Time-Frequency Acoustic Processing and Recognition:
Analysis and Analog VLSI Implementations

by

Robert Timothy Edwards

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

March, 1999

© Robert Timothy Edwards 1999

All rights reserved

Acknowledgements

In order of appearance, I would like to thank the cast of characters who inspired this thesis, either directly or indirectly:

My parents and sister, who taught me critical thinking, studying, and writing skills, and encouraged me to explore my universe in all forms: Art, music, literature, language, and science. In spite of all their efforts, I was enamored of computers and technology, and became an engineer.

Dr. Hisham Massoud at Duke University, who inadvertently launched me on the path of neuromorphic engineering by recommending and loaning to me a book by Carver Mead called *Analog VLSI and Neural Systems*, and who affected my choice of graduate schools by advising me that the top five electrical engineering programs in the country were, in order, "Stanford, Stanford, Stanford, Berkeley, and Berkeley."

Dr. Michael Godfrey at Stanford University, who enthusiastically encouraged me to pursue research in Analog VLSI. He and I, Boyd Fowler, and Neal Bhadkamkar put together an Analog VLSI research laboratory in the basement of Durand Building and held our own against the naysayers in the department.

My wife, Linglan, who shared with me the occasionally tortuous life of the married graduate student. Under mutual moral support, we kept roughly the same schedule from beginning to end of our respective theses. She beat me to it by a couple of months.

My thesis advisor, Dr. Gert Cauwenberghs, who was forced to bear the brunt of my moments of extreme aggravation that typically accompany thesis research.

All the people whose support, friendship, experiences and discussions have shaped many aspects of my research and this thesis: Among them Pamela Abshire, Andreas Andreou, Marc Cohen, Abbas El Gamal, Paul Furth, Wolfgang Himmelbauer, Mark Martin, Amjad Obeidat, Fernando Pineda, Philippe Pouliquen, Michael Sebert, Shihab Shamma, and many, many others.

2.18 Results and implementation . . . 46
2.19 Modulation Multiplier . . . 49
2.20 Switch-Cap Wavelet Gaussian Function . . . 50
2.21 Output Time Multiplexing . . . 52
2.22 Wavelet chip slice . . . 53
2.23 Experimental Results . . . 55
2.24 Extensions of the Research . . . 58
2.25 Summary . . . 61

3 Current-Mode Filterbank Frontend 62
3.1 Time-Frequency Representations using Filterbanks . . . 62
3.2 Parallel Filterbanks for Transient Classification . . . 63
3.3 Current-Mode Filters for Current-Mode Applications . . . 64
3.4 High-Level Simulations of the Filterbank Frontend . . . 66
3.5 Introduction to Translinear Circuits and Log-domain Filtering . . . 70
3.6 Principles of log-domain synthesis . . . 73
3.7 First-Order Circuit synthesis . . . 76
3.8 Designing second-order sections . . . 78
3.9 Technology limitations for low-frequency filter design . . . 83
3.10 Layout Considerations for VLSI Log-Domain Circuits . . . 86
3.11 Current-Mode Circuits for Non-Filtering Applications . . . 89
3.12 Signal Rectification and Smoothing . . . 89
3.13 Signal Rectifier . . . 90
3.14 Signal Peak-Peak Detector . . . 91
3.15 L-1 Normalization Array . . . 94
3.16 Experimental Results . . . 97
3.17 Summary . . . 105

4 Acoustic Transient Processing 106
4.1 Introduction . . . 106
4.1.1 The Problem of Speech Recognition . . . 106
4.1.2 Acoustic Transients . . . 107
4.2 Algorithms . . . 108
4.2.1 Simplifying The Correlation Equation . . . 112
4.3 Simulations . . . 119
4.3.1 The Hopkins Electronic Ear (HEEAR) Processor . . . 120
4.3.2 Simulating the Acoustic Transient Baseline Algorithm . . . 121
4.3.3 Optimizing Correlation Algorithms . . . 123
4.3.4 Simulations of different zero-mean representations . . . 125
4.3.5 High-Level Simulations of ATP Mixed-Mode VLSI Hardware . . . 128
4.3.6 Optimization of the classifier using per-class gains . . . 128
4.3.7 System Robustness . . . 131
4.3.8 Research Directions . . . 133
4.3.9 Remarks . . . 134
4.4 Hardware Implementation of the Acoustic Transient Processor . . . 135

List of Figures

1.1 Some mappings of the TF plane. 7

2.1 Output sampling. Points marked '$\times$' represent center of the time-frequency area covered by that sampled output. 17
2.2 Frequency-time representation of the input as overlapping Gaussian filters. 17
2.3 Gabor sine and cosine logons, or wavelets. 19
2.4 Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H. 23
2.5 Complex demodulation (2 channels shown). 24
2.6 Complex modulation reconstruction (2 channels shown). 25
2.7 Circuit diagram of the frequency-division toggle flip-flops. 29
2.8 Gilbert multiplier with cascodes. 30
2.9 Frequency-domain transfer function of the final output of n cascaded lowpass filters as a function of the number of stages n. 32
2.10 Impulse response of the final output of n cascaded lowpass filters as a function of the number of stages n. 32
2.11 n cascaded stages of a filter approximating a half-Gaussian function, using continuous-time transconductance-C filters. 33
2.12 Wavelet decomposition block diagram for a single sine/cosine pair. 35
2.13 Wavelet reconstruction block diagram for a single sine/cosine pair. 35
2.14 Wavelet Transform Chip block diagram. 36
2.15 Architecture of Moreira-Tamayo *et al.* 38
2.16 The analog oscillator architecture of Lu *et al.* [26] containing a 2nd-order delta-sigma modulator. 40
2.17 *Top*: Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H. *Bottom*: Demodulation of an input X in the frequency domain with an oversampled sine wave S' using an ideal smoothing filter G and modulation filter H. 45
2.18 The oversampled sine sequence. 47
2.19 Optimized 64-bit oversampled sine sequence, first quadrant. 47
2.20 Frequency domain properties of the raw (**o**) and filtered (**x**) bit sequences. 48
2.21 Use of Gray code to generate sine and cosine sequences. 49
2.22 Multiplexing *vs.* Multiplying. 50

3.23 L-1 normalization circuit, after Gilbert [50]. 98
3.24 Measured magnitude response of the log-domain first-order lowpass filter from the test chip. 99
3.25 Measured magnitude response of one bandpass channel in the filterbank system, made of two cascaded second-order log-domain bandpass filters, over three tunings of the center frequency. 101
3.26 Measured magnitude response of all bandpass channels in the filterbank system, as measured at the output after peak-detection and smoothing. 102
3.27 Measured center frequencies of all bandpass channels in the filterbank system (circles), compared to the ideal exponential spacing (solid line). 103
3.28 Photograph of the fifteen-channel bandpass filterbank fabricated in 1.2 µm technology inside a 2.2 mm × 2.2 mm padframe. 104

4.1 Template correlator, as a direct implementation of the baseline algorithm, Equation (4.1). 110
4.2 Template correlation. 111
4.3 Filterbank output after applying an L-1 normalization across all channels. A single channel has been added to the system (top trace), containing the result of a constant value less the instantaneous sum of the remaining channel values. 113
4.4 Template correlator with pipelined architecture and multiplexors replacing the multipliers. 115
4.5 Values in the pipelined delay registers in the correlation algorithm simulation. . . . 116
4.6 Template correlator with [0,1] encoding of template values. 117
4.7 Block diagram of the temporal current correlator. 119
4.8 Example transient recorded data. 120
4.9 Example HEEAR-processed transients. 122
4.10 Example templates learned by the ATP algorithm. 123
4.11 Example of a real-valued template. 124
4.12 The same template reduced to trinary values. 124
4.13 The same template reduced to binary values. 125
4.14 Effect of decreasing the number of time-bins. 132
4.15 Effect of white noise added to the correlator inputs. 133
4.16 Block diagram of the temporal current correlator. 136
4.17 Efficient dynamic memory cell for the correlator array. 138
4.18 Details of the bucket brigade device (BBD). 139
4.19 A switched capacitor circuit to compute channel differences at the output. 141
4.20 The complete correlator array including both bucket brigade devices. Representative MOS device sizes are given as W/L in units of $\lambda = 0.6\,\mu m$. 142
4.21 Linearity between output voltage and input current. 143
4.22 Impulse response at each tap of the bucket brigade device. 144
4.23 Matching between devices throughout the template at 1 µA input. 145
4.24 Matching between devices throughout the template at 100 nA input. 145
4.25 Photomicrograph of the acoustic transient correlator, a 2.2 mm × 2.2 mm die fabricated in a 1.2µm CMOS technology. 146

C.1 Block diagram of the temporal current correlator, using time-differencing operations on the input. 197
C.2 Switch-cap time-differencing circuit. 199
C.3 Time-differencing correlation architecture. 200

Chapter 1

Introduction

1.1 Introduction

In the late 1980's, as the computer revolution was getting into full swing, Carver Mead wrote a book called *Analog VLSI and Neural Systems* [1] espousing a view of analog circuit design which was different from the generally accepted norm (which might be described as "better amplifiers and D/A converters"). The underlying premise is that biological systems outperform electrical/electronic systems by many orders of magnitude in terms of both compactness of design and consumption of power resources. Loss of speed is compensated by massive parallelism of architecture, and precision is not compromised. Indeed, the raw computational power and often precision [4] of many biomechanical and neurological systems is unparalleled.

The reasoning behind the dichotomy in power dissipation between biological and electronic computers is relatively simple. At one extreme, biological systems compete for resources and the forces of natural selection drive them towards the extremes of processing power (smarter is better) and energy efficiency (the least energy efficient creatures tend to be the first to starve). These requirements are strongly interdependent, as the process of thinking requires considerably more energy than, for instance, being a potted plant. But there are obvious competitive advantages to being perceptive of one's environment, perceptive of one's condition, mobile and, at the top of the heap, rational. But regardless of the tradeoff made between processing power and energy efficiency, natural selection always ensures that in biological systems, one is continuously minimized with respect to the other. Small size and compact design is a variable closely related to power efficiency, as larger organisms consume more resources, but larger bodies support larger brain cases housing more powerful brains.

analog design to the margins, and even that typically meaning operational amplifier design, and A/D (analog-to-digital) conversion. The modern engineer deals with the real world through a transducer and an A/D, after which the sky's the limit on signal processing possibilities. Except for simple CMOS imagers and some newer applications based on capacitive sensors, transducers are the realm of Microelectromechanical systems (MEMS) research and tend to require complicated or exotic fabrication processes that are usually incompatible with analog CMOS or BiCMOS circuit layout. Thus the sought-after application of "smart" analog-to-digital conversion, with sophisticated analog processing occurring at the source on the analog signal transducer, remains elusive.

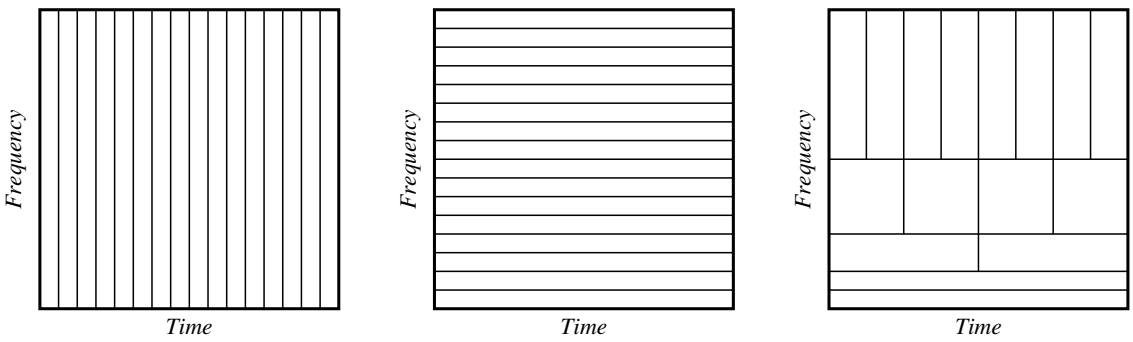
On the other hand, the problem of power consumption and heat dissipation works against the prospect of digital systems increasing in size and complexity without bound. In addition, ultra-low power circuits are becoming increasingly necessary as mobile applications increase in popularity and use, especially in consumer electronics (mobile phones, laptop computers, GPS receivers), but also in space electronics (micro-satellites, deep-space probes) military electronics, and medical electronics (pacemakers, hearing aids). Power constraints determine the difference between a battery life of hours and one of days, or one of months *vs.* one of years, or can mean the difference between a device which can operate exclusively off of solar power and one which needs alternate sources of power. Micropower electronics keep digital watches running for years; this kind of performance should and will be demanded of many more applications. Even for non-mobile applications, power regulation and heat dissipation become major problems when electronics begin consuming dozens of watts, forcing designers to find ways to cut back on power use.

Analog VLSI designers often are able to overcome some of the drawbacks of the power-hungry digital domain by emulating biological processes in electronic circuits which make optimal use of the analog domain. Mimicking neurological processes in electronics is sometimes referred to as "neuromorphic engineering" [2], a term which can be applied to processes on the level of individual neurons up to entire systems based on psychophysical data, and which is irrespective of the medium of implementation, be it analog custom integrated circuits, DSPs, microcontrollers, or computer software. There is a seemingly endless debate as to whether the human brain is most accurately described as a digital or analog signal processing system (it contains elements of both), and while it is possible that quasi-digital, spike-based processing may be the ultimate solution to approaching the efficiency and processing power of biological neural systems, as of now only specially designed analog systems come close to this goal. Neuromorphic systems have the capability to couple massively parallel computation with low power consumption and a large degree of robustness in the presence of both noise and mismatch or failure of components. Achieving this goal is a

- A mixed-mode architecture for implementing efficient template correlation, especially for use in conjunction with the frontend filterbank for acoustic transient classification [55, 56, 57].
- Analysis and optimization of acoustic transient classification algorithms using template correlation methods, and investigation of training methods for such classifiers [58].
- Fabricated VLSI circuits for each of the indicated architectures, including results and measurements which are detailed in this thesis, as well as test hardware and software which is beyond the scope of the thesis to describe, but which has been made publicly available and has been of benefit to many others.

Due to the variety of material encountered in my research, I have organized this dissertation into four major topics:

1. Chapter 2. Mapping the time-frequency plane with efficient analog and mixed-signal computation: The Continuous Wavelet Transform Processor.
2. Chapter 3. Continuous-time, current-mode circuits for acoustic-band filterbanks: Log-Domain Circuits, Filter Design and Synthesis.
3. Chapter 4. Algorithms, architectures, and mixed-signal circuits for pattern recognition using template correlation: The Acoustic Transient Classifier.
4. Chapter 5. Methods for training the acoustic transient classifier, extensions and directions of the research.



(a) Dirac (b) Fourier (c) Wavelet

Figure 1.1: Some mappings of the TF plane.

filter center frequencies f_c are spaced exponentially, with the bandwidth of each filter proportional to its center frequency. If we use a simple filterbank of bandpass filters to generate the wavelet map, then to adequately capture the output in a sampled-data system we must sample each filter channel at twice (*i.e.*, the Nyquist rate of) the upper cutoff frequency of its transfer function. However, in this architecture the upper cutoff is typically twice the value of the bandwidth, which results in an output bandwidth which is larger than the input bandwidth. The wavelet decomposition, however, describes a *transform*, so no information has been added to the system and it should not be necessary to increase the bandwidth of the system. Preferably, the system should shift the frequency content of each channel toward zero, allowing it to be sampled at the Nyquist rate corresponding to the channel bandwidth rather than the channel upper cutoff frequency. Thus the total output bandwidth is equal to the input bandwidth.

We adopted the method of shifting the center frequency f_c of the band to zero using *complex demodulation* [9]. The result of this process is two orthogonal outputs each of which can be sampled at twice the lowpass filter cutoff frequency. Re-modulation of the two outputs by the same process is equivalent to bandpass-filtering the original signal. We can approximately reconstruct the original input by applying this manipulation to each filterbank channel output and adding them together.

The Gaussian function has the most compact representation in the time-frequency plane, and so it forms the core filtering process of the Continuous Wavelet Transform processor. An approximation to a half-Gaussian-shaped lowpass function is quite easy to create using a series of

1.2.2 Acoustic Transient Recognition

Simple patterns often can be successfully recognized (classified) by directly correlating each input pattern with a stored template: compute the distance between each input and template entry using an appropriate metric, sum the result over all components of the feature space, and compare the totals among all templates to select a winner. This method is a linear classifier encoding one separating hyperplane per template. For acoustic signal processing, the feature space is typically a time-frequency map.

Speech signals are normally too complicated to enable a successful recognition system using correlation, due to the necessity of coping with large variance in pitch, timbre, and time for the same target output. *Acoustic transients*, on the other hand, are signals which by definition occur over very short periods of time (less than approximately 1/10 second). Examples of acoustic transients are the sound of a handclap, a door closing, sonar echos, and certain events in speech which occur on sub-phonetic time scales, such as the sudden bursts and silences associated with hard consonants like 'p' and 't'. Time-frequency decomposition gives a relatively stable description of an acoustic transient from instance to instance.

Together with Dr. Fernando Pineda, at the Johns Hopkins University Applied Physics Laboratory (JHU-APL), we have experimented with acoustic transient recognition by correlating time-frequency-mapped outputs with templates formed by averaging together examples of the transients from a training set [54, 55]. In addition, we have experimented with ways of reducing the complexity of the computation and the amount of data stored in each template. The result is a system which lends itself very nicely to efficient implementation in analog hardware [57].

We begin with a baseline algorithm which is the direct correlation between an input and a template

$$c_z[\,t\,] = \sum_{m=1}^{M} \sum_{n=1}^{N} x[t-n, m]\, p_z[n, m] \tag{1.2}$$

where M is the number of frequency channels of the input, N is the maximum number of time bins in the window (enough to cover the transient event at 1 to 2 ms per bin), x is the array of input signals split into frequency bands, p_z is the matrix of template pattern values for pattern z, and t is the current time. This formula produces a running correlation $c_z[\,t\,]$ of the input array with the template z. A system implementing the correlation in this form requires an analog (or multibit, if digital) storage for $M \times N$ values for *each* template, and $M \times N$ multiplications at every time step, also for each template. The computational requirements are formidable: for example, a 10-template classifier with a 16-channel filterbank frontend, operating at 2 ms per time step and allowing approx-

tectures. The architecture described above represents only one of a number of different possibilities. First, any combination of input or template can be made binary or trinary. Second, although a zero-mean transformation is the best way to get a simple decision boundary for choosing a binary value, the choice of pairwise frequency channel differences is only one of many such transformations. Other possibilities include time differences between samples, center-surround computation for each frequency channel, or some combination of time and frequency differences. The architecture of Figure 4.16 was chosen as the simplest representation maintaining acceptable performance on the classification task. However, results of the analysis of different architectures showed that the use of a channel difference computation in addition to a binary representation of *both* input and template resulted in similar classification rates to the proposed mixed-mode architecture. A system with both binary inputs and templates can be implemented entirely in the digital domain (except for the front-tend processing of the input prior to binarization). The optimal form of a binary-binary correlator would be a custom digital architecture, using a parallel architecture similar to that proposed for the analog-binary correlation. The slow speed at which the system operates (1 ms per sample) allows the digital system to be operated at extremely low power. On the other hand, the speed is slow enough to allow the architecture to be completely serial, which precludes low-power operation but permits the system to be built with discrete IC parts. A system based on discrete parts and field-programmable gate arrays (FPGAs) can be designed and built within the space of about two months at a cost similar to that of a custom chip. I designed and constructed a digital ATP system based on a trinary-trinary correlation method (an extension of the binary-binary case), described in detail in Chapter 4, both as a way to have a working real-time version of the algorithm, and as a way to directly compare area, power consumption, and other critical properties between the digital and the mixed-mode systems.

1.2.4 Overview: Learning and Continuous Speech Recognition

While Chapter 4 presents in detail the algorithms and architectures which perform robust acoustic transient classification, they do not cover the subject of how the templates are generated, which is part of the broad topic of *machine learning*. Chapter 5 takes a look at the average-value method used to train the classifier in simulation. The baseline algorithm calls for simple learning of template values by aligning and averaging examples from the training dataset. The method requires a segmentation algorithm to detect transient events in the input and determine the point at which the correlation output should be examined for a classification result. More intelligent methods

of the auditory cortex which encode maps of the time-frequency domain. Certain neurons in this region of the brain respond strongly to wavelet-like regions in time-frequency space, of differing size, scale, and angular orientation. Thus they are more efficient than the wavelet processor at capturing specific auditory events. It is likely that they encode the independent components of all auditory events which are meaningful to humans. The wavelet-like regions of response include areas of excitation and other areas of inhibition. The complex responses of the neurons are a result of many synaptic connections and are not easy to describe in neural network terms. However, they can be easily approximated by templates. A trinary encoding scheme such as that used by the digital transient processor allows encoding regions of excitation as positive bits and regions of inhibition as negative bits. Zero values mask out the regions of little or no response and fill in the corners of the artificially rectangular template, leaving elliptical areas like those encountered in auditory cortex neurons. Although the binary inputs and templates are only an approximation to the information-rich spike trains delivering auditory information through the brain and the continuous-valued response of the neurons, the template correlator should be able to provide additional insight into the processing of auditory signals in the brain, and may shed light on how these complicated feature detectors in the brain can work together to enable the robust sound and speech recognition of which the brain is so magnificently capable.

may come to mind: Either vary the size of the window according to the demands of the moment, or perform the analysis in parallel with windows of several different lengths. The first solution is adaptive, and while it is a very interesting solution, it is difficult to compute and requires information about the adaptation to be kept with the decomposed signal if the signal is to be analyzed or reconstructed. Advanced adaptive techniques are beyond the scope of this thesis. The other solution, parallel analysis at different scales, works well, but the result is redundantly computed, which increases the bandwidth of the decomposed signal n-fold for n parallel analyses. If we remove the redundant parts of the decomposition, the part that is left is a wavelet decomposition. Parts of the decomposition capture features of the signal which occur over short time (*transients*) and other parts capture features of the signal which have fine frequency structure (*formants*). Natural signals, particularly acoustic ones which are the primary topic of the thesis, require the accurate detection of both transient-like and formant-like features for automatic classification and interpretation of the signal.

Each tile of the time-frequency plane represents a single "wavelet coefficient" computed by applying a filtering function centered on that area and having the correct aspect ratio between time span and bandwidth. The so-called "mother function" describes a family of functions at different scales (a) and temporal offsets (b) which determine the position and aspect ratio of each tile covering the time-frequency plane:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right). \tag{2.1}$$

It is desirable for the wavelet function to have *compact support*; that is, the function should be bounded or generally localized in time and frequency. In formal terms, the wavelet should be able to meet the criteria [14]

$$|\psi(t)| \leq c\,(1+|t|)^{-1-\epsilon} \tag{2.2}$$

$$|\Psi(\omega)| \leq c\,(1+|\omega|)^{-1-\epsilon} \tag{2.3}$$

for some $\epsilon > 0$. Existence of an inverse transform depends on the relationship

$$\int_{-\infty}^{\infty} \frac{|\psi(\omega)|^2}{|\omega|} d\omega < +\infty. \tag{2.4}$$

Equation (2.4) implies that $\Psi(\omega = 0) = 0$, which in turn implies an oscillatory function in time.

The Discrete Wavelet Transform (DWT) is a discrete-time function which derives from certain families of orthonormal basis functions which satisfy the conditions of (2.3) and (2.4). The

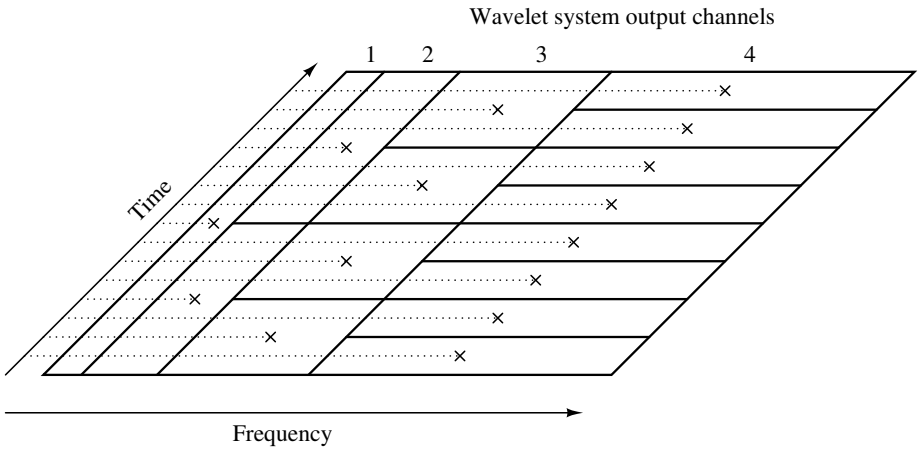


Figure 2.1: Output sampling. Points marked '×' represent center of the time-frequency area covered by that sampled output.

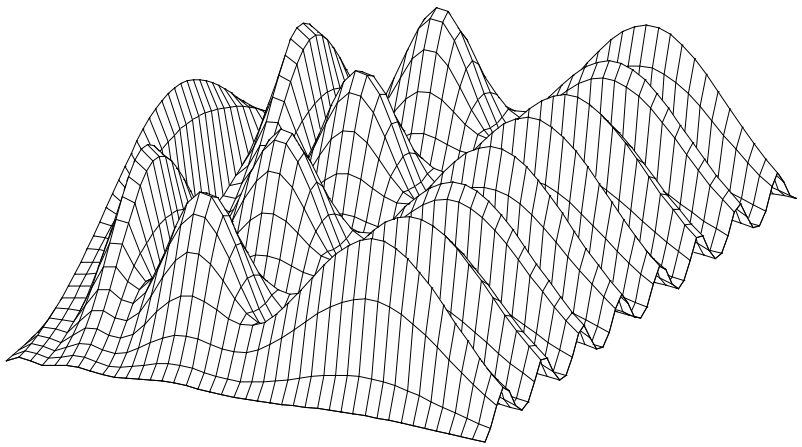


Figure 2.2: Frequency-time representation of the input as overlapping Gaussian filters.

The complex exponential in either transform can be split into real and imaginary parts, in which case the function appears as a sinusoid modulated by a Gaussian envelope, as shown in Figure 2.3. A family of curves can be generated on this basis, using the Gaussian as an envelope around a sinusoidal signal of differing frequencies. The meaning of these curves should be intuitively clear: The more cycles of a sinusoid fit into the Gaussian envelope, the better the frequency is defined, but the poorer the time is defined. A pure sinusoid of infinite duration represents one extreme, for which frequency is known exactly but time is not known at all. Likewise, the Gaussian itself represents the other extreme: the cosine function for which time is known exactly but for which frequency is entirely unknown, having no reference signal to which it can be compared. The "wavelet tiling" of the time-frequency plane (Figure 1.1 (c)) dictates the ratio of ω_a, the modulating frequency, to σ_b, the width of the Gaussian envelope. As one gets bigger, the other gets smaller.

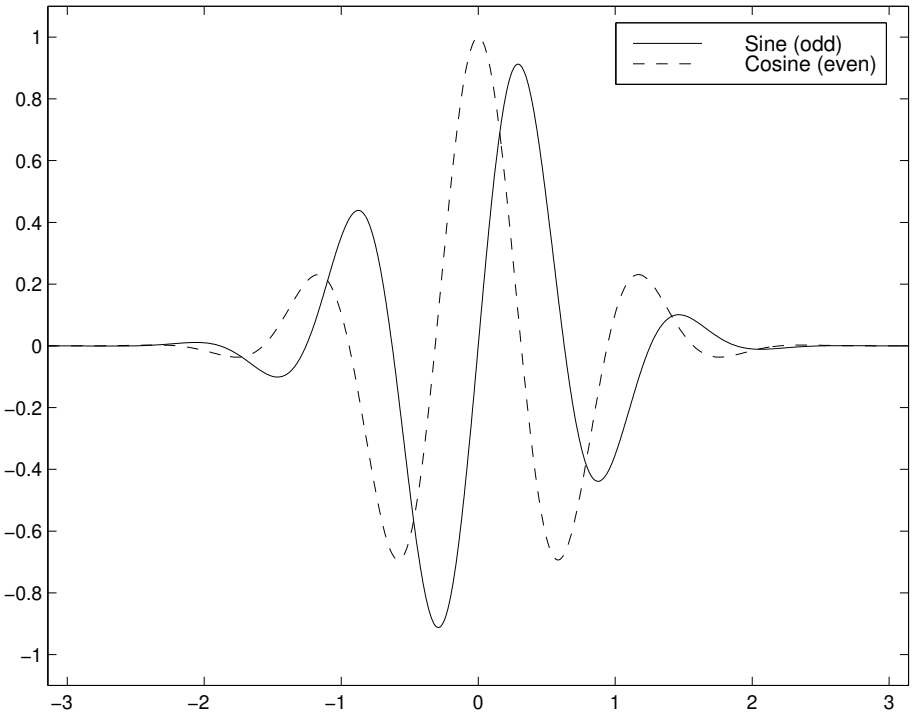


Figure 2.3: Gabor sine and cosine logons, or wavelets.

The Fourier transform property of time and frequency shifts yields an interesting insight about the connection to dilations and shifts of the discrete wavelet transform, since they are manifestations of the Fourier and inverse Fourier transforms. Unlike the discrete wavelet transform families, the Fourier transform pair of the Gabor logon family is beautifully symmetric.

Demodulation of a signal by a carrier

An input time-varying signal $x(t)$ is multiplied by a sinusoidal reference signal $s(t) = \cos(\omega_0 t)$. As viewed in the frequency domain, the result splits the input around the reference to generate the sum (reference + input) and difference (reference − input) components. This result can be easily seen by considering a input consisting of a single sinusoidal function of arbitrary phase relationship to the reference, for instance $x(t) = \cos(\omega t + \phi)$. The modulation is then

$$x(t)s(t) = \cos(\omega t + \phi)\cos(\omega_0 t). \tag{2.8}$$

Now, adding the trigonometric identities

$$\cos(u+v) = \cos(u)\cos(v) - \sin(u)\sin(v) \tag{2.9}$$
$$\cos(u-v) = \cos(u)\cos(v) + \sin(u)\sin(v) \tag{2.10}$$

together gives

$$\cos(u+v) + \cos(u-v) = 2\cos(u)\cos(v) \tag{2.11}$$

such that, when used with Equation (2.8), we get

$$x(t)s(t) = \frac{1}{2}\left(\cos((\omega_0 + \omega)t + \phi) + \cos((\omega_0 - \omega)t - \phi)\right) \tag{2.12}$$

which contains one sinusoid describing the sum $\omega_0 + \omega$ of the carrier and signal frequencies, and one describing the difference $\omega_0 - \omega$. Extending the result to arbitrary inputs involves viewing the arbitrary input as a Fourier series of sine components; since the modulation multiplication is a linear function, superposition applies, and every Fourier component of the arbitrary input is split into sum and difference components across the carrier frequency. Filtering the resultant signal with a highpass filter which accepts the sum component but rejects the difference component is the process known as *modulation*. Filtering the same signal instead with a lowpass filter which accepts the difference component but rejects the sum component is the process known as *demodulation*.

A useful application of the principle of signal modulation involves performing the demodulation *first*. In such case, the signal to be encoded is multiplied by an in-band carrier frequency in order to shift the desired frequency band down to zero. The resulting signal is lowpass filtered to remove the component representing the sum of carrier and modulator. If desired, the signal can then be modulated back into its original band, at which point the result is a bandpassed signal. Not only is it bandlimited, but the band to which it is limited can be made arbitrarily small, since there is no limit (other than the practical limits of noise, jitter of the carrier frequency, *etc*.) to the cutoff frequency of the lowpass function. The tight band limit translates to an arbitrarily large effective value of Q. The position of the band is placed at the carrier frequency, so it can be made very accurate (particularly since in our architecture the carrier frequency is driven by a quartz oscillator).

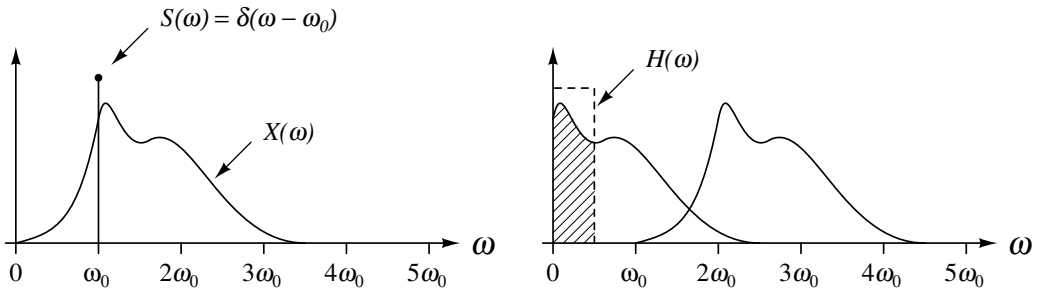


Figure 2.4: Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H.

Reconstruction involves first multiplying each function by the same sine and cosine carriers:

$$h(t) * (x(t)s_1(t))s_1(t) = \cos(\omega_0 t)\frac{1}{2}\cos((\omega_0 - \omega)t - \phi) \quad (2.21)$$

$$h(t) * (x(t)s_2(t))s_2(t) = \sin(\omega_0 t)\frac{1}{2}\sin((\omega_0 - \omega)t - \phi). \quad (2.22)$$

Finally, without filtering, these two parts are added together to produce the reconstruction (in this case, exact reconstruction, due to the use of perfect filters):

$$x'(t) = h(t) * (x(t)s_1(t))s_1(t) + h(t) * (x(t)s_2(t))s_2(t) \quad (2.23)$$

$$= \cos(\omega_0 t)\frac{1}{2}\cos((\omega_0 - \omega)t - \phi) + \sin(\omega_0 t)\frac{1}{2}\sin((\omega_0 - \omega)t - \phi). \quad (2.24)$$

The trigonometric identity (2.10) applies directly, giving the final result:

$$x'(t) = \frac{1}{2}\cos((\omega_0 - \omega_0 + \omega)t + \phi) \quad (2.25)$$

$$= \frac{1}{2}\cos(\omega_0 t + \phi) \quad (2.26)$$

$$= \frac{1}{2}x(t) \quad (2.27)$$

showing that the reconstruction is exact except for the required application of a gain of two. Again, this example can be extended to arbitrary inputs by the application of Fourier series and superposition.

2.5 Complex Demodulation in the Continuous Wavelet Processor

The architecture used for complex demodulation in the wavelet processor is depicted in Figures 2.5 and 2.6 and described below.

