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.

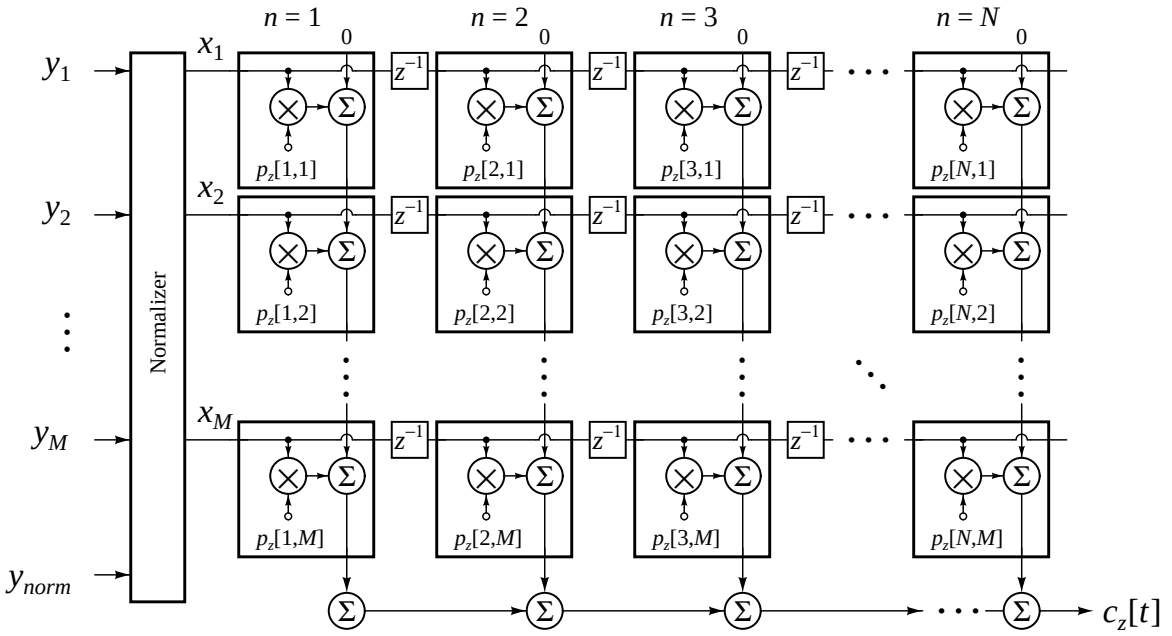

Figure 4.1: Template correlator, as a direct implementation of the baseline algorithm, Equation (4.1).

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

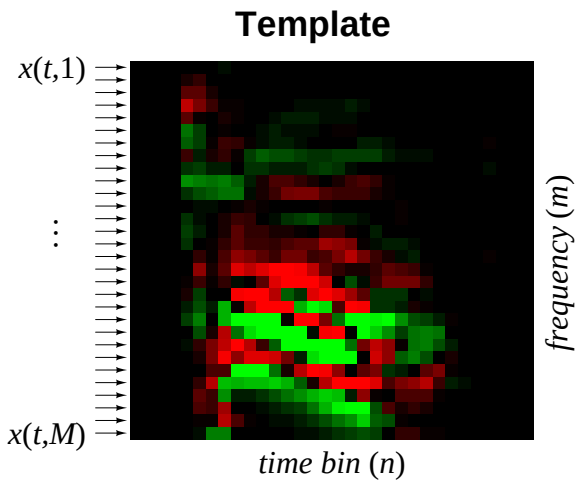

Figure 4.2: Template correlation.

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 pairwise difference of channels with respect to the original.

3. We replace the template values with binary values.

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

4.2.1 Simplifying The Correlation Equation

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

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

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

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

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.

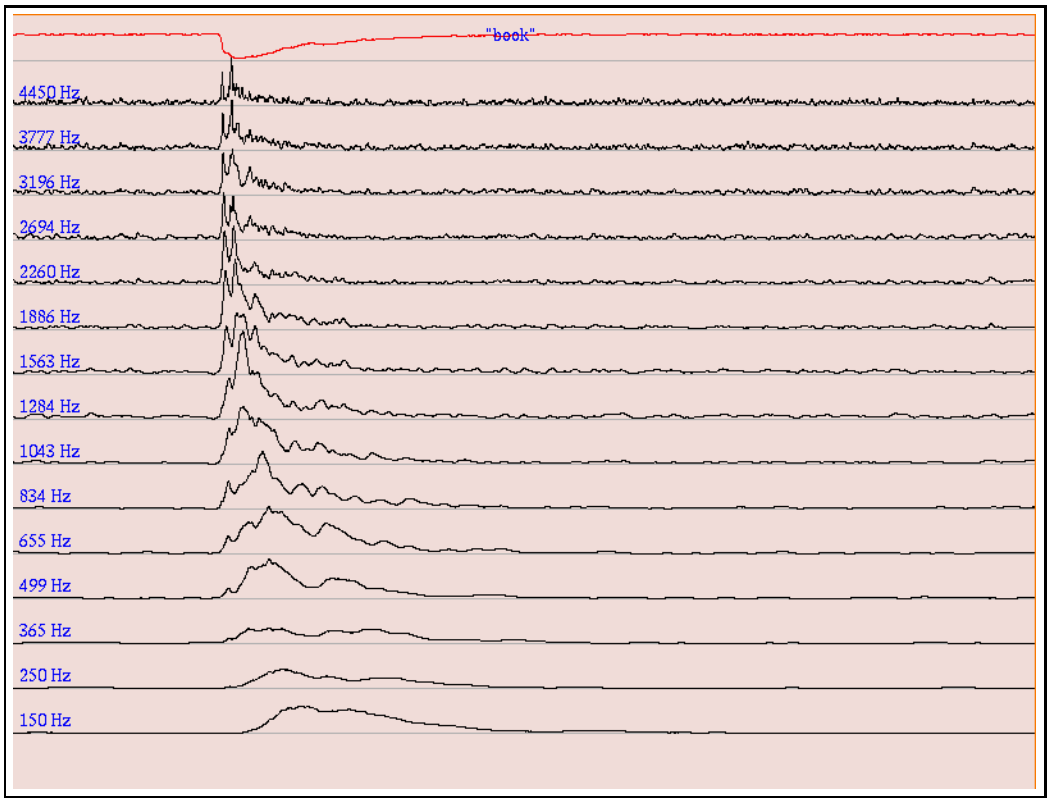

Figure 4.3: Filterbank output after applying an L-1 normalization across all channels. A single channel has been added to the system (top trace), containing the result of a constant value less the instantaneous sum of the remaining channel values.

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

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

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

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

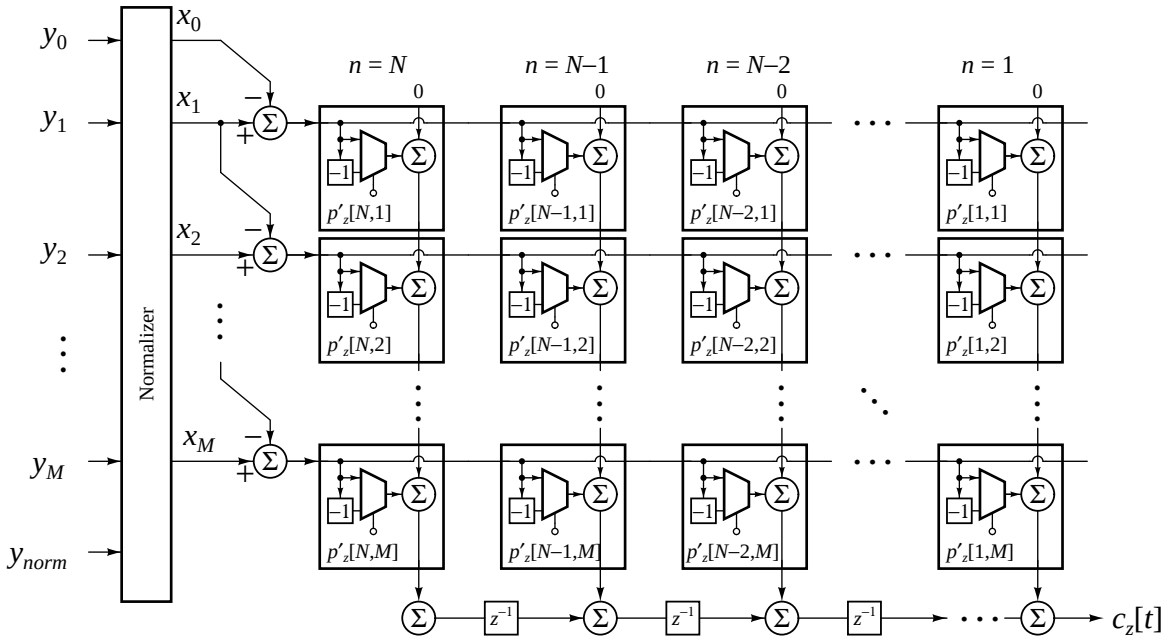

Figure 4.4: Template correlator with pipelined architecture and multiplexors replacing the multipliers.

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.

