in the iteration is obtained by assuming that $\alpha_1(V_O[n])^2$ is 0. Since α_1 is usually much less than 1, this term will be close to 0. A first-order simplification is to calculate $V_O[n]$ under these assumptions, then subtract the $\alpha_1(V_O[n])^2$ term out to obtain an estimate for $V_O[n]$. This will allow the integration loop to consist of in-line code without the need for extra iterations. Experience has shown that this simplification does not adversely affect the simulation results.

4.2.1.3 Combining Finite Gain and Capacitor Nonlinearity

The effects of finite gain and nonlinearity can be combined. To simplify the equations, the factors K and M are defined and the voltage notations simplified.

$$K = 1 + \frac{1}{A_{OL}} \qquad M = 1 + \frac{1}{A_{OL}} + \frac{C_S}{C_L}\frac{1}{A_{OL}} \tag{4.8}$$

$$V_O = V_O[n] \qquad V_{OI} = V_O[n-1] \qquad V_I = V_I[n-1] \tag{4.9}$$

If capacitor nonlinearity and finite opamp gain are both accounted for, the charge conservation equation is written as in Equation 4.10.

$$C_{I0}(1 + \alpha_1 K V_O) K V_O = C_{I0}(1 + \alpha_1 K V_{OI}) K V_{OI} + C_{S0}\left(1 + \alpha_1\left(V_I - \frac{V_O}{A_{OL}}\right)\right)\left(V_I - \frac{V_O}{A_{OL}}\right) \tag{4.10}$$

Solving for V_O gives a rather complicated equation.

$$V_O = \frac{K}{M}V_{OI} + \alpha_1\frac{K^2}{M}V_{OI}^2 + \frac{C_S}{C_I}\frac{1}{M}V_I + \frac{C_S\alpha_1}{C_I\,M}V_I^2 - 2\alpha_1\frac{C_S}{C_I}\frac{1}{MA_{OL}}V_IV_O + \frac{1}{A_{OL}^2M}\left(\frac{C_S}{C_I}\alpha_1 - \alpha_1K^2\right)V_O^2 \tag{4.11}$$

Since V_O appears on both sides of the equation, some changes must be made to implement Equation 4.11 in a simulation model. To avoid iterations in solving the equation, once again it is assumed that α_1 is much less than 1 and that V_O can be approximated by assuming it is 0 on the right hand side. Implementing this in a simulation then requires the evaluation of the following equations:

$$\begin{aligned} \text{tempout} &= f_1 V_{OI} + f_2 V_{OI}^2 + f_3 V_I + f_4 V_I^2 \\ \text{idealout} &= f_5 V_I \text{tempout} + f_6 \text{tempout}^2 \end{aligned} \quad (4.12)$$

The constant factors f can be evaluated from known circuit data and are expressed as follows:

$$f_1 = \frac{A_{OL} + 1}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.13)$$

$$f_2 = \frac{\alpha_1 (A_{OL} + 1) \left(1 + \frac{1}{A_{OL}}\right)^2}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.14)$$

$$f_3 = \frac{\alpha_1 \frac{C_S}{C_I} A_{OL}}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.15)$$

$$f_4 = \frac{A_{OL} \frac{C_S}{C_I}}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.16)$$

$$f_5 = -\frac{2\alpha_1 \frac{C_S}{C_I}}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.17)$$

$$f_6 = \frac{\frac{\alpha_1 \frac{C_S}{C_I}}{A_{OL}} - \alpha_1 A_{OL} \left(1 + \frac{1}{A_{OL}}\right)^2}{A_{OL} + 1 + \frac{C_S}{C_I}} \quad (4.18)$$

It has been shown that simple models like these are sufficient to study behavior of most switched-capacitor oversampling modulators [79], [44], [40], [42], [46]. These equations can easily be coded in a variety of programming and simulation environments.

4.2.1.4 Parasitic Input Capacitance of the Opamp

The parasitic capacitance at the input of the opamp can be quite large. In some opamps, the parasitic can be of the same magnitude as C_S or C_I. These effects have been analyzed [80],[81] and can be incorporated in the simulation. Since an opamp has finite gain, charge redistribution will leave charge on both the sampling capacitor and the parasitic at the input of the opamp. This diminishes the overall apparent gain of a switched capacitor integrator stage.

4.2.1.5 Integrator Settling

Settling time effects of the opamps should also be carefully modelled since they can provide a signal dependent offset when the settling is slew limited. It has been shown that if a single pole model is used for settling the performance is hardly degraded even if the time constant is quite small [79]. A simple settling model can be implemented by determining the ideal output, and then applying rules to determine how well the opamp settles.

Rather than using a single pole model, the settling period can be divided into a slewing region and a linear settling region. One model for slew rate limiting has been proposed for both the one pole and two pole systems with slew limiting [81]. A similar model has been used in simulating oversampling A/D converters [44]. This simplified model does not account for the initial capacitive feedthrough in the integrator, but does provide a simple model for simulation. In this model, there are 3 regions of operation, as shown in Figure 4.3. If the integrator were ideal, the output would change by an amount Vo. It is assumed that the time constant τ of the single pole system is less than the sampling period T_S and the integrator has a maximum slew rate S_R. When the output step is small, the integrator should settle with a one pole response. When slew limiting occurs, there will be a period of slewing until the output is close to the final value, and then linear settling. In this model, the cutoff point between these regions is chosen to be at the time τ. When there is slewing in the model, the output slews to $1/(\tau S_R)$ of the final value and then settles linearly in the remaining time. If the output is fully slew limited, then the output can only change by the amount $T_S S_R$ regardless of Vo.

This model can be implemented together with the other models for nonlinearity and finite opamp gain. The quantity Vo is calculated based on the equations developed previously for the integrator. The region of operation is then determined based on the inequalities given in Figure 4.3, and the integrator output is calculated based on the output value, also given in the figure. The same definitions could be made for a 2 pole model of settling.

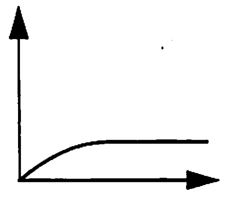
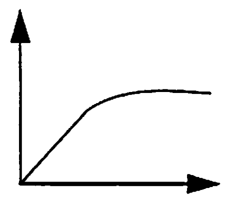
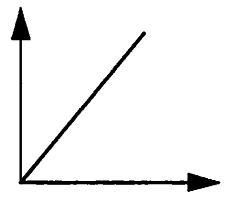
Time Domain View	Region of Operation	Final Simulation Output Value
	linear settling $V_O \le \tau S_R$	$V_O(1 - \exp(-\frac{T_S}{\tau}))$
	Combination slewing with some linear settling $\tau S_R < V_O \le (T_S + \tau) S_R$	$V_O - \tau S_R \exp\left(\frac{V_O}{\tau S_R} - 1 - \frac{T_S}{\tau}\right)$
	Slew limited $V_O > T_S + \tau$	$T_S S_R$

Figure 4.3 Regions of operation for integrator settling.

4.2.1.6 Integrator Clipping

Switched-capacitor integrators are limited in swing. Implementing this in simulation requires only a minor addition. The output of the integrator must be monitored and a hard limit applied when the output goes beyond a fixed range. This should be done after the correction for settling effects. Extra features could be added to model the soft saturation effects normally found in opamps.

4.2.1.7 Sampling Jitter

Clock jitter affects the sampling process causing an effect similar to gain errors. In simulation, jitter is easily modelled by adding random jitter to the input sinusoid generator. Results from experimental circuits showed that timing jitter leads to whitened noise in the output spectrum of the modulator which significantly degrades performance. This approach has been applied to determine the effects of jitter in audio applications [82], where it was shown that with respect to clock jitter, oversampling A/D converters are comparable to Nyquist-rate converters.

4.2.1.8 Models for Switched-Capacitor Δ–Σ Modulators

The integrator is the key element for simulation. To create a simulation model, the function accumulate must be defined based on the material described in the previous sections. Additionally, an initialization section is needed to store values for the capacitor sizes, the opamp gain, the capacitor nonlinearity, opamp slew rate, and swing limits. After this has been done, the code given in Figure 4.1 is sufficient to implement a basic simulation model for a first order Δ–Σ modulator. Higher-order models can be created by adding more integrators or more modulator loops. Other effects like comparator offsets and other noise sources can be added in the code. Noise generators are discussed in the next sections.

4.2.2 Modelling Continuous Time Integrators for Oversampling A/D Converters

The integrator used in oversampling A/D converters does not need to be implemented using switched-capacitor technology. Recently, there has been development of an A/D converter in using superconducting circuits [83] and also using mechanical Δ–Σ modulators for accelerometer applications [84]. These types of circuits require continuous time integrator models. Candy has pointed out that in highly oversampled systems, the input cannot change very rapidly so a discrete time approximation should be adequate for analysis and simulation [27]. Algorithms based on Newton-Raphson iterations, such as the method in SPICE, are costly in CPU time and should be avoided.

4.2.2.1 Digital System Simulation of the Integrator

For the accelerometer simulation, the basic difference equation model was modified so that the entire modulator was simulated at a high sampling rate. The system is already oversampled, but the integrator model is allowed to run at an even higher sampling rate, allowing the integrator state to evolve as the input changes. The integrator model can be developed from a digital simulation of a continuous time integrator. If the integrator model runs at 32 times the sampling rate, 32 times more arithmetic must be performed than the basic difference equation model. This presents some problems, since the high effective oversampling ratio forces long simulation times. In addition to this overhead, it may also be difficult to establish the oversampling factor that will allow adequate modelling of integrator dynamics.

4.2.2.2 Solution to the Differential Equations

If the input waveform is assumed to be either DC or a sinusoid and if noise effects can be ignored, it is possible to solve the differential equations governing the state trajectory for an integrator. The simulation then consists of using the initial conditions at the start of the sampling period and then calculating the state trajectory for the circuit based on the input function and the initial conditions. The only point that needs to be calculated in the trajectory is the value at the end of the sampling period. The state of the circuit at this point becomes the initial condition for the next sampling period. Since the equations are deterministic, the simulation does will be similar to the difference equation model in execution time.

Since the trajectory is known for all time points within the sampling interval, it becomes possible to study the effects of the jitter in the clocks used to control the feedback voltages in a modulator. This can be done by including a small perturbation in the time value assigned to the clock edge instead of always calculating the trajectory at a fixed interval. These problems need to be studied in the superconducting modulator, since it can be difficult to control the precision of the clocking logic.

4.2.3 Modelling Noise Sources

Even when modelling finite gain and nonidealities, the difference equation model predicts a higher SNR than observed from actual circuits. For a second order Δ–Σ modulator, the model predicts that an oversampling ratio of about 150 is required to achieve 16 bit performance. Circuit data acquired during this project and from other reported circuits [85] indicate that an oversampling ratio closer to 250 is required. We have found that by adding models for 1/f noise and white noise, agreement between the circuits and the simulation model can be achieved.

4.2.3.1 Generating White Noise

When a stream of uncorrelated random variables with variance σ^2 is generated, they will have the autocorrelation function given by Equation 4.19.

$$R[n] = \sigma^2\delta[n] \tag{4.19}$$

The discrete Fourier transform of the autocorrelation function gives the power spectral density (PSD).

$$S[f] = \sigma^2 \tag{4.20}$$

The total noise power of the signal is determined by integrating over the frequency range of interest. The actual distribution used to generate the numbers does not matter, but Gaussian distributions tend to be used since they are fully characterized by first and second order statistics.

For switched-capacitor integrators, white-noise generators are used to model the noise in the switches. As an example, consider the switch noise in a sampling switch connected to a capacitor. For simplicity, assume this total noise is kT/C and the sampling frequency is f_S. For a proper simulation in the digital domain, the noise power of the continuous time process must be set equal to the discrete time process, so the proper value of variance to use in the simulation is kT/C. Note that this parameter is independent of the sampling frequency.

Since computers work with deterministic algorithms, it can be difficult to generate a random number sequence. Some functions use Linear congruential random number generators which are based on modulo arithmetic and will exhibit periodicity after a certain number of points have been generated. Since many points need to be generated for simulation of oversampling A/D converters, some attention must be given to the particular random number generator being used. The **rand()** function supplied with most C compilers may not be able to generate long enough sequences without correlation. One solution is to use a different function or to use modifications that improve the randomness [86]. Other more involved methods are also available.

4.2.3.2 Generating 1/f Noise

Although 1/f noise is found throughout the natural world, algorithms for 1/f noise generation are not as pervasive [87], [88]. Modelling the noise near DC for oversampling A/D converters is important since this noise determines the overall baseband noise floor. A promising approach for 1/f noise generation is to use the summation of Lorentzian spectra. This approach has been used in instrumentation to generate continuous time 1/f noise over a specified range of frequencies. It has been shown that a constant distribution of 1.4 poles per decade gives a 1/f spectrum with less than 1% error [89]. By changing the pole distribution, $1/f^{\gamma}$ noise can be modelled. This model allows simulation of 1/f noise in over a frequency region which can be defined to include the areas near DC.

For discrete time, the spectral density can be approximated using sampling concepts. Equation 4.21 gives the power spectral density for summation of Lorentzian spectra. S(f) only has an approximate 1/f spectra over a range defined by the values of ϕ_n.

$$S(f) = \kappa \sum_{n=1}^{N} \frac{\varphi_n}{f^2 + \varphi_n^2} \approx \frac{\kappa}{f} \text{ for } \varphi_1 < f < \varphi_N \tag{4.21}$$

In continuous time, the Lorentzian spectrum can be implemented by passing white noise through a one-pole filter. The magnitude response H and the power spectral density for white noise filtered by H is given in Equation 4.22.

$$H(j\omega) = \frac{\sqrt{p}}{j\omega + p} \qquad S(f) = |H(jf)|^2\sigma^2 = \frac{p\sigma^2}{f^2 + p^2} \tag{4.22}$$

The DC gain for the filter must be adjusted to give the Lorentzian spectrum. We can obtain a discrete time approximation by using the s-to-z mapping of Equation 4.23 where T_S is the sampling period.

$$s = \frac{1 - z^{-1}}{T_S} \tag{4.23}$$

This represents backward-Euler integration and provides adequate results at the low frequencies of interest. If higher frequencies need to be considered, the bilinear s-to-z mapping could be used. The approximate magnitude response H' for the filter H is given in Equation 4.24.

$$H'(z) = \frac{\frac{T_S\sqrt{p}}{1 + T_S p}}{1 - \frac{1}{1 + T_S p} z^{-1}} = \frac{\frac{\sqrt{T_S}\sqrt{x}}{1 + x}}{1 - \frac{1}{1 + x} z^{-1}} \text{ where } x = T_S\, p \tag{4.24}$$

The corresponding power spectral density when white noise is filtered is given in Equation 4.25.

$$S'(f) = \frac{T_S\left(\frac{\sqrt{x}}{1 + x}\right)^2 \sigma^2}{1 - 2\left(\frac{1}{1 + x}\right)\cos\left(\frac{2\pi f}{f_S}\right) + \left(\frac{1}{1 + x}\right)^2} \tag{4.25}$$

Substituting this result into Equation 4.21 gives Equation 4.26 which is the power spectral density of the proposed 1/f noise generator.

$$S'(f) = \sum_{n=1}^{N} \frac{\left(\frac{x}{1 + x}\right)^2}{1 - 2\left(\frac{1}{1 + x}\right)\cos\left(\frac{2\pi f}{f_S}\right) + \left(\frac{1}{1 + x}\right)^2} \sigma^2 T_S \tag{4.26}$$

In order to match the magnitude of the spectral density S'(f) with the continuous time counterpart, the input white noise generators must have the same noise power. As argued in the previous section, the variance or noise power in a certain bandwidth must be set equal. In this case, the input white noise generators in the continuous time model of Equation 4.21 have spectral density κ. In the bandwidth f_S, they will have noise power κf_S. Thus in the discrete time simulation, the proper value to use for κT_S is the noise power of the continuous time white noise generators, which in this case is $\kappa f_S T_S = \kappa$, a quantity that is independent of frequency.

The noise generator can be implemented as shown in Figure 4.4. The frequency response of

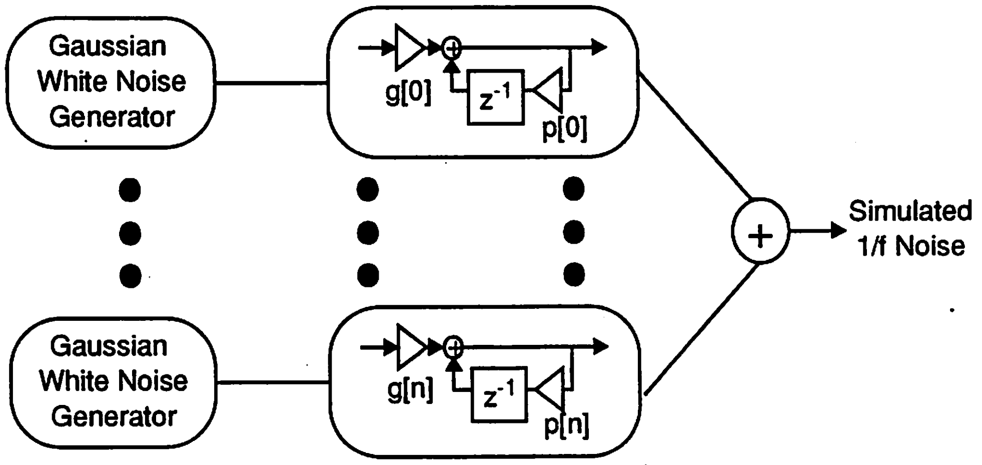

Gaussian White Noise Generator
g[0]
z^{-1}
p[0]
Gaussian White Noise Generator
g[n]
z^{-1}
p[n]
+
Simulated 1/f Noise

Figure 4.4 Summation of approximate Lorentzian spectra to obtain a 1/f noise model.

one implementation is shown in Figure 4.5 along with the 12 approximate pole values used in the

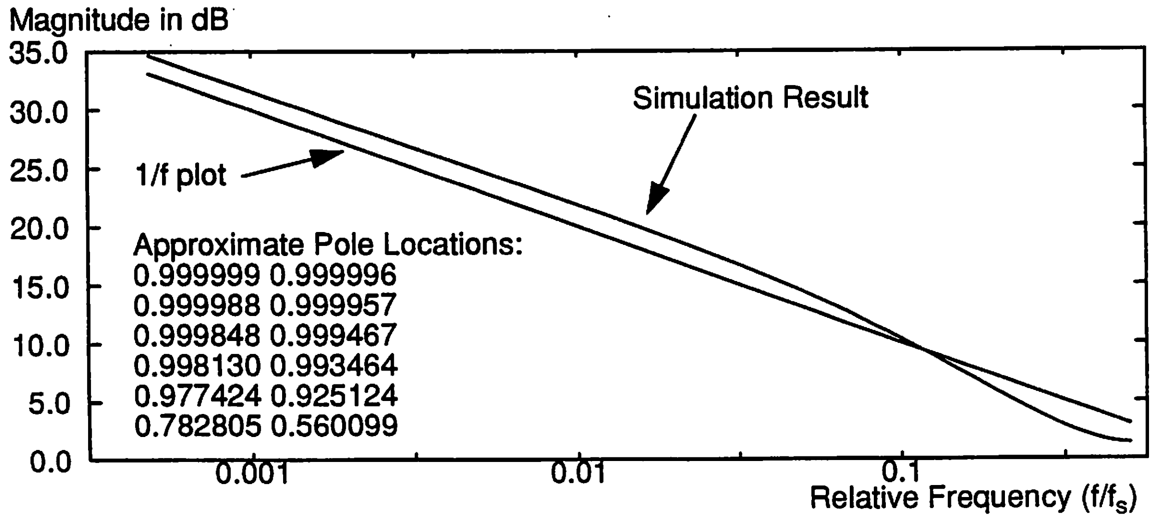


Figure 4.5 Frequency response of a 1/f noise generator.

generator model. In addition, a plot of the deviation from a true 1/f response shown in Figure 4.6

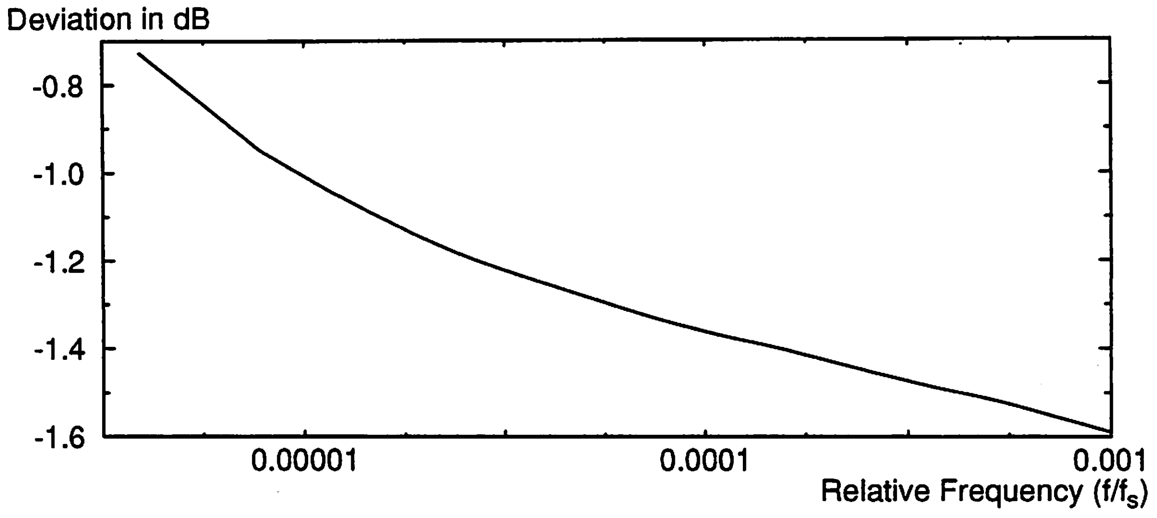


Figure 4.6 Deviation between the true 1/f response and the simulation result.

which shows little variation over the low frequency range of interest. The downward trend indicates that the simulation model provides noise with a $1/f^{\nu}$ characteristic where ν is slightly less than 1. There is little ripple since a high pole density was used across the interval.

To set the value of variance for the input white noise generator, consider the example of a MOS transistor. The equivalent input-referred noise can be modelled as in Equation 4.27.

$$v_{eq}^2 = \frac{K_f}{WLC_{ox}f} \tag{4.27}$$

The parameter $\sigma^2 T_S$ should be set equivalent to the factor $K_f/(WLC_{ox})$ in magnitude. For the simulation, the output of the 1/f noise generator is placed so that it adds to the input signal of the MOS transistor. It is interesting to note that a set of samples from the 1/f noise generator does not give any information about the sampling rate. In fact, a single set of samples can be used as 1/f noise regardless of the sample rate. This scalability is a characteristic of 1/f noise. It will look the same no matter what the resolution is used. This phenomenon has been exploited in the generation of fractal images.

Several theories have been advanced for the cause of 1/f noise in MOSFETs. A widely accepted view attributes it to generation-recombination noise at the silicon-oxide interface [90] where the traps are characterized by a distribution of time constants. The model simulates this physical mechanism since a distribution of time constants gives rise to something similar to a summation of Lorentzian spectra. Analysis using these assumptions gives rise to Equation 4.27 for a MOSFET which shows that 1/f noise is inversely proportional to gate capacitance.

4.3 Simulation and Analysis of Digital Filters

Computer simulation is a powerful tool and is an integral part of design management. Rather than overspecifying design parameters, system issues can be studied to refine the specifications. In addition, interactions between system components are more easily studied in this type of simulation environment.

4.3.1 PTOLEMY

The PTOLEMY simulation system provides this capability for our design system [91]. PTOLEMY allows designers to assemble flowgraphs of block-level functions, then to simulate the flowgraphs. The block-level models for the design system were implemented using synchronous data flow (SDF) [92]. Whenever a Synchronous Data Flow block is invoked or fired, it consumes or processes a number of tokens and produces a number of tokens, and the number of these tokens is known before execution. This type of description is adequate for describing most synchronous systems, including multirate systems. While all the cells related to the design system were implemented in the SDF domain, PTOLEMY also allows mixing of simulation schemes. For example, discrete event simulation models can be mixed with synchronous data flow (SDF) or even timestep driven simulation models found in most circuit simulators.

The models developed for PTOLEMY generally fall into the classes of FIR filters, Biquadratic filters, and Lattice Wave Digital Filters. In PTOLEMY terminology, the models are available to users as stars. With a graphical interface, users connect the stars into flowgraphs that perform various signal processing applications.

Since PTOLEMY is written in C++, fixed point models can be implemented using a fixed point data class. The particular implementation of the fixed point class limits values to be in the range (1, -1]. An initialization statement can be used so that fixed point arithmetic can be either saturating or wrap-around. While the fixed point calculations do simulate the actual operations of a chip, they cannot exactly replicate the round off errors, since the ordering of operations may be compiler dependent. Experiments are being performed to find the best means of incorporating fixed point simulations in PTOLEMY.

In addition to simulation models, a translator is provided to generate flowgraphs for filtering systems developed in the design system. With the simulation power in PTOLEMY, complete data and signal acquisition systems can be simulated. Effects like finite wordlength effects can be studied in depth on actual data sequences. This makes the study of the interactions between the different components of a system much easier.

4.3.2 Stars for Oversampling A/D Simulation

PTOLEMY supplies many stars that can be used in simulating oversampling A/D converters. However, the granularity of the stars is an issue, since oversampling ratios are large and many output sample points are desired. Rather than using a large collection of primitive stars, more complex stars were written to speed up simulations. An example of this trade-off is shown in the next section.

The basic modulator models described in the previous sections of this chapter were coded into stars for first, second, and third order modulators. The 1/f noise model is not included in the modulator, but in a separate star. Since 1/f noise is actually generated inside the modulator, the magnitude used in simulation must be modified to reflect the gain transfer function from the actual point where the noise is generated. The 1/f noise generator is included in a star called Vgnsin.

When simulating oversampling A/D converters, high resolutions in the simulation force users to take precautions to maintain resolution. This is true in the case of sinusoid generation. The normal SDF sinusoid generator in PTOLEMY creates a sinusoid by adding a fixed fraction of the value π at every time step, and then taking the sine of that value. Since the value of π is quantized, roundoff errors accumulate at each time step limiting SNR to about 85 dB. This effect smears the purity of the sinusoid, distorting the spectral line so it has a finite width. A new star was created to fix this problem. Rather than adding a fraction of π, an integer counter value is multiplied by the rounded value of π to create the argument for the sine function at each time step allowing high precision simulation.

For filters, PTOLEMY SDF models provide support for FIR filters and filters based on direct form II biquads. This library was augmented by stars for implementing CIC filters, and polyphase N-th band filters. The CIC filter star is the direct mapping of a set of integrators and differentiators. Since all arithmetic is performed using integers, finite wordlength effects are correctly modelled for bit-parallel implementations. Different wordlengths can be used in the integrators and the differentiators.

A generalized model was created for multirate polyphase N-th band filters. The basic structure of a polyphase filter is illustrated in Figure 4.7. Two variations can be modelled, one with external

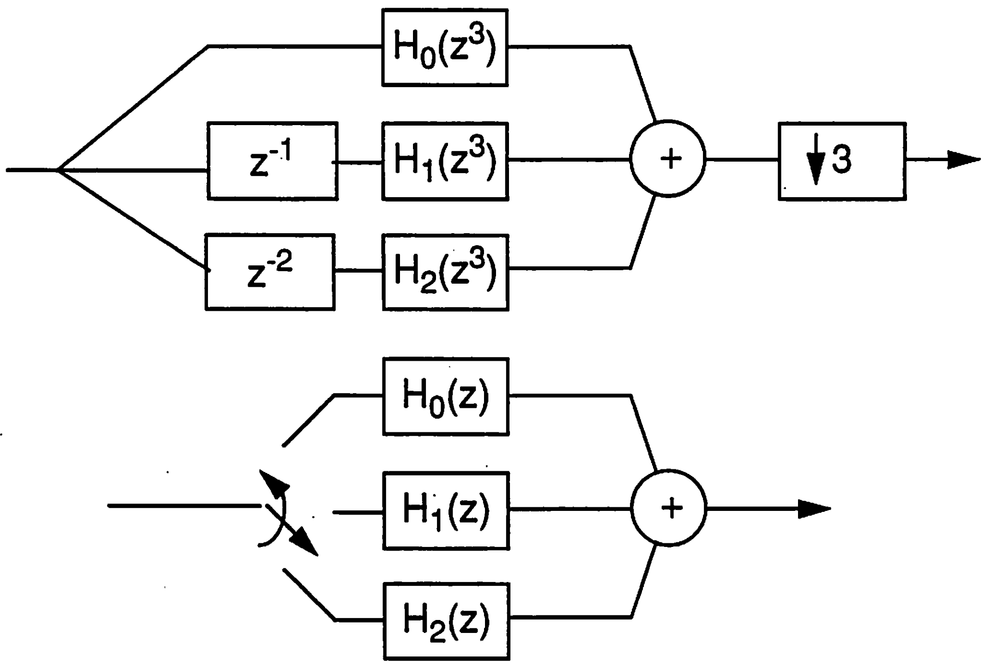


Figure 4.7 Block diagrams of a polyphase 3rd band filter using external decimation and the counter-clockwise commutation

decimation and one with decimation incorporated in the filter. Using the commutator model, only one branch per input sample must be evaluated. In the external decimation model, complications arise, since each state variable in the branch filters must have a depth of N to accommodate the delays, and all branches must be evaluated for each input sample.

Each of the branches consists of an all-pass filter which can be implemented as cascaded biquads or lattice wave digital filters. Both filter models were implemented as separate stars both set up for the decimation case. Interpolation can be handled in a similar fashion, with the commutator on the output. The basic algorithm is centered around evaluation of a single branch filter. The star takes N tokens from the input. The first token is used as input to the (N-1)th branch, the last token as input for the zeroth branch. The branches are then evaluated in order and the branch outputs are summed. The stars take as parameters the decimation ratio and an array of coefficients formatted according to the policy described in Appendix D. As special cases, the filters can be used to implement a standard IIR filter, either biquadratic or lattice wave digital. Special code is needed in the lattice wave digital case, since 2 branches are needed while for biquads, only a single branch is needed.

For spectral analysis, PTOLEMY provides an FFT and the option of generating power spectral density. It is desirable to use DFT magnitude and sinusoidal curve fitting programs to perform analysis for A/D converters to obtain signal to noise ratios. Rather than rewriting the code, PTOLEMY stars make **system()** calls to these programs and some scripts for post-processing and graph preparation. A star for the Minimum Sinusoidal Error method of determining SNR [71] was integrated as well as an FFT post processor that can estimate SNR based on the magnitude plot of an FFT [78].

4.3.3 Simulation Example

Figure 4.8 shows a flowgraph for first order Δ–Σ modulator flowgraph designed using the PTOLEMY Interactive Graphical Interface (PIGI). This particular model uses PTOLEMY primi-

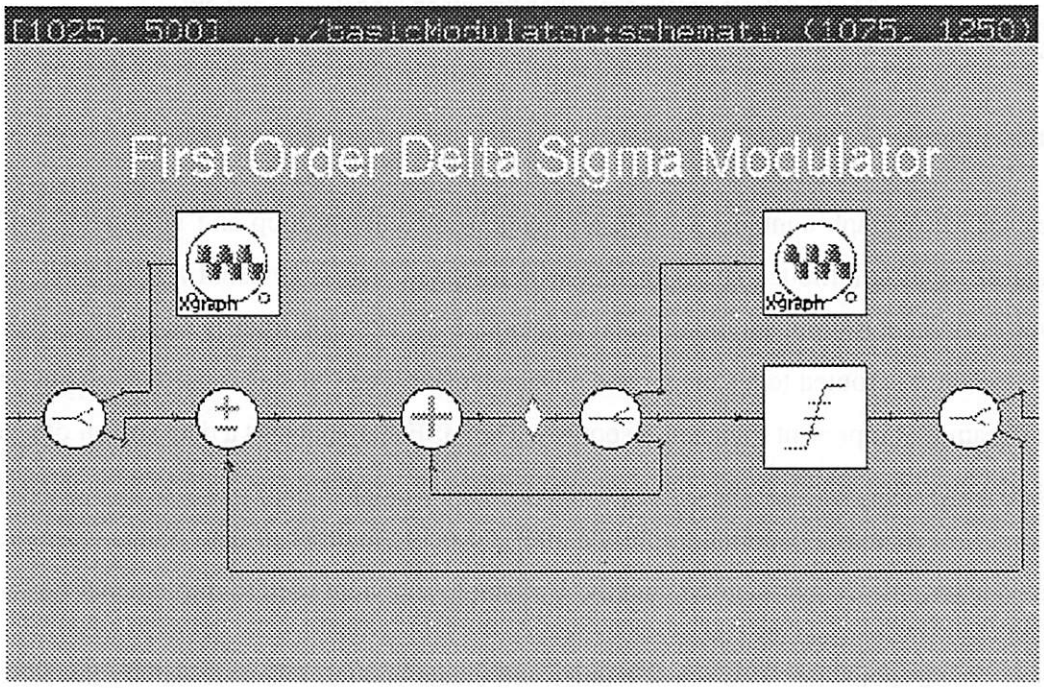

[1025, 500] .../basicModulator:schematic (1075, 1250)
First Order Delta Sigma Modulator
Xgraph
Xgraph

Figure 4.8 Flowgraph of a first order Δ–Σ modulator implemented in PTOLEMY.

tive functions to implement the modulator. Simulation overhead makes it advisable to encapsulate a model like this one as a single piece of code. This has been done in the flowgraph shown in Figure 4.9 which shows a complete oversampling A/D converter simulation with a single stage of

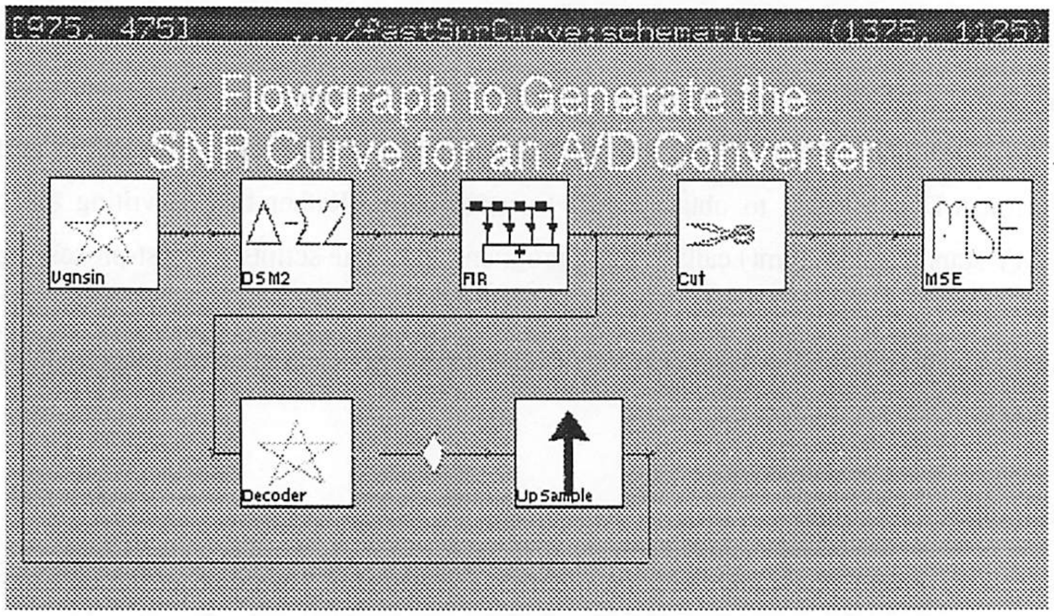

[975, 475] .../FastSnrCurve:schematic (1375, 1125)
Flowgraph to Generate the SNR Curve for an A/D Converter
Vgnsin
DSM2
FIR
Cut
MSE
Decoder
UpSample

Figure 4.9 Complete simulation flowgraph for determining SNR.

decimation filtering.

The basic flowgraph contains 8 basic elements in a feedback loop and will create a plot of SNR against input amplitude. The Vgnsin star is used to create a sinusoid and additive 1/f and white noise. A second order modulator and a decimate by 128 FIR filter is added after that. The output of the filter is routed to a Cut star and a Decoder star. The Cut star is used to cut out parts of the data stream that represent transient responses. The MSE star analyzes a data stream of a certain length and calculates SNR. Since the system operates in synchronous dataflow, a method was needed to allow changing of input amplitude after a specific number of data points was obtained. The Decoder star, the delay diamond, and the UpSample star were used to implement one scheme.

Under synchronous dataflow, the scheduler will create enough tokens to satisfy the MSE star. For example, suppose that 2 sets of 32 points are anticipated at the MSE star, representing 2 different input amplitudes. The parameters in the MSE star are set so that on each iteration, 32 tokens are consumed. The Cut star can be set to discard the first 32 samples of 64 total that it consumes. The Decode star is then used to detect when 64 total samples have been produced, and produces a nonzero output on the cycle when this occurs. This value is delayed by 1 cycle using the delay diamond, and then upsampled by a factor of 128 with zero insertion. A factor of 128 is used to match the decimation ratio in the filter which keeps the sample rate consistent in the feedback loop. When the Vgnsin star detects a non-zero input, it moves to the next amplitude to be generated. When the flowgraph is run, it will need 2 iterations to complete the required calculations. The result of the simulation will be a plot of SNR vs. input amplitude, as shown in Figure 4.10.

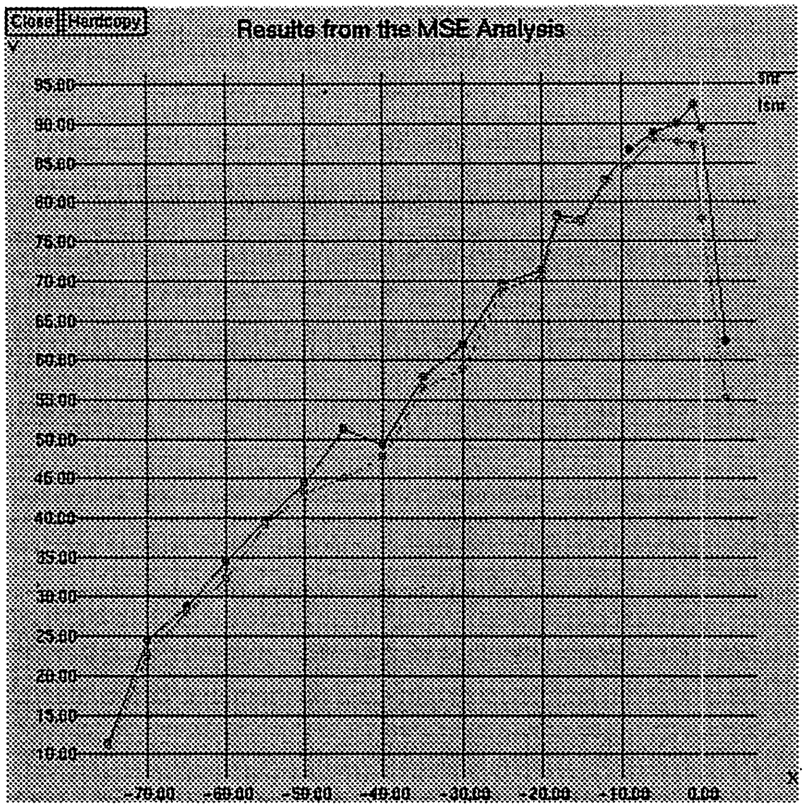


Figure 4.10 Output from the flowgraph shown in Figure 4.9

4.4 System Simulation Strategy

Simulation is needed for 3 levels of design abstraction: algorithm, architecture, and circuits. Algorithmic verification is performed using the models in PTOLEMY. Floating point coefficients and data values are used to provide feedback on system design. Algorithmic changes can be easily studied by using the design system to create filter coefficients and then changing stars accordingly.

For architecture simulation, bit-true models are needed to investigate finite wordlength effects. Some finite wordlength models exist in PTOLEMY, but they are limited. While finite wordlength effects are important, ordering of operations can affect the accumulation of roundoff errors. These effects are highly architecture dependent, requiring a specific model for each architecture. Models for FIR filters are easy to generate, but for IIR filters, the task can be more challenging. Design systems like HYPER and C-to-silicon provide these features. The C-to-silicon tools generate C language modules that can be used in system simulations using finite wordlength models. The architecture level is the lowest level at which full system simulations are currently feasible.

At the circuit level, simulations are made on models derived from layout extractions. Transient circuit simulations are used to verify connectivity and functionality of analog circuits, while gate level simulations are used for digital circuits. To verify system functionality, the analog portion is modelled by the difference equation model. A bit stream is generated and used as input for the gate level digital simulation. Since simulation takes quite a bit of time, full simulation runs are not performed. A number of output data samples are generated and then compared to the results of the architecture level simulations. This is repeated for a few different modulator input conditions. A match between the circuit simulations and architecture simulations verifies the design.

4.5 Summary

Several methods have been presented for developing models for noise-shaping modulators. The techniques presented here are meant to be a guide in developing new models. The key in simulation is not to force a given model to fit physical phenomena, but to abstract the physical phenomena using modelling mechanisms that capture the relevant system characteristics. This method allows large reductions in simulation time while still providing accurate simulation necessary to guide system development. As design complexity increases, designers will need to use these higher-level simulation techniques to shorten design times and to provide better design exploration. In addition, digital filtering simulation in PTOLEMY and system level simulation for verification were discussed.

CHAPTER 5

Coefficient Design Techniques

5.1 Overview

In the first phase of detailed design, coefficient design is performed based on algorithmic information from high-level design. This chapter provides an overview of the coefficient design tools used in the design system. Since most of the coefficient design process is hidden by the framework, the various design algorithms are discussed, highlighting various strengths and weaknesses.

5.2 Tools for Filter Design

As mentioned in chapter 2, there are several filtering algorithms, including FIR, biquadratic IIR, polyphase, and lattice wave digital filters. Each of the these has advantages in certain applications. In order to study the effects of choosing a certain filter type, design aids are needed so that filter coefficients can be obtained, and various properties can be studied by simulation or analysis. This section describes various CAD tools that were incorporated into the design system for analysis and implementation of digital filters for multirate applications.

5.2.1 Coefficient Design

Coefficient design is the process of finding a set of coefficients for a given filtering algorithm that will meet a set of frequency domain requirements and is sometimes called the filter approximation problem. Filter design programs are based on 2 different approaches, one using approximation theory, while the other approach uses numerical optimization techniques to force a transfer function to approximate a given response. Classical methods will first be discussed, followed by numerical optimization methods.

5.2.2 Classical Design Methods

When approximation techniques are used, the problem is stated in terms of solving Feldtkeller's equation.

$$H(j\omega) = 1 + K(j\omega) \tag{5.1}$$

H(jω) is the desired continuous time frequency response and is a function that approximates 1 in the passband. Thus, K(jω) must then approximate 0 in the passband. Given a tolerance for passband and stopband ripple, approximation techniques are used to find the locations for poles and zeroes of the transfer function. The definition of K(jω) allows the approximation of 0 on an interval and methods are available for solving this problem. Optimality of the approximation is defined by the user and several criteria are used, including maximally flat approximations and Chebyshev or minimax approximations. Classical filter design techniques only allow flat specifications for magnitude deviations in the passbands and stopbands. These methods can been applied to designs for RLC filters, active filters, and discrete time filters. Full discussions of the techniques can be found elsewhere [93] so the details of the design process will not be discussed further.

Since classical techniques are based on the approximation of 0 over a region, they are not easily modified to handle magnitude equalization problems. Instead of approximating 0 over a region, an arbitrary magnitude response must be approximated as is the case when designing multirate filters using the sinc filter approach. Classical methods cannot handle this problem, so other methods are needed to provide magnitude or phase equalization.

Extensive results have been tabulated for filters based on Elliptic, Chebyshev I and II, and Butterworth responses. However, these tables are only able to meet specifications in discrete steps. Coding the design equations in computer programs allows filters to be designed to custom specifications. The program DOREDI uses this approach in the design of cascaded biquadratic IIR filter sections [94]. DOREDI also provides facilities for determining optimal pairing of poles and zeroes, and determining coefficient wordlength.

While filters based on cascades of biquads are flexible and easy to implement, wave digital filters are often favored since they tend to exhibit better numerical properties. There can be less sensitivity of coefficient wordlengths, and the structure minimizes limit cycles. The classical design methods have been adapted to the design of lattice wave digital filters [95]. While design programs based on these equations do exist [96], [97], they were not incorporated in this project. The design equations are simple to use, and the design can be performed using a small programmable calculator.

5.2.3 Remez's Second Algorithm

Several numerical optimization methods are based on the Remez exchange algorithm, which is also known as Remez's second algorithm [98]. This method was applied by McClellan and Parks to the design of equiripple FIR filters [99]. The basic algorithm uses iterative approximations of a transformed magnitude response using a trigonometric polynomial. The polynomials are generated by interpolation using the set of points defined by the extrema points from the previous iteration. In each iteration, the zero crossings of the magnitude response are altered until an equiripple response is obtained. The approximation problem has been elegantly solved since a single polynomial is used and the magnitude deviation can be evaluated in both the passband and stopband using the same function. Since the filter coefficients are symmetric, linear phase is achieved and no phase equalization is necessary. In addition, the problem has been stated in terms of minimizing the maximum ripple rather than approximating 0, so arbitrary ripple specifications can be given as well as approximation of arbitrary magnitude responses. This makes the approximation optimal in the Chebyshev sense and it is also known as the minimax approximation.

The concepts used in the Remez algorithm have been applied to IIR filter design problems and are referred to as modified Remez methods since polynomial approximation techniques are no longer applied. The Iterative Stopband Method is an example of an algorithm that uses a modified Remez algorithm [93]. In this method, the passband is uniquely determined by the placement of the poles of the transfer function. Formulas were developed to design Chebyshev and maximally-flat passbands given the pole locations. Rather than using a polynomial approximation obtained from interpolation, partial derivatives for the transfer function are evaluated and used to guide the placement of the poles. This method can have convergence problems and it only provides for arbitrary ripple specifications. It cannot approximate arbitrary magnitude responses since the passband approximations are still based on classical techniques which seek to approximate 0 in a region.

A similar modified Remez algorithm was used for the design of polyphase IIR N-th band filters [57]. Instead of placing pole or zero locations, the position of the attenuation zeroes of the transfer function are varied. Approximation techniques from all-pass filter theory are used to find the new transfer functions based on the results of a design iteration [100]. Again, partial derivatives are used to drive the design iterations towards convergence. A version of this program was implemented in C during the project.

5.2.4 Least p-th Error Approximation

Remez methods provide solutions that are optimal in the Chebyshev or minimax sense. The Least p-th or Minimum p-th Error criterion provides another definition of optimality [101],[102] and it can be defined as in Equation 5.2.

$$E(c) = \sum_{i=1}^{m} \{ w_i [F(x_i) - F'(x_i, c)] \}^p \tag{5.2}$$

The error criterion E is calculated at m discrete points and the function F'() that minimizes this criterion is considered optimal for a given value of p. F(x) is the desired frequency response at a point x. F'(x, c) is the response of the current filter which is characterized by the set of parameters denoted by c and w_j is a weighting that may vary with each point in the frequency response. When p is equal to 2, the set of parameters that minimizes the error criterion is the weighted least squares solution.

It can be shown that as p approaches infinity, the Least p-th approximation tends to the best minimax approximation [103]. Based on this concept, Deczky wrote a digital filter design program to generate coefficients. The Fletcher-Powell algorithm [104] was used to minimize the error criterion for a fixed value of p. The program assumed that the filter would be implemented using cascaded biquads. Gradients could be calculated as a function of the coefficient set since the form of the transfer function was fixed. The Fletcher-Powell algorithm traces the gradients until an optimum is found. When the convergence was achieved, the value of p could be increased and Fletcher-Powell could be applied again.

The Least p-th filter design program was rewritten in C for this project. This implementation of the program was difficult to work with, and did not always converge to a desirable point. Convergence can be improved by first finding the least squares solution and using that solution as the starting point for subsequent calls of the Fletcher-Powell algorithm. However, the program still has difficulty converging for values of p above 4. This appears to be due to numerical problems.

This design method was favored because it could provide arbitrary magnitude and phase approximation, allowing for magnitude and phase equalization. However, due to the difficulties in use, this method was abandoned in favor of other approaches. It seems more reasonable to implement a least squares design program since it was observed that few benefits were gained from increasing the value of p. The Least p-th error algorithm still finds use in the design of filters and has recently been applied to the design of bandpass Δ–Σ modulators [14]. The source code provided in this project could be modified to handle these applications.

5.2.5 Nonlinear Programming

The problem of magnitude equalization was addressed by using a program based on nonlinear programming techniques. The approximation problem was reformulated to a problem that could be solved through the use of sequential unconstrained optimizations [105]. This program also makes use of the Flecther-Powell algorithm but another formulation of the problem allows better convergence and better results for filter design than the Least p-th error method. The techniques used in the program are fairly complex, so recoding was not performed and the program was left in FORTRAN. This program can only modify an existing cascade biquadratic filter function so another design program must be used in conjunction with this one.

There has been recent work on using nonlinear programming schemes to optimize transfer functions according to the power spectral density of oversampling modulators. Rather than approximating a magnitude response, coefficients are varied to minimize the aliasing of noise in decimation filters. Quadratic programming was used to design optimal FIR filters [106], while filters based on polyphase decompositions of IIR filters are designed using multi-objective optimization [107].

5.3 Coefficient Design

While the general filter design problem has been characterized as "solved", there still exist many practical problems when implementing digital filters. The filter design problem has traditionally been formulated as an approximation problem. Classical methods solve this problem elegantly, but only for the case of constant ripple bandpass filters. For other specifications, approximations are based on numerical optimization with a goal of minimizing an error criterion. Since the value of the minimum is not known, design programs must be used iteratively until design specifications are met. This complicates efforts at design automation. Scripts can be devised to direct iterations, but this was not attempted in this project.

Another problem is the fact that almost all design programs provide floating point coefficients. For VLSI implementation, fixed point arithmetic provides many advantages in lowering area and power consumption. Designing floating point coefficients to just fit the specifications leaves little latitude for coefficient quantization. It is difficult to judge how much extra room to leave in the filter specification for coefficient quantization. This can create a larger design iteration loop, and further complicates design automation.

Once the floating point coefficients have been determined, they need to be quantized to allow finite wordlength implementations. While the choice of filtering algorithm is generally independent of the architecture, the proper type of coefficient quantization technique to apply depends heavily on the implementation of the filter. For bit-serial and microcoded implementations, the multiply operations in the filter can be implemented using shift-add techniques. In this case, minimizing the number of shifts is important. In contrast, implementations that use dedicated multipliers require minimization of the wordlength. In the design system, little support is given for minimizing the wordlengths of coefficients. It was felt that the use of dedicated multipliers should be avoided since they tend to take up considerable area.

To minimize the number of shifts, the coefficients are encoded in canonical sign digit (CSD) notation [108]. The coefficient bits are assigned a value from the set (-1, 0, 1), and it can be shown that a minimal coding exists if certain conditions are satisfied. The optimization process seeks to minimize the number of nonzero bits in the CSD representations of all the coefficients in a filter while still maintaining the specification. For this project, the program CANDI was used for coefficient quantization [109]. CANDI uses information created by the simulation and analysis program DIGEST [110]. Optimization is performed by generating sets of coefficients via searches and then evaluating the sets to see which satisfy the design specification. CANDI was written as an interactive program so the user must control the types of searches that are used. Since DIGEST must be used before CANDI, frequency response information and other analyses can be performed.

In order to automate the coefficient quantization process, CANDI was modified. Only a simple univariate search is performed in the automated mode. This was found to be adequate in most cases since it isn't always necessary to find the optimal solution. When implementing microcoded processors, there are a given number of machine cycles for evaluation of a filter. It was found that the univariate search often provided results that could be implemented in the desired number of machine cycles. Further minimization would only leave idle processor cycles.

Since CANDI searches perturbations to the coefficient set, execution time is combinatorial with respect to the number of coefficients. Extra specification points do not increase the execution time excessively, but high order filters can require considerable amounts of execution time.

Other coefficient quantization programs are available, but were not integrated into this project. For the case of FIR filters, better results were obtained by allocating an extra bit to larger coefficients and using a bivariate local search for the CSD coefficients [66]. This program has been used in the FIRGEN system for implementing FIR filters [70], and is available in the tools distributed with this project. Another approach based on simulated annealing have also been developed [111].

5.4 Coefficient Design for Analog Modulators

As mentioned in Chapter 2, only one topology requires an extensive coefficient design procedure and this topology makes use of higher order loop filter. By changing the placement of the poles and zeroes in this filter, the noise shaping can be controlled. A design methodology for calculating the coefficients has been presented previously [44]. One researcher has reported success in using the Least p-th Error criterion routine for designing these types of modulators [45].

Since the code for the Least p-th Error criterion algorithm was developed in C for the project, it should be possible to adapt it for the design of these higher order modulators.

5.5 Summary

Many tools are available for coefficient design and quantization. A good understanding of the design methods helps users to identify where changes could be made to the design system to increase performance. Only a small sampling of programs were incorporated into the design system, but they represent most of the major techniques in modern filter design.

The designer uses the coefficient design tools after performing high-level design analysis. A common set of design parameters are stored in the design database, so the designer need only specify the proper design program. Coefficient design and quantization can be fully automated, as described in this section. After coefficient design, the designer will be ready for architecture mapping.

CHAPTER 6

Architecture Design

6.1 Architecture Selection

Algorithmic changes can affect the design in different ways. For example, a clever implementation can free up enough cycles to allow time multiplexing when it was initially thought implausible or an algorithmic transformation can reduce power consumption. While the basic signal processing functions remain the same, the architecture can be vastly different. For our design system, we have chosen to support 3 design styles to encourage the exploration of algorithm mapping techniques to different architectures: specialized architectures, general purpose digital signal processors, and high-performance architectures. This chapter discusses and documents some of the digital filter architectures that were developed and investigated during the course of the project. Before discussing the architectures used in this project, a brief discussion is given for some of the other design tools available.

The architecture tools are invoked after coefficient design and quantization. Most of the process is hidden from the designer, but the basic process is to convert the quantized coefficients into parameters required by the architecture templates. As with coefficient design, most of the process is automated. The designer only needs to select the correct design method, and then let the design manager take over.

6.1.1 Bit-Serial Architectures

Bit-serial and digit-serial techniques are attractive for low sampling rate applications. Parallelism is exploited both in execution units and on a time basis. Much work has been performed in this area, and many design systems have been developed. General Electric has made extensive use of bit-serial circuits in development of products based on Δ–Σ technology [112], [113]. The development of these circuits was made possible by the GE Bitserial compiler. The compiler is able to take a C-like program description and map it to a layout of CMOS bit-serial circuits. While bit-serial architectures were not heavily investigated in this project, a simple interface could be written to generate the input code for the compiler based on coefficients generated by the filter design programs.

6.1.2 High-performance Architectures

High-performance architectures implement algorithms where there are relatively few machine cycles per sampling interval. This means that multiplies can't be implemented using a single shift-add unit and must be mapped to either pipelined or parallel units. Design trade-offs must be studied to determine whether full multipliers must be used or if additional hardware units can be allocated so that the algorithm can be scheduled in the appropriate amount of time. These techniques are needed when the oversampling ratio is low and the sample rate is high, which is the situation when using higher order modulators for high-speed oversampling A/D conversion.

These types of filter architectures can be implemented using the HYPER system [2]. HYPER takes a Silage [114] description of the filter and allows the user to examine the design and apply transformations to achieve a better design. Silage is a applicative language developed to facilitate the expression of algorithms meant for IC implementation. Hyper is able to find the minimum bounds on the number of execution units needed to perform an algorithm under a given sampling rate constraint. The HYPER system is currently being expanded to handle a variety of architecture styles. The output of HYPER is an structural description of the processor that can be used as input for the Lager Silicon Assembly System [67]. Code generators have been developed to generate silage descriptions for digital filters to allow the use of HYPER from within the design system.

6.2 Time Multiplexed FIR filter family.

6.2.1 Background

FIR filters for multirate signal processing can be efficiently implemented in hardware. As explained in Chapter 2 for the case of decimation, only the desired output samples need to be calculated and thus only a limited number of state variables need be stored. For a filter of length L and for a decimation ratio of D, only L/D registers are needed. Since the coefficients still need to be stored in a ROM, these filters are often assumed to consume excessive amounts of area.

Several researchers have proposed architectures that perform FIR filtering for cases where L/D is small. One version was based on a single ROM for coefficient storage and L/D parallel datapaths to perform the computation [115]. This architecture can be inefficient in area since it requires L/D parallel accumulator units.

To avoid the coefficient ROM, Candy found a way to calculate the coefficients of a sinc^2 filter with only 4 registers, 4 adders, and some peripheral logic [116]. This structure can be quite efficient in area, but further improvements might be achievable. Since digital circuits tend to be faster than analog circuits in a given technology, it is possible to time multiplex the operations so only a single adder is required. Structures based on this approach were developed for the case where L/D = 2 [61] and L/D = 3 [60]. Both of these approaches made use of counters to generate the coefficient values. In addition, the input was assumed to be a single bit so an AND gate served as a multiplier. Since there is overhead for counter circuits, it isn't clear if Candy's architecture is smaller. However, the time-multiplexed architectures provide more flexibility since they can be modified to implement an arbitrary impulse response and they don't rely on modulo arithmetic.

6.2.2 Overview and Black-Box view

For this project, a more flexible approach was sought. A reusable and parameterizable architecture family was developed based on a single adder and a variable number of registers in the datapath. The block diagram for the filter family is shown in Figure 6.1. A coefficient ROM is used in most cases. If L/D = 2 and a sinc squared filter is implemented or if a simple accumulate and dump filter is desired, counters can be substituted to reduce area and power consumption. The coefficient ROM allows filters to be implemented with decimation ratios that are not powers of 2. Additionally, designs are not restricted to have frequency responses that are powers of the sinc function. It is assumed that the impulse response of the desired FIR filter is symmetric, which

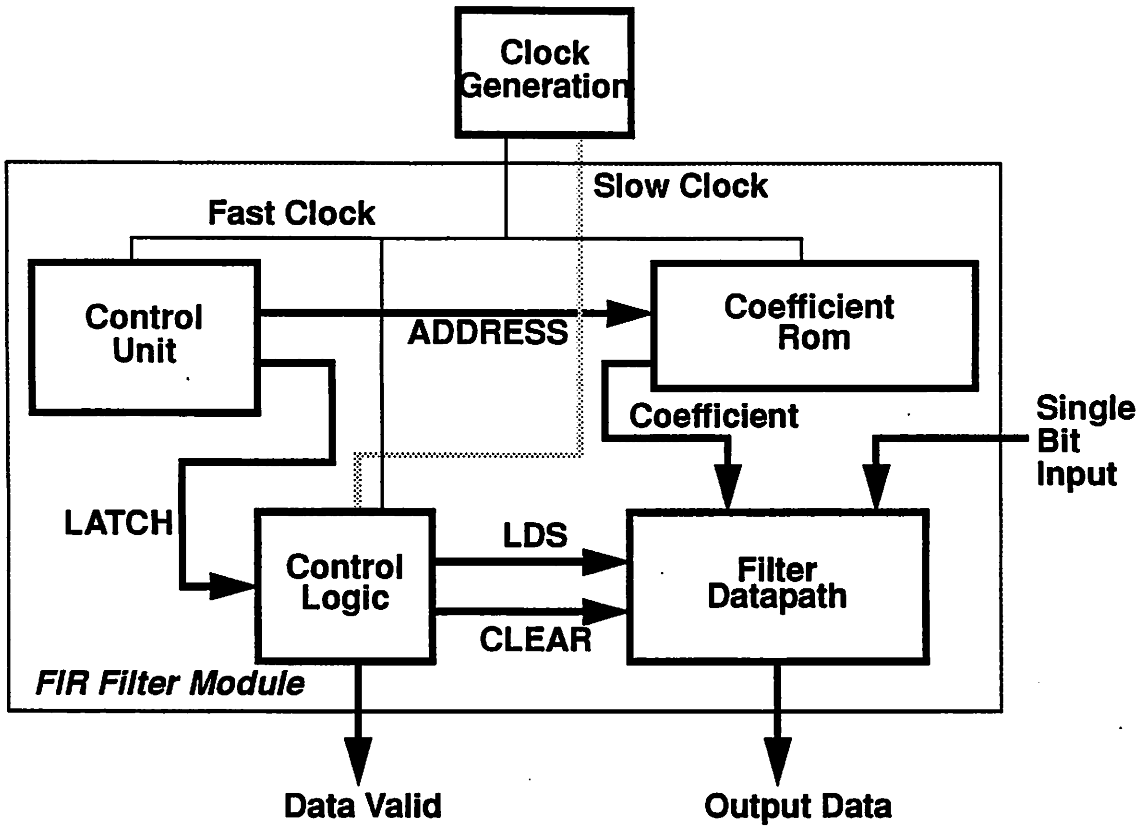


Figure 6.1 Block diagram of the FIR filter family.

allows only half of the response to be stored in the ROM. The filter family currently has structural descriptions for L/D values of 1, 2, and 3.

The filters are designed to handle single bit inputs and to work internally with 2's complement arithmetic. The clock generators are not included in the structure descriptions and must be connected externally. The current implementation is based on 2 phase clocking. Most of the registers are based on master-slave registers and it would not be difficult to translate the design to a single phase clocking scheme. Internally, the filter runs at D times the input rate, so a fast clock running at D times the input rate is needed. A clock running at the input sampling rate is also needed to aid in synchronization of signals.

The basic timing relations for the filter family are shown in Figure 6.2. No input synchronization signal is required for the filter since 1 input is required during each slow clock cycle. The input data is latched on the falling edge of the slow ϕ_2 phase. Output data is generated at 1/D times the input rate so a new output is available every D cycles. The new output sample is valid on the

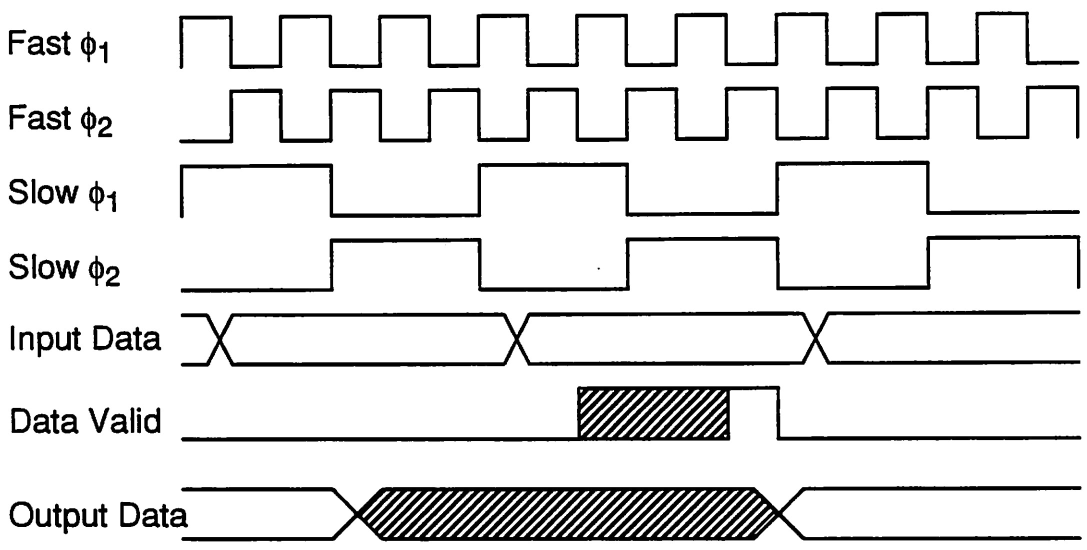


Figure 6.2 Timing relations for the FIR filter family for the case L/D=3.

cycle after the falling edge of the Data Valid signal, and remains valid until 1 slow clock cycle before the next falling edge of Data Valid. The falling edge of the Data Valid signal is periodic and occurs every D slow clock cycles, or about every L fast clock cycles. The rising edge of DataValid can occur at slightly different times due to the nature of the circuitry that generates the signal. After asserting RESET on the filter, the output data will not be correct until 2 output samples have been generated.

