Abstract

Time-frequency analysis techniques, such as wavelet decomposition and Gabor filtering, are a tool for efficient coding of short-term acoustical features, and so are fundamental to acoustic pattern classification and speech recognition.

This thesis addresses issues of efficiency and robustness in the design and implementation of acoustic signal processors and small-vocabulary speech recognition systems for applications where power dissipation and integration density are primary design constraints. We couple time-frequency signal representations with massively parallel architectures using analog VLSI technology to design compact special-purpose systems with power efficiency surpassing conventional DSPs.

We present the design of, and results from, several prototyped VLSI systems, including processors for time-frequency decomposition and template-correlation-based acoustic transient pattern classification. We present methods for automatically training a template correlator and discuss potential research directions for this architecture, including biological modeling and continuous-speech recognition.

Contents

Abstract ii

Acknowledgements iii

List of Figures vii

List of Tables xii

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Mapping the Time-Frequency Plane . . . 6
  - 1.2.1 Current-mode filterbanks . . . 8
  - 1.2.2 Acoustic Transient Recognition . . . 9
  - 1.2.3 Analog VLSI implementation of the transient classifier . . . 10
  - 1.2.4 Overview: Learning and Continuous Speech Recognition . . . 11

2 Continuous Wavelet Transform 14
- 2.1 Introduction to the 1-Dimensional Continuous Wavelet Transform . . . 14
- 2.2 CWT *vs.* DWT . . . 16
- 2.3 Gabor Logons and Wavelets . . . 18
- 2.4 Complex Demodulation . . . 20
- 2.5 Complex Demodulation in the Continuous Wavelet Processor . . . 23
- 2.6 Post-processing . . . 27
- 2.7 An Analog CWT Processor . . . 28
- 2.8 Generating carrier sinusoids . . . 29
- 2.9 Analog multiplication . . . 30
- 2.10 Wavelet Gaussian Function . . . 30
- 2.11 Wavelet chip slice . . . 34
- 2.12 Chip Specifications . . . 34
- 2.13 Limitations of the Architecture . . . 36
- 2.14 Variations on an Architecture . . . 37
- 2.15 A Mixed-Mode Wavelet Processor . . . 38
- 2.16 Details of the Bit-Sequence-Finding Algorithm . . . 42
- 2.17 Sequence generation . . . 44

4.4.1 Current-switching Memory Array . . . 135
4.4.2 Bucket Brigade Device . . . 138
4.4.3 Circuit input section . . . 140
4.4.4 Characterizations of the VLSI Hardware . . . 141
4.4.5 Experimental Results . . . 144
4.4.6 Summary . . . 147
4.5 The Digital ATP . . . 149
4.6 Digital correlator custom VLSI architecture . . . 149
4.7 Digital correlator semicustom FPGA architecture . . . 151
4.8 The Switch-Capacitor Frontend . . . 156
4.9 Experimental results of the trinary-trinary correlation hardware . . . 161
4.9.1 Method of input segmentation . . . 161
4.9.2 Method of template generation . . . 163
4.9.3 Experimental Results . . . 164

5 Learning and Speech Recognition **171**
5.1 Automatic Template Learning for Template-Based Correlation . . . 171
5.2 Average-Value Templates . . . 172
5.3 Deterministic Methods: Statistical Component Analysis . . . 174
5.4 Support Vector Machines . . . 177
5.5 Heuristic Methods (Unnikrishnan/Hopfield) . . . 178
5.6 Biologically-Inspired Methods . . . 183

A Linearity of a Transconductance Amp **187**

B MATLAB Code for Sine Sequence Generation **190**
B.1 Commentary . . . 193

C Correlation with Time Differentiation **195**
C.1 Proof of validity of the pipelined architecture . . . 198

D Simulation of the ATP Frontend **201**
D.1 Commentary . . . 207

E Simulation of the ATP Correlator **209**
E.1 Commentary . . . 212

F Optimizing per-class gains in the ATP **214**
F.1 Commentary . . . 219

Bibliography **220**

Vita **227**

2.23 Single discrete-time lowpass filter section. 52
2.24 A scheme for controlling time-multiplexing of the outputs. 53
2.25 Sixteen-channel architecture using the 7-to-5 frequency ratio. 54
2.26 The wavelet chip, block diagram. 55
2.27 Photomicrograph of the mixed-mode continuous wavelet transform processor, a 4 mm×6 mm die size fabricated in a 2 µm CMOS p-well process. 56
2.28 Signal demodulation using the wavelet chip. The top trace is the input signal. The middle trace is the input multiplied by the binary sequence. The bottom trace is the output after filtering. 57
2.29 Gaussian filter magnitude response: predicted and measured. 58
2.30 Gaussian filter phase response: predicted and measured. 59
2.31 Wavelet responses to isolated sine wave input. 59
2.32 An arbitrary tiling of the time-frequency plane. 60

3.1 Frontend filterbank system—block diagram. 64
3.2 Sampled-data input from a recording of an acoustic transient—that of a book being dropped onto a desk. 67
3.3 Bandpass-filtered acoustic transient input using two cascaded second-order filters on each channel. 69
3.4 Parallel filterbank output after rectification and smoothing of the acoustic transient input signal across all frequency bands. 71
3.5 Translinear loop with common-base and common-emitter configurations. 74
3.6 Filter pole formed using a transconductor. 76
3.7 First-order log-domain filter circuit. 77
3.8 Computing a current difference at a log-domain filter input. A) The underlying idea, which is physically unrealizable. B) An equivalent working implementation. . . . 80
3.9 Bandpass structure formed from first-order sections. 81
3.10 An alternative common-emitter circuit generating I'_{DC} (see text for discussion). . . 82
3.11 Complete circuit schematic for the second-order bandpass filter. 83
3.12 Another BiCMOS Log-domain bandpass filter, using the common-emitter configuration for all feedback circuits. 84
3.13 Base compensation (B) eliminates undesirable behavior due to significant base current which occurs in (A). 85
3.14 Complete circuit schematic for the second-order bandpass filter. Bipolar transistors are minimum size, and MOS dimensions are indicated as W/L in units of $\lambda = 0.6$ µm. 86
3.15 Circuit of Figure 3.12, showing base compensation circuits, cascode connections, and the Q-generating circuit 87
3.16 Structure of a bandpass filterbank channel and stacking of channels to form the whole filterbank. 89
3.17 Simplified schematic of the adaptive current full-wave rectifier. 91
3.18 Cascaded log-domain lowpass filters. 92
3.19 Simple diode-based peak detector. 92
3.20 Behavior of the peak-peak detector. 94
3.21 Peak-peak detector circuit, simplified. 95
3.22 Peak-peak detector with cascodes, as fabricated. 96

4.26 Measured correlation of repeated sounds "can" (left) and "snap" (right) with the "can" template loaded into chip memory. 148
4.27 Measured correlation of repeated sounds "can" (left) and "snap" (right) with the "snap" template loaded into chip memory. 148
4.28 Block diagram of the sequential digital correlator architecture. 151
4.29 Memory allocation in the digital correlator SRAM. 153
4.30 SRAM address generator for the digital correlator. 154
4.31 Digital correlator controller state diagram, simplified. 155
4.32 Serial digital correlator structure for one template. 155
4.33 Digital ATP correlator system, configurable for up to sixteen templates. 157
4.34 Switched-capacitor bandpass filterbank, single channel. 160
4.35 Audio input circuit for the frontend system. 160
4.36 One board of the switched capacitor frontend filterbank, bandpass filtering and encoding the audio input into sixteen parallel channels. 161
4.37 A mixed-signal method for detecting the onset of a transient event from the output of an analog frontend filterbank. 162
4.38 A simple finite state machine which detects the onset of transient events in the trinary-valued frontend filterbank output. 163
4.39 Templates for twelve transient classes, determined from data obtained from the frontend hardware. Frequency is on the x-axis, with the lowest frequency channel on the right, and time is on the y-axis, with the transient onset at the bottom of the template. Colors correspond to trinary values as follows: black = -1, white = $+1$, gray = 0. 165
4.40 System input and output for sound "bar" (left) and "book" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment. 166
4.41 System input and output for sound "shelf" (left) and "tub" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment. 167

5.1 Unnikrishnan, Hopfield, and Tank correlator architecture, block diagram. 179
5.2 Unnikrishnan *et al.* correlator simulation results on isolated digit recognition. . . . 180
5.3 Response of architecture in simulation to the same input as Figure 5.2 but with white noise added to the input. The system remains robust in the presence of noise. . . . 180
5.4 Gaussian kernels which diffuse the input in the Unnikrishnan *et al.* model. 181
5.5 Example of a biologically-inspired template encoding a falling tone of specific frequency and rate. *left:* Continuous-valued encoding. *right:* Trinary encoding. 184

A.1 Simple differential pair transconductance amplifier. 188
A.2 Linear error in a simple transconductance amplifier as a function of the differential input voltage. 189

List of Tables

2.1 Technologies used for the Wavelet processors. 35

3.1 Measured filterbank characteristics. 100

4.1 Classes and descriptions of the recorded transient dataset. 121
4.2 Confusion matrix for leave-one-out cross-validation loop, using the baseline algorithm. 124
4.3 Simulation results with different architectures. 126
4.4 Confusion matrix for leave-one-out cross-validation loop, using binary $(1, 0)$ templates and channel differencing. 126
4.5 Simulation results for different methods of computing channel differences 127
4.6 System accuracy with and without per-class normalization. 131
4.7 Transistor, cell, and array sizes corresponding to Figure 4.16. 137
4.8 Left: logic for the binary-binary correlation operation. Right: logic for the trinary-trinary correlation. "X" represents a don't-care condition. 150
4.9 2-bit Sign-amplitude trinary representation of input and template values in the digital ATP. "X" represents a don't-care condition. 159
4.10 Offline recognition task: Cross-validation on the transient dataset using a non-real-time segmenter. 168
4.11 Real-time recognition task: Cross-validation on the transient dataset using the simple finite state machine segmenter. 169
4.12 Real-time recognition task: Cross-validation on the transient dataset using the simple finite state machine segmenter. In this transient set, spurious transient events were hand-tagged as "unknown" classes (denoted by a question mark). 170

A digital system has what is known as "restoring logic." The system is powered by positive and negative (ground) supplies which also represent the ideal logic values. Whenever a digital operation occurs, a node capacitance is forced to the value of one of the power rails by charging or discharging completely. At a glance, there is seemingly little difference between the discharging capacitor of a digital operation and the rush of current through a sodium channel in a spiking neuron. The difference is one of method, scale, dimension, and material. In the end, it is the material of which electronic circuits are made that determines the ultimate constraints on scale and dimension. As integrated electronic devices get smaller, power consumption per operation does drop by leaps and bounds. But the operation of digital circuits demands that each circuit be able to maintain two distinct states, and the particular nonlinearity of the transistor (exponential for bipolar devices and also for MOS devices below the threshold voltage) requires a voltage margin to keep the binary states distinct. Attempts to lower the transistor device threshold only shorten the voltage margin by shifting the exponential curve downward, decreasing the dynamic power dissipated but increasing the static power dissipated. For a circuit of a given size (number of transistors) and frequency, there is an optimum threshold and power supply voltage (there is still much room for improvement in today's digital fabrication processes), which minimizes power consumption, but the fundamental limit is imposed by the physical properties of the silicon p-n junction itself, and the power savings cannot be continued indefinitely. For silicon circuits to get power efficiency above this limit, a difference in method is required. Analog processing is one way to go about it. If the system is continuous-valued and not binary, then there is no need for restoring logic and the system can operate at rates of power dissipation which are orders of magnitude below those of digital systems. Of course there is a tradeoff in moving to low-power in the analog domain, and that is the presence of noise (thermal noise, shot noise, and $1/f$ noise), nonlinear distortion, mismatch, drift, slewing, parasitic inductive and capacitive coupling, and temperature dependence. These unwanted effects have to be addressed in every circuit design.

Today, however, as digital systems become increasingly fast and densely packed and decreasingly expensive (though expense reflects more the economy of scale than the cost of production or operation), it is necessary to choose analog applications wisely. There is no point in creating an analog system which will be outperformed by an equivalent digital (microprocessor or DSP) system in a year or two when digital fabrication processes move to the next smaller feature size and system designs move to a lower power supply standard. The incredible advances of CMOS fabrication processes (as per Amdahl's Law) [7] remain virtually unchecked as yet by their fundamental limits. These advances and the prospect of their continuation have the tendency to reduce the subject of

daunting but often greatly rewarding endeavor. This thesis explores one small corner of the space of neuromorphic engineering design: I hope that in addition to educating the reader, it conveys a sense of the scope of possibilities contained in the field, and the sense of excitement and exploration felt by those whose research touches the neuromorphic engineering community.

I have directed my research, which began while I was in the Masters degree program at Stanford, towards investigating methods in auditory signal processing which lend themselves to implementation in analog hardware, specifically large-scale fabrication in silicon. A general goal of all my projects, and of the research groups of which I have been a part, has been to examine systems which are not easily implemented in a digital environment due to size, scalability, or power constraints, and find systems in the analog or mixed-mode domain which perform the same task at low power and high robustness without compromising performance. Typical methods include using current-domain subthreshold circuits for ultra-low power operation [1, 3], and using translinear circuits to compute many arithmetic functions of current and voltage in a minimal area [34]. These methods, along with thoughtful system design, make efficient use of space on silicon. In order to compare analog and digital systems fairly, it is critically important not to stop at merely simulating these systems, but to fabricate and test them as well. Each system described in this thesis has been put onto silicon and characterized by measurements made in the laboratory.

My contributions to the field of electrical engineering and associated fields include:

- An analog architecture for implementing the Continuous Wavelet Transform using the method of complex demodulation, and the use of a cascade of lowpass filters in conjunction with complex demodulation to realize a Gaussian-shaped bandpass filter [17].

- A mixed-mode architecture which used oversampling methods to perform linear analog multiplications [20].

- A mixed-mode architecture for a Continuous Wavelet Transform processor, demonstrating the use of the oversampling method for sine wave generation and complex modulation and achieving low power consumption [18, 21].

- A method for synthesis of log-domain filter circuits, especially as relates to audio-frequency applications [48].

- Circuit architectures for current-mode audio-frequency filterbanks using log-domain filter techniques [49].

1.2 Mapping the Time-Frequency Plane

The time-frequency plane is a powerful concept in signal processing [8, 13, 15]. All acoustic signals can be represented as functions of both time and frequency; speech recognition and acoustic pattern classification systems require that acoustic information be mapped into a function of both time and frequency. In Chapter 2, I summarize the problem of mapping the time-frequency plane, and show how different mappings can yield different information about acoustic signals.

A signal can be divided into discrete units (samples) which adequately describe the signal; *i.e.*, the original signal can be reconstructed from its component units. This is true only if all samples cover the total area of interest (total bandwidth and timespan of the signal) in the time-frequency plane, where *coverage* is determined by the area of uncertainty of each sample,

$$\Delta f \Delta t \geq \frac{1}{2}, \tag{1.1}$$

the acoustic corollary of Heisenberg's uncertainty principle in which matter waves are replaced by acoustic pressure waves. Every sampled output of an acoustic signal-processing system can be viewed as covering a small area in the time-frequency plane. The representation of the area of support of a system as a rectangle with width Δt and height Δf is an idealization of real systems which may be primarily localized in a small area of time-frequency space but spread well beyond the boundaries. The Gaussian is in fact the only function satisfying the minimum $\Delta f \Delta t$ area [8], yet it is nonzero to infinity: a true Gaussian is in fact noncausal, but causal approximations to the Gaussian profile have near-minimum area.

The two most common divisions of the time-frequency plane are the *Dirac map* and the *Fourier map* [15], shown in Figures 1.1(a) and (b). The Dirac map corresponds to a sampled signal, and the area covered by each sample is the frequency spread times the sample period. The Fourier map comes from the Fourier transform, which divides the frequency plane into even parts. The non-windowed Fourier transform assumes an infinite timespan for the signal, and therefore does not have minimum area samples in the time-frequency plane.

Each of these methods yields different information: The Dirac map yields the *best* time resolution but the worst frequency resolution; the Fourier map achieves the best frequency resolution at the expense of having the worst time resolution. Overview: Continuous wavelet decomposition

This view of the time-frequency plane naturally leads to ideas of ways to create time-frequency maps which achieve the optimal tradeoff between time and frequency resolution. One of these is the *wavelet map* (Figure 1.1(c)) [15]. An important property of the wavelet map is that the

simple lowpass filters in cascade: the total transfer function converges to a Gaussian function as the cascade length is increased.

We have designed and fabricated a series of analog and mixed-signal chips to perform the wavelet transform. Two major obstacles to overcome were determining 1) how to generate a sine wave with low harmonic distortion and 2) how to implement a highly linear multiplier. We developed a robust system by using a hybrid analog and digital architecture [18, 20, 21]. We solved both problems of the original design by using an *oversampled binary representation* of the modulating sine wave [20].

1.2.1 Current-mode filterbanks

It is not always necessary to preserve phase information in order to perform acoustic recognition and classification tasks. In such cases a simple filterbank of bandpass filters may suffice, with the outputs rectified and smoothed to yield an estimate of signal energy in a particular band at a specific instant in time. In Chapter 3, I consider the problem of designing such a filterbank using *current-mode* circuits as a frontend processor for the acoustic recognition system described in Chapter 4. The filter transfer functions in a current-mode filterbank are the ratio of output to input *current* rather than voltage. I chose a current-mode architecture partly because it is convenient to interface to the backend systems, which are also current-mode circuits, but more importantly because current-mode filters have a larger dynamic range (on the order of 10^6) than voltage-mode circuits which have similar power-consumption and layout area. Subthreshold analog voltage-mode filterbanks have been well-researched and well-developed [1, 5, 51], but remain subject to low dynamic range.

Current-mode filterbanks, on the other hand, are only beginning to be extensively investigated. Recent work on the relatively new subject of *log-domain filters* [39, 47] has provided some circuit designs and methods for current-mode filters. Working with circuit ideas from Philippe Pouliquen and Wolfgang Himmelbauer, I developed a new method of generating log-domain filter circuits given a transfer function, and fabricated some low-power BiCMOS designs on a test chip which proved to implement the filter functions correctly. Figure 3.11 shows one bandpass filter design.

A useful aspect of log-domain designs is that filter parameters such as center frequency f_0 and resonance Q are controlled directly by bias currents, permitting the same circuit to be used for each filter of the bank, with bias currents appropriately scaled for each one.

imately 1/10 second (64 bins at 2 ms) per template requires over 10 million multiplies per second. This is feasible using dedicated software. However, we view the template correlation as a fundamental low-level task of potentially many kinds of acoustic recognition and classification systems, and as such it should be made as simple, compact, and power-efficient as possible.

We simplified the correlation equation from the standpoint of analog hardware implementation through a series of steps, arriving at an algorithm with requirements for memory storage reduced to a single bit per template value (thus, $M \times N$ bits per template) plus N analog values stored in a shift-and-accumulate register. The accuracy of the algorithm in classification tasks remains little affected by the sparse encoding of information in the template.

1.2.3 Analog VLSI implementation of the transient classifier

Figure 4.7 shows a block diagram of the correlator architecture we developed. The details of the circuits which implement the frontend section, normalizer, and correlator are explored in Chapter 4. Simple memory circuits such as those found in RAM chips provide the binary template storage. Currents from each channel input are switched onto N common lines depending on the template value for channel m and time n. Merging currents together performs the summation across the M channels. Two *bucket brigade devices* (BBDs), circuits usually used for analog delay lines in high-fidelity audio and other acoustic applications [60, 62], perform summation across time n. Summed currents are converted to a voltage on each bucket brigade node by integrating onto the bucket brigade capacitor for a fixed period of time, and we compute the difference between the last bucket brigade cell of the two BBDs with a simple switched capacitor circuit. The bucket brigades are fully pipelined so that for each time t the system produces one complete correlation computation at its output node.

We have made a simple estimate of power consumption based on detailed knowledge about the circuit components used in the VLSI layout. This power consumption typically should be less than 10 microwatts for each template. We expect a system with a large "vocabulary" of 200 transient events using the normal timestep of 2 ms to consume less that 2 mW of power. This system computes all correlations on all templates in parallel, and therefore does not need to employ any measures to reduce the search space of the solution; and it produces a complete correlation at every timestep, so segmentation of the input is not necessary, though it may be desired for robustness, particularly in instances where the transients are presented to the system in isolation.

I made an analysis of the performance of the algorithm in simulation under different archi-

of template generation can eliminate the need for a segmenter, allowing the correlation system to operate continuously and increasing system robustness to situations such as overlapping transient events. In Chapter 5, we investigate training methods based on information-theoretic and neural-network-like methods. Another step is to incorporate a learning algorithm into the system so that template values can be trained by on-chip adaptation. In this thesis, however, I will only consider the intermediate step of "chip in the loop" training, in which a computer evaluates the template adaptation based on correlation results of the hardware system itself.

Currently, the most successful and most widely used method for automated speech recognition is the Hidden Markov Model (HMM) [65]. Systems based on Hidden Markov Models must incorporate many intricate strategies to reduce the search space in order to make computation times reasonable. Digital systems incorporating large-scale parallelism can investigate many possibilities in real time, but are bound to be power-consuming, especially as real-time speech recognition systems constantly push the limits of available technology. The human brain, however, remains the ideal against which all of today's state-of-the-art systems are measured, and it succeeds at this amazing task using a tiny fraction of the power of the digital systems which it unfailingly outperforms. The potential gains of realizing speech processing systems in analog hardware make investigating them worthwhile. Template correlation algorithms and architectures have the potential to provide a fundamental layer of processing upon which a speech processing system may be built. The thesis concludes with a look at two different approaches to achieving this goal. The first is an architecture proposed by Unnikrishnan, Hopfield, and Tank [68, 69]. Like the acoustic transient processor, the algorithm was developed with analog hardware implementation in mind. Also like the acoustic transient processor, it is based on template correlation. It is presented as a neural network system, but after a small amount of algebraic manipulation, one can arrive at a description which bears strong resemblance to the template correlation equations. The main difference between the two is that the Unnikrishnan *et al.* system is a continuous-speech digit recognizer, able to cope with some of the problems of complex signals of long duration, but not requiring the solution of complicated syntactic and semantic issues faced by more general speech recognition systems. In simulation, the system achieves accuracies of 98 to 99% on the TIDIGITS database. Its requirements of the hardware are formidable, however, which is one reason that the system has never been implemented outside of computer simulation, where it runs very slowly and inefficiently due to the analog nature of the design.

The second approach is a biological model of auditory processing in the human brain, based on physiological data about the neural connections in the auditory cortex [70]. There are areas

Chapter 2

Continuous Wavelet Transform

2.1 Introduction to the 1-Dimensional Continuous Wavelet Transform

Wavelet transforms are time-frequency transforms which map the time-frequency plane in the manner of Figure 1.1 (c) using specific dilations and translations. Tilings of higher-frequency areas of the plane have larger bandwidth and thus, in accordance with the uncertainty principle, shorter timespan. The bandwidth is proportional to the position of the tile along the frequency axis. The relationship between frequency position and bandwidth (and inversely, timespan) is in keeping with the Nyquist sampling theory which states that in order to accurately capture the information about a signal (so as to be able, for instance, to reconstruct that signal), the signal must be sampled at a twice the value of the highest frequency in the signal. In other words, the timespan covered by the samples must be inversely proportional to the bandwidth of the signal. This makes wavelet tiling more "natural" in a sense, and while it cannot guarantee the most efficient representation of any arbitrary signal, it will tend to be a good representation for most natural signals.

It is important to explain what I mean by a "good representation." A voice-band recording sampled at 8 kHz can be called a "good representation" in the sense that it contains all the information about the signal necessary to reconstruct (*e.g.*, play back) the recording. On the other hand, a sampled signal (without any further transformations) is rarely used for speech recognition because the speech signal is encoded into the time-domain signal in complex ways that cannot be detected by a purely time-domain analysis (the Dirac basis). Equivalently, a Fourier transform (Fourier basis) of the signal is of limited use. A windowed Fourier transform does fairly well at capturing the essence of the speech recording, but if the window is of fixed length, then there will always be some events which are too short in time to be accurately represented. Two solutions to that problem

wavelet mother function is bound is space by satisfying the stronger relationship $\psi(t) = 0$ outside of a small region of compact support. Bounding the frequency is a matter of filtering. Because the system is discrete-time, the filtering functions are FIR filters and the bandpass characteristic results from sequential application of a lowpass and a highpass filter [10], a common practice in signal processing. An elegant "pyramid" algorithm [11] defines filtering over different frequency scales in a recursive manner in a way that allows information from higher frequencies to be pushed into lower frequencies as the recursion progresses.

Using well-developed discrete-time filter theory, the FIR highpass and lowpass filters can be transformed into an efficient pipelined butterfly filter (also known as a lattice filter) [12]. The FIR filter coefficients or the equivalent lattice filter coefficients are uniquely determined by the mother wavelet basis function. The length of the FIR or butterfly filter is the wavelet order. Higher-order wavelet functions typically yield better distribution of information among the wavelet dilations, leading to better data compression when the wavelet system is used for that purpose.

2.2 CWT *vs.* DWT

The Continuous Wavelet Transform (CWT) is an analog filtering function and is similar to what is known as the Gabor spectrogram [13]. Similarly to the Discrete Wavelet Transform, it requires operations of lowpass and highpass filtering at different scales. However, the filtering functions are performed on the input in parallel, as a filterbank, with the lowpass and highpass functions combined into a single bandpass function. The different scales are interpreted as adjacent bands in the frequency domain, with the bandwidth increasing proportionally with the center frequency of the band: thus the CWT is described by a constant-Q filterbank. That description is shown graphically in Fig. 2.1.

The DWT produces the equivalent result by starting with a block of discrete data and performing successive high- and lowpass digital filtering. The filtering is repeated on the lowpass output in order to bandpass the signal in a series of stages called "dilations." Each dilation divides the frequency space of the current interval in half while doubling the time span, thus keeping the time-frequency product constant. In contrast, the CWT divides a signal into a set of logarithmically-scaled frequency bands by passing it through a bank of constant-Q bandpass filters. Both the CWT and the DWT are filterbanks, although the CWT takes the more intuitive form of a physical filter with a transfer function in the frequency and time domains. DWT filters are carefully formulated mathematical constructs whose transfer functions are recursive and which can only be said to per-

form "lowpass" and "highpass" functions in a vaguely-defined sense. Due to the constraints of the Discrete Wavelet Transform specification, all DWT filters have compact support. The fact that the DWT can be confined absolutely in time and frequency is proof that it cannot be implemented by physical filters in a continuous time domain. By being physically realizable, the CWT cannot achieve perfect compact support. Instead, analogously to the way elliptical filters such as the Chebyshev and Butterworth are generated, CWT functions are constructed in order to achieve maximal compactness with respect to some criterion.

A major difference between the DWT and the CWT is that the CWT loses the concept of *compact support*. The continuous-time nature of the transform implies the use of causal filters, whose frequency response cannot be perfectly limited to a given bandwidth. Instead, the filters must overlap. The shape and complexity of the filter determines the amount of overlap and thus higher-order filter functions can be used to describe higher-order continuous wavelet transforms, analogously to the way that higher-order functions operate in the discrete wavelet transform.

2.3 Gabor Logons and Wavelets

The beginnings of the theory of the continuous wavelet transform begin with a seminal work by Gabor [8], a paper with the rather bold title "Theory of Communication" (1946) which outlines the physical basis behind limitations of time-domain and Fourier analysis and shows how both time and frequency can be incorporated into a function providing the optimal tradeoff in resolution between the two domains, and how the time-frequency plane can be effectively represented by tiling with these functions. Gabor calls the functions "logons," those functions which have minimum $\Delta\omega\Delta t$. The simplest logon form (lowest order CWT filter) is the Gaussian function,

$$\Psi(\omega) = e^{-\omega^2/2\sigma^2}. \tag{2.5}$$

The laws of Fourier transforms dictate that a Gaussian in the time domain is also a Gaussian in the frequency domain, thus this function is smooth in all directions on the time-frequency plane. The properties of Fourier transforms also dictate a symmetry between time and frequency shifts of a signal and multiplication by a complex sinusoid in frequency and time, respectively:

$$\Psi(\omega) = e^{-j\omega t_a}\, e^{-(\omega-\omega_b)^2/2\eta\sigma_b^2} \tag{2.6}$$

$$\psi(t) = e^{-j\omega_a t}\, e^{-(t-t_b)^2/2\sigma_b^2} \tag{2.7}$$

Other useful continuous wavelet filter functions can be derived starting from different criteria for compactness. One such case is the chirplet [16], which allows rotation of the filter with respect to the time and frequency axes.

The frequency shift of the Gabor logon, which is a multiplication by a complex sinusoid in the time domain, amounts to a signal demodulation, where in order to maintain the symmetry between the time and frequency sides of the transform, the complex nature of the frequency shift must be maintained; that is, the modulation must be performed with both a sine and a cosine to represent the real and imaginary parts of the multiplication with the complex exponential.

2.4 Complex Demodulation

Continuous-time bandpass filter functions with arbitrarily high Q values are notoriously difficult to design with precision (a fact which will be discussed in detail in Chapter 3, Section 3.5). It is not especially difficult to build a second-order filter section which can be tuned externally to the desired specifications, but managing to get an entire filterbank of 16 or 32 independent sections to all match within a given tolerance can be difficult or impossible, depending on the strictness of the specifications. Adding a requirement of low power consumption compounds the problem. Generally, the only practical circuit solution is to leave the continuous-time domain and instead enter the discrete-time domain, with the use of switched capacitor (S-C) circuits. The result is accurate (for analog) computation with modest power consumption at the expense of die area, which tends to be high due to the use of large numbers of capacitors.

We did, in fact, resort to switched capacitor architectures for all of our wavelet filter functions. However, it was not necessary to stop optimizing at the architectural (circuit) level. On the algorithmic level, we were further able to make better bandpass filters using a method called *complex demodulation* [9]. Demodulation is a well-established method used widely in modems and radios, based on the principle that the frequency content of a signal can be shifted up or down by multiplying it by a pure sinusoidal "carrier" signal and then filtering appropriately. In those applications, the signal to be broadcast is first modulated on a carrier signal to push it into a high-frequency broadcast band, and then demodulated by the receiver to retrieve the original signal. The purpose of modulating is twofold: First, the high-frequency signal can be broadcast much further without significant attenuation, and second, a large number of signals can be transmitted at the same time by assigning each one a non-overlapping portion of the electromagnetic spectrum.

The method just described is also a well-established method, used in subband coders and to make "lock-in" amplifiers able to analyze miniscule frequency bands for signal and noise analysis.

There is one small mathematical catch to this method. When a signal is demodulated to zero frequency, parts of the signal extend into the negative frequency domain, which in terms of the actual measured signal means that these frequencies are folded back (aliased) into the positive frequency domain. There they would be inseparable except for the "complex" part of complex demodulation. The method requires two carrier signals, one of which is offset from the other by 90 degrees phase. Thus if one carrier describes sin(ωt), then the other describes cos(ωt). Two demodulations are performed in parallel, each with one of the two carriers. The two resulting signals both have frequencies aliased into apparent mush, but between the two signals is all the information necessary to separate out the aliased parts after subsequent modulation. In fact, it can be shown that this requires nothing more than separately modulating the two results, again with the modulation carrier signals offset in phase, and adding the two modulation results together. No filtering is necessary for the modulation step, which is another consequence of the math.

Complex Demodulation and Reconstruction of a signal by a carrier

Returning to our previous example: An input time-varying signal $x(t)$ is multiplied by two sinusoidal reference signals $s_1(t) = \cos(\omega_0 t)$ and $s_2(t) = \sin(\omega_0 t)$. Choosing for $x(t)$ the simple form of a sinusoid of arbitrary phase ϕ:

$$x(t)s_1(t) = \cos(\omega t + \phi)\cos(\omega_0 t) \tag{2.13}$$

$$x(t)s_2(t) = \cos(\omega t + \phi)\sin(\omega_0 t) \tag{2.14}$$

Applying the following trigonometric identities—

$$\cos(u+v) + \cos(u-v) = 2\cos(u)\cos(v) \tag{2.15}$$

$$\sin(u+v) + \sin(u-v) = 2\sin(u)\cos(v) \tag{2.16}$$

yields the following expressions:

$$x(t)s_1(t) = \frac{1}{2}\left(\cos((\omega_0+\omega)t+\phi) + \cos((\omega_0-\omega)t-\phi)\right) \tag{2.17}$$

$$x(t)s_2(t) = \frac{1}{2}\left(\sin((\omega_0+\omega)t+\phi) + \sin((\omega_0-\omega)t-\phi)\right). \tag{2.18}$$

Now perform a lowpass filter by assuming an ideal filtering function, $h(t)$, having a Fourier transform $H(\omega)$, which perfectly rejects the sum of frequencies while perfectly passing the difference of frequencies (see Figure 2.4):

$$h(t) * x(t)s_1(t) = \frac{1}{2}\cos((\omega_0-\omega)t-\phi) \tag{2.19}$$

$$h(t) * x(t)s_2(t) = \frac{1}{2}\sin((\omega_0-\omega)t-\phi). \tag{2.20}$$

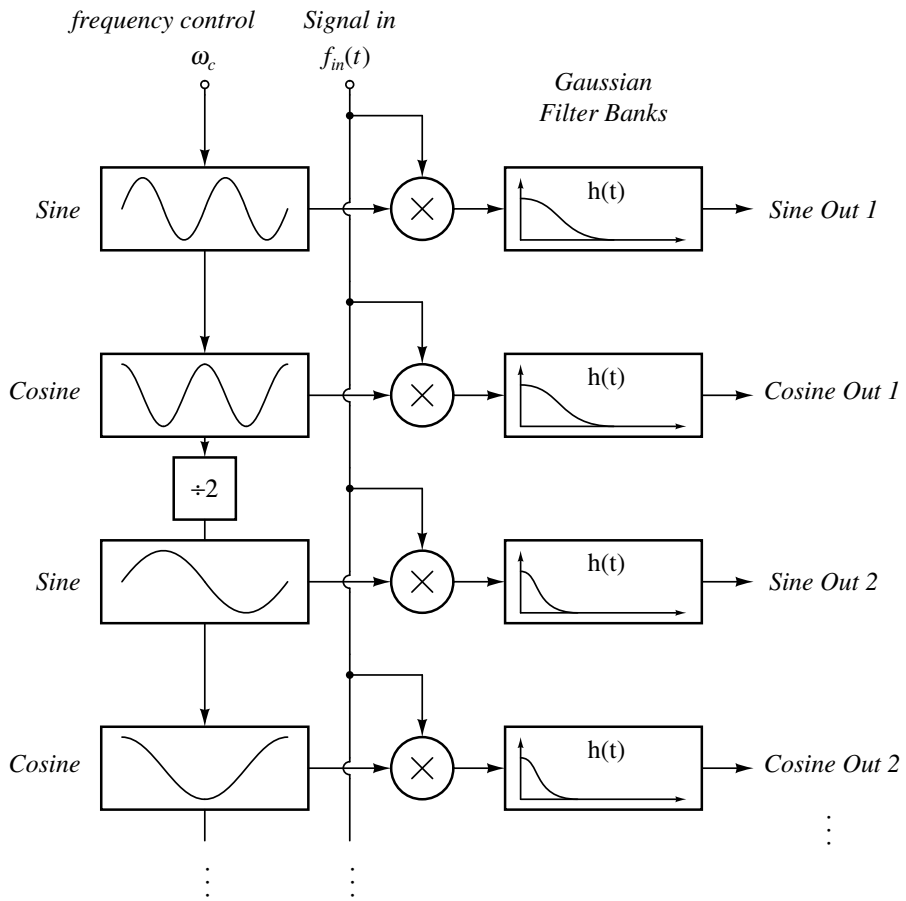

Figure 2.5: Complex demodulation (2 channels shown).