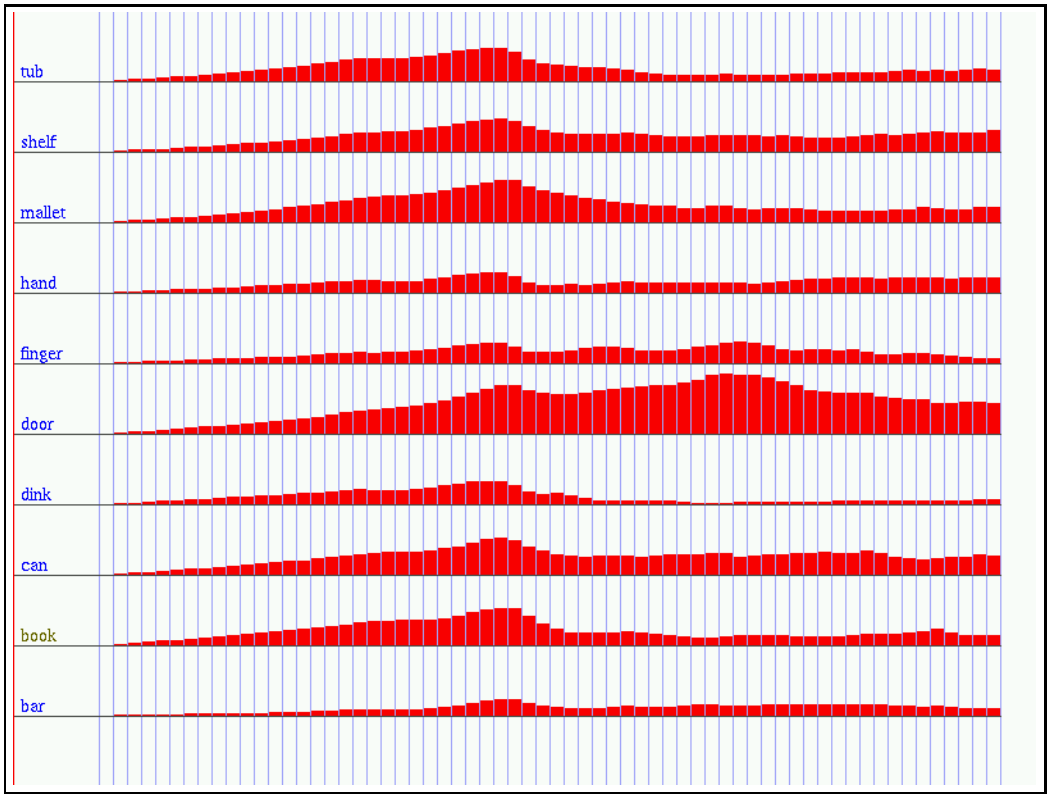

Figure 4.5: Values in the pipelined delay registers in the correlation algorithm simulation.

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.

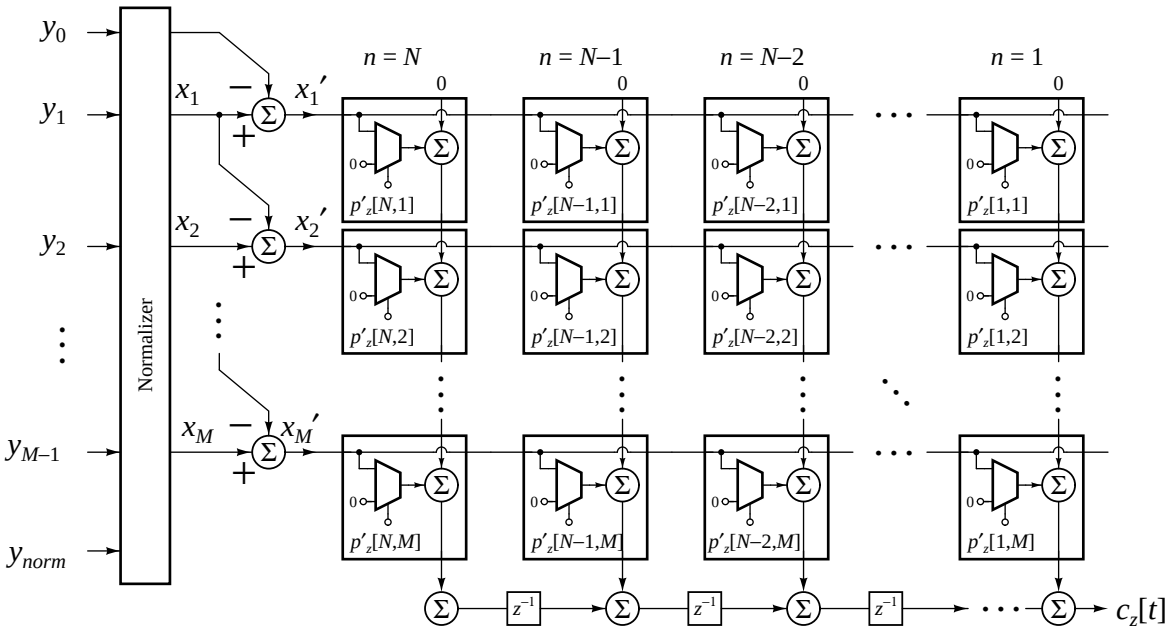

Figure 4.6: Template correlator with [0,1] encoding of template values.

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:

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

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

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

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

Thus, the correlation output becomes:

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

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

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

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

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.

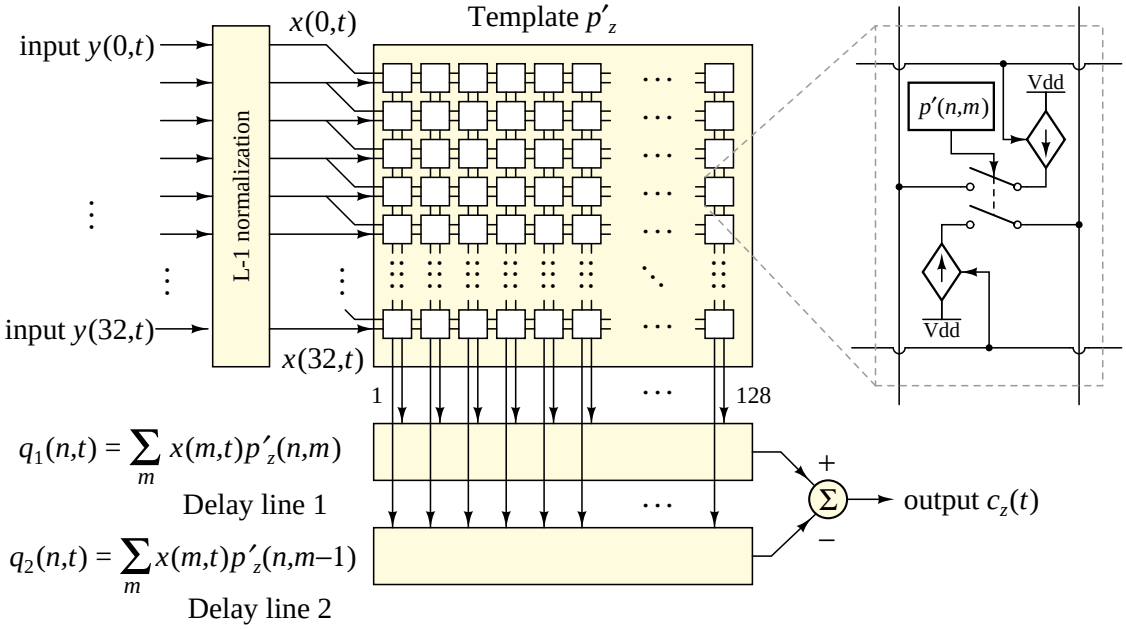

Figure 4.7: Block diagram of the temporal current correlator.

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.

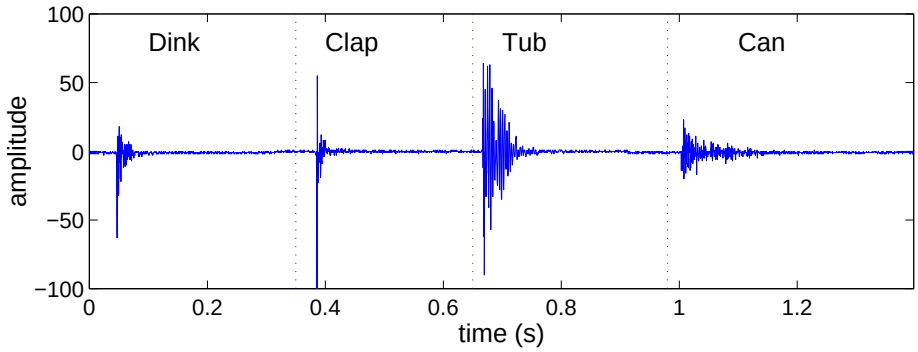

Figure 4.8: Example transient recorded data.

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

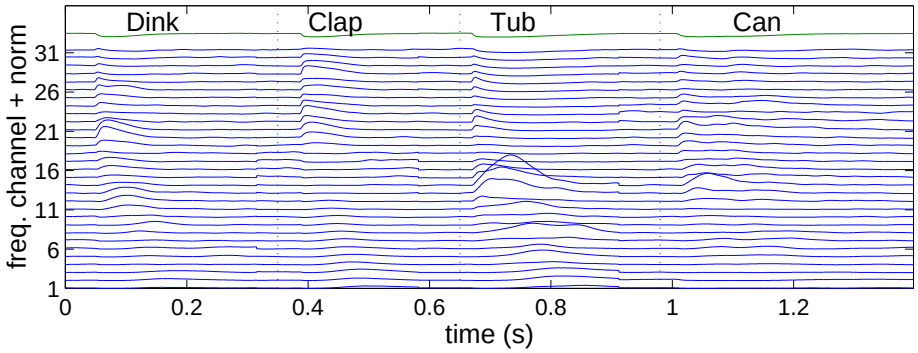

Figure 4.9: Example HEEAR-processed transients.

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.

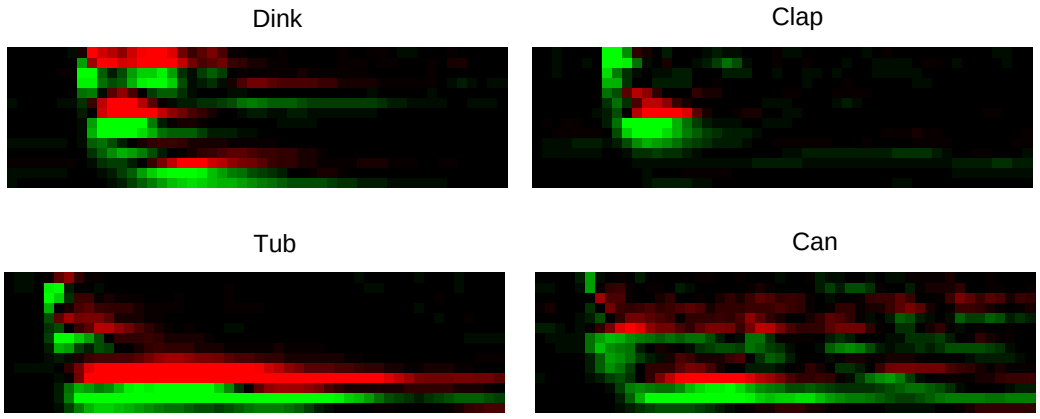

Figure 4.10: Example templates learned by the ATP algorithm.