The pinout for the structure description is given in Table 6-1.The filter was designed with a scanpath for testing. Test vectors can be generated by using data from an IRSIM or THOR simulation model generated from the structure description. To use the test mode, the pin SHIFT should be set high and the appropriate serial data placed on SCANIN.

6.2.3 Details of the Implementation.

The control unit for the filter family is comprised of a finite state machine that controls an address calculation unit. L/D state variables must be updated for each input cycle, and the proper sequencing of coefficients must be generated. This can be easily achieved by using a pipelined controller, as depicted in Figure 6.3 for the L/D = 3 case. If each state variable is examined individually, it is found that the address for the coefficients need only be incremented by 1 each time an input sample is processed. Since the impulse response was assumed to be symmetric, address generation can be implemented using a counter that counts from 0 to L/2, then back down. When

TABLE 6-1 Pinout for the FIR filter family.

Pin Name	Sig. Type	Description
VDD	Power	Positive power supply terminal
GND	Power	Negative power supply terminal
PHI1	Clock	Phase 1 of the clock
PHI2	Clock	Phase 2 of the clock.
p1slow	Clock	Phase 1 of the slow clock.
p2slow	Clock	Phase 2 of the slow clock.
RESET	Control	Reset the processor.
SHIFT	Test	When asserted, scanpath is enabled.
SCANIN	Test	Input for the scan chain.
SCANOUT	Test	Output for the scan chain.
DSMINBAR	Input	Complement of the single bit input.
DSMIN	Input	The single bit input.
FIROUT	N bit output	Bit-parallel output.
dvalid	Control	Signals new output valid.
dvalidbar	Control	Complement of dvalid.

the address generation is time-multiplexed, the counters can be implemented as shown in Figure 6.3. Preset values are needed to start the 3 generators in the correct place. Within the controller, preset values set the datapath control signals so that the adder either increments, decrements, or stalls.

As an example, consider the case where L/D = 3, and L = 9. Figure 6.4 shows the progression of addresses generated for this case. It was assumed that the impulse response was symmetric, so only 5 coefficients are stored, numbered in order from 0 to 4. During the first input sample, the addresses are generated as 0, 2, 3. The controller registers are set so that the adder in the address calculation unit increments on the first fast clock cycle, then decrements, then adds. As the calculations progress, output samples will be calculated in clock cycles 8, 18, and 25. Data Valid will go low at the end of cycles 9, 18, and 27.

The data path for the filter is shown in Figure 6.5. The LDS signal is a delayed version of the LATCH signal generated in the controller. LDS and CLEAR can occur in the same cycle, since latching of the data indicates that the register storing that particular state should be cleared. The controller and datapaths are fully pipelined. The critical path of the entire filter is dependent on the

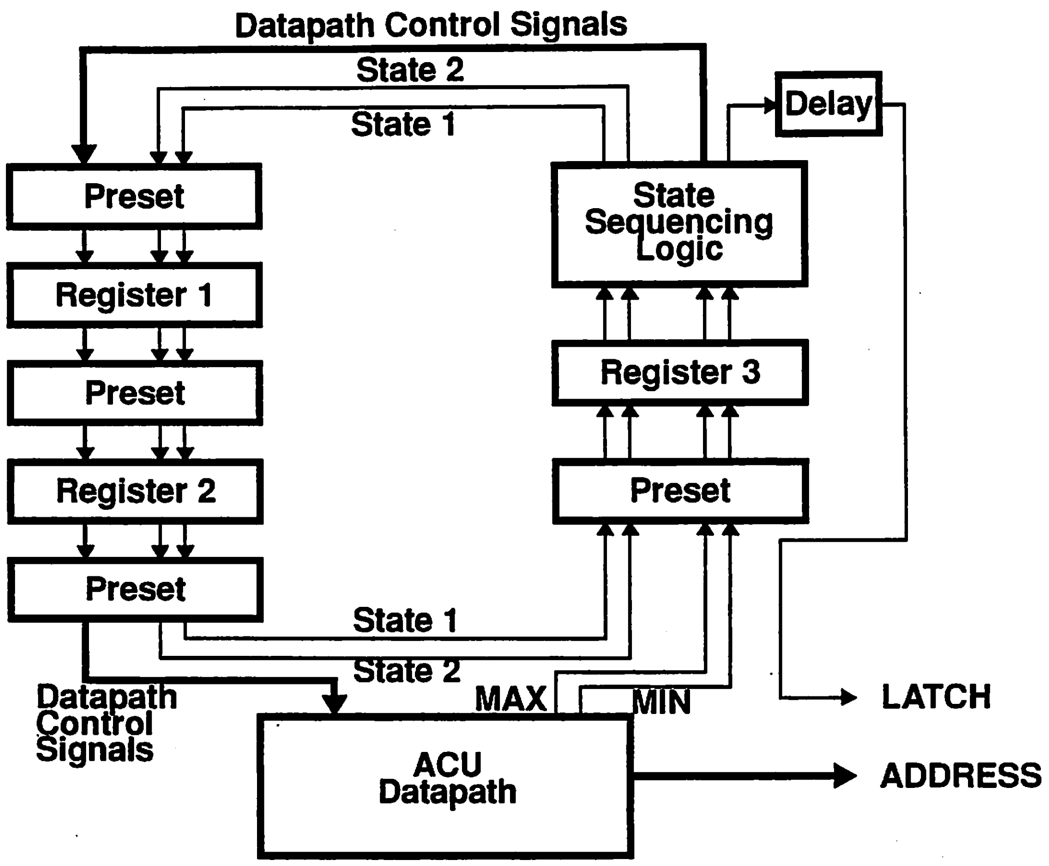


Figure 6.3 Block diagram of the controller for the case where L/D = 3.

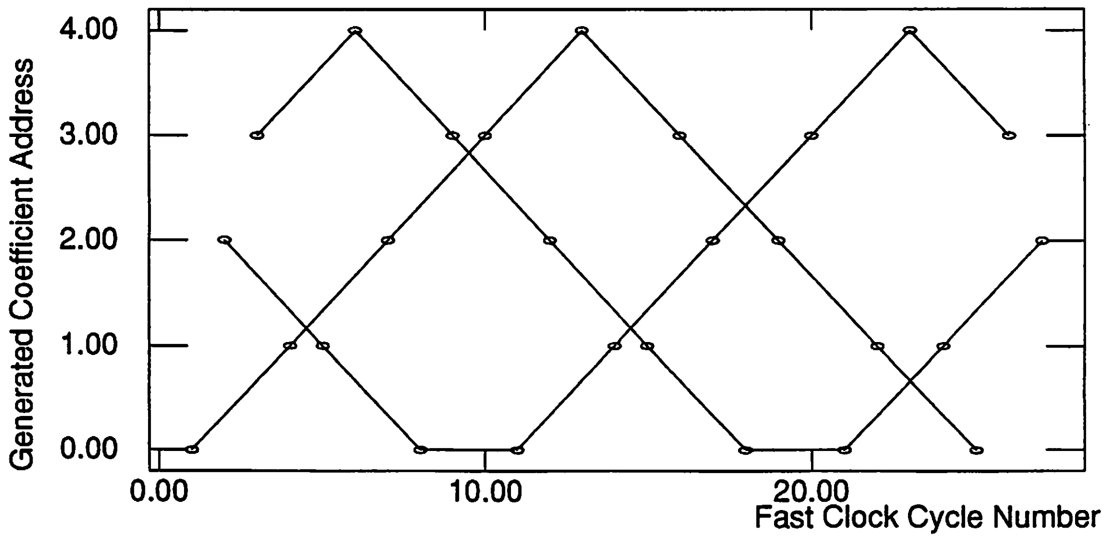


Figure 6.4 Sequence for generating coefficient addresses for the case L/D = 3.

various parameters used in the design, but is dominated by either the time for the coefficient ROM

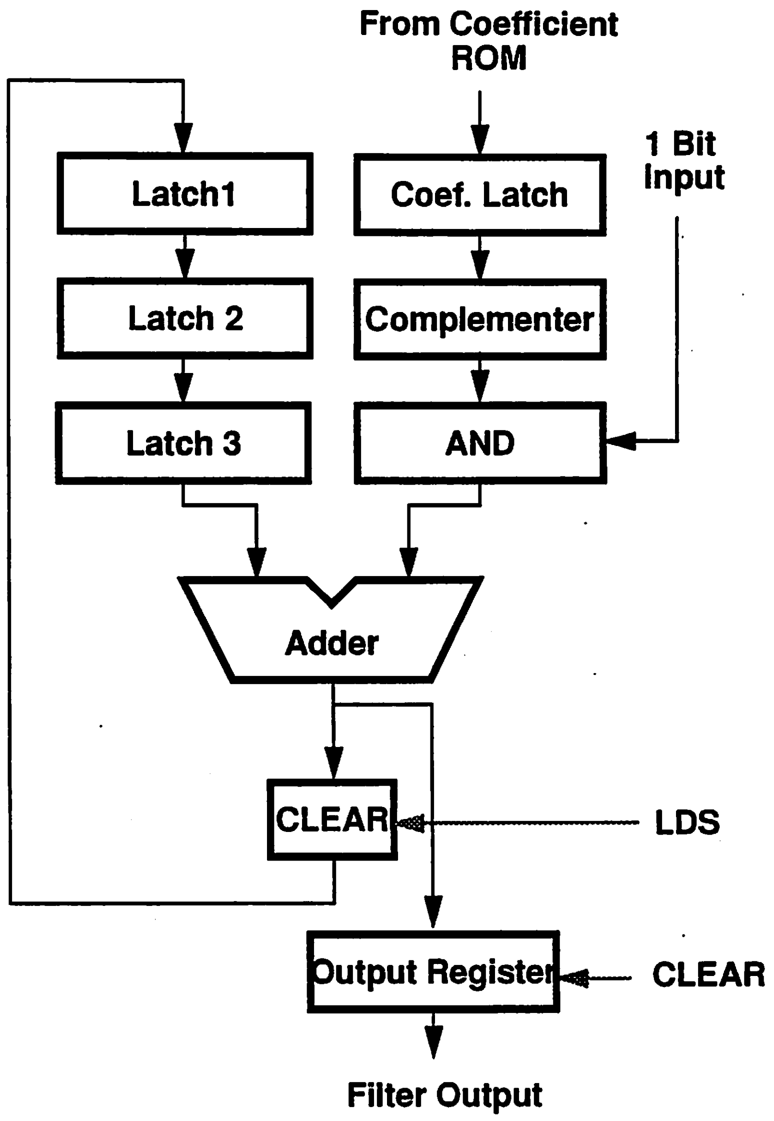


Figure 6.5 Datapath for the FIR filter family for the case L/D = 3.

to evaluate or the sum of ripple carry adder delay and the register access times. In all fabricated versions so far, the filter was found to be faster than the modulator connected to it.

In retrospect, this filter family can waste quite a bit of power. The registers in both the datapath and the controller are set up in such a way that they are always clocked at the fast sample rate. A parallel implementation where each datapath register is loaded only once per input sample could be more favorable. However, the adder would still be used 3 times per input sample.

6.2.4 DECFIR

The program DECFIR was written to handle parameter generation for this family of architectures. The basic parameters needed to generate the filter parameters are the tap values, the decimation ratio, and the wordlength to use in the filter datapath. DECFIR generates the initial state information for the controller and coefficient tables based on this information.

Since the layout of the datapaths and PLA structures is based on tiling algorithms, it is not difficult to estimate the required area for a design. Routing area is not deterministic since an interactive floorplanner is used. It was found by experimentation that adequate area estimates could be obtained using the sum of the macrocell area and scaling it by a constant. A similar result has also been obtained in another project for more random layouts [117]. Table 6-2 shows the results of

TABLE 6-2 Area estimation results for the case of L/D = 3. Results are for 2μm CMOS.

Dec. Ratio	Estimated Area	Area from an actual layout
32	4.7 mm^2	4.8 mm^2
64	5.9 mm^2	5.9 mm^2
128	6.9 mm^2	6.7 mm^2

area estimation for the L/D = 3 case for differing values of D, the decimation ratio.

6.3 CIC Filter family.

6.3.1 Background

CIC (Comb Integrator Cascade) filters have been used in digital filtering for many years. The basic approach is to use a cascade of integrators and differentiators (or comb filters) as pictured in Figure 6.6 for the decimation case with 3 stages. The filter decimates by a factor D and directly implements the transfer function shown in Equation 6.1.

$$H(z) = \left(\frac{1 - z^{-DM}}{1 - z^{-1}}\right)^3 \tag{6.1}$$

An interpolating filter can be obtained by interchanging the placement of the differentiators and the integrators. The value of M can be used to shape the filter response, but when it is set to 1, the entire differentiator chain can be run at the lower sampling rate [118]. When the feedback fac-

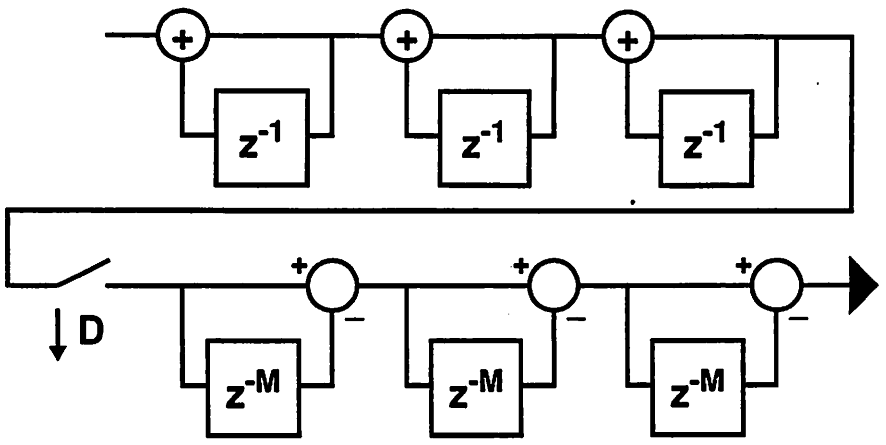


Figure 6.6 Block diagram of a CIC decimation filter.

tor in the integrator is 1, the filter is not necessarily stable since the poles lie on the unit circle. It has been shown that if the filter is implemented using 2's complement arithmetic, and certain constraints on the register wordlengths are met, the filter will be stable [119].

These filters are very efficient for pure sample rate changes since they provide good stopband rejection but very little control over the passband. Transmission zeroes are placed at multiples of the output sampling rate and the frequencies near these points alias into the baseband, as illustrated for the case of a decimate-by-8 sinc^3 filter in the magnitude response plot of Figure 6.7. In this

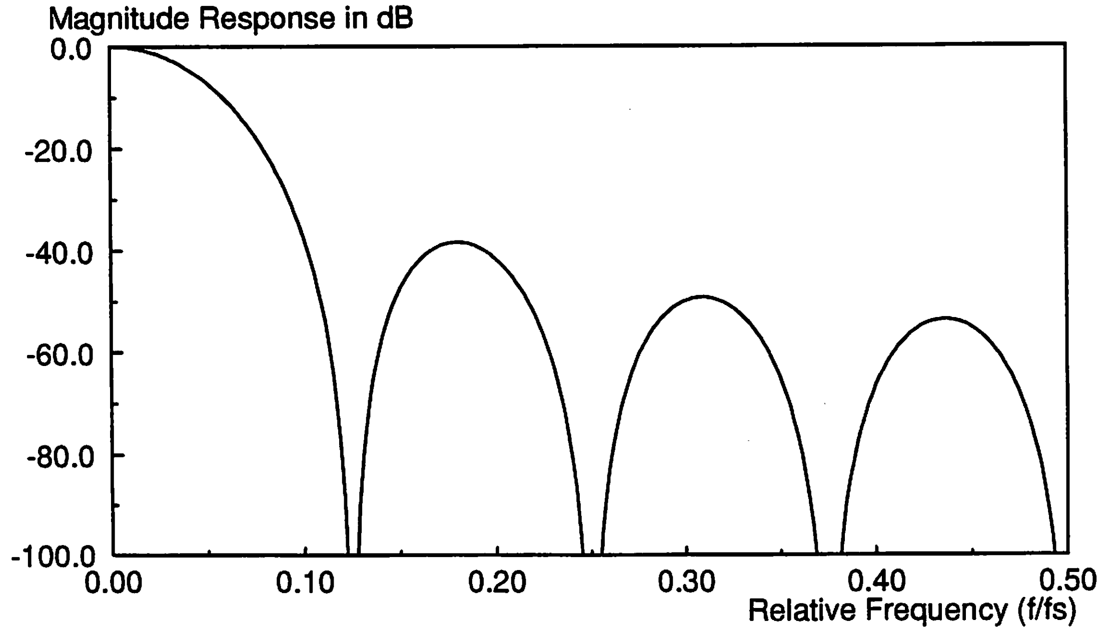


Figure 6.7 Magnitude response for a decimate-by-8 sinc^3 filter.

case, 3 transmission zeroes are placed at 0.125, 0.250, and 0.375. The passband has a monotonic droop, but for signal processing applications, this droop may not be tolerable and it will have to be corrected in the final filtering stage.

These filters have no coefficient storage, few control circuits are necessary, and they can be fully pipelined. However, to insure correct operation, the first register must be set to a minimum wordlength, which for N stages and an input that is in the range (R, -R], the wordlength required is given in Equation 6.2.

$$\text{wordlength} = N\log 2\,(R + 1) \tag{6.2}$$

The wordlength can be significantly large for high decimation ratios. The wordlengths in successive stages can be truncated to reduce area, but this introduces quantization noise. This effect can be analyzed quite easily [119], but for oversampling converters, the extra noise due to truncation can wipe out the gains from noise shaping.

The transfer function of Equation 6.1 can be implemented using an accumulate-and-dump stage when D=1. This circuit has the same transfer function and can be thought of as an FIR filter of length D. This eliminates the need for 1 register and 1 subtracter. It can be applied in any CIC implementation for the middle integrator differentiator combination. It simplifies the datapath at the expense of extra control logic.

Instead of using a single CIC filter for sample rate changes, several CIC filters can be used in cascade. By using several stages, the wordlength requirements can be minimized which can lead to a smaller overall implementation. This approach was implemented in the DECGEN filter compiler, which was developed by Hang and Jain at UCLA [51]. A heuristic algorithm was used to minimize the area required by the composite filter cascade.

6.3.2 Overview and Black Box View

Two versions of CIC filters were created for the DECGEN project, and both have been added to the design system library. Both versions used digit-serial differentiator implementations but one implementation used bit-parallel integrators with a final accumulate and dump stage, while the other used digit-serial integrators. Since the filters used digit-serial arithmetic, the number of I/O pins is dependent on the wordlength used in the filter and the digit size used for processing.

Digit-serial arithmetic is a generalization of bit-serial arithmetic. Instead of using a single bit, several bits are processed at the same time. For example, suppose the wordlength is 16 bits and the digit size is 2 bits. On each clock cycle, the input to the digit-serial processor will be 2 adjacent bits from the input word and 8 cycles total will be required for processing a single word. Depending on the implementation, the digits may start from the MSB side or the LSB side of the original word, much like in bit-serial processing. The CIC filters in the DECGEN library work with the least-significant digit first format.

The structure descriptions for both of these architectures are written by programs based on input parameters taken from the design database or provided by the user on the command line. Floorplans are also generated by these programs so that automatic place and route options can be used. Several modifications were made to these programs to increase functionality and to provide rudimentary area estimation. Since the clock generators are based only on divider circuits, only decimation ratios that are a power of 2 can be implemented with these circuits. The two architecture versions will be discussed separately.

6.3.2.1 Bit-Parallel Integrator version

For the bit-parallel integrator version, there is an option for having bit-parallel or digit-serial outputs from the filter. The output wordlength and the digit size must be set so that their product is a multiple of the decimation ratio. It is not necessary for the input and output wordlengths to be the same since the wordlength can be adjusted in the parallel to serial convertor after the last integrator. The module generators will add logic for testing the filters if desired. The extra logic consists of multiplexers on the input to allow data to come from 2 sources which is useful for isolating filters for individual testing. The pinout for the architecture template is given in Table 6-3 and the timing relations for the output data are shown in Figure 6.9.

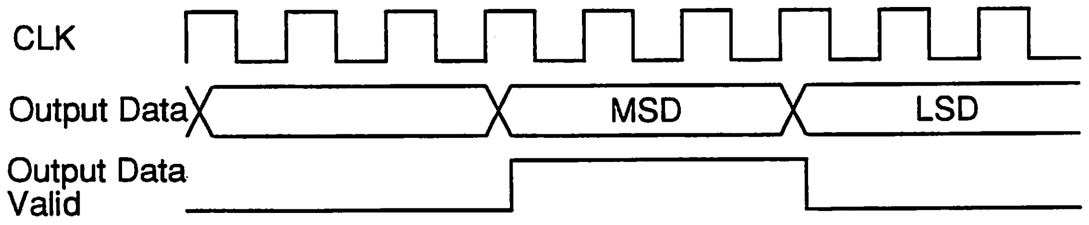


Figure 6.8 The output timing relations for the input signals for a CIC filter with digit-serial integrators. Digit-size was assumed to be 4.

6.3.2.2 Digit-Serial Integrator version

Since both the integrators and the differentiators are implemented using digit-serial arith-

TABLE 6-3 Pin names for the bit-parallel integrator version of the CIC filter.

Pin Name	Sig. Type	Description
VDD	Power	Positive power supply.
GND	Power	Negative power supply.
CLK	Clock	Input clock, frequency is equal to the input sampling rate.
LOAD1	Control	Signal is high when most sig. digit is present at the output.
gRSTb	Control	Reset signal, active low.
CLKp	Test	Alternate clock input for testability.
TESTC1	Test	If asserted, use CLKp rather than CLK as input.
IN1p	N-bit test input	Alternate data input for testability.
TEST1	Test	If asserted, use IN1p rather than IN1 as input.
IN	N-bit input	Input data, wordlength is N.
OUT	M-bit output	Output with digit or word length M depending on choice of output style.

metic, there are restrictions on the value of the wordlengths that can be used. One of the design parameters for the filter is the clock ratio, which specifies the number of clock cycles per input sample (not input digit). The input and output wordlengths must be a multiple of the clock ratio. The digit size for the input is then the input wordlength divided by the clock ratio, and a similar argument holds for the output wordlength. The word size can be truncated after the integrator section, so the input and output wordlengths need not be the same Table 6-4 gives the pin information used in the structure description and Figure 6.9 shows the timing relations that must be obeyed

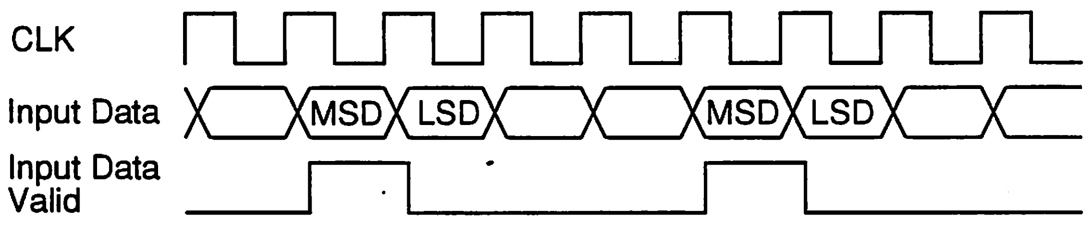


Figure 6.9 The expected timing relations for the input signals for a CIC filter with digit-serial integrators. Digit-size was assumed to be 4.

when providing inputs to the filter. The input data valid signal should last as long as the input digit is valid. The data are latched on the falling edge of the input clock. The filter generates data with output timing relations as shown in Figure 6.9.

TABLE 6-4 Pin names for the digit-serial integrator version of the CIC filter.

Terminal	Sig. Type	Description
Vdd	Power	Positive supply.
GND	Power	Negative supply.
fCLK	Clock	Input clock running at frequency (clock ratio) * (input sampling rate).
LOAD1	Control	Input for data valid signal of the previous digit serial stage.
LOAD2	Control	Output data valid signal. Goes high when most sig. digit is on output.
gRSTb	Control	Reset, active low.
CLKp	Test	Alternate input for fCLK.
CLKdivp	Test	Input for testing differentiator stage.
TESTC2	Test	When asserted, CLKp and CLKdivp are used as the clock inputs rather than fCLK.
INp	N-bit test input	Alternate data input.
LOAD1p	Test	Alternate input for input data valid signal.
TEST2	Test	If asserted, data is taken from INp and LOAD1p rather than from IN and LOAD1.
IN	N-bit input	Digit-serial input data with digit size N.
diffOUT	M-bit output	Digit-serial output data with digit size M.

6.3.3 Details of the Implementation

The design and implementation of these filters has been previously documented in detail [51]. The bit-serial integrators were modified to provide zeroing when the gRSTb signal is asserted. This extra circuitry is not necessary, but simplifies simulation and testing, since registers can be set to a known state. The modified register circuit diagram is shown in Figure 6.10.

Currently, the circuits are only marginally functional. The key problem lies in the circuits used for generating internal clocks which is built from divide-by-2 circuits. The circuit, shown in Figure 6.11, has been reported previously [120] and is currently not a part of the standard Lager libraries. It is based on a single phase latch which has a lower bound on hold times after the clock edge has fallen and the data was latched. The latch can allow data to shoot through, so when direct feedback is used, a race condition exists. The solution to this problem is to add an extra buffer on the latch output to insure that the hold time constraint is met.

These errors in single phase circuits are difficult to find by simulation. To date, the cells in the

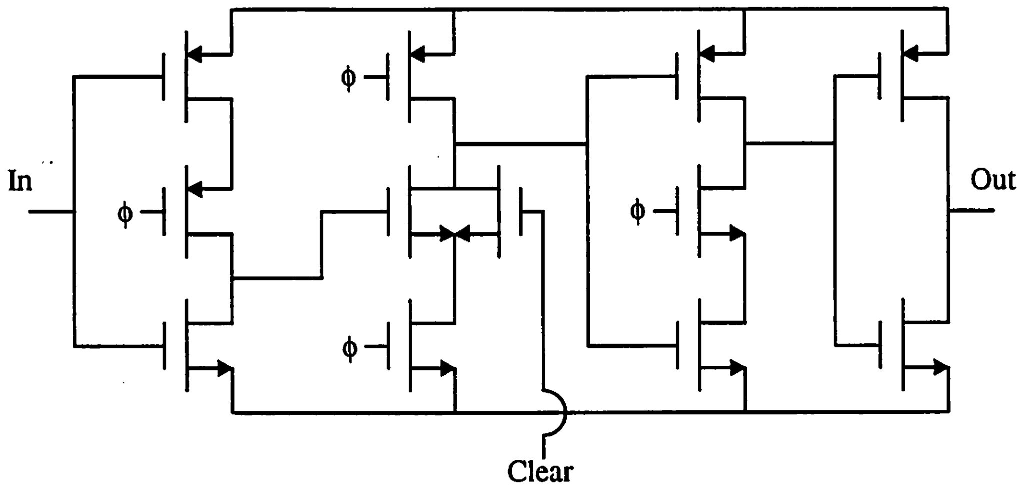


Figure 6.10 Modified TSPC latch with reset.

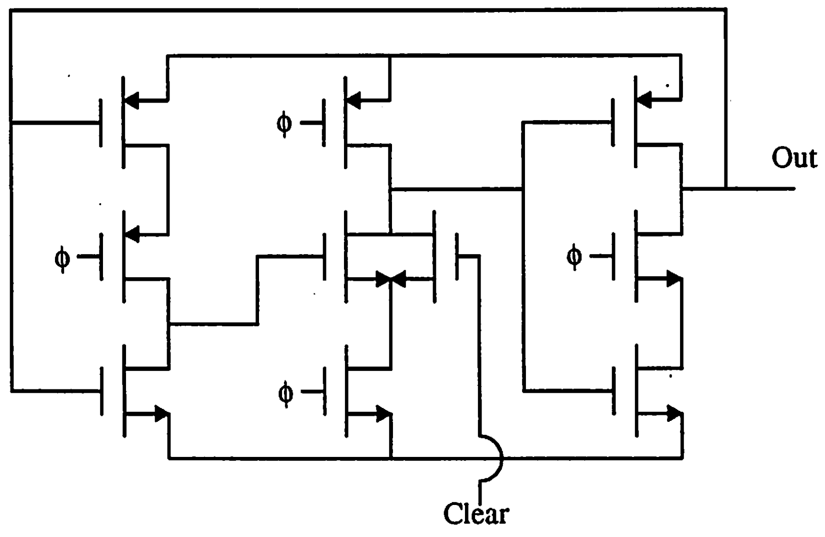


Figure 6.11 Divide by 2 circuit.

Lager libraries have generally ignored clock edges since this is not a crucial issue in the design of 2-phase circuits. Switch-level simulators such as IRSIM were not able to pick up problems due to slow clock edges. Circuit-level simulations are not easy, since the latch should be simulated within a large system design. Routing contributes a significant capacitive load to the clock lines and must be accurately modelled. Another problem is that the single phase logic circuits are not scalable. As the circuit speed gets faster, clock edges need to be sharper, but from experience, it appears that the propagation delay of the latch decreases faster than the buffer driver delay.

The filters were implemented using a scalable buffer strategy. Centralized clock buffers are generated for each module and a single clock line feeds the datapath. This technique should be examined more closely if the filters are redesigned. While the gate loading for the registers is accurately modelled, it is more difficult to calculate the parasitic capacitance associated with the routing from the buffer to the registers. A better strategy would be to use dedicated buffers in control slices on the datapath. This eliminates the extra capacitance loading due to routing, and the buffers can be programmed to drive a known capacitance.

6.4 Custom Decimate-by-2 FIR filter

6.4.1 Overview

As mentioned in chapter 4, decimation can be incorporated in an FIR filter implementation so that only the required output samples need to be calculated. This requires only L/D registers for state variable storage when L is the filter length and D is the decimation ratio. While it appears that the computation rate drops by a factor of D, it will not decrease this much since address computation becomes more complicated. Usually, only one copy of the coefficients is stored in a coefficient ROM so when decimation is incorporated, the state variable updates no longer follow a simple incremented progression. One solution is to keep a single address pointer for each state variable and then update this pointer each time the state variable is updated. This solution was adopted in the custom FIR solution described in Section 6.2. Since L/D was assumed to be small, few address pointers are required and the computation overhead and extra hardware are not excessive.

For FIR filters with larger L/D ratios, another solution is available. Filters with large L/D ratios tend to be used in situations where there are many processor cycles in each sampling period. In this case, a time-multiplexed architecture is desired. A general purpose processor is one solution, but if L is too large, a full multiplier may be required. A custom solution to this problem was developed for DECGEN. This solution used a full parallel multiplier and a hardware realization of an address calculation algorithm.

6.4.2 Black-box view

This filter can take either digit-serial or bit-parallel inputs. The structure description files for this filter are generated using a program that requires only the quantized coefficient values, the internal wordlength to use, and the input and output wordlengths. If digit-serial input are used, the basic timing relations are the same as for the CIC filters. For bit-parallel inputs, the data is latched internally on the falling edge of the input clock as shown in Figure 6.12. The timing for generation

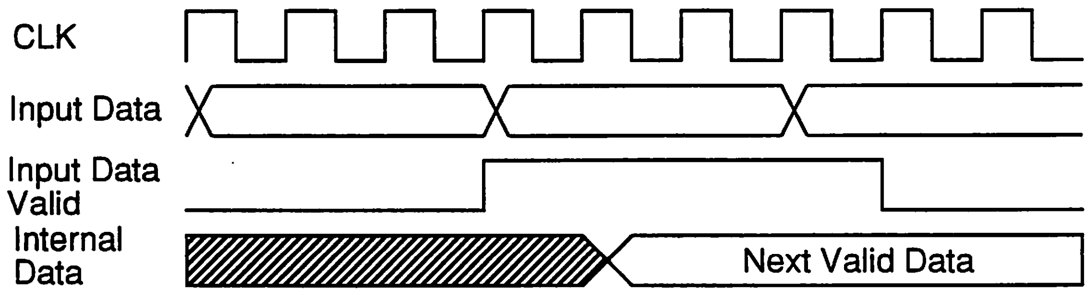


Figure 6.12 Timing diagram for latching of the bit-parallel input.

of the output

DataValid signal is shown in Figure 6.13 and the pinout is given in Table 6-5.

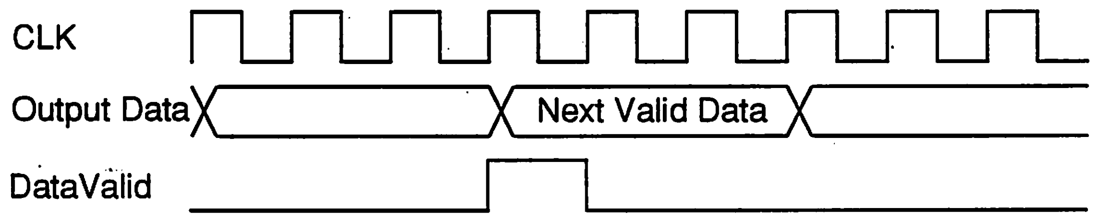


Figure 6.13 Timing relations for the output data for the decimate-by-2 filter.

6.4.3 Implementation details

The architecture and circuit designs are given along with the documentation for the other DECGEN modules [51]. The controller was modified to account for a global RESET synchronization and differs slightly from previous documentation. When RESET is asserted, the controller will wait for the next time that LOAD2 is asserted and then begins the filtering operations. The implementation also makes use of the div2N cell described in the previous section. Care must be taken to insure that this cell will operate correctly. Several registers within the datapath make use of single phase clocking, so clock buffers must be simulated and adjusted accordingly when changing feature size.

TABLE 6-5 Pin names for decimate-by-2 FIR filter.

Terminal	Sig. Type	Description
Vdd	Power	Positive power supply.
GND	Power	Negative power supply.
CLK	Clock	Input clock, must run at (L/D)*(output sampling rate).
LOAD2	Control	Data Valid signal from the last stage.
RESET	Control	Active HIGH reset signal.
SCAN	Test	If HIGH, enables the scanpath.
SCANIN	Test	Input for scan data.
SCANOUT	Test	Output for scan data.
CLKp	Test	Alternate clock input.
TESTC3	Test	When asserted, enables CLKp as clock input.
LOAD2p	Test	Alternate for input data valid signal.
SInp	N-bit test input	Alternate for data input.
TEST3	Test	When asserted, allows data entry from LOAD2p and SInp rather than LOAD2 and SIn.
SIn	N-bit input	Input data with word or digit size N.
OUT	M-bit output	M bit output data.

6.5 C-to-silicon

6.5.1 Overview

The C-to-silicon project began with the Kappa processor developed for Lager III [121]. Lager III provided support for silicon compilation of structural descriptions of systems to layout [122]. Azim and Shung developed an assembler for this architecture to allow mapping of generic DSP algorithms to a parameterizable architecture. Rimey furthered this work by providing a retargetable compiler which could accommodate user-defined processor architectures [1]. With Lager III, it was possible for a user to write a C language description and have it compiled to a variety of user defined architectures. In LagerIV, the software and processor libraries were updated and the tools were designated as the C-to-silicon toolset by Thon [123]. Thon refined the Kappa processor and renamed it the Puma processor. The C-to-silicon process is illustrated in Figure 6.14.

In order for the compiler to generate code for different architectures, an architecture description must be supplied. This description delineates the processor resources and the valid register

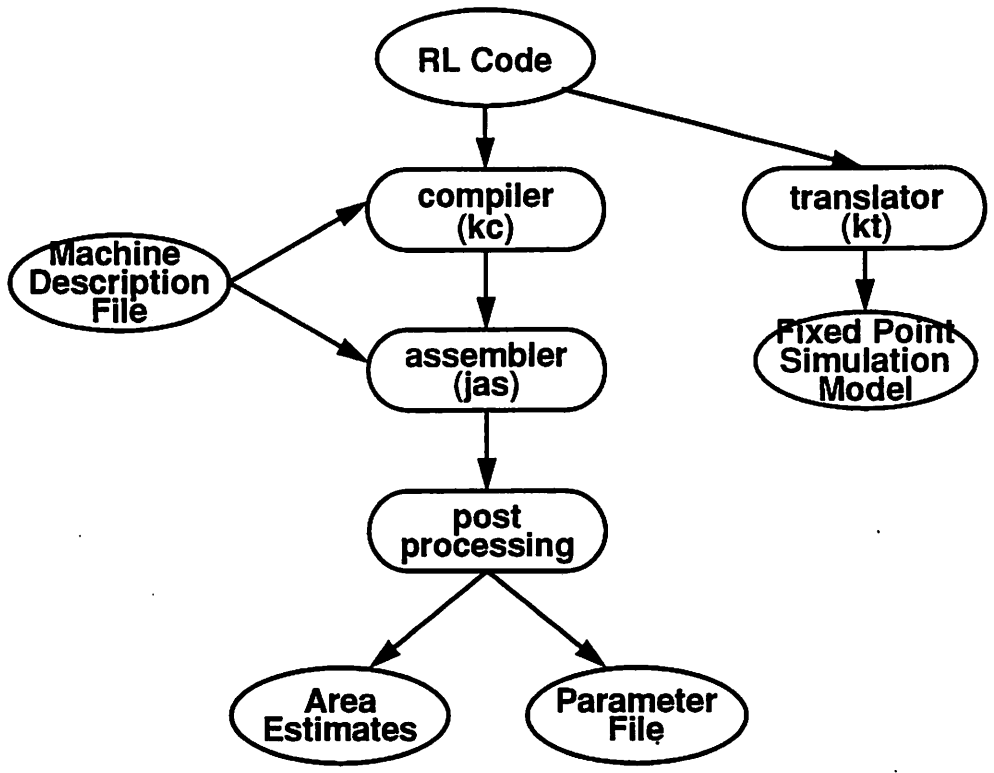


Figure 6.14 Design flow for the C-to-silicon design tools.

transfers that can take place on the datapath. The compiler assumes a fixed 2 level control structure, but extra datapaths and memory units can be added. The output of the compiler is a register transfer language description of the algorithm that must be processed by an assembler to obtain the microcode and parameters necessary for layout. The assembler also needs an architecture description that specifies the control signals that activate individual register transfers. A previous effort sought to generate this information automatically [124].

The C-to-silicon tools are not integrated under a design manager. The user must perform each step of the compilation process by hand. A script was written for this project so that the process can be automated to some extent, but this assumes that all steps proceed without errors. Retargeting the compiler and assembler for new architectures can be somewhat tedious.

6.5.2 Black-box description.

As mentioned previously, the basic architecture defined for the C-to-silicon tools is called Puma. A block diagram of the processor is shown in Figure 6.15. The basic Puma architecture has been used to implement several projects including a pitch extractor, a PID controller, and several

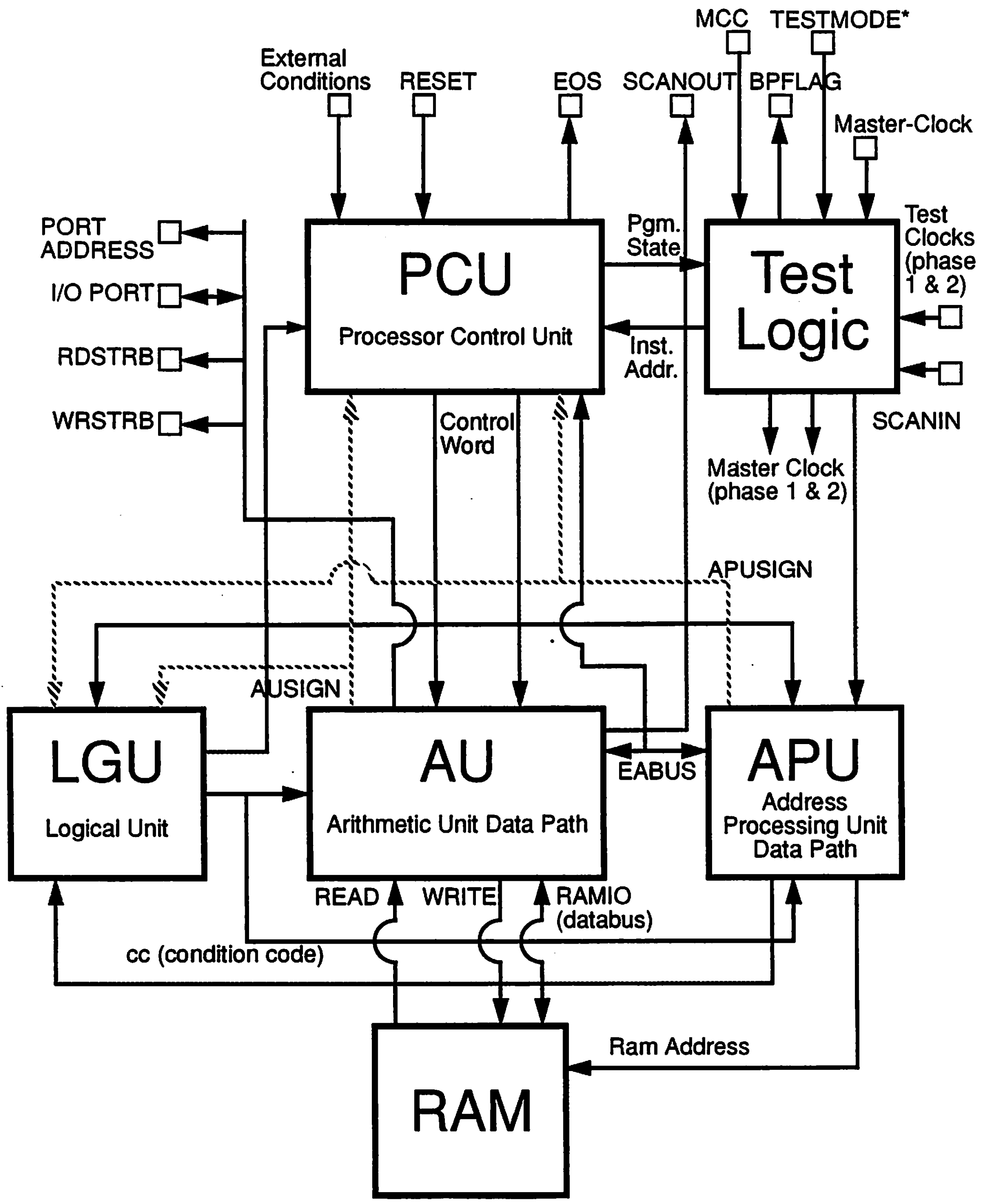


Figure 6.15 Block diagram of the Puma processor.

digital filters. In addition to basic processor I/O, several control lines can be defined to control the processor or detect conditions within the processor. The basic pinout is given in Table 6-6. An internal clock generator is used to create the internal 2 phase clocks used in the processor. Only a single clock input is required.

TABLE 6-6 Pinout for the PUMA processor.

Pin Name	Sig. Type	Description
Vdd	Power	Positive power supply.
GND	Power	Negative power supply.
CLOCKIN	Clock	Input master clock.
RESET	Control	Processor reset signal.
MCC	Control	Master clock control. When low, processor is in normal operating mode.
BPFLAG	Control	Output signal used in testing.
READSTRB	Control	Asserted by the processor when the processor reads DATAINPORT.
WRITESTRB	Control	Asserted by the processor when the processor reads DATAOUTPORT.
WRPORT	Control	Signal set high when writing data port and low when reading port. Can be used for tristate control of a bidirectional I/O pad.
OFFCHIP2CFSM	Control Bus	Signals that were defined as input boolean flags in the RL code.
LGU2OFFCHIP	Control Bus	Signals that were defined as output boolean flags in the RL code.
TESTMODEINV	Test	Active low signal that enables testing mode.
TPHI1	Test	Clock phase 1 for use during testing.
TPHI2	Test	Clock phase 2 for use during testing.
SCANIN	Test	Input for the scan chain.
SCANOUT	Test	Output for the scan chain.
PORTADDRESS	4-bit input	Supplies address for reading and writing external data ports.
DATAINPORT	N-bit input	Data input to processor.
DATAOUTPORT	N-bit output	Data output from processor.

The timing diagram for processor input and output is shown in Figure 6.16. The input data to the processor must be valid at the end of ϕ_2, at which point the data is latched. The output data is held valid over both phases of the clock cycle. The signal WRPORT was originally used to control a the signal direction of a bidirectional I/O pad for the stand-alone version of the processor.

For digital filtering, it was found that the several of the resources in the Puma architecture were underutilized and that the compiler does not allow control of scheduling processor I/O. The external world must be able to supply on the cycle when it is needed by the processor and to latch data when it is generated. A buffer was needed to store data since data was often not generated or

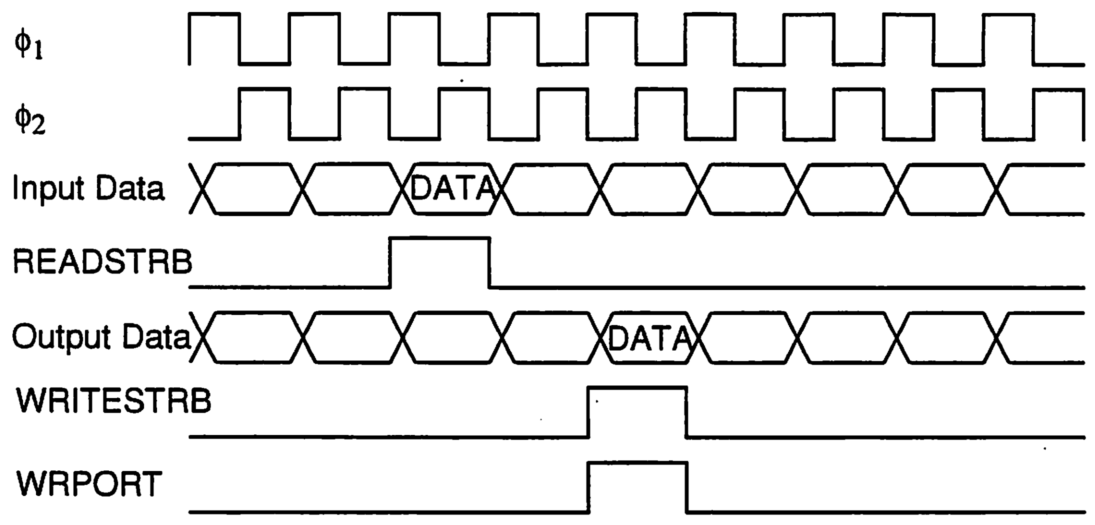


Figure 6.16 Timing relations for the Puma processor.

latched on the cycle when it was needed. The intent of the C-to-silicon tools was to allow minor architectural changes so designs could be optimized for different applications.

The Lambda architecture was developed with these ideas in mind. The Lambda processor is a stripped version of the Puma processor. The address calculation unit was deleted, so only immediate addressing schemes could be used. This is not a problem since addresses are known at compile time for digital filters. It is also design trade-off, since addresses are now stored in ROM rather than being calculated on the fly making this choice favorable when the additional ROM size is less than the size of the address calculation unit. The datapath functionality was reduced, and an I/O buffering unit was added. The block diagram for the Lambda architecture is shown in Figure 6.17 and the pinout, which has changed slightly from Puma, is shown Table 6-7. The processor can have different widths for the input and output data but the processor datapath must have the same wordlength as the output data.

The processor timing is now data driven and determined by the input program to the compiler. Input data is latched on the falling edge of ϕ_2 after the falling edge of IDataValid.The input data must be held valid over the entire ϕ_2 cycle time. In addition, the output data is latched and remains valid until the rising edge of the next ODataValid signal. No detection circuitry was incorporated for detecting read/write conflicts internal to the processor.

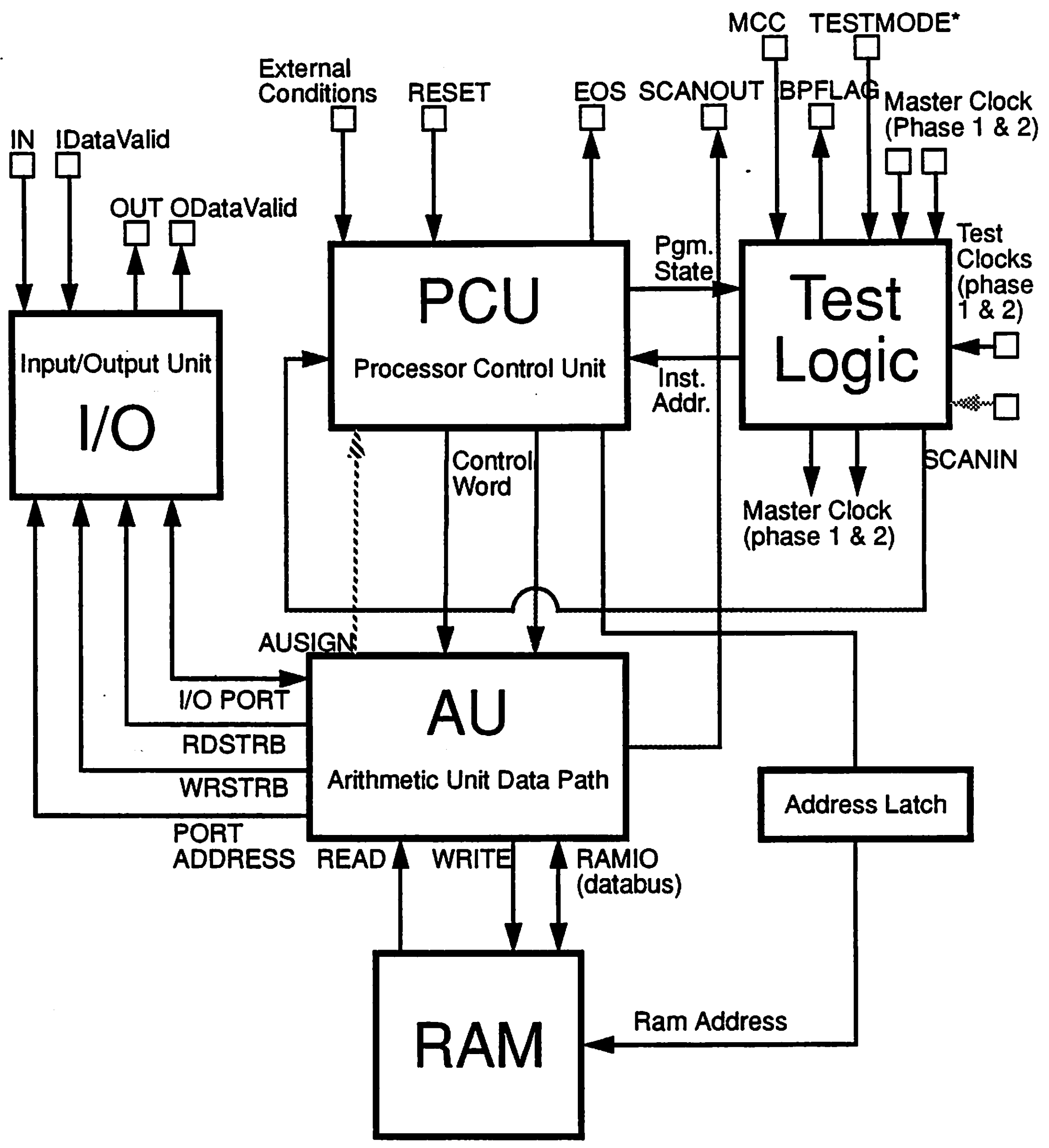


Figure 6.17 Block diagram of the Lambda processor.

6.5.3 Implementation details

The Puma architecture has been fully documented in previous work [121], [125]. Some of the leafcells were updated to allow more feedthrough paths in the datapath cells allowing better routing and smaller datapath implementations. The Lambda architecture has been optimized for digital filtering. Using one filter design, it was found that the Lambda architecture provided an implementation that was 60% smaller in area than the Puma implementation.

TABLE 6-7 Pinout of the Lambda Processor.

Pin	Sig. Type	
Vdd	Power	Positive power supply.
GND	Power	Negative power supply.
PHI1	Clock	Processor clock phase 1.
PHI2	Clock	Processor clock phase 2.
RESET	Control	Global processor reset signal.
IDataValid	Control	External source asserts this signal when input data is valid.
ODataValid	Control	Asserted by the processor when output value is valid.
MCC	Control	Controls testing modes.
BPFLAG	Control	Status signal during testing mode.
TESTMODEINV	Test	Active low signal that enables testing mode.
TPHI1	Test	Clock phase 1 for use during testing.
TPHI2	Test	Clock phase 2 for use during testing.
SCANIN	Test	Input for the scan chain.
SCANOUT	Test	Output for the scan chain.
IN	M-bit input	Input for the processor.
OUT	N-bit output	Output for the processor.

A block diagram of the Lambda processor was shown in Figure 6.17. The logical unit and address calculation unit were removed, but an extra unit for I/O buffering was added. The I/O unit is targeted towards applications where several input data values are consumed for each output data value generated. The I/O unit consists of a single register buffering the output values, and a small register file, as shown in Figure 6.18. The processor controller provides addresses for reading data from the register file. The local controller uses a counter to generate addresses for the register file.

The basic operation is as follows. When an external source strobes IDataValid high, the controller detects the falling edge, latches the data into the register file, and the counter is incremented by one. The processor signal READSTRB controls the multiplexer which selects either the counter value or the address coming from the processor. When READSTRB is asserted, the register file places data on the output port using the address provided by the processor. No extra circuitry was provided to detect conflicts between the IDataValid signal and the READSTRB signal from the processor controller. Since the program execution is periodic and deterministic, conflicts can detected from simulation. If a conflict is present, the algorithm must be rescheduled, either by

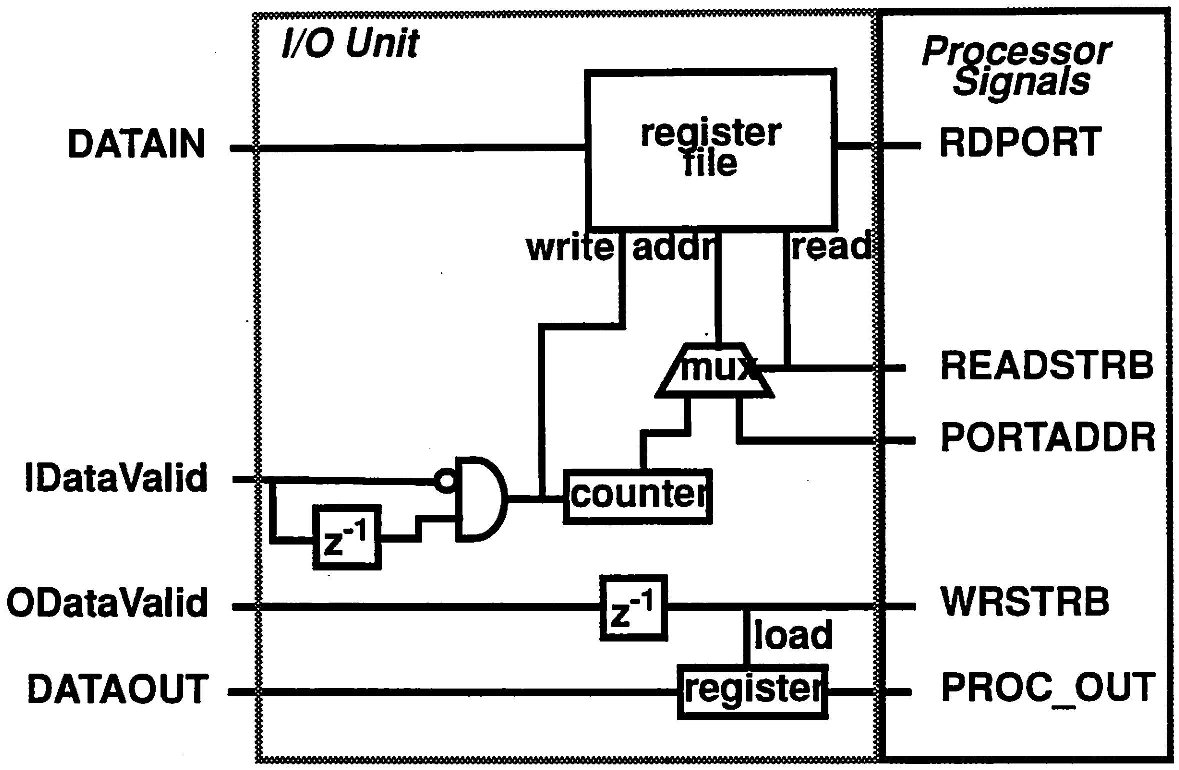


Figure 6.18 Block diagram of the I/O unit in the Lambda Processor.

recompilation or by modifying the assembly code. When the processor asserts WRSTRB, the data is latched in a register, and the WRSTRB signal is delayed by 1 clock cycle to create the OData-Valid signal.

In the current implementation, the input data is aligned with the processor data at the LSB. Some implementations may favor MSB alignment, so the template can be changed accordingly. Also, the I/O unit clock is derived from the processor clock. To achieve synchronization, the processor clock must be at least as fast as the clock that generates the input data valid signal.

The PUMA datapath and address calculation unit are shown in Figure 6.19. The Lambda datapath is shown in Figure 6.20. The functionality was reduced by lowering the maximum right shift from 15 to 7 bits. Additionally, the shifter is now controlled only by the controller rather than allowing variable multiplies, which were implemented through the shift register called RCOEF shown in Figure 6.19. Removing the address calculation unit provided area advantages because it removed a major routing bottleneck in the processor layout. Only minor changes were needed in the architecture description for correct compilation to this new processor architecture.

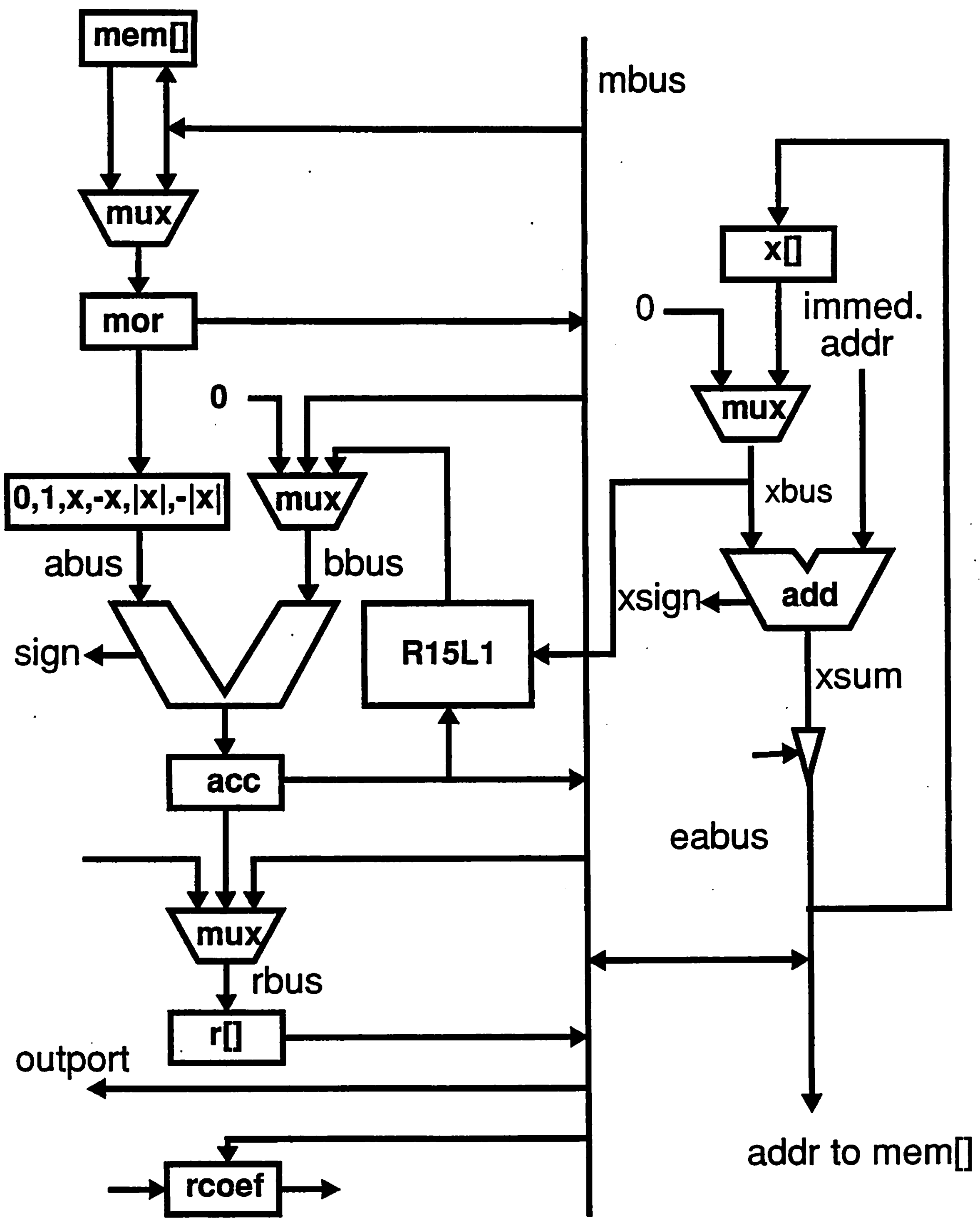


Figure 6.19 Block diagram of the PUMA datapath and Address Calculation Unit.

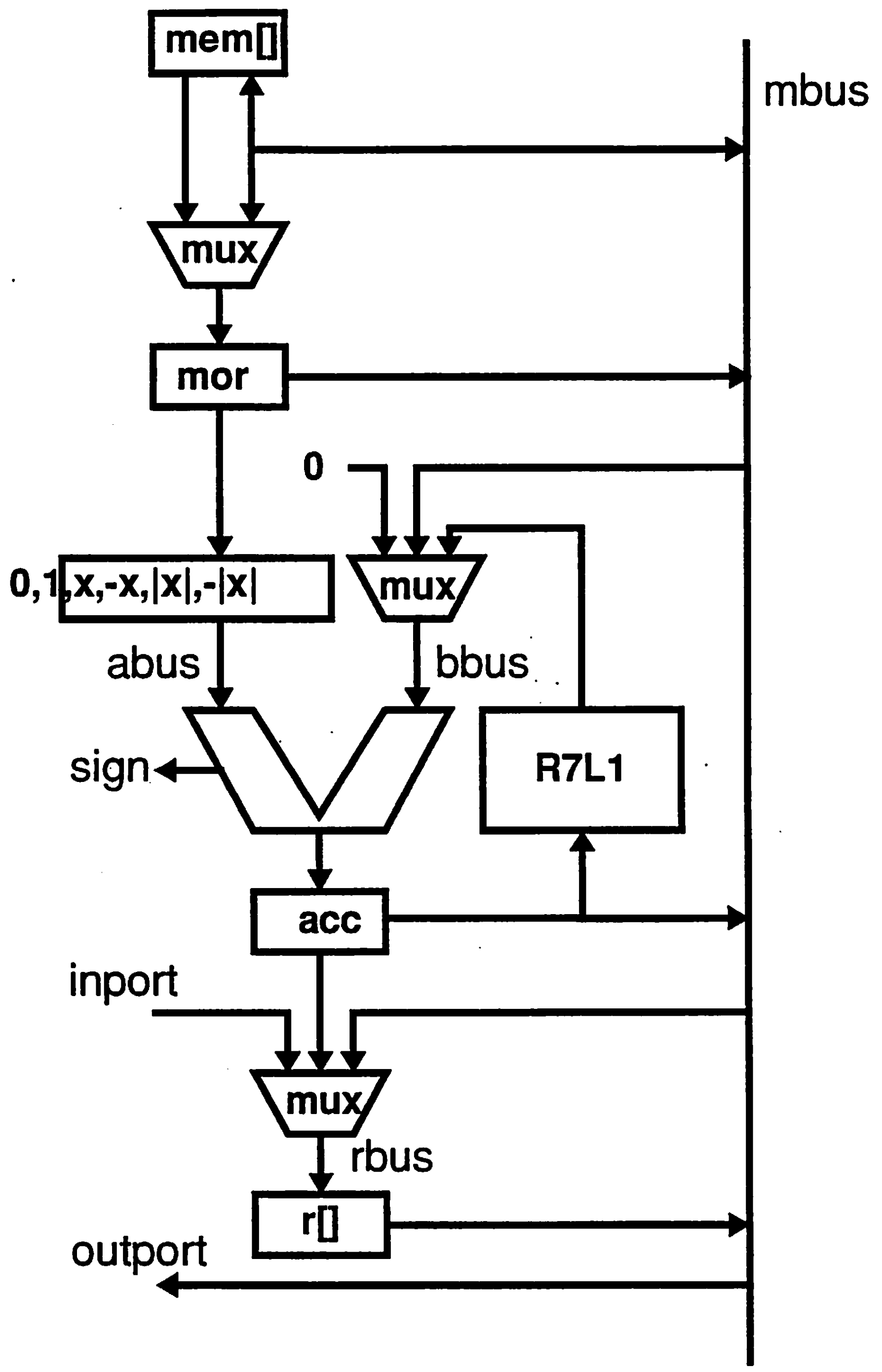


Figure 6.20 Block diagram of the Lambda datapath.

