Analysis and Application of the Spectral Warping Transform to Digital Signal Processing

Warwick Peter Malcolm Allen

6th July 2007

Acknowledgements

I thank my wife, Wendy, for the sacrifice she has made to enable me to write this thesis.

I very sincerely thank my supervisors, Associate Professor Donald Bailey and Professor Serge Demidenko. Thank you both for your patience and persistence; this thesis would have never eventuated had you not cared so much to help me succeed. Thank you, Don, for your diligent guidance and insightful suggestions. Thank you, Serge, for your continual encouragement and belief in my ability.

I thank my Lord and my God, the Lord Jesus Christ, for His grace and His strength. His grace made it possible for me to study; His strength caused me to persevere. May all glory be given to Him.

Abstract

This thesis provides a thorough analysis of the theoretical foundations and properties of the *Spectral Warping Transform.* The spectral warping transform is defined as a time-domain-to-time-domain digital signal processing transform that shifts the frequency components of a signal along the frequency axis. The z-transform coefficients of a warped signal correspond to z-domain 'samples' of the original signal that are unevenly spaced along the unit circle (equivalently, frequency-domain coefficients of the warped signal correspond to frequency-domain samples of the original signal that are unevenly spaced along the frequency axis). The location of these unevenly spaced frequency-domain samples is determined by a z-domain mapping function. This function may be arbitrary, except that it must map the unit circle to the unit circle.

It is shown that, in addition to the frequency location, the bandwidth, duration and amplitude of each frequency component of a signal are affected by spectral warping. Specifically, frequency components within bands that are expanded in frequency have shortened durations and larger amplitudes (conversely, components in compressed frequency bands become longer with smaller amplitudes).

A property related to the expansion and compression of the duration of frequency components is that if a signal is time delayed (its digital sequence is prepended with zeroes) then each of the frequency components will have a different delay after warping. This time-domain separation phenomenon is useful for separating in time the frequency components of a signal. Such

separation is employed in the generation of spectrally flat chirp signals. Because spectral warping will generally expand the duration of some frequency components within a signal, the transform must produce more output samples than there are (non-zero) input samples in order to avoid time-domain aliasing. A discussion of the necessary output signal length is presented.

Particular attention is given to spectral warping using all-pass mapping function, which can be realised as a cascade of all-pass filters. There exists an efficient hardware implementation for this all-pass SW realisation [1, 2]. A proof-of-concept application-specific integrated circuit that performs the core operations required by this algorithm was developed.

Another focus of the presented research is spectral warping using a piecewise-linear mapping function. This type of spectral warping has the advantage that the changes in frequency, duration and amplitude between the non-warped and warped signals are constant factors over fixed frequency bands.

A matrix formulation of the spectral warping transformation is developed. It presents the spectral warping transform as a single matrix multiplication. The transform matrix is the product of the three matrices that represent three conceptual steps. The first step is to apply a discrete Fourier transform to the time-domain signal, providing the frequency-domain representation. Step two is an interpolation to produce the signal content at the desired new frequency samples. This interpolation effectively provides the frequency warping. The final step is an inverse DFT to transform the signal back into the time domain. A special case of the spectral warping transform matrix has the same result as a linear (finite-impulse-response) filter, showing that spectral warping is a generalisation of linear filtering. The conditions for the invertibility of the spectral warping transformation are derived.

Several possible realisation of the SW transform are discussed. These include two realisation using parallel finite-impulse-response filter banks and a realisation that uses a cascade of infinite-impulse-response filters.

Finally, examples of applications for the spectral warping transform are

given. These include: non-uniform spectral analysis (and signal generation), approximate spectral analysis in the time domain, and filter design.

This thesis concludes that the SW transform is a useful tool for the manipulation of the frequency content of digital signals, and is particularly useful when the frequency content of a signal (or the frequency response of a system) over a limited band is of interest. It is also claimed that the SW transform may have valuable applications for embedded mixed-signal testing.

Contents

List of Figures

Nomenclature

The following is an index of symbols used in this thesis, with the page number of their first occurrence.

$\mathcal{F}^{-1}\{\}$	The inverse Fourier transform operator.	16
$\hat{f}[m]$	Discrete-time output (warped) sequence.	20
$\mathbf{f}, \mathbf{f}_N$	Input vector.	47
$G[m]$	Discrete Fourier transform of $g[m]$.	45
$G[m]\Big\|_{m=i}$	A single sample in the warped frequency domain.	49
$g[m]$	Discrete-time output (warped) sequence.	46
$\mathbf{g}, \mathbf{g}_M$	Warped output vector.	47
$\mathbf{H}_{M,N}$	The $M \times N$ SW z-transform matrix.	46
$\lvert H_n(\omega)\rvert$	Magnitude gain response for N^{th}-order warping; $N = 1$ if omitted.	44
i, k	Integers.	54
$\mathbf{I}_N$	The $N \times N$ identity matrix.	68
L	Length of window sequence.	36
M	Integer number of output samples.	29
$\mathbf{m}$	Vector of row (output) indices.	47
N	Integer number of input samples.	47
n	Integer used as a discrete time domain index; typically used for input (unwarped) sequences.	20
N_0, N_1, N_2	Specific sample locations of the input sequence.	36
$\hat{N}_0, \hat{N}_1, \hat{N}_2$	Specific sample locations of the output sequence.	40
$\hat{N}_{\text{first}}$	The location of the first non-zero sample of the output sequence.	42
$\hat{N}_{\text{last}}$	The location of the last non-zero sample of the output sequence.	42
ω	Angular frequency in the z-domain.	20
ω_0	Centre frequency of a narrow-band signal.	35
$\omega_m, \hat{\omega}_m$	The m^{th} element of the vector $\boldsymbol{\omega}$ or $\hat{\boldsymbol{\omega}}$, respectively.	30
$\omega[m]$	The sequence of frequencies at which the Fourier transform of the input signal is evaluated.	45
$\hat{\omega}$	Angular frequency in the $\hat{z}$-domain.	20
$\omega_{\min}, \omega_{\max}$	Lowest and highest frequency components of a unwarped signal.	34
$\hat{\omega}[m]$	The sequence of warped frequencies at which the Fourier transform of the output sequence is evaluated.	59
$\hat{\omega}_{\min}, \hat{\omega}_{\max}$	Lowest and highest frequency components of a warped signal.	34
$\boldsymbol{\omega}, \hat{\boldsymbol{\omega}}$	Vector representation of the sequences $\omega[n]$ and $\hat{\omega}[m]$.	29
$S(\omega_{\max}, \omega_{\min})$	Bandwidth distortion function for first-order real warping.	34

Chapter 1

Introduction and Motivation

Mankind owes its success to its ability to understand and manipulate its environment. We are able to manipulate our environment to such a large extent because of the technology we have developed; our ability to develop technology is a result of having a good understanding of our world. When studying our world, and when designing technology to change it, it is very helpful to have standard frameworks with which to describe what we see and what we design. The power of a framework lies in its ability to describe realities (albeit approximately) with precise models that can be analysed mathematically. One such framework is that of *signals and systems*.

As an example of systems and signals, consider a temperature-controlled room, where the input for the temperature controller is the air temperature at the thermometer and the output is the energy released by the heating units. The temperature input and the energy output are signals and the controller is a system, according to the definition of signals and systems used by this text:

- A *signal* is a function of one or more independent variables that contains information about some phenomenon [3].

- A *system* is an entity that produces one or more output signals in response to one or more input signals [3].

The study of signal processing operates within the framework of signals and systems. The tools that are provided by signal processing theory are powerful enough to be applied to complex systems. One such set of complex systems, which is of particular concern to this thesis, is that of very-large-scale integrated circuit (VLSI) systems (with a particular view towards mixed-signal integrated circuits [1]). These are of interest because of the importance and challenges of testing such systems. There are three main reasons that make VLSI testing difficult:

1. VLSI systems are highly complex.

2. The entire surface of the chip (except over the pads) is sealed with an overglass layer preventing the circuit nodes from being available for monitoring or excitation.

3. It is not possible to modify a circuit during tests to make it work (except in unusual situations).

Design and fabrication faults in VLSI systems can be very costly, both in terms of time and money. Therefore test circuitry is often built into the chip's architecture. In fact, testability is so important that many designers are prepared to dedicate over 30% of a chip's area to this purpose [5]. This thesis will suggest a technique for testing VLSI systems.

To explain this technique (and other related procedures), the framework of signals and systems needs to be clearly defined. This is done in the following sections. The concept of signals is discussed, along with descriptions of important classes of signals. The representation of a signal in the time and the frequency domains is explained and a brief introduction to sampling is given. Then important properties of systems are outlined, leading on to a description of transforms and to an introduction to the spectral warping

[1] A mixed-signal integrated circuit combines analogue circuits with digital signal processing (DSP) circuits on a single semiconductor die. An integrated circuit that includes both analogue and digital circuitry (but no DSP), like a 555 timer, is generally not considered a mixed-signal integrated circuit [4].

transform. This chapter concludes with reasons for further investigation of the spectral warping transform and an outline of the remaining chapters.

1.1 Signals

Signals are ubiquitous, appearing in a vast plethora of forms. The act of speaking creates an acoustic signal — compression and expansion of air as a function of time. A microphone might receive the voice and translate it into an electrical signal — voltage as a function of time. Ants communicate their alarm to peers via pheromone signals — chemical concentrations as functions of time. Of course, the independent variable need not be time. Ants also communicate the location of food using pheromone trails — chemical concentrations as functions of position [6]. Another example is a newspaper: a newspaper presents a signal that consists of ink intensity as a function of position on the page. This is actually an example of a two-dimensional signal: ink intensity by horizontal position and vertical position. For an example of a higher-dimensional signal, consider a barometer attached to a weather balloon. It records a signal of four dimensions: air pressure as a function of latitude, longitude, altitude and time. This thesis is only concerned with one-dimensional signals. Time and frequency (and complex frequency) appear as the independent variables in this thesis. The theoretical results that are presented are still valid if time is substituted with some other independent variable.

The following sections give an overview of types of single-dimensional signals, and some operations that can be applied to those signals.

1.2 Continuous-Domain and Discrete-Domain, Analogue and Digital

A *continuous-domain* signal is defined for every point over some interval in the domain. Similarly, the amplitude of a *continuous-range* signal may as-

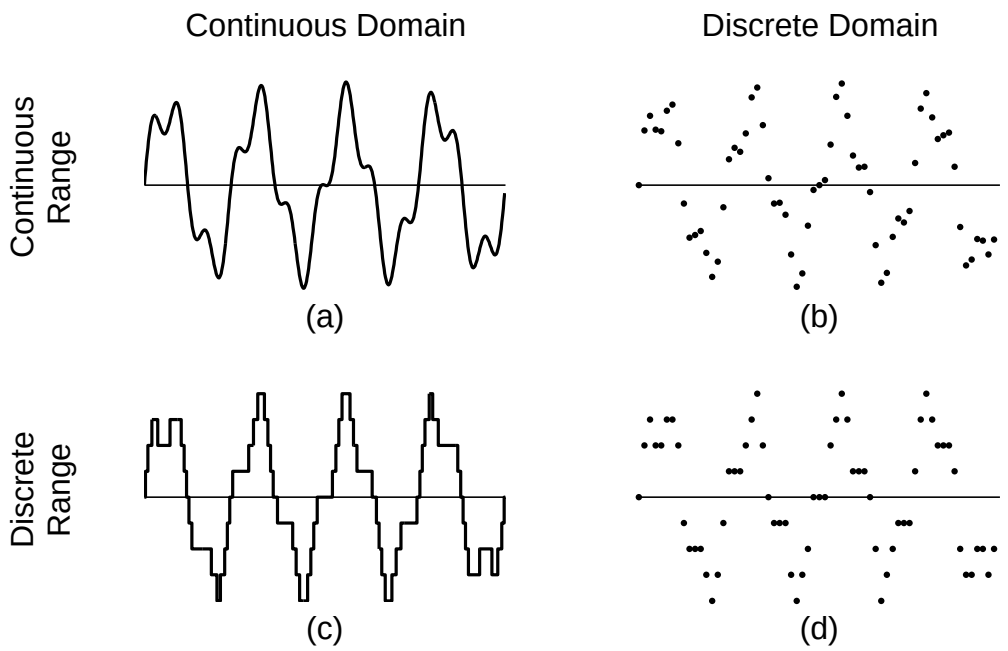

Figure 1.1: **Types of signals: continuous and discrete.** A signal may be (a) continuous in both its domain and its range (analogue), (b) sampled so as to be in a discrete domain but still have a continuous range, (c) in a continuous domain but have a discrete (quantised) range, or (d) a digital signal — discrete in both domain and range.

sume any value in a continuous range. A signal that has a continuous range over a continuous domain is an *analogue* signal. In contrast, a *discrete-domain* signal is defined only for discrete points in the domain (these points are usually, but not necessarily, evenly spaced). A *digital* signal can be represented by a sequence of numbers, so in addition to its domain being discrete, its range must also be discrete (*quantised*). That is, its amplitude may only assume a finite number of values. Figure 1.1 shows four signals: analogue (continuous-range/continuous-domain) signals, discrete-domain signals with continuous ranges, quantised continuous-domain signals and quantised discrete-domain (digital) signals.

The theory sections of this thesis deals with analogue signals and continuous-range/discrete-domain signals. For the realisation and implementations

sections, digital circuitry is used, requiring digital signals. In this text, a function of a continuous domain is represented using parentheses (for example, $f(t)$) and a function of a discrete domain is represented using square brackets (for example, $f[n]$).

1.3 Sinusoids and Complex Exponentials

A very important identity in signal processing is *Euler's equation*, as given below.

$$e^{j\omega t} = \cos\omega t + j\sin\omega t \tag{1.1}$$

Because $e^{u+v} = e^u e^v$ for any u and v, we can derive the following trigonometric identities:

$$\cos(\omega t) = \tfrac{1}{2}\left(e^{j\omega t} + e^{-j\omega t}\right) \tag{1.2}$$

$$\sin(\omega t) = \tfrac{1}{2i}\left(e^{j\omega t} - e^{-j\omega t}\right) \tag{1.3}$$

Hence, sinusoidal signals (which appear everywhere in nature) can be expressed in terms of the exponential $e^{j\omega t}$.

1.4 Real Exponentials — Decaying Signals

The function

$$h(t) = e^{-t/\tau} \qquad t \in \mathbb{R}, \quad \tau \geq 0 \tag{1.4}$$

represents a decaying signal, whose value approaches zero as time t increases. τ is the time constant — the time it takes for the signal to decay to 36.8% of its initial value (e^{-1}). If τ is negative, the signal is unbounded and approaches infinity as t increases. A simple decaying signal is plotted in figure 1.2.

