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

Abstract

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

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

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

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.

Contents

List of Figures

List of Tables

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.

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

Today, however, as digital systems become increasingly fast and densely packed and decreasingly expensive (though expense reflects more the economy of scale than the cost of production or operation), it is necessary to choose analog applications wisely. There is no point in creating an analog system which will be outperformed by an equivalent digital (microprocessor or DSP) system in a year or two when digital fabrication processes move to the next smaller feature size and system designs move to a lower power supply standard. The incredible advances of CMOS fabrication processes (as per Amdahl's Law) [7] remain virtually unchecked as yet by their fundamental limits. These advances and the prospect of their continuation have the tendency to reduce the subject of 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 daunting but often greatly rewarding endeavor. This thesis explores one small corner of the space of neuromorphic engineering design: I hope that in addition to educating the reader, it conveys a sense of the scope of possibilities contained in the field, and the sense of excitement and exploration felt by those whose research touches the neuromorphic engineering community.

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

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

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

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

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

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

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

- 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.

1.2 Mapping the Time-Frequency Plane

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

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

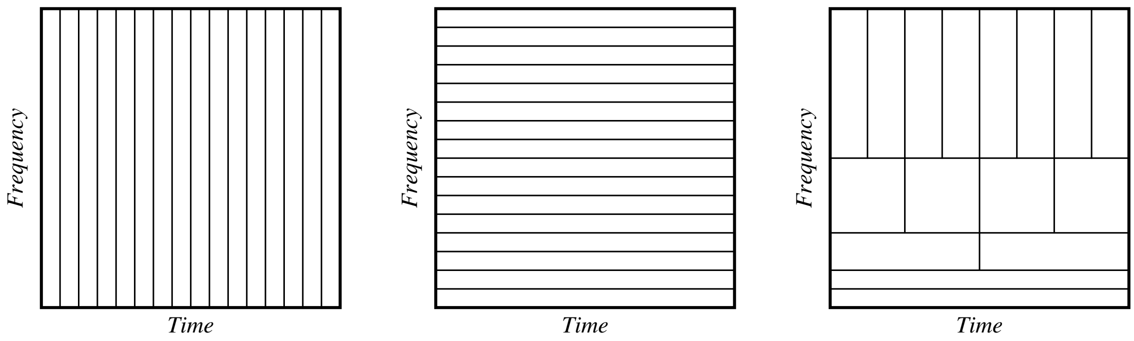
$$\Delta f \Delta t \geq \frac{1}{2}, \tag{1.1}$$

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

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

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

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



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

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

1.2.1 Current-mode filterbanks

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

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

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

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

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

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

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

1.2.3 Analog VLSI implementation of the transient classifier

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

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

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

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

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

Chapter 2

Continuous Wavelet Transform

2.1 Introduction to the 1-Dimensional Continuous Wavelet Transform

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

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

$$\begin{aligned} |\psi(t)| &\leq c\,(1+|t|)^{-1-\epsilon} & (2.2)\\ |\Psi(\omega)| &\leq c\,(1+|\omega|)^{-1-\epsilon} & (2.3) \end{aligned}$$

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

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

2.2 CWT *vs.* DWT

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

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

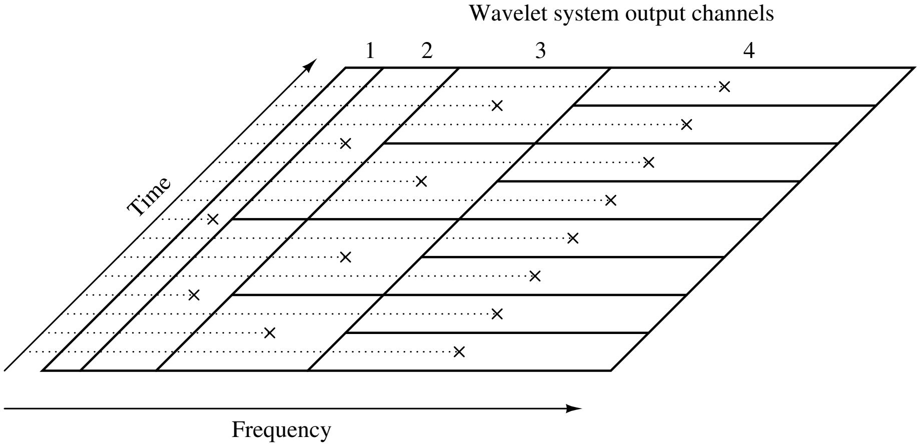


Figure 2.1: Output sampling. Points marked '×' represent center of the time-frequency area covered by that sampled output.

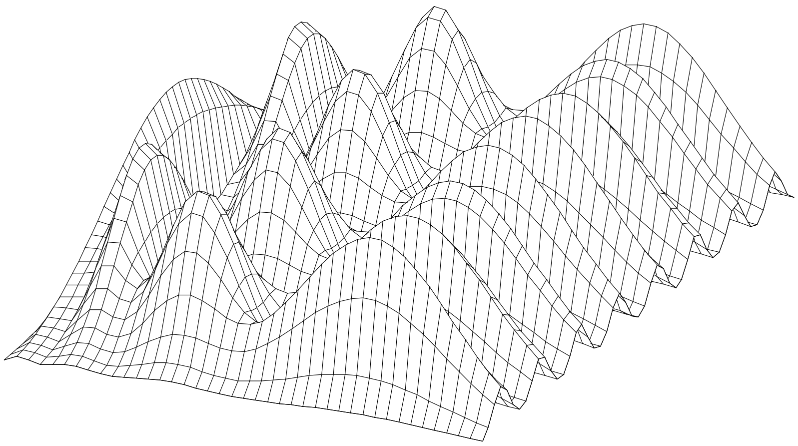


Figure 2.2: Frequency-time representation of the input as overlapping Gaussian filters.

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

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

2.3 Gabor Logons and Wavelets

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

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

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

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

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

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.

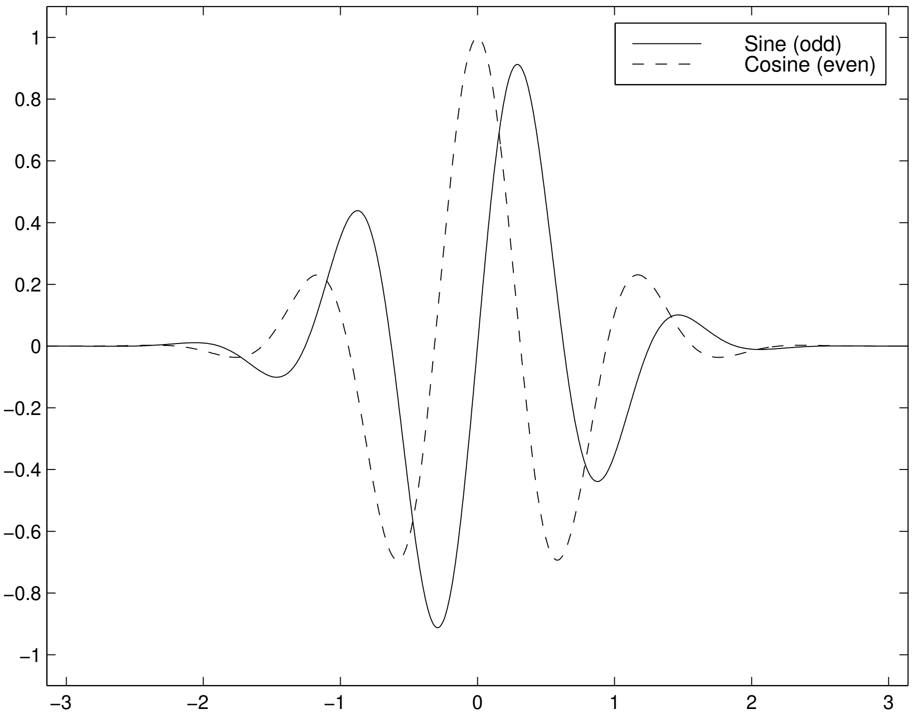


Figure 2.3: Gabor sine and cosine logons, or wavelets.

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.

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

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

2.4 Complex Demodulation

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

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

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
>
> $$\begin{aligned} \cos(u+v) &= \cos(u)\cos(v) - \sin(u)\sin(v) && (2.9)\\ \cos(u-v) &= \cos(u)\cos(v) + \sin(u)\sin(v) && (2.10) \end{aligned}$$
>
> 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).

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

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

Complex Demodulation and Reconstruction of a signal by a carrier

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

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

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

Applying the following trigonometric identities—

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

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

yields the following expressions:

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

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

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

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

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

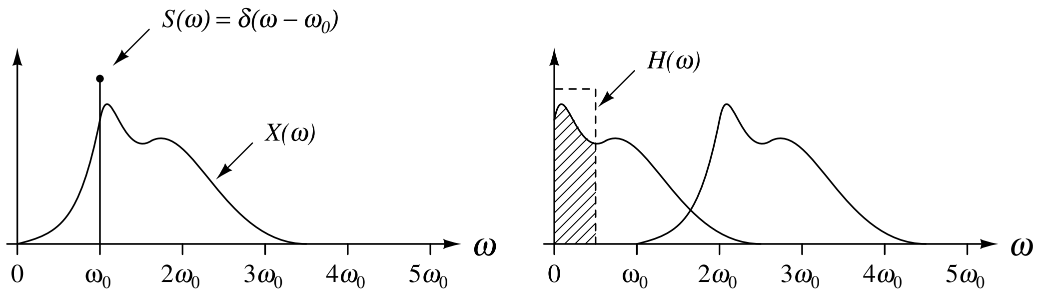


Figure 2.4: Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H.

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):

$$\begin{aligned} x'(t) &= h(t) * (x(t)s_1(t))s_1(t) + h(t) * (x(t)s_2(t))s_2(t) && (2.23) \\ &= \cos(\omega_0 t)\frac{1}{2}\cos((\omega_0 - \omega)t - \phi) \\ &\quad +\sin(\omega_0 t)\frac{1}{2}\sin((\omega_0 - \omega)t - \phi). && (2.24) \end{aligned}$$

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

$$\begin{aligned} x'(t) &= \frac{1}{2}\cos((\omega_0 - \omega_0 + \omega)t + \phi) && (2.25) \\ &= \frac{1}{2}\cos(\omega_0 t + \phi) && (2.26) \\ &= \frac{1}{2}x(t) && (2.27) \end{aligned}$$

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.

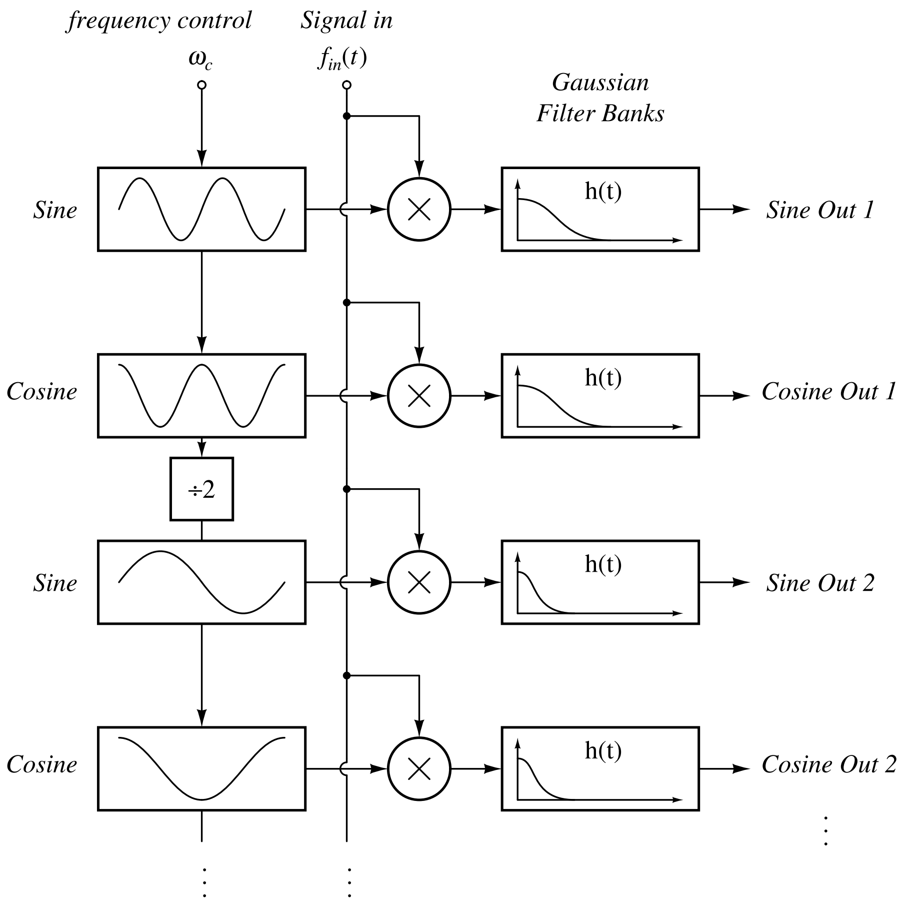


Figure 2.5: Complex demodulation (2 channels shown).

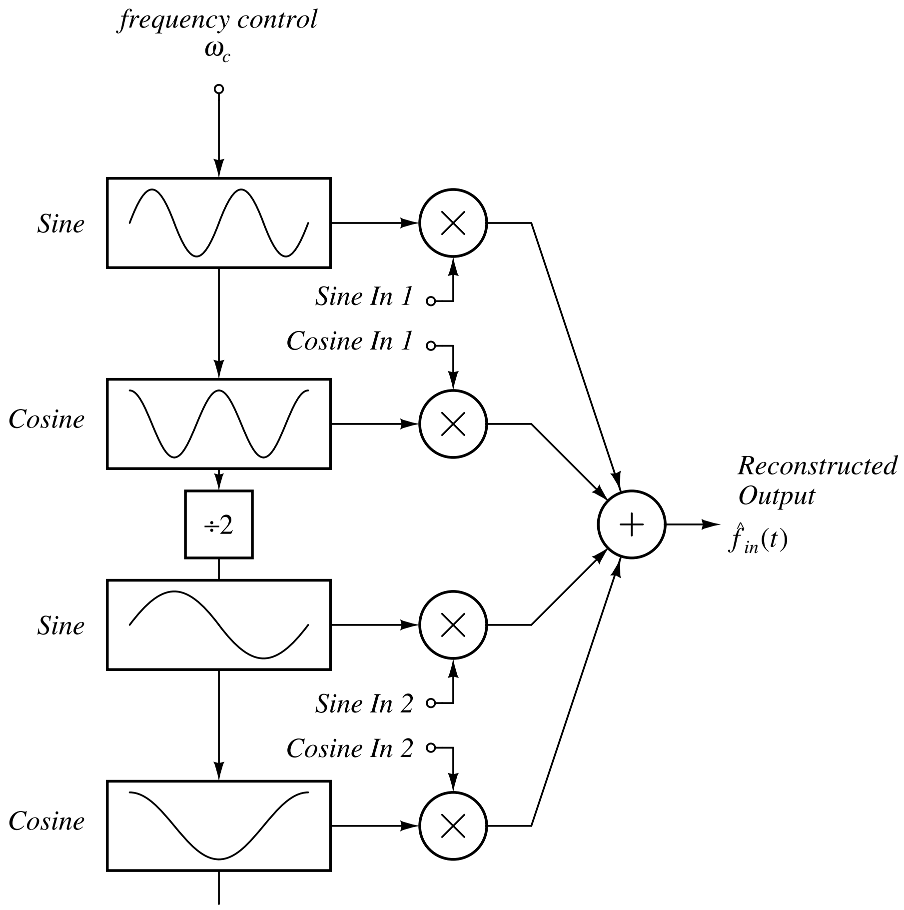


Figure 2.6: Complex modulation reconstruction (2 channels shown).

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

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

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

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

Therefore,

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

and

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

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

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

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

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

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:

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

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

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

2.7 An Analog CWT Processor

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

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

2.8 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.

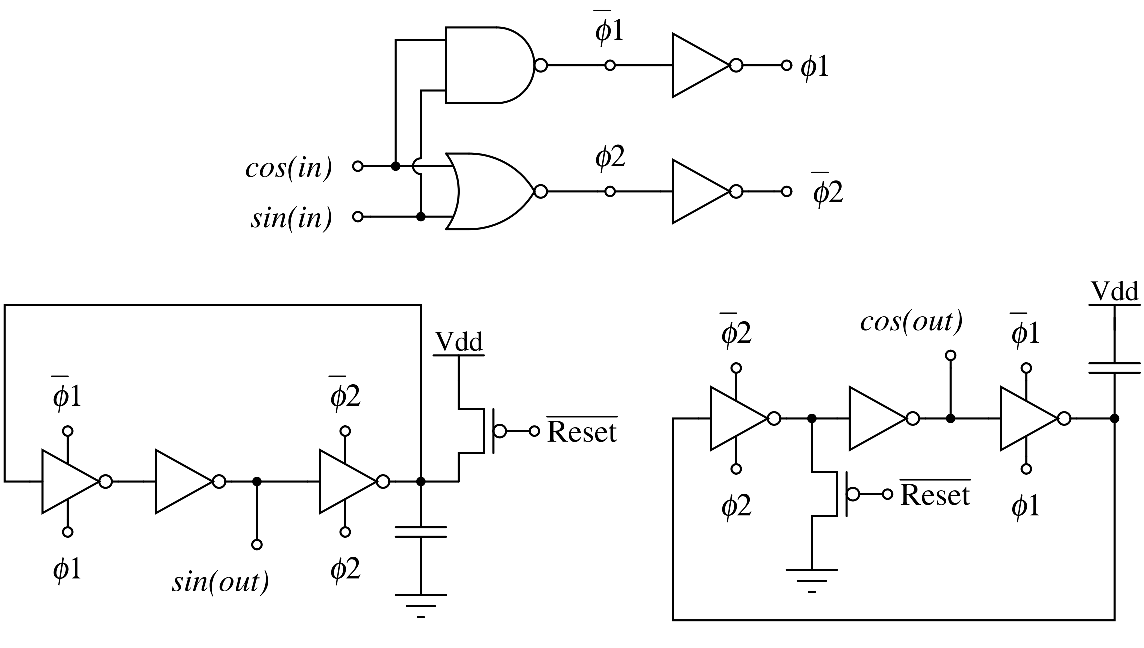


Figure 2.7: Circuit diagram of the frequency-division toggle flip-flops.

2.9 Analog multiplication

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

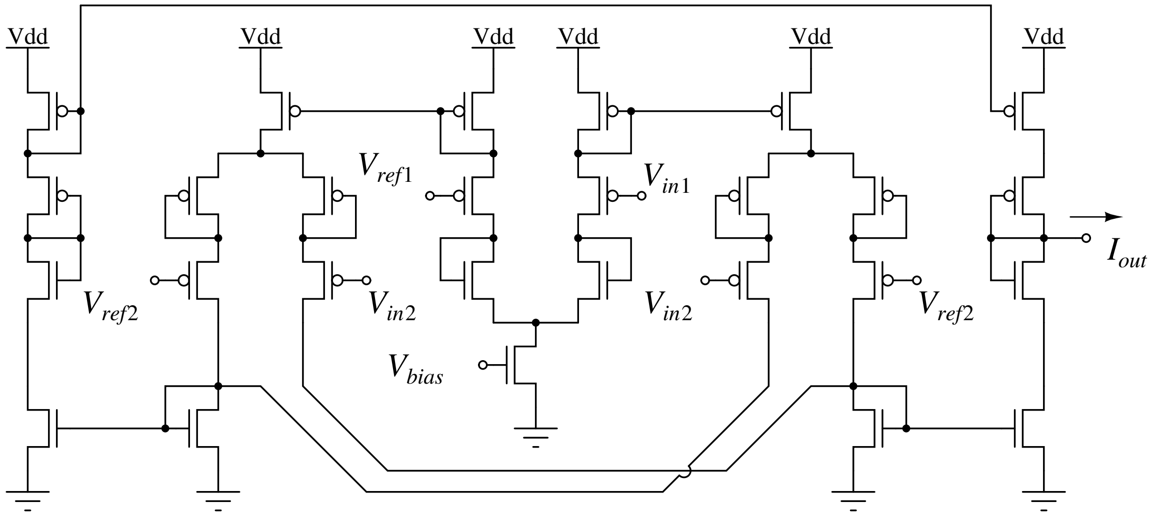


Figure 2.8: Gilbert multiplier with cascodes.

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

2.10 Wavelet Gaussian Function

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

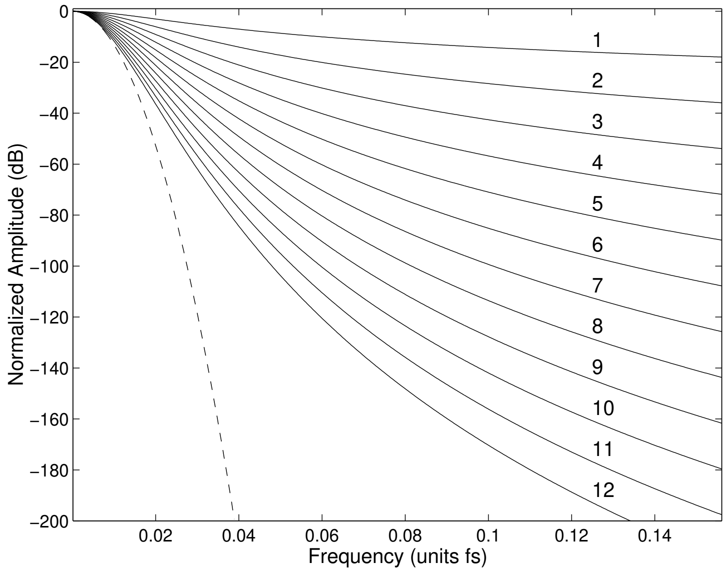


Figure 2.9: Frequency-domain transfer function of the final output of n cascaded lowpass filters as a function of the number of stages n.

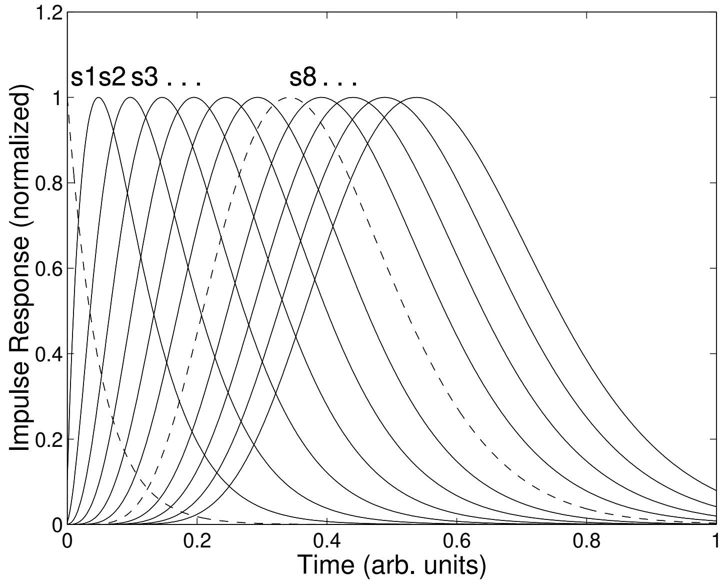


Figure 2.10: Impulse response of the final output of n cascaded lowpass filters as a function of the number of stages n.

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

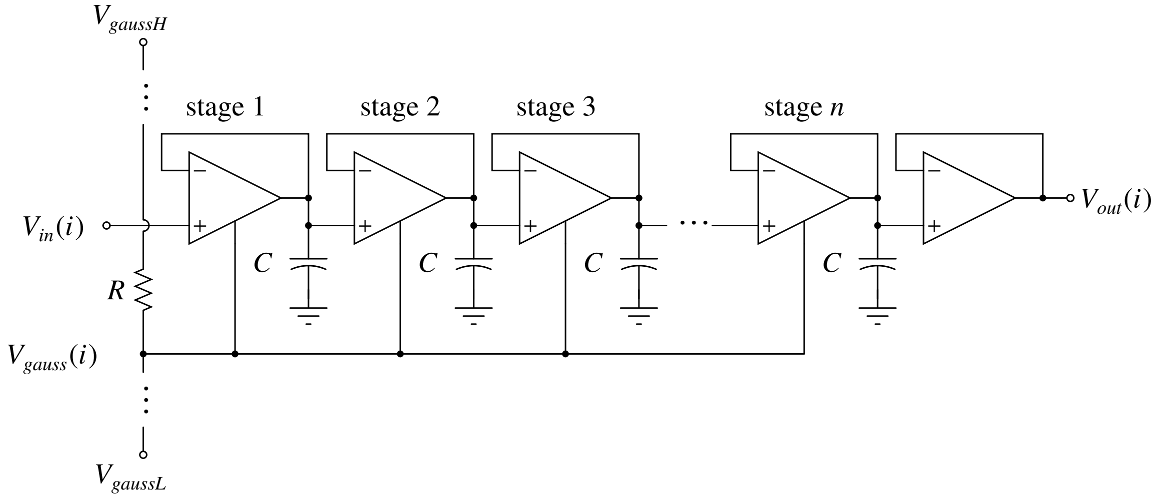


Figure 2.11: n cascaded stages of a filter approximating a half-Gaussian function, using continuous-time transconductance-C filters.

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.

2.11 Wavelet chip slice

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

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

2.12 Chip Specifications

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

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

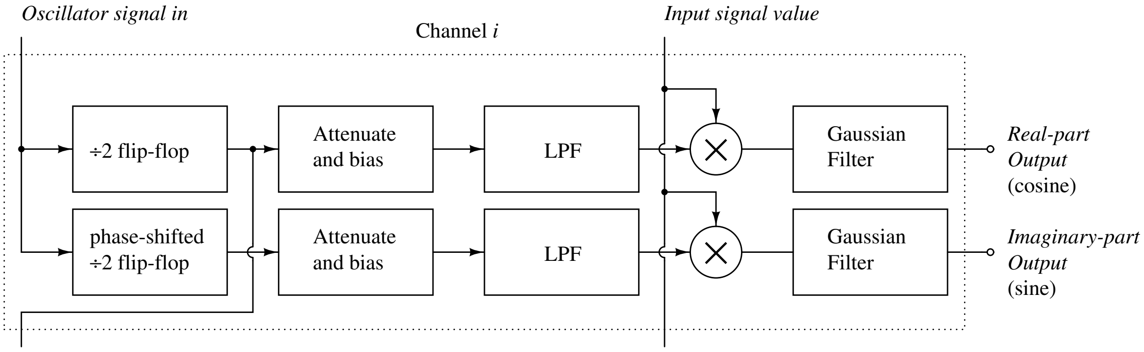


Figure 2.12: Wavelet decomposition block diagram for a single sine/cosine pair.

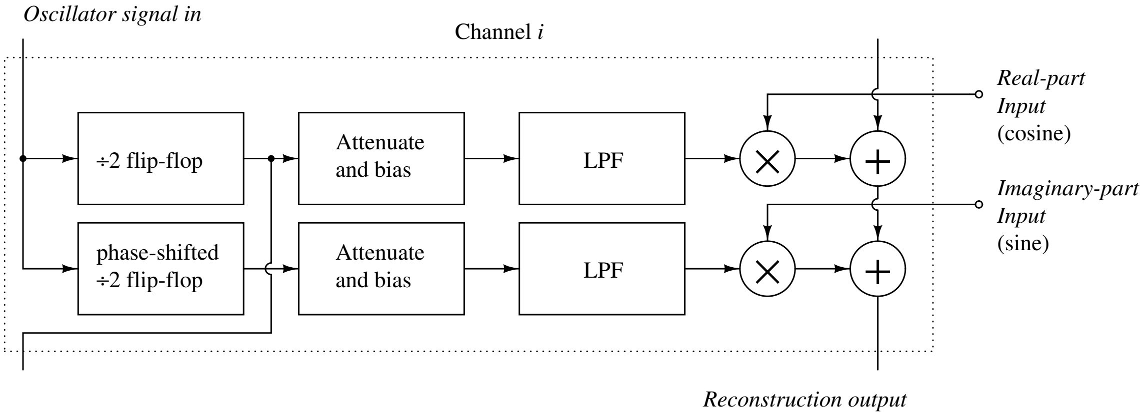


Figure 2.13: Wavelet reconstruction block diagram for a single sine/cosine pair.

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

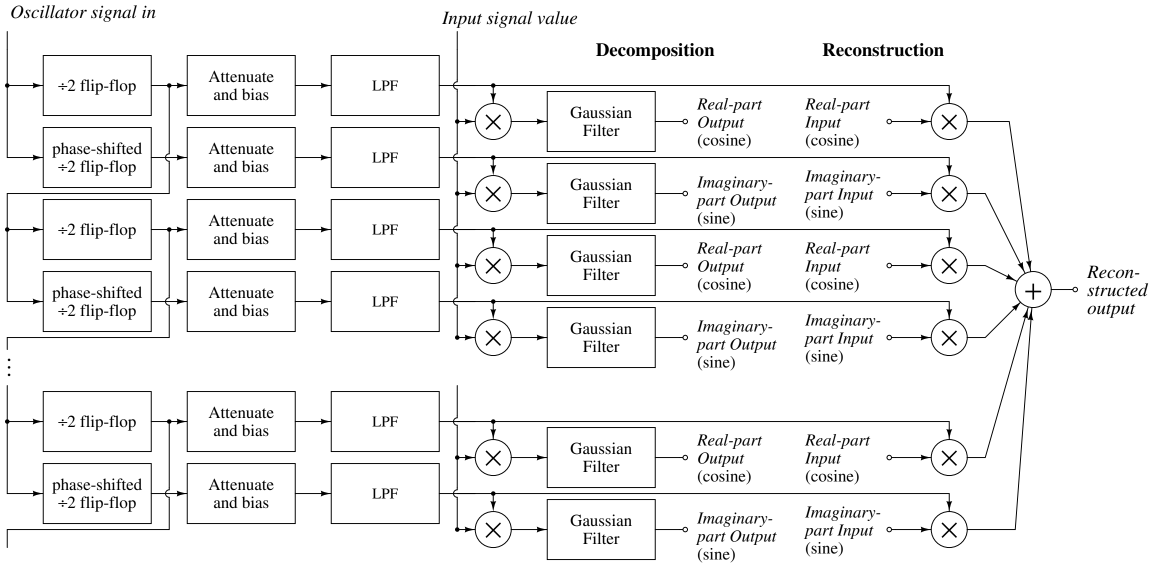


Figure 2.14: Wavelet Transform Chip block diagram.