The input function is designated by $f_{\text{in}}(t)$. In order to demodulate it with respect to some given frequency ω_c (one of the center frequencies of the N channels of the wavelet decomposition), we use the following multiplication:

$$f_{\text{out}}(t) = 2h(t) * (f_{\text{in}}(t)(\cos\omega_c t + j\sin\omega_c t)) \tag{2.28}$$

where the lowpass filter time-domain transfer function $h(t)$ is shown without reference to a specific equation and is assumed for the sake of argument to be a filter which passes all frequencies below its cutoff unattenuated, and blocks all frequencies above its cutoff. Since the lowpass filter is assumed to be real, and the input $f_{\text{in}}(t)$ is real, then the output $f_{\text{out}}(t)$ of Equation (2.28) must necessarily be complex-valued, and can be represented by separating it into real and imaginary parts:

$$f_{\text{out}}(t) \equiv f_{\text{real}}(t) + jf_{\text{imag}}(t). \tag{2.29}$$

Therefore,

$$f_{\text{real}}(t) = 2h(t) * (f_{\text{in}}(t)\cos\omega_c t) \tag{2.30}$$

and

$$f_{\text{imag}}(t) = 2h(t) * (f_{\text{in}}(t)\sin\omega_c t). \tag{2.31}$$

Note that Equations (2.30) and (2.31) are themselves both real-valued. Both results are obtained easily by multiplying the input by two sinusoids which are 90° out of phase with each other.

Each part (sine and cosine) of the demodulation process produces a new signal which contains the sum and the difference of the original and "carrier" frequencies. Since we are demodulating the carrier frequency down to zero, we are interested only in the difference, so we use the lowpass filter $h(t)$ to get rid of the part containing the sum of the two signal frequencies. From the remaining difference, one cannot separate signals on one side of ω_c from those on the other since ω_c is now at zero and negative frequencies have no physical meaning. In the real-valued signal, negative frequencies are flipped over the frequency axis and alias into the positive frequency spectrum. However, the information necessary to separate the positive from the negative frequency components is preserved in the phase relationship between $f_{\text{real}}(t)$ and $f_{\text{imag}}(t)$, as shown by the exact reconstruction below.

In order to remodulate the signal back to its original frequency, we perform the following multiplication:

$$\hat{f}_{\text{in}}(t) = f_{\text{out}}(t)(\cos\omega_c t - j\sin\omega_c t). \tag{2.32}$$

quantizes the time domain. A true wavelet transform needs to quantize both. A bandpass filterbank whose outputs are not sampled is not, strictly speaking, a wavelet transformer.

Since the bandwidth of each channel is proportional to the center frequency as required in a constant-Q system, and the contents of each band have been shifted down in frequency until the band is centered around zero frequency, the most efficient description in terms of the time-frequency uncertainty relation in a sampled-data representation requires that each frequency band be sampled at a rate proportional to its bandwidth. The result of the sampling is that each rectangle in the time-frequency plane shown in Figure 2.1 has an equal area representing the effective time and frequency bandwidths of the Gaussian filter (with some overlap). The overlap of filter functions is depicted in Figure 2.2. If the filter channels are sampled in the binary-tree fashion shown when the channels are centered on a $\log_2$ scale, the samples can be easily time-division multiplexed into a single output stream [17].

2.7 An Analog CWT Processor

The first attempt to build a Continuous Wavelet Transforming processor consisted primarily of a continuous-time, subthreshold analog design fabricated in a standard CMOS process. The analog circuits were based on the analog VLSI techniques described by Carver Mead in *Analog VLSI and Neural Systems* [1]. The underlying idea was to generate a set of exponentially-spaced clock (square) waveforms which would then be shaped (via filtering) into sine and cosine pairs, and multiplied directly with the continuous-time, continuous-valued input using a translinear (analog) multiplier.

This chip was designed in subthreshold analog as an alternative to using a computer or DSP system to perform the same transform. The advantages of this approach are reduced size and power consumption. The resulting implementation is inflexible in terms of ability to reprogram the type or order of the wavelet function, and requires dealing with the problems of temperature sensitivity, nonlinearity of analog computation, variable process parameters, and noise injection throughout the circuit (particularly that caused by the digital circuits generating the square wave). As will be seen presently, not all of these problems could be overcome sufficiently, resulting in a move toward a more mixed-mode architecture (Chapter 2 Section 2.15).

2.9 Analog multiplication

Purely analog multiplication is very problematic due to the limited range of linearity in such circuits, especially when the circuits are realized in subthreshold CMOS technology. A standard circuit for analog multiplication of voltages is the Gilbert multiplier (Figure 2.8). The Gilbert multiplier is based on the translinear principle and in terms of linearity and noise performance is best implemented using bipolar transistors rather than MOSFETs for the critical components of the translinear loop (for a full discussion of translinear circuits and the tradeoffs between MOSFET and bipolar devices, see Chapter 3).

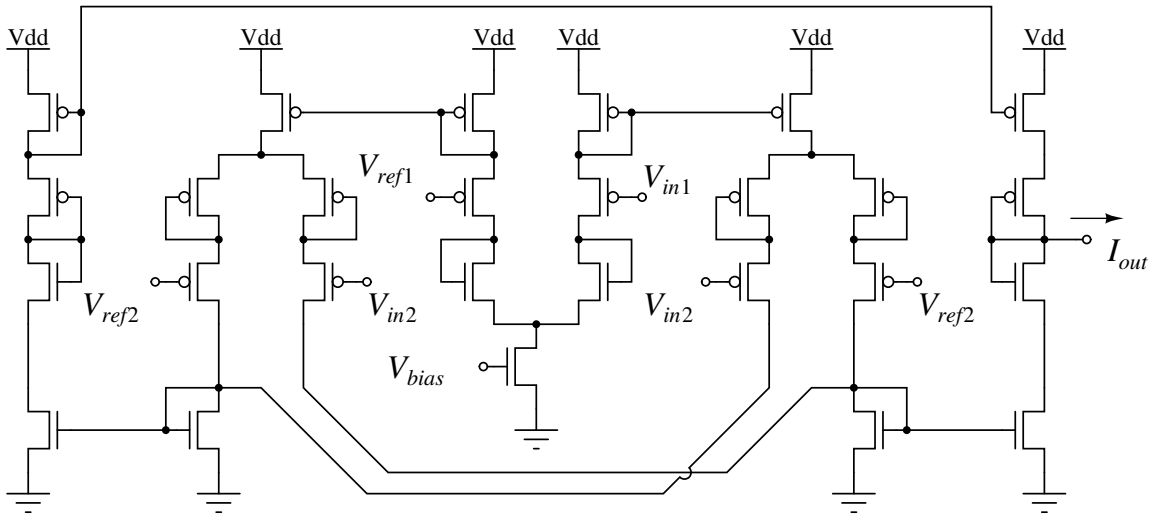

Figure 2.8: Gilbert multiplier with cascodes.

The Gilbert multiplier circuit executes a nonlinear function which, for input signals close to the reference, can be described by the linear approximation $I_{out} \propto (V_{in1} - V_{ref1}) \cdot (V_{in2} - V_{ref2})$. As with most subthreshold MOS translinear circuits, the input voltage swing is limited to a few kT/q, or about 50 mV. and is difficult to extend by more than a factor of two or so through the use of increasingly complicated linearization techniques [5]. A simple derivation of this limiting voltage can be found in Appendix A.

2.10 Wavelet Gaussian Function

In addition to allowing simple sampling of each output channel, the use of complex demodulation simplifies the process of designing bandpass filters that maintain a Gaussian shape while

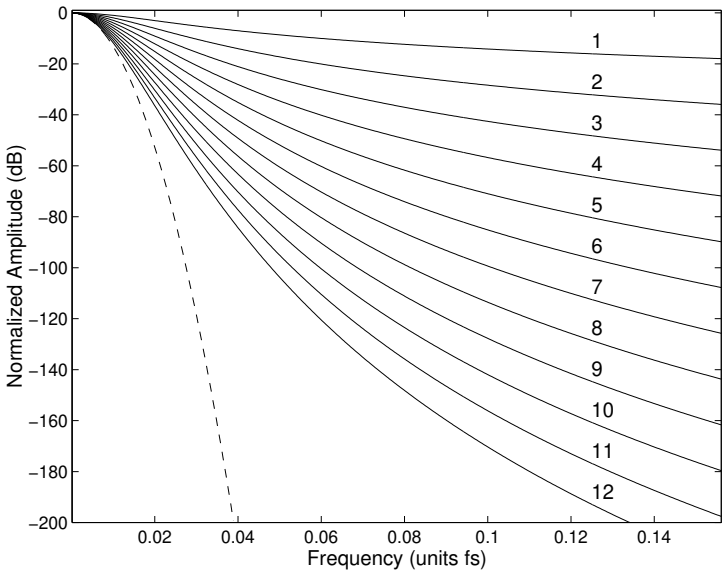

Figure 2.9: Frequency-domain transfer function of the final output of n cascaded lowpass filters as a function of the number of stages n.

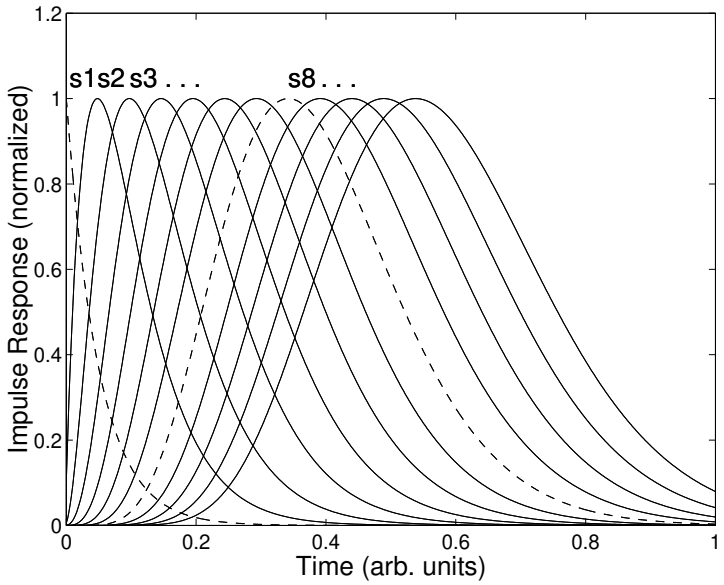

Figure 2.10: Impulse response of the final output of n cascaded lowpass filters as a function of the number of stages n.

2.11 Wavelet chip slice

The parallel nature of the analog wavelet computation allows the VLSI layout to be generated easily by abutting slices of circuitry. The filtering functions are identical for the sine and cosine parts of the transform, so slices are logically grouped in pairs, with each slice containing one transparent latch such that the pair of slices forms the flip-flop for dividing down the clock input to that section. Thus the path of the frequency stepping runs vertically across all channels while the signal path runs from left to right. In addition to generating the divide-by-two oscillator, each slice is responsible for shaping (filtering) the oscillator signal to produce a smooth sine (or cosine) wave, multiplying this modulating signal with the input, and filtering the result through a half-Gaussian-shaped lowpass function. Finally, depending on whether the section is configured for decomposition or reconstruction, the circuit samples the filter outputs and multiplexes them into a stream (decomposition), or else aggregates all the outputs together to produce the final result (reconstruction).

A block diagram of decomposition and reconstruction for a wavelet transform processor sine/cosine pair (single channel, or "slice") is shown in Figures 2.12 and 2.13, respectively. The entire analog Wavelet Transform chip is shown in Fig. 2.14.

2.12 Chip Specifications

- Power Supply: +5 V DC $\pm$5%
- Input mean value: 2.5 V $\pm$0.5 V
- Input p-p amplitude: 0.4 V
- Input frequency range: 80 Hz to 10 kHz
- Number of output channels: 12 (6 pairs)
- Silicon area: 1.96 $\times 10^6$ μm^2 (in a 2.0 micron CMOS process)
- Chip package: 68-pin PLCC

The technologies used to fabricate the several versions of the analog Continuous Wavelet Transform processor are as follows:

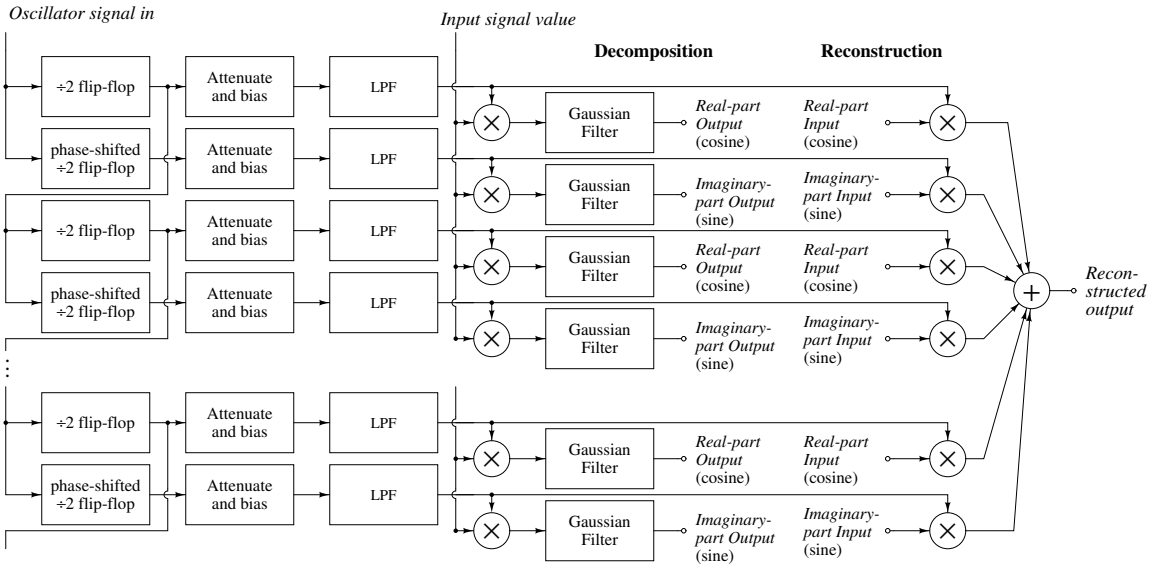

Figure 2.14: Wavelet Transform Chip block diagram.

2.13 Limitations of the Architecture

The major drawback of this analog architecture involves the use of filtered square wave signals as the modulating sinusoids. The use of a square wave as a modulation carrier is well-documented (see, for example, Horowitz & Hill, 2nd ed., p. 437) [28]. However, its use is restricted to modulation in the usual sense wherein a low-frequency signal is pushed up into a high-frequency band. The Fourier series describing a square wave contains an infinite series of odd harmonics beginning with the fundamental, which attenuates only as $1/n$, where n is the harmonic number (1, 3, 5, ...). The modulation performed by multiplying the square wave by the input signal can be broken down by superposition into the multiplication of each of the square wave harmonics with the input. If the destination band (carrier frequency) is large enough, then all of the intermodulation products (multiplication of the input by all harmonics greater than the fundamental) are very far out of band and can be easily attenuated with a simple filter. Unfortunately, if the function performed is demodulation instead of modulation, then by definition the carrier frequency is in the band of the signal and so any of its harmonics may be also. Consequently, intermodulation products will be in-band and cannot be filtered out. Shaping a square wave into a sinusoid by filtering out the higher harmonics is a difficult prospect at best involving complicated high-order elliptic filters; the filter cutoff must be prohibitively sharp to pass the fundamental frequency but attenuate the third harmonic to reasonable levels for clean signal processing (at least 40 to 50 dB for most audio

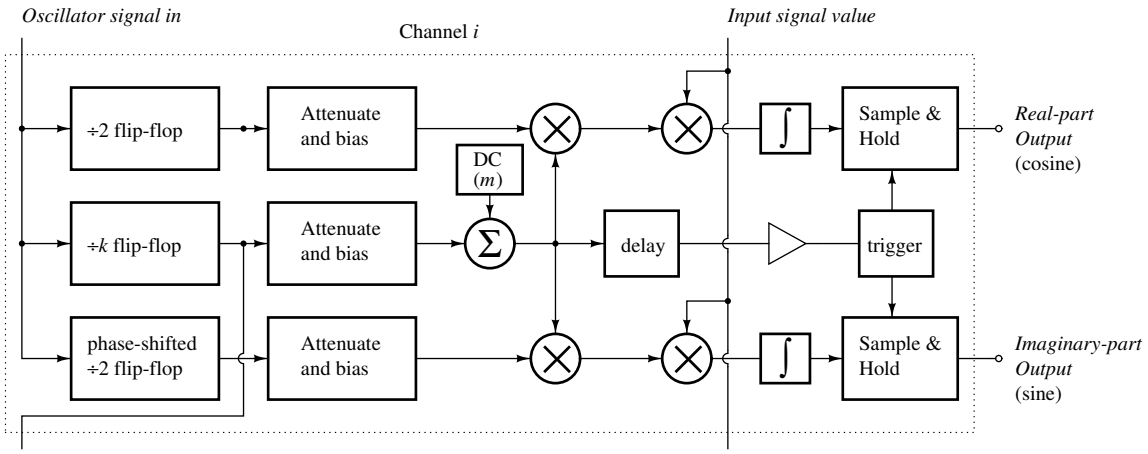

Figure 2.15: Architecture of Moreira-Tamayo *et al.*

The Moreira-Tamayo architecture quite nicely demonstrates the duality of the wavelet transform: Because it incorporates both the time domain and the frequency domain, efficient architectures can be realized in either one. The response of both systems to the same input is the same.

2.15 A Mixed-Mode Wavelet Processor

Several hardware implementations of the Continuous Wavelet Transform and related bandpass filter bank architectures have been reported in recent years [17, 19, 23], including the analog architecture discussed at the beginning of this chapter. The remaining sections of the chapter highlight a novel architecture we developed for the continuous wavelet transform processor which is unique in its encoding of the decomposition output and use of oversampling techniques.

The goal of the new architecture was to overcome the obvious drawbacks of the analog subthreshold MOS circuits, namely the quality of the carrier sinusoid signal and the linearity of the modulation multiplication. The new architecture relies more on digital processing and as such is more appropriately considered a "mixed-mode" design.

Other than the novel circuit methods used, the new design retains the essential characteristics of the original wavelet architecture. To recap: The processor performs a demodulation of the audio-frequency input signal in parallel across N channels, where the channels are adjacent with minimal overlap, cover the audio frequency band of interest, and are centered on a logarithmic scale. The process of demodulation shifts the signal frequencies from each channel center frequency

many different frequency axes. Each produces its own sum and difference components, and some are bound to end up in the signal band at the system output. In order to get predictable system performance, it is necessary to have a sinusoid carrier signal with a fixed and controllable amount of harmonic distortion. To this end we considered the possibility of generating an oversampled version of a sinusoid using a delta-sigma modulator.

The delta-sigma circuit of Lu *et al.* [26], an excellent example of the method, begins with a standard resonator system consisting of a simple loop of two integrators. Implemented in the digital (z) domain, each integrator requires a delay and accumulate operator, and one multiplication by a constant coefficient. The delta-sigma method eliminates the multi-bit multiplications by inserting a delta-sigma modulator into the loop which renders part of the resonator circuit a single-bit value, where the single bit represents an oversampled sinusoidal oscillation. Figure 2.16 shows the architecture, where no actual multiplications are required: One multiplication is reduced to a multiplexing operation, and the other in the digital domain is a bit shift operation. The remainder of the system still requires precision digital delay and accumulate operations, as does the implementation of the second-order delta-sigma modulator.

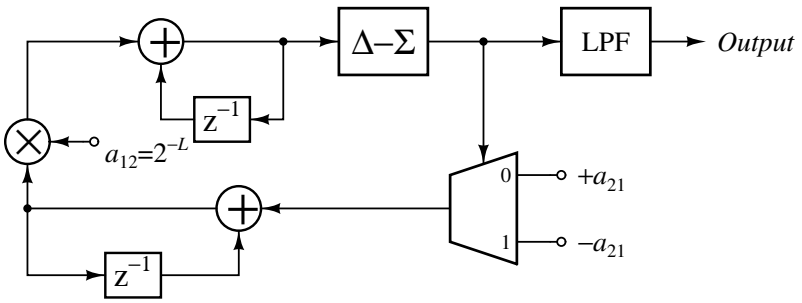

Figure 2.16: The analog oscillator architecture of Lu *et al.* [26] containing a 2nd-order delta-sigma modulator.

Eventually it became obvious to us that our system is much more specific than the general delta-sigma method described above. More to the point, the delta-sigma method is rather a bit of overkill for the purpose of generating a sine wave, particularly when only a finite set of frequencies is required rather than a full range of arbitrary values. We need to generate a number of sinusoids of known fixed frequency and amplitude. Presumably there exists a fixed sequence of bits which describes the optimal n-times oversampled sinusoid in the context of our system for a given value of n. If the optimal sequence can be determined, then it has a fixed amount of harmonic distortion which should decrease with increasing n as the oversampled sequence contains increasing amount

It should be clear at once that a full search of the n-bit space of the sequence is an "np-hard" problem, and furthermore that each search involves a large amount of computation for simulating the antialiasing filter and output filter, computing an FFT, and measuring harmonic distortion. For the purpose of finding the optimal sequence, we define harmonic distortion as the ratio of the amplitude of the largest distortion product to the amplitude of the fundamental frequency.

On the other hand, it should also be clear that there are a number of factors in our favor:

1. Inverted sequences have the same response as their non-inverted forms, and do not need to be investigated. This is also true of reversed sequences.

2. Sinusoids have quarter-wave symmetry, and therefore the sequences should also (symmetric signals can be closely approximated by non-symmetric sequences, as is generally the case in a 2nd-order delta-sigma system, but there is no reason to explore this space). This cuts the search space by a factor of four.

3. Only a small subset of 2^n sequences look anything like a sinusoid, allowing the possibility of heuristic approaches.

The first consideration involves generating a quarter sequence, and then generating the other three quarters by (in turn) reversing the sequence, inverting and reversing, and reversing once again. A reasonable heuristic approach is to start with a known sequence which looks something like a sinusoid, namely a square wave (for which the first quarter sequence is all '1' bits). From that point we need an iterative method which will explore the local space of nearby bit sequences ('nearby' meaning in a Hamming distance sense) which will (hopefully) converge quickly to a solution without encountering local minima along the way.

As is often true for these sorts of methods, we do not present (nor have we even attempted) a proof of convergence. Our method successfully found bit sequences which exceeded the system constraints and which met with our approval (*i.e.*, the sequence was short enough that it was conceivable to design a simple sequence generator to fit the wavelet system on a 2mm $\times$ 2mm layout).

2.16 Details of the Bit-Sequence-Finding Algorithm

It is possible to ascertain some properties of the solution before running the algorithm. For instance, the average of ones and zeros in the bit sequence must match the integration under the

where $g(t)$ the impulse response of the lowpass prefilter and $h(t)$ is the postmultiplication filter as before (Section 2.4). The binary sequence $s'(t)$ is assumed periodic with quarter-wave symmetry, and therefore can be described in the frequency domain by a Fourier series containing only harmonics of odd orders in ω_0:

$$S'(\omega) = \sum_{k=0}^{\infty} {S'}_k(\omega) \,, \tag{2.40}$$

$${S'}_k(\omega) = jc_k \left[\delta\left(\omega - (2k+1)\omega_0\right) - \delta\left(\omega + (2k+1)\omega_0\right)\right]. \tag{2.41}$$

The fundamental component ${S'}_0(\omega)$ corresponds to the desired sinusoidal signal $s(t)$.

Figure 2.17 shows how the two systems operate under the assumption that the lowpass filter functions $H(\omega)$ and $G(\omega)$ are ideal, *i.e.*, flat in the passband and with infinite rolloff at the cutoff frequency. Under this assumption, it can be seen from the figures that an arbitrary input spectrum $X(\omega)$ corresponding to the time-domain input $x(t)$ produces the same output $y(t)$ for both systems if and only if the prefilter bandwidth $\mathrm{BW}(G(\omega))$ is constrained by

$$\omega_0 + \mathrm{BW}(H(\omega)) < \mathrm{BW}(G(\omega)) < 3\omega_0 - \mathrm{BW}(H(\omega)) \,. \tag{2.42}$$

if the last inequality is not satisfied, convolution products of $X(\omega) * {S'}_k(\omega)$ for $k > 0$ will be aliased into the output.

In reality, the filters $H(\omega)$ and $G(\omega)$ have finite rolloffs, and the equivalence between the systems in Figure 2.17 is only approximate. The quality of the approximation depends on the harmonic coefficients c_k corresponding to the binary sequence, which can be optimized for minimum distortion. There is a trade-off between the complexity of the sequence and that of the filters G and H, as illustrated in Section 2.18.

2.17 Sequence generation

The output spectrum generated by the modulation scheme contains intermodulation products between the prefiltered input $G(\omega) \cdot X(\omega)$ and the harmonics of ${S'}_k(\omega)$, with terms of the form

$$jc_k \; G\left(\omega \pm (2k+1)\omega_0\right) \cdot X\left(\omega \pm (2k+1)\omega_0\right) \cdot H(\omega). \tag{2.43}$$

Only the fundamental term $k = 0$ is desired, and distortion arises from the higher-order intermodulation products, $k > 0$. To reduce distortion, the coefficients c_k need to be small for $k > 0$,

$s'(nT)$, for $n = 1 \ldots N$. With quarter-wave symmetry, only $N/4$ bits (one quadrant) need to be determined. Still, this problem has a combinatorial complexity, and becomes intractable for large N. We obtain approximate solutions using a technique of iterative block optimization, where a full search is repeatedly conducted over randomly selected blocks of consecutive bits in the sequence.

Appendix B lists a short Matlab program which performs the block-iterative sequence search. The procedure for determining the harmonic distortion is as follows: A quarter-wave bit sequence is expanded by inverting and reversal into a full wave and described in terms of values $+1$ and -1. An FFT is applied to this sequence and its magnitude computed. The resulting spectrum is multiplied by the frequency-domain transfer function of the lowpass filter (described as an attenuation coefficient per FFT bin). Then we can directly compute total harmonic distortion as the magnitude of the second FFT bin (the signal) divided by the sum of the magnitudes of all the other FFT bins (the distortion).

In lieu of computing every one of the 2^n possible sequences, we combine exhaustive search with a random perturbative method. The exhaustive search is performed over a tractable subspace of m bits, where $m < n$ (generally, $m < 16$ to compute in reasonable time), where the remaining $n - m$ bits are held fixed. On each iteration, the starting point of the subsequence to exhaustively search is chosen at random, the search space of 2^m sequences computed for a minimum, then the minimizing sequence is chosen as the new sequence and the process is repeated. There is no guarantee of success, and in practice the error surface is rife with local minima. Usually, however, the algorithm produces acceptable results, meaning total harmonic distortion of -60 to -70 dB, which suffices for most applications. The formulation of the algorithm leaves open the possibility of variants based on genetic algorithms: at every iteration of the algorithm, the best z solutions are kept rather than than the single best solution, allowing population statistics to determine the course of the optimization. By searching a broader solution space at each step, the system is less likely to become trapped in local minima, and the convergence time is significantly reduced.

2.18 Results and implementation

We demonstrate the principle with the following example and describe a simple and elegant implementation. The filter $G(\omega)$ is third order, implemented as a cascade of three single-pole filter stages, each pole located at $z = 15/16$. The total sequence length is $N = 256$. Figure 2.19 shows the 64 bits of the first quarter of the sequence obtained using the iterative block optimization method with block size 8. Figure 2.20 shows the FFT results for the filtered and unfiltered binary

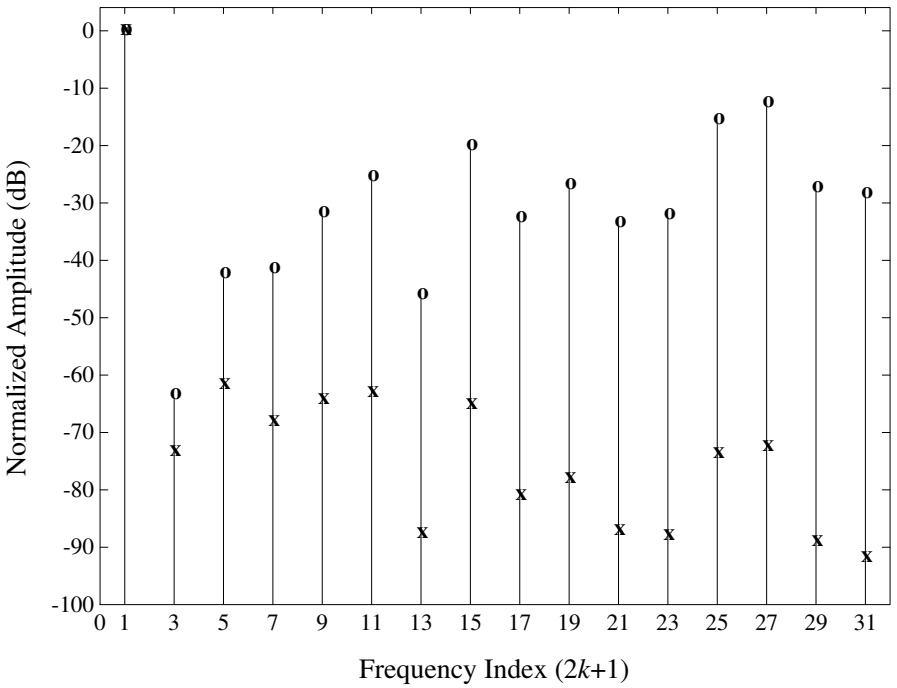

Figure 2.20: Frequency domain properties of the raw (**o**) and filtered (**x**) bit sequences.

sate for the sizable harmonics in the square wave. Such a filter would be more complicated to build than the simple cascade of single-pole filters in the above example, and would be more sensitive to mismatches in the implementation. On the other hand, the oversampled sequence is fairly easy to generate, as outlined below.

The first-quadrant sequence of Figure 2.19 consists of 11 zeros and 53 ones, as expected from the integral calculation in Equation 2.38. The asymmetry allows a simple implementation using a sparse address decoder. A binary counter counts from 0 to $r - 1$, where r is the length of one quarter of the full binary sequence. The address decoder, a wired-'or' implementation of nMOS transistors and a pMOS pullup device with a small layout footprint, generates the 11 'zero' bits at the proper points of the count. The inversion and reversing operations needed to obtain the rest of the full sequence can be elegantly realized by using a gray-code count rather than a binary count. In an n-bit gray code, as illustrated in Figure 2.21, the lower $n - 2$ bits describe a sequence which counts out forwards and then backwards. The inversion of the sequence is determined through an exclusive-or operation of the sequence bit with the n-th bit of the gray-code counter. It is also quite straightforward to generate the addresses of a sequence 90° out of phase with the original, for

arbitrary continuous-valued input with a *known* periodic function (the sine wave), we are able to use the oversampled sine sequence directly to achieve a highly linear analog multiplier.

Simple but practical modulation schemes make use of multiplication of a continuous-valued signal and a binary-valued signal. An exact multiplication of an arbitrary input by a binary-valued (± 1) function can be realized as a multiplexer which is controlled by the function and alternates between the input and its inverse. This is shown in Figure 2.22. In the figure, the multiplexer is

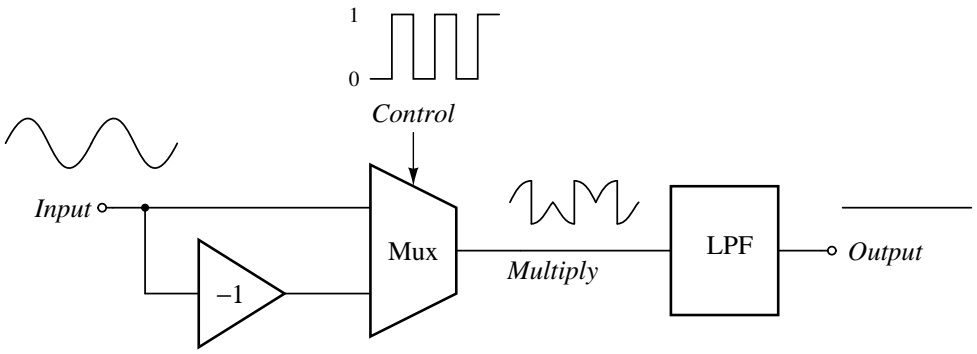

Figure 2.22: Multiplexing *vs.* Multiplying.

controlled by a square wave [28]. The square wave is used as an approximation to a sine wave with all harmonics other than the fundamental considered to be "error" terms. As mentioned in the previous section, this works as long as the intermodulation products of the input signal and these "error" harmonics fall well outside of the passband of the final result after the usual lowpass filtering.