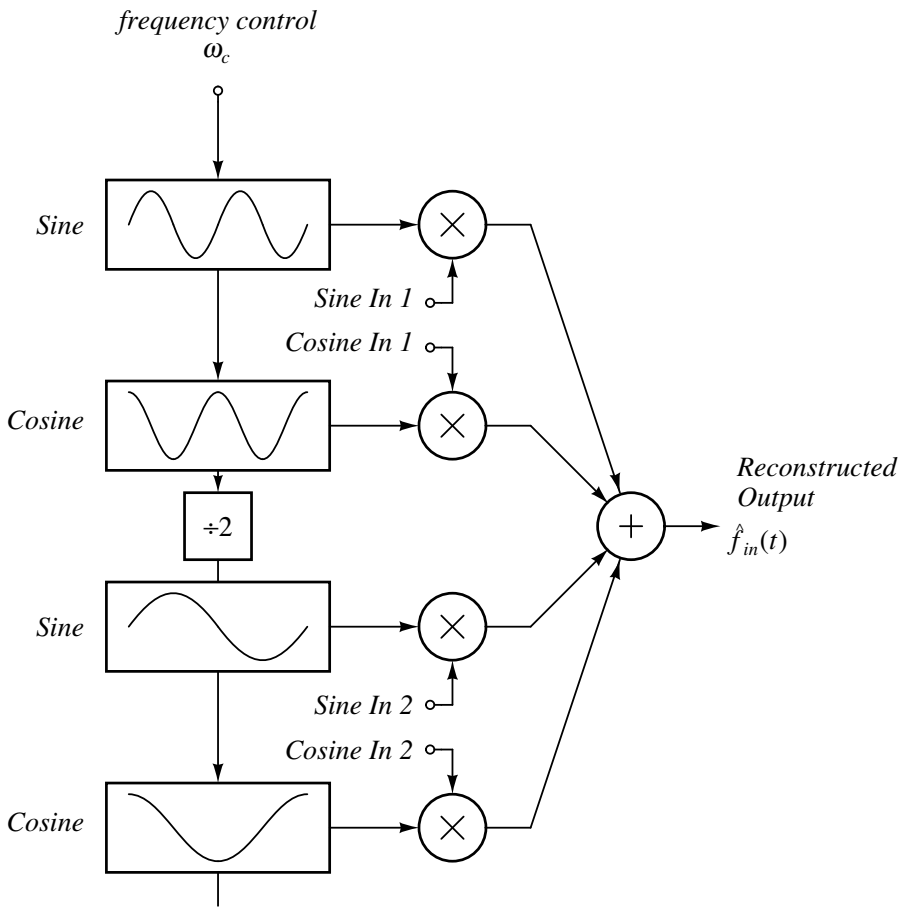


Figure 2.6: Complex modulation reconstruction (2 channels shown).

This is the same function as the demodulation (2.28) except for the change in sign and the lack of a lowpass filter function. We multiply out the real and imaginary parts of this equation to get a purely real result:

$$\hat{f}_{\text{in}}(t) = f_{\text{real}}(t) \cos \omega_c t + f_{\text{imag}}(t) \sin \omega_c t. \tag{2.33}$$

This step separates the negative from the positive frequency components as it pushes the center frequency up from zero to its original value. The result is the exact reconstruction of the original input (within the limits of a physical lowpass filter to approximate the ideal one used here).

The signs have worked out such that the remodulating sinusoids have exactly the same phase relation as the demodulating sinusoids. This fact suggests that an efficient architecture should make use of the same hardware to perform both the demodulation and the remodulation. In other words, a single chip can be configured either as the function decomposer or as the function reconstructor. In the instance of the continuous wavelet transform, the lowpass filter for the demodulation can be combined with the Gaussian filter of the transformation such that no additional filter is required.

2.6 Post-processing

Subsequent to complex demodulation and Gaussian filtering the system outputs are in a form useful for signal processing: for instance, analog methods of compression wherein signal bands with energy less than a critical threshold can be eliminated to save bandwidth prior to transmission and reconstruction. As described thus far, the Gaussian CWT is a band equalizer, but with all the outputs occupying frequency space around zero frequency. Thus the outputs are not in a form suitable for efficient transmission, as they all overlap in the frequency domain. There are two ways to arrange the outputs for transmission to the reconstruction system:

1. Modulate the signals into nonoverlapping frequency bands

2. Sample the system and time-multiplex the samples into an efficient representation.

The first method is the method of reconstruction, although the separated channels can be modulated to any desired transmission frequency and ordered in any manner or compressed by the scheme mentioned above to reduce the total transmission bandwidth. The second method allows more flexibility in the representation by interleaving samples. It is also more faithful to the idea of *tiling* the time-frequency plane: the bandpass filterbank quantizes the frequency domain; a sampler

2.8 Generating carrier sinusoids

This chip, as mentioned above, used square waves (clock signals) as its basis for generating sinusoids at specific frequency and phase. Specifically, the architecture called for each channel center frequency to be half the value of the neighboring channel. Additionally, the architecture was made such that each wavelet processor core would generate center frequencies for six channels. The master clock is applied to the system most conveniently as a two-phase, non-overlapping signal which becomes the first input to a cascade of toggle flip-flops (T-FFs) made in the standard way from two transparent latches in series. The two-phase flip-flops conveniently give outputs from the first and second latches which are 90° out of phase with each other (this is necessarily true for the input of the second stage and beyond because each of the two-phase outputs of the previous stage is forced to be 25% duty cycle, and the two phases are exactly 180° apart. The same can be ensured for the first stage by doubling the master clock frequency and preceding the first stage with another toggle flip-flop). The flip-flop configuration is shown in Figure 2.7. Signals *cos(in)* and *sin(in)* are the pulse trains corresponding to the frequency and phase of the sine and cosine components of the preceding channel. Signals *cos(out)* and *sin(out)* are the pulse trains which are shaped by filtering and become the modulating signals for the current channel. The reset login ensures that the sine and cosine parts have the correct phase relative to each other upon initialization of the circuit.

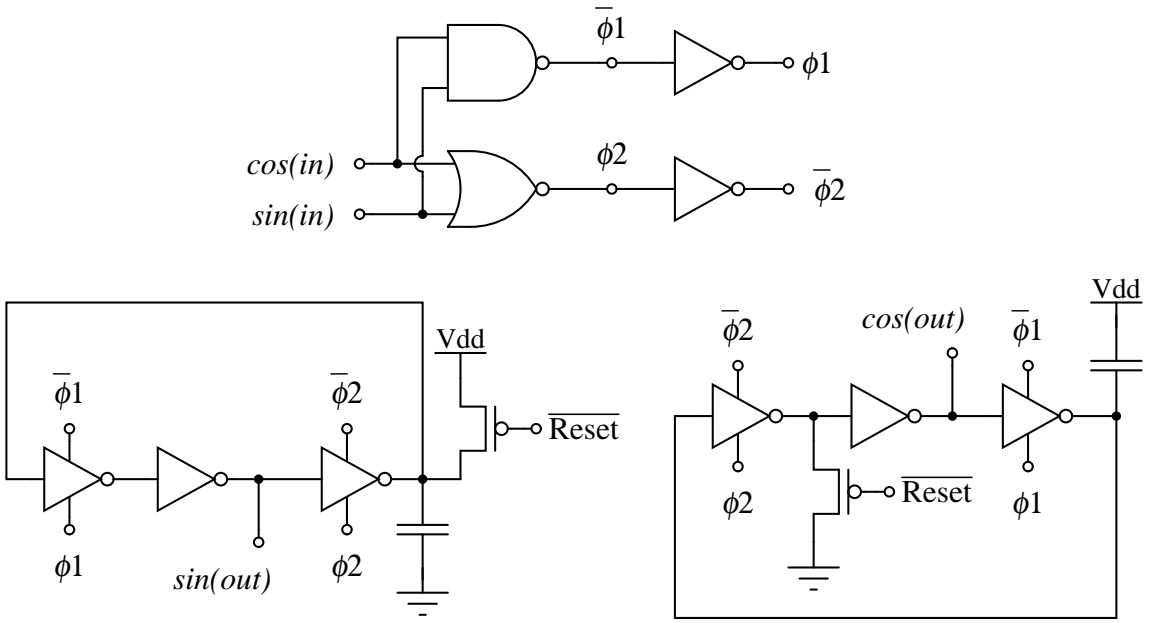


Figure 2.7: Circuit diagram of the frequency-division toggle flip-flops.

allowing variable center frequency and width. Because the signals are first demodulated until the center of the frequency band of interest is placed at zero frequency, it is only necessary to create a Gaussian lowpass filter, which is one half of a Gaussian function placed at zero frequency. Two identical filters are required for the sine and cosine parts of the complex demodulation. The signals are remodulated to their respective center frequencies during reconstruction using the same method (and preferably the same hardware, if feasible). The resulting output will be reflected symmetrically across the modulation frequency, behaving as if it had been passed directly through a bandpass filter with Gaussian characteristics. For demodulation used in the context of the continuous wavelet transform, the lowpass filter for the demodulation can be combined with the Gaussian filter required by the wavelet transformation. Note that, according to Equation 2.33, reconstructing the signal does not require any filtering subsequent to remodulation.

We based the design of the circuit which approximates the half-Gaussian lowpass filter function (as described by Grossman [13]) on a probability argument. First, I present Equation (2.34) which describes the filter transfer function of the circuit:

$$H(s) = \frac{V_{\text{out}}}{V_{\text{in}}} = \left(\frac{1}{\tau s + 1}\right)^n. \tag{2.34}$$

This filter function describes a cascade of n first-order lowpass filter sections in series. Although Equation (2.46) converges to a delta function in the limit as $n \to \infty$ for constant τ, it can be shown that when τ is replaced by an expression which maintains constant bandwidth, the transfer function approaches the Gaussian function (2.5) as $n \to \infty$. A proof of this equation can be found in [31] which shows that the Gaussian shape is an example of the central limit theorem of probability: In other words, it is a result of the use of cascaded stages and is relatively independent of the shape of the filter.

For n sections in cascade, the relationship between τ and the Gaussian bandwidth σ from the wavelet mother function, Equation (2.5), is

$$\sigma = \frac{1}{\tau\sqrt{n\ln(2)}}. \tag{2.35}$$

The circuit is shown in part in Fig. 2.11. Although not an architectural necessity, we chose to implement each first-order filter as a transconductance-C filter using transconductance amplifiers operating in the subthreshold region. The transconductance amplifiers are connected as voltage followers, which in conjunction with the capacitor at the output is a configuration called a "follower-integrator." Due to considerations of signal-to-noise ratio and the ability to generate a

given constant τ from a voltage applied to the follower-integrators, we chose a cascade order of five sections.

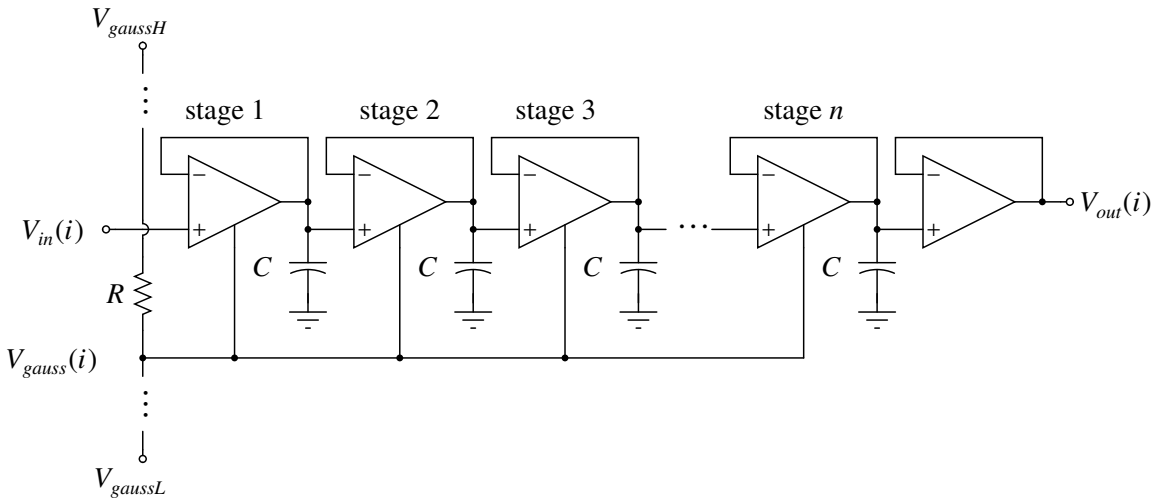


Figure 2.11: n cascaded stages of a filter approximating a half-Gaussian function, using continuous-time transconductance-C filters.

The center frequencies of the filters in the filter bank are spaced on a $\log_2$ scale. Assuming speech-quality bandwidth for the input signal, we decided that six outputs would be sufficient, giving typical center frequencies of 9kHz, 4.5kHz, 2.25kHz, 1.125kHz, 562.5Hz, and 281.25Hz (unless more than one system is interleaved in the manner described in Section 2.22). The center frequency of the highest-frequency filter is determined by an oscillator which can be generated either on-chip for a voltage-controlled frequency, or off-chip for a stable frequency. The center frequencies of the rest of the filters are determined by dividing down the oscillator appropriately.

The bandwidth of each Gaussian filter is set automatically with respect to the others with the exception of the first and last filters, which have widths adjustable using two control voltages V_{gaussH} and V_{gaussL}. In terms of the transfer function (2.46) for the filter, the parameters τ of the highest- and lowest-frequency filters are fixed by these control inputs. τ should be calculated to assure that the width of each Gaussian is proportional to the value of the center frequency, as it is not determined automatically from the other system parameters.

More recently, this idea has been expanded upon by Harris *et al.* [22] for the creation of so-called "gamma-tone" filters and filter structures which use carefully calculated weighted feedback and feedforward connections to cause the filter transfer function to reach a given accuracy of approximation of the Gaussian shape in significantly fewer stages.

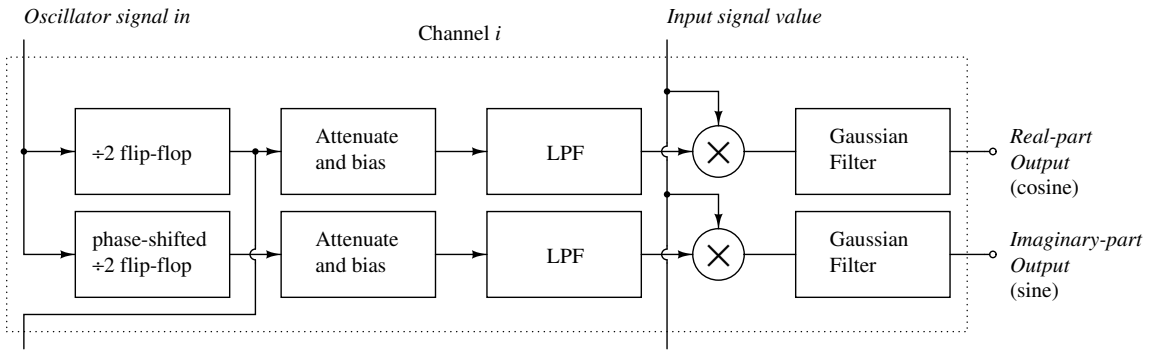


Figure 2.12: Wavelet decomposition block diagram for a single sine/cosine pair.

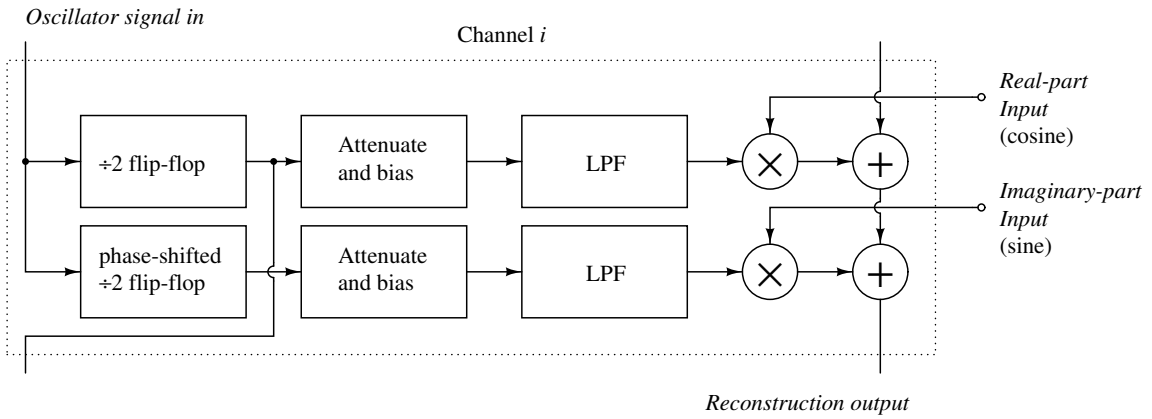


Figure 2.13: Wavelet reconstruction block diagram for a single sine/cosine pair.

Table 2.1: Technologies used for the Wavelet processors.

Chip name	*Foundry*	*Min. feature size*	*Well type*	*Poly*
WaveChip2	Zilog	2.0 micron	twin-well	single
Z89c55ba	Zilog	1.2 micron	twin-well	single
WaveChip5a	Orbit	2.0 micron	N-well	double
WaveChip5b	Orbit	2.0 micron	P-well	double

applications; 60 to 70 dB for high fidelity). Complicated filter architectures were not addressed in this series of wavelet processors, and so results did not reach acceptable levels of performance.

2.14 Variations on an Architecture

One interesting variation on this design was created and developed by Moreira-Tamayo *et al.* [23, 24] at Texas A&M University. Using primarily the same complex demodulation technique based on our architecture, the authors translated the entire problem into the time domain. In place of the wavelet Gaussian function, they used a rectified cosine (also known as a *Hanning window*, a function which is reasonably easy to generate in the time domain and which is reasonably close to the Gaussian in its time and frequency support. Its time-frequency product is approximately 0.513, or 2.6% larger than the minimum area (1/2) of the Gaussian. The rectified cosine itself becomes the envelope function; the wavelet function itself is generated by multiplying the rectified cosine by a sinusoid of higher frequency. Effectively, this is a double demodulation, and simplifies the implementation somewhat by making use of the same hardware architecture for generating both the Gaussian (or in this case, Hanning) envelope and the wavelet modulating function. Their wavelet family can be described by the function:

$$\psi(t) = Ae^{-j\omega_c t}\left(1 + m\cos\left(\omega_p t\right)\right), \qquad -\pi < \omega_p t < \pi \tag{2.36}$$

where this function is repeated (chained) in time, and the system repeated over dilations of the frequency. The wavelet computation on an input function $f(t)$ for frequency scale a and time shift b is written

$$CWT(f,a,b) = \frac{1}{\sqrt{a}}\int_{\ell_i}^{\ell_f} f(t)\cdot\left(g\left(\frac{t-b}{a}\right)\ v\left(\frac{t-b}{a}\right)\right)dt \tag{2.37}$$

where $g(t) = \exp\left(-j\omega_c t\right)$ is implemented as a complex demodulation by separating the function into $\sin\left(\omega_c t\right)$ and $\cos\left(\omega_c t\right)$, and $v(t) = 0.5(1 + m\cos\left(\omega_p t\right))$. The architecture of Moreira-Tamayo *et al.* also follows our architecture in its use of a master clock divided down by flip-flops for generation of multiple frequency square wave signals subsequently filtered to generate the sinusoid modulator. However, the system is built at the component level rather than as a VLSI processor. The analog multiplications are implemented with MC1494 analog multiplier integrated circuits. The integration operation in Equation (2.37) is implemented by a transconductance-C circuit. A slightly simplified block diagram of one channel of the system is shown in Figure 2.15. Only the wavelet decomposition was reported in [23]. Compare Figure 2.15 to the block diagram of the decomposition half of our wavelet decomposition architecture, Figure 2.12.

down to zero. Each demodulation result is then filtered by a lowpass version of the wavelet function, which also serves as the postmultiplication filter for the demodulator. Signal information on both sides of the channel center frequency is preserved by using complex demodulation, in which each channel is split into two parts, with the modulating sinusoid of one being 90° out of phase with that of the other. The CWT is invertible: The reconstruction process consists of modulating the channel outputs back to their respective center frequencies, and summing them all together. The decomposition method is shown in Figure 2.5 and the reconstruction method in Figure 2.6.

The wavelet nature of the output allows the channel outputs to be efficiently encoded. The outputs of the decomposition are time sampled at the Nyquist rate of each channel, and all the sampled channel outputs are time-multiplexed into a single stream. The reconstruction processor decodes the data stream prior to reconstruction.

As we developed the oversampling architecture described in the previous section, it became clear that, as the carrier signal was both binary and discrete-time based on a well-defined synchronous clock, the most efficient architecture would maintain the discrete-time nature of the carrier, and retain a digital mode of processing throughout as much of the architecture as possible. Following this line of reasoning leads to an elegant and extremely efficient design for a complex demodulation multiplier and Gaussian filter.

In the introduction to this chapter, I mentioned that by using complex demodulation, very accurate center frequencies for the bandpass function derive directly from the carrier signal frequency, which itself is derived from a quartz oscillator. In the previous section, the wavelet transform architecture was developed, but the circuits using that architecture could not achieve acceptable performance due to intermodulation products caused by poorly attenuated harmonics of the carrier signal. Analog multipliers built from analog circuits by nature have a limited range of operation due to nonlinearities in the circuit function. It remains to be explained how to get from the frequency-accurate, digital-domain square wave clock signal to an accurate and repeatable sinusoidal signal. That is the purpose of this section.

We developed a method for sinusoidal modulation of analog signals which does not require an explicit multiplication, and hinges on generation of accurate analog sine wave using the mixed-mode technique of oversampling.

As demonstrated by repeated failure of performance of previously fabricated versions of the wavelet processor, obtaining a sinusoidal signal by lowpass filtering a square wave introduces too much distortion to make the system usable. The modulation function is sensitive to distortion harmonics, which get multiplied by the input signal and act like many separate modulations about

of information about the encoded signal.

The above phrase "in the context of our system" is critical in that it is not to be assumed that the same sequence is optimal for every possible implementation. The frequency-domain representation of the original (unfiltered) single-bit oversampled sequence can be determined by converting the bit sequence directly to a sequence of voltages in the time domain according to the following rules:

- bits '1' are represented by some fixed positive voltage
- bits '0' are represented by some fixed negative voltage
- each voltage is applied for a fixed length of time t such that the entire sequence of n bits has length $T = nt$.

We then compute the FFT of this sequence. A sequence which accurately encodes a sinusoid should show a large FFT component at the frequency of the sinusoid (which without loss of generality we consider to be the inverse of the sequence length, $f_0 = 1/T$), show a highly attenuated response at all frequencies close to the fundamental, with the harmonic distortion increasing at large frequencies (an inherent property of oversampled signals). The quality of the sinusoid in the context of the system cannot be determined from this FFT result. The oversampled sequence must be lowpass filtered to retrieve the encoded signal, and so the quality of the resulting signal depends both on the binary sequence used and the properties of the lowpass function used to retrieve the analog signal from its oversampled representation. Furthermore, the input signal must be prefiltered with a lowpass to attenuate components which would mix with the oversampled sequence and add to the resulting harmonic distortion. The point is that the harmonic distortion is determined by the process of the modulation itself—specifically, the way that it causes frequencies of the input to mix with harmonic distortion components of the carrier to become error components at the output—and so the binary sequence required for optimal performance is intimately tied to the system itself.

One fallout of this consideration is that the bit sequence used to encode the sinusoid cannot be determined until the system is known in detail. On the other hand, it is not possible to know the exact requirements for the filtering system (for instance, size requirement of the VLSI layout) until we know what the sequence is. Some iterations can be expected before a solution is agreed upon. This fact makes it necessary to have a method which can fairly rapidly determine the optimal sequence.

sinusoid, and the difference between the two gives a rough indication of the minimum distortion possible using a length-n sequence.

The way we have defined the system in Section 2.15, a DC value of zero is represented exactly by a bit sequence of alternating ones and zeros ($\pm$ voltage). A DC value of one is represented exactly by a bit sequence of all ones. Given this consideration, the expected ratio of ones to zeros in a bit sequence in the first case is 1/2, and in the second case 1. Following this line of reasoning, the expected ratio of ones to zeros in the first quarter of the sine sequence is

$$\int_0^{\frac{\pi}{2}} (1 + 0.5 \sin x) dx. \tag{2.38}$$

The total integration divided by the interval comes to $1/2 + 1/\pi$, or 0.8183. This multiplied by 64 bits, for example, yields an expected value of between 52 and 53 'one' bits, leaving 12 or 11 'zero' bits. This is what we can expect the makeup of near-optimal sequences to be.

Comparing the integration under the curve to the instantaneous average of the bit sequence at each step and then determining the next bit of the sequence accordingly is exactly the method of sigma-delta modulation. The bit sequence produced is always changing from cycle to cycle due to the residual error between the filtered bit sequence and the integral under the sine curve, which is never zero.

A sequence of n bits which is repeated exactly on every cycle can never achieve the accuracy of an n-bit delta-sigma modulator. On the other hand, as n increases, so does the accuracy, so that if the system requirement is a maximum fixed value of total harmonic distortion, there exists a fixed bit sequence of some minimum length n which will meet that requirement.

The problem in finding the sequence is this: A sigma-delta system is deterministic: The next bit in the output sequence is an exact function of the current state of the system. But a sigma-delta-like method cannot be used to find an optimal fixed sequence, because changing any bit in the fixed sequence changes the past, present, and future state of the system. Instead, it is necessary to search the space of 2^n bit sequences to find one which achieves the desired accuracy.

Knowing the expected number of 'one' and 'zero' bits in the sequence still doesn't help much in finding the optimal sequence: The set of all combinations of m 'zero' bits in a sequence of length n remains computationally intractable for reasonable values of n (such as 64).

The modulation system for the CWT, when using the oversampled sequence method, is described by

$$y(t) = \left[(x(t) * g(t)) \cdot s'(t)\right] * h(t), \tag{2.39}$$

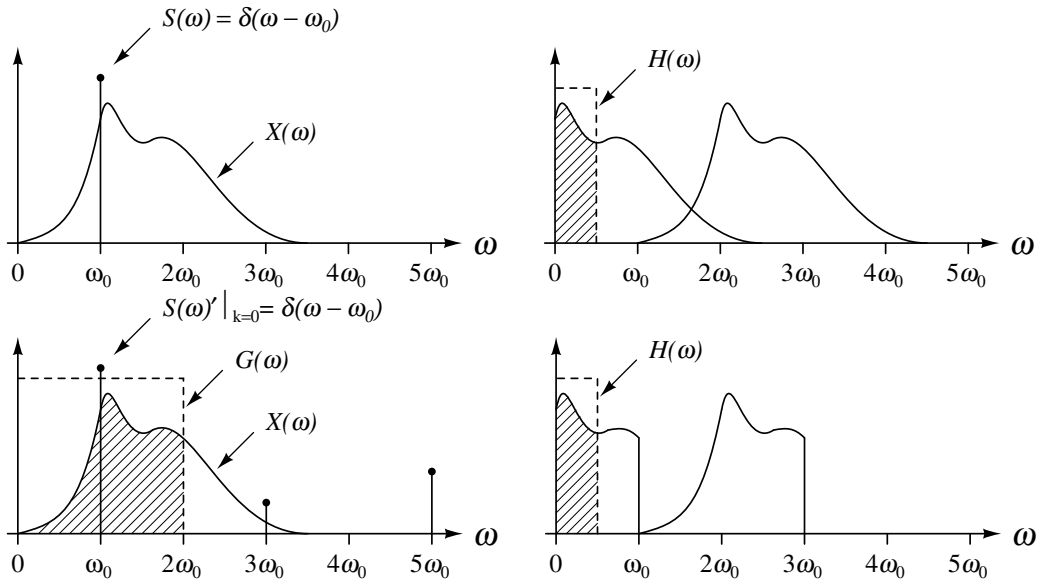


Figure 2.17: *Top*: Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H. *Bottom*: Demodulation of an input X in the frequency domain with an oversampled sine wave S' using an ideal smoothing filter G and modulation filter H.

except for large values of k for which the terms of (2.43) are reasonably small due to the attenuation by the prefilter G. In other words, the low-frequency components of the binary sequence $s'(t)$ need to approximate the sine wave $s(t)$ as closely as possible. Qualitatively, this corresponds to an oversampled noise-shaped sine wave, as produced by the delta-sigma modulator method [26].

Techniques for deriving periodic sequences with several zero or small harmonic components of c_k are presented in [33]. We formulate the problem of finding an optimal binary sequence directly from a minimum distortion criterion on the intermodulation components (2.43).

In general, the amount of distortion is input dependent, and assumptions need to be made on $X(\omega)$ to formulate an optimization criterion. Our criterion is to maximize the ratio of energy in the fundamental harmonic modulation component ($k = 0$) to the combined energy of the distortion components ($k > 0$). Assuming a narrow bandwidth of $H(\omega)$ and an input spectrum $X(\omega)$ which in the worst case is flat in amplitude, the criterion becomes:

$$\text{Maximize}: \quad \frac{c_0^2 \left|G(\omega_0)\right|^2}{\sum_{k=1}^{\infty} c_k^2 \left|G\left(\left(2k+1\right)\omega_0\right)\right|^2}. \tag{2.44}$$

which is equivalent to minimizing the harmonic distortion of the sequence $s'(t)$, filtered with the same prefilter $G(\omega)$. The criterion can be applied, in principle, to select the optimal bit sequence

sequences. All harmonics of the filtered sequence are more than 60 dB below the fundamental. The magnitude of the prime harmonic of the sequence is approximately 1.02, which is within 2% of unity. This indicates that if the binary sequence is made of voltage levels $\pm V_{seq}$, then the resulting sine wave will have a zero-to-peak amplitude within 2% of V_{seq}.

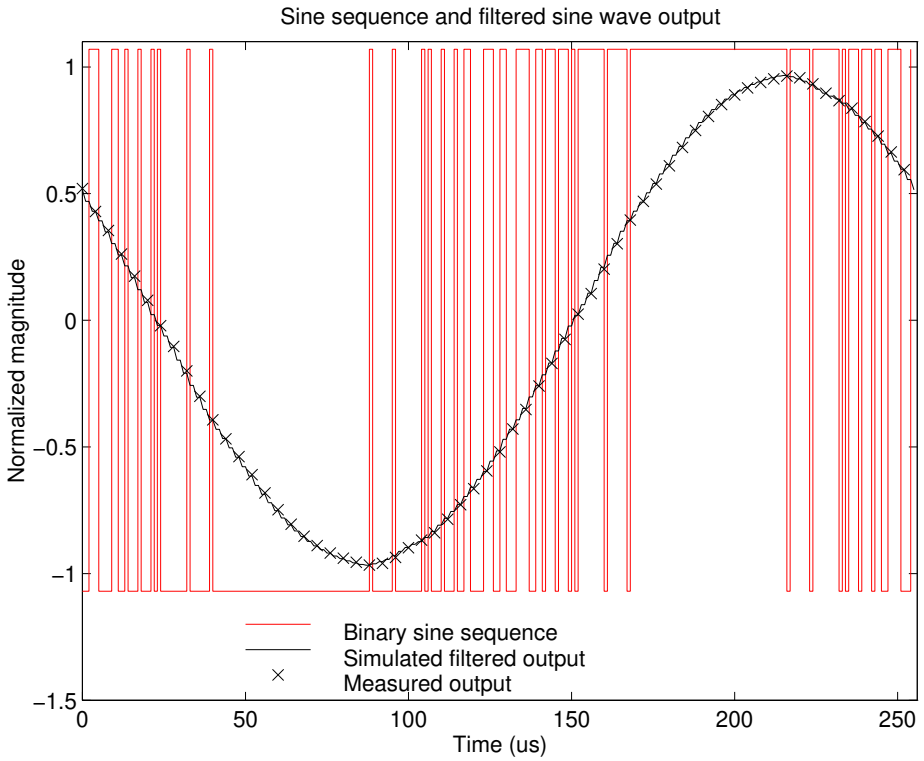


Figure 2.18: The oversampled sine sequence.

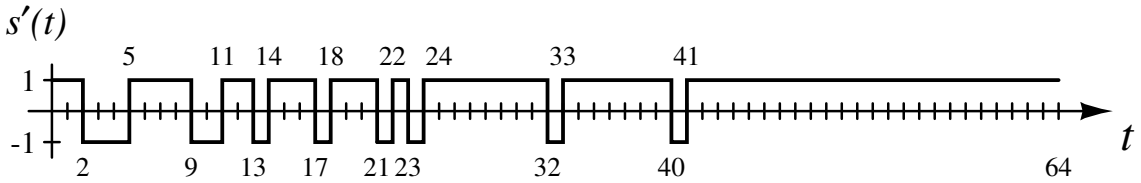


Figure 2.19: Optimized 64-bit oversampled sine sequence, first quadrant.

The advantages of using an oversampled modulation sequence rather than a simpler binary sequence can be appreciated by following comparison. To obtain the same 60 dB linearity performance with, say, a square wave modulator, a premultiplication filter G with much sharper rolloff such as a fourth-order Chebyshev or a 6th-order Butterworth would be required to compen-

complex modulation with both sine and cosine components, by performing inversion as above but using bit $n - 1$ of the counter. One technique is to use a sigma-delta modulator to transform an arbitrary signal into binary form [25, 26]. However, in the case of modulation the (carrier) signal is known *a priori*, so a sigma-delta modulator is unnecessary and can be replaced by a simple digital circuit which provides the multiplexer with a predetermined fixed sequence.

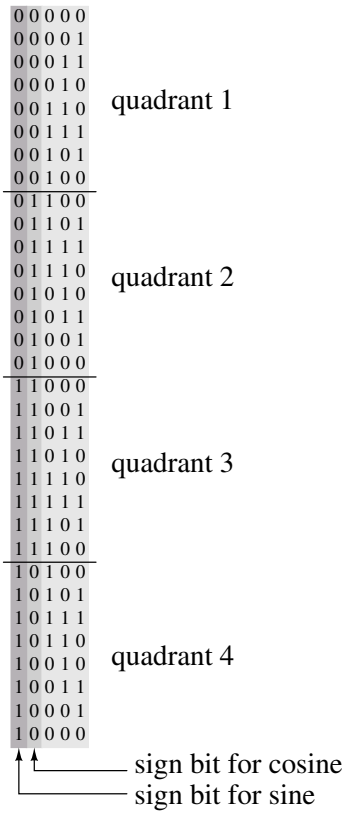


Figure 2.21: Use of Gray code to generate sine and cosine sequences.