2.13 Limitations of the Architecture

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

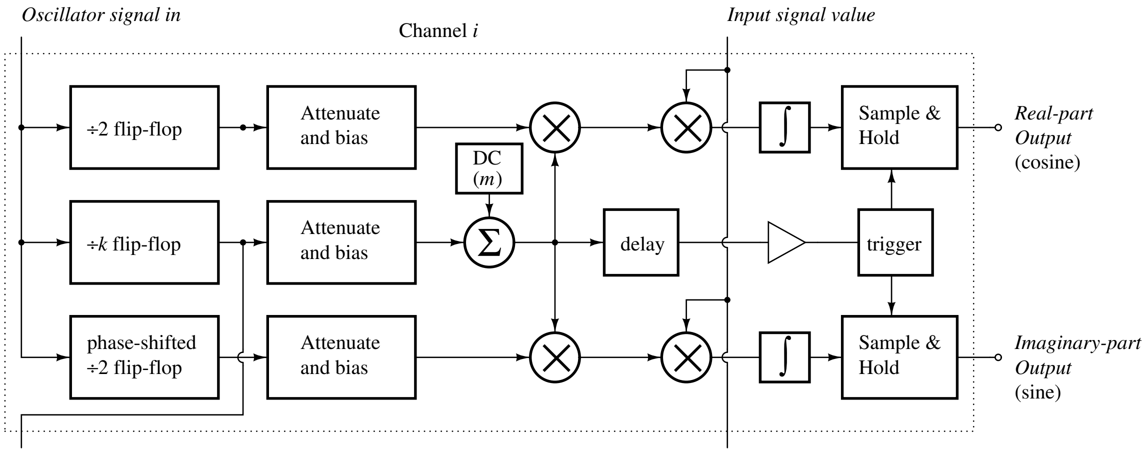


Figure 2.15: Architecture of Moreira-Tamayo *et al.*

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

2.15 A Mixed-Mode Wavelet Processor

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

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

Other than the novel circuit methods used, the new design retains the essential characteristics of the original wavelet architecture. To recap: The processor performs a demodulation of the audio-frequency input signal in parallel across N channels, where the channels are adjacent with minimal overlap, cover the audio frequency band of interest, and are centered on a logarithmic scale. The process of demodulation shifts the signal frequencies from each channel center frequency 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 many different frequency axes. Each produces its own sum and difference components, and some are bound to end up in the signal band at the system output. In order to get predictable system performance, it is necessary to have a sinusoid carrier signal with a fixed and controllable amount of harmonic distortion. To this end we considered the possibility of generating an oversampled version of a sinusoid using a delta-sigma modulator.

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

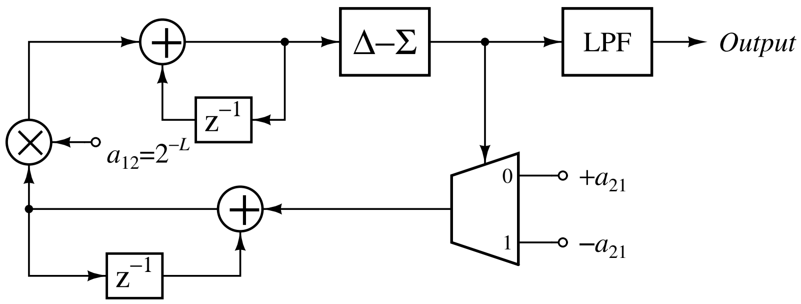


Figure 2.16: The analog oscillator architecture of Lu *et al.* [26] containing a 2nd-order delta-sigma modulator.

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

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

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

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

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

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

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

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

2.16 Details of the Bit-Sequence-Finding Algorithm

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

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

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

$$
\begin{aligned}
S'(\omega) &= \sum_{k=0}^{\infty} S'_k(\omega) \ , && (2.40)\\
S'_k(\omega) &= jc_k \left[\delta\left(\omega - (2k+1)\omega_0\right) - \delta\left(\omega + (2k+1)\omega_0\right)\right] . && (2.41)
\end{aligned}
$$

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

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

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

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

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

2.17 Sequence generation

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

$$
jc_k \, G\left(\omega \pm (2k+1)\omega_0\right) \cdot X\left(\omega \pm (2k+1)\omega_0\right) \cdot H(\omega). \qquad (2.43)
$$

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

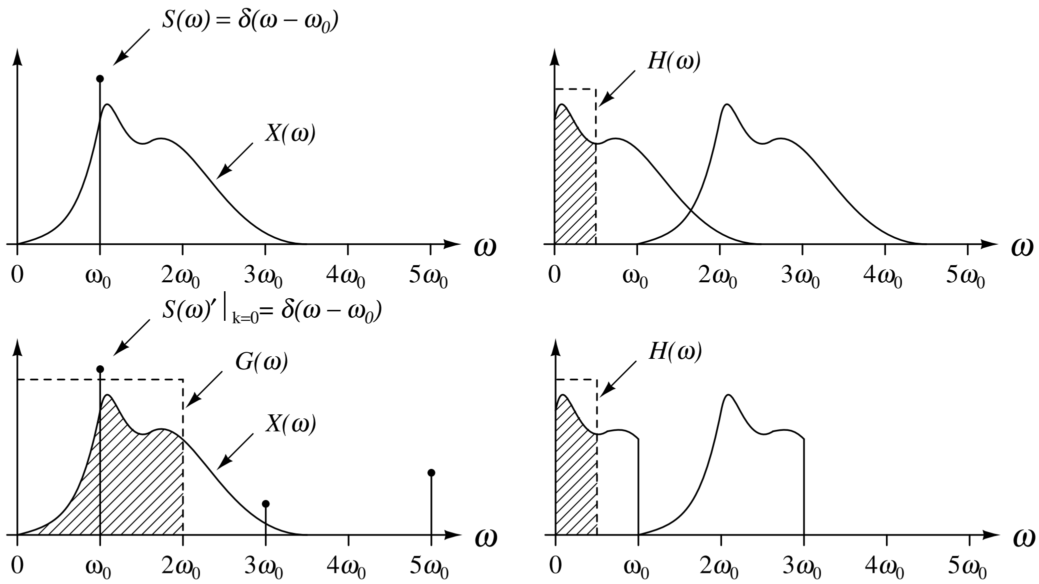


Figure 2.17: *Top*: Demodulation of an input X in the frequency domain with a perfect sine wave S and ideal modulation filter H. *Bottom*: Demodulation of an input X in the frequency domain with an oversampled sine wave S' using an ideal smoothing filter G and modulation filter H.

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:

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

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

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

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

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

2.18 Results and implementation

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

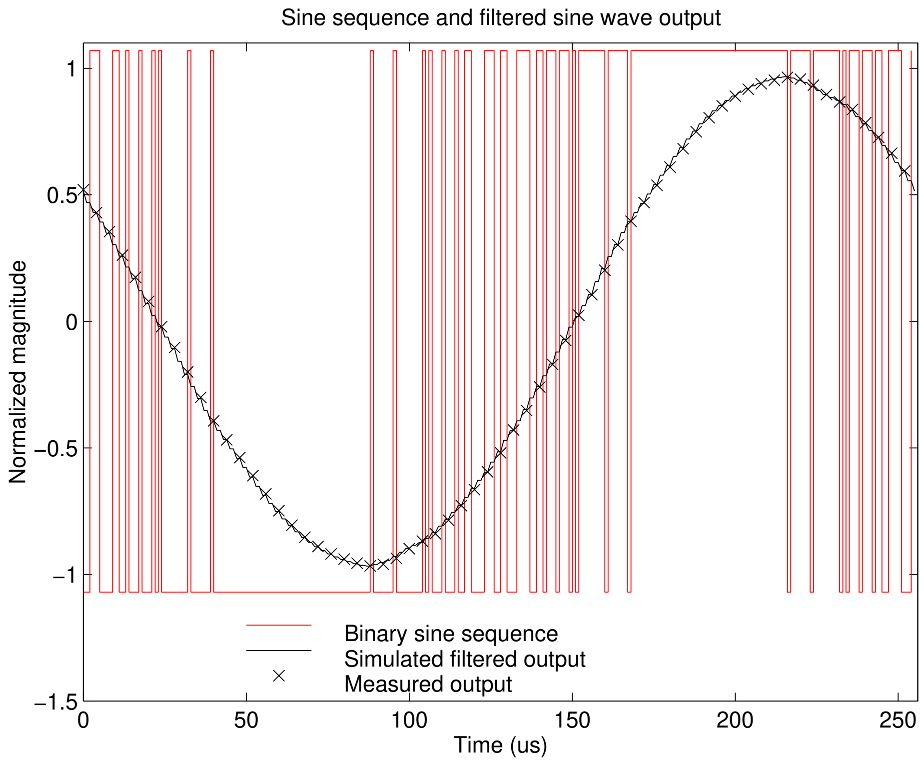


Figure 2.18: The oversampled sine sequence.

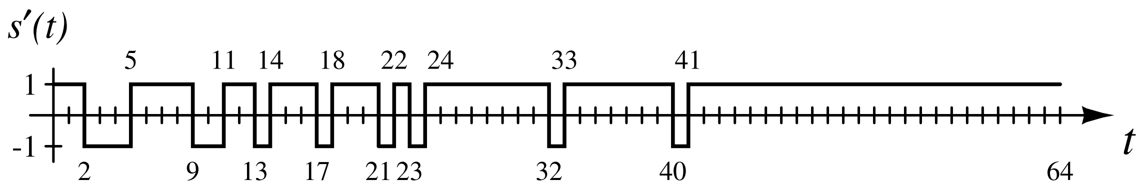


Figure 2.19: Optimized 64-bit oversampled sine sequence, first quadrant.

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-

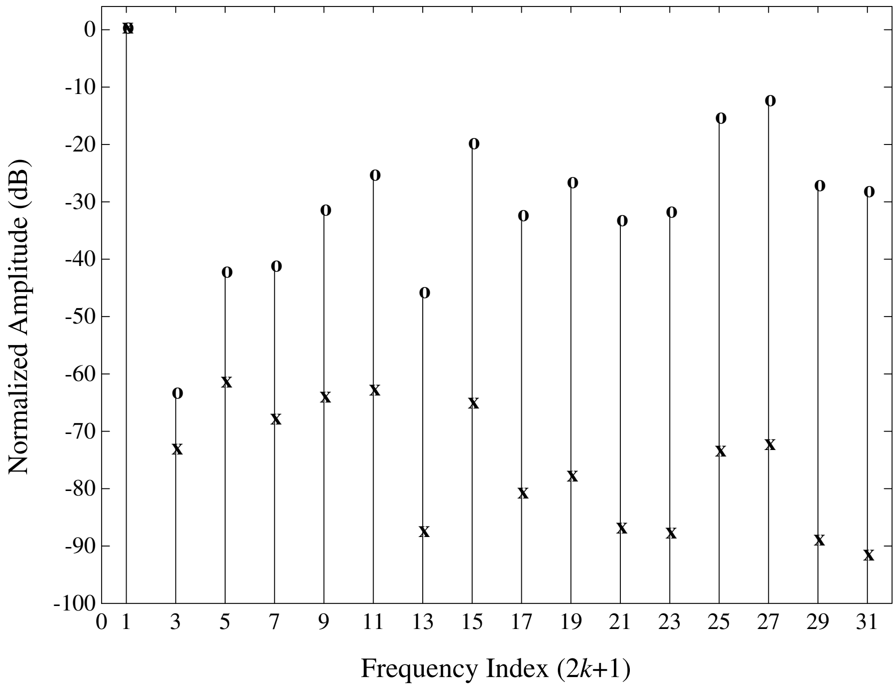


Figure 2.20: Frequency domain properties of the raw (**o**) and filtered (**x**) bit sequences.

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

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

0 0 0 0 0	
0 0 0 0 1	
0 0 0 1 1	
0 0 0 1 0	
0 0 1 1 0	quadrant 1
0 0 1 1 1	
0 0 1 0 1	
0 0 1 0 0	
0 1 1 0 0	
0 1 1 0 1	
0 1 1 1 1	
0 1 1 1 0	quadrant 2
0 1 0 1 0	
0 1 0 1 1	
0 1 0 0 1	
0 1 0 0 0	
1 1 0 0 0	
1 1 0 0 1	
1 1 0 1 1	
1 1 0 1 0	quadrant 3
1 1 1 1 0	
1 1 1 1 1	
1 1 1 0 1	
1 1 1 0 0	
1 0 1 0 0	
1 0 1 0 1	
1 0 1 1 1	
1 0 1 1 0	
1 0 0 1 0	quadrant 4
1 0 0 1 1	
1 0 0 0 1	
1 0 0 0 0	

sign bit for cosine
sign bit for sine

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

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

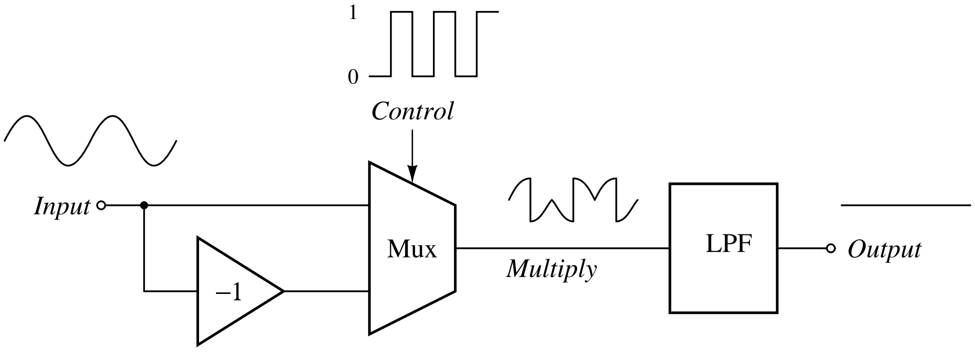


Figure 2.22: Multiplexing *vs.* Multiplying.

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

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

2.20 Switch-Cap Wavelet Gaussian Function

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

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

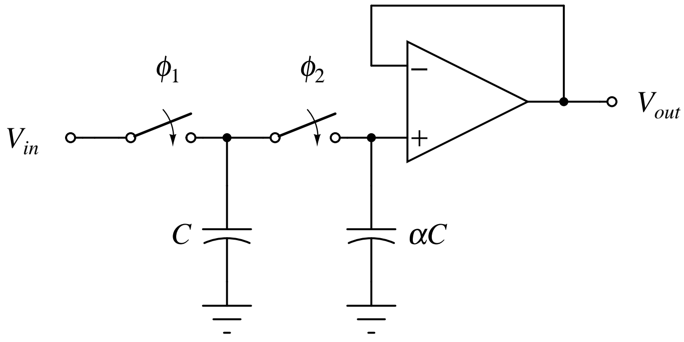


Figure 2.23: Single discrete-time lowpass filter section.

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

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

2.21 Output Time Multiplexing

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

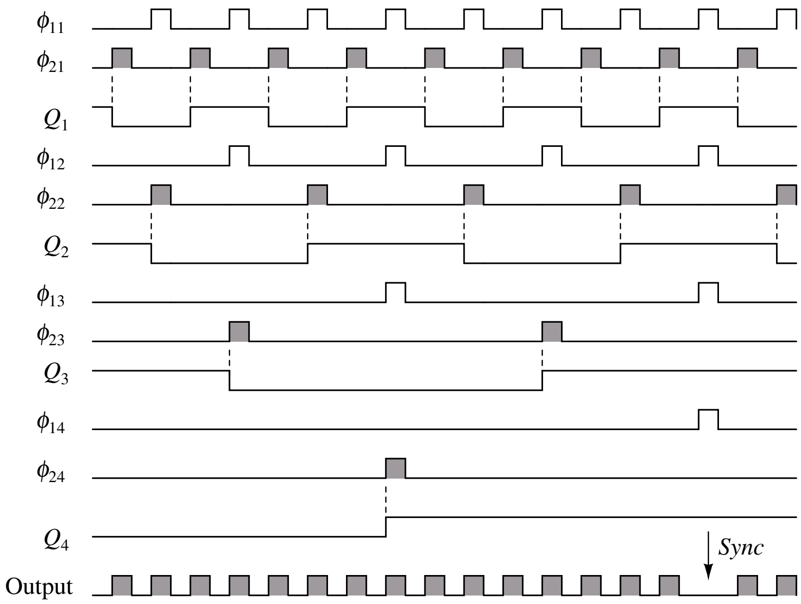


Figure 2.24: A scheme for controlling time-multiplexing of the outputs.

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

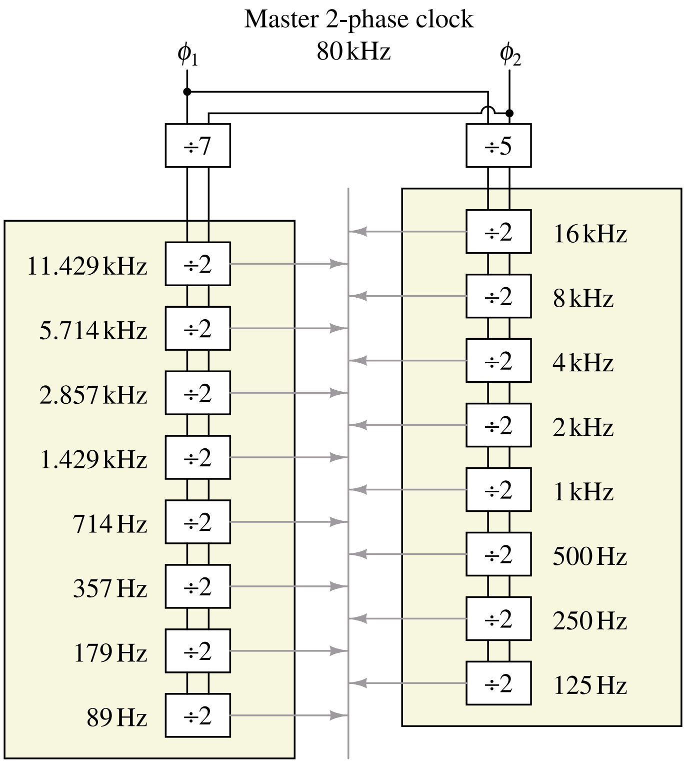


Figure 2.25: Sixteen-channel architecture using the 7-to-5 frequency ratio.

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

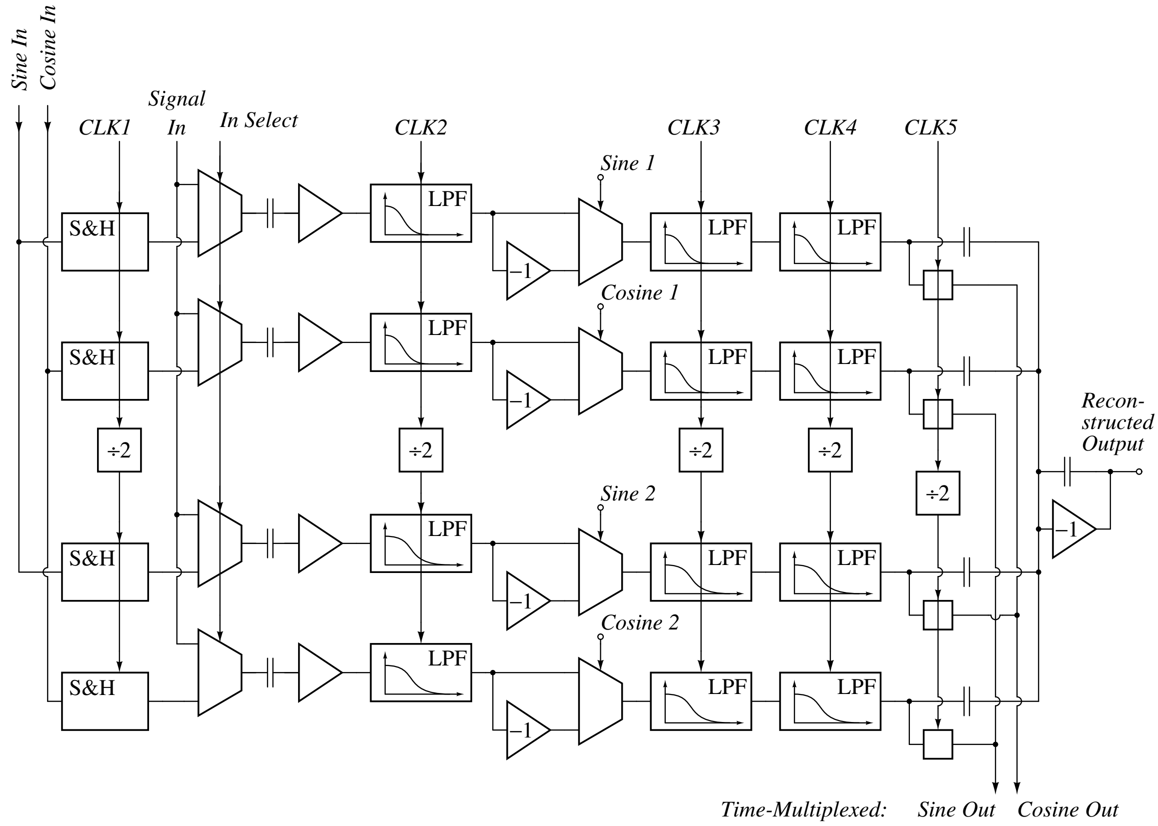


Figure 2.26: The wavelet chip, block diagram.

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**")

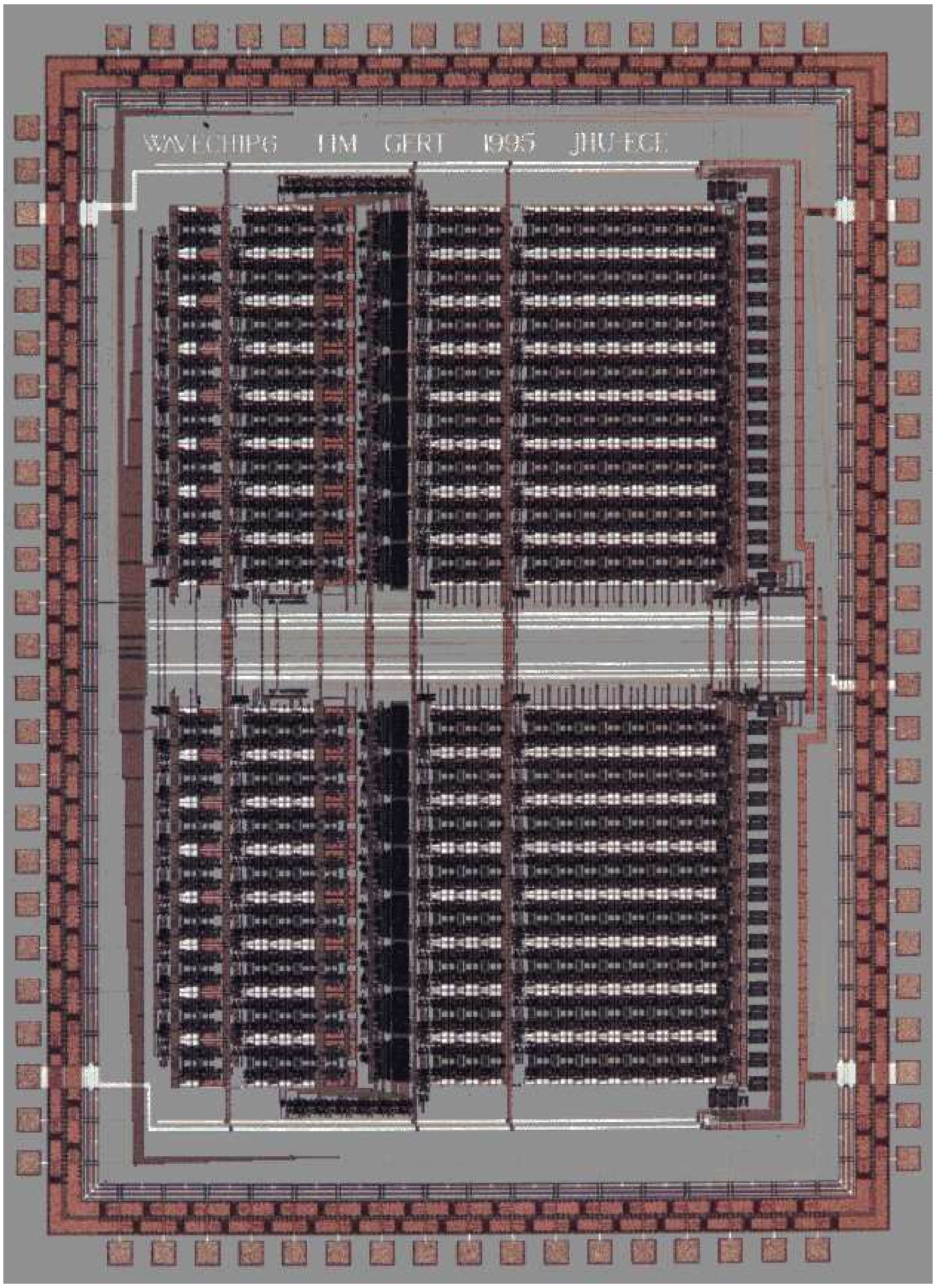


Figure 2.27: Photomicrograph of the mixed-mode continuous wavelet transform processor, a 4 mm×6 mm die size fabricated in a 2 μm CMOS p-well process.

lowpass filtering. Distortion components of the binary sequence are attenuated by the filter to below $-60\,\text{dB}$.

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.

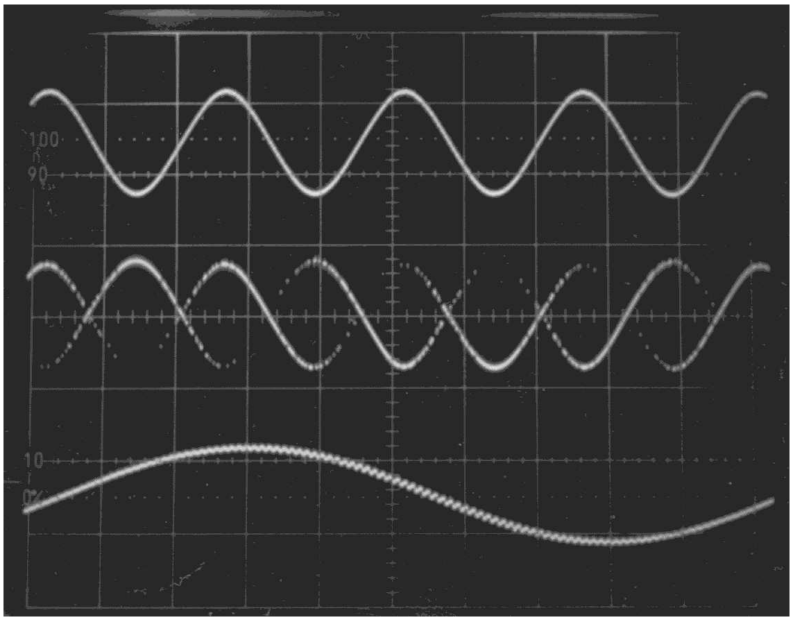


Figure 2.28: Signal demodulation using the wavelet chip. The top trace is the input signal. The middle trace is the input multiplied by the binary sequence. The bottom trace is the output after filtering.

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.

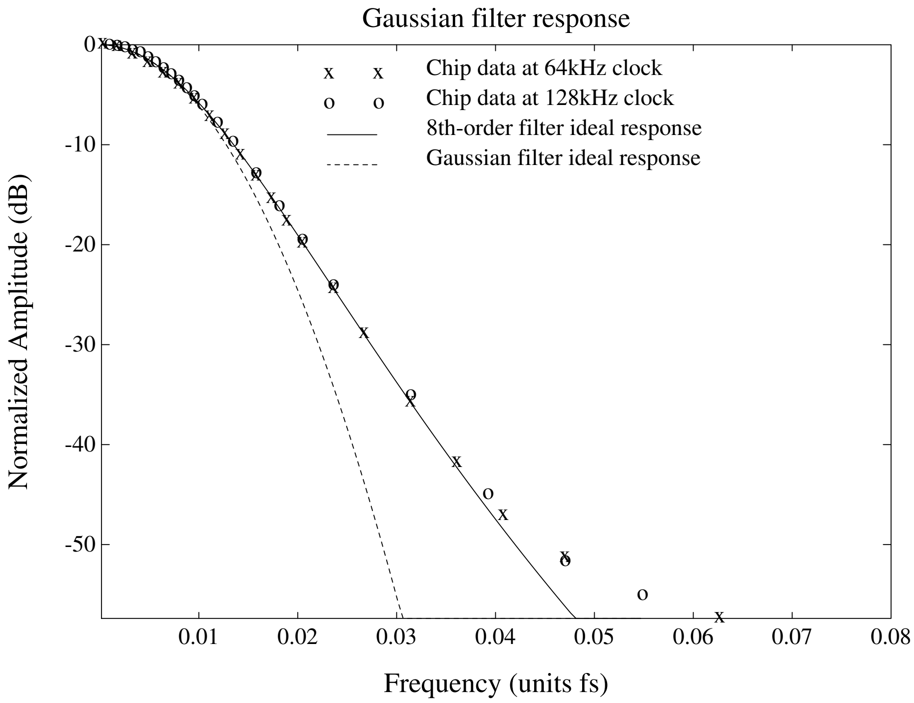


Figure 2.29: Gaussian filter magnitude response: predicted and measured.