In the same manner as the square wave example, any arbitrary binary sequence can be used as a control, insofar as any unwanted intermodulation products fall outside the passband of the postmultiplication filter. Since the purpose of oversampled representations of waveforms is to push unwanted harmonics as far away as possible from the harmonics of the desired waveform, binary-valued oversampled representations work extremely well in place of the square wave in Figure 2.22. It is assumed that the oversampling noise is negligible in the frequency band of interest, and can therefore be filtered from the multiplication output.

2.20 Switch-Cap Wavelet Gaussian Function

The circuit which approximates a Gaussian filter is based on the same architecture as described in Section 2.10 (shown in Figure 2.11), which is in turn based on the Central Limit Theorem of statistics: A half-Gaussian profile is produced by an infinite cascade of lowpass filters.

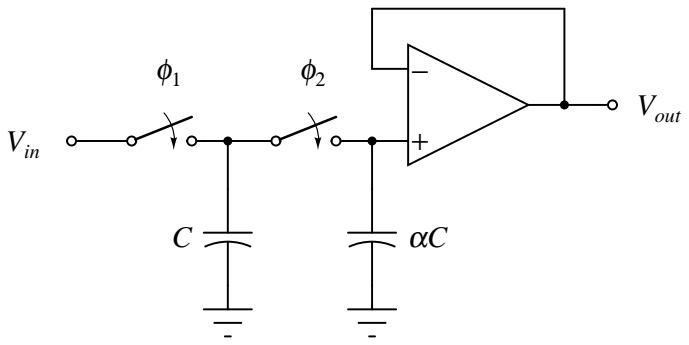

Figure 2.23: Single discrete-time lowpass filter section.

sections due to a primary constraint of accuracy. For eight stages, the resulting transfer function matches a true Gaussian to about −60 dB.

The center frequencies of the effective bandpass functions are determined by the frequency of the modulator sine waves (described below), which are spaced on a $\log_2$ scale. The bandwidth of each wavelet filter is determined by the cutoff frequency of the switched capacitor lowpass filter, and therefore is proportional to the period of its driving two-phase clock. The frequency of this clock signal, like the clock which generates the oversampled carrier signal, is spaced on a $\log_2$ scale. The fixed relationship between the carrier and the lowpass cutoff gives the wavelet filterbank a constant-Q characteristic. The clock frequency can be derived from the clock driving the modulator sequence generator (in our implementation, this is accomplished by dividing down the master clock off-chip). The bandwidths typically are set so that adjacent channels overlap at the half-magnitude point of the Gaussian function.

2.21 Output Time Multiplexing

Outputs of the Gaussian filter sections should be sampled in a manner which generates the time-frequency distribution shown in Figure 2.1. The binary-tree time multiplexing shown is quite easy to generate from a Gray code. As shown in Figure 2.24, control signals for taking the output from each wavelet channel can be achieved by a counter which uses two-phase clocking and generates Gray code as output. This control signal is the trigger signal, ϕ_2, for each Gray code digit Q_i. It can be seen that if ϕ_{21} controls sampling of Channel 1, ϕ_{22} controls sampling of Channel 2, *etc*., then Channel 1 is sampled twice as often as Channel 2, which is sampled twice as often as Channel 3, and so forth, and none of the signals overlap. One time slot per sequence of outputs is

sixteen channel system, each channel has a center frequency value which is $\sqrt{2}$ of the neighboring channel, with the bandwidth of each channel narrowed by a factor of two to account for the fact that there are twice as many channels squeezed into the same total frequency span. The value of $\sqrt{2} = 1.41421\ldots$ is approximated reasonably well by the integer fraction $7/5 = 1.4$, which is only about 1.0% low. The sixteen-channel system is enabled by starting with a master clock which is first divided down by five and by seven, each result becoming the master clock for one eight-channel section. The outputs of each eight-channel section are interleaved, and the sampling scheme of Figure 2.1 must be modified so that the timesteps are halved so as to allow both sections to be sampled. The sixteen-channel architecture is shown in Figure 2.25. Wavelet decompositions of any multiple of eight channels can be realized in a similar manner.

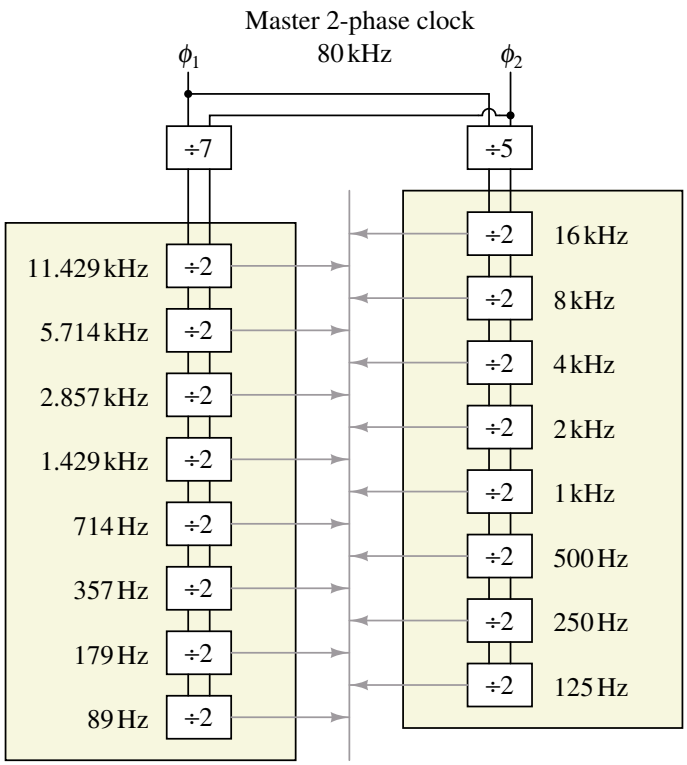

Figure 2.25: Sixteen-channel architecture using the 7-to-5 frequency ratio.

Decomposition and reconstruction functions are similar and therefore are able to share the same multiplier and filter circuitry, and the chip can be configured for either function. Reconstruction requires sample and hold circuitry on the front end to demultiplex the sine and cosine inputs, and a capacitive adder at the output. During reconstruction, the Gaussian filters do not shape the

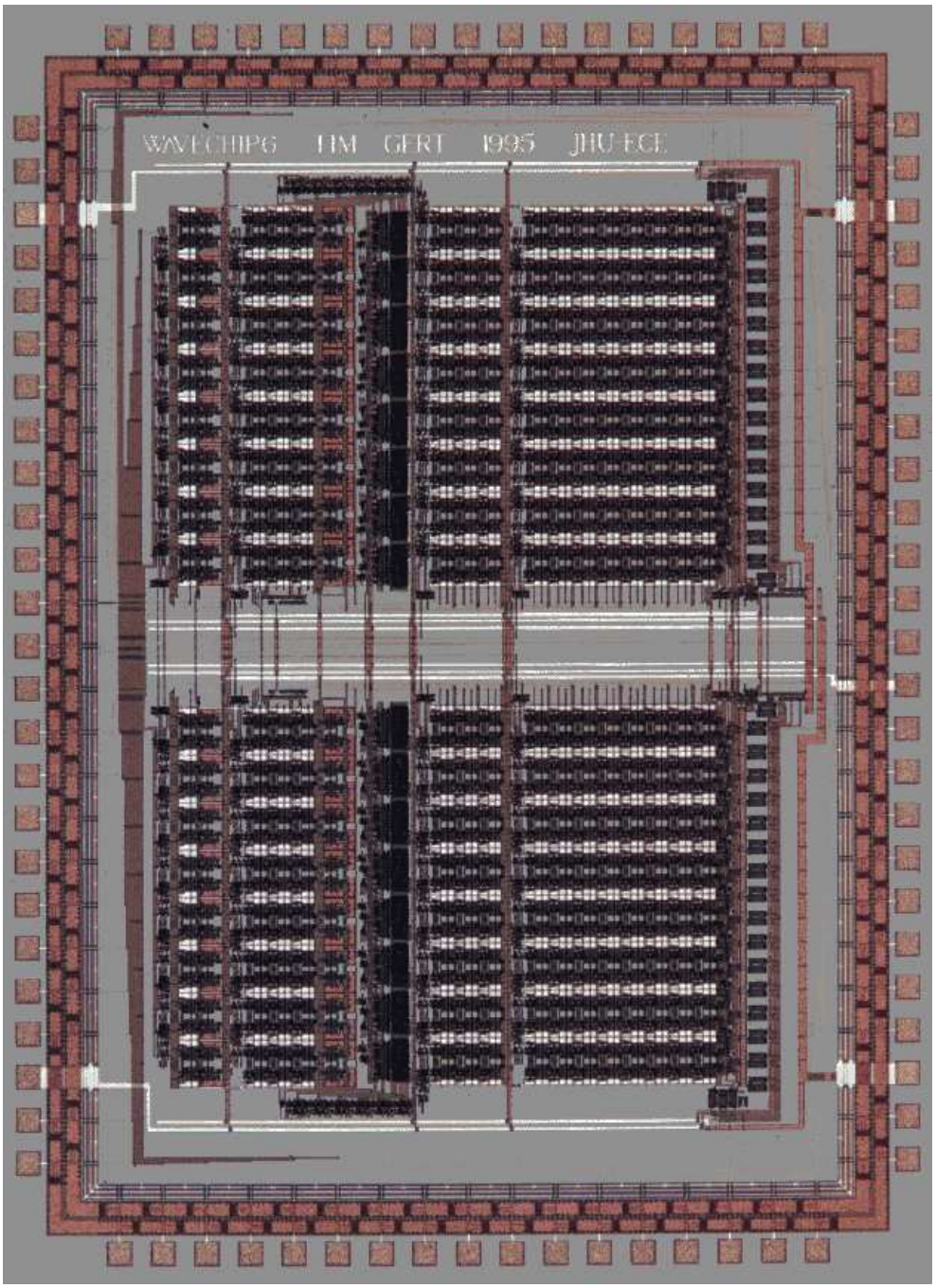

Figure 2.27: Photomicrograph of the mixed-mode continuous wavelet transform processor, a 4 mm×6 mm die size fabricated in a 2 µm CMOS p-well process.

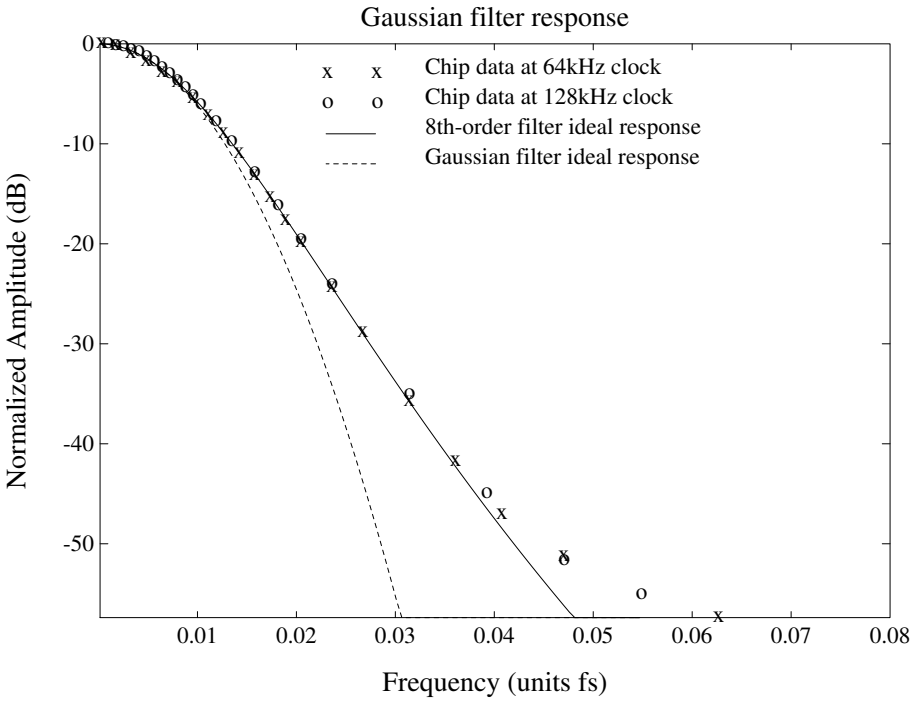

Figure 2.29: Gaussian filter magnitude response: predicted and measured.

The mismatch between channels of a sixteen-channel wavelet decomposition (the setup of which is described in the previous section) was tested by measuring the peak-to-peak magnitude of the output of each channel *vs.* the system input frequency. This measurement clearly shows the bandpass nature of the wavelet channels. The response of the first eight channels of a sixteen-channel wavelet output (channel frequencies spaced on a $\log\sqrt{2}$ scale) to a single sine wave input of variable frequency is plotted in Figure 2.31, along with the theoretical response (solid lines) in which the Gaussian function is approximated by cascaded lowpass filters as it is on the chip. The center frequency of each channel is exact, as it must be by design; the mismatch between filter bandwidths is negligible, and the mismatch in amplitude is acceptable.

2.24 Extensions of the Research

It is possible to apply the same complex demodulation scheme in a system designed to create an *arbitrary* mapping of the time-frequency plane, which unlike those depicted in Figure 1.1, has no regular structure, but a structure determined by the instantaneous information content of the

input signal itself. One such possible instantaneously computed map is depicted in Figure 2.32. The purpose of such a mapping is to optimally isolate the information content of the input signal in order to minimize the amount of bandwidth at the output. For example, if the input signal were a pure tone, then the output would be restricted to a single filter of minimum bandwidth and maximum timespan, sampled appropriately. Although algorithms to compute the optimal arbitrary coverage of the time-frequency plane are difficult to derive (formulas in [15] are nonintuitive and not recognizably amenable to hardware implementation), a system could potentially be designed based on heuristic methods.

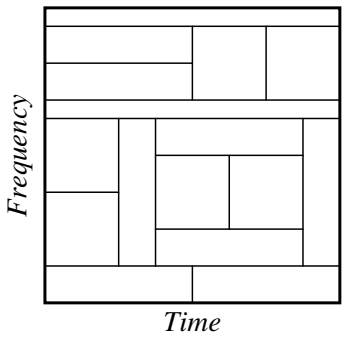

Figure 2.32: An arbitrary tiling of the time-frequency plane.

Another possible extension of the work consists of exploring algorithms which can make use of the demodulated outputs for acoustic pattern (speech/sound) recognition applications. One benefit of the wavelet processor *vs*. a standard filterbank approach normally used for speech recognition applications is that the wavelet output preserves phase information in the orthogonal sine and cosine outputs. Phase information does play an important role in some speech events and many other kinds of acoustic events (such as music and sonar), and may be used to the advantage of recognition systems. Another benefit, when the wavelet sampling scheme is used, is that the communication bandwidth required between the filterbank and the recognition system is greatly reduced, unlike standard filterbanks in which all channels are sampled at the same rate or are not demodulated to reduce the total required bandwidth for sampling. In any situation in which the frontend processor cannot be close to the recognition system (or other backend processor), this method constitutes an increase in efficiency and reduction in power in signal transmission.

Chapter 3

Current-Mode Filterbank Frontend

3.1 Time-Frequency Representations using Filterbanks

There is a corollary to the observation at the conclusion of the previous chapter that our wavelet transform architecture is perhaps better utilized for things other than time-frequency decomposition: Not every time-frequency decomposition needs to be a wavelet transform. Of course, as the term "wavelet transform" or "wavelet decomposition" gets ever more loosely applied (not unlike the term "neural network"), almost any time-frequency decomposition can be described as a wavelet transform. This tends to blur the lines between what is a wavelet filter, a cochlear filter, or just a bandpass filterbank.

In my mind—and this is clearly a subjective interpretation—the three filterbank types mentioned above *should* be more clearly defined along the following lines:

- A *Bandpass Filterbank* is any architecture which splits its input into multiple, simultaneous outputs by passing the signal through a parallel set of bandpass channels, preferably with minimal overlap. The band centers are often spaced on a logarithmic scale, although that is not a necessity.

- A *Cochlear Filterbank* is any architecture reasonably attempting to model the signal processing properties of the mammalian auditory system, at very least including a model of basilar membrane mechanics. This differs from a bandpass filterbank in that the input passes through a long cascade of lowpass filter sections, from which the output is tapped at various intervals. In keeping with the psychology of auditory perception and known facts about the biological structure of the cochlea and its neural connections, the center frequency of the (highpassed or

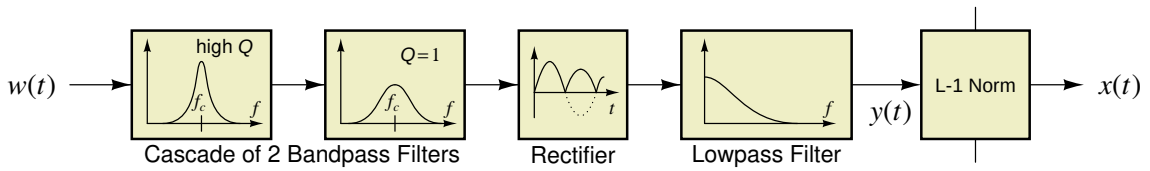

Figure 3.1: Frontend filterbank system—block diagram.

currents at the input. It appeared that it was going to be a difficult task to figure out how to couple a voltage-mode circuit like the HEEAR chip, or even the continuous wavelet transform processor, to the template correlator subsystem. Here is the fundamental problem: Currents are easily turned into voltages by feeding the current through a resistor to ground and buffering the resulting voltage across the resistor. Turning voltages into currents (that is, creating a linear voltage-controlled current source) is a more involved matter involving active feedback. Both methods require that the resistances required are compatible with the process: resistances larger than about 10 kΩ are inconvenient to realize in VLSI, while resistances larger than about 1 MΩ become incompatible with discrete component design.

Subthreshold circuits and, in general, current-mode circuits aimed at ultra-low power applications, typically have currents in the nanoamp range. The correlation processor to be described in detail in Section 4.4.1 and following is such a circuit, with the input currents topping out around 1 μA. Such a system should operate marginally well using an off-chip interface of discrete components between the frontend analog signal processing section and the template correlator. But since one of our goals is to realize single-chip solutions for large systems, we prefer to have the frontend interface directly to the correlator without the need to send signals off-chip, incurring the associated load capacitance, noise pickup, and pad-limited design. This being the goal, either we must devise a voltage-mode correlator or else devise a current-mode frontend.

3.3 Current-Mode Filters for Current-Mode Applications

From as far back as the first few years I began designing CMOS circuits operating in the subthreshold regime, I was aware of the strong duality between voltage-mode and current-mode circuit design. The idea of duality, however, cannot be too rigorously applied and is incomplete in a number of areas. Roughly speaking, voltage-mode circuits typically are based on transconductance amplifiers, whereas current-mode circuits typically are based on current conveyors. Because cur-

3.4 High-Level Simulations of the Filterbank Frontend

Preliminary research focused on simulating variations of the filterbank frontend using sampled recordings of acoustic transient events (see Chapter 4, Section 4.3.1). The purpose of the research was to confirm that the parallel, linear filterbank channels we anticipated using would be functionally equivalent to the cascaded, compressively-nonlinear HEEAR chip filters. Proving this in simulation essentially meant converting the continuous-time filter equations into the z-domain and using a computer to apply the filter equations to the original data presented to the HEEAR chip. Then both the simulated parallel filterbank outputs and the HEEAR chip outputs were used as inputs to a simulation of the acoustic transient correlation. Results, which can be found in Chapter 4, Section 4.3.5, showed that at least under the ideal conditions of simulation, the parallel filterbank frontend outputs yield similar results on the classification task, maintaining the classification rates of the HEEAR outputs and often doing better. The improved classification rates almost certainly are due in part to the difference between the noisy outputs of the HEEAR chip with all of its physical limitations versus the clean digital processing of the filterbank simulation.

The filterbank system described in the remainder of this chapter performs the steps shown in Figure 3.1. Input to the system is an acoustic signal. The simulation takes a sampled, digitized version of the acoustic signal. The HEEAR data acquisition system stored its outputs and a sampled version of the original input as 16 bit values sampled at 32 kHz. The input was bandlimited to under 16 kHz to prevent aliasing during sampling. Recording gain was not adjusted to maximize the resolution of the recording so that the output was effectively 8 bits resolution rather than 16. The fact that the HEEAR system copied its input to the output along with its own filtered outputs was useful for making sure that the simulated parallel filterbank system received the same inputs as the HEEAR system.

Figure 3.2 shows the raw sampled input signal for an example transient recording. This and the following figures are screen captures from the X11 interface to our simulation software.

The parallel filterbank frontend processes its inputs through N channels ($n = 1 \ldots N$), where each channel contains two cascaded second-order bandpass filters, as shown in Figure 3.1. The frequency-domain (Laplace domain) transfer function for each filter section is

$$H_n(s) = \frac{(1/Q)\tau_n s}{1 + (1/Q)\tau_n s + \tau_n^2 s^2} \tag{3.1}$$

where $\tau_n = 1/2\pi\omega_{c,n}$ and $\omega_{c,n}$ is the center frequency for channel n. Resonance value Q may be made a function of channel n; however, we enforced the restriction that this would be a *constant* Q

filterbank.

Simulation of the bandpass filter requires a transformation from the s-domain into the z-domain. A standard way to do this is to use the *bilinear transform*,

$$H(z) = H(s)\Big|_{s=2f_s\frac{(z-1)}{(z+1)}} \tag{3.2}$$

which is valid when the sampling frequency is much larger than the filter bandwidth. The result of applying the bilinear tranform to Equation (3.1) is:

$$H_n(z) = \frac{b_{0,n} + b_{2,n}z^{-2}}{1 - a_{1,n}z^{-1} - a_{2,n}z^{-2}} \tag{3.3}$$

where

$$b_{0,n} = \frac{\tau_n}{1 + (\tau_n/Q) + \tau_n^2} \tag{3.4}$$

$$b_{2,n} = -b_{0,n} \tag{3.5}$$

$$a_{1,n} = \frac{2(\tau_n^2 - 1)}{1 + (\tau_n/Q) + \tau_n^2} \tag{3.6}$$

$$a_{2,n} = \frac{-(1 - (\tau_n/Q) + \tau_n^2)}{1 + (\tau_n/Q) + \tau_n^2} \tag{3.7}$$

and

$$\tau_n = \frac{f_s}{\pi f_{c,n}}. \tag{3.8}$$

with f_s being the sampling frequency (32 kHz), and $f_{c,n}$ being the center frequency of channel n. Thus each channel has its unique transfer function.

Center frequency values are rather arbitrarily defined; they are usually defined recursively, using

$$f_{c,n} = \gamma f_{c,n-1} \qquad n = 2 \ldots N \tag{3.9}$$

$$f_{c,1} = f_1 \tag{3.10}$$

for a purely exponential scale, where γ is constant and f_1 is the lower frequency bound of the filterbank, or

$$f_{c,n} = \gamma_0 f_{c,n-1} + \gamma_1 \qquad n = 2 \ldots N \tag{3.11}$$

$$f_{c,1} = f_1 \tag{3.12}$$

for a "mel" scale, where both γ_0 and γ_1 are constant and f_1 is the lower frequency bound of the filterbank. The values γ should ensure that channels overlap in the frequency domain based on some criterion such as -3 dB bandwidth or half-bandwidth.

for which application of the bilinear transform yields

$$H_n(z) = \frac{d_n + d_n z^{-1}}{1 - c_n z^{-1}} \tag{3.15}$$

having coefficients

$$c_n = \frac{1}{1 + \alpha_n} \tag{3.16}$$

$$d_n = \frac{\alpha_n - 1}{\alpha_n + 1}. \tag{3.17}$$

The value α_n may be simply a constant value across all channels or it may vary across channels as a value proportional to the bandpass constant τ_n. It may be computed from the sampling rate f_s and the desired lowpass cutoff frequency $f_{3\mathrm{dB}}$ by the relationship:

$$\alpha_n = \frac{f_s}{\pi f_{3\mathrm{dB},n}}. \tag{3.18}$$

In cascade, the cumulative lowpass filter transfer function for M cascaded stages is

$$G_n(s) = (H_n(s))^M. \tag{3.19}$$

Since this is not a Gaussian filter, usually two or three stages will suffice for the smoothing operation. In simulation, we used three stages. The result of applying rectification and smoothing in simulation can be seen in Figure 3.4.

3.5 Introduction to Translinear Circuits and Log-domain Filtering

Electrical engineering has a long history of attempts to create linear functions from highly nonlinear components. This may be due largely to the fact that engineering began with the (nearly) perfectly linear passive devices: resistors, capacitors, and inductors. The fundamental theories of linear filter design were fleshed out well before the advent of transistors or, for that matter, vacuum tubes. When these "active" devices were invented, designers capitalized on certain obvious nonlinear properties such as transistor switching (for digital circuits) and diode rectification (for AC to DC conversion). For most analog design, and almost exclusively for analog filter design, the goal was to force linear behavior from these devices through the use of high-gain amplification and feedback. The fundamental circuit element of analog integrated filter design became the operational amplifier, made of dozens or more transistors.

Linearization techniques work well, but the nonlinearities complicate circuit analysis considerably. One of the first techniques learned in amplifier and filter design involves the notion of

"small signal" analysis *vs*. "large signal" analysis. Nonlinear circuits act linear around a certain DC operating point, but can achieve only a limited accuracy over a given range. As the signal applied to the circuit departs from the operating point, the circuit departs from linear behavior. Circuit analysis proceeds in two stages by necessity: The nonlinear circuit equations can be used to find the quiescent operating point of the circuit given no input. Unfortunately, the nonlinear circuit equations, although they do describe the complete behavior of the system, are intractable for nonzero input functions. The linear circuit equations by nature are tractable but only approximate the behavior of the system near its operating point. Fortunately the principle of superposition allows these two sets of equations to be solved independently and added together to get symbolic solutions such as transfer function equations.

Log-domain filters are a different beast entirely. They comprise a subclass of circuits having externally linear transfer functions but internally nonlinear components (occasionally referred to by the acronym "ELIN") [40, 36]. This property, of course, could be claimed in reference to any circuit built with operational amplifiers, where every transistor used to build the amplifier is a nonlinear component. However, operational amplifiers are so firmly ingrained into engineering design that they are preferentially regarded as irreducible, fundamental devices without regard to their internal complexities. This view, which hides the internal complexity of the device behind a relatively few parameters like gain, unity-gain bandwidth, slew rate, and output impedance, is quite sensible. It may even be that one day certain subcircuits of log-domain filters, in particular current conveyors, will be regarded in the same way with the same universal scale of acceptability. For now, though, mostly due to the small number of transistors required for simple log-domain filters, they are scrutinized at the transistor level where the nonlinearities cannot be ignored.

As the name implies, log-domain filters are specifically those circuits whose internal state is a logarithmic function of the input and output. The circuit design exploits this particular nonlinearity directly rather than attempting linearization around an operating point. Such manipulations can only be done for specific types of nonlinearity: One is the square law, described by the drain current through a MOS transistor operating in strong inversion relative to its gate-to-source voltage [37]; another is the exponent, described by the collector current into a bipolar junction transistor relative to its base-to-emitter voltage, or the drain current through a MOS transistor operating in weak inversion (subthreshold) relative to its gate-to-source voltage. However, within the range of operation for which the nonlinearity of interest dominates all other second-order corrections to the device behavior, *large-signal linearity* is possible in circuits made from these devices. When a circuit has large-scale linearity, its transfer function uniquely describes the overall behavior of

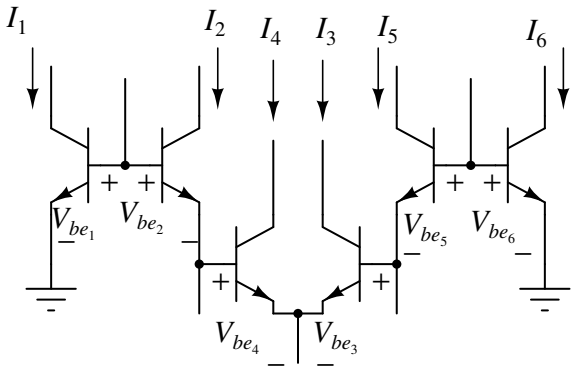

Figure 3.5: Translinear loop with common-base and common-emitter configurations.

however, the elements are nonlinear elements with exponential I-V relationships, namely diodes, bipolar transistors under normal biasing conditions, or MOS transistors in weak inversion. Each voltage drop around the loop is the voltage drop across the p-n junction of the diode or BJT or the voltage drop from the gate to the source in a MOSFET. By applying the I-V relationship of the nonlinear devices to the KVL equation, substituting the currents through the devices (which are not part of the KVL loop) for the voltage drop across the junction, it becomes immediately apparent that the exponential function allows very simple expressions for certain relationships between the currents.

Figure 3.5 shows a simple example of a translinear loop formed by the base-emitter junctions on bipolar transistors. For purposes of analysis and synthesis, we will consider all transistors (whether BJT or, equivalently, MOSFET operating in weak inversion[1]) to be ideal, perfectly matched devices implementing the simplified exponential function $I_c = I_s\, e^{V_{be}/V_t}$, or inversely, $V_{be} = V_t\, \ln\left(I_c/I_s\right)$, where V_{be} is the BJT base-to-emitter voltage, or the gate-to-source voltage of a corresponding subthreshold MOS transistor, V_t is the thermal voltage (0.025 V at room temperature) and I_s is the reverse saturation current of the transistor. Circuit modifications necessary to deal with the nonidealities of real devices will be addressed in Section 3.9. By Kirchoff's voltage law, the sum of the V_{be} voltages around the loop is zero:

$$\sum_{i\,\mathrm{odd}} V_{be_i} = \sum_{i\,\mathrm{even}} V_{be_i}. \tag{3.20}$$

[1] I do not intend to suggest that the physical mechanisms by which BJTs and MOSFETs in weak inversion operate are equivalent, which they are not [34], only that the I-V characteristic of both can be closely approximated by an exponential function which allows either transistor type to become the fundamental translinear circuit component for realizing equivalent (to first order) linear systems.

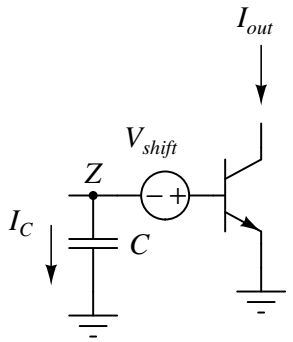

Figure 3.6: Filter pole formed using a transconductor.

Note that while neither equations (3.22) and (3.27) describe any circuit behavior that cannot be derived by directly solving the (simplified) I-V equations, they both can be used as methods to quickly analyze translinear loop circuits without explicitly referring to either exponent and logarithm functions or to device parameters (*e.g.*, I_s), and also to synthesize filters from building blocks based on simple translinear loops.

3.7 First-Order Circuit synthesis

To show how these circuit concepts can be used in practice to synthesize complete filter circuits, consider for instance a generic first-order system with the (current) transfer function:

$$\frac{I_{out}(s)}{I_{in}(s)} = \frac{1}{A + \tau s} \tag{3.28}$$

Equation (3.28) is not in a canonical form, but is presented here as a useful building block for generating higher-order circuits, as we will show in Section 3.8. Working in the time domain and substituting, from Equation (3.27), $\dot{I}_{out}$ for sI_{out}, gives:

$$AI_{out} + \left(\frac{\tau\, I_C}{V_t\, C}\right) I_{out} = I_{in}, \tag{3.29}$$

$$I_{out}\left(A + \frac{\tau\, I_C}{V_t\, C}\right) = I_{in}. \tag{3.30}$$

We then can define the time constant τ in terms of some (constant) bias current I_b:

$$\tau = \frac{V_t\, C}{I_b}. \tag{3.31}$$

This may seem like an arbitrary step, but the units of a time constant are of course equal to RC, and $R = V/I$, and V_t is the thermal voltage with units of volts ($kT/q \approx 0.025$ V), so this expression

As an approximation to a true exponential, this equation breaks down at currents in the range of I_s, which for most applications can be considered equal to zero compared to the nominal operating current through the device. Translinear devices cannot behave properly at all for negative input currents. An exception is the use of carefully constructed *class A-B* log-domain circuits, in which the input is centered around zero current, and two symmetric log-domain circuits handle the signal alternately in the positive and negative regions [40]. Class A-B circuits are particularly interesting for reasons of their ability to act as instantaneous companding circuits to reduce internally-generated noise as seen at the filter output. However, they are beyond the scope of this thesis, which is concerned mainly with efficient implementations and only covers single-ended, class A filters. Correct operation of a class A first-order section requires that the entire circuit be biased by adding a positive DC current to the input large enough to keep the input positive and therefore keep the transistor base-emitter (or gate-source) junctions forward biased at all times.

One noticeable consequence of class A operation is that while noise on the signal generated at or before the filter input is logarithmically compressed and expanded along with the signal, internally generated noise only experiences the expansion. Thus a portion of the noise seen at the filter output increases with the instantaneous signal level. Fortunately, there are few sources of noise internal to the log-domain circuit, and according to the equipartition theorem [6], they are all referred to the capacitor node. Thus the magnitude of the exponentially-rectified portion of the noise at the output can be calculated, and the capacitor size can be increased to make the value as small as required by the filter specifications.