1.5 Impulses

Impulse signals are another very important group of signals. An impulse in the continuous domain is defined as a signal which has an amplitude of zero

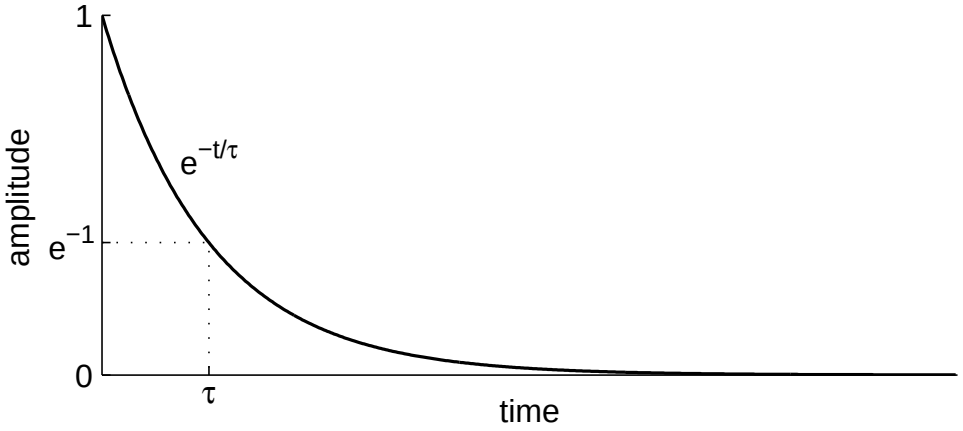

Figure 1.2: **A simple decaying signal.**

everywhere except for one time instance, at which there is an infinitesimally short burst of energy such that the total area under the signal is one. An impulse signal is represented mathematical using the *Dirac-delta function*, as defined below [7, 8].

$$\delta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-j\omega t} \, \mathrm{d}\omega \tag{1.5}$$

This definition is the inverse Fourier transform of $\omega = 1$ for all ω. The Dirac-delta function can also be expressed by the two statements below.

$$\begin{aligned} \delta(t) &= 0 \qquad t \neq 0 \\ \int_{-\infty}^{\infty} t \, \mathrm{d}t &= 1 \end{aligned} \tag{1.6}$$

An impulse at time τ is represented by $\delta(t - \tau)$.

One pertinent point that can be noted from the first definition is that the Dirac-delta function is the sum (integral) of the real cosine and imaginary sine functions of all frequencies (compare (1.5) with (1.1)). This concept will be referred to when discussing the effects of applying the spectral warping transform to an impulse signal.

The discrete equivalent of the Dirac-delta function is the *unit impulse*,

defined thus:

$$\delta[n] = \begin{cases} 0 & n \neq 0 \\ 1 & n = 0 \end{cases} \tag{1.7}$$

This can also be expressed as a sum of sinusoids:

$$\delta[n] = \frac{1}{2\pi} \sum_{\omega=-\infty}^{\infty} \cos \omega n - j \sin \omega n \tag{1.8}$$

For practical applications, a finite-length sequence representing a unit impulse is needed — that is, a sequence for which each sample has an amplitude of zero, except for one sample, which has an amplitude of one. An N-length unit impulse sequence that has its impulse a sample index N_0 can be expressed using just real cosine terms, as below.

$$\delta[n - N_0] = \sum_{i=0}^{N-1} f_i^+[n - N_0] + f_i^-[n - N_0] \tag{1.9}$$

where

$$\begin{aligned} f_i^+[n - N_0] &= \cos \frac{2\pi i(n - N_0)}{N} \\ f_i^-[n - N_0] &= \begin{cases} -\cos \frac{2\pi i(n-N_0)}{N} & n \geq N_0 \\ 0 & n < N_0 \end{cases} \end{aligned} \tag{1.10}$$

This representation is used to explain the effects of spectrally warping a unit impulse.

1.6 Impulse Responses

An *impulse response* is the signal a system produces on its output in response to an impulse signal being applied to its input. Impulse response signals are often decaying sinusoids — products of real and complex exponential functions. The equation below is an example of a typical simple impulse response signal.

$$\begin{aligned} g(t) &= e^{\frac{t_0 - t}{\tau}} \frac{e^{-j\omega(t-t_0)} + e^{j\omega(t-t_0)}}{2} \\ &= e^{\frac{t_0 - t}{\tau}} \cos \omega(t - t_0) \end{aligned} \tag{1.11}$$

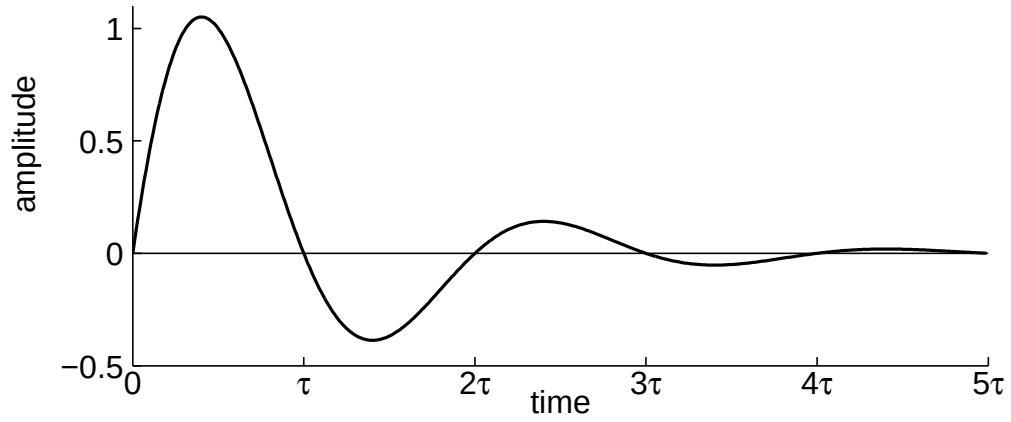

Figure 1.3: **An example impulse response signal.** The signal plotted is the decaying sinusoid $e^{\frac{t_0-t}{\tau}} \cos \omega(t-t_0)$, setting $\omega = \pi/\tau$ and $t_0 = \tau/2$.

This signal is plotted in figure 1.3 using $\omega = \pi/\tau$ and $t_0 = \tau/2$. Equation (1.11) can be rewritten by introducing the complex variable z, where

$$z = |z|\, e^{j\angle z} \tag{1.12}$$

Now

$$g(t) = \frac{|z|\,(e^{-j\angle z} + e^{j\angle z})}{2} \qquad \begin{cases} |z| &= e^{\frac{t_0-t}{\tau}} \\ \angle z &= \omega(t-t_0) \end{cases} \tag{1.13}$$

A useful feature about the impulse response is that its Fourier transform is the system (frequency) response of the system that generated it. This is because an impulse contains all the possible frequency components, all with the same energy (if the impulse occurred at time $t = 0$, then all the impulse's constituent frequency components have the same phase).

1.7 Sampling and Reconstruction

A band-limited continuous signal can be represented exactly by a discrete sequence. The values of the elements of the sequence are obtained by sampling the value of the continuous signal at certain times. When interpreting the discrete sequence, it is usually assumed that it represents the signal of the

lowest frequency that fits to the sampled points. If the samples are taken at regular intervals, then the duration of the sampling interval T_s must be no longer than half the wavelength of the highest frequency component in the signal, or

$$T_s < \frac{\pi}{\omega_{\max}} \tag{1.14}$$

where T_s is in seconds and $\omega_{\max}$ is the signal's highest frequency component in radians per second [9, 10, 11].

The sampling process effectively multiplies the continuous signal with an impulse train (a signal that has zero amplitude everywhere except at the times at which the samples are measured, at these points the impulse train has a value of one). Multiplying a signal in the time domain with the impulse train

$$h(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s) \tag{1.15}$$

is equivalent to convolving the signal in the frequency domain with the impulse train

$$H(\omega) = \frac{2\pi}{T_s} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{2\pi}{T_s}\right) \tag{1.16}$$

This time/frequency correspondence is show symbolically as

$$g(t) = f(t)h(t) \iff G(\omega) = F(\omega) \otimes H(\omega) \tag{1.17}$$

where $\otimes$ is the convolution operator and $F(\omega)$, $G(\omega)$ and $H(\omega)$ are the respective Fourier transforms of $f(t)$, $g(t)$ and $h(t)$. This convolution of the signal's spectrum causes an image of the spectrum to be repeated at intervals of $2\pi/T_s$ along the frequency axis. So if the signal's bandwidth is greater than $2\pi/T_s$, the spectral images will overlap and cause aliasing (to avoid aliasing the highest frequency component must be no higher than π/T_s, accounting for the negative and positive sides of the spectrum). This sampling process is shown in figure 1.4, in both the time and the frequency domains.

To reconstruct the continuous signal from the sampled signal, the spectral images need to be removed. This can be done by multiplying the spectrum

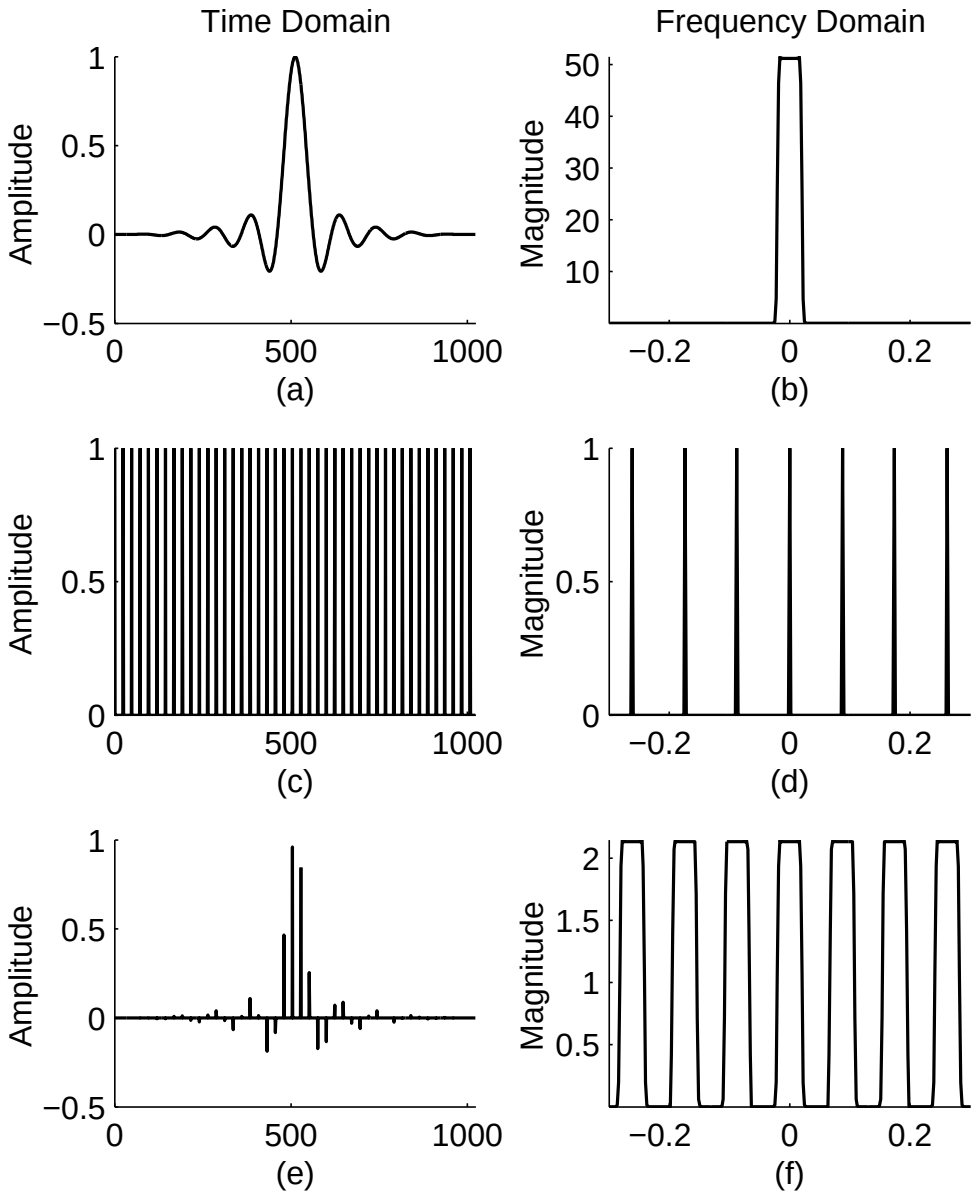

Figure 1.4: **The sampling process in the time and frequency domains.** The time-domain signal (a) is multiplied by an impulse train (c) to give the sampled signal (e). In the frequency domain, the signal's spectrum (b) is convolved with the impulse train shown in (d) to give the the spectrum of the sampled signal (f).

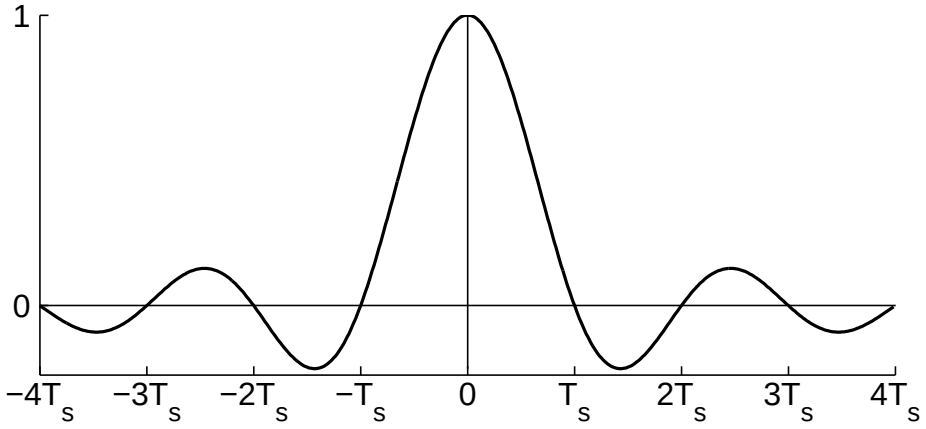

Figure 1.5: **A sinc signal used for reconstructing continuous signals from their discrete sequence representation.** T_s is the sampling period.

of the sampled signal with a *reconstruction filter*. A ideal reconstruction filter has a value of zero for all frequencies of magnitude greater than π / T_s and a value of one for all frequencies from $-\pi / T_s$ to π / T_S — that is, a rectangular window of width $2\pi / T_S$, centred on the origin. In the time domain, this rectangular window equates to a sinc that has a main-lobe width of $2\,T_s$, like that shown in figure 1.5. Multiplication in the frequency domain is convolution in the time domain, so to recover the original signal, the sampled signal is convolved with the sinc. This convolution is described mathematically below.

$$\begin{aligned} f(t) &= f[n] \otimes \frac{\sin \pi \frac{t}{T_s}}{\pi \frac{t}{T_s}} \\ &= \sum_{n=-\infty}^{\infty} f[n] \frac{\sin \pi \left(n - \frac{t}{T_s} \right)}{\pi \left(n - \frac{t}{T_s} \right)} \end{aligned} \tag{1.18}$$

where $f[n]$ are the samples of $f(t)$.