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

2.24 Extensions of the Research

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

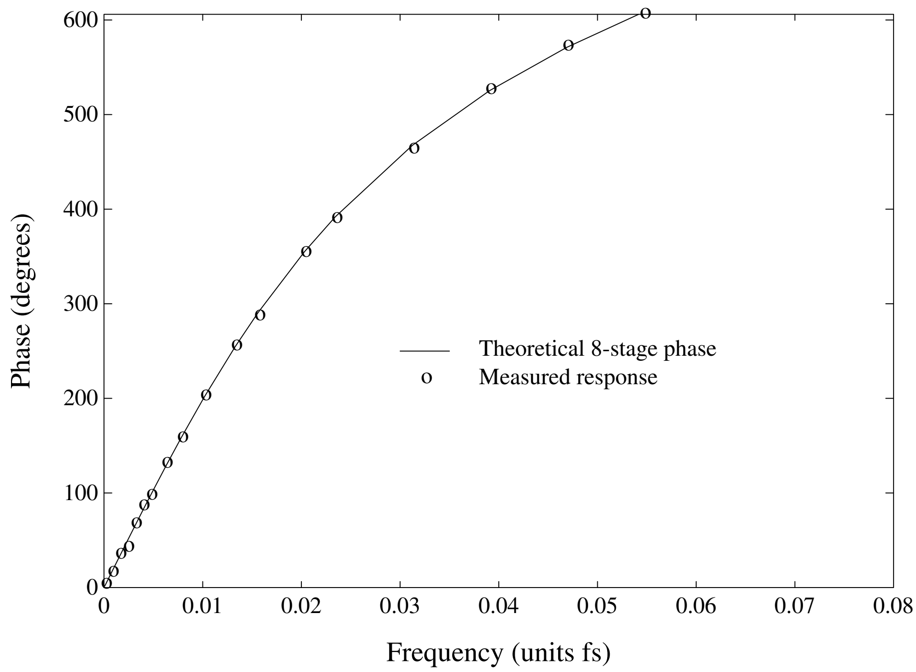


Figure 2.30: Gaussian filter phase response: predicted and measured.

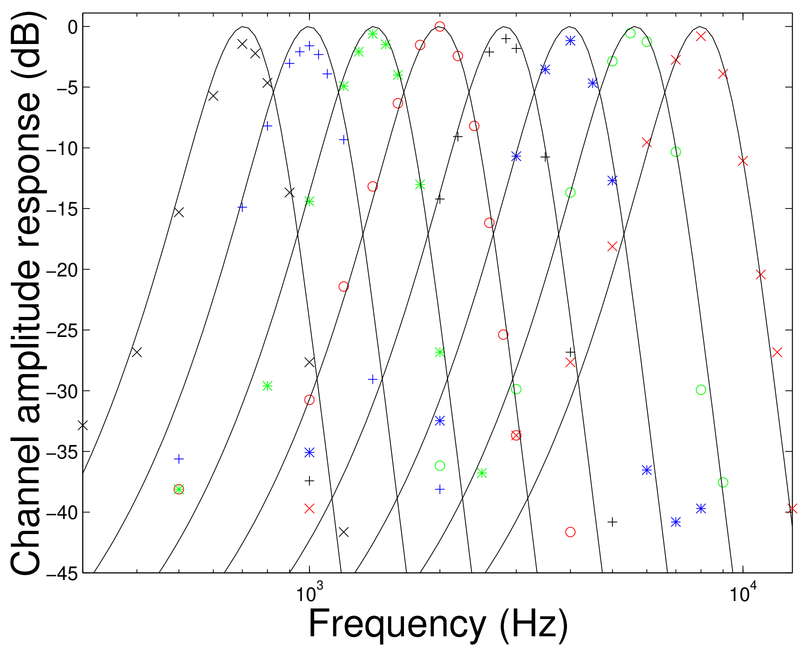


Figure 2.31: Wavelet responses to isolated sine wave input.

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

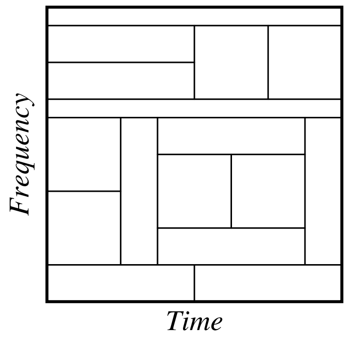


Figure 2.32: An arbitrary tiling of the time-frequency plane.

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

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.

Chapter 3

Current-Mode Filterbank Frontend

3.1 Time-Frequency Representations using Filterbanks

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

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

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

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

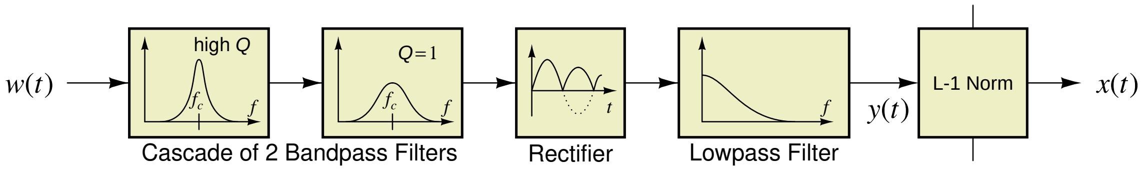


Figure 3.1: Frontend filterbank system—block diagram.

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

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

3.3 Current-Mode Filters for Current-Mode Applications

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

3.4 High-Level Simulations of the Filterbank Frontend

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

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

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

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

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

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

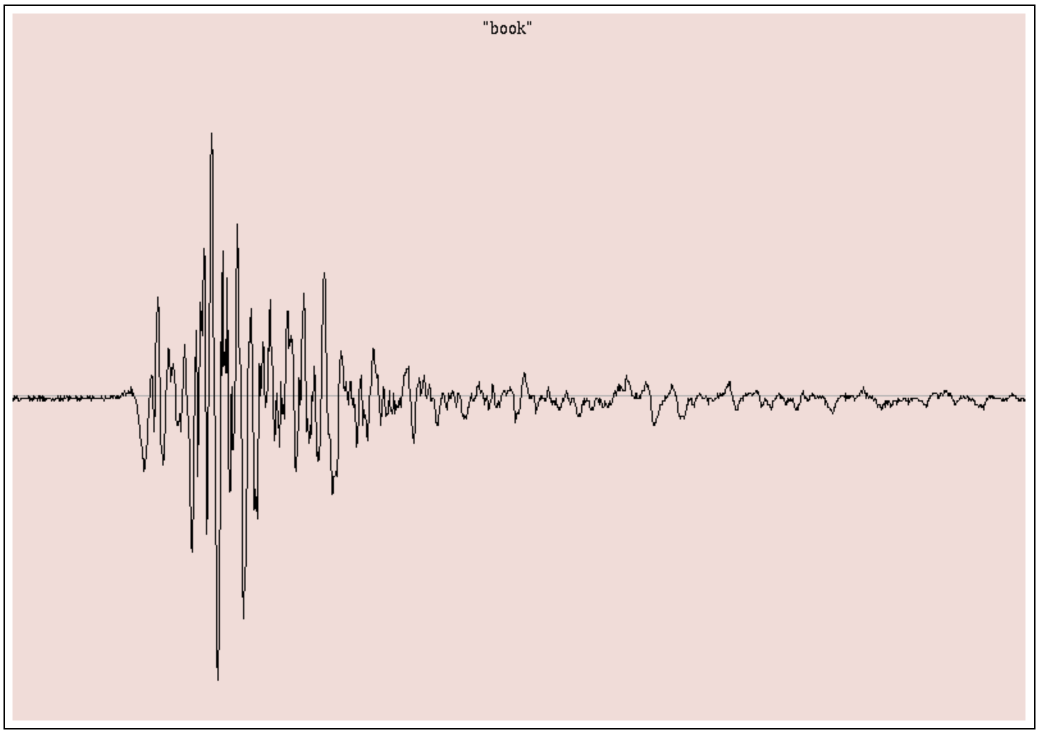


Figure 3.2: Sampled-data input from a recording of an acoustic transient—that of a book being dropped onto a desk.

filterbank.

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

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

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

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

where

$$\begin{aligned} b_{0,n} &= \frac{\tau_n}{1 + (\tau_n/Q) + \tau_n^2} && (3.4)\\ b_{2,n} &= -b_{0,n} && (3.5)\\ a_{1,n} &= \frac{2(\tau_n^2 - 1)}{1 + (\tau_n/Q) + \tau_n^2} && (3.6)\\ a_{2,n} &= \frac{-(1 - (\tau_n/Q) + \tau_n^2)}{1 + (\tau_n/Q) + \tau_n^2} && (3.7) \end{aligned}$$

and

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

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

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

$$\begin{aligned} f_{c,n} &= \gamma f_{c,n-1} \qquad n = 2 \ldots N && (3.9)\\ f_{c,1} &= f_1 && (3.10) \end{aligned}$$

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

$$\begin{aligned} f_{c,n} &= \gamma_0 f_{c,n-1} + \gamma_1 \qquad n = 2 \ldots N && (3.11)\\ f_{c,1} &= f_1 && (3.12) \end{aligned}$$

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

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.

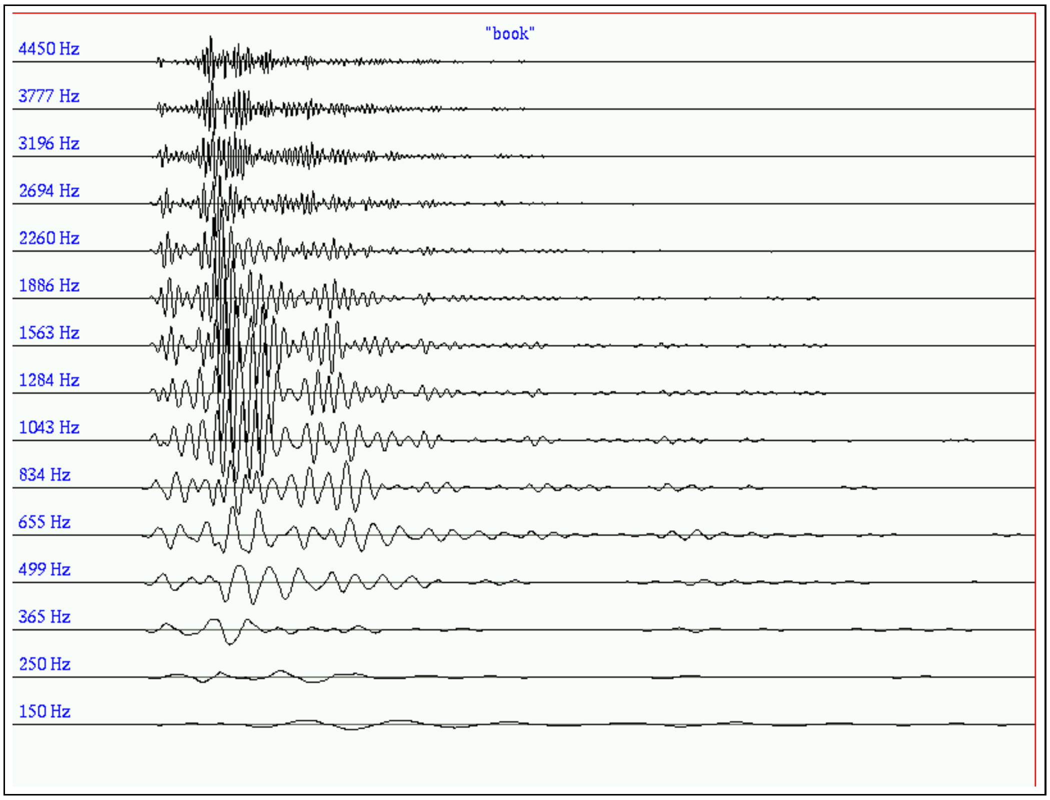


Figure 3.3: Bandpass-filtered acoustic transient input using two cascaded second-order filters on each channel.

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

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

for which application of the bilinear transform yields

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

having coefficients

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

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

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

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

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

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

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

3.5 Introduction to Translinear Circuits and Log-domain Filtering

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

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

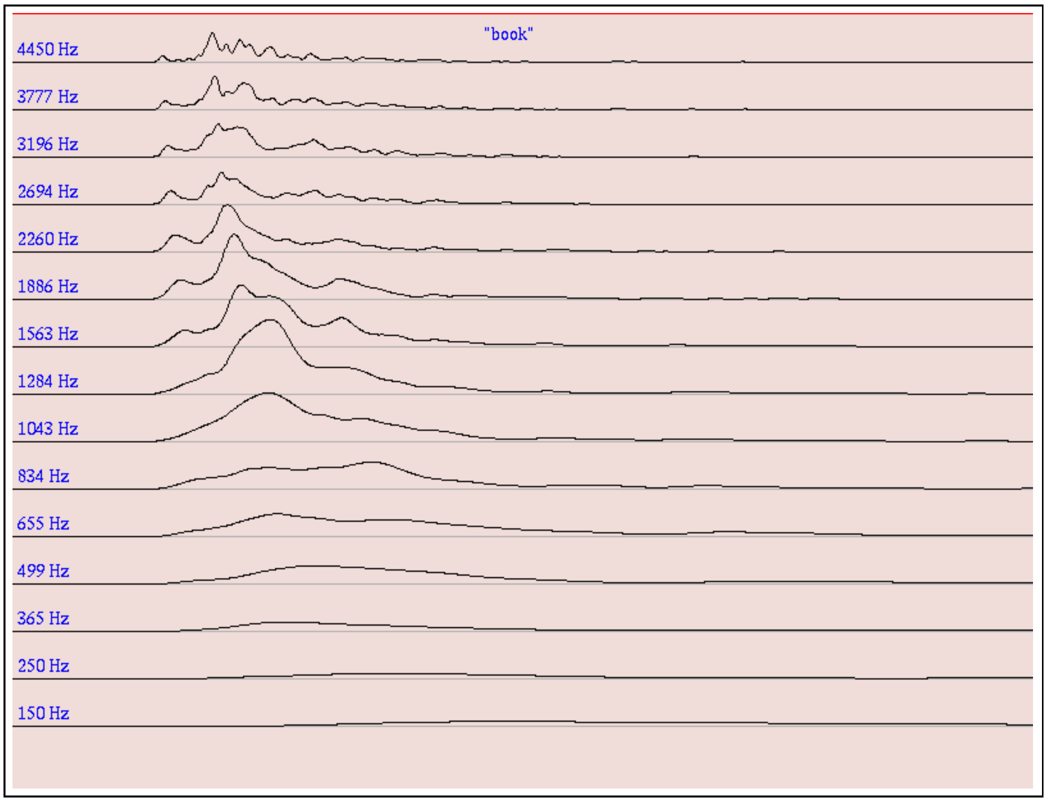


Figure 3.4: Parallel filterbank output after rectification and smoothing of the acoustic transient input signal across all frequency bands.

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

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

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

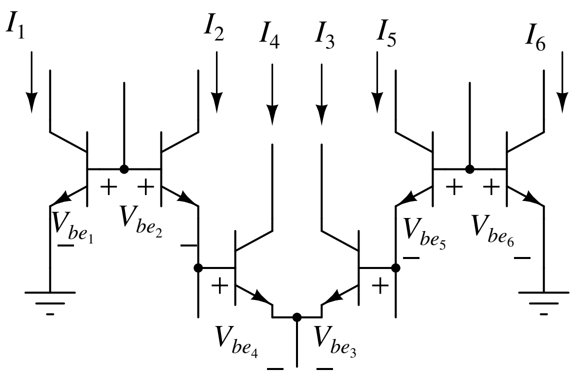


Figure 3.5: Translinear loop with common-base and common-emitter configurations.

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

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

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

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

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.

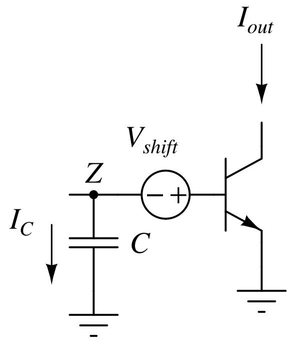


Figure 3.6: Filter pole formed using a transconductor.

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

3.7 First-Order Circuit synthesis

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

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

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

$$\begin{aligned} AI_{out} + \left(\frac{\tau\, I_C}{V_t\, C}\right) I_{out} &= I_{in}, \qquad (3.29) \\ I_{out}\left(A + \frac{\tau\, I_C}{V_t\, C}\right) &= I_{in}. \qquad (3.30) \end{aligned}$$

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

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

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

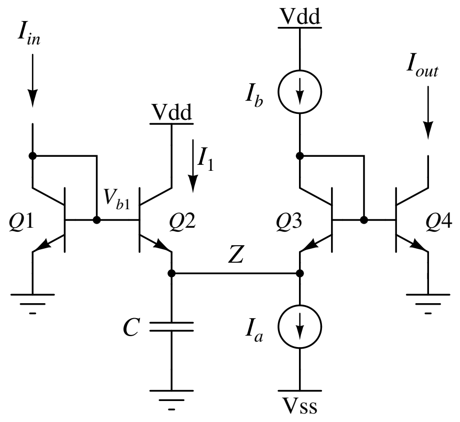


Figure 3.7: First-order log-domain filter circuit.

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:

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

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

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

3.8 Designing second-order sections

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

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

This describes a bandpass function with a gain and resonance of Q, except that the output is biased to a positive DC operating point. Since the bandpass function itself eliminates DC components 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, however: The negative feedback term can be separated from the positive term, and with the proper transformation, it can be referred to the capacitor node as a dependent current source. This solution is shown in Figure 3.8B. To find the equation for this dependent current source, the translinear loop

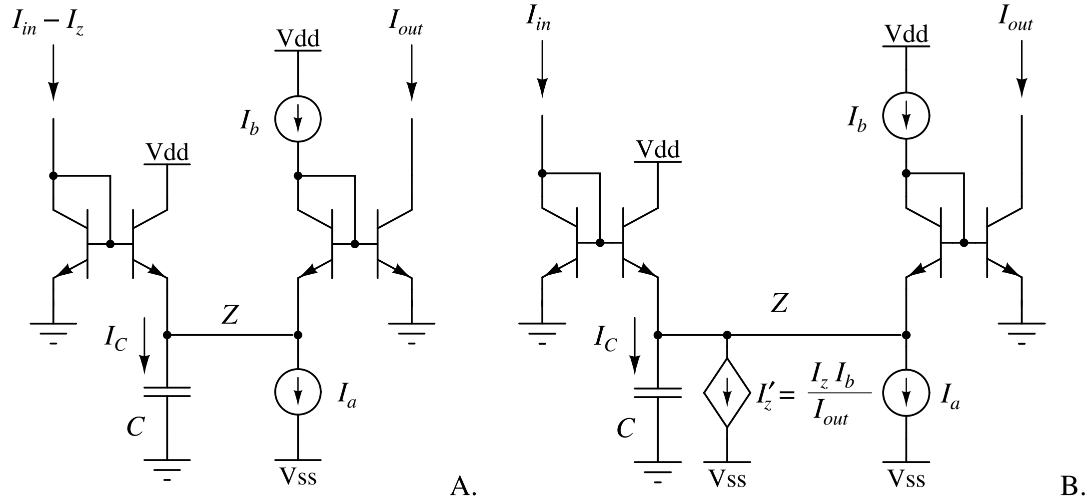


Figure 3.8: Computing a current difference at a log-domain filter input. A) The underlying idea, which is physically unrealizable. B) An equivalent working implementation.

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

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

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

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

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

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

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

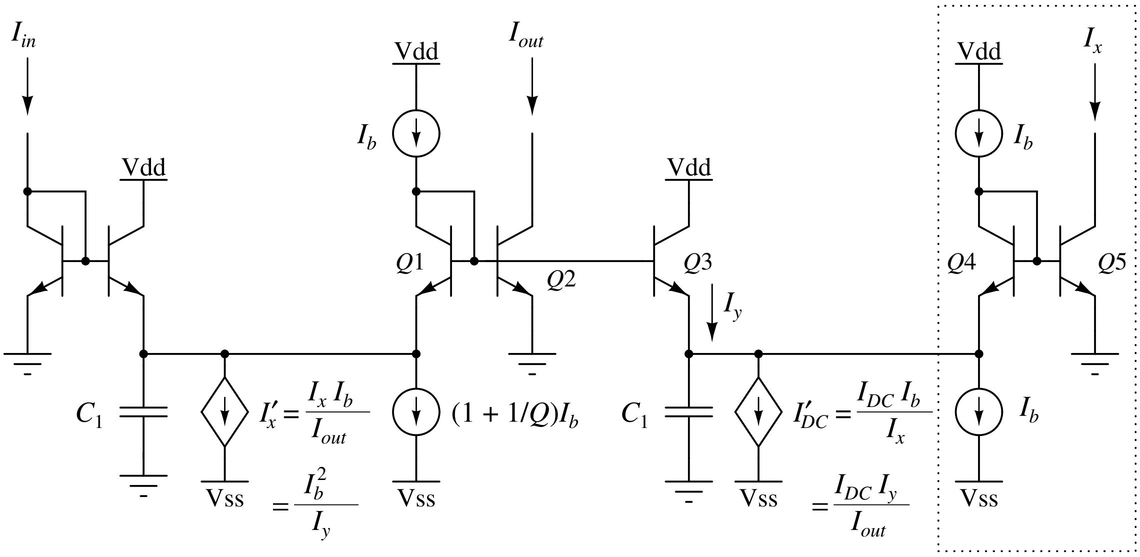


Figure 3.9: Bandpass structure formed from first-order sections.

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

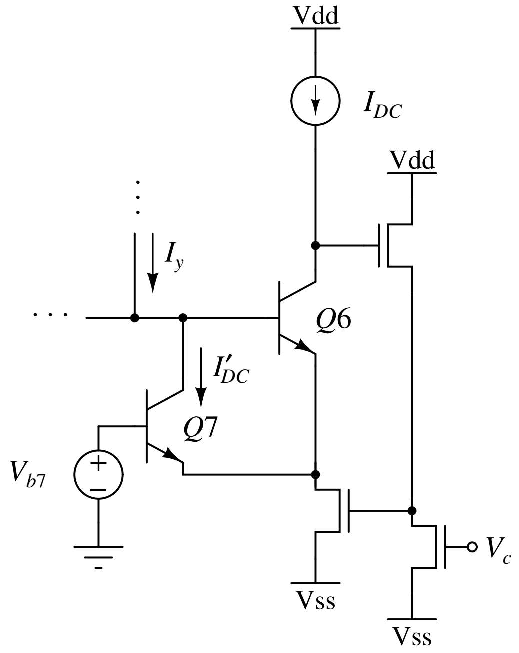


Figure 3.10: An alternative common-emitter circuit generating I'_{DC} (see text for discussion).

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

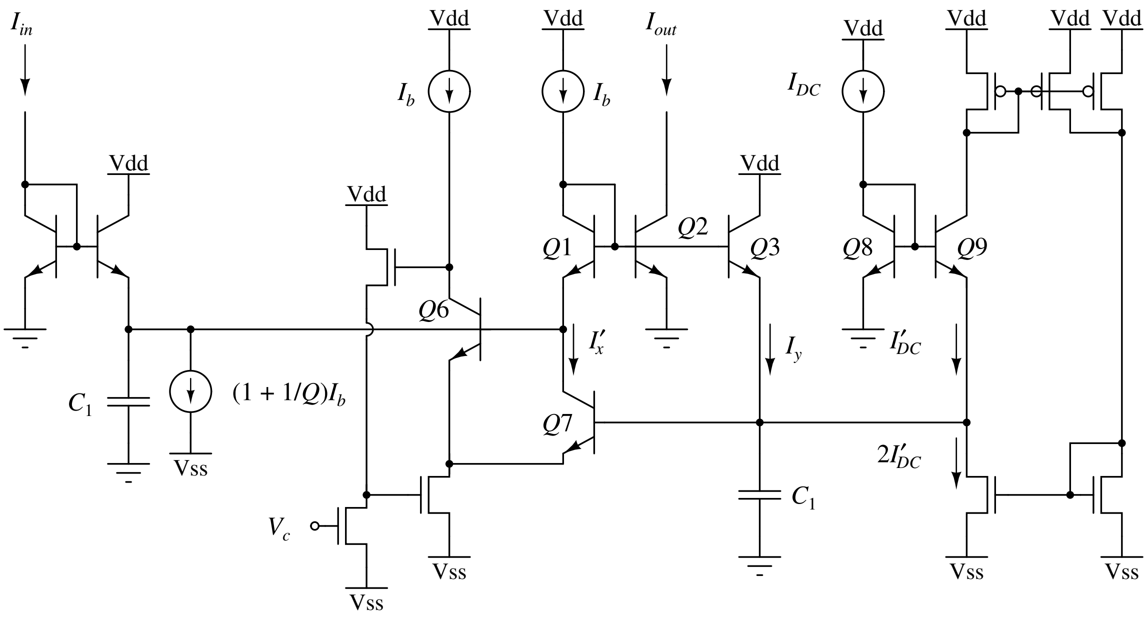


Figure 3.11: Complete circuit schematic for the second-order bandpass filter.

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

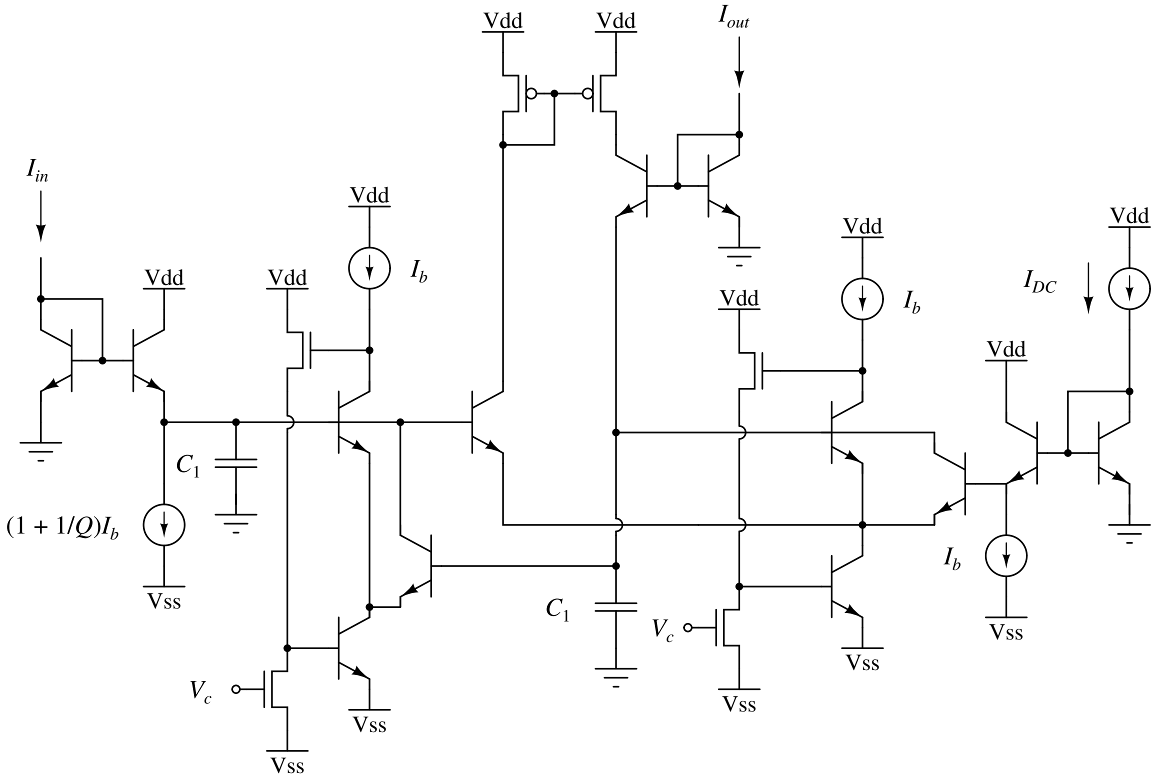


Figure 3.12: Another BiCMOS Log-domain bandpass filter, using the common-emitter configuration for all feedback circuits.