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

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

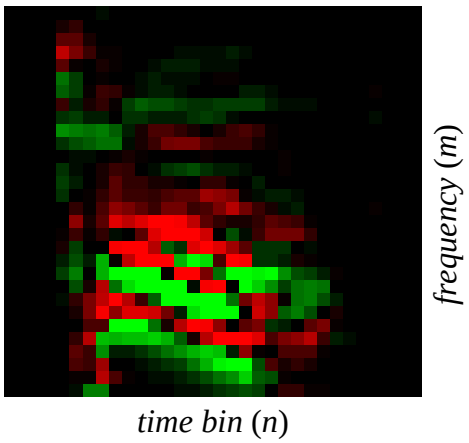

Figure 4.11: Example of a real-valued template.

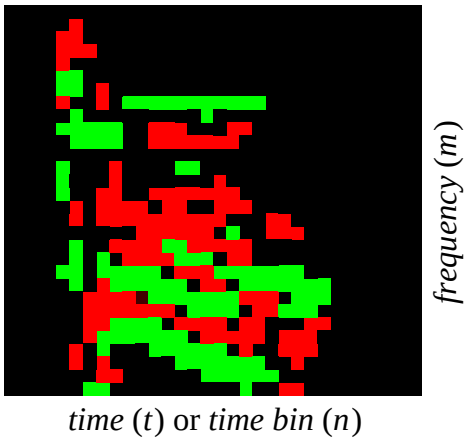

Figure 4.12: The same template reduced to trinary values.

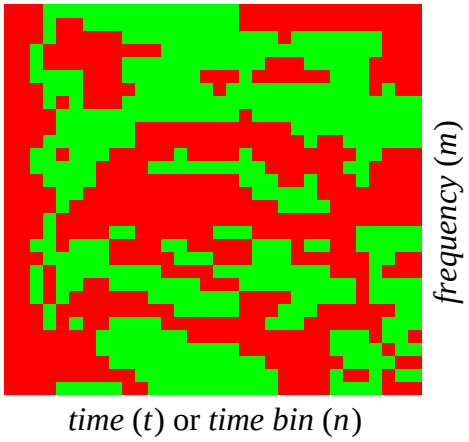

Figure 4.13: The same template reduced to binary values.

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

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

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

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).

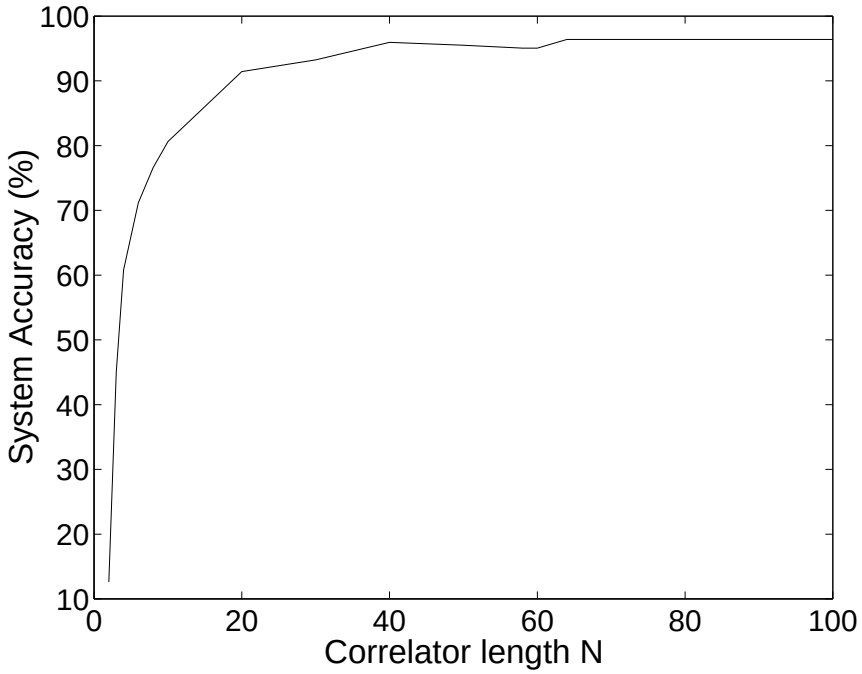

Figure 4.14: Effect of decreasing the number of time-bins.

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.

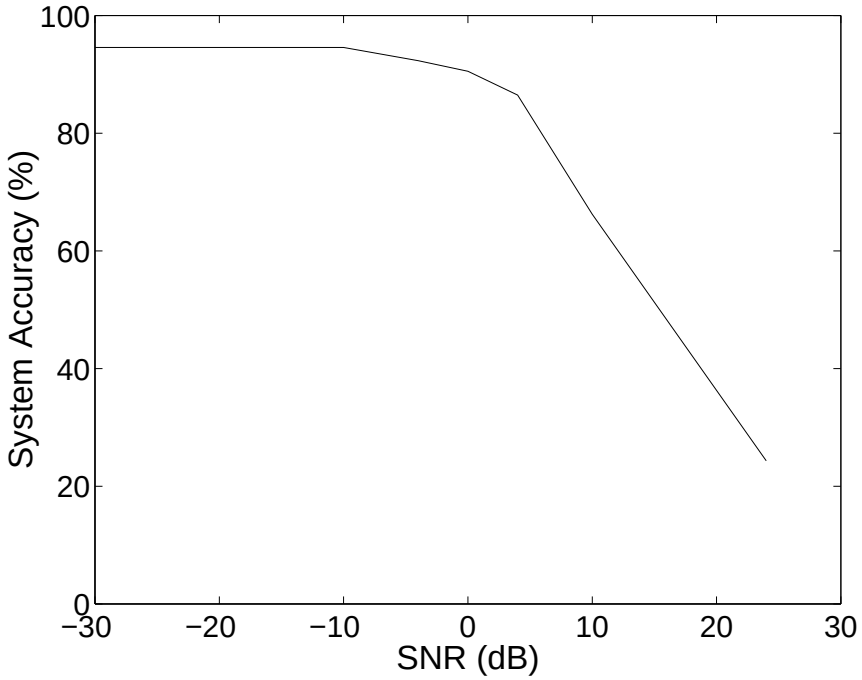

Figure 4.15: Effect of white noise added to the correlator inputs.

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 $Write(m)$ running horizontally across the template array and two lines for the input $bit(n)$ and $\overline{bit}(n)$ values. The programming transistors and corresponding data and control lines appear gray in the figure. The remainder of the core cell represents the bulk of the correlator processing, which has been somewhat complicated by the requirements of the channel differencing operation. The algorithm requires two switches per template memory location $t(n, m)$, one (M_2) which switches the input current of row m to the first (positive) delay line, and one (M_3) which switches the input current of row $m - 1$ to the second (negative) delay line. The positive and negative parts share the same wire carrying the current down to the delay lines by time-multiplexing based on the signal Φ.

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.

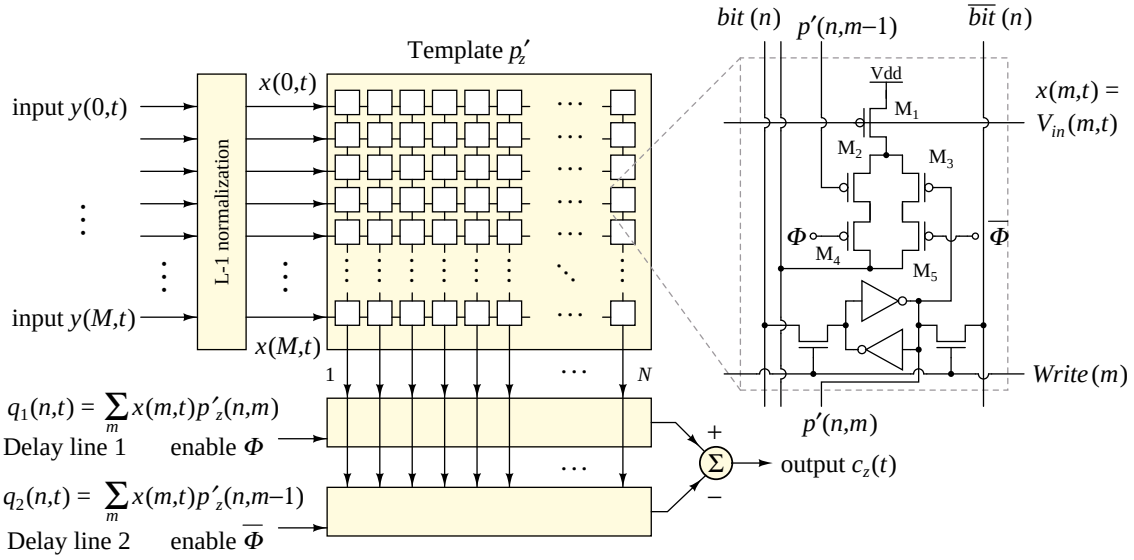

Figure 4.16: Block diagram of the temporal current correlator.

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

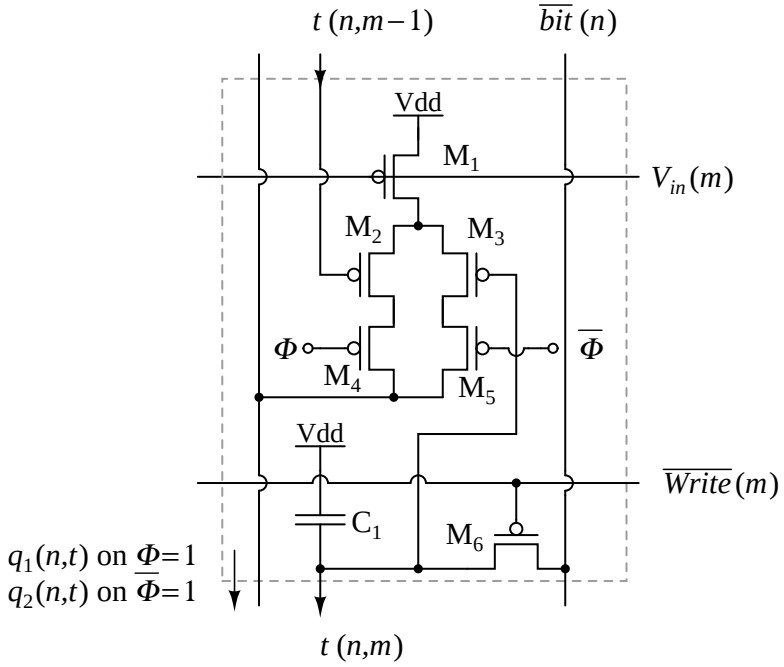

Figure 4.17: Efficient dynamic memory cell for the correlator array.