2.19 Modulation Multiplier

The previous section described a method by which an accurate sine wave can be generated by filtering an oversampled binary sequence. Utilizing this technique immediately alleviates one of the two major problems of the analog wavelet processor. This section addresses the other problem, that of producing a linear multiplication of the input with the sinusoidal carrier signal.

Because implementation of the demodulation multiplication requires multiplication of an

A finite cascade of simple filters will approximate the Gaussian transfer function to the degree of accuracy required, provided that there is enough space in the VLSI layout to accommodate the number of filters required.

Once it was determined that the output of the modulation multiplication would be a synchronous discrete-time signal, the decision was made to change the continuous time filters into switched capacitor filters. This is done for three reasons:

- The discrete-time system scales correctly to any clock speed, so the system can be run accurately either with real-time inputs or with non-real-time inputs, such as test inputs generated by a computer. Non-real-time inputs do not even need to be strictly synchronous.
- The switched capacitor architecture has much larger linear input and output ranges than the transconductance-C filters used in the analog architecture.
- The bandwidth is controlled digitally and so can be computed directly from the system clock driving the center-frequency oscillators, as opposed to being an independently-controlled variable.

As with the continuous-time filter architecture, it is only necessary to create one half of a Gaussian centered around zero. The same argument relating to the Central Limit Theorem applies, so the switched capacitor filter retains the architecture of a set of cascaded first-order lowpass sections. Figure (2.23) shows a section of the filter we designed. This circuit is an RC lowpass filter using a simple switched capacitor simulated resistor [27]. The filter design is a discrete-time circuit and directly implements the (z-transformed) lowpass function through distribution of charge:

$$\beta = \left(\frac{1}{1+\alpha}\right), \tag{2.45}$$

$$V_{out} = \beta V_{in} z^{-\frac{1}{2}} + (1-\beta)\, V_{out} z^{-1} \tag{2.46}$$

where in the figure ϕ_1 and ϕ_2 are nonoverlapping clock signals of period T which maps to the z-domain unit delay. This circuit maps to an equivalent (s-domain) RC lowpass filter by $RC = \alpha T$.

The Gaussian lowpass filter consists of a cascade of n sections, which have a combined transfer function which converges to a Gaussian in the limit $n \to \infty$. The choice of the number of sections is a tradeoff between the accuracy of the filter with respect to a true Gaussian and the delay (and also noise) incurred by a signal passing through the cascade. We chose a cascade of eight

unused, but could be used to encode the DC level of the input. Note that a Gray code counter is as simple to design as a binary counter. In the usual implementation of a binary counter as a cascade of toggle flip-flops where each toggle flip-flop is a pair of transparent latches in series, the output of the first latch in each pair encodes a Gray code while the output of the second latch in each pair encodes the binary sequence.

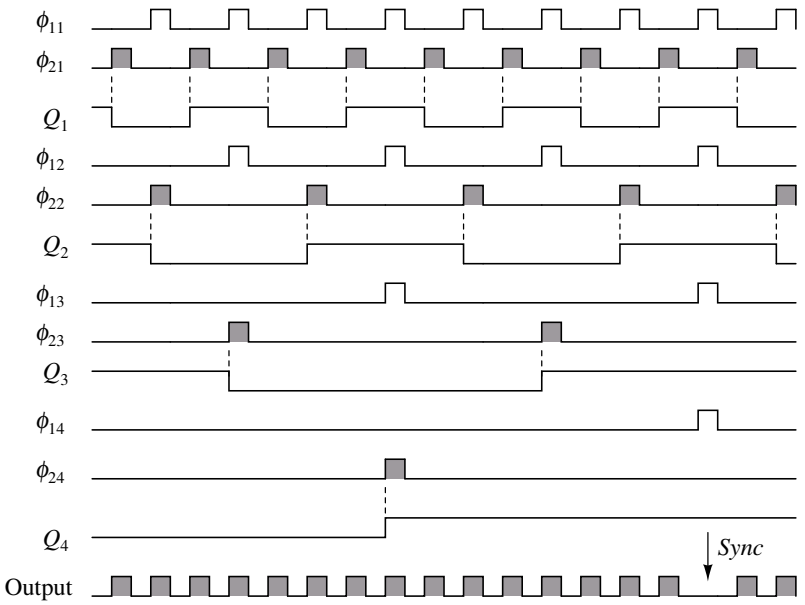


Figure 2.24: A scheme for controlling time-multiplexing of the outputs.

The same method is used throughout the chip to generate clocks which ensure that the modulator frequencies and filter bandwidths differ by a factor of 2 for each channel. An additional bonus of using Gray code is that digital noise is kept low because only one channel is clocked at a time, and power consumption is kept to a minimum due to the event-driven nature of the process.

2.22 Wavelet chip slice

The parallel nature of the analog wavelet computation makes the chip easy to create using abutting slices of circuitry. We built a bank of eight slices (channels), each containing the logic to divide down the incoming clock signals, generate the sine and cosine sequences, multiply these modulating signals with the chip inputs, filter the result with a Gaussian-shaped function, and time-multiplex the outputs on two buses (one for sine, one for cosine parts). I devised a simple way to expand the system from eight to sixteen channels covering the same total frequency span: In a

signal but are used only to reject the high-frequency noise output of the multiplier. A block diagram showing two channels of the Wavelet Transform chip is shown in Figure 2.26.

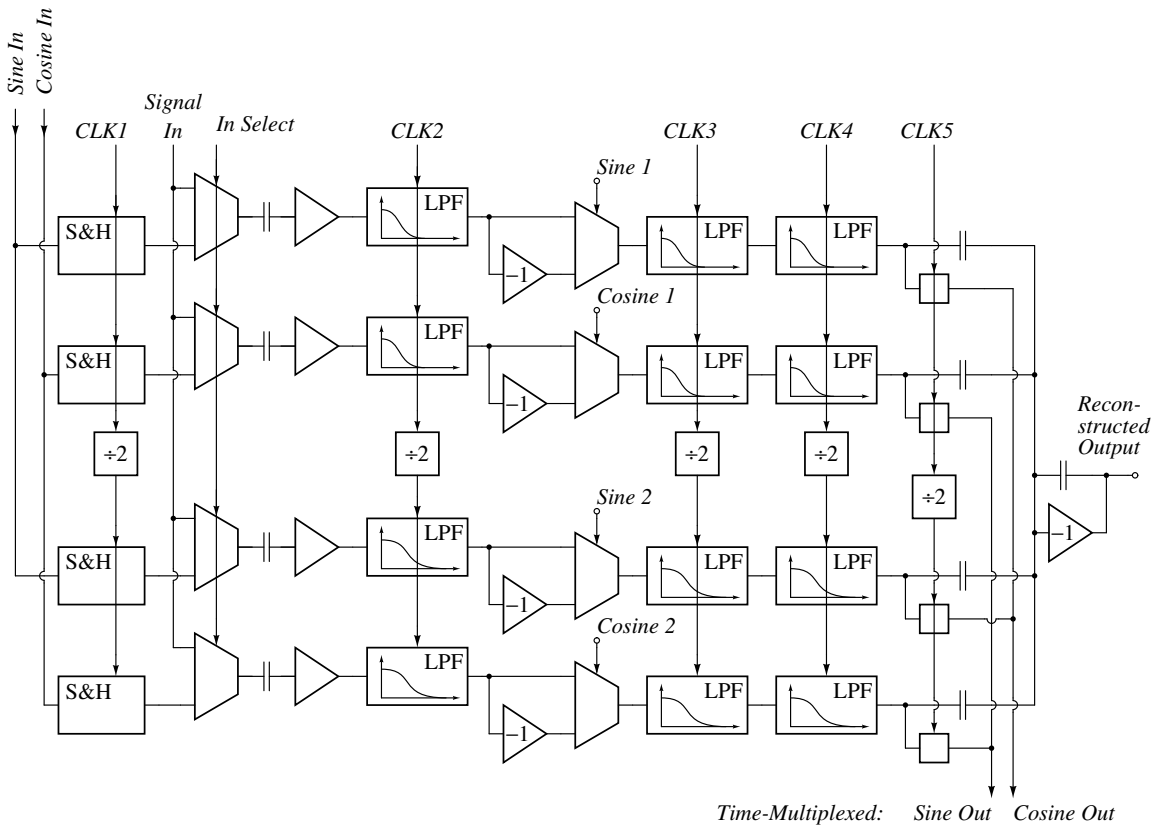


Figure 2.26: The wavelet chip, block diagram.

The circuit layout of the CWT processor fits two sections of eight channels each on a single 4 mm×6 mm die in a 2 µm CMOS p-well process, packaged in an 84-pin PGA. Figure 2.27 is a photomicrograph of the integrated circuit. Test results reported here are from this chip and a separate test chip containing a single channel.

2.23 Experimental Results

In a test of the modulation multiplication (including the postmultiplication filter), the sinusoid modulator is multiplied by a constant input. A binary sequence representing the oversampled sinusoid is produced at the multiplexer output, and is smoothed into a sine wave by the lowpass filter. Figure 2.20 shows the FFT of the test chip modulation output before ("**o**") and after ("**x**")

lowpass filtering. Distortion components of the binary sequence are attenuated by the filter to below -60 dB.

Figure 2.28 is an oscilloscope photograph showing the demodulating hardware in action. The top trace is the input signal, a sine wave generated by a function generator for the purposes of evaluating the system. The middle trace is the result of multiplying the input by the binary sequence. The function flips rapidly from positive to negative voltages, which is too fast to be captured in the oscilloscope photograph where it appears to be two overlapping sinusoids. The bottom trace is the output after filtering through the Gaussian filters, and is a sinusoid of a frequency which is the difference between the input and the carrier. The bottom trace is jagged due to the discrete-time nature of the switched capacitor hardware. Because the output is sampled at the same rate prior to time-division multiplexing into the output stream, there is no need at this point to apply a smoothing filter to the output.

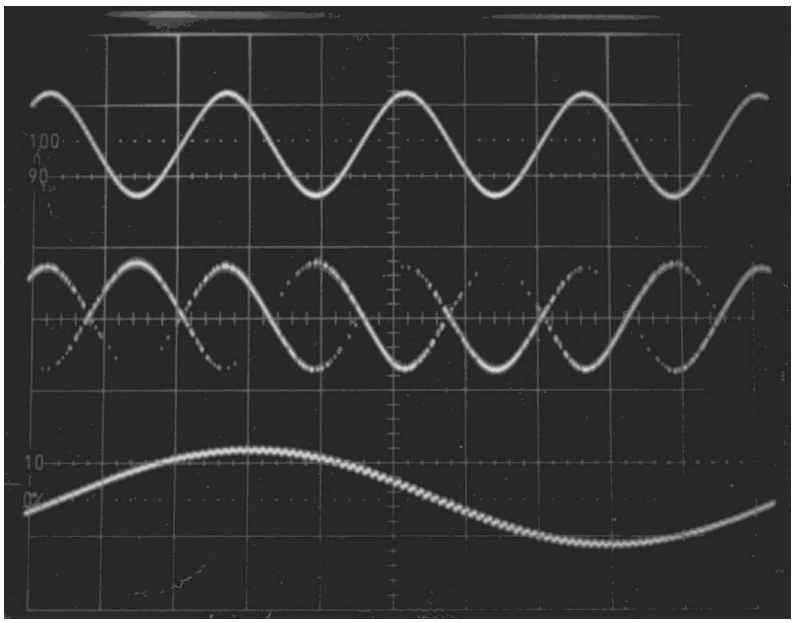


Figure 2.28: Signal demodulation using the wavelet chip. The top trace is the input signal. The middle trace is the input multiplied by the binary sequence. The bottom trace is the output after filtering.

Figures 2.29 and 2.30 show the measured transfer function (magnitude and phase) of the Gaussian filter, a cascade of eight single-pole switched capacitor filters, as compared to the predicted result for an ideal eight-stage lowpass filter cascade. Bandwidth is normalized to the clock frequency of the filter switches, showing that the shape of the filter is independent of the corner frequency.

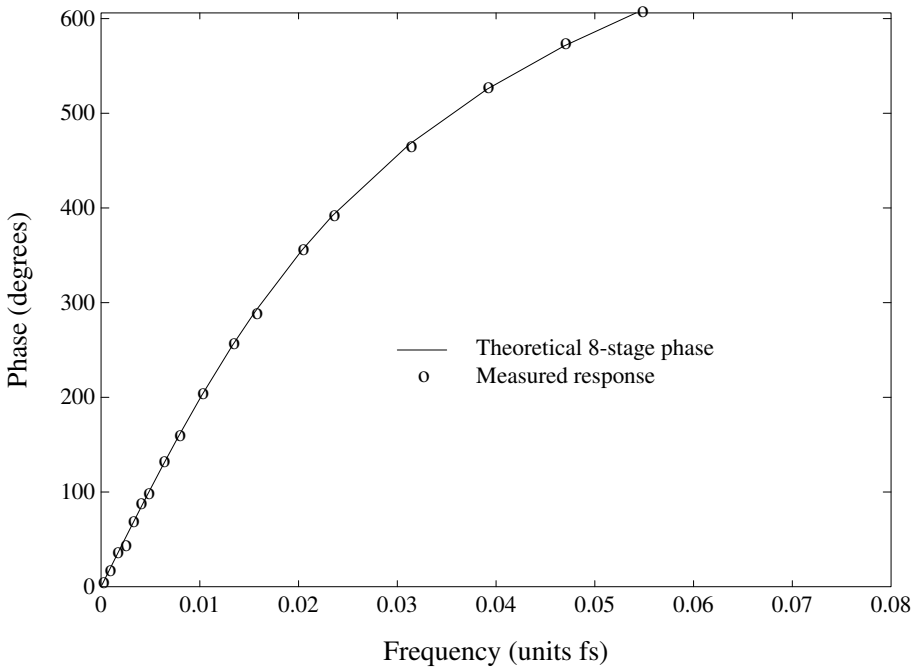


Figure 2.30: Gaussian filter phase response: predicted and measured.

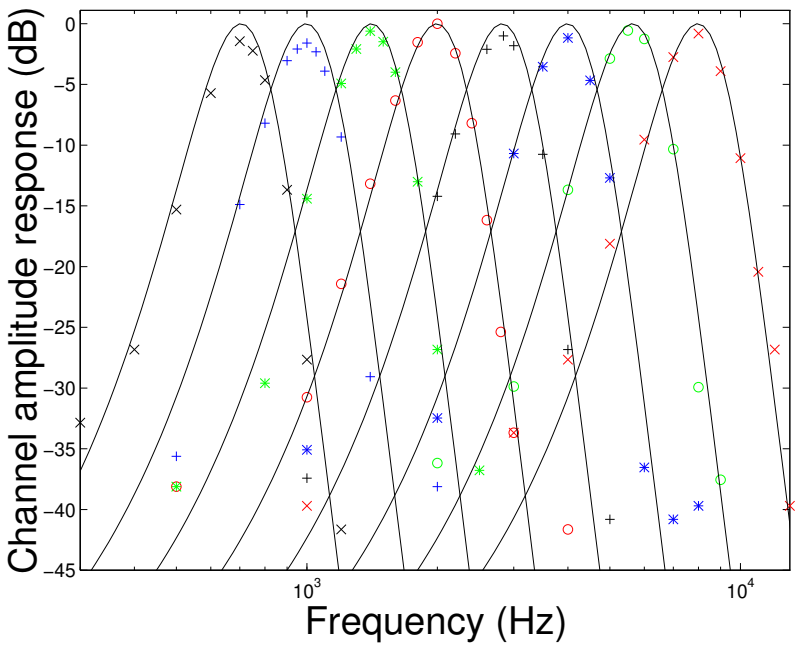


Figure 2.31: Wavelet responses to isolated sine wave input.

2.25 Summary

There is an interest in the signal processing community for hardware to efficiently perform the Continuous Wavelet Transform for applications such as speech recognition, sound processing, and data compression. The Wavelet Chip described herein is able to compute both the CWT and its inverse (wavelet decomposition and reconstruction) in hardware, generating analog, discrete-time outputs. It is built using a mixed analog/digital architecture which is small and energy-efficient when compared to possible all-digital implementations. We show that our system computes a highly linear analog multiplication with a sine wave of low harmonic distortion by using oversampling techniques to implement the modulation multiplication. We utilize the Central Limit Theorem in order to generate a Gaussian-shaped filter from a cascade of lowpass filter sections. An additional benefit of the mixed analog/digital approach is precise control over the center frequencies and bandwidths of the channels. The output of the transform is realized by time-multiplexing the multiresolution outputs of the filters in an efficient manner. Experimental results on a sixteen-channel prototype have demonstrated the effectiveness of the new architecture.

The continuous wavelet processor architecture is perhaps given more significance by ignoring the "wavelet" aspect of it altogether. That is, there are uses for the architecture beyond the decomposition and reconstruction of signals for compression, frequency shifting, and so forth. The architecture represents a novel method of modulation and demodulation which can be used, to give an instance, for simple but effective lock-in amplification (extremely high-Q bandpass filtering). The oversampling method of sine wave generation can be used for precision function generation, or even a form of parameterized audio synthesis. The use of oversampling methods in this way to circumvent limitations of analog multiplication is a method that could bear considerably more scrutiny and has a wide range of potential applications.

bandpassed) taps typically are spaced logarithmically (or more accurately, on a mel scale).

- A *Wavelet Filterbank* is any architecture which implements a wavelet transform by attempting to create a distribution of outputs which covers the time-frequency plane with greatest efficiency. By necessity, the frequency bands are spaced on a logarithmic scale.

Note that the above definitions do not *preclude* any overlap. For example, although the outputs of a cochlear filter do not encode the input with anything approaching maximal efficiency, that does not necessarily mean that the mechanics of the inner ear plus subsequent neural processing do not eventually extract a maximally efficient wavelet-like representation from them. In a way, the cascaded serial processing of the cochlear filter is more faithful to the idea of "dilations" in wavelet transformations, particularly as adopted in DWT implementations, than is the parallel filterbank structure we used for the CWT processor. Cascaded processing avoids redundancy in the filtering and is therefore decidedly more power-efficient than parallel processing. However, problems with cascaded processing, such as cumulative noise and cumulative DC offset error, often make circuit implementations prohibitively difficult.

3.2 Parallel Filterbanks for Transient Classification

About the same time that we were realizing practical circuits for the continuous wavelet transform processor and achieving good results, I began work on a project involving the implementation of an algorithm for acoustic transient classification. The algorithm, initially conceived and developed by Fernando Pineda at the Johns Hopkins University Applied Physics Laboratory, was intended to be most conveniently implemented by analog circuits. The algorithm consisted of two major parts: First, a filterbank system, only roughly specified, from which the bandpassed outputs would be rectified and smoothed to encode the average energy in each channel over time (see Figure 3.1); and second, a classification system based on template correlation. The correlation system is described in detail in Chapter 4. This chapter concerns the design and development of an analog filterbank frontend system targeted for use with the acoustic transient correlator.

Preliminary trials of the classification algorithm used a cochlear filterbank circuit dubbed with the pseudo-acronym "HEEAR" (for the "Hopkins Electronic EAR"), developed in Andreas Andreou's group at Johns Hopkins University by Weimin Liu and later developed into a capable audio signal processing system by Nagendra Kumar and Paul Furth. Once I began work designing the template correlation system, however, an elegant design began to evolve, one which required

rent conveyors have fewer transistors than amplifiers, current-mode circuits can be made extremely compact compared to their voltage-mode equivalents, if such an equivalence exists. For example, adding two currents requires no components at all, just the merging of two current-carrying wires, an operation which is perfectly linear at all scales. Adding voltages requires an amplifier circuit and has a limited range of linearity, especially when the amplifiers are CMOS transconductance amplifiers operating in the subthreshold regime (see Appendix A). The Gilbert multiplier (see Chapter 2), requires at least 17 transistors and has a rather limited linear range due to the same differential input pair configuration as a transconductance amplifier. By contrast, multiplication of currents is easily obtained with a translinear multiplier (see Section 3.6) consisting of four transistors and two current sources, and is linear over many orders of magnitude of the input, especially if implemented with bipolar transistors.

Each "school" of design has its strong and weak points, though. While the strength of current-mode design is in the compactness and elegant simplicity of its circuits and the capability of wide linear dynamic range, its weakness is in filter circuit design. One factor hampering the design of our original analog circuit-based wavelet transform chip was the combination of simple arithmetic functions such as addition and multiplication which would be most conveniently implemented in the current domain, with filtering functions best implemented with voltage-mode transconductance-C filters.

For a long time, very few current-mode filter designs were available, and those that did exist were not amenable or adaptable to CMOS VLSI technology. However, in the early 1990s, interest began to grow around the topic of "log-domain" filters. Researchers discovered simple and elegant circuit topologies which lent themselves readily to VLSI implementation, and suddenly the possibility existed to couple current-mode arithmetic with current-mode filtering to realize all manner of novel systems. Because I was aware of many potential applications for current-mode filtering in analog and mixed-mode VLSI design, I was among the first researchers to begin attempting to build complete systems around log-domain filters and current-mode arithmetic.

Unfortunately, embarking on systems design while the underlying circuit design theory was still in its infancy was a daunting and difficult task, and required research that lead to novel contributions to the field of log-domain circuit theory. The following sections summarize our investigations and contributions to this burgeoning field, directed primarily toward the goal of building practical current-mode filters and related circuits for an acoustic-frequency analog signal processing system.

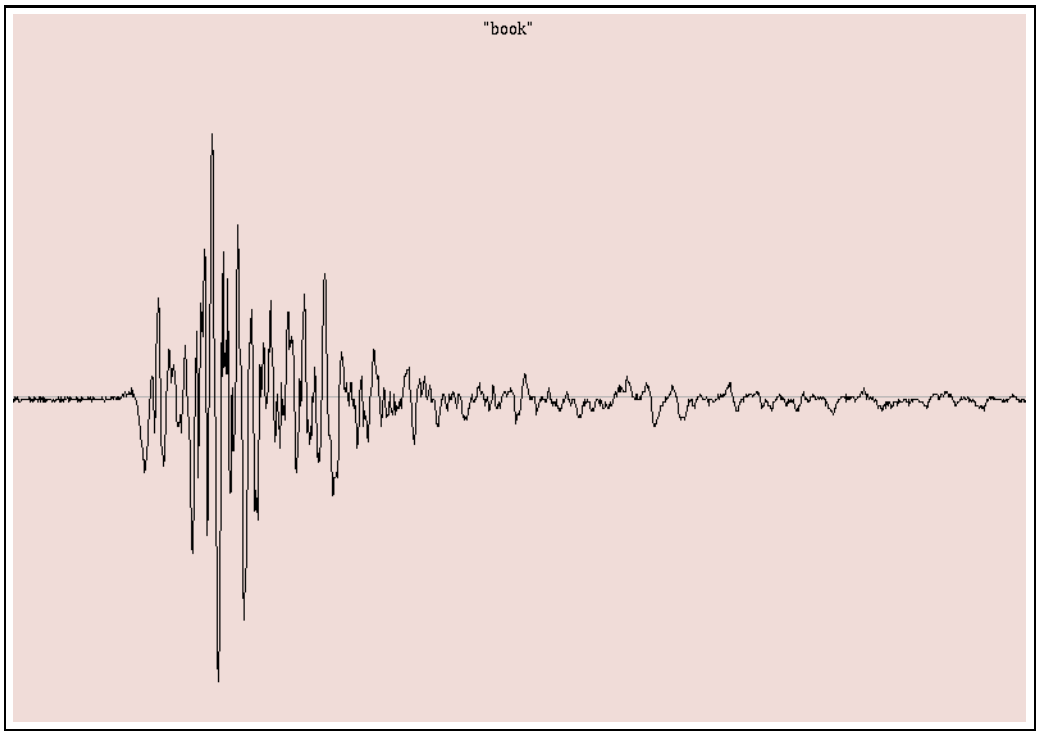


Figure 3.2: Sampled-data input from a recording of an acoustic transient—that of a book being dropped onto a desk.

The two cascaded sections have the combined transfer function

$$G_n(s) = (H_n(s))^2. \tag{3.13}$$

The result of this filtering operation on an example transient signal in simulation is shown in Figure 3.3. The algorithms which generate the filter characteristics and simulate the filter operation are detailed in Appendix D.

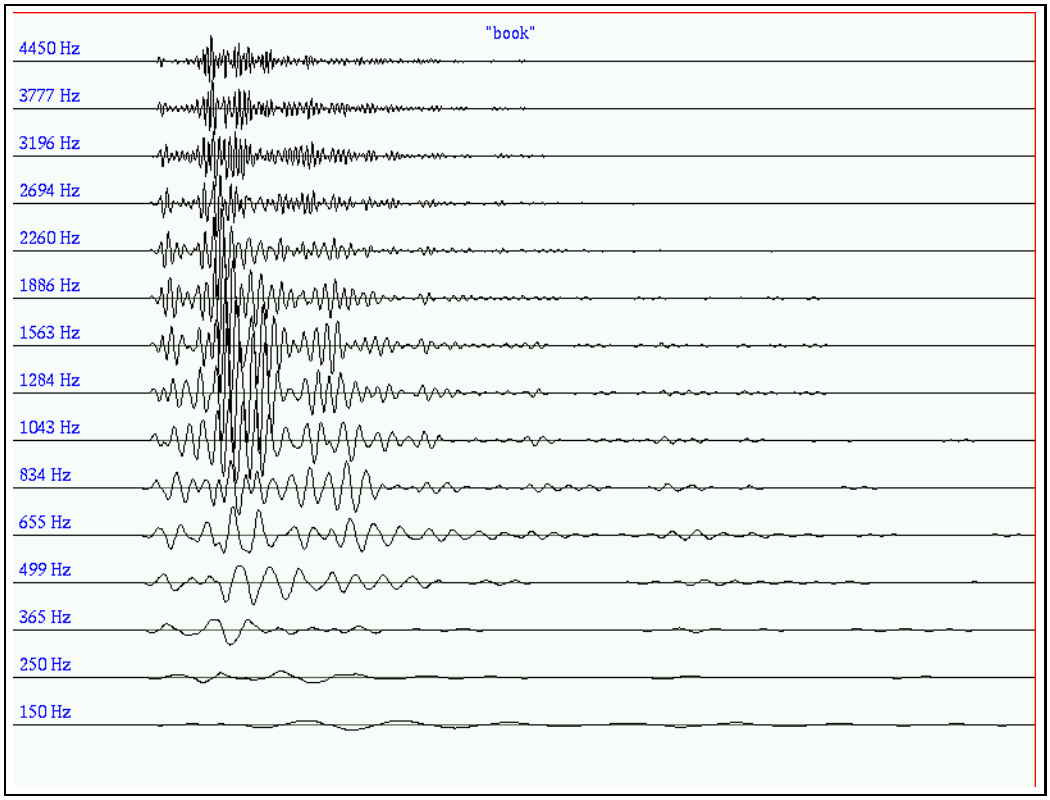


Figure 3.3: Bandpass-filtered acoustic transient input using two cascaded second-order filters on each channel.

The signal rectification process is trivial in simulation since the data are zero-mean after bandpass filtering. The lowpass function is, like the bandpass function, an s-domain filter function describing the analog hardware, transformed into the z domain via bilinear transform. Our choice of smoothing filter is exactly the same as that used for the continuous wavelet transform system: simple first-order lowpass sections in cascade. Each first-order filter has the transfer function

$$H_n(s) = \frac{1}{1 + \alpha_n s} \tag{3.14}$$

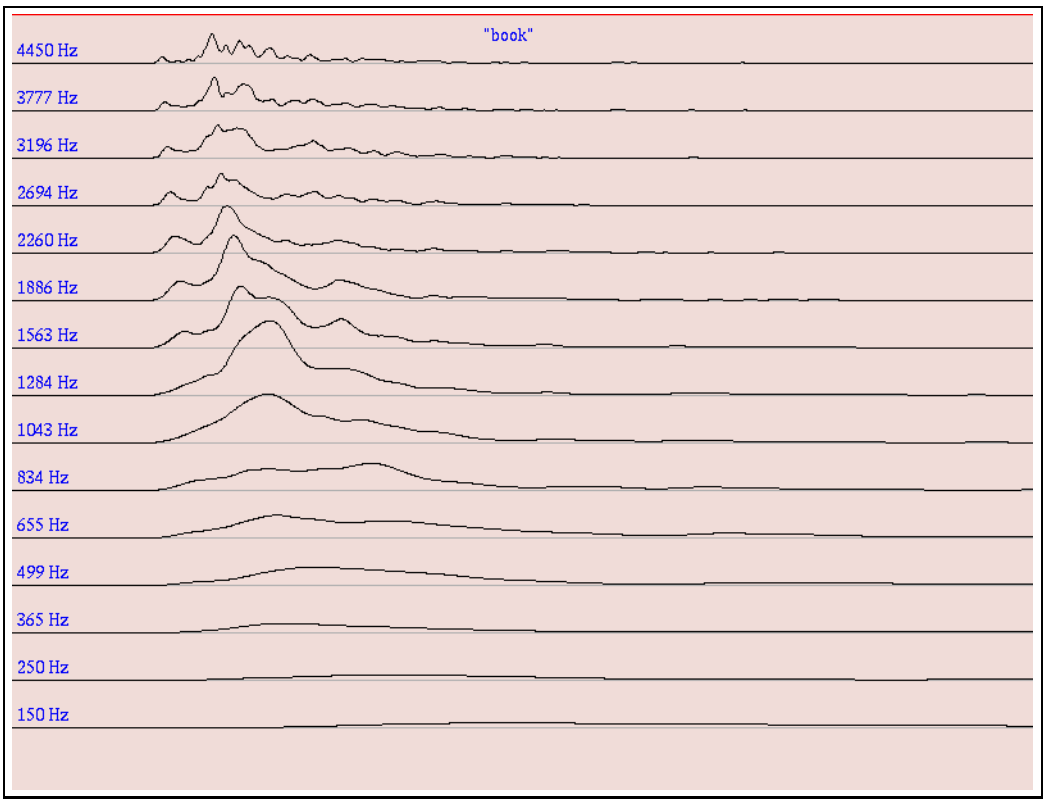


Figure 3.4: Parallel filterbank output after rectification and smoothing of the acoustic transient input signal across all frequency bands.

the system. The equations governing the internal nonlinearity of the system are generally tractable, leading to complete solutions which do not require separate DC and transient analyses.

The foundation of log-domain filter theory was laid more than 20 years ago with the formulation of *translinear loop* circuits by Barry Gilbert [3, 35], which made use of the fact that the I-V equation of the bipolar transistor is an excellent approximation to an exponential over a large number of decades of current; coupled with Kirchoff's Current Law (KCL), it enables a number of mathematical operations to be performed by a small number of transistors. One important circuit based on the translinear principle is the Gilbert multiplier 2.8, whose operation is based on the fact that a multiplication of n numbers can be performed by taking the logarithm of the numbers, adding them all together, and then taking the antilog (exponential) of the result. Conceptually, this seems like a roundabout way to multiply numbers together. But because it relies on one easily-obtained linear operation (summation via KCL) and two easily-obtained nonlinear operations (log and antilog via the bipolar transistor I-V characteristic), the result is that two currents can be multiplied quite simply by this indirect method, while any attempt at a direct multiplication using linearization through feedback is extraordinarily complicated by comparison.

Substantial progress has been made in simplifying the processes of synthesis and analysis of log-domain filters [38, 39, 41, 42, 43, 44] by recognizing that they are by necessity composed of translinear loop circuits. The use of translinear circuit theory circumvents the necessity of explicitly solving transistor I-V equations, instead replacing them with mathematical functions on a more symbolic level, such as log, antilog, sum, and multiply. The translinear loop circuit analysis, as described by Gilbert, pertains only to static circuits, and only goes so far in describing time-dependent circuits such as filters. The crucial step in simplifying log-domain filter synthesis and analysis is to formulate a translinear principle which applies to time-derivative systems. This "dynamic translinear principle" was developed in Mulder *et al.* [38], and is worth paraphrasing here in the context of our method of circuit synthesis, where it allows us to synthesize filter circuits without resorting to state-space manipulation (*c.f.* [41]).

3.6 Principles of log-domain synthesis

A translinear loop is a circuit loop in exactly the same sense as Kirchoff's Voltage Law (KVL): it describes a circuit which originates at some point, passes through a number of circuit elements with associated voltage drops, and returns to the same point with a total sum of zero volts around the whole loop (*i.e.*, potential, and therefore energy, is conserved). In a translinear loop,

Substituting the idealized transistor equation results in an expression for the currents in which the logarithm function can be eliminated due to the property of logarithms relating the sum of logs to the log of products. This is the "static translinear principle":

$$\sum_{i\,\mathrm{odd}} V_t \, \ln\left(I_i/I_s\right) = \sum_{i\,\mathrm{even}} V_t \, \ln\left(I_i/I_s\right); \tag{3.21}$$

$$\prod_{i\,\mathrm{odd}} I_i = \prod_{i\,\mathrm{even}} I_i. \tag{3.22}$$

As Equation (3.22) shows, the static translinear principle gives a simple rule for computing products of currents, revealing a minimalist way of multiplying the value of two currents (always scaled by another current, so that the result is a current, not an unmeasurable or impractically-measurable unit like amperes squared). To design filters, or, for that matter, any dynamic system, we need an equivalent, simple rule to generate equations involving time-derivatives of currents, *i.e.*,

$$I_{out} \propto \dot{I}_{in}. \tag{3.23}$$

Fortunately, such a simple rule exists, one which again hinges on the use of an arithmetic property of logarithms/exponentials to simplify resulting expressions. The key to the problem is the property of the derivative of an exponential function. Applying the time derivative to the simplified transistor equation results in the "dynamic translinear principle":

$$I_{out} = I_s \, e^{(V_{be})/V_t} \tag{3.24}$$

$$\dot{I}_{out} = \frac{d}{dt}\left(I_s \, e^{(V_b - V_e)/V_t}\right) \tag{3.25}$$

$$= \frac{1}{V_t} I_{out} \frac{d}{dt}\left(V_b - V_e\right). \tag{3.26}$$

Grounding the BJT emitter ($V_e = 0$) and adding a capacitor C to the system at node V_b ($I_C = C\dot{V}_b$) yields an equation composed entirely of current-mode variables. Figure 3.6 shows such a system, a basic building-block of log-domain filters (*c.f.* [39], Fig. 1, and [38], Fig. 1). It should be noted at this time that a constant voltage V_{shift} (as shown in Figure 3.6) may be inserted between the capacitor node Z and the transistor base. This constant is canceled on both sides of the equation and therefore does not affect the solution:

$$\dot{I}_{out} = \frac{1}{V_t} I_{out} \frac{I_C}{C}. \tag{3.27}$$

From this equation we can compute the derivative of a current by multiplying two currents (I_{out} and I_C) together. As mentioned earlier, multiplication of two currents can be easily accomplished with a translinear loop circuit.