Other functions may be used to interpolate samples to approximately reconstruct the continuous signal. A thorough review of sampling techniques can be found in [12].

1.8 Important Properties of Systems

There are some systems that have certain properties that make them extremely useful. These properties are *stability*, *linearity*, *time-invariance* and *invertibility*, as explained below. Another important classification of systems is whether or not they have a finite-length response to a finite-length excitation (also explained below). Whenever a system is investigated, it is important to consider which of these properties apply to that system.

1.8.1 Stability

A system is stable if all its output signals are bounded whenever all input signals are bounded. The poles of a stable system are always inside of the unit circle.

1.8.2 Finite and Infinite Response

The impulse response of a *finite-impulse-response* (FIR) system eventually settles to zero, whereas an *infinite-impulse-response* (IIR) system may respond indefinitely to an impulse. Useful features of FIR systems include:

1. They are always stable. All the poles of an FIR system are located at the origin (inside the unit circle).

2. FIR systems have no feedback. The output is only dependent on the inputs, not on previous outputs.

An IIR system, on the other hand, has internal feedback, so the output is determined by the inputs and previous output. The poles of an IIR system can be anywhere on the z-plane, so they are not guaranteed to be stable.

1.8.3 Linearity and Superposition

A very important class of system is that of linear systems. Linear systems possess the property of superposition. Superposition states that if the input

to a system is a weighted sum of two or more signals, then the output of the system is equal to an equivalently weighted sum of the system's response to each of the components of the input. To express this more precisely, consider a system that produces an output $y(t)$ in response the input $x(t)$. The system is linear if the following is true: if $y_1(t)$ and $y_2(t)$ are the system's responses to $x_1(t)$ and $x_2(t)$ respectively, and $x(t) = A_1x_1(t) + A_2x_2(t)$ (where A_1 and A_2 are any constants), then $y(t) = A_1y_1(t) + A_2y_2(t)$.

The superposition condition can also hold for systems that operate in discrete domains: a discrete system H is linear if

$$H\left(A_1x_1[n] + A_2x_2[n]\right) = H\left(A_1y_1[m]\right) + H\left(A_2y_2[m]\right) \tag{1.19}$$

When dealing with discrete sequences, each output sample of a linear system is a linear combination of some or all of the input samples. This means that a linear FIR system can be represented by a matrix multiplication.

1.8.4 Time (Shift) Invariance

A system is time invariant if the output to a given input does not depend on when that input is applied. An example of a physical system that is not time invariant is a transistor that generates heat while it is used and whose parameters change with a change in temperature. An input applied to the transistor when it has just been turned on and is cold may produce a different output from the same input applied to the transistor when it has been running a while and is hot.

A continuous-time system H is time invariant if

$$H\left(x(t)\right) = y(t) \Rightarrow H\left(x(t-\tau)\right) = y(t-\tau) \qquad \forall \quad \tau \in \mathbb{R} \tag{1.20}$$

The equivalent of time invariance in discrete-time systems is shift invariance; a discrete-time system H is shift invariant if

$$H\left(x[n]\right) = y[n] \Rightarrow H\left(x[n-n_0]\right) = y[n-n_0] \qquad \forall \quad n_0 \in \mathbb{I} \tag{1.21}$$

1.8.5 Invertibility

A system is invertible if unique inputs always produce the unique outputs. This implies that there is a one-to-one mapping between the input and the output. An invertible system has an inverse system: the original signal can be recovered by applying the output signal to the inverse system.

1.9 Transforms

The useful or interesting information in a signal can often be made more accessible by rearranging the signal in some manner. This is the core job of signal processing. An example is transforming a signal so that the frequency components contained within the signal may be clearly seen. Such information, once retrieved, can be used to make decisions — either by a human or by an automatic control system. These transformations usually take place in the digital domain, where powerful or specialised computers can be used for the task. The term *transform* is given to a system that performs some such transformation on a signal.

There are many transforms that are commonly applied to signals, each revealing some particular information of interest. The Fourier transform, for example, rearranges a signal to show its frequency composition. Examples of transforms that can be applied to discrete-time signals include

- the discrete-time Fourier transform (which expresses signals in terms of real frequency components [13]),
- the z-transform (which expresses signals in terms of complex, or decaying, frequency components [13]),
- the Hadamard transform (which expresses signals in terms of a symmetrical pattern of values of positive and negative one [14]),
- and wavelet transforms (which express signals in terms of shorter signals of different frequencies [15]).

Each transform is useful in certain situations, and each has limitations that can restrict its usefulness at times. For this reason, it is beneficial to investigate transforms to discover what type of information they reveal and in what circumstances they are superior to other transforms.

Some very common transforms — the continuous and discrete-time Fourier transforms and the z-transform — are introduced now, followed by the spectral warping transform.

1.10 Time and Frequency Domains and the Fourier Transform

Any signal can be constructed as a weighted superposition of complex exponential functions of different frequencies. For a continuous function, this superposition is given by the following integral, where $f(t)$ is any real signal.

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)\, e^{j\omega t}\, \mathrm{d}\omega \tag{1.22}$$

where $F(\omega)$ is the function giving the weight of the frequency component at ω. Equation (1.22) can be recognised as the inverse *Fourier transform* formula and $F(\omega)$ as the Fourier transform, or frequency-domain representation, of $f(t)$. The (forward) Fourier transform expresses the frequency-domain representation of a signal as a weighted superposition of complex exponential functions of different time values.

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, \mathrm{d}t \tag{1.23}$$

In this work, the time-domain representation of a signal will be labelled with a lowercase letter and the frequency-domain representation with the corresponding uppercase letter. The Fourier transform operation is denoted by the $\mathcal{F}\{\}$ symbol.

$$F(\omega) = \mathcal{F}\{f(t)\}(\omega) \tag{1.24}$$

The inverse Fourier transform is denoted by $\mathcal{F}^{-1}\{\}$.

$$f(t) = \mathcal{F}^{-1}\{F(\omega)\}(t) \tag{1.25}$$

The Fourier transform of a discrete-time signal is referred to as the *discrete-time Fourier transform* (DTFT). Its definition is similar to that of the continuous-time Fourier transform, with the integration being replaced by a summation.

$$F(\omega) = \sum_{n=-\infty}^{\infty} f[n]\, e^{-j\omega n} \tag{1.26}$$

Its inverse is

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega)\, e^{j\omega n}\, \mathrm{d}\omega \tag{1.27}$$

Note that the integral in the inverse DTFT formula is taken over the interval $(-\pi, \pi)$. This is because the DTFT is periodic over that interval (that is, $F(\omega + 2k\pi) = F(\omega)$ for any integer k).

1.11 The z-Transform

The sequence $g[n]$ can be expressed in terms of the complex variable z by calculating its z-*transform*. The transform equation is

$$G(z) = \sum_{n=0}^{\infty} g[n] z^{-n} \tag{1.28}$$

$G(z)$ is the z-transform of $g[n]$ [16]. It is defined as a power series of z^{-1}. Each z^{-n} can be interpreted as indicating the nth sample of $g[n]$. Therefore, multiplying a sequence by z^{-n} shifts the sequence to the right by n samples.

The inverse z-transform equation is

$$g[n] = \frac{1}{2\pi j} \oint F(z) z^{n-1}\, \mathrm{d}z \tag{1.29}$$

The value of the z-transform of $g[n]$ for $|z| = 1$ (the unit circle) is the discrete-time Fourier transform of $g[n]$.

1.12 The Spectral Warping Transform

The spectral warping (SW) transform is another discrete-time transform, first proposed by Oppenheim, Johnson and Steiglitz [2] and further investigated in [1, 17, 18].

The SW transform rearranges a signal in such a way that each frequency component is mapped to a new frequency. Consequentially, the duration and amplitude of each frequency component are also affected. These properties potentially make the SW transform a useful tool in signal analysis, especially for extracting information from a signal about its frequency composition over a restricted frequency band.

1.13 Motivation for Investigating the Spectral Warping Transform

The hypothesis of this thesis is that the SW transform is indeed a useful addition to the suite of tools available for digital signal analysis and manipulation, particularly as a technique for system characterisation and high-resolution frequency analysis. To test this hypothesis, an in-depth mathematical analysis of the transform is presented (chapter 2). The representations derived from the analysis are used to develop implementations of the transform (chapter 3). Thus equipped with an understanding of the properties of the SW transform, some applications for it are explored in chapter 4. A more specific proposition that this thesis addresses is that SW-based techniques are useful for testing mixed-signal systems (particularly mixed-signal integrated circuits).

A secondary aim of this thesis is to test the usefulness of implementing a realisation of a SW transform using an application-specific integrated circuit (ASIC). The fitness of a such an implementation to digital and mixed-signal integrated circuit characterisation is evaluated. To this end, a proof-of-concept ASIC was designed, fabricated and tested, the details of which are given

in appendix A.

For the SW transform to take its place alongside other digital signal processing transforms, a rigorous mathematical analyses must be provided. Such an analysis is presented in the next chapter.

Chapter 2

Analysis of the Spectral Warping Transform

This chapter provides a theoretical underpinning of the spectral warping (SW) transform, detailing considerations that must be taken into account when applying the transform.

The transform is first discussed in general terms, described by an arbitrary frequency-mapping function. Then some specific SW transforms, described by certain families of frequency-mapping functions, are discussed. Some of these variations of the transform are useful for practical applications and some are not, for reasons that will be explained. The various effects of the transform are covered in section 2.2. Section 2.3 explains how the SW transform can be viewed as a linear transformation. The implications of the length of the input and output signals are then discussed. The final section of this chapter provides an alternative way to calculate the SW transform.

2.1 Digital Spectral Warping

The SW transform is a digital signal processing (DSP) transform that shifts the frequency content of a signal by manipulating its time samples. It is a time-domain-to-time-domain transform, wherein the discrete Fourier trans-

form (DFT) of the warped signal is equal to non-uniformly spaced samples of the Fourier transform of the original signal. This can be considered to be a warping of the frequency axis. This warping is governed by the frequency warping function $\theta(\omega)$

$$\hat{\omega} = \theta(\omega) \tag{2.1}$$

where ω and $\hat{\omega}$ are the original and warped frequencies, respectively.

Spectral warping can also be viewed in the z-domain. In the z-domain, sinusoids map onto the unit circle. Therefore SW effectively warps the z-plane such that points on the unit circle are mapped to other points on the unit circle. Thus, (2.1) can be restated in terms of z.

$$\begin{aligned} \hat{z} &= \Theta(z) \\ |\hat{z}| &= 1 \quad \forall \quad |z| = 1 \end{aligned} \tag{2.2}$$

where z is the unwarped, and $\hat{z}$ is the warped, complex frequency variable.

A general spectral warping formula can be developed as follows. Beginning with the input sequence $f[n]$, the z-transform is taken:

$$\sum_{n=-\infty}^{\infty} f[n] z^{-n}$$

the z-domain is warped according to the warping function $\Theta(z)$:

$$\sum_{n=-\infty}^{\infty} f[n] \Theta(z)^{-n}$$

and the inverse z-transform is taken to give the output sequence $\hat{f}[m]$:

$$\hat{f}[m] = \frac{1}{2\pi j} \oint_{C'} \sum_{n=-\infty}^{\infty} f[n] \Theta(z)^{-n} z^{m-1} \, \mathrm{d}z \tag{2.3}$$

where C' is the anticlockwise contour of integration defined by $|z| = 1$ [13]. Although (2.3) describes spectral warping succinctly as a single equation, it can be difficult to calculate and is not used directly in this thesis. Rather,

the spectral warping transform is calculated by working out each of the steps sequentially.

The DFT of a signal consists of samples of the signal's z-transform evenly spaced around the unit circle. Because of the warping of the z-domain, evenly spaced samples around the unit circle in the $\hat{z}$-domain equate to unevenly spaced samples around the unit circle in the z-domain. Therefore the DFT of a spectrally-warped signal consists of unevenly spaced samples of the original signal's z-transform. A conceptual explanation of the process of spectral warping, then, is as follows:

1. The z-transform of the original signal is taken.
2. This is sampled at arbitrary points on the unit circle.
3. A frequency warping function is applied that redistributes the arbitrary samples evenly around the unit circle.
4. Finally, the inverse z-transform is evaluated at the sample locations. As the samples are now evenly spaced, the inverse z-transform reduces to an inverse DFT.

Steps 2 and 3 are illustrated in figure 2.1. Note that steps 2 and 3 can be swapped — it is equally valid to consider the warping of the z-plane happening first, which is then sampled at evenly spaced locations. It is only a conceptual difference.

Some key features of the SW transform, aside from the frequency warping effect, are listed below:

Time domain to time domain. Although the SW transform is largely characterised by its effect in the frequency domain, it is a time-domain-to-time-domain transform.

Not time (shift) invariant. Delaying an input signal to the SW transform will affect the shape of the output signal. This property is discussed further in section 2.2.2 and in chapter 4.

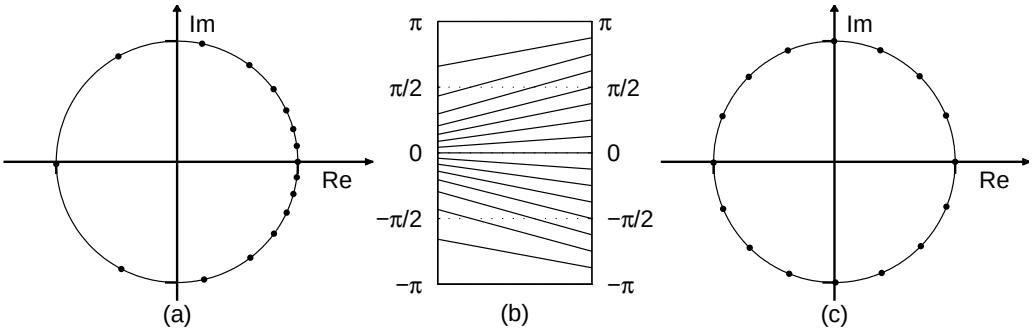

Figure 2.1: **The steps involved in spectral warping.** The unevenly distributed z-domain samples (a) are redistributed by the spectral warping function (b) to be evenly spaced around the unit circle (c). The inverse z-transform now reduces to an inverse DFT.

Linearity. The SW transform is mathematically linear: each output sample is a linear combination of the input samples. This is shown in section 2.3. It is not linear, however, in the common engineering sense, in that it can introduce frequencies in the output that were not present in the input. The introduction of these new frequencies is a result of the transform not being shift invariant; this will also be explained in section 2.4.

Amplitude and duration is affected. The amplitude and the duration of the signal's frequency components change as the frequencies are warped. These effects are discussed in the following sections.