3.8 Designing second-order sections

Construction of higher-order log-domain circuits from simple component cells has been described in various ways, such as the Bernoulli cell of [41] and the E+ and E− cells of [44]. We generate higher-order functions in a similar way, factoring the desired current transfer function into equations which can be directly implemented by the first-order subcircuit of Figure 3.7. Consider, for example, a second-order bandpass equation (Frey [39] uses a more general form of the same function for state-space synthesis of a second-order section):

$$I_{out}(s) = I_{in}(s)\frac{\tau s}{1 + (1/Q)\tau s + \tau^2 s^2} + I_{DC}(s). \tag{3.34}$$

This describes a bandpass function with a gain and resonance of Q, except that the output is biased to a positive DC operating point. Since the bandpass function itself eliminates DC components

however: The negative feedback term can be separated from the positive term, and with the proper transformation, it can be referred to the capacitor node as a dependent current source. This solution is shown in Figure 3.8B. To find the equation for this dependent current source, the translinear loop

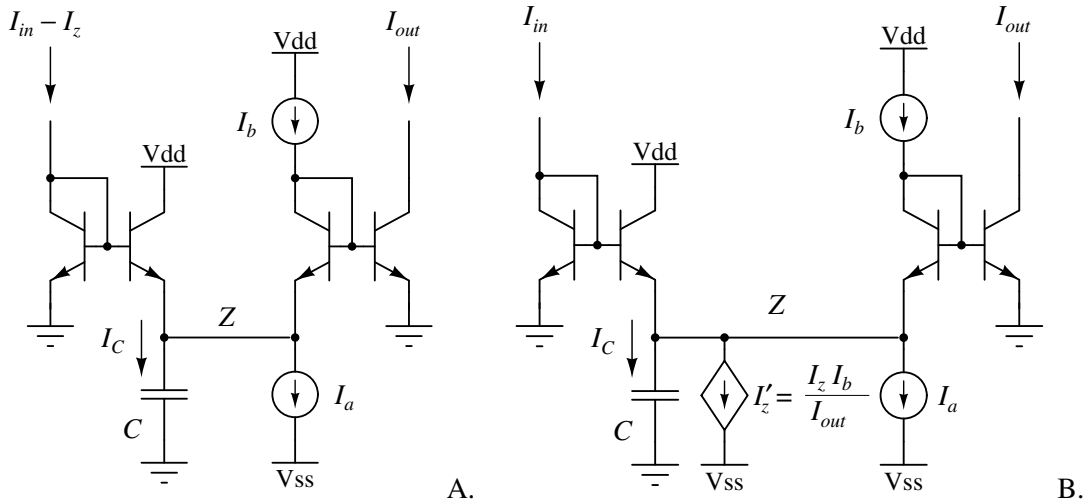

Figure 3.8: Computing a current difference at a log-domain filter input. A) The underlying idea, which is physically unrealizable. B) An equivalent working implementation.

equations for circuits A and B can be written, respectively:

$$(I_{in} - I_z)I_b = (I_a + I_C - I_b)I_{out}; \tag{3.39}$$

$$I_{in}I_b = (I'_z + I_a + I_C - I_b)I_{out}. \tag{3.40}$$

Solving for I'_z, we find that the dependent current source must have the value $I'_z = (I_z\, I_b)/I_{out}$. It should be immediately apparent that this familiar form can be conveniently implemented by a simple four-transistor translinear loop circuit.

Now the bandpass filter can be drawn as a cascade of two first-order circuits, with the output of the first section shared with the input of the second, a method which is described briefly in [39]. The connected sections are shown in Figure 3.9.

As a further simplification, by noting the translinear loop $Q2$-$Q3$-$Q4$-$Q5$ which gives the equation $I_x I_y = I_b I_{out}$, the dependent current sources I'_{DC} and I'_x can be written in terms of I_y rather than I_x, and the entire subcircuit which generates I_x (dotted box in Figure 3.9) can be eliminated, avoiding the need for one set of matched current sources.

It remains to generate the translinear loops implementing the dependent current sources. Generally, two different solutions are possible for any simple translinear loop of this type: A

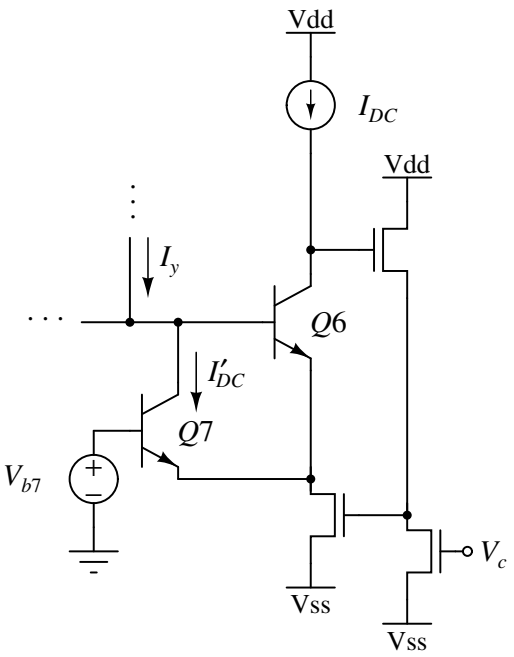

Figure 3.10: An alternative common-emitter circuit generating I'_{DC} (see text for discussion).

important to reduce the number of matched current sources in the filter circuit, which is the primary source of gain and offset errors and inter-channel mismatch in a filterbank. The common-base circuit also incurs a signal delay around the current mirror loop, and has been observed to latch up when the MOSFETs in the mirrors are made too large, increasing the signal delay to the point at which the circuit becomes unstable. A full analysis of this circuit's stability has not been attempted. On the right-hand side of the filter, use of a common-emitter circuit to replace the $Q8 - Q9$ common-base pair (Figure 3.10, with $V_{b7} = 0\,\mathrm{V}$) results in current draw through the base of $Q6$, turning the lossless integrator into a lossy integrator. At audio frequencies, where the bias current I_b can be smaller than the base current I_{DC}/β of 10 to 100 nA, the circuit will fail. Note, however, that I'_{DC} is proportional to $I_{DC} \cdot e^{V_{b7}/V_t}$. I_{DC} can be reduced to the order of magnitude of I_b if V_{b7} is raised to compensate. This will allow the common-emitter circuit to work, though with the drawback that the output bias level is no longer equal to I_{DC}, but is a derived function of I_{DC} and V_{b7}. The derived function can in turn be generated from the real values of I_{DC} and V_{b7} using yet more translinear circuits 3.12. The resulting circuit has a minimum number of matched currents throughout, and therefore potentially has the best behavior in terms of mismatch. We took this approach in our latest prototype of the bandpass filterbank. Unfortunately, this approach merely shifted the mismatch problem from one

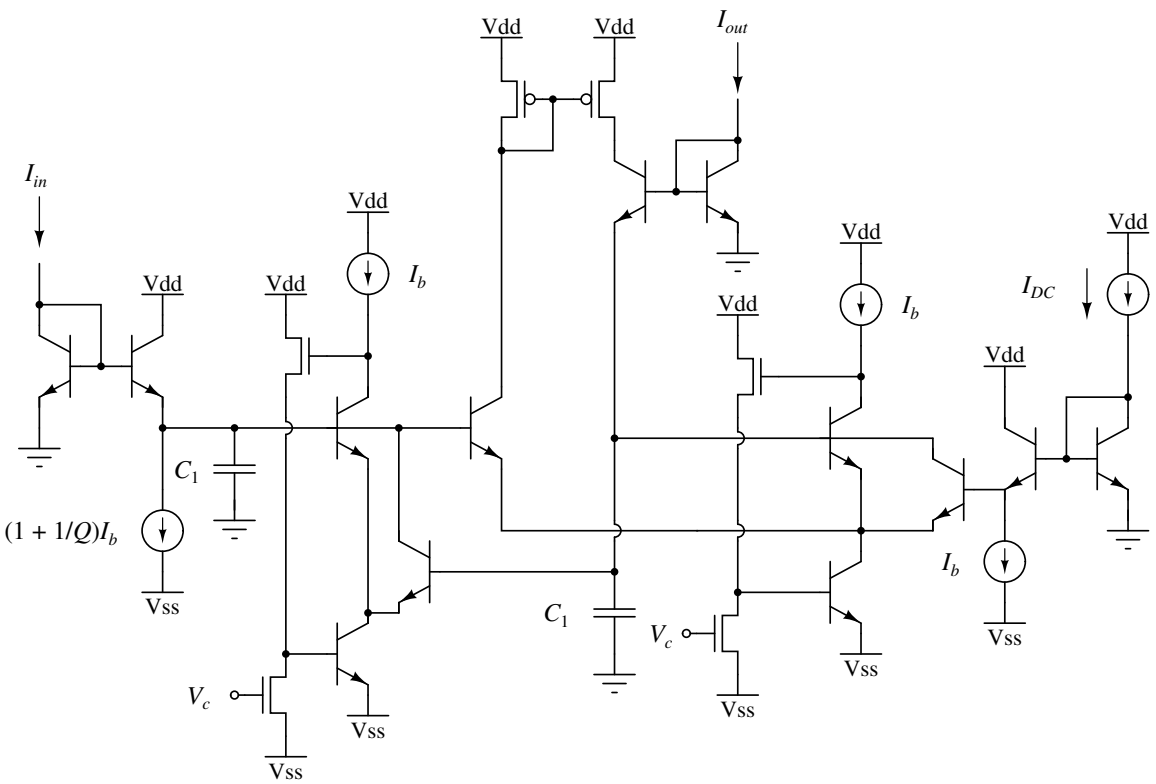

Figure 3.12: Another BiCMOS Log-domain bandpass filter, using the common-emitter configuration for all feedback circuits.

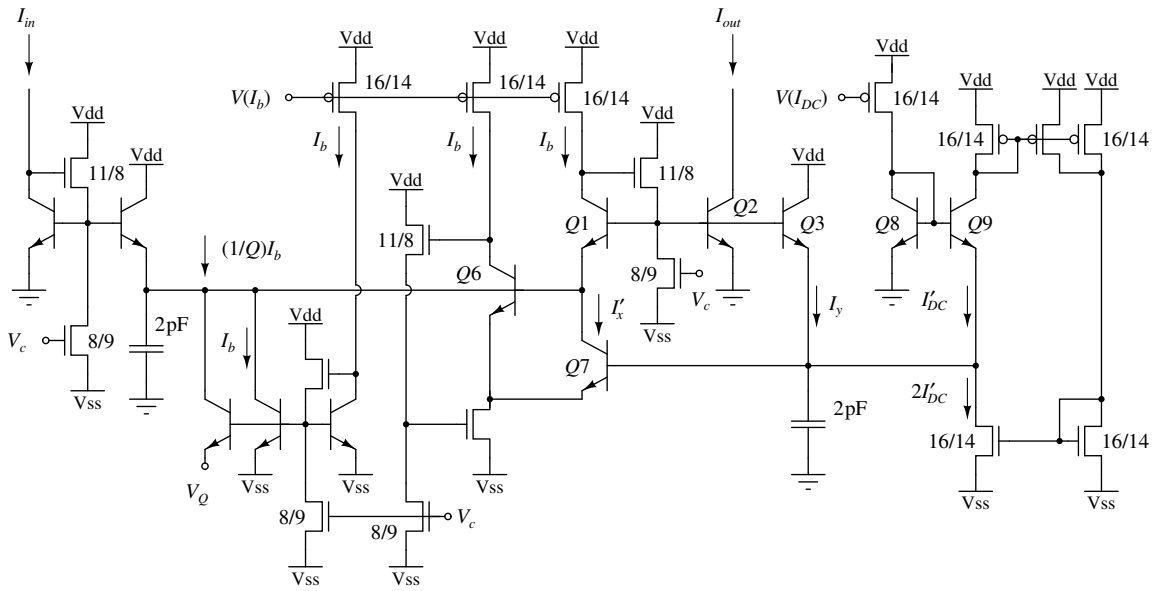

Figure 3.14: Complete circuit schematic for the second-order bandpass filter. Bipolar transistors are minimum size, and MOS dimensions are indicated as W/L in units of $\lambda = 0.6\,\mu$m.

base-current compensation circuit can be shared with the V_c bias needed for the common-emitter translinear circuit of Figure 3.10.

Figure 3.14 shows the complete bandpass filter, including all base compensation circuits and current sources, as it was fabricated and from which the measurements in Section 3.16 were taken. Note the circuit implementation of the current $(1 + 1/Q)I_b$: in this configuration, $Q = e^{(V_Q - \mathrm{V_{ss}})/V_t}$, making tuning of Q independent of I_b, the center frequency tuning control, and giving Q a natural range from 1 to the practical maximum allowed by the circuit.

Figure 3.15 shows the alternative bandpass circuit from Figure 3.12 with all the circuits used to compensate for nonidealities. The use of single MOSFET transistors as the current sources requires that cascode connections be added to compensate for the differing source voltages on otherwise matched transistor pairs.

3.10 Layout Considerations for VLSI Log-Domain Circuits

The log-domain filter circuits are so sensitive to mismatch error that it was deemed necessary to invent a regular layout structure to use throughout the chip. Conventional wisdom is that the most obvious measures to take against potentially bad matching of paired transistors are: use the same layout length and width for each device, keep the devices facing the same direction, keep the

devices close together, and use quadrature symmetry when practical. After considerable experience in chip design and testing, I have concluded that the greatest influence on transistor matching, after ensuring the same length and width of each device, is the effect of "same surround." This appears to be more critical even than having transistors facing the same direction (at least this is true for anti-symmetry; I have not looked at 90-degree rotations as the opportunity for this kind of layout while maintaining same surround is rare) and having transistors close together. Indeed, transistors which are displaced by half the length of a chip (say, 1 mm) but which are surrounded by exactly the same layout to a radius of about 50 to 100 μm in all directions tend to be much better matched than two transistors placed next to each other but with different circuits to the right and left of the pair.

Generally speaking, one would expect that the more regular and symmetric the layout, the better the matching between transistors throughout. The parallel nature of the filterbank aids considerably in the matching between channels. Edge effects which might cause increased mismatch in the highest- and lowest-frequency channels can be nearly eliminated by placing the elements most sensitive to mismatch as far away from the edge as possible. Double polysilicon capacitors are the least sensitive to mismatch caused by differences in the surrounding layout. Also, the capacitors are large compared to the size of transistors and most of them have one side grounded. The width of the capacitors usually satisfies the distance of 50 to 100 μm over which circuit differences can affect transistor matching. If the capacitors face outward to the edges of the chip, MOS transistors placed behind the capacitors will not suffer any mismatch due to edge effects. The structure of this design is shown in Figure 3.16.

The edge-effect phenomenon was noticed between two versions of the frontend filterbank. One version contained fifteen channels and the other, sixteen. Each filterbank was designed with pairs of channels stacked antisymmetrically, facing each other, as shown in Figure 3.16. The fifteen-channel filterbank, however, had one unpaired channel on the high-frequency end. The lack of pairing was the only factor influencing mismatch that was not present in the sixteen-channel filter. In the sixteen-channel filter, all measured center frequencies were within 2% of their nominal values. In the fifteen-channel filter, all measured center frequencies were within 2% of their nominal values *except for the unpaired channel*, whose center frequency was over 30% too high! By contrast, no systematic difference occurred between even and odd channels, revealing that antisymmetry has no apparent effect on transistor mismatch.

rectification followed by a log-domain lowpass filter for smoothing. However, we were aware from prior experience that rectification requires that the midpoint of the signal be known, and offsets introduced by mismatch and circuit nonidealities can severely complicate the task. So in addition to developing circuits for the direct implementation of the simulation method, we also invented novel circuits which would determine the true peak-to-peak height of each bandpass output, avoiding the necessity of knowing the signal midpoint.

3.13 Signal Rectifier

Figure 3.17 shows a simplified version of the circuit used for the current full-wave signal rectifier. This circuit is adaptive on a fairly long time scale (about 1/10 s to 1 s would be typical), using a *charge pump* for the adaptation. The charge pump (including the inverter, transistors Q_2 and Q_3, and capacitor C_1) acts to keep the current I_1 through transistor Q_1 equal to the input current I_{in} using feedback from the operational amplifier: If the input current exceeds the current through Q_1, then V_{fb} drops, passing V_{ref}, as the system driving the input is driven towards cutoff. Consequently, the op-amp output rises, causing the inverter output to be driven to ground. When that side of the charge pump is grounded, the nMOS transistor in the charge pump pair is activated and sinks current, drawing charge off of the capacitor node. The lowering voltage on the gate of Q_1 increases its drain-to-source current I_1. This negative feedback process continues until I_1 matches I_{in}. The biases Vp_n and Vp_p on the gates of transistors Q_2 and Q_3, respectively, determine the rate at which charge can be placed on or removed from C_1. Thus, the system acts like a crude, nonlinear lowpass filter with the biases determining the time constant of the filter. Time constants are so large in this system that the nonlinearity of the system is irrelevant. I_1 changes very slowly compared to I_{in}, and thus will eventually drift until it settles at the DC average of I_{in}. It will adapt to any DC drift in the input which is longer than the time constant of the charge pump adaptation.

The architecture of the system ensures that the current difference between I_1 and I_{in} always has somewhere to go, leaving V_{fb} virtually unchanged (high gain on the op-amp, of course, makes the system respond faster and more cleanly). When I_{in} exceeds its midpoint I_1, Q_5 is cut off and current is drawn through Q_4, mirrored, and produces a positive I_{out}. When I_{in} falls below I_1, Q_4 is shut off (and the mirror Q_6, Q_7 with it), and positive current again flows to the output, this time directly through Q_5. Apart from mismatch errors, the output current always equals $|I_{in} - I_1|$.

The main differences between the simplified schematic and the one used on the fabricated IC are a source-degenerated ("diode-connected") pMOS transistor between Q_1 and Vdd to raise

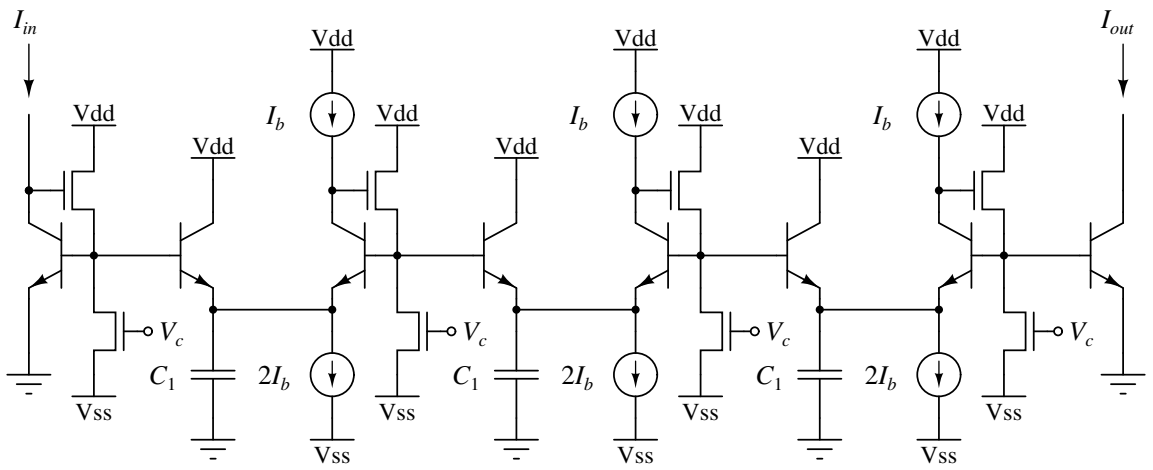

Figure 3.18: Cascaded log-domain lowpass filters.

small current source to cause the capacitor to leak slowly to ground so that the circuit will track the input rather than remain stuck at the largest input voltage. Putting the diode in the feedback loop of an amplifier causes the circuit to be unaffected by the voltage drop across the diode, so that the output very closely follows the rising edge of the input when the diode is not reverse biased. This circuit is shown in Figure 3.19.

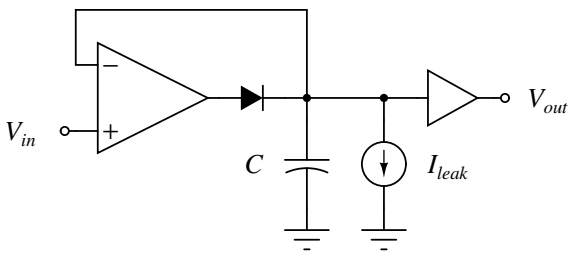

Figure 3.19: Simple diode-based peak detector.

One novel aspect of our circuit is that it operates *differentially* by containing two subcircuits, one which computes the maximum peak, and one which computes the minimum peak (technically speaking, the "trough"). The maximum peak-detecting circuit leaks toward ground, stopping when its value equals that of the input. The minimum peak-detecting circuit leaks toward the positive power rail, also stopping when its value equals that of the input. The circuit output is the result of subtracting the minimum peak-detecting circuit output from the maximum peak-detecting circuit output, which gives a reasonably accurate measurement of the input signal's instantaneous

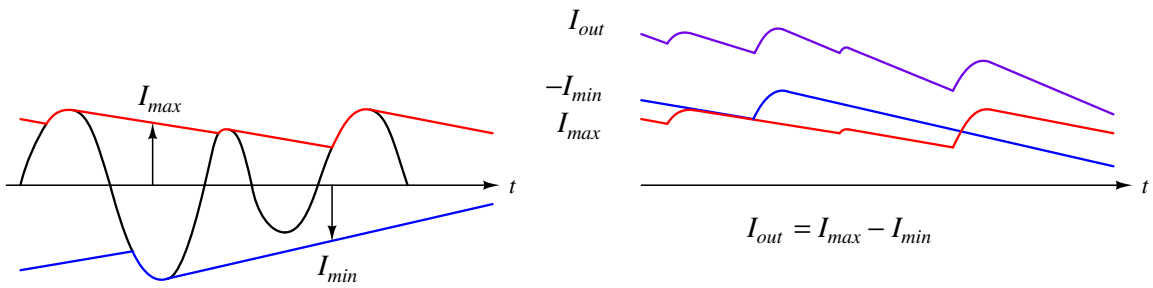

Figure 3.20: Behavior of the peak-peak detector.

rate of charge leakage equal in the two subcircuits, the same size capacitor and pMOS transistor gated by the same bias voltage was used for both. The capacitor voltage always leaks toward Vdd. The capacitor node voltage drives the output transistor gate directly. Where it drives a pMOS transistor, the (maximum) current drifts toward lower values; where it drives an nMOS transistor, the (minimum) current drifts toward higher values.

Figure 3.22 is the final version of the circuit, as implemented on the chip. Extra circuitry has been added to minimize any offset seen at the output due to mismatch between the maximum-detecting circuit and the minimum-detecting circuit. The design takes into account all the major systematic sources of mismatch, primarily by cascoding devices to reduce the Early effect. A significant part of the extra circuitry (Q_4, M_{15}, M_{16}, M_{17}) exists merely to compute the proper cascode bias for transistor M_{21} to keep I_{max} consistent with I_{maxl}.

We employ a short cascade of log-domain first-order lowpass filters at the end of the circuit, just as was done for the signal rectification and smoothing described in the previous section, to smooth out the bumps in the peak detector output. This presents a much smoother signal to the system output at the expense of only a very small amount of circuitry.

3.15 L-1 Normalization Array

The simulation model calls for a final transformation of the energy envelope: a normalization across all channels. The normalization is the "L-1 norm," which follows the following equation (written in units relevant to the current-mode system):

$$I_{out}(i) = I_{norm} \frac{I_{in}(i)}{\sum_{j=1}^{N} I_{in}(j)} \tag{3.41}$$

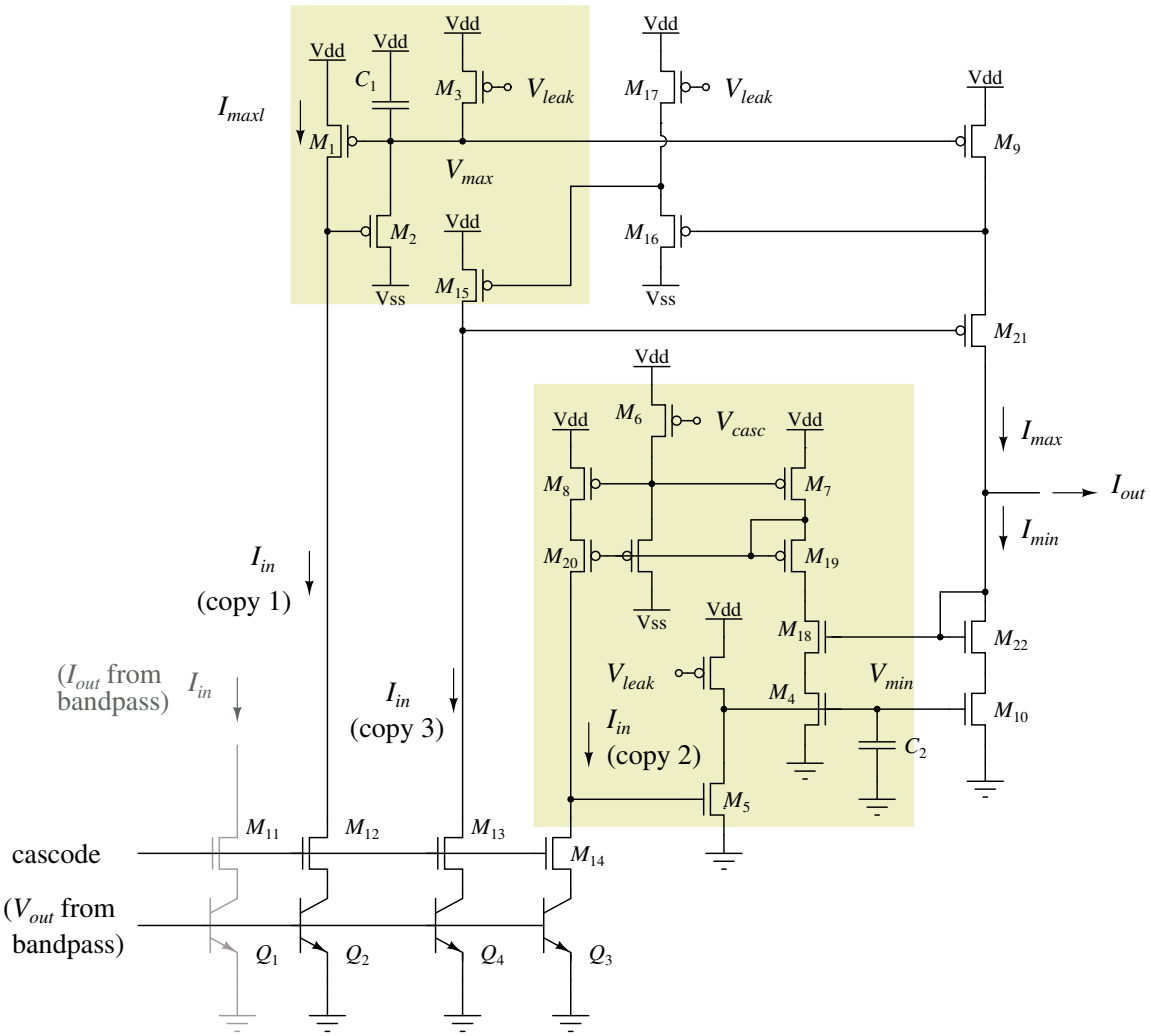

Figure 3.22: Peak-peak detector with cascodes, as fabricated.

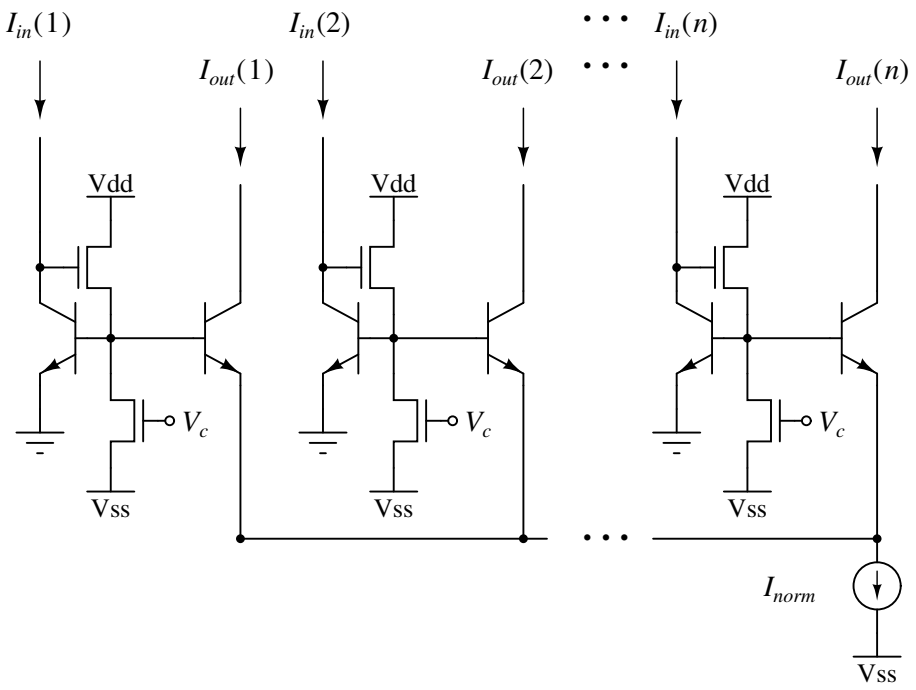

Figure 3.23: L-1 normalization circuit, after Gilbert [50].

filterbank of log-domain bandpass filters (Figure 3.28). The filterbank consists of 15 channels each consisting of two cascaded bandpass filters followed by rectification and smoothing of the signal at the output; The classifier circuit for which this filterbank was designed (Chapter 4) is a current-mode system which takes the current outputs of the frontend system directly, without converting to voltage. The chips were fabricated on 2 mm dies in 2 µm (test chip) and 1.2 µm (filterbank) analog n-well BiCMOS processes, with double metal, double poly, and a p-base layer for creating vertical NPN bipolar transistors. For high-density integration, we limited capacitors to about 2 pF and made all BJTs with a minimum size well. The size of the capacitors implies (from Equation (3.31)) bias currents of 30 pA to 3 nA for center frequencies spaced from 100 Hz to 10000 Hz, respectively.

Figure 3.24 shows measurements taken from a first-order lowpass section on the test chip. The circuit is that shown in Figure 3.7, with a capacitor value of about 1 pF, a bias current ratio $I_a = 2I_b$ and bias currents generated by applying a voltage to the gate of a (cascoded) nMOS transistor in weak inversion. The gate voltage was stepped in linearly-spaced increments, which ideally should yield exponentially-spaced corner frequencies. The data show a falloff of approximately 22 dB/decade and confirm the exponential spacing of corner frequencies. The response maps well

Table 3.1: Measured filterbank characteristics.

Die Size	2.2 mm × 2.2 mm
Process	1.2 µm n-well BiCMOS double-poly, double-metal
Power supply $V_{dd} - V_{ss}$	4.0–6.0 V
V_{ss}	−0.5 V
Number of channels	15
Dimensions of bandpass filter	75 µm × 234 µm
Capacitor size	2.0 pF
Input DC bias	10 µA
Output bias I_{DC}	10 µA
Max. input AC signal	8 µA peak-peak.
Power consumption	200 µW at 5.0 V
Dynamic range	35 dB at listed conditions
Bias current I_b	30 pA to 3 nA
Frequency tuning range	50 Hz to 15 kHz
Q tuning range	1 to 10
Gain nonuniformity across channels	18% at Q = 6

over the entire audio frequency range, although the dynamic range is limited by an approximately 50 dB noise floor. Note that the transfer function at 100 Hz corner frequency implies that with the use of base compensation, the bipolar transistors maintain a well-defined exponential I-V relationship even at collector currents as low as 30 pA.

The remaining data were measured from the filterbank system chip, the characteristics of which are summarized in Table 3.16. All measurements shown were made with a power supply of 5 V, though the system showed similar characteristics at 6 V and 4 V. Cascoded MOS transistors implementing the current sources and mirrors plus the p-n junction voltage drops in the translinear loops limit circuit operation at less than approximately 4 V. Power consumption of the circuit is dominated by the DC bias added to the filter input and output, which affects the SNR at the filter output. For the 5 V power supply and an input/output bias of 10 µA, the filter circuit of Figure 3.14 dissipates about 200 µW and has a dynamic range of about 35 dB.

Figure 3.25 shows characteristics of the bandpass filter at three different frequencies in the audio band and three different Q values. The filter consists of two second-order bandpass sections of the type shown in Figure 3.14, connected in cascade. Both filters in the cascade have adjustable Q values; for our measurements we varied the Q of the first filter while keeping the Q of the second filter fixed. The Q of the second filter has a value approximately equal to but strictly less than one and is used primarily to give the response a 40 dB/decade drop on the skirts of the passband.

to each. The variation is due primarily to error in matching I_b to $I_b + (1/Q)I_b$ in Figure 3.14. The dependency on $1/Q$ results in increasingly large gain differences between channels as Q becomes larger. A filterbank with sufficient number of channels cannot necessarily utilize precision trimming of current sources separately for each channel. Although much of the mismatch can be attributed to the quality of the fabrication process, better matching should be sought through circuit designs which minimize the number of matched current sources, especially those which must match a pMOS current source to an nMOS current sink.