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

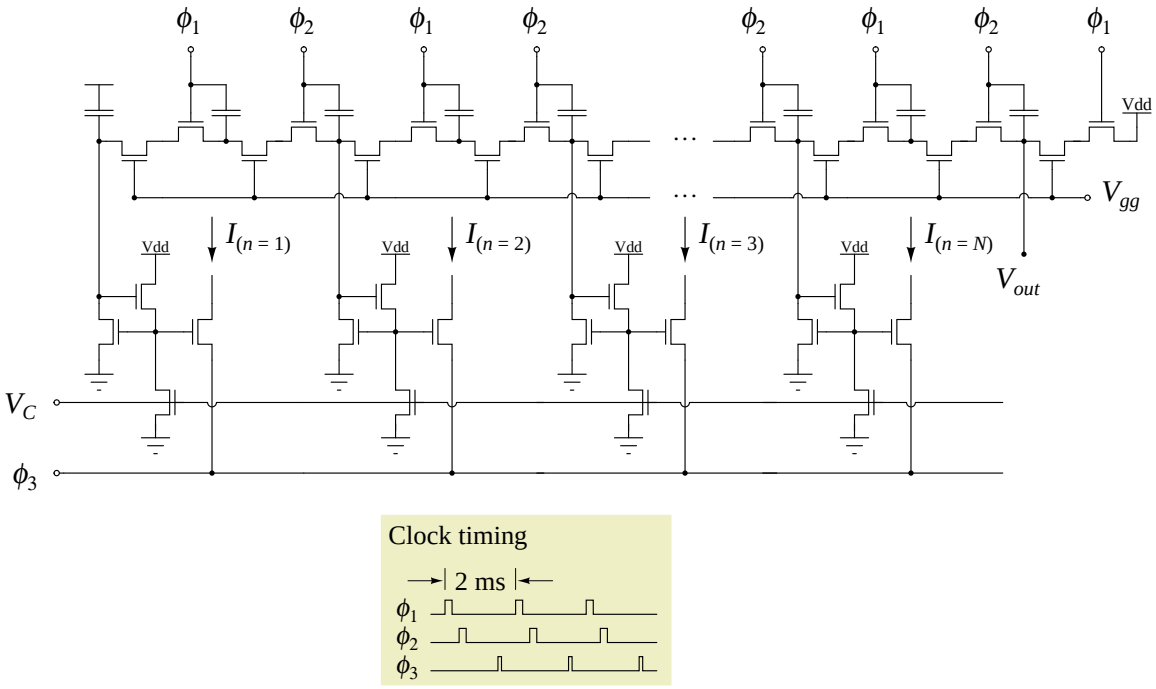

Figure 4.18: Details of the bucket brigade device (BBD).

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

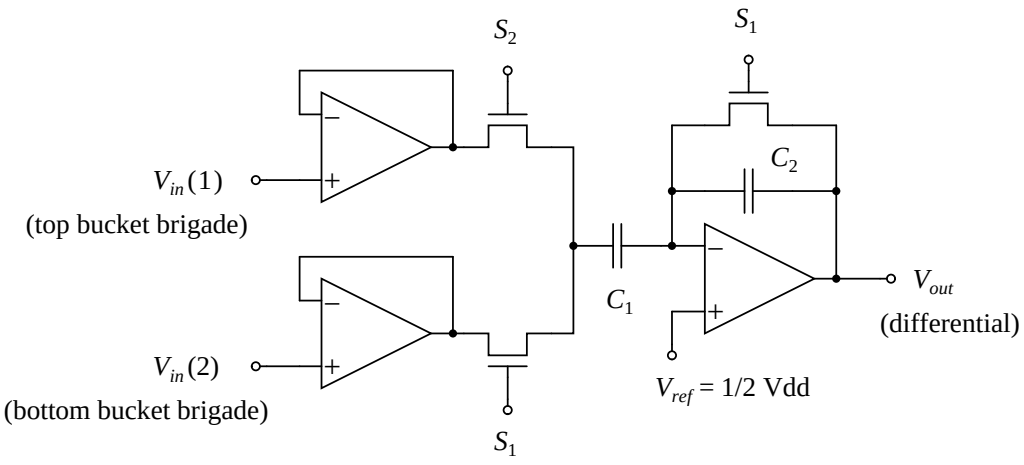

Figure 4.19: A switched capacitor circuit to compute channel differences at the output.

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.

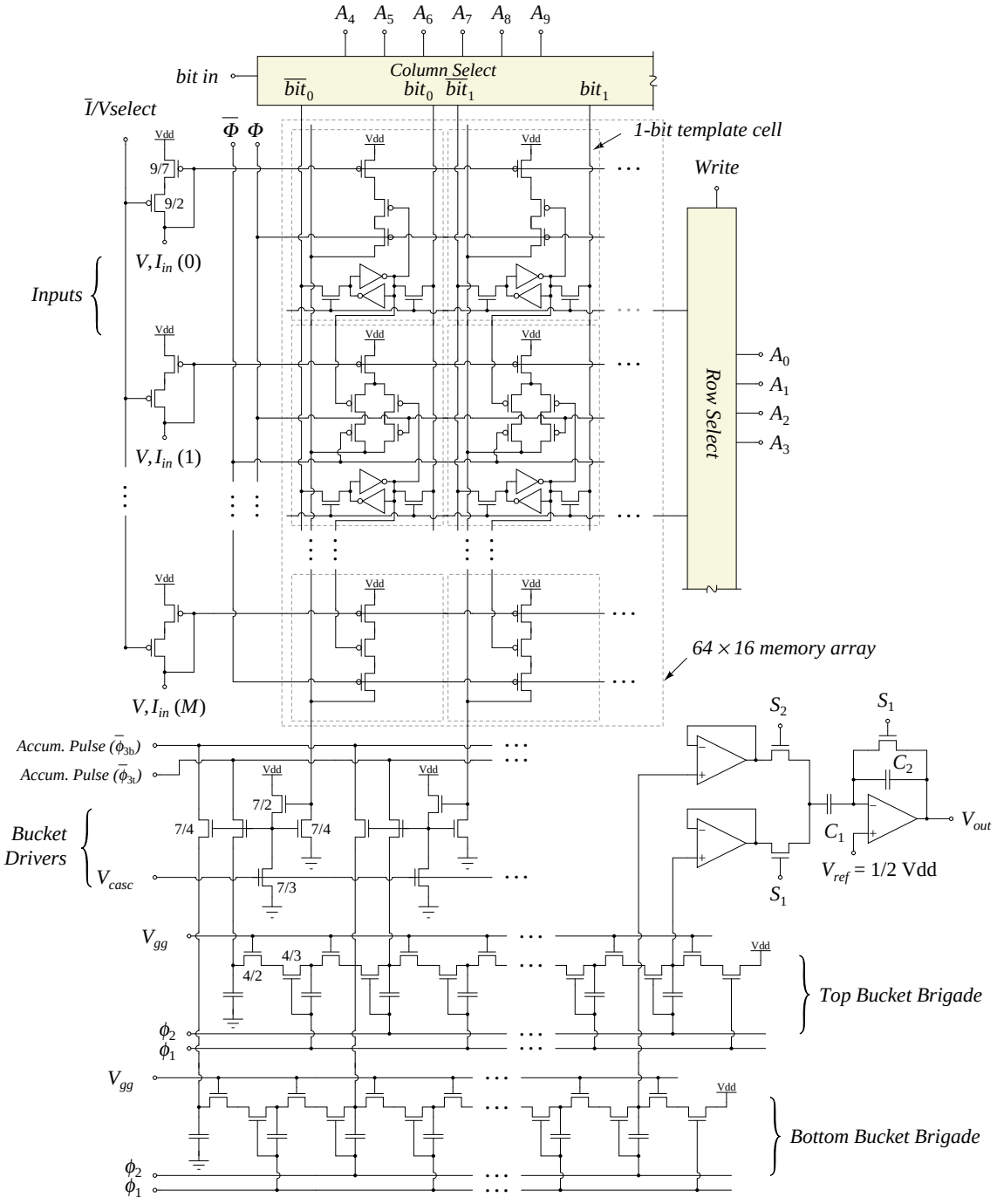

Figure 4.20: The complete correlator array including both bucket brigade devices. Representative MOS device sizes are given as W/L in units of λ = 0.6 µm.

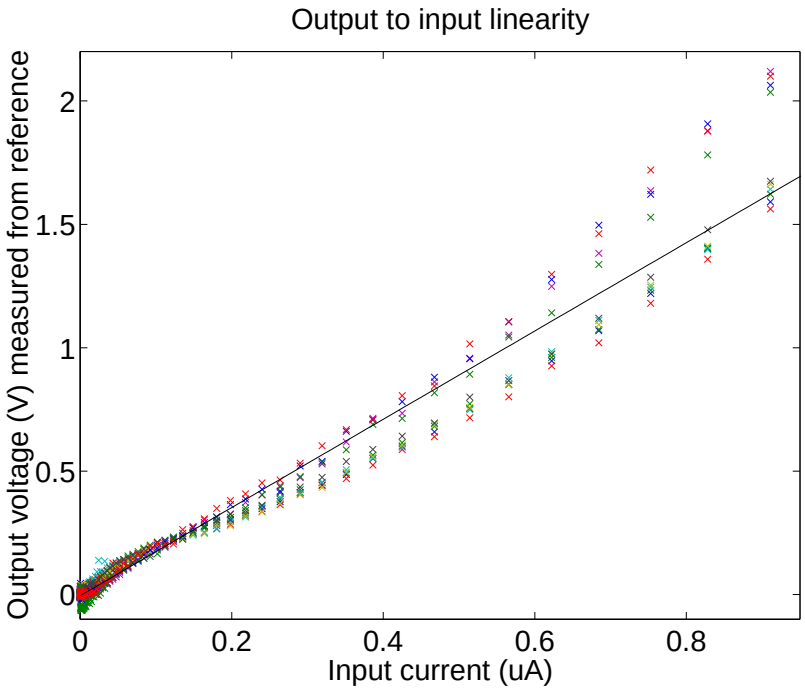

Figure 4.21: Linearity between output voltage and input current.

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.