6.6 Parallel to Serial Converter

6.6.1 Description

For chip development, serially transmission of signals cuts down the number of pins required for packaging. A simple implementation of a parallel-to-serial converter is included in the architecture template library. The converter is implemented with a programmable counter and a shift register. The program pisogen can be used to generate the state table for the converter. Note that for an N bit parallel to serial conversion, N+1 clock cycles are needed for the converter to work.

The pinout for the converter is shown in Figure 6-8 and the timing diagram is as shown in

TABLE 6-8 Pinout for the parallel to serial converter.

Pin Name	Sig. Type	Description
Vdd	Power	Positive power supply.
GND	Power	Negative power supply.
PHI1IN	Clock	Input clock, phase 1.
PHI2IN	Clock	Input clock, phase 2.
OCLOCK	Clock	Output serial data burst clock.
RESET	Control	Reset the converter and wait for next DataValid.
DatatValid	Control	Input signal to denote that input data is valid.
TSHIFTIN	Test	When asserted, scan test mode enabled.
SCANIN	Test	Input for the scan chain.
SCANOUT	Test	Output for the scan chain.
WORDIN	N-bit Input	Parallel data input.
SERIALOUT	Output	Output serial data stream.

Figure 6.21

6.6.2 Details

The converter is implemented using 2 parallel registers, a standard cell logic block, and a PLA as shown in Figure 6.22. The counter was implemented in the PLA since a it required less area than a dedicated counter. The entire converter could be implemented using standard cells, providing arbitrary aspect ratios and denser routing. It is uncertain whether this version change would make the overall circuit smaller. It appears that the density of the PLA will be traded for the white space due mismatch in aspect ratios and the implementations will be roughly the same size. A sim-

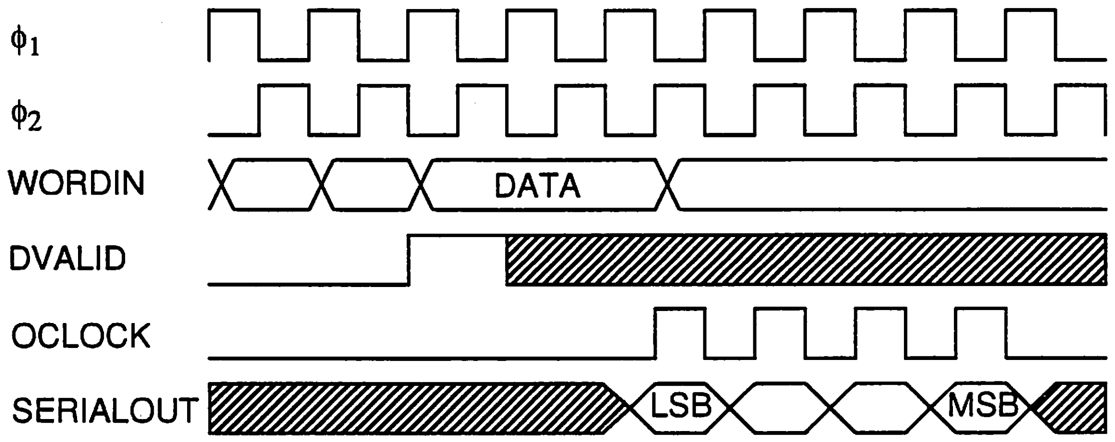


Figure 6.21 Timing diagram for the parallel to serial converter.

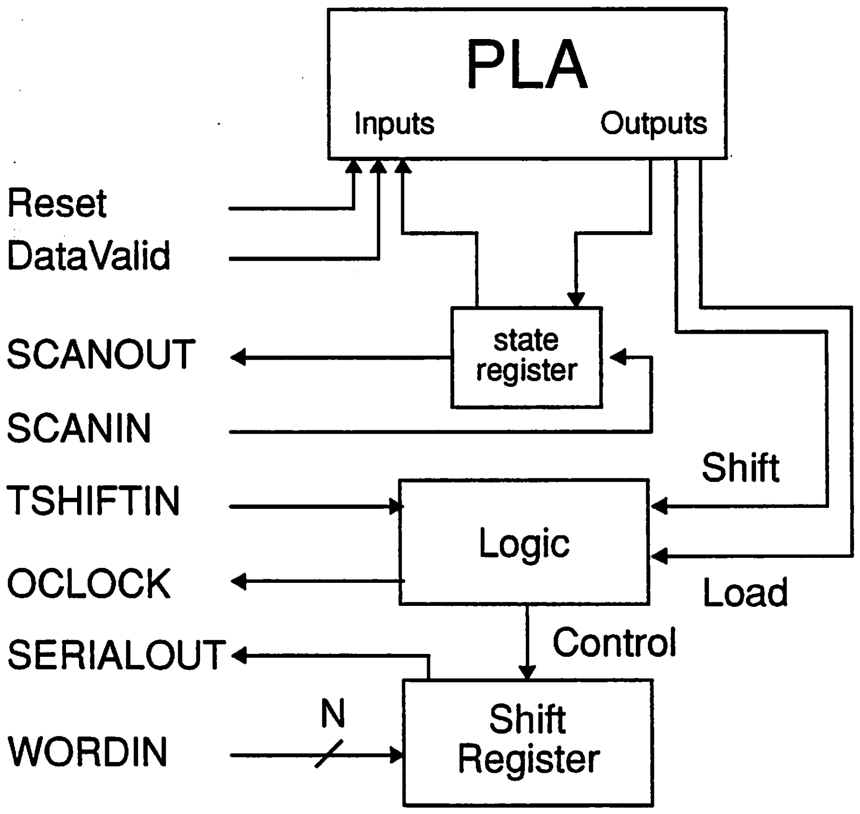


Figure 6.22 Block diagram of the parallel to serial converter.

ilar version could be developed using a single phase register. This would be helpful, since only one clock phase would need to be routed to the circuit.

6.7 Clock Generators

The clock generator module contained within the TimLager Cell library does not provide adequate non-overlap time if the clock buffer is heavily loaded. The circuit diagram is shown in Figure 6.23. Some caution must be observed when using this clock generator since the amount of

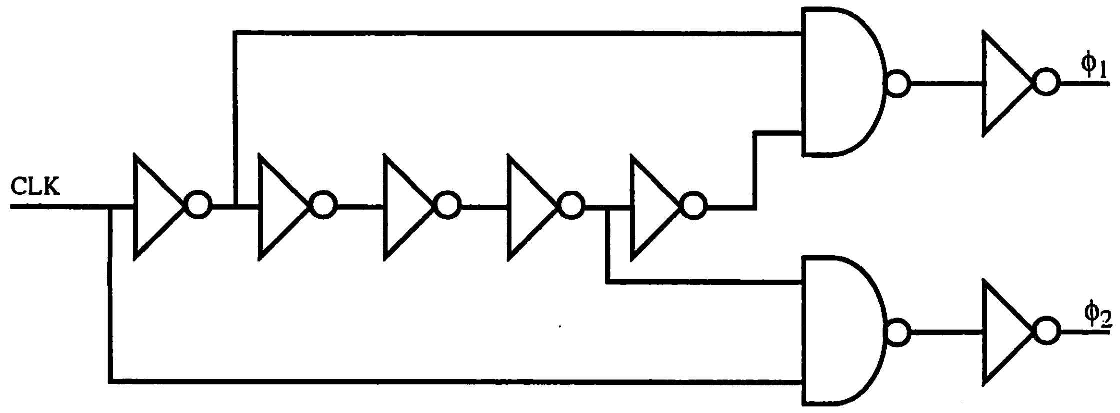


Figure 6.23 Clock generator circuit for generating 2 non-overlapping phases.

non-overlap time is dictated by the delay in the inverter chain. If the clock driver is heavily loaded, the edges will have longer rise and fall times and will lessen the non-overlap time. Cross-coupled NOR gates overcome this limitation, and should be considered if the library is redesigned. Simulators like IRSIM do not have the accuracy to determine if an adequate non-overlap time has been maintained so hand calculations should be performed to verify buffer drive capacity.

As a standard methodology to increase testability, clocks lines are always routed through a multiplexer so that if there is a problem with non-overlap time, off-chip clocks can be used. The output of the on-chip clock generator is multiplexed with a pad input. After the multiplexer, the clock signal is run into the on-chip clock buffers.

A master clock is used to generate all other clocks on chip. D flip-flops can be used to divide the master clock to generate some clock values. If the master clock must be divided by a factor that is not a power of 2, custom logic must be designed. In the current architecture library, there is a need for a divide by 6 circuit. This can be realized using 3 flip-flops and some random logic, as shown in Figure 6.24. This circuit provides 2 output clocks, which are 1/2 and 1/6 the input clock rate.

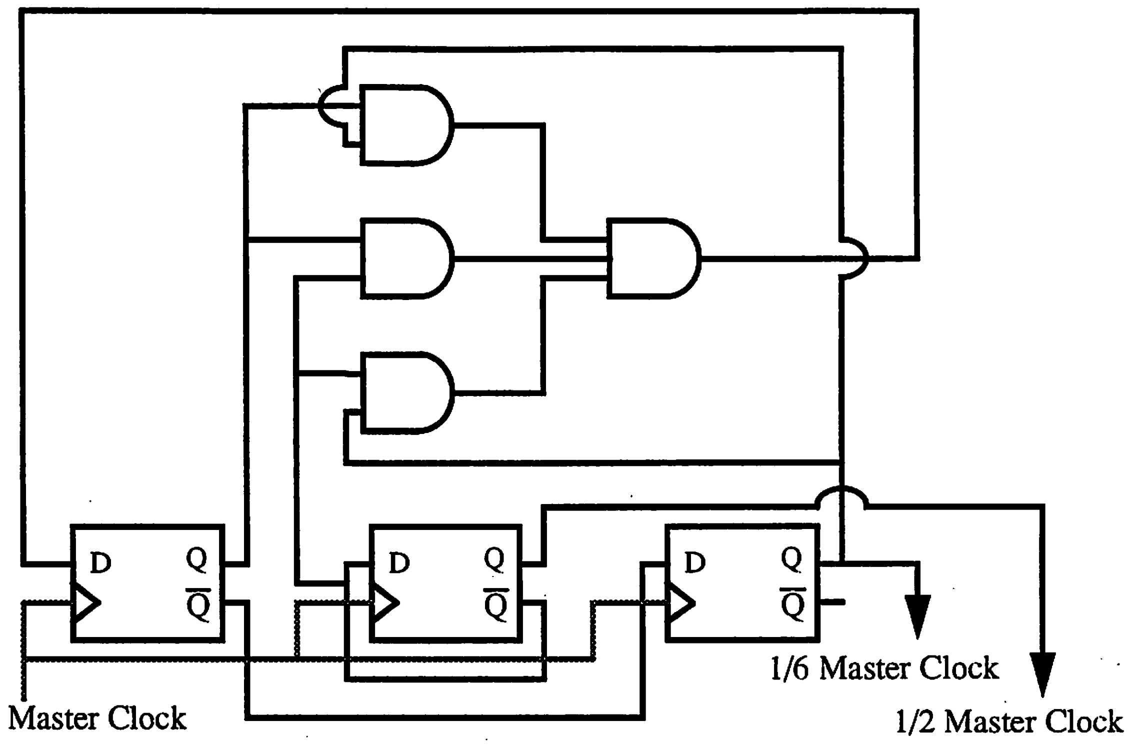


Figure 6.24 Block diagram of circuit used to generate 1/6 and 1/2 clock phases.

6.8 Analog Modulator Architecture Selection

6.8.1 Overview

While minimizing the area of the digital filter is an important concern, the choice of modulator determines the overall converter performance. The noise shaping ability and circuit imperfections dictate the lowest possible noise floor. The various types of oversampling modulators were discussed in Chapter 2 and various trade-offs were highlighted. Currently, only the second order Δ–Σ modulators are available. An experimental second order first order cascade was developed, but was not fully debugged.

6.8.2 Black Box Description

The modulators in the cell library are not scalable. Various cells exist in 2µm CMOS with different options for capacitors and opamps. A single version exists in 1.2µm CMOS. The variations are listed in Table 6-9. Selection of the proper modulator should be guided by the process in which

TABLE 6-9 Modulators in the Cell Library

Cell Name	Features
dsm2abn	2μm n-well layout, metal1-poly caps, Class AB opamp, int. gain = 0.25.
dsm2ab	2μm p-well layout, double poly caps, Class AB opamp, int. gain = 0.25.
cds2pab	2μm p-well layout, double poly caps, Class AB opamp, int. gain = 0.25, offset zeroing in the first integrator.
dsm2p12	2μm p-well layout, double poly caps, Class AB opamp, int. gain = 0.5.
fcdsm2p	2μm p-well layout, double poly caps, Folded Cascode opamp, int. gain = 0.25.
dsm2pc2	1.2μm n-well layout, metal1-poly caps, Class AB opamp, int. gain = 0.5.

the circuit will be fabricated and by resolution and speed considerations. All of the modulators conform to the basic pinout listed in Table 6-10. There are many pins listed, but most are related to clocking and biasing. Appendix F provides details on how to connect the various supply and bias lines to chip pins in order to minimize the effects of cross-talk. The current sources for biasing were set nominally to 40 μA for Class AB opamps and 275 μA for folded cascode opamps. Analog ground should not be confused with analog Vss and is defined to be the potential halfway between Vdd! and GND!.

For timing, the analog input is sampled on the falling edge of phi1 in all cases except for cds2pab, for which the input is sampled on the falling edge of phi2. A single bit output is generated every clock cycle, and is latched and valid during phi2.

6.9 Estimation Techniques

6.9.1 Area

Area estimates can easily be generated from most target architectures. Since a cell library based approach is used, area for datapaths and tiled structures can be estimated by adding up all of the leafcell area. It has been shown that the total area of a design is proportional to the total leafcell area plus a factor to account for global buses and controller area [117]. This result was obtained for systems that use a simple control structure based on a single PLA.

TABLE 6-10 Basic Modulator Pinout.

Pin Name	Sig. Type	Description
Vdd!	Supply	Analog Vdd
GND!	Supply	Analog Vss, lowest potential
DVdd!	Supply	Vdd for digital comparator
DGnd!	Supply	GND for digital comparator
dout	Output	Single bit output from modulator
doutb	Output	Logical complement of dout
phi1	Clock	Phase 1 of clock
phi1b	Clock	Logical complement of phi1
phi2	Clock	Phase 2 of clock
phi2b	Clock	Logical complement of phi2
phi1d	Clock	Delayed version of phi1
phi1db	Clock	Logical complement of phi1d
phi2d	Clock	Delayed version of phi2
phi2db	Clock	Logical complement of phi2d
shield1 and shield2	Shield	Connect these pins to a clean supply
capshield	Shield	Connect this pin to the well supply
vcmo1 and vcmi1	Bias	Analog ground inputs
vref_pos1 and vref_pos2	Bias	Set to positive full scale voltage
vref_neg1 and vref_neg2	Bias	Set to negative full scale voltage
nbias_in1 and nbias_in2	Bias	Input tail current for n transistor mirror
pbias_in1 and pbias_in2	Bias	Input tail current for p transistor mirror
intin_pos1	Analog Input	Positive analog input
intin_neg1	Analog Input	Negative analog input

Our layout generation tools make use of an interactive floorplanner. Poor initial placement of the leafcells results in layouts that have large amounts of whitespace that inflate area numbers. Thus area estimates will have an accuracy that can be dependent on the designer. To improve on this strategy, 2 area numbers are reported for most architectures. The total leafcell area is reported as well as an area estimate based on a fixed floorplan. It has been found that for most designs, the actual total area is about 1.5 to 3 times the leafcell area for all of the architecture templates in the system. This is slightly lower than the figure reported by Schultz [117], but this is due to the fact that the architectures used in this project have fewer global buses and also to the fact that regular structures are used, so estimates are quite good.

6.9.2 Speed

Digital circuit speed can be estimated by finding the critical path. For parameterized structures, this can be difficult since different critical paths exist for different combinations of parameters. In this project, the analog circuit speed turns out to be much less than the digital circuit speed. While the critical paths of the architectures are generally known, a formalized set of estimators were not developed. Speed only is a problem for the C-to-silicon architectures that run at rates significantly higher than the analog sampling rate with very large programs requiring a large RAM.

6.9.3 Power

Methods for power estimation were being concurrently developed in other projects at Berkeley. The results were not available for inclusion in this project. The basic method is to estimate the loading capacitance due to logic gates and interconnect. Based on a statistical measure, an activity factor can be defined to designate how many bits are changing for an assumed input. Power numbers for the current library are skewed, since a pseudo static PLA is a large contributor to power consumption in most current designs. To guide the design process now, decisions can be guided by the material presented in recent papers on low-power design considerations based on algorithms and architectures [126], [65].

6.10 Adding New Architecture Templates

6.10.1 Different Techniques

The architecture templates are stored in structure description language (SDL) files. SDL files are the input for the Lager Silicon Assembly System. The Lager System is based on assembly of various circuit structures which at the lowest level are simply circuits from a cell library. While circuit synthesis programs can be used in the layout generation phase, to date, only hand-designed leafcells are found in the cell library. SDL is hierarchical, allowing partitioning of complex designs. SDL captures the type of blocks used in a design and their connectivity. In addition, SDL provides programming constructs to a large degree of parameterization.

Since SDL contains information about the component circuits and the connectivity, it is merely one expression of very general information. SDL could be translated to other hardware design language formats to provide for layout generation in other design systems. The previous sections show 3 basic techniques for developing architecture templates in SDL: fixed library cells, parameterized SDL files, and SDL generators.

The analog modulators were developed using the fixed library cell approach. Custom layout was generated and then an SDL file was written to make the cell available within the Lager System for place and route. The PUMA and Lambda architectures and the FIR family were all developed using parameterizable SDL files. Key features like ROM values and wordlengths are parameters to these files. This means that the SDL files always remain the same, but different circuit configurations are generated by altering the input parameter file. The CIC filters and the decimate-by-2 FIR filter were implemented using C programs that write the proper SDL files based on the design parameters. In this case, subtle changes in the architecture were not able to be expressed in terms of parameters.

Each technique provides some advantages. The examples currently in the design system can be used as models for developing new architectures. The philosophy in developing a new digital architecture for the design system is to make it parameterizable so it can handle a range of applications. This often leads to design trade-offs that increase flexibility at the cost of performance. However, it has been found that the penalties are small and the benefits to other potential users is large.

6.10.2 System Integration Issues

It is often necessary to use many different clock frequencies in the various architectures to allow better performance. The FIR family with L/D=3 uses 2 clocks, one at 3 times the frequency of the other. Clocking issues must be considered with care, since multiple clock frequencies increase clock generation circuit complexity, and can increase noise coupling with the analog circuits. Clock skew can also become a problem. For these reasons, clock generation circuits were left external to the architectures so that if centralized clock generation circuitry is desired, it can be implemented.

Digital filters and analog modulators are complex systems on their own. Combining them into a usable integrated circuit requires good interface specification to insure that data is properly interchanged. Within the current architecture library, an attempt has been made to outline some basic interface timing constraints so that all the blocks can interact without requiring extra glue logic.

One basic problem within the design system is the use of different clocking schemes for sequential logic. Four-phase, two-phase, single-phase, and edge-triggered clocking schemes are used throughout the various architectures. Basic I/O timing relations have been developed and nearly all of the bit-parallel architectures obey this scheme. The clock phase phi1 is considered to be the reference clock phase. All circuits should generate a signal to indicate that new output data is present and the output data should be valid until the rising edge of the next this data valid signal. The new output data is guaranteed to be valid on the falling edge of the data valid signal, at which time it can be latched.

The basic scheme is shown in Figure 6.25. While this scheme is rather conservative, it should

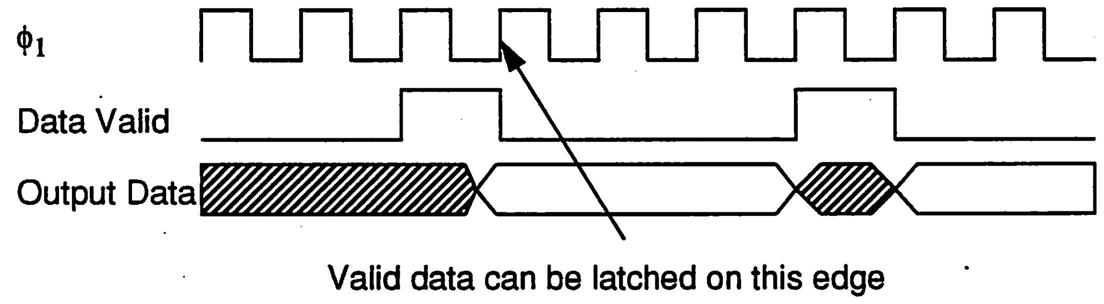


Figure 6.25 Timing relations for system design.

guard against setup and hold time violations, allowing arbitrary mixing of architecture blocks. Multiple clock frequencies also cause complications for the I/O timing. If a slower clock is used in the processor receiving data, it may be difficult to determine when the data is valid, since the data valid pulse may be contained within a single clock cycle. This means that the detection scheme should be based on an edge triggered scheme or an SR latch that processes the data valid signal and the uses local clocks to determine when data valid has gone low. A block diagram of such a circuit is shown in Figure 6.26. The circuit is able to detect the falling edge regardless of what clock rate was used to generate it. If the processor is known to be faster than the processor supplying the data valid signal, a simpler circuit can be used, as shown in Figure 6.27. This is the circuit that was used in the I/O unit of the Lambda processor architecture.

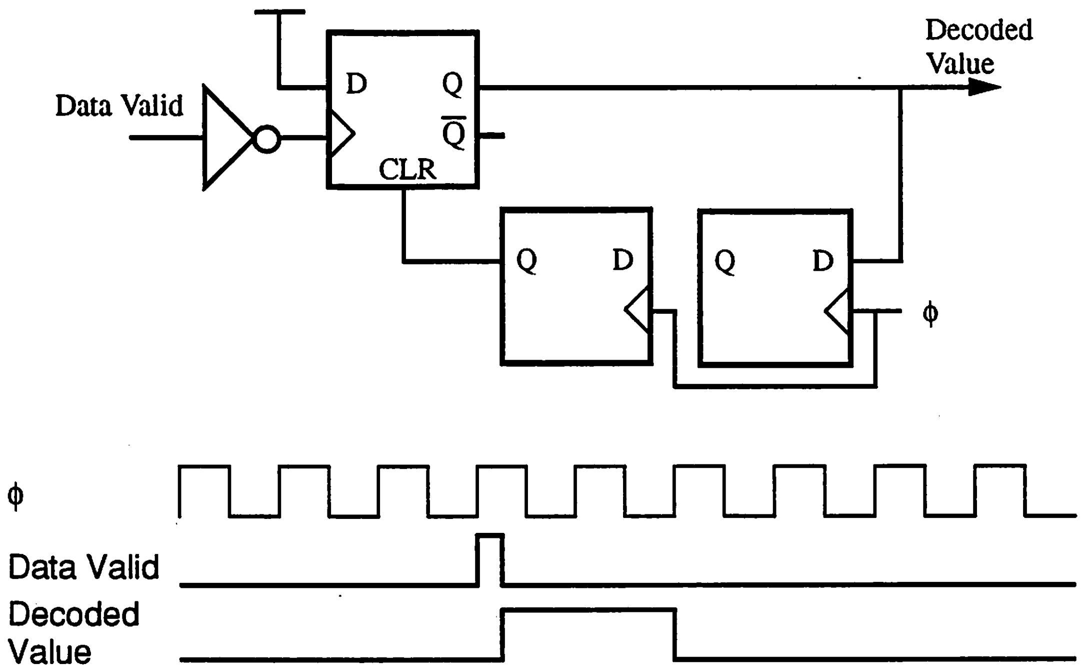


Figure 6.26 Circuit for detecting the falling edge of a data valid signal.

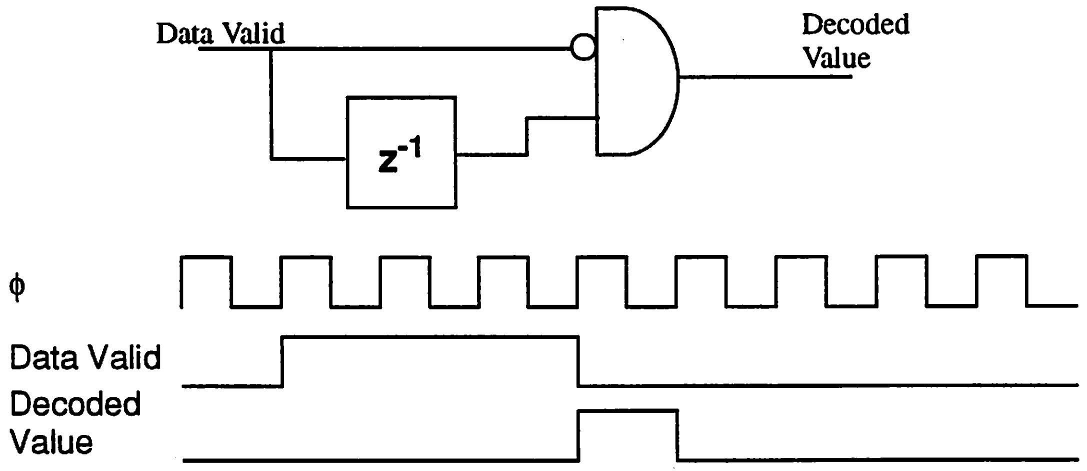
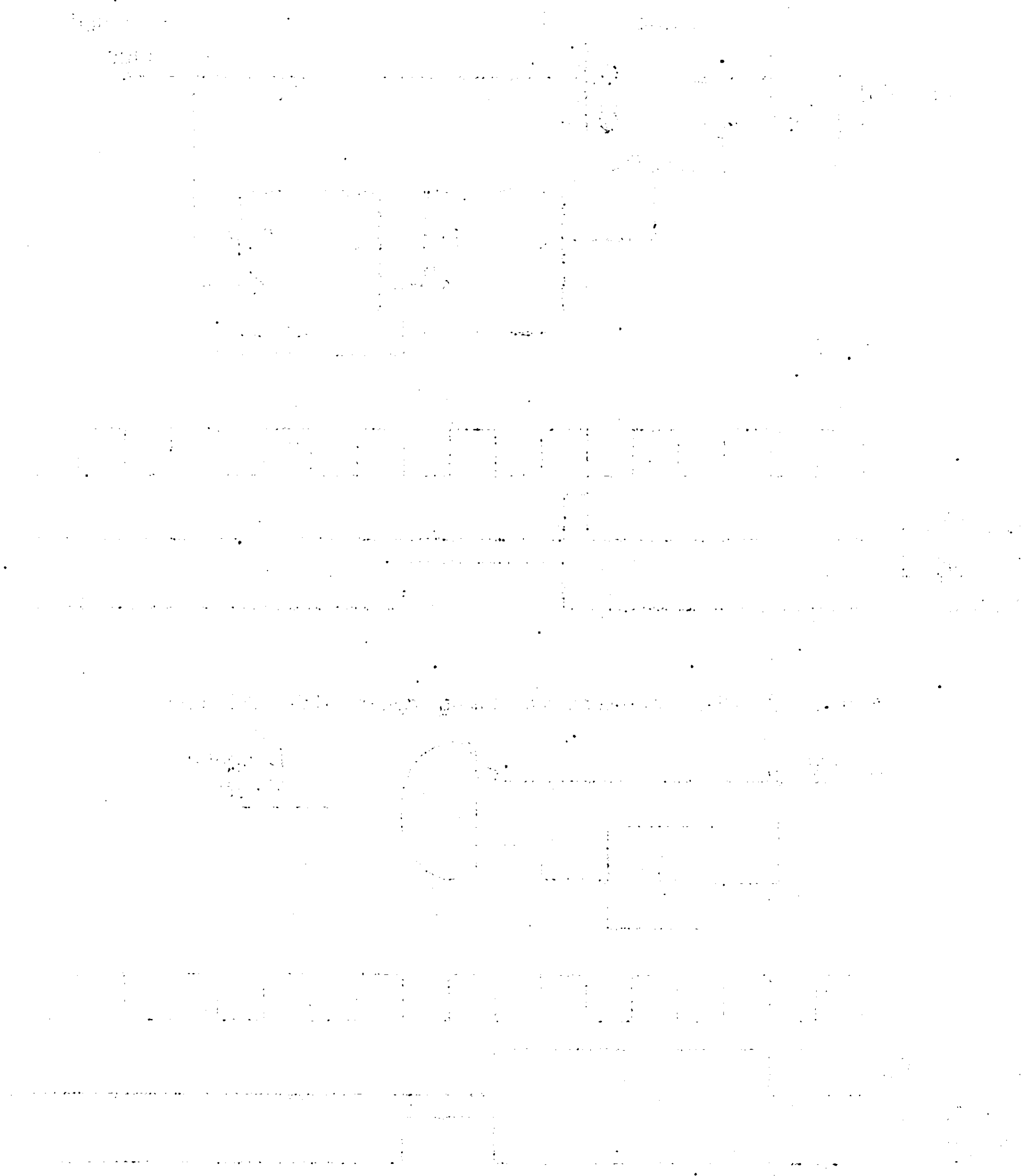


Figure 6.27 Simpler circuit for detecting a falling edge.

6.11 Summary

This chapter presented information on the architecture templates developed in this project. For each template, an overview was given along with information necessary for using the template in a larger design.



[illegible]

[illegible]

CHAPTER 7

Design and Implementation of the Analog Circuits

The design of oversampling modulators is not a simple task. Although there are only a few analog components, it can be difficult to achieve the high resolution promised by theoretical results. New theories are being proposed, but the behavior of a quantizer in a non-linear feedback is still not fully understood. This chapter contains a discussion of the library cells developed in this project and some thoughts on developing module generators for these circuits.

7.1 Design of the Experimental Circuits.

The quality of the analog modulator design is critical to the overall interface performance. Analog circuit imperfections have been shown to limit the performance that can be achieved at high oversampling ratios. Uncertainty in simulation results can lead to over-design which can be costly. Previously, models based on difference equations have tended to overestimate dynamic range and peak signal-to-noise ratio. For a second order modulator, a difference equation model including basic non-idealities predicts that an oversampling ratio of about 150 is required to achieve 16 bit performance. The estimators presented in Chapter 2, Equation 2.9, predict that an oversampling ratio of about 100 is required. In practice, oversampling ratios above 250 are required [85]. A better correlation of modulator performance and simulation models will enhance the design process and allow better design approaching theoretical limits.

For this project, several designs were fabricated and characterized to help formulate a better simulation model for design. The basic second order Δ–Σ modulator was chosen as the test configuration, and variations were made to the opamps, capacitors, capacitor ratios, and processes. One of the advantages of oversampling A/D converters is the tolerance of component mismatch allowing them to be implemented in standard digital CMOS processes. Part of the design process was to establish what circuits are necessary to implement high resolution conversion without the need for precision analog techniques.

7.1.1 Modulator Overview

The circuit diagram for the modulator is shown in Figure 7.1 This is a 2 phase circuit, with the input sampled on ϕ_1 and accumulation and update of the feedback voltage occurring on ϕ_2. The input switches are clocked with delayed phases to implement the bottom plate sampling scheme. This type of sampling decreases the distortion caused by the effects of charge injection since the sampling switches open into the fixed potential provided by the summing node [77]. A modular circuit layout was used to facilitate changes in the modulator components, as detailed in Section 7.5. Figure 7.2 shows the organization of the layout for one of the modulators along with a

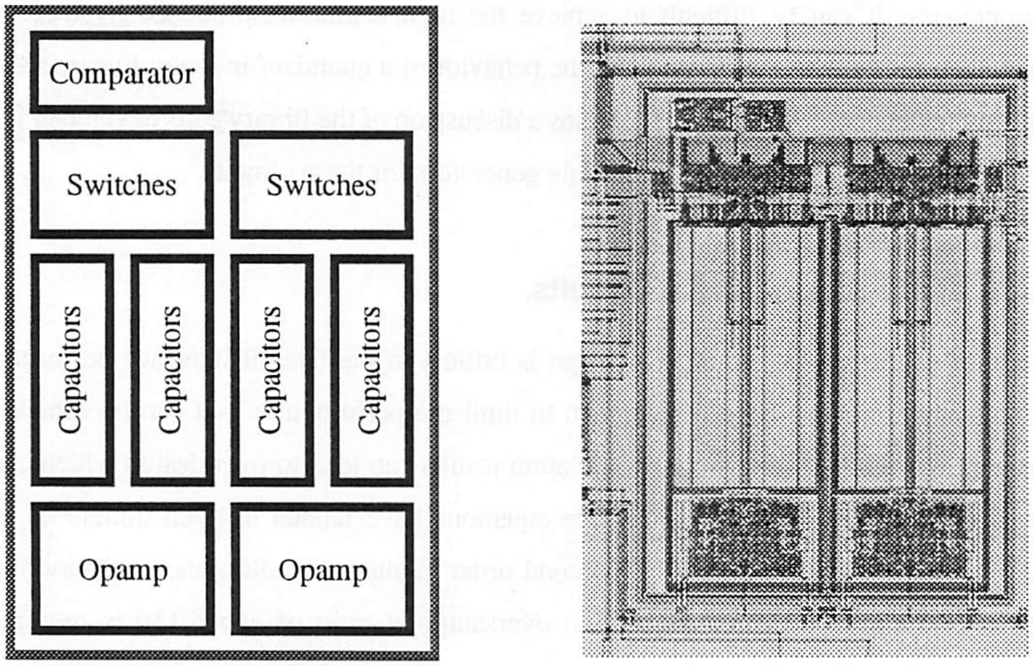


Figure 7.2 Block diagram of modulator layout and a die photo from a fabricated chip.

die photo of a modulator. To explore the effects of different circuit phenomena, several variations detailed in Table 7-1 were designed and fabricated. All circuits were designed to operate from a nominal 5V supply.

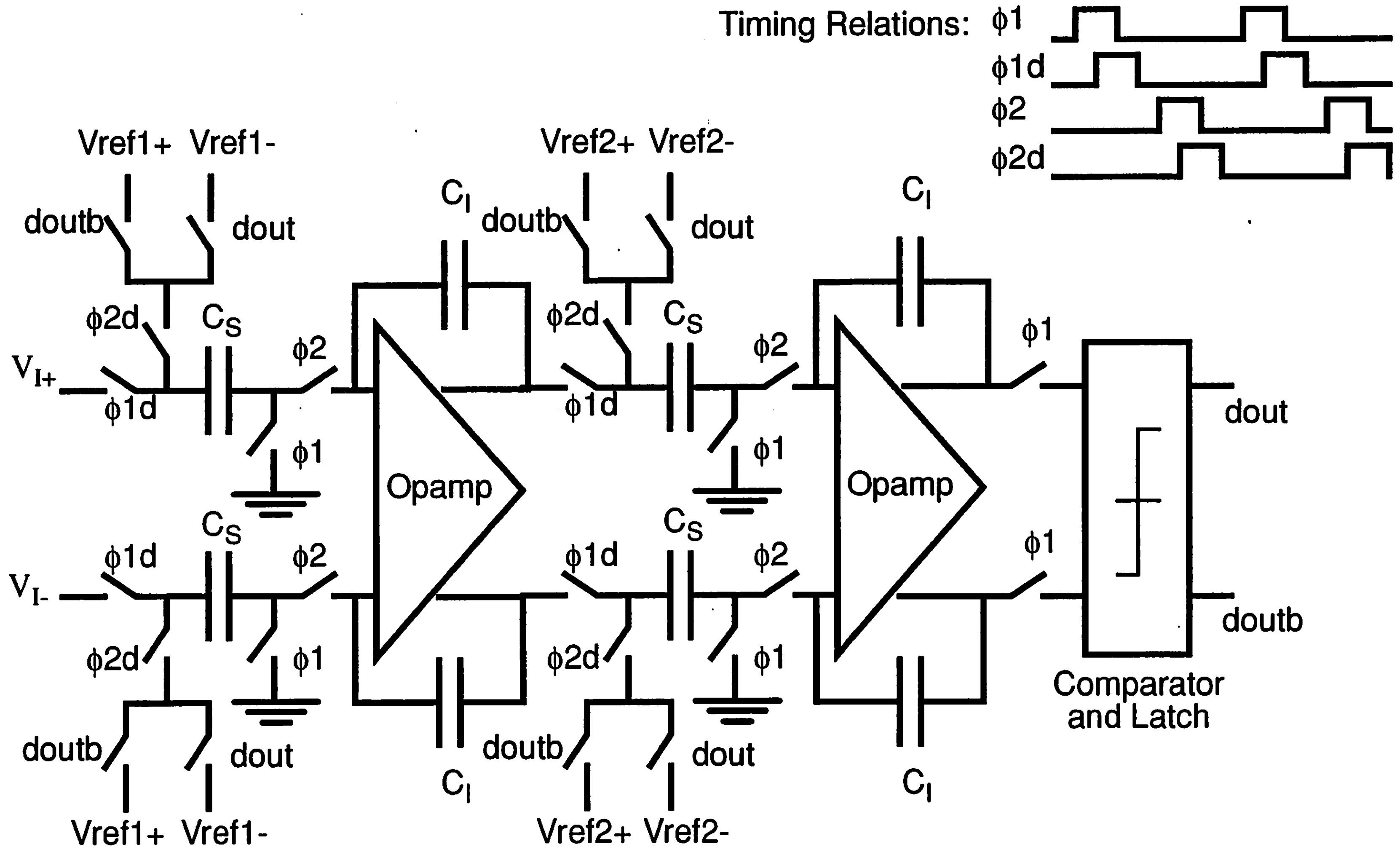


Figure 7.1 Basic diagram of a second order Δ–Σ modulator.

TABLE 7-1 Modulator variations studied in this project.

Variation	Opamp type	Capacitor Type	Process	Extra Features
1	Class AB	Double Poly	Orbit 2μm, pwell	none
2	Class AB	Double Poly	Orbit 2μm, pwell	Correlated Double Sampling Integrator
3	Folded Cascode	Double Poly	Orbit 2μm, pwell	none
4	Class AB	Metal1-Poly	VTI 2μm, nwell	none

7.1.2 Integrator Design

Differential circuits provide many advantages, including larger signal swings and better rejection of power supply fluctuations. Fully differential circuits were used in all the integrator designs since this also provides cancellation of odd order harmonic distortion. Opamp nonlinearity can lead to significant performance degradation, so single ended opamps are avoided. Simulation results have shown that large signal swings are needed by the modulators [79], on the order of twice the input signal swing. Additionally, opamp gain should be larger than the desired decimation ratio and settling will not alter behavior if it is characterized by a single pole response. Since single-stage CMOS opamps can satisfy these requirements, class AB and folded cascode opamps were chosen.

Previous results have shown that the configuration with 2 integrators of gain of 0.5 maximizes the SNR if both feedback voltages are of the same size [79]. However, by allowing the feedback voltages to vary independently, integrator gains can be changed to take advantage of voltage scaling in switched capacitor networks. This increases the complexity of the circuit design, since more reference voltages must be generated. Table 7-2 summarizes some simulation results for the case where the feedback voltage vref2, as defined in Figure 7.1, and the integrator gains are varied. These simulations were performed using an input sinusoid of fixed amplitude 40 dB down from full scale. Using different values for the integrator gains in the first and second integrators does not have much of an affect on the simulation since changing the second or inner integrator gain only affects the size of the input to the comparator. The number of design choices can be reduced by forcing them to be the same. In the simulations, the feedback voltage denoted by vref1 was held to the value 2.8 while vref1 was allowed to vary.

The data in Table 7-2 indicate that the SNR can be maximized by lowering the feedback volt-

TABLE 7-2 Values for SNR in dB as a function of feedback voltage vref2 and integrator gains for a 2nd order Δ–Σ modulator with fixed sine wave input of amplitude -40dB down from full scale.

		Values for Vref2						
		2.8	**2.5**	**2.2**	**1.9**	**1.6**	**1.3**	**1.0**
Values for Integrator Gain	0.10	34.3	35.8	37.2	39.2	41.5	44.9	50.1
	0.15	39.6	41.0	42.6	45.2	49.0	54.8	61.7
	0.20	43.8	46.1	48.3	51.6	56.9	61.2	60.7
	0.25	47.8	49.8	54.1	58.2	62.6	60.6	57.0
	0.30	52.3	56.6	61.7	62.0	60.0	62.2	12.9
	0.35	58.1	61.2	61.6	60.3	59.0	59.2	8.0
	0.40	60.9	61.9	59.5	61.6	56.6	24.9	15.5
	0.45	61.0	58.9	62.2	61.5	19.7	5.5	16.1
	0.50	60.8	61.2	62.8	59.7	3.5	3.5	-30.4
	0.55	61.5	61.8	59.6	6.2	3.8	6.1	-32.2

age vref2 and decreasing the integrator gain. The results are not very sensitive to the absolute value of the vref2, but the value should be less than the feedback voltage vref1. With lower values of integrator gain, there is increased sensitivity to the integrator gain variation since the range over which the SNR is maximized grows smaller. Other simulations have shown that lower integrator gain values allow larger input signals and more dynamic range. Using this data, it was decided to set the nominal integrator gains to 0.25, rather than the 0.5 value previously used. The sampling capacitor size was chosen to be 0.5 pF.

Integrators can also be designed to use a technique called correlated double sampling (CDS), which is a form of offset cancellation. This technique allows cancellation of DC offset and low frequency noise which limits the achievable dynamic range in switched-capacitor circuits and has been used previously in Δ–Σ modulators [127]. To implement CDS, extra switches were added to provide the offset sampling feature as shown in Figure 7.3. The charge conservation equation can be written for this integrator assuming that the opamp has a time varying offset voltage $V_{OS}[n]$.

$$C_S((V_I[n-1]-V_{OS}[n-1])-V_{OS}[n])-C_I\left(V_O[n-1]+V_{OS}\left[n-\frac{1}{2}\right]\right)=C_IV_O[n] \qquad (7.1)$$

Solving for $V_O[n]$ gives:

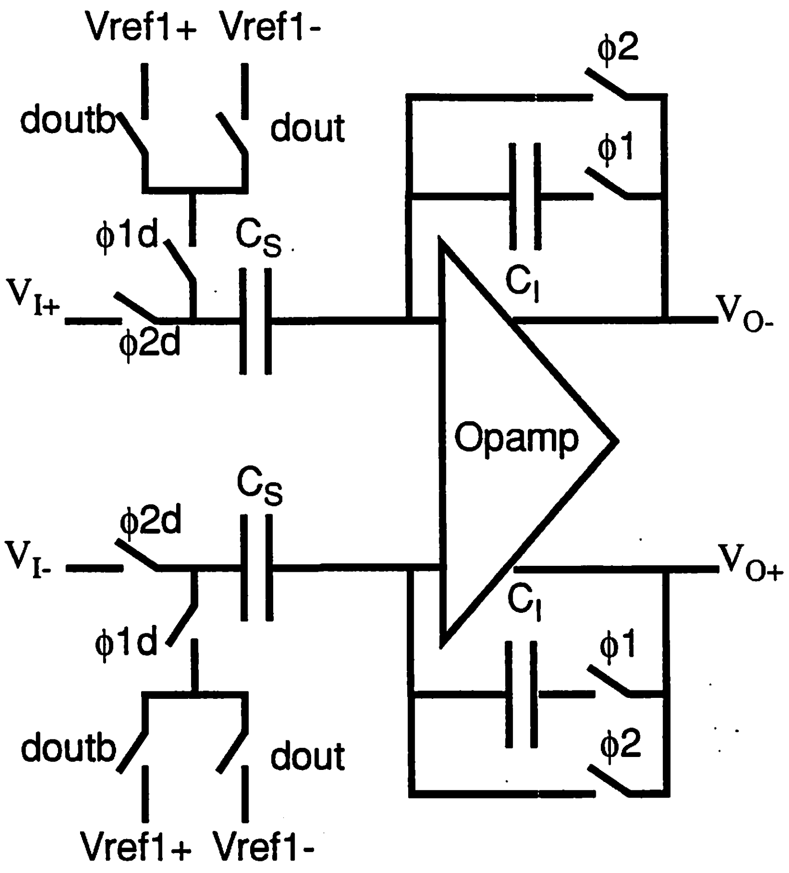


Figure 7.3 Integrator with correlated double sampling.

$$V_O[n] = V_O[n-1] - \frac{C_S}{C_I}V_I[n-1] - C_S(V_{OS}[n-1] + V_{OS}[n]) + C_IV_{OS}\left[n - \frac{1}{2}\right] \tag{7.2}$$

If the offset is not time varying and if $C_I = 2C_S$, perfect cancellation is achieved. Low frequency noise is only averaged and so cancellation isn't achieved. For the test circuit, the correlated double sampling scheme was implemented with $C_I = 4C_S$ since previous integrators had been designed with this capacitor ratio. Perfect DC cancellation is not achieved, but 1/f noise would be reduced if it were the dominating factor. In the modulator, only the first integrator was changed to the correlated double sampling configuration. The sampling clocks were changed by 1/2 clock phase for this modulator to ensure correct sampling in the second stage.

7.1.3 Opamps

In previous designs reported in the literature, both class AB and folded cascode opamps have been used in oversampling A/D converters. Class AB opamps are favored because they can pro-

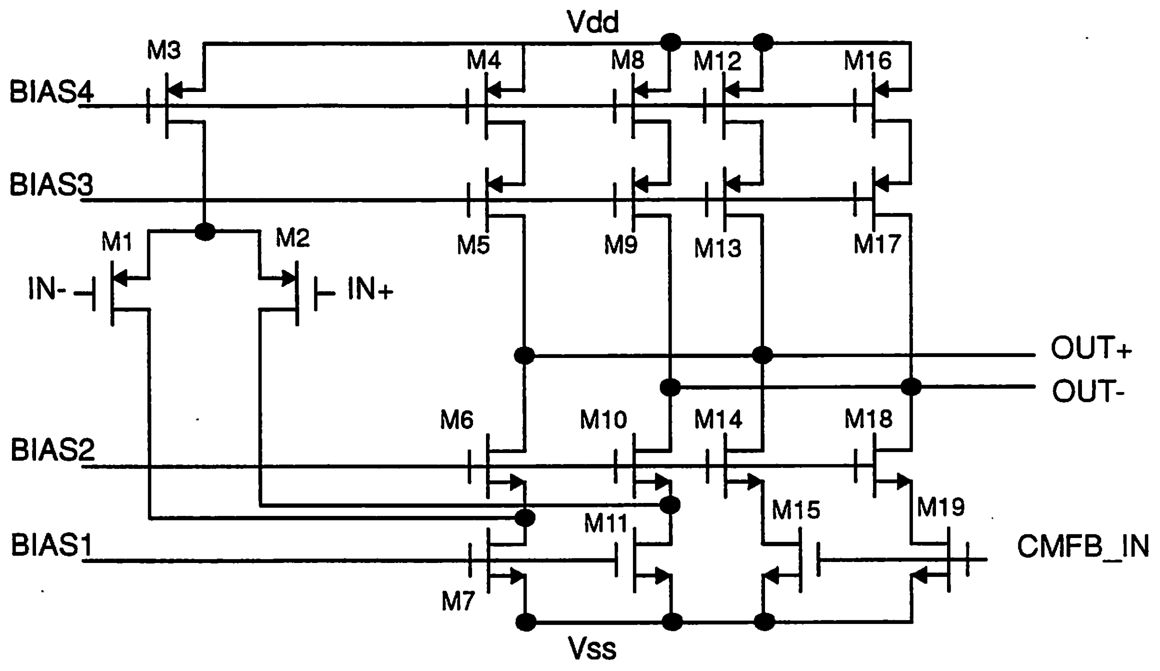


Figure 7.4 Folded Cascode Opamp.

vide large slew rates and large output swings. Folded cascode amps have fewer transistors in the signal path so they have wider bandwidths than class AB opamps. Successful oversampling A/D converters have been designed using both types of opamps. The schematic for the folded cascode opamp used in this study is shown in Figure 7.4 along with device sizes in Table 7-3. The sche-

TABLE 7-3 Device sizes for the opamp shown in Figure 7.4.

Devices	Size	Device	Size
M1, M2	438/2	M7, M11	180/3
M3	450/3	M12, M16	44/3
M4,M8	225/3	M13, M17	44/3
M5, M9	225/3	M14, M18	18/3
M6, M10	90/3	M15, M19	18/3

matic for the class AB opamp is shown in Figure 7.5 and device sizes shown in Table 7-4. It is similar to the one used by Castello [128].

A third opamp variation was designed for use in a 1.2μm process and is shown in Figure 7.6 and device sizes shown in Table 7-5. This variation of the class AB opamp uses a different configuration for the input devices. Both class AB opamps have similar input referred noise contribu-

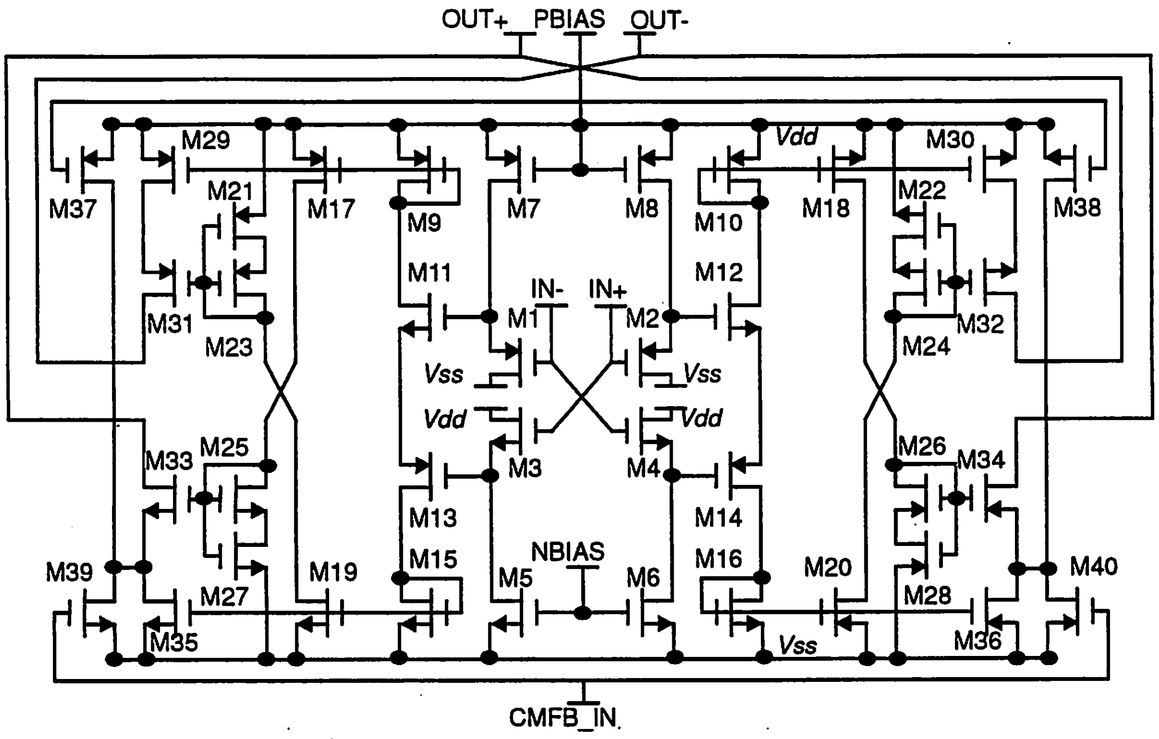


Figure 7.5 Class AB opamp, variation 1.

TABLE 7-4 Device Sizes for the Class AB opamp, variation 1.

Device	Size	Device	Size	Device	Size
M1, M2	6/2	M15, M16	10/2	M29, M30	96/2
M3, M4	6/2	M17, M18	24/2	M31, M32	96/2
M5, M6	20/4	M19, M20	10/2	M33, M34	40/2
M7, M8	40/4	M21, M22	9/2	M35, M36	40/2
M9. M10	24/2	M23, M24	9/2	M37, M38	32/4
M11, M12	16/2	M25, M26	4/2	M39, M40	32/3
M13, M14	16/2	M27, M28	4/2		

tions. The second variation allows better matching between devices in the input stage, making it more immune to power supply variations while the first variation has the advantage that the input common-mode voltage can be at half the supply voltage. For maximum dynamic range, the second variation needs to have the input common mode voltage set about a PMOS threshold voltage below half the supply voltage.

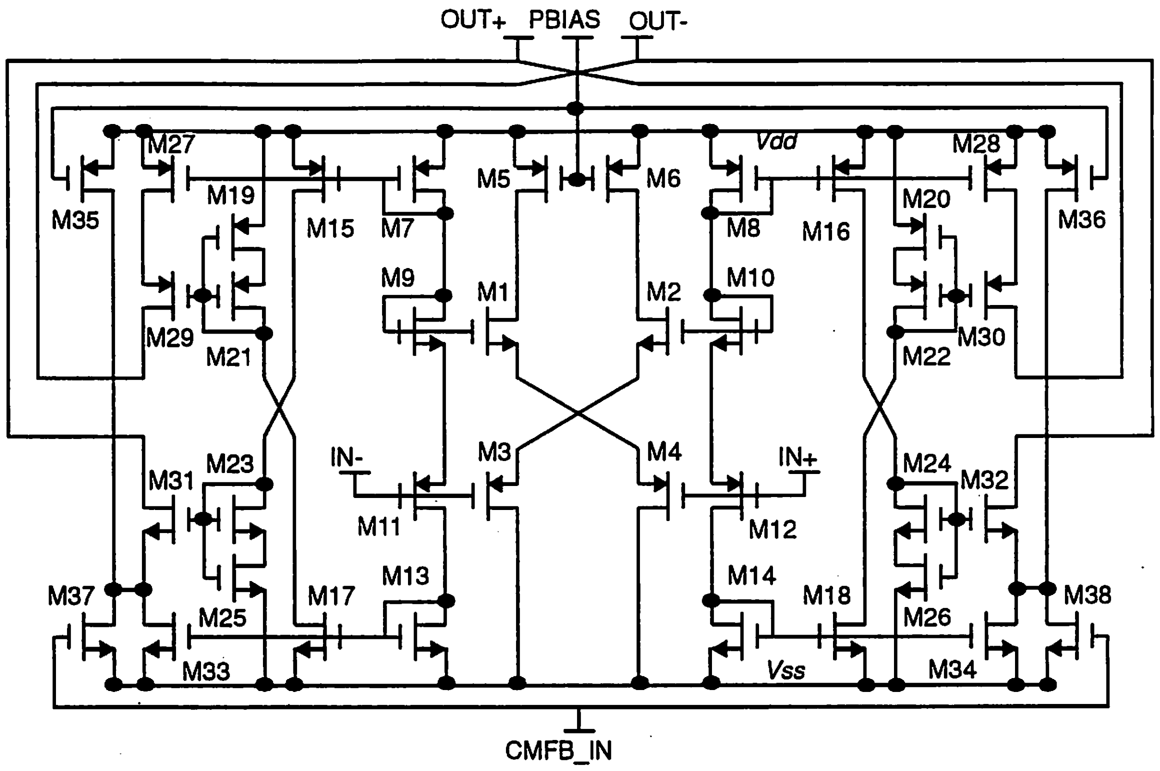


Figure 7.6 Class AB opamp, variation 2.

TABLE 7-5 Device sizes for the Class AB opamp, variation 2.

Device	Size	Device	Size	Device	Size
M1, M2	30/1.2	M15, M16	24/1.2	M29, M30	48/1.2
M3, M4	30/1.2	M17, M18	9.6/1.2	M31, M32	19.2/1.2
M4, M6	18/2.4	M19, M20	5.4/1.2	M33, M34	19.2/1.2
M7, M8	24/1.2	M21, M22	5.4/1.2	M35, M36	9.6/2.4
M9, M10	24/1.2	M23, M24	2.4/1.2	M37, M38	9.6/1.8
M11, M12	24/1.2	M25, M26	2.4/1.2		
M13, M14	9.6/1.2	M27, M28	48/1.2		

All opamps were simulated with SPICE providing the results listed in Table 7-6. For bandwidth and settling simulations, estimates were made of the loading due to routing and bottom plate capacitor parasitics. For the Class AB 1 and folded cascode opamps, capacitor parasitics were calculated assuming double poly capacitors, while metal1-poly capacitors were assumed for the Class AB 2 opamp. Device models for the simulations were taken from process information supplied by MOSIS. Additional simulations were made to characterize performance of the Class AB opamps since a wide range of variation can be tolerated for the bias current, allowing a trade-off between

TABLE 7-6 Summary of simulation results.

Parameter	Folded Cascode	Class AB 1	Class AB 2
Bias currents	275 μA	40 μA	40 μA
DC gain over full scale swing	1270 / 5.3 V	823 / 5.9 V	995 / 6.0 V
AC low freq. small signal gain	692	908	371
Small signal bandwidth	87 MHz	50 MHz	64 MHz
Phase margin	61°	40°	54°
Settling time, 0.5%	28.3 ns	61.1 ns	24.0 ns
SPICE device models	Level 2, 2.0 μm	Level 2, 2.0 μm	BSIM, 1.2 μm

gain and bandwidth. Settling time was measured by placing feedback capacitors on the opamp and then capacitively coupling a step signal into the opamp inputs. This ignores the switch resistances usually found in switched-capacitor integrators, providing an optimistic estimate of the settling time. Settling time was measured for a 1 V transition at the output of the opamp.

The common-mode feedback circuitry is based on a dynamic capacitive update scheme [129] shown in Figure 7.7. The capacitive update is applied during ϕ_2 to provide correction by the end of

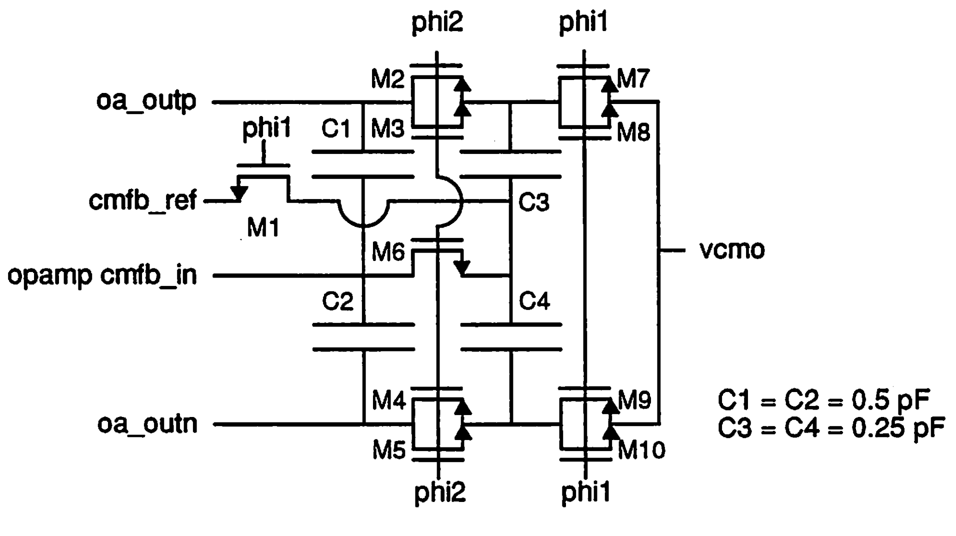


Figure 7.7 Common mode feedback circuit using capacitive updates.

the phase when the output is considered valid. A value for the reference cmfb_ref is generated using a set of ratioed devices usually included in the opamp layout. The output common mode voltage vcmo is set to analog ground.

7.1.4 Comparators

Since a relatively large offset can be tolerated, a simple comparator based on cross coupled inverters is sufficient for this application. Early designs made use of the comparator shown in Figure 7.8 along with device sizes shown in Table 7-7 while later versions of the modulator incor-

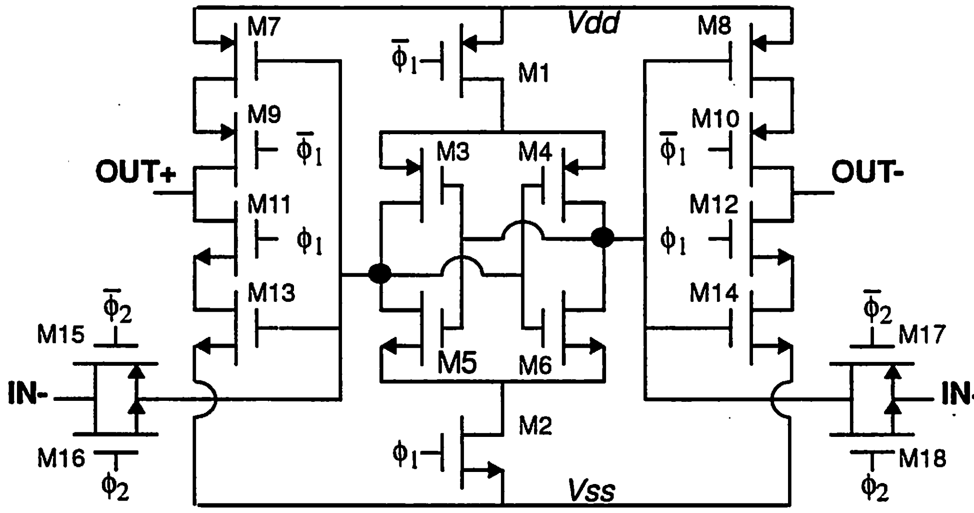


Figure 7.8 Circuit diagram for comparator variation 1.

TABLE 7-7 Device sizes for comparator variation 1.

Devices	Size	Devices	Size	Devices	Size
M1	20/3	M7, M8	6/2	M15, M17	4/2
M2	10/3	M9, M10	6/2	M16, M18	4/2
M3, M4	20/3	M11, M12	4/2		
M5, M6	10/3	M13, M14	4/2		

porated the comparator shown in Figure 7.9 along with device sizes in Table 7-8. This comparator

TABLE 7-8 Device sizes for comparator variation 2.

Device	Size	Device	Size	Device	Size
M1, M2	4.8/2.4	M7, M8	28.8/1.2	M13, M14	4.8/1.2
M3, M4	14.4/1.2	M9, M10	28.8/1.2		
M5, M6	9.6/1.2	M11, M12	9.6/1.2		

was used in both 2 μm and 1.2 μm without modification. Both comparators rely on cross coupled inverters and regeneration. The second comparator allows the source of each transistor to be con-

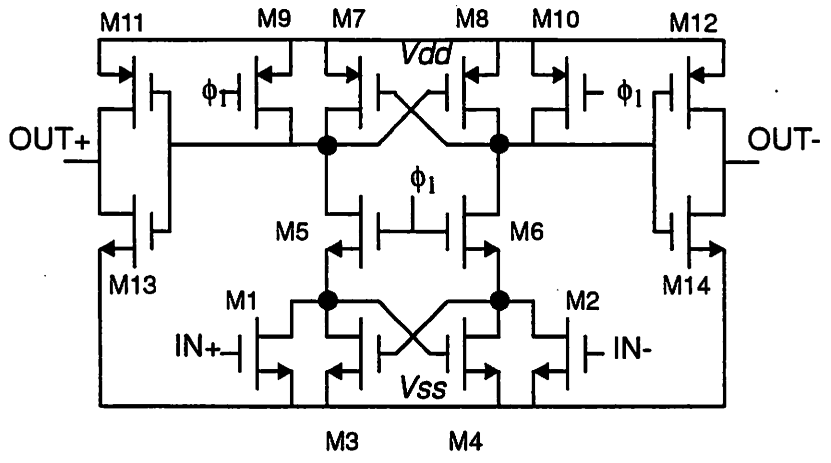


Figure 7.9 Circuit diagram for comparator, version 2.

nected to ground or the power supply. This eliminates the body effect for the transistors in the feedback configuration, allowing faster regeneration resulting in a faster comparator response [130]. Offset is also lower when body effect is lessened.