2.1.1 The Frequency Warping Function

As stated by (2.2), the frequency warping function $\Theta(z)$ maps the unit circle onto the unit circle ($|\Theta(z)| = 1 \ \forall \ |z| = 1$). Any function that has this property can be used for spectral warping, although the mapping must have a one-to-one correspondence if the spectral warping is to be invertible. Some candidate functions are considered below.

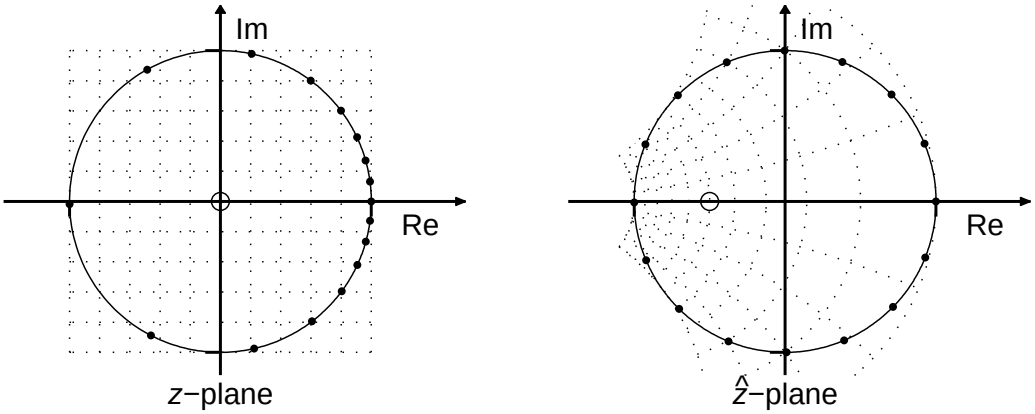

Figure 2.2: **Warping of the z-plane using a real first-order all-pass mapping function.**

2.1.2 First-Order All-Pass Mapping

A first-order all-pass mapping is described by

$$\hat{z} = \Theta_1(z) = \frac{z - a}{1 - a^* z} \qquad |a| < 1 \tag{2.4}$$

where a is a warping factor and a^* is the complex conjugate of a. This function has a pole at $(a^*)^{-1}$, causing the z-plane to warp away from $(a^*)^{-1}$ (where z maps to $\hat{z} = \infty$). A value of $|a|$ close to unity gives maximum warping, and negating a gives the inverse warping. An example of first-order all-pass warping of the z-plane, for which $\hat{z}$ has a real pole, is shown in figure 2.2. In this example the z-plane is warped into the $\hat{z}$-plane using the first-order warping function

$$\hat{z} = \frac{z - \frac{1}{2}}{1 - \frac{z}{2}} \tag{2.5}$$

The inverse mapping is given by

$$z = \frac{\hat{z} + \frac{1}{2}}{1 + \frac{\hat{z}}{2}} \tag{2.6}$$

which has a pole at $\hat{z} = -2$. The warping 'gathers' the z-plane towards this pole. A second example of first-order all-pass warping of the z-plane (where

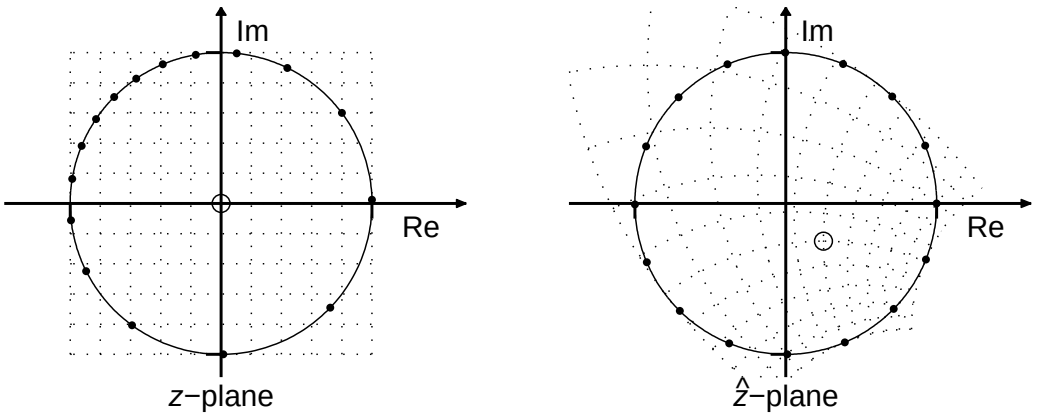

Figure 2.3: **Warping of the z-plane using complex first-order all-pass mapping.**

$\hat{z}$ has a complex pole) is shown in figure 2.3. In this case, the z-plane is warped into the $\hat{z}$-plane using the complex warping parameter $a = {}^{(j-1)}/_{4}$, giving a first-order warping function

$$\hat{z} = \frac{z - \frac{j-1}{4}}{1 + z\frac{j+1}{4}} \tag{2.7}$$

The inverse mapping

$$z = \frac{\hat{z} + \frac{j-1}{4}}{1 - \hat{z}\frac{j+1}{4}} \tag{2.8}$$

has a pole at $z = 2(1-j)$, towards which the z-plane is 'gathered'.

To derive the warping of the frequency axis, $\hat{\omega} = \theta(\omega)$, from the warping of the z-plane, the substitution $z = e^{j\omega}$ is made.

$$e^{j\hat{\omega}} = \Theta(e^{j\omega}) = \frac{e^{j\omega} - a}{1 - a^* e^{j\omega}} \tag{2.9}$$

$$\begin{aligned} \Rightarrow \quad \theta(\omega) = \hat{\omega} &= -j \ln\left(\Theta(e^{j\omega})\right) \\ &= j \ln\left(1 - a^* e^{j\omega}\right) - j \ln\left(e^{j\omega} - a\right) \end{aligned} \tag{2.10}$$

The equation (2.10) is simplified observing that the magnitude of (2.9) is

unity for all frequencies, as proved below.

$$\begin{aligned}\left|\Theta(e^{j\omega})\right|^2 &= \frac{\left|e^{j\omega}-a\right|^2}{\left|1-a^*e^{j\omega}\right|^2} \\ &= \frac{\left(e^{j\omega}-a\right)\left(e^{-j\omega}-a^*\right)}{\left(1-a^*e^{j\omega}\right)\left(1-ae^{-j\omega}\right)} \\ &= \frac{1-ae^{-j\omega}-a^*e^{j\omega}+\left|a\right|^2}{1-a^*e^{j\omega}-ae^{-j\omega}+\left|a\right|^2} = 1\end{aligned} \tag{2.11}$$

Therefore the frequency warping is the phase angle of the z-plane warping function:

$$\begin{aligned}\theta(\omega) = -j\ln\Theta(e^{j\omega}) &= -j\ln\left|\Theta(e^{j\omega})\right| - j\ln\left(e^{j\angle\Theta(e^{j\omega})}\right) \\ &= \angle\Theta(e^{j\omega}) \\ &= \tan^{-1}\frac{\mathrm{Im}\left\{\Theta(e^{j\omega})\right\}}{\mathrm{Re}\left\{\Theta(e^{j\omega})\right\}}\end{aligned} \tag{2.12}$$

If the complex part of $\Theta(e^{j\omega})$ is moved to its numerator by multiplying the numerator and denominator by the denominator's complex conjugate,

$$\begin{aligned}\Theta(e^{j\omega}) &= \frac{\left(e^{j\omega}-a\right)\left(1-a^*e^{j\omega}\right)^*}{\left(1-a^*e^{j\omega}\right)\left(1-a^*e^{j\omega}\right)^*} \\ &= \frac{\left(e^{j\omega}-a\right)\left(1-ae^{-j\omega}\right)}{\left|1-a^*e^{j\omega}\right|^2}\end{aligned} \tag{2.13}$$

then the denominator can be ignored (it is real and will cancel out). Separating the numerator into its real and imaginary parts produces

$$\begin{aligned}\left(e^{j\omega}-a\right)\left(1-ae^{-j\omega}\right) &= e^{j\omega}-2a+a^2e^{-j\omega} \\ &= \cos\omega + j\sin\omega - 2a_R - j2a_I + \\ &\quad \left(a_R^2 + j2a_Ra_I - a_I^2\right)\left(\cos\omega - j\sin\omega\right) \\ &= \left(1+a_R^2-a_I^2\right)\cos\omega + 2a_R\left(a_I\sin\omega - 1\right) + \\ &\quad j\left(\left(1-a_R^2+a_I^2\right)\sin\omega + 2a_I\left(a_R\cos\omega - 1\right)\right)\end{aligned} \tag{2.14}$$

where $a_R = \mathrm{Re}\left\{a\right\}$ and $a_I = \mathrm{Im}\left\{a\right\}$.

Inserting this result into (2.12) gives an expression for the frequency warping function:

$$\theta(\omega) = \tan^{-1} \frac{(1 - a_R^2 + a_I^2)\sin\omega + 2a_I\,(a_R\cos\omega - 1)}{(1 + a_R^2 - a_I^2)\cos\omega + 2a_R\,(a_I\sin\omega - 1)} \tag{2.15a}$$

An alternative form of this expression is derived here:

$$\begin{aligned}
\Theta(e^{j\omega}) = \frac{e^{j\omega} - a}{1 - a^* e^{j\omega}} &= e^{j\omega}\frac{1 - ae^{-j\omega}}{1 - a^* e^{j\omega}} \\
&= e^{j\omega}\frac{(1 - ae^{-j\omega})\,(1 - ae^{-j\omega})}{(1 - a^* e^{j\omega})\,(1 - ae^{-j\omega})} \\
&= e^{j\omega}\frac{(1 - ae^{-j\omega})^2}{|1 - a^* e^{j\omega}|^2}
\end{aligned}$$

$$\begin{aligned}
\theta(\omega) = \angle\Theta(e^{j\omega}) &= \angle e^{j\omega} + \angle\left(\frac{(1 - ae^{-j\omega})}{|1 - a^* e^{j\omega}|}\right)^2 \\
&= \omega + 2\tan^{-1}\frac{\mathrm{Im}\,\{1 - ae^{-j\omega}\}}{\mathrm{Re}\,\{1 - ae^{-j\omega}\}} \\
&= \omega + 2\tan^{-1}\frac{a_R\sin\omega - a_I\cos\omega}{1 - a_R\cos\omega - a_I\sin\omega}
\end{aligned} \tag{2.15b}$$

Because a is real for most practical implementations of the SW transform, the following simplified frequency warping function is normally used, which is valid for real a.

$$\theta(\omega) = \tan^{-1}\frac{(1 - a^2)\sin\omega}{(1 + a^2)\cos\omega - 2a} \qquad a \in \mathbb{R}, \quad |a| < 1 \tag{2.16a}$$

or

$$\theta(\omega) = \omega + 2\tan^{-1}\frac{a\sin\omega}{1 - a\cos\omega} \qquad a \in \mathbb{R}, \quad |a| < 1 \tag{2.16b}$$

where $\mathbb{R}$ is the set of all real numbers. Both forms of (2.16) are stated in [1].

The inverse of (2.15) is found by negating a (i.e., warping the signal in the opposite direction).

$$\theta^{-1}(\omega) = \tan^{-1}\frac{(1 - a_R^2 + a_I^2)\sin\omega + 2a_I\,(a_R\cos\omega + 1)}{(1 + a_R^2 - a_I^2)\cos\omega + 2a_R\,(a_I\sin\omega + 1)} \tag{2.17}$$

The inverse warping for real a, then, is

$$\theta^{-1}(\omega) = \tan^{-1}\frac{(1 - a^2)\sin\omega}{(1 + a^2)\cos\omega + 2a} \qquad a \in \mathbb{R}, \quad |a| < 1 \tag{2.18}$$

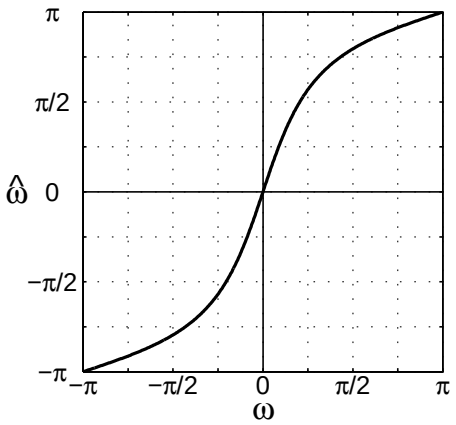

Figure 2.4: **First-order all-pass warping of the frequency axis with a real warping parameter.** The frequency warping curve is given by (2.15), where $a = 1/2$.

The warping of the frequency axis that is equivalent to the z-plane warping of figure 2.2 is shown in figure 2.4. Two other first-order frequency warping functions are plotted in figure 2.5. It can be clearly seen from these figures that increasing the magnitude of a makes the warping more severe, and negating a causes warping in the reverse direction.

It can also be observed from figures 2.4 and 2.5 that the frequency axis is stretched where $\theta(\omega)$ has a slope greater than 1 and is condensed where the slope is less than 1. The slope of $\theta(\omega)$ is given by

$$\begin{aligned}\frac{\mathrm{d}\,\theta(\omega)}{\mathrm{d}\,\omega} &= \frac{\mathrm{d}}{\mathrm{d}\,\omega}\left(j\ln\left(1-a^*e^{j\omega}\right) - j\ln\left(e^{j\omega}-a\right)\right)\\ &= \frac{a^*e^{j\omega}}{1-a^*e^{j\omega}} + \frac{e^{j\omega}}{e^{j\omega}-a}\\ &= \frac{1-|a|^2}{\left|e^{j\omega}-a\right|^2}\\ &= \frac{1-a_R^2-a_I^2}{1-2a_R\cos\omega-2a_I\sin\omega+a_R^2+a_I^2}\end{aligned} \tag{2.19}$$

$$\Rightarrow \qquad \frac{\mathrm{d}\,\theta(\omega)}{\mathrm{d}\,\omega} = \frac{1-a^2}{1-2a\cos\omega+a^2} \qquad a\in\mathbb{R}, \quad |a|<1 \tag{2.20}$$

This slope has a maximum at $\omega = \angle a$ ($\omega = 0$ for positive, real a) and a

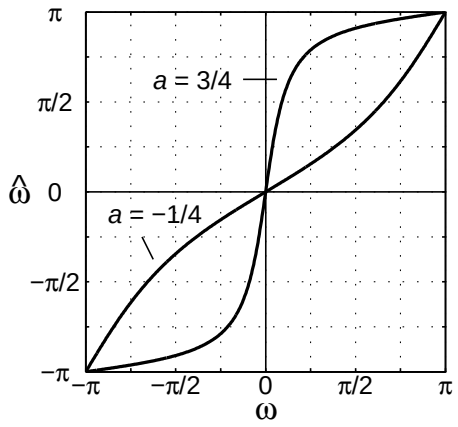

Figure 2.5: **Frequency warping functions for two real values of a.**

minimum at $\omega = \angle -a$ ($\omega = \pi$ for positive, real a). Maximum stretching occurs at the slope's maximum, and maximum compression occurs at the slope's minimum.