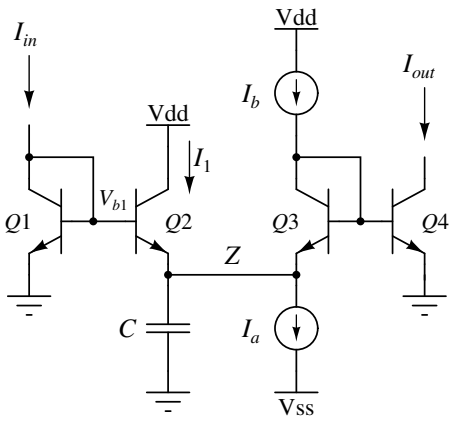


Figure 3.7: First-order log-domain filter circuit.

does indeed have the correct units of time. Substituting τ into Equation (3.30) and multiplying through by I_b, we get

$$I_{out}\left(AI_b + I_C\right) = I_{in}I_b. \tag{3.32}$$

Equation (3.32) has a familiar form, that of the four-component translinear loop, Equation (3.22), with the constraint that I_C and I_{out} must have the relationship shown in Figure 3.6. One of several possible implementations is the "up-down" (common-base) configuration shown in Figure 3.7, in which transistors $Q1$ through $Q4$ form the translinear loop $I_{out}\, I_1 = I_{in}\, I_b$. $Q3$ is a voltage level-shifter (*c.f.* [39], Fig. 2), and Equation (3.28) is satisfied if $I_a = I_b\,(1 + A)$. Thus, the value A is tunable and controlled by the ratio of I_a to I_b, both of which are unconstrained constant bias voltages. Useful values of A are $A = 1$ at $I_a = 2I_b$ (a lowpass filter) and $A = 0$ at $I_a = I_b$ (an integrator, useful in systems with feedback such as second-order filters).

The negative power supply, $\mathrm{V_{ss}}$, is a somewhat arbitrary value which needs to be kept far enough below ground that any transistors used as current sources which draw current off of the capacitor node Z can operate correctly. For a simple, single nMOS transistor used as the current source I_b, $\mathrm{V_{ss}}$ should be about $-0.5\,V$.

Translinear loop equations break down as voltages across the translinear element approach zero because the diode equation, which holds (with different premultipliers) for all three types of translinear elements, differs from a true exponential by a single "-1", which is the manifestation of the impossibility of having currents move against the direction of the applied voltage across the junction:

$$I_c = I_s\left(\exp\left(\left(V_b - V_e\right)/V_t\right) - 1\right). \tag{3.33}$$

from the input, the output bias must be provided separately, denoted here as a constant current I_{DC}, independent of the input signal. The class A filter circuit requires only that the input be strictly positive. The output can potentially be bidirectional. However, there are several reasons for biasing the output to make it strictly positive: The filter section is then in a structural form simple to cascade, and the circuit implementations are less prone to output errors caused by device mismatch.

An elegant way to implement the above equation is to multiply the DC current term by the denominator of the bandpass function, $1/1 + (1/Q)\tau s + \tau^2 s^2$, which by itself is a second-order lowpass filter function. A lowpass operation on the DC term retains the DC term unaffected (except for possible offsets as a result of nonidealities in the actual circuit). On the other hand, the additional term makes the equation much easier to factor.

$$\begin{aligned} I_{out}(s) &= I_{in}(s)\left(\frac{\tau s}{1+(1/Q)\tau s+\tau^2 s^2}\right) \\ &+ I_{DC}(s)\left(\frac{1}{1+(1/Q)\tau s+\tau^2 s^2}\right). \end{aligned} \tag{3.35}$$

Equation (3.35) can be easily factored into two simple first-order current transfer functions by introducing an intermediate current term I_x:

$$\frac{I_{out}(s)}{I_{in}(s)-I_x(s)} = \frac{1}{1/Q+\tau s}; \tag{3.36}$$

$$\frac{I_x(s)}{I_{out}(s)-I_{DC}(s)} = \frac{1}{\tau s}. \tag{3.37}$$

The right-hand side of (3.36) and (3.37) takes the form of the first-order function (3.28): for the first section, a lossy integrator, set $I_a = I_b(1 + 1/Q)$, and for the second section, a lossless integrator, set $I_a = I_b$. Replacing τ by the expression of Equation (3.31) shows that the bandpass filter's center frequency is

$$f_c = 2\pi\omega = \frac{1}{\tau} = \frac{I_b}{2\pi V_t C}. \tag{3.38}$$

The input to each section has a negative feedback term from the other section. As presented in the equations above, and shown in Figure 3.8A, the positive and negative feedback terms combine to generate an input which can be a positive or negative value. But this presents a problem: it violates the class A log-domain circuit structure in which input currents must be strictly positive, driving the transistors out of the active mode and invalidating the translinear model. In this case of carefully orchestrated feedback which is the bandpass filter, the inputs to the log-domain sections cannot simply be biased upward: The lossless integrator, in particular, would not work at all, since it would continuously integrate the positive input until its output saturated. There is a solution,

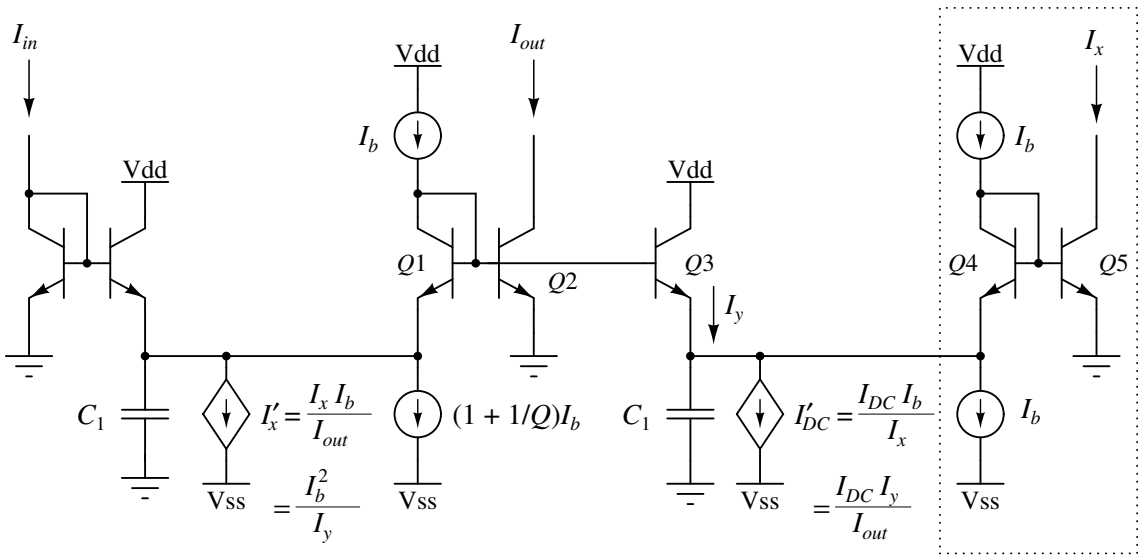


Figure 3.9: Bandpass structure formed from first-order sections.

common-base configuration, and a common-emitter configuration. Thus far we have used common-base configurations exclusively to design first-order log-domain sections. Naturally, connecting transistors together at the emitter works just as well to create KVL loops, with the preferred structure depending on the circumstance. Both structures are illustrated in the bandpass circuit schematic of Figure 3.11: In the first solution, $I'_x = I_b^2/I_y$ is generated by the internal translinear loop $Q3$-$Q7$-$Q6$-$Q1$ using the common-emitter configuration of transistors $Q6$ and $Q7$. This configuration has been used by Frey [39] and others. The other dependent current source, $I'_{DC} = I_{DC}I_y/I_{out}$, is implemented through the translinear loop $Q8$-$Q9$-$Q3$-$Q2$ using a common-base instead of a common-emitter configuration, in which the sign of the current is reversed, as required in Figure 3.9, by mirroring, doubling, and subtracting from the same node. This configuration has been described previously by Fox [44]. Notice that both translinear loops implement the same function with different circuits.

The author of each of the works mentioned above used the same circuit configuration (common-base or common-emitter) twice. It is possible to mix both types of structures to take advantage of the strengths of each in the appropriate context. This is the approach we used for our bandpass filter after encountering some stability problems with our first prototype, which used two common-base structures. On the left-hand side of the filter, a common-base configuration is generally less desirable due to the required extra pair of matched I_b current sources. It is critically

place to another in the circuit, resulting in improved matching of Q but not in gains.

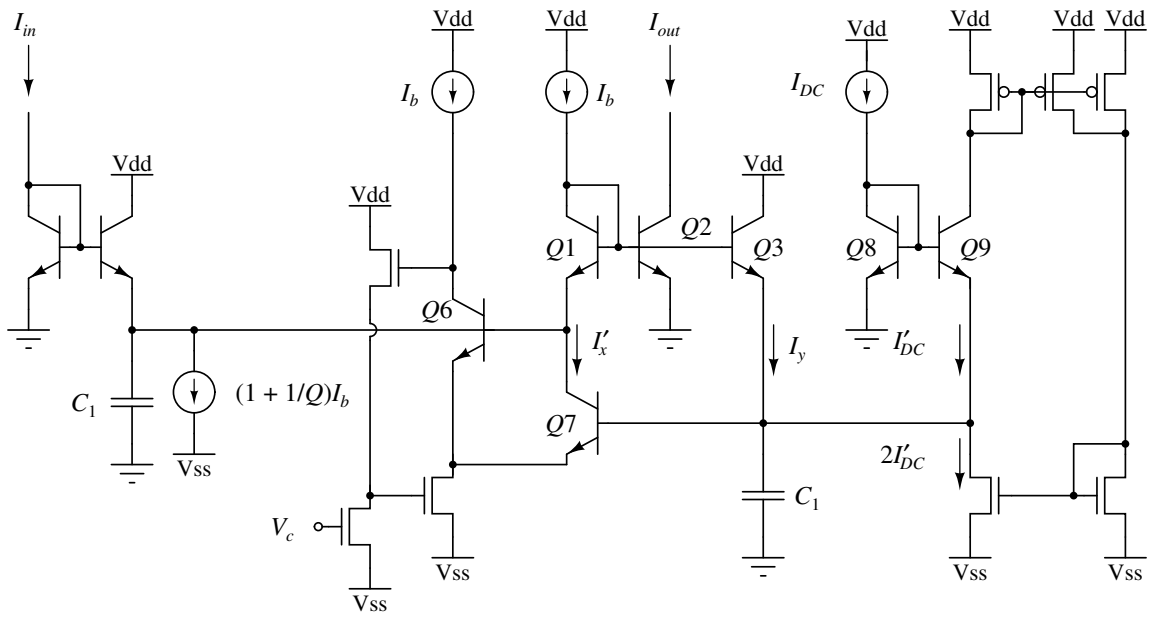


Figure 3.11: Complete circuit schematic for the second-order bandpass filter.

3.9 Technology limitations for low-frequency filter design

Much previously published work on log-domain filters has extolled their potential for replacing conventional filters in high-frequency filter design. Our work instead concentrates on the use of log-domain filters for audio-frequency applications [44, 45, 46], the use of current-mode filtering for current-mode applications, and the application of log-domain filters in system design [57]. The low frequency range of audio requires a large RC time constant, which for the log-domain filters described here is inversely proportional to the bias current I_b. For large-scale integrated systems where capacitors cannot reasonably be made larger than a few picofarads, the bias current can be as low as several tens of picoamps on the low end of the audio frequency range, which places some important restrictions on circuit technology. Problems arising from established designs are:

1. Traditional bipolar designs fail due to base current draw and can suffer from β mismatch at low emitter current values.

2. MOSFET designs have poor matching of currents in mirrors, and require that the input bias

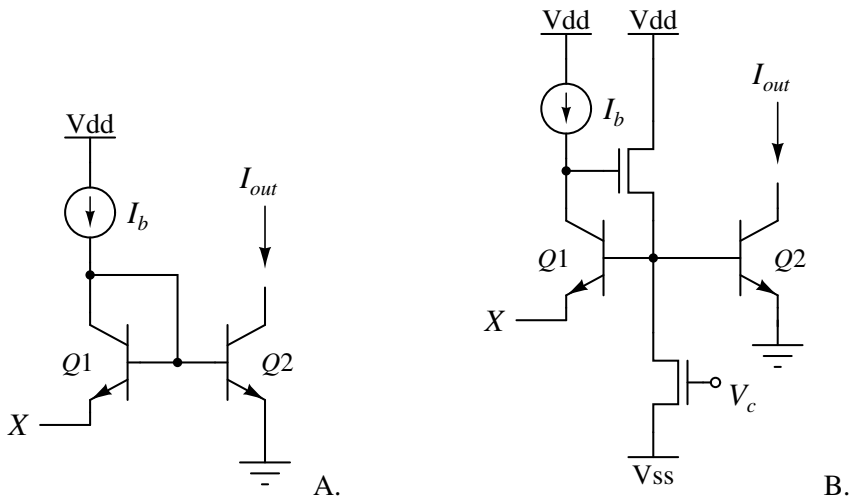


Figure 3.13: Base compensation (B) eliminates undesirable behavior due to significant base current which occurs in (A).

current be very low to keep the MOSFET transistors in weak inversion. The input and output are then close to the noise floor and it becomes difficult to retrieve a clean output signal.

The first of these problems deserves some explanation: The failure of the circuit stems from the design assumption that base current is negligible in translinear-loop circuits. This assumption becomes invalid in situations where the transistor $Q1$ in Figure 3.13A attempts to drive $Q2$ when the collector current through $Q2$ is many orders of magnitude higher than that through $Q1$, due to the emitter voltage of $Q2$ being significantly lower than that of $Q1$. The result is that $Q2$ requires more current through its base than the current source above $Q1$ can provide. This situation occurs in a number of critical places in the bandpass filter circuit; for instance, at the output, where the (DC current biased) output is many orders of magnitude higher than bias current I_b which is driving its base.

We have fabricated and tested both MOSFET and BiCMOS designs. Results from the MOSFET versions indicate that the large noise floor is pervasive and difficult to impossible to eliminate by design. On the other hand, there exist a number of established circuit techniques collectively known as *base-current compensation* for dealing with unwanted base current draw, making the first problem more likely to be overcome. We devised a technique by which base current loss can be eliminated using base-current compensation on all diode-connected transistors in the translinear loops as shown in Figure 3.13. Voltage V_c is a constant bias which generates a bias current that reduces the impedence of the base node. In the second-order filter, the V_c bias on the

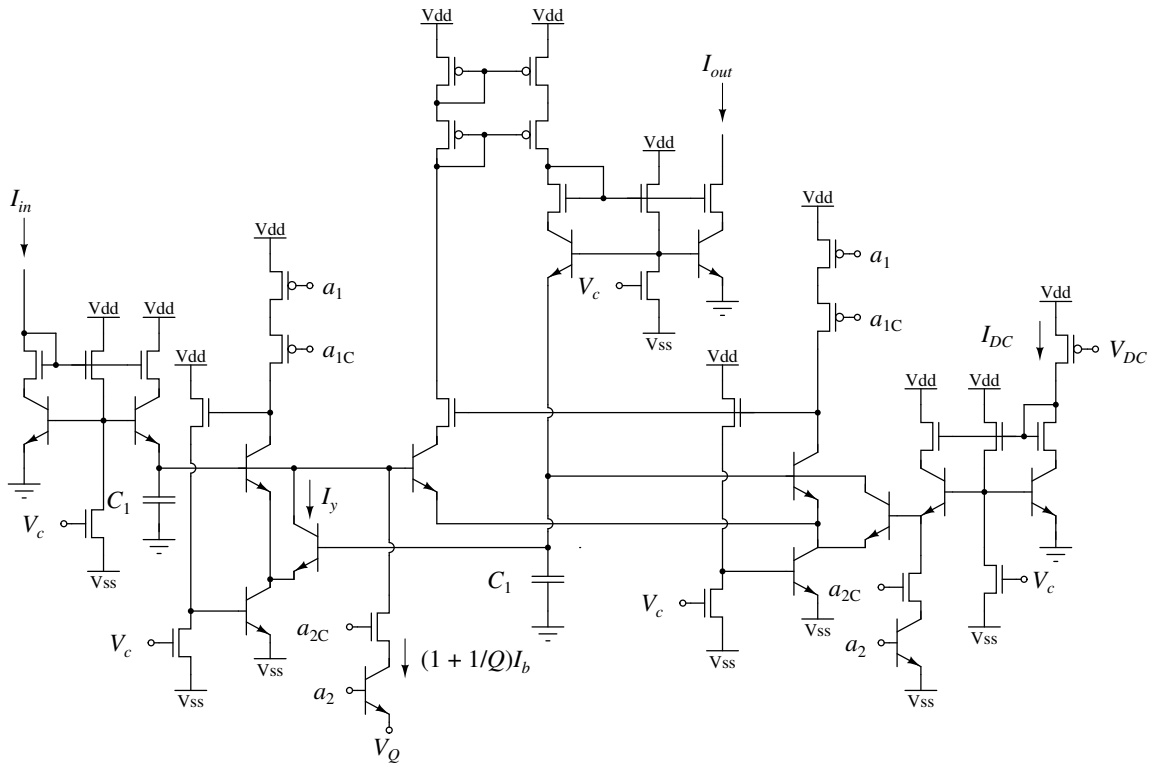


Figure 3.15: Circuit of Figure 3.12, showing base compensation circuits, cascode connections, and the Q-generating circuit

.

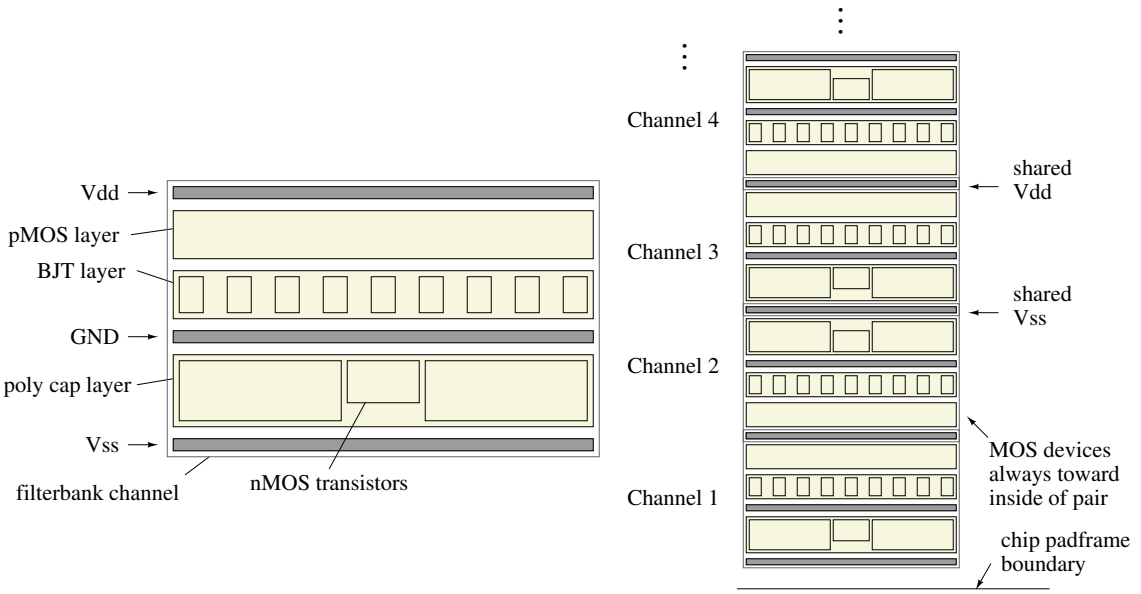


Figure 3.16: Structure of a bandpass filterbank channel and stacking of channels to form the whole filterbank.

3.11 Current-Mode Circuits for Non-Filtering Applications

So far in this chapter, we have developed log-domain theory for the purpose of analyzing and synthesizing filters. Much of the novel aspect of the research work involved in developing the current-mode filterbank, however, was concerned with the other critical circuits necessary to extend the simple parallel filterbank into a useful signal processing frontend.

In keeping with the desire to reduce transistor mismatch through the use of regular and symmetric layout, every attempt was made to maintain that structure not only for the filter, but for all the peripheral circuits as well.

3.12 Signal Rectification and Smoothing

The application for which the frontend filterbank was designed called for rectification and smoothing of each bandpass filter channel output for the purpose of evaluating the short-term average energy over the frequency band of the channel. The signal was smoothed (integrated) with a time constant of 1 to 2 ms.

We considered two ways to compute the short-term energy envelope: The first way was to implement the method of the simulation exactly; that is, to perform a half-wave or full-wave

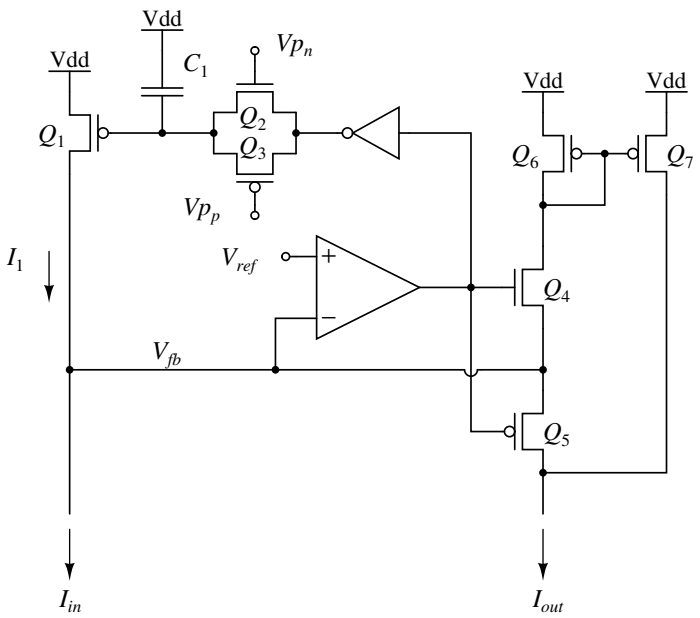


Figure 3.17: Simplified schematic of the adaptive current full-wave rectifier.

the required voltage across C_1, the replacement of the current mirror formed by Q_6 and Q_7 by a current-conveyor mirror of the type shown in Figure 3.13 (B), and the addition of a small coupling capacitor between the op-amp negative input and the output.

After signal rectification, smoothing is a matter of using a log-domain lowpass filter, a structure which has been developed and is known to work well (Figure 3.24). The first-order lowpass is, in fact, the most robust of all the log-domain filter circuits. It can be easily cascaded to form an n-th order filter, as shown in Figure 3.18. Because the translinear circuits operate down to practically zero input current, the log-domain lowpass filter can operate directly on the output of the signal rectifier without requiring any further biasing of the input. The smoothed output can reach arbitrarily low current values during quiet periods of the system input.

3.14 Signal Peak-Peak Detector

We invented and developed a novel analog circuit for determining the height of a signal without requiring any knowledge of the mean value of the input waveform. The operation of the circuit is based on the usual "leaky" integrator peak detector, which consists of a diode used to rectify the input, a capacitor to hold the peak voltage value when the diode is reverse-biased, and a

peak-to-peak amplitude (see Figure 3.20). Due to the leaky capacitor architecture, the circuit responds quickly to sharp rising edges in the input but relaxes slowly to zero on the falling edge of the input. This should be a good model for establishing the energy envelope of acoustic transients, which contain much of the detailed information required for classification in the shape and frequency distribution of the rising edge.

Another novel aspect of our circuit is that it operates in the current domain, in keeping with the mode of the log-domain filters preceding it and the normalizer and correlation processor following it. The use of currents allows us to implement the diode rectifier very simply with a current conveyor structure. These structures are shown in Figure 3.21.

This circuit qualifies as a "log-domain" circuit. The capacitor nodes (V_{max}, V_{min}) hold voltage values which are the logarithm of the maximum (or minimum) input current (I_{in}) and are expanded exponentially when converted from the voltage back into a current (I_{max}, I_{min}). A major consequence of this choice of architecture is that although the leaky current source (M_3, M_6) on each of the capacitors (C_1, C_2) causes a linear change of voltage per unit time, as does the simple voltage-mode peak detector of Figure 3.19, the effect is that the output current changes *exponentially* with time (or as a square-law, depending on the bias on the MOS transistors (M_9, M_10) which do the expansion). This effect is difficult to avoid in a current-mode circuit. We have minimized the effect by raising the DC bias of I_{in}, so that the signal amplitude is considerably smaller than its bias. The differential current change of the maximum and minimum currents is small, and so traverses a shorter range of the nonlinear expansion function, rendering the output nearly linear as a function of time.

Here is a brief description of how the circuit operates: On the maximum current-finding side, when current I_{in} tries to exceed I_{maxl}, voltage Vfb_p drops to near ground to reduce the collector current of Q_2 to match I_{maxl}. The lowering of Vfb_p turns on transistor M_2, causing the circuit formed by transistors M_1 and M_2 to become a current conveyor. V_{max} drops immediately until I_{maxl} reaches the same value as I_{in}. This current-conveyor feedback ensures that as long as the input current is equal to I_{maxl} and rising, I_{max} will continue to track it. On the other hand, when I_{in} tries to drop below I_{maxl}, Vfb_p rises toward Vdd so that M_1 is driven toward cutoff and I_{maxl} drops until it becomes equal to I_{in}. When Vfb_p is close to Vdd, transistor M_2 is cut off, and leaves V_{max} floating save for the small leakage through M_3. So although I_{maxl} is forced to track the input current, the output I_{max} remains at its maximum level, only drifting slowly toward zero until either it falls below the level of the input current, or the input current catches up with its value.

The minimum-tracking circuit is symmetric to the maximum-tracking circuit. To keep the

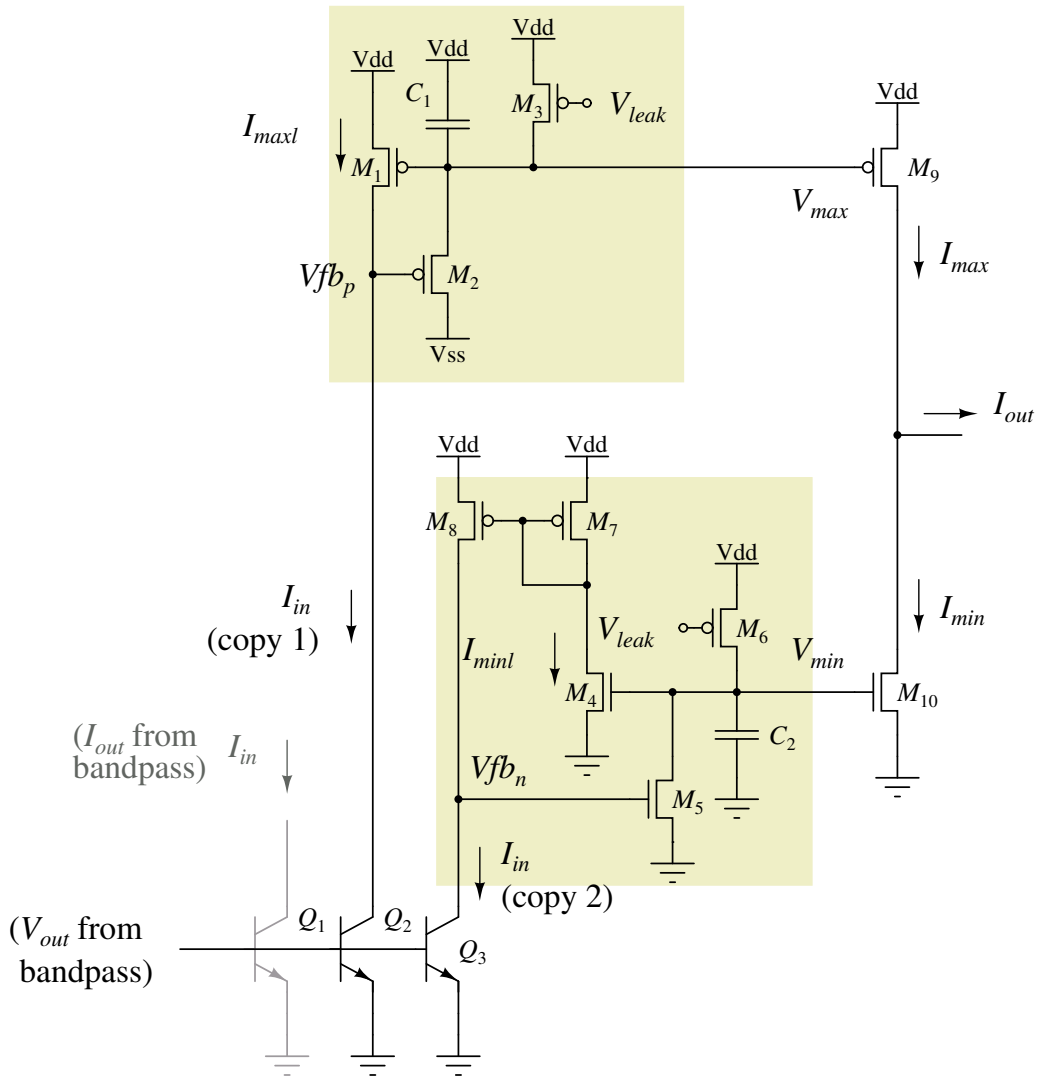


Figure 3.21: Peak-peak detector circuit, simplified.

The point of the L-1 norm is to force the outputs to maintain the same total value as a form of automatic gain control.

$$\sum_{j=1}^{N} I_{out}(j) = I_{norm} \tag{3.42}$$

Such a circuit has been known for many years: it is one of the "original" translinear circuits. It has a simple and elegant form and as such makes a voltage-mode implementation almost unthinkable. It was, in fact, the initial impetus for developing the current-mode, log-domain bandpass filters. The circuit was invented by Barry Gilbert [50] and is known, at least in analog VLSI circles, as the "Gilbert normalizer" [1]. Figure 3.23 shows this circuit. The bias current I_{norm} is a simple way to assure the relationship of Equation (3.42) (excepting the slight difference between the desired I_{norm} and the true I_{norm} due to the BJT β values). Otherwise, the circuit is simply made up of many translinear loops, for which

$$I_{in}(i)I_{out}(j) = I_{in}(j)I_{out}(i) \qquad \forall\, i, j \tag{3.43}$$

Solve by adding together all of the loop equations for input/output pair i:

$$I_{out}(1) = I_{in}(1)\frac{I_{out}(i)}{I_{in}(i)} \tag{3.44}$$

$$I_{out}(2) = I_{in}(2)\frac{I_{out}(i)}{I_{in}(i)} \tag{3.45}$$

$$\vdots$$

$$I_{out}(N) = I_{in}(N)\frac{I_{out}(i)}{I_{in}(i)} \tag{3.46}$$

$$\sum_{j=1}^{N} I_{out}(j) = \sum_{j=1}^{N} I_{in}(j)\frac{I_{in}(i)}{I_{out}(i)} \tag{3.47}$$

$$I_{norm} = \sum_{j=1}^{N} I_{in}(j)\frac{I_{in}(i)}{I_{out}(i)} \tag{3.48}$$

$$I_{out}(i) = I_{norm}\frac{I_{in}(i)}{\sum_{j=1}^{N} I_{in}(j)} \tag{3.49}$$

3.16 Experimental Results

We have fabricated and tested several different chips, one of which was a test chip with individual filter structures on it, and one a complete signal-processing system [57] containing a

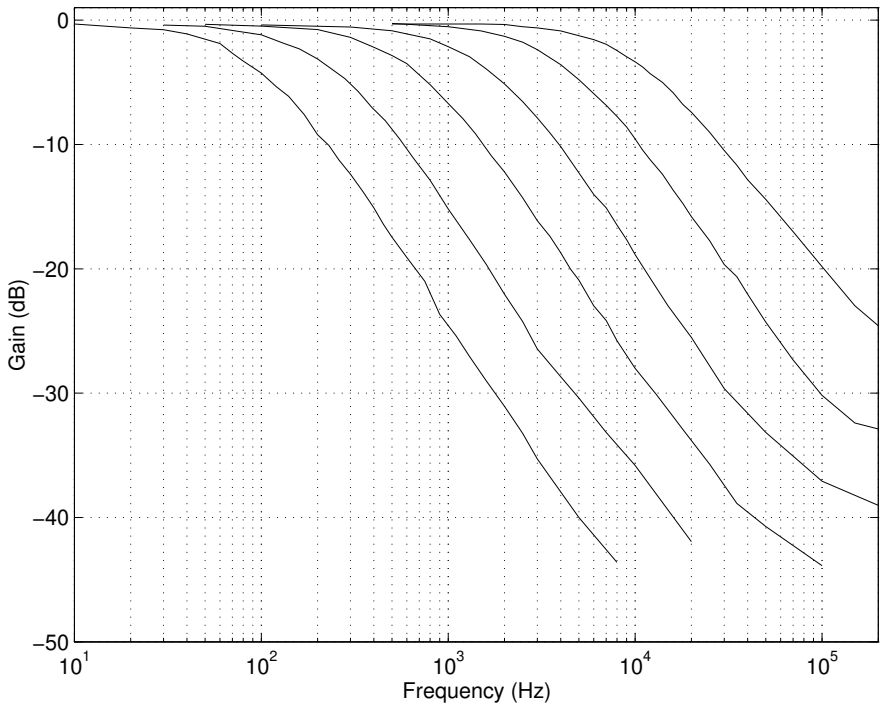


Figure 3.24: Measured magnitude response of the log-domain first-order lowpass filter from the test chip.