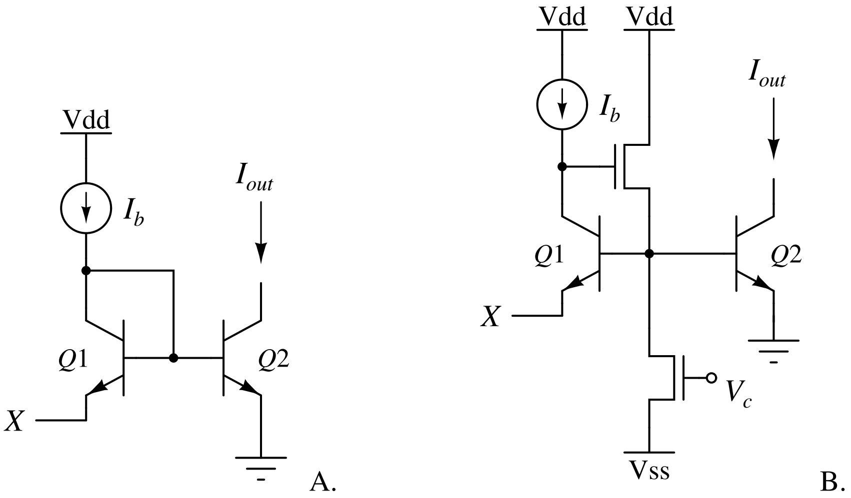


Figure 3.13: Base compensation (B) eliminates undesirable behavior due to significant base current which occurs in (A).

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

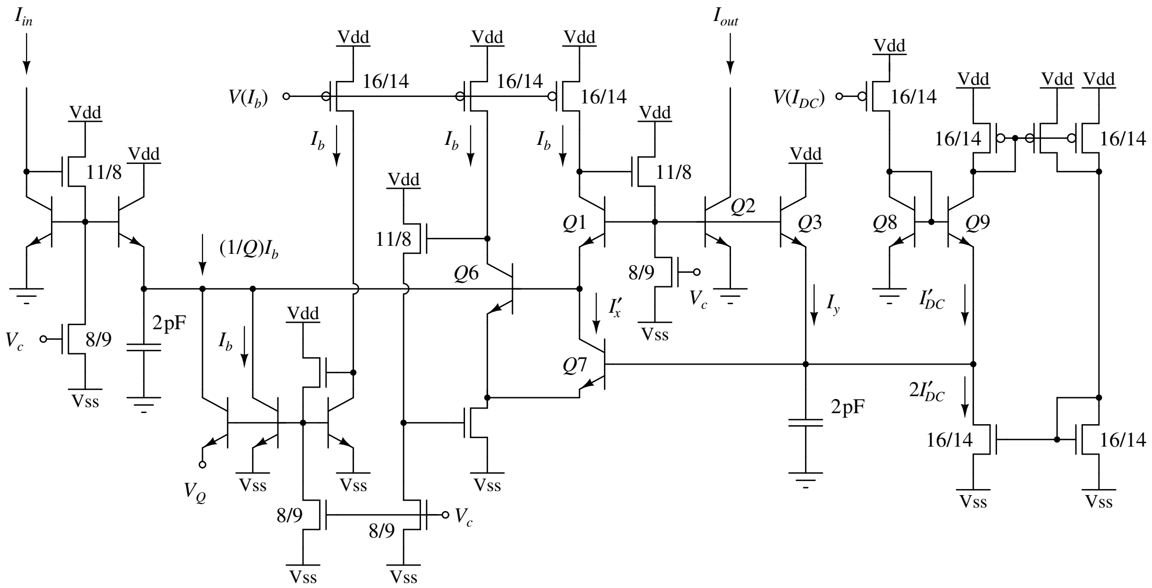


Figure 3.14: Complete circuit schematic for the second-order bandpass filter. Bipolar transistors are minimum size, and MOS dimensions are indicated as W/L in units of $\lambda = 0.6\,\mu$m.

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

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

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

3.10 Layout Considerations for VLSI Log-Domain Circuits

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

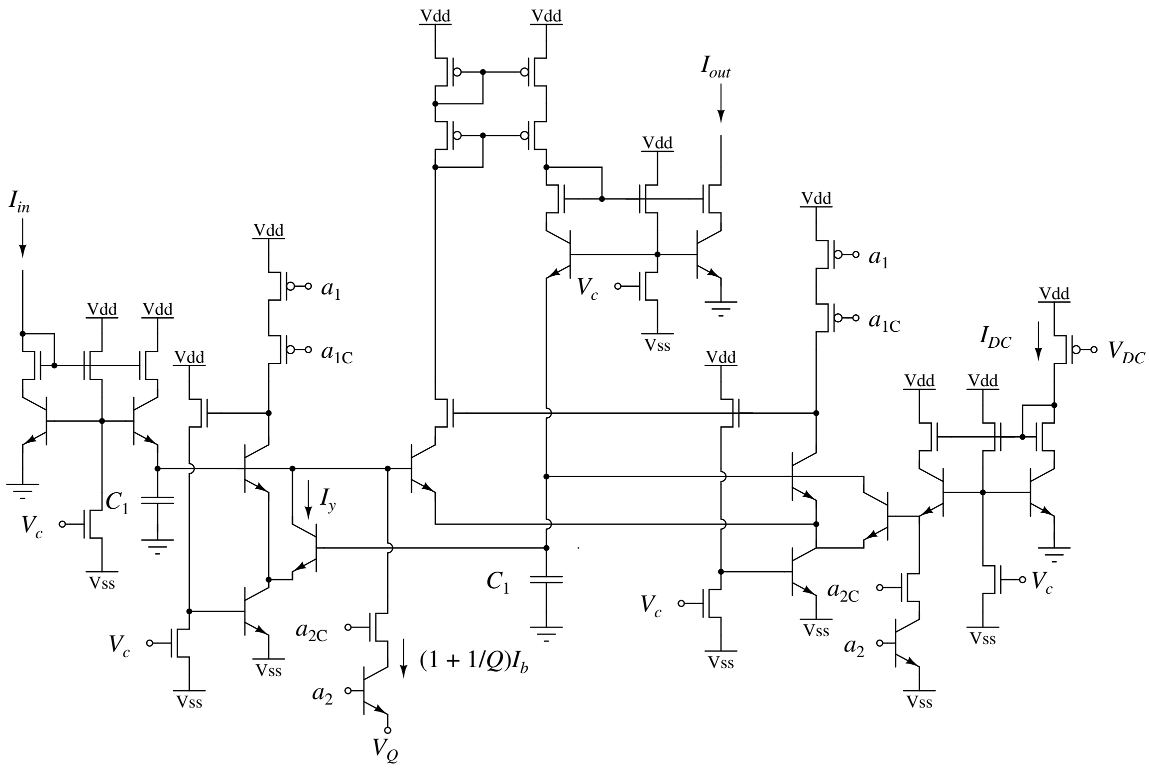


Figure 3.15: Circuit of Figure 3.12, showing base compensation circuits, cascode connections, and the Q-generating circuit

.

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

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

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

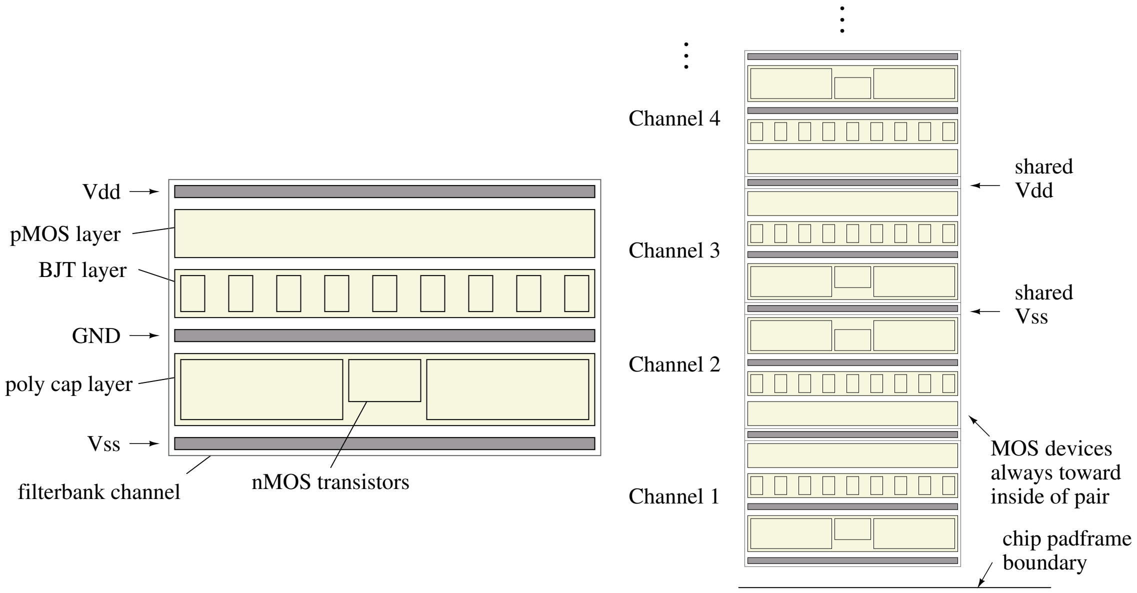


Figure 3.16: Structure of a bandpass filterbank channel and stacking of channels to form the whole filterbank.

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

3.13 Signal Rectifier

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

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

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

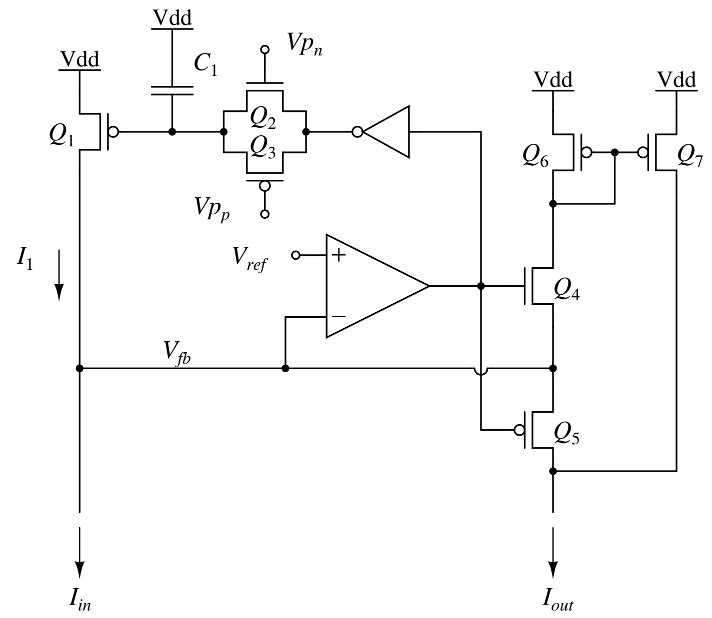


Figure 3.17: Simplified schematic of the adaptive current full-wave rectifier.

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

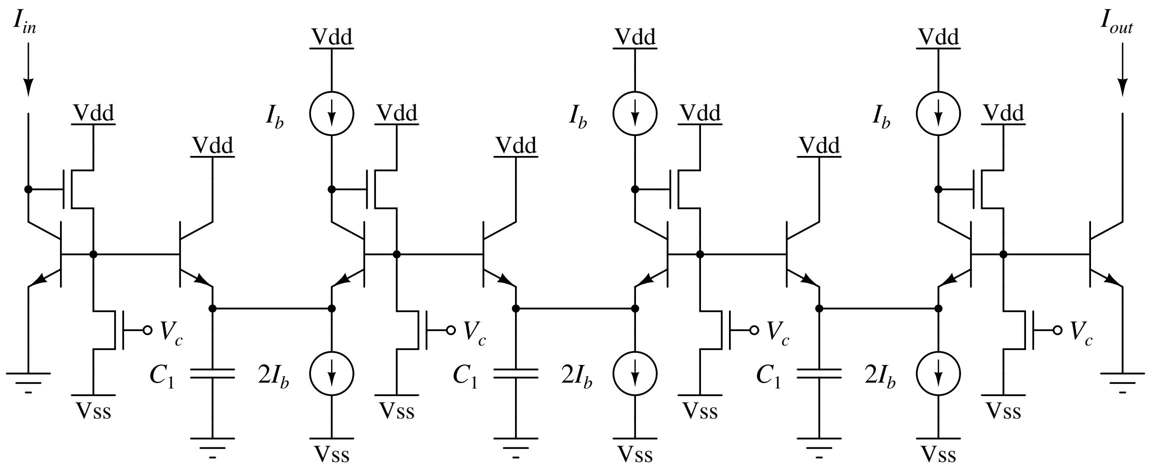


Figure 3.18: Cascaded log-domain lowpass filters.

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

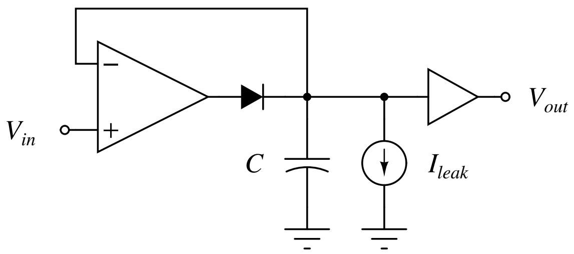


Figure 3.19: Simple diode-based peak detector.

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

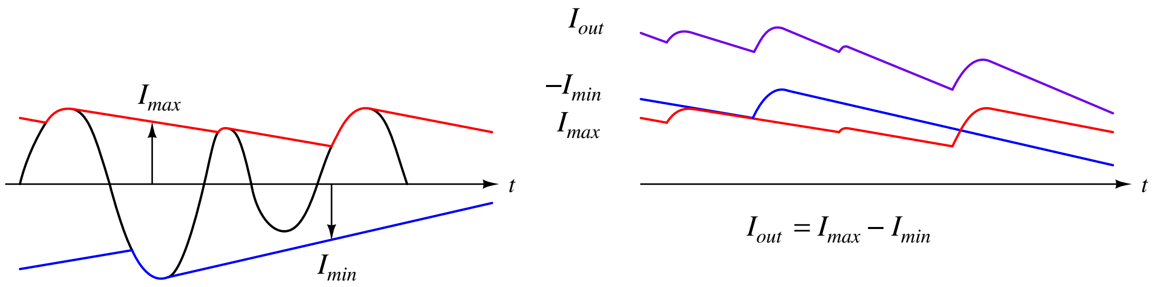


Figure 3.20: Behavior of the peak-peak detector.

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

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

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

3.15 L-1 Normalization Array

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

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

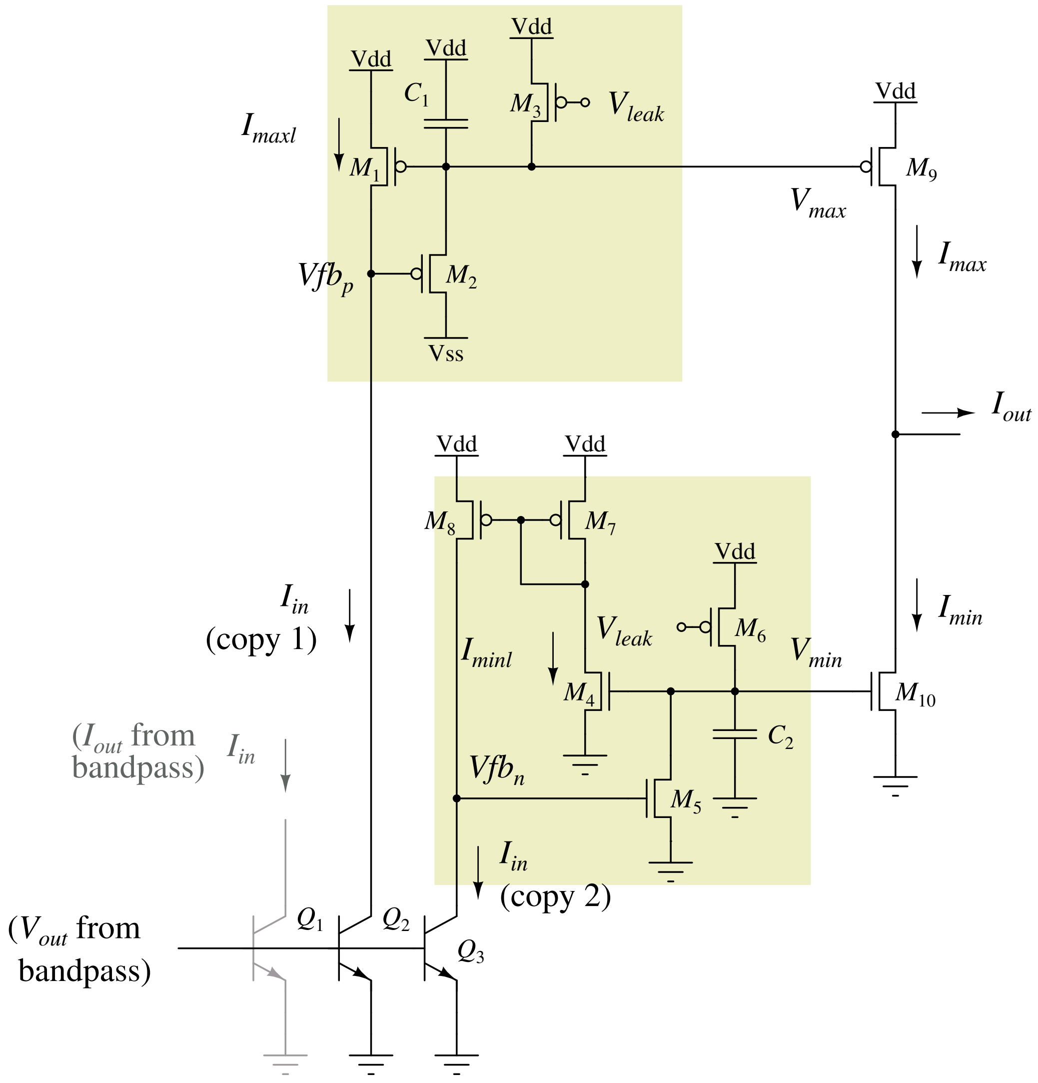


Figure 3.21: Peak-peak detector circuit, simplified.

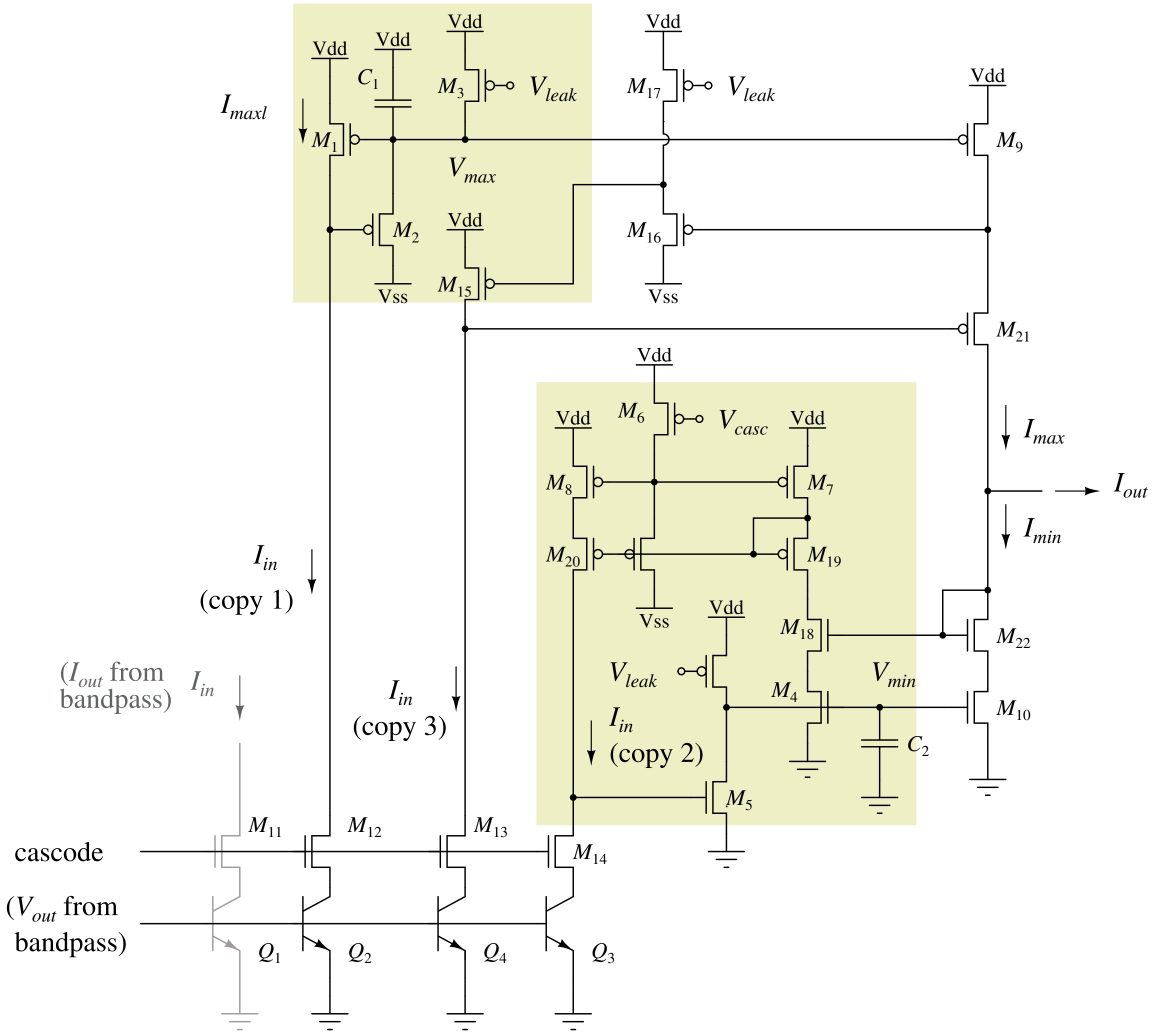


Figure 3.22: Peak-peak detector with cascodes, as fabricated.

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 band-pass 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

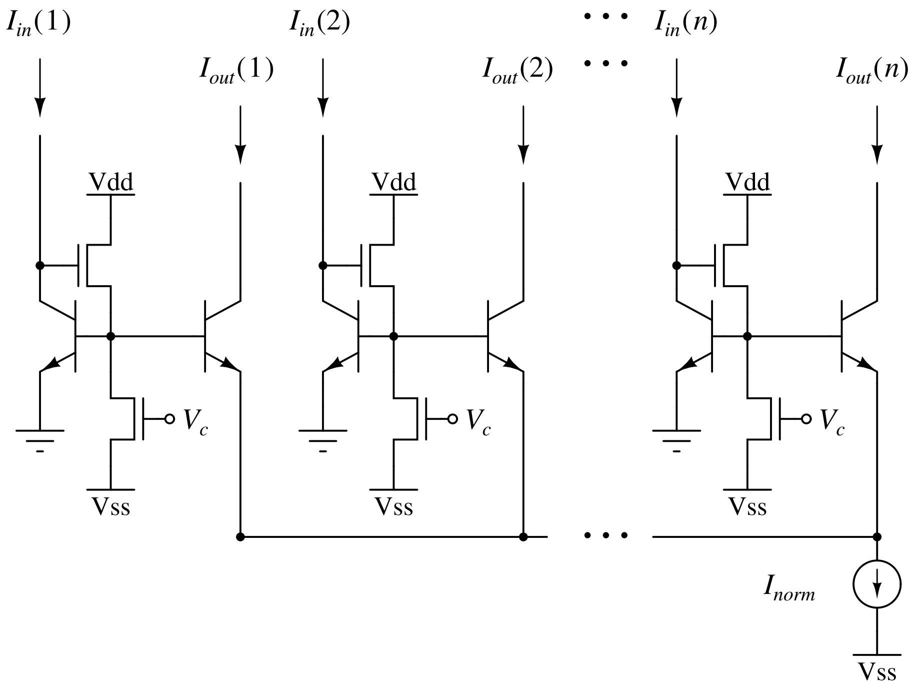


Figure 3.23: L-1 normalization circuit, after Gilbert [50].

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

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

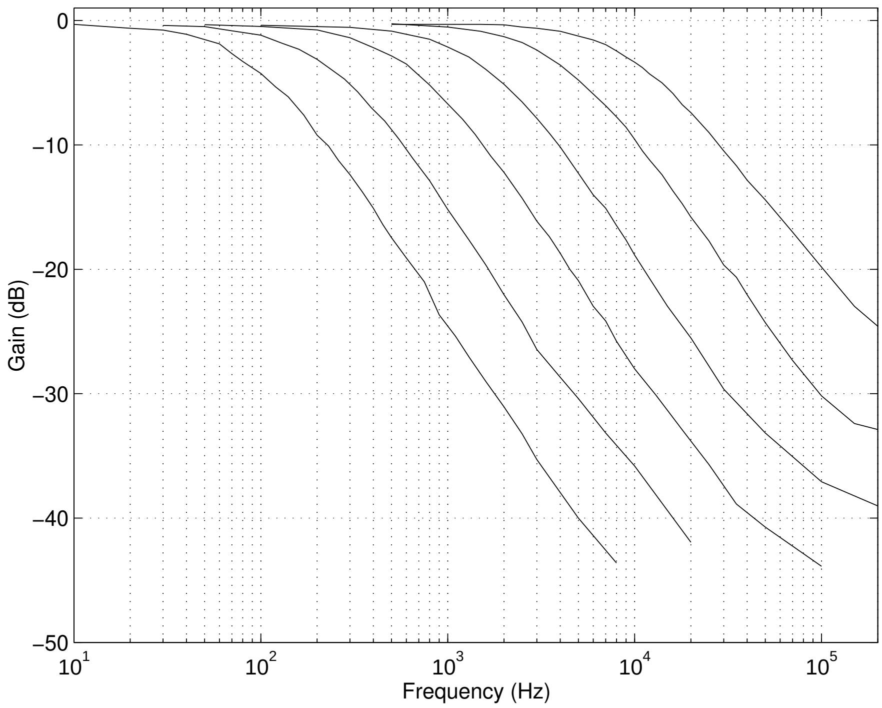


Figure 3.24: Measured magnitude response of the log-domain first-order lowpass filter from the test chip.

Table 3.1: Measured filterbank characteristics.

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

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

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

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

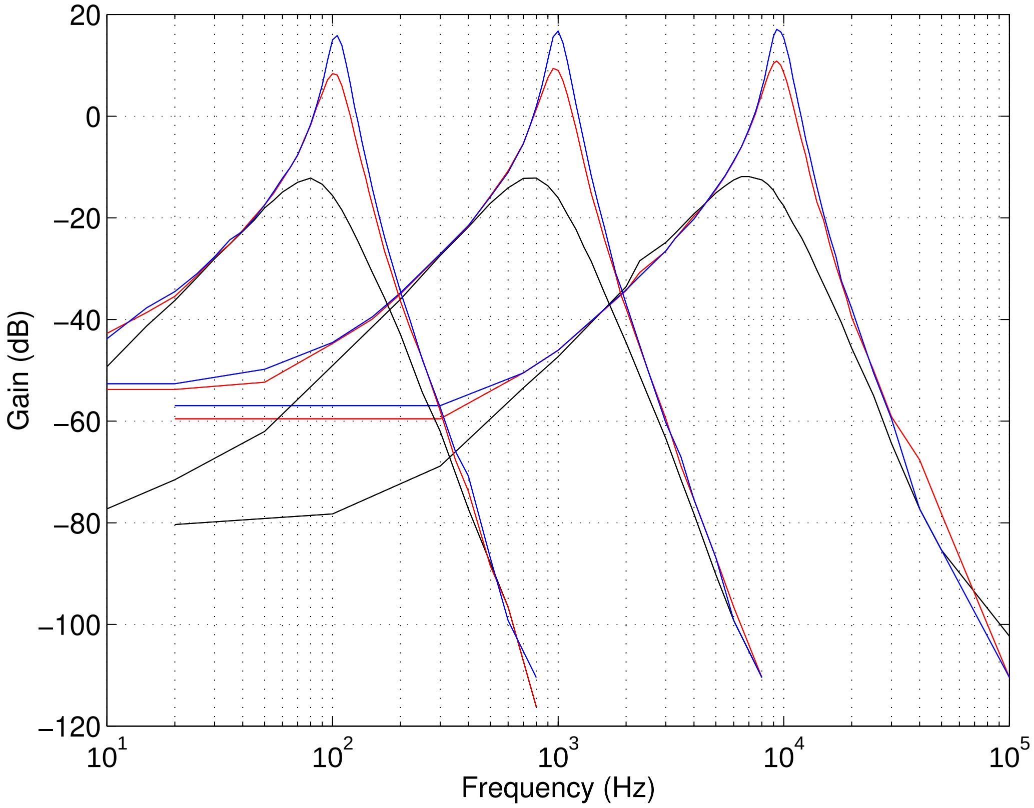


Figure 3.25: Measured magnitude response of one bandpass channel in the filterbank system, made of two cascaded second-order log-domain bandpass filters, over three tunings of the center frequency.