In both cases, ϕ_1 is the evaluate stage and ϕ_2 is the precharge stage. Both comparators can be designed to be much faster than the settling time of the opamps so the choice between circuits is not crucial. The decision from the comparator must be stored for use on ϕ_2 to select the reference voltage for the integrator. In early circuits, a dynamic latch was used while in later circuits, an SR latch was adopted.

7.1.5 Capacitors

Capacitor nonlinearity has been shown to cause harmonic distortion and performance degradation. Capacitor nonlinearity for most MOS processes is not well characterized, so metal1-poly and double-poly capacitor variations were used in the experimental circuits. The double-poly capacitor option is not available in all processes and involves extra processing steps and digital CMOS processes are beginning to favor a third layer of metal rather than a second poly layer. If metal1-poly capacitors provide good linearity performance, it will be more economical to use standard CMOS processes over those which supply double-poly capacitors.

In simulation, values for first order capacitor nonlinearity were set to 10 to 100 parts per million per Volt, based on numbers reported in the literature [131], [132]. Below 20 ppm/V, only small amounts of harmonic distortion are observed in simulation. These values for nonlinearity reflect the use of single-ended circuits. For differential circuits, first order nonlinearity cancels and smaller numbers should be used. No attempt was made to account for higher order capacitor nonlinearity.

The capacitance between the metal1 and poly layers is much lower than for double-poly so large amounts of area are needed to implement these capacitors. The large bottom plate parasitic loads the opamp and limits the achievable speed of modulators. In this case, Class AB opamps become more favorable since they can drive the larger capacitances. Other capacitor options do exist, such as metal1-metal2 capacitors. They may provide better linearity since both plates are matched, but the bottom plate parasitic is similar. This choice was not tried in any circuits, but results from the metal1-poly capacitors should reflect the performance achievable by metal1-metal2 capacitors. A third choice is a metal2-metal1-poly sandwich capacitor. This structure could reduce the bottom-plate parasitic and has been used in past work [133]. Layouts were developed using these capacitors, but circuits were not fabricated.

7.2 Results from the Test Circuits

Each of the modulators was measured using the experimental setup shown in Figure 7.10,

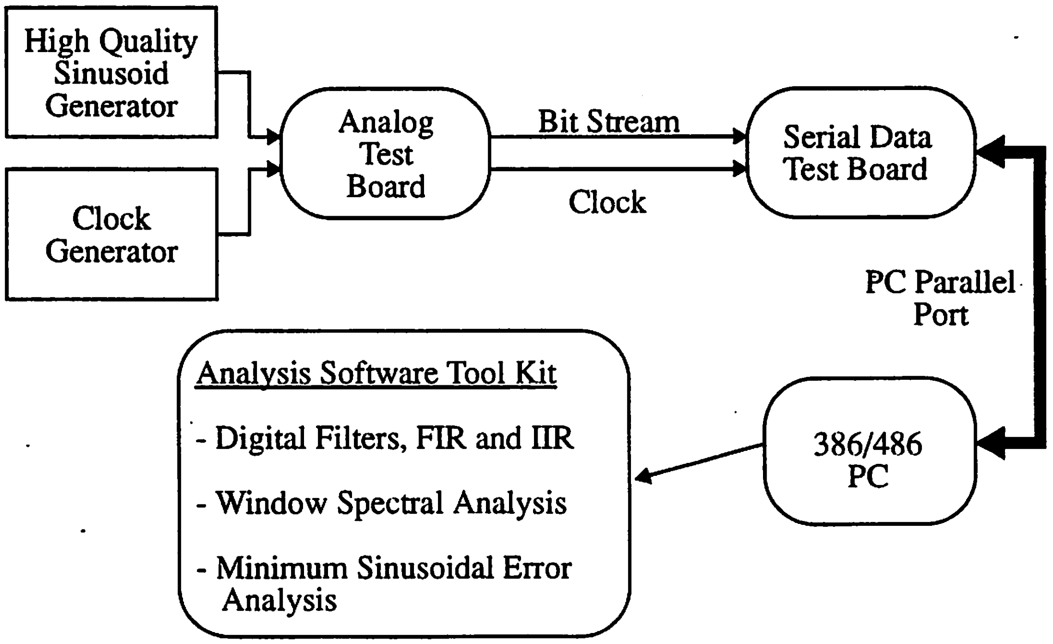

High Quality Sinusoid Generator
Clock Generator
Analog Test Board
Bit Stream
Clock
Serial Data Test Board
PC Parallel Port
386/486 PC
Analysis Software Tool Kit
- Digital Filters, FIR and IIR
- Window Spectral Analysis
- Minimum Sinusoidal Error Analysis

Figure 7.10 Block diagram of the measurement setup.

which is described in Appendix G. Care was taken to minimize extra sources of noise in the test circuit since these can greatly alter the measurement results. Measurements were made at an oversampling frequency of 1.23 MHz and the resulting bit streams were filtered in software with a sinc^3 FIR filter using a decimation ratio of 256. The SNR values were calculated using the Minimum Sinusoidal Error (MSE) method [71] on data sets of 1024 points. The sampling frequency was chosen so that all of the modulators could be compared under similar operating conditions and was limited by the modulator using correlated double sampling.

To determine dynamic range, a single set of measurements was taken on a representative chip to obtain a plot of SNR against input amplitude. The 0 dB SNR point was determined by fitting a line to the data using linear regression and extrapolating. To verify that this method was satisfactory, repeated measurements were made on a single chip. While there was variation between measurements, the deviation was not large, almost always less than 3 dB.

The measured results for the chips are found in Table 7-9. The peak Signal to Noise + Distor-

TABLE 7-9 Measured results from the test chips.

Circuit Features	measured dynamic range	simulated dynamic range	measured peak SNDR	simulated peak SNDR	measured peak SNR
Class AB opamp, double poly capacitors, pwell	82.0 dB	85.1 dB	72.9 dB	74.0 dB	82.2 dB
Class AB opamp, double poly capacitors, pwell, correlated double sampling	91.6 dB	91.1 dB	78.8 dB	78.8 dB	91.2 dB
Class AB opamp, metal1-poly capacitors, nwell	85.7 dB		77.9 dB		86.4 dB
Folded Cascode opamp, double poly capacitors, pwell	93.8 dB	97.4 dB	77.7 dB	80.1 dB	93.9 dB

tion ratio (SNDR) for the folded cascode version was limited by reduced signal swing in the integrators. When quantization effects only are included in simulation, the dynamic range in simulation is found to be about 104 dB. Comparison of the results for the various opamp combinations indicates that 1/f noise is a strong factor in the dynamic range. The high values for peak SNR indicate that harmonic distortion is the main source of degradation rather than excess noise generated at high input levels.

7.3 Comparing Circuits and Simulation Results

Measured 1/f noise data was available for the 2 μm p-well process. Hand calculations were made to estimate the total input referred noise of the various opamps. Equation 7.3 was used to model the total input referred noise density, n(f), for MOS transistors. From fabricated devices, the measured value of K_f was found to be $6x10^{-12}$ V^2pF for an NMOS device.

$$V_{eq}^2 = \frac{K_f}{WLC_{OX}f}\frac{V^2}{Hz} \qquad (7.3)$$

The resulting numbers were used to characterize simulation models developed using the techniques described in Chapter 4. Figure 7.11 shows measured data and the data from the simulation

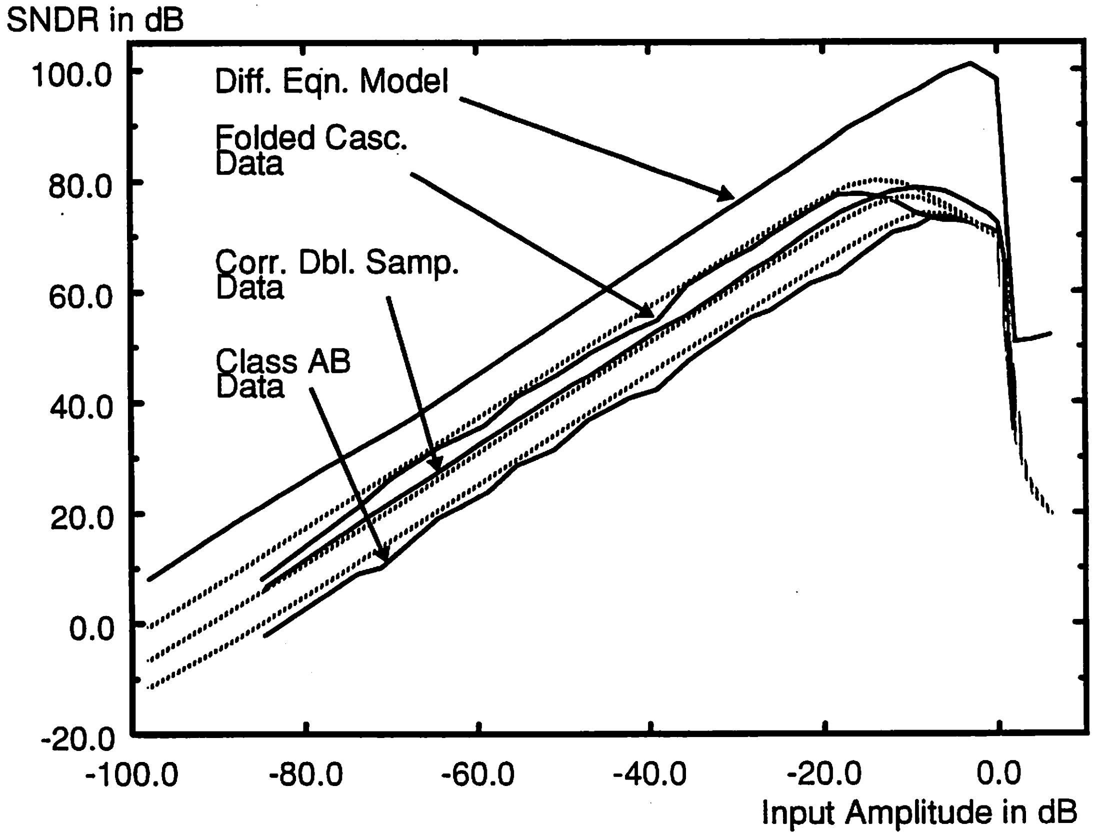


Figure 7.11 Comparison of measured data (solid lines) and results from simulation models (dotted lines).

models for SNDR as a function of input amplitude for a fixed decimation ratio of 256. The simulation model provides much better information than the basic difference equation model. Simulations match measured data to within about 3dB.

An additional nonlinear term was used to model the distortion at high input levels. Simulation values of around 300 ppm/V of nonlinearity were required to provide a good degree of matching. In examining the experimental data, it is found that most of the distortion is due to third harmonics which is expected for a differential implementation. At large input signal levels, there is a considerable amount of second harmonic distortion.

The large nonlinearity coefficient cannot be accounted for by the capacitors. Voltage coefficients for double poly capacitors are expected to be well below 100 ppm/V since both plates contain similar materials [131]. The 4 phase clocking scheme minimizes the effect of signal dependent charge injection so this is probably not the cause. Tests were run with only 2 clock phases, which should have forced charge injection from the switches to have an effect. The distortion increased but only by a small amount, but not much above that which was already present. Nonlinearity in the opamp gain function must be the probable cause of the distortion. SPICE transient simulations using these opamps and level 2 models from MOSIS did exhibit a similar amount of harmonic distortion. While low gain can be tolerated in these modulators, these numbers indicate that low gain must be achieved with good linearity and low variation over the full scale range.

Parameters used in the simulation model are listed in Table 7-10. Excellent matching was

TABLE 7-10 Parameters used in the simulations.

Parameter	Class AB Model	Class AB Model with CDS	Folded Cascode Model
Input standard deviation for the 1/f noise generator	0.0000920	0.0000920	0.0000208
Standard deviation for additive white noise	0.0001287	0.0001287	0.0001287
Combined opamp/capacitor nonlinearity	325 ppm/V	325 ppm/V	500 ppm/V
Open loop opamp gain	1000	1000	1000
Positive opamp saturation level	3.5 V	3.5 V	3.5 V
Outer Feedback Voltage	2.8 V	2.8 V	2.8 V
Inner Feedback Voltage	1.8 V	1.8 V	1.8 V
Time Constants allowed for opamp settling	4	4	4
Integrator Gain	0.25	0.25	0.25

achieved between circuits and simulation models. Even in the correlated double sampling case, the simulation was still able to predict the noise averaging effects. These results show that the presence of 1/f noise dominated the circuit performance in actual circuit implementations. However, dynamic range on the order of 15 bits still can be achieved in these circumstances without the need for trimming or special circuit design techniques. Additionally, there is no evidence to suggest that the nonlinearity of metal1-poly capacitors is a problem. This shows that relatively high resolution converters can be integrated in standard digital CMOS processes. However, 1/f noise must be characterized to allow simulations and to predict worst case performance.

Figure 7.12 shows results for various decimation ratios for the first circuit listed in Table 7-9,

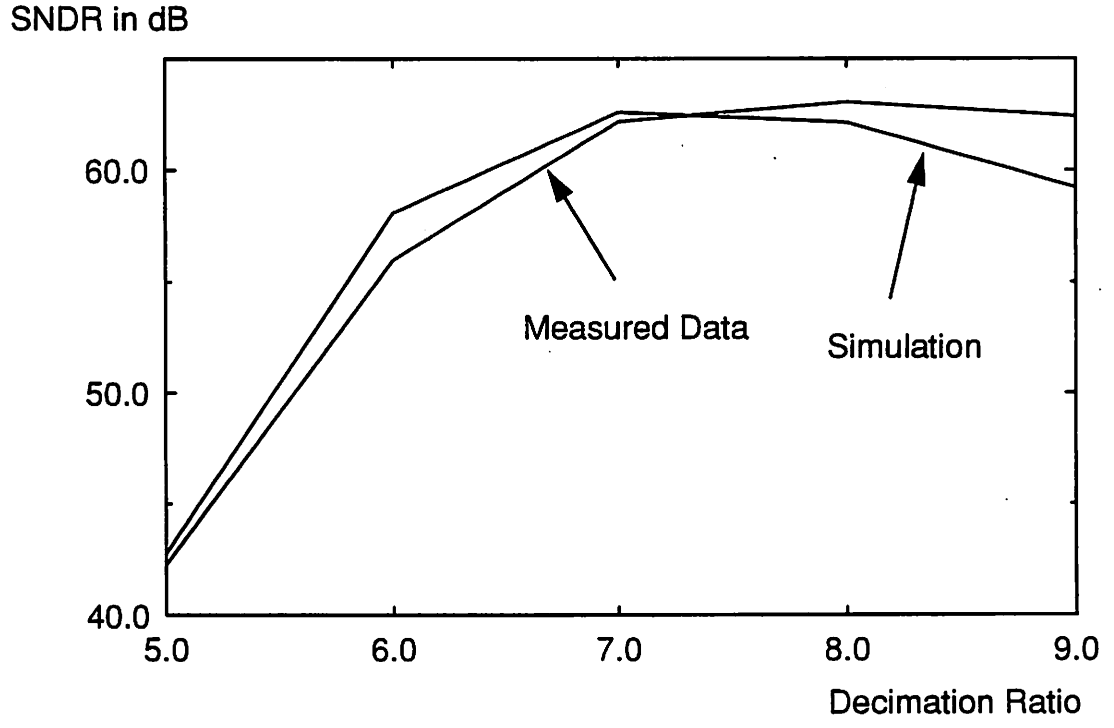
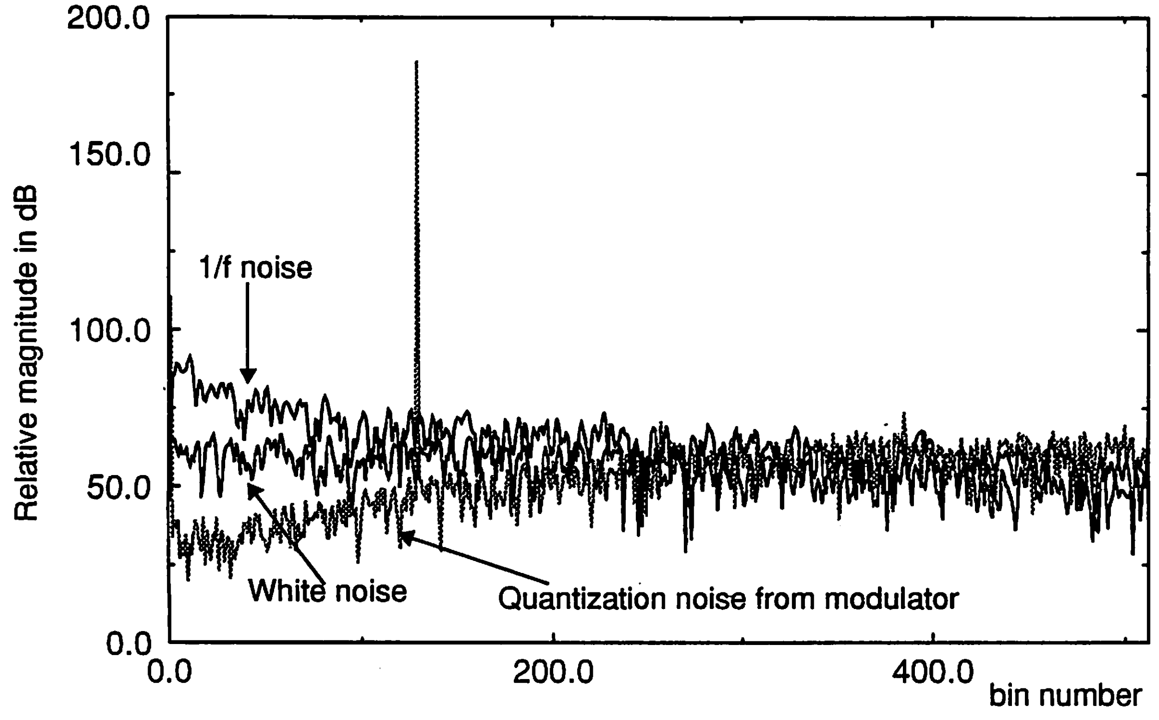


Figure 7.12 Comparison of simulation and measured results for changes in decimation ratio. Input is a sinusoid, 20 dB down from full scale.

with fixed input amplitude of 20 dB down from full scale. The simulation model predicts the decrease in performance that actual modulators experience when large decimation ratios are used. As the oversampling ratio is increased, the residual 1/f noise and white noise from the amplifiers and the switches limits the performance of the modulator. Accurate simulation of this degradation is necessary so designers can improve the circuits used in high resolution conversion.

FFT plots can be used to illustrate the degradation and also to show the degree of matching between circuits and simulation. The top plot in Figure 7.13 shows the relative magnitudes of



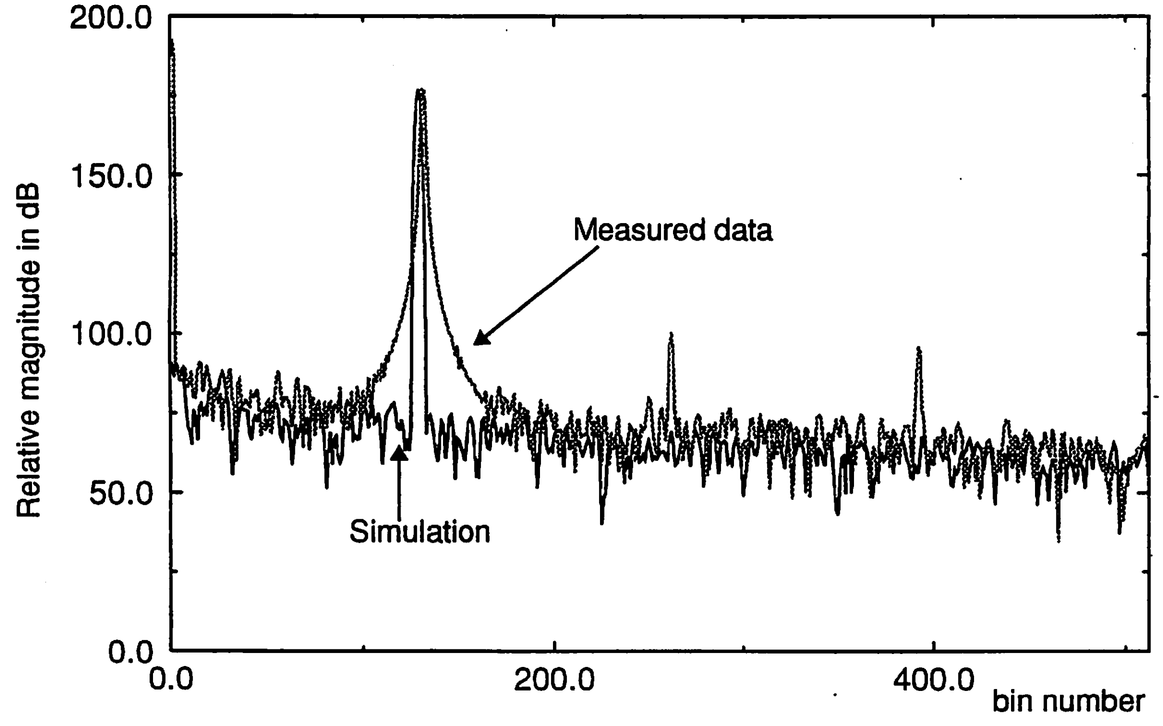


Figure 7.13 FFT plots of simulated and measured data. The top plot shows the ideal response from a modulator, along with white noise and 1/f noise contributions. The bottom plot shows the composite simulation along with an FFT plot from taken from one of the test circuits.

white noise, 1/f noise, and quantization noise for a second order Δ–Σ modulator, with output decimated by 256 using a sinc^3 filter. The simulations assume that the size of the sampling capacitor is 0.5 pF. The 1/f noise contribution is clearly larger than the other two. However, even if the 1/f contribution were removed, the white noise contribution is still dominant. Decimating by a factor of 2 would only cut the noise in half, leading to a 3 dB gain at best. In the lower half of Figure 7.13, a plot is shown comparing actual data from a chip and the simulation model combining all the noise sources. The chip data shows harmonic distortion, but the matching for the noise floor is quite good.

These results show that modelling techniques for oversampling A/D converters can provide accurate results that match circuits. Issues such as the choice of opamp, capacitors, capacitor ratios, or comparator have little effect on the overall design since 1/f and white noise dominate the results. These results point to the fact that very high resolution conversion can be achieved if chopper stabilization techniques are adopted to minimize 1/f noise. These results were derived from second order Δ–Σ modulators, but the simulation techniques can be applied to the design of higher order modulators. Since circuits and simulation models have been shown to correlate, a future step is to derive module generators for modulators to allow truly automated generation of ASIC oversampling A/D converters.

7.4 Module Generators for Oversampling Modulators

While oversampling modulators are tolerant to process variations, they still must be redesigned when implemented in a new process technology. Analog design is time consuming and can be the bottleneck in mixed signal designs. Module generators would allow designers to quickly implement new modulators, avoiding the need to rely on the choices presented by a fixed cell library. This would encourage even more design exploration to find trade-offs for interfaces comprised of low and high order modulators. As mentioned in chapter 1, there has been recent progress in the development of analog layout generation tools. Based on the design experience presented in this chapter, accurate module generators for the analog loops could be developed. The next sections discuss how to accomplish this.

7.4.1 Developing a Module Generator

Developing module generators for analog oversampling modulators should not be a difficult task. A module generator for the basic integrator configuration has already been developed for use in automatic layout generation for algorithmic A/D converters [9]. The current integrator synthesis routines allow for optimization of opamp properties like bandwidth, gain, and power. Simulation results for oversampling A/D converters have provided rules of thumb for opamp gain and settling time. For oversampling modulators, it is known that opamp swing should be maximized and input referred 1/f noise should be minimized. To handle the design of oversampling modulators, the synthesis routines for the opamps could must modified to optimize these additional parameters.

Synthesis and place-and-route tools are needed to integrate the module generators for integrators, voltage references, and comparators. The synthesis routines need to analyze the given specifications and then translate the specifications to requirements on the integrators and comparators. In addition to this, a generalized module generator will need to be able to synthesize several topologies for the modulator. There are several different problems when synthesizing each modulator topology, and these were briefly discussed in the previous.

It is not clear how this type of module generator should be used in a design system. A large amount of knowledge is needed to successfully run most module generators. Often, some effort is required to touch up or compact the layouts generated by these module generators. One of the attractive features of oversampling modulators is that a single design can be used with different decimation filters to give different resolutions. It seems reasonable to encourage reuse rather than module generation. The best choice will probably be to include optional module generators in the design system. If libraries need to be designed or if expert users want to experiment with new topologies or higher performance, they can make use of the module generators. However, shorter design cycles will be available to those who use library based cells.

Selecting a good topology for design is much more difficult than implementing a module. While a common layout generation and opamp synthesis program could be used, a general module generator would need different high-level design tools. The MASH and Δ–Σ topologies can make use of basic template designs. However, the modulators with higher order loop filters require a coefficient design program to find the placements of the poles and zeroes, and the proper capacitor sizes. In addition, tools that can handle the complex routing and capacitor sizing are needed. There are also the issues of stability and component requirements to consider.

7.4.2 Module Generation and Topology Choice

The choice of topology is influenced by the application. The basic strategy is as follows. First select a low order modulator and attempt to increase the decimation ratio to meet performance requirements. If this can't be done, then increase the order of the modulator. Increasing the decimation ratio does not always bring higher resolution, and these effects can be studied. Opamp nonidealities, sampling noise, and 1/f noise all limit performance so that practical decimation ratios are less than 500 for 5 Volt systems. The use of high decimation ratios provides a good system trade-off, since this provides more cycles for digital signal processing. However, this approach does not work for very high speed conversion.

For higher sampling rate applications, there is no alternative but to use higher order modulators or modulators with multi-bit A/D and D/A converters. Higher order modulators are often reported in the literature, but emphasis is placed on the analog portion. While lower decimation ratios may be used for a given resolution, higher order digital filters are required to suppress the additional noise. These filters require more hardware and power. This trade-off is important since well over 80% of the chip area usually is required for digital filtering.

While module generators can probably be easily developed, it isn't clear that they will be helpful in the design automation process. In the end, designers will tend to rely on methods that they are comfortable with. Unconditional stability at the expense of both analog and digital circuit complexity is available with multistage modulators. Simple circuits with high linearity, but with unconditional stability are available from the modulators with higher order loop filter. Stability and simple circuits can be achieved with first and second order Δ–Σ modulators, but this comes with spurious tones and high decimation ratios. No perfect solution exists, and good designs will require more than a module generator to balance these conflicting demands.

7.5 Layout Techniques for Oversampling Modulators

While module generation would have aided the rapid prototyping of modulators, custom layout was used in this project. A modular design style was developed to simplify the layout tasks [78]. The modulator was partitioned into integrators, a comparator, and a latch. The integrator was further partitioned into sections for switches, capacitors, and the opamp.

Opamps were designed to be symmetric with common centroid geometry. The layout for the Class AB 2 opamp is shown in Figure 7.14. All devices were surrounded with guard rings to pro-

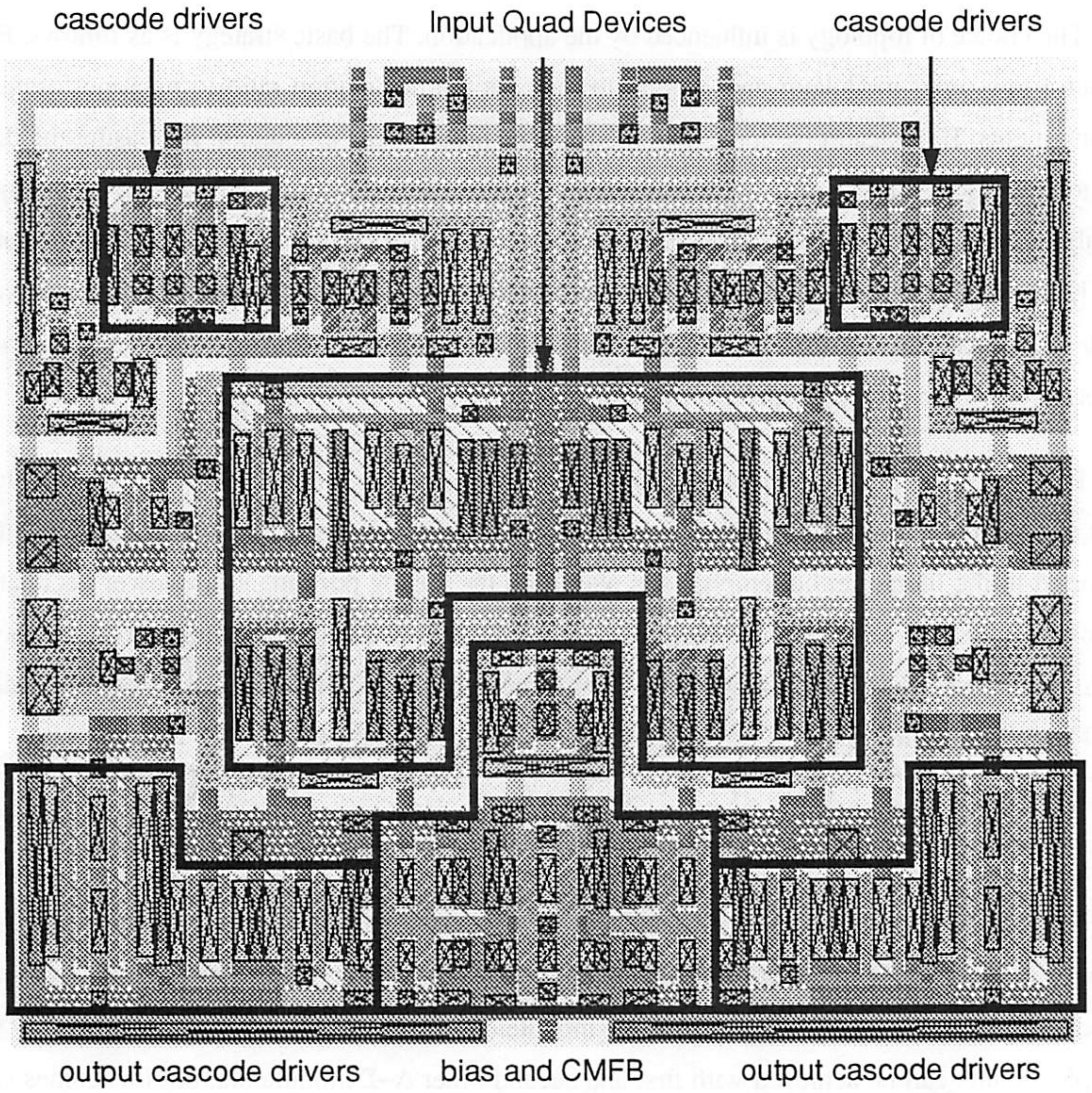


Figure 7.14 Layout of the Class AB 2 opamp.

vide isolation. The width of the layout was set by the size of the capacitor array, while the height could be varied arbitrarily. A layout showing both metal1-poly and double-poly capacitors is shown in Figure 7.15. Both arrays in the figure implement 0.5 pF sampling caps, 2.0 pF integrator caps, and the 2 common mode feedback caps. The size discrepancy shows why metal1-poly caps have such a large bottom plate parasitic. The basic switch array for an integrator is shown in Figure 7.16. The top routing channel is used for clocks and other digital signals while the lower channel is used for analog signals. Both channels have a poly shield beneath them to minimize coupling of signals to the substrate. The switches are routed so that no analog and digital lines

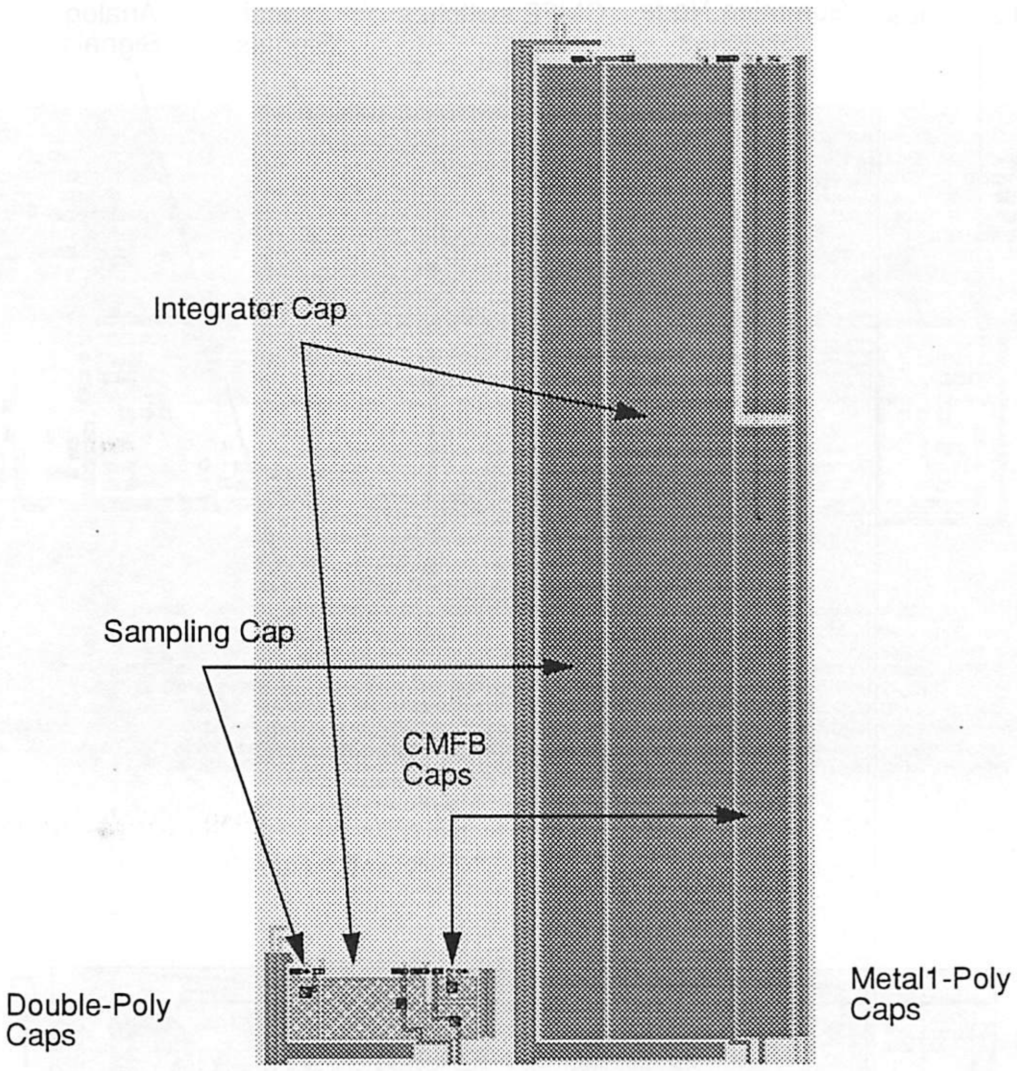


Figure 7.15 Layout for double-poly and metal1-poly capacitors.

cross. The switch array is composed of 2 symmetric blocks containing the input and summing node switches and a third block containing the cmfb switches. Integrator structures are created by tiling the 3 modules together.

For a full modulator, a comparator is needed for the feedback loop. The layout for the second comparator variation is shown in Figure 7.17. A full first-order Δ–Σ modulator can be assembled by tiling all the components to form an integrator in a feedback loop. Layout for a full modulator is shown in Figure 7.18. The distribution tape for the design system provides the various modules used in the project along with examples for tiling the modules into complete modulators with orders from 1 to 3.

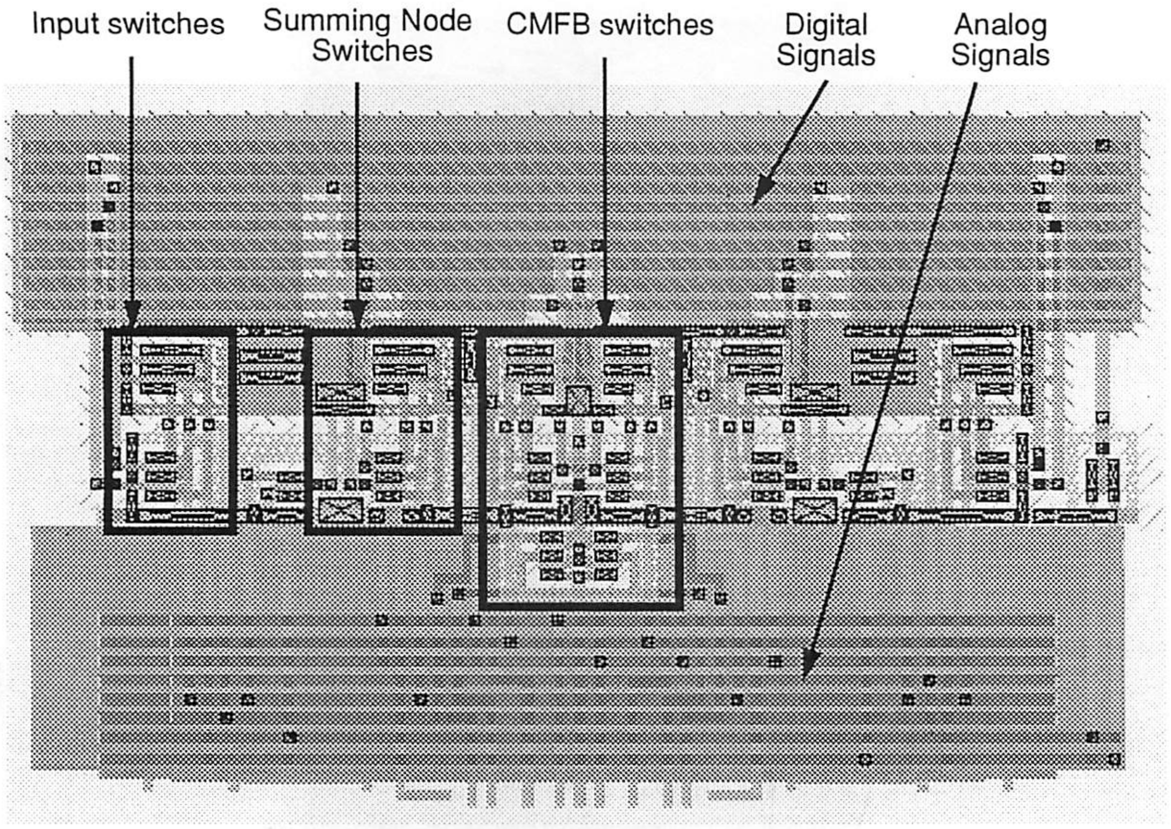


Figure 7.16 Layout for the integrator switch array.

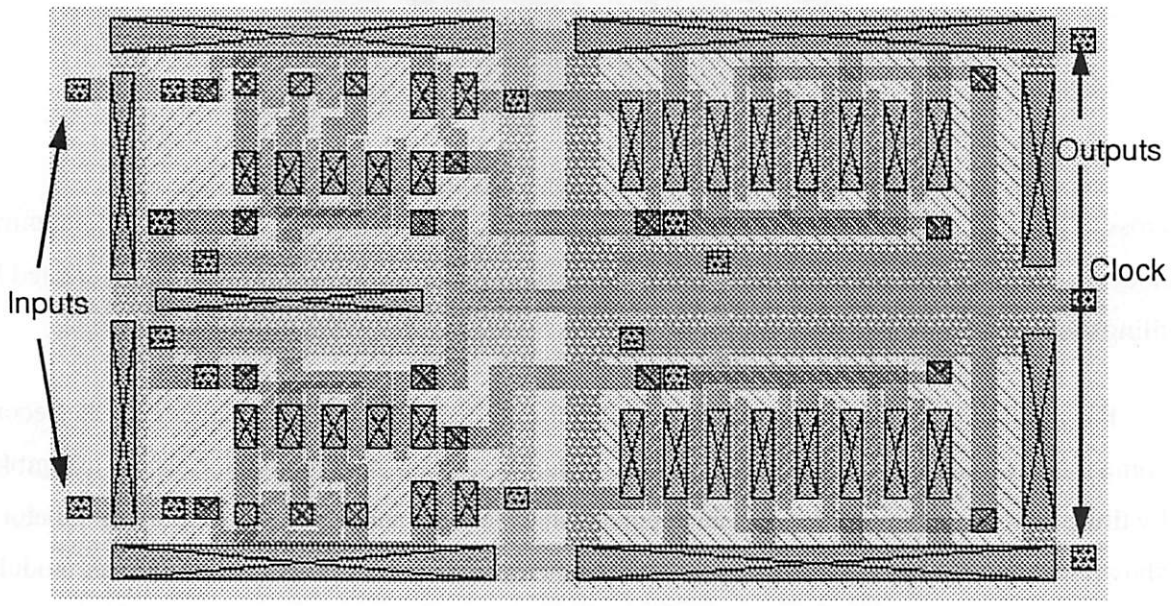


Figure 7.17 Layout for the second comparator variation.

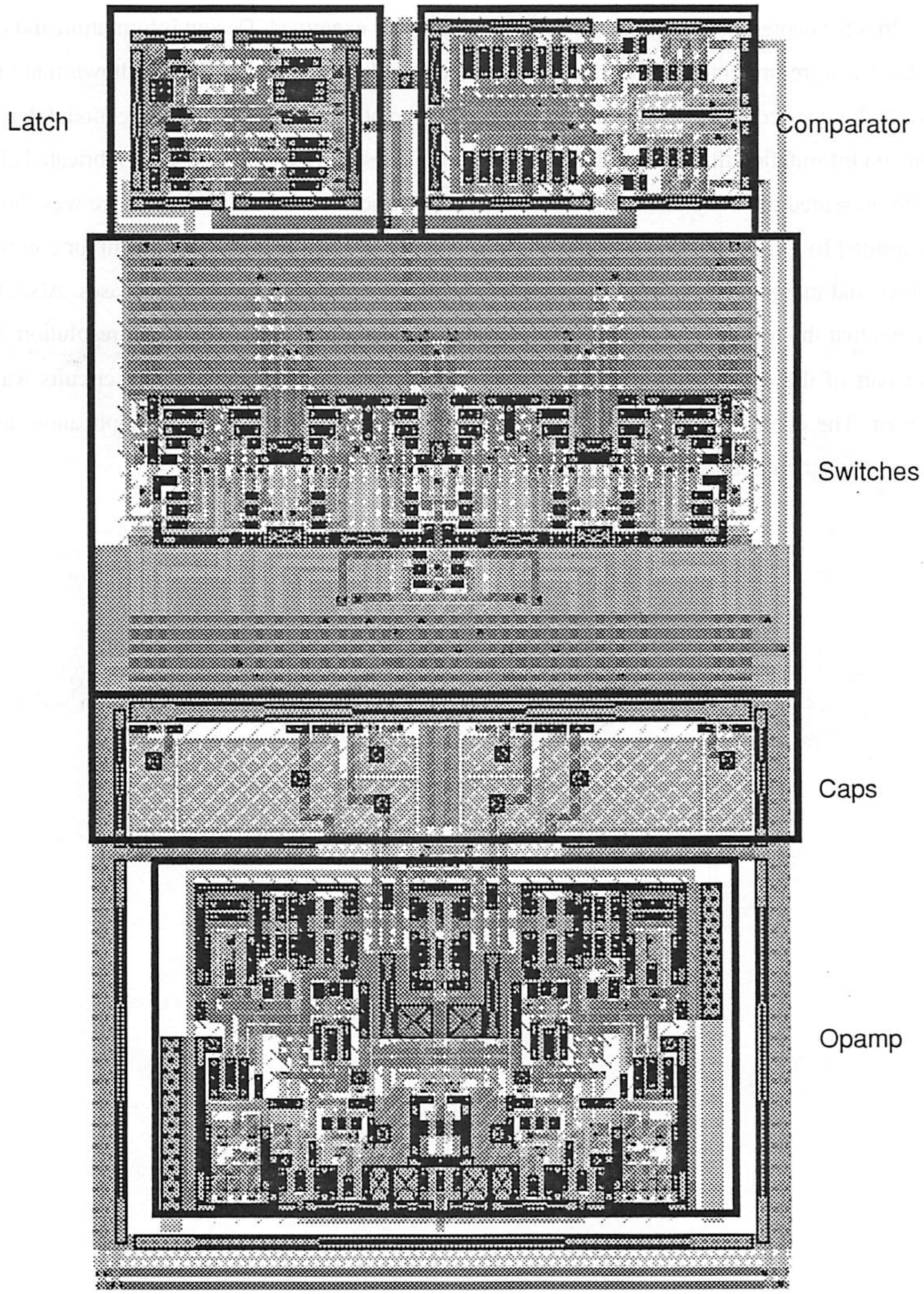


Figure 7.18 Layout for a first-order modulator.

7.6 Summary

In this chapter, the design of the analog circuits was examined. Design information and circuit diagrams were presented to document the cells currently in the library. It was shown that modelling techniques could be used to accurately predict the performance of the analog modulator based only on information that can be obtained from normal test devices. Results from fabricated circuits were presented to verify that the models work for a variety of circuits. Performance was shown to be limited by excess noise associated with the opamps. Design features like opamp or comparator choice and integrator gain were shown to be secondary considerations in these cases. Also, it was shown that the linearity of metal1-poly capacitors is not a problem up to 15 bit resolution. In the last part of the chapter, the development of module generators for the analog circuits was discussed. The choice of modulator was found to be highly dependent on the application and the entire implementation should be considered when making the choice.

CHAPTER 8

Design Examples

Several design examples are examined in this chapter. The examples progress in complexity and mirror the advances made during the project. As more sophisticated tools were developed, better analyses could be performed. In the discussion, emphasis is placed on describing the design process and what trade-offs were considered.

8.1 Analysis Principles for Oversampling A/D Converter Design

Before discussing design examples, a brief review of applications for oversampling converters will be given. These converters are not memoryless. They must be run for a period of time before the output samples are fully up to date which means that a single analog modulator cannot be time-multiplexed among several channels unless ample time is provided for transients to settle out. In data acquisition applications, bandlimiting is provided for free when linear filters are used for the decimation and averaging to increase resolution. Oversampling converters must use oversampling which limits them to applications where the final sampling rate is many times slower than the achievable gate delay in the process. At present, video rate converters seem to be out of the question.

While oversampling converters may not be useful for the fastest applications, they are quite useful for high resolution applications. The 1 bit A/D and D/A converters used in the modulators are by definition the highest linearity that can be achieved. With careful design, this fact can be exploited to give resolutions greater than 20 bits [31], [134]. For modulators that use multi-bit A/D and D/A converters, oversampling and noise-shaping can provide increases in resolution beyond the limitations of matching in a process.

In designing oversampling A/D converters, the basic design principle is the trade-off of resolution for speed. Simple but accurate models can be developed to predict the amount of resolution possible for a given oversampling ratio for a variety of modulator configurations. This can be based on simulation results or on process information, as was shown earlier. Basic circuit techniques can be used to predict the maximum sampling rate that a modulator can achieve in a given technology [135]. Once this is done, the design process can be simply stated as choosing a modulator, an oversampling ratio, and a set of digital filters that minimize the loss of resolution due to aliasing.

The variety of modulator topologies and digital filtering options clearly complicates the initial topology decision. Higher order modulators provide better noise shaping, and thus more resolution, at a lower oversampling ratio. However, the increased noise shaping requires higher order digital filters to suppress the out of band noise. Higher order filters may increase area, but area could be reduced by time-multiplexing operations and running the digital filters at higher rates than the analog loop. This trade-off might not be favorable due to the increase in power dissipation due to the overhead of sharing circuitry. As discussed in chapter 4, there are complex trade-offs for digital decimation filter design between speed, power, area, and the amount of noise aliased back in band.

8.2 Example 1: Data Acquisition

8.2.1 Application

Data acquisition is the process of collecting data points from a slowly varying signal. Measurements are not necessarily made in a periodic time fashion. Since the input signal varies slowly and since relatively large amounts of time are available, high resolution is usually desired. The solution to this problem has been to use an integrating A/D converter with a precision current source. However, resolution is limited by mismatches and very high resolutions are obtained through trimming.

Oversampling A/D converters are attractive because they can achieve high resolution without the need for trimming. Since there is quite a bit of time for conversion, large oversampling ratios are not a problem. Also, relatively simple, low order modulators provide the resolution without having to worry about instability in higher order loops. A critical problem is the interference of noise sources, such as 60 Hz AC power noise. Since a digital decimation filter is a necessity, it can be designed to provide suppression at these critical frequencies, improving the performance of the overall system.

8.2.2 Requirements

The first project undertaken with the design system was the design of fully integrated data acquisition IC. The goal was to investigate some of the cross talk issues and to verify the functionality of the designs. In the actual design of a data acquisition converter, several specifications need to be considered.

The choice of the digital filter is not very difficult. Since there is interest in DC signals, the passband of the filter only has to pass DC without attenuation and there are not other passband ripple requirements. For stopband attenuation, the main goal is to limit the amount of noise aliasing back in band from the noise shaping process and to attenuate any out of band signals mixed with the input. The filtering function can easily be satisfied by a sinc filter.

For the modulator, a second order Δ–Σ modulator provides good resolution without stability problems. There is some concern about the noise tones created by the nonlinear behavior of the loop, but these should be minor since circuit noise will provide enough dithering to randomize the patterns. For very high resolution applications, the first integrator should be chopper stabilized or auto-zeroed to minimize the effects of 1/f noise. Capacitors can be made quite large to reduce switch noise. In this case, an extra dithering source might be considered to randomize the limit-cycle patterns.

The data acquisition system should try to minimize area and power but this can be a tricky proposition. Bit-serial circuits seem favorable at these speeds, since they are small but they must be run at significantly higher clock rates, which may make the power trade-off unfavorable.

8.2.3 Implementation

No real specifications were created for this project and add-hoc methods were used in the optimization process. The standard second Δ–Σ modulator with a Class AB opamp was used. Measured data showed that the modulator would provide resolution gains up to decimation ratios of about 128. After this point, the 1/f noise of the first opamp dominated and further decimation would only provide marginal gains in performance.

The FIR filter with coefficients implementing a $sinc^3$ filter was chosen for decimation. The coefficients were not rounded since rounding adds noise to the response and can diminish the overall converter resolution. The CIC comb filter was shown to require about the same area for decimation ratios of 64 to 128, and was not a viable candidate at the time since it was still being developed. A decimation ratio of 64 was chosen, since it allowed the resulting chip to fit into the cavity of a 40 pin DIP package when the chip was implemented in a 2 μm process. If further decimation was required, it could be implemented off chip in subsequent processing. This is a feature found in most commercial data acquisition chips based on Δ–Σ converters. A parallel to serial converter was added so data could be collected using the standard test setup.

During layout generation, some care was taken to isolate the analog portion of the circuit. While the place-and-route tool used a true digital router, clever placement still allowed definition of channels dedicated to analog signals. No digital and analog signals were allowed to cross and some distance was allowed between the analog part and the rest of the digital circuitry.

8.2.4 Verification Strategy

To verify the circuit, extensive simulations were performed using IRSIM. The entire chip was extracted, including the pads. The circuit was designed with a multiplexer to allow testing of the digital filter independent of the analog modulator. In the simulation, the modulator does not interfere since the input to the digital filter was tied high when the simulation was run. It was verified that the output sum for the FIR filter was correct and that the data valid pulses generated by the filter were periodic and equally spaced in time. In addition, extensive simulations were run on the scan path to verify functionality and to generate test vectors.

8.2.5 Results

The chip layout occupied 5.5 x 4.5 mm^2 with pads and it runs off a single 5 V supply. It was fabricated in a double poly 2 μm process. Figure 8.1 shows the die photo of the chip. The chip was found to be functional during early testing so the scan test mode was never exercised.

On this chip, it was possible to disable the digital circuits without affecting the analog circuits allowing examination of crosstalk effects. Measurements were made with the filtering performed

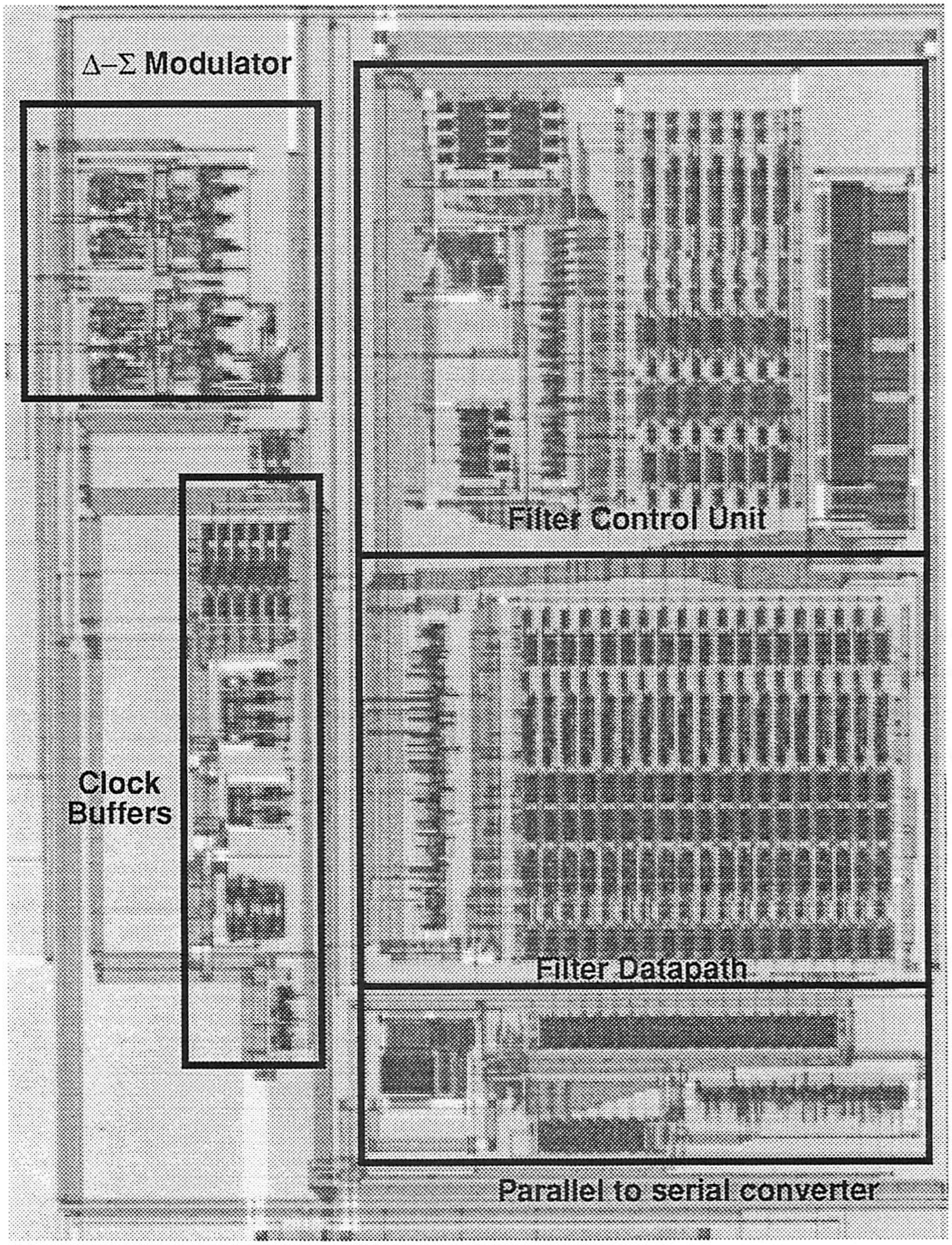


Figure 8.1 Die photo of the chip for data acquisition.

in software and with the on-chip filter. The SNR curves are shown in Figure 8.2 for the case where

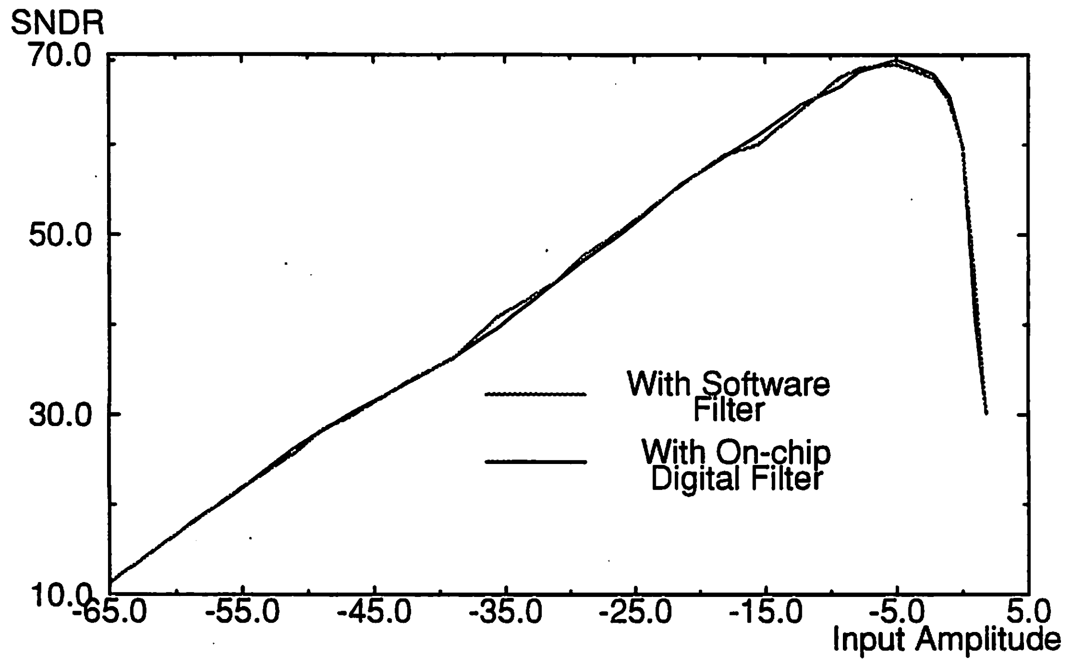


Figure 8.2 Comparison of SNR curves when the filtering is performed on-chip and off-chip.

the modulator is sampling at 1.23 MHz. It can be seen that there is relatively little difference between the 2 cases so crosstalk is not a major problem. However, it was noticed that extra tones can be seen in the spectrum of the output of the chip. These tones are small in magnitude and contribute at most 1 dB of performance degradation. The probable cause of these tones is clock feedthrough from the clock buffers. As seen in Figure 8.1, the clock buffers were placed relatively close to the analog modulator.

Table 8-1 summarizes chip performance. A large portion of the power consumption is due to

TABLE 8-1 Summary of Results for the data acquisition test chip with decimation ratio 64.

Parameter	**Value**
Technology	Double Poly 2μm CMOS
Area of the chip core	3.6 x 3.0 mm^2
Extrapolated dyn. range @ f_s=1.23 MHz	74.9 dB
Measure Peak SDNR @ f_s=1.23 MHz	69.4 dB
Maximum Output Sampling Rate	125 kHz

the digital portion of the chip. To investigate the possibilities, measurements were made with reduced supply voltages. The critical path of the digital filter is quite small and is not limited by the current sampling rate. This means that extra speed is available and lowering the supply will not cause a circuit failure. Table 8-2 shows a plot of measured power for different supplies. Power

TABLE 8-2 Measured power dissipation. Output data rate was 52 kHz.

Supply Voltage	Digital Power	Analog Power
5.0 V	67.8 mW	4.1 mW
4.5 V	52.7 mW	3.1 mW
4.0 V	38.2 mW	2.2 mW
3.5 V	27.2 mW	1.5 mW

consumption can be decreased significantly by lowering the supply on the digital portion. Analog dynamic range is lost when lowering the supplies, so a separate analog power supply could be used for these circuits. Calculations have shown that over half the digital power is dissipated in the PLA structures which use pseudo static circuits. These figures show that low power A/D conversion should be easily implemented. Digital power can be reduced so that it is the same order of magnitude as the analog power.

Since this chip was done in the early development stages of the design system, design time was relatively long, about a 2 month design cycle for a single designer. Most of the time was spent in developing the digital filter architecture and in simulation and verification. The modulator had been developed previously.

8.3 Example 2: Linear Phase A/D converter

8.3.1 Background

In some cases, linear phase is required to maintain signal characteristics. An all FIR filter implementation of the decimation filters is necessary to achieve this. Design programs are readily available for FIR filters and the coefficient rounding problems are not difficult to handle. This makes them conceptually easy to deal with and implement. However, they can consume an enormous amount of resources, as will be shown in this example.

The all FIR case is not always a bad choice since FIR filters can be competitive with IIR filters when the transition region is large. Also, for the case where phase response is important, FIR filters may provide an advantage. In terms of complexity, an IIR filter with phase equalization may require just as many resources as an FIR filter for the same specifications. For this reason, it is important to support the design of FIR filters for all stages of filtering.

The DECGEN compiler was developed to address this class of problems [51]. An A/D converter was designed to verify the design of the compiler and the supporting circuits. The specifications for the circuit were for a 12 bit converter with 72 dB of stopband attenuation. The specification is generic and can be used for speech coding. To reduce die area, the total decimation ratio was limited to 64, but the circuits were designed to function with an output sampling rate of in excess of 150 kHz. The basic frequency domain specification is shown in Table 8-4.

TABLE 8-3 Specifications for the Linear Phase Coder Design.

Parameter	Value
Sampling Rate	140 kHz
Passband Edge	63 kHz
Passband ripple	< 0.01 dB
Anti-alias attenuation above 67 kHz	70 dB

8.3.2 Implementation

Rather than using design estimators, the design was implemented using the DECGEN compiler. DECGEN assumes a fixed architecture of cascaded CIC filters followed by a decimate by 2 FIR filter. A heuristic model is used to estimate the area consumed by various combinations of filters. Once an optimal choice has been selected, netlist generators are invoked to write the structure descriptions of the functional units.

The chip is comprised of 4 functional units as shown in Figure 8.3. The compiler calculated

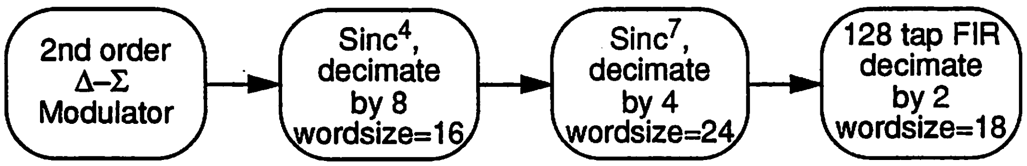

2nd order Δ–Σ Modulator
Sinc^4, decimate by 8 wordsize=16
Sinc^7, decimate by 4 wordsize=24
128 tap FIR decimate by 2 wordsize=18

Figure 8.3 Functional units in the linear phase A/D converter.

the necessary wordlengths in each of the CIC filters in an effort to minimize area and the wordlengths used in each filter are shown in the figure. The circuits and architecture templates have been discussed in Chapter 6. The use of single phase logic allows very high clock rates, certainly much higher than required in this application.

Care must be taken when rounding the output values of each filter stage. Rounding adds quantization noise to the signal. If not enough bits are retained, the noise floor is set by the quantization noise of the digital filter, which dominates over the noise from the modulator. No decimation filtering can then improve the signal to noise ratio above the theoretical 3 dB gain per decade of decimation. The DECGEN compiler currently takes care of these calculations, but the calculations do not provide guard bits. In the current implementation, only the upper 12 bits of the first filter and the upper 10 bits of the second filter were retained and used in the next stage. It would be better to add 1 or 2 bits to the numbers that DECGEN provides.

The circuit was implemented with several testability features. Each of the filters can be isolated through multiplexers. In addition, clocks and input signals can be supplied from the pads and intermediate outputs can be observed. Since many of the signals are serial, few extra pads were needed to implement this strategy.

8.3.3 Verification

With this chip, several verification problems became evident. The size of simulation files required many hours to perform a simulation run. The design was partitioned so that each functional block was verified independently. As was the case with the previous chip, a DC signal was used to verify that the FIR filters provided the correct sum at the output. This proved to be an insufficient test vector, since errors in the reset of the accumulator cannot be detected. Digit serial data formats also complicated testing. Rather than observing outputs at a given time in the simulation, the proper start bit must be identified, and then parallel data must be reconstructed from the digit data after several samples. To get the internal clock generators to start, a rather complicated simulation sequence was required. Simulation times in IRSIM measured on the order of 4 hours using an IBM RS 6000.