A complex warping parameter allows the region of maximum stretching to be placed at any frequency, but a real signal, which has a spectrum that is symmetrical about $\omega = 0$, will be transformed into a complex signal, for which $F(\hat{\omega}) \neq F^*(-\hat{\omega})$. This is demonstrated by the complex warping function in figure 2.6. To use complex warping parameters while keeping the signal real, a higher order system is required in which each complex warping parameter is matched with its complex conjugate.

Recall that the SW transform samples the unit circle of the z-plane, and redistributes the samples evenly around the unit circle in the $\hat{z}$-plane. To calculate where the samples are located in the z-plane, the inverse of (2.4) is applied to the evenly spaced samples in the $\hat{z}$-plane. By defining a vector of the output frequency samples

$$\hat{\boldsymbol{\omega}} = \begin{pmatrix} 0 \\ \frac{2\pi}{M} \\ \vdots \\ \frac{2\pi(M-1)}{M} \end{pmatrix} \tag{2.21}$$

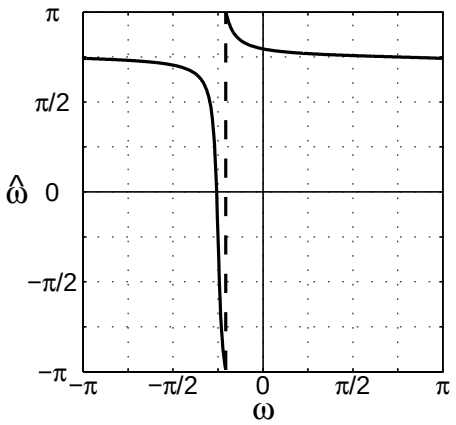

Figure 2.6: **A complex first-order all-pass frequency warping function.** The warping parameter is $a = {}^{3}/_{4}(1-j)$. A real signal ($F(\omega) = F^*(-\omega)$) will become complex ($F(\hat{\omega}) \neq F^*(-\hat{\omega})$) after warping with a first-order complex all-pass warping function.

(where M is the number of samples) and defining the special vector/matrix exponent operations

$$x^{\mathbf{y}} = \begin{pmatrix} x^{y_0} \\ x^{y_1} \\ \vdots \end{pmatrix} \qquad x^{\mathbf{Y}} = \begin{pmatrix} x^{Y_{0,0}} & x^{Y_{0,1}} & \cdots \\ x^{Y_{1,0}} & x^{Y_{1,1}} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{2.22}$$

for any scalar x, any vector $\mathbf{y}$ and any matrix $\mathbf{Y}$, $\hat{z}$-plane sample locations can be specified as

$$\hat{\mathbf{z}} = e^{j\hat{\boldsymbol{\omega}}} \tag{2.23}$$

Applying the inverse warping (assuming real a):

$$\mathbf{z} = \Theta^{-1}(\hat{\mathbf{z}}) = \frac{\hat{\mathbf{z}} + a}{1 + a\hat{\mathbf{z}}} \tag{2.24}$$

$$\Rightarrow \quad \boldsymbol{\omega} = \theta^{-1}(\hat{\boldsymbol{\omega}}) = \tan^{-1} \frac{(1-a^2)\sin\hat{\boldsymbol{\omega}}}{(1+a^2)\cos\hat{\boldsymbol{\omega}} + 2a} \tag{2.25}$$

$$\Rightarrow \quad \omega_m = \theta^{-1}(\hat{\omega}_m) = \tan^{-1} \frac{(1-a^2)\sin\frac{2\pi m}{M}}{(1+a^2)\cos\frac{2\pi m}{M} + 2a} \qquad m \in \{0, 1, ..., M-1\} \tag{2.26}$$

A first-order all-pass mapping with real poles is of particular interest, because there exists an IIR (infinite impulse response) implementation of the SW transform for mapping functions of this form. This will be discussed in chapter 3.

2.1.3 Higher-Order All-Pass Mapping

More control over how the z-plane is warped can be achieved by adding additional poles. An N^{th}-order all-pass function of the form

$$\Theta_N(z) = \prod_{n=1}^{N} \frac{z - a_n}{1 - a_n^* z} \qquad |a| < 1 \tag{2.27}$$

has N poles at $(a_n^*)^{-1}$ and N zeroes at a_n. By extension of equations (2.12) and (2.13), it can be seen that the frequency warping given by N^{th}-order functions is

$$\theta_N = \tan^{-1} \frac{\text{Im}\left\{\prod_{n=1}^{N}(e^{j\omega} - a_n)(1 - a_n e^{-j\omega})\right\}}{\text{Re}\left\{\prod_{n=1}^{N}(e^{j\omega} - a_n)(1 - a_n e^{-j\omega})\right\}}. \tag{2.28}$$

Figure 2.7 plots a second-order all-pass warping with parameters $a_1 = {}^{-3}/_4$ and $a_2 = {}^{3}/_4$. The figure illustrates that, in general, a real signal ($F(\omega) = F^*(-\omega)$) becomes complex ($F(\hat{\omega}) \neq F^*(-\hat{\omega})$) after warping with a second-order all-pass function. Also note the many-to-one (non-invertible) nature of the $\theta_N(\omega)$.

A real output is achieved from a second-order warping by making the warping parameters conjugate symmetrical, such as the warping function plotted in figure 2.8.The warping parameters are $a_1 = {}^{(1+j)}/_{2\sqrt{2}}$ and $a_2 = {}^{(1-j)}/_{2\sqrt{2}}$. Note how a real signal remains real after warping ($F(\omega) = F^*(-\omega) \Rightarrow F(\hat{\omega}) = F^*(-\hat{\omega})$).

The z-plane mapping shown in figure 2.9 is equivalent to the frequency warping of figure 2.8. The warping parameters are $a_1 = {}^{(1+j)}/_{2\sqrt{2}}$ and $a_2 = {}^{(1-j)}/_{2\sqrt{2}}$. The grid is not shown in this figure because it wraps right around

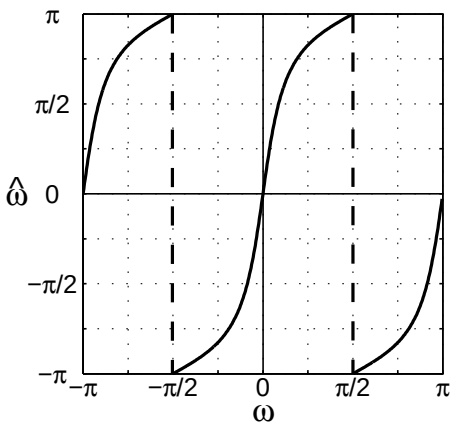

Figure 2.7: **A second-order all-pass frequency warping function.**

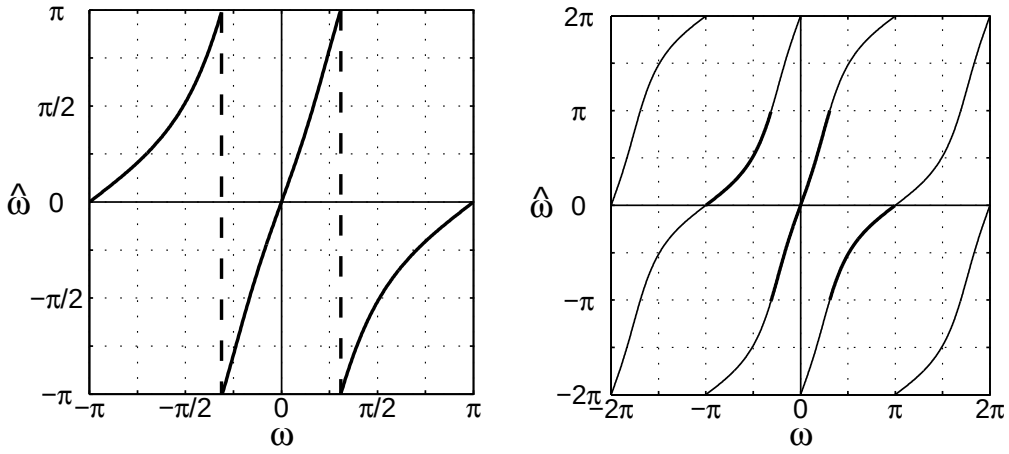

Figure 2.8: **A complex-conjugate second-order all-pass frequency warping function.** The two graphs show the same function, plotted on different scales.

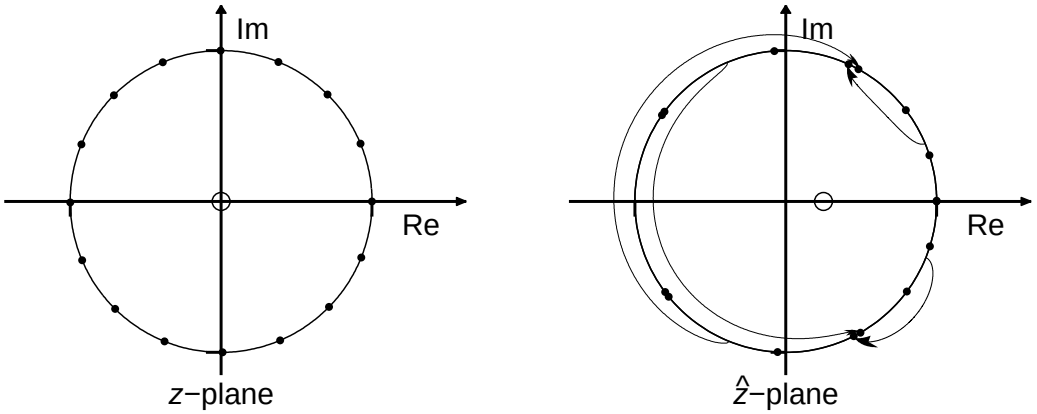

Figure 2.9: **A complex-conjugate second-order all-pass mapping of the z-domain.**

the unit circle and crosses over itself. Instead, arrows are drawn for two complex-conjugate sample pairs in the $\hat{z}$-plane, showing where they came from in the z-plane. These arrows illustrate that although some samples are moved from negative frequencies to positive, conjugate symmetry is preserved. They also show that distant z-plane samples can map to the same $\hat{z}$-plane location, highlighting the aliasing that occurs from this many-to-one mapping.

Figures 2.7 and 2.8 show that there are two possible frequencies ω that could have been transformed to create each warped frequency $\hat{\omega}$. Thus it is not possible to calculate the original frequency ω from a given warped frequency $\hat{\omega}$, which means second-order warping functions are not invertible. A consequence of this many-to-one mapping is that $\Theta_N(z)$ and $\theta_N(\omega)$ are not generally invertible for $N > 1$. This makes it difficult to correctly position the samples in the z-plane to make them evenly spaced in the $\hat{z}$-plane.

2.1.4 Piecewise-Linear Mapping

It is possible to choose any arbitrary function $\theta(\omega)$, rather than deriving it from a known z-domain warping function $\Theta(z)$. In fact, $\Theta(z)$ does not need to be defined for all z — it only needs to be defined on the unit circle. The

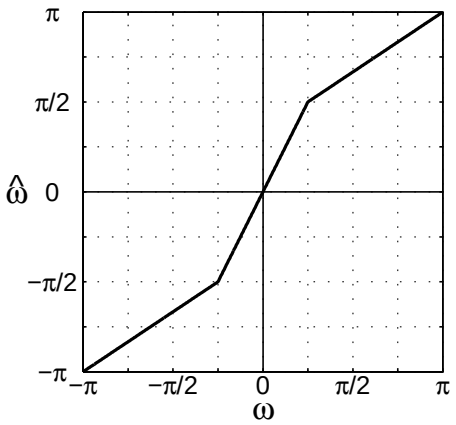

Figure 2.10: **A piecewise-linear frequency warping function.**

z-domain warping function $\Theta(z)$ is derived from a given frequency warping $\theta(\omega)$ using the mapping $z = e^{j\omega}$ at its inverse $\omega = -j \ln z$:

$$\Theta(z) = e^{j\theta(\omega)} = e^{j\,\theta(-j \ln z)} \tag{2.29}$$

Piecewise-linear functions are useful in that their slope is constant (and, therefore, the extent of the warping) over frequency bands. Consider the function

$$\theta(\omega) = \begin{cases} {}^{(2\omega-\pi)}/_{3} & -\pi \;\; < \omega \leqslant -{}^{\pi}/_{4} \\ 2\omega & -{}^{\pi}/_{4} < \omega \leqslant \;\; {}^{\pi}/_{4} \\ {}^{(2\omega+\pi)}/_{3} & \;\; {}^{\pi}/_{4} < \omega \leqslant \;\; \pi \end{cases} \tag{2.30}$$

displayed in figure 2.10. Frequencies below $|\omega| = \frac{\pi}{4}$ are expanded by a factor of 2 and frequencies above $|\omega| = \frac{\pi}{4}$ are compressed by a factor of 1.5.

2.2 Bandwidth, Time and Amplitude Distortion Effects

The centre frequency, bandwidth, duration and amplitude of a signal (or component of a signal) are intimately related to each other — changing one

of these parameters must affect the other parameters. This section describes how the SW transform changes a signal with regard to these parameters.