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

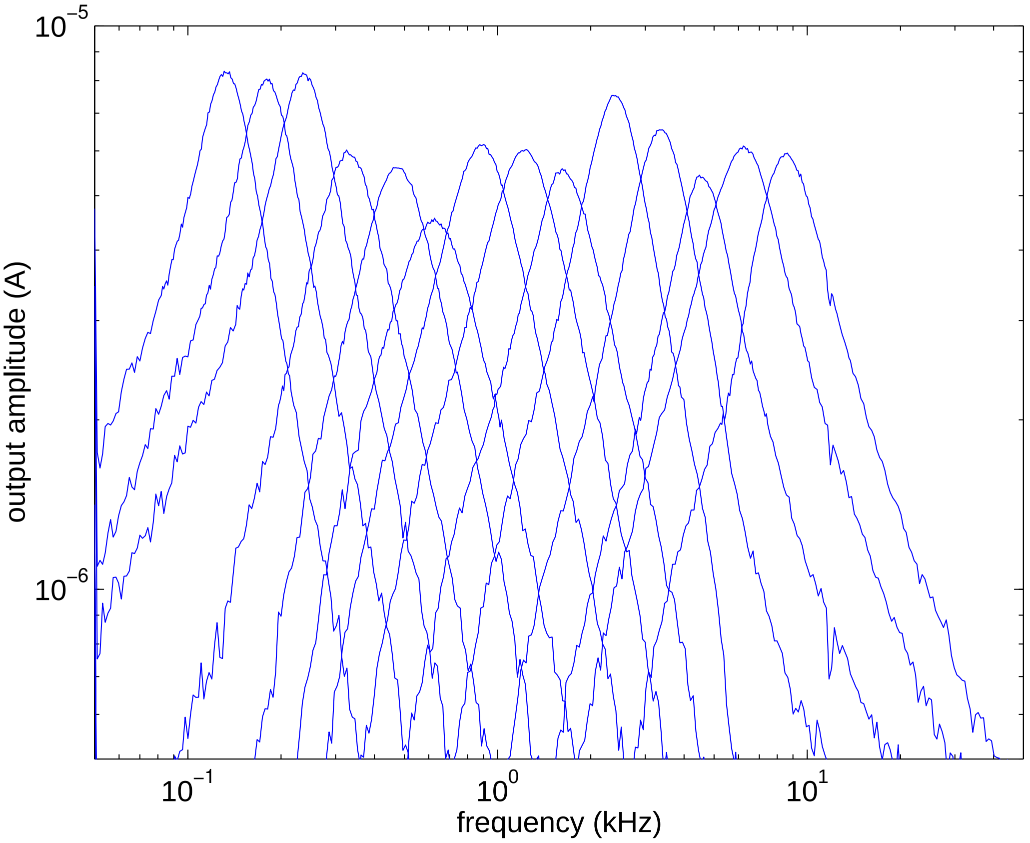


Figure 3.26: Measured magnitude response of all bandpass channels in the filterbank system, as measured at the output after peak-detection and smoothing.

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

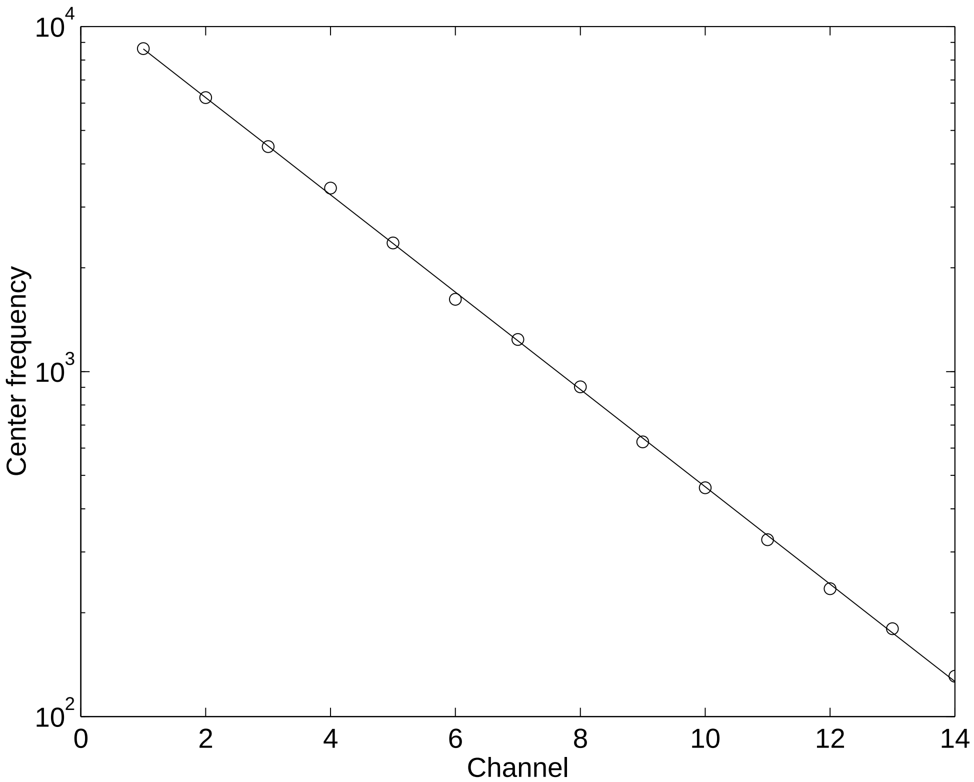


Figure 3.27: Measured center frequencies of all bandpass channels in the filterbank system (circles), compared to the ideal exponential spacing (solid line).

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.

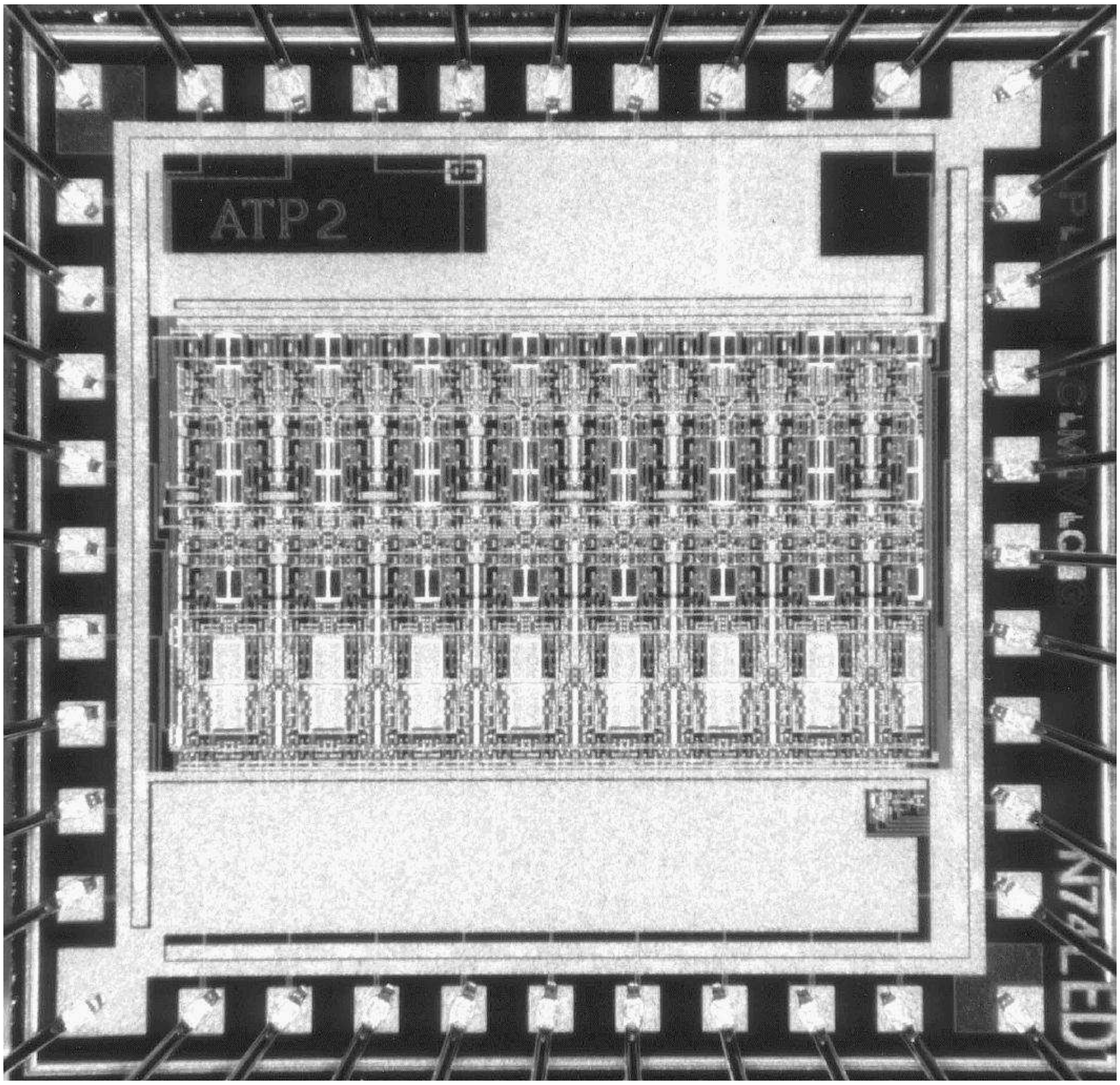


Figure 3.28: Photograph of the fifteen-channel bandpass filterbank fabricated in 1.2 µm technology inside a 2.2 mm × 2.2 mm padframe.

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].

Chapter 4

Acoustic Transient Processing

4.1 Introduction

4.1.1 The Problem of Speech Recognition

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

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

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

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

4.2 Algorithms

There exist many kinds of pattern classification algorithms. Some of them work by finding the correlation between a target pattern and a stored prototype pattern, called the *template*. Correlation is one of the simplest classification schemes, both conceptually and mathematically. The correlation is a linear system effectively reducing the dimensionality of a large input space into a 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 topics of how to perform the classification from the correlation outputs and how to determine the optimal template for each class are discussed further in Section 4.3.6 and following.

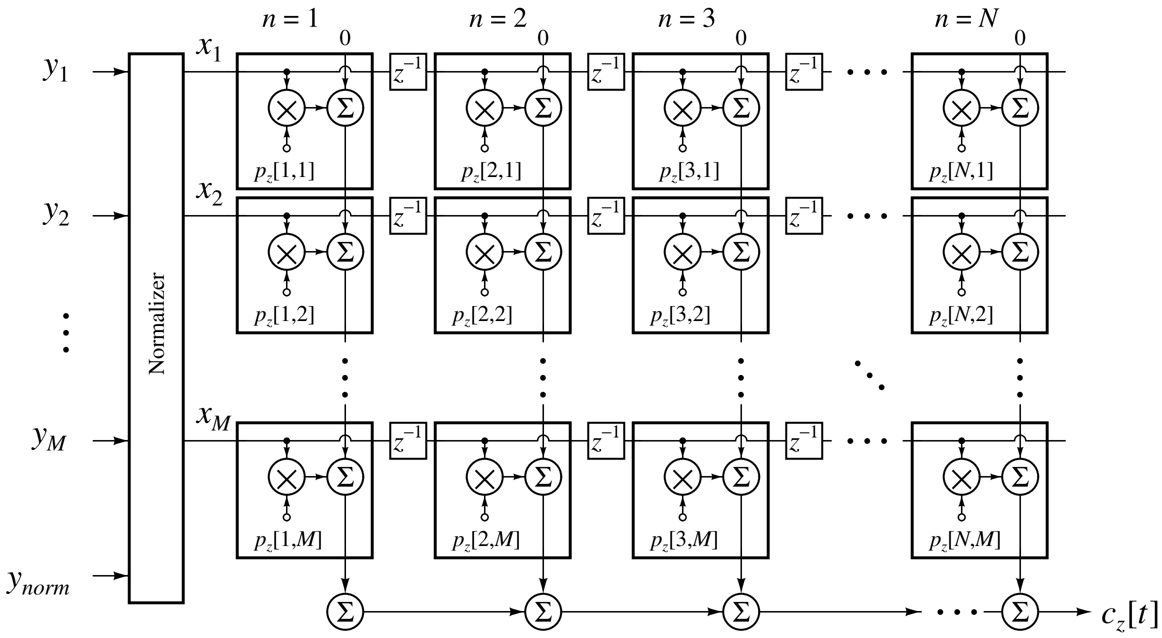


Figure 4.1: Template correlator, as a direct implementation of the baseline algorithm, Equation (4.1).

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

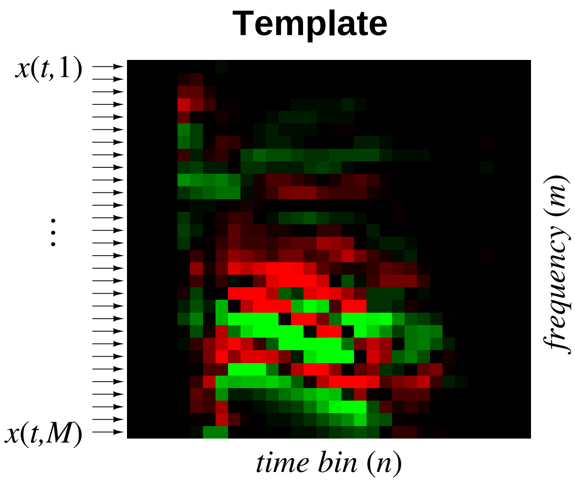


Figure 4.2: Template correlation.

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:

1. First, we normalize the input.

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

3. We replace the template values with binary values.

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

4.2.1 Simplifying The Correlation Equation

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

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

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

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

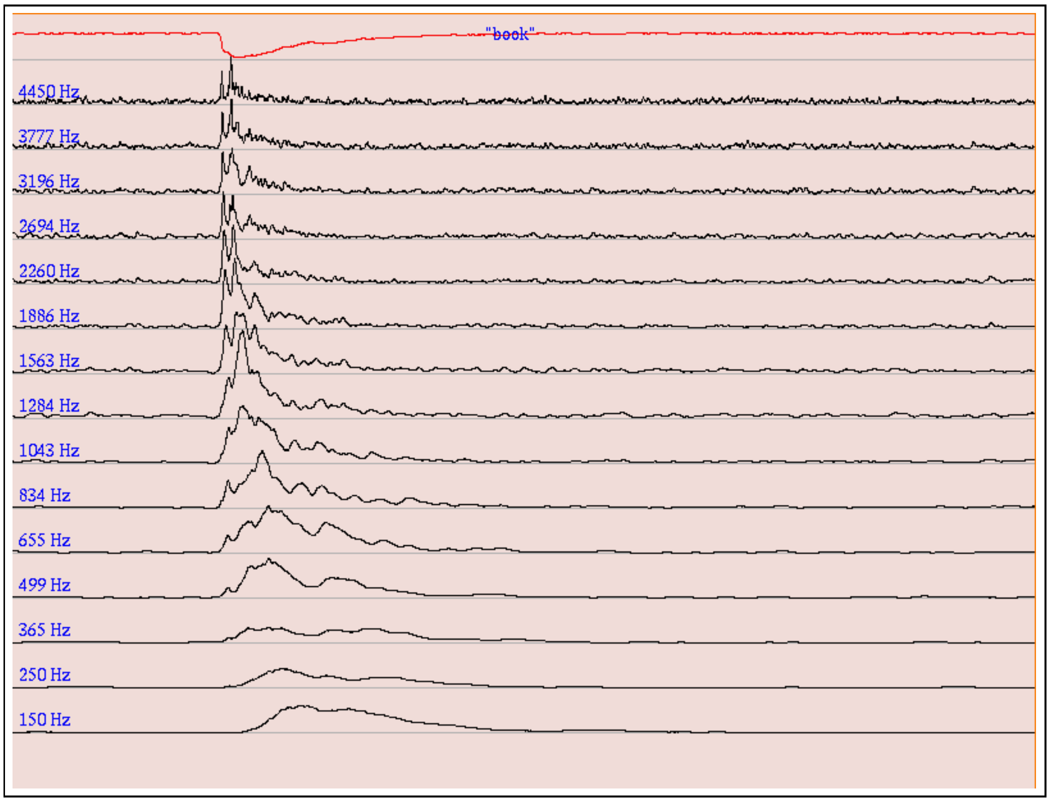


Figure 4.3: Filterbank output after applying an L-1 normalization across all channels. A single channel has been added to the system (top trace), containing the result of a constant value less the instantaneous sum of the remaining channel values.

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

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

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

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

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

where

$$\begin{aligned} x'[t,m] &= (x[t,m] - x[t,m-1]) && (4.4)\\ {p'}_z[n,m] &= \mathrm{sign}(p_z[n,m] - p_z[n,m-1]). && (4.5) \end{aligned}$$

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

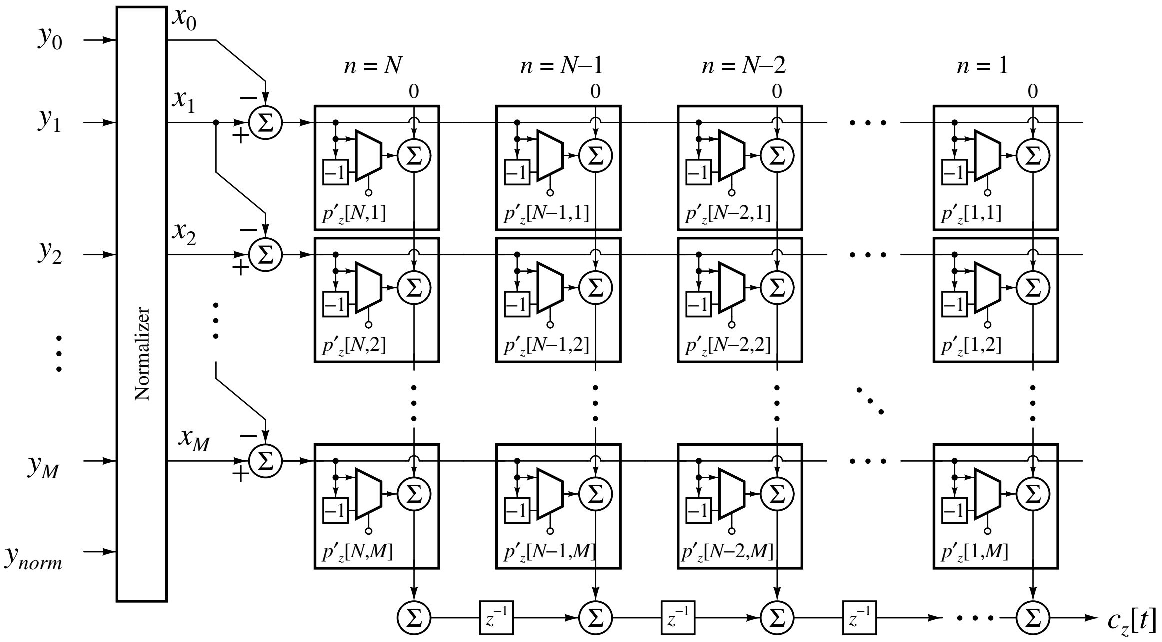


Figure 4.4: Template correlator with pipelined architecture and multiplexors replacing the multipliers.

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:

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

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

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

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

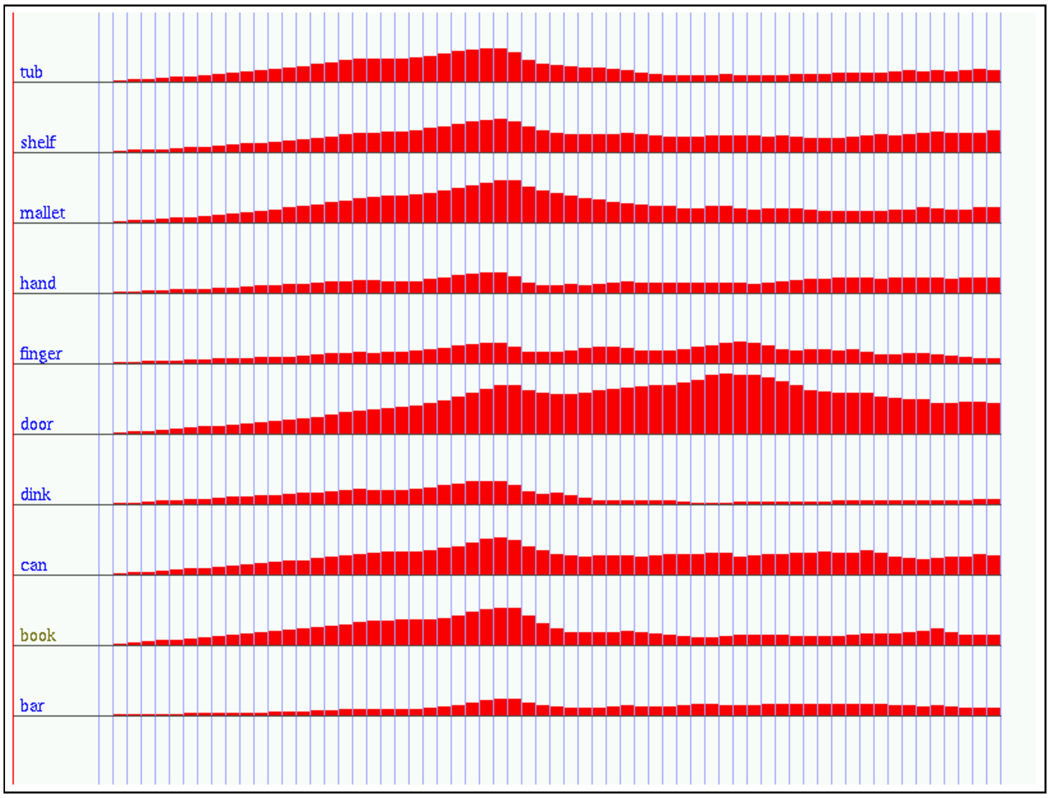


Figure 4.5: Values in the pipelined delay registers in the correlation algorithm simulation.

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

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.

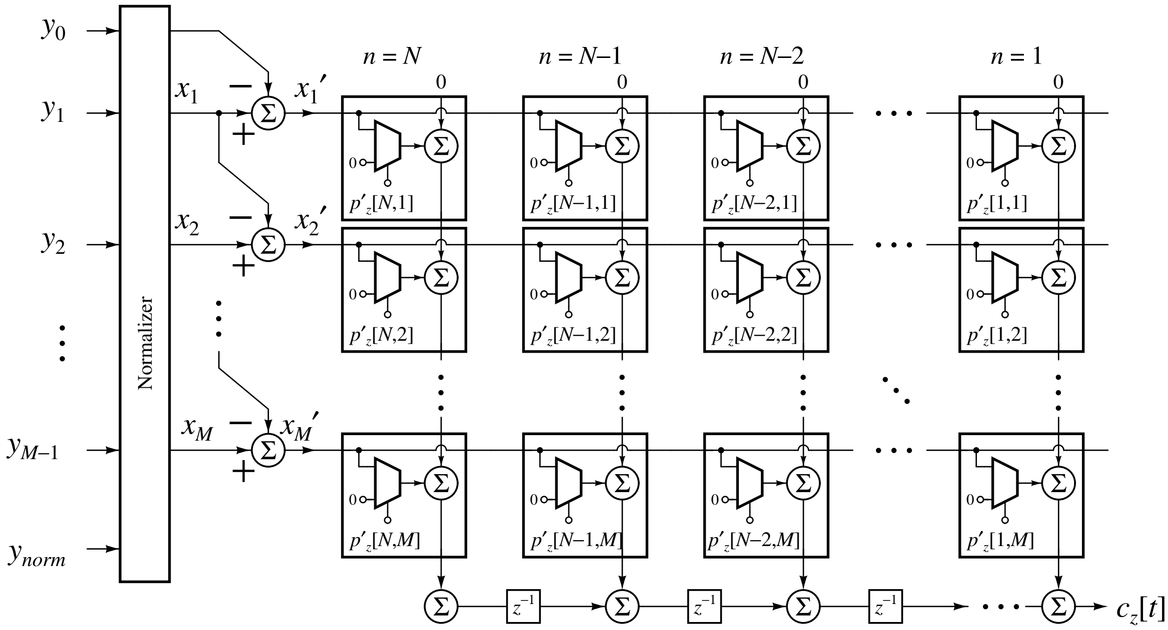


Figure 4.6: Template correlator with [0,1] encoding of template values.

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

$$
\begin{aligned}
q_1[n,t] &= q_1[n-1,t-1] + \sum_{m=1}^{M} x[t,m]\, {p'}_z[n,m] \qquad \forall n \neq 1 && (4.9)\\
q_2[n,t] &= q_2[n-1,t-1] + \sum_{m=1}^{M} x[t,m-1]\, {p'}_z[n,m] \qquad \forall n \neq 1 && (4.10)\\
q_1[1,t] &= \sum_{m=1}^{M} x[t,m]\, {p'}_z[1,m] && (4.11)\\
q_2[1,t] &= \sum_{m=1}^{M} x[t,m-1]\, {p'}_z[1,m] && (4.12)
\end{aligned}
$$

Thus, the correlation output becomes:

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

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

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

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

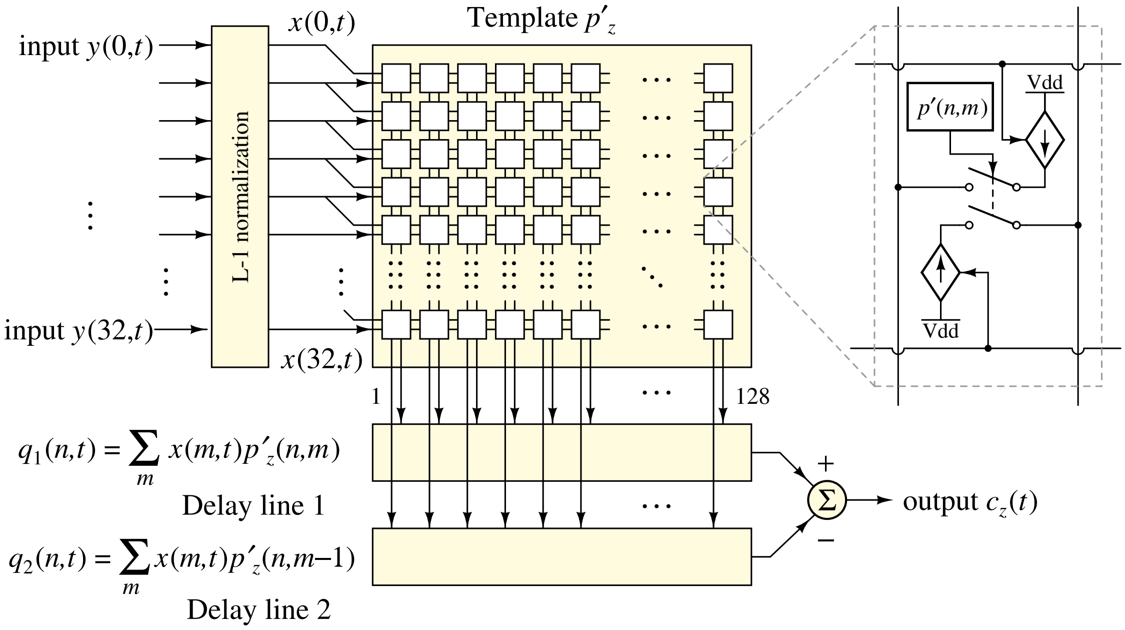


Figure 4.7: Block diagram of the temporal current correlator.

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

4.3.1 The Hopkins Electronic Ear (HEEAR) Processor

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

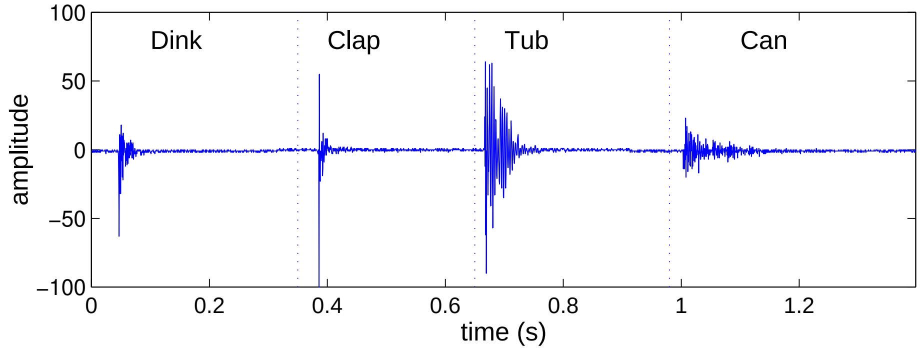


Figure 4.8: Example transient recorded data.

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

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

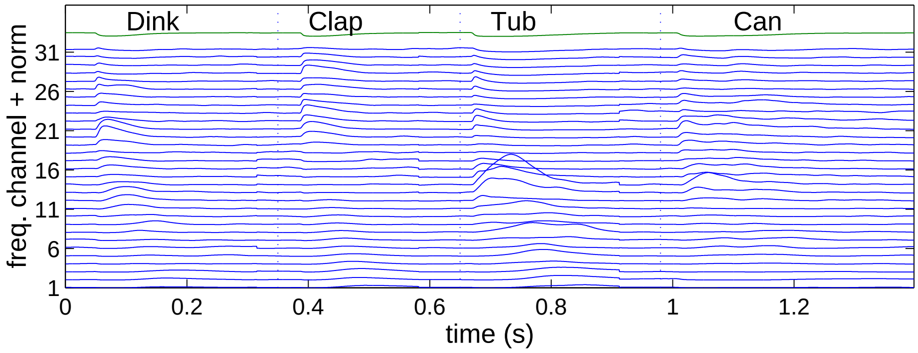


Figure 4.9: Example HEEAR-processed transients.

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

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

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

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

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

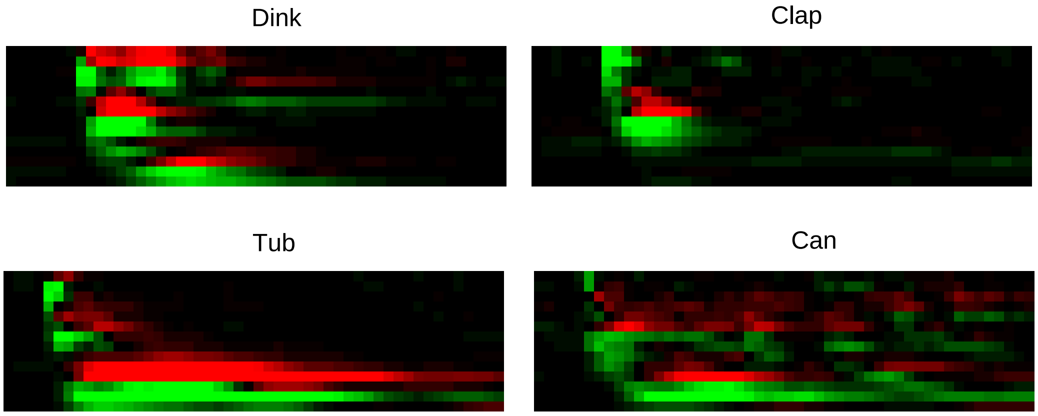


Figure 4.10: Example templates learned by the ATP algorithm.