8.3.4 Results

The chip was 9.9 x 9.6 mm^2 in size in 2.0um CMOS and the die photo is shown in Figure 8.4. The core size was 8.3 x 8.1 mm^2. Quite a bit of area was lost due to mismatch in aspect ratios. An accumulator reset error in the final FIR filter went undetected in simulation, so the chip was not

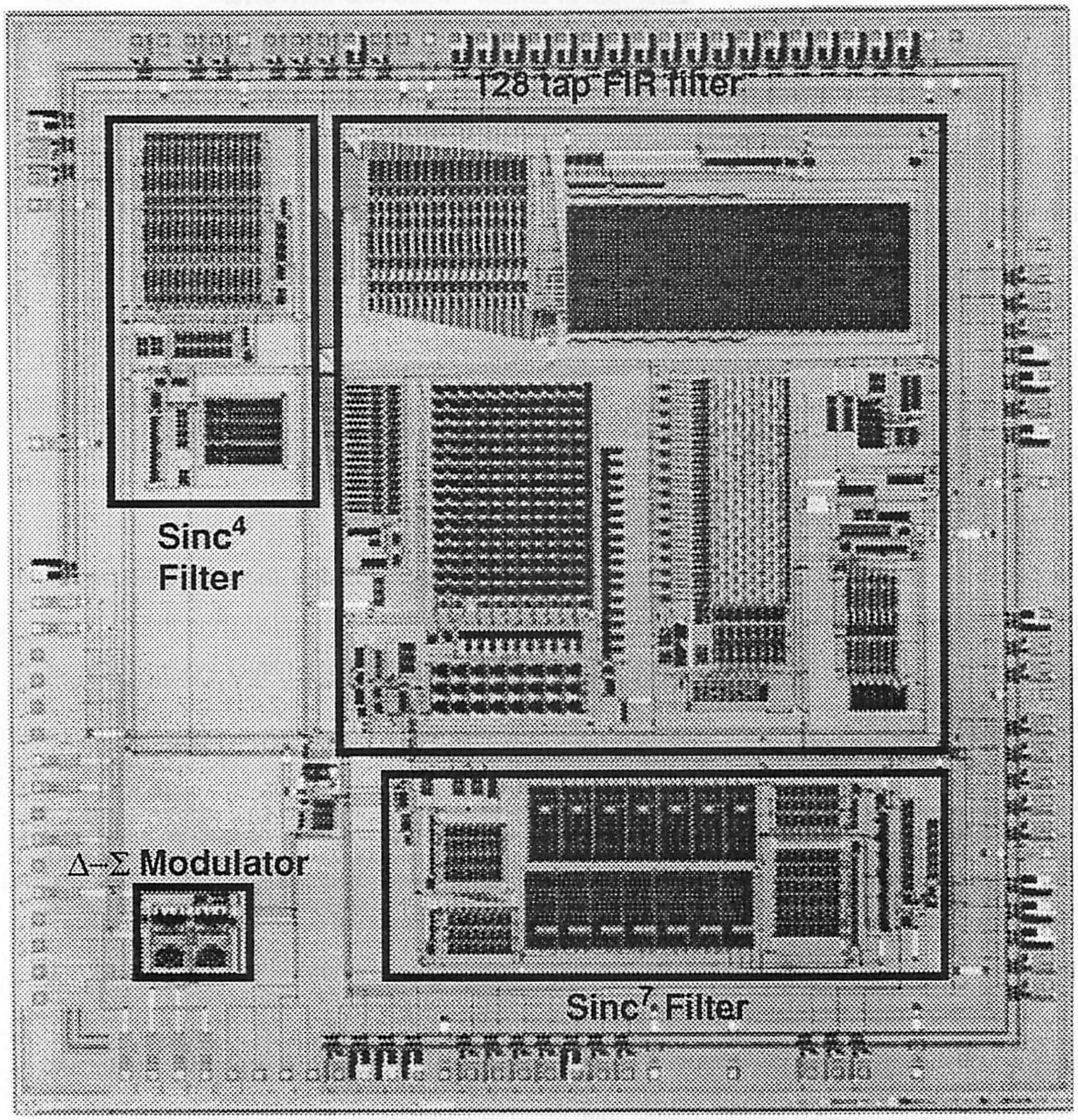


Figure 8.4 Die photo of the linear phase A/D converter.

fully functional. The error manifested itself by distorting 6 out of 64 samples when examining the finite impulse response. Output data was examined and was correct for the 58 samples that were valid. A sample of the measured chip output is shown in Figure 8.5. This error has been fixed in the architecture templates and a new version of the layout was generated and simulated but a new chip was not fabricated.

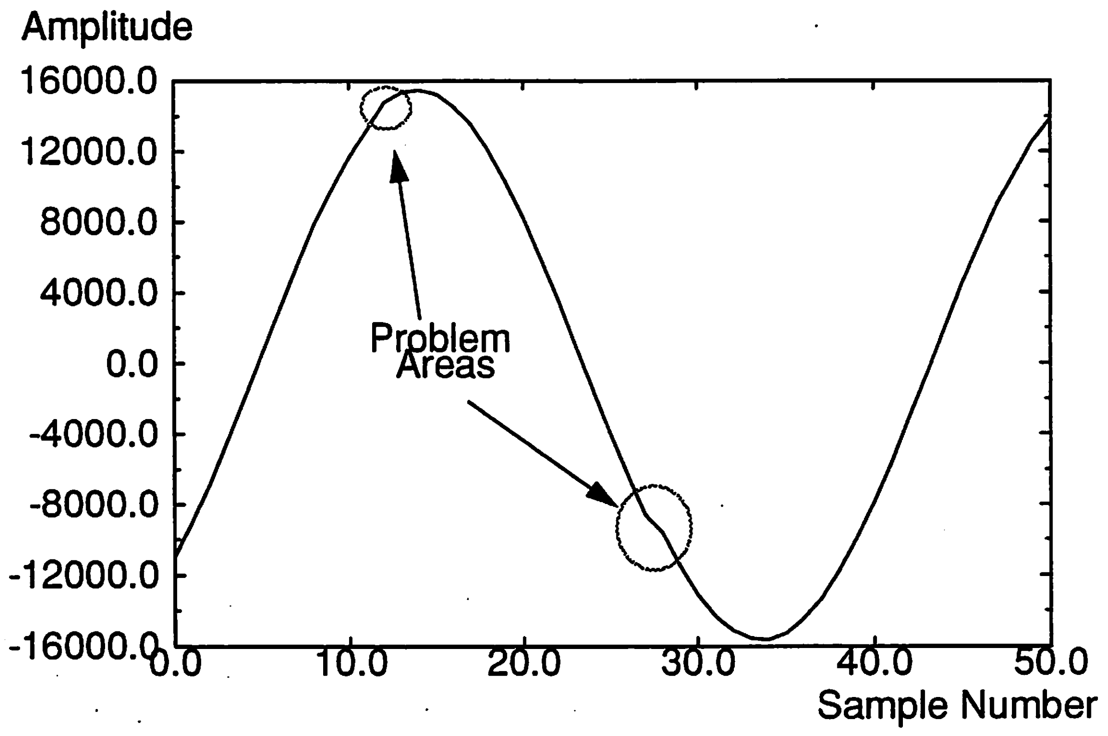


Figure 8.5 Measured output from the linear phase A/D chip.

8.4 Example 3: Speech Coder

8.4.1 Requirements

This circuit was designed to investigate the possibility of integrating a full signal acquisition system as part of a real-time speech recognition system. A basic specification was defined for the A/D response. The system designer specified a 2 sets of frequency responses summarized in Table 8-4, and requested a response in between the 2. The basic design strategy was to minimize

TABLE 8-4 Frequency domain specifications for the interface for speech coding.

Parameter	Relaxed Value	Tightly Specified Value
Sampling Rate	20 kHz	20 kHz
Passband	120 Hz - 9 kHz	120 Hz - 9 kHz
Passband Ripple	0.125 dB	0.05 dB
Magnitude at 60 Hz	-12 dB	-20 dB
Magnitude at 10 kHz	-14 dB	-30 dB
Anti-alias above 11 kHz	-28 dB	-72 dB

area, but to also maximize the digital filter performance. The design called for an output sampling rate of 20 kHz, and as much dynamic range as possible. Harmonic distortion is not a pressing issue. The large dynamic range is more important since the system must be able to pick up soft and loud voices. To study the efficiency of the design tools, a 2 week constraint was placed on total design time.

8.4.2 Design Study

To minimize area and power consumption, a 1.2 μm process was chosen for implementation since it was the smallest feature size available. The high-level design tools were applied and design candidates were formed. Estimates for resolution were obtained using simulation models and process data. Simulations indicated that a decimation ratio of 256 would provide about 90 dB of dynamic range when a second order modulator was used. While a slight redesign of the modulator would have allowed true 16 bit dynamic range, it was felt that the present modulator was sufficient, since redesign time would be on the order of 3 weeks. For similar reasons, a third order modulator was rejected. First order modulators were rejected due to the problems with tones in baseband.

At this point, several design candidates were enumerated. Using the anti-alias requirements and passband specifications, estimates for filter order were obtained. A partial list of the design candidates is shown in Table 8-5. In terms of area, the all FIR filter solution is the worst case since

TABLE 8-5 Partial list of design candidates.

No.	Filter 1	Filter 2	Filter 3	Filter 4	Filter 5
1	Sinc^4, dec. by 8	Sinc^7, dec. by 4	128 tap FIR, dec by 8		
2	Sinc^4, dec. by 8	Sinc^7, dec. by 4	8th Order Elliptic		
3	Sinc^3, dec. by 32	Polyphase, dec. by 8, Order 31	3rd Order Elliptic		
4	Sinc^3, dec. by 32	Polyphase, dec. by 4, Order 7	Polyphase, dec. by 2, Order 11	3rd Order Elliptic	
5	Sinc^3, dec. by 32	Polyphase, dec. by 2, Order 3	Polyphase, dec. by 2, Order 5	Polyphase, dec. by 2, Order 11	3rd Order Elliptic

speed requirements dictate that all the filters be implemented on separate processing units. The same is true for the case where an IIR filter is used as a decimating filter. The remaining combinations made use of a dedicated hardware processor for the FIR filter, and a processor core to implement the remaining filters.

For the FIR filter, we wish to implement a sinc^3 response. This can be achieved with an FIR filter or with a CIC filter. Area estimators showed that for a decimation ratio of 32, the CIC filter would require 1.7 mm^2 and the FIR filter would require 2.0 mm^2 of area. Although the FIR filter requires more power and area, it was chosen over the CIC filter since the area penalty was small and since there was some doubt about making the CIC filter work in the 1.2 μm process.

To establish the choice for the remaining digital filters, estimators were used. Since it is essentially a software implementation, it is fair to compare the total number of operations based on guesses for coefficient coding numbers. Using rules of thumb, the FIR filter approaches were found to be too costly. What remained were the approaches based on polyphase N-th band filters.

The coefficient design tools run fairly fast, so it was possible to perform detailed design to investigate properties of the remaining implementations. The aggressive specification provided by the designer allowed 0.05 dB of ripple in the passband. The sinc^3 filter introduces about 0.14 dB of droop across the passband which must be compensated in the final filter. It was thought that only a minor deviation from the estimated order would be needed to achieve these results. Using an optimization program, it was found that excessive effort was needed to compensate to 0.05 dB of ripple in the passband and that the polyphase N-th band filters introduce negligible ripple. After some consideration, we decided not to compensate the 0.14 dB of droop and to use a lattice wave digital filter as the final stage. This filter required fewer operations to implement than the corresponding biquad, and had better numerical rounding properties.

The remaining problem was to establish which combination of N-th band filters was the best for this application. Detailed design was performed on the 3 remaining candidates to obtain coefficients coded in the CSD notation. As a study in design metrics, the total number of nonzero bits required to evaluate each of the 3 cases was tabulated as in Table 8-6. Since the designs are similar, all 3 were translated to C and the C-to-silicon compiler was used to obtain the total number of operations required. These numbers are also listed in Table 8-6.

Since the total decimation ratio is 256, the processor can be designed to allow multiples of 256 cycles for implementing the digital filters. If 512 cycles are required, then the processor must run at twice the rate of the modulator and will consume more power. If all of the implementations had

TABLE 8-6 Summary of detailed design data for design candidates.

Filter Candidate (Refer to Table 8-5)	Number of Non-zero coefficient bits per output sample	Estimated Number of machine cycles required per output sample
3	149	365
4	82	253
5	78	277

required less than 256 cycles, they would all be equally viable at this point since they would require roughly the same amount of ROM space. However, only 1 design candidate met this criterion, so it was selected for implementation.

Using the results of the compiler, a finite wordlength simulation was performed. Since high resolution was desired, we decided not to truncate the data words at the outputs of filters. This meant that at least 16 bits were needed at the output of the filters. Figure 8.6 shows the results of

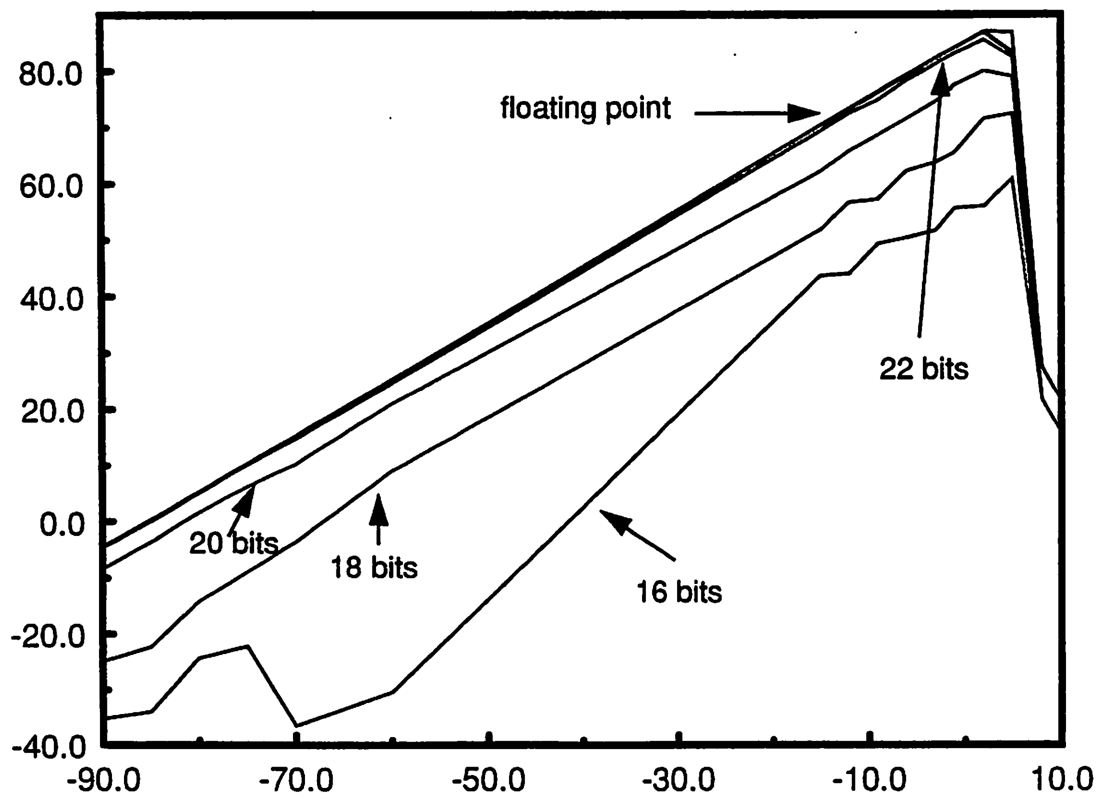


Figure 8.6 Results from finite wordlength simulations for the overall chip.

finite wordlength simulations for the SNR curves using the modulator simulation model. We chose 22 bits as the wordsize of the datapath for the processor.

The composite frequency response for the filtering combination was calculated to show how coefficient quantization affects the filter design. Figure 8.7 shows the composite frequency

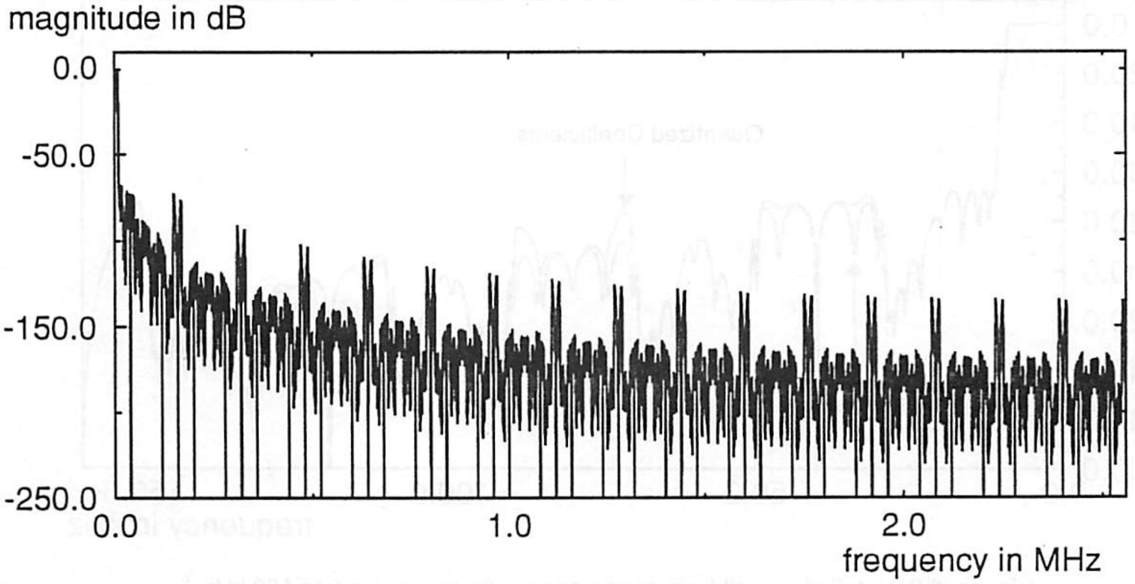


Figure 8.7 Composite frequency response for the decimate by 256 filter cascade.

response for the decimate by 256 filter. The calculations were made assuming that the sampling rate was 20 kHz at the output, or 5.12 MHz at the input. Figure 8.8 shows the detailed passband

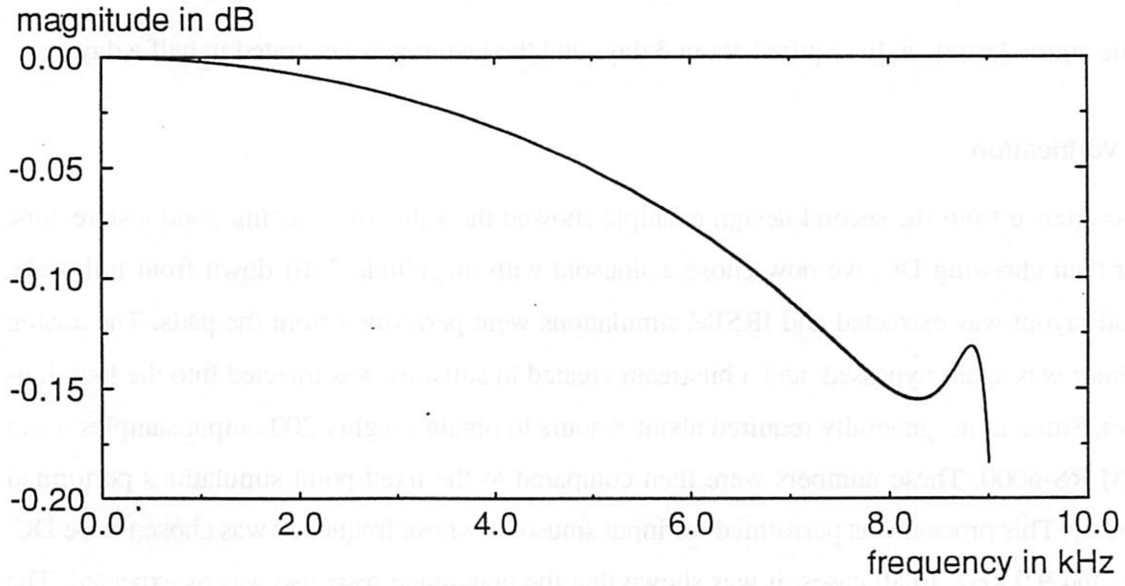


Figure 8.8 Detailed passband response.

response. There is less than 0.2 dB of ripple in the passband from 0 to 9 kHz and the peaking in the passband is caused by the ripple in the elliptic filter at the end of the cascade. The magnitude responses from both the ideal and quantized coefficients are plotted in the figure. The discrepancies between the 2 are very minor. Figure 8.9 isolates the region of the filter response from 0 to 160

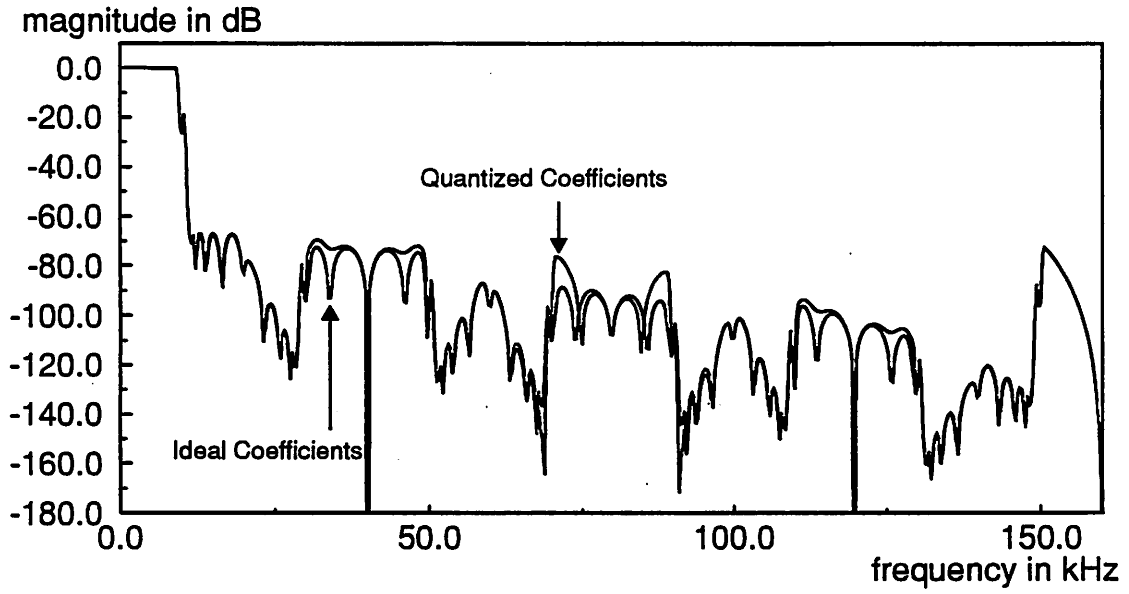


Figure 8.9 Detail of overall frequency response for the region 0 to 160 kHz.

kHz. The quantization of the filter coefficients degrades the magnitude response significantly but still maintains the anti-aliasing requirement and overall frequency domain specifications.

The entire design study required about 3 days and the layout was generated in half a day.

8.4.3 Verification

Experience from the second design example showed the value of choosing good test vectors. Rather than choosing DC, we now chose a sinusoid with magnitude 3 dB down from full scale. The full layout was extracted and IRSIM simulations were performed from the pads. The analog modulator was again bypassed, and a bitstream created in software was injected into the first digital filter. Simulations generally required about 8 hours to obtain roughly 200 output samples using an IBM RS-6000. These numbers were then compared to the fixed point simulations performed previously. This process was performed for input sinusoids whose frequency was chosen to be DC, 1 kHz, and 9.9 kHz. In all cases, it was shown that the magnitude response was as expected. The sinusoid at 9.9 kHz was rejected, since it lies in the stopband of the filter. A difference of about 5

LSB values was found between the fixed-point and the IRSIM simulations. This difference was attributed to slightly different simulation methods in the 2 models.

Verification and simulation required about 1 week of time with most of the time spent waiting for the simulations to finish.

This method of verification was shown to be very useful. A slight bug in the filter code generators was detected in the first simulation of the extracted layout. While it is costly in CPU time, the results are worth the investment. More work needs to be applied developing tools for correlating high level simulations to the circuit level simulation results.

8.4.4 Results

The total chip area required was 5.1 x 6.0 mm^2. The die photo for the signal acquisition module is shown in Figure 8.10. The chip contains 2 parallel to serial converters to allow examination from the output of both digital signal processing units. Performance results are tabulated in Table 8-7. The modulator made use of metal1-poly capacitors, allowing fabrication in a standard

TABLE 8-7 Summary of chip results.

Parameter	Value
Technology	1.2 μm standard CMOS
A/D Interface Area	3.2 x 4.0 mm^2 = 12.9 mm^2
Extrapolated Dynamic Range	87 dB
Measured Peak SNDR	74 dB
Output Sampling Rate	20 kHz
Oversampling Ratio	256
Anti-alias performance	> 65 dB over 0 to 9 kHz
Passband Ripple	< 0.15 dB over 0 to 9 kHz

digital IC process.

Figure 8.11 shows a comparison of measured and simulation data for the chip. There is excellent agreement between the simulation for the signal acquisition module and the actual measured data. At most, there is less than 3 dB deviation over the entire 87 dB of dynamic range, indicating that crosstalk is negligible. The FFT of the output data was examined, and appears to be free of interfering tones.

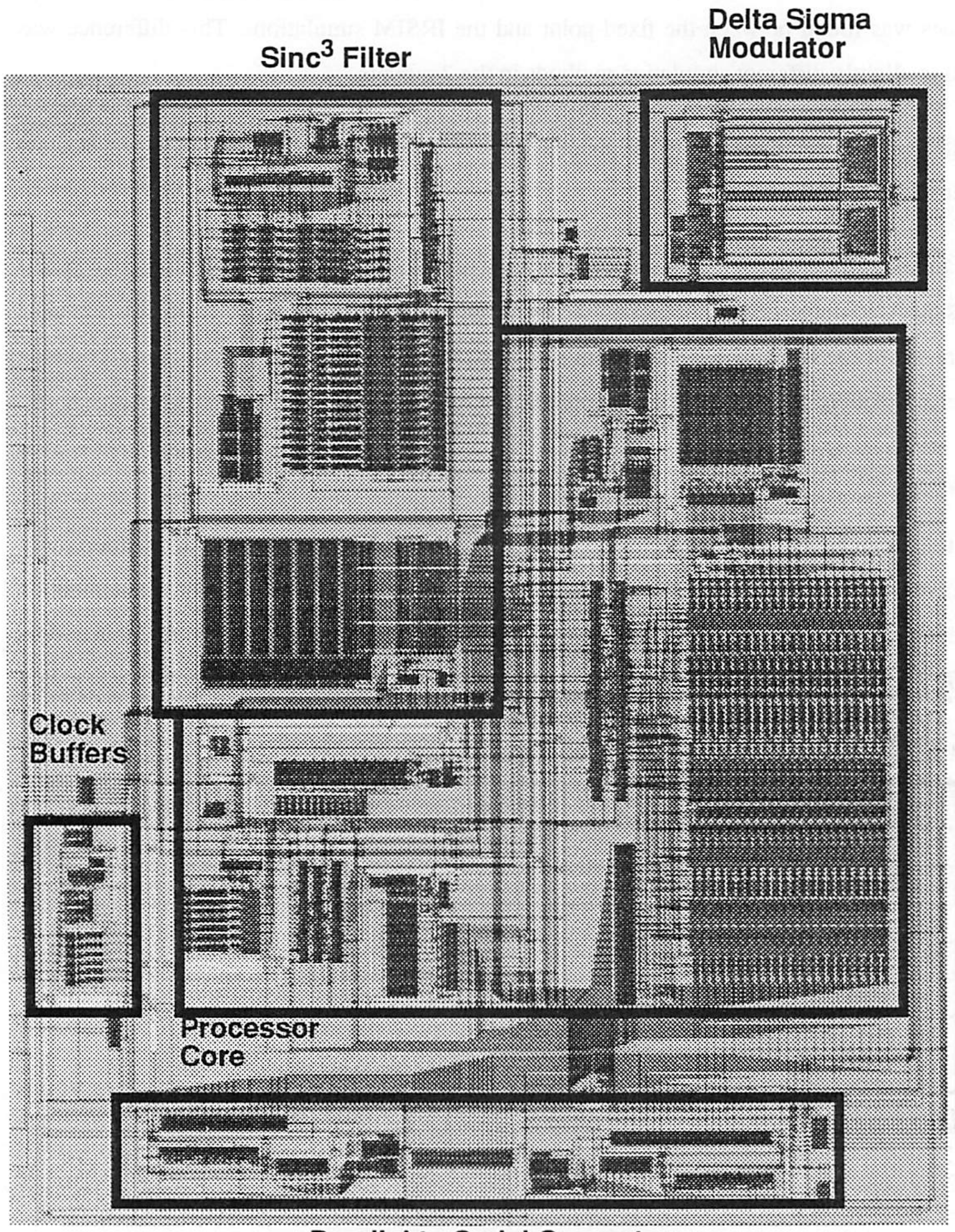


Figure 8.10 Die photo for the speech coder A/D interface.

In Figure 8.11, a comparison is made with data from a modulator whose output was processed in software with a sinc^3 filter. This data was taken from a test chip that contained only the modula-

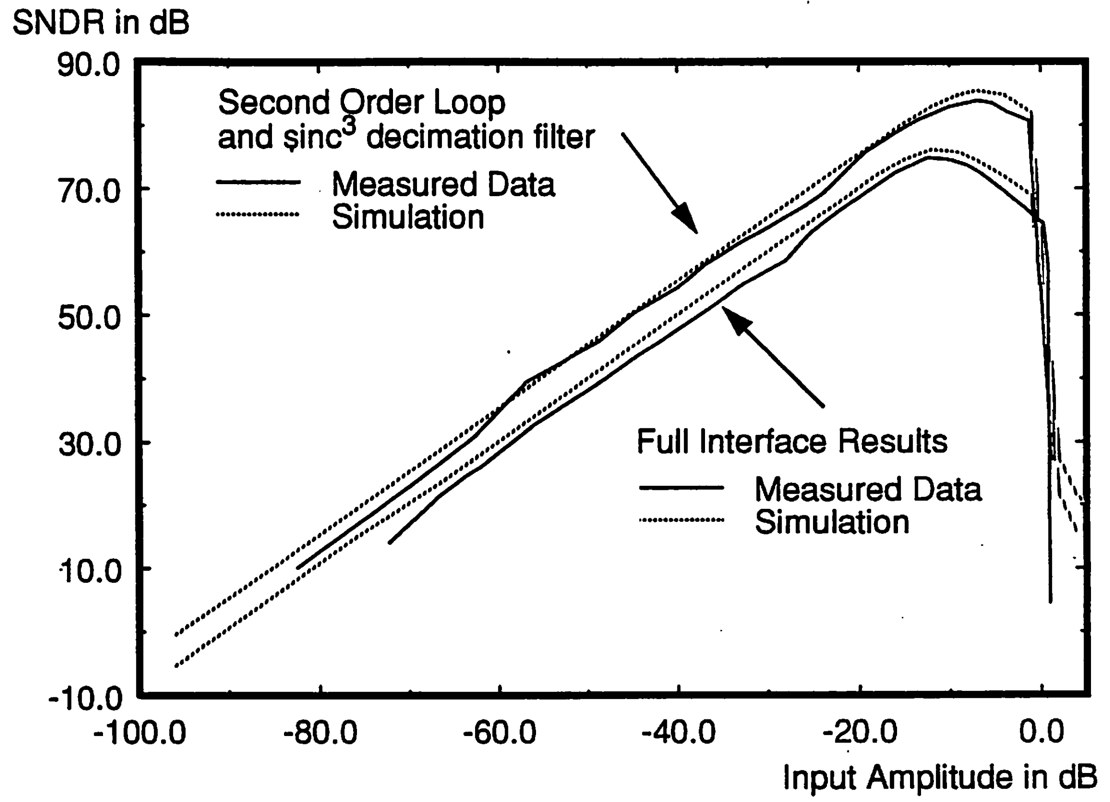


Figure 8.11 Measured results from test chips

tor without the digital filter. Note that there is roughly a 5 dB difference between the sinc^3 case and the chip case. This was observed in simulation and is attributed to 2 factors. The sinc^3 filter does have a droop attenuation, and thus decreases the total in band noise at high frequencies. The filter designed in our case has very little droop across the pass band, so in band noise is at full strength. The other factor is the aliasing of the shaped quantization noise from the modulator. The sinc3 transfer function is more efficient at rejecting this noise than the cascade filter, and thus a performance loss is expected.

The chip was functional up to an 8 MHz sampling rate with the speed being limited by the analog modulator. Experiments were run to see how lowering the power supply affects power consumption and the results are shown in Table 8-8.

8.5 Other Examples

Elements of the design system have been applied to several other projects. A summary of projects is listed in Table 8-9

TABLE 8-8 Measured data for power consumption.

Supply Voltage	Digital Power	Analog Power	Measured Peak SNDR
5 V	145 mW	1.2 mW	74.6 dB
4 V	72 mW	0.6 mW	70.3 dB
3 V	46 mW	0.3 mW	49.4 dB

TABLE 8-9 Projects using the design system.

Project	Comments
Wavelet Transform Filter Bank	Detailed design programs were used to analyze the implementation of a wavelet transform filter bank.
Accelerometer	Simulation environment was used to study the feasibility of using noise shaping methods in a control loop for an accelerometer.
Zoomer	Implementation of a nonlinear filtering scheme for optimal decoding in data acquisition applications.
Superconducting Δ–Σ Converters	Simulation models and detailed design tools were used to design and implement high resolution converters using superconducting circuits.

Wavelet filter banks can be designed using the tools for digital filtering [136]. A set of scaling filters is needed to implement a wavelet transform processor. The design method for the filters makes use of design techniques for quadrature mirror filters. One implementation of a wavelet filter bank was studied using the framework. Estimates for different types of FIR filters were examined to determine the feasibility of integrating a filter bank on a single chip. Mask layouts were generated to compare FIR filters generated by the FIR decimate-by-2 architecture template and the design tool XFIR, which uses the FIRGEN program [70]. It was found that design exploration using the HYPER system provided a smaller implementation than either of the architecture templates examined.

Other projects have made use of various design tools to study implementations of new features in oversampling techniques. The basic trend is to use the design tools in 1 of 2 ways. Some designers use the high-level design tools to investigate applications of noise-shaping techniques to new technologies. This is happening in the case of the accelerometer [84] and the superconducting A/D converter [83].

Other designers make use of the library cells as they prototype new ideas. This approach was used in the zoomer [37]. The C-to-silicon tools were used to study the implementation of a nonlinear filter. The chip was to be generated using a modified version of a library modulator. The non-linear decoding algorithm is easily expressed in C.

8.6 Summary

This chapter presented 3 extensive design examples using the oversampling converter design framework. All examples exhibited a greatly shortened design cycle. The improved simulation models were used to predict how the actual chip would perform based on process information for 1/f noise. The final section illustrated how various parts of the design system could be used to rapidly develop new design ideas.

CHAPTER 9

Conclusion

9.1 Design System Features

In this project, a design system has been developed to facilitate the design and analysis of signal acquisition modules based on oversampling A/D converters. The design system has demonstrated 3 major concepts: effective design partitioning of mixed signal designs, the possibility of accurate modulator modelling, and centralization of a variety of design styles to allow better design comparisons.

9.1.1 Partitioning of the Mixed Signal Design Process

Providing common levels of abstraction for both analog and digital circuits provides the basis for high-level design and a seamless design framework. The mixed signal design process was shown to be complex with quality solutions being spread out over a large part of the design space. To speed up the search and to allow comparison of widely differing solutions, the design process was partitioned into tasks that could be easily automated and decision tasks where human designers were needed to resolve complex design issues.

The automated tasks were defined with strict data interface specifications but limited definitions for functionality. This allows provides the tools designers with the freedom to use any technique to solve a problem and to further divide the task into smaller problems. Since interface specifications were defined between the primary steps in the design system, the design tools can localize data so that the designer must only deal with a small amount of information during the decision tasks. In addition, the interface specification allows designers to try several design tools based on a single design specification facilitating design exploration. Within the design system, the coefficient design, architecture mapping, and layout generation tasks were all highly automated.

The choice of algorithm, choice of architecture, and high-level design tasks all involved decision making. By examining the limited information required for the automated design tasks, designers are able to rapidly define new design ideas and then focus on the more abstract aspects of the design comparing the relative strengths and weaknesses of designs. This strategy was taken one step further by providing tools that could estimate the performance that a detailed design tool would give without having to run the tool. The strategy of hierarchical design estimation greatly speeds up design, increases estimation accuracy, and limits the amount of design information that must be stored from each design candidate.

Effective partitioning of the design process provided the means for a block oriented design system. Design elements could be abstracted to simple black boxes that could be implemented in a variety of ways. Designs can be mixed and matched to provide the best solution for a given application.

9.1.2 Improved Modulator Modelling

Improved models were presented for the noise-shaping modulators. The models were able to generate data that had an excellent degree of matching with data from fabricated circuits. 1/f noise was found to be a limiting factor in all of the designs examined. While chopper stabilization can be used to minimize this noise, it can be desirable to implement modulators with only simple opamps since it decreases design complexity and fairly high resolutions are still possible. Modelling can provide good insight in these cases. The new models will have their largest effect in system design. Using these models, system designers will be able to examine how specifications affect the implementation of a design. In addition, designers will be able to study how aliasing in suboptimal decimation filters affects overall resolution.

The success in matching models and circuits shows that simple simulation models will be useful for other applications. These models have already been applied for the design of mechanical Δ–Σ loops and to implementations of superconducting oversampled A/D converters. The real problem will be in correctly identifying the limiting factors to incorporate in the model.

9.1.3 Centralized Design Templates

Many different filtering styles have been incorporated in the design system. In addition, the definition of a CAD framework facilitates the integration of new design tools. Designers now have the opportunity to focus on comparing the merits of different architecture implementations and algorithms. The reuse of parameterized designs encourages design exploration and shortens the design cycle considerably. It also encapsulates information, allowing designers to use a module without having to work out the details of the implementation. The centralization of this design information should make it easier for digital system designers to incorporate these interfaces on-chip with a minimal investment in design time. In addition, it will allow designers to experiment on a single facet of the design and then to rely on the library to supply the other modules to implement a full design. The final design example of the last chapter shows that the design cycle can be reduced to a matter of days.

9.2 Extensions for D/A Conversion

In many applications, it is desirable to have on-chip A/D and D/A converters to form a codec. The benefits of oversampling and noise shaping can still be exploited for D/A conversion and many of the design techniques used in this project can be used. The basic oversampling D/A channel consists of the components shown in Figure 9.1. Interpolation filters are needed to bandlimit a

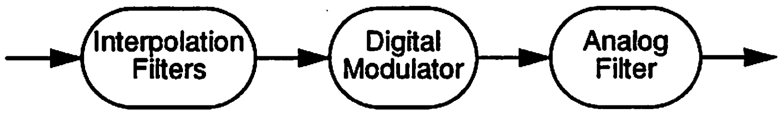


Figure 9.1 Block diagram of an oversampling D/A channel.

signal after extra zero samples are inserted to increase the sampling rate. The process is called interpolation because the filters replace the zero samples with a value that is interpolated from the original samples. The digital modulators take the higher sampling rate signal and perform the noise shaping. They are similar to the analog modulators and implement the same discrete time difference equations. A key portion of the modulator design is the implementation of the low resolution D/A converter. While it may be only a one bit output, attention must be given to the rise and fall times of the output bitstream since the shape of the waveform can determine the ultimate resolution. Finally, analog filters must be used for reconstruction and smoothing of the signal.

Most of the digital filter design programs can be adapted for interpolation filter design. Implementation can be performed with the HYPER and C-to-silicon tools. The digital modulator is easily implemented as an architecture template if the D/A converter cell is carefully designed. The remainder of the modulator consists of adders and registers. The most difficult problem will be designing and integrating the analog filters.

Switched-capacitor filters can be designed through ADORE [7]. However, these filters are single-ended and differential architectures are more desirable. At Berkeley, efforts are underway to attack the general analog place and route problem [11], so it should soon be possible to incorporate architecture templates for general analog filters.

9.3 Extensions for General Mixed Signal Design

As analog CAD tools mature, it will be interesting to see if they can be integrated within the current design framework. This would allow the comparison of oversampling and Nyquist-rate converters for the same design specifications. Oversampling converters do consume quite a bit of area, but this may not be the deciding factor in fine-line technologies. In addition, higher resolution without trimming is possible. The trade-offs in placement of different signal processing tasks in the analog and digital domains can also be effectively studied.

In the analog domain, it appears that module generators will arise around architectures that have desirable properties. Issues like testability and characterization will drive the development of these architectures. It may be that few design alternatives will be available, and the value of comparing many implementations at a high level will be diminished. It remains to be seen what advances will be made in analog CAD and this will determine how system designers choose to approach the mixed signal design problem.

9.4 Future Directions

Several future directions can be pursued with respect to the design concepts presented in this project.

9.4.1 Graphical Interfaces and Design

Within this project, PTOLEMY was used as the simulation system. However, something similar to the PTOLEMY graphical interface could be used as the design interface for a hierarchical design exploration system. PTOLEMY makes good use of hierarchy, and simulation can currently be performed with mixed levels of abstraction. PTOLEMY does encourage the use of a block diagram design methodology and reuse. However, the data structures in PTOLEMY are geared towards simulation. Adequate interconnect information is not readily available in the schematic views of PTOLEMY unless the simulation is being performed at the gate level. A set database views with different facets would need to be created as connectivity is discovered during the top down design process.

9.4.2 Refinements in the Libraries and Tools

The task of library definition and management needs to be addressed with a more unified approach. Currently, the task of macrocell generation for a library is a long process. As new libraries are formulated, the functionality of cells may change so that architecture templates become obsolete. A more desirable approach would be to define ways of expressing architectural information so that designs could easily be ported from library to library. With VHDL, this is becoming a possibility, but it is by no means a standard.

In addition, a method is needed for evaluating designs in the architecture library. Most users will not pay attention to the implementation details. Some circuit techniques may be marginal or problematic, especially as designs are scaled and moved between different process technologies. Reuse means that good and bad designs will be reused, so some facility is needed to judge designs and to dictate when redesigns should be considered.

There is a pressing need for tool development in the formal verification of designs. The design tools presented in this project speed up design, but verification of the layout and of the functionality now take most of the design time. Methods are needed for establishing appropriate test vectors at all levels of simulation and for easing the process of comparing the results at different levels. This problem is not as simple as for pure digital circuits since the analog circuits need to be simulated and verified concurrently with the digital portions.

9.4.3 Automating the Design Process

The title of this dissertation alluded to fully automatic generation. While much of the design process was automated, several key steps were left out on the assumption that a human designer would be better able to address the problem. It is possible to fully automate the design exploration at each level of the hierarchy, but there is a problem of defining a proper objective function for optimization. The cost function will need to cover several objectives and methods must be established for assigning relative costs to quantities to allow fair comparisons. It will most likely be a highly nonlinear cost function, so mathematical optimization will not be easy. Other optimization schemes may provide better insight. Recent developments in the field of Operations Research and in Systems Engineering show promise for studying these problems [137].

9.5 Conclusion

We have presented a CAD framework for the analysis, design, and implementation of oversampling analog to digital interfaces. High-level analysis methods were incorporated with modularity and reuse of structures to provide a framework that allows rapid design space exploration with expert advice supplied by estimation tools. Instead of relying on black box compilers, the designer becomes actively involved in finding and exploiting beneficial design trends. Case studies for 3 chip designs implemented with the design system tools were presented to illustrate the variety of problems that can be addressed. The framework builds on the previous work in silicon assembly to provide rapid prototyping of designs. Examples have shown that designers from many fields can take advantage of this open framework to prototype new ideas with out having to invest in the design effort to create the surrounding components or tools.

References

[1] K. Rimey and P. N. Hilfinger. A Compiler for Application-Specific Signal Processors. In *VLSI Signal Processing III*, pages 341–351. IEEE Press, November 1990.

[2] C. Chu, M. Potkonjak, M. Thaler, and J. Rabaey. HYPER: An Interactive Synthesis Environment for High Performance Real Time Applications. In *Proc. of ICCD'89*, pages 432–435, October 1989.

[3] M. G. R. Degrauwe, B. L. A. G. Goffart, C. Meixenberger, M. L. A. Pierre, J. B. Litsios, J. Rijmenants, O. J. A. P. Nys, E. Dijkstra, B. Joss, M. K. C. M. Meyvaert, T. R. Schwarz, and M. D. Pardoen. Towards an Analog System Design Environment. *IEEE JSSC*, 24(3):659–671, June 1989.

[4] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: A Framework for Analog Circuit Synthesis. *IEEE Trans. on CAD*, 8(12):1247–1266, December 1989.

[5] H. Koh, C. Sequin, and P. R. Gray. Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models. In *Proc. of ICCAD*, pages 502–505, November 1987.

[6] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley. KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing. *IEEE JSSC*, 26(3):330–343, March 1991.

[7] H. Yaghutiel, S. Shen, P. R. Gray, and A. S. Sangiovanni-Vincentelli. Automatic Layout of Switched-Capacitor Filters for Custom Applications. In *Proc. of ISSCC*, pages 170–171, 1988.

[8] P. E. Allen and P. R. Barton. A Silicon Compiler For Succesive Approximation A/D And D/A Converters. In *Proc. of CICC*, pages 552–555, 1986.

[9] G. Jusuf, P. R. Gray, and A. Sangiovanni-Vincentelli. CADICS - Cyclic Analog-To-Digital Converter Synthesis. In *Proc. of ICCAD*, pages 286–289, 1990.

[10] W. J. Helms and B. E. Byrkett. Compiler Generation of A To D Converters. In *Proc. of CICC*, pages 161–164, 1987.

[11] H. Chang, A. Sangiovanni-Vincentelli, F. Balarin, E. Charbon, U. Choudhurry, G. Jusuf, E. Liu, E. Malavasi, R. Neff, and P. R. Gray. A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits. In *Proceedings of CICC*, 1992.

[12] R. A. Walker and R. Camposano, editors. *A Survey of High-Level Synthesis Systems.* Kluwer Academic, Boston, 1991.

[13] B. Schweber. IF Stages Are Going Digital for both Analog and Digital Singals. *Analog Dialogue*, (1):3–6, 1992.

[14] S. A. Jantzi, M. Snelgrove, and P. F. Ferguson Jr. A 4th-Order Bandpass Sigma-Delta Modulator. In *Proc. of CICC*, pages 16.5.1–16.5.4, 1990.

[15] J. P. Calvez. *Embedded Real-Time Systems: A Specification and Design Methodology.* IRESTE and ECS Langues and Communications S. A., Nantes, France, 1990.

[16] A. M. Dewey and S. W. Director. *Principles of VLSI System Planning: A Framework for Conceptual Design.* Kluwer Academic, Norwell, Massachusetts, 1990.

[17] R. Jain, H. Samuelli, P. T. Yang, C. Chien, G. G. Chen, L. K. Lau, B.-Y. Chung, and E. G. Cohen. Computer-Aided Design of a BPSK Spread-Spectrum Chip Set. *IEEE JSSC*, 27(1):44–58, January 1992.

[18] E. Dijkstra, L. Cardoletti, O. Nys, C. Piguet, and M. Degrauwe. Wave digital filters in oversampling a/d converters. In *Proc. of ISCAS*, pages 2327–2330, June 1988.

[19] T. Saramaki and H. Tenhunen. Efficient VLSI-Realizable Decimators for Sigma-Delta Analog-to-Digital Converters. In *Proc. of ISCAS*, 1988.

[20] S.-S. Hang and R. Jain. Decimation Filter Compiler for Oversampling A/D Applications. In *Proceedings of ICASSP 1992*, pages V537–V540, March 1992.

[21] E. Berkcan. MxSICO: A Mixed Analog Digital Compiler: Application To Oversampled A/D Converters. In *Proceedings of CICC*, May 1990.

[22] A. Sherstinsky and C. G. Sodini. A Programmable Demodulator for Oversampled Analog-to-Digital Modulators. *IEEE Trans. on CAS*, 37(9):1092–1103, September 1990.

[23] T. Karema, T. Husu, T. Saramaki, and H. Tenhunen. A Filter Processor for Interpolation and Decimation. In *Proceedings of CICC*, pages 19.2.1–19.2.4, 1992.

[24] R. Steele. *Delta Modulation Systems.* John Wiley and Sons, New York, 1975.

[25] H. Inose and Y. Yasuda. A Unity Bit Coding Method by Negative Feedback. *Proc. IEEE*, 51:1524–1535, November 1963.

[26] N. S. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video.* Prentice Hall, Englewood Cliffs, 1984.

[27] J. C. Candy and O. J. Benjamin. The Structure of Quantization Noise from Sigma-Delta Modulation. *IEEE Trans. on Comm.*, COM-29(9):1316–1323, September 1981.

[28] B. P. Agrawal and K. Shenoi. Design Methodology for Sigma Delta Modulators. *IEEE Trans. on Comm.*, COM-31(3):361–369, March 1983.

[29] R. M. Gray, W. Chou, and P. W. Wong. Quantization Noise in Single-Loop Sigma-Delta Modulation with Sinusoidal Inputs. *IEEE Trans. on Comm.*, 37(9):956–968, September 1989.

[30] H. Wang. A Geometric View of Sigma-Delta Modulation. *IEEE Trans. on Cir. and Sys. II*, 39(6):402–405, June 1992.

[31] D. A. Kerth and D. S. Piasecki. An Oversampling Converter for Strain Gauge Transducers. *IEEE JSSC*, 27(12):1689–1696, December 1992.

[32] B. P. Brandt and B. A. Wooley. A 50-MHz Multibit Sigma-Delta Modulator for 12-b 2-MHz A/D Conversion. *IEEE JSSC*, 26(12):1746–1756, December 1991.

[33] F. Op't Eynde, G. M. Yin, and W. Sansen. A CMOS Fourth-Order 14b 500k-Sample/s Sigma-Delta ADC Converter. In *Proc. of ISSCC*, pages 62–63, 1991.

[34] V. Friedman. The Structure of the Limit Cycles in Sigma Delta Modulation. *IEEE Trans. on Comm.*, 36(8):972–979, August 1988.

[35] R. M. Gray. Oversampled Sigma-Delta Modulation. *IEEE Tran. on Comm.*, 35(5):481–489, May 1987.

[36] R. M. Gray. Spectral Analysis of Quantization Noise in a Single-Loop Sigma-Delta Modulator with dc Input. *IEEE Tran. on Comm.*, 37(6):588–599, June 1988.

[37] S. Hein and A. Zakhor. Lower Bounds on the MSE of the Single and Double Loop Sigma Delta Modulators. In *Proc. of ISCAS*, May 1990.

[38] S. H. Ardalan and J. J. Paulos. An Analysis of Nonlinear Behavior in Delta-Sigma Modulators. *IEEE Trans. on Cir. and Sys.*, CAS-34(6):593–603, JUNE 1987.

[39] E. F. Stikvoort. Some Remarks on the Stability and Performance of the Noise Shaper or Sigma-Delta Modulator. *IEEE Trans. on Comm.*, 36(10):1157–1162, October 1988.

[40] K. Uchimura, T. Hayashi, T. Kimura, and A. Iwata. Oversampling A-to-D and D-to-A Converters with Multistage Noise Shaping Modulators. *IEEE Trans. on ASSP*, 36(12):1899–1905, December 1988.

[41] D. B. Ribner. A Comparison of Modulator Networks for High-Order Oversampled Sigma-Delta Analog-to-Digital Converters. *IEEE Trans. on CAS*, 38(2):145–159, February 1991.

[42] L. A. Williams III and B. A. Wooley. Third-Order Cascaded Sigma-Delta Modulators. *IEEE Trans. on CAS*, 38(2):489–497, May 1991.

[43] W. Chou, P. W. Wong, and R. M. Gray. Multistage Sigma-Delta Modulation. *IEEE Trans. on Inf. Theory*, 35(4):784–796, July 1989.

[44] K. C.-H. Chao, S. Nadeem, W. L. Lee, and C. G. Sodini. A higher order topology for interpolative modulators for oversampling a/d converters. *IEEE Trans. on Cir. and Sys.*, 37(3):309–318, MARCH 1990.

[45] S. Jantzi, R. Schreier, and M. Snelgrove. Bandpass Sigma-Delta Analog-to-Digial Conversion. *IEEE Trans. on CAS*, 38(11):1406–1409, November 1991.

[46] Y. Sakina. Mulit-Bit Sigma-Delta Analog-to-Digital Converters with Nonlinearity Correction Using Dynamic Barrel Shifting. Master's thesis, UC Berkeley, May 1990.

[47] L. R. Carley. An Oversampling Analog-to-Digital Converter Topology for High-Resolution Signal Acquisition Systems. *IEEE Trans. on CAS*, CAS-34(1):83–90, January 1987.

[48] B. H. Leung. Multibit Sigma-Delta A/D Converter Incorporating a Novel Class of Dynamic Element Matching Techniques. *IEEE Trans. on CAS II*, 39(1):35–51, January 1992.

[49] R. E. Crochiere and L. R. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall, Inc., Englewood Cliffs, 1983.

[50] S. Chu and C. S. Burrus. Multirate Filter Designs Using Comb Filters. *IEEE Trans. on CAS*, CAS-31(11):913–924, November 1984.

[51] S.-S. Hang. DECGEN: Decimation Filter Compiler for Oversampling A/D Converter Applications. Master's thesis, UCLA, 1991.

[52] E. Viscito and J. P. Allebach. On Determining Optimum Multirate Structures for Narrow-Band FIR Filters. *IEEE Trans. on ASSP*, 36(8):1255–1271, August 1988.

[53] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, 1975.

[54] H. G. Martinez and T. W. Parks. A Class of Infinite-Duration Impulse Response Digital Filters for Sampling Rate Reduction. *IEEE Trans. on ASSP*, ASSP-27(2):154–162, April 1979.

[55] M. G. Bellanger, G. Bonnerot, and M. Coudreuse. Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks. *IEEE Trans. on ASSP*, ASSP-24(2):109–114, April 1976.

[56] R. Ansari and B. Liu. Efficient Sampling Rate Alteration Using Recursive (IIR) Digital Filters. *IEEE Trans. on ASSP*, ASSP-31(6):1366–1373, December 1983.

[57] M. Renfors and T. Saramaki. Recursive Nth-Band Digital Filters, Parts I and II. *IEEE Trans. on CAS*, CAS-34(1):24–51, January 1987.

[58] F. Mintzer. On Half-Band, Third-Band, and Nth-Band FIR Filters and Their Design. *IEEE Trans. on ASSP*, ASSP-30(5):734–738, October 1982.

[59] J. C. Candy. A Use of Double Integration in Sigma Delta Modulation. *IEEE Trans. on Comm.*, COM-33(3):249–258, March 1985.

[60] V. Friedman, D. M. Brinthaupt, D.-P. Chen, T. W. Deppa, Jr. J. P. Elward, E. M. Fields, J. W. Scott, and T. R. Viswanathan. A Dual-Channel Voice-Band PCM Codec Using Sigma Delta Modulation Technique. *IEEE JSSC*, 24(2):274–280, April 1989.

[61] B. H.-K. Leung, R. Neff, P. R. Gray, and R. W. Brodersen. Area-Efficient Multichannel Oversampled PCM Voice-Band Coder. *IEEE JSSC*, 23(6):1351–1357, December 1988.

[62] R. G. Lerch, M. H. Lamkemeyer, H. L. Fiedler, W. Bradinal, and P. Becker. A Monolithic Sigma-Delta A/D and D/A Converter with Filter for Broad-Band Speech Coding. *IEEE JSSC*, 26(12):1920–1927, December 1991.

[63] T. L. Satty. A Scaling Method for Priorities in Hierarchical Structures. *Journal of Mathematical Psychology*, 15:234–281, 1977.

[64] S. L. Padula and J. Sobieszczanski-Sobieski. A Computer Simulator for the Development of Engineering System Design Methodologies. Technical Report LAR-13580, NASA Langley Research Center, 1987.

[65] P. E. Landman and J. R. Rabaey. Power Estimation for High Level Synthesis. In *Proceedings of EDAC-EUROASIC*, 1993.

[66] H. Samuelli. An Improved Search Algorithm for the Design of Multiplierless FIR Filters

with Powers-of-Two Coefficients. *IEEE Trans. on CAS*, CAS-36(7):1044–1047, July 1989.

[67] R. W. Brodersen, editor. *Anatomy of a Silicon Compiler*. Kluwer Academic, Norwell, Massachusetts, 1992.

[68] A. Casotto, editor. *OCTTOOLS 5.1 Part I and II*. U. C. Berkeley Electronics Research Laboratory, 1991.

[69] A. Casotto, A. R. Newton, and A. Sangiovanni-Vincentelli. Design Management Based on Design Trees. In *27th ACM/IEEE Design Automation Conference*, pages 136–141, June 1990.

[70] R. Jain, P. T. Yang, and T. Yoshino. Firgen: A computer-aided design system for high performance fir filter integrated circuits. *IEEE Trans. on SP*, 7:1655–1668, July 1991.

[71] B. E. Boser, K.-P. Karmann, H. Martin, and B. A. Wooley. Simulating and Testing Oversampled Analog-to-Digital Converters. *IEEE Trans. on CAD*, CAD-7:668–674, June 1988.

[72] C. M. Wolff. *The Analysis and Simulation of Delta-Sigma Modulators*. PhD thesis, Carnegie Mellon University, December 1990.

[73] M. W. Hauser and R. W. Brodersen. Circuit and Technology Considerations for MOS Delta-Sigma A/D Converters. In *Proc. of ISCAS*, May 1986.

[74] B. E. Boser and B. A. Wooley. The Design of Sigma-Delta Modulation Analog-to-Digital Converters. *IEEE JSSC*, 23(6):1298–1308, December 1988.

[75] M. Rebeschini, N. R. van Bavel, P. Rakers, R. Greene, J. Caldwell, and J. R. Haug. A 16-b 160-kHz CMOS A/D Converter Using Sigma-Delta Modulation. *IEEE JSSC*, 25(2):431–440, April 1990.

[76] G. T. Brauns, R. J. Bishop, M. B. Steer, J. J. Paulos, and S. H. Ardalan. Table-Based Simulation of Delta-Sigma Modulators Using ZSIM. *Trans. on CAD*, 9(2):142–150, February 1990.

[77] K.-L. Lee and R. G. Meyer. Low-Distortion Switched-Capacitor Filter Design Techniques. *IEEE JSSC*, SC-20(6):1103–1113, December 1985.

[78] M. W. Hauser. From unpublished work and personal correspondence. 1989.

[79] B. E. Boser. *Design and Implementation of Oversampled Analog-to-Digital Converters*. PhD thesis, Stanford, October 1988.

[80] S. H. Lewis. *Video-Rate Analog-to-Digital Conversion Using Pipelined Architectures*. PhD thesis, U C Berkeley, November 1987.

[81] O. Agazzi and A. A. Adan. An Analog Front End for Full-Duplex Digital Transceivers Working on Twisted Pairs. *IEEE JSSC*, 24(2):229–240, April 1989.

[82] S. Harris. The Effects of Sampling Clock Jitter on Nyquist Sampling Analog-to-Digital Converters, and on Oversampling Delta-Sigma ADCs. *J. Audio Eng. Soc.*, 38(7/8):537–542, July/August 1990.

[83] P. H. Xiao and T. Van Duzer. Superconducting Delta-Sigma Oversampling A/D Converter. *IEEE Transactions on Applied Superconductivity*, 1993.

[84] W. Yun, R. T. Howe, and P. R. Gray. Surface Micromachined, Digitally Force-Balanced

Accelerometer with Integrated CMOS Detection Circuitry. In *Proc. of the IEEE Solid-State Sensor and Actuator Workshop*, pages 126–131, 1992.

[85] B. P. Brandt and B. A. Wooley. Second-Order Sigma-Delta Modulation for Digital-Audio Signal Acquisition. *IEEE JSSC*, 26(4):618–627, April 1991.

[86] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 1988.

[87] P. Gruber. 1/f-Noise Generation. In *Noise in Physical Systems and 1/f Noise*, pages 357–360, 1985.

[88] M. Gardner. White and brown music, fractal curves and one-over-f fluctuation. *Scientific American*, pages 16–32, April 1978.

[89] B. Pelligrini, R. Saletti, B. Neri, and P. Terreni. 1/f^v Noise Generators. In *Noise in Physical Systems and 1/f Noise*, 1985.

[90] A. van der Ziel. *Noise in Measurments*. John Wiley and Sons, New York, 1976.

[91] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Platform for Heterogeneous Simulation and Prototyping. In *Proc. 1991 European Simulation Conference*, Copenhagen, Denmark, 1991.

[92] E. A. Lee and D. G. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. *IEEE Trans. on Computers*, C-36(1):24–35, January 1987.

[93] A. S. Sedra and P. O. Brackett. *Filter Theory and Design: Active and Passive.* Matrix Publishers, Inc., Beaverton, 1978.

[94] G. F. Dehner. *Program for the Design of Recursive Digital Filters*, pages 6.1.1–6.3.105. IEEE Press, New York, 1979.

[95] L. Gazsi. Explicit Formulas for Lattice Wave Digital Filters. *IEEE Trans. on CAS*, CAS-32(1):68–88, January 1985.

[96] L. C. Liu. A silicon compiler for high-performance lattice wave digital filters integrated circuits. Master's thesis, UCLA, 1991.

[97] W. Lao. The Design of High-Speed CMOS 15th-Order Recursive Digial Filter. Master's thesis, UCLA, 1988.

[98] E. Remez. Sur le calcul effectif des polynomes de Tchebicheff. *Compt. Rend. Acad. Sci. (France)*, 199:337–340, July 1934.

[99] J. H. McClellan, T. W. Parks, and L. R. Rabiner. A Computer Program for Designing Optimum FIR Linear Phase Digital Filters. *IEEE Trans. on Aud. and Elec.*, AU-21(6):506–515, December 1973.

[100] T. Henk. The generation of arbitrary-phase polynomials by recurrence formulae. *Int. J. Circuit Theory Appl.*, 9:461–478, September 1981.

[101] G. C. Temes and D. Y. F. Zai. Least pth Approximation. *IEEE Trans. Circuit Theory*, CT-16:235–237, May 1969.

[102] J. W. Bandler and C. Charalambous. Theory of Generalized Least pth Approximation. *IEEE Trans. Circuit Theory*, CT-19:287–289, May 1972.

[103] A. G. Deczky. Synthesis of Recursive Digital Filters Using the Minimum p-Error Criterion. *IEEE Trans. on Aud. and Elec.*, AU-20(4):257–263, October 1972.

[104] R. Fletcher and M. J. D. Powell. A Rapidly Convergent Descent Method for Minimization. *Comput. J.*, 6:163–168, 1963.

[105] M. T. Dolan and J. F. Kaiser. *An Optimization Program for the Design of Digital Filter Transfer Functions*, pages 6.3.1–6.3.23. IEEE Press, New York, 1979.

[106] B. H. Leung. Design Methodology of Decimation Filters for Oversampled ADC Based on Quadratic Programming. *IEEE Trans. CAS*, 10:1121–1132, October 1991.

[107] Z. P. Ma and B. Leung. Decimation iir filter for oversampled a/d based on multiobjective optimization. In *34th Midwest Symposium on Circuits and Systems*, 1991.

[108] K. Hwang. *Computer Arithmetic: Principles, Architecture, and Design.* Wiley, New York, 1979.

[109] R. Jain, J. Vandewalle, and H. J. DeMan. Efficient and Accurate Multiparameter Analysis of Linear Digital Filters Using a Multivariable Feedback Representation. *IEEE Trans. on CAS*, CAS-32:225–235, March 1985.

[110] L. J. M. Claesen, H. J. DeMan, and J. Vandewalle. DIGEST: A Digital Filter Evaluation and Simulation Tool for MOS VLSI Filter Implementations. *IEEE JSSC*, SC-19(3):414–424, June 1984.

[111] E. Dijkstra, M. Degrauwe, J. Rimenants, and O. Nys. A Design Methodology for Decimation Filters In Sigma Delta A/D Converters. In *Proceedings of ISCAS*, pages 479–482, May 1987.

[112] S. L. Garverick, K. Fujino, D. T. McGrath, and R. D. Baertsch. A Programmable Mixed-Signal ASIC for Power Metering. *IEEE JSSC*, 26(12):2008–2016, December 1991.