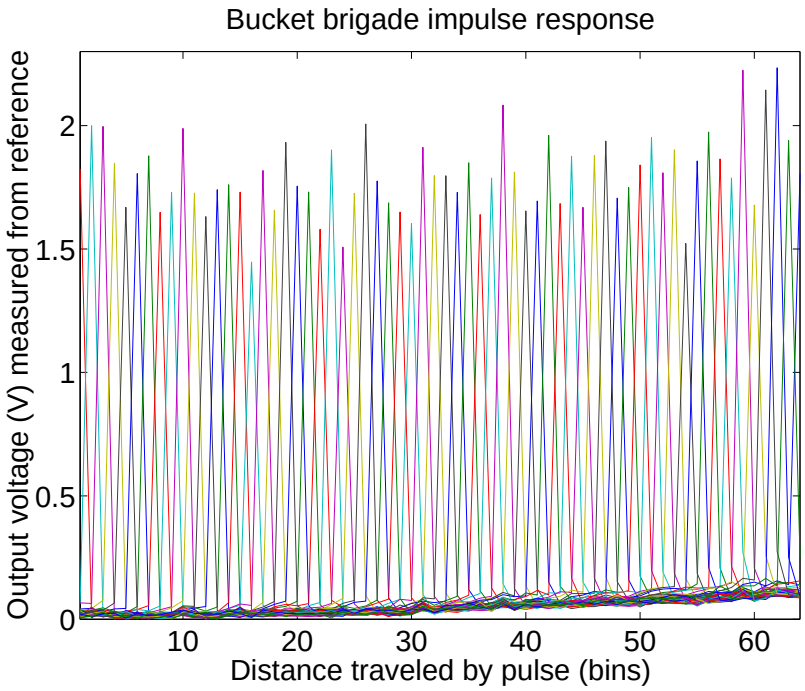

Figure 4.22: Impulse response at each tap of the bucket brigade device.

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

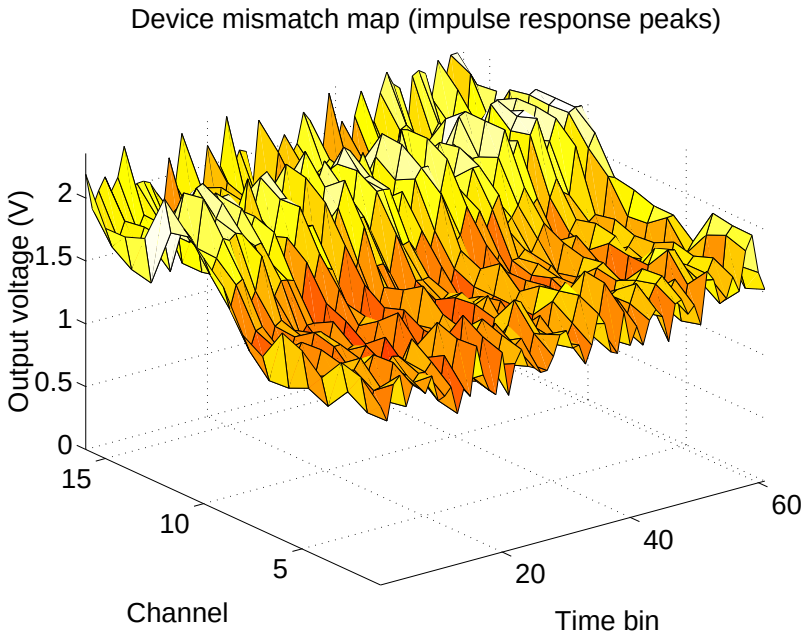

Figure 4.23: Matching between devices throughout the template at 1 µA input.

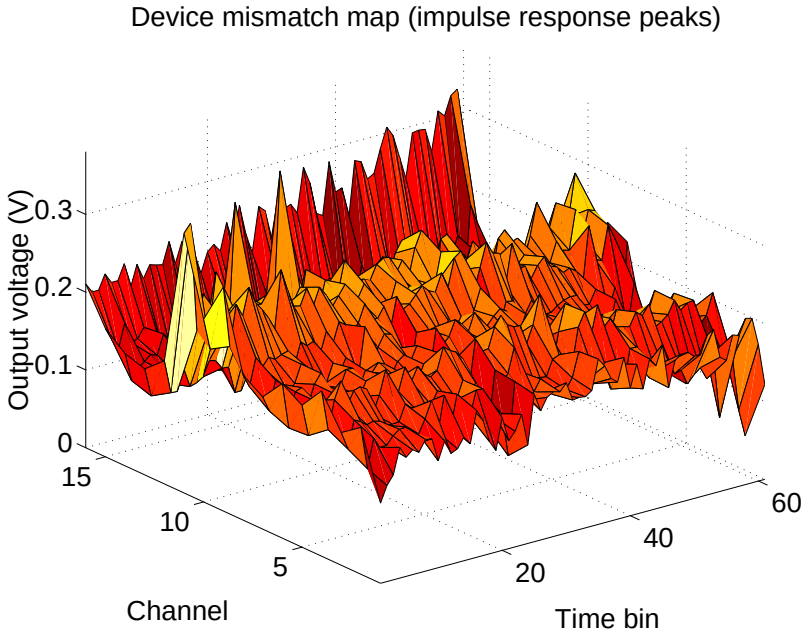

Figure 4.24: Matching between devices throughout the template at 100 nA input.

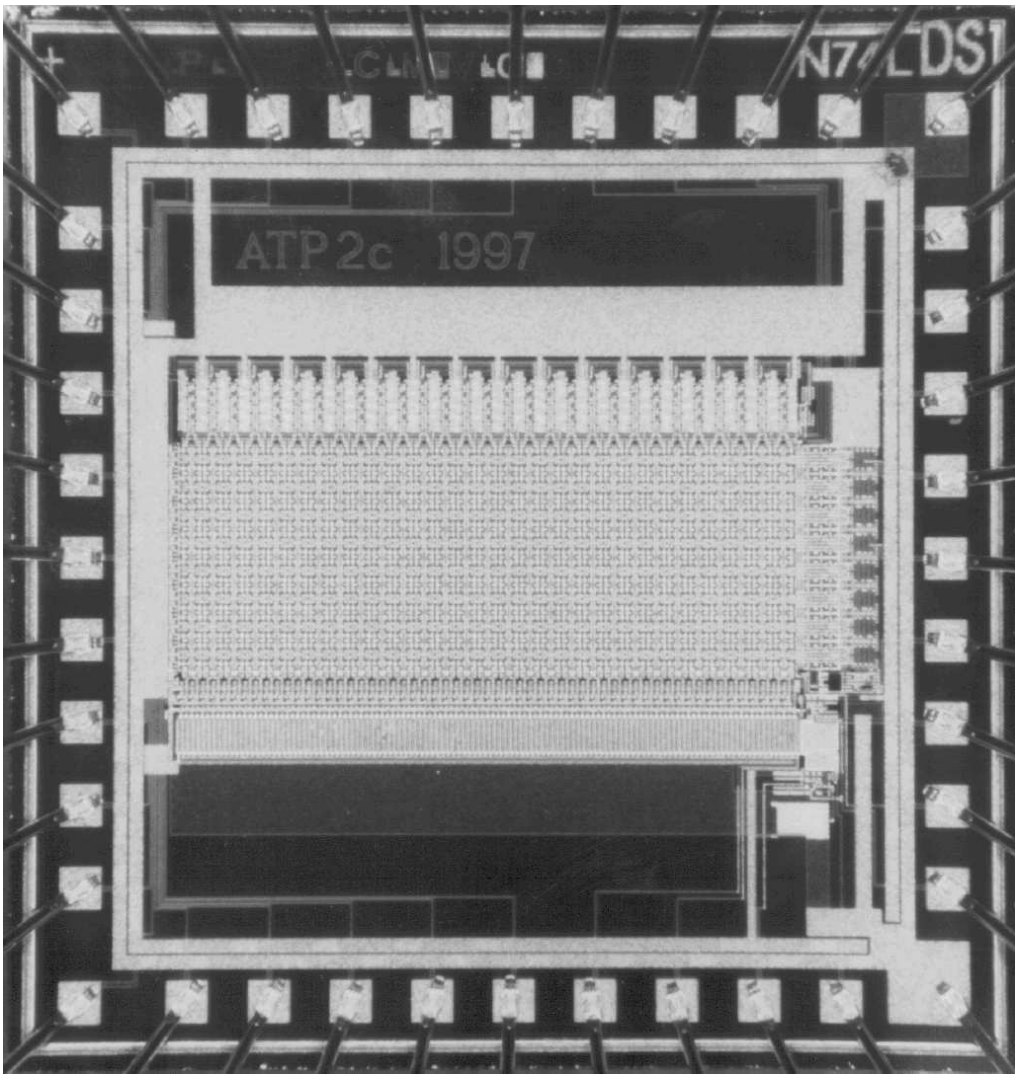

Figure 4.25: Photomicrograph of the acoustic transient correlator, a 2.2 mm $\times$ 2.2 mm die fabricated in a 1.2µm CMOS technology.

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 × 1170 μm per template allows a high integration density.

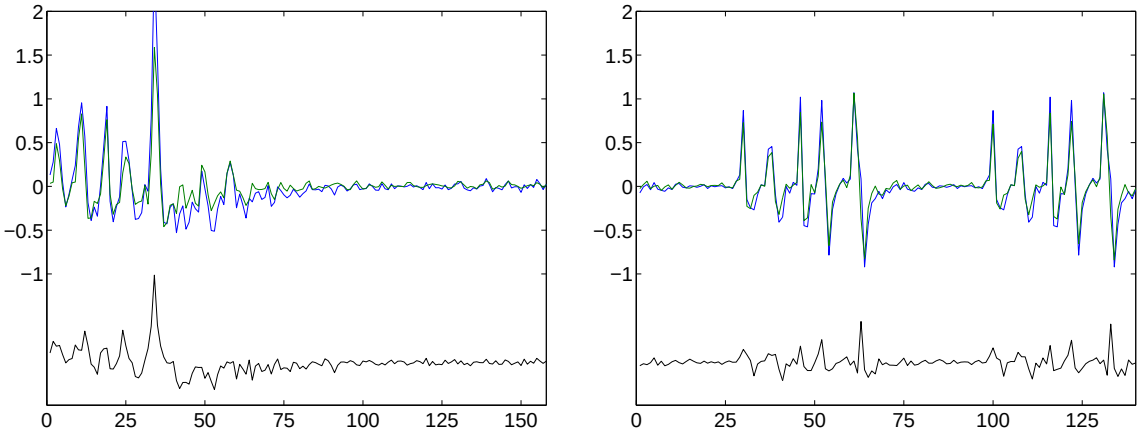

Figure 4.26: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “can” template loaded into chip memory.