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

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

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

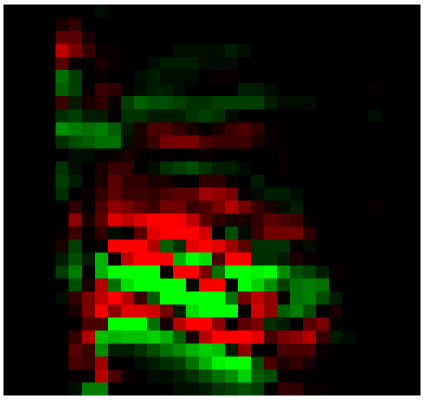


Figure 4.11: Example of a real-valued template.

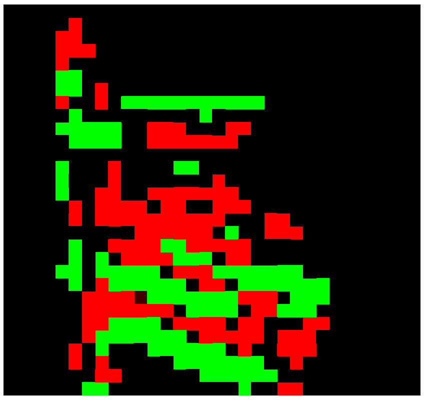


Figure 4.12: The same template reduced to trinary values.

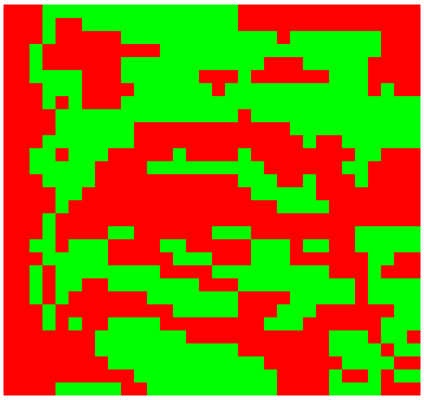


Figure 4.13: The same template reduced to binary values.

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 on the time scale of transients.

Table 4.3: Simulation results with different architectures.

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

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

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

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

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

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

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

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

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

4.3.6 Optimization of the classifier using per-class gains

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

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

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

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

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

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

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

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

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

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

while at the same time minimizing the expressions

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

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

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

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

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

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

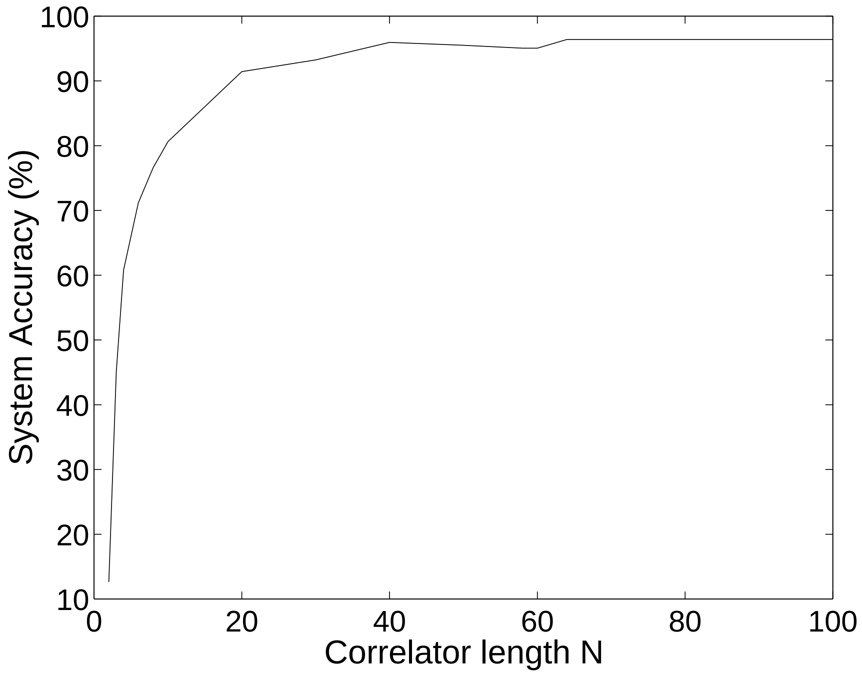


Figure 4.14: Effect of decreasing the number of time-bins.

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

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

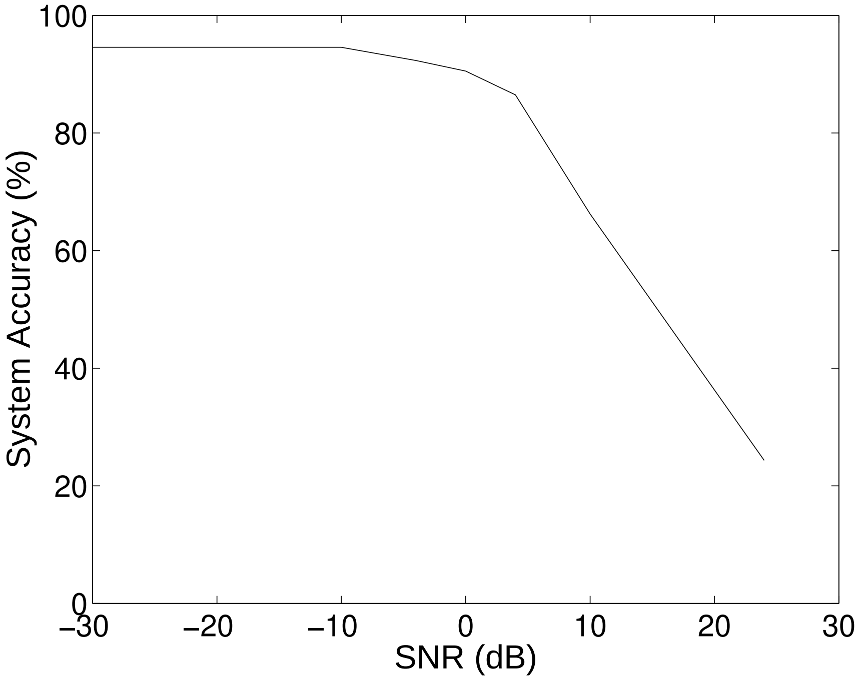


Figure 4.15: Effect of white noise added to the correlator inputs.

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

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

4.3.9 Remarks

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

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

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

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

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 $\mathit{Write}(m)$ running horizontally across the template array and two lines for the input $\mathit{bit}(n)$ and $\overline{\mathit{bit}}(n)$ 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.

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

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

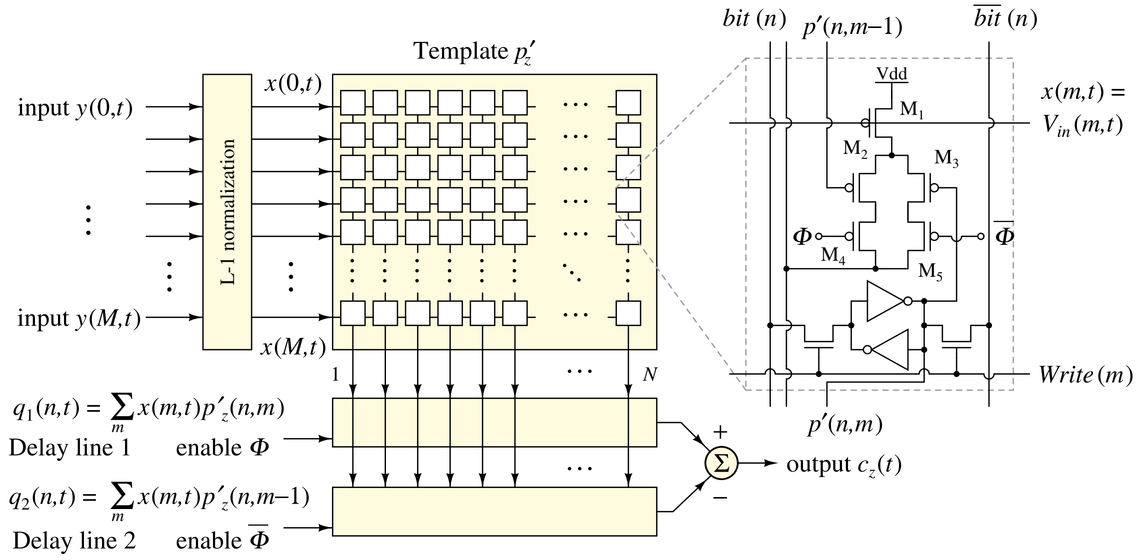


Figure 4.16: Block diagram of the temporal current correlator.

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

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

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

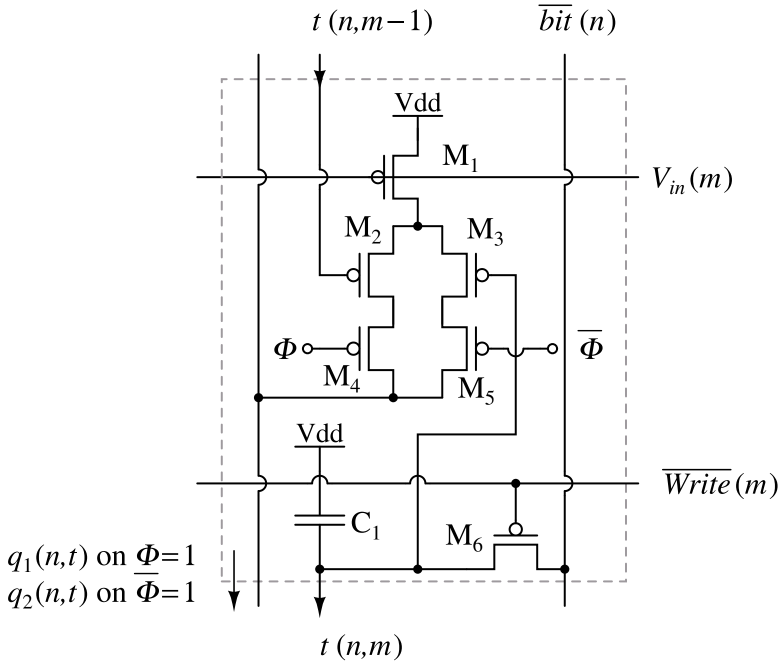


Figure 4.17: Efficient dynamic memory cell for the correlator array.

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

4.4.2 Bucket Brigade Device

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

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

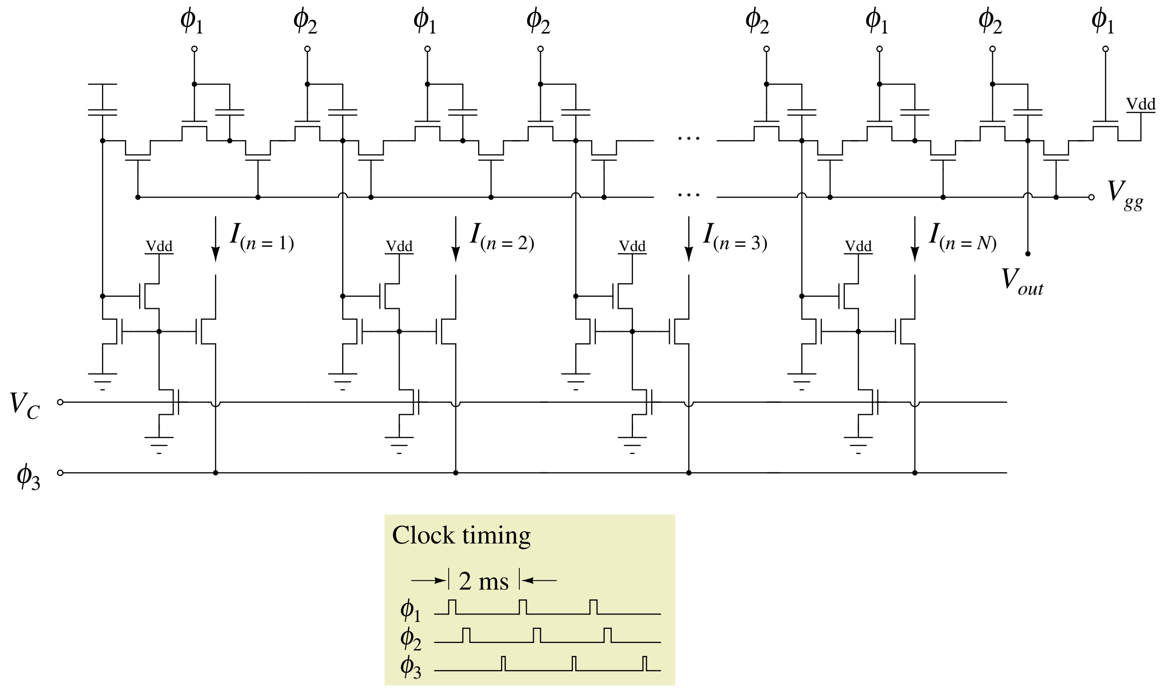


Figure 4.18: Details of the bucket brigade device (BBD).

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

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

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

4.4.3 Circuit input section

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

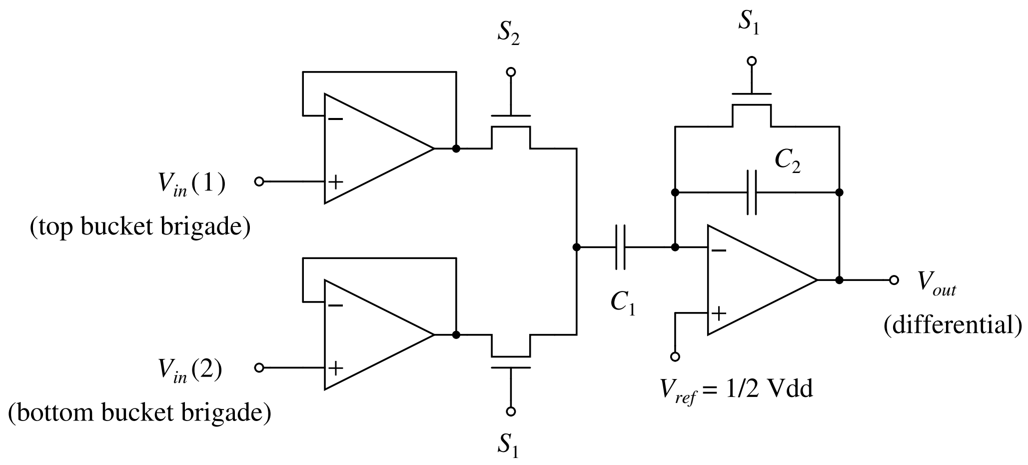


Figure 4.19: A switched capacitor circuit to compute channel differences at the output.

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.

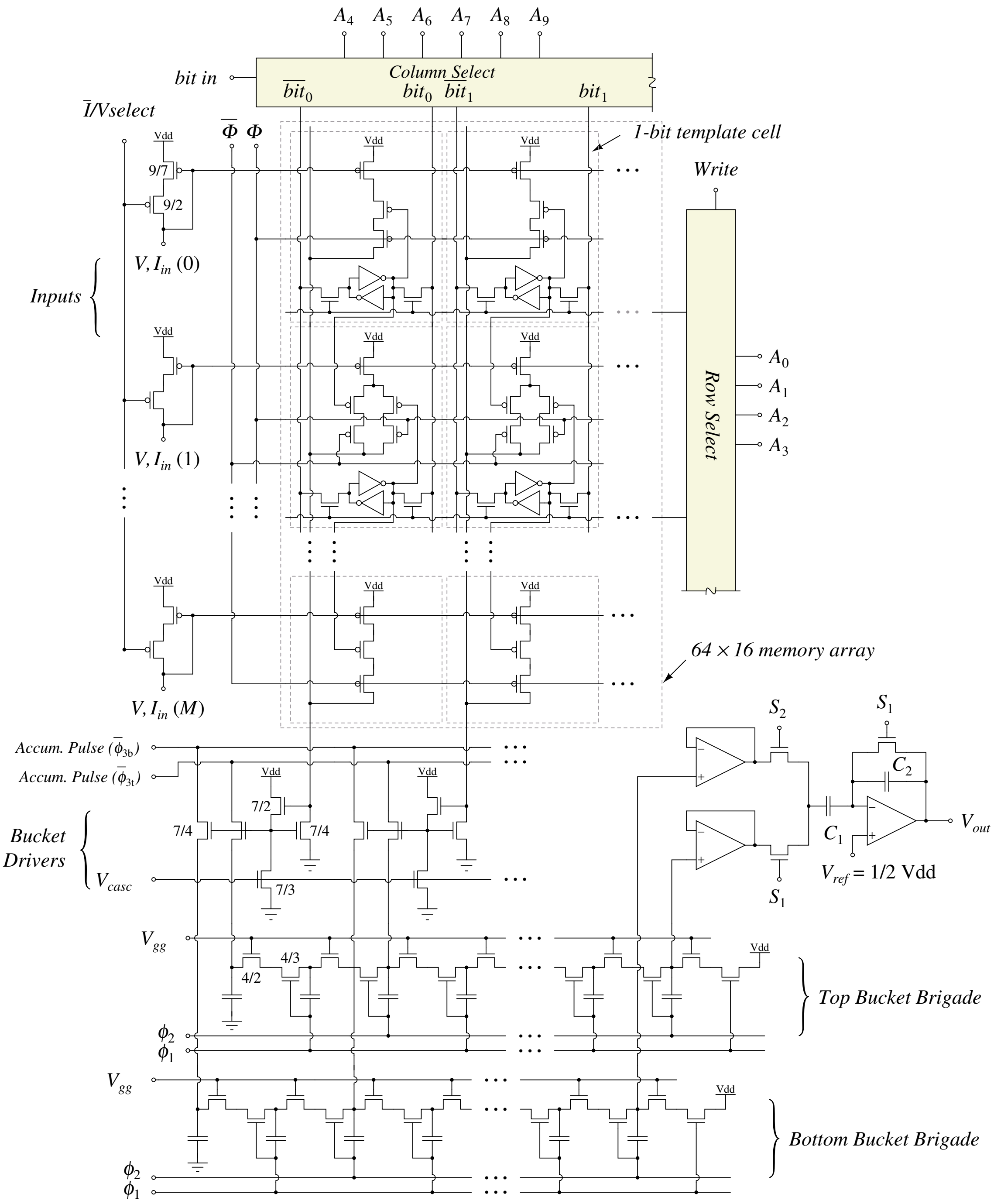


Figure 4.20: The complete correlator array including both bucket brigade devices. Representative MOS device sizes are given as W/L in units of $\lambda = 0.6\,\mu\text{m}$.

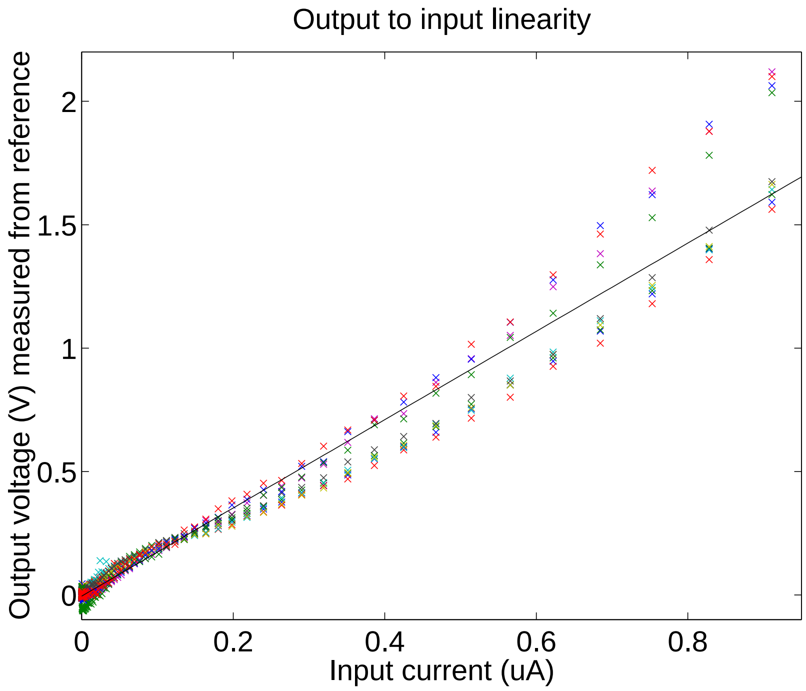


Figure 4.21: Linearity between output voltage and input current.

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.

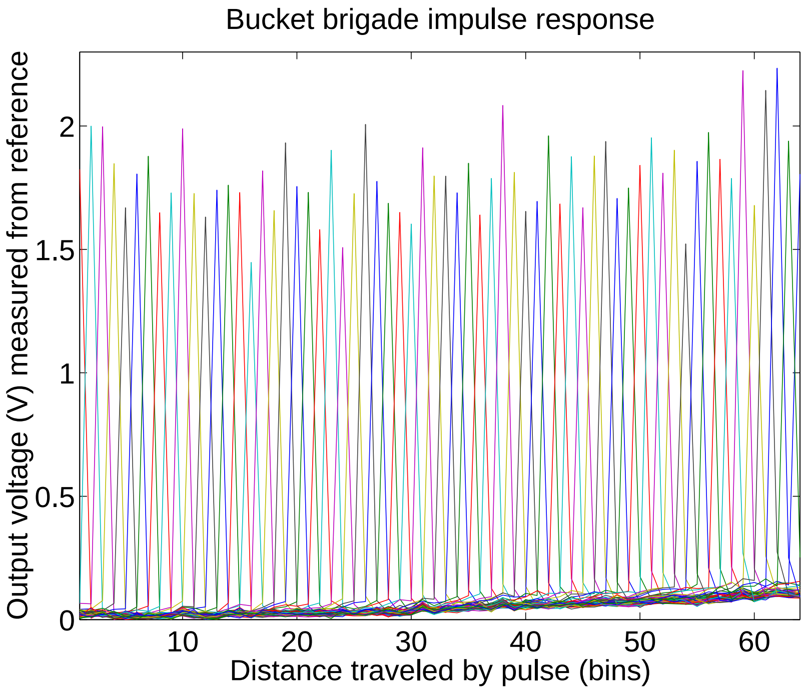


Figure 4.22: Impulse response at each tap of the bucket brigade device.

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

4.4.5 Experimental Results

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

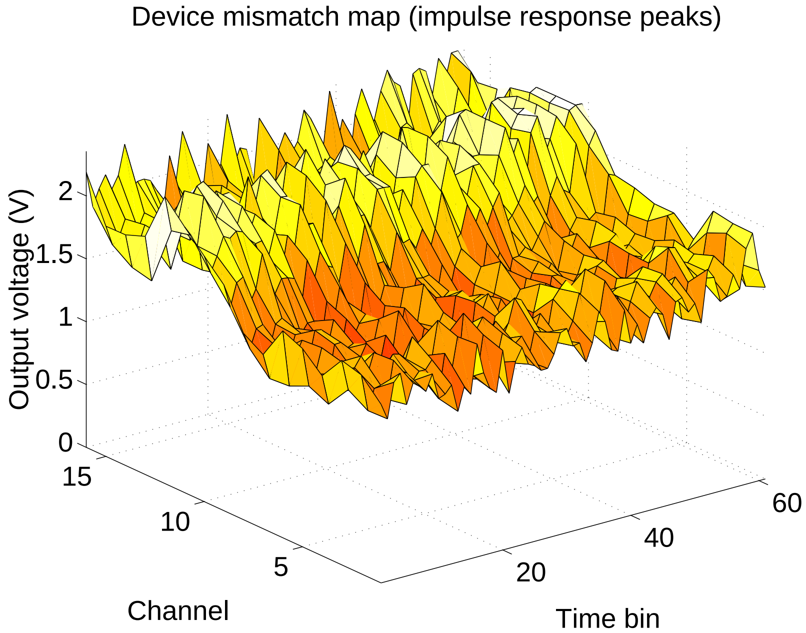


Figure 4.23: Matching between devices throughout the template at 1 µA input.

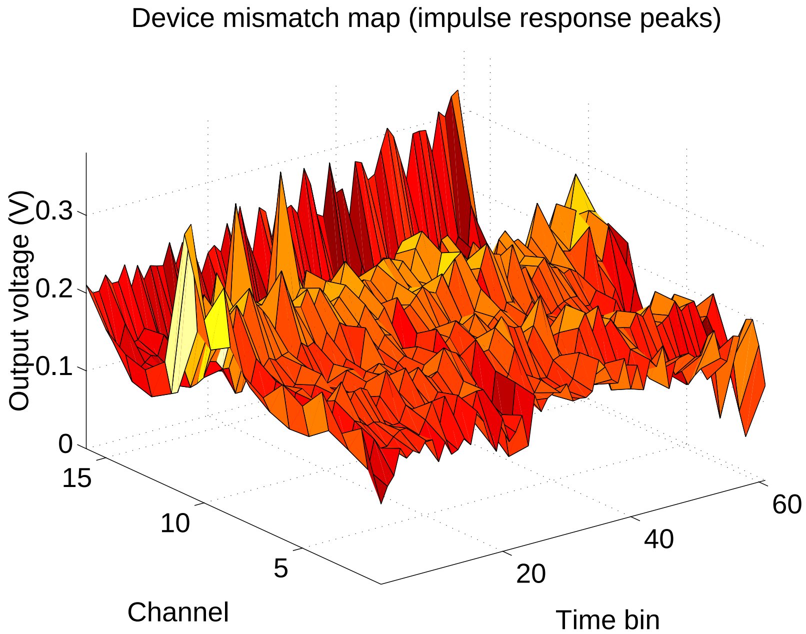


Figure 4.24: Matching between devices throughout the template at 100 nA input.

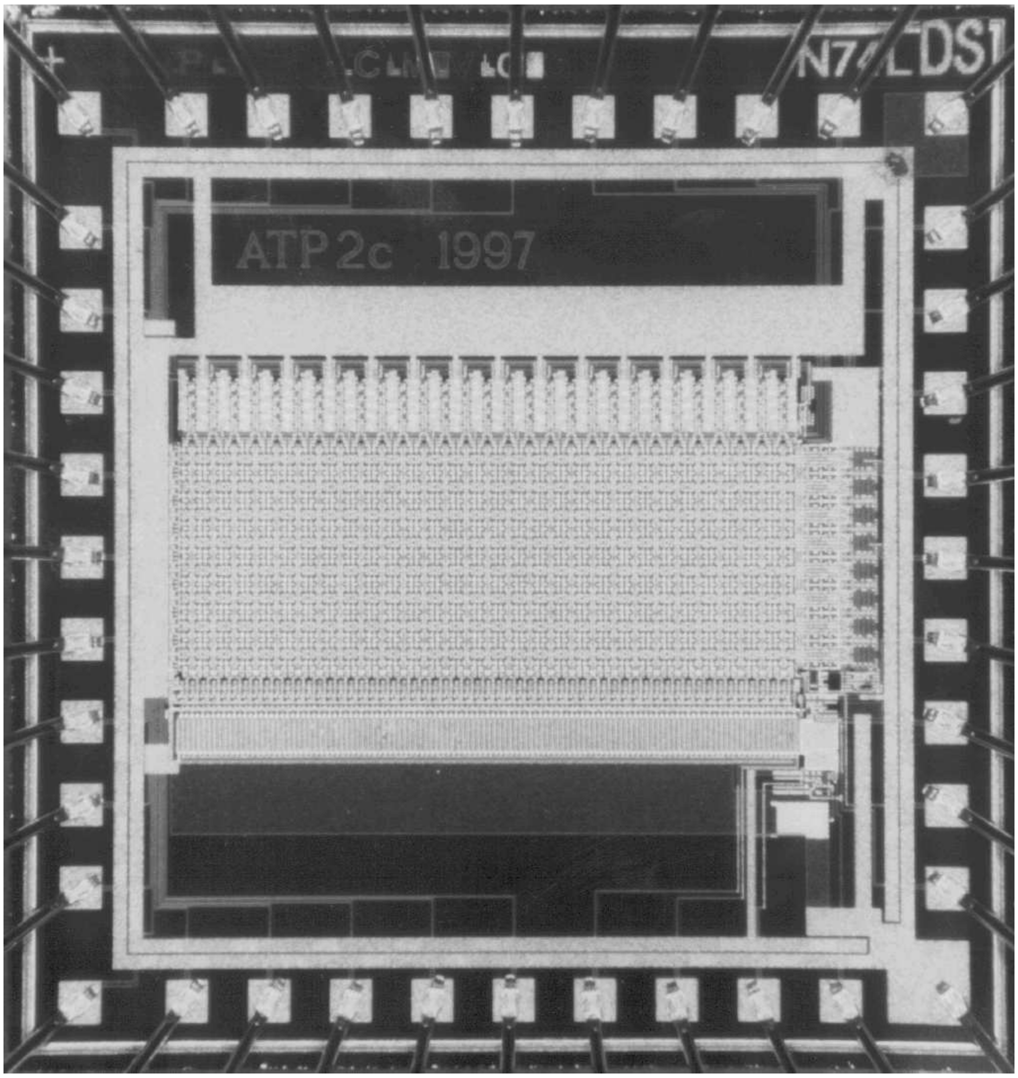


Figure 4.25: Photomicrograph of the acoustic transient correlator, a 2.2 mm × 2.2 mm die fabricated in a 1.2μm CMOS technology.