[113] F. F. Yassa and S. L. Garverick. A Multichannel Digital Demodulator for LVDT/RVDT Position Sensors. *IEEE JSSC*, 25(2):441–450, April 1990.

[114] P. Hilfinger and J. Rabaey. DSP Specification Using the Silage Language. In R. W. Brodersen, editor, *Anatomy of a Silicon Compiler*, chapter 15, pages 199–220. Kluwer Academic, 1992.

[115] M. W. Hauser, P. J. Hurst, and R. W. Brodersen. MOS ADC-Filter Combination That Does Not Require Precision Analog Components. In *Proc. of ISSCC*, February 1985.

[116] J. C. Candy, B. A. Wooley, and O. J. Benjamin. A Voiceband Codec with Digital Filtering. *IEEE Trans. Comm.*, COM-29:815–830, June 1981.

[117] D. P. Schultz. The Influence of Hardware Mapping on High-Level Synthesis. Master's thesis, U C Berkeley, May 1992.

[118] E. Dijkstra, O. Nys, C. Piguet, and M. Degrauwe. On The Use of Modulo Arithmetic Comb Filters in Sigma Delta Modulators. In *Proc. of ICASSP*, pages 2001–2004, April 1988.

[119] E. B. Hogenauer. An Economical Class of Digital Filters for Decimation and Interpolation. *Trans. on ASSP*, ASSP-29:155–162, April 1981.

[120] J. Yuan and C. Svensson. High-Speed CMOS Circuit Technique. *IEEE JSSC*, 24(1):62–70, February 1989.

[121] S. K. Azim. *Application of Silicon Compilation Techniques to a Robot Controller Design.*

PhD thesis, UC Berkeley, May 1988.

[122] C. S. Shung. *An Integrated CAD System for Algorithm-Specific IC Design.* PhD thesis, U. C. Berkley, June 1988.

[123] L. E. Thon and R. W. Brodersen. C-to-silicon Compilation. In *Proceedings of CICC*, pages 11.7.1–11.7.4, 1992.

[124] M. F. Mar. A Retargetable Microcode Generator for the Lager Environment. Master's thesis, UC Berkeley, June 1989.

[125] L. E. Thon. *Application Specific Processors for Numerical Algorithms.* PhD thesis, UC Berkeley, December 1992.

[126] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-Power CMOS Digital Design. *IEEE JSSC*, 27(4):473–484, April 1992.

[127] P. J. Hurst and R. A. Levinson. Delta-Sigma A/Ds with Reduced Sensitivity to Op Amp Noise and Gain. In *Proc. of ISCAS*, 1989.

[128] R. Castello and P. R. Gray. A High-Performance Micropower Switched Capacitor Filter. *IEEE JSSC*, SC-20(6):1122–1132, December 1985.

[129] D. Senderowicz, S. F. Dreyer, J. H. Huggins, C. F. Rahim, and C. A. Laber. A Family of Differential NMOS Analog Circuits for a PCM Codec Filter CHIP. *IEEE JSSC*, SC-17(6):1014–1023, December 1982.

[130] A. Yukawa. A CMOS 8-Bit High-Speed A/D Converter IC. *IEEE JSSC*, SC-20(3):778–779, June 1985.

[131] J. L. McCreary. Matching Properties, and Voltage and Temperature Dependence of MOS Capacitors. *IEEE JSSC*, SC-16(6):608–616, December 1981.

[132] A. B. Grebene. *Bipolar and MOS Analog Integrated Circuit Design.* John Wiley and Sons, New York, 1984.

[133] B. H.-K. Leung. *Multi-Channel PCM A/D Interfaces Using Oversampling Techniques.* PhD thesis, UC Berkeley, November 1987.

[134] J. P. O'Connell. Monolithic Sigma-Delta A/D Converters Offer 21-Bit Resolution. *Analog Dialogue*, (1):7–9, 1992.

[135] M. W. Hauser. Technology scaling and performance limitations in delta-sigma analog-digital converters. In *Proc. of ISCAS*, May 1990.

[136] O. Rioul and M. Vetterli. Wavelets and Signal Processing. *IEEE SP Magazine*, pages 14–38, October 1991.

[137] W. J. Fabrycky. Indirect Experimentation for System Optimization: A Paradigm Based on Design Dependent Parameters. In *Proc. of the 2nd Annual Int. Symp. of the National Council on Systems Engineering*, pages 357–364, July 1992.

APPENDIX A

The OCDS Distribution

A.1 Introduction

The tools described in this dissertation were gathered into the Oversampling Converter Design System (OCDS) distribution. Included on this tape are magic layouts, design programs, SDL design templates, PTOLEMY models, C simulation models, documentation, and examples. The distribution requires about 80 Mb of disk space.

Throughout this appendix, it is assumed that the environment $OCDS has been defined. $OCDS refers to the home directory of the OCDS installation. All files are referenced from this directory. For the LAGER and OCTTOOLS packages, references are made assuming the installation is similar to that of the zabriskie cluster at Berkeley.

The OCDS distribution does not contain the C-to-silicon tools. These tools are found in the Lager directories, under ~lager/common/c-to-silicon and ~rl, which is held within the c-to-silicon directory.

A.2 Requirements

For a full installation, several tools and conditions must be met. They are listed here.

1. The install makefiles will work with GNU make version 3.58. Earlier versions may work, but Lager Gnumake did not correctly detect dependencies that were up to date.
2. Prior to beginning the install, the OCDS tape should be read into some directory. You must then set $OCDS to the root directory before proceeding.

3. OCTTOOLS 5.1 should be installed.
4. LAGER 2.1 or later should be installed and available, with the c-to-silicon tools.
5. PERL should installed. If you don't have it, get it from an ftp site such as uunet.uu.net or jpl-devvax.jpl.nasa.gov.
6. f77, the fortran compiler should be available.
7. PTOLEMY version 0.4-alpha should be installed.

Subsets of the full install are available. If you don't have PTOLEMY, you can use the C simulators provided. LAGER is only needed if you plan to generate layouts. The C-to-silicon tools, however, are necessary for some architecture mapping and compilation tasks. It may be possible to convert the FORTRAN code to C using the f2c compiler. This would eliminate the need for f77. All of the programming code was written with OCTTOOLS 5.1. Later or earlier versions may work, but there is not guarantee.

To begin the install process, read the file $OCDS/README and $OCDS/Makefile. All the information for the install in found there. The distribution comes with binaries compiled for Sparc architectures running SunOS 4.1.3.

A.3 Typical User Setup

If OCDS has already been installed, you must add several lines to your .cshrc file before you can use the system. First, be sure that $OCDS/bin is in your path. You can do this by executing commands similar to those listed below in your .login file:

```
# Gives path to root directory of OCDS
setenv OCDS /usr/tools/siera/OCDS
# Adds $OCDS/bin to path
set path = ($OCDS/bin $path)
```

In addition, you must add commands similar to the following ones in your .cshrc to set up your account for running the OCDS version of pigiRpc, necessary for PTOLEMY simulations:

```
### For PTOLEMY ###
if (! $?ARCH) setenv ARCH '/bin/arch'
setenv PTOLEMY ~ptolemy
setenv PIGIRPC $OCDS/bin/pigiRpc
set path = (. ~ptolemy/bin "ptolemy/bin.$ARCH $OCDS/bin /usr/ucb $path)
```

Also, append the contents of $OCDS/lager to the lager file in your home directory. This will allow LAGER to find the architecture libraries.

Be sure to check that these things are set in the correct places when you have problems executing parts of the design system.

A.4 Organization

In $OCDS, there are many directories. This section will discuss the highlights and point out various features available. The section will start with documentation and tutorials, and then move on to the standard programs and cell libraries. The next section provides a discussion of some features not integrated into the design system.

A.4.1 Tutorials and Documentation

Tutorials are found in the directory $OCDS/hypertext. The files are in Framemaker format, and meant to be read using Framemaker hypertext features. The README file gives a quick overview of the material. The tutorials are set up to introduce design with oversampling A/D converters, then to guide a designer through the process from specification to layout. Postscript versions of the files are given for those who don't have Framemaker.

Most of the detailed documentation is found in this dissertation. The manual pages reproduced in Appendix B are stored in Framemaker format in $OCDS/manpages. Use the file manpages.book to access all the files and print them out at the same time.

A.4.2 Examples

Examples are contained in $OCDS/examples. There are 4 directories of examples: 8bit, comp2schem, design, and speech. 8bit contains the files for the example used in the tutorial and speech contains the files for the example given in Appendix C. In the design directory, there is a README file that describes the examples, which include the design of biquads, linear phase equiripple FIR filters, polyphase N-th band filters, sinc filters, and coefficient quantization examples. In the comp2schem directory, there are 2 examples of filter design and automatic translation to PTOLEMY flowgraphs.

There is an exercise to introduce the oversampling A/D converter stars in PTOLEMY. This is found in $OCDS/ptolemy/tutorial. The file, in Framemaker format, is called ptlab.doc. The exercise will show you how to assemble your own simulations using the added stars. It is assumed that the user knows how to used PTOLEMY already.

A.4.3 Design Programs

The Microsoft Excel macro sheets are found in $OCDS/excel. The README file in that directory describes how to load the files on a PC compatible running Windows and Excel v4.0.

The filter design programs are contained in $OCDS/design. The basic set of programs consists of biquads, doredi, optiir, and polyiir for IIR design, and sinwin and lpfir for FIR design. The program fr is used to plot magnitude and phase responses. sCandi is used for coefficient quantization along with digest. Most users will not use doredi, optiir, and digest since scripts have been written so that users will not have to deal with the input and output formats. They can, however, be used if you know how to set up the input files.

The $OCDS/lib directory holds the dblib package. This package is further described in Appendix D, and is used by programmers to implement interfaces to the design database.

Programs for architecture mapping are contained in $OCDS/mapper. The OCDS installation provides the programs c2sil, coef2rl, coef2sil, decfir, pisogen, and writeSDL. Architecture templates are found in $OCDS/architecture. This directory holds these architectures:

- clkmodule - clock generator.
- firld1, firld2, firld3 - FIR filters for L/D = 1, 2, or 3
- lambda - a C-to-silicon target architecture
- piso - parallel to serial converter

System support programs reside in $OCDS/system. The programs vmake, parser, and the programs used for documentation all have sources here.

A.4.4 Layout Generation

A single layout generator program was supplied with OCDS for LAGER. This program is called leafgen, and will instantiate a single leafcell in a design. Some architecture templates use this, and source is found in $OCDS/layout_gen/leafgen.

A.4.5 Cell Libraries

Cell libraries are provided with SDL descriptions and magic layouts. In $OCDS/cellib, several dpp libraries and a Tim.Lager library are supplied. These cells are used by the various architecture templates supplied with OCDS. For analog cells, modulator layouts are found in $OCDS/analog/

leafcells. They are fully compatible with LAGER and can be instantiated in any LAGER design. Some of these cells use double poly capacitors, so use the technology file $OCDS/technology/ scmos2p4.tech21 with magic v4 to view them. Also, for generating OCT from magic files, use $OCDS/technology/mag2oct for translation. From OCT to magic, use $OCDS/technology/to-magic-gen.

A.4.6 PTOLEMY

The files to run the oversampling A/D stars in PTOLEMY are found in $OCDS/ptolemy. The program comp2schem for translating component view data to PTOLEMY schematic views is found in $OCDS/ptolemy/comp2schem. The rest of the directory is set up like the PTOLEMY distribution for SDF stars. $OCDS/ptolemy/doc holds the documentation for the stars in troff format. There is a makefile in the directory for printing things. $OCDS/ptolemy/osad holds various flow-graphs for demonstrations. $OCDS/ptolemy/pigiRpc holds the sources for making the OCDS version of pigiRpc. $OCDS/ptolemy/src contains the source for the MSE and PFF4 programs called by stars in the OCDS distribution. Manual pages for both of these programs are found here. $OCDS/stars contains the source code for the stars added into PTOLEMY.

A.5 Extra Features

A.5.1 Design Programs

In $OCDS/design/extras, you will find some programs that were examined, but not integrated into the design system. There is a C translation of Deczky's least-pth error criterion design program along with some programs for rounding FIR coefficients.

A.5.2 Mapping

In $OCDS/mapper/laker, there is source for mapping biquad descriptions on to a switched-capacitor filter architecture. This program could be used with ADORE to generate switched-capacitor filters automatically.

In $OCDS/packages/decgen, the complete decgen distribution is given. While none of the decgen design programs were integrated into OCDS, several enhancements were made including area estimation routines. Only I/O interfaces for the database need to be added to incorporated these architecture mapping techniques into OCDS. Decgen provided a high-level design interface that can be used without OCDS. See $OCDS/README for more details.

A.5.3 Analog Layout

In $OCDS/analog/test_leafcells, a number of older versions of modulator layout are found. These layouts have some feature that enhanced the testability of the modulators. Most modulators have switch structures that allow observation of different internal analog nodes.

The file $OCDS/analog/templates.tar.Z is a compressed tar file containing various magic layouts for building modulators. Included in this archive are the 3 opamps mentioned in this dissertation along with spice simulation decks from extracted layouts. Several versions of capacitors, comparators, and modulator layouts round out the archive.

A.5.4 C simulation

In $OCDS/Csimulation, there is a complete stand alone simulation package. See the README file in that directory for details.

A.5.5 Chip Layouts

In $OCDS/chips/chipcif, the CIF files for all of the chips described in Appendix H are given. The SDL files for generating these chips are not given in the release, but can be found at Berkeley on optical disk.

APPENDIX B

Design System Program Descriptions

B.1 Description

The design system integrates many different types of CAD programs. This appendix will give the manual pages for the design system programs.

NAME

biquads

program to design coefficients for cascaded biquad IIR filters.

SYNOPSIS

biquads [options] view

OPTIONS

-L [logfile]	file to use to log output information.
-O	run OPTIIR after synthesis.
-s	save intermediate input and output files.

DESCRIPTION

biquads is a front end to a collection of programs for design cascaded biquad IIR filters. The program takes as input a component view with sufficient properties to direct the design process and a valid frequency specification. These properties must be set in the component view.

FILTER_CLASS - one of LOWPASS, HIGHPASS, BANDPASS, or BANDSTOP
ORDER - defines the order of the filter to be designed.
FILTER_STRUCTURE - must be set to BIQUAD_CASCADE
MAG_APPROX - one of ELLIPTIC, BUTTERWORTH, CHEBYSHEVI, or CHEBYSHEVII
SAMPLING_FREQ - set to the sampling frequency in Hz. This value is used to reference the frequency points given in the frequency specification.

See parser(1) for information about the frequency specification declaration. If a deviation from a classical design is desired, enter the final frequency domain specification for the filter. *biquads* uses the band edge information and the band edge deviation specification to perform the initial design. If the -O option was requested, OPTIIR is then invoked, and the program tries to move the classical design towards the desired one. Note that none of the programs used by *biquads* has the ability to distinguish if it has met the specifications or not. The user must check to verify that the design has succeeded. If not, another iteration must be used and the filter order can be increased, or the specification made more lenient. The program fr(1) can be used to automatically check the frequency specification.

The program works by translating component view information into ASCII text files that serve as inputs to the filter design programs. The programs used are DOREDI, by G. F. Dehner and OPTIIR, by M. T. Dolan and J. F. Kaiser. The resulting ASCII output files are parsed, and information stored into the component view. By saving the intermediate files, users can run these programs by hand.

AUTHOR

Monte F. Mar, mmar@zabriskie.berkeley.edu

SEE ALSO

parser(1), fr(1)

REFERENCES

Programs for Digital Signal Processing, IEEE Press, 1979.

NAME

c2sil

script used to compile a program written in RL to parameters for an SDL file description of a processor.

SYNOPSIS

c2sil [options] viewname

OPTIONS

-v	Verbose mode. Shows information about the compilation process.
-L [filename]	Log file for output.
-s	Saves all intermediate files created in the compilation process.
-a	Specifies that the subroutine stack should be left out of the processor.
-l	Specifies that the logical unit and the address processing unit should be left out of the processor (i.e. use the Lambda processor architecture rather than PUMA).

DESCRIPTION

c2sil is a Perl script written to automate the process of compiling a program written in RL onto a processor architecture using the C-to-silicon tools. The script requires that an RL file with the same name as the input view must be in the same working directory as the input view, and the file must have an extension .k.

To make this script work correctly, you must have the following programs in your path: kc, jas, compact-parw, espresso, KAest, and LAest.

When the script is invoked, the first step is to call the RL compiler with the .k file as input. When the .k file was generated, information about the target processor was included in the .k file. This script makes the assumption that the processor architecture was either PUMA or Lambda. PUMA is the default, and Lambda is used when the -l flag is specified. The input code is compiled to a register transfer language description with a .s extension and a summary of the compilation information is printed to stdout (or to the log file if selected) and to a file with the extension .script. Next, the jas assembler is called on the .s file to create a representation of the microcode. If the -a option was used, jas will attempt to analyze the state transitions and assign unique state representations so that a stack is not necessary. Jas creates a Lager style parameter file with the extension .par, which is immediately transformed by the script to the extension .par.org. This file is then processed by the script compact-parw, which attempts to compact the PLA descriptions using the program espresso. The output of this process is a file with the .par extension. This file can then be used in DMoct to get the processor layout. As a final step, the .par file is parsed by an area estimation program, either KAest or LAest, depending on the architecture. The estimates are then written to the log file and into the database.

While the process is highly automated, several features can be exploited by performing the compilation by hand. The kc compiler was written in common lisp, and can create simulation models, both fixed point and floating point. These can be very useful in debugging program descriptions. A profiler is also available for the simulators generated by the compiler. The profiler provides information about machine cycle counts for the target processor.

BUGS

The script relies on the user's environment for program paths.
Abnormal exits from programs may not be handled correctly.

AUTHOR

Monte F. Mar

REFRENCES

For details on using the C-to-Silicon Tools, refer to

L. E. Thon, K. Rimey, and L. Svensson, "From C to Silicon", in R. W. Brodersen, editor, "Anatomy of a Silicon Compiler", pages 251-268, Kluwer Academic, Norwell, Mass., 1992.

NAME

candi

program to quantize coefficients, providing CSD coding.

SYNOPSIS

candi [options]

OPTIONS

-a	Run candi in an automatic mode
-i [filename]	input file created by digest
-o	output file listing information about the candi session
-c	output file listing only the final coefficients

DESCRIPTION

candi is a program that will allow interactive quantization of filter coefficients according to a point-wise specification. The program uses the specification to guide the process and to maintain a feasible set of coefficients. The input for the program is generated using the program digest, which creates a matrix representation of the network.

This version of *candi* has been modified to accept command line arguments and to provide an automated design mode. Within the program, there are basically 3 methods for examining the search space: a univariate search, a local search, and a multivariate search. In the interactive mode, users can try the different search methods in an effort to minimize the number of nonzero bits in the coefficients. In the automated mode, only a univariate search is performed.

The program searches the coefficient space to establish feasible sets. Thus execution time is directly proportional to the number of coefficients being quantized. For large numbers, the search time can be quite long.

AUTHOR

Rajeev Jain

REFERENCES

See the candi and digest manuals for more detailed explanations.

R. Jain, et. al., "Efficient and Accurate Mutiparameter Analysis of Linear Digital Filters Using a Multivariable Feedback Representation", IEEE Trans. on CAD, CAD-32:225-235, March, 1985.

NAME

coef2rl

code generator to translate quantized coefficients to rl code descriptions

SYNOPSIS

coef2rl [options] viewname

OPTIONS

-L [filename]	write log to filename
-m [integer]	integer indicates ordering when merging several RL files

DESCRIPTION

Coef2rl is a code generator program for digital filters. It expects an input the component view format containing quantized coefficients. The program works for FIR, Biquad IIR, Lattice Wave Digital IIR, and polyphase N-th band IIR filters and could be extended for other topologies. A straightforward mapping from coefficients to code is performed. No optimization is applied to the code, compilers should be able to figure the proper procedure. The output code is in the RL (Rimey's Language) format, suitable for the kc compiler.

In the current state, the program handles decimation correctly, but this may not be true for interpolation.

The output code is valid RL code, but might be modified to add extra flags or data conditions. The wordlength used in the processor is established using the INTERNAL_WORDLENGTH property. The proper architecture file to use is indicated through the use of the ARCH_FILE property. When the -m flag is used, the generated code forms a subroutine that can be called by another module. The program will annotate the database with a property called MERGE that allows subsequent calls to *coef2rl* to identify the ordering of the subroutines. If coef2rl is called on a view that has instances, it will generate code in the correct order by using the MERGE property to order the module calls. This capability is only supported for 1 level of subroutine calls.

This program will generate an output with the same name as the input view with the extension .k appended.

AUTHOR

Monte Mar

REFERENCE

For more information on the syntax of RL, see

Kenneth E. Rimey, "A Compiler for Application-Specific Signal Processors", Ph. D. Thesis, U. C. Berkeley, January, 1990.

NAME

coef2sil

code generator to translate quantized coefficients to silage code descriptions

SYNOPSIS

coef2sil [options] viewname

OPTIONS

-L [filename]	write log to filename
-m [integer]	integer indicates ordering when merging several silage files

DESCRIPTION

Coef2sil is a code generator program for digital filters. It expects an input the component view format containing quantized coefficients. The program works for FIR, Biquad IIR, Lattice Wave Digital IIR, and polyphase N-th band IIR filters and could be extended for other topologies. A straightforward mapping from coefficients to code is performed. No optimization is applied to the code, compilers should be able to figure the proper procedure. The output code is in the silage format, suitable for the kc compiler.

In the current state, the program handles decimation correctly, but this may not be true for interpolation.

The output code is valid RL code, but might be modified to add extra flags or data conditions. The wordlength used in the processor is established using the INTERNAL_WORDLENGTH property. The proper architecture file to use is indicated through the use of the ARCH_FILE property. When the -m flag is used, the generated code forms a subroutine that can be called by another module. The program will annotate the database with a property called MERGE that allows subsequent calls to *coef2sil* to identify the ordering of the subroutines. If *coef2sil* is called on a view that has instances, it will generate code in the correct order by using the MERGE property to order the module calls. This capability is only supported for 1 level of subroutine calls.

This program will generate an output with the same name as the input view with the extension .sil appended.

BUGS

The output programs have not been verified with a silage compiler. This should be done before the code is released

AUTHOR

Monte Mar, mmar@zabriskie.berkeley.edu

REFERENCE

For more information on the syntax of silage, see

P. Hilfinger and J. Rabaey. DSP Specification Using the Silage Language. In R W Brodersen, editor, *Anatomy of a Silicon Compiler*, chapter 15, pages 199–220. Kluwer Academic, 1992.

NAME

comp2schem

program to convert information in the component view to a PTOLEMY flowgraph.

SYNOPSIS

comp2schem [options] view:type

OPTIONS

-L [logfile]	file to use to log output information.
-t	attempt to trace the nets, recreating the connectivity in the component view.

DESCRIPTION

comp2schem translates information found in the component view to the schematic policy followed by PTOLEMY. The output of this program can be used as a galaxy in a PTOLEMY simulation. In order to map the instances in the component view, *comp2schem* makes use of the PTOLEMY_MODEL property. The value of this property is the actual ptolemy star or galaxy that should be implemented in the flowgraph. A file called cellpath must be placed in the current working directory. In this file, the paths to the PTOLEMY stars must be included, so comp2*schem* can locate the stars and find information about terminals and model parameters.

In order to map component view data to a PTOLEMY star, *comp2schem* uses a lookup table to establish relationships between component view properties and PTOLEMY model parameters. There are 2 sources for establishing the lookup table. The file comp2schem.def outlines system wide correspondences for mapping properties. In addition, mappings can be specified in the input text file for the component view. These values are stored in the PTOLEMY_PARAMETERS bag.

If the -t option is used, the program assumes that each component view instance has only 1 terminal with the property INPUT and 1 terminal with the property OUTPUT. A 1 to 1 mapping is then used to create a connected flowgraph with terminals for a galaxy. If the -t option is not used, the stars are placed randomly in the flowgraph and the user can connect them in any order.

Besides the output schematic view, the program will sometimes create files that will be loaded as parameters when the flowgraph is run. This is due to the fact that long strings of coefficients can't be stored in the OCT database according to the policy implemented by PTOLEMY. These files are given names like component_view_name.array, and there format is explained in the definition of the component view policy.

AUTHOR

Monte F. Mar, mmar@zabriskie.berkeley.edu

SEE ALSO

DMoct(1), vmake(1), sdl(5)
Oversampling Converter Design System documentation for details on OCT policies.
Oversampling Converter Design System documentation on PTOLEMY SDF models.
The ALMAGEST: PTOLEMY Users Manual

NAME

decfir

program to calculate area estimates and parameters for a family of FIR decimation filters.

SYNOPSIS

decfir [options] filename

OPTIONS

-d	debug mode. Shows information about the impulse response on stderr.
-L [filename]	Log file for output.
-A	If this flag is used, the remaining command line options are enabled.
-f [filename]	File containing the integer coefficients
-m [string]	An arbitrary name for the design. Output files will use this as a base name.
-l [integer]	Length of the impulse response for the filter.
-r [integer]	The decimation ratio for the filters.
-w [integer]	The internal word length to use in the filter.
-c [integer]	The number of digital clock cycles for each analog cycle, or the L/D ratio.
-S [integer]	Use coefficients to implement a sinc window. The power of the window is determined by the argument to the -c flag. Input to the -f option is ignored.
-a [integer]	Only do the area estimation.

DESCRIPTION

decfir calculates the parameters necessary for the implementation of a decimating FIR filter. The architectures have been described in the Lager structure description language (SDL). The FIR filters make use of a structure similar to those reported by Friedman and Leung.

If the decimation ratio is large and the filter length is suitably short, it is possible to use a very effective polyphase decomposition of an FIR filter. Since only samples at the output rate need to be calculated, the filter only needs to store the number of variables given by the L/D ratio. The L/D ratio is the length of the filter divided by the decimation ratio. To save on hardware, it is possible to update the state variables at a higher rate than the incoming sample rate.

Normally, the program will take component view information for input. Only the view name needs to be specified; all the information should be found in the view. The program can also be used in stand-alone mode by specifying the -A flag. As a design example, suppose you wanted to implement a filter of length 164 and you want to decimate by 64. The L/D ratio is 2.56, but it must be rounded up to 3. We choose to place the coefficients in a file called fcoefs and to call the design newfir. The command line to use is:

```
elias 21> decfir -f fcoefs -m newfir -l 164 -r 64 -c 3
```

decfir will return some statistics on area consumption based on the current implementation of SCMOS cells using lambda = 1.0µm. It will also create several files ending with .esp and one file ending with .par. These files are the parameter files used by the Lager Silicon Assembly system.

Instead of specifying a value for length and a file of coefficients, the -S flag can be used. In this case, coefficients for a sinc window will be calculated. For the L/D=1 and L/D=2 cases, a counter scheme is used eliminating the need for a coefficient ROM.

AUTHOR

Monte F. Mar mmar@zabriskie.berkeley.edu

REFERENCES

Friedman, V., et. al., "A Dual-Channel Voice-Band PCM Codec Using Sigma Delta Modulation Technique", IEEE Journal of Solid State Circuits, vol. 24, No. 2, April, 1989, pp. 274-280.

Leung, B. H-K., "Multi-Channel PCM A/D Interfaces Using Oversampling Techniques", Ph.D. Thesis, U. C. Berkeley, Nov. 1987.

NAME

doc

program to document steps in the design process

SYNOPSIS

doc [options] design_name

OPTIONS

-L [logfile]	file to use to log output information.
-s [string]	string names the stage of the design being completed.
-e	use emacs rather than vi for data entry.
-r	record data found in the estimates bag into the documentation.

DESCRIPTION

This program creates a separate view called documentation in the current working directory where consecutive entries are held. A template is written describing the design and asking various questions. After a design stage is completed, extra information can be added to the file through a vi or emacs editing window. The program can be used in the vmake script to force designers to make documentation entries.

The results of this program can be printed out using the printdoc program.

AUTHOR

Monte F. Mar

SEE ALSO

printdoc(1)

NAME

fr

program to calculate magnitude responses for a variety of filters.

SYNOPSIS

fr [options] view

OPTIONS

-L [logfile]	file to use to log output information.
-f [string]	use the string to find the name of the first filter instance in a cascaded implementation.
-i	use ideal coefficients to calculate the response (default).
-c	use coded coefficients to calculate the response.
-n [integer]	specify the number of point to calculate the frequency response (default 256).
-m	use xgraph(1) to display the magnitude data.
-p	display phase data. Only works if -f option is not specified.
-a	display aliasing data. Only works if -f option is specified.

DESCRIPTION

This program is meant to be used as part of the Oversampling Converter Design System. The input should be an OCT component view with information about some filter designs, especially the FILTER_-STRUCTURE property, which determines the algorithm used to calculate the response. Depending on the design state, the frequency response can be calculated using ideal coefficients or coded coefficients.

The program is able to calculate the composite response of a cascade of filters if they are specified in the input component view. If decimation is used in a cascade, the program will calculate the response with respect to the highest sampling rate in the system, i.e. the input sampling rate. The number of points specified will refer to the number of points in used in at the lowest sampling rate. Thus if the number of points is 256, and a cascade consists of a decimate by 8 followed by a single rate filter, there will be 2048 total points in the output response.

The program writes out the following files, using the viewname for the file stub.

view.mgn - magnitude response in decibels.
view.phase - phase response.
view.alias - composite alias if a full scale signal were applied.

The output files are written in two columns, with the frequency parameter first and the data parameter second. This format is suitable as input for xgraph(1). The alias file consists of a summation of all the images that will be aliased into the low pass band.

If frequency domain specifications are found in the input view, the program will calculate the response at these points and check to make sure that the magnitude specifications are within the tolerances. If the response is outside these bounds, error messages are logged, and the program returns a non-zero status.

Optionally, the data created by the program can be displayed automatically if the proper command line options are specified.

BUGS

The program is currently set only for decimation. Modifications are needed to extend to the case that handles interpolation in the filter cascades.

AUTHOR

Monte F. Mar

NAME

lpfir

program to calculate coefficients for equiripple FIR filters.

SYNOPSIS

lpfir [options] filename

OPTIONS

-c	interpret the input file as an oct view (default).
-a [filename]	interpret the input file as ASCII input.

DESCRIPTION

lpfir implements the equiripple FIR design program presented by Parks, McClellan, and Rabiner. The input specification can either be in an OCT component view or ASCIII. The program has been extended to allow for arbitrary band descriptions to be entered. This feature has been used to design filters with droop compensation.

The ASCII input description is almost identical to the one accepted by the FORTRAN implementation of this program published in the original paper. Two examples of ASCII inputs will be given here.

```
32 1 3 0 0
0.0 0.1 0.2 0.35
0.425 0.5
0.0 1.0 0.0
10.0 1.0 10.0
```

This example comes form the original FORTRAN source code for the algorithm. Note that this version requires no commas between data entries. This input data specifies a length 32 bandpass filter with stopbands from 0 to 0.1 and 0.425 to 0.5 and passband from 0.2 to 0.35 with weighting of 10 in the stopbands and 1 in the passband. The grid density defaults to 16.

The new mode is called type 4. To use the type 4 filter specification, you must specify the number of bands. The type 4 specification is meant to modify a regular band pass filter to compensate for droop or to provide droop in the passband. Three groups of points are stored. The first group gives a listing of the frequency points. The second group gives the expected magnitude of the filter response. The third group give the weights associated with the magnitude values. An example is given below.

```
128 4 2 0 21
0.000000 0.250000 0.275000 0.500000
0.000000 0.025000 0.050000 0.075000
0.100000 0.125000 0.150000 0.175000
0.200000 0.225000 0.250000 0.275000
0.300000 0.325000 0.350000 0.375000
0.400000 0.425000 0.450000 0.475000
0.500000
1.000000 1.018466 1.033445 1.050814
1.070699 1.093252 1.118650 1.147104
1.178857 1.214194 1.267958 0.000000
```

```
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000
1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000
16.000000 81.000000 256.000000 625.000000
1296.000000 2401.000000 4096.000000 6561.000000 10000.000000
```

This example describes a 128 tap low pass filter with a passband modified to compensate for a sinc shaped droop in the passband. Note that there are 21 frequency points defined, and that the standard stop-band is not weighted uniformly.

The program writes output data similar to what the FORTRAN program originally printed. If an OCT input view was specified, the impulse response will be stored in the same view.

OCT component views suitable for lpfir can be generated using the vmake utility.

BUGS

The code has not been tested thoroughly. The maximum length for a filter can be set in a header file in the source code.

AUTHOR

Monte F. Mar

REFERENCES

J. H. McClellan, T. W. Parks, and L. R. Rabiner, "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters", IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No. 6, December 1973, pgs. 506-515.

NAME

parser

parser for converting .adi files to OCT component view. .adi files are the input to the oversampling converter design system.

SYNOPSIS

parser filename

DESCRIPTION

parser is called by the vmake program. The main function is to map text entries into the policy defined for the OCT component view. The input syntax for the parser is defined in terms of several blocks which are described in the following paragraphs. The parser supports C-style syntax for comments. A sample input file is given here.

```
name speech;
subcells {
 dsm2pb2 DSM;
 firld1 FILTER1;
 fourth FILTER2;
 half FILTER3;
 e2 FILTER4;
}
pinlist {
 parent IN INPUT;
 parent OUT OUTPUT;
 parent CLOCK CLOCK;
 input parent IN;
 input DSM IN;
 mid1 DSM OUT;
 mid1 FILTER1 IN;
 mid2 FILTER1 OUT;
 mid2 FILTER2 IN;
 mid3 FILTER2 OUT;
 mid3 FILTER3 IN;
 mid4 FILTER3 OUT;
 mid4 FILTER4 IN;
 out FILTER4 OUT;
 out parent OUT;
property {
 SAMPLING_FREQ 1.0;
 DEC_RATIO 256;
 ARCH_FILE frisc2;
}
freqSpec {
 0.000 2.00 0.05dB 1.0 edge;
 0.225 2.00 0.05dB 1.0 edge;
 0.275 0.00 60.0dB 1.0 edge;
 0.500 0.00 60.0dB 1.0 edge;
design {
 fr -f FILTER2;
}
mapping {
 coef2rl;
}
```

The name line gives the name of the component view that the data will be written to. This need not correspond with the actual name of the file. In what follows, ordering of the lists must be maintained, but some lists are optional.

The instances contained in the view are given in the subcells list. Each entry in the list consists of 2 fields. The first field gives the name of the view to be instantiated. The second field is the instance name. If a component view is not found using the information in first field, the parser will attempt to create the component view. The file cellpath in the current working directory is used to provide paths to search for this file. The component view will then be created in the current working directory. This list is optional.

The pinlist consists of entries with 3 fields separated by spaces and terminated by a semi-colon. It is used to specify connectivity of instances found in the view. If the terminal is found on the parent, the first field is set to parent and the second field gives the name of the terminal. The third field denotes the property to be associated with the terminal. Valid property values are INPUT, OUTPUT, CONTROL, CLOCK, TEST, MISC. If the terminal is found on an instance, the first field is used to specify the instance name that the terminal is connected to. The second field gives the name of the terminal. The third field assigns a net name to the terminal. Two terminals are connected by assigning them the same net name in the pin list. This list is optional.

The freqSpec list provides the ability to specify custom frequency responses. Each entry in the frequency specification consists of 5 fields. The first field is for the frequency point. The second field is the expected magnitude value. Optionally after this entry, you can put the character string 'dB' to indicate that the value is in dB. The next field is the deviation from the expected magnitude value. Again the string 'dB' may follow this. The field following the deviation is a weight value to specify the emphasis to put on meeting the specification at that point. This is currently only used by the lpfir design program. The final keyword 'edge' should be added to entries that make up the band edges. This list is optional.

The property list consists of entries with 2 fields. The first field gives the property name, and the second gives the property value. Only a single field is read after the property name, so strings with spaces are not allowed.

Following the property list, 3 lists can be given. These lists are titled design, quantize, mapping, and smv. Each list consists of entries that are command lines for a given program, without the view name specified. The parser will add the view name automatically. These 4 lists are translated to the DEPENDENCY bag that will be used by vmake for time stamp information.

The parser works by using 2 passes. In the first pass, data is parsed in to internal data structures. On the second pass, data is written to the component view. If an error is encountered, the parser will attempt to continue until a new section is recognized. At the end, the parser will print the number of errors encountered. It returns a non-zero value if parsing was not successful.

AUTHOR

Monte F. Mar

SEE ALSO

vmake(1)
Oversampling Converter Design System documentation for details on OCT policies.

NAME

pisogen

program to create parameters for the library parallel-to-serial converter.

SYNOPSIS

pisogen [options]

OPTIONS

-w [integer]	wordsize of the bit parallel input.
-m [filename]	name for the parameter file to be created, use an .esp extension.

DESCRIPTION

Pisogen creates the state machine description needed by the piso architecture template. The output file is suitable for as input for espresso, allowing the resulting PLA to be minimized. In addition to this file, a second parameter file must be created by the user. This file must include the following information:

```
(statewidth 7)
(wordlength 17)
(inwidth 7)
(outwidth 7)
(espresso "filename.esp")
```

Statewidth, inwidth, and outwidth all have the same value, which can be calculated as log2(-wordlength) + 2.

AUTHOR

Monte F. Mar

NAME

polyiir

program to design N-th band polyphase IIR filters

SYNOPSIS

polyiir [options] viewname

OPTIONS

-L [logfile]	file to use to log output information.
-A	Use the ascii command line. If this flag is used, must specify at least then,c,a,b flags. If not used, o,n,c,a,b,w,l,s flags are ignored and viewname is not necessary.
-d	debug at level given by integer.
-m [filename]	write magnitude response to this file.
-p [filename]	write phase response to this file.
-o [filename]	output file for the design data.
-s [filename]	save high precision coefficients to this file.
-n [integer]	number of points in freq. response (default 256).
-c [integer]	fraction of 1.0 for the filter cutoff frequency.
-a [integer]	number of attenuation zeroes desired.
-b [integer]	number of branches (ie decimation ratio).
-w [filename]	save wdf coefs. in save file rather than biquad coefs.
-l	approximate linear phase response.

DESCRIPTION

Polyiir is based on the algorithm presented by Renfors and Saramaki. It will design case 1 Nth band IIR filters. Only a subset of the entire algorithm was implemented, so convergence problems may be encountered for high orders of attenuation zeroes.

The program can generate coefficients for biquad implementations, or wave digital filter implementations using all pass filters in the branches.

From the component view, the property ORDER defines the number of attenuation zeroes, DEC_RATIO determines the number of branches, and FILTER_STRUCTURE determines whether biquads or wave digital coefficients are calculated. The cutoff is determined from the frequency specification.

BUGS

Since only part of the algorithm was implemented, convergence problems are encountered above 4 attenuation zeroes for the nonlinear phase case, and above about 9 attenuation zeroes for the approximate linear phase case.

AUTHOR

Monte F. Mar

REFERENCES

M. Renfors and T. Saramaki, “Recursive Nth-Band Digital Filters - Part I:Design and Properties”, IEEE Transactions on Circuits and Systems, Vol. CAS-34, No. 1, Jaunuary 1987, pgs. 24-39.

NAME

printdoc

program to print documentation collected with the doc program

SYNOPSIS

printdoc [options] documentation_view

OPTIONS

-L [logfile]	file to use to log output information.
-s [filename]	print documentation to this file.

DESCRIPTION

This program prints the documentation entered using the doc program. Entries are sorted according to the order that they were entered, and according to name. The output file is in ASCII format.

AUTHOR

Monte F. Mar

SEE ALSO

doc(1)

NAME

sCandi

script to quantize coefficients, providing CSD coding.

SYNOPSIS

sCandi [options] viewname

OPTIONS

-L [logfile]	run candi in an automatic mode
-s	save intermediate files created for digest.
-a	run candi in automatic mode.

DESCRIPTION

sCandi is a Perl script written to automate the process of file processing for using the *candi* program. This allows complete automation from filter coefficients stored in the component view to quantized coefficients written to the component view.

The script first calls the program *coef2dig* to create an input for *digest*. Next digest is run to generate an input file for *candi*, which is called immediately after. *Candi* can be run in automated mode, where only a univariate search is performed. In the interactive mode, other searches can be performed that will minimize the number of nonzero coefficients further. After *candi*, *candi2db* is called to write the coefficients into the database.

AUTHOR

Monte Mar

SEE ALSO

coef2dig(3), digest manual, candi(1), candi manual, candi2db(3)

REFERENCES

See the candi and digest manuals for more detailed explanations.

NAME

sincwin

program to design FIR coefficients for filters that have powers of sin(x)/x for the magnitude response.

SYNOPSIS

sincwin [options] view

OPTIONS

-L [logfile]	file to use to log output information.
-o	order for the sinc window (power that sin(x)/x is raised to).
-b	number of bits for the coefficient representation.
-r	decimation ratio to be implemented.
-t	use truncation to obtain output coefficients. Default is to use rounding.

DESCRIPTION

Sincwin calculates the filter coefficients to implement a window that has a sin(x)/x response raised to a power. The coefficients are easy to generate. If the decimation ratio is R and the order is N, the length will be R - N + 1, and the coefficients are given by the N convolutions of a pulse train of length R. The program only works for orders from 1 to 6.

The program will optionally round the coefficients or truncate them. Quantization will only occur if the number of bits in the coefficient representation is set smaller than that required by the largest coefficient.

If any of the flags o, b, r, or t are not specified, the program will look for the information in the input view. The ORDER property corresponds to the o flag, COEF_WORDLENGTH to the b flag, DEC_RATIO to the r flag, and CODING_METHOD to the r flag.

AUTHOR

Monte F. Mar

NAME

vmake

design manager program for the Oversampling Converter Design System

SYNOPSIS

vmake [options] view:type

OPTIONS

-L [logfile]	file to use to log output information.
-d	print out verbose information on the make process.
-n	run vmake without execution of commands.
-t [design I quantize I mapping I smv]	run the make process, but stop at the end of this target.

DESCRIPTION

vmake is a design manager for the Oversampling Converter Design System. Starting with a input text file, vmake converts the text file to an OCT component view, then proceeds to track design progress, verifying that targets are up to date. This program was meant to be a general purpose program that could be extended to handle general processing and time stamp checking of arbitrary OCT views.

The programs expects to find a time stamp information within a DEPENDENCY bag in the OCT views that it processes. The DEPENDENCY bag is created by the program parser(1) which is called by vmake. All time stamps are handled by vamke itself, so none of the programs executed by vmake must handle this. In the case of input text files and OCT views that do not require DEPENDENCY bags, time stamps are inferred from the last date of modification obtained from the stat(3) function.

AUTHOR

Monte F. Mar

SEE ALSO

parser(1), stat(3)
Oversampling Converter Design System documentation for details on OCT policies.

NAME

writeSDL

program to convert top level component view to SDL description of a module.

SYNOPSIS

writeSDL [options] view:type

OPTIONS

-L [logfile]	file to use to log output information.

DESCRIPTION

Once all the design steps have been finished in the Oversampling Converter Design System, a the final component view has enough information to allow implementation of each of the instances in the view. But without extra information about connectivity, a full structure_master view cannot be developed from the component view. The purpose of this program is to attempt to write a skeletal version of an SDL file for the input component view.,

The basic algorithm for writeSDL is as follows. Each instance in the view is examined to see if contains mapping information specifying a structure_master view corresponding to the instance. If it does not, the property LIBRARY_CELL is searched for. Once a structure_master view has been located, information about the formal terminals and formal properties is recorded. If the structure_master view does not exist, an attempt is made to call DMoct to generate it. After this information is gathered for each instance, the sdl file is generated. Since information about interconnect is not guaranteed in the component view, writeSDL creates the SDL file in pinlist format, which can be easily edited.

This program assumes that mapping tools create parameter files by taking the SMV parameter name and then prepending the component instance name. This must be done to insure uniqueness of the property names in the case where 2 SMVs have properties with a common name, but different values. Within the pinlist, the program sorts the terminals and groups them according to the terminal type assigned in the SMV by the TERMTYPE property or also by the NETTYPE property, if the formal property was attached to a net. This was done to help users sort out the terminals when editing the output sdl file.

The program will attempt to match names on component instances with formal SMV terminals in order to preserve connectivity. If there is a match between the names, the program will attempt to find the net that the component view terminal is attached to, and then assign that net name to the net in the SDL file. This means that if the correct terminal net specification is placed in the component view, the resulting SDL file will be valid, and DMoct can be invoked as part of the vmake process.

To connect modules that have different wordlengths, writeSDL provides the information specified in the INTERNAL_WORDLENGTH, BINARY_POINT, and WORD_ALIGNMENT properties. For the input net of an instance, correct connection is specified by aligning the binary point with that of the preceding instance. The binary points should be offset to the right by the amount specified in the WORD_ALIGNMENT property. This alignment is performed using the net-base and term-base properties of the SDL language.

AUTHOR

Monte F. Mar

SEE ALSO

DMoct(1), vmake(1), sdl(5)
Oversampling Converter Design System documentation for details on OCT policies.

NAME

candi2db

program to parse the output coefficient file from candi, writing results to the database

SYNOPSIS

candi2db [options] -i [filename] viewname

OPTIONS

-L [filename]	write log to filename
-i [filename]	input file created by candi

DESCRIPTION

candi2db is a support program written so that the *sCandi* script could be realized. The main problem was to get the coefficient information from the candi output into the database in the component view. This translation presents a problem, since the database names for the coefficients and the names used in *Candi* are usually not the same. To get around this problem, the program *coef2dig* creates tags and annotates these to the database. The job of *candi2db* is then to read the tags out of the database and then sort the coefficients from the output coefficient file so that the proper quantized value is stored in the database.

AUTHOR

Monte Mar

SEE ALSO

sCandi(1), candi(1), coef2dig(3)

NAME

coef2dig

program to translate database coefficients to format suitable for the program digest

SYNOPSIS

coef2dig [options] viewname

OPTIONS

-L [filename]	write log to filename
-o [filename]	input file created by candi

DESCRIPTION

Coef2dig is a support program written so that the *sCandi* script could be realized. *Candi* is used to generate quantized coefficients that still meet a design specification. This program translates a set of coefficients of a known filter type to an input file suitable for *digest*. In order to do this, the program reads the database, identifying the proper filter type using the FILTER_STRUCTURE property, and then annotates the coefficients with the tag that is used in the *digest* file, and the description is written.

This program need not only be used in the *sCandi* script. The file generated by this program can be modified and used with *digest* to obtain frequency response information and other analyses can be performed. See the *digest* users guide for more information.

Note that the frequency specification points that *candi* uses for evaluation are the ones held in the FREQUENCY_SPEC bag in the component view.

AUTHOR

Monte Mar

SEE ALSO

sCandi(1), candi(1), candi2db(3), digest manual

NAME

KAest

program to estimate area that a PUMA processor will take

SYNOPSIS

KAest [options] viewname

OPTIONS

-L [filename]	write log to filename
-w	write results into the database
-l [float]	size of lambda in microns

DESCRIPTION

KAest performs area estimation based on the parameter file for a C-to-silicon design. The area of the major functional blocks is calculated, and then overall area is estimated using an assumed floorplan.

The routine is based on simple mathematical formulas.

AUTHOR

Monte Mar

SEE ALSO

c2sil(1)

NAME

LAest

program to estimate area that a Lambda processor will take

SYNOPSIS

KAest [options] viewname

OPTIONS

-L [filename]	write log to filename
-w	write results into the database
-l [float]	size of lambda in microns

DESCRIPTION

LAest performs area estimation based on the parameter file for a C-to-silicon design. The area of the major functional blocks is calculated, and then overall area is estimated using an assumed floorplan.

The routine is based on simple mathematical formulas.

AUTHOR

Monte Mar

SEE ALSO

c2sil(1)

NAME

optiir

program to optimize the design coefficients for cascaded biquad IIR filters.

SYNOPSIS

optiir

DESCRIPTION

Optiir is a program that will attempt to alter a transfer function to meet a specified deviation. The program makes use of sequential unconstrained optimizations to force convergence. Each of the optimization problems has been formulated so it can be solved used the Fletcher-Powell algorithm.

This program is taken directly from the IEEE collection, and requires input data to be in the format described in the documentation. It reads stdin and writes stdout. A call can be made to *optiir* from within the *biquads* program. *Biquads* formats the information in the proper format.

It has been noticed that this program will give differentt output results on different computers. It appears to be compiler dependent.

SEE ALSO

biquads(1)

REFERENCES

Programs for Digital Signal Processing, IEEE Press, 1979.

APPENDIX C

An Example Design Trace

Tutorials for the design system can be found in the distribution. The root directory of the installation should be set in the environment variable OCDS. The tutorials can be found in the directory $OCDS/hypertext. You can view these documents using Framemaker or Frameviewer. If you do not have Framemaker, you can print out the postscript versions also found in that directory. In addition to the tutorials, several examples are provided in the directory $OCDS/examples. You can run these examples to see how the system works. Separate README files are included with each example to tell how to run the example and what it should do.

C.1 Creating Design Specifications

The design starts by specifying the peak absolute and minimum voltages that need to be processed by the converter. Additionally, the designer should be able to answer these questions:

```
Magnitude Information
The peak magnitude of the signal is: 1 Volt
The minimum signal magnitude of interest is: 100 microVolts

Frequency Characteristics of the Signal
The highest frequency of interest is: 4.75 kHz

Desired Signal Conditioning
Is a magnitude droop acceptable across the passband? No
If not, how much passband ripple can be tolerated? 0.1 dB
How much anti-alias attenuation is needed? 60 dB
How much bandwidth is needed for a transition region? 0.25 kHz
What output sampling rate is acceptable? 10 kHz
Should the phase response be linear? No
```

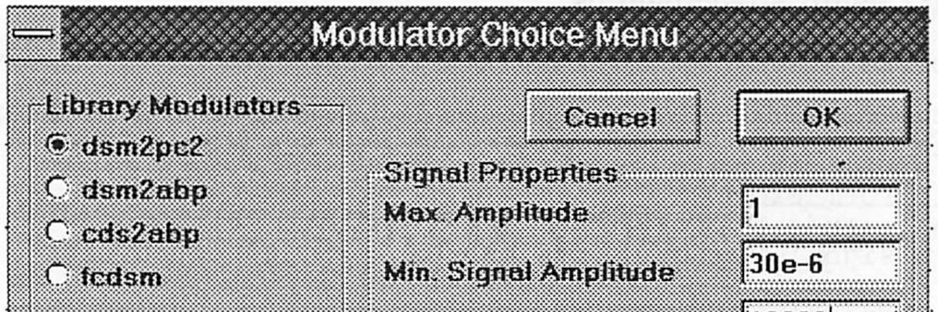
A comment should be made about the definition of transition region. When bandlimiting a signal, we cannot provide an infinitely steep rolloff. The transition region defines the point from the edge of the passband to half of the sampling frequency. Making this bandwidth small will greatly increase the complexity of filters but will allow more usage of the sampled bandwidth.

Some default parameters were given, and these will be used throughout this design trace. Using these parameters, some estimates can be made using some tools implemented in Microsoft Excel.

C.2 Spreadsheet Analysis Tools

The macro sheets for the spreadsheet tools are found in $OCDS/excel. They were created using MS Windows 3.1 and MS Excel 4.0. There are 3 versions of each macro sheet with extensions .xla, .xlm, and .csv. The .xla format is the add-in format. If you transfer these files to a PC or a Macintosh and install them in the excel/xlstart directory, they will be automatically loaded when you start Excel. The .xlm format is editable. You can load these macros using the open command from the file menu in Excel. The .csv format stands for comma separated values. These files were included to document the macros and equations used in estimation. The remainder of this discussion assumes that you are on a PC or a Macintosh, you have Excel running, and the macros were loaded either as add-ins or by opening all the .xlm macro sheets.

Using the data from the last section, we can choose a modulator and estimate the necessary oversampling ratio for achieving the specification. Figure C.1 shows the data entry menu for mod-


Modulator Choice Menu

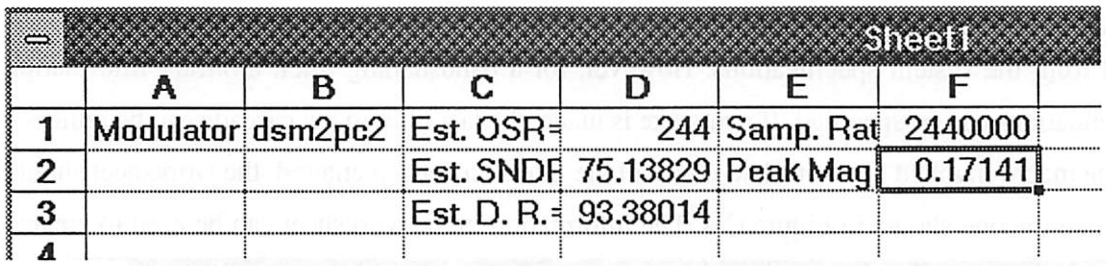
Library Modulators
- (●) dsm2pc2
- ( ) dsm2abp
- ( ) cds2abp
- ( ) fcdsm

Cancel OK

Signal Properties
- Max. Amplitude: 1
- Min. Signal Amplitude: 30e-6
- Output Sampling Rate: 10000

Figure C.1 Sample data entry box for estimating modulator parameters.

ulator analysis. This menu can be invoked by typing control-m. The macro returns estimation information to the current worksheet, and provides the information as shown in Figure C.2. The



Sheet1

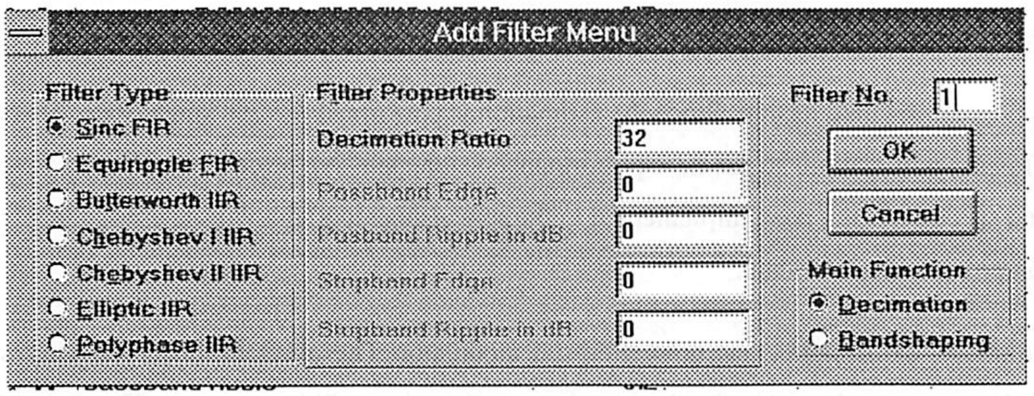
	A	B	C	D	E	F
1	Modulator	dsm2pc2	Est. OSR=	244	Samp. Rat	2440000
2			Est. SNDF	75.13829	Peak Mag	0.17141
3			Est. D. R.=	93.38014		
4						

Figure C.2 Results from the modulator estimation macro.

estimated oversampling ratio is 244, and the peak SNDR is expected to be only 75.1 dB. The usable dynamic range is 93.4 dB, but the user only requested 20log(1/10e-6) = 80 dB. Signal magnitudes about 0.17 V will have excessive harmonic distortion, and this voltage is estimated to be the peak magnitude for linear conversion. The information provided by this macro is based on ideal digital filtering, so some margins should be allowed for losses in digital filtering.

If droop is acceptable, like in low frequency applications, a simple CIC filter response will satisfy the digital filter requirements. No further estimation is needed, since the CIC or sinc FIR filters can be selected with order set 1 degree higher than the order of the modulator. The filter must implement a decimation ratio at least as large as the estimated oversampling ratio.

Since droop was not acceptable in the specification presented in Section C.1, a set of decimation filters need to be designed. A new worksheet can be created in Excel, and a template drawn by invoking a macro with control-t. Once the template is drawn, the user can enter values for the desired output sampling rate, passband edge, passband ripple, and anti-aliasing. Once this has been done, the Add Filter menu can be invoked by typing control-a and is shown in Figure C.3. The user



Add Filter Menu

Filter Type
- [✓ Sinc FIR]
- Equiripple FIR
- Butterworth IIR
- Chebyshev I IIR
- Chebyshev II IIR
- Elliptic IIR
- Polyphase IIR

Filter Properties

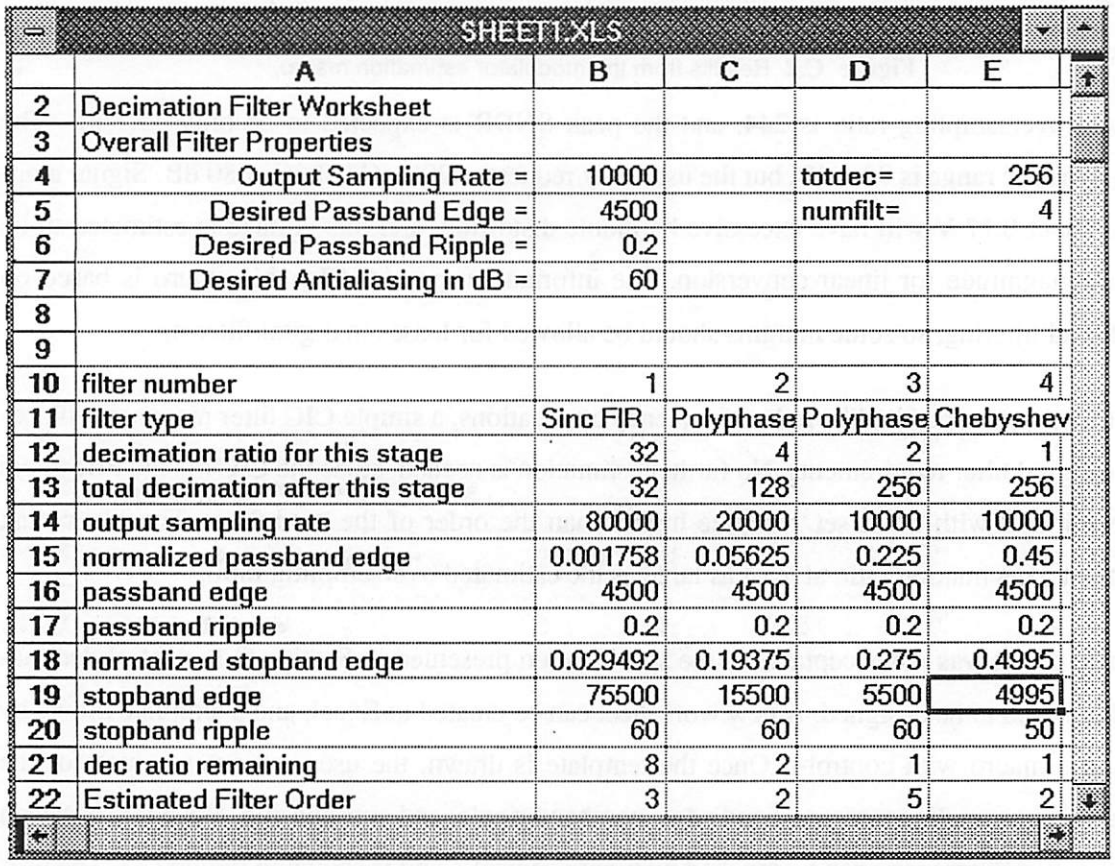
Decimation Ratio	32
Passband Edge	0
Pasband Ripple in dB	0
Stopband Edge	0
Stopband Ripple in dB	0

Filter No 1

Main Function
- [✓ Decimation]
- Bandshaping

Figure C.3 The Add Filter menu for filter parameter estimation.

has the choice of adding a filter as a decimation filter or a bandshaping filter. For a decimation filter, only the decimation ratio needs to be entered since the band edges and the attenuation can be derived from the system specifications. However, for a bandshaping filter, arbitrary information and attenuations can be specified. If a mistake is made, the last filter in the cascade can be removed using the macro invoked by control-w. Once all the filters have been entered, the worksheet should appear like the one shown in Figure C.4. The information in the worksheet can be used to created


SHEET1.XLS

	A	B	C	D	E
2	Decimation Filter Worksheet				
3	Overall Filter Properties				
4	Output Sampling Rate =	10000		totdec=	256
5	Desired Passband Edge =	4500		numfilt=	4
6	Desired Passband Ripple =	0.2			
7	Desired Antialiasing in dB =	60			
8					
9					
10	filter number	1	2	3	4
11	filter type	Sinc FIR	Polyphase	Polyphase	Chebyshev
12	decimation ratio for this stage	32	4	2	1
13	total decimation after this stage	32	128	256	256
14	output sampling rate	80000	20000	10000	10000
15	normalized passband edge	0.001758	0.05625	0.225	0.45
16	passband edge	4500	4500	4500	4500
17	passband ripple	0.2	0.2	0.2	0.2
18	normalized stopband edge	0.029492	0.19375	0.275	0.4995
19	stopband edge	75500	15500	5500	4995
20	stopband ripple	60	60	60	50
21	dec ratio remaining	8	2	1	1
22	Estimated Filter Order	3	2	5	2

Figure C.4 Completed worksheet for decimation filter design.

the input files for detailed design. Since the estimation is performed using a spreadsheet, it is easy to try different combinations to see how performance is affected. Performance estimators for area, power, and speed have not been implemented in the spreadsheet tool, but will be used in the design estimation process at a later date.

In summary, the designer has now chosen a modulator. The oversampling ratio guides the choice of decimation ratio and in the final implementation, they will be set equal. The digital filter or filters have been chosen through estimation, and all the basic parameters necessary for coefficient design are now available.

C.3 Input File Descriptions For Detailed Design

The input file descriptions are written in the design language used by the design system parser. The files should be created with a .adi extension, which stands for Analog to Digital Interface. Sample input files based on information in the last section are presented. There is a discrepancy for the file e2.adi, which is used to specify the final filter in the cascade. The filter implemented in this block is a lattice wave digital filter. No design program for lattice wave digital filters was incorporated in the system, but it is easy to design the filter coefficients according to Gaszi's method. The program a2oct can insert the coefficients into the database, and this is what was done. The file e2.adb lists these coefficients and is the input for a2oct.

Some other discrepancies will be noted between the data in Figure C.4 and the input descriptions listed in this section. The frequency response data in filter.adi has been altered so that more aliasing is allowed in the transition region. This is an acceptable practice since the noise aliased in this band is attenuated by the final 2 filters in the cascade. The sampling rate was changed from 10 kHz used in Figure C.4 to 20 kHz in the input descriptions. There are other discrepancies, but this reflects the fact that the design can be altered to take advantage of design properties to gain better implementations.