2.2.1 Bandwidth Distortion

As a band-limited signal undergoes SW, its bandwidth is altered according to the slope of $\theta(\omega)$, as illustrated in figure 2.11. A signal's bandwidth $B(\omega_{\max}, \omega_{\min})$ is defined as the difference between its lowest non-zero frequency component $\omega_{\min}$ and its highest non-zero frequency component $\omega_{\max}$.

$$B(\omega_{\max}, \omega_{\min}) = \omega_{\max} - \omega_{\min} \tag{2.31}$$

Similarly, a warped signal bandwidth is given by

$$\hat{B}(\hat{\omega}_{\max}, \hat{\omega}_{\min}) = \hat{\omega}_{\max} - \hat{\omega}_{\min} \tag{2.32}$$

where $\hat{\omega}_{\min}$ and $\hat{\omega}_{\max}$ are the lowest and highest frequency components of the warped signal, respectively. For a first-order real all-pass warping, $\hat{\omega}_{\min} = \theta_1(\omega_{\min})$ and $\hat{\omega}_{\max} = \theta_1(\omega_{\max})$ (this is not necessarily true for higher-order warping functions). The ratio of the warped signal's bandwidth to that of the original signal, denoted by $S(\omega_{\max}, \omega_{\min}, \hat{\omega}_{\max}, \hat{\omega}_{\min})$, describes the bandwidth distortion resulting from the SW.

$$S(\omega_{\max}, \omega_{\min}, \hat{\omega}_{\max}, \hat{\omega}_{\min}) = \frac{\hat{B}(\omega_{\max}, \omega_{\min})}{B(\omega_{\max}, \omega_{\min})} = \frac{\hat{\omega}_{\max} - \hat{\omega}_{\min}}{\omega_{\max} - \omega_{\min}} \tag{2.33}$$

For first-order real warping the bandwidth distortion is a function of only two frequency variables:

$$\begin{aligned} S(\omega_{\max}, \omega_{\min}) &= \frac{\theta_1(\omega_{\max}) - \theta_1(\omega_{\min})}{\omega_{\max} - \omega_{\min}} \\ &= \frac{\tan^{-1}\frac{\left(1-a^2\right)\sin\omega_{\max}}{\left(1+a^2\right)\cos\omega_{\max}-2a} - \tan^{-1}\frac{\left(1-a^2\right)\sin\omega_{\min}}{\left(1+a^2\right)\cos\omega_{\min}-2a}}{\omega_{\max} - \omega_{\min}} \end{aligned} \tag{2.34}$$

If a signal's bandwidth $B(\omega_{\max}, \omega_{\min})$ is sufficiently narrow, then an approximation to $\hat{B}(\hat{\omega}_{\max}, \hat{\omega}_{\min})$ can be found by multiplying $B(\omega_{\max}, \omega_{\min})$ with

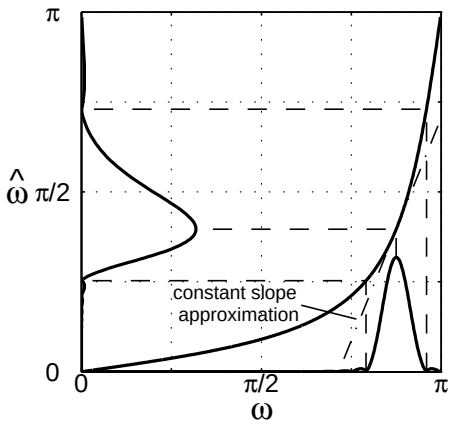

Figure 2.11: **Distortion of a signal's bandwidth.** An example signal spectrum is warped using a first-order warping with parameter $a = 3/4$.

the slope of $\theta(\omega)$ at the centre frequency ω_0. This approximate bandwidth distortion function is denoted by $\tilde{S}(\omega_0)$.

$$\begin{aligned}\tilde{S}(\omega_0) &= \left.\frac{\mathrm{d}\,\theta_1(\omega)}{\mathrm{d}\,\omega}\right|_{\omega=\omega_0} \\ &= \frac{1-a^2}{1-2a\cos\omega_0+a^2} \approx S(\omega_{\max},\omega_{\min})\end{aligned} \tag{2.35}$$

$$\hat{B}(\hat{\omega}_{\max},\hat{\omega}_{\min}) \approx B(\omega_{\max},\omega_{\min})\ \tilde{S}(\omega_0) \tag{2.36}$$

The error ratio $\mathcal{E}\left(S(\omega_{\max},\omega_{\min}),\tilde{S}(\omega_0)\right)$, defined as the absolute difference between the actual and approximate values divided by the actual value, is a four-dimensional function of ω_0, $\omega_{\min}$, $\omega_{\max}$ and a.

$$\begin{aligned}&\mathcal{E}\left(S(\omega_{\max},\omega_{\min}),\tilde{S}(\omega_0)\right) \\ &= \frac{\left|S(\omega_{\max},\omega_{\min})-\tilde{S}(\omega_0)\right|}{S(\omega_{\max},\omega_{\min})} \\ &= \frac{\left|\theta_1(\omega_{\max})-\theta_1(\omega_{\min})-\frac{(\omega_{\max}-\omega_{\min})(1-a^2)}{1-2a\cos\omega_0+a^2}\right|}{\theta_1(\omega_{\max})-\theta_1(\omega_{\min})}\end{aligned} \tag{2.37}$$

As an example, the accuracy of the constant-slope approximation for $f[n]$, where $f[n]$ is a sinusoid of frequency ω_0 that has been windowed using

a Hanning window function of length L, beginning at sample N_1, is

$$f[n] = \begin{cases} \cos\omega_0 n \times \frac{1}{2}\left[1 + \cos\frac{2\pi}{L}\left(n - N_1 - \frac{L}{2}\right)\right], & |n - N_1| \leq \frac{L}{2} \\ 0, & \text{otherwise} \end{cases} \tag{2.38}$$

This signal has the frequency spectrum:

$$F(\omega) = e^{-j\omega\left(N_1 + \frac{L}{2}\right)} \frac{L}{4\,\pi}\left[\tilde{F}(\omega + \omega_0) + \tilde{F}(\omega - \omega_0)\right] \tag{2.39}$$

where

$$\tilde{F}(\omega) = \frac{\sin(\pi L\omega)}{L\omega} + \frac{1}{2}\left(\frac{\sin\left(\pi\left(L\omega + 1\right)\right)}{L\omega + 1} + \frac{\sin\left(\pi\left(L\omega - 1\right)\right)}{L\omega - 1}\right)$$

The main lobe of this spectrum has a width of $B_{hanning} = 8\pi/L$, spanning the band $\omega_0 - {}^{4\pi}/_{L}$ to $\omega_0 + {}^{4\pi}/_{L}$. Note that the value for the bandwidth given here is not strictly the difference between the maximum and minimum frequencies (because the side lobes have a finite amplitude), but it is a useful measure of bandwidth for this example. The warped spectrum, $\hat{F}(\omega) = F\left(\theta^{-1}(\omega)\right)$, has a main lobe width of

$$\begin{aligned} \hat{B}(\hat{\omega}_{\max}, \hat{\omega}_{\min}) = \tan^{-1}\frac{(1-a^2)\sin\left(\omega_0 + \frac{4\pi}{L}\right)}{(1+a^2)\cos\left(\omega_0 + \frac{4\pi}{L}\right) - 2a} - \\ \tan^{-1}\frac{(1-a^2)\sin\left(\omega_0 - \frac{4\pi}{L}\right)}{(1+a^2)\cos\left(\omega_0 - \frac{4\pi}{L}\right) - 2a} \end{aligned} \tag{2.40}$$

Figure 2.12 shows bandwidth stretching, $\hat{B}(\hat{\omega}_{\max}, \hat{\omega}_{\min})/B_{hanning}$, for three signals of the form of (2.38), each having a different initial bandwidth $B_{hanning}$ (resulting from different window lengths L). Figure 2.13 plots the error resulting from the bandwidth stretching approximation $\tilde{S}(\omega_0)$ for each of the three signals. Three important features of the bandwidth stretching approximation are highlighted by figure 2.13. The first is that the approximation is exact along the line $a = 0$. This is sensible, because for $a = 0$ the function $\theta_1(\omega)$ is a straight line, so the slope over the whole band is equal to the slope at ω_0. The second feature that can be noted is that the approximation is very good for

$$a = cos(\omega_0) \tag{2.41}$$

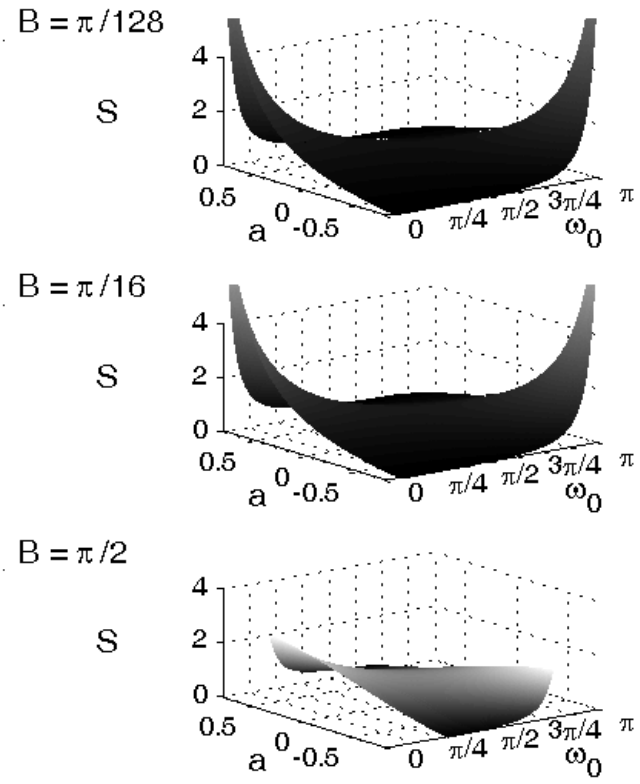

Figure 2.12: **The bandwidth stretching function.** The independent-variable axes are the warping factor a and the centre frequency ω_0.

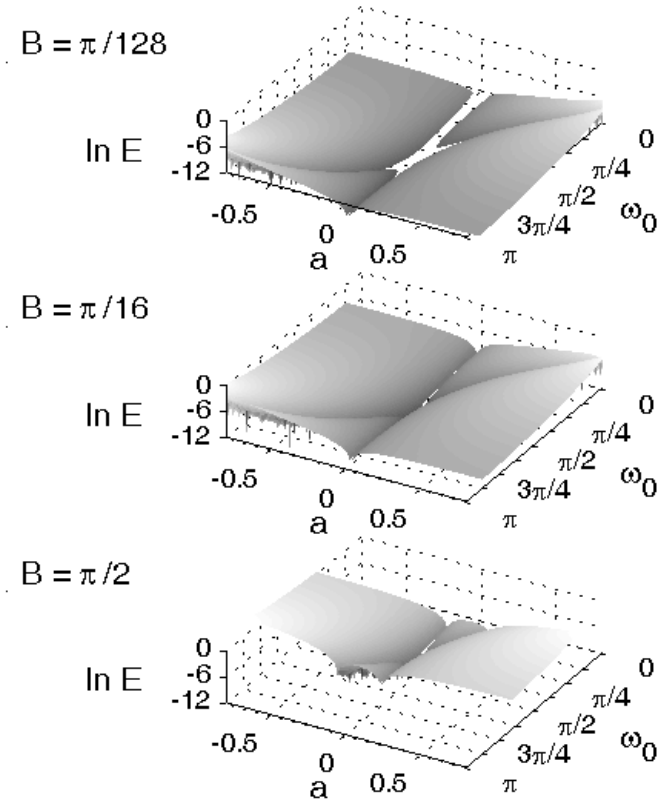

Figure 2.13: **The error $\mathcal{E}$ resulting from using the approximation $\tilde{S}(\omega_0)$.** The independent-variable axes are the warping factor a and the centre frequency ω_0. The logarithm of the error is plotted to more clearly show the features of the error function, which are: there is no error where $a = 0$, the error is small when $a = cos(\omega_0)$ and the error increases with the bandwidth of the signal (B).

This relates to the "knee" of the curve $\theta_1(\omega)$ — the point where the slope equals unity. Around this point, the error on one side of the centre frequency is mostly cancelled out by the error on the other side. Finally, the error is small for narrow-band signals. This is what is expected, because, for a narrow-band signal, the slope of $\theta_1(\omega)$ does not change much over the signal's spectrum.

2.2.2 Time Distortion

There is a reciprocal relationship between the time and frequency domains: the duration of the signal is reduced if the bandwidth is increased, and vice versa. If the duration of the unwarped signal is T and the duration of the warped signal is $\hat{T}$, then

$$\hat{T} = T s(\omega_{\min}, \omega_{\max}) \tag{2.42}$$

where $s(\omega_{\min}, \omega_{\max})$ is the duration distortion function, given by

$$s(\omega_{\min}, \omega_{\max}) = \frac{B(\omega_{\min}, \omega_{\max})}{\hat{B}(\hat{\omega}_{\min}, \hat{\omega}_{\max})} \tag{2.43}$$

By the same argument as that used in the previous section, the approximate change-of-duration function for a narrow-band signal is the reciprocal of the derivative of $\theta(\omega_0)$, and is denoted by $\tilde{s}(\omega_0)$.

$$\tilde{s}(\omega_0) = \left(\frac{\mathrm{d}\,\theta(\omega_0)}{\mathrm{d}\,\omega_0} \right)^{-1} \tag{2.44}$$

So, for a real first-order all-pass warping, the function is:

$$\tilde{s}_1(\omega_0) = \frac{1 - 2a\cos\omega_0 + a^2}{1 - a^2} \qquad a \in \mathbb{R} \tag{2.45}$$

($\mathbb{R}$ being the set of all real numbers). The subscript 1 in the notation $\tilde{s}_1(\omega_0)$ indicates that function relates to a first-order warping. If this subscript is absent, a first-order warping is assumed.

A discrete sinusoidal input of frequency ω_0, starting at sample N_1 and finishing at N_2, will produce a sinusoidal output of frequency $\hat{\omega}_0 = \theta(\omega_0)$,

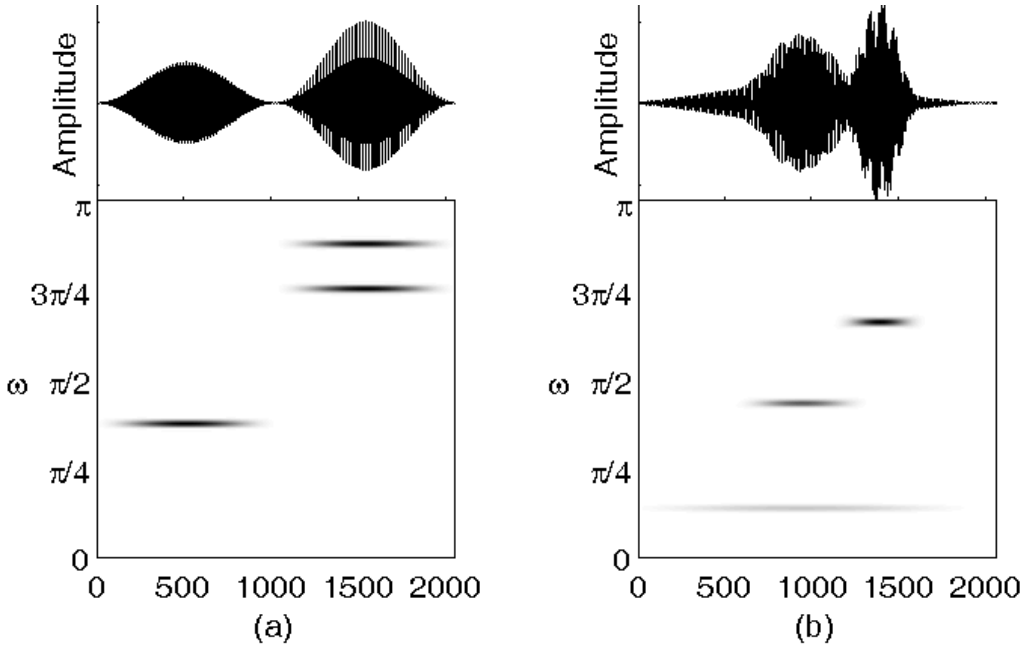

Figure 2.14: **Different frequency components within a signal are stretched/compressed and shifted by different amounts.** (a) Before warping. (b) After warping.

spanning the time $\hat{N}_1$ to $\hat{N}_2$. Substituting N_1 and N_2 for N_0 in (2.46) below will give approximations to these values.

$$\begin{aligned}\hat{N}_0(N_0, \omega_{\min}, \omega_{\max}) &= N_0 s(\omega_{\min}, \omega_{\max}) \\ &\approx N_0 \tilde{s}(\omega_0) = N_0 \frac{1 - 2a\cos\omega_0 + a^2}{1 - a^2} \qquad a \in \mathbb{R}\end{aligned} \tag{2.46}$$

where ω_0 is the centre frequency.