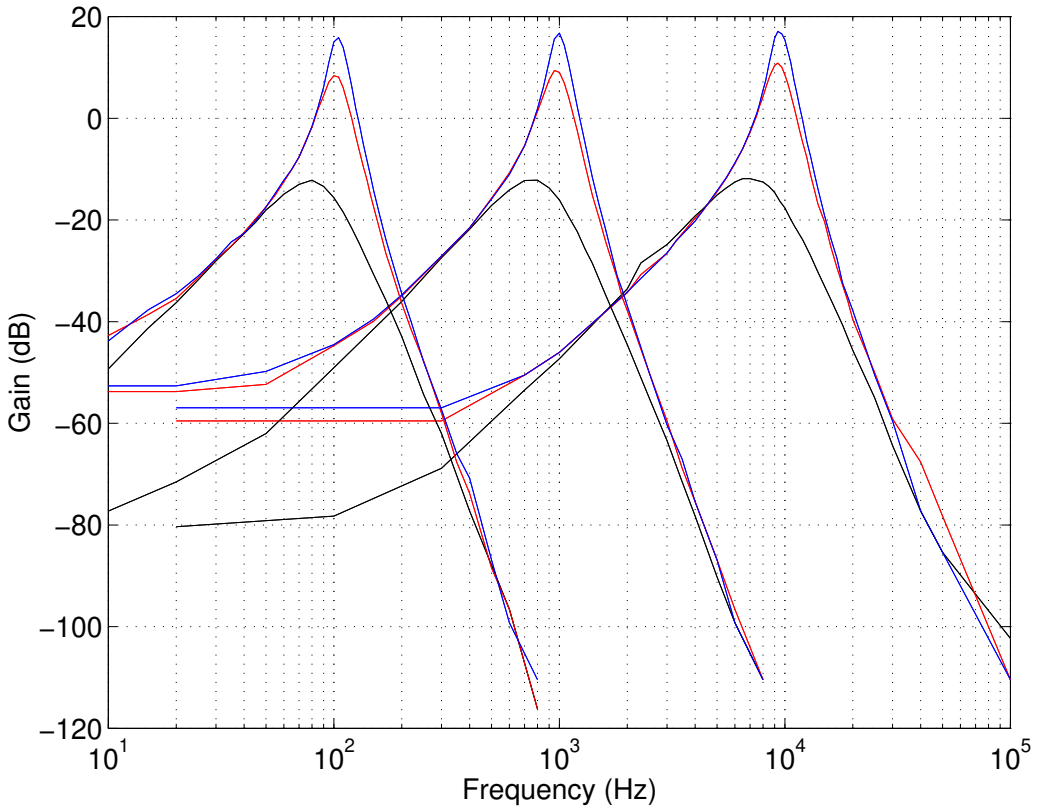


Figure 3.25: Measured magnitude response of one bandpass channel in the filterbank system, made of two cascaded second-order log-domain bandpass filters, over three tunings of the center frequency.

The first filter is responsible for the sharp bandpass response near the center frequency. Both filters are biased to have the same center frequency. Results show that considerable reduction in rolloff occurs on the lower side of the response, due to mismatch of components in the circuit, in particular the matching of bias currents I_b. These data confirm the simulation results of [44], although it is important to note that the effective gain (Q) of the filter appears to be completely independent of the center frequency (bias current I_b). The full tuning range of the circuit is from about 50 Hz to 15 kHz, limited on the low end by $1/f$ and thermal noise and on the top end by voltage drops across the CMOS current sources, which were sized appropriately for audio-frequency operation. Bipolar-based log-domain filters of similar design are capable of high-frequency operation in the tens of MHz.

The distribution of Q values, shown in Figure 3.26 and listed in Table 3.16, shows considerable variation across all channels in the filterbank with the same V_Q (see Figure 3.14) applied

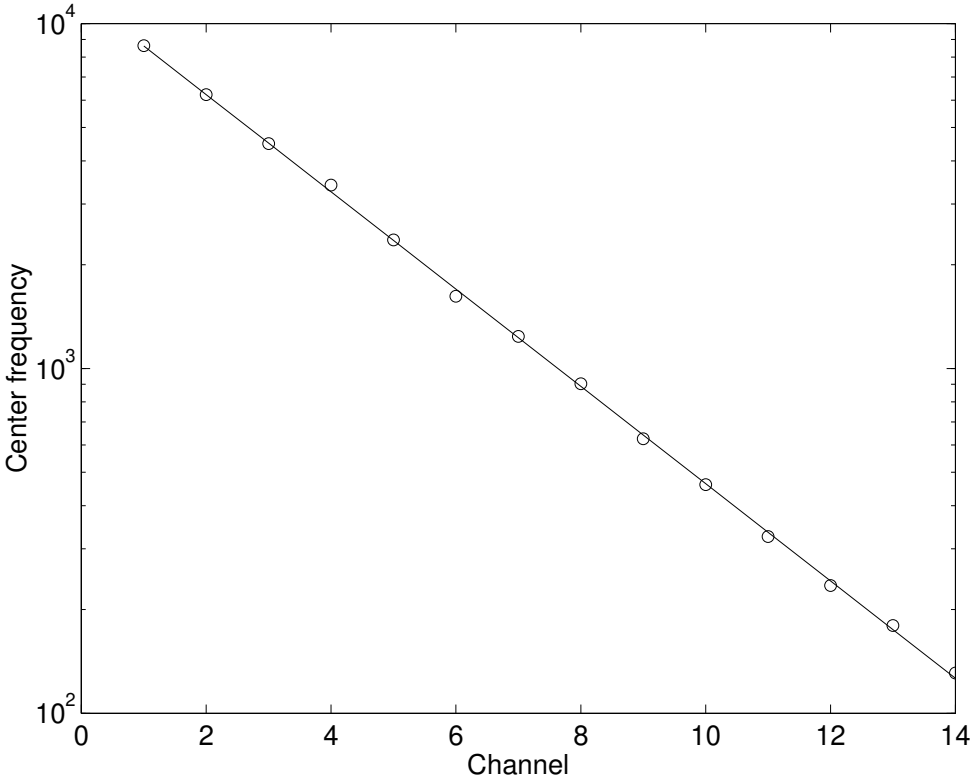


Figure 3.27: Measured center frequencies of all bandpass channels in the filterbank system (circles), compared to the ideal exponential spacing (solid line).

spacing, and mismatch between the pMOS transistors which are gated by the tapped voltages and directly generate I_b. Symmetry in the layout guarantees good matching between transistors at the same position in each channel. This is, in fact, true even between odd and even channels which are antisymmetric (one is flipped across the horizontal axis with respect to the other). The result is nearly perfect exponential spacing seen in Figure 3.27. The poorer matching in Q values derives from two sources: One is that there are two currents which are compared against each other. While good matching between I_b in one channel against I_b in another channel is guaranteed by large-scale symmetry, good local matching of two currents is not so easy to obtain, as the transistors which generate the two matched currents do not have the same physical environs. Particularly in this circuit, the current which generates I_b is sourced from the positive power supply into the capacitor node while the current which generates Q by its ratio to I_b is drawn from the capacitor node to the negative power supply, implying that one current is generated by a pMOS transistor and the other by an nMOS transistor. This situation gives notoriously poor matching properties.

3.17 Summary

We have described how first-order systems can be synthesized from static and dynamic translinear principles, and how to generate higher-order filter transfer functions from these first-order building blocks. In particular, we have addressed circuit design issues relating to audio-frequency applications. We used these synthesis methods to design and fabricate VLSI analog signal processing systems of log-domain filters with current-domain input and output. Results from a first-order lowpass filter and a second-order bandpass filter fabricated in standard BiCMOS technologies show that these filters have transfer functions which map consistently over the entire audio frequency band. Design considerations for low-frequency operation ensure correct translinear operation of the bipolar transistors even at collector currents lower than 10 pA. We have characterized performance of the bandpass filters in the context of a high layout density 15-channel filterbank system. Performance of these filters is adequate for the applications for which they were designed [57].

of increased absolute accuracy with fixed resources, which ultimately renders the systems more fragile. Undoubtedly, what is required for better speech recognition is greater hardware resources in size and complexity. This is almost certainly the future of computer speech recognition as the discipline matures. It is analogous to the evolving state of computer display systems: Years ago, computer CPUs were responsible for the task of dealing with just about all tasks concerned with the visual output. All tasks more involved than piping bits to the monitor were performed in software. Eventually, it became clear that unless the bulk of graphic-related processing was taken away from software and given over to high-speed, highly parallel dedicated hardware, visual displays were not going to get much beyond the stage of text and occasional simple line drawing. Today, video card technology is practically running away from the ability of programmers to make use of it. Video cards have their own large stores of memory (typically 8 MB to run full-motion video in 24-bit color on a large display) and handle tasks from MPEG compression of video to color Gouraud shading of millions of 3-dimensional triangles per second. Speech recognition has not been an absolute necessity for computers, and so the development of speech recognition hardware has not been driven like the development of video display hardware. At some point in the future, though, it is bound to happen. The massive parallelism required for robust speech recognition will require dedicated hardware, something that can wander off on its own processing time and resources and return probabilities of the occurrence of various words and phrases, topical information, speaker identification, language recognition, and the like. The microprocessor will once again be free to work on "more important tasks."[1]

4.1.2 Acoustic Transients

Let us step back a moment from the problem of speech recognition. Current software programs for speech recognition pursue the problem in a very software-specific way. That is to say, knowing that the underlying microprocessor is a serial device with limited parallelism, what steps are most important to make optimal use of this limited resource? The result is systems which tackle the problem primarily in a top-down manner. On the other hand, suppose that we want to look at the problem in a bottom-up manner, making no assumptions about the underlying hardware, but developing the hardware to be that which best solves the problem.

Now let us take another look at biology. Long before there was speech recognition, there

[1] Eventually, the dedicated peripheral hardware will eclipse the microprocessor itself, which will primarily be concerned with managing the interactions between them. Such a distributed system ends up, for obvious reasons of efficiency of design, looking structurally rather like the human brain.

single dimension, from which a classification is determined by placement of a threshold value. Because the classification represents a separating hyperplane, correct classification is possible only for linearly separable data. In addition, template correlation achieves good results as a classifier only if the signal matches the template at the resolution of the template repeatably over many independent instances. This practically never works at the time resolution of the raw acoustic input signal itself, due to the complex interaction of numerous frequency components with phases which may differ from one instance to the next. A template correlation is best suited to operate on the time scale at which the energy envelope of individual frequency components of the input changes. This implies that first the input acoustic signal must be transformed into a time-frequency description to capture the envelope of separate frequency bands, integrated over the required time span. This in turn implies that the template will be two-dimensional, a prototypical mapping of the input class in both time and frequency.

As a way to perform classification of continuous speech, such a simple algorithm fails miserably. Speech and other complex long-term signals may be stretched or compacted in time significantly from one instance to the next (even after computing short-term energy envelopes of the frequency bands of the input decomposition). Unless the template can be stretched or shrunk (known as *dynamic time warping*) to match the input (or vice versa), there may be very little correlation between a signal and its template [66]. Acoustic transients, which are short-term events of less than approximately a tenth of a second, do not suffer the problem of time warping on the time scale of the interesting information content of the signal, which is about one or two milliseconds. A simple correlation in the time-frequency domain followed by a search for the maximum correlation across all templates over the time window of the input yields accurate classification results [54].

A template correlation in the time-frequency domain has the simple general form:

$$c_z[t] = \sum_{m=1}^{M} \sum_{n=1}^{N} x[t-n, m]\, p_z[n, m] \tag{4.1}$$

where M is the number of frequency channels of the input (having already been processed by a filterbank followed by rectification and smoothing, or equivalent frontend processing), N is the maximum number of time bins in the window, x is the array of input signals from the frontend processor, p_z is the matrix of template pattern values for pattern z, t is the current time, normalized to discrete units of the sampling time. This formula produces a running correlation $c_z[t]$ of the input array with the template z. A signal may be interpreted as belonging to class z when the output $c_z[t]$ exceeds a threshold, or by evaluating some function of the vector of outputs over all classes. The

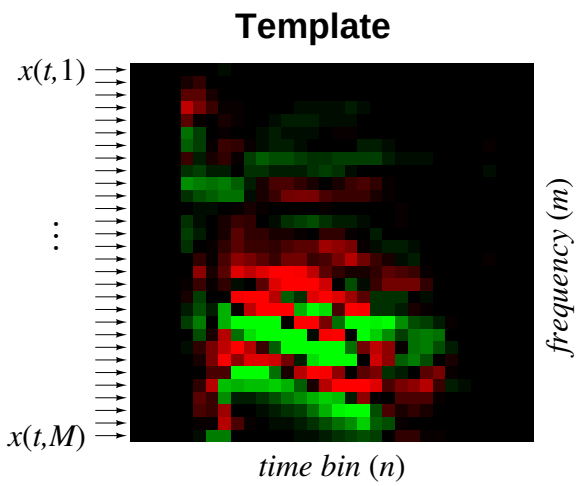


Figure 4.2: Template correlation.

and robust, but are neither simple, cheap, nor power efficient. Second, the template correlation as presented is not necessarily the best solution to the problem of transient classification, and although it has the soundness of common sense about it, there are numerous variations to try which may work as well or better, but which first must be investigated in full.

Our algorithm addresses both issues. It is an approach that lends itself elegantly to low-power parallel mixed-mode computation in the form of MOS transistor circuits operating primarily in the subthreshold mode. It is also an approach which results from an intensive look at variations on the theme of template correlation and results from considerations balancing the competing characteristics of speed, power, cost, and robustness. One consequence of looking at a number of variations is the development of two interesting variations of the algorithm, the alternate one which lends itself to efficient digital computation on dedicated hardware. In Section 4.2.1, we present the mixed-mode algorithm. In Section 4.5 and following, we present the digital algorithm, and we include a summary of the variations on the correlation algorithm which eventually led to the choice of each in Section 4.3.4. In the following discussion, we present our algorithm as a set of incremental modifications to Equation (4.1), which we will refer to as the "baseline algorithm." Software simulations (see Section 4.3) show the baseline algorithm to be both accurate and robust. Here we are primarily concerned with modifications which render the template correlation easier to compute, faster, cheaper, or more power-efficient, either in general or with respect to a specific type of hardware implementation.

We summarize the steps taken as follows:

and degrades gracefully rather than catastrophically with increasing noise (see Section 4.3.7). The effect normalization has on an example acoustic transient is shown in Figure 4.3, which is the normalization step applied to the rectified and smoothed filterbank outputs from the example of Figure 3.4.

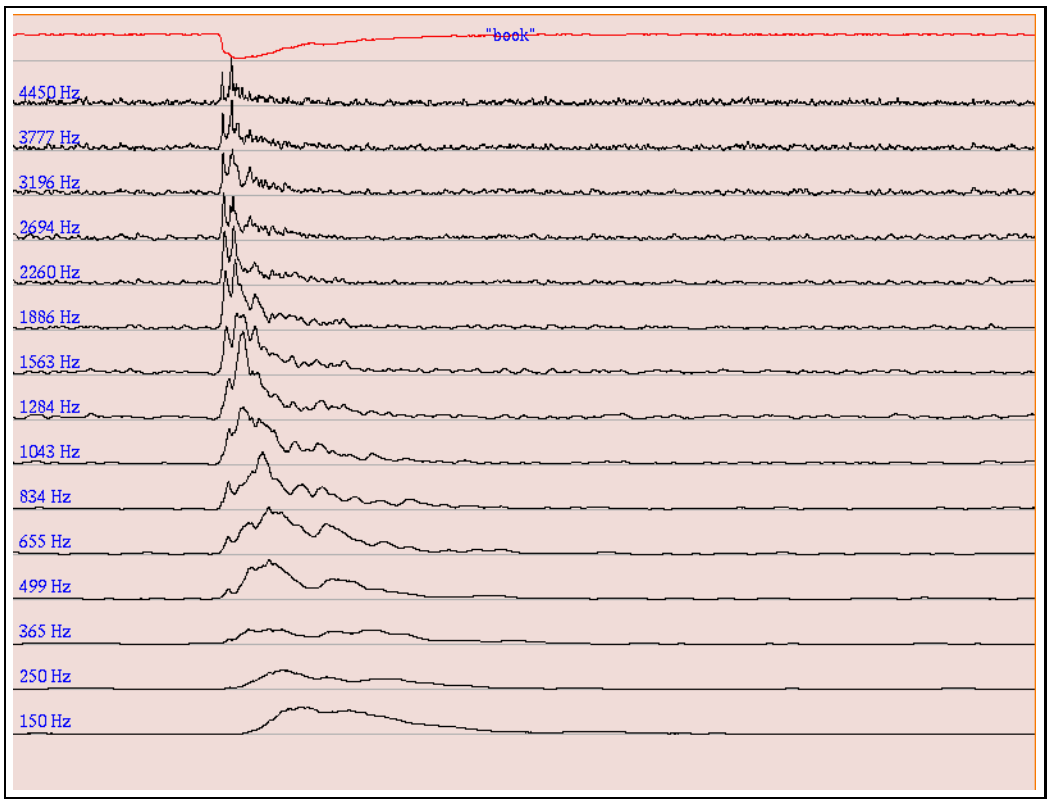


Figure 4.3: Filterbank output after applying an L-1 normalization across all channels. A single channel has been added to the system (top trace), containing the result of a constant value less the instantaneous sum of the remaining channel values.

While automatic gain control and reduced false alarm rates are a beneficial consequence of normalization, the *motivation* behind the particular choice of L-1 normalization is that it is essential for significant simplification of the pattern classifier algorithm [55] as discussed below.

The hardware complexity of the correlation computation is determined by the multiply-accumulate function performed at each template value. Because digital multiplication is a particularly complicated area and time consuming function, it should be immediately apparent that the circuit complexity can be greatly reduced if it is possible to encode the template values as a single bit each. While this consideration leads to efficient digital implementations of the correlator, it is an especially efficient form for an analog, current-mode implementation. It remains to be proved that

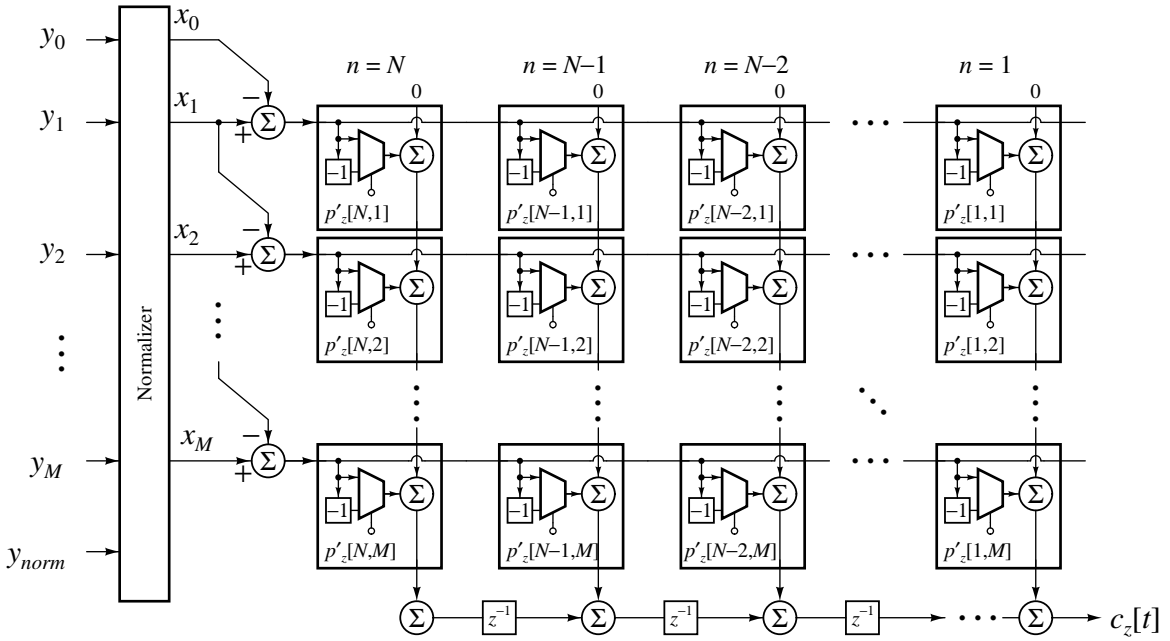


Figure 4.4: Template correlator with pipelined architecture and multiplexors replacing the multipliers.

Figure 4.4 shows an architecture implementing Equation (4.3) directly. In addition to minimizing die area required for storing template weights, binary template values reduce the correlation multiplication to a multiplexing function. However, a multiplication by -1 is still required in Figure 4.4. While the negation operator is not nearly as complicated as a multiplier and can be handled efficiently by various tricks, some circuitry overhead is inevitable.

A further simplification of the architecture is also shown in the same figure. This simplification involves transforming the correlation into a pipelined process. In the pipelined architecture, input $x[t]$ is multiplied by all template values simultaneously and is not used again: it does not require any memory for the input vector. Instead, the system stores *partial column sums* from the inner loop evaluation, which we denote $q[n,t]$. The partial column sums require a single shift-and-accumulate register. The correlation output at time t is the value at the end (position N) of the register, delayed by one time step:

$$q[n,t] \quad = \quad q[n-1,t-1] + \sum_{m=1}^{M} x'[t,m]\, p'_z[n,m] \qquad \forall n \neq 1 \tag{4.6}$$

The fact that the input has been normalized by the L-1 normalization function allows a further simplification of the architecture which again does not measurably affect system performance. This step is to make the template values binary $[0, 1]$ rather than binary $[-1, 1]$. In the case of $(0, 1)$ encoding, the problematic -1 gain factor disappears and the multiplexing function may be reduced to a simple switch. This results in a simplified floorplan for both analog and digital implementations, although the greatest gain is realized for current-mode analog systems, where each switch can be a single MOS transistor and the accumulation can be accomplished by simple addition of currents onto a single wire. Figure 4.6 shows this architecture.

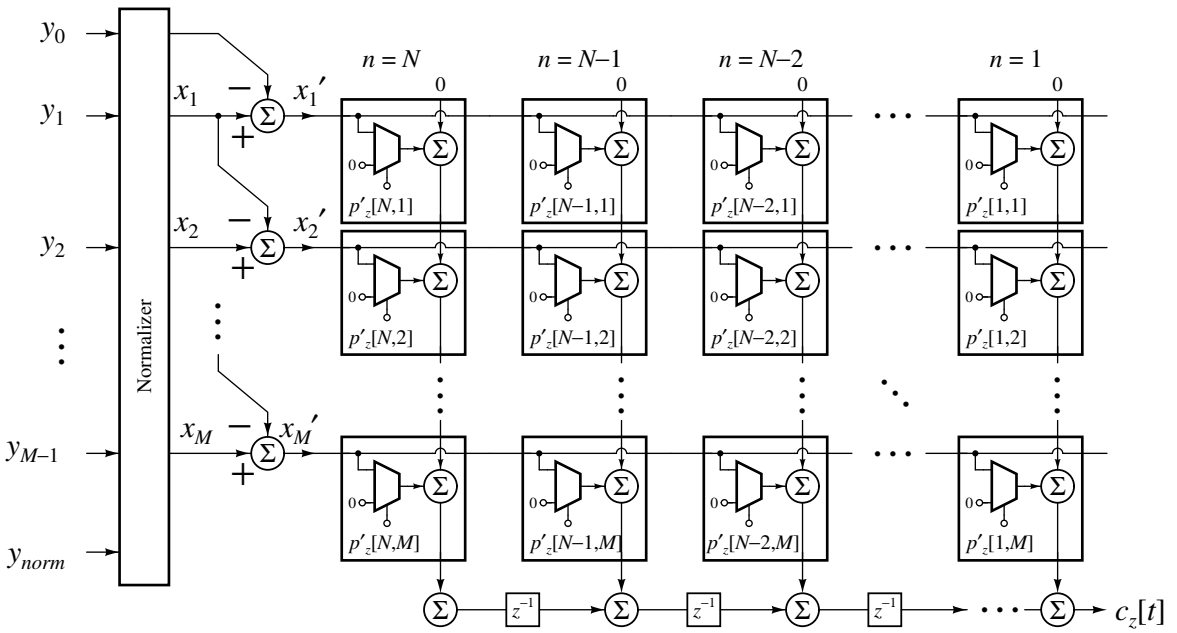


Figure 4.6: Template correlator with [0,1] encoding of template values.

Still, the input x' has both positive and negative values, so even though the correlation has been reduced to a simple array of switches, there remains the problem of bidirectional currents. We must duplicate (mirror) the input current in each template cell, and mirroring of bidirectional currents is much more complicated than mirroring of unidirectional currents, which requires only that a single gate voltage be distributed to the array to be transformed back into the input at each cell by a single matched transistor. The best choice in this case is to avoid the problem by commuting the channel differencing operation to the output. This requires that the differenced input $x'[t, m]$ be expanded and multiplied through by the template separately. There are few good architectural choices; the best one relies on the accuracy of the accumulate-delay register and assumes that two

system is fully parallel, and only about as complex as a RAM array. Figure 4.7 shows a diagram of the system as presented, for a current-mode implementation. The correlator accepts inputs in the form of unidirectional currents, one for each channel m. The input current is mirrored simultaneously in all cells across the template array, shown here by voltage controlled current sources. Each template cell contains a single bit controlling the multiplexer, a dual-pole, single-throw switch which adds either zero current or the copy of the input current to the sum. Rather than compute the channel difference directly, the positive and negative parts are kept separate. Inputs from frequency channels n and $(n+1)$ are kept separate, each added to its own partial column sum line and accumulated over time on two different delay-accumulate lines. The channel difference computation is deferred, occurring at the output.

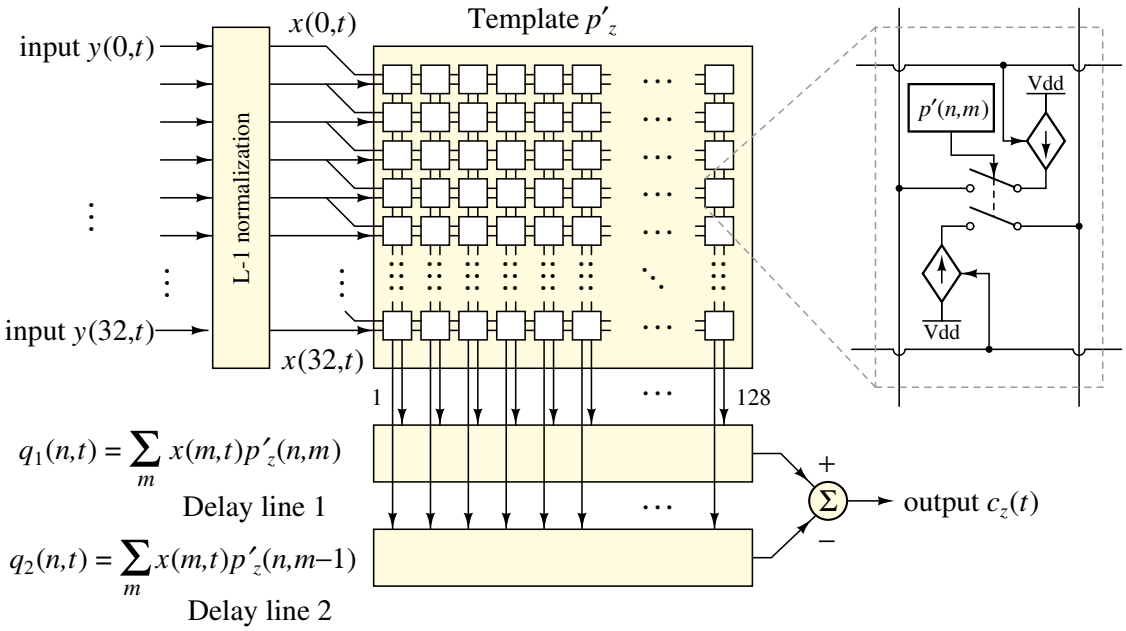


Figure 4.7: Block diagram of the temporal current correlator.

4.3 Simulations

In the previous sections of this chapter, we introduced an algorithm for classifying acoustic transient signals using template correlation, and showed the development of the algorithm for efficient and accurate operation under the constraints of the proposed analog VLSI hardware. In the course of both algorithm and hardware development, we were led to reevaluate the algorithm in

Bar	A heavy metal bar
Book	Sound of a book being closed, loudly
Can	An empty aluminum can hit against a desk
Dink	A lightweight metal bar
Door	Sound of a door closing, with the latch snapping into place
Finger	Finger snap
Hand	Handclap
Tub	A plastic tub
Mallet	A wooden mallet
Shelf	Sound of knocking on a wooden bookshelf

Table 4.1: Classes and descriptions of the recorded transient dataset.

cascade rather than second-order filters with gain. Because the taps then do not have bandpass-like characteristics, each tap is followed by a bandpass filter.

The cascade of filters begins with a high-frequency cutoff and each successive stage has a progressively lower cutoff frequency. Consequently, the stopband attenuations get multiplied and after a few stages in the cascade, the filter transfer function acquires an extremely large slope for the rolloff into the stopband. The use of first-order filters rather than resonant filters increases the stability of the signal as it passes along the cascade, and helps control mismatch between the stages. The bandpass-filtered output of the taps comprise a thirty-one channel filterbank with center frequencies spaced on a logarithmic frequency scale from 100 Hz to 6 kHz. The filter is more or less constant-Q with $Q = 3.5$ at the higher frequencies but drops to unity resonance at the lowest frequencies, in keeping with the physical model of the mammalian cochlear frequency response.

The HEEAR chip extends the original electronic cochlea by modeling the function of the inner hair cells surrounding the mammalian cochlea. The purpose of the hair cells is signal rectification and some nonlinear gain control. These hair cell circuits were bypassed for the acoustic transient recordings. The HEEAR response to the recorded dataset was sampled and saved to disk along with the original recording.

4.3.2 Simulating the Acoustic Transient Baseline Algorithm

The acoustic transient processor algorithm begins with digital post-processing of the HEEAR frontend filterbank data on a computer. The samples from the thirty-one output channels of the HEEAR processor are rectified, smoothed with a lowpass filter function using a 1 ms time constant, and then decimated from 32 kHz to 1 kHz. The smoothed, decimated outputs are

smoothing time constant of 1 ms, and the final results combined with an OR logic function. The longer time constant ensures that all noise is eliminated from the segment estimate, while the shorter time constant acts to detect the transient onset with less delay [55].

Template values p_z are learned automatically by aligning all examples of the same class in the training set using the result of the automatic segmenter, and averaging the values together. The template size is set to be the same size as the maximum-length segmented example, and the remaining examples are padded out to match.

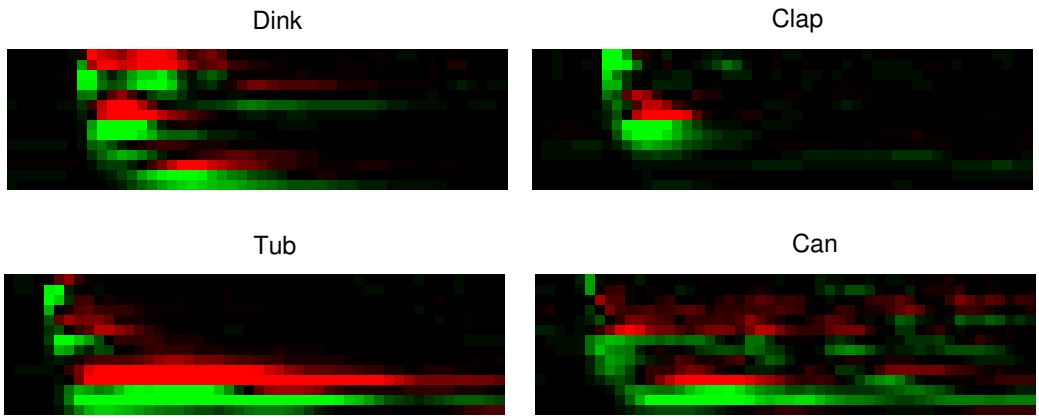


Figure 4.10: Example templates learned by the ATP algorithm.

We evaluated the accuracy of the system with a leave-one-out cross-validation loop in which we train the system on all of the database except one example of one class, then test on that remaining example, repeating the test for each of the 222 examples in the database. The baseline algorithm gives a classification accuracy of 96.4%. Table 4.2 shows the resulting confusion matrix for the cross-validation loop.

4.3.3 Optimizing Template Correlation Algorithms for Acoustic Transient Classification

A major consideration for hardware implementations (both digital and analog) is the memory storage required by the templates, one of which is required for each class. Minimal storage space in terms of bits per template is practical only if the algorithm can be proved to perform acceptably well under decreased levels of quantization of the template values.

At one bit per template location (*i.e.*, $M \times N$ bits per template), the complexity of the hardware is greatly simplified, but it is no longer obvious what method is best to use for learning

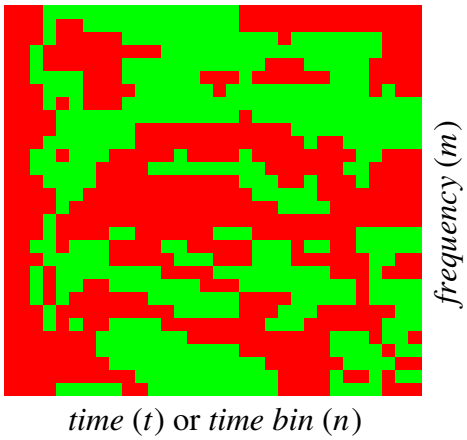


Figure 4.13: The same template reduced to binary values.

the template values, or for calculating the per-class gains. The choice of the method is guided by knowledge about the acoustic transients themselves, and simulation to evaluate its effect on the accuracy of a typical classification task.

4.3.4 Simulations of different zero-mean representations

One bit per template value is a desirable goal, but realizing this goal requires reevaluating the original correlation equation. The input values to be correlated represent band-limited energy spectra, and range from zero to some maximum determined by the L-1 normalization. To determine the value of a template bit, the averaged value over all examples of the class in the training set must be compared to a threshold (which itself must be determined), or else the input itself must be transformed into a form with zero average mean value. In the latter method, the template value is determined by the sign of the transformed input, averaged over all examples of the class in the training set.

The obvious transformations of the input which provide a vector of zero-mean signals to the correlator are the time derivative of each input channel, and the difference between neighboring channels. Certain variations of these are possible, such as a center-surround computation of channel differences, and zero-mean combinations of time and channel differences. While there is evidence that center-surround mechanisms are common to neurobiological signal processing of various sensory modalities in the brain, including processing in the mammalian auditory cortex [70], time derivatives of the input are also plausible in light of the short time base of acoustic transient events. Indeed, there is no reason to assume *a priori* that channel differences are even meaningful

for the center-surround in some instances is most likely due only to the fact that one less channel contributes information to the correlator than in the pairwise channel difference computation. When accuracy is constant, a hardware implementation will always prefer the simpler mechanism.

Very little difference in accuracy is seen between the use of a binary $(1, -1)$ representation and a binary $(1, 0)$ representation, in spite of the fact that all zero-valued template positions do not contribute to the correlation output. This lack of difference is a result of the choice of the L-1 normalization across the input vector, which ensures that the part of the correlation due to positive template values is roughly the same magnitude as that due to negative template values, leading to a redundant representation which can be removed without affecting classification results. In analog hardware, particularly current-mode circuits, the $(1, 0)$ template representation is much simpler to implement.