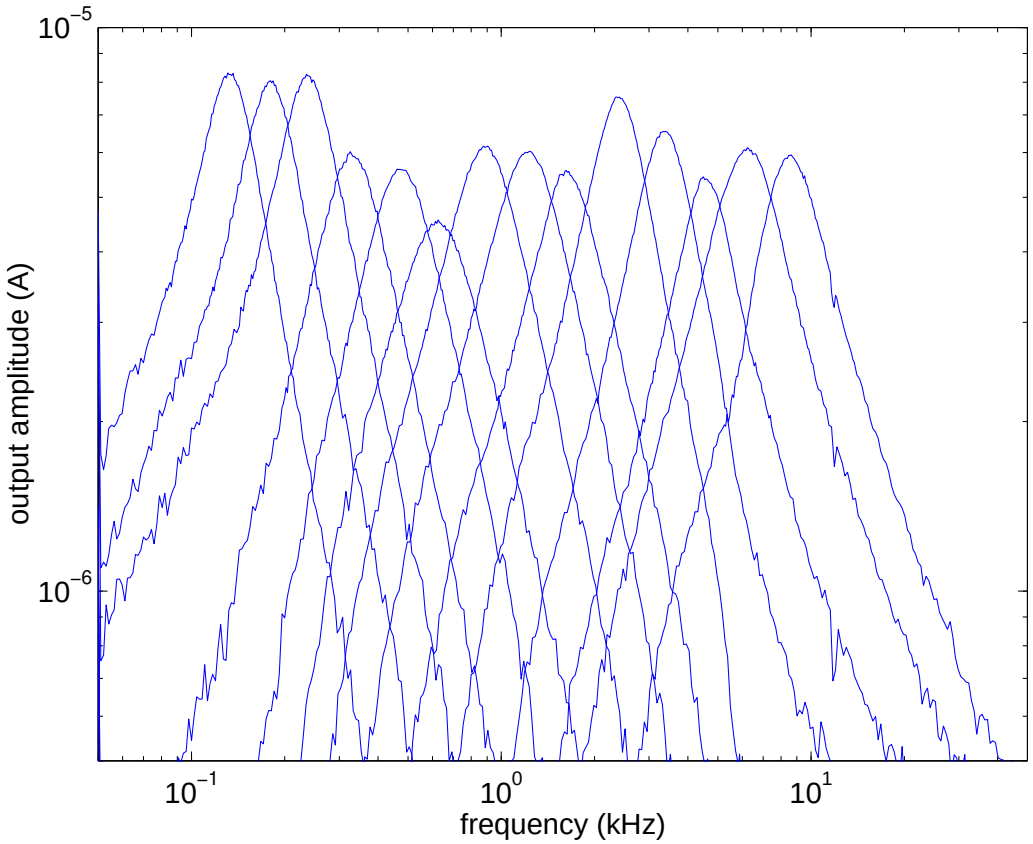

Figure 3.26: Measured magnitude response of all bandpass channels in the filterbank system, as measured at the output after peak-detection and smoothing.

The distribution of center frequencies derives from the same value I_b as the distribution of Q values, and follows the same exponential behavior (as discussed previously in Section 3.9). The main difference is that the filter center frequency is directly proportional to I_b, thus variance in center frequency is directly proportional to variance in I_b. Variations in I_b can come from only two sources: Physical nonidealities in the polysilicon strip which may cause the voltage taps to deviate from linear

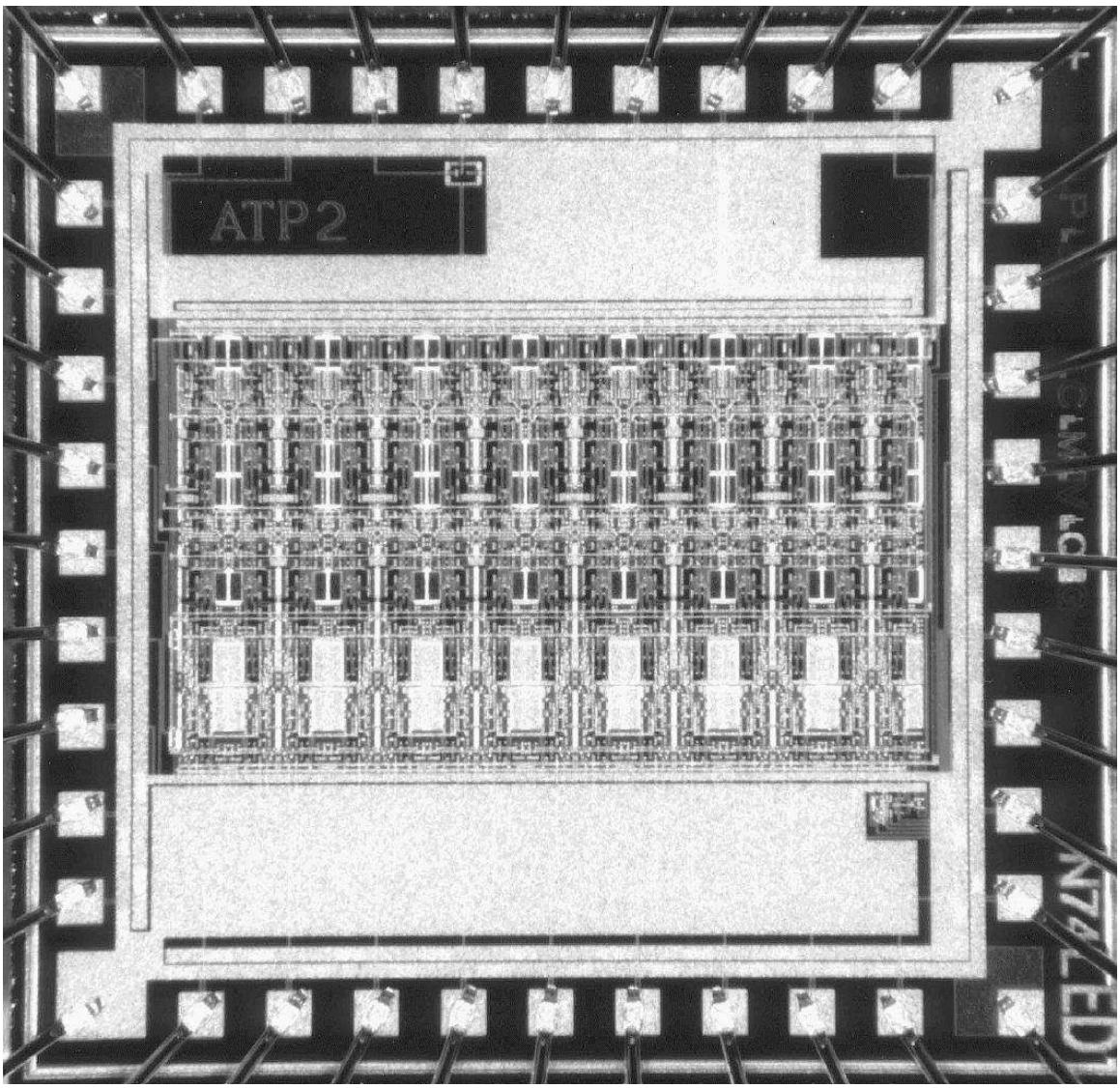

Figure 3.28: Photograph of the fifteen-channel bandpass filterbank fabricated in 1.2 µm technology inside a 2.2 mm × 2.2 mm padframe.

Chapter 4

Acoustic Transient Processing

4.1 Introduction

4.1.1 The Problem of Speech Recognition

Speech recognition is an inherently difficult problem on many different levels. For many years, thousands of companies worldwide have put tremendous money and effort into the problem. Undeniably, they have made many important gains, and coupled with the phenomenal advances in speed and efficiency of computer hardware, have brought the first rudiments of practical speech recognition from the research centers to the office and home. Unfortunately, speech recognition is the type of problem that requires exponentially increasing resources for linear gains in accuracy. It is generally acknowledged that human-level accuracy in speech recognition requires the resources currently available only in the human brain—measured in the range of terabits and processed with a high degree of parallelism and amazing efficiency. It is very likely that the reason no other animal on Earth uses language is a specific threshold in complexity of the brain which only humans exceed. The fact that the number of neural synapses in the brains of other mammals is also in the trillions suggests that speech recognition comes about from a combination of factors including brain size, structure, complexity, and the capabilities of the sensory systems which connect the organism to its environment.

Consequently, in regard to electronic speech recognition, perhaps the proper question to be framed is not how well they perform in an absolute sense, but how well they are doing given the resources they have to operate, and how robust they are under non-ideal conditions. This latter point is not a minor one. Current research in automated speech recognition tends toward the goal

was sound recognition, just one part of a large array of mechanisms to help an organism interpret its environment. Most species of animal larger than insects have some form of auditory system, often quite complex as in the case of birds and birdsong. Humans have developed the capability of speech at most in the last few hundred thousand years, maybe much more recently. Evolutionary forces have little capability for sweeping changes over this timespan, and indeed, the auditory systems of our evolutionary neighbors, the primates, are little different from ours. This means that much of the biological hardware used for speech recognition evolved for other reasons. Many of those reasons, including speech recognition, can be bundled under the term "communication" which itself is an extension of "perception." If we want to know how to design speech recognition hardware from bottom up, and if we believe that evolutionary biology can be a useful field guide for this kind of problem, then we should start with the basic problems of perception and communication before tackling the complexities and intricacies involved in continuous speech recognition.

Since one of the major difficulties of speech recognition is the problem of the length and variation of sounds which represent the same perceived word or phrase, a good starting point would be to look at the perception of acoustic events which are much shorter and have less variance between individual instances. These events are called *acoustic transients*. They comprise the bangs, clicks, pops, thuds, clinks, snaps, and other sounds which make up an important part of the daily perception of our environment. In addition, they include the transients from the world of biological echolocation, sets of sound reflections encoding visual information for animals such as bats and porpoises, and the contemporary electromechanical equivalent, sonar. Long before humans could speak to one another, they were concerned with the snap of a twig telling of an approaching enemy or friend, the drip of sought-after water in the desert, and the sounds of all the vocalizing animals to tell the difference between predator and prey. These capabilities go way down the food chain, and given evolutionary nature's tendency to build more complex systems on top of less complex ones rather than developing new systems from scratch, they can be considered fundamental to the problem of speech recognition and communication.

4.2 Algorithms

There exist many kinds of pattern classification algorithms. Some of them work by finding the correlation between a target pattern and a stored prototype pattern, called the *template*. Correlation is one of the simplest classification schemes, both conceptually and mathematically. The correlation is a linear system effectively reducing the dimensionality of a large input space into a

topics of how to perform the classification from the correlation outputs and how to determine the optimal template for each class are discussed further in Section 4.3.6 and following.

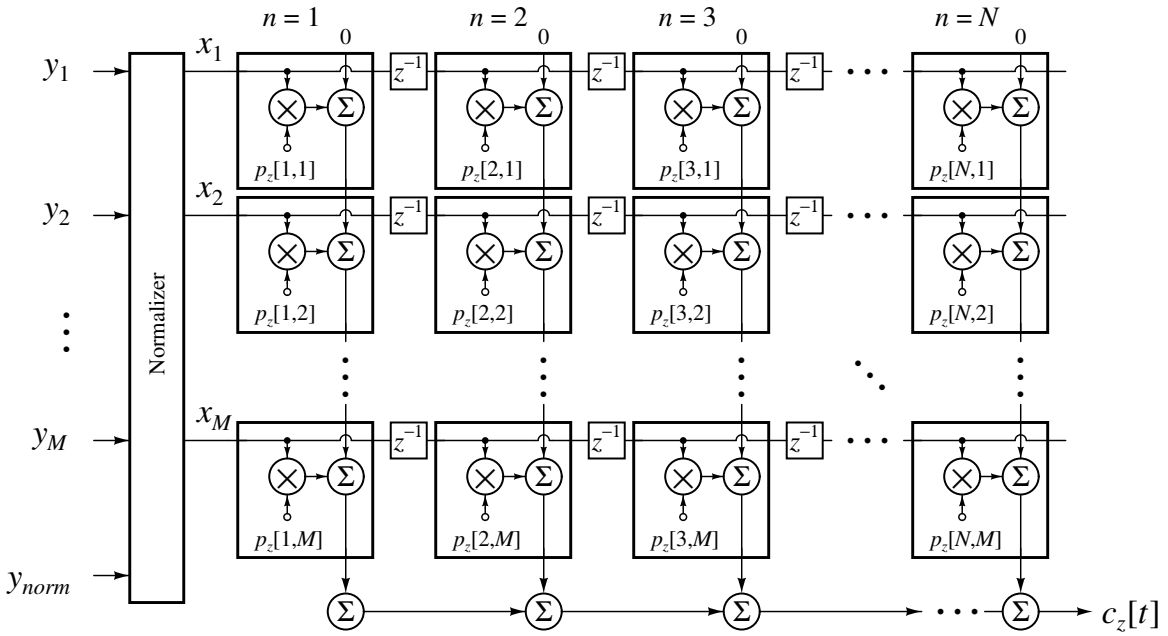

Figure 4.1: Template correlator, as a direct implementation of the baseline algorithm, Equation (4.1).

Figure 4.1 shows the architecture as a direct implementation of Equation (4.1). For large M and N, this algorithm can be expensive to execute on a DSP in terms of speed and power requirements, since it calls for a fixed-point or floating-point multiply and accumulate at every template cell, followed by delay stages implying fixed-point or floating-point storage, and additional accumulates. In practice, workable solutions require that the incoming signal be segmented such that the correlation need be computed only at certain points in time. This causes the classification to be heavily dependent on the quality of the segmentation algorithm and reduces the robustness of the system. For reasonably-sized problems, a full correlation potentially could be done in real time with dedicated hardware. However, there are two points I would like to make in that regard: First, in light of the preceding discussion on biology and speech recognition, I view the problem of acoustic transient recognition and its potential solution, template correlation, as being a fundamental building block of bottom-up methods for attacking the problem of speech recognition. The fundamental parts of the hardware may be required by higher-level processing hardware or software to execute many times, and should be simple, cheap, fast, power efficient, and relatively robust. DSPs are fast

1. First, we normalize the input.

2. Next, we transform the input and template into a zero-mean representation by using the pairwise difference of channels with respect to the original.

3. We replace the template values with binary values.

4. For considerations of simplified analog hardware, we commute the differencing operation between channels to the output.

4.2.1 Simplifying The Correlation Equation

We assume that the template correlation system receives an input vector y which is the two-dimensional output of a filterbank system, as described in detail in Chapter 3. If w is a live or recorded acoustic signal (see Figure 3.1), the frontend filterbank processor splits it into M bandpassed frequency bands, after which the short-term energy envelope y for each band is estimated by rectification and smoothing.

Our first step, normalization, is essential for the steps which follow, but it is also motivated by a need for some type of gain control to increase system robustness. Consequently, the optimal template values must be computed using the normalized rather than the original inputs during training of the templates for each class. We normalize the system inputs by the *L-1 norm* function

$$x[t,m] = \frac{y[t,m]}{\theta + \sum_{k=1}^{M} y[t,k]}, \tag{4.2}$$

where we have simplified the presentation of the algorithm by assuming a dimensionless output normalized relative to unity. The constant value θ suppresses noise during quiet intervals in the input. If this value is not added into the system, then during intervals of silence at the input, all channels will be amplified until the random noise on the input reaches unity value. While one may correctly assume that amplified noise is not likely to correlate with the template any better than silence (zeros) itself, white or (in particular) babble noise does have a non-zero probability of generating false alarm errors (see Section 4.3.7). If θ is larger than the input, then the input will remain at a low level and not be amplified significantly, as θ takes most of the amplification instead. This helps reduce the false alarm rate of the system, particularly if the noise level at the input is well known and the signal-to-noise ratio is reasonably large. However, note that the only consequence of a low signal-to-noise ratio is an elevated false alarm rate: The system is robust

acceptable classification performance is possible with such a minimal representation of the template, which is presented in Section 4.3.2.

To maintain the system accuracy using single-bit templates it is necessary to have a criterion by which a choice may be made as to whether a particular template bit should be $+1$ or -1. A natural choice can be made if the input (on which the template is trained) has a zero-mean representation. Unfortunately, the normalized output of the frontend filterbank processor (Figure 4.3) is a strictly positive-valued representation. A further transformation is required to produce a zero-mean representation, for which the most convenient method is differentiation (or differencing, in discrete space). Numerous options are available, including differentiation in time, computing pairwise differences between channels, computing a center-surround (sometimes referred to as a "Mexican hat") function, or some weighted combination of time sample and frequency channel differences. Simulations have proven that the calculation of pairwise channel differences has by far the best tradeoff in simplicity and robustness. Time differentiation is not as effective for acoustic transient classification, but as it has an especially efficient implementation and may be useful for certain recognition tasks, the algorithm and architecture is included in Appendix C.

We calculate the pairwise difference between channels of the input and compare it to the corresponding difference between channels of the template. The effect of this step upon the recognition task is negligible, as it amounts to having run the input through a spatial highpass filter, subtracting off what amounts to a constant factor. The templates are static values, so the difference between channels of each template replaces the original values.

Now that the (transformed) input and template values are zero-mean over time, we may replace the template values by their *sign*, indicating whether the energy in a particular channel is expected to be larger or smaller than its neighbor. Thus, the new formula is written:

$$c_z[t] = \sum_{n=1}^{N} \sum_{m=1}^{M} x'[t-n, m]\, p'_z[n, m] \tag{4.3}$$

where

$$x'[t, m] = (x[t, m] - x[t, m-1]) \tag{4.4}$$

$$p'_z[n, m] = \text{sign}(p_z[n, m] - p_z[n, m-1]). \tag{4.5}$$

Through simulations, we have shown that binarization of the template has a negligible effect on classification performance [56]. Interestingly, this result does not hold for a binary input and a continuous-valued template.

$$q[1,t] = \sum_{m=1}^{M} x'[t,m]\, p'_z[1,m] \tag{4.7}$$

$$c_z[t] = q[N, t-1] \tag{4.8}$$

A short proof that the pipelined architecture is equivalent to the original architecture can be found in Appendix C, Section C.1. The main difference between Figure 4.1 (the original, non-pipelined architecture) and subsequent figures (4.4, 4.6) is that the template is reversed from left to right. In the non-pipelined architecture, the oldest input vector $x'[t-n, m]$ is closest to the output, having passed through N delays while propagating from left to right. In the pipelined architecture, the *most recent* input vector $x'[t, m]$ is closest to the output, being the last state to be accumulated on the delay-accumulate register before the output is read.

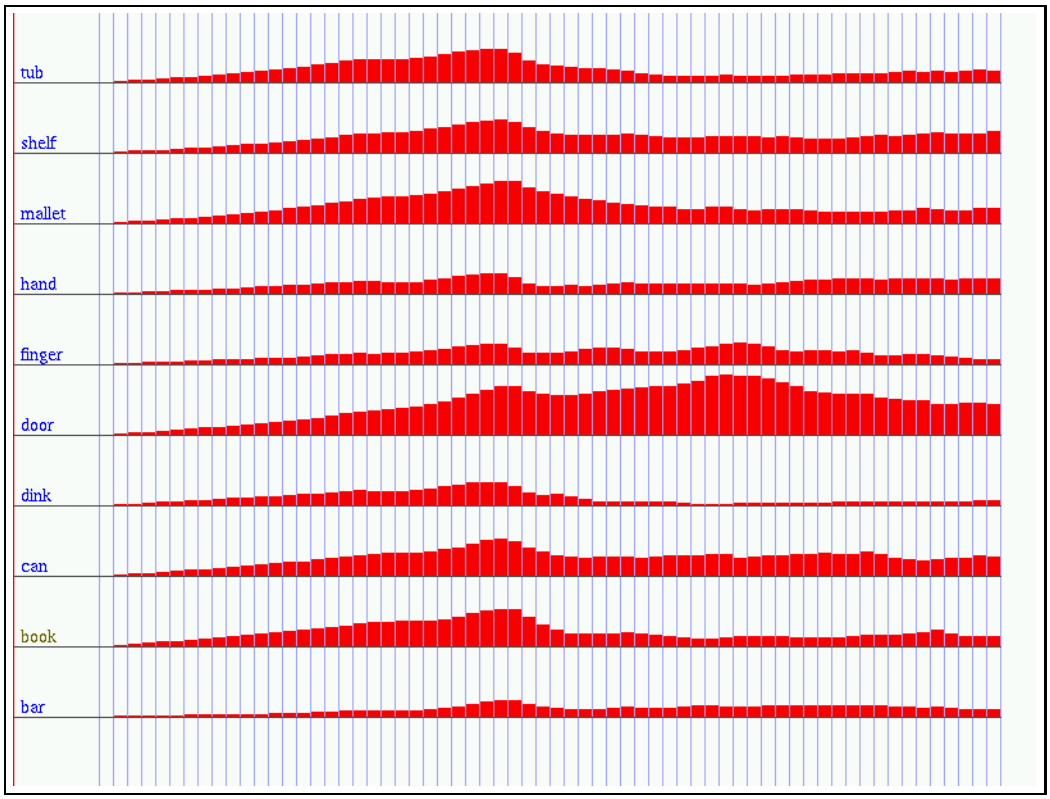

Figure 4.5: Values in the pipelined delay registers in the correlation algorithm simulation.

For all-digital ATP systems, the algorithm of Equations (4.6–4.8) represent the most efficient architecture. For digital systems, the signed arithmetic for the multiplications can be handled easily by XOR logic, so that the overhead is minimal. The four-quadrant multiplication is still problematic for analog implementations, however.

registers can be built that will be reasonably well-matched. Call these two registers q_1 and q_2:

$$q_1[n,t] = q_1[n-1,t-1] + \sum_{m=1}^{M} x[t,m]\, p'_z[n,m] \qquad \forall n \neq 1 \tag{4.9}$$

$$q_2[n,t] = q_2[n-1,t-1] + \sum_{m=1}^{M} x[t,m-1]\, p'_z[n,m] \qquad \forall n \neq 1 \tag{4.10}$$

$$q_1[1,t] = \sum_{m=1}^{M} x[t,m]\, p'_z[1,m] \tag{4.11}$$

$$q_2[1,t] = \sum_{m=1}^{M} x[t,m-1]\, p'_z[1,m] \tag{4.12}$$

Thus, the correlation output becomes:

$$c_z[t] = q_1[N,t-1] - q_2[N,t-1] \tag{4.13}$$

Note that in this case the input $x[t,m]$ which comes straight from the frontend filterbank processing and has *not* been transformed by a differencing operation, is multiplied directly with the template value $p'_z[n,m]$, which been transformed.

One distinct advantage of commuting the difference operation is that only one difference needs to be evaluated rather than M separate differences, one for each channel, and so while we must be concerned with how well matched the two delay-accumulate registers are, and how well matched the input mirroring transistors are, we do not have to worry about matching and accuracy of the channel difference calculation across all the channels. Most importantly, since the inputs x are rectified and therefore strictly positive, the product xp', where p' is binary $[0,1]$, is always nonnegative and equals either x or zero. The implied multiplication is in the positive quadrant only, which allows us to conveniently implement the entire correlation as an array of single-transistor current sources and single-transistor current switches. The overhead required by splitting the correlation into two parts for the delay-accumulate registers q_1 and q_2 is not as large as one might expect (see Section 4.4.1 and following). The resulting high density layout of templates enables an entire acoustic pattern recognition system to be placed on a single chip in conventional CMOS technology. A typical application consisting of an analog current-mode acoustic frontend processor and 1024-bit templates (M = 16, N = 64) to classify a dozen different transient sounds can fit in a 4 mm $\times$ 6 mm die area in a 1.2 µm technology.

Our choice of an analog implementation stems from the ability to make such a system compact and extremely power-efficient. With a separate correlator circuit for each template, the

the light of the possibilities and the limitations of the chosen hardware. This and the following sections specifically addresses improvements in classification performance achievable by algorithmic modifications, tailored to the constraints and strengths of the implementation medium.

4.3.1 The Hopkins Electronic Ear (HEEAR) Processor

To train and evaluate the classification system, we acquired a database of approximately twenty recorded samples of each of ten different classes of "everyday" transients using materials obtained from a typical office/laboratory environment at the time the recordings were made [54]. The ten classes are listed in Table 4.1.

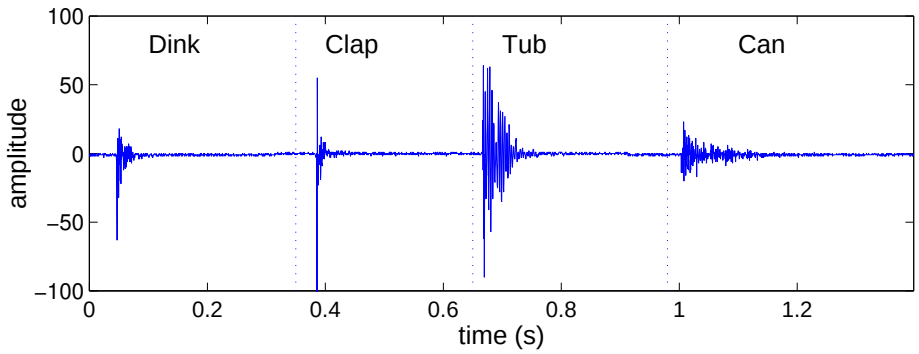

Figure 4.8: Example transient recorded data.

The complete recording of the transients was played back into a data acquisition system, sampled at 32 kHz, and saved to disk. Then the sampled stream was processed through a system based around the analog VLSI *HEEAR* chip [53]. The HEEAR processor is an electronic cochlear model, a modification and extension of the original design presented in *Analog VLSI and Neural Systems* [1]. The HEEAR chip is like the original electronic cochlea in that it models the mechanics of the mammalian basilar membrane using a long cascade of lowpass filters, with outputs tapped along the length of the cascade. It differs in that it makes use of first-order lowpass filters along the

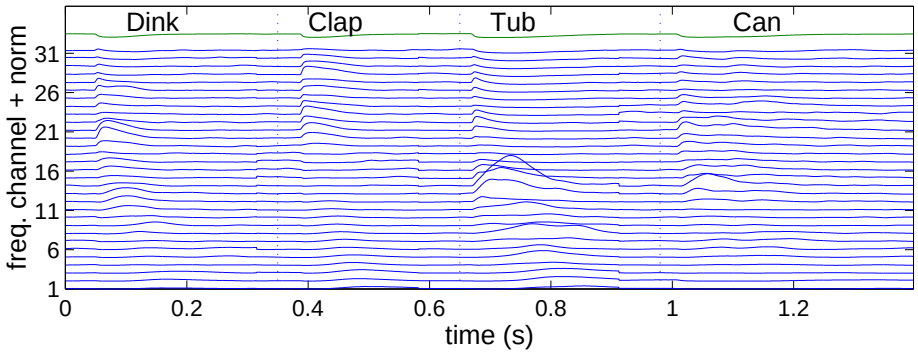

Figure 4.9: Example HEEAR-processed transients.

normalized with the L-1 norm function (Equation (4.2)).

The value θ itself can be normalized and used as another output channel:

$$x[t, M+1] = \frac{\theta}{\theta + \sum_{k=1}^{M} y[t,k]}, \tag{4.14}$$

For typical values of θ, the additional output $x[t, M+1]$ inversely follows the energy envelope of the input signal, which is the sum of the individual channel outputs. It becomes maximum during the periods of silence and minimum during presentation of a transient event. This extra output can be used to detect onsets of transients, but is not used in the correlation computation of Equation (4.9–4.12).

The next stage of processing involved automatic segmenting of the recordings into individual transient examples. The automatic segmenting algorithm finds transient boundaries using the normalized channel $x[t, M+1]$. Quantizing this signal with a hard-limiting threshold function produces a noisy estimate of the segment, which is then smoothed and thresholded again to produce a clean segmentation signal lasting the duration of the transient. The smoothing and thresholding step is performed twice in parallel, once with a smoothing time constant of 10 ms and once with a

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub
Bar	28	0	0	0	0	0	0	0	0	0
Book	0	19	0	0	0	0	0	0	0	1
Can	0	0	27	0	2	0	0	0	0	0
Dink	1	0	0	25	0	0	1	1	0	0
Door	0	0	0	0	9	0	0	1	0	0
Finger	0	0	0	0	0	17	1	0	0	0
Hand	0	0	0	0	0	0	21	0	0	0
Mallet	0	0	0	0	0	0	0	12	0	0
Shelf	0	0	0	0	0	0	0	0	28	0
Tub	1	0	0	0	0	0	0	0	0	28

Table 4.2: Confusion matrix for leave-one-out cross-validation loop, using the baseline algorithm.

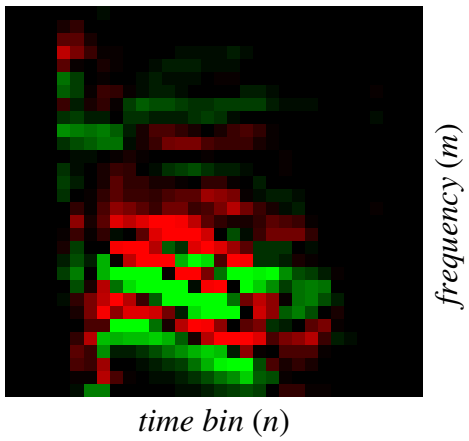

Figure 4.11: Example of a real-valued template.

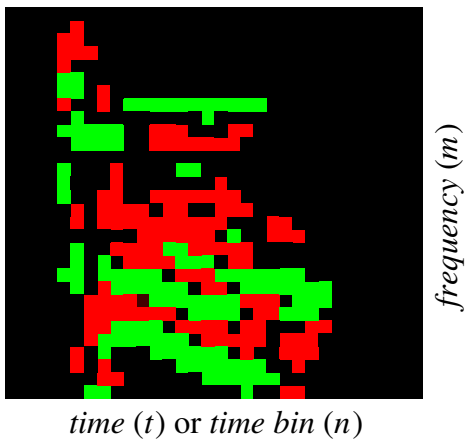

Figure 4.12: The same template reduced to trinary values.

on the time scale of transients.

Table 4.3: Simulation results with different architectures.

Method	Both Cont.	Binary Input	Both Binary	Binary $(1, -1)$ Template	Binary $(1, 0)$ Template
One-to-One	96.40%	—	—	—	—
Time Difference	85.59%	65.32%	59.46%	82.43%	81.98%
Channel Difference	90.54%	53.60%	95.05%	94.59%	94.14%
Center-Surround	92.79%	53.60%	95.05%	92.34%	92.34%

Table 4.3 shows simulation results, where classification accuracy on the cross-validation test is given for different combinations of continuous-valued and binary inputs and templates, and different zero-mean transformations of the input. There are several significant points to the results of these classification tasks. The first is to note that in spite of the fact that acoustic transient events are short-term and the time steps between the bins in the template as low as 2 ms, using time differences between samples does not yield reliable classification when either the input or the template or both is reduced to binary form. However, reliability remains high when the correlation is performed using channel differences. The implication is that even the shortest transient events have stable and reliable structure in the frequency domain, a somewhat surprising conclusion given the impulsive nature of most transients.

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub
Bar	27	0	0	0	1	0	0	0	0	0
Book	0	19	0	0	0	0	0	0	0	1
Can	1	0	22	0	4	0	0	0	1	0
Dink	1	0	0	26	0	0	1	0	0	0
Door	0	0	0	0	10	0	0	0	0	0
Finger	0	0	0	0	0	18	0	0	0	0
Hand	0	0	0	0	0	3	18	0	0	0
Mallet	0	0	0	0	0	0	2	10	0	0
Shelf	0	0	0	0	1	0	0	0	27	0
Tub	0	0	0	0	0	0	0	0	0	28

Table 4.4: Confusion matrix for leave-one-out cross-validation loop, using binary $(1, 0)$ templates and channel differencing.

Another interesting point is that we observe no significant difference between the use of pairwise channel differences and the more complicated center-surround mechanism (twice the channel value minus the value of the two neighboring channels). The slight decrease in accuracy

4.3.5 High-Level Simulations of ATP Mixed-Mode VLSI Hardware

We independently confirmed the results of the software trials using a separate system more accurately representing the hardware being developed for the ATP project. Primarily, this involved a full high-level simulation of the log-domain parallel bandpass filterbank to confirm that results of the classification algorithm were independent of the specific features of the frontend system used; in other words, that the performance was not specifically linked to the behavior of the HEEAR chip in some way that would render the algorithm less accurate on the proposed parallel bandpass filterbank system.

To train and evaluate the system, we used the same database of recorded samples of 10 different classes of "everyday" transients described in the preceding section. We simulated the frontend filterbank described in Chapter 3, a sixteen channel filterbank having a Q of 5.0 and with center frequencies spaced on a mel scale from 100 Hz to 4500 Hz. The bandpass filtering was followed by rectification and smoothing with a lowpass filter function with a cutoff frequency scaled logarithmically across channels, from 60 Hz to 600 Hz. The channel output data were decimated to a 500 Hz rate (2 ms period). Half of the database was used to train the system, and half used to test.

In general, performance on classification tasks was similar to that of the system using HEEAR chip outputs, in spite of the fact that the time period of samples was doubled, the number of channels cut in half, and the number of training examples also cut in half. Slight gains in performance on certain tasks are most likely due to the cleaner digital filtering of the recorded data.

4.3.6 Optimization of the classifier using per-class gains

The baseline algorithm simulation reported in Appendix E returns a correlation value equal to the dot product divided by the number of time samples in the template. Thus:

$$c_z[t] = K_z \sum_{m=1}^{M} \sum_{n=1}^{N_z} x[t-n, m]\, p_z[n, m] \tag{4.15}$$

$$K_z = \frac{1}{N_z} \tag{4.16}$$

This slight difference in the correlation equation was not included in Equation (4.1) due to the fact that the VLSI hardware has a fixed number of time samples per template N, so there is no point in dividing each result by a constant value. The point of the normalization, however, is obvious for the baseline algorithm, considering that some acoustic transient classes have inherently more energy than others. If the outputs for each template are compared solely on their correlation values,

just prior to quantizing. Thus:

$$K_z = \sqrt{\sum_{m=1}^{M}\sum_{n=1}^{N_z} {p''}_z[n,m]\,{p'}_z[n,m]} \tag{4.18}$$

$${p''}_z[n,m] = p_z[n,m] - p_z[n,m-1] \tag{4.19}$$

$${p'}_z[n,m] = \mathrm{sign}({p''}_z[n,m]) \tag{4.20}$$

where $p_z[n,m]$ are the same template values used for the template in the baseline algorithm Equation (4.1). This formula results in different K_z values when the inputs are continuous-valued and the templates are binary-valued.

There is yet another consideration to be made in determining values K_z. An alternate strategy is based on the realization that while normalization of all template autocorrelations is important, there is also valuable information to be gained from the *cross-correlations* between class templates or between class templates and class example averages, as used in Equation (4.20). The cross-correlation of class average ${p''}_i$ with template ${p'}_j$ yields the expected average output value when an input example of class i is correlated against the template for class j. Clearly, what we want is to maximize the expression

$$K_i \sum_{m=1}^{M}\sum_{n=1}^{N} {p''}_i[n,m]\,{p'}_i[n,m] \tag{4.21}$$

while at the same time minimizing the expressions

$$K_j \sum_{m=1}^{M}\sum_{n=1}^{N} {p''}_i[n,m]\,{p'}_j[n,m] \qquad \forall j \neq i. \tag{4.22}$$

We do this by inventing an error expression by which we can evaluate the state of the condition above, and then maximizing or minimizing the expression with respect to K_i and K_j. Note that for all of the following equations, we will use the value N to denote the width of the template. This assumes either that all templates have the same width N or that the shorter of the two expressions is padded to match the other.