Individual frequency components of a signal are shifted in time and altered in duration, as demonstrated in figure 2.14. Figure 2.14(a) shows the time domain plot and spectrogram of a signal consisting of three frequency components: $\omega_1 = 3\pi/8$, $\omega_2 = 6\pi/8$ and $\omega_3 = 7\pi/8$. The warped signal, warped using a first-order all-pass function with warping factor $a = -1/2$, is shown in figure 2.14(b). This particular spectral warping causes all the components to have a lower frequency. It also causes the two components

of frequencies above $\pi/2$ to be condensed in duration and shifted towards the origin. The low-frequency component is expanded in duration. If this component originally had its starting point away from the origin, the warping would have caused its starting point to be shifted even further from the origin. Such separation of frequencies can be exploited for useful purposes; some of these uses will be mentioned in chapter 4.

The change in the starting location of frequency components that do not start at time zero can be simply explained by decomposing such a component into two parts: a cosinusoid that begins at the origin and extends to where the frequency component under consideration ends, and another cosinusoid of the same frequency but opposite phase that also begins at the origin but ends where the frequency component under consideration begins. This second sub-component cancels the first until the start of the frequency component under consideration. For example, say a signal contained a frequency component $f_i[n]$ of frequency ω_i that began at time N_1 and finished at time N_2.

$$f_i[n] = \begin{cases} 0 & n < N_1 \\ \cos\omega_i & N_1 \geq n \leq N_2 \\ 0 & n > N_2 \end{cases} \tag{2.47}$$

Decomposing into the two sub-components, denoted by $f_i^+[n]$ and $f_i^-[n]$, gives

$$\begin{aligned} f_i[n] &= f_i^+[n] + f_i^-[n] \\ f_i^+[n] &= \begin{cases} \cos\omega_i & n \leq N_2 \\ 0 & n > N_2 \end{cases} \\ f_i^-[n] &= \begin{cases} -\cos\omega_i & n < N_1 \\ 0 & n \geq N_1 \end{cases} \end{aligned} \tag{2.48}$$

The sub-component $f_i^-[n]$ is subject to the change-of-duration function $s(\omega)$ given by (2.43), therefore $s(\omega)$ also describes the change in starting location. So, for the above example, the warped frequency component will begin at $\hat{N}_1$

where

$$\begin{aligned}\hat{N}_1 &= N_1 s(\omega_i) \\ &\approx N_1 \tilde{s}(\omega_i) = \frac{1 - 2a\cos\omega_i + a^2}{1 - a^2}\end{aligned} \tag{2.49}$$

Of particular interest is the time separation of the frequency components that comprise an impulse signal. This is interesting because it produces a *spectrally-flat* chirp signal. What is meant by spectrally flat is that the Fourier transform of the chirp signal is constant. Different frequency components in the SW chirp signal have different amplitudes, but they all have the same energy (low-amplitude components have a longer duration). The use of such chirp signals will be described in chapter 4.

A unit impulse contains all frequencies, each of which is shifted to a different location in time when the signal is warped (provided that the impulse does not occur at time zero). An impulse at sample N_0, when warped, will spread out to range between $N_0 \min(s(\omega))$ and $N_0 \max(s(\omega))$. For real first-order all-pass warping,

$$\begin{aligned}\min(s(\omega)) \approx \max(\tilde{s}(\omega)) &= \begin{cases} \tilde{s}(0) = \frac{1-2a+a^2}{1-a^2} & a \geq 0 \\ \tilde{s}(\pi) = \frac{1+2a+a^2}{1-a^2} & a \leq 0 \end{cases} \\ &= \frac{1 - |a|}{1 + |a|}\end{aligned} \tag{2.50}$$

and

$$\max(s(\omega)) \approx \max(\tilde{s}(\omega)) = \frac{1 + |a|}{1 - |a|} \tag{2.51}$$

This means that the first and last non-zero samples will occur at approximately

$$\hat{N}_{\text{first}} = N_0 \frac{1 - |a|}{1 + |a|} \tag{2.52}$$

and

$$\hat{N}_{\text{last}} = N_0 \frac{1 + |a|}{1 - |a|} \tag{2.53}$$

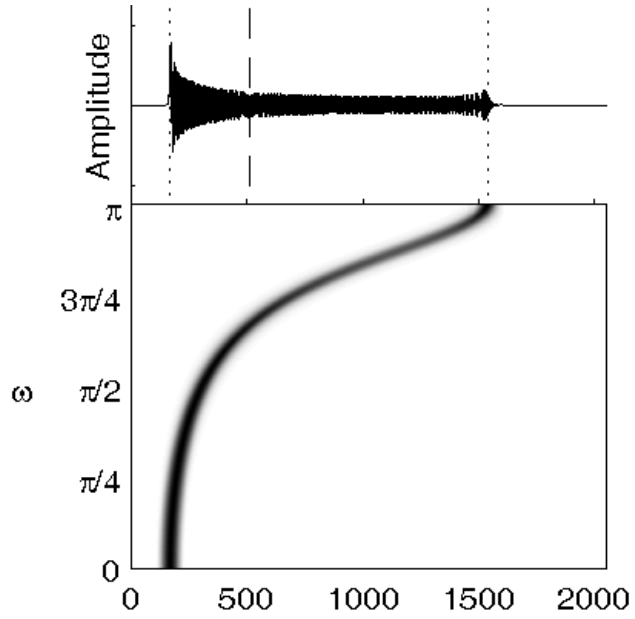

Figure 2.15: **Time domain plot and spectrogram of the result of spectrally warping a unit impulse** with warp parameter $a = 1/2$.

One way to understand this result is to consider a unit impulse as the sum of anti-phase cosinusoid pairs, as described in (2.48). If $\delta[n - N_0]$ represents a unit impulse at $n = N_0$, then

$$\delta[n - N_0] = \sum_{i=0}^{N-1} f_i^+[n - N_0] + f_i^-[n - N_0] \tag{2.54}$$

where

$$\begin{aligned} f_i^+[n - N_0] &= \cos \frac{2\pi i(n - N_0)}{N} \\ f_i^-[n - N_0] &= \begin{cases} -\cos \frac{2\pi i(n-N_0)}{N} & n \geq N_0 \\ 0 & n < N_0 \end{cases} \end{aligned} \tag{2.55}$$

The spreading of a unit impulse is shown in figure 2.15. The dashed line indicates the location of the impulse (at sample 512). The input signal was 1024 samples long, but the output was 2048 samples long to avoid time-domain aliasing (wrap-around). A positive warp parameter of $a = 1/2$ was

used, causing the low frequencies to be compressed in time and moved to the start of the sequence, and the high frequencies expanded and moved towards the end. All the frequencies are warped towards the Nyquist frequency. By applying (2.52) and (2.53), the first and last non-zero samples are found to occur at approximately sample position $512 \times 1/3 \approx 171$ and sample position $512 \times 3 = 1536$, respectively. These sample locations are indicated by the dotted lines.

2.2.3 Amplitude Distortion

The area under the (absolute value of) the time-domain curve of each frequency component is approximately conserved during a SW transformation (providing there are enough output samples to ensure that the signal is not truncated[1]). Therefore when a frequency component is stretched out in time it is attenuated. Similarly, it is amplified if it is squashed up in time. The expression for the magnitude gain response is the reciprocal of the change-of-duration factor:

$$\begin{aligned} |H(\omega)| &= \frac{1}{s(\omega_{\min}, \omega_{\max})} = \frac{\hat{B}}{B} = S(\omega_{\min}, \omega_{\max}) \\ &\approx \frac{\mathrm{d}\,\theta(\omega)}{\mathrm{d}\,\omega} \end{aligned} \tag{2.56}$$

For first-order warping with real a, the expression becomes

$$|H_1(\omega)| \approx \frac{1 - a^2}{1 - 2a\cos\omega + a^2} \tag{2.57}$$

2.3 Matrix Representation

The SW transform can be represented as a matrix multiplication. The following sub-sections contain a derivation of a matrix that performs spectral warping.

[1] See section 2.4 for a discussion on required signal lengths

2.3.1 The z-Transform Matrix

In order to derive the SW matrix, the z-transform matrix must be defined. To this end, let $f[n]$ be N samples of an input signal, and $F(z)$ the z-transform of $f[n]$, where $n = 0, 1, \ldots, N-1$ and $z \in \mathbb{Z}$, with $\mathbb{Z}$ being the set of all complex numbers.

$$F(z) = \sum_{n=0}^{N-1} f[n] z^{-n} \tag{2.58}$$

Now let M be the number of output samples to be generated, indexed by $m = 0, 1, \ldots, M-1$, and let $G[m]$ be the result of evaluating $F(z)$ at the M points on the unit circle given by $e^{-j\,\omega[m]}$.

$$G[m] = F(z)\Big|_{e^{-j\,\omega[m]}} \tag{2.59}$$

The frequency sample locations $\omega[m]$ are calculated according to the inverse of some frequency warping function $\theta(\omega)$:

$$\omega[m] = \theta^{-1}\big(\hat{\omega}[m]\big) \tag{2.60}$$

where

$$\hat{\omega}[m] = \frac{2\pi m}{M} \tag{2.61}$$

If the vectors $\mathbf{n}$, $\mathbf{f}_N$, $\boldsymbol{\omega}_M$ and $\mathbf{G}_M$ are defined as

$$\mathbf{n} = \begin{pmatrix} 0 & 1 & \cdots & N-1 \end{pmatrix} \tag{2.62}$$

$$\mathbf{f}_N = \begin{pmatrix} f[0] & f[1] & \cdots & f[N-1] \end{pmatrix}^T \tag{2.63}$$

$$\boldsymbol{\omega}_M = \begin{pmatrix} \omega[0] & \omega[1] & \cdots & \omega[M-1] \end{pmatrix}^T \tag{2.64}$$

and

$$\mathbf{G}_M = \begin{pmatrix} G[0] & G[1] & \cdots & G[M-1] \end{pmatrix}^T \tag{2.65}$$

then equations (2.58) and (2.59) can be rewritten as

$$\mathbf{G}_M = \mathbf{H}_{M,N}\mathbf{f}_N \tag{2.66}$$

where

$$\mathbf{H}_{M,N} = e^{-j\,\boldsymbol{\omega}_M \mathbf{n}} = \begin{pmatrix} 1 & e^{-j\,\omega_0} & \cdots & e^{-(N-1)j\,\omega_0} \\ 1 & e^{-j\,\omega_1} & \cdots & e^{-(N-1)j\,\omega_1} \\ 1 & e^{-j\,\omega_2} & \cdots & e^{-(N-1)j\,\omega_2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-j\,\omega_{M-1}} & \cdots & e^{-(N-1)j\,\omega_{M-1}} \end{pmatrix} \tag{2.67}$$

(using the vector/matrix exponent operations of (2.22)).

The elements of the vector $\boldsymbol{\omega}_M$ are the frequencies at which the Fourier transform of the input signal is sampled, and the $M \times N$ matrix $\mathbf{H}_{M,N}$ is the z-*transform matrix* representing (2.58). The m^{th} row of $\mathbf{H}_{M,N}$ gives the weights for determining the m^{th} sample in the z-domain.

2.3.2 The Spectral Warping Matrix

The elements of $\boldsymbol{\omega}_M$ are points that are arbitrarily placed on the unit circle in the z-plane, but are uniformly-distributed on the unit circle in the $\hat{z}$-plane. The inverse DFT translates the evenly spaced samples in the $\hat{z}$-domain to the time domain. Spectral warping is implicit when the arbitrarily-placed z-domain samples are treated as evenly spaced ($\hat{z}$-domain) samples. Taking an inverse DFT of $G[m]$, then, results in the time-domain samples of the spectrally warped signal. The inverse DFT of $G[m]$ is given by

$$g[m] = \frac{1}{M} \sum_{i=0}^{M-1} G[i]\, W_M^{-im} = \frac{1}{M} \sum_{i=0}^{M-1} F\left(e^{j\omega_i}\right) W_M^{-im} \tag{2.68}$$

where

$$W_M = e^{-j2\pi/M} \tag{2.69}$$

The M-point DFT matrix is defined as

$$\mathbf{D}_M = e^{-j\,\hat{\boldsymbol{\omega}}_M \mathbf{m}} \tag{2.70}$$

where

$$\mathbf{m} = \begin{pmatrix} 0 & 1 & \cdots & M-1 \end{pmatrix}. \tag{2.71}$$

The inverse M-point DFT matrix, then, is

$$\begin{aligned}\mathbf{D}_M^{-1} &= \frac{e^{j\,\hat{\boldsymbol{\omega}}_M \mathbf{m}}}{M} \\ &= \frac{1}{M}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_M^{-1} & W_M^{-2} & \cdots & W_M^{-(M-1)} \\ 1 & W_M^{-2} & W_M^{-4} & \cdots & W_M^{-2(M-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_M^{-(M-1)} & W_M^{-2(M-1)} & \cdots & W_M^{-(M-1)^2} \end{pmatrix}\end{aligned} \tag{2.72}$$

The vector $\mathbf{g}_M$ contains the time-domain output samples. Using the definition of (2.72), equation (2.68) can be written in matrix notation to find $\mathbf{g}_M$.

$$\mathbf{g}_M = \mathbf{D}_M^{-1}\mathbf{G}_M \tag{2.73}$$

Substituting (2.66) into (2.73) gives

$$\begin{aligned}\mathbf{g}_M &= \mathbf{D}_M^{-1}\mathbf{H}_{M,N}\,\mathbf{f}_N \\ &= \mathbf{S}_{M,N}\,\mathbf{f}_N\end{aligned} \tag{2.74}$$

where N is the number of input samples and

$$\mathbf{S}_{M,N} = \mathbf{D}_M^{-1}\mathbf{H}_{M,N}. \tag{2.75}$$

$\mathbf{S}$ is referred to as the *SW transform matrix*. Equation (2.74) shows that each output sample is a linear combination of the input samples — hence the SW transform is indeed a linear transform.

The n^{th} column of $\mathbf{S}$ is the response of the transform to an impulse at the n^{th} input sample location; these N impulse responses are scaled by the corresponding input samples and added to produce the output sequence. The fact that the columns of $\mathbf{S}$ are not related by simple delays, but are quite different from each other, indicates that the transform is not shift invariant — delaying the input signal will change the shape of the output signal. Figure 2.16 shows the 1^{st}, 6^{th}, 11^{th} and 16^{th} columns of an example 32×16 SW transform matrix $\mathbf{S}$. This particular matrix is for a first-order all-pass warping, with a warping parameter $a = 1/8$.