Time differencing of the input can be efficiently realized in analog hardware by commuting the time-difference calculation to the end of the correlation computation and implementing it with a simple switched capacitor circuit. Taking differences between input channel values, on the other hand, is no so easily reduced to a simple hardware form. To find a reasonable solution, we simulated a number of different combinations of channel differencing and binarization. Table 4.5 shows a few examples. The first row is our standard implementation of channel differences using binary $(1, 0)$ templates and continuous-valued input. The drawback of this method in analog hardware is the matching between negative and positive parts of the correlation sum. We found two ways to avoid this problem without greatly compromising the system performance: The first, shown in the second row of Table 4.5 is to add to the correlation sum only if the channel difference is positive and the template value is 1 (one-quadrant multiplication). Another (shown in the last row) is to add the maximum of each pair of channels if the template value is 1, which is preferable in that it uses the input values directly and does not require computing a difference at all. Unfortunately, it also adds a large component to the output which is related only to the total energy of the input and therefore is common to all class outputs, reducing the dynamic range of the system.

Table 4.5: Simulation results for different methods of computing channel differences

method	accuracy
channel difference	94.14%
one-quadrant multiply	92.34%
maximum channel	93.69%

certain classes will have a tendency to win unconditionally. Since in general, classes which have longer timespans contain more overall energy, dividing the correlation result by the timespan of the template is a sensible way to normalize. However, it remains an ad hoc solution. The proper treatment is presented below. It is fundamental to the optimizations described in Section 4.2.1.

The template has been generated by averaging all examples of the class in the training set. Therefore the template can be considered to represent the *prototypical* class example. The correlation of the template with itself (autocorrelation) therefore represents a prototypical correlation for that class with its own template, under condition of optimal alignment, for which we expect the correlation result to be maximum. If all correlation results for template z are normalized by the autocorrelation of template z, then all correlations should have an average maximum value of one, and correlation outputs from different templates may be compared directly to one another.

The gain factor K_z is computed from the template values using the autocorrelation function:

$$K_z = \sqrt{\sum_{m=1}^{M} \sum_{n=1}^{N_z} {p'}_z[n,m]^2}. \tag{4.17}$$

The per-class gain values K_z using the autocorrelation normalization are optimal for the baseline algorithm. Autocorrelation applied to *binary* templates (when the template value is assumed to be either $+1$ or -1) yields $K_z = M\,N_z$, which is the same value for all classes z when the template is a fixed size ($N_z = N\;\forall z$). This indicates that autocorrelation tells us nothing interesting about per-class gains other than that the optimal case is that of no normalization. Unity gain is assumed in all the simulations of the previous section, and the assumption is upheld by the excellent system accuracy in simulation where no gains K_z were applied at the outputs.

A careful evaluation of errors from several runs indicated the possibility that different gains on each channel potentially could improve recognition rates. That is, if errors are histogrammed by type (class a expected, incorrectly classified as class b, for all combinations of $a \neq b$), a nonuniform distribution results. Simple experiments with K_z gain values tweaked by hand proved that reducing the gain of classes which had a greater tendency to be chosen in error could cause error rates to drop and the error distribution to be more uniform.

The apparent contradiction can be resolved by realizing that the autocorrelation of the binary template with itself only makes sense if the input is also binary. When inputs are continuous-valued, then the correct computation is not the template autocorrelation, but the correlation of the template against the averaged examples of the class, which are the values obtained for the template

C_{ii} with respect to all other elements in the same column, C_{ij}. Since the templates are fixed by the averaging algorithm we used to create them, the only degree of freedom available for minimizing or maximizing anything is the premultiplication coefficients K_j on each template, one per row of C. The per-class gain mechanism is easily transferred to the analog or digital hardware domain.

In the case of both continuous-valued templates and input, an optimal solution can be directly evaluated and yields the autocorrelation normalization of Equation (4.17). However, for all binary forms of the template and/or input, direct evaluation is impossible and the solution must be found by choosing an error function $\mathcal{E}$ to minimize or maximize. The error function must assign a large error to any off-diagonal element in a column that approaches or exceeds the diagonal element in that column, but must not force the cross-correlations to arbitrarily low negative values. A minimizing function that fits this description is

$$\mathcal{E} = \sum_j \sum_{i \neq j} \exp\left(K_i C_{ij} - K_j C_{jj}\right). \tag{4.24}$$

This function unfortunately has no closed-form solution for the coefficients K_j, which must be determined numerically using Newton-Raphson or some other iterative method. A software routine which performs the optimization is printed in Appendix F.

Improvements in the recognition rates of the classification task using this optimization of per-class gains is shown in Table 4.6, where we have considered only the case of inputs and templates encoding channel differences. Although the database is small, the gains of 2 to 4% for the quantized cases are significant, particularly as they render the histogram of errors more uniform.

Table 4.6: System accuracy with and without per-class normalization.

binarization	accuracy, optimized	accuracy, non-optimized
none	100%	100%
template only	93%	91%
template & input	95%	91%

4.3.7 System Robustness

We performed several additional experiments in addition to those covered in the previous sections. One of these was an evaluation of recognition accuracy as a function of the template length N (number of time bins), to determine what is a proper size for the templates. The result is shown in Figure 4.14. This curve reaches a reliable maximum at about 50 time bins, from which our chosen size for the hardware implementation of 64 bins provides a safe margin of error (as

rather than catastrophically with additional noise.

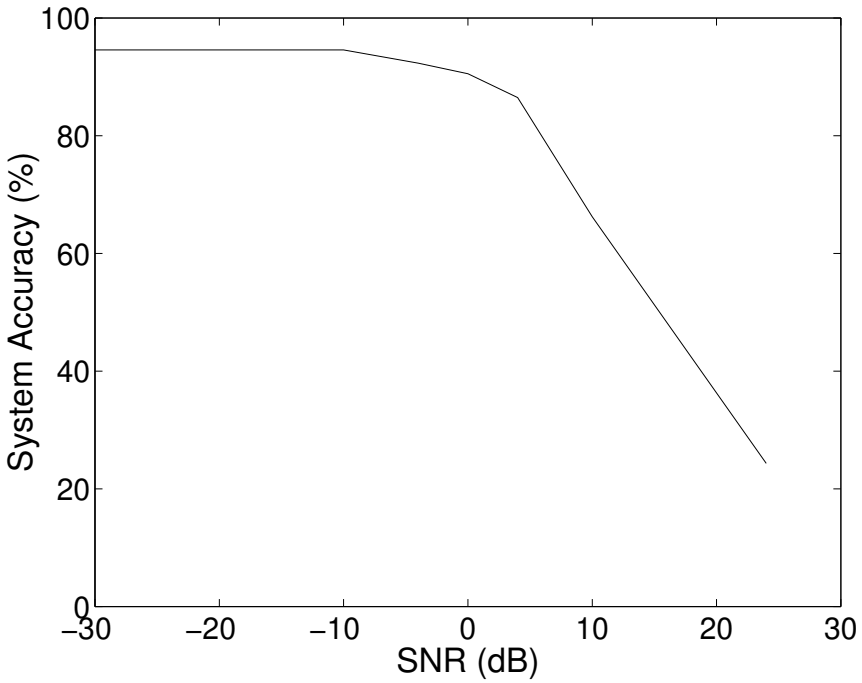


Figure 4.15: Effect of white noise added to the correlator inputs.

An interesting question to ask about the L-1 normalization at the frontend is how the normalization value (θ) affects the classification performance. If this channel is omitted, then the total instantaneous value of all outputs must equal the same value, even during periods of silence, in which low-level noise gets amplified. The nominal value of this channel was chosen to match the levels of noise in the transient recordings. For one of the cases of Table 4.3 (real input, binary $(1, 0)$ template, channel differencing at the input), we tried two other tests, one with the normalization constant doubled, and one with it omitted (zero). Doubling the normalization constant had no effect on the error rate, while omitting it caused the accuracy to drop only from 94.1% to 92.3%. The conclusion is that for large templates, random noise has a low probability of producing a spurious positive correlation that would be classified as a transient. The classification algorithm is not largely dependent on input signal normalization.

4.3.8 Research Directions

The optimizations of the correlation algorithm we have presented are by no means exhaustive; in fact, they represent only the tip of the iceberg. All the optimization strategies considered here

4.4 Hardware Implementation of the Acoustic Transient Processor

We now turn to the problem of creating the VLSI circuits which implement the algorithms presented in the preceding sections. Much of the algorithm development took into consideration the problem of efficient mixed-mode VLSI design.

Our approach lends itself elegantly to low-power, massively parallel analog computation in the form of MOS transistor circuits operating primarily in the subthreshold mode. Our choice of an analog implementation stems from the ability to make such a system compact and extremely power-efficient. Using a separate correlator circuit for each template, the system is fully parallel, and only about as complex as a RAM array.

4.4.1 Current-switching Memory Array

To implement the summation over m in Equations (4.9–4.12), we utilize the simplest form of summation available to analog circuits, that of summing currents onto a single wire. Each cell in the array contains a conventional static memory circuit storing the single-bit template value, a pMOS transistor switch with the template bit controlling its gate, and a pMOS transistor current source which generates a copy of the input current for row m. The switch allows or disallows the copied current to be added to the total current for column n. Each array position is individually addressable for programming (so-called "random access") to allow simple chip-in-the-loop learning under computer control. Unlike a true RAM array, there is no need (other than diagnostic) to be able to read the value of the memory cell. In a way, these cells are "write-only," although the value of each cell can be determined indirectly through the system input and output.

Figure 4.16 shows the layout of the template array and the circuit used at each template cell. The single-bit template value is stored in a standard 6-transistor SRAM cell, made of back-to-back inverters and two nMOS programming transistors connected to a row-programming enable line $Write(m)$ running horizontally across the template array and two lines for the input $bit(n)$ and $\overline{bit}(n)$ values. The programming transistors and corresponding data and control lines appear gray in the figure. The remainder of the core cell represents the bulk of the correlator processing, which has been somewhat complicated by the requirements of the channel differencing operation. The algorithm requires two switches per template memory location $t(n, m)$, one (M_2) which switches the input current of row m to the first (positive) delay line, and one (M_3) which switches the input current of row $m - 1$ to the second (negative) delay line. The positive and negative parts share the same wire carrying the current down to the delay lines by time-multiplexing based on the signal Φ.

M_1	W/L (λ) = 16/8
$M_2 \ldots M_5$	W/L (λ) = 4/2
core cell	W/L (λ) = 28/86
16×64 array	W/L (λ) = 1796/1380

Table 4.7: Transistor, cell, and array sizes corresponding to Figure 4.16.

are positioned like cascode transistors in relation to the current source (M_1), and will act as such over a large range of voltages. The implications are:

1. The dynamic circuit should ensure that the dynamic memory capacitor node ($t(n, m)$) tends to move toward the positive power supply (V_{dd} in response to charge leakage. If this is true, then cells which are programmed *off* will remain *off*.

2. A cell which is programmed *on* will tend to leak slowly toward the *off* state. However, the switch will be effectively *on*, passing all input current, for a large range of voltages (for a 5 V supply, one can expect about four volts range, above which the input current will be reduced). Consequently, the dynamic memory does not need to be refreshed often, reducing overall power consumption.

3. Another consequence of the large range of voltages for which the switch is *on* is that the memory does not need to be programmed down to zero volts. As a result, the core cell does not need any nMOS transistors at all: The $bit(n)$ value to store can be programmed through a single pMOS transistor (M_6). Due to the limitations of unipolar devices used as switches, the range of voltage values which can be stored on the capacitor is limited to the positive power supply down to somewhere in the range of 1 to 2 V above ground. This value suffices to keep the switch open.

4. Because the correlator array operates on a slow 1 to 2 ms clock, there is plenty of time for refreshing between cycles of the correlator, so that the digital switching which occurs during dynamic memory refresh does not have any effect on the analog processing. All N columns in a single channel (row) m are refreshed together. On each correlation cycle, one of the rows is refreshed, so that the entire memory is refreshed every M cycles ($M = 16$ on our prototype chips).

We have used both standard static (back-to-back inverters) memory and the dynamic cell described above in prototype designs. As is generally the case with VLSI memory arrays, the

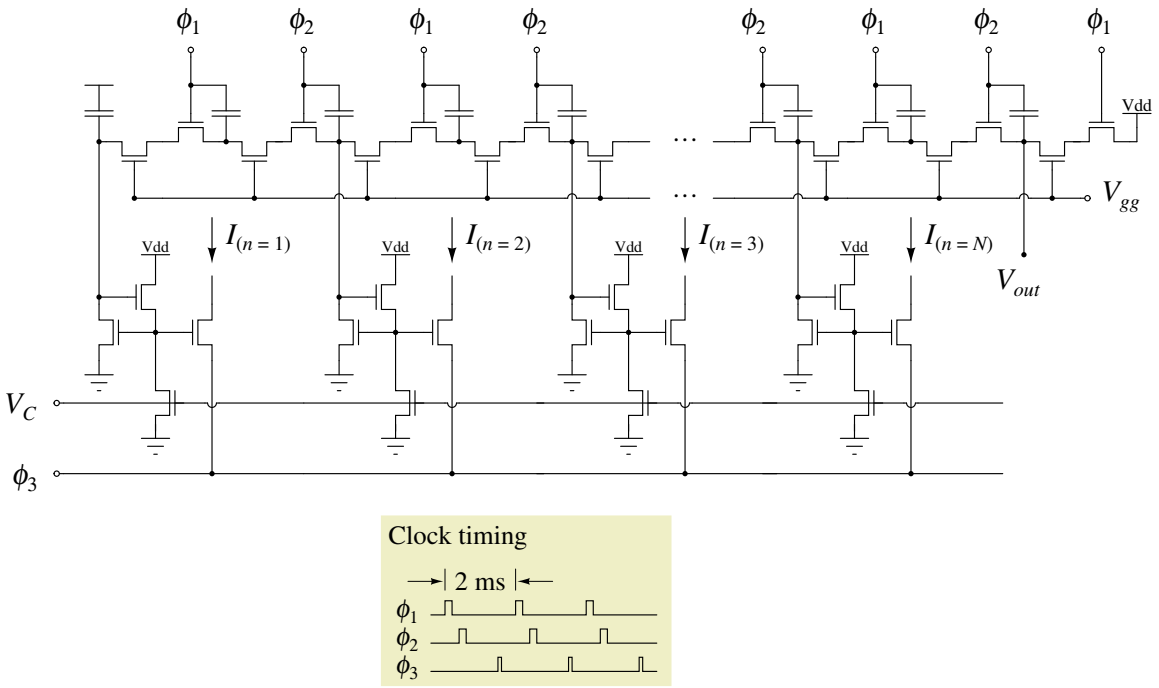


Figure 4.18: Details of the bucket brigade device (BBD).

clock. Each BBD transistor's gate is coupled to its drain through a capacitor (preferably several hundred fF or greater). At the start of a cycle, the analog value to be transferred (a voltage measured negatively from V_{dd}) is stored on the transistor source, and a value of $V_{dd} - V_{th}$ is stored on the transistor drain, where V_{th} is the transistor threshold voltage. The clock raises the gate-to-source voltage V_{gs} of the transistor while ensuring a positive drain-to-source voltage V_{ds}, initiating current flow from drain to source. Provided that the capacitors at drain and source of the transistor are the same size, the voltage drop at the drain will equal the voltage rise at the source. Current flow becomes negligible when $V_{gs} = V_{th}$. The final voltage at the source is therefore $V_{dd} - V_{th}$, and the final voltage at the drain, when the clock voltage returns to ground, is the value originally at the source.

Although the explanation above describes the theoretical operation of the bucket brigade device, in practice such a system suffers from significant losses due to *charge transfer inefficiency*, primarily because on any given cycle, some of the charge remains trapped under the transistor gate and is injected back into the previous stage at the end of the cycle. An additional transistor per BBD stage, gated by a constant voltage V_{gg}, alleviates the problem and can increase charge transfer efficiency from 97% to 99.7% The voltage V_{gg} has an optimal value which can be determined

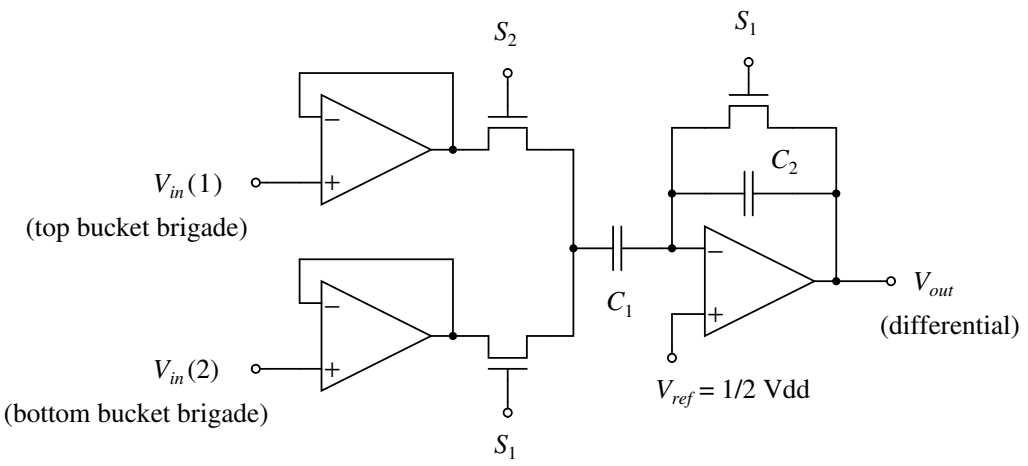


Figure 4.19: A switched capacitor circuit to compute channel differences at the output.

currents throughout the array. The remaining template correlators act as "slaves," receiving the voltages produced by the master template correlator as input rather than currents. Fanout is not a major problem due to the small size of the currents and the slow speeds (1 to 2 ms time constant) at which the input signal changes. Figure 4.20 shows the simple 2-transistor circuit made of pMOS devices which acts as a switchable current mirror. Making the voltage/current enabling switch the same size as the switch devices in the memory cells helps to maintain similar surrounds for the input with respect to the core of the correlator and so improves matching characteristics.

4.4.4 Characterizations of the VLSI Hardware

The integration of current onto the bucket brigade nodes closely approximates a current-to-voltage conversion, resulting in a linear voltage change at the bucket brigade stage proportional to the input current. The current-conveyor driver circuits ensure a quick response to the ϕ_3 pulse even for very low-level input currents. The current-to-voltage conversion ideally should be linear; thus an important characterization of the hardware is determining this response. The linearity can be determined by randomly selecting one template memory cell, enabling it, disabling all the other cells, and presenting the system with a single pulse on the same channel as the selected memory cell, having a duration of one cycle of the bucket brigade. The system is clocked until the output changes in response to the input pulse, and the height of the output is measured. This procedure is repeated for a range of amplitudes of the input pulse, and for different randomly-selected array positions. The response of our system is shown in Figure 4.21.

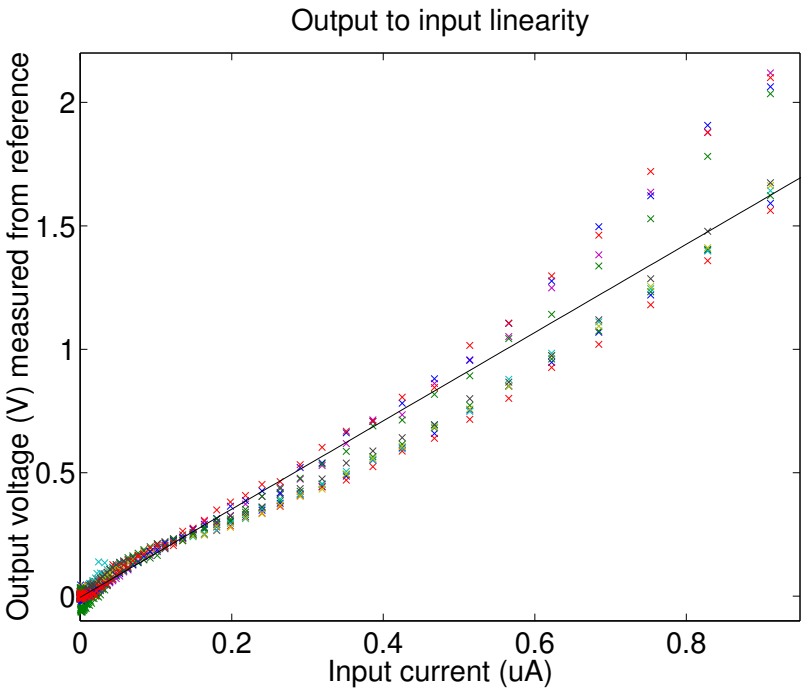


Figure 4.21: Linearity between output voltage and input current.

The charge-transfer efficiency of a BBD is not as high as that for a CCD; the measured efficiency of our BBD is 99.7% per stage.[3] A signal passing through the entire bucket brigade (64 stages) loses 15% of its charge, but this does not significantly affect the classification performance of the system. Figure 4.22 shows the response of the BBD to a fixed current of 1 µA integrated onto one of the 64 taps in the BBD over a fixed duration (width of ϕ_3 pulse) of 3.4 µs, then relayed to the output. We repeat the measurement for each of the 64 BBD taps. The randomly distributed variation in outputs (14.5%) is typical for device mismatch in the MOS transistors which mirror the input current onto the bucket brigade capacitor (see Figure 4.20). The charge transfer efficiency is determined by matching the tail of the impulse response (the slight rise in the baseline reference toward higher taps) to a simple model of charge transfer.

Disregarding the losses due to charge transfer inefficiency, we can use the output of the BBD to generate a map of the transistor mismatch across the template, as shown in Figures 4.23 and 4.24 for input levels of 1 µA and 100 nA, respectively. Systematic offsets exist in both channel number and time bin, following a vaguely sinusoidal pattern similar to that reported in [63]. These

[3] This represents the best response out of several fabrication runs. Why the other fabrications showed decreased charge transfer efficiency for essentially the identical circuit layout has not been determined.

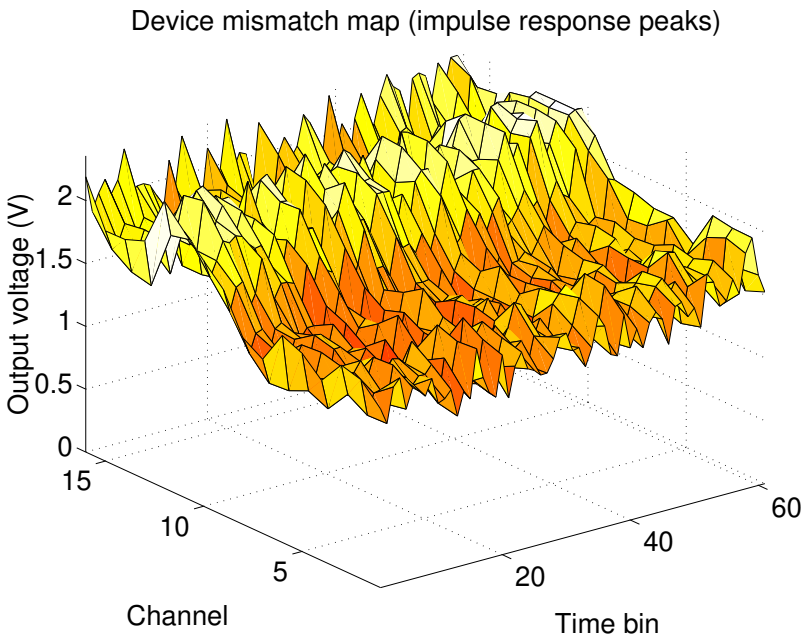


Figure 4.23: Matching between devices throughout the template at 1 µA input.

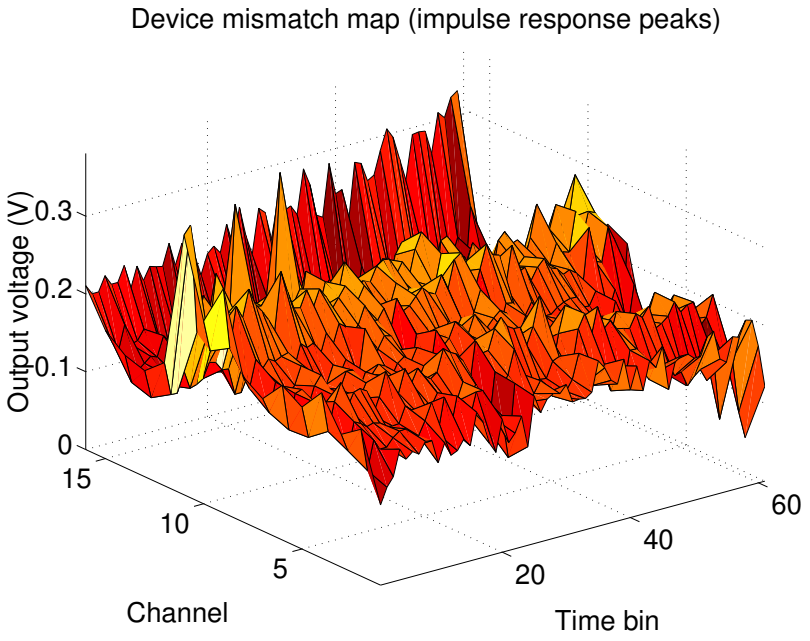


Figure 4.24: Matching between devices throughout the template at 100 nA input.

Among the tests we performed was to take a template for a pre-recorded acoustic transient from our simulation program, download it to the chip, then compare its output to that of the simulation program, where both are given the same input. We used two different transients for the input, one corresponding to the template class and the other a different class example. Figure 4.26 shows both measured and simulated output of the correlation between the template for "can" and a "can" sound and a "finger snap" sound as input. Figure 4.27 shows the opposite case, the correlation between the template for "snap" and the same transient inputs. Residual errors between the simulated and measured analog outputs are shown at the bottom of each plot. Most of the residual error is systematic and can be traced to the nonlinearity in the current to voltage conversion (Figure 4.21).

Total power dissipation of the acoustic transient processor depends primarily on the static dissipation of the amplifiers in the switched capacitor circuit and the bias current in the current conveyors driving the bucket brigade inputs. At minimal values of these biases allowing correct system operation (V_{casc} = 0.4 V and amplifier bias = 0.6 V), average power dissipation is 30 µW with a peak dissipation of approximately 50 µW during the onset of a transient input.

4.4.6 Summary

We have designed and fabricated a chip intended for use as a classifier of transient (short-term) acoustic signals. A signal to be classified is decomposed into an array of energy envelopes across a set of frequency channels, and normalized across all channels. The chip uses analog current-mode circuits to produce an output voltage encoding the running correlation between this time-frequency representation of the input and a stored template of binary values. The pattern-classification algorithm which the chip implements has been shown through simulation to be robust to binarization of the template values when the template values encode the expected sign of the time sample or pairwise channel difference of the normalized inputs.

Test results show that the chip output voltage closely follows predicted values from simulations. The chip can be used to classify acoustic transient events quickly, accurately, and with a high degree of robustness due to a parallel, pipelined architecture. The power consumption of 30 µW per template is significantly lower than that for a DSP or microprocessor performing the same computation, and the layout area of 700 µm × 1170 µm per template allows a high integration density.

4.5 The Digital ATP

A notable conclusion from the software trials of the Acoustic Transient Processor was the observation that comparing binary inputs against binary templates, contrary to early results (which were flawed), yields results comparable to the continuous-valued input case. In some trials, the results from the binary-binary case were better than those for any other configuration.

Consequently, it is possible to build a correlator which is entirely digital. It is not "digital" in the traditional sense of a DSP or microprocessor, with floating-point operations on wide data busses, but rather involves operations efficiently and simply implemented at the bit level with opportunities for significant amounts of parallelism. The frontend processor of such a system remains analog (though not necessarily continuous-time), and may in fact be the same frontend as designed for the original ATP system. Three changes to the frontend architecture should be noted:

- Because the L-1 norm function does not change the relative signs of the outputs, but only serves to scale the entire output, the normalization does not change the binary output and may be omitted.
- Each output is compared against that of its neighboring channel using a comparator and the result is a single bit per channel. There is one less output than there are channels.
- The outputs also may be compared against a single threshold value resulting in a second bit per channel. This gives some minimal amplitude information, and can be used to prevent the system from responding to noise during quiet periods of the input. Typically, the two-bit output will be evaluated as a trinary number taking on values -1, 0, and 1.

The frontend filterbank system as proposed can be considered a "smart A/D" which takes a continuous-valued input and produces a set of, say, 16 to 32 bits comprising a feature vector evaluated at 1 ms (or similar) intervals.

4.6 Digital correlator custom VLSI architecture

Because a binary-valued input significantly decreases the complexity of the associated correlator circuitry, it is possible to explore more complex encoding schemes for both the input and template, such as binary-trinary and trinary-trinary correlations. There are additional reasons to investigate alternate ways to produce an efficient correlation value. In the additional software trials,

In digital design, particularly in regard to adders and multipliers, a convenient way to reduce circuit complexity is to shift the burden of the processing from space to time by serializing part or all of the process. This is exactly what is called for in the case of the ATP correlator, since once the system has been pipelined column-wise, the number of serial operations performed per template per unit time is small: in fact, it is one exclusive-or followed by a single addition followed by a register shift. This leaves considerable opportunity for serial operation. Namely, the addition at each column can be replaced by a counter, in which the input is presented one channel at a time; this architecture is shown in Figure 4.28. There are two lines per column: one to hold the input value, selected by row, and one to hold the template value, also selected by row. Only one exclusive-or gate is required, at the bottom of the column, and the result of the exclusive-or sets the direction of an up-down counter that performs the summation serially for each row. At the end of each input cycle, the contents of each counter are shifted over to the next column, where the counting continues on the next cycle. The counter under each column needs only be as many bits wide as the maximum count possible at that column, so the size of the counter (in bits) is proportional to the log of the column number.

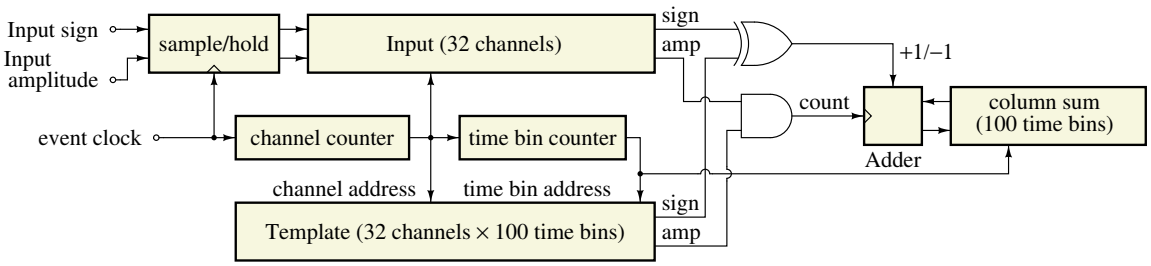


Figure 4.28: Block diagram of the sequential digital correlator architecture.

4.7 Digital correlator semicustom FPGA architecture

Even under the condition of an input presented serially rather than in parallel, the computational requirements are not great and the digital system can be operated on a long cycle time and at low power. The entire correlation can be made serial without exceeding the capabilities of off-the-shelf TTL and/or CMOS IC components. While a board-level design cannot be as power-efficient, compact, or scalable as a custom digital VLSI design, it can be made quite cheaply and with a short development time. Consider a rather large system, equivalent to the largest systems simulated in

The solution I adopted was to use the same template SRAM chip to keep the intermediate column results. The most simply implemented solution is to allow the intermediate store area to take away from the column space used by the templates. This leaves the number of columns unknown, but with the number of channels fixed at 32, determined by the following relation,

$$8192 \geq 2(32n) + (16n), \tag{4.25}$$

where we allow 16 bits for each intermediate result for the sake of simplicity in implementation, even though only 13 bits are necessary. The largest number n which satisfies the relation is $n = 102$. In the implementation we rounded this to 100, for reasons of convenience, one of which was to allow room to store the final correlation result. The memory allocation in the SRAM is depicted in Figure 4.29.

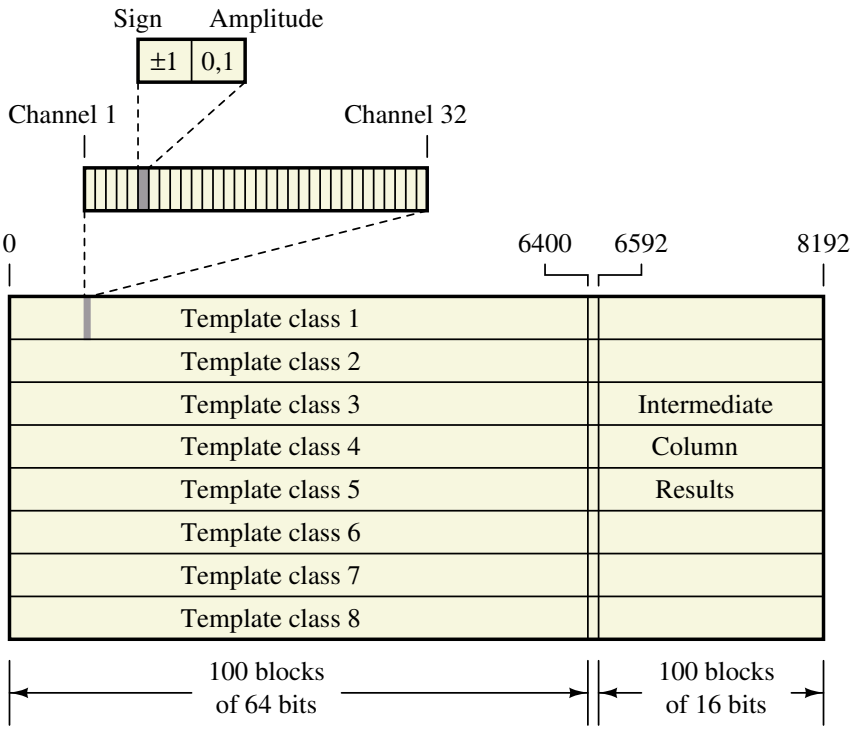


Figure 4.29: Memory allocation in the digital correlator SRAM.

Accessing memory is a matter of building a semicustom chip with a variety of binary counters. One seven-bit counter is needed to count from 100 to zero. The count must be downward, in order to effect a shift towards the output by reading each intermediate column sum, accumulating over the number of input channels, and placing the result into the storage area for the next column

template and column sum storage, one Actel 1020 part for the finite state machine controller, and another Actel 1020 part for the correlator itself.