We start with a $Z \times Z$ matrix of cross-correlations, denoted C, where Z is the total number of classes:

$$C_{ij} = K_j \sum_{m=1}^{M}\sum_{n=1}^{N} {p''}_i[n,m]\,{p'}_j[n,m] \quad i = 1 \ldots Z,\ j = 1 \ldots Z \tag{4.23}$$

Matrix element C_{ij} is the expected value for the correlation between a typical example of a transient input i and the template for its own class j. Therefore we wish to maximize each diagonal element

well as a convenient power of 2 for addressing the memory). However, it is interesting to note that recognition accuracy does not drop to that of random chance until only two time bins are used (a total of 64 bits per template), and accuracy is nearly 50% with only 3 time bins (a total of 96 bits per template).

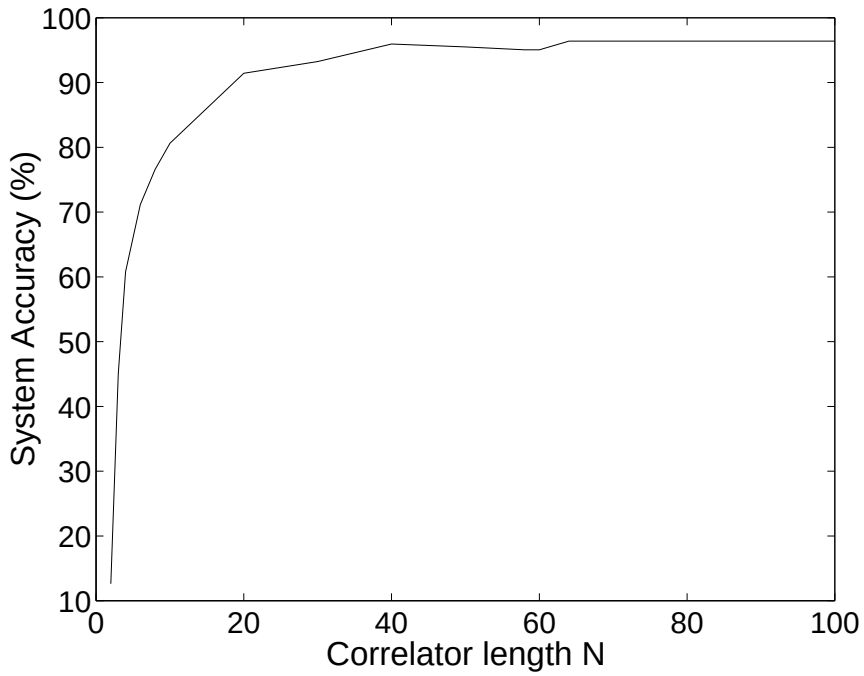

Figure 4.14: Effect of decreasing the number of time-bins.

Examining the effect of a different number of frequency channels is a difficult process due to the requirement of regenerating the input data for each set, since the filter placement and bandwidth must change when the number of filter channels changes. However, we did make one critical measurement which was to note the difference between operation at 32 channels (the original simulations) *vs.* 16 channels (the size of the VLSI hardware) with all other parameters of the recognition task fixed. The recognition task which maintained an accuracy of 94.1% under 32 channels dropped only slightly to 91.9% when using only 16 channels. In both input sets, the Q of the filters was adjusted to maintain similar overlap between adjacent filters, and the frequency span covered by all filters in the filterbank was approximately the same.

We made one evaluation of the robustness of the algorithm in the presence of noise by introducing additional white noise at the correlator inputs. The graph of Figure 4.15 shows that accuracy remains high until the signal-to-noise ratio is roughly 0 dB, after which it degrades gracefully

have been based upon the assumption that the template values arrived at by aligning and averaging all class examples is immutable. By considering template values fixed, normalization becomes a relatively simple problem. We have one degree of freedom and we can be certain that we have found an optimal solution under these constraints.

The acquisition of templates by aligning and averaging all examples of a class is a reasonable and simple method based on the fact that a function's autocorrelation peaks at the position of exact alignment. However, it does not guarantee that the resulting template values are in any way "optimal" over all time and against all competing class inputs. It does not consider correlation peaks which might occur away from the position of alignment of the templates, and it does not take into account any information about competing classes. Releasing the constraint of fixed templates expands the optimization problem into so many dimensions that finding a solution becomes a formidable task. Some strategies toward this goal are discussed in Chapter 5.

4.3.9 Remarks

Starting from a template correlation architecture for acoustic transient classification targeted for high-density, low-power analog VLSI implementation, we have investigated several variants on the correlation algorithms, accounting for the strengths and constraints of the VLSI implementation medium while maintaining acceptable classification performance.

Reduction of input and templates to binary form does not significantly affect performance, as long as they are transformed to encode the difference in neighboring channels of the original filterbank frontend outputs. This suggests that acoustic transient classification is not only amenable to implementation in simple analog hardware, but also in reasonably simple digital hardware.

In looking for zero-mean representations of the input compatible with a binary template, we found that computing pairwise differences between channels gives a more robust representation than a time-differential form, as was reported previously in [55]. We have found that computing a center-surround function of the inputs yields virtually the same results as taking pairwise channel differences. Where hardware implementation is the goal, the pairwise difference function is preferred due to its greater simplicity.

We have additionally shown that cross-correlations between aligned, averaged inputs and templates can be used with an iterative method to solve for optimal gain coefficients per class output, which yield better classification performance. This is a method which can be applied in general to all template correlation systems.

Two additional switches (M_4 and M_5) determine the phase at the memory cell. On the delay lines, the same signal Φ determines which delay line performs the accumulation (Figure 4.20).

Through the various optimizations to the algorithm described in Section 4.2.1, we have reduced the array core cell to five transistors. It could be made with as few as three transistors by having two metal lines carrying the positive and negative parts of the column sum currents to the two delay lines at the bottom of the array rather than sharing the one column line as shown in the figure. However, for the two-metal layer, two-poly layer process we used to fabricate the circuit, the five-transistor cell is actually more compact than the three-transistor cell due to the design rule constraints on the separation of metal lines, and the fact that the four transistors used as switches (M_2 through M_5) can be minimum-size devices.[2] By contrast, the input current mirror transistor M_1 should *not* be minimum size, as transistor area is inversely related to device mismatch. Transistor and core cell sizes are reported in Table 4.7. Dimensions are shown in lambda (λ) where the fabrication process parameter $\lambda = 0.6\,\mu$m.

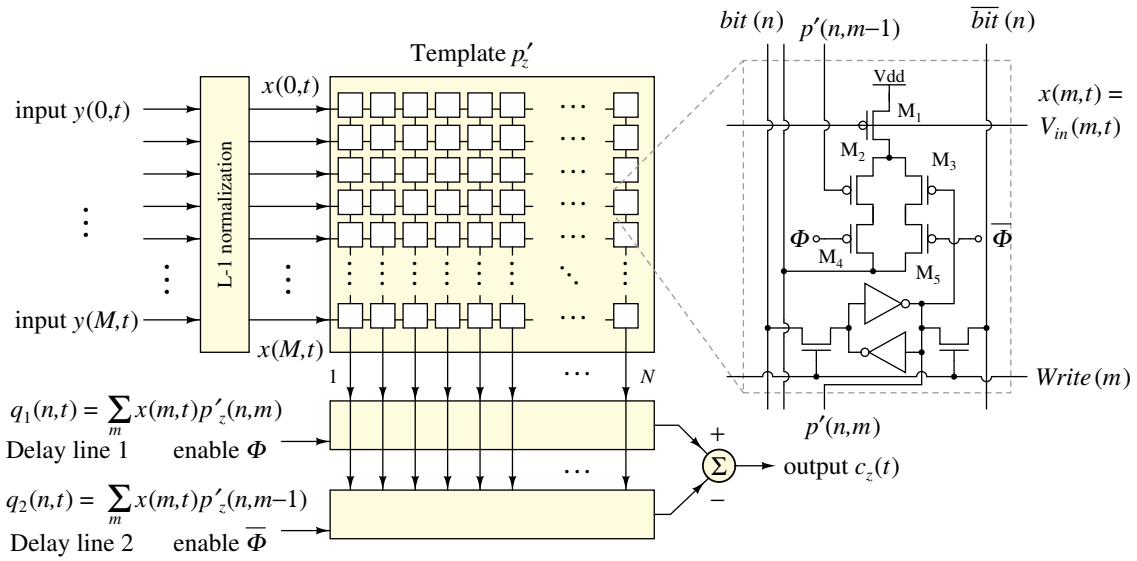

Figure 4.16: Block diagram of the temporal current correlator.

The current-mode correlator cell is an ideal circuit architecture for a dynamic, rather than static, memory. Figure 4.17 shows how a dynamic memory cell can be constructed (although it does not show the sensing circuit, one per column, used to determine the value of the memory during refresh and set the $\overline{bit}(n)$ line accordingly). The switches in the core cell (M_2 through M_5)

[2] This is consistent with the usual rule of thumb about the expected fraction of a digital circuit which will be occupied by routing (metal) layers *vs.* the space occupied by the gates (transistors) themselves.

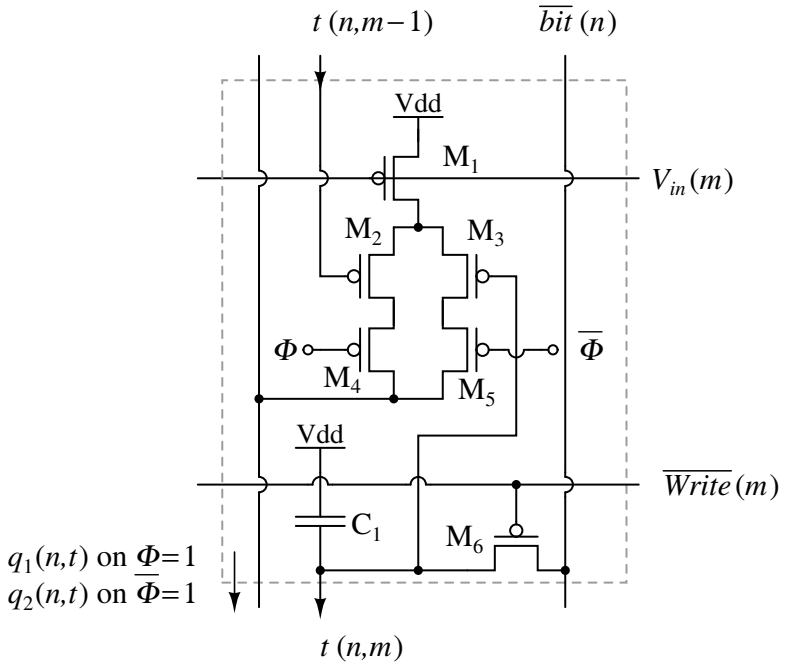

Figure 4.17: Efficient dynamic memory cell for the correlator array.

dynamic version consumes more power due to the constant leakage and refreshing, but takes considerably less layout area than its static counterpart, due to the fewer transistors required but also due to the fact that the cell can be made with a single type of transistor (pMOS, in the figure) which allows a much more compact core cell.

4.4.2 Bucket Brigade Device

Pipelined, sampled-time analog delay lines have a number of different possible implementations, many of them rather complicated. The task of the delay line in the acoustic transient processor requires both a pipelined delay and an addition (Figure 4.6). We implement the delay-accumulate register using a *bucket brigade device* (BBD) [60]. This device is similar to a CCD line, but is more appropriate for this application, in which the system is clocked at a rate of 1 or 2 ms: while the charge-transfer efficiency in a bucket brigade is less than that of a CCD [61], the CCD is adversely affected by dark currents in the quiescent state and cannot operate at slow (auditory) rates. Large poly1-poly2 capacitors, which are significantly less affected by leakage currents than CCD capacitors, store the charge at each bucket brigade node.

Figure 4.18 shows the bucket brigade line. It is driven by a nonoverlapping two-phase

experimentally but which is typically a few tenths of a volt below the positive power supply V_{dd} [62].

The bucket brigade device may be put into service as an accumulator quite conveniently. Most bucket brigade circuits in the literature are used as analog delay lines with the sampled voltage applied to one end of the BBD and the output detected n stages later. Our use of the bucket brigade as a pipelined accumulator is apparently a novel application of the device. Our bucket brigade receives input at every single stage in the form of the partial column sum current provided by the correlator template array. The input current is switched on and off through a source-switched transistor. Integration occurs during a short, fixed-width pulse determined by the signal ϕ_3 (Figure 4.18). The transistor is switched by the source rather than the gate to prevent charge injection into the sensitive bucket brigade capacitor node. A similar treatment to that given to the bipolar mirrors of the log-domain structures of Chapter 3 is given to the source-switched mirror. Rather than being a base compensation scheme, its primary purpose is to provide sufficient current to the mirror at all times to enable it to charge up its own parasitic capacitances quickly and give the fastest current-switching response. The parasitic capacitance between the source and gate of the source-switched transistor will tend to pull down the gate voltage when the source is lowered. If the gate is part of a simple mirror, then the circuit recovers by charging the parasitic capacitance directly from the input current, which often is very small. If the mirror cannot recover fully in a short time compared to the period of ϕ_3, then the linearity of the current-to-voltage conversion suffers.

According to the channel-differencing algorithm, there are two matched bucket brigade devices in the system. The input current from the template array is shared, with the integration pulse ϕ_3 alternating between the two BBDs. The output of one bucket brigade must be subtracted from that of the other, according to Equation (4.13). We accomplish this with the switched capacitor circuit shown in Figure 4.19. We measure the output relative to the voltage V_{ref}, and scale it by the ratio C_1/C_2 (the scale is arbitrary). The circuit must be reset by S_1 at the same frequency as the bucket brigade clocks. The output is valid between the ϕ_2 and S_1 clocks. The bucket brigade is fully pipelined, yielding one full correlation at every time step.

4.4.3 Circuit input section

The circuit receives its input as an array of currents from the filterbank frontend system. We have adopted a system which allows multiple template correlator integrated circuits to be connected to a single filterbank frontend. One template correlator acts as the "master," receiving the currents from the frontend filterbank, and producing voltages which are used to mirror the input

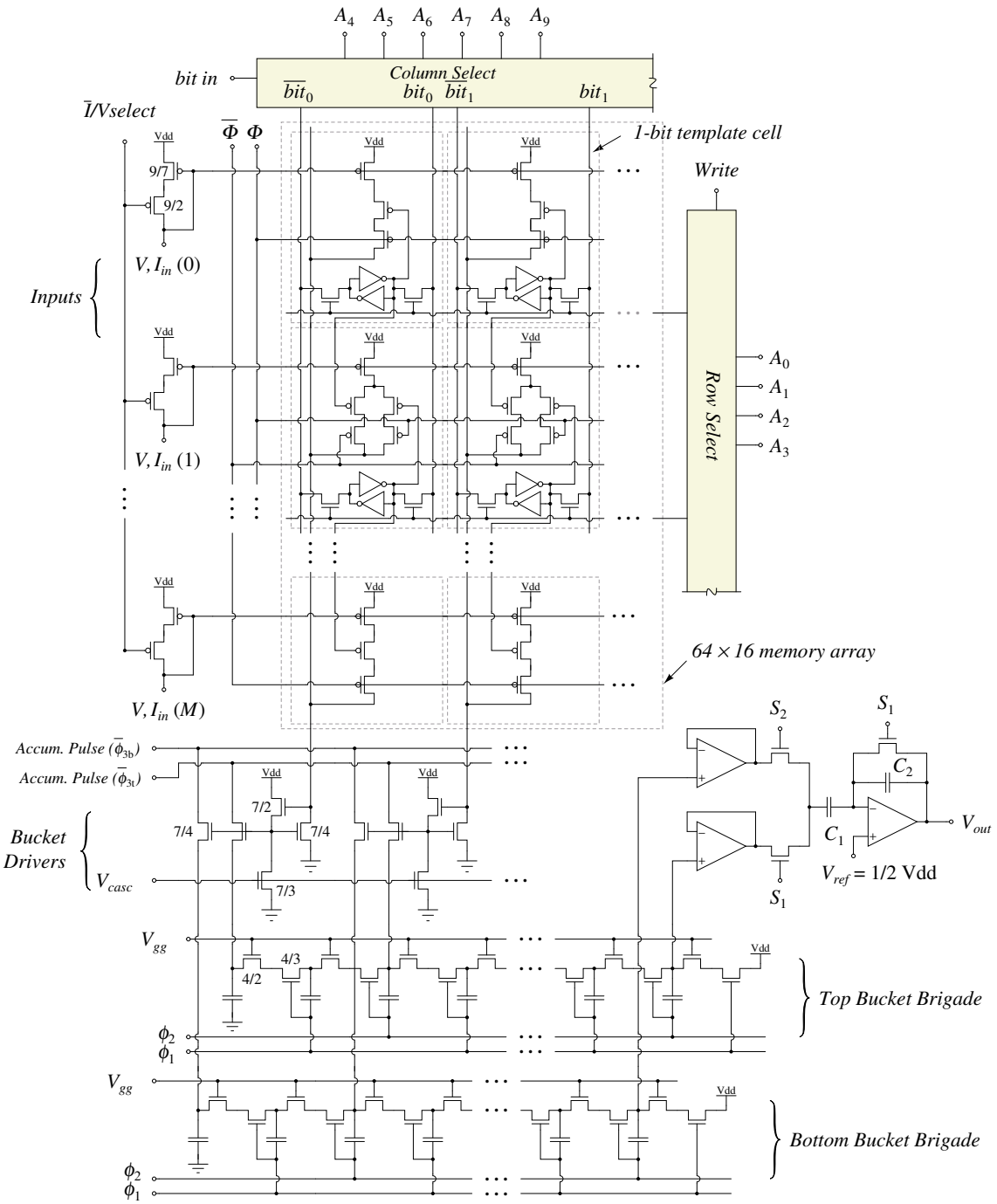

Figure 4.20: The complete correlator array including both bucket brigade devices. Representative MOS device sizes are given as W/L in units of $\lambda = 0.6\,\mu$m.

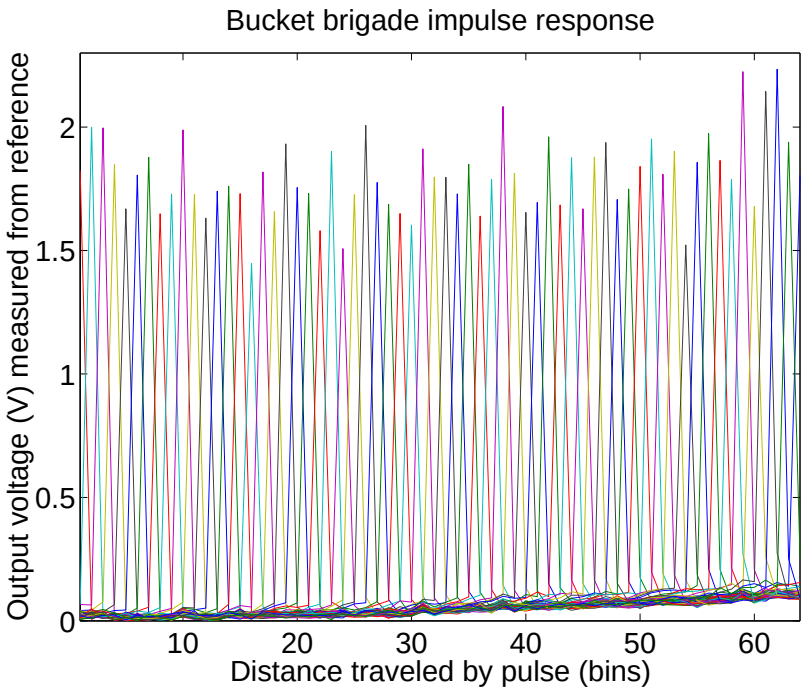

Figure 4.22: Impulse response at each tap of the bucket brigade device.

mismatches vary significantly with current level and thus are impossible to correct to any useful extent. Fortunately, the correlation is a distributed computation by which most of the random variation of individual components gets averaged out: we depend on the massively parallel nature of the correlation to self-correct circuit mismatch.

4.4.5 Experimental Results

We designed and fabricated a chip containing a single template correlator in a standard 1.2µm CMOS technology. The size of the correlator is 700 µm $\times$ 1170 µm. in a 2.2 mm die. Figure 4.25 is a photomicrograph of the integrated circuit. We tested this chip using an experimental setup which implements the frontend system (bandpass filterbank and subsequent rectification and smoothing of the input signal) digitally, on a workstation. This digital signal processing is performed offline, with the resulting sixteen channels of output saved to a file. They are downloaded as needed to a 16-channel current-mode D/A chip, which produces the nanoamp-level currents used by the correlator chip. This setup allows us to evaluate the correlator independently of the frontend filterbank system.

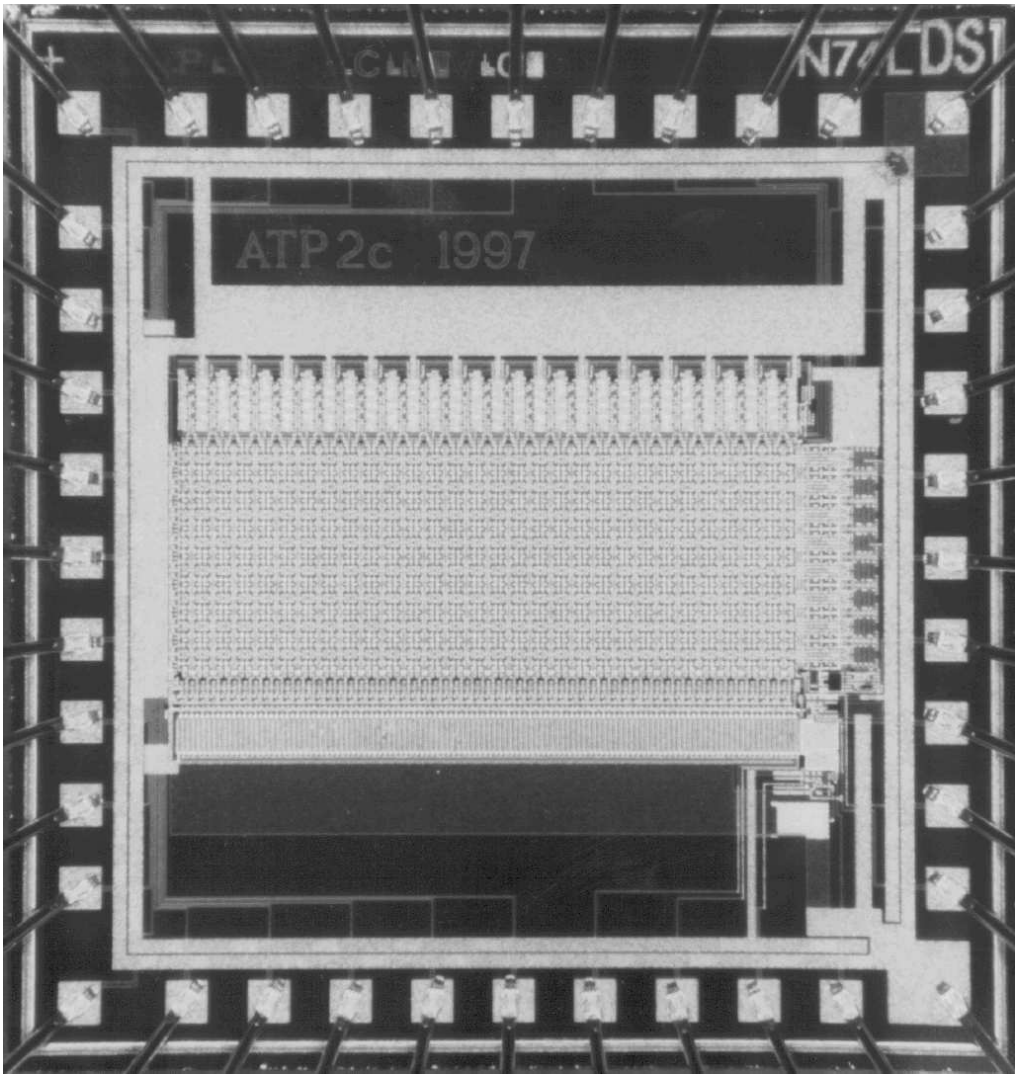

Figure 4.25: Photomicrograph of the acoustic transient correlator, a 2.2 mm $\times$ 2.2 mm die fabricated in a 1.2µm CMOS technology.

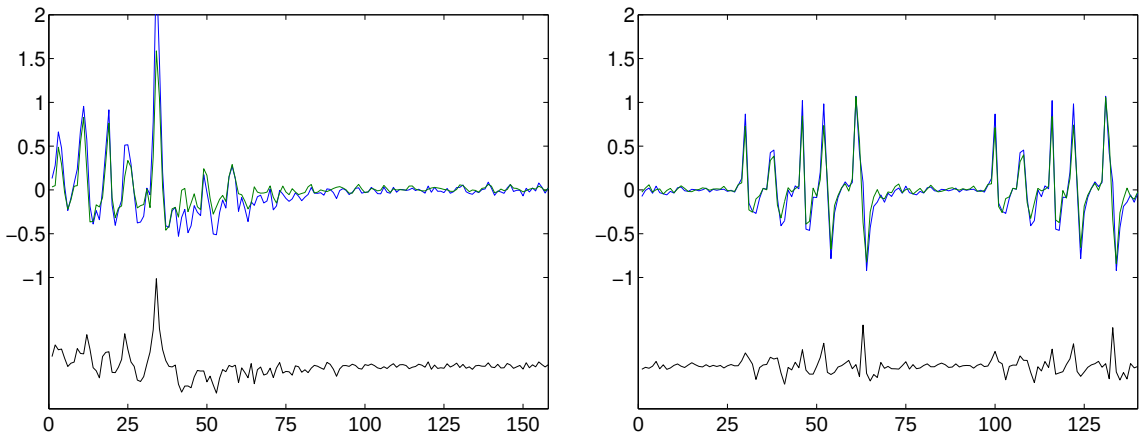

Figure 4.26: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “can” template loaded into chip memory.

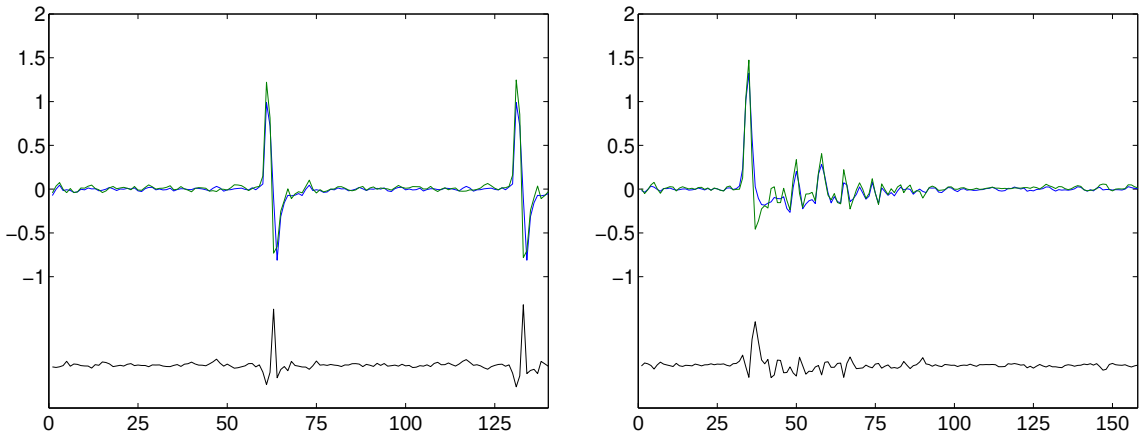

Figure 4.27: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “snap” template loaded into chip memory.

binary-binary correlation resulted in excellent classification accuracy. However, continuous values of the inputs were used to determine the binary templates, even though only binary inputs were presented to the system during the classification test. Alternate methods of template generation are required when a quantized representation of the input class is all that is available during training. The methods used are explained in Section 4.9.

In a system using continuous-valued inputs, channel differencing produces both positive and negative results, and so it is beneficial to commute the difference operation to the end of the system to avoid the necessity of manipulating signed values throughout the system. In a system with binary or trinary inputs, however, signed-value computations are reduced to simple logic operations (see Table 4.8. Thus arithmetic manipulation of signed values is not a problem, and it is desirable to reduce the input channels to binary form as soon as possible in the stages of processing. The channel difference operation optimally should be assumed by the frontend system, computed prior to the correlation summation, thus greatly reducing the complexity of the correlator. The multiplication takes the form of an exclusive-or operation between each input bit and the corresponding template bit for the case of binary-binary correlation. The trinary-trinary case uses the same exclusive-or between the sign bits of the input and template, but uses an "and" operation between the amplitude bits of input and template. This correlator logic is clarified in Table 4.8.

template sign	input sign	accumulate
0	0	$+1$
0	1	-1
1	0	-1
1	1	$+1$

template		input		accumulate
ampl	sign	ampl	sign	
0	X	X	X	0
X	X	0	X	0
1	0	1	0	$+1$
1	0	1	1	-1
1	1	1	0	-1
1	1	1	1	$+1$

Table 4.8: Left: logic for the binary-binary correlation operation. Right: logic for the trinary-trinary correlation. "X" represents a don't-care condition.

A good strategy for the digital system if a custom chip were to be designed and built would be to keep the same parallel column-wise configuration as the mixed-mode design, and to keep the template SRAM cell incorporated as an integral part of the system. A first impression might be to perform the correlation as an addition using adder circuits. In other words, the bucket brigade device would be replaced by a combination adder and shift register. However, multi-bit arithmetic consumes space, and in a layout it is impossible to fit the pitch of the adder/shift register cells to the pitch of the SRAM cells.

the software trials of the ATP, which has a 32-channel input and a 32 $\times$ 128 binary template. An entirely serial operation for a single template requires counting 4096 times within the 2 ms input sample timeframe. If the counter can be run flat-out at one count per clock cycle, a driving clock of only 2 MHz is required. This is slow enough to allow the system definition to be expanded to include trinary operations, so that template and input values are 2 bits wide and take on the effective values $[-1, 0, +1]$. Trinary-valued operations require twice as much memory and twice the clock rate.

The resulting template size of 8192 bits is not chosen arbitrarily, as we consider a design using semicustom FPGA and off-the-shelf parts. A good choice for the FPGA is an Actel 1000-series part, which features one-time-programming (OTP) using fuse-like connections which are activated with externally applied voltages. The cheaper types of FPGAs come with numerous configurable combinatorial and sequential logic modules, but on-board memory is only available through the use of sequential modules as latches or flip-flops. Large blocks of memory are not possible (except on the more expensive FPGAs). However, it is relatively simple to build the state-machine circuitry necessary to interface the FPGA to external memory, particularly as there are plenty of input/output ports available for address and data buses. SRAM chips come in a small number of standard sizes, one of which is 8K $\times$ 8 (that is, 13 address bits ($2^{13} = 8192$) and 8 data bits), a configuration which allows 8 templates to be stored and read in parallel. In this configuration, rather than treating the 8-bit data bus as a single byte, we treat the bus as 8 separate 1-bit values. There are, of course, alternate ways to arrange the templates in memory, but having all templates addressed in parallel is the most efficient for several reasons which should be obvious from the following discussion.

The next step is to determine how to deal with stored values during the course of the correlation computation. The system has to compute a correlation based on the current and past values of the input, so either it must store the input values over the time span of the template, or it must store intermediate column results in a pipelined manner, as described above for the custom VLSI architecture. Storing the input requires 32 bits $\times$ 128 samples. On the other hand, storing intermediate values requires only 13 bits $\times$ 128 columns (13 bits are required instead of 12 due to the trinary operations: the total range of the output is from -4096 to $+4095$, which is a 13-bit result).[4]

[4]This value is an upper limit. Fewer bits are required if a convenient way can be found to pack the results, since the first column result requires only 6 bits, with the number of bits growing as the log of the column number as noted above. In practice, packing the columns is a procedure too complicated to be worthwhile using in a serial implementation.

up. Template values are counted in blocks of 64 (32 channels $\times$ 2 bits per template value) starting at SRAM address zero and counting up. Intermediate column sums are counted in blocks of 16, starting at SRAM address 8191 and counting down. The two values are easily obtained by inverting and shifting the output of the seven-bit counter and multiplexing, as shown in Figure 4.30. Additionally, the column address to write to is easily obtained by latching the seven-bit counter output before each new count.

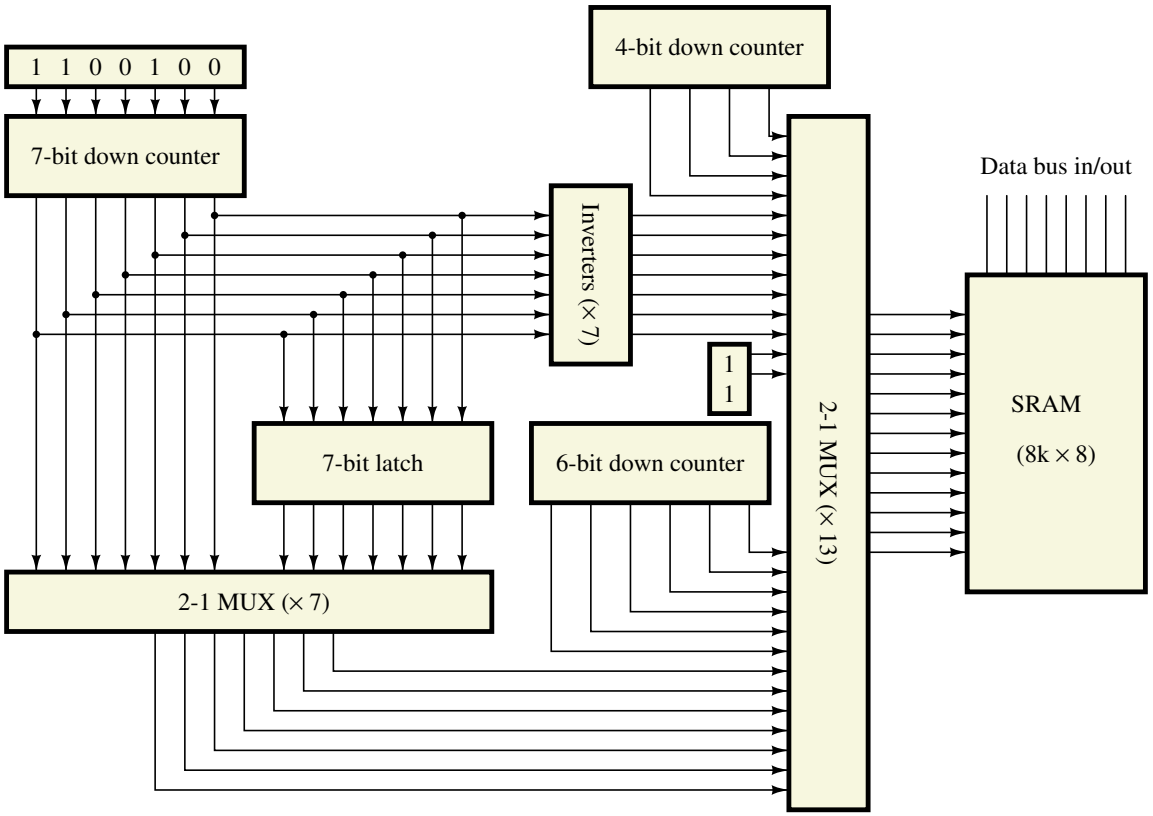

Figure 4.30: SRAM address generator for the digital correlator.