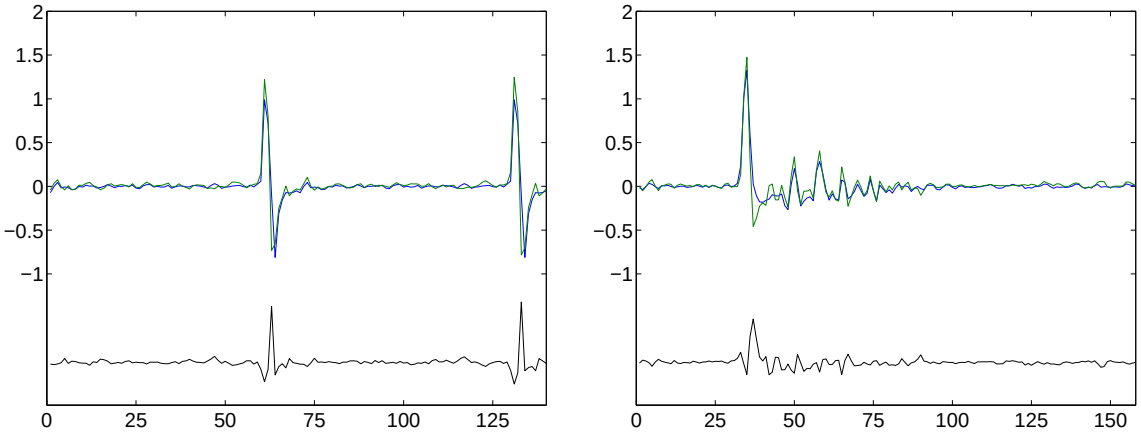

Figure 4.27: Measured correlation of repeated sounds “can” (left) and “snap” (right) with the “snap” template loaded into chip memory.

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.

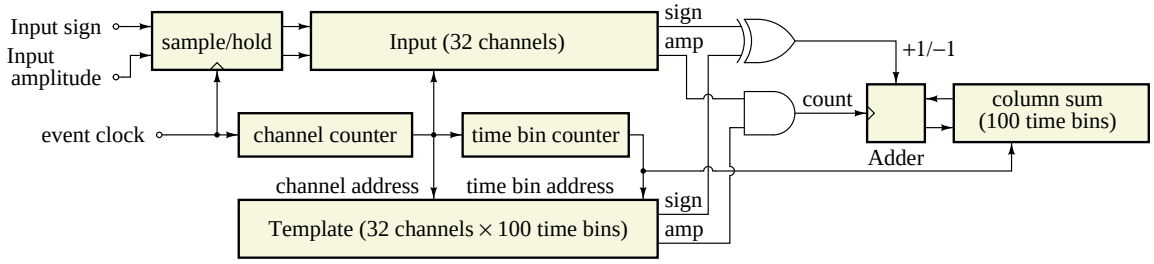

Figure 4.28: Block diagram of the sequential digital correlator architecture.

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.

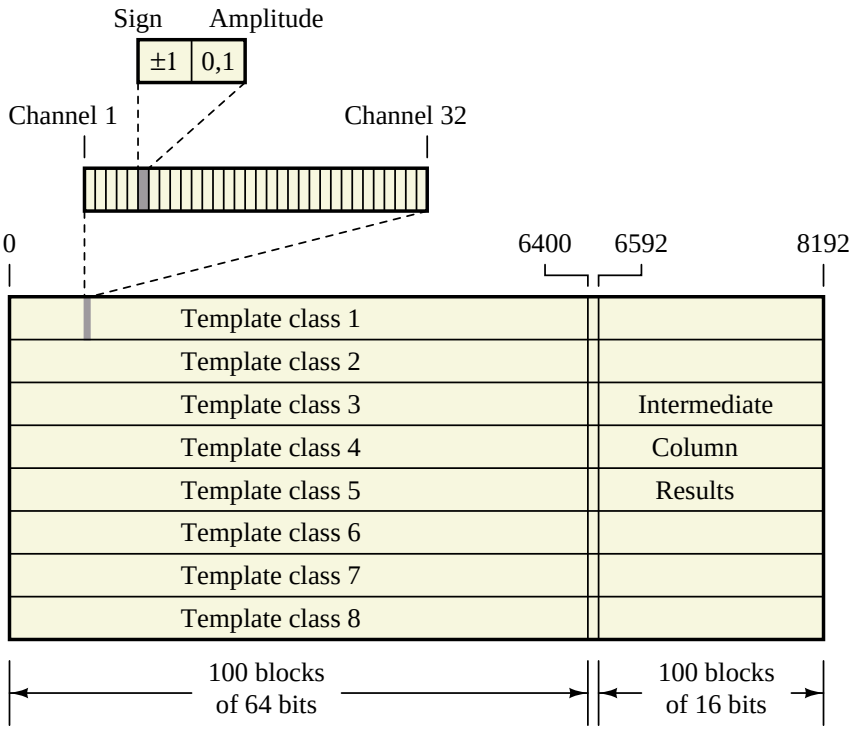

Figure 4.29: Memory allocation in the digital correlator SRAM.

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.

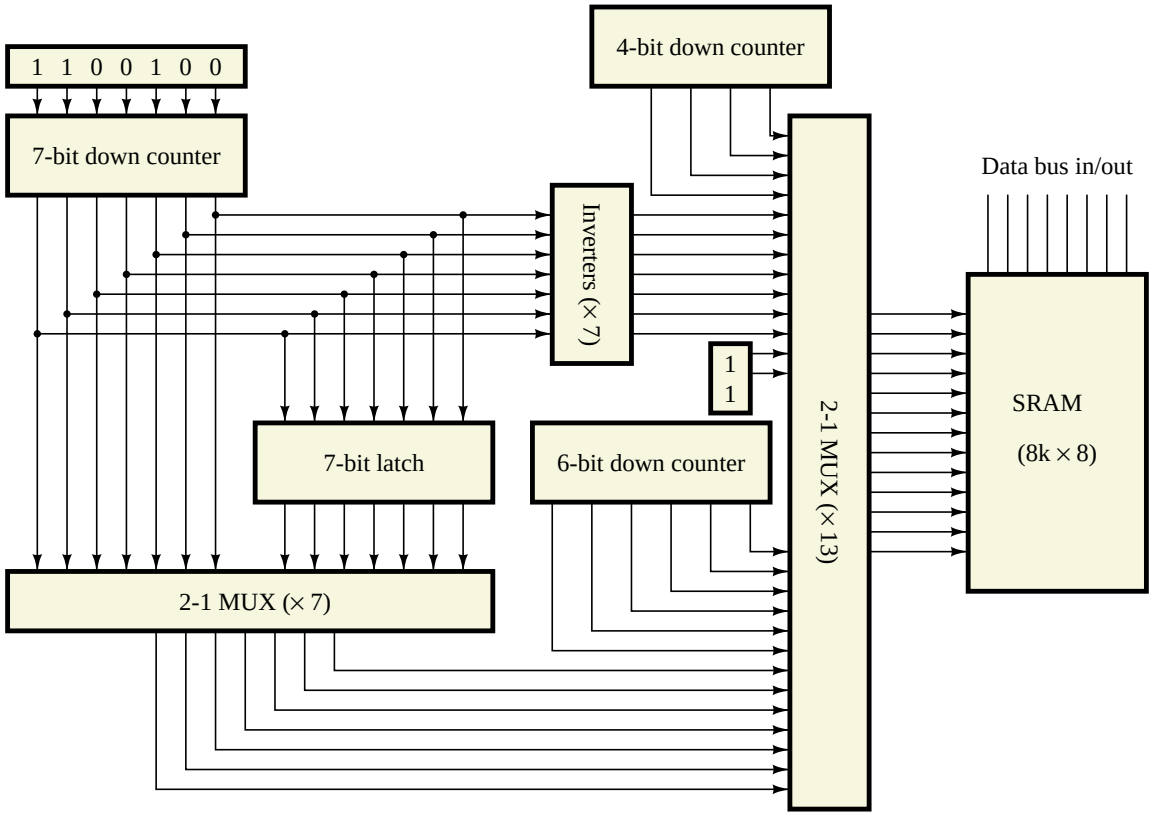

Figure 4.30: SRAM address generator for the digital correlator.

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.

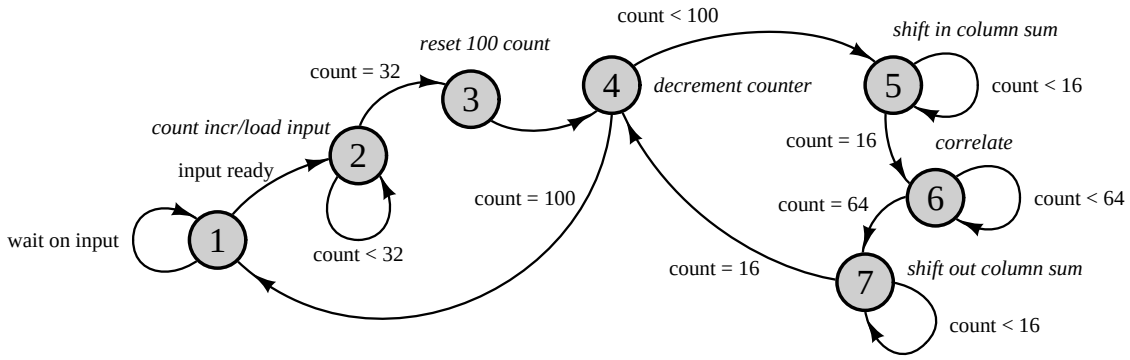

Figure 4.31: Digital correlator controller state diagram, simplified.

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.

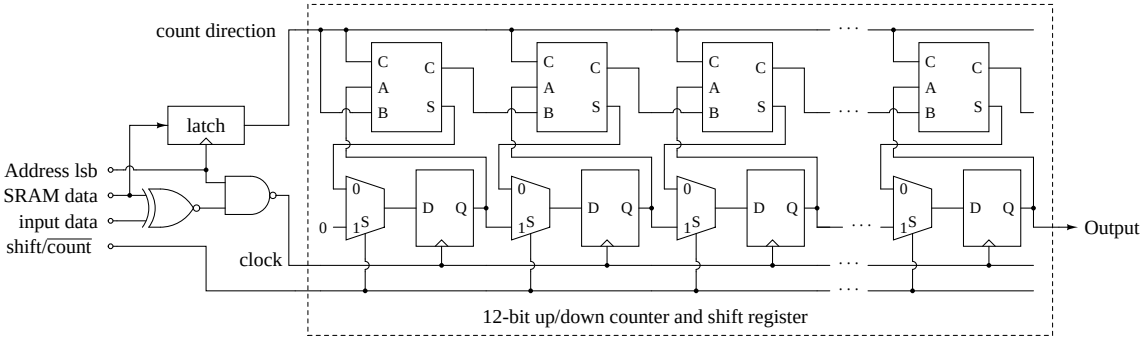

Figure 4.32: Serial digital correlator structure for one template.