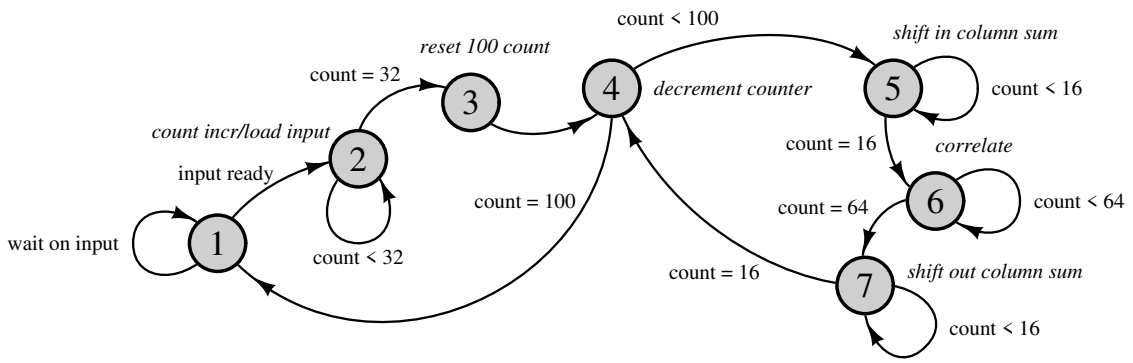


Figure 4.31: Digital correlator controller state diagram, simplified.

The correlator semicustom chip consists of eight registers in parallel, one for each data bit of the SRAM. Each register is a 13-bit arithmetic unit capable of 2's-complement increment and decrement and a right shift/rotate operation. At the front of each register is logic for translating the two bits of input and template into their trinary values and determining the appropriate counting operation (increment, decrement, or no operation). The correlator is depicted in Figure 4.32.

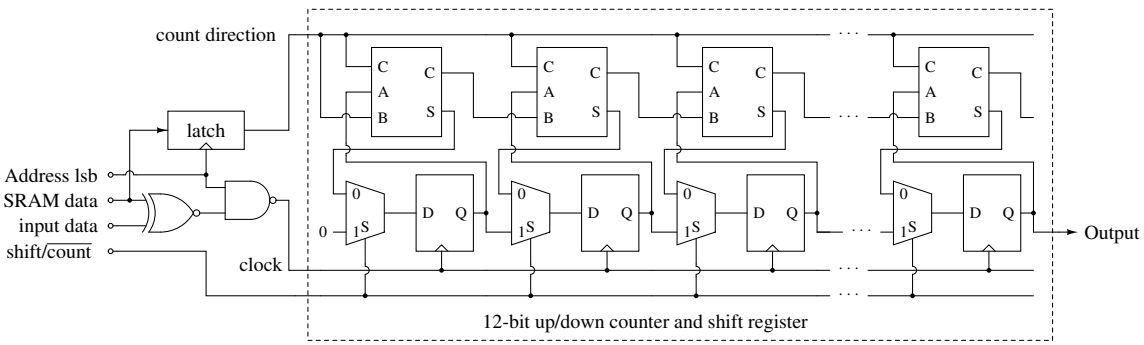


Figure 4.32: Serial digital correlator structure for one template.

In addition to the functions directly associated with the correlation, the correlator chip is also capable of performing a simple analysis of the output at each timestep and determine if any output exceeds a threshold and, if so, which output has the maximum value. The serial fashion in which the rest of the architecture is implemented makes this function especially simple. At the end of the first accumulate cycle, after a new input vector is presented to the system, the last column

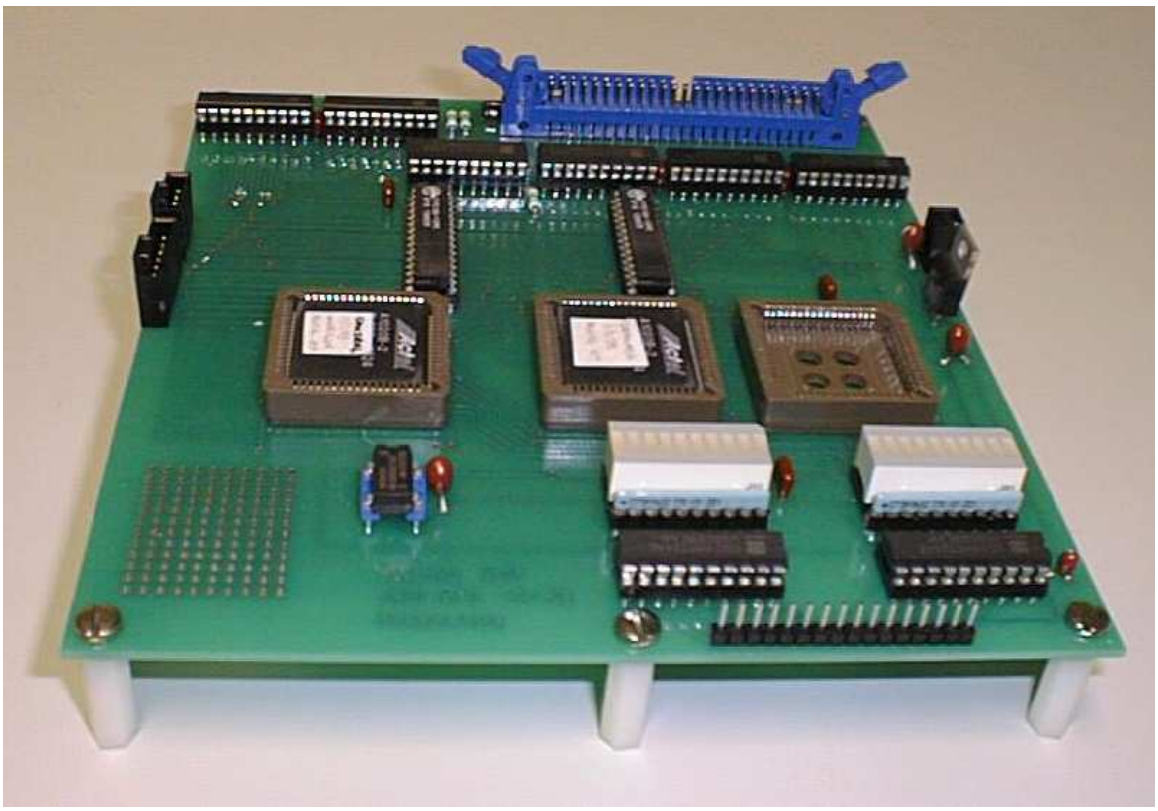


Figure 4.33: Digital ATP correlator system, configurable for up to sixteen templates.

ampl	sign	trinary
L	X	0
H	L	-1
H	H	$+1$

Table 4.9: 2-bit Sign-amplitude trinary representation of input and template values in the digital ATP. "X" represents a don't-care condition.

drives all the filter chips (plus perhaps a few clock signals derived by dividing down the master clock by powers of two through a series of flip-flops), but each filter requires a specific set of resistor values to create the correct gain and Q for the channel. Generated this way, filters are difficult to match. The other method is to generate a separate clock for each filter. A convenient way to generate the necessary clocks is to divide a suitably large master clock signal by integer values, one for each channel. This method is hardware-intensive, but it is completely digital and therefore can be programmed into a semicustom part such as an Actel chip, a small one of which is about the right size for driving a 32-channel system. It is also possible to make the system programmable, so that the board-level system enjoys an additional gain over the custom chip of allowing Mel-scale spacing (or, for that matter, arbitrary spacing) between bandpass centers.

Figure 4.34 depicts a complete bandpass channel. Each channel requires two chips: One MF10CCN dual general second-order filter and one MF4CN 4th-order Butterworth lowpass filter (both device numbers refer to National Semiconductor parts; pin-compatible devices are made by Linear Technologies). Three additional chips are shared between channels: one LM319N dual high-speed comparator, one LM339N quad comparator for the output, both shared between two channels, and one CD4053 triple 2-to-1 analog multiplexer, shared between three channels. External components are limited to five resistors for the bandpass filters (of which four have the same value; DIP-style resistor packs help improve matching characteristics), three pullup resistors (in SIP packages for minimum space) to interface the CMOS and TTL-compatible parts, and a resistor-capacitor pair to create a simple continuous-time antialiasing filter between the rectifier and the lowpass filter. To interface to the serial input method of the digital correlator, a few additional digital multiplexers are required at the output (a tree of four 8-to-1 muxes followed by one 4-to-1 mux, for a 32-channel system).

A separate system drives the filter inputs. This system consists of a microphone connector, a single-chip audio preamplifier (Analog Devices SSM-2017) followed by a simple R-C antialiasing filter and buffer (see Figure 4.35) [64]. The audio input has programmable gain and can be coupled to a buffered controller such as a sound card PCM codec on the controlling computer to generate

Figure 4.36 is a photograph of one of the two filterbank boards.

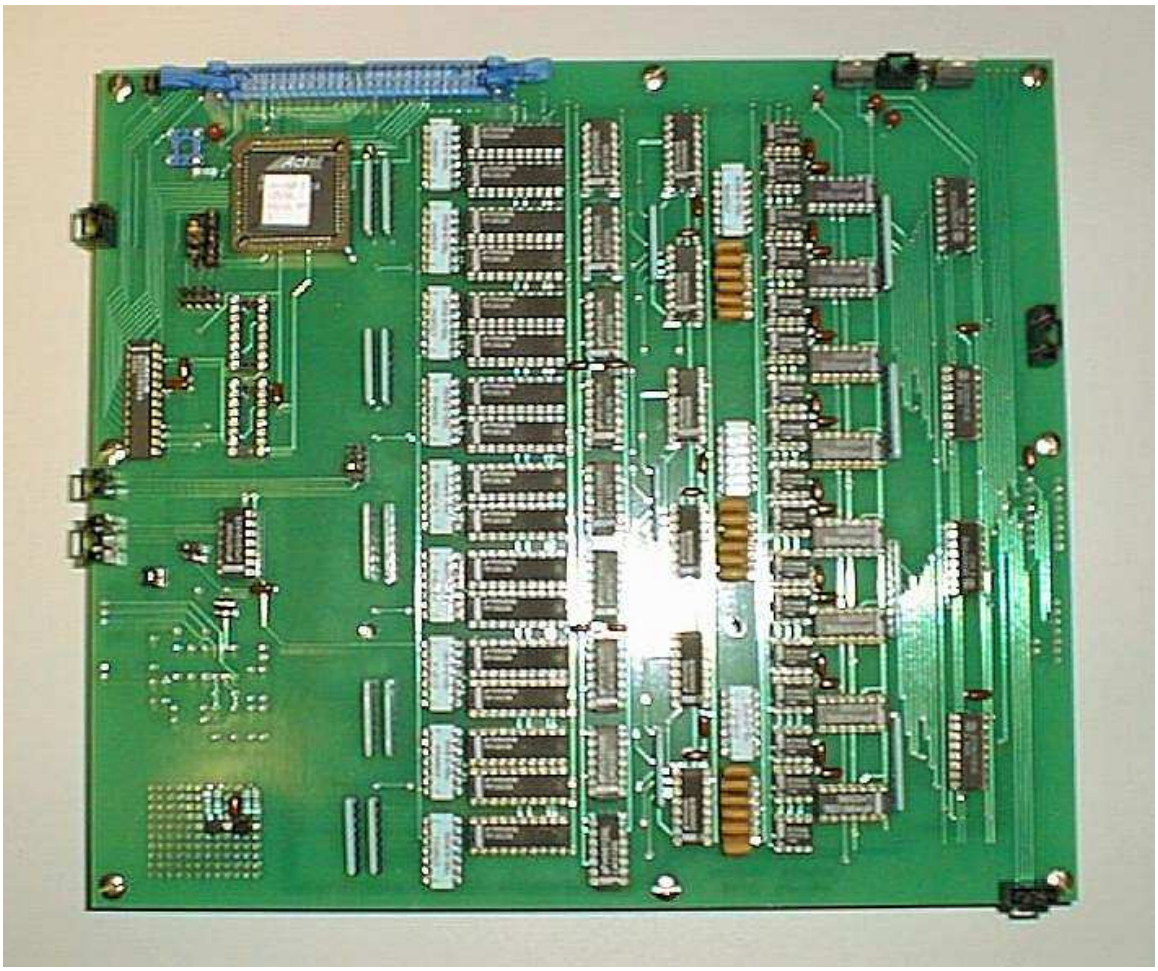


Figure 4.36: One board of the switched capacitor frontend filterbank, bandpass filtering and encoding the audio input into sixteen parallel channels.

4.9 Experimental results of the trinary-trinary correlation hardware

4.9.1 Method of input segmentation

When training a template correlation system using the average value of the training set examples, the output can only be expected to produce a reliable classification when the input is perfectly aligned to the template, not shifted in time to the right or to the left. This is very clearly shown in the output response of the trinary-trinary correlator as shown in Figures 4.40 and 4.41, in

which is denoted by a in the figure. The value a is thresholded by comparing it to a constant value, which was chosen as 7 for a system of 32 frontend filterbank channels. The two filters are approximated by a system of two counters whose timeouts allow the system to ignore short glitches in the "noisy" segmentation. Although the segmentation algorithm must operate in real-time, it can operate with a large amount of delay, because the transient onset occurs 100 samples (1/10 second) before the point of optimal alignment of input and template.

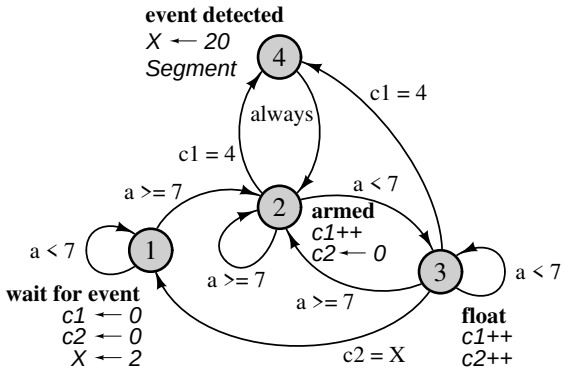


Figure 4.38: A simple finite state machine which detects the onset of transient events in the trinary-valued frontend filterbank output.

4.9.2 Method of template generation

Trinary input values and trinary template values allow numerous variations on both the method of tempate generation and the method of correlation computation. Here we present only the most effective of those training algorithms we investigated. The method presented below assumes that all events in the training data have been "tagged" by a segmenter, whether real-time or non-real-time, automatic or human, and that values in the input array $x[t, m]$ take on values $\{-1, 0, +1\}$:

1. For all events in the training dataset:

   (a) Align the training data on tags and fill to length of template (100 samples).

   (b) Sum all aligned instances of each class using the relation

$$p[n, m] = p[n, m] + x[t - n, m]. \tag{4.26}$$

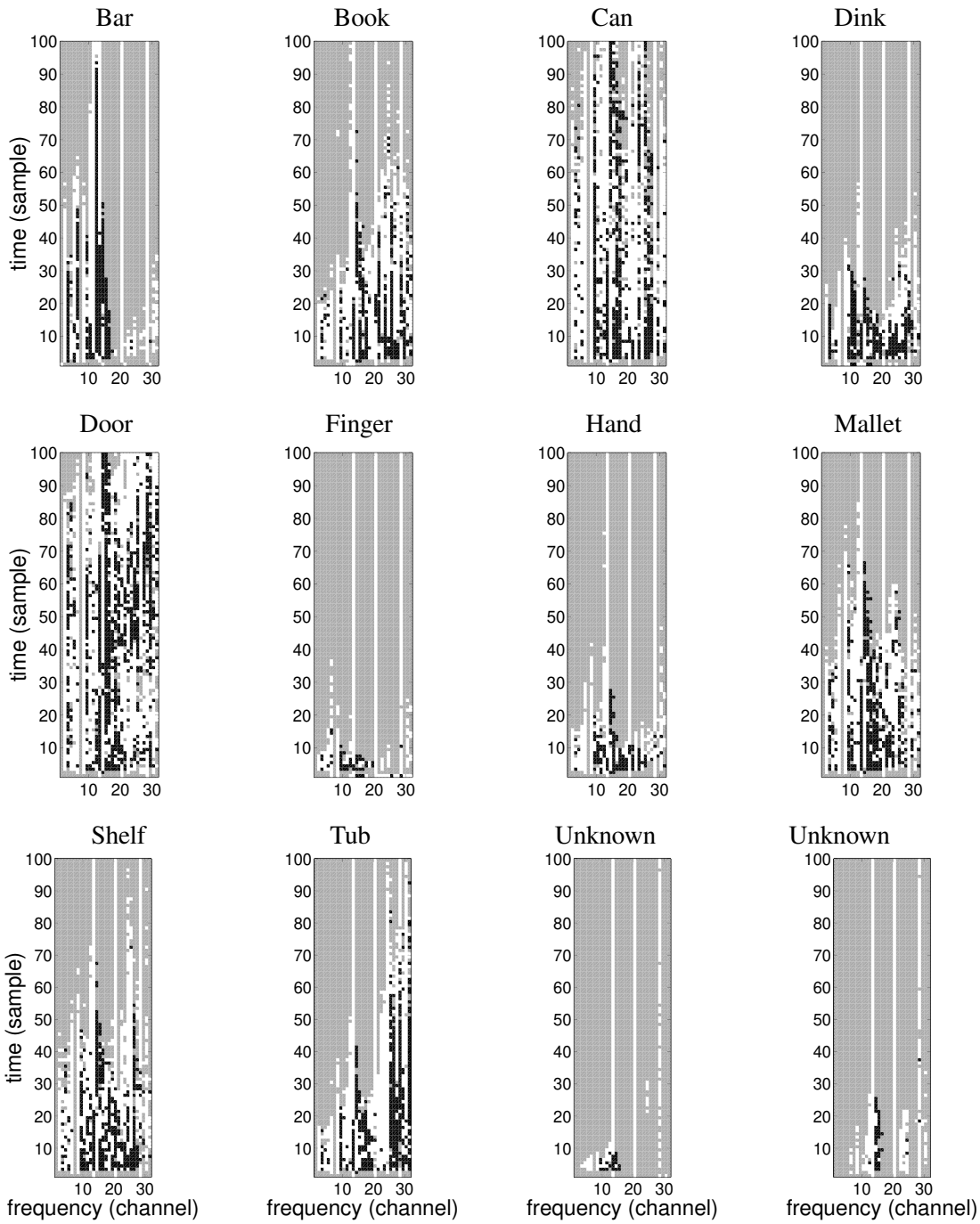


Figure 4.39: Templates for twelve transient classes, determined from data obtained from the front-end hardware. Frequency is on the x-axis, with the lowest frequency channel on the right, and time is on the y-axis, with the transient onset at the bottom of the template. Colors correspond to trinary values as follows: black = -1, white = $+1$, gray = 0.

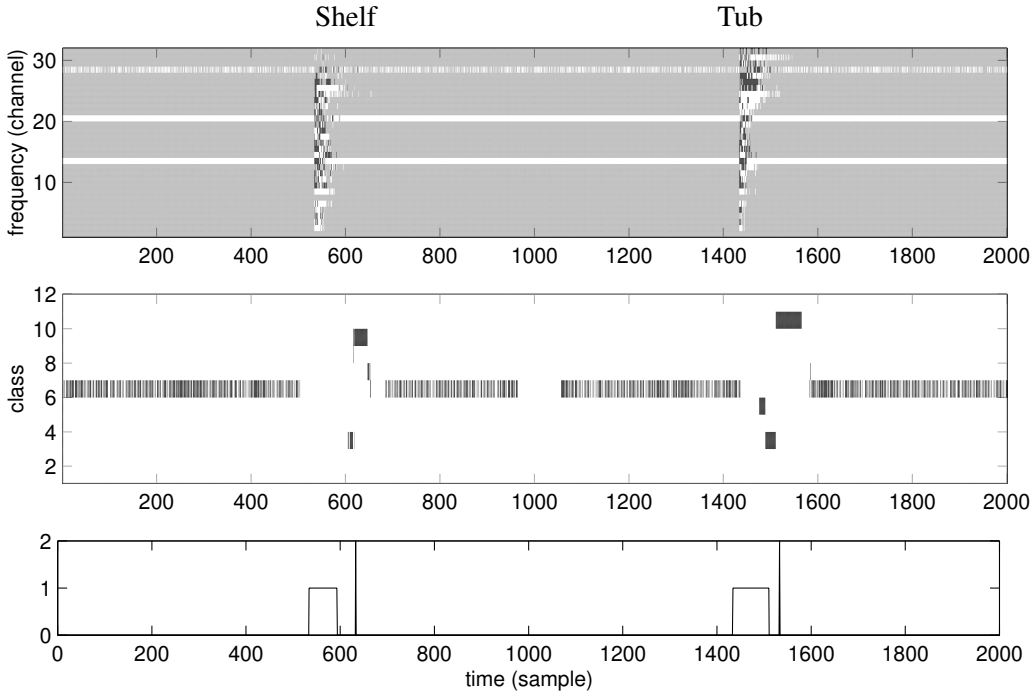


Figure 4.41: System input and output for sound "shelf" (left) and "tub" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

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	19	0	0	0	0	1	0	0
Dink	0	0	0	18	0	0	3	0	0	0
Door	0	0	4	0	20	13	6	4	0	0
Finger	0	0	0	0	0	19	1	0	0	0
Hand	0	0	0	0	0	5	14	0	0	0
Mallet	0	0	1	0	0	0	1	17	1	0
Shelf	0	0	0	0	0	0	0	0	19	0
Tub	0	0	0	0	0	0	0	0	0	20

Total instances presented:	199
Total events found:	226
Correct:	186
Incorrect:	11
Missed:	2
Extra event inserted:	25
Correctly identified:	93.5%
Incorrectly identified:	5.5%
Missed:	1.0%
Insertions:	12.6%

Table 4.11: Real-time recognition task: Cross-validation on the transient dataset using the simple finite state machine segmenter.

Chapter 5

Learning and Speech Recognition

5.1 Automatic Template Learning for Template-Based Correlation

Up to this point, I have not dwelled on the method of obtaining a valid template to use for the class example in transient classification. The acoustic transient processor is a purely feed-forward system which has no local hardware for on-chip (or on-board) learning. Speed is of utmost importance for classification once the templates are known, and so the hardware has been developed precisely for this purpose. Training can be performed off-line and not in real-time, as long as we can assume that a classification system does not need to reconfigure or retrain itself during operation (which may not be a completely valid assumption for speech, but should be true for simple transient applications). This does not mean that training is or should be entirely separate from the classifier hardware. In the preceding chapter, I have alluded to "chip in the loop" learning. The method of "chip in the loop" learning assumes that even the best feed-forward network has nonlinearities and mismatch, and that the best model of such a system is the system itself. To train the classifier, the training system sets all the weights (template values), presents the training data to the classifier, runs the classifier, and captures the output. In an iterative learning scheme, it adapts the weights according to some learning rule, then performs the learning cycle again.

The correlation algorithm works under the assumption that a template should be a "prototype" of its class: it should have all the characteristics of an example of its class. This could be interpreted in several ways. One interpretation is that the template is the average value of many examples of the class, aligned by a segmentation procedure. Another interpretation is that the template contains features which correlate positively with features of all examples of its own class, but may be missing features which would correlate positively with examples of a different class, thus

frame events (*i.e.*, the speed at which the door is closed). Assume that the recorded examples are perfectly aligned on the door latch event. When they are all averaged together and copied to the template, the door latch event will remain distinct while the door frame event will be spread out into a Gaussian shape describing the probability of timing of the door frame event relative to the door latch event rather than describing anything about the door frame event itself. This does not invalidate the average-template method. But it does show how template averaging can lose valuable information about the transient class.

In the case of the door it should be obvious that the direct template method is not sophisticated enough to capture all the nuances of a closing door. Another way to put this is that a closing door is not strictly an acoustic transient, but is a composite event which displays some of the properties of complicated acoustic events such as speech. The closing door, by the way, is not an isolated example. Many such composite events exist in nature. A piano key is another example of a composite event: The sound of one part of the wooden hammer mechanism striking another occurs so close to the sound emanating from the vibrating strings that it is impossible to separate the two sounds perceptually. But the sound of a piano key with the transient removed simply does not sound much like a piano.

The example just described shows that ensemble averaging can produce a probability distribution which, while masking features on a smaller time scale, can contribute important information about the relative timing of events on a larger time scale. This can be repeated on many hierarchical levels, possibly down to the level of zero-crossings (which in the acoustic transient processor we have eliminated by smoothing the signal out to a 2 ms time scale). By creating the acoustic transient processor as a one-level, non-hierarchical architecture, we are forced to find a single time scale which is optimal for transient processing rather than making use of information at all scales. In that sense, what the acoustic transient processor is missing is a wavelet-like interpretation of its input. The situation is illustrative of why a filterbank in and of itself is not a wavelet transformer: In the template correlator, the filterbank outputs are sampled evenly across the spectrum, which forces the system either to oversample the low frequency channels or undersample the high frequency channels. Encoding a wavelet-like representation in the template correlator is a problem without an obvious solution; at very best, it is inconvenient to format into hardware. But the wavelet representation doesn't tell the whole story. In the continuous wavelet representation, larger time scales are captured by filtering lower frequency bands. But if we wish to interpret the behavior of a complex acoustic event over a period of one second, we certainly do not mean that we wish to view the signal spectrum in the 1 Hz range. In this case, the interpretation is more

tures. Under many situations, one of which includes the template correlator, the independent feature detector is easier to determine than the independent components themselves.

PCA and ICA tend to give qualitatively similar results, and formulating each in the framework of acoustic transient classification encounters the same problems, so I will focus here on only one of the two methods, ICA. Bell and Sejnowski [71, 72] have shown how ICA may be applied as a model of visual and auditory perception.

Independent Component Analysis is a purely linear construct. Is assumes that all inputs are a linear mixture of statistically independent parts, jumbled together in different amounts by some unknown natural process. Because the mixture is linear, the generating process is an unknown matrix of weights determining how much of each independent source is added to the output:

$$x = \mathbf{A}s \tag{5.2}$$

where x are the data presented to the input of the classifier, having been produced by a matrix of independent source vectors $\mathbf{A}$ and a mixing vector s (for simplicity, we assume that s is one-dimensional). Bell and Sejnowski present the problem as on of *blind source separation*; that is, the goal is to find the mixing matrix s and thus determine the original independent sources $\mathbf{A}$. This is useful, for example, to separate one speaker's voice from others in the background of a recording of a crowded room. ICA recovers the sources by applying the inverse of the problem,

$$u = \mathbf{W}x, \tag{5.3}$$

where u is a vector of weights which is a scaled and permuted version of s (the linear system cannot differentiate between different scales and orders of the input vectors, nor does it need to for the blind signal separation problem), and the inverse of $\mathbf{W}$ yields the similarly scaled and permuted version of matrix $\mathbf{A}$:

$$u = \mathbf{WA}s. \tag{5.4}$$

A classification system poses the same problem in a different way. In an ideal situation, each class of input is independent from the rest. Each class is one independent source, and real examples presented to the system consist of the class independent source (the "prototype") mixed with lesser amounts of other sources (including noise). The classifier itself is $\mathbf{W}$, and given an input set x, it produces an output u, consisting of elements u_i representing the classification result for class i. In the ideal case, u is very nearly a binary vector which is (nearly) one for the element i of u which corresponds to the class to which input x belongs, and (nearly) zero elsewhere. Consequently, Equation (5.4) which forms the core of the ICA algorithm is itself a classifier.

fewer outputs than weights per template, it does so by reducing the dimensionality of the input space and in that way is very similar to Linear Discriminant Analysis (LDA) techniques Initialization is a problem, but generally satisfactory performance results from initializing the templates with the original ensemble-average values.

2. The weight update rule applies only to continuously-valued weights. No ICA weight-update rule has been demonstrated for binary valued weights. This problem occurs for many kinds of learning systems, including the backpropagation algorithm for neural networks. All of these algorithms are related in one way or another to gradient descent, which requires an estimate of the slope of an error surface with respect to the derivatives of the weights. If the weights are binary, their derivatives are discontinuous functions, and an error surface cannot be evaluated. Solutions usually involve using continuous-valued weights but passing them through a sigmoidal function such as a hyperbolic tangent which has a well-defined derivative but can approximate a step function by scaling the x-axis. This tends to force weights to resolve to binary values but leaves open issues such as how to change the sigmoid scaling during training. This solution requires that training be performed on a model of the system rather than the system itself if the system (such as our acoustic transient processor hardware) contains only binary values. Trinary values are, naturally, even more complicated to deal with.

3. The ICA update rule is static in time and is best suited for static applications such as natural scene analysis. It does not adequately deal with statistical correlation in time that arises regularly in a system such as the acoustic transient classifier. It is not sufficiently powerful enough to know that independent components appear and disappear with transitions from one to another. A temporally-based ICA algorithm cannot be blind to its own history and must take such information into account. It may even require sufficient delay so as to observe both past and future inputs around the input which causes the weight update. This would significantly increase the complexity of the system.

5.4 Support Vector Machines

Another promising training method is the use of *Support Vector Machines* [73]. The theory of Support Vector Machines (SVM) provides a statistical framework for pattern classification for which most standard methods such as multilayer neural networks and radial basis functions are

mind, although the project never went further than the software simulation stage. The frontend to the system is quite similar to the ATP frontend (see Section 1.2.1): a filterbank of bandpass filters (in this case, 32) followed by rectification and smoothing, with the channel outputs modified by a center-surround computation (which in our tests of ATP architectures resulted in system behavior almost identical to that using pairwise channel differences).

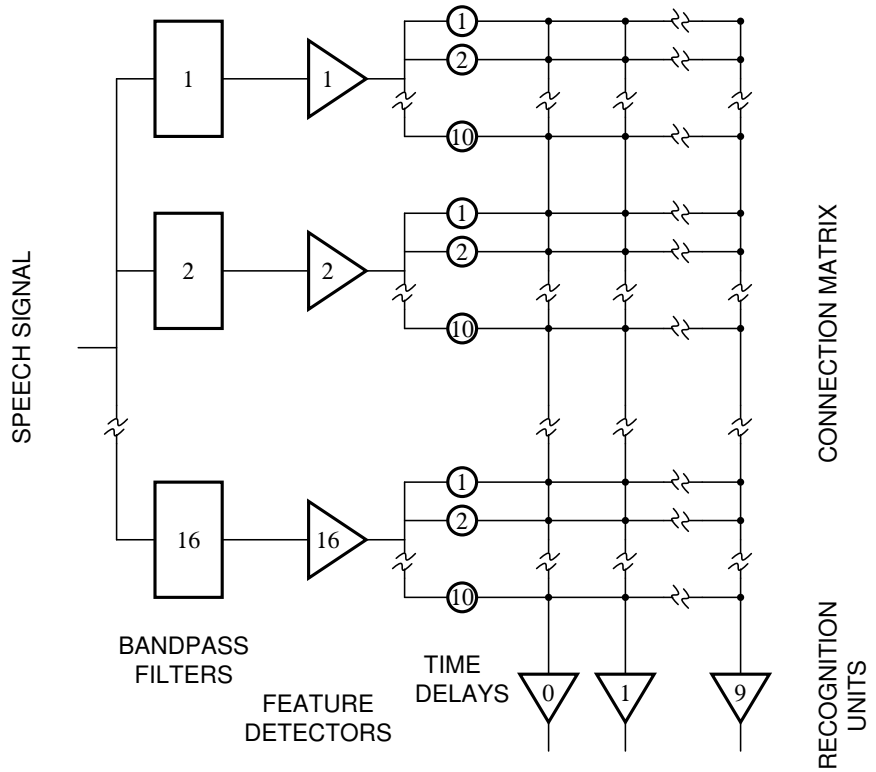


Figure 5.1: Unnikrishnan, Hopfield, and Tank correlator architecture, block diagram.

The correlation between input and template is also similar to that of the ATP, except the input signal is binarized, not the template. However, there are some key differences by which the Unnikrishnan et al. architecture copes with time variations in the input pattern: it treats event times as normally distributed variations around a fixed mean. Input signals from each (frequency) channel travel through a delay line where they diffuse as they travel, and the correlation is computed at ten "taps" spaced at regular time intervals (rather than at every time sample as in the ATP). The form of the correlation is (following the naming conventions of Equation (1.2)):

$$c_z[\,t\,] = \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{\tau=1}^{T} x[t-n,m]\, g[\tau,n]\, p_z[\tau,m] \tag{5.6}$$

where T is the number of taps, and $g[\cdot]$ is the set of amplitudes defining the Gaussian diffusion profile at each tap. Here I have taken the liberty of replacing a continuous integral, as it is presented in the original paper ([68], Equation (3)), with a discrete sum, as it is implemented in their software simulation. The Gaussian diffusion kernels (Figure 5.4) follow the equation

$$g(n,\tau) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(\tau - n\tau_{\text{delay}})^2}{2\sigma^2}\right) \tag{5.7}$$

where $\sigma = \sigma_0^2(n - \lfloor N/2 \rfloor)^2$ defines the Gaussian width at each tap. Parameters for scaling the Gaussian width σ_0 and the spacing between taps τ_{delay} are the main free variables of the system. If

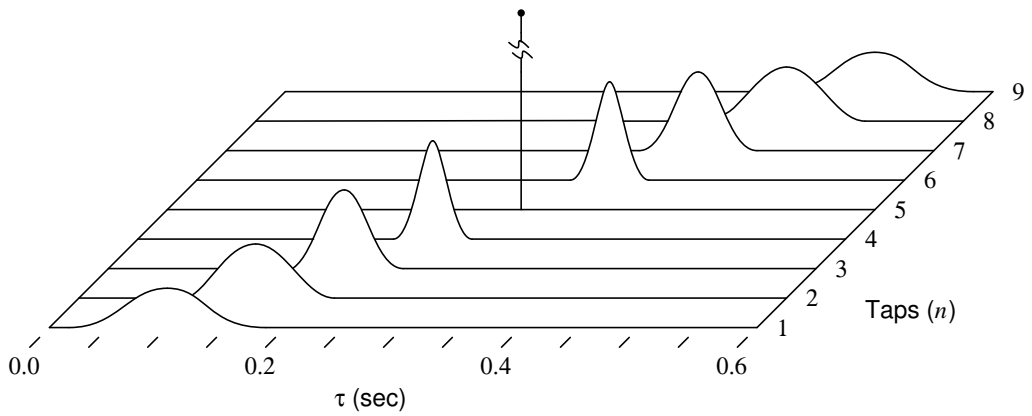


Figure 5.4: Gaussian kernels which diffuse the input in the Unnikrishnan *et al.* model.

we note that the weights and the Gaussian profiles are fixed, then we can define a function in which the weights are premultiplied by the Gaussian profiles to generate a new set of weights:

$$p'_z[n,m] = \sum_{\tau=1}^{T} g[\tau,n]\, p_z[\tau,m], \tag{5.8}$$

then the correlation takes a form equivalent to Equation (1.2):

$$c_z[\,t\,] = \sum_{m=1}^{M} \sum_{n=1}^{N} x[t-n,m]\, p'_z[n,m]. \tag{5.9}$$

This reveals the close connection with the ATP algorithm. In the previous section of this chapter, I explained how ensemble averaging over many inputs can produce templates which encode the probability distribution of the timing of events relative to the point of segmentation. This architecture does something very similar but assumes that the timing probability distributions are normal and encodes the normal distribution as a function (the Gaussian kernels) rather than part of the template. The arrangement of the equation in (5.9) moves the Gaussian functions back into the template. The

of that phoneme are duplicated for every instance. A hierarchical structure could easily remove that redundancy. Using one correlator per *phoneme* and employing a state-based model to detect transitions between phonetic states is a much more efficient use of hardware, and should create a system more robust to differences in the timespan of different instances of the same spoken digit.