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 $\times$ 1170 μm per template allows a high integration density.

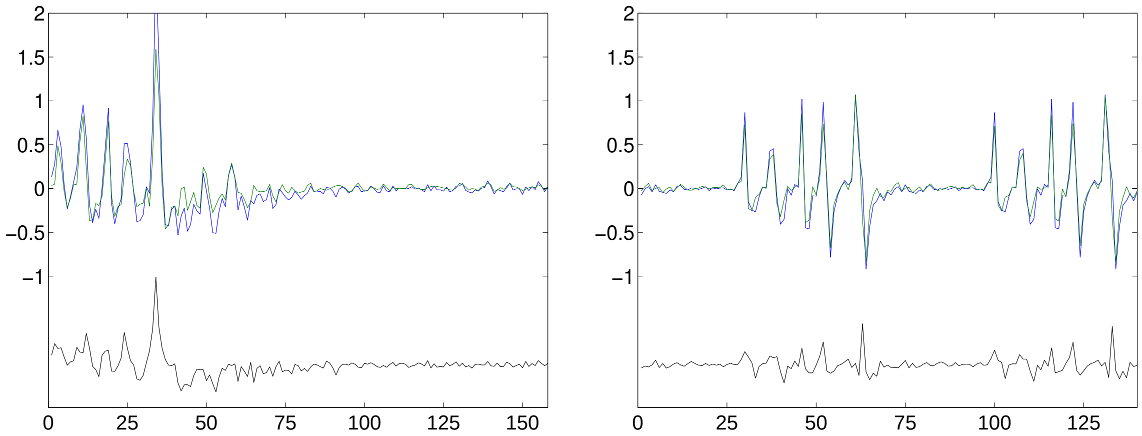


Figure 4.26: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “can” template loaded into chip memory.

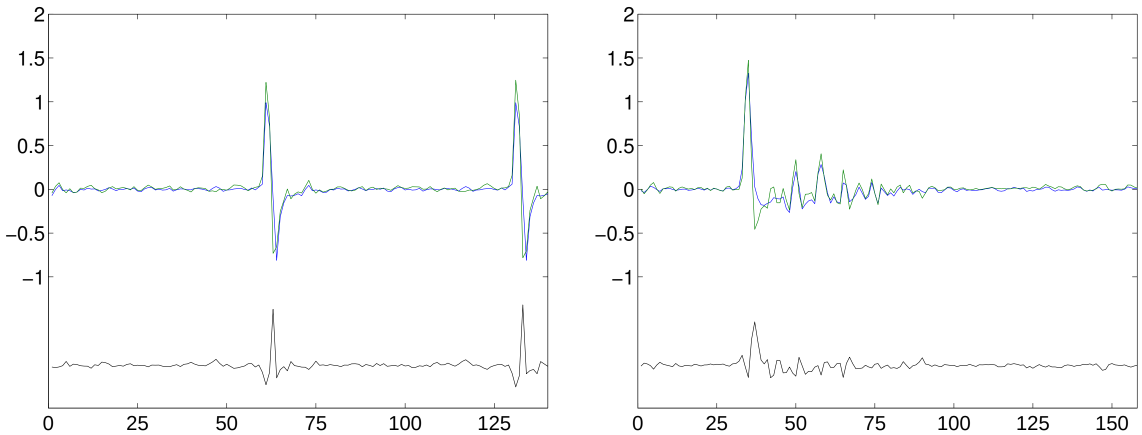


Figure 4.27: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “snap” template loaded into chip memory.

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

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

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

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

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

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

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.

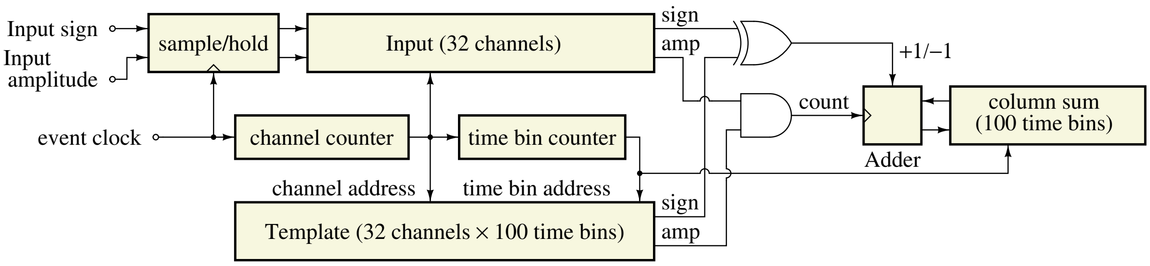


Figure 4.28: Block diagram of the sequential digital correlator architecture.

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

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

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

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

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.

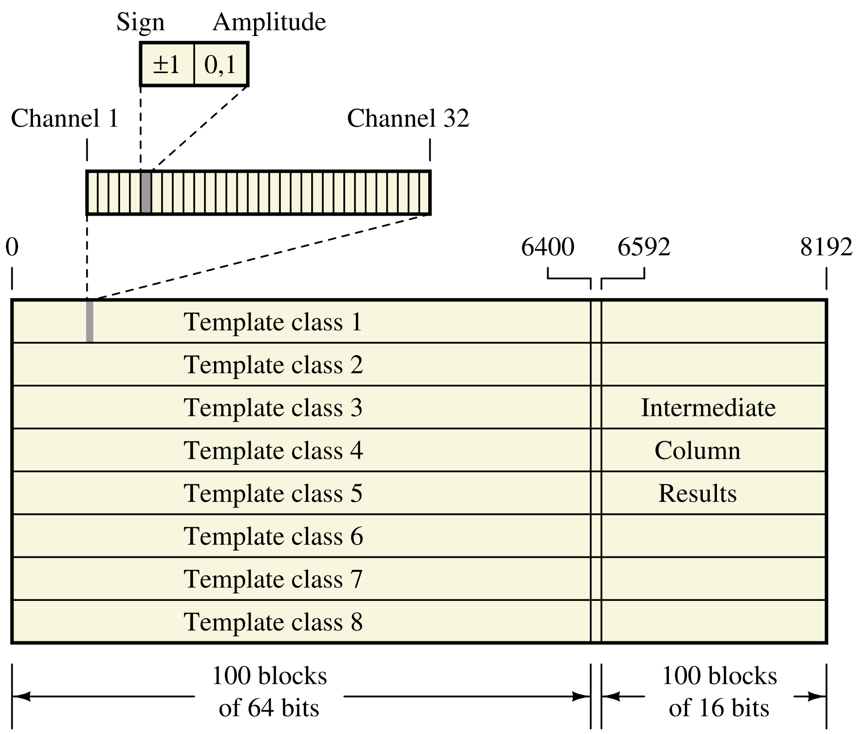


Figure 4.29: Memory allocation in the digital correlator SRAM.

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

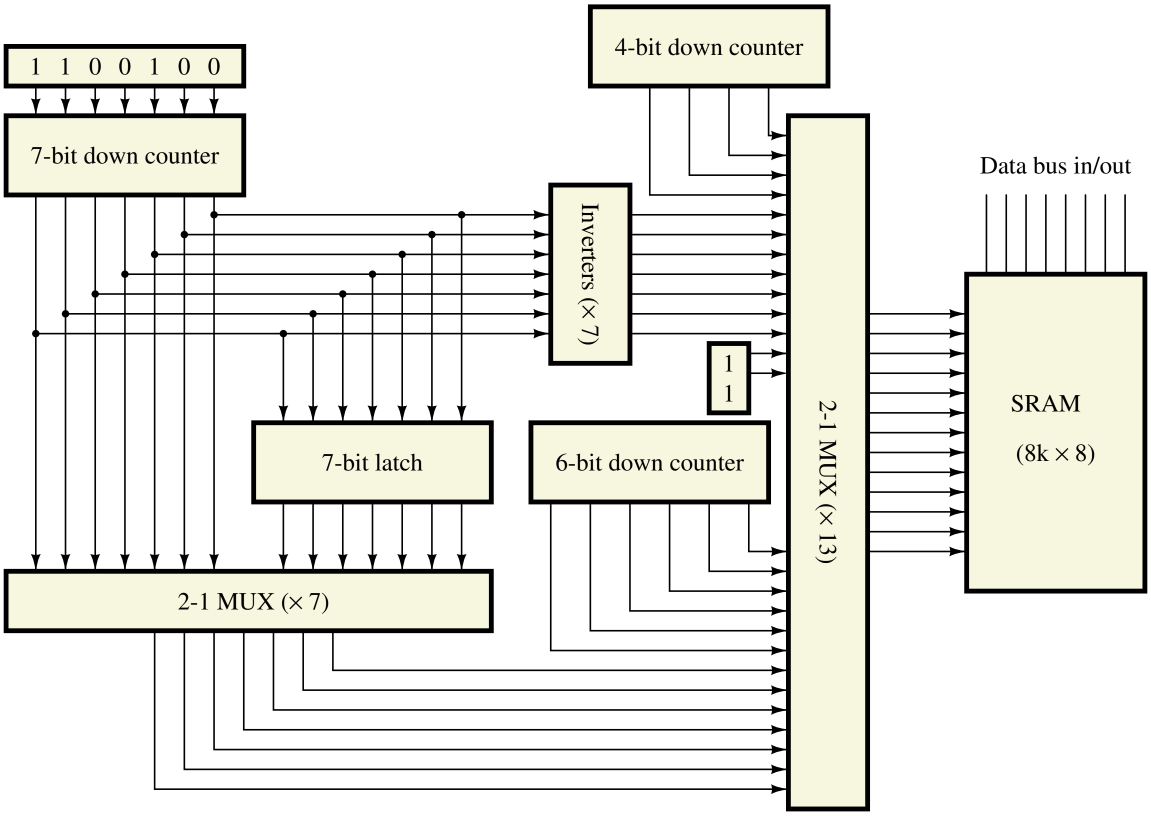


Figure 4.30: SRAM address generator for the digital correlator.

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

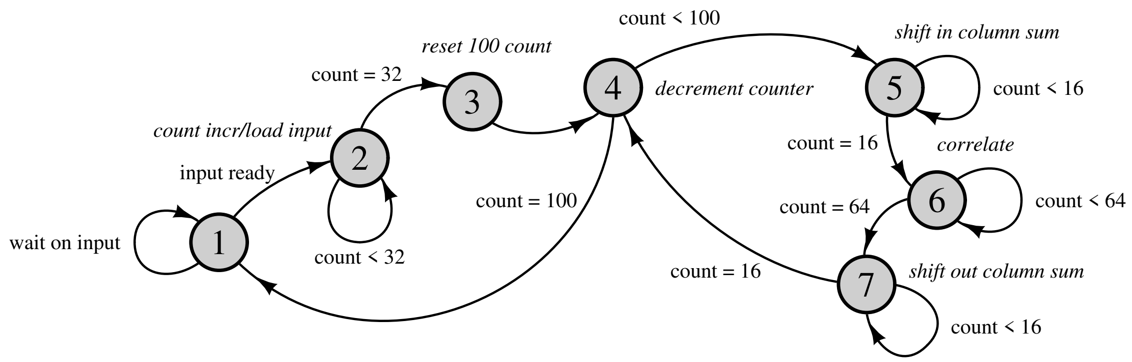


Figure 4.31: Digital correlator controller state diagram, simplified.

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.

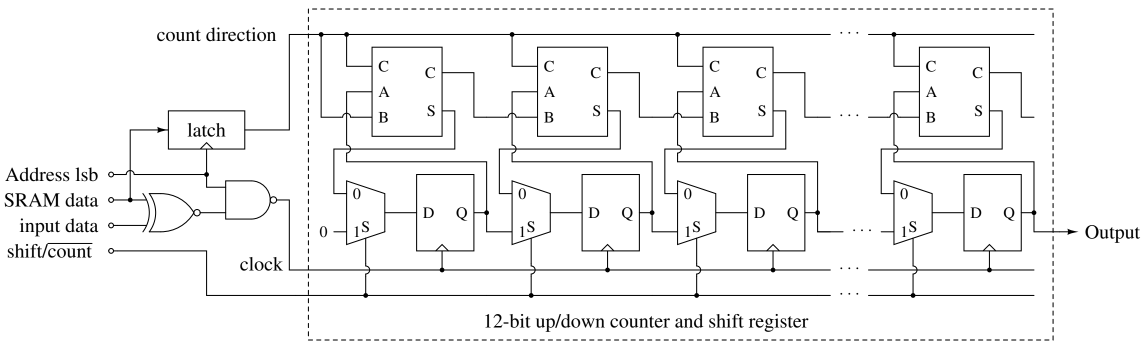


Figure 4.32: Serial digital correlator structure for one template.

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

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

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

4.8 The Switch-Capacitor Frontend

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

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

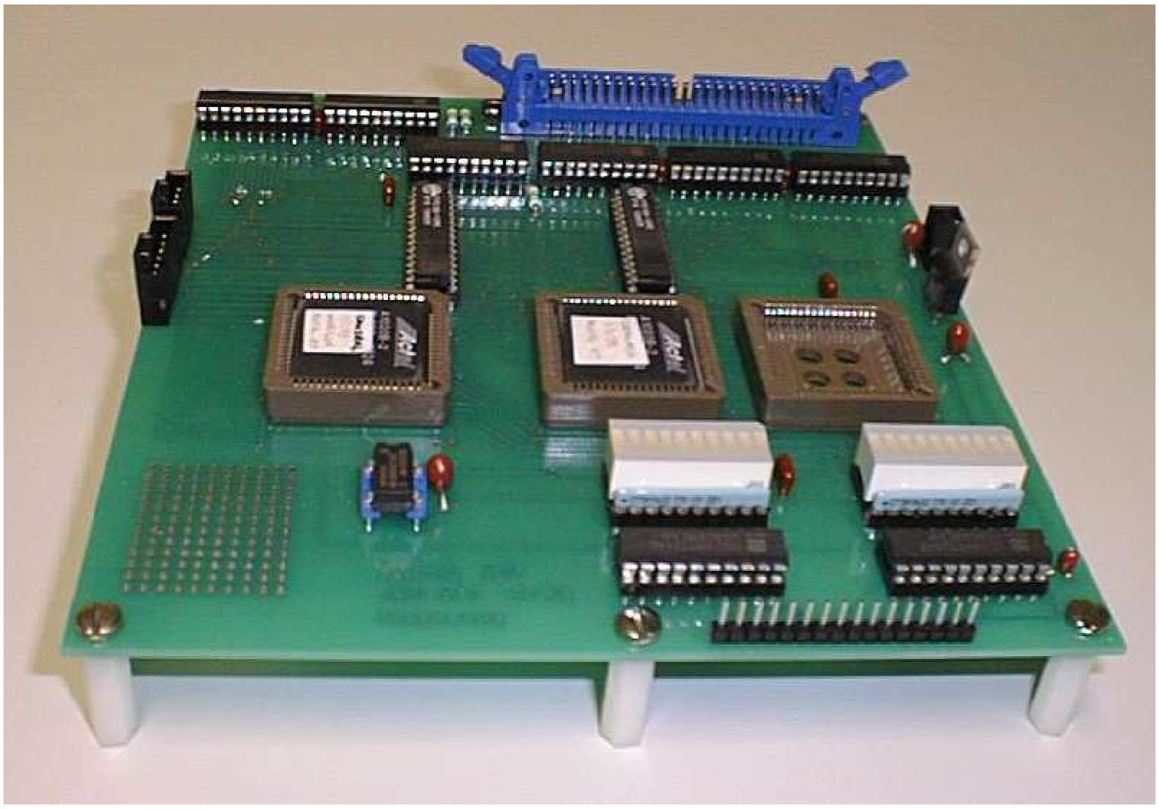


Figure 4.33: Digital ATP correlator system, configurable for up to sixteen templates.

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

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

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

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

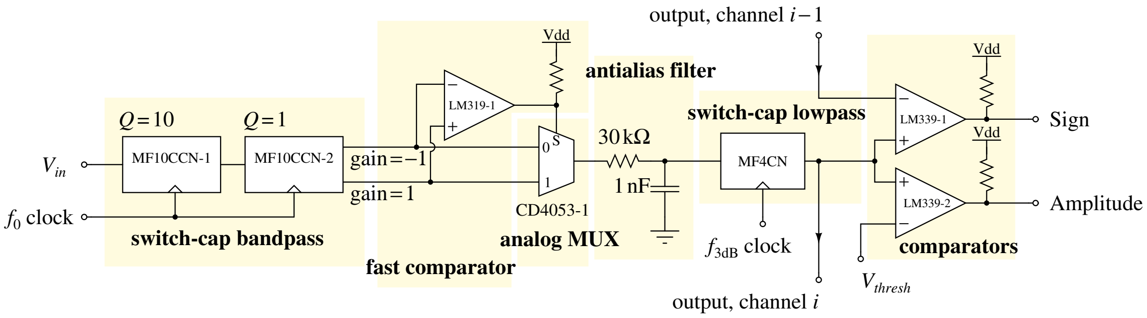


Figure 4.34: Switched-capacitor bandpass filterbank, single channel.

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

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

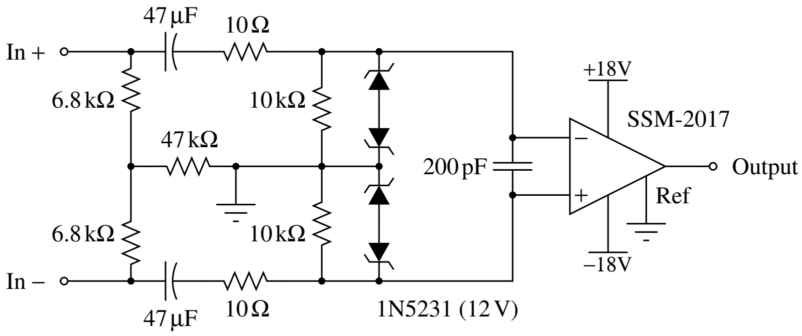


Figure 4.35: Audio input circuit for the frontend system.

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

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

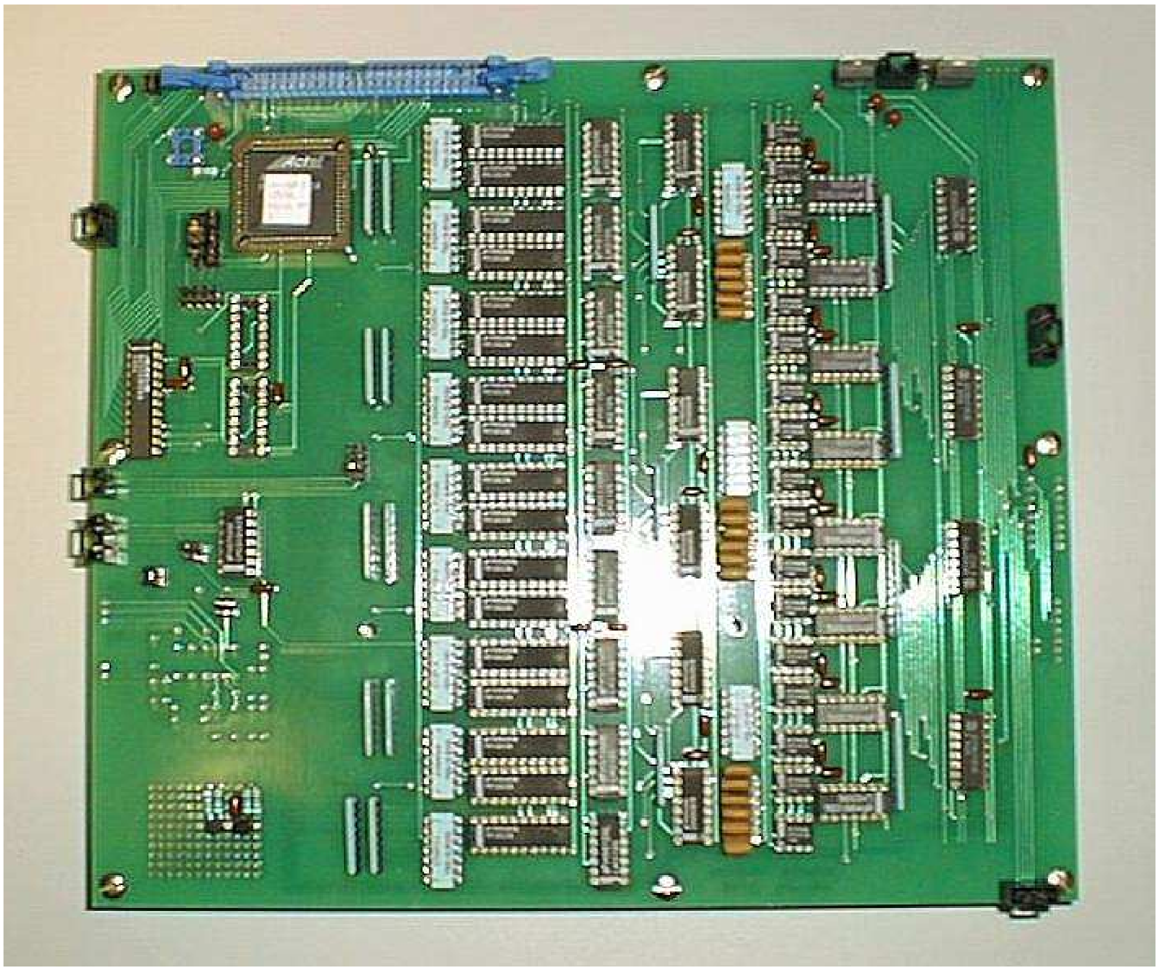


Figure 4.36: One board of the switched capacitor frontend filterbank, bandpass filtering and encoding the audio input into sixteen parallel channels.

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

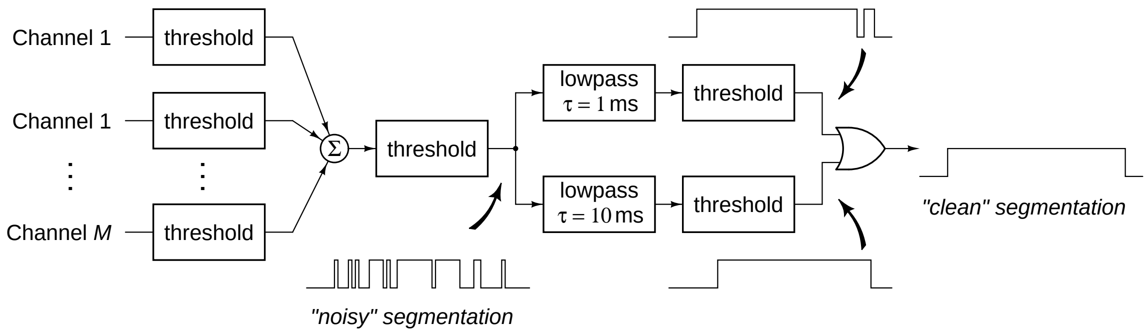


Figure 4.37: A mixed-signal method for detecting the onset of a transient event from the output of an analog frontend filterbank.

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

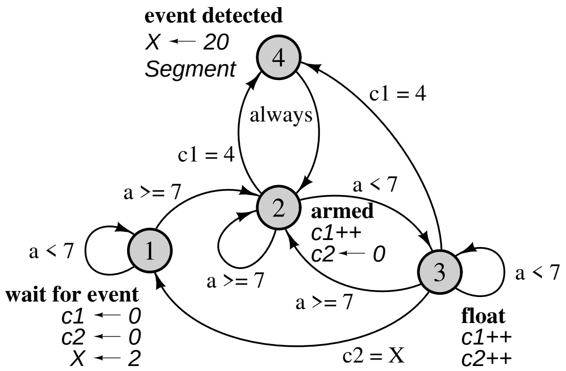


Figure 4.38: A simple finite state machine which detects the onset of transient events in the trinary-valued frontend filterbank output.

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

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

2. Threshold the template values using the relation

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

4.9.3 Experimental Results

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

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

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

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

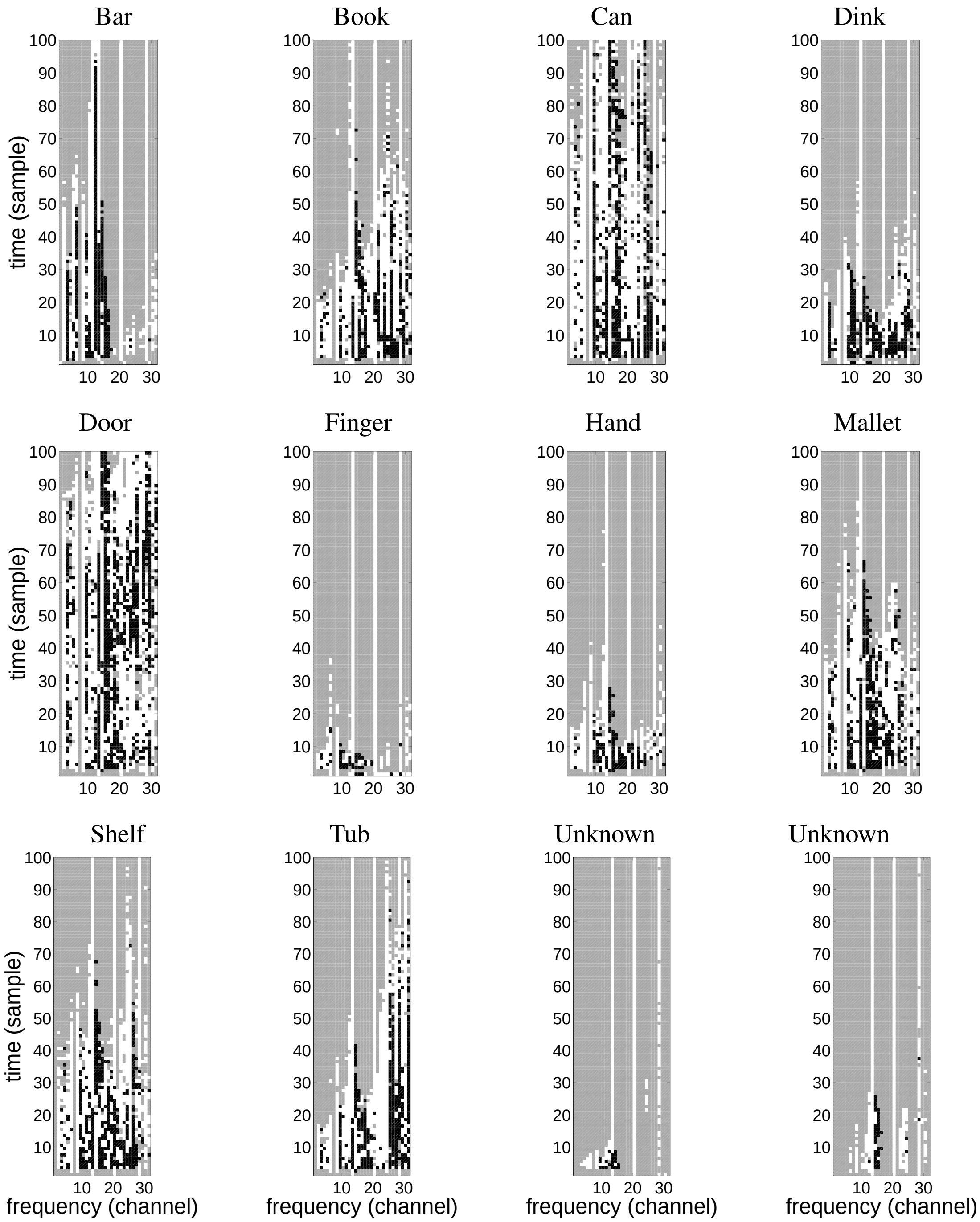


Figure 4.39: Templates for twelve transient classes, determined from data obtained from the frontend hardware. Frequency is on the x-axis, with the lowest frequency channel on the right, and time is on the y-axis, with the transient onset at the bottom of the template. Colors correspond to trinary values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

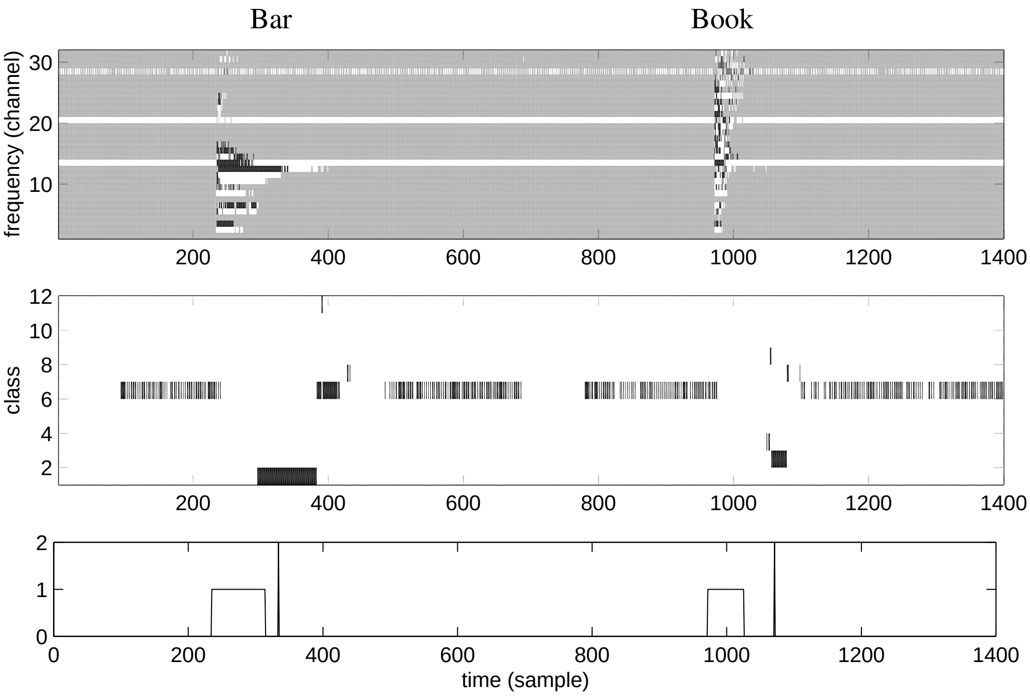


Figure 4.40: System input and output for sound "bar" (left) and "book" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