The input files can be found in the distribution in the directory $OCDS/examples/speech. They are reproduced here:

```
###
### file: speech.adi
###
name speech;
subcells {
                dsm DSM;
                firl FILTER1;
                procl PROC;
}
pinlist {
                parent IN INPUT;
                parent OUT OUTPUT;
                parent CLOCK CLOCK;
                parent EN CONTROL;
                input parent IN;
                input DSM IN;
                mid1 DSM OUT;
                mid1 FILTER1 IN;
                mid2 FILTER1 OUT;
                mid2 PROC IN;
                out PROC OUT;
                out parent OUT;
}
quantize {
                fr -c -n 32 -f FILTER1 -m -a;
                doc -s quantization_done;
}
mapping {
                writeSDL;
                doc -s mapping_done -r;
}

###
### file: dsm.adi
###
name dsm;
pinlist {
                parent IN INPUT;
                parent OUT OUTPUT;
                parent CLOCK CLOCK;
                parent EN CONTROL;
}
properties {
                PTOLEMY_MODEL DSM2;
                GAIN 1000.0;
                LIBRARY_CELL dsm2pb2;
}

###
### file: fir1.adi
###
name fir1;
pinlist {
                parent IN INPUT;
                parent OUT OUTPUT;
                parent CLOCK CLOCK;
                parent EN CONTROL;
}
properties {
                PTOLEMY_MODEL FIR;
                FILTER_STRUCTURE LINEAR_ARRAY;
                FILTER_CLASS LOWPASS;
                DEC_RATIO 32;
                SINC_ORDER 3;
                CODING_METHOD ROUNDING;
                INTERP_RATIO 1;
}
design {
                sincwin;
}
mapping {
                decfir;
                doc -r -s mapping_done;
```

```
}

###
### file: proc1.adi
###
name proc1;
subcells {
                fourth FILTER2;
                half FILTER3;
                e2 FILTER4;
}
pinlist {
                parent IN INPUT;
                parent OUT OUTPUT;
                input parent IN;
                input FILTER2 IN;
                midx FILTER2 OUT;
                midx FILTER3 IN;
                midy FILTER3 OUT;
                midy FILTER4 IN;
                out FILTER4 OUT;
                out parent OUT;
}
properties {
                WORD_ALIGNMENT 4;
                BINARY_POINT 2;
}
design {
                fr -m -n 256 -f FILTER2;
                doc -s design_done;
}
quantize {
                fr -m -c -n 256 -f FILTER2;
                doc -s quantize_done;
}
mapping {
                coef2rl;
                c2sil -l -a;
                doc -r -s mapping_done;
}

###
### file: fourth.adi
###
name fourth;
pinlist {
                parent IN INPUT;
                parent OUT OUTPUT;
                parent CLOCK CLOCK;
}
properties {
                SAMPLING_FREQ 1.0;
                DEC_RATIO 4;
                INTERP_RATIO 1;
                ORDER 2;
                FILTER_STRUCTURE POLYPHASE_LWDF;
                FILTER_CLASS LOW_PASS;
```

```
               PHASE_APPROX NON_LIN_PHASE;
               INTERNAL_WORDLENGTH 22;
               PTOLEMY_MODEL Polywdf;
               PROCESSOR_NAME proc1;
               ARCH_FILE lambda;
               BINARY_POINT 3;
               WORD_ALIGNMENT 4;
}
freqSpec {
0.0 4.00 0.05dB 1.0 edge;
0.0625 4.00 0.05dB 1.0 edge;
0.1875 0.00 53dB 1.0 edge;
0.1983 0.00 53dB 1.0;
0.2295 0.00 53dB 1.0;
0.2705 0.00 53dB 1.0;
0.3018 0.00 53dB 1.0;
0.3125 0.00 53dB 1.0;
0.4375 0.00 53dB 1.0;
0.5 0.00 53dB 1.0 edge;
}
design {
               polyiir;
               fr -m;
               doc -s design_done;
}
quantize {
               sCandi -a;
               fr -c -m;
               doc -s quantize_done;
}
mapping {
               coef2rl -m 1;
}

###
### half.adi
###
name half;
pinlist {
               parent IN INPUT;
               parent OUT OUTPUT;
               parent CLOCK CLOCK;
}
properties {
               SAMPLING_FREQ 1.0;
               DEC_RATIO 2;
               INTERP_RATIO 1;
               ORDER 5;
               FILTER_STRUCTURE POLYPHASE_LWDF;
               FILTER_CLASS LOW_PASS;
               PHASE_APPROX NON_LIN_PHASE;
               INTERNAL_WORDLENGTH 22;
               PTOLEMY_MODEL Polywdf;
               PROCESSOR_NAME proc1;
               ARCH_FILE lambda;
               WORD_ALIGNMENT 4;
               BINARY_POINT 3;
```

```
}
freqSpec {
0.0 2.00 0.05dB 1.0 edge;
0.225 2.00 0.05dB 1.0 edge;
0.275 0.00 60dB 1.0 edge;
0.2773 0.00 60dB 1.0;
0.280 0.00 60dB 1.0;
0.285 0.00 60dB 1.0;
0.2891 0.00 60dB 1.0;
0.295 0.00 60dB 1.0;
0.300 0.00 60dB 1.0;
0.3086 0.00 60dB 1.0;
0.315 0.00 60dB 1.0;
0.320 0.00 60dB 1.0;
0.325 0.00 60dB 1.0;
0.330 0.00 60dB 1.0;
0.335 0.00 60dB 1.0;
0.3418 0.00 60dB 1.0;
0.345 0.00 60dB 1.0;
0.350 0.00 60dB 1.0;
0.360 0.00 60dB 1.0;
0.370 0.00 60dB 1.0;
0.380 0.00 60dB 1.0;
0.3906 0.00 60dB 1.0;
0.410 0.00 60dB 1.0;
0.430 0.00 60dB 1.0;
0.450 0.00 60dB 1.0;
0.4609 0.00 60dB 1.0;
0.480 0.00 60dB 1.0;
0.5 0.00 60dB 1.0 edge;
}
design {
               polyiir;
               fr -m;
               doc -s design_done;
}
quantize {
               sCandi -a;
               fr -c -n 1024 -m;
               doc -s quantize_done;
}
mapping {
               coef2rl -m 2;
}

###
### file: e2.adi
###
name e2;
pinlist {
               parent IN INPUT;
               parent OUT OUTPUT;
               parent CLOCK CLOCK;
}
properties {
               FILTER_CLASS LOWPASS;
               ORDER 2;
               FILTER_STRUCTURE BIQUAD_CASCADE;
               FILTER_CLASS lowpass;
               MAG_APPROX chebyshevii;
               DEC_RATIO 1;
               SAMPLING_FREQ 20000;
               INTERNAL_WORDLENGTH 22;
               PTOLEMY_MODEL Polybiq;
               PROCESSOR_NAME proc1;
               ARCH_FILE lambda;
               BINARY_POINT 1;
               WORD_ALIGNMENT 2;
}
freqSpec {
               0.000000 0.000000dB 0.200000dB 1.0 edge;
               300.000000 0.0dB 0.200000dB 1.0;
               600.000000 0.0dB 0.200000dB 1.0;
               900.000000 0.0dB 0.200000dB 1.0;
               1200.000000 0.0dB 0.200000dB 1.0;
               1500.000000 0.0dB 0.200000dB 1.0;
               1800.000000 0.0dB 0.200000dB 1.0;
               2100.000000 0.0dB 0.200000dB 1.0;
               2400.000000 0.0dB 0.200000dB 1.0;
               2700.000000 0.0dB 0.200000dB 1.0;
               3000.000000 0.0dB 0.200000dB 1.0;
               3300.000000 0.0dB 0.200000dB 1.0;
               3600.000000 0.0dB 0.200000dB 1.0;
               3900.000000 0.0dB 0.200000dB 1.0;
               4200.000000 0.0dB 0.200000dB 1.0;
               4500.000000 0.0dB 0.200000dB 1.0;
               4800.000000 0.0dB 0.200000dB 1.0;
               5100.000000 0.0dB 0.200000dB 1.0;
               5400.000000 0.0dB 0.200000dB 1.0;
               5700.000000 0.0dB 0.200000dB 1.0;
               6000.000000 0.0dB 0.200000dB 1.0;
               6300.000000 0.0dB 0.200000dB 1.0;
               6600.000000 0.0dB 0.200000dB 1.0;
               6900.000000 0.0dB 0.200000dB 1.0;
               7200.000000 0.0dB 0.200000dB 1.0;
               7500.000000 0.0dB 0.200000dB 1.0;
               7800.000000 0.0dB 0.200000dB 1.0;
               8100.000000 0.0dB 0.200000dB 1.0;
               8400.000000 0.0dB 0.200000dB 1.0;
               8700.000000 0.0dB 0.200000dB 1.0;
               9000.000000 0.0dB 0.200000dB 1.0 edge;
               9900.000000 0.000000 30.0dB 1.0 edge;
               9990.000000 0.000000 30.0dB 1.0 edge;
}
design {
               a2oct;
               fr -m;
               doc -s design_done;
}
quantize {
               sCandi -a -s;
               fr -c -m;
               doc -s quantize_done;
}
```

```
mapping {
               coef2rl -m 3;
}

###
### file: e2.adb
###
name e2
instances
end
terminals
IN INPUT
OUT OUTPUT
CLOCK CLOCK
end
netlist
end
structure
FILTER_TYPE string LOWPASS
ORDER integer 3
FILTER_STRUCTURE string LATTICE_WDF
FILTER_CLASS string lowpass
MAG_APPROX string chebyshevi
DEC_RATIO integer 1
SAMPLING_FREQ real 20000
INTERNAL_WORDLENGTH integer 22
PTOLEMY_MODEL string Polywdf
PROCESSOR_NAME string proc1
end
specProps
end
frequencySpec
0.000000 2.000000 0.100000dB 1.0 edge
1000.000000 2.000000 0.100000dB 1.0
2000.000000 2.000000 0.100000dB 1.0
3000.000000 2.000000 0.100000dB 1.0
4000.000000 2.000000 0.100000dB 1.0
5000.000000 2.000000 0.100000dB 1.0
6000.000000 2.000000 0.100000dB 1.0
7000.000000 2.000000 0.100000dB 1.0
8000.000000 2.000000 0.100000dB 1.0
9000.000000 2.000000 0.100000dB 1.0 edge
9800.000000 0.000000 24.0dB 1.0 edge
9950.000000 0.000000 24.0dB 1.0
9990.000000 0.000000 24.0dB 1.0 edge
end
idealCoefs
latticeWdf
2
1
1
0
0
1
1
5
.237694593
```

```
0
0
5
.158963109
5
.023291743
end
codedCoefs
end
Destimates
end
connect
end
parameters
end
Mestimates
end
status
end
```

C.4 Output From A Design Session

The output from the design session using the input files of the last section is presented below. The script output has been annotated to indicate what actions were performed. Subtitles are used to break up the flow and to designate new steps initiated in the design process.

C.4.1 Preliminaries

These lines show the content of the current working directory, and the contents of the cellpath and comp2schem.def files. Both must be in the current working directory for the program to work.

```
Script started on Wed Jan 27 10:43:44 1993
elias 21> ls
cellpath        e2.adb          fourth.adi    script.out
comp2schem.def  e2.adi          half.adi      speech.adi
dsm.adi         fir1.adi        proc1.adi

elias 22> m cellpath
~ptolemy/src/domains/sdf/dsp/icons:~ptolemy/lib/technology/ptolem
y:~ptolemy/src/domains/sdf/icons:~cad/ADsystem/ptolemy/stars:~pto
lemy/src/domains/sdf/stars:~ptolemy/src/domains/sdf/dsp/stars:~/l
ambda:~cad/ADsystem/architecture/firld3:~/analog/leafcells:~/arch
tools/lambda:.

elias 23> ls
cellpath        e2.adb          fourth.adi    script.out
comp2schem.def  e2.adi          half.adi      speech.adi
dsm.adi         fir1.adi        proc1.adi

elias 24> m comp2schem.def
# This is a temporary version of comp2schem.def
# This should contain bindings between component view and ptolemy
model
# property names.
coefs values
taps values
values values
decimation DEC_RATIO
decRatio DEC_RATIO
interpolation INTERP_RATIO
order ORDER
attZero ORDER
intwl INTERNAL_WORDLENGTH
diffwl INTERNAL_WORDLENGTH
outwl OUTPUT_WORDLENGTH
```

C.4.2 The DESIGN Phase

In this phase, coefficient design is performed. When the files are run, the fr -m command forces vmake to display the frequency response on the users screen using xgraph. The doc program provides a template to record information.

```
elias 25> vmake -t design speech
vmake> parser speech.adi
Parsing completed successfully.
viewname = speech
parser> Opened facet with name: speech
parser> View proc1/component not found.
parser> Attempting to parse file proc1.adi.
Parsing completed successfully.
```

...Intermediate output was deleted...

C.4.3 The QUANTIZE Phase

In the second phase of detailed design, coefficient quantization was performed using the program candi. The automated mode was chosen. Once again, the fr program was used to verify that the resulting designs met specifications.

```
elias 26> vmake -t quantize speech
vmake> parser speech.adi
Parsing completed successfully.
viewname = speech
```

...Intermediate output was deleted...

C.4.4 A Small Digression

Before going to the mapping phase, 2 files were placed in the current working directory. It has been found that the kc compiler used in the mapping phase has a problem performing tilde expansion. To circumvent this, we place the 2 files in the current working directory. Since the kc compiler finds them there, no search and no tilde expansion are necessary. If we hadn't placed them there, an obscure message about a segmentation fault would be reported by the kc compiler.

```
elias 27> cp ~rl/etc/lambda.md .
elias 28> cp ~rl/etc/lambda.lsp .
```

C.4.5 The MAPPING Phase

In the final mapping phase, all parameters are determined. The output from the kc compiler has been trimmed.

```
elias 29> vmake -t mapping speech
vmake> Running vmake on master of instance PROC
vmake> Running vmake on master of instance FILTER4
vmake> coef2rl -m 3 -L e2.log e2
```

...Intermediate output deleted...

```
script done on Wed Jan 27 12:42:15 1993
```

Area estimators were run during this part of the design. The results are found in the next section in the reports generated by calls to the doc program.

C.5 Documentation

The doc program allows users to add additional information as the design process progresses. Here are some comments entered on this particular design run. The program stores all the design information in a single view, and labels the reports as they are added. Reports are sorted according to the instance that they were generated for.

```
********REPORT #1*****************************
Design module: e2
Design report time: Wed Jan 27 10:44:30 1993
Successful completion of design stage: design_done

The design meets specifications.

********REPORT #2*****************************
Design module: e2
Design report time: Wed Jan 27 10:48:05 1993
```

```
Successful completion of design stage: quantize_done

Checked specifications, no violations in the passband, but near
the Edythe ripple is close to the specification.

********REPORT #1******************************
Design module: half
Design report time: Wed Jan 27 10:44:57 1993
Successful completion of design stage: design_done

The design succeeded. There is a small margin in the stopband,
passband ripple is small.

********REPORT #2*****************************
Design module: half
Design report time: Wed Jan 27 10:51:46 1993
Successful completion of design stage: quantize_done

There is a slight violation of spec due to quantization. There is
not a spec point near the violation, and if one were added, candi
would handle it. It is not a very large violation, so I will
ignore it, but look at the overall quantized results.

********REPORT #1******************************
Design module: fourth
Design report time: Wed Jan 27 10:45:32 1993
Successful completion of design stage: design_done

The magnitude data shows that spec is exceeded in one region, but
this is due to Nth band nature of the design, and the fact that
the frequency spec was given in 2 bands without specifying the
don't care band.

********REPORT #2*****************************
Design module: fourth
Design report time: Wed Jan 27 10:54:17 1993
Successful completion of design stage: quantize_done

Quantization did not cause problems.

********REPORT #1******************************
Design module: proc1
Design report time: Wed Jan 27 10:46:32 1993
Successful completion of design stage: design_done

No specifications are given in the magnitude plot, but by eye-
balling the response, we do have about 65 dB of attenuation across
the passband of the composite plot.

********REPORT #2*****************************
Design module: proc1
Design report time: Wed Jan 27 10:54:38 1993
Successful completion of design stage: quantize_done

Composite for the last 3 filters looks good. The minor spec viola-
tion is not a problem, but I can see where it occurs. The overall
response looks good.
```

```
********REPORT #3*****************************
Design module: proc1
Design report time: Wed Jan 27 11:50:11 1993
Successful completion of design stage: mapping_done

Total Area Estimate: 20.263936 sq mm
Cell Area Estimate: 4.989446 sq mm

Area estimates seem large for total area. I would guess something
like 15 or 16 sq mm. There were 218 instructions in the cstore.

********REPORT #1*****************************
Design module: speech
Design report time: Wed Jan 27 10:55:51 1993

Examining the response shows that the quantized composite for all
filters Attenuation appears to be over 65 dB for composite filter.
Note that gain is about 54 dB in the plot, but this is due to gain
of sinc filter. Since we are using quantized coefficients, the
gain is not 1.


********REPORT #2*****************************
Design module: speech
Design report time: Wed Jan 27 13:34:36 1993
Successful completion of design stage: mapping_done
Couldn't find the area estimates.

Full design completed, writeSDL gave the usual message about no
pins on the parent.

********REPORT #1*****************************
Design module: fir1
Design report time: Wed Jan 27 11:51:22 1993
Successful completion of design stage: mapping_done

Total Area Estimate: 3.517398 sq mm
Cell Area Estimate: 2.344932 sq mm

Area estimates seem about right. Total converter area should be on
the order of 25 sq mm in 2um CMOS.
```

C.6 Final Design Steps

The program writeSDL created a skeleton SDL file in pinlist format. The parameter values are currently spread across several files, but the file speech.par will contain references to all of them. The user only needs to connect the terminals in the design, and then instantiate the design in a system. Layout can then be performed using the Lager Silicon Assembly System.

APPENDIX D

Detailed Database Documentation

D.1 Overview

This appendix contains detailed information on coefficient storage, the ESTIMATION bag, and the dblib programming library.

D.2 dblib Programming Library

The dblib programming library provides routines that aid programmers in dealing with the component view. Within the distribution for the design system, the dblib package can be found in the directory $OCDS/lib/dblib, where OCDS is an environment variable pointing to the root directory for the Oversampling Converter Design System installation.

D.2.1 Header File

The file dblib.h should be included in design programs so that proper exit codes, data structures, and function types are defined. The file is included here.

```
/* dblib.h */
#ifndef DBLIB_H #define DBLIB_H
#define DBL_PKG_NAME "dblib"

/* Constants for string size */
#define SMALL_ST 32
#define STR_SZ 256
#define LRG_ST 4096
#define MIN_DB -240.0
```

```
/* Error codes for errtrap */
#define BAD_OCT_VIEW 3
#define BAD_OCT_CALL 4
#define BAD_INPUT 5
#define BAD_SYS_CALL 7

enum coefClassType { COEF_NULL=0, LINEAR_ARRAY, BIQUAD_CASCADE,
     LATTICE_WDF, POLYPHASE_BIQUADS, POLYPHASE_LWDF,
     COMB_CASCADE };
typedef enum coefClassType coefClassType;

typedef struct FSPEC {
     double frequency;
     double magdB;
     double magdevdB;
     double weight;
} Fspec;

/* Function definitions */
extern void getProp();
extern void createOrModifyIprop();
extern void createOrModifyRprop();
extern void createOrModifySprop();
extern void farray2dd();
extern void tags2dd();
extern void codes2dd();
extern double *dd2farray();
extern char **dd2tags();
extern char **dd2codes();
extern coefClassType getCoefType();
extern char *getToken();
extern char *stringCat();
extern char *real2str();
extern char *int2str();
extern char *prop2str();
extern char *iArray2str();
extern char *rArray2str();
extern Fspec *fspec2array();

#ifndef OCT_H #include "oct.h"
#endif

extern octStatus myCreateRprop();
extern octStatus myCreateOrModifyRprop();
extern octStatus myGetByRpropName();

#endif /* DBLIB_H */
```

D.2.2 dblib Function Call Descriptions

The basic function calls are described in detail.

```
dblib.doc - description of top level functions in dblib package.

This is a collection of routines to ease the implementing of
```

software that will interact with the Oversampling Converter Design design System (OCDS).

```
/* General purpose property routines */
void getProp(octObject *facetp, char *propname,
     octObject *containerp, octObject *propp)
```

Given all the information, it will return the property. The facet is given primarily for error message reporting.

```
octObject *createOrModifySprop(octObject *facetp, char *propname,
     octObject *container, char *value)

octObject *createOrModifyIprop(octObject *facetp, char *propname,
     octObject *container, int value)

octObject *createOrModifyRprop(octObject *facetp, char *propname,
     octObject *container, double value)
```

Routines to set or change properties anywhere in the design. Has added error diagnostics that make them better to use than just the plain oh versions.

```
/* Real property routines */
```

Use these routines to handle all real properties. OCT only stores real variables in %lf format, which will not carry enough precision. These routines use %.14lg format to store real variables as strings, maintaining much higher precision. These routines mirror their counterparts given in the oh (OCT helper) package.

```
octStatus myCreateRprop(octObject *contp, octObject *propp,
     char *str, double val)
```

Create a real property contained by contp using propp with name given by str and value val.

```
octStatus myCreateOrModifyRprop(octObject *contp,
     octObject *propp, char *str, double val)
```

Create or modify a real property contained by contp using propp with name given by str and value val.

```
octStatus myGetByRpropName(octObject *contp,
     octObject *propp, char *str)
```

Get a real property contained by contp with name str and return it in propp.

```
/* Coefficient data class access */
void farray2dd(octObject *facetp, double *array, int length,
     coefClassType type, char *bagname)
```

Routine to convert a linear array into the proper COEF bag in the DESIGN_DATA bag. Assumption in the current code is that integer coefs are CODED, while real coefs are IDEAL.

Bagname should be either IDEAL_COEFS or CODED_COEFS.

void tags2dd(octObject *facetp, char **array, int length;
coefClassType type, char *bagname)

Routine to convert a annotate the IDEAL coefficient values with the tags contained in the array. Attaches the TAG property to the VALUE property for a coefficients in the bag given by bagname. Bagname should be either IDEAL_COEFS or CODED_COEFS.

void codes2dd(octObject *facetp, char **array, int *length,
coefClassType type, char *bagname)

Routine to convert a linear array into the proper COEF bag in the DESIGN_DATA bag. CODES are stored in array. Rather than creating a VALUE property, a CODE property is created. Bagname should be either IDEAL_COEFS or CODED_COEFS.

double *dd2farray(octObject *facetp, double *array, int *length, type)

Routine to convert COEF bag info about VALUE properties to a linear array for use by other programs. Space is allocated internally, and can be freed by the calling routine. Bagname should be either IDEAL_COEFS or CODED_COEFS.

char **dd2tags(octObject *facetp, char **array, int *length,
coefClassType type, char *bagname)

Routine to convert COEF bag info about TAG properties attached to VALUE properties to an array of characters for use by other programs. Space is allocated internally, and can be freed by the calling routine. Bagname should be either IDEAL_COEFS or CODED_COEFS.

char **dd2codes(octObject *facetp, char **array, int *length,
coefClassType type, char *bagname)

Routine to convert COEF bag info about CODE properties to an array of characters for use by other programs. Space is allocated internally, and can be freed by the calling routine. Bagname should be either IDEAL_COEFS or CODED_COEFS.

/* Strings */
char *stringCat(char *str1, char *str2, char delimiter)

Returns a string with the parameters concatenated and separated by the delimiter. Arguments are not modified, and the result can be freed by the calling routine.

char *real2str(double number)

char *int2str(int number)

char *prop2str(octObject *prop)

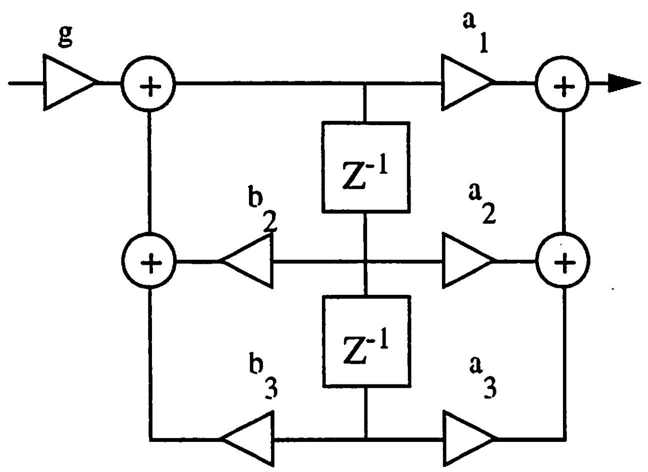
```
        Routines to convert the argument to a string. The result can
        be freed by the calling routine.

char *getToken(char *str, char **token)

        Token is stripped off of argument str and returned as token.
        Function returns pointer to remaining string. Do not free
        this, since it should have been allocated by the calling
        routine. The token can be freed in the calling routine.

char *rArray2str(double *array, int length, char delimiter)

char *iArray2str(double *array, int length, char delimiter)

        Routines to convert arrays to formatted strings. Space is
        allocated within the routine, and must be freed by the
        calling routine. The character delimiter separates entry
        in the single string.

/* Frequency Specification */
Fspec *fspec2array(octObject *facetp, int **sizes)

        This routine will retrieve the frequency specification from
        the component view and return it in an array of the structure
        Fspec. The pointer **sizes is used to return the number of
        specification points in each band. The zeroth entry will
        have the number of bands in the specification.
```

D.3 Coefficient Storage

Separate policies were created for FIR, Biquads, and Lattice Wave Digital Filter (LWDF) structures. Within programs, all coefficients must be stored in a linear array before they are written to the database. They also are formatted in an array when they are retrieved from the database.

For an FIR filter, the coefficients are stored in order in the array, which should have length N for an N tap response. Within the component view, the coefficients are also stored as a linear array contained in the appropriate coefficient bag. This policy is not generic, and will have to be extended to accommodate polyphase FIR filter realizations.

For Biquads and LWDFs, a policy was created that encompasses both single branch and multiple branch polyphase implementations. Biquad coefficients are defined as shown in Figure D.1. The array for storing biquad coefficients is set up as follows. Suppose there are N branches of biquads in the filter. The first entry in the array will then be the value N. The N entries that follow specify the number of biquadratic sections in each branch. The next N values denote the branch delay associated with each branch. Branch delays are necessary when implementing polyphase fil-



$$H(z) = \frac{g(a_1 + a_2 z^{-1} + a_3 z^{-2})}{1 + b_2 z^{-1} + b_3 z^{-2}}$$

Figure D.1 Correspondence between a biquad and the system equation.

ters. The next N entries specify the gain values to associate with each branch. After this, there will be 6 entries per biquad specifying the coefficients. Setting the coefClassType parameter to BIQUAD_CASCADE or POLYPHASE_BIQUADS in function calls to the database will force this format to be recognized. An example for a 2 branch filter follows.

```
ar[0] = 2 /* Number of branches in the filter */
ar[1] = 2 /* Number of sections in branch 1 */
ar[2] = 1 /* Number of sections in branch 2 */
ar[3] = 0 /* Delay associated with branch 1 */
ar[4] = 1 /* Delay associated with branch 1 */
ar[5] = 0.8 /* Gain factor associated with branch 1 */
ar[6] = 1.2 /* Gain factor associated with branch 2 */
/* coefficients for biquad 1 of branch 1 */
ar[7] = a1; ar[8] = a2; ar[9] = a3;
ar[10] = b1, ar[11] = -b1; ar[12] = -b2;
/* coefficients for bilinear section 2 of branch 1 */
ar[13] = a1; ar[14] = a2; ar[15] = 0;
ar[16] = b1, ar[17] = -b1; ar[18] = 0;
/* coefficients for biquad 1 of branch 2 */
ar[19] = a1; ar[20] = a2; ar[21] = a3;
ar[22] = b1, ar[23] = -b1; ar[24] = -b2;
```

For LWDFs, the array is similar except only 4 slots are used for coefficients. LWDFs use biquadratic sections that consist of 2 adaptors stacked on top of each other. Each adaptor has a type from 1 to 5 and a coefficient value. Setting the coefClassType parameter to LATTICE_WDF or POLYPHASE_LWDF causes this format to be recognized. An example follows.

```
ar[0] = 2 /* Number of branches in the filter */
ar[1] = 2 /* Number of sections in branch 1 */
ar[2] = 1 /* Number of sections in branch 2 */
ar[3] = 0 /* Delay associated with branch 1 */
ar[4] = 1 /* Delay associated with branch 1 */
ar[5] = 1 /* Gain factor associated with branch 1 */
ar[6] = 1 /* Gain factor associated with branch 2 */
ar[7] = 0.4123 /* Coefficient for lower adaptor of biquad section
1, branch 1 */
ar[8] = 1 /* lower adaptor type for biquad section 1, branch 1 */
ar[9] = 0.1234 /* Coefficient for upper adaptor of biquad section
1, branch 1 */
ar[8] = 5 /* lower adaptor type for biquad section 1, branch 1 */
ar[10] = 0.14 /* Coefficient for lower adaptor of biquad section
2, branch 1 */
ar[11] = 2 /* lower adaptor type for bilinear section 2, branch 1
*/
ar[12] = 0.0 /* Only a lower adaptor for section 2, branch 1 */
ar[13] = 0.0 /* Only a lower adaptor for section 2, branch 1 */
ar[14] = 0.21 /* Coefficient for lower adaptor of biquad section
1, branch 2 */
ar[15] = 4 /* lower adaptor type for bilinear section 2, branch 1
*/
ar[16] = 0.34 /* Coefficient for upper adaptor of biquad section
1, branch 2 */
ar[17] = 4 /* upper adaptor type for bilinear section 2, branch 1
*/
```

For the coefficient tags arrays, the header information must only contain the number of branches entry and the number of sections in each branch. Tags arrays are used to allow the programmer to assign arbitrary names to the coefficients. Tags for the coefficients should be in the same array position as the coefficient value would be.

Within the component view in the OCT database, the array is translated into the structure shown in Figure D.2. The array values for the 4 LWDF coefficient entries map into the SECTION[i] bag as follows:

```
VALUE[0] = lower adaptor coef. value
TYPE[0] = lower adaptor type
CODE[0] = coded coefficient representation
VALUE[1] = upper adaptor coef. value
TYPE[1] = upper adaptor type
CODE[1] = coded coefficient representation.
```

For a biquad SECTION[i] bag, the coefficients map as follows.

```
VALUE[0] = a1 VALUE[1] = a2 VALUE[2] = a3
VALUE[3] = 1.0 VALUE[4] = -b2 VALUE[3] = -b3
```

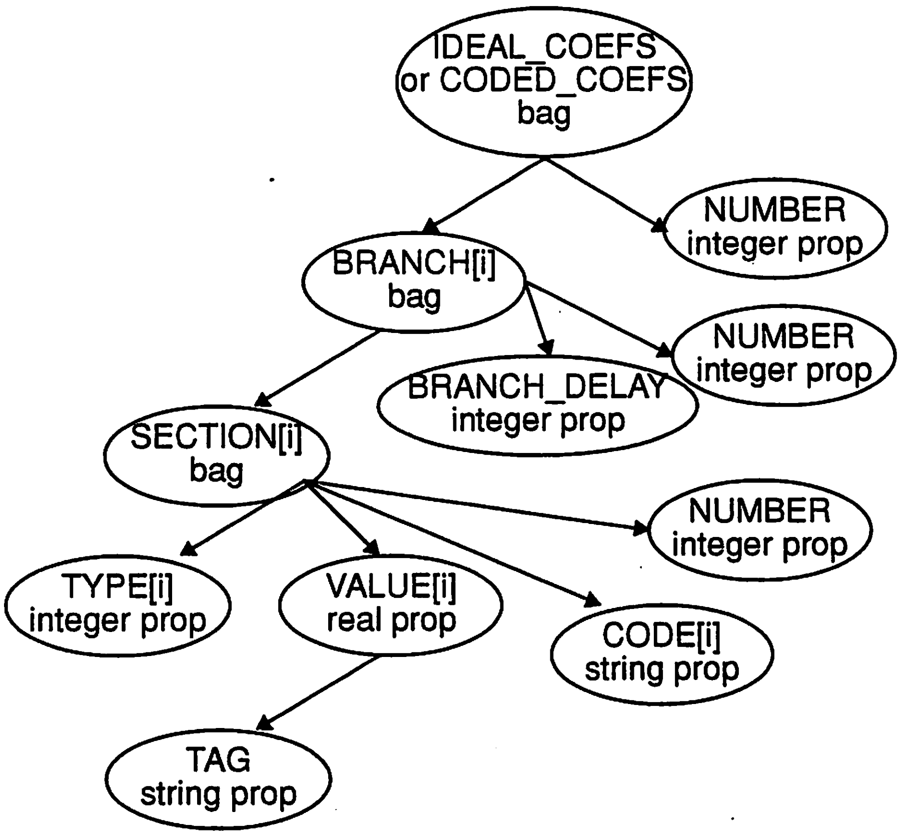


Figure D.2 OCT policy for storing biquad and LWDF coefficients.

The coded coefficients should be stored according to the COEF_CODE_FORMAT stored in the STRUCTURE bag. If the property value is SIGNED_BINARY or UNSIGNED_BINARY, the coded coefficients are stored as a binary number in string representation, for example "101101". If the property has value CSD, then the codes are stored with spaces between the values, like "1 . 0 - 1 1". The binary point is allowed in coded coefficients.

D.4 MAPPING bag

As mentioned in Chapter 2, no strict policy exists for the use of the mapping bag. However, if an ESTIMATES bag is generated within the mapping bag, it should follow the policy shown in Figure D.3. The ESTIMATES bag contains several bags that store the actual estimates which are retrieved by the doc program when generating reports about the design. Only the AREA bag has a set policy, with 3 values needed. The CELL_AREA property stores the total area of all the subcells excluding routing. The TOTAL property gives an estimate for the entire layout with routing. The UNITS property stores the scaling value to give the area in square microns.

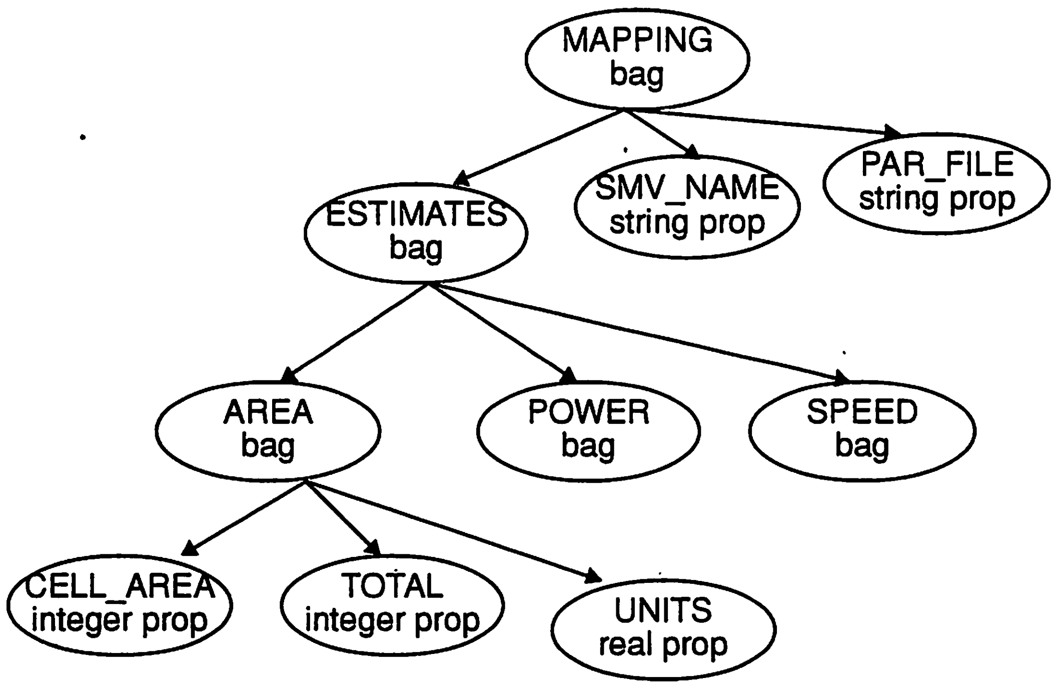


Figure D.3 Policy for the MAPPING bag and the ESTIMATES bag.

APPENDIX E

Architecture Library Summary

This appendix serves as a cross reference for the architecture template library. The templates are organized in alphabetical order. The various options for second order modulators are listed separately in the last section along with some basic design estimation equations.

E.1 CIC Filters

Description:

This architecture performs filtering with magnitude responses that are powers of sin(x)/x. They are built using cascades of integrator and differentiator stages. There is no control over the passband response, but attenuation in the stopband is good. The description for this architecture family starts on page 89

Typical Use:

CIC filters are used as the first stage of decimation after the modulator. They are very useful for implementing moderate decimation ratios, and for cases when there are few constraints on passband ripple.

Pinout and Timing Diagrams:

See Table 6-3 on page 93 and Table 6-4 on page 94 for pinouts and Figure 6.9 on page 93 and Figure 6.9 on page 93 for the timing relations.

E.2 C-to-Silicon

Description:

This template allows for the compilation of RL programs onto the PUMA processor architecture. A stripped version of PUMA called Lambda is also supported. The script c2sil provides full translation from filter coefficients to compiled microcode. The description for this template starts on page 98.

Typical Use:

C-to-silicon is used when there are many processor cycles available for a program implementation. Time-multiplexing of hardware can be achieved allowing several signal processing algorithms to be implemented in software.

Pinout and Timing Diagrams:

Pinouts for the PUMA processor are found in Table 6-6 on page 101. For the Lambda processor, the pinout is found in Table 6-7 on page 104. Timing relations are found in Figure 6.16 on page 102.

E.3 Clock Generators

Description:

Various clock generator cells are supported for the different architecture templates. There are macrocells for 2 phase clock generation and divide-by-6 clock generation. The descriptions for these circuits starts on page 110.

E.4 Decimate-by-2 FIR Filter

This template implements a custom FIR filter with decimation ratio of 2. Samples only need to be calculated at the output rate allowing the internal number of state variables to be cut in half. The description for this filter starts on page 96.

Typical Use:

This is a dedicated filter architecture, and for a filter length of N, N clock cycles will be required for evaluation of a single output. This limits the filter to relatively low speed applications. Since it does implement a decimate-by-2 function, it can be used in wavelet filter bank applications.

Pinout and Timing Diagrams:

The pinout is given in Table 6-5 on page 98. Timing relations are found in Figure 6.12 on page 97 and Figure 6.13 on page 97.

E.5 FIR Filter Family, Time Multiplexed

Description:

This template implements a custom family of FIR filters for cases where the ratio of number of coefficients to the decimation ratio is 1, 2, or 3. The input must be 1 bit wide. The implementation allows for arbitrary coefficient values. The description of this family starts on page 83.

Typical Use:

This template has been used for the first filter after the modulators. The architecture has been optimized for speed. These filters can be used in the same way as CIC filters, but they have added flexibility since arbitrary coefficient values can be stored.

Pinouts and Timing Diagrams:

Pinouts for the template can be found in Table 6-1 on page 86 and a timing diagram in Figure 6.2 on page 85.

E.6 Parallel to Serial Converter

Description:

This is an on-chip parallel to serial converter. The description starts on page 108.

Typical Use:

The converter is used to convert bit-parallel data to serial-burst data so that fewer pins are needed in a full chip implementation. It can also be used to take parallel words off a data bus and sent off chip for observability.

Pinout and Timing Diagrams:

The pinout can be found in Table 6-8 on page 108 and a timing diagram in Figure 6.21 on page 109.

E.7 Second Order Modulators

A set of 5 basic modulators were developed in this project, and all implement the basic second order Δ–Σ modulator. The overall performance varies across the modulators by about 12 dB in dynamic range, with most of the variation attributed to differences in 1/f noise contributions. No naming convention was followed. Each modulator macrocell was designed for a specific technology and was not meant to be scaled. Moving from N-well to P-well can be achieved by changing the appropriate shields and wells in the layout. The double poly cells were designed using a modified technology file that allows both layers of poly to overlap. Mosis technology files are not compatible and define an explicit capacitor layer. All circuits are assumed to be running off a single 5 V supply with analog ground set to 2.5 V.

The basic pinout can be found in Table 6-10 on page 113. Detailed design information is found in Chapter 7.

Estimation formulas make use of the parameter R = $\log_2$(Oversampling Ratio). The formulas were developed using regression based on data from simulation models, and should be valid for R in the range 2 to 9. Opamp and comparator schematics and other circuit diagrams are found in Chapter 7.

E.7.1 Modulator 1: dsm2pc2

Technology: 1.2 μm N-well CMOS
Nominal Sampling Cap. Size: 1.0 pF, metal1-poly
Nominal Integrator Gain: 0.5
Opamp: Class AB 1, bias nominal 20 μA
Comparator: variation 2
Voltage References: vref_pos1 = vref_pos2 = 3.5 V, vref_neg1 = vref_neg2 = 1.5V
Estimated Dynamic Range: $-1.4789R^2 + 26.857R - 31.221$
Estimated Peak SNDR: $-1.5691R^2 + 26.784R - 34.235$
Simulation parameter, 1/f noise input std. dev.: 0.000021
Simulation parameter, additive white noise std. dev.: 0.000091
Simulation parameter, nonlinearty: 325 ppm
Simulation parameter, inner and outer feedback voltages: 2.0
Area: 971 x 705 = 0.685 mm^2
Maximum sampling rate: 8 MHz

E.7.2 Modulator 2: dsm2abp

Technology: 2 μm P-well CMOS

Nominal Sampling Cap. Size: 0.5 pF, double poly

Nominal Integrator Gain: 0.25

Opamp: Class AB 1, bias nominal 20 μA

Comparator: variation 1

Voltage References: vref_pos1 = 3.9V, vref_pos2 = 3.6 V,
vref_neg1 = 1.1 V, vref_neg2 = 1.4 V

Estimated Dynamic Range: $-1.6899R^2 + 27.037R - 26.902$

Estimated Peak SNDR: $-1.6453R^2 + 26.013R - 30.239$

Simulation parameter, 1/f noise input std. dev.: 0.000092

Simulation parameter, additive white noise std. dev.: 0.000129

Simulation parameter, nonlinearty: 325 ppm

Simulation parameter, outer feedback voltage: 2.8

Simulation parameter, outer feedback voltage: 1.8

Area: 705 x 914 = 0.644 mm^2

Maximum sampling rate: 5 MHz

E.7.3 Modulator 3: dsm2abn

Technology: 2 μm N-well CMOS

Nominal Sampling Cap. Size: 0.5 pF, metal1-poly

Nominal Integrator Gain: 0.25

Opamp: Class AB 1, bias nominal 20 μA

Comparator: variation 1

Voltage References: vref_pos1 = 3.9V, vref_pos2 = 3.6 V,
vref_neg1 = 1.1 V, vref_neg2 = 1.4 V

No data for estimating dynamic range or SNDR.

Simulation parameter, additive white noise std. dev.: 0.000129

Simulation parameter, nonlinearty: 325 ppm

Simulation parameter, outer feedback voltage: 2.8

Simulation parameter, outer feedback voltage: 1.8

Area: 1373 x 846 = 1.16 mm^2

Maximum sampling rate: 3 MHz

E.7.4 Modulator 4: cds2pab

Technology: 2 μm P-well CMOS

Nominal Sampling Cap. Size: 0.5 pF, double poly

Nominal Integrator Gain: 0.25, with correlated double sampling.

Opamp: Class AB 1, bias nominal 20 μA

Comparator: variation 1

Voltage References: vref_pos1 = 3.9V, vref_pos2 = 3.6 V, vref_neg1 = 1.1 V, vref_neg2 = 1.4 V

Estimated Dynamic Range: $-1.5747R^2 + 26.951R - 28.324$

Estimated Peak SNDR: $-1.6504R^2 + 26.690R - 32.389$

Simulation parameter, 1/f noise input std. dev.: 0.000092

Simulation parameter, additive white noise std. dev.: 0.000129

Simulation parameter, nonlinearty: 325 ppm

Simulation parameter, outer feedback voltage: 2.8

Simulation parameter, outer feedback voltage: 1.8

Area: 704 x 800 = 0.563 mm^2

Maximum sampling rate: 3 MHz

E.7.5 Modulator 5: dsm2fc2p

Technology: 2 µm P-well CMOS

USE the cell fcbias to provide biasing with this cell!

Nominal Sampling Cap. Size: 0.5 pF, double poly

Nominal Integrator Gain: 0.25

Opamp: Folded Cascode, bias nominal 275 µA

Voltage References: vref_pos1 = 3.9V, vref_pos2 = 3.6 V,

vref_neg1 = 1.1 V, vref_neg2 = 1.4 V

Estimated Dynamic Range: $-1.2447R^2 + 24.844R - 25.440$

Estimated Peak SNDR: $-1.4194R^2 + 24.173R - 27.296$

Simulation parameter, 1/f noise input std. dev.: 0.000020

Simulation parameter, additive white noise std. dev.: 0.000129

Simulation parameter, nonlinearty: 550 ppm

Simulation parameter, outer feedback voltage: 2.8

Simulation parameter, outer feedback voltage: 1.8

Area: 849 x 833 = 0.707 mm^2

Maximum sampling rate: 3 MHz

Note that numbers for maximum sampling rate are based on measured data from circuits. Bandwidth can be increased by using different bias conditions.

APPENDIX F

Mixed Signal VLSI Design

F.1 Overview

When designing mixed signal VLSI chips, care must be taken to minimize the amount of noise that is coupled from the analog to digital circuits. Several precautions were taken in this project in all the chips implemented. This appendix documents some of the steps taken to reduce crosstalk. In all cases, crosstalk was not found to limit performance. Differences between simulation and measured chip results were within experimental error.

F.2 Cell Design

The Lager System was designed as a digital assembly system. Library cells are assembled using place and route tools. Little attention is given to routing or the sensitivity of nets. This must be considered when designing cells for inclusion in the Lager System.

For high resolution A/D conversion, it is best to isolate the analog and digital circuitry to prevent capacitive coupling of digital signals into sensitive nodes. Consider the floorplan of the modulator shown in Figure F.1. The digital circuits, the switches and comparators, are separated from the analog circuits. In addition, shields are placed underneath the switches and the capacitors. Within the opamp layout, all devices are shielded using guard rings. The floorplan was also oriented so that all digital signals and clocks enter from one side of the cell. Analog signals enter on the opposite side. Power and biasing signals were also placed on separate sides. The entire cell is surrounded by a guard ring.

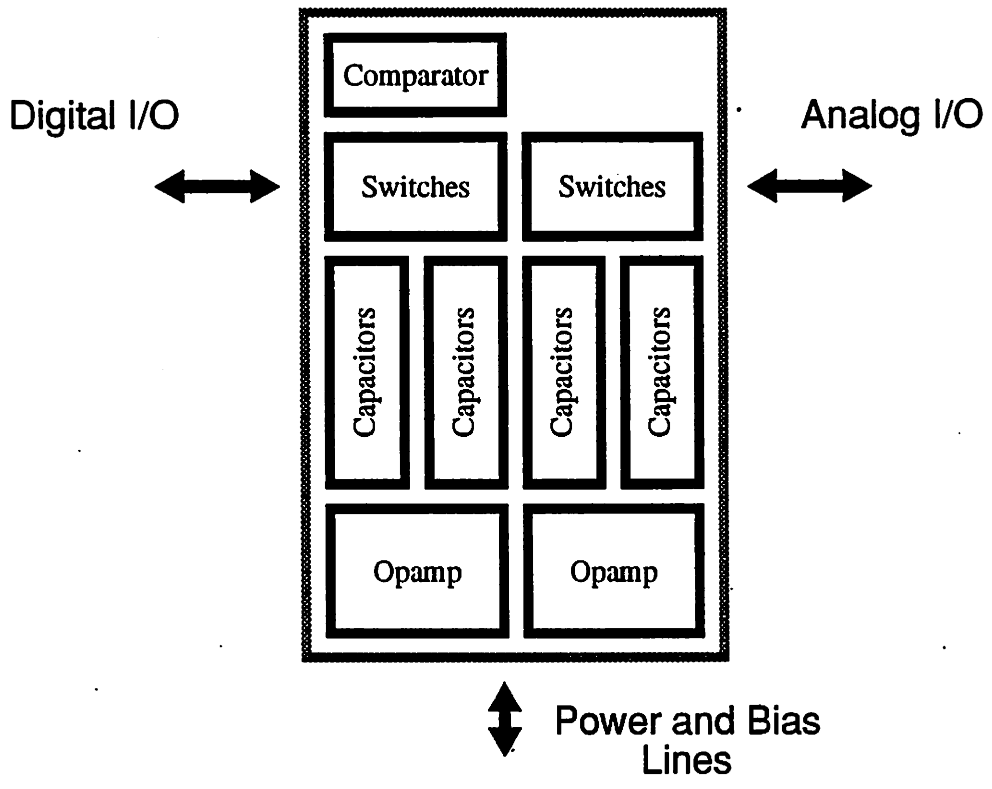


Figure F.1 Floorplan for the modulator.

The macrocell could have been designed as separate entities within the Lager System, with separate cells for opamps, switch arrays, capacitors, and comparators. However, at the present time, only digital place and route tools are available. Since parasitics associated with poor routing can affect the behavior, the entire cell layout was performed by hand. At this level of abstraction, routing connections will have only a minor affect on the overall performance.

F.3 Global Place and Route

Since the cell was created with digital and analog I/O on separate sides, it is possible to dedicate specific routing channels for analog signals. This usually requires designers to use the manual placement and channel definition modes when using FLINT in the Lager System. It is possible to route chips so that no digital lines cross analog signal lines.

The analog modulator should be placed close to the edge of the chip to shorten the runs for the analog signals and allow for some isolation. The modulator was usually placed close to a corner so that 2 sides would be shielded from digital circuits. On the other 2 sides, an effort is made to place relatively quiet circuits or white space to increase the distance from active circuits and clock buffers. With the experimental circuits, crosstalk was not a major issue. Crosstalk was only observed when the clock generators were placed to close to the modulator, which was the case in example 1 of Chapter 8. The crosstalk caused a 2 dB degradation at the 12 bit level of resolution. From

F.4 Pad Ring Design

To further decrease crosstalk, separate power supply pads were used for digital and analog circuits and for shields. To isolate the supplies, the pad ring surrounding the chip core must be broken at some point so that the power lines for the pads are isolated. In this project, no protection was used on the analog pads, so no analog pad ring was necessary. While this decreases the chance of noise coupling from a supply, it also decreases reliability since chips are susceptible to ESD. The pad ring for most chips was developed based on the pinout required by the test board discussed in Appendix A.

F.5 Board Level

All analog supplies and bias lines were carefully filtered close to the pin on the package. A combination of a 10 µF tantalum capacitor and a 100 nF ceramic capacitor was used. Measurements made with and without these filter caps showed quite a bit of improvement. Separate power planes were assigned for analog and digital supplies with the ground point for both supplies established on the board close to the device. The supply lines were tied together at the power supply to minimize the number of loops.

APPENDIX G

Modulator Test Setup

G.1 Testing Oversampling A/D converters.

All of the modulators and A/D converters in this project were tested using a custom test system. Initially, logic analyzers were used, but they did not have sufficient memory and it was difficult to transfer data to a workstation for analysis. The current test system was designed to be simple yet accurate enough to test 16 bit A/D interfaces. The system needed to be flexible enough to allow testing of modulators as well as full interfaces, allowing arbitrary wordlengths and large sample sizes.

The test system consists of 2 boards and IBM compatible PC. A block diagram of the system is shown in Figure G.1. The device under test is placed on the analog test board. The data acquisition board serves as a memory buffer between the PC and the analog test board. A Khron-Hite signal generator provides a low distortion sinusoid for a test signal. Clock signals for the analog modulators are generated from a crystal oscillator. A single HP power supply is used to supply +12 V and -12 V for driving the analog bias circuitry. The +5 V output is used to power the rest of the boards. Ground is established on the analog test board. The signal generator and power supply are left floating.

G.2 Chip Testing Board

The analog chip testing board provides the following features:

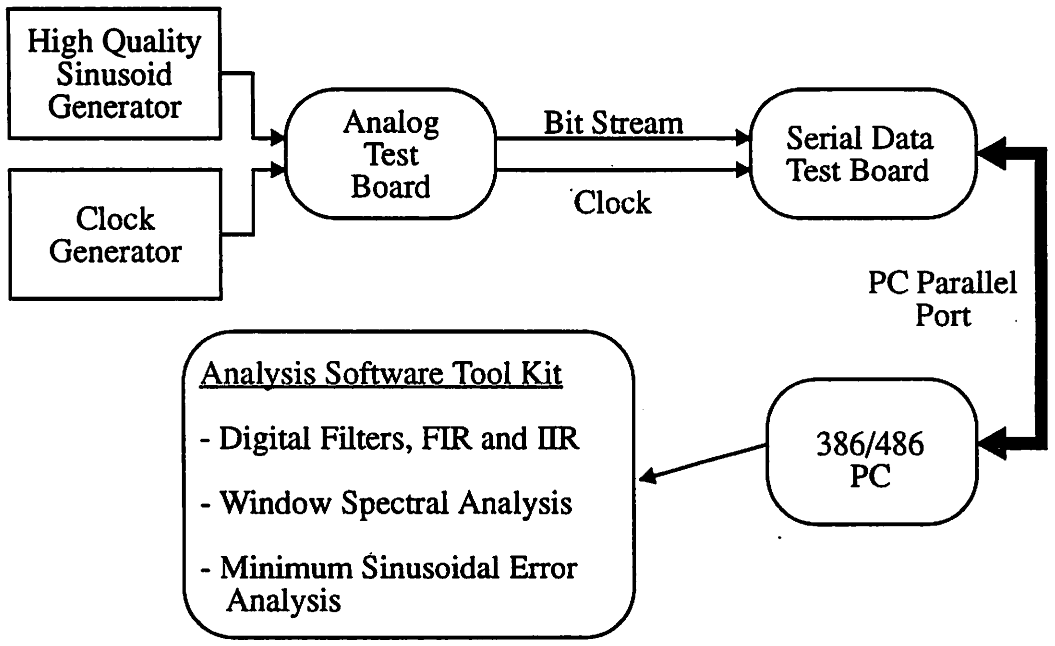

High Quality Sinusoid Generator
Clock Generator
Analog Test Board
Bit Stream
Clock
Serial Data Test Board
PC Parallel Port
386/486 PC
Analysis Software Tool Kit
- Digital Filters, FIR and IIR
- Window Spectral Analysis
- Minimum Sinusoidal Error Analysis

Figure G.1 Block diagram of the test system.

- 2 phase non overlapping clock generator.
- Delayed versions of the 2 phases.
- Digital buffering for driving data to the data acquisition board.
- Generation of 5 analog bias levels.
- Single-ended to differential signal conversion, with level shifting.
- Separate analog and digital power planes.
- Separate analog and digital power supplies, and a separate supply for biasing the well.

A 40 pin dip package was selected for the analog test board. The pinout is shown in Table G-1. The pinout was initially defined for testing 2 modulators packaged in a single chip. Note that most analog and digital signals are placed on separate sides of the package. By planning the internal chip placement correctly, all analog signals can be routed to one side. Internal crossovers of analog and digital signals can be avoided. The separation also allows the use of a broken pad ring, so digital and analog power lines can be isolated in the pad ring. While the analog test board only contains a 40 pin ZIF package, this should not be a limitation. Daughter boards have been designed. With good design, minimal degradation was observed. A block diagram of the board is shown in Figure G.2.

TABLE G-1 Pinout for the 40 pin DIP socket on the test board.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (0 or 5V selectable)	1	40	Capshield (5 or 0V selectable)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output	6	35	Analog Vss
1 bit output	7	34	unused
Input for phi1d	8	33	unused
Input for phi2d	9	32	unused
Input for phi1	10	31	unused
Input for phi2	11	30	unused
Digital GND	12	29	unused
Digital Vdd	13	28	unused
unused	14	27	unused
unused	15	26	Analog Vdd
unused	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
unused	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
unused	18	23	2Analog In-
unused	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

G.2.1 Digital Clock Generation Circuitry.

The circuit used for clock generation is straightforward. The schematic is shown in Figure G.3. A crystal is preferred for clock generation, since it creates low phase noise. Standard pulse generators were found to have excessive amounts of phase noise, which caused an increase in the noise floor observed in the modulators. A high precision sinusoid generator can also be used for a clock generator. Full TTL levels were used to clock the circuit. Lower clock levels may lower noise levels.

G.2.2 Single-Ended to Differential Converter

All of the modulators were designed for differential input signals. To achieve single-ended to differential conversion, several methods can be applied. The easiest is to use a signal generator with differential outputs. Another option is to use a transformer, but this is generally used for

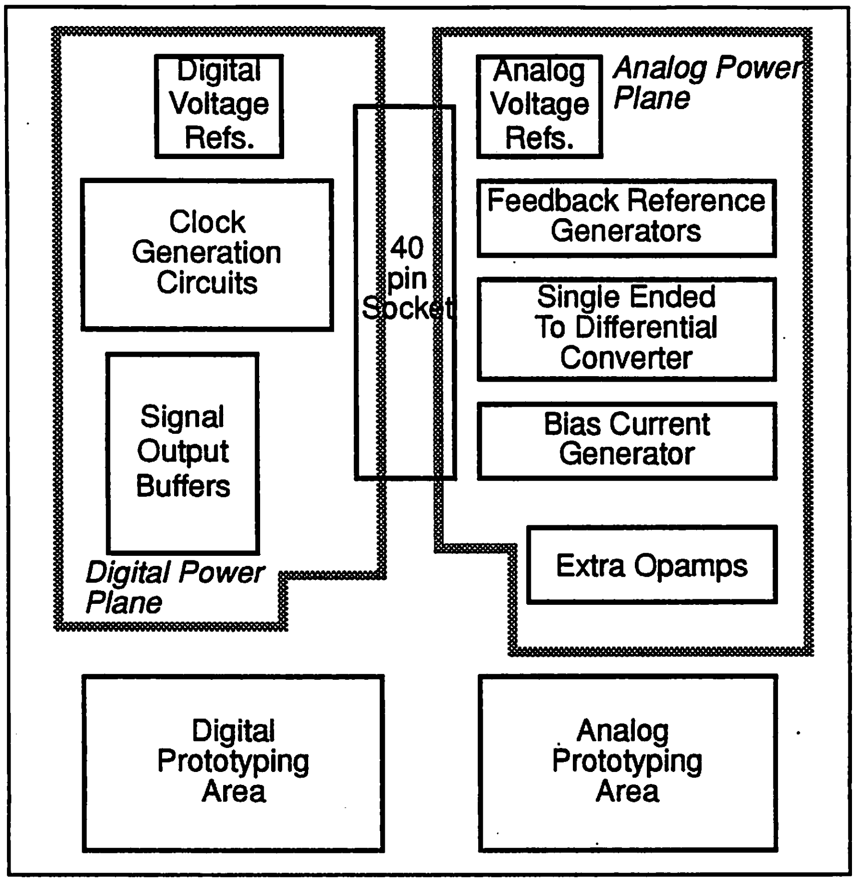


Figure G.2 Test board layout.

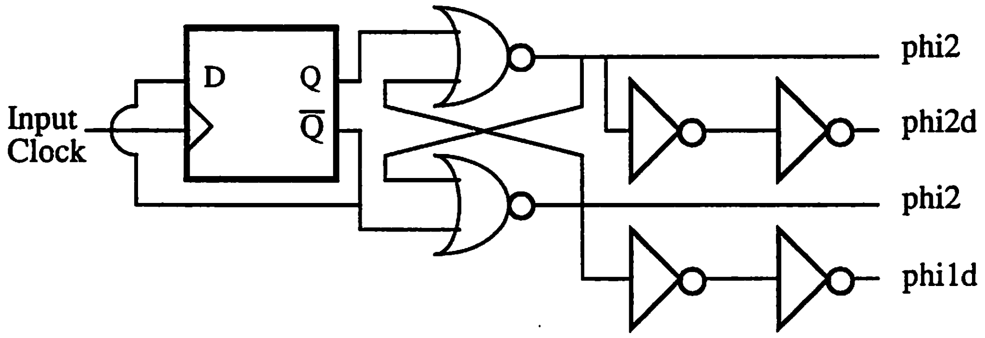


Figure G.3 Clock generation circuitry.

higher frequency signals. The final approach is to use a set of opamps as shown in Figure G.4. Several opamps were used during development of the test board. It was found that the opamps do add

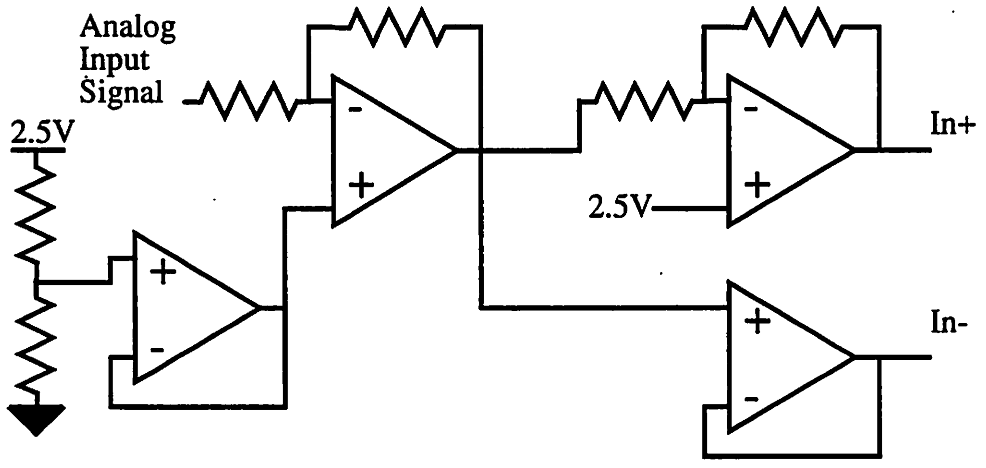


Figure G.4 Single-ended to differential converter.

some harmonic distortion, but this tends to be small compared to the distortion added by the circuit.

G.2.3 Analog Reference Generators

Simple resistive dividers with good bypass capacitor networks were found to be adequate for this application. Buffer amplifiers can be added, but they also add noise. Bias current generation was performed using a simple IC transistor array.

G.3 Serial Data Acquisition Board

A serial acquisition card was designed to allow collection of data from modulators and complete A/D converters. Once data is collected, it can be analyzed to determine the performance of a converter. The board was designed to interface to an IBM compatible personal computer. This provides an inexpensive, easy to use system. The board was used with a 33MHz, 486 based system with 5Mb of memory. It also worked with a 20Mhz, 386SX laptop with 6Mb of memory. Minimum requirements are a 386SX system with at least 2Mb of memory. The software might be rewritten to allow less memory usage with longer program execution times.

G.3.1 Overview of the Serial Data Acquisition Board

The block diagram for the board is shown in Figure G.5. It consists of a bank of memory orga-

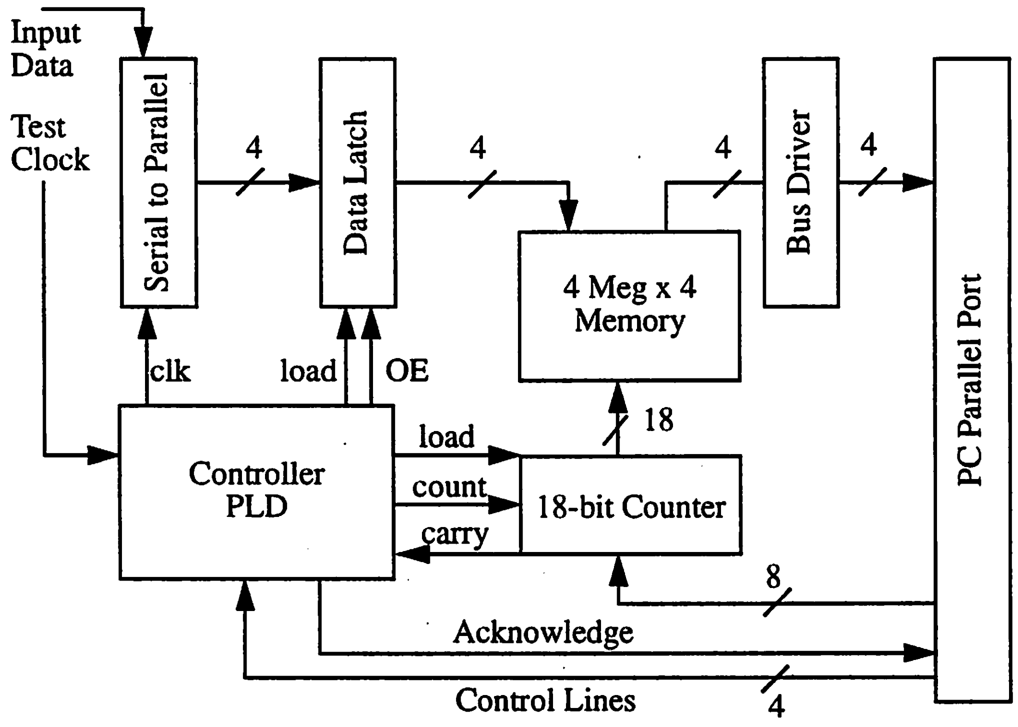


Figure G.5 Block diagram of the Serial Data Acquisition Board.

nized into 4 bit words, an address generator, a controller, and a serial to parallel converter. The board has 4 modes of operation, listed as follows:

8. Idle mode - do nothing.
9. Load address - load an address offset into the address generator.
10. Acquire mode - Acquire data and save it into the memory.
11. Read mode - Allow PC to read data sequentially from the memory.

The board has been implemented with 4 Megabits of memory organized as 4 bit words. The address generator was implemented using a 20 bit up counter.

G.3.2 Communication Protocols

Communication between the PC and the Serial Data Acquisition board is controlled using the lines normally associated with the printer interface. The signals carried on the 25 pin cable were redefined to conform to the current application. Rather than using the data lines normally associated with the parallel port, data are collected through the status lines since they can be read without the need for latching data. This should speed up the data transfer. Data for the initializing the address in the 18-bit counter are put on the normal data lines. 3 consecutive 8-bit words are sent to the counter to provide the 18-bit initial address. The controller is designed to respond to the rising edge of a strobe signal, making the system asynchronous.