A simple finite state machine can determine the correct cycle of addressing, reading, and writing the SRAM. The state diagram of this machine is shown in Figure 4.31. It requires only 4 state variables and can be easily designed by hand in one evening. The complexity of the FSM is enough to prohibit the use of gate-level IC components. However, it is just about the right size for a semicustom FPGA part such as an Actel 1020, which at the time of writing could be bought for under $15 each. The digital architecture was designed with cost and time of development as the primary constraints. To keep the costs down, the design was split into three parts: An SRAM for

sum, which holds the correlation result, is shifted out MSB-first for writing to SRAM. At that point, each of the eight correlations can be compared one bit at a time from the most significant bit to the least significant bit in order to select the maximum (there can be more than one winner if several templates produce the same result). A threshold value can be injected into the system serially, as if it were a ninth correlation output. Because this particular output selection mechanism is kept simple on purpose in order to fit on a single chip, there is only one global threshold rather than a threshold for each channel. The global threshold value is presented in parallel to the controller chip, which broadcasts the value serially to the correlator chip at the appropriate time on each input cycle. This setup preserves the scalability of the architecture, in which a single controller chip can service multiple correlator chips. Each correlator chip sends and receives one bit of information necessary to spread the maximum-finding calculation over all the correlator chips in the system in order to produce a global maximum.

Actel parts are able to drive moderate loads such as LEDs. The output maximizing circuit generates separate outputs for each template in the system, which can in turn drive an LED bar-graph array for a demonstration system capable of lighting a specific LED in response to one of several input transients.

The printed circuit board developed for this application (Figure 4.33) includes space for one controller FPGA, two correlator FPGAs, an interface to the frontend system which provides the inputs, and an interface to a digital I/O card on a computer through which the template values can be programmed. Fully populated, the board can simultaneously compute 16 template correlations at 1 ms intervals.

4.8 The Switch-Capacitor Frontend

The constraints on the architecture of the frontend system are somewhat looser with the digital ATP used as a backend instead of the analog correlator. For instance, the frontend does not need to compute the L-1 normalization, as mentioned above in Section 4.5. The rest of the system remains the same, and the output requires comparators to perform the pairwise channel differences and produce the binary result.

The custom analog VLSI frontend system had several problems, the main one being the factor of (up to) three difference in gain across channels. Unfortunately, large component mismatch is a fact of life with MOSIS processes such as the AMI 1.2 µm process used to fabricate the frontend chips, and heightened sensitivity to mismatch is a fact of life with high-Q systems. And although

some gain mismatch can be tolerated by the transient classification algorithm, mismatch of the sort encountered in our fabricated versions of the log-domain circuit cannot (without adaptation or other corrective circuitry).

I investigated a board-level design using switched capacitor filters to generate real-time frequency decompositions of an audio or computer-generated input. This design effort was in part to explore a different frontend architecture for use with the digital correlator, and partly for comparison with the custom integrated circuit log-domain frontend. Of particular interest is how good the matching between channels is for the switched capacitor system, and whether the gain in performance is worth the tradeoff in integration density and power consumption. Like the design of the correlator, a custom VLSI analog integrated circuit would ultimately achieve much better power efficiency and area efficiency, but a board-level system is an excellent demonstration of the concept which can be developed in short time at relatively low cost. The cost of the component-level system is, in fact, about the same as a small custom chip ($4\,\mathrm{mm}^2$) fabrication (through a cost-efficient, shared process such as MOSIS). The bandpass filter configuration of the log-domain chip can be efficiently reproduced in a commercially available dual general-purpose second-order switched capacitor filter IC. Each of the two filters on each chip can be wired as a bandpass filter. The first section uses 3 external resistors to form a fixed-Q ($Q = 10$) bandpass filter with a gain equal (or proportional) to the Q and a center frequency determined solely by the clock frequency. The second section uses 2 external resistors to form a bandpass filter of $Q = 1$ with a gain of 1 and center frequency equal to that of the first bandpass filter. The second filter uses the mode 1A configuration, which yields two bandpass outputs, one inverted, one not. When $Q = 1$, these two outputs are of equal magnitude. These symmetric outputs can be combined with a comparator and a multiplexer which passes the maximum function to its output, thus realizing a full-wave signal rectifier. Another switched capacitor filter chip, a 4th-order Butterworth lowpass, smooths the rectified output. Two more comparators complete the channel: One computes the difference with the neighboring channel, and the other computes the difference with a global threshold. The thresholding comparison allows the signal input to have trinary values of two bits each encoding values of $[-1, 0, +1]$. I designate these two bits "sign" and "amplitude", with the trinary encoding shown in Table 4.9. The digital semicustom system described in the previous section is designed specifically to allow any combination of trinary and binary correlations.

There are two choices for clocking the bandpass filters. One is to place the burden of determining the center frequencies on the filter design, by using mode 2 or mode 3 filters, whose center frequency can be adjusted relative to the clock frequency. That way, only one master clock

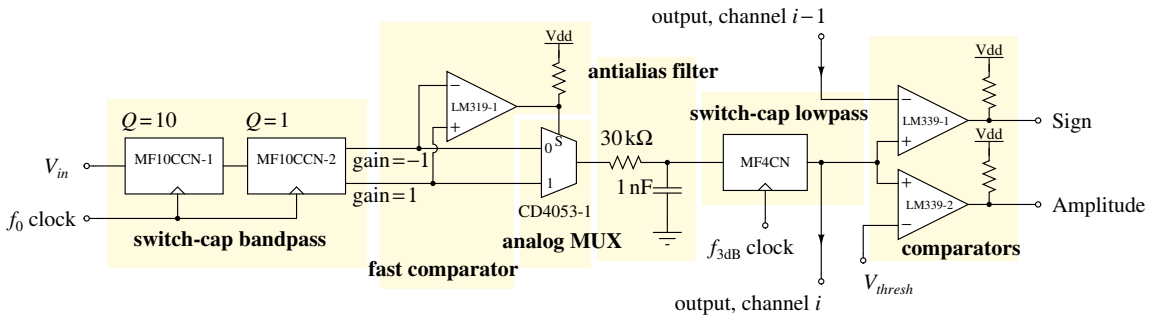

Figure 4.34: Switched-capacitor bandpass filterbank, single channel.

analog outputs which feed the audio preamplifier in similar arrangement to the analog microphone input.

The digital correlator system provides handshaking signals necessary to keep the correlator and computer in synchrony; otherwise, using microphone or other real-time input, the handshaking must be generated on the frontend side with a 1 ms-period clock (generated from the 20 MHz master clock using four four-bit decade counters in series). This setup assumes that the correlator system is clocked fast enough to keep up with the input, so that the "wait-on-input" handshaking signal provided by the correlator is not required.

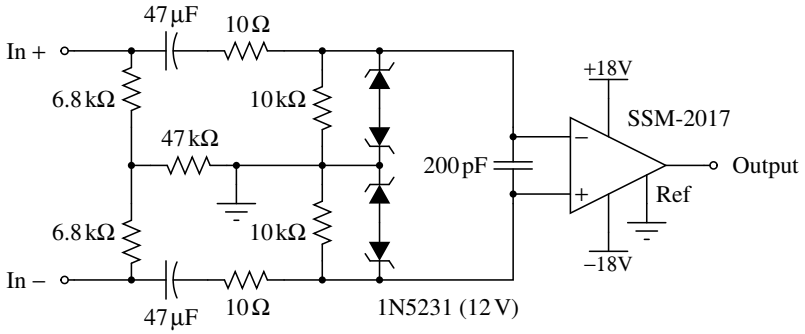

Figure 4.35: Audio input circuit for the frontend system.

A reasonably-sized printed circuit board can handle around 16 channels, so the system was designed for two boards, each of which produces half of the 32 outputs, and which work together such that only one board has an input section, one board contains the master 20 MHz clock, one board contains the final 4-to-1 multiplexer that communicates directly with the correlator system, and one board contains a 12-bit D/A converter (Maxim MAX530) generating the bias voltage for determining the threshold at which output signal level causes the amplitude bit to change state.

which the output for the target class, though strong and persistant at and near the point of best alignment, is accompanied by spurious false classifications away from the alignment point. These false classifications can be ignored only if the segmentation algorithm correctly identifies the presence of a transient event and predicts the correct time at which to accept the classification result. Making the correlation output temporally shift-invariant is the purpose of some of the training methods discussed in Chapter 5, such as Independent Component Analysis, Support Vector Machines, and Perceptron Neural Networks. Any system which assumes exact alignment of the input and template requires a robust segmentation algorithm to predict the best point of alignment. Software trials of the continuous-valued input, binary-valued template architecture used a segmentation scheme amenable to analog hardware, shown in Figure 4.37. The raw output of all channels of the frontend system (after normalization) were thresholded, summed together, and binarized by another threshold, resulting in a "noisy" segmentation. The noisy segmentation result was filtered in parallel by two lowpass functions with time constants of 1 ms and 10 ms, respectively, thresholded once more, and combined with an OR operation. The longer time constant ensured a clean segmentation for the duration of the input transient, while the shorter time constant ensured a quick reponse of the system to the onset of a transient event.

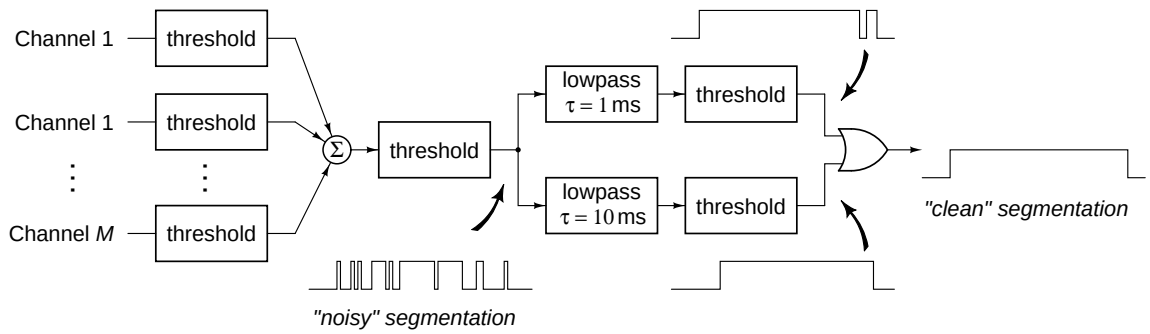

Figure 4.37: A mixed-signal method for detecting the onset of a transient event from the output of an analog frontend filterbank.

For the trinary-trinary correlator, it was desirable to have a similarly simple algorithm for segmentation, but one which could operate on the trinary frontend system output using purely logical operations in keeping with the rest of the correlator system rather than relying on analog computation. An algorithm which proved adequate to the task is the finite state machine shown in Figure 4.38. This is essentially a binary version of the original segmentation algorithm. The first threshold function is taken care of by the frontend system itself in the production of the amplitude bit on each channel. The segmenter sums the amplitude bits across all channels, producing a value

2. Threshold the template values using the relation

$$p[n,m] = \begin{cases} 1 & \text{if } p[n,m] > 0 \\ -1 & \text{if } p[n,m] < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.27}$$

4.9.3 Experimental Results

The performance of the trinary-trinary correlator was confirmed using the same cross-validation test as for the original software simulations of the algorithm, except that the hardware itself performed the frontend time-frequency decomposition and the correlation summation. Templates values were determined offline using data obtained from the frontend hardware. Figure 4.39 shows the templates derived from frontend system data for one step of the cross-validation test.

Figures 4.40 and 4.41 show example outputs of the frontend system, the correlator, and the segmenter at two times during the cross-validation test. For these four input transients, the correlator performed correct classification. The threshold value which the frontend system uses to determine the amplitude bit for each channel output is a degree of freedom affecting system performance in trinary-trinary correlation tasks (in binary-binary correlation, the amplitude bit is ignored, so the threshold has no effect on system performance). If the threshold is too high, then the trinary frontend output values become zero always, and the correlation fails, setting a clear upper limit on the optimal threshold value. If the threshold is zero, then the frontend output becomes the same as for the binary case. It is not obvious that this should degrade performance, particularly since in the original software trials binary-binary correlation performance was very close to that for the baseline algorithm. In practice, however, averaging the binary training data produces templates inferior to those produced using continous-valued training data, and performance suffers. The optimal threshold voltage is only meaningful in the context of input transduction, audio preamplifier gain, and bandpass filter gain, but may be determined empirically from correlation performance under conditions of different threshold voltage.

The remaining tables (4.10 through 4.12) represent the accumulated statistics from several runs of the cross-validation test. The first, Table 4.10, is a simple test of the performance of the system in the absense of a real-time segmenter, in which the correlation value is accepted at the time of best known alignment of each input transient. The result represents the best performance the system can achieve in an off-line recognition task, which is 99.0%.

Table 4.11 shows the result from the same task repeated using the finite state machine

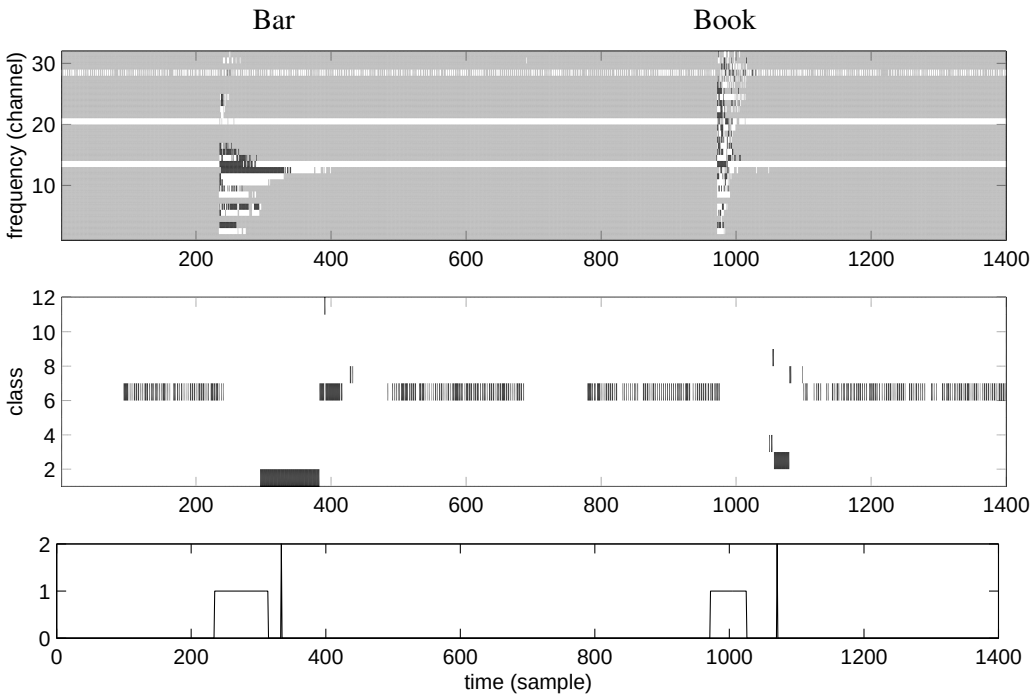

Figure 4.40: System input and output for sound "bar" (left) and "book" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub
Bar	20	0	0	0	0	0	0	0	0	0
Book	0	20	0	0	0	0	0	0	0	0
Can	0	0	20	0	0	0	0	0	0	0
Dink	0	0	0	20	0	0	0	0	0	0
Door	0	0	0	0	20	0	0	0	0	0
Finger	0	0	0	0	0	20	0	0	0	0
Hand	0	0	0	0	0	1	18	0	0	0
Mallet	0	0	1	0	0	0	0	19	0	0
Shelf	0	0	0	0	0	0	0	0	20	0
Tub	0	0	0	0	0	0	0	0	0	20

Total instances presented:	199
Correct:	197
Incorrect:	2
Accuracy:	99.0%

Table 4.10: Offline recognition task: Cross-validation on the transient dataset using a non-real-time segmenter.

segmenter shown in Figure 4.38. The main loss in performance is due to the detection of transients which are not part of the input data set classes. Inspection of the original recordings revealed that these "spurious" transients are at least as loud as some of the actual transient class examples, and so the segmenter should be expected to detect them and consider them to be transients in the input. The majority of these transients belonged to the recording "door" and correspond to the sound of the door being opened (however quietly) between each target instance of the door being closed. The spurious transient events were hand-tagged as new transient classes labeled "unknown," and the cross-correlation task was performed again. The result of hand-tagging the data, shown in Table 4.12, is over 95% accuracy. Apparently some of the spurious events were indistinguishable from handclaps, as shown by the mistaken identity of all three examples of one "unknown" class. Problems like this account for the relatively low recognition rate (96.4%) of the baseline algorithm, which otherwise should be guaranteed virtually perfect results on linearly separable data. The cross-validation tests results confirm that the simple trinary-trinary correlation method approaches the baseline algorithm in accuracy, even when only trinary-valued time-frequency decompositions of the audio input are available for training, and when a simple finite state machine consisting of a few small binary counters and assorted logic is used to segment the input.

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub	?	?
Bar	20	0	0	0	0	0	0	0	0	0	0	0
Book	0	20	0	0	0	0	0	0	0	0	0	0
Can	0	0	24	0	0	0	0	0	0	0	0	0
Dink	0	0	0	20	0	0	0	0	0	0	0	0
Door	0	0	0	0	20	0	0	0	0	0	0	0
Finger	0	0	0	0	0	17	1	0	0	0	0	2
Hand	0	0	0	0	0	1	19	0	0	0	0	0
Mallet	0	0	2	0	0	0	0	18	0	0	0	0
Shelf	0	0	0	0	0	0	0	0	20	0	0	0
Tub	0	0	0	0	0	0	0	0	0	20	0	0
?	0	0	0	0	0	3	0	0	0	0	0	0
?	0	0	0	0	0	0	0	0	0	0	0	7

Total instances presented:	215
Total events found:	214
Correct:	205
Incorrect:	9
Missed:	1
Correctly identified:	95.3%
(In set of known targets):	97.2%
Incorrectly identified:	4.2%
Missed:	0.5%

Table 4.12: Real-time recognition task: Cross-validation on the transient dataset using the simple finite state machine segmenter. In this transient set, spurious transient events were hand-tagged as "unknown" classes (denoted by a question mark).

decreasing the likelihood of a false identification of the second class. Thus, the template may not look anything like the decomposed class examples against which it is correlated, but produces an output which is much larger than that for any other class input. Class averaging requires the least amount of training, and can be done with only the frontend filterbank system as part of the training system. Once the idea of simple class averages is abandoned and a more robust solution sought on the basis of standard pattern recognition, a number of possibilities arise and must be investigated for their applicability to the transient classification problem. This chapter provides an overview of what training methods are available and how they may be applied to the acoustic transient processor.

5.2 Average-Value Templates

The first interpretation of the form of the template, that of class average, is the one that I have been assuming throughout the text to this point. Average-value template learning computes the baseline template p_z for class z as

$$p_z(n,m) = \frac{1}{K}\sum_{k=1}^{K} y_{z,k}(t_0 - n, m) \qquad \forall n, m \tag{5.1}$$

where the training set contains K examples of class z. The examples are distinct and their boundaries known, and they are aligned to some time t_0 relative to the start of the event. The values $y_{z,k}$ are the response of the frontend filterbank to the k-th example of class z. This is at least a simple method to implement. The main difficulty in implementation arises from the need to accurately segment each example of the training set used to generate the template. The average of examples which are not properly aligned quickly becomes flat and featureless. Unfortunately, all acoustic signals have some temporal variation, and so the average value will always become increasingly flat and featureless away from the point of alignment.

To give a concrete example: The sound of a door closing (one of the transient classes in our recorded set) consists of two separate events: The sound of the door latch striking the metal plate, followed by the sound of the door itself bumping against the frame. The result of the two sounds occurring in sequence is the familiar "ker-chunk" of a closing door. Either of the two sounds, if separated from the other, fail to sound like a door. Clearly the sound that we recognize as that of a door closing is as much dependent on the juxtaposition of these two events as it is on the events themselves. On the other hand, the timing between the two events depends on the radial velocity of the door, which may vary considerably from one event to the next. Suppose the recorded transient examples are in every way exact except for the distance between the door latch and door

like a discrete wavelet transform where information is actively pushed from one dilation into the next, but where each filtering function is optimized for a particular transient event at each dilation (timescale). Those functions may be approximated by template correlators to give a high level of flexibility to function selection, but the templates no longer encode any function as simple as an ensemble average. What they encode are functions which extract features at each dilation which, collectively over all dilations, optimally isolate one acoustic class from all other classes.

Clearly, capturing the nuances of composite acoustic events requires that composite sounds be broken down into their fundamental, unchanging parts at each timescale. The template correlation should be affected by temporal shifts as little as possible, and probabilities between the timing of fundamental events probably should be dealt with separately, in a hierarchical manner, by using the outputs of the current timescale as the input to the next timescale. The largest timescale has Z templates (one for each class) and ideally separates the acoustic classes in a robust manner. The outputs of all timescales up to the last are effectively "hidden nodes" whose outputs do not necessarily have an obvious physical meaning. They are like hidden nodes in a neural network, and may be treated in a similar manner.

5.3 Deterministic Methods: Statistical Component Analysis

A major difficulty with finding the fundamental components of composite acoustic events lies in the fact that the fundamental transient parts are not necessarily separable. Training a network to learn to separate classes is made difficult when the training examples cannot be isolated from one another. A related problem is perhaps the most problematic of all for ensemble average training: the system is trained to recognize not only features which are unique to each transient class, but also features which are common to many or all of the transient classes. Unfortunately, these common features can dominate the structure of the sound input. Most transients are characterized by a sharp rise in total energy followed by a slower decay back to zero. Proper training should suppress the similarities and accentuate the differences between transient classes.

Fortunately, the techniques of *Principle Component Analysis* (PCA) and *Independent Component Analysis* (ICA) address some of these problems. They provide a mathematical framework for addressing the issues mentioned above. ICA operates under the assumption that every signal is formed of statistically independent parts which have been put together in various weighted distributions. The ICA algorithm determines what are the independent components of its input (after presentation of many training examples, of course), but its *inverse* is a *detector* of those fea-

In a more realistic case, transient events are themselves composed of other independent sources. As mentioned in the previous section, many transients have similar structures. Roughly speaking, the similar structures are independent components which are shared between transient classes, and there are other independent components which correspond to structures unique to certain classes. Each class produces a unique output u representing the unique mixture of independent sources which encode that acoustic transient class.

The acoustic transient processor baseline algorithm (4.1) is a linear matrix multiplication and so is directly applicable to ICA. The matrix **W** is the set of templates p, where each template is a one-dimensional vector of values, ignoring the relationships between the two dimensions of time and frequency. From the discussion above, it should be clear that in an ICA framework each template p encodes an independent component of the source, and does not encode a single class discriminator function unless the class happens to be representable by a single independent component (which would be extremely rare in practice). The output c of all the template correlators encodes the instantaneous mixtures of independent components present in the input.

It should be noted that standard ICA theory assumes a square matrix **W**; that is, the assumption is that there are as many independent sources as there are weights in the system (for each template). However, the final weight update rule derived does not require a square matrix **W**. Indeed, logically the size of **W** should correspond exactly to the true number of independent components of the source. When there are more weights than sources, the encoding is redundant but should have the positive effect of increasing system robustness in the presence of noise; the matrix **A** may be recovered if necessary using the matrix pseudo-inverse.

The weight update rule is [72]:

$$\Delta w_{ij} = \mu w_{ij} + 2 \tanh(u_i) \sum_k w_{kj} u_k \tag{5.5}$$

where μ is a learning constant. The weight update rule is similar to Hebbian learning.

Applied directly to the transient test data, this rule does an excellent job in breaking up the input into distinct output patterns. There are several major problems with the application of the ICA technique, however:

1. Equation (5.5) is an *unsupervised learning* rule. Because the true independent components of the input are unknown, knowing the class of the input is unhelpful to the algorithm. The problem of classification learning is really deferred, not solved. ICA can be viewed as a way to warp the input space into a form more amenable to simple classification. When there are

a subset. The main difference is that SVM theory formulates a *deterministic* algorithm for learning based on Lagrangian multipliers. Solutions are iterative (because large nonlinear networks rarely have symbolic solutions) but not haphazard: quadratic programming methods can find the optimal separating hyperplanes for any classification task, given properly tagged training data. As usual, of course, it is necessary to choose the optimal complexity of the feedforward classifier such that it generalizes well and does not over-fit the data, but the ease of quadratic programming allows one to train and test networks of different complexity and determine exactly what system complexity yields the best generalization in a purely deterministic sense.

There are only two failures of standard SVM to meet the needs of the acoustic transient processor, and they are the same ones encountered with Independent Component Analysis: The temporal nature of the problem which gives rise to a high degree of correlation between successive values of the outputs in time, and the binary (or trinary) nature of the template values which plays havoc with learning algorithms due to the presence of discontinuous derivatives. Hopefully, extensions of SVM exist which can include both these cases. Also, SVM training uses a model of the feed-forward classification system and should be reformulated to allow the physical system to be its own model (as with "chip in the loop" learning) to account for properties of the real system which may not be captured in a simple model.

5.5 Heuristic Methods (Unnikrishnan/Hopfield)

Speech signals are quite complicated and contain transients as well as long-term (vowel) events of variable length. Successful recognition of transients can be viewed as a step towards recognition of continuous speech. Another important step is the detection of vowel formants. When separated from the more general task of speech recognition, it is a relatively easy problem, particularly when the vocabulary is small. The task of digit recognition provides a useful small-vocabulary system which has been used often as a benchmark for speech recognition systems. One system optimized to the task of continuous-speech digit recognition is the architecture by Hopfield, Tank, and Unnikrishnan [68, 69]. The architecture of [68] achieves a 99.3% recognition rate on continuous spoken digits when trained for a single speaker, and a similar architecture reported in [69] achieves 97.5% in a speaker-independent task (male speakers only). These figures are for the author's own recorded database, which is similar to the industry standard *TIDIGITS*, a database of both isolated and continuously-spoken digits, using male, female, and child speakers from around the country.

The authors claim that their architecture was designed with analog implementation in

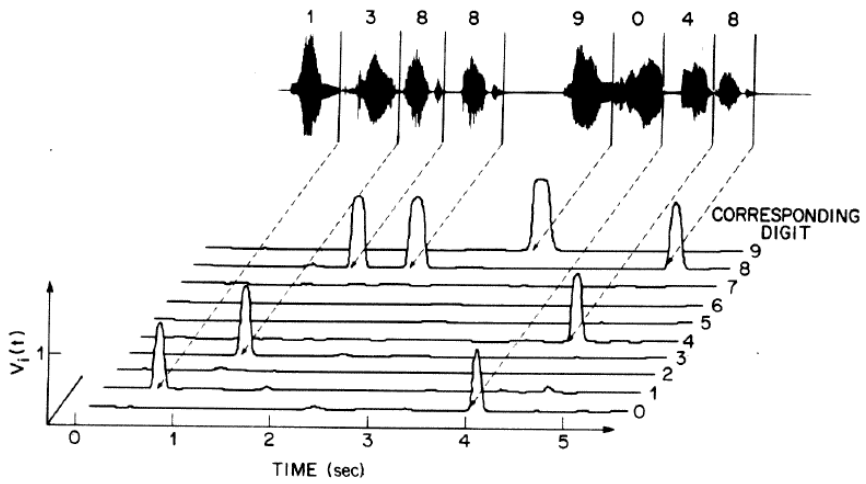

Figure 5.2: Unnikrishnan *et al.* correlator simulation results on isolated digit recognition.

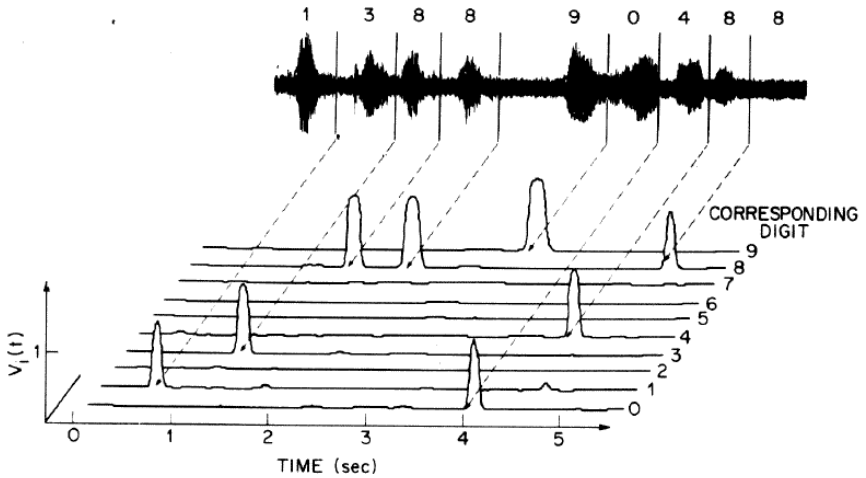

Figure 5.3: Response of architecture in simulation to the same input as Figure 5.2 but with white noise added to the input. The system remains robust in the presence of noise.

step may be regarded as an improvement or not depending on the capabilities of the hardware. The tapped-delay model of Unnikrishnan et al. requires much less storage per template, but on the other hand it depends heavily on the Gaussian multiplications. The authors envision the Gaussian kernel multiplications as diffusion profiles in an analog circuit; however, the model shows diffusion occurring both directions in time, an impossibility for a circuit implemented as a physically diffuse medium (such as the bucket brigade). We have emphasized throughout the thesis that analog multiplication is difficult and imprecise. The architecture chosen for the acoustic transient processor is one which avoids the problem of analog multiplication and it is not unreasonable to assume that the above equation, which casts the Unnikrishnan *et al*. model into the framework of the acoustic transient processor, is a reasonable compromise between system size and accuracy. It remains to be investigated whether the performance of this algorithm remains high if the same manipulations applied to the ATP algorithm are applied to it; namely, if the weights take binary values instead of, or in addition to, the input. If so, then an ATP-like architecture can be used to implement and test the algorithm.

The scheme used by the Unnikrishnan *et al.* architecture for training the system lies somewhere between the simplicity of ensemble averaging and the complexity of independent component analysis. Segmentation of each input example is performed by hand, but the system learns the correct position of the segmentation through a bootstrapping method. Segmentation is defined as the point near or at the end of the spoken word at which the detector should signal a correct classification. The set of weights for each word recognizer is updated with a gradient-descent equation (perceptron rule), where the word recognizer matching the training word presented is updated to increase its output at the segmentation point, while the word recognizers for all other words are updated to decrease their outputs at the segmentation point and also at other fixed intervals during the presentation of the input training example. When the training is near completion, the system can use its own output to determine the segmentation point, thus allowing it to overcome small errors in the estimated segmentation. The negative training on out-of-class weights helps the system isolate those parts of each class which are different from all the others, unlike ensemble averaging which will additionally encode those features which are common to all transient classes.

Because spoken digits are not transients, they have timespans on the order of half a second or longer, requiring a delay length (and template size) more than four times that of the ATP. This is because the Unnikrishnan *et al.* architecture is not hierarchical. It is not even phonetic, which is a simple form of hierarchy used by many systems. If a phoneme occurs twice in the same word, or if the same phoneme is shared among several words, the stored weights encoding the recognizer

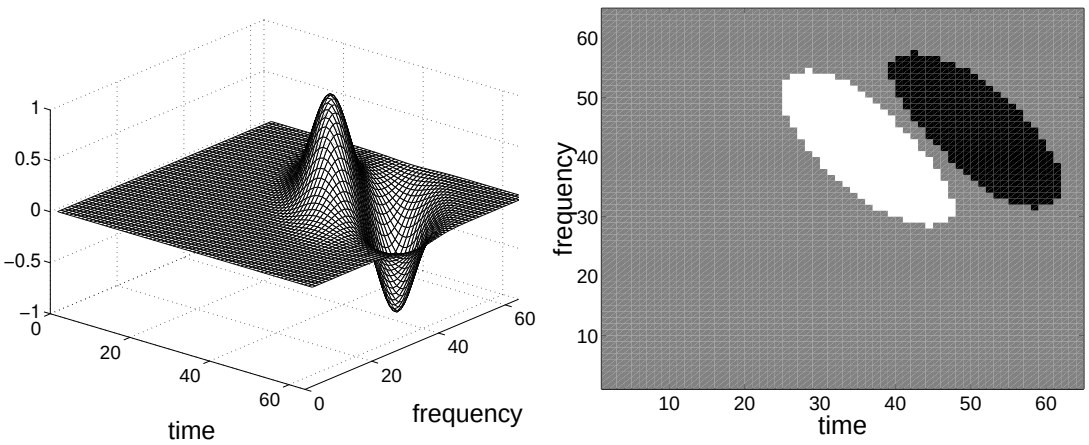

Figure 5.5: Example of a biologically-inspired template encoding a falling tone of specific frequency and rate. *left:* Continuous-valued encoding. *right:* Trinary encoding.

kind of instantaneous spectral information. The Gaussian is defined by the equation

$$\frac{1}{2\pi\sqrt{|U|}}\exp\left((\mathbf{x}-\mathbf{x}_0)U(\mathbf{x}-\mathbf{x}_0)^T\right) \tag{5.10}$$

where $\mathbf{x}$ is a 2-dimensional vector in time and frequency, $\mathbf{x}_0$ is the distribution center, and U is a covariance matrix

$$U = \begin{bmatrix} (\sigma_t)^2 & \rho \\ \rho & (\sigma_f)^2 \end{bmatrix} \tag{5.11}$$

having components for variance in time σ_t, in frequency σ_f, and the covariance coefficient ρ, which effectively represents a rotation of the distribution.