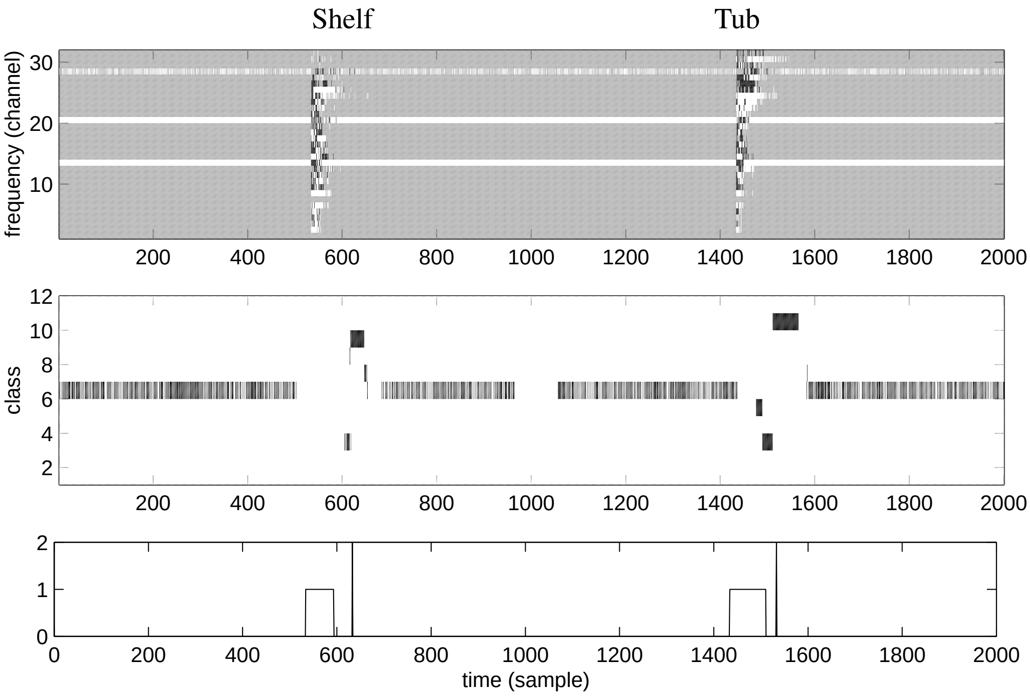


Figure 4.41: System input and output for sound "shelf" (left) and "tub" (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub
Bar	$\boxed{20}$	0	0	0	0	0	0	0	0	0
Book	0	$\boxed{20}$	0	0	0	0	0	0	0	0
Can	0	0	$\boxed{20}$	0	0	0	0	0	0	0
Dink	0	0	0	$\boxed{20}$	0	0	0	0	0	0
Door	0	0	0	0	$\boxed{20}$	0	0	0	0	0
Finger	0	0	0	0	0	$\boxed{20}$	0	0	0	0
Hand	0	0	0	0	0	1	$\boxed{18}$	0	0	0
Mallet	0	0	1	0	0	0	0	$\boxed{19}$	0	0
Shelf	0	0	0	0	0	0	0	0	$\boxed{20}$	0
Tub	0	0	0	0	0	0	0	0	0	$\boxed{20}$

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

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

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

Event	Bar	Book	Can	Dink	Door	Finger	Hand	Mallet	Shelf	Tub
Bar	$\boxed{20}$	0	0	0	0	0	0	0	0	0
Book	0	$\boxed{20}$	0	0	0	0	0	0	0	0
Can	0	0	$\boxed{19}$	0	0	0	0	1	0	0
Dink	0	0	0	$\boxed{18}$	0	0	3	0	0	0
Door	0	0	4	0	$\boxed{20}$	13	6	4	0	0
Finger	0	0	0	0	0	$\boxed{19}$	1	0	0	0
Hand	0	0	0	0	0	5	$\boxed{14}$	0	0	0
Mallet	0	0	1	0	0	0	1	$\boxed{17}$	1	0
Shelf	0	0	0	0	0	0	0	0	$\boxed{19}$	0
Tub	0	0	0	0	0	0	0	0	0	$\boxed{20}$

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.

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

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

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

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

5.2 Average-Value Templates

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

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

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

To give a concrete example: The sound of a door closing (one of the transient classes in our recorded set) consists of two separate events: The sound of the door latch striking the metal plate, followed by the sound of the door itself bumping against the frame. The result of the two sounds occurring in sequence is the familiar “ker-chunk” of a closing door. Either of the two sounds, if separated from the other, fail to sound like a door. Clearly the sound that we recognize as that of a door closing is as much dependent on the juxtaposition of these two events as it is on the events themselves. On the other hand, the timing between the two events depends on the radial velocity of the door, which may vary considerably from one event to the next. Suppose the recorded transient examples are in every way exact except for the distance between the door latch and door 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 like a discrete wavelet transform where information is actively pushed from one dilation into the next, but where each filtering function is optimized for a particular transient event at each dilation (timescale). Those functions may be approximated by template correlators to give a high level of flexibility to function selection, but the templates no longer encode any function as simple as an ensemble average. What they encode are functions which extract features at each dilation which, collectively over all dilations, optimally isolate one acoustic class from all other classes.

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

5.3 Deterministic Methods: Statistical Component Analysis

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

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

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

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

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

The weight update rule is [72]:

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

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

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

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

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

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

5.5 Heuristic Methods (Unnikrishnan/Hopfield)

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

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

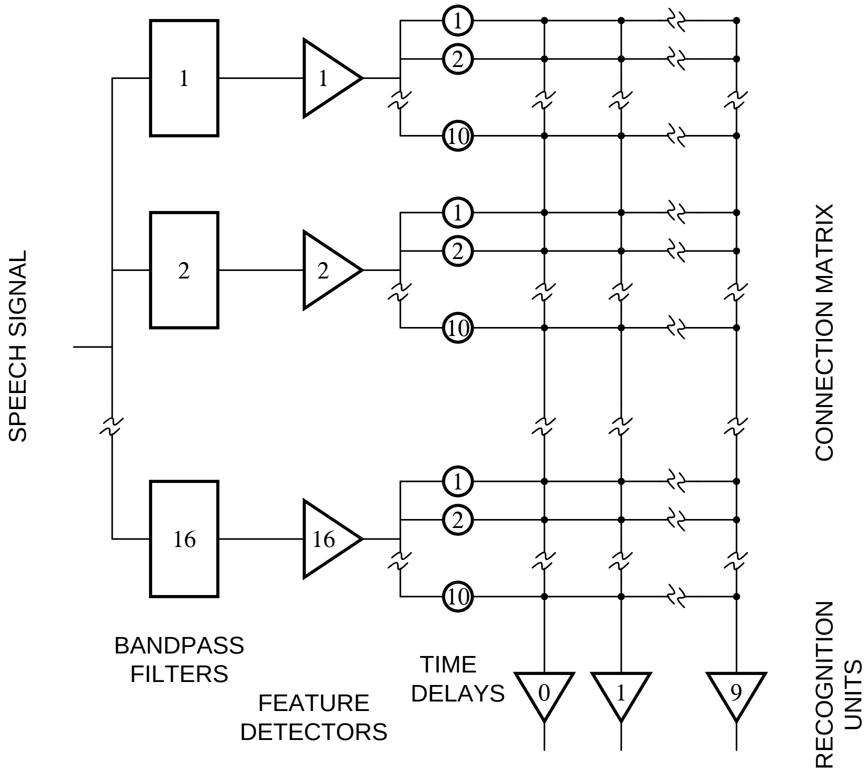


Figure 5.1: Unnikrishnan, Hopfield, and Tank correlator architecture, block diagram.

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)):

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

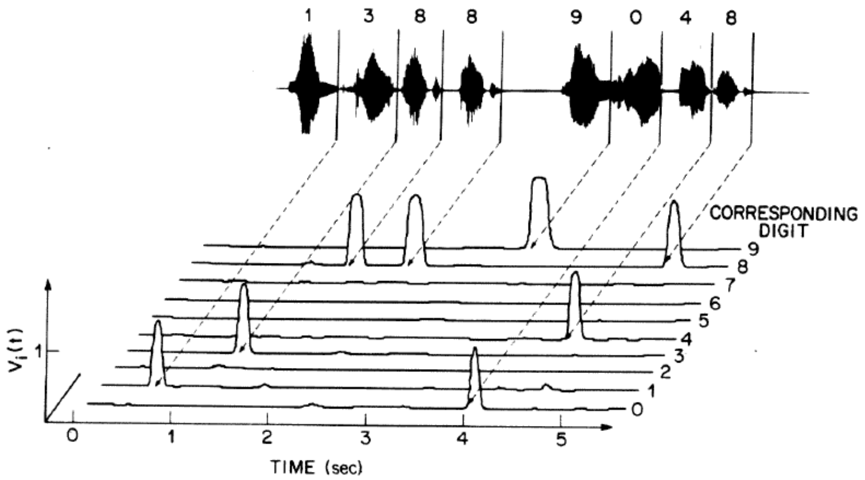


Figure 5.2: Unnikrishnan *et al.* correlator simulation results on isolated digit recognition.

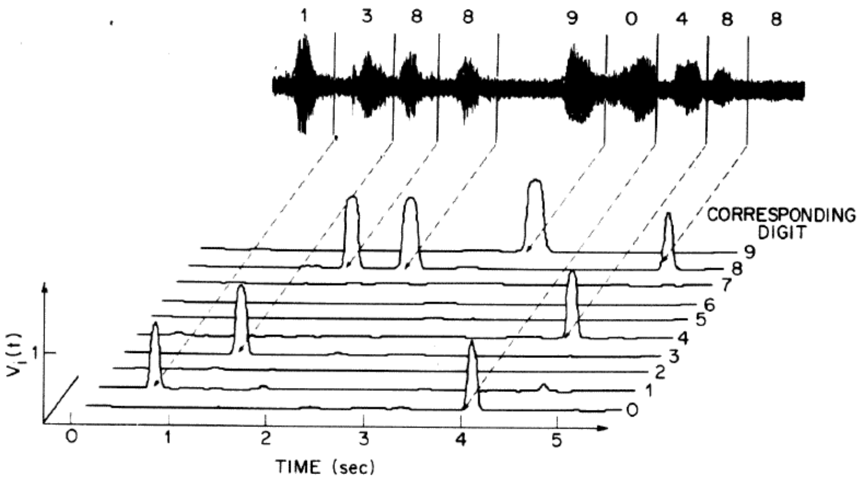


Figure 5.3: Response of architecture in simulation to the same input as Figure 5.2 but with white noise added to the input. The system remains robust in the presence of noise.

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

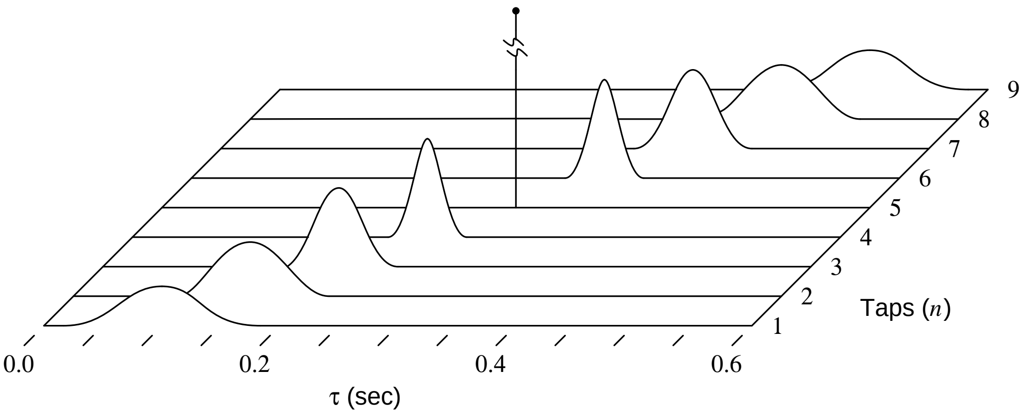


Figure 5.4: Gaussian kernels which diffuse the input in the Unnikrishnan *et al.* model.

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

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

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

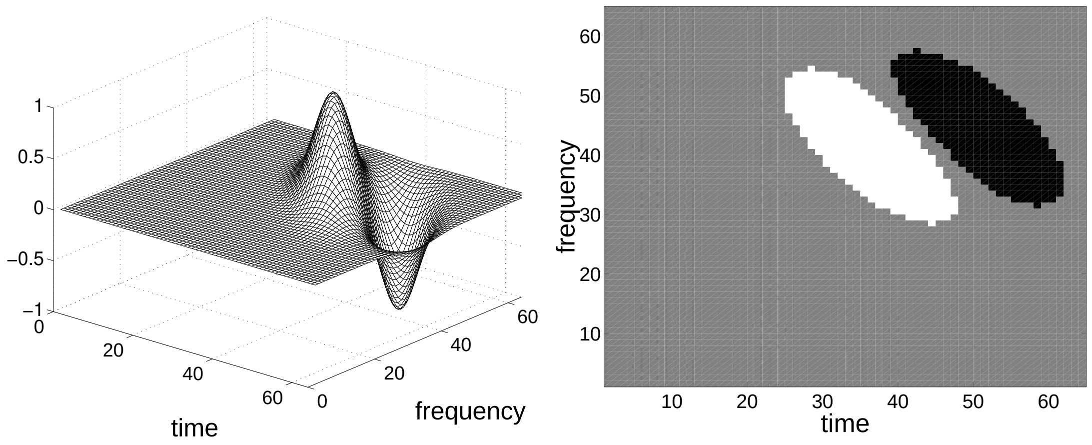


Figure 5.5: Example of a biologically-inspired template encoding a falling tone of specific frequency and rate. *left:* Continuous-valued encoding. *right:* Trinary encoding.

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

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

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

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

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

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

Conclusion

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

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

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

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,

$$\begin{aligned} I_2 - I_1 &= \Delta I = I_0 e^{-V_s/V_t} \left(e^{(\kappa V_2)/V_t} - e^{(\kappa V_1)/Vt} \right) && \text{(A.2)} \\ I_2 + I_1 &= I_b = I_0 e^{-V_s/V_t} \left(e^{(\kappa V_2)/V_t} + e^{(\kappa V_1)/Vt} \right) && \text{(A.3)} \end{aligned}$$

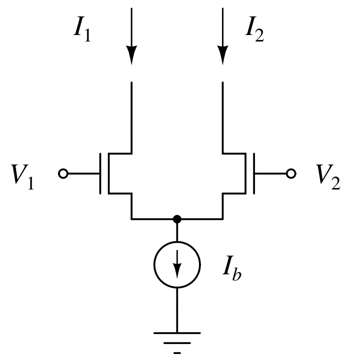


Figure A.1: Simple differential pair transconductance amplifier.

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

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

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

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

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

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

such that a linear approximation to the output is

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

Linear error is

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

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

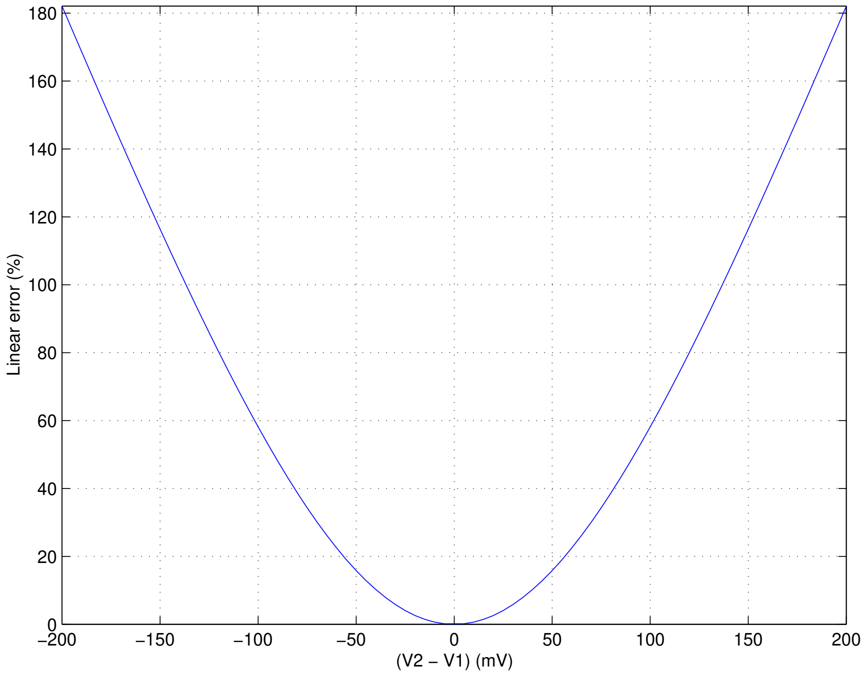


Figure A.2: Linear error in a simple transconductance amplifier as a function of the differential input voltage.

Appendix B

MATLAB Code for Sine Sequence Generation

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

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

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

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

norm = 2 * pi / period;
```

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

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

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

%--------------------------------------------------------------
% Derived values

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

% Initialize bit sequence and perturbative sequence

bits = ones(nBits, 1);

% Initial values for iteration

bestSNR = -100;         % in dB, conservative starting point
iter = 0;

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

while 1,
   iter = iter + 1;

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

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

   maxbits = bits;
```

```
for pattern = 1: maxPattern,

   % Generate the perturbed bit sequence

   bitsP = bits;

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

   % Generate the reverse of the bit sequence bitsP()

   bRev = bitsP(length(bitsP):-1:1);

   % Generate the full sequence from its quarter parts and
   % compute the SNR after filtering with Atten().

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

   % "Signal" is the Fundamental harmonic
   % "Noise" is all the other harmonics

   Signal = Amp(2);
   Noise = sum([Amp(1) Amp(3:length(Amp))]);

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

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

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

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

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

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

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.

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

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

where

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

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

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

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

If we let

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

then

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

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

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.

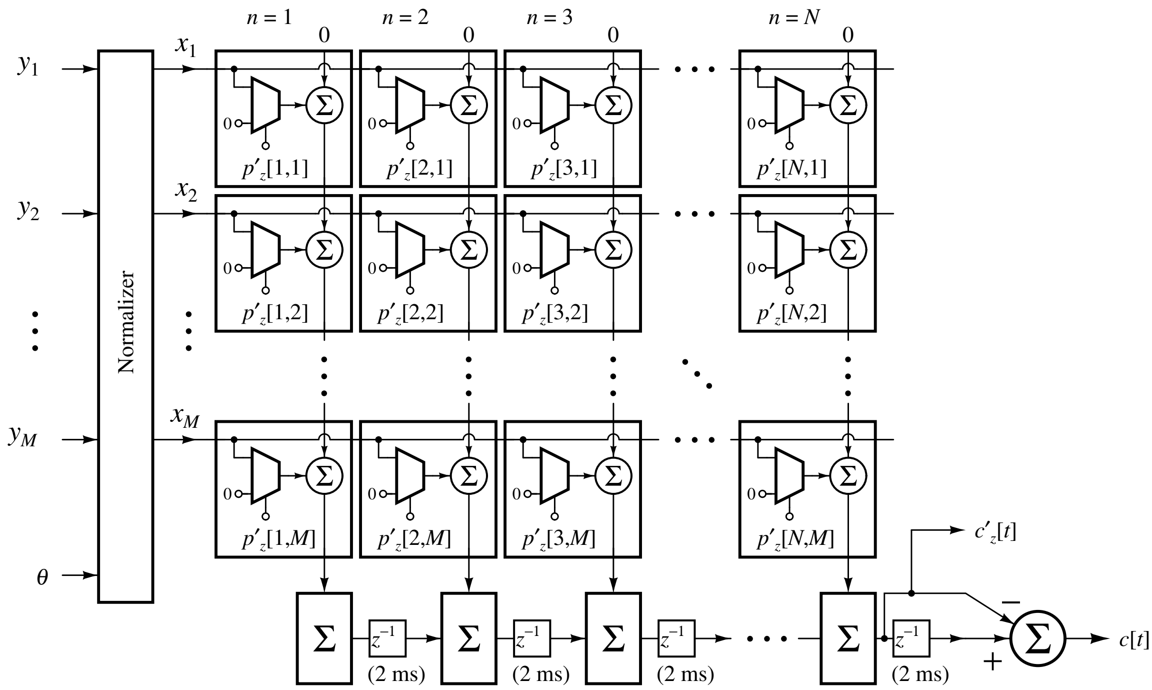


Figure C.1: Block diagram of the temporal current correlator, using time-differencing operations on the input.

C.1 Proof of validity of the pipelined architecture

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

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

$$\begin{aligned}
q[n,t] &= q[n-1,t-1] + \sum_{m=1}^{M} x'[t,m]\, {p'}_z[n,m] \qquad \forall n \neq 1 && \text{(C.7)}\\
q[1,t] &= \sum_{m=1}^{M} x'[t,m]\, {p'}_z[1,m] && \text{(C.8)}\\
c_z[t] &= q[N,t-1] && \text{(C.9)}
\end{aligned}$$

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

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

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

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

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 $n \to N-n+1$ (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.

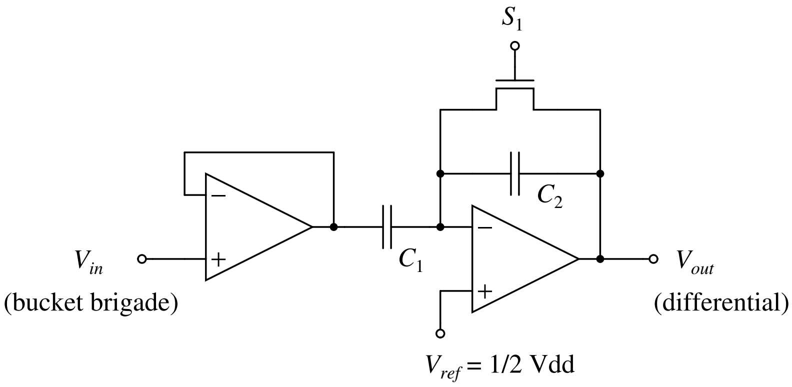


Figure C.2: Switch-cap time-differencing circuit.

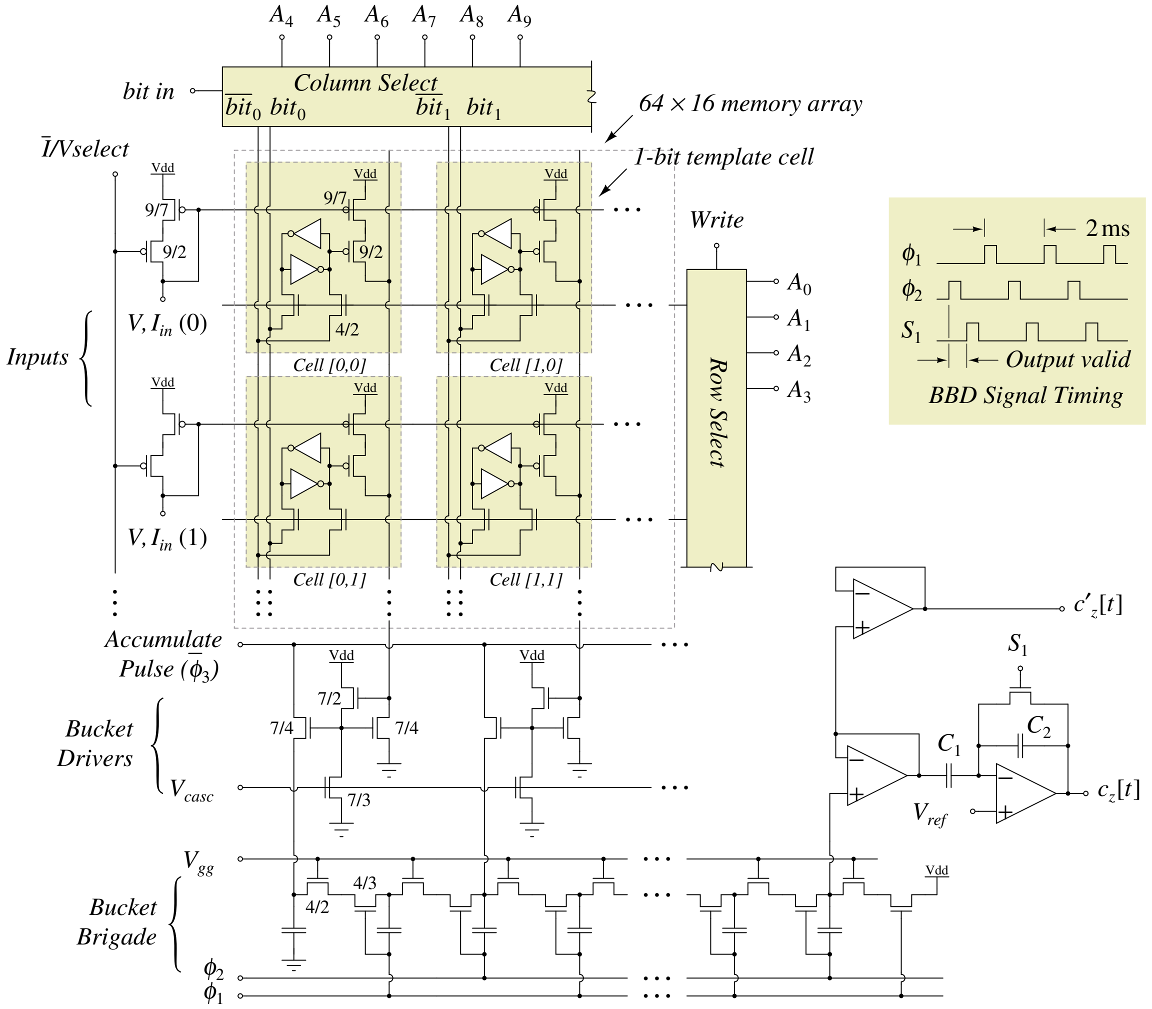


Figure C.3: Time-differencing correlation architecture.

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))
```

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

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

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

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

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

float minout[NFILTERS], maxout[NFILTERS];

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

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

void makefilters()
{
   short nfilt;
   float X, Y, x1;
   double nn = 1.0 / (double) (NFILTERS - 1);
   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 */
   /* solution of this equation, so we'll start above 1   */
   /* and iterate downward to the solution.               */

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

   Y = STEPFREQ - BOTFREQ * (X - 1);

   filterfreq[0] = BOTFREQ;

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

/*------------------------------------------------------*/
/* Computer filter coefficients for each frequency band */
/*------------------------------------------------------*/

void filtinit()
{
   short nfilt;
   float sfreq = SFREQ;
   float Q, denominv, tau, tausq, tauQ;

   makefilters();

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

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

        /* compute filter cutoff and Q value */

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

        /* compute bandpass filter coefficients */

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

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

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

        /* initialize filter endpoints */

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

        /* Initialize envelope-detection parameters */

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

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

void filterbank()
{
    short nfilt;
```

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

      /* 1st bandpass filter */

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

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

      /* 2nd (cascaded) bandpass filter */

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

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

makesmooth()
{
   short nfilt;
   float ntau;

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

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

      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];
```

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

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

      lpfiltdata[nfilt][itime] = abcs[4][nfilt]
                * (lfiltdata[2][nfilt] +
                lfiltdata[3][nfilt]) + abcs[5][nfilt] *
                lpfiltdata[nfilt][itimen1];
   }
}

/*-------------------------------------------------------*/
/* Simulate peak-peak detector in hardware version of ATP */
/*-------------------------------------------------------*/

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

peakdet()
{

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

      /* maximum peak find */

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

      /* minimum peak finder */

      minout[nfilt] += ntau;
      if (filtdata[nfilt][itime] < minout[nfilt])
         minout[nfilt] = filtdata[nfilt][itime];

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

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;
}
```

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

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

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

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

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

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

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

```

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

            dot += (t1 > t0) ? d1 : d0;

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

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

float correlate_new(float *template, float *data,
        int numchans, long samples)
{
    float dot = 0.0;
    float t0, t1, t2, d0, d1, d2;
    int i, j;

    for(j = 0; j < samples - 1; j++) {
        for(i = 0; i < numchans - 2; i++) {

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

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

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

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

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

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

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

E.1 Commentary

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

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

Appendix F

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

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

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

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

int channels, timebins, filters, classes;

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

int quantize;

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

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

```
   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;
```

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

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

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

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

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

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

#define alpha 0.1
#define ahpla 0.9
```

```
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];
```

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

            newgain = alpha
                 * (-log(nsum / dsum) / cgains[i][i])
                 + ahpla * nngain;
            ndiff = fabs(newgain - nngain);
            nngain = newgain;
            bcount++;
         } while(ndiff > 0.0001 && bcount < 10000);
         if (bcount == 10000) break;

         gains[i] = nngain;
      }
      fprintf(stdout, "\n");

      /* Renormalize the gain coefficients */

      gainnorm();

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

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

   } while (nndiff > 1e-8);

   printgains(cgains);

   /* free up allocated memory */

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

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

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.

Bibliography

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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.