Custom software provides the interface to control data acquisition using the Serial Data Acquisition board. The software was developed using a port of the GNU gcc compiler. Saving long streams to disk is time consuming so the program provides 2 filtering tasks after acquisition, FIR decimation filtering and serial to parallel conversion. Computation using in memory operands speeds up the acquisition and analysis tasks. Additionally, the program provides for an arbitrary batch file to be executed after the filtering tasks have been completed.

G.3.3 Parallel to Serial Converters

This system will only acquire serial data. In order to collect data from bit parallel systems, a parallel to serial converter is required. The parallel to serial converter can be implemented on chip using a library module, or on the test board using the simple circuit programmed in a PLD. The PLD is used to implement a state machine similar to the one found in the library module.

APPENDIX H

Chip Descriptions

H.1 Overview

This appendix provides detailed information about chips generated in this project. The CIF files for each of these chips is included on the design system distribution tape. Each sheet provides a pinout, details on the chip functionality, and information about chip yields. The descriptions are arranged in alphabetical order according to the name assigned to the chip. MOSIS run numbers are also included to ease identification of chips.

Chip #1: ad12
Mosis ID Number: N28O AJ1

1.1 Introduction

This chip contains a second order modulator, an FIR filter, and a version of the Lambda processor created by the C-to-silicon tools. It implements an oversample and decimate by 256 A/D converter. Design notes indicate that this chip was created during 7/92. The chip was fabricated using HP's CMOS34 1.2 μm, n-well process. It also contains an on-chip clock generator.

1.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 1.1. The pinout conforms to the standard for the analog test board.

TABLE 1.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 0V)	1	40	Capshield
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Guard	5	36	vcmi (set to about 1.5V)
1 bit modulator output	6	35	Analog Vss
PSEROUT	7	34	vref_neg2 (tie to pin 25)
Input for slow phi1d	8	33	vref_pos2 (tie to pin 24)
Input for slow phi2d	9	32	AVdd (tie to pin 26)
Input for slow phi1	10	31	ODSMIN
Input for slow phi2	11	30	FSERIALOUT
Digital GND	12	29	FOCLOCK
Digital Vdd	13	28	POCLOCK
CLOCKIN	14	27	PDataValid
ofPHI1	15	26	Analog Vdd
ofPHI2	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
CLOCKSEL (H for on-chip)	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
RESET	18	23	Unused
ODSMSEL (H for on-chip)	19	22	Unused
BUFSEL (H for normal operation)	20	21	Unused

1.3 Chip Contents

The chip contains an on-chip clock generator, a second order modulator (the same one used on dsm12), a decimate by 32 FIR filter, and a DSP core. To operate the chip in normal mode with the on chip clock generators, place a clock that is 6 x 256 times as fast as the desired output rate on the pin CLOCKIN. Then set CLOCKSEL and ODSMSEL high and BUFSEL low. The modulator output will be available on pin DSMOUT and the A/D output will be available as a 22 bit output in a serial burst. The serial data is found on pin PSERIALOUT, the serial burst clock on POCLOCK, and a synchronization signal on DataValid. In addition, the output of the FIR filter is available as 16 bit serial words in serial bursts with data on FSERIALOUT and the serial clock on FOCLOCK.

To use external clocks, set CLOCKSEL low and set PHI1, PHI2, PHI1d and PHI2d at 256 times the desired output rate. Set ofPHI1 and ofPHI2 3 times faster than PHI1 and PHI2.

The input data to the filter can be supplied from off-chip. Set pin ODSMSEL low and place synchronized input data on ODSMIN.

1.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was cdsdsm.cif. Of the 12 chips, all were found to be fully functional. The chips were tested by applying an input sinusoid and measuring SNR response. However, there is a circuit problem that makes the chips fail when operating slightly above 5 V. This appears to be highly temperature dependent. This only affects the Lambda processor. The modulator and FIR filter work fine, and were tested up to 5.5 V.

1.5 Comments

Under speed tests, the chips were found to be functional up to about 8 MHz. It may go faster, but a 50 MHz crystal was the highest speed available for testing. The modulator definitely dies out above 8 MHz. The references were set so that vref_pos1 = 3.5V, vref_neg1 = 1.5V, vref_pos2 = 3.5 V, and vref_neg2 = 1.5 V. Bias currents were set to a nominal 40 μA. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 1.0 pF. Dynamic range using the on-chip digital filter was extrapolated to be 87 dB.

The pad frame for this chip is set up for a 108 pin package. If extra die are rebonded, it is possible to enable some other test features, such as the ability to see what is on the main processor bus. Please refer to the SDL design files for more detail.

The chip will only work when the on-chip clock generator was used. These is a flaw in the design so that PHI1 and PHI1d are shorted together. The filter should still work using off-chip clocks, but there appears to be a synchronization problem in the interface for the processor. I did not debug this.

1.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 2 comparator was used as well as the type 2 Class AB opamp. Details on the design trade-offs for the digital filter can also be found there. Other details on device sizes can be obtained by examining the CIF file.

Chip #2: bcap
Mosis ID Number: N15W FA1

2.1 Introduction

This chip contains 2 second order modulators, both using class AB opamps and double poly capacitors with integrator gains set to 0.25. The first modulator uses 1 pF sampling caps, while the second modulator uses 4 pF sampling caps. Design notes indicate that this chip was created during 5/91. The chip was fabricated using Orbit's 2.0 µm, p-well, double poly process.

2.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 2.1. The pinout conforms

TABLE 2.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	boutp1
Input for phi1d	8	33	boutn1
Input for phi2d	9	32	boutn2
Input for phi1	10	31	boutp2
Input for phi2	11	30	soutp2
Digital GND	12	29	soutn2
Digital Vdd	13	28	soutp1
outenab	14	27	soutn1
tvinp	15	26	Analog Vdd
tvinm	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
tvo1	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
tvo2	18	23	2Analog In-
tbias	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

to the standard for the analog test board. The chip will function properly if placed in the test board and data from pins 6 and 7 will be valid.

2.3 Chip Contents

Two copies of the modulator were placed on the chip. On the left side of the chip is the modulator with 1 pF sampling caps. Outputs from the first and second integrators were routed to pads 27, 28, 29, and 30. On the right side of the chip is the modulator with 4 pF sampling caps. Outputs from the second modulator were routed to pads 31, 32, 33, and 34. This chip made use of a set of switches on the integrator outputs so the routing to the pads could be switched out when parasitic effects weren't desired. This feature is controlled by pin 14, outenab. When set high, the outputs of the integrators were enables.

In addition to the modulators, a test opamp was placed on the chip. The schematic for the opamp will be given in section 6.

2.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was called bcap.cif. Of the 12 chips, all were found to be fully functional. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 12 chips were found to be functional with a spread of about 7 dB in performance across the chips.

2.5 Comments

During testing, the references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.4 V, and vref_neg2 = 1.6 V. Bias currents were set to a nominal 40 μA. On this chip, I tried using unit capacitors to create the big caps. I also found the noise floor only slightly lower than dsm2p. The larger capacitances did slow down the devices.

2.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 2 comparator was used as well as the type 1 Class AB opamp. The schematic for the extra test opamp is shown in Figure 2.1. Other

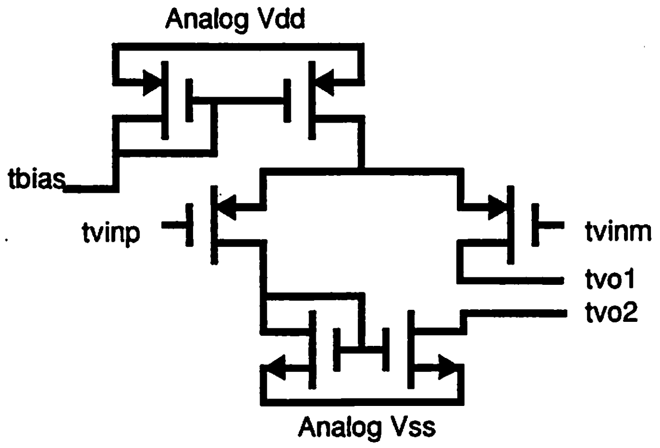


FIGURE 2.1Schematic for the test opamp placed on the chip.

details on device sizes can be obtained by examining the CIF file.

Chip #3: cdsdsm
Mosis ID Number: N0BO CC1

3.1 Introduction

This chip contains 2 second order modulators, both using class AB opamps and double poly capacitors with integrator gains set to 0.25. The first integrator in both modulators made use of correlated double sampling. Design notes indicate that this chip was created during 10/90. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. It also contains an on-chip clock generator.

3.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 3.1. The pinout conforms

TABLE 3.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	Unused
Input for phi1d	8	33	Unused
Input for phi2d	9	32	Unused
Input for phi1	10	31	Unused
Input for phi2	11	30	Toutp1
Digital GND	12	29	Toutn1
Digital Vdd	13	28	Toutp2
phi1	14	27	Toutn2
phi2	15	26	Analog Vdd
phi1d	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
phi2d	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
select (H for on-chip)	18	23	2Analog In-
compbar	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

to the standard for the analog test board. The chip will function properly if placed in the test board and data from pins 6 and 7 will be valid.

3.3 Chip Contents

Two copies of the modulator were mirrored on the chip. On the left side of the chip, outputs from the first and second integrators as well as the comparator output were routed to pads 27, 28, 29, 30, and 19. To use the on-chip clock generator, set pin 18 high. The 4 clock phases are then derived from the input on pin 10 and buffered an placed on pins 14, 15, 16, and 17. The output of the left modulator is routed to pin 7. The right modulator has no parasitic loading and the output is routed to pin 6.

3.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was cdsdsm.cif. Of the 12 chips, all were found to be fully functional. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 12 chips were found to be functional with a spread of about 4 dB in performance across the chips.

3.5 Comments

Under speed tests, the chips were found to be functional up to about 3 MHz. However, this can be changed by adjusting bias to lower opamp gain. Absolute speed numbers are difficult to gauge since performance rolls off slowly. The references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.6 V, and vref_neg2 = 1.4 V. Bias currents were set to a nominal 40 μA. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF. The peak SNR was measured at 78 dB and the dynamic range was extrapolated to be 91 dB.

3.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the type1 Class AB opamp. The circuit diagram for the correlated double sampling integrator is also given. The on-chip clock generator is as documented. Other details on device sizes can be obtained by examining the CIF file.

Chip #4: df3s2
Mosis ID Number: N1CF JA1

4.1 Introduction

This chip contains 2 second order modulators, both using class AB opamps and double poly capacitors with integrator gains set to 0.25. The first integrator in both modulators made use of correlated double sampling. Design notes indicate that this chip was created during 10/90. The chip was fabricated using Orbit's 2.0 µm, p-well, double poly process. It also contains an on-chip clock generator.

4.2 Pinout

The chip was packaged in a 132 pin PGA and the pinout is given in Table 4.2, Table 4.2, Table 4.2,

TABLE 4.1Pinout for the chip, north side of the chip.

Pin No.	Pin Description	Type	Pin No.	Pin Description	Type
1	pbias	Analog in	18	iLOAD1	Test out
2	nbias	Analog in	19	IN2P1	Test out
3	AVdd	Analog pwr	20	IN2P0	Test out
4	AVss	Analog pwr	21	Vdd	Dig. pwr
5	iop2	Analog out	22	Unused	
6	ion2	Analog out	23	Unused	
7	ctrl2	Ana. ctrl.	24	Gnd	Dig. pwr
8	ctrl2*	Ana. ctrl.	25	Unused	
9	TESTC1	Test ctrl	26	Unused	
10	s1OUT1	Test out	27	CLR2	Test ctrl
11	s1OUT0	Test out	28	PRE2	Test ctrl
12	iLOAD2	Test out	29	PRE3	Test ctrl
13	GND	Dig. pwr	30	Unused	
14	TESTC3	Test ctrl	31	Vdd	Dig. pwr
15	TESTC2	Test ctrl	32	Unused	
16	TEST2	Test ctrl	33	Unused	
17	LOAD1p	Test in			

and Table 4.2.

TABLE 4.2Pinout for the chip, west side of the chip.

Pin No.	Pin Description	Type	Pin No.	Pin Description	Type
34	SCANIN	Test in	51	Unused	
35	LOAD2p	Test in	52	Unused	
36	S2OUT0	Test out	53	Vdd	Dig. pwr
37	S2OUT1	Test out	54	Unused	
38	S2OUT2	Test out	55	iNext	Test out
39	Vdd	Dig. pwr	56	RESET	Ctrl in
40	IN3p[0]	Test in	57	ilatch2	Test out
41	IN3p[1]	Test in	58	ilatch1	Test out
42	IN3p[2]	Test in	59	SCAN	Test ctrl
43	TEST3	Test ctrl	60	PRE4	Test ctrl
44	RESETb	Ctrl in	61	DL1	Test ctrl
45	GND	Dig. pwr	62	GND	Dig. pwr
46	CLK64p	Test in	63	Unused	
47	CLK16p	Test in	64	Unused	
48	iCLK	Test out	65	Unused	
49	Unused		66	Unused	
50	Substrate	Dig. pwr			

TABLE 4.3Pinout for the chip, south side of the chip.

Pin No.	Pin Description	Type	Pin No.	Pin Description	Type
67	GND	Dig. pwr	84	OUT2	Dig. out
68	Vdd	Dig.pwr	85	OUT1	Dig. out
69	OUT17	Test out	86	OUT0	Dig. out
70	OUT16	Test out	87	Vdd	Dig. pwr
71	OUT15	Test out	88	GND	Dig. pwr
72	OUT14	Test out	89	fCLK	Clock
73	OUT13	Test out	90	SCANOUT	Test out
74	OUT12	Test out	91	TEST0	Test ctrl
75	OUT11	Test out	92	XPHI2d	Test clock
76	OUT10	Test out	93	XPHI1	Test clock

TABLE 4.3Pinout for the chip, south side of the chip.

Pin No.	Pin Description	Type	Pin No.	Pin Description	Type
77	OUT9	Test out	94	Vdd	Dig. pwr
78	OUT8	Test out	95	XPHI1d	Test clock
79	OUT7	Test out	96	XPHI2	Test clock
80	OUT6	Test out	97	GND	Dig. pwr
81	OUT5	Test out	98	CLR1	Test in
82	OUT4	Test out	99	Unused	
83	OUT3	Test out			

TABLE 4.4Pinout for the chip, east side of the chip.

Pin No.	Pin Description	Type	Pin No.	Pin Description	Type
100	DataValid	Ctrl out	117	DGnd	Dig. pwr
101	TEST1	Test ctrl	118	DVdd	Dig. pwr
102	GND	Dig. pwr	119	ctrl1*	Ana. ctrl
103	Unused		120	ctrl1	Ana. ctrl
104	Unused		121	ion1	Ana. out
105	Unused		122	iop1	Ana. out
106	Unused		123	guard	Ana. ref.
107	Unused		124	shiled	Ana. ref.
108	Unused		125	in1p	Analog in
109	Unused		126	AGnd	Ana. ref.
110	Unused		127	inn	Analog in
111	Vdd	Dig. pwr	128	Cshield	Ana. ref.
112	Unused		129	vref_pos1	Ana. ref.
113	Unused		130	vref_neg1	Ana. ref.
114	Unused		131	vref_pos2	Ana. ref.
115	Unused		132	vref_neg2	Ana. ref.
116	dout	Ana. out			

The pinout does not conform to the standard for the analog test board. A separate test board was used for testing this device.

4.3 Chip Contents

This chip contains a second order modulator and 3 digital FIR filters implementing an oversample and decimate by 64 A/D converter. The chip contains many testability features, allowing all 4 of the functional units to be tested individually. To get the overall chip to run, start by placing the master clock on the pin fclk. It should be twice as fast as the desired sampling rate. Set these signals low: ctrl2, TESTC1, TESTC3, TESTC2, TEST2, CLR2, PRE2, PRE3, TEST3, RESET, SCAN, PRE4, DL1, TEST0, CLR1, TEST1, and ctrl1. Set these signals high: ctrl1*, RESETb, and ctrl2*. You may have to toggle RESET and RESETb to get the processors started. All analog references should be connected as in other examples. The differential analog input then should then be placed on the pins inp and inn.

To work with the modulator alone, use the following bits of information:

1. The signals ctrl1 and ctrl1* control switches connected to the output of the first integrator. The output can be observed on the pins ion1 and iop1. Setting ctrl1 high and ctrl1* low enables the switches so the output can be observed. For the second integrator, the signals ctrl2, ctrl2*, ion2, and iop1 serve similar functions.
2. If TEST0 is set high, then the modulator clock inputs must come from XPHI1, XPHI2, XPHI1d, and XPHI2d. If TEST0 is low, then the modulator clocks are derived from fclk.
3. The output of the modulator is available on the pin labelled dout.

All the digital filters were designed with 2 sets of inputs. The signals with TEST at the beginning control these functions.

To isolate the first filter after the modulator, a 4th order CIC filter, use this information:

1. If you set TESTC1 high, the signal CLK64p must be supplied to clock this circuit. CLK64p has a frequency that is 64 times the output rate.
2. If you set TEST1 high, then the 1 bit filter input is taken from the pin IN1p.
3. The digit serial output is available on pins S1OUT0 and S1OUT1. The signal iLOAD1 is generated internally and is synchronized with the most significant digit.
4. The signal RESETb resets this filter.

To isolate the second filter, a 7th order CIC filter, use this information:

1. If you set TESTC2 high, then you must supply CLK64p and CLK16p. CLK64p has frequency 64 times the output rate, and CLK16p has frequency 16 times the output rate. If TESTC2 is low, then clocks are derived from fclk.
2. If you set TEST2 high, then you can supply the digit serial input data on the pins IN2p0 and IN2p1. You must also supply LOAD1p, which is a signal that is synchronized with the most significant digit of the input. If TEST2 is low, data is taken from the preceding CIC filter.
3. The digit serial outputs are available on pins S2OUT0, S2OUT1, and S2OUT2. The signal iLOAD2 is generated by this filter and is synchronized with the most significant digit of the input data.
4. The signal RESTb resets this filter.

To isolate the third filter, a 128 tap FIR filter, use this information:

1. If you set TESTC3 high, then you must supply the clock CLK16p as described previously, and fclk must still be connect. The signal iCLK will show you the internally generated clock.

2. If you set TEST3 high, you can supply the digit serial data input data on the pins IN3p[0-2]. You must also supply LOAD2p, which is synchronized with the most significant digit.
3. The output data is bit parallel and found on pins OUT[0-17]. In addition, the signals iNEXT, iCLK, iLATCH1, and iLATCH2 are generated in this filter and can be used to detect chip problems.
4. The signal RESET is used to reset this filter.
5. This filter has a scanpath. By setting SCAN high and putting vectors on SCANIN, you can load the scan chain. Output scan data is found on SCANOUT.

Several signals called PREX, CLRX, and DL1 also exist. The CLR signals can be used to reset the integrators in the CIC filters. The other signals are associated with clock generation circuitry.

4.4 Yield Statistics

This chip was hard to test, and some devices exhibited problems when run at 5V. Only the digital portions of the chips were tested. No numbers on yield were tabulated.

4.5 Comments

The chip contains an error in the RESET circuitry which causes 6 out of every 64 output samples to be corrupt. This error only occurs in the final 128 tap FIR filter. The intermediate filtering sections do work. This chip was packaged in a 132 pin PGA. A wire-wrapped test board was used. From the limited amount of data presented here, it is clear that this is a very complex chip.

4.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the type1 Class AB opamp. Details on the chip and digital circuits can be found in Shoei-Shin Hang's Masters Degree Report, available from UCLA. Other details on device sizes can be obtained by examining the CIF file.

Chip #5: dsm12
Mosis ID Number: N1BA AD1

5.1 Introduction

This chip contains a second order modulator using a class AB opamps and metal1-poly capacitors with integrator gains set to 0.5. Design notes indicate that this chip was created during 11/91. The chip was fabricated using HP's CMOS34 1.2 µm, n-well process. It also contains 3 variations of a clock driver to allow variation of the sampling edge.

5.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 5.1. The pinout conforms

TABLE 5.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 0V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	Unused
1 bit output	6	35	Analog Vss
Unused	7	34	ngate
Input for phi1d	8	33	ndrain
Input for phi2d	9	32	pgate
Input for phi1	10	31	pdrain
Input for phi2	11	30	intout1
Digital GND	12	29	intout2
Digital Vdd	13	28	Unused
outenab	14	27	vcmi
enable1	15	26	Analog Vdd
enable2	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
enable3	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
intout3	18	23	Unused
intout4	19	22	Unused
Unused	20	21	Unused

to the standard for the analog test board.

5.3 Chip Contents

The chip contains a second order modulator designed for 1.2 μm CMOS. The outputs of the integrators were routed to pins 18, 19, 29, and 30. Switches were placed between the opamp outputs and the pads to isolate the opamps from extra parasitic loading. The signal outenab is used to close and open the switch.

3 clock buffers with different sizes were placed on the chip. They have tristate outputs, and a single set of buffers must be enabled for the chip to work. The signals enable1, enable2, and enable3 are the buffer enables, with enable1 activating the slowest clock buffer and enable3 the fastest.

The signal vcmi should be set about Vt below analog ground. Other values can be tried.

Also on this chip, an n and a p transistor were supplied so that the process could be characterized if necessary. Figure 5.1 shows the pin connections for the devices..

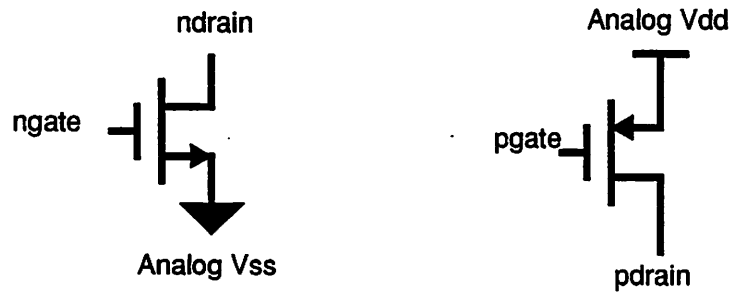


FIGURE 5.1Extra devices put on the chip for device characterization.

5.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 50 chips were provided. The cif file used to fabricate the chip was called dsm12.cif. Of the 50 chips, 40 were found to be fully functional. Devices 10, 11, 12, 13, 27, 38, and 39 did not work at all. Devices 8, 14, and 41 showed some life, but poor performance. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 40 chips were found to be functional with a spread of less than 2 dB in performance across the chips.

5.5 Comments

Under speed tests, the chips were found to be functional up to about 8 MHz. However, this can be changed by adjusting bias to lower opamp gain. Absolute speed numbers are difficult to gauge since performance rolls off slowly. The references were set so that vref_pos1 = 3.5V, vref_neg1 = 1.5 V, vref_pos2 = 3.5 V, and vref_neg2 = 1.5 V. Bias currents were set to a nominal 40 μA. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 1.0 pF. The peak SNDR was measured at 83 dB and the dynamic range was extrapolated to be 92 dB.

5.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 2 comparator was used as well as the type 2 Class AB opamp. Other details on device sizes can be obtained by examining the CIF file.

Chip #6: dsm2p
Mosis ID Number: N04Y EG1

6.1 Introduction

This chip contains 2 second order modulators, both using Class AB opamps and double poly capacitors with integrator gains set to 0.25. Design notes state that the chip was created during 3/90. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. 2 cascoded n transistors and 2 capacitors were included as test structures.

6.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 6.1. The pinout conforms

TABLE 6.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	ncdrain
Input for phi1d	8	33	ndrain
Input for phi2d	9	32	ncgate
Input for phi1	10	31	ngate
Input for phi2	11	30	bottom1 (bottom plate of capacitor)
Digital GND	12	29	top1 (top plate of capacitor)
Digital Vdd	13	28	top2 (top terminal of capacitor)
tphi1d (test phi1d phase)	14	27	bottom2 (bottom plate of capacitor)
Toutn2	15	26	Analog Vdd
Toutp2	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
comp	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
Toutp1	18	23	2Analog In-
Toutn1	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

to the standard for the analog test board. The chip will function properly if placed in the test board, but only data from pin 6 will be valid.

6.3 Chip Contents

Two copies of the modulator were mirrored on the chip. On the left side of the chip, outputs from the first and second integrators as well as the comparator output were routed to pads 18, 19, 15, 16, and 17. In addition, the first modulator has the phi1d clock phase separated and routed to pin 14 to allow an additional testing mode. If this pin is set low, the first integrator is isolated and can be run using the vref1 pins as input. In normal operation, pin 14 was tied to pin 8. The output of the left modulator is routed to pin 7. The right modulator has no parasitic loading and the output is routed to pin 6.

Several test devices were placed on the chip and their schematics are shown in Figure 6.1.

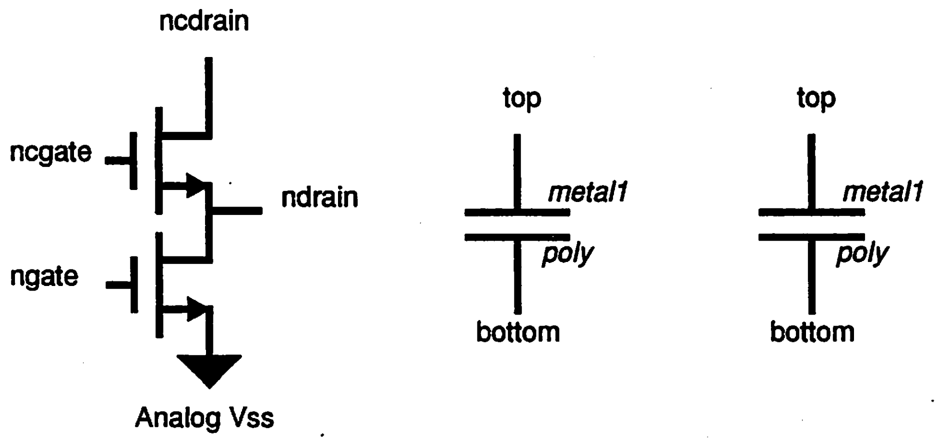


FIGURE 6.1Extra devices put on the chip for device characterization.

6.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was dsm2p.cif. Of the 12 chips, 10 were found to be fully functional. Chip 6 was not functional, and chip 8 functioned at a lower performance level. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 10 chips were found to be functional with a spread of about 2 dB in performance across the chips.

6.5 Comments

Under speed tests, the chips were found to be functional up to about 8 MHz. Absolute speed numbers are difficult to gauge since performance rolls off slowly. The references were set so that vref_-pos1 = 3.8V, vref_neg1 = 1.2V, vref_pos2 = 3.0 V, and vref_neg2 = 2.0 V. Bias currents were set to a nominal 40 µA. For a sampling rate of 1 MHz and a decimate by 256 sinc^3 decimation filter, typical peak SNR was 72 dB and dynamic range was 82 dB. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF.

The cif file was written with an old version of the magic technology file, which causes an error when n transistors are read in. Use the cif istyle lambda=1.0(oldpwell) if this happens.

6.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the type 1 Class AB opamp. Other details on device sizes can be obtained by examining the CIF file.

Chip #7: dsmad2
Mosis ID Number: N12E HB1

7.1 Introduction

This chip contains a second order modulator with integrator gains set to 0.25 and an FIR decimate-by-64 filter. Design notes state that the chip was created during 1/91. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. An on-chip clock generator was included to simplify clock generation.

7.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 7.1. The pinout conforms

TABLE 7.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Unused	5	36	nbias (input for n current mirror)
1 bit output	6	35	Analog Vss
SERIALOUT	7	34	Analog Vdd (tie to pin 26)
Input for slow phi1d	8	33	SCANIN
Input for slow phi2d	9	32	SHIFT (H enables scantest)
Input for slow phi1	10	31	RESET
Input for slow phi2	11	30	filter GND
PAD GND	12	29	filter Vdd
PAD Vdd	13	28	OCDSM
input for fast phi1	14	27	DSMSEL (H for on-chip)
input for fast phi2	15	26	Analog Vdd (tie to pin 34)
CLOCKIN	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
SOCLOCK	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
CLOCKSEL (H for on-chip)	18	23	Unused
SCANOUT	19	22	Unused
Unused	20	21	Unused

to the standard for the analog test board. See the notes in the next section to achieve proper operation of the chip.

7.3 Chip Contents

This chip contains an on-chip clock generator, a second order modulator with double poly capacitors, a decimate-by-64 FIR filter, and an output parallel to serial converter.

The chip requires a set of fast clocks and slow clocks and the fast clock must be 3 times faster than the slow clock. These clocks can be generated off-chip and supplied to the chip at pins 8, 9, 10, 11, 14, and 15. Pin 18 must be set low to enable off-chip clock inputs. For on-chip clocks, a clock that is twice the desired fast clock frequency should be connected to pin 16 and pin 18 should be set high. The on-chip clock generator was functional.

Power and ground leads were separated into pad, filter, and analog pins. This allows direct measurement of the power consumed by the filter core, without adding the power in the pads.

The digital filter can operate in a scan test mode. The input for the scan chain is pin 33 and the output is pin 19. Pin 32 should be set high when scantest mode is used. Another testability feature allows the filter input to come either from the on-chip modulator or from an external source. To use the on-chip modulator, set pin 27 high. For off-chip data, place the data stream on pin 28 and set pin 27 low.

The filter output comes out as a 19 bit serial burst with LSB first data. The output stream appears on pin 7 and the corresponding serial clock appears on pin 17. Since the clock is not active very often, there is little trouble in synchronizing the data acquisition board and a data valid signal is not required. When operating the filter, the RESET pin 18 must be set high and then low to reset the filter. Use a scope to visually verify that the SOCLOCK signal is periodic. If it is not, the RESET sequence was not correct and must be repeated.

7.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 24 chips were provided. The cif file was dsmad2.cif. Of the 24 chips, 20 were found to be fully functional. Chips 16, 17, 18, and 20 were found to be problematic. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 20 chips were found to be functional with a spread of about 3 dB in performance across the chips.

7.5 Comments

The chips performed similar to dsm2p. The references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.0 V, and vref_neg2 = 2.0 V. Bias currents were set to a nominal 40 μA. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF. The chip design was discussed as example 1 of the dissertation.

Chips 23 and 24 were given to Pao Chen, EG&G Reticon.

7.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the folded cascode opamp. Other details on device sizes can be obtained by examining the CIF file.

Chip #8: dsmchip2 Mosis ID Number: M98X DK1

8.1 Introduction

This chip was created by placing the modulators left by Max Hauser and Bosco Leung on a single die. Design notes state that the chip was created about 11/89. The chip was fabricated using Orbit's 2.0 μm, p-well process.

8.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 8.1. The pinout does not

TABLE 8.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Unused	1	40	testin
Unused	2	39	testout
2Analog In+	3	38	Vdd
2Analog In-	4	37	GND
Unconnected	5	36	dout2
1Analog In+	6	35	phi1
1Analog In-	7	34	phi1d
shield	8	33	dout1
Analog Vdd	9	32	phi2d
Unused	10	31	phi2
Analog Vss	11	30	dout3
vref_pos1	12	29	dith-
vref_neg2	13	28	dith+
pbias	14	27	Unused
nbias	15	26	Unused
Unused	16	25	Unused
1Analog Ground	17	24	3Analog Ground
3Analog In-	18	23	s21
vref_neg2	19	22	s24
vref_pos2	20	21	3Analog In+

conform to the standard for the analog test board.

8.3 Chip Contents

This chip contains the first order modulator described in Bosco Leung's Ph. D. dissertation, the last versions of the second order modulator and a first order modulator that Max Hauser worked on.

8.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were received. The cif file used to fabricate the chip was called dsmchip2.cif. There was much difficulty in testing this chip since a wire wrapped test board was used. No specific notes were kept on yield, but at least 1 chip was shown to be functional.

8.5 Comments

Initial studies were performed using a DAS. The serial-to-parallel converter board developed by Max Hauser for the SPUDS system was used for testing.

The cif file was written with an old version of the magic technology file. The old tech file caused all of the n transistors to exhibit design rule violations. To view this file, use the cif istyle lambda=1.0(oldpwell).

8.6 Circuit Diagrams

Circuit diagrams can be found in Bosco Leung's dissertation for his first order loop.

Chip #9: dsmtest
Mosis ID Number: N15W FB1

9.1 Introduction

This chip contains 2 second order modulators, both using class AB opamps and double poly capacitors. One modulator has integrator gains of 0.5. The other modulator has features that allow the isolation and testing of a single opamp. Design notes indicate that this chip was created during 5/91. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process.

9.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 9.1. The pinout conforms

TABLE 9.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	cmfbin
Input for phi1d	8	33	oaouten
Input for phi2d	9	32	oainen
Input for phi1	10	31	io1en
Input for phi2	11	30	oa_inp
Digital GND	12	29	oa_outm
Digital Vdd	13	28	oa_outp
comp_eval	14	27	oa_inn
douten	15	26	Analog Vdd
indout	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
intout*	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
comp	18	23	2Analog In-
comp*	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

to the standard for the analog test board. The chip will function properly if placed in the test board and data from pin 6 will be valid.

9.3 Chip Contents

Two modulators were placed on the chip. On the right side, the modulator uses 1.0 pF sampling caps and integrators with gain set to 1/2. The output of this modulator is routed to pin 6. The modulator on the left was designed so the opamp could be isolated. The integrators are configured for a gain of 1/4, and the sampling capacitor is sized to 2 pF. Figure 9.1 shows a wiring diagram to illustrate how the pads should be connected for various testing modes. A fully differential circuit implementation is used but only a single ended version is shown in the drawing. The signals comp, indout, oa_inp, and oa_outm all have complementary versions.

Setting the signals compen, oaouten, oainen, and io1en all high enables the modulator to work as desired. There are basically 3 modes of operation that can be used.

1. The opamp can be isolated by setting io1en, oaouten, and oainen low. The input to the opamp is accessible through the pins oa_inp and oa_inm and the outputs at oa_outm and oa_outp. The clocked common-mode feedback is disabled, and the voltage placed on cmfbref will control the output common mode point. Also in this mode, the comparator is isolated. Response can be studied by connecting compeval to phi1, and injecting signals with the opamp used as a buffer in unity gain feedback mode. Output of the comparator is available on the pins comp and comp*.
2. Using the same conditions as in mode 1, but setting oaouten high enables the clocked common mode feedback, allowing examination of the opamp under normal operating conditions. Again, the comparator can be tested.
3. Using the same conditions as in mode 2, but setting oainen high forces the opamp into an integrator configuration. The signal compen can be set high to enable the feedback loop, or set low to allow injection of signals into the second opamp input. This is achieved by placing complementary logic values on indout and indout*.

9.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was called dsmtest.cif. Of the 12 chips, all were found to be fully functional. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 12 chips were found to be functional with a spread of about 1 dB in performance across the chips.

9.5 Comments

During testing, the references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.6 V, and vref_neg2 = 1.4 V. Bias currents were set to a nominal 40 μA. Full characterization of these chips was not performed. However, it was noted that basic SNR performance was similar to that obtained with dsm2p. Examining the FFT showed that there was slightly lower noise when using the integrators with gain of 1/2. This is expected since under the additive white noise analysis, the noise shaping is not as steep for the gain of 1/2 case. The larger capacitances did slow down the opamps.

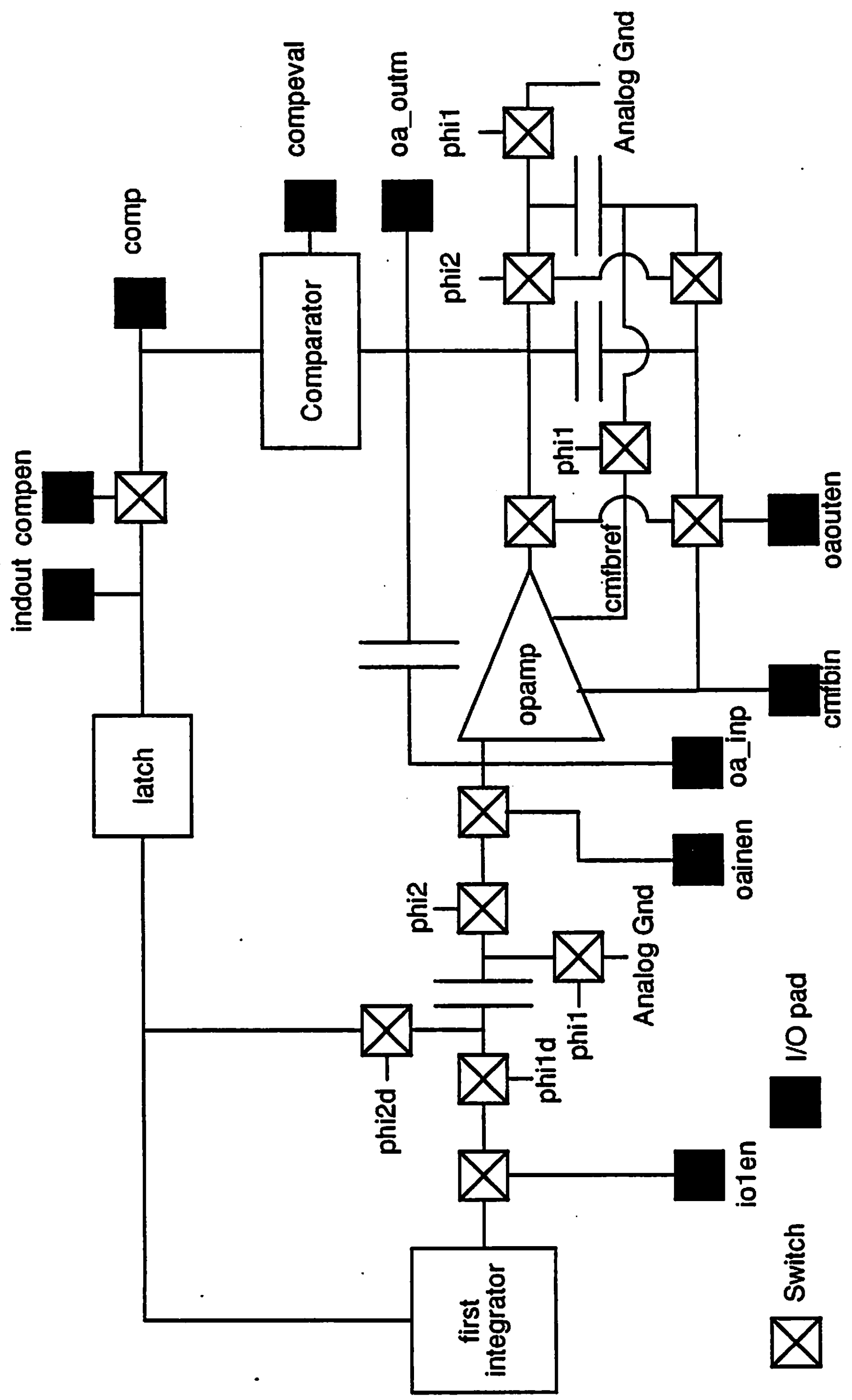


FIGURE 9.1Wiring diagram for the left modulator. A single ended signal path is shown here. Circuits are fully differential.

9.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 2 comparator was used as well as the type 1 Class AB opamp. The circuit diagram for the correlated double sampling integrator is also given. Other details on device sizes can be obtained by examining the CIF file.

Chip #10: fc2p2
Mosis ID Number: N0BO CB1

10.1 Introduction

This chip contains 2 second order modulators, both using folded cascode opamps and double poly capacitors with integrator gains set to 0.25. This chip was created during 10/90 and is a redesign of fcdsm2p. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. It also contains a replica of the biasing circuitry used in the folded cascode opamp and an on-chip clock generator.

10.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 10.1. The pinout con-

TABLE 10.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	bias1
Input for phi1d	8	33	bias2
Input for phi2d	9	32	bias3
Input for phi1	10	31	bias4
Input for phi2	11	30	Toutp1
Digital GND	12	29	Toutn1
Digital Vdd	13	28	Toutp2
phi1	14	27	Toutn2
phi2	15	26	Analog Vdd
phi1d	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
phi2d	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
select (H for on-chip)	18	23	2Analog In-
comp	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

forms to the standard for the analog test board. The chip will function properly if placed in the test board and data from pins 6 and 7 will be valid.

10.3 Chip Contents

Two copies of the modulator were mirrored on the chip. On the left side of the chip, outputs from the first and second integrators as well as the comparator output were routed to pads 27, 28, 29, 30, and 19. To use the on-chip clock generator, set pin 18 high. The 4 clock phases are then derived from the input on pin 10 and buffered an placed on pins 14, 15, 16, and 17. The output of the left modulator is routed to pin 7. The right modulator has no parasitic loading and the output is routed to pin 6.

The replica high swing cascode biasing circuit is shown with pin names in Figure 10.1.

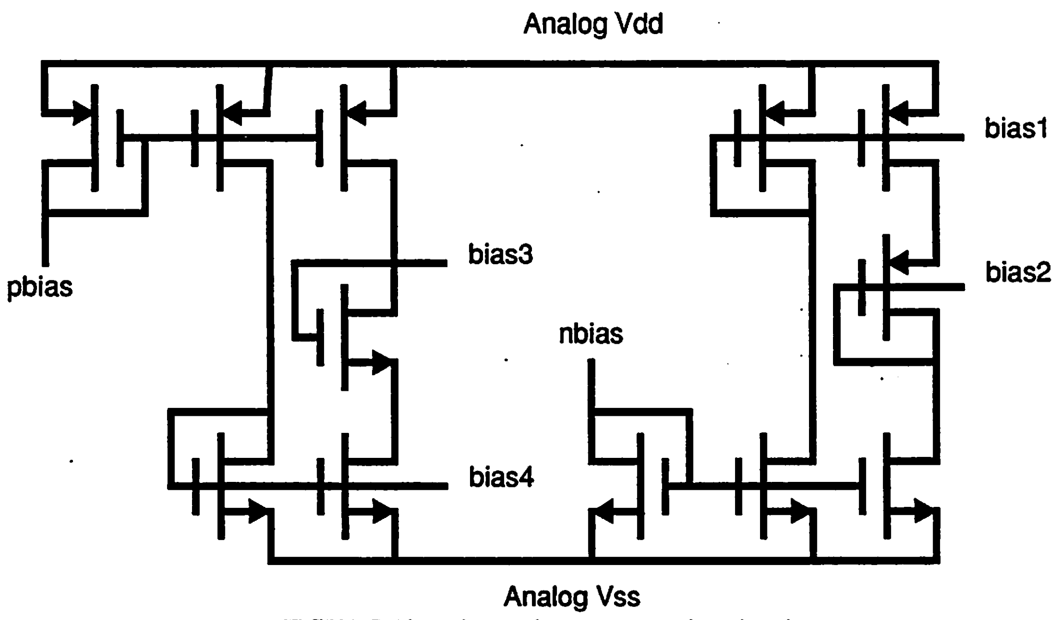


FIGURE 10.1High swing cascode bias circuit.

10.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was fc2pc2.cif. Of the 12 chips, 8 were found to be fully functional. Chips 1, 4, 7, and 12 functioned at a lower performance level with large amounts of harmonic distortion evident in the output spectrum. Performance could be increased by adjusting the bias conditions. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 8 chips were found to be functional with a spread of about 3 dB in performance across the chips.

10.5 Comments

Under speed tests, the chips were found to be functional up to about 3 MHz. Absolute speed numbers are difficult to gauge since performance rolls off slowly. The references were set so that vref_-pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.0 V, and vref_neg2 = 2.0 V. Bias currents were set to a nominal 275 μA, but the integrator outputs for the test modulator were found to be in the wrong operating region. Bias was readjusted until the common mode output was at 2.5V. By measuring the values of bias2 and bias3, it was found that the high swing bias was not correct. SPICE models did not give the correct results for the linear region operation of the devices. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF. The peak SNR was measured at 77 dB and the dynamic range was extrapolated to be 93 dB.

Devices 7 and 12 were given to Pao Chen, EG & G Retcion.

10.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the folded cascode opamp. The on-chip clock generator is as documented. Other details on device sizes can be obtained by examining the CIF file.

Chip #11: fcdsm2p Mosis ID Number: N06M GE1

11.1 Introduction

This chip contains 2 second order modulators, both using folded cascode opamps and double poly capacitors with integrator gains set to 0.25. Design notes state that the chip was created during 6/90. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. 2 cascoded n transistors and 2 capacitors were included as test structures.

11.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 11.1. The pinout con-

TABLE 11.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	ncdrain
Input for phi1d	8	33	ndrain
Input for phi2d	9	32	ncgate
Input for phi1	10	31	ngate
Input for phi2	11	30	bottom1 (bottom plate of capacitor)
Digital GND	12	29	top1 (top plate of capacitor)
Digital Vdd	13	28	top2 (top terminal of capacitor)
tphi1d (test phi1d phase)	14	27	bottom2 (bottom plate of capacitor)
Toutn2	15	26	Analog Vdd
Toutp2	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
comp	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
Toutp1	18	23	2Analog In-
Toutn1	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

forms to the standard for the analog test board. The chip will function properly if placed in the test board, but only data from pin 6 will be valid.

11.3 Chip Contents

Two copies of the modulator were mirrored on the chip. On the left side of the chip, outputs from the first and second integrators as well as the comparator output were routed to pads 18, 19, 15, 16, and 17. In addition, the first modulator has the phi1d clock phase separated and routed to pin 14 to allow an additional testing mode. If this pin is set low, the first integrator is isolated and can be run using the vref1 pins as input. In normal operation, pin 14 was tied to pin 8. The output of the left modulator is routed to pin 7. The right modulator has no parasitic loading and the output is routed to pin 6.

Several test devices were placed on the chip and their schematics are shown in Figure 11.1.

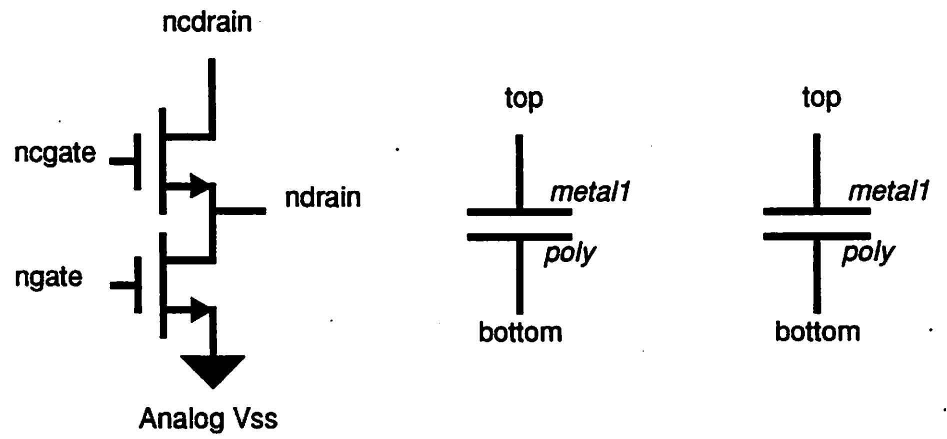


FIGURE 11.1Extra devices put on the chip for device characterization.

11.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was fcdsm2p.cif. Of the 12 chips, 8 were found to be fully functional. Chips 1, 3, 5, and8 functioned at a lower performance level with large amounts of harmonic distortion evident in the output spectrum. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 8 chips were found to be functional with a spread of about 3 dB in performance across the chips.

11.5 Comments

Under speed tests, the chips were found to be functional up to about 3 MHz. Absolute speed numbers are difficult to gauge since performance rolls off slowly. The references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.0 V, and vref_neg2 = 2.0 V. Bias currents were set to a nominal 275 µA, but the integrator outputs for the test modulator were found to be in the wrong operating region. Bias was readjusted until the common mode output was at 2.5V. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF. Using chip 10, peak SNR was measured at 67.5 dB.

After testing this chip, a small design flaw was found in the common mode feedback circuitry. The redesigned version of this chip was called fc2p2. In spite of this error, it is possible to gather quite a bit of useful data from this chip. The design flaw decreases the integrator swing giving rise to large harmonic distortion.

The cif file was written with an old version of the magic technology file, which causes n transistors to be displayed with design rule violations. Use the cif istyle lambda=1.0(oldpwell) if this occurs.

11.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the folded cascode opamp. Other details on device sizes can be obtained by examining the CIF file.

Chip #12: m1pn2
Mosis ID Number: N03W EE1

12.1 Introduction

This chip contains 2 second order modulators, both using Class AB opamps and metal1-poly capacitors, with integrator gains set to 0.25. Design notes state that the chip was created about 3/90. The chip was fabricated using VLSI Technology's 2.0 µm n-well process. 2 cascoded n transistors, a p transistor, and a capacitor were included as test structures.

12.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 12.1. The pinout con-

TABLE 12.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 0V)	1	40	Capshield (Connect to 5V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
1 bit output (test)	7	34	pgate
Input for phi1d	8	33	ncdrain
Input for phi2d	9	32	pdrain
Input for phi1	10	31	ndrain
Input for phi2	11	30	ncgate
Digital GND	12	29	ngate
Digital Vdd	13	28	top (top terminal of capacitor)
tphi1d (test phi1d phase)	14	27	bottom (bottom plate of capacitor)
Toutn2	15	26	Analog Vdd
Toutp2	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
comp	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
Toutp1	18	23	2Analog In-
Toutn1	19	22	2Analog In+
Shield	20	21	Analog Ground (2.5V)

forms to standard for the analog test board. The chip will function properly if placed in the test board, but only data from pin 6 will be valid.

12.3 Chip Contents

Two copies of the modulator were mirrored on the chip. On the left side of the chip, outputs from the first and second integrators as well as the comparator output were routed to pads 18, 19, 15, 16, and 17. In addition, the first modulator has the phi1d clock phase separated and routed to pin 14 to allow an additional testing mode. If this pin is set low, the first integrator is isolated and can be run using the vref1 pins as input. In normal operation, pin 14 was tied to pin 8. The output of the left modulator is routed to pin 7. The right modulator has no parasitic loading and the output is routed to pin 6.

Several test devices were placed on the chip and their schematics are shown in Figure 12.1.

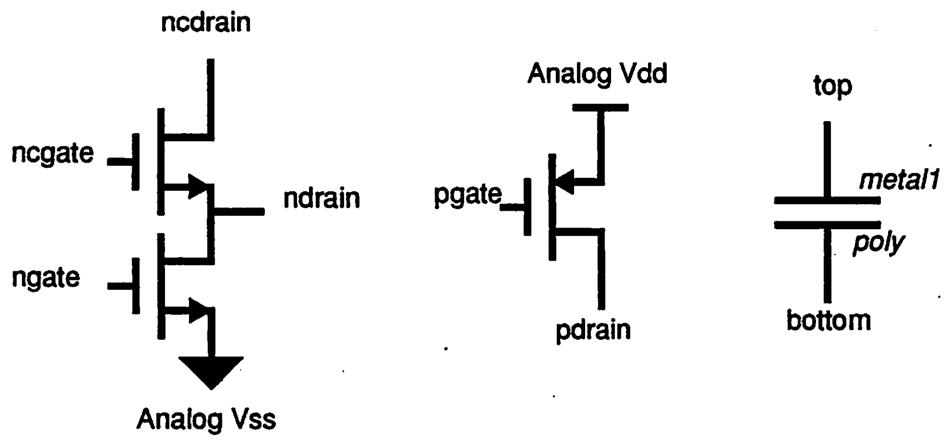


FIGURE 12.1Extra devices put on the chip for device characterization.

12.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was m1pn2.cif. All 12 were found to be functional. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, all chips were found to be functional with a spread of about 1 dB in performance across the 12 chips.

12.5 Comments

Under speed tests, the chips were found to be functional up to about 3 MHz. Speed seems to be limited primarily by the large parasitic capacitance (from the bottom plate of the integrating cap) on the output of the opamps. The references were set so that vref_pos1 = 3.8V, vref_neg1 = 1.2V, vref_pos2 = 3.0 V, and vref_neg2 = 2.0 V. Bias currents were set to a nominal 40 μA. For a sampling rate of 1 MHz and a decimate by 256 sinc^3 decimation filter, typical peak SNR was 77 dB and dynamic range was 85 dB. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF.

12.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the type 1 Class AB opamp. Other details can be obtained by examining the CIF file.

Chip #13: oatest
Mosis ID Number: N0CT DF1

13.1 Introduction

This chip contains a second order modulator and an stand alone integrator, both using class AB opamps and double poly capacitors with integrator gains set to 0.25. Design notes indicate that this chip was created during 12/90. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. It also contains an on-chip clock generator. This chip was designed so that the opamp in the stand alone integrator could be isolated for performance measurements.

13.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 13.1. The pinout con-

TABLE 13.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield (Connect to 0V)
1Analog In-	2	39	vref_neg1 (Adjustable, 0 - 2.6V)
1Analog In+	3	38	vref_pos1 (Adjustable, 2.4 - 5V)
Analog Ground (2.5V)	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
1 bit output (clean)	6	35	Analog Vss
CLKIN	7	34	oa_outp
Input for phi1d	8	33	oa_outn
Input for phi2d	9	32	vcmo
Input for phi1	10	31	cmfbin
Input for phi2	11	30	oa_inp
Digital GND	12	29	oa_inn
Digital Vdd	13	28	to_oap
cdout	14	27	to_oan
cmfben	15	26	Analog Vdd
csp_pos	16	25	vref_neg2 (Adjustable, 0 - 2.6V)
csn_pos	17	24	vref_pos1 (Adjustable, 2.4 - 5V)
csp_neg	18	23	2Analog In-
csn_neg	19	22	2Analog In+
CLKSEL (H for on-chip)	20	21	Analog Ground (2.5V)

forms to the standard for the analog test board. The chip will function properly if placed in the test board and data from pins 6 will be valid.

13.3 Chip Contents

To use the on-chip clock generator, set pin 20 high. The 4 clock phases are then derived from the input on pin 7 and buffered an placed on pins 8, 9, 10, and 11. These pins use bidirectional pads so that if the internal clock generator is bad, the 4 clock phases can be supplied from an off-chip source. The output of the modulator is routed to pin 6.

By configuring connections on the chip, the opamp can be configured as an integrator. Figure 13.1. The layout was developed so these integrator portions could be wired for 4 modes of operation.

1. Test of the opamp core. In this mode, set phi2 and cmfben low. Common mode feedback is controlled through the use of the cmfb_in pin. A full differential opamp can be measured.
2. Test the opamp core with clocked common mode feedback. In this mode, enable the clocks and set cmfben high.
3. Test the opamp in an integrator configuration. Set the clocks and cmfben as in mode 2. In addition, use off chip capacitors and the switch array to create an integrator. This can be done as follows. Short the following pairs of pins: 35 and 32, 30 and 28, 29 and 27. Connect capacitors across these pins: 34 and 29, 33 and 30, 16 and 17, 18 and 19. Set pin 14 to either high or low and set vref_pos2 and vref_neg2 to Analog Ground. Now pins 23 and 22 serve as the inputs to the integrator.
4. Test the opamp as a first order modulator. Set up the pins as in mode 3, but take pins 33 and 34 and use them as the inputs to the a comparator. Take the output of the comparator and connect it to pin 14. This signal is also the output of the modulator.

13.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was oatest.cif. No formal tests were made to see if all of the chips were functional. The chip was made primarily to provide a testable opamp.

13.5 Comments

A single device was characterized for performance. The references were set so that vref_pos1 = 3.9V, vref_neg1 = 1.1V, vref_pos2 = 3.4 V, and vref_neg2 = 1.6 V. Bias currents were set to a nominal 40 μA. Nominal size of the integrating capacitor was set to 2.0 pF and the sampling caps to 0.5 pF. The performance was similar to that obtained from the dsm2p chips.

13.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 1 comparator was used as well as the type1 Class AB opamp. The on-chip clock generator is as documented. Other details on device sizes can be obtained by examining the CIF file.

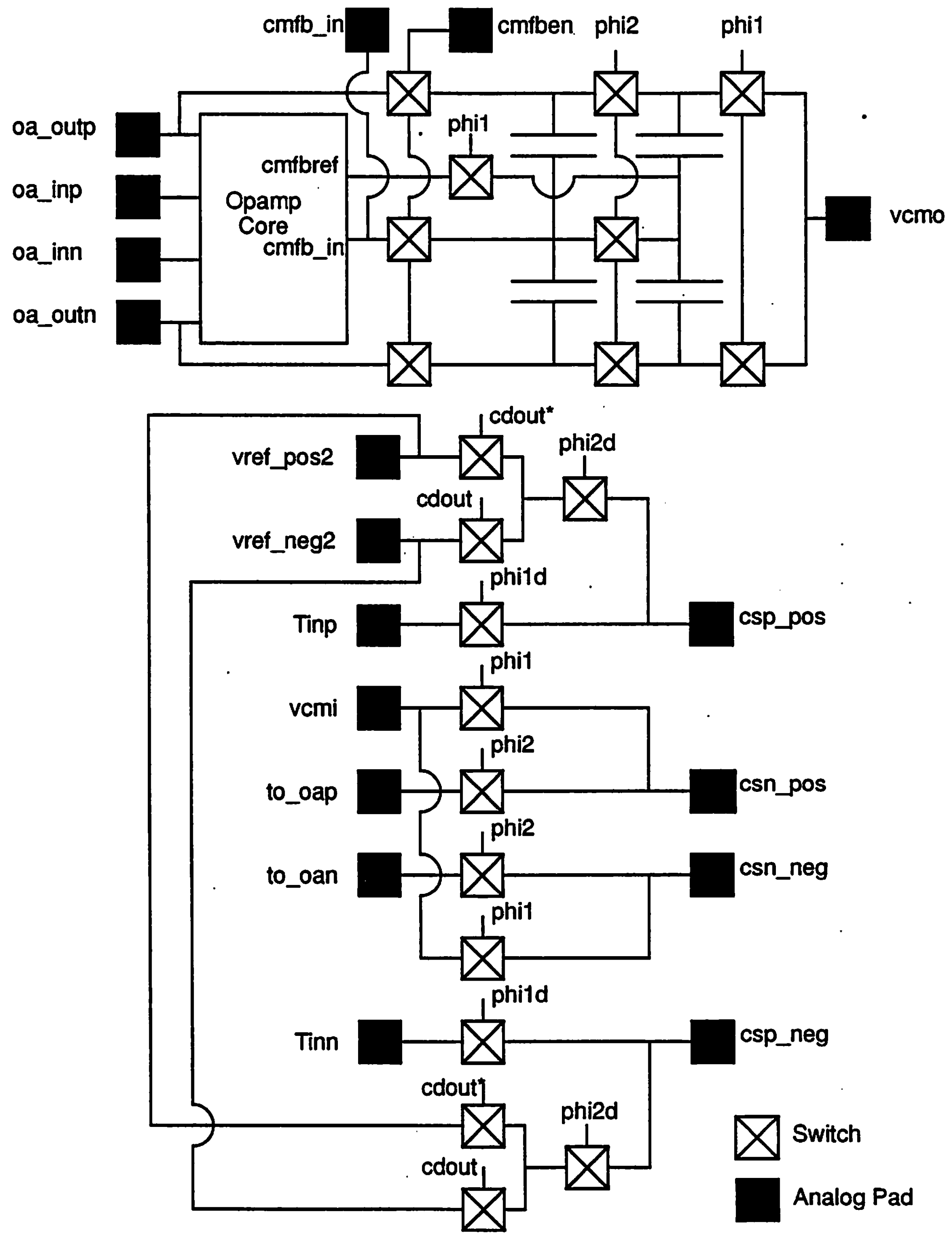


FIGURE 13.1 Wiring diagram for the integrator components.

Chip #14: order3
Mosis ID Number: N15W FC1

14.1 Introduction

This chip contains second order and first order modulators set up in a cascade. When properly configured, the composite circuits can be used as a third order multistage modulator. All integrators use class AB opamps and double poly capacitors with integrator gains set to 0.5. Design notes indicate that this chip was created during 3/91. The chip was fabricated using Orbit's 2.0 μm, p-well, double poly process. It also contains logic to allow collection of both data streams.

14.2 Pinout

The chip was packaged in a 40 pin dip and the pinout is given as in Table 14.1. The pinout con-

TABLE 14.1Pinout for the chip.

Pin Description	Pin No.	Pin No.	Pin Description
Substrate (Connect to 5V)	1	40	Capshield
Unused	2	39	vref_neg1
Unused	3	38	vref_pos1 (
Unused	4	37	pbias (input for p current mirror)
Shield	5	36	nbias (input for n current mirror)
dout1	6	35	Analog Vss
SEROUT	7	34	ioen3
Input for phi1d	8	33	intout_pos3
Input for phi2d	9	32	intout_neg3
Input for phi1	10	31	vref_pos3
Input for phi2	11	30	vref_neg3
Digital GND	12	29	ioen2
Digital Vdd	13	28	intout_pos2
Unused	14	27	intout_neg2
dout2	15	26	Analog Vdd
io1en	16	25	vref_neg2
intoutp1	17	24	vref_pos1
intoutn1	18	23	2Analog In-
Shield	19	22	2Analog In+
CSH	20	21	Analog Ground (2.5V)

forms to the standard for the analog test board.

14.3 Chip Contents

The left modulator is a standard second order modulator. The output is routed to pin 6 as signal dout1. The differential outputs of the last integrator in the left modulator serve as the inputs for the first order modulator placed on the right. The output of the first order modulator is accessible on pin 15 as the signal dout2. The signals io1en, io2en, and io3en control switches placed on the outputs of the 3 integrators. When disabled, the switches block the capacitive loading associated with the routing wires to the chip pads. Io1en and io2en control the isolation of the outputs of the first and second integrators in the left modulator and io3en controls the integrator in the first order modulator. The outputs appear on the pins as labeled. The additional signals vref_pos3 and vref_neg3 serve as the D/A inputs for the first order modulator. They can be connected to the references for the second order modulator if desired.

The signal SEROUT is obtained by combining the signals dout1 and dout2 and the circuit used for this is shown in Figure 14.1.

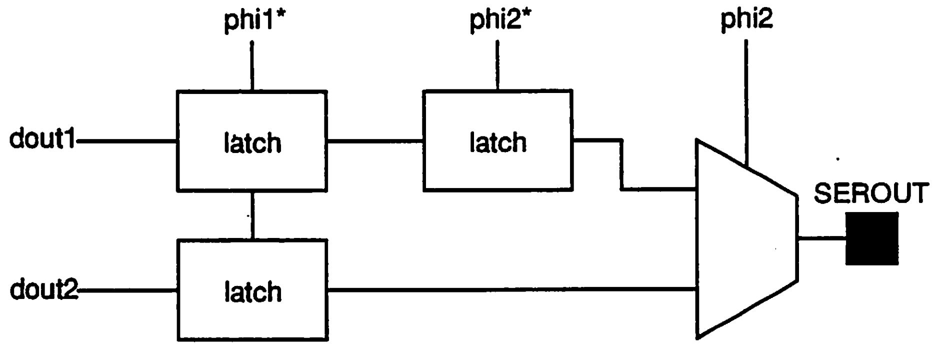


FIGURE 14.1Logic used to combine the output bit streams into a single stream.

14.4 Yield Statistics

Chips are numbered in pencil, usually near the label where the Mosis ID number is found. On this run, 12 chips were provided. The cif file was cdsdsm.cif. Of the 12 chips, all were found to be fully functional. The chips were tested by applying an input sinusoid and measuring SNR response. Under these conditions, the 12 chips were found to be functional with a spread of about 3 dB in performance across the chips.

14.5 Comments

During testing, the references were set so that vref_pos1 = vreg_pos2 = vref_pos3 = 3.2V, and vref_neg1 = vref_neg2 = vreg_neg3 = 1.8V. Bias currents were set to a nominal 40 μA.

The master clock on the test board is divided by 2, so it can be used as the clock to collect the signal SEROUT. Once the data stream was collected in software, a filtering algorithm was implemented to combine the bit streams into a single stream with third order noise shaping. See the papers by Ribner for information about this type of second-order first-order cascade. It was found that the performance was not better than a single second order modulator since the third order noise shaping was masked by the noise floor caused by 1/f noise in the opamps.

14.6 Circuit Diagrams

Circuit diagrams can be found in the dissertation. The type 2 comparator was used as well as the type 1 Class AB opamp. The circuit diagram for the correlated double sampling integrator is also given. Other details on device sizes can be obtained by examining the CIF file.