5.6 Biologically-Inspired Methods

A potential use of the acoustic transient processor is the investigation of acoustic processing in the brain. Recent research by Shamma [70] shows that certain areas of the mammalian auditory cortex encode time-frequency maps in neural "wetware." Shamma subjected neural recordings to analysis and was able to produce the time-frequency mapping encoded by single neuron outputs. These maps are qualitatively similar to spatial mappings of the independent components of natural images as produced by Bell and Sejnowski [72]. A large proportion of both the neural and natural scene ICA outputs have Gabor logon-like characteristics (that is, sine- and cosine-modulated Gaussians in 2-dimensional space). This is not entirely surprising given the relationship of all three systems (wavelets, independent component analysis, and biological neural processing) to information theory and the efficient representation of signals.

A typical example of a map in the auditory cortex is shown in Figure 5.5. The figure represents the level of neural activity in response to auditory signals over a range of frequency and change of frequency with time. The neural activity is measured in spikes per second and averaged over numerous trials. Most of the map is unencoded—that is, the neuron does not respond at all, either with excitation or inhibition, to inputs in that range of time and frequency. Other areas produce an excitatory response (red) and others, an inhibitory response (blue). In the figure, the mapping is tuned to a falling tone. If the mapping represents the input directly (that is, without differentiation in time or frequency), then the mapping is strongly tuned to a particular falling formant but at the same time is strongly inhibiting a tone slightly below it in frequency and/or slightly lagging in time. On the other hand, if the mapping represents a channel-differenced encoding of the input, then the mapping represents a simple response to a single falling tone. A simple model of the distributions which covers most cases observed in the auditory cortex can be formed by one or two Gaussians encoding regions of either odd symmetry, as shown in the figure, or even symmetry consisting of a large excitatory response surrounded by inhibitory side lobes. The angle of the Gaussians corresponds to the rate of the rise or drop. At the extremes, a Gaussian with no rotation (horizontal) encodes a constant tone, and a vertically-aligned Gaussian encodes a very fast transient or some

to the system, whereas the correlator is a linear system (certain methods of training, such as independent component analysis, depend on this fact). Despite the differences between a real neuron and its model implemented as a linear, trinary correlation, we can expect that the system is in most ways like the acoustic transient system and should therefore remain reasonably robust after the same algorithmic manipulations.

Appendix A

Calculation of Linear Error in a Transconductance Amplifier

A fundamental limitation of analog multipliers such as those used in the original implementation of the continuous wavelet transform modulator (Section 4.3.2) is the small linear range of the output with respect to the differential input. This is also one of the primary reasons to consider the use of log-domain filters, with their large-scale linearity, as a replacement for transconductance-C filters (Chapter 3). Transconductance amplifiers operated in subthreshold have inherent limits on linear range. The approximate linear range of a simple transconductance amplifier can be calculated as shown below. Source degeneration and other methods [5] can extend the linear range by a factor of two to perhaps ten. Above-threshold operation extends the linear range because the square-law transistor behavior gives rise to a flatter amplifier transfer function than does the subthreshold exponential behavior, but the range is nevertheless severely limited.

Figure A.1 shows a MOS differential pair as used in a simple transconductance amplifier. The output $I_2 - I_1$ is shown differentially, although normally the output is made single-ended by mirroring I_2 and subtracting I_1 from it.

We use a somewhat simplified MOS subthreshold equation for drain current:

$$I_{ds} = I_0 e^{(\kappa V_g - V_s)/V_t}; \tag{A.1}$$

making other simplifying assumptions, such as perfectly matched input transistors,

$$I_2 - I_1 = \Delta I = I_0 e^{-V_s/V_t} \left(e^{(\kappa V_2)/V_t} - e^{(\kappa V_1)/Vt} \right) \tag{A.2}$$

$$I_2 + I_1 = I_b = I_0 e^{-V_s/V_t} \left(e^{(\kappa V_2)/V_t} + e^{(\kappa V_1)/Vt} \right) \tag{A.3}$$

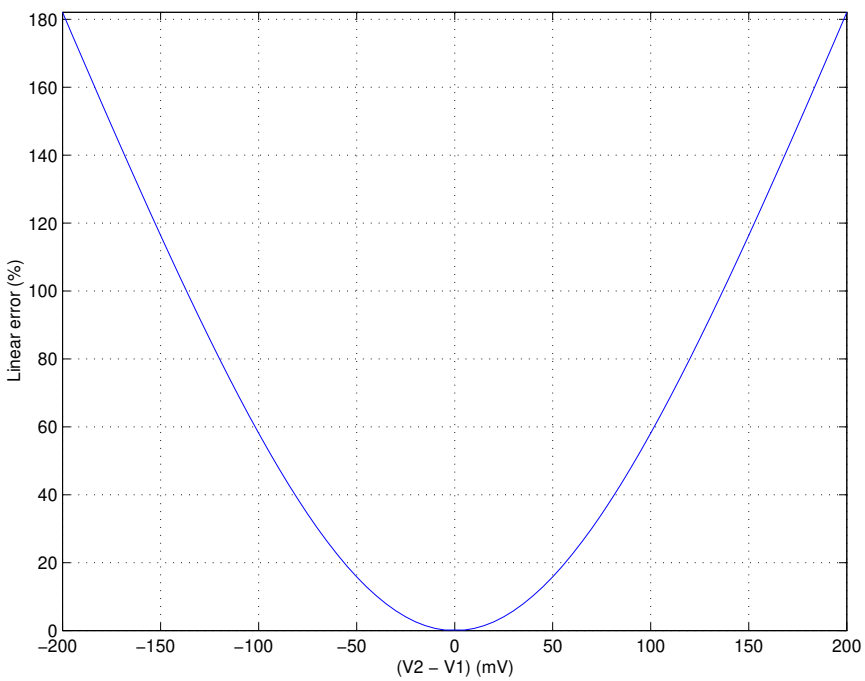


Figure A.2: Linear error in a simple transconductance amplifier as a function of the differential input voltage.

```
    factor = 2 * cap * (cap + 1);

30  ival = [0: (period / 2) - 1];
    singleAtten = 1 ./  (1 + factor * (1 - cos(norm * ival)));
    Atten = exp(log(singleAtten) * stages);

    figure(1);
35  semilogy(Atten);
    title('Lowpass filter function');

    %-------------------------------------------------------------
    % Derived values
40
    maxPattern = 2^nOpt - 1;
    nBits = period / 4;
    xbase = [1: nBits];  % for graphing

45  % Initialize bit sequence and perturbative sequence

    bits = ones(nBits, 1);

    % Initial values for iteration
50
    bestSNR = -100;         % in dB, conservative starting point
    iter = 0;

    %-------------------------------------------------------------
55  % Iterate:
    % Non-terminating loop---just print out results as they come
    %-------------------------------------------------------------

    while 1,
60     iter = iter + 1;

       % Compute optimal SNR:
       %  updates bitsF, assigns a random index, and performs
       %  partial exhaustive optimization with leftmost given
65     %  index. Yields the minimum SNR under all combinations.

       maxSNR = -100;
       index = random('Discrete Uniform', nBits, 1, 1);

70     maxbits = bits;
```

```
        if  maxSNR > bestSNR,
           bestSNR = maxSNR;
           bits = maxbits;
           fprintf('iteration #%d: SNR = %f\n', iter, bestSNR);
120        fprintf('bit pattern: ');
           for i = 1: nBits,
              fprintf('%1d', (bits(i) > 0));
           end;
           fprintf('\n');
125
           % Plot the FFT outputs for this iteration

           figure(2);
           semilogy(xbase, Raw(1:length(xbase)), 'bx',...
130                 xbase, Amp(1:length(xbase)), 'ro');
        end;
     end;

     %-------------------------------------------------------------
```

B.1 Commentary

This code is completely self-contained, and for the most part self-explanatory. It attempts to find a sequence of n bits, where n is a power of 2, which best represents a sine wave. The bulk of the processing is taken care of by the Matlab built-in Fast Fourier Transform (FFT) in line 93. The iteration loop (line 58) does not terminate but prints out the bit sequence (lines 112 to 120) whenever it reaches a new minimum.

The algorithm, which has no proof of convergence and is not in any way guaranteed to find a global minimum, performs an exhaustive search over a subset of bits within the target sequence. The position of the subsequence is randomly determined (line 67), with the subsequence itself being a loop size of `maxPattern`, calculated from the given number of subsequence bits `nOpt` in line 40. The inner loop exhaustively searching the space of bit patterns in the subsequence (0 to `maxPattern` $-$ 1) comprises lines 71 to 110. Only one quarter of the sequence, `bitsP`, needs to be generated directly from the current best sequence and the current subsequence pattern. The remaining three-quarters of the sequence are formed by reversing the sequence (line 88) and negating both the original and reversed sequences (line 93, where the total sequence is constructed

Appendix C

Template Correlation Algorithm with Time Differentiation

The algorithm for template correlation presented in Chapter 4 uses a computation of the pairwise difference between neighboring channels to transform the input and template into a zero-mean representation, from which a binary form may be easily extracted for either or both as required by the implementation.

In addition to frequency channel difference calculations, another way to put the input and template in to an easily quantizable form is a time differentiation. In the time-sampled system, this amounts to taking the difference between successive samples of the input or neighboring time bins of the template. Time differentiation did not prove to be especially effective by itself for acoustic transient classification, but in combination with channel difference computations, it can increase overall system robustness, and may also be useful by itself for certain classification tasks. Its form is especially efficient for hardware implementation, and it is the method used for the first prototypes of the mixed-mode VLSI transient classifier system.

The input is treated in the same manner as it is for the channel differencing system:

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

where the constant value θ suppresses noise during quiet intervals in the input.

1. We need to compute only one time difference rather than M differences (one for each frequency channel).

2. Architecturally, the algorithm is less affected by device mismatch when computing the output based on the difference of successive outputs rather than the absolute value of the output.

3. Most importantly, since the inputs x are rectified, the product xp' (when p' is binary $[0, 1]$ is always positive and equals either x or zero, which allows us to conveniently implement the entire convolution as an array of simple on/off current switches carrying current in one direction only.

Figure C.1 shows the system as described, where each template value is a single bit controlling a switch (multiplexer) which adds either zero or the unidirectional current input to the sum. The bucket brigade device is responsible for accumulating the summed current over time, and a simple switched capacitor circuit takes the difference at the output.

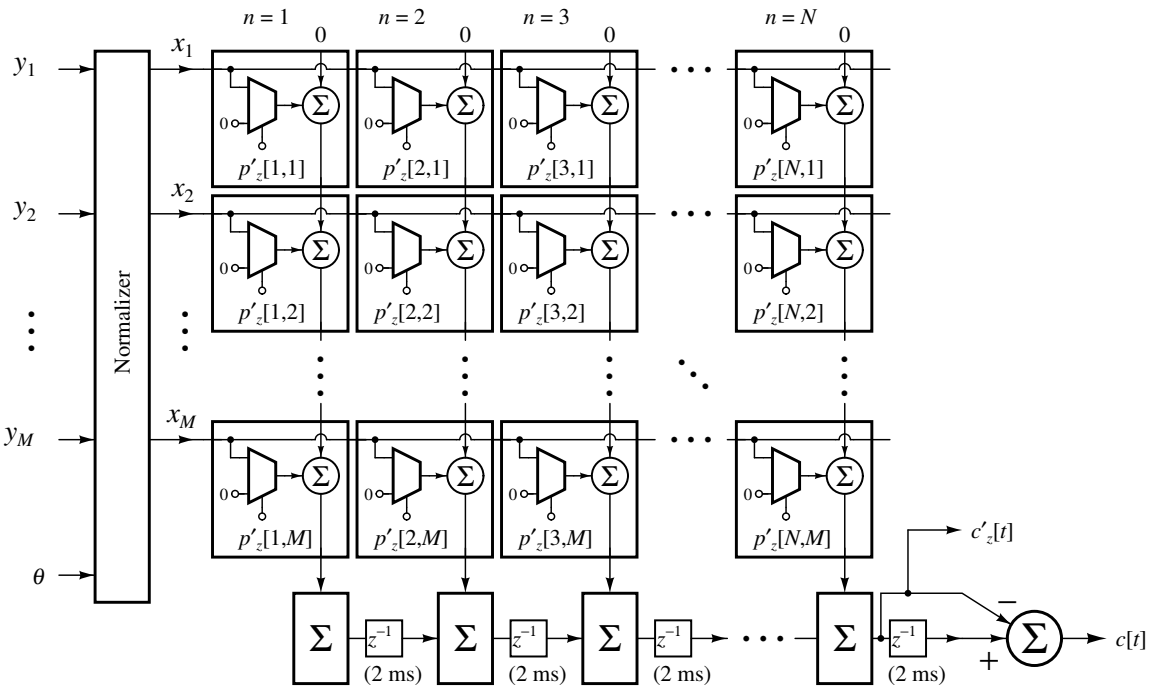


Figure C.1: Block diagram of the temporal current correlator, using time-differencing operations on the input.

This is the original correlation equation, although the indices for x' and p' do not match up one-to-one; as noted in the text, the template values must be reversed from right to left to make the template for the pipelined architecture equivalent to the original template. That is, $p'_z[n,m]$ becomes $p'_z[N-n+1,m]$ for all n, and

$$c_z[t] = \sum_{n=1}^{N} \sum_{m=1}^{M} x'[t-N+n-1,m]\, p'_z[N-n+1,m] \tag{C.17}$$

With a change of variables $n \rightarrow N-n+1$ (n counts N to 1 rather than 1 to N):

$$c_z[t] = \sum_{n=1}^{N} \sum_{m=1}^{M} x'[t-n,m]\, p'_z[n,m] \tag{C.18}$$

which is the original correlation equation.

The time-differential correlator circuit thus requires only one bucket-brigade device and avoids the problem of matching. Matching is, in fact, quite good due to the differential output which compares the difference between the values on the bucket brigade output node at successive time samples. The differential measurement reduces errors due to systematic offsets in the bucket brigade output. A simple switch-cap circuit which computes the time difference is shown in Figure C.2.

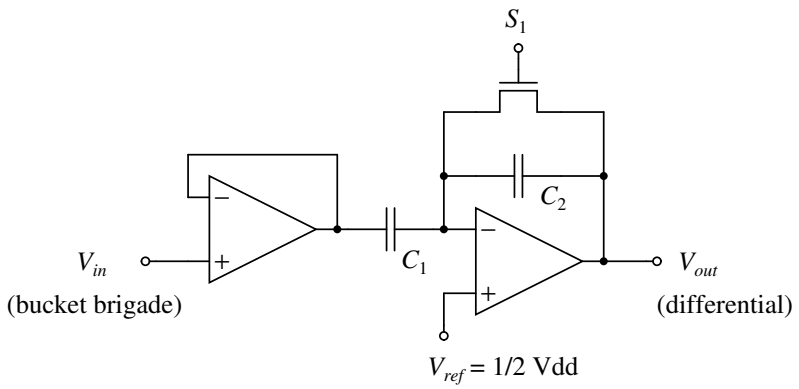


Figure C.2: Switch-cap time-differencing circuit.

Appendix D

C-code fragment from simulation of the Acoustic Transient Processor frontend filterbank

```
/*-------------------------------------------------------*/
/*  frontend.c                                           */
/*  A modification of the original frontend.f program.   */
/*-------------------------------------------------------*/
/*  This program computes the bandpass filter and        */
/*  detector outputs for PCM input data files.           */
/*  The file determined by the  command line filename    */
/*  is sequentially read for the names of speech files   */
/*  to be converted.                                     */
/*-------------------------------------------------------*/

#include "atp.h"
#include "defines.h"
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <sys/wait.h>

#define abs(a) ((a) < 0 ? -(a) : (a))
#define max(a,b) ((a) < (b) ? (b) : (a))
#define min(a,b) ((a) < (b) ? (a) : (b))
```

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

70  void makefilters()
    {
       short nfilt;
       float X, Y, x1;
       double nn = 1.0 / (double) (NFILTERS - 1);
75     double K = (TOPFREQ - BOTFREQ) / STEPFREQ;

       /* The equation for X has no symbolic solution, so     */
       /* compute the coefficient X by iteration.             */
       /* Note that X is generally close to 1 but 1 is also a */
80     /* solution of this equation, so we'll start above 1   */
       /* and iterate downward to the solution.               */

       X = 2.0;
       do {
85        x1 = X;
          X = (float) exp(nn * log(1.0 + K * (double)(X - 1)));
       } while ((x1 - X) > 0.000001);

       Y = STEPFREQ - BOTFREQ * (X - 1);
90
       filterfreq[0] = BOTFREQ;

       for (nfilt = 1; nfilt < (NFILTERS + PREPROC); nfilt++)
          filterfreq[nfilt] = filterfreq[nfilt - 1] * X + Y;
95  }

    /*-------------------------------------------------------*/
    /* Computer filter coefficients for each frequency band */
    /*-------------------------------------------------------*/
100
    void filtinit()
    {
       short nfilt;
       float sfreq = SFREQ;
105    float Q, denominv, tau, tausq, tauQ;

       makefilters();

       /* for now, fixed-Q filters */
110    Q = 5.0;
```

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

           /* 1st bandpass filter */
160
           bfiltdata[2][nfilt] = bfiltdata[1][nfilt];
           bfiltdata[1][nfilt] = bfiltdata[0][nfilt];

           bfiltdata[0][nfilt] =
165                abcs[0][nfilt] * (float)shortpcm[itime]
                 + abcs[1][nfilt] * (float)shortpcm[itimen2]
                 + abcs[2][nfilt] * bfiltdata[1][nfilt]
                 + abcs[3][nfilt] * bfiltdata[2][nfilt];

170        /* 2nd (cascaded) bandpass filter */

           filtdata[nfilt][itime] =
                   abcs[0][nfilt] * bfiltdata[0][nfilt]
                 + abcs[1][nfilt] * bfiltdata[2][nfilt]
175              + abcs[2][nfilt] * filtdata[nfilt][itimen1]
                 + abcs[3][nfilt] * filtdata[nfilt][itimen2];
        }
    }

180 /*-------------------------------------------------------*/
    /* Smooth with a 3rd-order (cascaded) lowpass filter     */
    /*-------------------------------------------------------*/

    makesmooth()
185 {
        short nfilt;
        float ntau;

        /* also for now, we don't decimate data down to a new */
190     /* timescale based on the Nyquist rate of the data    */
        /* smoothing (i.e., samples at 2ms each)              */

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

195        lfiltdata[1][nfilt] = lfiltdata[0][nfilt];
           lfiltdata[0][nfilt] = abcs[4][nfilt]
                     * (abs(filtdata[nfilt][itime])
                     + abs(filtdata[nfilt][itimen1]))
                     + abcs[5][nfilt] * lfiltdata[1][nfilt];
```

```
245  /*-------------------------------------------------------*/
     /* compute L-1 Norm and normalization channel            */
     /*-------------------------------------------------------*/

     computenorm()
250  {
        short nfilt;
        float nsum = NCHAN;

        for (nfilt = 0; nfilt < NFILTERS; nfilt++)
255        nsum += lpfiltdata[nfilt][itime];

        for (nfilt = 0; nfilt < NFILTERS; nfilt++)
           normdata[nfilt][itime] = NORMTO
                     * (lpfiltdata[nfilt][itime] / nsum);
260
        normdata[NFILTERS][itime] = NORMTO * (NCHAN / nsum);
     }

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

D.1 Commentary

The software simulation of the ATP frontend filterbank processor sufficed for tests of the ATP correlator in the absence of a hardware frontend processor. It is a direct implementation of the intended function of the hardware. Filter transfer functions have been converted from the continuous $j\omega$ frequency domain into the discrete-time z-domain using bilinear transforms.

The channels of the bandpass filterbank can have either a mel-scale or exponential center frequency distribution (90 to 92). Routine `makefilters()` facilitates the computation by determining the frequency spacing from the parameters `TOPFREQ`, `BOTFREQ`, and `STEPFREQ`. The coefficients of the recursion have no symbolic solution and so are determined iteratively (lines 80 to 88).

Routine `filtinit()` (line 98) determines the coefficients of the z-domain IIR filters from the center frequency distribution and the designated constant Q value.

Routine `filterbank()` (line 152) performs a single time sample computation of the filters, retaining the internal values for subsequent computations. In keeping with the hardware spec-

Appendix E

C-code fragment from simulation of the ATP correlator

```
/*-------------------------------------------------------*/
/*                    template classifier                */
/*-------------------------------------------------------*/

#define MIN(a,b) ((a) < (b)) ? (a) : (b)

/*-------------------------------------------------------*/
/* correlate() -- correlates template with sample        */
/*-------------------------------------------------------*/

float correlate(float *template, float *data, int numchans,
        long samples)
{
    float dot = 0.0;
    int i, j;

    for(j = 0; j < samples; j++) {
        for(i = 0; i < numchans - 1; i++) {
            dot += (*template++) * (*data++);
        }
        template++;
        data++;
    }
    return dot/samples;
}
```

```
                t0 = *template++;
                d0 = *data++;

                dot += (t1 > t0) ? d1 : d0;
75
    /*          dot += (t1 > t0) ? (d1 - d0) : 0;           */
    /*          dot += (t1 - t0) * (d1 - d0);               */
            }
            data++;
80          template++;
        }
        return dot;
    }

85  /*---------------------------------------------------------*/
    /* correlate_new() --- correlates template w/data          */
    /*---------------------------------------------------------*/

    float correlate_new(float *template, float *data,
90          int numchans, long samples)
    {
        float dot = 0.0;
        float t0, t1, t2, d0, d1, d2;
        int i, j;
95
        for(j = 0; j < samples - 1; j++) {
            for(i = 0; i < numchans - 2; i++) {

                t0 = *(template + numchans);
100             t1 = *template++;
                t2 = *(template + numchans);
                d0 = *(data + numchans);
                d1 = *data++;
                d2 = *(data + numchans);
105             dot += ((t1 + t2 - 2 * t0) > 0) ?
                            (d2 + d1 - 2 * d0) : 0;
            }
            data += 2;
            template += 2;
110     }
        return dot;
    }

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

classes and determines which class is the "winner" based on the maximum returned dot product. Several versions of the correlation routine have been included here: Tests of variations of the algorithm were performed by uncommenting the necessary parts of the code and calling the correct correlation subroutine from `classify()`.

Line 128 is the original software simulation which allowed templates to be of variable length, depending on the maximum length of the recorded examples of each class. Because the hardware does not allow such flexibility, line 127 is a replacement which more accurately reflects the hardware by limiting the number of time bins in each template to a specific value (shown here set to 64).

The correlation routine at line 11 is the baseline algorithm, based on the direct multiplication of template and input. The routine `correlate_dt()` at line 30 returns a correlation between the binary, time-differenced template and the continuous-valued, time-differenced input. Routine `correlate_old()` at line 56 returns a correlation using channel differences, although several variations are explored in the commented lines (lines 73 and 74). Finally `correlate_new()` (line 86) includes information from both the time difference and channel difference and calculates a binary template value based on a zero-mean representation derived from both.

```
      for (i = 0; i < filters; i++) {
         gsum += gains[i];
      }
      gscale = (double)filters / gsum;

      /* adapt toward the solution */

      for (i = 0; i < filters; i++) {
         gains[i] *= (0.5 + gscale * 0.5);
/*       gains[i] *= gscale;                        */
         fprintf(stdout, "%8.3f", gains[i]);
      }
      fprintf(stdout, "\n");
}

/*-------------------------------------------------------------*/
/* Print the value of the minimization function.               */
/* return column number with lowest value                      */
/*-------------------------------------------------------------*/

double printerr(double **cgains)
{
   int i, j, k;
   double cval, sum = 0.0, lval;

   for (i = 0; i < classes; i++) {
      lval = gains[i] * cgains[i][i];
      for (j = 0; j < classes; j++) {
         if (j == i) continue;
         sum += exp(gains[j] * cgains[j][i] - lval);
      }
   }
   fprintf(stderr, "%8.3e  ", sum);

   return sum;
}

/*-------------------------------------------------------------*/
/* print out cross-correlations of weight matrices             */
/*-------------------------------------------------------------*/

void printgains(double **cgains)
{
   int i, j, k, t;
```

```
findgains()
{
   int i, j, k, t;
   int bcount;
   double **cgains;
   double tout, newgain, nsum, dsum, ndiff;
   double nngain, nndiff;
   double newerr, lasterr;

   cgains = (double **)malloc(filters * sizeof(double *));
   for (i = 0; i < filters; i++)
      cgains[i] = (double *)malloc(filters * sizeof(double));

   fprintf(stdout, "\nPer-template gain calculations:\n\n");

   for (i = 0; i < filters; i++) {
      gains[i] = 1.0;
   }
   printgains(cgains);
   lasterr = printerr(cgains);

   fprintf(stdout,
      "Computing gains by minimizing cross-correlations.\n");

   for (i = 0; i < filters; i++) {
      gains[i] = 1.0;
   }
   do {
      for (i = 0; i < filters; i++) {
         nngain = 1.0;
         bcount = 0;
         do {
            nsum = 0.0;
            dsum = 0.0;
            for (j = 0; j < filters; j++) {
               if (j == i) continue;
               dsum += exp(gains[j] * cgains[j][i]);
               if (cgains[i][j] > 0)
                  nsum += cgains[i][j]
                        * exp((nngain * cgains[i][j])
                        - (gains[j] * cgains[j][j]));
            }
            dsum *= cgains[i][i];
```

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

F.1 Commentary

This is a portion of the code used to independently confirm Fernando Pineda's results using a simulation of the frontend filterbank rather than the HEEAR chip output and operating in continuous time rather than depending on a segmenter, the main point of which was to look for false positives which might occur between presented examples.

This code fragment is the part of the program which determines gains for each class in an attempt to optimize the classification accuracy by maximizing the difference among cross-correlations of the class templates.

The main optimization loop has no proof of convergence and so cannot be guaranteed to find a solution; however, it seems to work well in practice.

Routine `findgains()` is the main iterative loop which seeks to minimize an error function as described in Chapter 4, Section 4.3.6. The error function is computed in lines 141 through 155. The cross-correlations `cgains[i][j]` between templates are computed using the current estimate of the per-channel gains `gains[i]` within the function `printgains()`, lines 66 to 101.

The choice of error function does not compensate for one problem: If all gains are multiplied by a constant, the error measure does not change, which is reasonable since the system accuracy does not change either. This extra degree of freedom allows the adapting gains to slowly drift up or down. The function `gainnorm()`, lines 22 to 38, keeps the gains centered around unity. The routine moves the gains toward the desired mean rather than jumping there in a singe step to avoid throwing the adaptation loop into a limit cycle.

Routine `printerr()` finds a global estimate of the error function which the main loop of the program uses both to report the status of the iterative loop and to automatically determine when the iterative process has converged.

[11] S. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **11**, 1989, pp. 674–693.

[12] P. P. Vaidyanathan, "Passive Cascaded-Lattice Structures for Low-Sensitivity FIR Filter Design, with Applications to Filter Banks," *IEEE Trans. Circuits and Systems*, **CAS-33** (11), November 1986, pp. 1045–1064.

[13] A. Grossmann, R. Kronland-Martinet, and J. Morlet, "Reading and Understanding Continuous Wavelet Transforms," *Wavelets: Time-Frequency Methods and Phase Space*, Springer-Verlag (1989), pp. 2–20.

[14] A. Cohen and J. Kovačević, "Wavelets: The Mathematical Background," *Proc. IEEE*, Special Issue on Wavelets, **84** (4), April 1996, pp. 514–522.

[15] N. Hess-Nielsen and M. V. Wickerhauser, "Wavelets and Time-Frequency Analysis," *Proc. IEEE*, Special Issue on Wavelets, **84** (4), April 1996, pp. 523–540.

[16] R. Bracewell, "Adaptive chirplet representation of signals on the time-frequency plane," *Electronics Letters*, **27** (13), pp. 1159–1161, June 1991.

[17] T. Edwards and M. Godfrey, "An Analog Wavelet Transform Chip," *IEEE Int'l Conf. on Neural Networks*, **3**, 1993, pp. 1247–1251.

[18] R. T. Edwards and G. Cauwenberghs, "Analog VLSI Processor Implementing the Continuous Wavelet Transform," *Neural Information Processing Systems (NIPS)*, Denver, 1995.

[19] J. Lin, W. Ki, T. Edwards, and S. Shamma, "Analog VLSI Implementations of Auditory Wavelet Transforms Using Switched-Capacitor Circuits," *IEEE Trans. Circuits and Systems—I*, **41** (9), Sept. 1994.

[20] R. T. Edwards and G. Cauwenberghs, "Oversampling Architecture for Analog Harmonic Modulation," *IEEE Circuits and Systems*, **43** (8), Aug. 1996, pp. 696–698, Aug. 1996.

[21] R. T. Edwards and G. Cauwenberghs, "A VLSI Implementation of the Continuous Wavelet Transform," *Proc. ISCAS '96*, **4**, 368–371, May 1996.

[34] A. Andreou and K. Boahen, "Neural Information Processing I: The Current-Mode Approach," *Analog VLSI: Signal and Information Processing*, M. Ismail and T. Fiez, eds. McGraw Hill, Inc., 1994, pp. 311–357.

[35] B. Gilbert, "Translinear circuits: a proposed classification," *Electron. Lett.*, **11**, pp. 14–16, 1975.

[36] E. Seevinck, "Companding current-mode integrator: A new circuit principle for continuous-time monolithic filters," *Electron. Lett.*, **26**, pp. 2046–2047, Nov. 1990.

[37] R. Wiegerink, *Analysis and Synthesis of MOS Translinear Circuits*, The Hague, Netherlands, Cip-Gegevens Koninklijke Bibliotheek (1992).

[38] J. Mulder, A. van der Woerd, W. Serdijn, and A. Roermund, "General Current-Mode Analysis Method for Translinear Filters," *IEEE Trans. Circuits Syst. I*, **44** (3), pp. 193–197, Mar. 1997.

[39] D. Frey, "Log Domain Filtering for RF Applications," *IEEE J. Solid-State Circuits*, **31** (10), pp. 1468–1475, Oct. 1996.

[40] Y. Tsividis, "Externally Linear, Time-Invariant Systems and Their Application to Companding Signal Processors," *IEEE Trans. Circuits Syst. II*, **44** (2), pp. 65–85, Feb. 1997.

[41] E. M. Drakakis, A. Payne, and C. Toumazou, "Log-domain filters, translinear circuits and the Bernoulli Cell," *Proc. ISCAS '97*, **1**, pp. 501–504, 1997.

[42] M. Punzenberger, C. Enz. "A new 1.2V BiCMOS Log-domain integrator for companding current mode filters," *Proceedings of ISCAS 1996, Atlanta, Georgia*, **1**, pp. 125–128, 1996.

[43] D. Perry and G. Roberts, "The Design of Log-Domain Filters Based on the Operational Simulation of LC Ladders," *IEEE Trans. Circuits Syst. II*, **43**, No. 11, pp. 763–774, Nov. 1996.

[44] R. Fox, M. Nagarajan, and J. Harris, "Practical Design of Single-Ended Log-Domain Filter Circuits," *Proc. ISCAS '97*, **1**, pp. 341–344, 1997.

[45] C. Toumazou, J. Ngarmnill, and T. S. Lande, "Micropower Log-Domain Filter for Electronic Cochlea," *Electronic Letters*, **30**, No. 22, pp. 1839–1841, Oct. 27, 1994.

[56] F. J. Pineda, G. Cauwenberghs, and R. T. Edwards, "Bang, click, thud, or whack?," *Johns Hopkins APL Technical Digest* **18** (2), pp. 244–253, April–June, 1997.

[57] R. T. Edwards, G. Cauwenberghs, and F. J. Pineda, "A Mixed-Signal Correlator for Acoustic Transient Classification," *Proc. ISCAS '97*, **1**, pp. 621–625, Jun. 1997.

[58] R. T. Edwards, G. Cauwenberghs, and F. J. Pineda "Optimizing Correlation Algorithms for Hardware-based Transient Classification," *Neural Information Processing Systems (NIPS)*, 1998.

[59] R. T. Edwards and G. Cauwenberghs "A VLSI Processor for Acoustic Transient Correlation" (to appear in) *IEEE Journal of Solid-State Circuits*.

[60] R. Coggins, M. Jabri, B. Flower, and S. Pickard, "A Hybrid Analog and Digital VLSI Neural Network for Intracardiac Morphology Classification," *IEEE Journal of Solid-State Circuits*, **30**, pp. 542–550, 1995.

[61] E. R. Fossum, "Wire Transfer of Charge Packets Using a CCD-BBD Structure for Charge-Domain Signal Processing," *IEEE Transactions on Electron Devices*, **38**, Feb. 1991.

[62] Panasonic, Inc., *Audio Signal Delay BBD Series* Data Book, 1994.

[63] A. Pavasović, A. G. Andreou, and C. R. Westgate, "Characterization of of Subthreshold MOS Mismatch in Transistors for VLSI Systems," Journal of VLSI Signal Processing, 1994.

[64] Analog Devices, Inc., "Self-Contained Audio Preamplifier" SSM-2017 Data Sheet, Rev. C, 1997.

[65] F. Jelinek, *Statistical Methods for Speech Recognition*, Cambridge, Massachusetts, MIT Press (1997).

[66] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-23, pp. 66–72, 1975.

[67] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, **37** (3), pp. 328–339, March 1989.

Vita

R. Timothy Edwards received the degree of Bachelor of Science in Electrical Engineering (BSEE) from Duke University, Durham, North Carolina in 1990, graduating Summa Cum Laude and with Honors. In 1992, he received the degree of Master of Science in Electrical Engineering (MSEE) from Stanford University, Stanford, California. In 1993, he came to Johns Hopkins University to pursue a Ph.D. degree in the department of Electrical and Computer Engineering, Whiting School of Engineering, under the direction of Professor Gert Cauwenberghs. He is currently employed as a senior staff researcher in the Space Department at the Johns Hopkins University Applied Physics Laboratory.