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

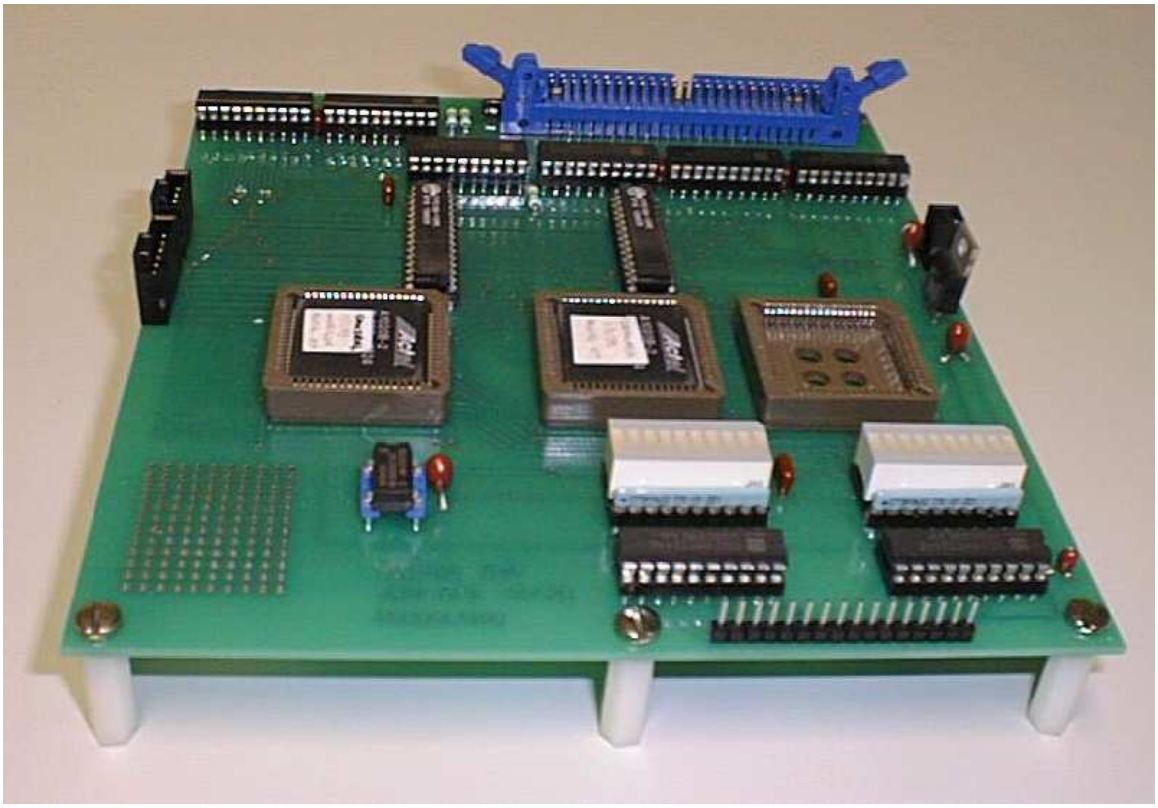

Figure 4.33: Digital ATP correlator system, configurable for up to sixteen templates.

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

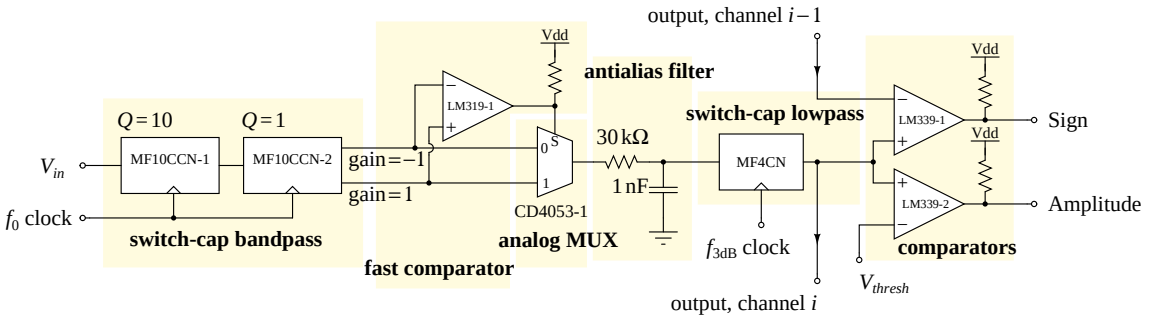

Figure 4.34: Switched-capacitor bandpass filterbank, single channel.

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.

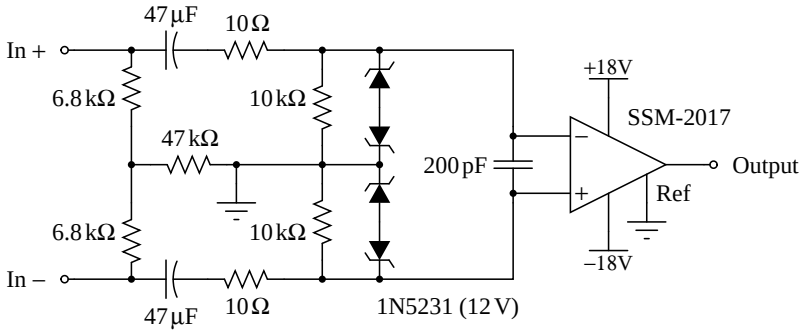

Figure 4.35: Audio input circuit for the frontend system.

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.

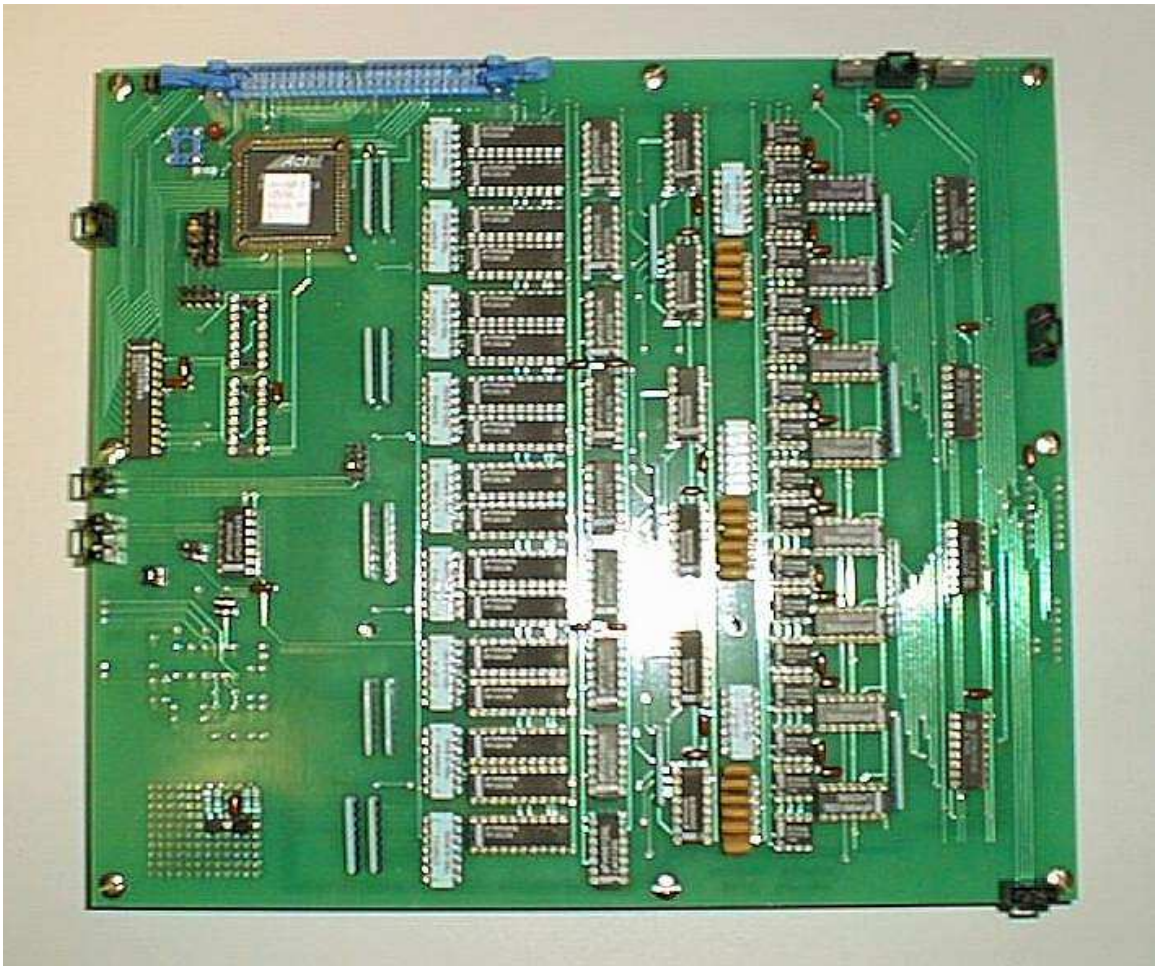

Figure 4.36: One board of the switched capacitor frontend filterbank, bandpass filtering and encoding the audio input into sixteen parallel channels.

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.

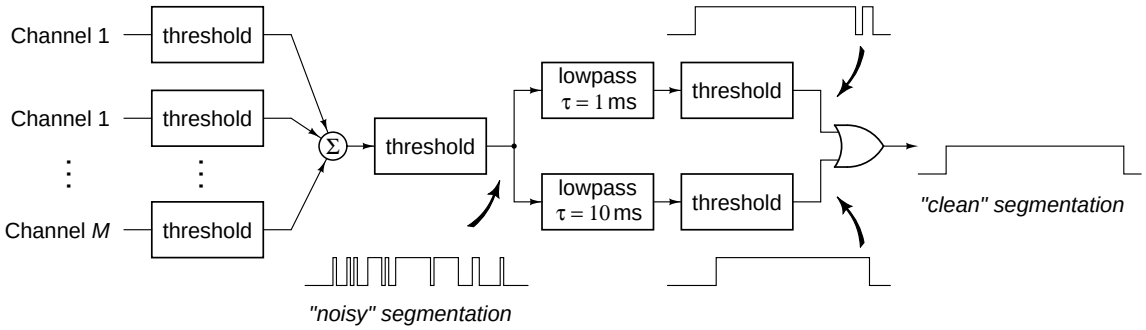

Figure 4.37: A mixed-signal method for detecting the onset of a transient event from the output of an analog frontend filterbank.