The acoustic transient architecture algorithm provides a way to actively test theories associated with the observations from the auditory cortex. In particular, the trinary encoding of the digital transient classifier hardware can encode regions of positive and negative response inside a region of zero response, as shown on the right-hand side of Figure 5.5. This is essentially an attempt to map the correlation algorithm onto a neuron, where each location of the template maps more or less directly to a synapse connection between the delayed output of the cochlear frequency mapping and the neuron under study in the auditory cortex. The neuron is more efficient than the correlator because it presumably uses no more synaptic connections than necessary for each response. It also has a higher resolution than the correlator because the synaptic weights encode continuous-valued weights: a neat little trick of nature that is, as noted in previous chapters, rather difficult to duplicate in silicon. Also, the neurons of the brain make optimally efficient use of the nonlinearities inherent

Conclusion

This thesis has sought to portray analog VLSI as a medium for computation and a viable alternative to digital signal processing. These aspects of analog design were taken for granted many years ago before the start of the so-called "computer revolution," but today require justification in an increasingly high-speed digital world.

Our use of the Analog VLSI medium concentrates on acoustic signal processing. We show how efficient use of mixed-mode processing, translinear circuit design, and other techniques can lead to compact, power-efficient systems which can rival DSPs in many aspects of performance. What we espouse is a view of system design which starts at the level of algorithms and looks at all ways, both analog, digital, or mixtures of both, which can best implement a specific system under the constraint of limited resources, namely, size and power. I believe that there is still a need for compact, power-efficient analog computing systems even in a primarily digital world. The conflicting demands of energy efficiency and raw processing power must be satisfied, and so system design must avoid many of the pitfalls of brute-force digital design and achieve ever greater levels of finesse from devices to circuits to algorithms and systems.

In this thesis, we have demonstrated the application of efficient analog and mixed-signal design to several acoustic signal processing systems, starting with ways of perceiving and mapping the time-frequency domain. Throughout the thesis, we have shown how each system deals with this mapping, from wavelet transforms to filterbanks and pattern classification systems. We have "zoomed in" from this view of signal processing to look at specific circuits for implementing time-frequency transformations of acoustic inputs and acoustic pattern recognition and classification. And we have "zoomed out" from this view to look at the larger picture of learning machines, speech recognition, and biological models of neural processing. We could continue on in either direction, from transistor physics to the human brain, because time-frequency mapping of acoustic data occurs ubiquitously in both nature and in engineering. At this important juncture of circuits, acoustics, biology, and machine and human learning, we hope we have provided some valuable insights and pointed the way to better engineering design.

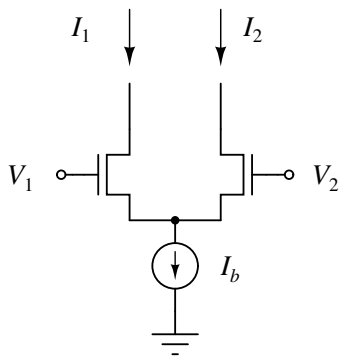

Figure A.1: Simple differential pair transconductance amplifier.

which can be combined to eliminate the common source voltage V_s results in an expression for the output:

$$\Delta I = I_b \left(\frac{e^{(\kappa V_2)/V_t} - e^{(\kappa V_1)/V_t}}{e^{(\kappa V_2)/V_t} + e^{(\kappa V_1)/V_t}} \right). \tag{A.4}$$

As a function of differential input voltage $\Delta V = V_2 - V_1$,

$$\Delta I = I_b \left(\frac{1 - e^{-(\kappa \Delta V)/V_t}}{1 + e^{-(\kappa \Delta V)/V_t}} \right). \tag{A.5}$$

This is a sigmoidal function, with odd symmetry about $\Delta V = 0$, with asymptotes at $\pm I_b$. To find linear error, first find the linear fit to this function at the origin:

$$\left. \frac{\partial \Delta I}{\partial \Delta V} \right|_{\Delta V = 0} = \frac{1}{2} \frac{\kappa}{V_t} I_b \tag{A.6}$$

such that a linear approximation to the output is

$$I_{lin} = \left(\frac{1}{2} \frac{\kappa}{V_t} I_b \right) \Delta V. \tag{A.7}$$

Linear error is

$$\frac{I_{lin} - \Delta I}{\Delta I} = \frac{\kappa \Delta V}{2 V_t} \left(\frac{1 + e^{-(\kappa \Delta V)/V_t}}{1 - e^{-(\kappa \Delta V)/V_t}} \right) - 1. \tag{A.8}$$

This expression is independent of I_b. It is plotted in Figure A.2. From the plot it can be seen that the function is within 10% linear to about $\pm 50\,\mathrm{mV}$. By approximating the exponents in the above equation, it can be shown that this is a few V_t not by coincidence, but by physical character.

Appendix B

MATLAB Code for Sine Sequence Generation

```
%-------------------------------------------------------------
% sequence.m
%-------------------------------------------------------------
%   Searches for optimal single-bit string approximating a
%   sinusoid.  Algorithm is a stochastic partial search, in
%   blocks of consecutive bits.
%
%   Original Pascal program by Gert Cauwenberghs, 1994
%   Converted to Matlab by Tim Edwards 1998
%-------------------------------------------------------------

%-------------------------------------------------------------
% Settings for this run
%-------------------------------------------------------------

period = 256;   % should be power of 2, range from 4 to 2048
nOpt = 2;       % block length optimized per iteration; <= 16
cap = 15;       % capacitance ratio of filter, Cmax / Cmin
stages = 3;     % number of cascaded 1st-order filter stages

%-------------------------------------------------------------
% Create the lowpass filter function
% This generates attenuation factors in the frequency domain
% for each fft bin
%-------------------------------------------------------------

norm = 2 * pi / period;
```

```
      for pattern = 1: maxPattern,

         % Generate the perturbed bit sequence
75
         bitsP = bits;

         mask = 1;
         for i = 1: nOpt,
80          if bitand(pattern, mask) == mask,
               place = mod(i + index, nBits) + 1;
               bitsP(place) = -bitsP(place);
            end;
            mask = bitshift(mask, 1);
85       end;

         % Generate the reverse of the bit sequence bitsP()

         bRev = bitsP(length(bitsP):-1:1);
90
         % Generate the full sequence from its quarter parts and
         % compute the SNR after filtering with Atten().

         CData= abs(fft([bitsP' bRev' -bitsP' -bRev'], period));
95       Raw = CData(1:length(CData)/2);
         Amp = Raw .* Atten;

         % "Signal" is the Fundamental harmonic
         % "Noise" is all the other harmonics
100
         Signal = Amp(2);
         Noise = sum([Amp(1) Amp(3:length(Amp))]);

         if Signal <= 0,
105         Signal = 1e-30;
         end;

         SNRtrial = -20 * log10(Noise / Signal);

110      if SNRtrial > maxSNR,
            maxSNR = SNRtrial;
            maxbits = bitsP;
         end;
      end;
115
```

out of its parts as it is passed to the FFT function).

The lowpass filter function which determines the output after lowpass filtering can be applied in the frequency domain. Knowing that the sequence is periodic and therefore its FFT is discrete, consisting of only the harmonics of the sequence period, it is only necessary to know by what factor the lowpass filter function attenuates each of these harmonics. Thus the filter function may be computed once at each of the harmonics out to some sufficiently high harmonic and stored as array of attenutation factors. This filter computation takes place in lines 20 to 31.

The code could be modified to search for optimal bit patterns representing any arbitrary function by applying an appropriate lowpass filter and comparing the resulting FFT to the FFT of the original function. This has not been attempted, though, and its behavior with regard to becoming trapped at local minima is unknown.

We take the time difference between successive samples of the input on each channel and multiply it by the corresponding template value, which itself is computed from the original template value of the baseline algorithm by taking the difference of values between neighboring time bins. After this step, both input and template have a zero-mean form. We can then quantize the template values by replacing each with its *sign*, indicating whether the energy is expected to be rising or falling at each time sample:

$$c_z[t] = \sum_{m=1}^{M} \sum_{n=1}^{N} (x[t-n,m] - x[t-n-1,m])\, p'_z[n,m] \tag{C.2}$$

where

$$p'_z[n,m] = \text{sign}(p_z[n,m] - p_z[n-1,m]). \tag{C.3}$$

Moving from a positive-valued, magnitude-encoding input and template to a differential form has no effect on classification performance, although it complicates the algorithm by introducing negative values on both sides and requiring four-quadrant multiplications. Rendering each template value binary has a negligible effect on classification performance. Binarization without differentiation severely degrades system performance, and interestingly, system performance suffers dramatically if the input is made binary but the templates are kept continuous valued [56].

When the input is transformed by a time differencing operation, it is possible to rearrange the correlation equation and move the differencing operation so that it becomes the final processing step. By noting that the time difference commutes with the summation, we can write Equation (4.1)

$$\begin{aligned} c_z[t] &= \sum_{m=1}^{M} \sum_{n=1}^{N} x[t-n,m]\, p'_z[n,m] \\ &- \sum_{m=1}^{M} \sum_{n=1}^{N} x[(t-1)-n,m]\, p'_z[n,m]. \end{aligned} \tag{C.4}$$

If we let

$$c'_z[t] = \sum_{m=1}^{M} \sum_{n=1}^{N} x[t-n,m]\, p'_z[n,m], \tag{C.5}$$

then

$$c_z[t] = c'_z[t] - c'_z[t-1]. \tag{C.6}$$

Commuting the time difference operation to the end has several distinct advantages for hardware implementation:

C.1 Proof of validity of the pipelined architecture

This is a short proof which verifies that the proposed pipelined architecture produces a correlation result which is exactly equivalent to the result of the non-pipelined correlation.

The pipelined architecture follows the ruleset presented in Chapter 4, Section 4.2.1, and repeated here for clarity:

$$q[n,t] = q[n-1,t-1] + \sum_{m=1}^{M} x'[t,m]\, {p'}_z[n,m] \qquad \forall n \neq 1 \tag{C.7}$$

$$q[1,t] = \sum_{m=1}^{M} x'[t,m]\, {p'}_z[1,m] \tag{C.8}$$

$$c_z[t] = q[N,t-1] \tag{C.9}$$

Now expand out the final correlation result $c_z[t]$:

$$\begin{aligned}
c_z[t] &= q[N,t-1] && \text{(C.10)}\\
&= q[N-1,t-2] + \sum_{m=1}^{M} x'[t-1,m]\, {p'}_z[N,m] && \text{(C.11)}\\
&= q[N-2,t-3] + \sum_{m=1}^{M} x'[t-2,m]\, {p'}_z[N-1,m] \\
&\quad + \sum_{m=1}^{M} x'[t-1,m]\, {p'}_z[N,m] && \text{(C.12)}\\
&\vdots \\
&= q[1,t-N] + \sum_{m=1}^{M} x'[t-N+1,m]\, {p'}_z[2,m] \\
&\quad + \sum_{m=1}^{M} x'[t-N+2,m]\, {p'}_z[2,m] + \ldots + \sum_{m=1}^{M} x'[t-1,m]\, {p'}_z[N,m] && \text{(C.13)}\\
&= \sum_{m=1}^{M} x'[t-N,m]\, {p'}_z[1,m] + \sum_{m=1}^{M} x'[t-N+1,m]\, {p'}_z[2,m] && \text{(C.14)}\\
&\quad + \sum_{m=1}^{M} x'[t-N+2,m]\, {p'}_z[3,m] + \ldots + \sum_{m=1}^{M} x'[t-1,m]\, {p'}_z[N,m] && \text{(C.15)}
\end{aligned}$$

Now we can collect the terms back under a summation over n:

$$c_z[t] = \sum_{n=1}^{N} \sum_{m=1}^{M} x'[t-N+n-1,m]\, {p'}_z[n,m] \tag{C.16}$$

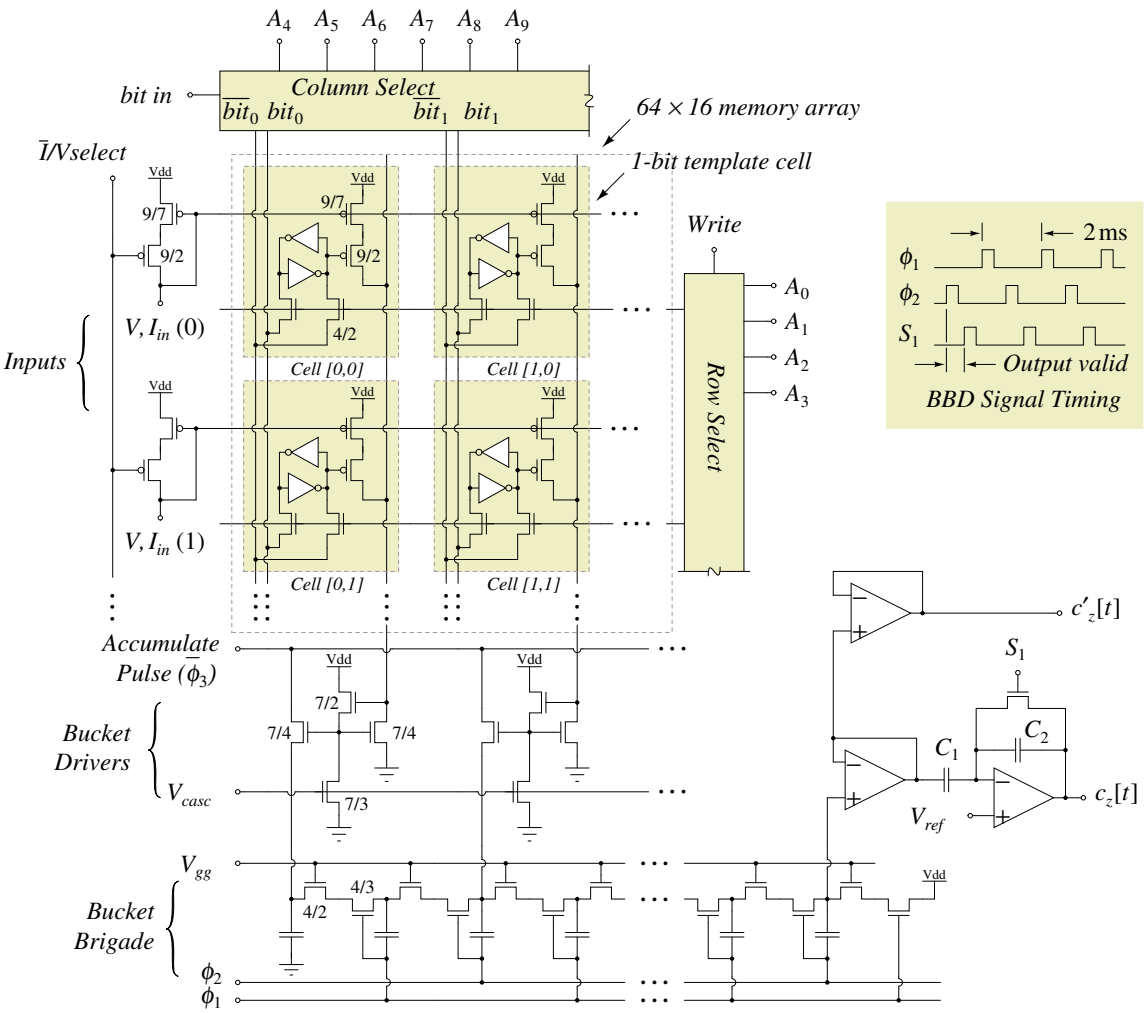

Figure C.3: Time-differencing correlation architecture.

```
#define sign(a) ((a) < 0 ? -1 : 1)

/*-----------------------------------------------------------*/
/*      External variable declarations                       */
/*-----------------------------------------------------------*/
/* itime            = current sample                         */
/* itimen1, itimen2 = last sample                            */
/* filtdata         = bandpass-filtered data                 */
/* lpfiltdata       = time-averaged filter data              */
/* normdata         = peak-detector + time-averaged output   */
/* samprate         = sample rate of input                   */
/* filterfreq       = filter center frequency values         */
/* abcs             = filter coefficients                    */
/* bfiltdata        = intermediate bandpass filter results   */
/* lfiltdata        = intermediate lowpass filter results    */
/* minout, maxout   = peak-detector minimum and maximum      */
/*-----------------------------------------------------------*/

extern long itime, itimen1, itimen2;
extern float *filtdata[NFILTERS];
extern float *lpfiltdata[NFILTERS];
extern float *normdata[DETS];
extern short *shortpcm;
extern int samprate;

float filterfreq[NFILTERS];
float abcs[6][NFILTERS];

float bfiltdata[3][NFILTERS];
float lfiltdata[4][NFILTERS];

float minout[NFILTERS], maxout[NFILTERS];

/*-----------------------------------------------------------*/
/* Function for generating filter center frequencies      */
/*-----------------------------------------------------------*/
/*-------------------------*/
/*  Filter coefficients:   */
/*  Bandpass b0 = abcs[0]  */
/*  Bandpass b2 = abcs[1]  */
/*  Bandpass a1 = abcs[2]  */
/*  Bandpass a2 = abcs[3]  */
/*  Lowpass  b  = abcs[4]  */
/*  Lowpass  a  = abcs[5]  */
```

```
    for (nfilt = 0; nfilt < NFILTERS; nfilt++) {

        /* compute filter cutoff and Q value */

        tau = samprate / (3.14159 * filterfreq[nfilt]);

        /* compute bandpass filter coefficients */

        tausq = tau * tau;
        tauQ = tau / Q;
        denominv = 1 / (1 + tauQ + tausq);
        abcs[0][nfilt] = tau * denominv;
        abcs[1][nfilt] = -abcs[0][nfilt];
        abcs[2][nfilt] = 2 * (tausq - 1) * denominv;
        abcs[3][nfilt] = -(1 - tauQ + tausq) * denominv;

        /* Compute lowpass filter coefficients. "sfreq" */
        /* makes a graded cutoff on a geometric scale   */

        tau = samprate / (3.14159 * sfreq);
        abcs[4][nfilt] = 1 / (1 + tau);
        abcs[5][nfilt] = (tau - 1) / (tau + 1);
        sfreq *= FGRAD;

        /* initialize filter endpoints */

        bfiltdata[1][nfilt] = bfiltdata[0][nfilt] = 0.0;
        lfiltdata[0][nfilt] = 0.0;
        lfiltdata[2][nfilt] = 0.0;

        /* Initialize envelope-detection parameters */

        minout[nfilt] = 0;
        maxout[nfilt] = 0;
    }
}

/*-------------------------------------------------------*/
/*      Bandpass Filter-bank execution                   */
/*-------------------------------------------------------*/

void filterbank()
{
    short nfilt;
```

```
200
         /* filter the output 2x more, for 3rd-order result */

         lfiltdata[3][nfilt] = lfiltdata[2][nfilt];
         lfiltdata[2][nfilt] =
205                    abcs[4][nfilt] * (lfiltdata[0][nfilt]
                     + lfiltdata[1][nfilt]) + abcs[5][nfilt]
                     * lfiltdata[3][nfilt];

         lpfiltdata[nfilt][itime] = abcs[4][nfilt]
210                  * (lfiltdata[2][nfilt] +
                     lfiltdata[3][nfilt]) + abcs[5][nfilt] *
                     lpfiltdata[nfilt][itimen1];
      }
   }
215
   /*-------------------------------------------------------*/
   /* Simulate peak-peak detector in hardware version of ATP */
   /*-------------------------------------------------------*/

220 #define TAUSCALE  0.1   /* frequency-to-drop ratio */

   peakdet()
   {

225    for (nfilt = 0; nfilt < NFILTERS; nfilt++) {
         ntau = TAUSCALE * filterfreq[nfilt];

         /* maximum peak find */

230      maxout[nfilt] -= ntau;
         if (filtdata[nfilt][itime] > maxout[nfilt])
            maxout[nfilt] = filtdata[nfilt][itime];

         /* minimum peak finder */
235
         minout[nfilt] += ntau;
         if (filtdata[nfilt][itime] < minout[nfilt])
            minout[nfilt] = filtdata[nfilt][itime];

240      lpfiltdata[nfilt][itime] =
                  maxout[nfilt] - minout[nfilt];
      }
   }
```

ifications of Chapter 3, Section 3.4, there are two cascaded bandpass filters. They are followed by rectification and smoothing, which can either take the form of the signal full-wave rectifier followed by lowpass filtering (routine `makesmooth()`) or the peak-peak detector (routine `peakdet()`). All outputs are subject to L-1 normalization across all channels in routine `computenorm()`, with the θ value from Equation (4.2) provided by `NCHAN`.

```
/*-------------------------------------------------------*/
/* correlate_dt() -- correlates template with data       */
/*-------------------------------------------------------*/

float correlate_dt (float *template, float *data,
        int numchans, long samples)
{
    float dot = 0.0;
    float t0, t1, d0, d1;
    int i, j, bit;

    for (j = 0; j < samples - 1; j++) {
        for (i = 0; i < numchans; i++) {
            t1 = *(template + numchans);
            t0 = *template++;
            d1 = *(data + numchans);
            d0 = *data++;
            dot += ((t1 - t0) > 0) ? (d1 - d0) : 0;
        }
    }
    return dot;
}

/*-------------------------------------------------------*/
/* correlate_old() -- correlates template with data      */
/*-------------------------------------------------------*/
/* This is according to Fernando's original code, but    */
/*  repaired to skip the normalization channel and not to */
/* wrap around from top to bottom channel.               */
/*-------------------------------------------------------*/

float correlate_old(float *template, float *data,
        int numchans, long samples)
{
    float dot = 0.0;
    float t0, t1, d0, d1;
    int i, j;

    for (j = 0; j < samples; j++) {
        for (i = 0; i < numchans - 2; i++) {

            t1 = *(template + 1);
            d1 = *(data + 1);

```

```
115  /* Classification algorithm                                  */
     /*-----------------------------------------------------------*/

     int classify(FILEHEADER *fh, NEWSEGMENT *sh,
             GENERAL_RECORD *record)
120  {
         int class, bestclass;
         float *data, *template, dot;
         long int j, samples;
         int i, numchans;
125      float maxdot;
         class = 0;
         numchans = fh->numchans;
         maxdot = 0;

130      while (templates[class]) {
            data = record->rec.flt;
            template = templates[class];
     /*     samples = MIN(samples, 64);              */
            samples = MIN(sh->samples, template_samples[class]);
135         dotprod[class] = correlate(template, data, numchans,
                              samples);

            if(dotprod[class]>maxdot) {
                maxdot = dotprod[class];
140             bestclass = class;
            }
            class++;
         }
         return bestclass;
145  }

     /*-----------------------------------------------------------*/
```

E.1 Commentary

This code fragment is the central part of the ATP correlator simulations. The entire program, `classify.c`, originally written by Fernando Pineda, takes input from the database generated from the HEEAR chip, and performs a "leave-one-out" cross-validation loop over all of the recorded transient examples.

The correlation is called from routine `classify()`, line 113, which loops over all

Appendix F

C-code fragment from the optimizer for per-class gains in the ATP

```
/*-------------------------------------------------------------*/

#ifdef CHANDIFF
#define NCHANS  (channels - 1)
#else
#define NCHANS  channels
#endif

/*-------------------------------------------------------------*/

int channels, timebins, filters, classes;

double ***weights = NULL;  /* weights for reconstruction */
double     *gains = NULL;  /* weights on output vector   */

int quantize;

/*-------------------------------------------------------------*/
/* Normalize the gain coefficients                             */
/*-------------------------------------------------------------*/

void gainnorm()
{
   int i;
   double gsum = 0.0, gscale;
```

```
        fprintf(stdout, "Weight matrix cross-correlations\n");

        for (i = 0; i < filters; i++) {
           for (j = 0; j < filters; j++) {
              cgains[i][j] = 0.0;
              for (k = 0; k < channels; k++)
                 for (t = 0; t < timebins; t++)
                    switch (quantize) {

                       /* Quantized template and input */
                       case 2:
                         cgains[i][j] += gains[i]
                          * ((weights[j][k][t] > 0) ? .01 : -.01)
                          * ((weights[i][k][t] > 0) ? .01 : -.01);
                         break;

                       /* Quantized template only */
                       case 1:
                         cgains[i][j] += gains[i]
                             * weights[j][k][t]
                             * ((weights[i][k][t] > 0) ? 1 : -1);
                         break;

                       /* Continuous-valued input & template */
                       default:
                         cgains[i][j] += gains[i]
                             * weights[j][k][t]
                             * weights[i][k][t];
                         break;
                    }
              fprintf(stdout, "%5.3f  ", cgains[i][j]);
           }
           fprintf(stdout, "\n");
        }
     }

     /*-------------------------------------------------------------*/
     /* compute relative gain on each template output               */
     /*-------------------------------------------------------------*/

     #define alpha 0.1
     #define ahpla 0.9
```

```
                /* move gain in the direction of the optimal */
160             /* value by a factor alpha                   */

                newgain = alpha
                     * (-log(nsum / dsum) / cgains[i][i])
                     + ahpla * nngain;
165             ndiff = fabs(newgain - nngain);
                nngain = newgain;
                bcount++;
             } while(ndiff > 0.0001 && bcount < 10000);
             if (bcount == 10000) break;
170
             gains[i] = nngain;
          }
          fprintf(stdout, "\n");

175       /* Renormalize the gain coefficients */

          gainnorm();

          newerr = printerr(cgains);
180       nndiff = fabs(newerr - lasterr);
          lasterr = newerr;

          if (bcount == 10000) {  /* failed to converge */
             fprintf(stdout,
185                 "Gain calculation failed to converge.\n");
             for (i = 0; i < filters; i++) {
                gains[i] = 1.0;
             }
             break;
190       }

       } while (nndiff > 1e-8);

       printgains(cgains);
195
       /* free up allocated memory */

       for (i = 0; i < filters; i++)
          free(cgains[i]);
200    free(cgains);
    }
```

Bibliography

[1] C. A. Mead, *Analog VLSI and Neural Systems*, Reading, MA, Addison-Wesley (1989).

[2] C. A. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, **78** (10), October 1990, pp. 1629–1636.

[3] E. Vittoz, "Analog VLSI signal processing: why, where, and how?" *Analog Integrated Circuits and Signal Processing*, **6**, pp. 27–44, July 1994.

[4] F. Rieke, D. Warland, R. van Steveninck, and W. Bialek, "Spikes: Exploring the Neural Code," Cambridge, MA, The MIT Press (1997).

[5] P. Furth, "On the Design of Optimal Continuous-Time Filter Banks in Subthreshold CMOS," Ph.D. thesis, Johns Hopkins University, Baltimore, MD, 1996.

[6] R. Sarpeshkar, T. Delbrück, and C. A. Mead, "White Noise in MOS Transistors and Resistors," *IEEE Circuits and Devices*, **9** (6), pp. 23–29, Nov. 1993.

[7] J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach," 2nd ed. San Francisco, CA, Morgan Kaufmann Publishers, Inc. (1996).

[8] D. Gabor, "Theory of Communication," *J. Inst. of Electr. Eng.*, **93**, 1946, pp. 429–457.

[9] C. Bingham, M. D. Godfrey, and J. W. Tukey, "Modern Techniques of Power Spectrum Estimation," *IEEE Transactions on Audio and Electroacoustics*, **AU-15** (2), June 1967, pp. 56–66.

[10] G. Strang, "Wavelets and Dilation Equations: A Brief Introduction," *SIAM Review*, **31** (4), December 1989, pp. 614–627.

[22] C. -J. Pu and J. Harris, "Theory and Implementation of a Gaussian Decay Low-pass Filter," *Proc. ISCAS '98*, May, 1998.

[23] O. Moreira-Tamayo and J. Pineda de Gyvez, "Time-Domain Analog Wavelet Transform in Real-Time," *Proc. ISCAS '95*, **3**, pp. 76–79, May, 1995.

[24] O. Moreira-Tamayo and J. Pineda de Gyvez, "Analog Computation of Wavelet Transform Coefficients in Real-Time," *IEEE Trans. Circuits and Systems I*, **44** (1), pp. 67–70, Jan. 1997.

[25] P. W. Wong and R. M. Gray, "FIR Filters with Sigma-Delta Modulation Encoding," *IEEE Trans. Acoust., Speech, Signal Processing*, **38**, pp. 979–990, June 1990.

[26] A. Lu and W. Roberts, "A High-Quality Analog Oscillator Using Oversampling D/A Conversion Techniques," *IEEE Trans. on Circuits and Systems—II*, **41** (7), pp. 437–444, July 1994.

[27] R. Gregorian, K. Martin, and G. Temes, "Switched-Capacitor Circuit Design," *Proc. IEEE*, **71** (8), pp. 941–966, Aug. 1983.

[28] P. Horowitz and W. Hill, *The Art of Electronics*, 2nd ed. New York: Cambridge University Press, 1989, pp. 1030–1034.

[29] K. Martin, "Improved Circuits for the Realization of Switched-Capacitor Filters," *IEEE Trans. Circuits and Systems*, CAS-**27** (4), pp. 237–244, 1980.

[30] J. Bor and C. Wu, "Analog Electronic Cochlea Design Using Multiplexing Switched-Capacitor Circuits," *IEEE Trans. Neural Networks*, **7** (1), Jan. 1996, pp. 155–166.

[31] W. McC. Siebert, *Circuits, Signals, and Systems*. Cambridge, MA: The MIT Press, 1986, pp. 496–497.

[32] Q. T. Huang and G. S. Moschytz, "Analog FIR Filters with an Oversampled Σ–Δ Modulator," *IEEE Trans. on Circuits and Systems—II*, **39** (9), pp. 658–663, Sept. 1992.

[33] D. Lancaster, "The Quest for Magic Sine Waves," Circuit Cellar INK, issue 59, pp. 28–35, June 1995.

[46] W. Serdijn, M. Broest, J. Mulder, A. van der Woerd, and A. van Roermund, "Low-voltage Ultra-low-power Translinear Integrator for Audio Filter Applications," *IEEE Journal of Solid State Circuits*, **32**, No. 4, pp. 577–581, April, 1997.

[47] W. Himmelbauer, P. Furth, P. Pouliquen, and A. G. Andreou, "Log-domain Filters in Subthreshold MOS," *Technical Report TR-96/3*, ECE Department, Johns Hopkins University, Baltimore, MD, January 1996.

[48] R. T. Edwards and G. Cauwenberghs "Synthesis of Log-Domain Filters from First-Order Building Blocks" (to appear in) *Analog Integrated Circuits and Signal Processing*, special issue on translinear circuits.

[49] R. T. Edwards and G. Cauwenberghs, "A Second-Order Log-Domain Bandpass Filter for Audio Frequency Applications," *International Symposium on Circuits and Systems (ISCAS)*, Monterey, CA, June 1998.

[50] B. Gilbert, "A 16-channel array normalizer," *IEEE J. Solid-State Circuits.*, **SC-19**, pp. 956–963, 1984.

[51] W. Liu, A. G. Andreou, and M. H. Goldstein, Jr., "Voiced-Speech Representation by an Analog Silicon Model of the Auditory Periphery," *IEEE Trans. Neural Networks*, **3** (3), May 1992, pp. 477–487.

[52] A. van Schaik, E. Fragnière, E. Vittoz, "A Silicon Model of Amplitude Modulation Detection in the Auditory Brainstem," *Neural Information Processing Systems (NIPS)*, Denver, 1996.

[53] M. H. Goldstein, W. Liu, and R. E. Jenkins, "Speech Processing by Real and Silicon Ears," *Johns Hopkins APL Technical Digest*, **12** (2), 1991, pp. 115–128.

[54] F. J. Pineda, K. Ryals, D. Steigerwald, and P. Furth, "Acoustic Transient Processing using the Hopkins Electronic Ear," *World Conference on Neural Networks*, Washington, D.C., 1995.

[55] F. J. Pineda, G. Cauwenberghs, R. T. Edwards, "Bangs, Clicks, Snaps, Thuds, and Whacks: An Architecture for Acoustic Transient Processing," *Neural Information Processing Systems (NIPS)*, Denver, 1996.

[68] K. P. Unnikrishnan, J. J. Hopfield, and D. W. Tank, "Connected-Digit Speaker-Dependent Speech Recognition Using a Neural Network with Time-Delayed Connections," *IEEE Transactions on Signal Processing*, **39**, pp. 698–713, 1991.

[69] K. P. Unnikrishnan, J. J. Hopfield, and D. W. Tank, "Speaker-Independent Digit Recognition Using a Neural Network with Time-Delayed Connections," *Neural Computation*, **4**, pp. 108–119, 1992.

[70] K. Wang and S. Shamma, "Representation of Acoustic Signals in the Primary Auditory Cortex," *IEEE Trans. Audio and Speech Processing*, **3**(5), pp. 382–395, 1995.

[71] A. J. Bell and T. J. Sejnowski, "Learning the higher-order structure of a natural sound," *Network Computation in Neural Systems*, **7**(2), pp. 261–267, 1996.

[72] A. J. Bell and T. J. Sejnowski, "The 'Independent Components' of Natural Scenes are Edge Filters," *Vision Research* **37** (23), pp. 3327–3338, 1997.

[73] V. N. Vapnik, *The Nature of Statistical Learning Theory*, New York, NY, Springer-Verlag (1995).