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.

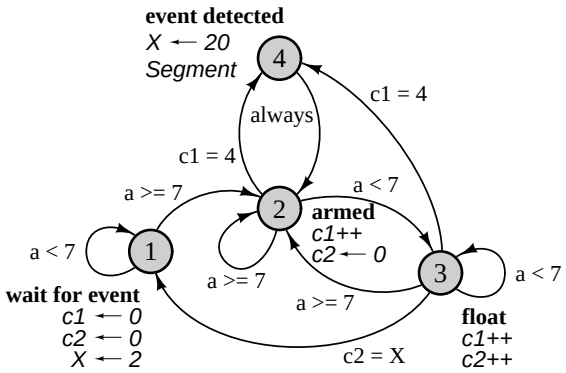

Figure 4.38: A simple finite state machine which detects the onset of transient events in the trinary-valued frontend filterbank output.

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

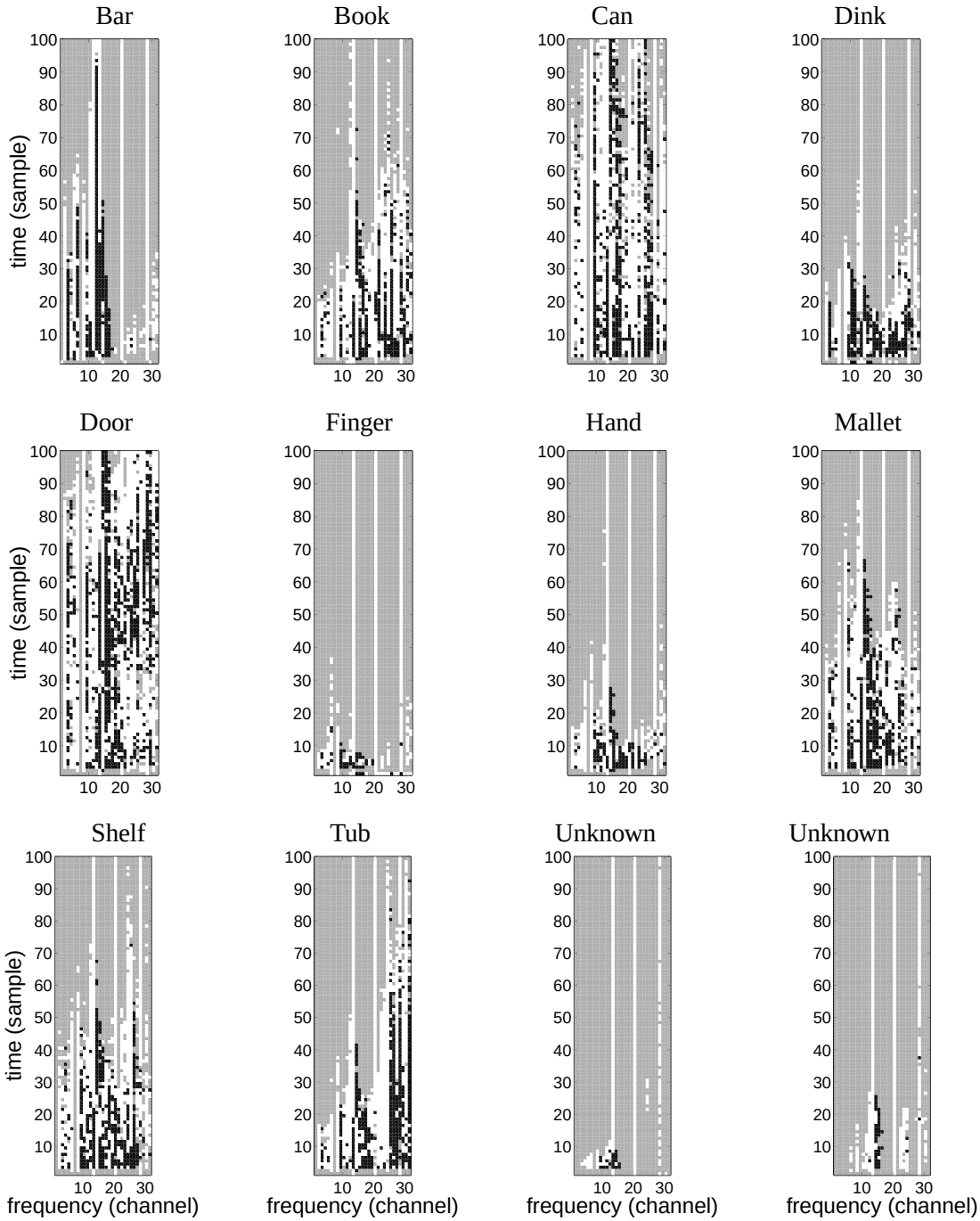

Figure 4.39: Templates for twelve transient classes, determined from data obtained from the frontend hardware. Frequency is on the x-axis, with the lowest frequency channel on the right, and time is on the y-axis, with the transient onset at the bottom of the template. Colors correspond to trinary values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

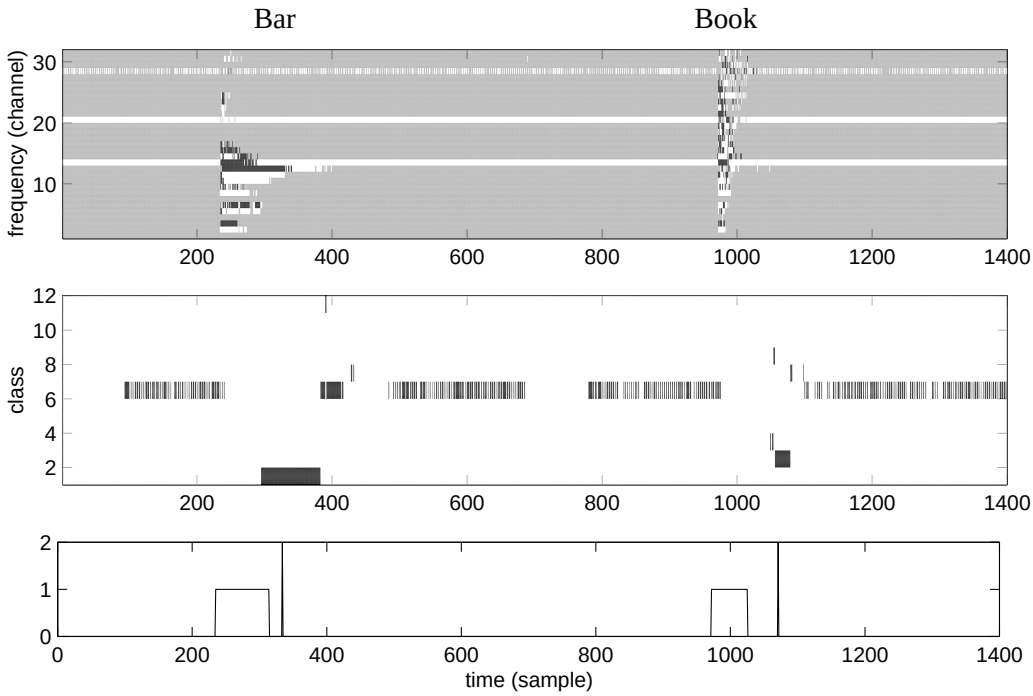

Figure 4.40: System input and output for sound “bar” (left) and “book” (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

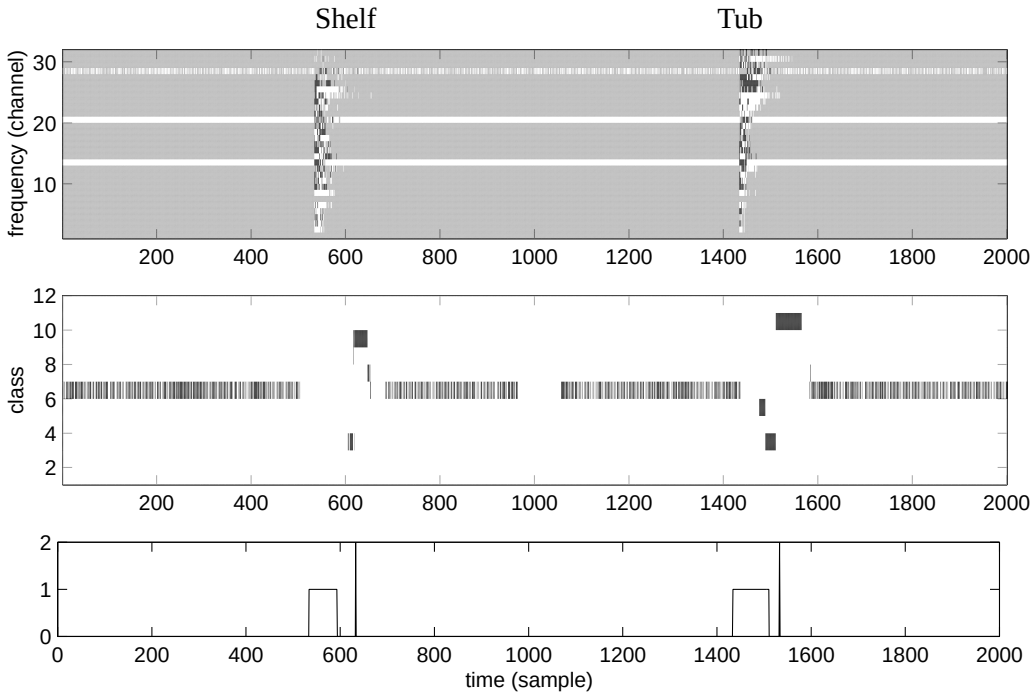

Figure 4.41: System input and output for sound “shelf” (left) and “tub” (right). Top graph is the frontend system output (32 channels); middle graph is the correlation output over 12 classes; bottom graph is the output of the segmenter showing both the segmented transient and the optimal point of alignment.

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

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

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

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

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

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

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

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

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

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