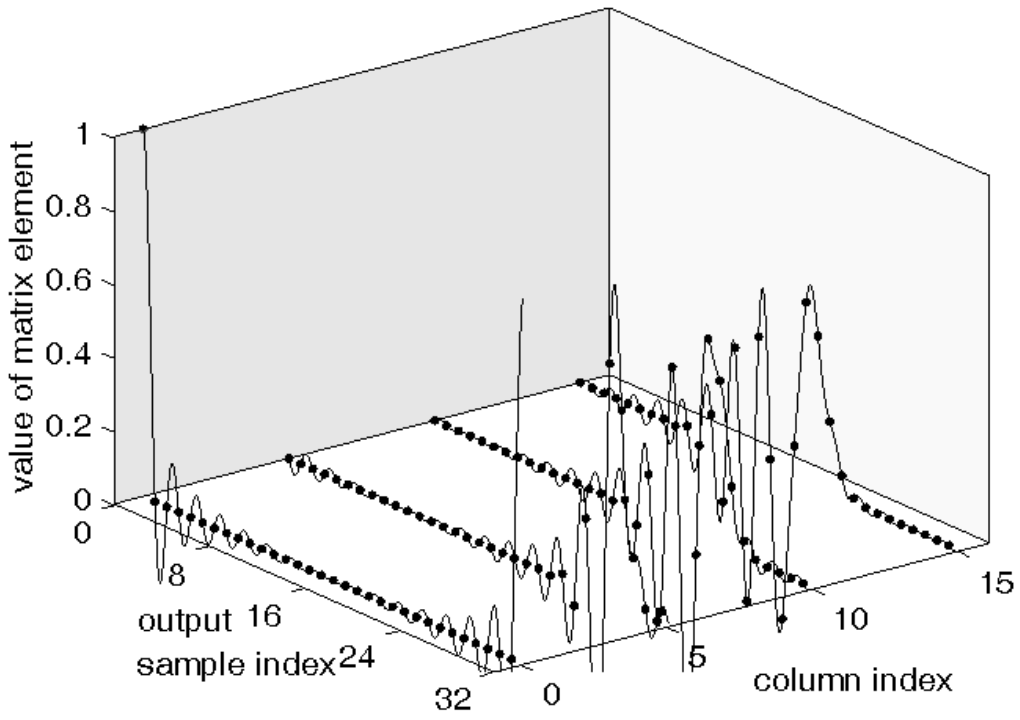

Figure 2.16: **Some columns of a SW transform matrix.**

2.3.3 The Frequency Interpolation Matrix

Rather than directly using the z-transform matrix, a DFT can be used to obtain N equally spaced samples. The M unequally spaced samples may be found by interpolation. This interpolation is made explicit by replacing the matrix $\mathbf{H}$ with a DFT matrix and an interpolation matrix. To explain this result, consider the conceptual process of spectral warping listed below.

1. A standard DFT of the N input samples is calculated, giving N uniformly spaced frequency domain samples.
2. These N frequency samples are interpolated at M arbitrary frequency locations.
3. The M samples are redistributed to be evenly spaced.
4. Finally, a standard inverse DFT transforms these M frequency samples into the M time-domain output samples.

The warping happens in step 3, when the M samples are moved (warped). The above list is essentially the same as the one on page 21, put into a form that directly corresponds to the following matrix operations:

$$\mathbf{g} = \overbrace{\mathbf{D}_M^{-1}}^{\text{step 4}} \overbrace{\mathbf{C}_{M,N}}^{\text{steps 2 \& 3}} \overbrace{\mathbf{D}_N}^{\text{step 1}} \mathbf{f}_N \tag{2.76}$$

where

$$\mathbf{C}_{M,N} = \mathbf{H}_{M,N}\mathbf{D}_N^{-1} \tag{2.77}$$

$$= \tfrac{1}{N}\, e^{-j\,\boldsymbol{\omega}_M \mathbf{n}}\, e^{j\,\boldsymbol{\omega}_N \mathbf{n}} \tag{2.78}$$

The matrix $\mathbf{C}$ is the *frequency interpolation matrix* for a given SW transform. The basic principle of the frequency interpolation is that, for each output frequency sample $G[m]\Big|_{m=i}$, $i \in \mathbf{m}$, an interpolation function is centred at the frequency location to which $G[m]\Big|_{m=i}$ corresponds, and the amplitude

of the function at the location of each input sample is the contribution that that sample makes to $G[m]\Big|_{m=i}$.

$$G[m] = \sum_{n=0}^{N} C[m,n]F[n] \tag{2.79}$$

$$\Rightarrow \qquad \mathbf{G}_M = \mathbf{C}_{M,N}\mathbf{F}_N \tag{2.80}$$

Therefore, each row of $\mathbf{C}$ gives one warped frequency sample as a linear combination of all of the non-warped (evenly spaced) samples given by the DFT. The type of interpolation performed by $\mathbf{C}$ will determine characteristics of the spectral warping. Some interpolation functions are described in the following sections.

2.3.4 Spectral Warping a Generalisation of Linear Filtering

If the frequency interpolation matrix $\mathbf{C}$ is square and diagonal, each output frequency sample equals the input sample of the same frequency scaled by a constant. In this case, SW is the same as filtering the input signal with an FIR filter whose coefficients are the diagonal elements of $\mathbf{C}$. SW can therefore be regarded as a generalisation of linear filtering.

A simple MATLAB function to test this is presented here:

```
function C = interp_matrix_as_filter( h, M, N )
R = M/N;
C = zeros( M, N );
C(1:R:M,:) = diag( fft( h, N ) )/R;
for i = 1:N
        C(:,i) = circonv( ones( R, 1 ), C(:,i) );
        C(:,i) = circonv( ones( R, 1 ), C(:,i) );
end
% rotate (first row --> last row)
C = [C(R:end,:); C(1:R-1,:)];
```

where the function CIRCONV performs circular convolution (see appendix C). If $M = N$ then the function returns a diagonal matrix with system response coefficients on the diagonal. If $M > N$ then the system response coefficients are interpolated down the columns to produce an almost diagonal non-square matrix. Using this function, filtering and up sampling can be performed together:

```
S = IDFT_MATRIX( M ) * ...
        interp_matrix_as_filter( h, M, N ) * ...
                DFT_MATRIX( N );
g = S*f;
```

where g is the result of filtering f with h, upsampled by a factor of M/N.

2.3.5 Sinc Interpolation

The Shannon-Whittaker-Kotel'nikov sampling theorem states that a band-limited signal may be reconstructed completely from time-domain samples of the signal[12], provided that the time interval between samples is less than half the period of the highest frequency component contained in the signal.

Sampling a signal causes it to become periodic in the frequency domain (the spectrum is repeated over an interval of the sample frequency f_s). To reconstruct the signal, the higher frequency images must be filtered out. To completely filter out the images, without affecting the amplitude of the primary image, the spectrum is multiplied by a rectangular filter function — this has the effect of convolving the time-domain samples with a continuous sinc function. That is,

$$f(t) = f[n] \otimes c(t) \tag{2.81}$$

where $c(t)$ is the interpolation function:

$$c(t) = \operatorname{sinc} t = \frac{\sin \pi t}{\pi t} \tag{2.82}$$

Therefore

$$f(t) = \sum_n f[n] \frac{\sin \pi(t - nT_s)}{\pi(t - nT_s)} \tag{2.83}$$

where T_s is the sample period. In words, (2.83) states that the value of the continuous function at time t is the sum of the value of the sinc interpolation functions that have been time shifted by nT_s (over all n), each evaluated at time t. If the sampling frequency is too low, such that the criterion for the sampling theorem is not met, then the spectral images will overlap, making it impossible to recover the original spectrum.

If the discrete-time signal is to be resampled, and the original continuous-time signal is not required, then alternative interpolation functions can be employed — they only need to satisfy

$$f[m] = f(t)\Big|_{t=mT_s} \tag{2.84}$$

That is, the result of the interpolation only needs to equal the original signal at the output sample locations.

The sinc interpolation function has the necessary property that it has a value of unity at the origin and zero at all the other sample locations: an output sample at the same location as an input sample will naturally be the same as that input sample (100% contribution from that input sample and no contribution from any of the other input samples).

2.3.6 Interpolation in the Frequency Domain

In this thesis, the Fourier domain of a sampled time-domain signal is referred to as the *discrete-time frequency domain*. This distinguishes between the domain of the non-periodic Fourier transform of the continuous signal (simply referred to as the *frequency domain*) and that of the periodic Fourier transform of the sampled signal. Taking the discrete Fourier transform (DFT) of a sampled signal produces a sampled representation of the signal's spectrum; this is the *discrete frequency domain* representation. Hence, the DFT operation is equivalent to a discrete-time Fourier transform, followed by sampling of the continuous discrete-time frequency domain.

Sampling and interpolation in the discrete-time frequency domain are similar to that in the time domain; the difference is that the discrete-time frequency domain is periodic over 2π, so any interpolation function used must be similarly periodic. The spectrum is sampled by multiplying by an impulse train of frequency-period $2\pi/N$. This multiplication causes the signal to repeat in time every NT_s (to be able to completely reconstruct the spectrum, NT_s needs to be at least the duration of the signal).

To interpolate between the frequency-domain samples, a process similar to reconstructing a signal from time-domain samples is applied. This interpolation requires a function that, when centred on a frequency-domain sample (time-domain period), will cause that sample to fully contribute to the output and every other sample have no contribution. This, again, corresponds to a sinc function (in this case, the function is in the frequency domain), with the exception that the function must be periodic over 2π (the frequency domain must still be considered periodic, because the signal is of discrete nature in the time domain). The interpolation function being periodic over 2π means that the contribution from a sample at a given location will include the contributions from the corresponding samples at all the periodic images.

For SW, the interpolation is for the purpose of resampling, so the relaxed constraint (2.84), translated to the frequency domain, applies:

$$F[m] = F(\omega)\Big|_{\omega=\omega[m]} \tag{2.85}$$

The sinc-like function that satisfies this constraint for M evenly spaced output samples is

$$C(\omega) = e^{jD\omega}\frac{\sin\frac{D\omega}{2}}{D\sin\frac{\omega}{2}} = \frac{1}{D}\frac{e^{\frac{j\,3D\omega}{2}} - e^{\frac{jD\omega}{2}}}{e^{\frac{j\omega}{2}} - e^{\frac{-j\omega}{2}}} \tag{2.86}$$

where D is any common multiple of M and N:

$$D = i\,\mathrm{lcm}\,\{M, N\} \qquad i \in \mathbb{I}, \quad i > 0 \tag{2.87}$$

$\mathbb{I}$ is the set of all integers and $\mathrm{lcm}\{M, N\}$ denotes the least common multiple of M and N. The least common multiple of the two integers M and N is

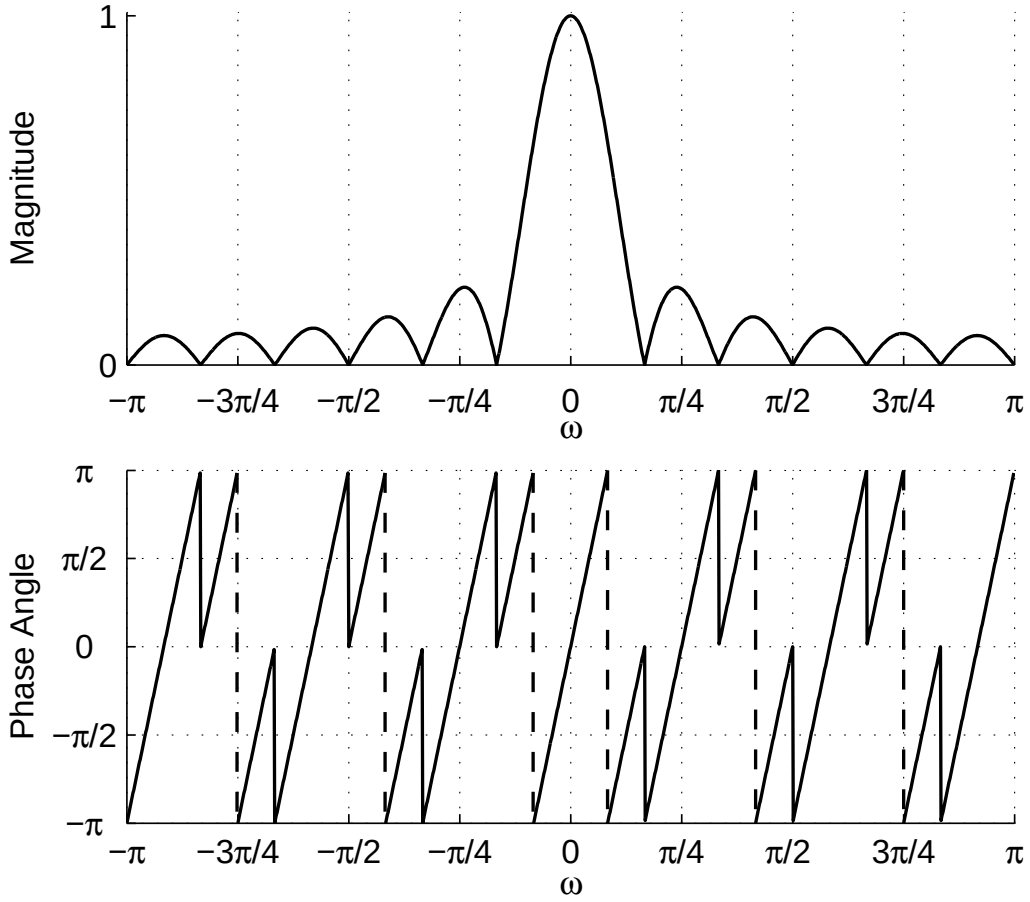

Figure 2.17: **The modified sinc interpolation function** $C(\omega)$.

$MN/\gcd\{M,N\}$, where $\gcd\{M,N\}$ is the greatest common divisor of M and N.[2]

The function (2.87) is plotted in figure 2.17, using $i = 1$ and with (number of input samples) $N = 3$ and (number of output samples) $M = 4$. The

[2]The greatest common divisor of the two integers a and b are found using the recursive Euclidean algorithm[19]:

$$\gcd\{a,b\} = \begin{cases} |a|, & b = 0 \\ |b|, & a = 0 \\ \gcd\left\{\min(|a|,|b|),\, ||a|-|b||\right\}, & \text{otherwise} \end{cases}$$

$(\sin {}^{\omega}/_{2})^{-1}$ term causes the function to be periodic over 2π. The phase term, $e^{jD\omega}$, causes the function to wrap around between samples, ensuring that the phase is correct at the sample locations. An example of this function used for resampling a spectrum is given in figure 2.18, where the function $e^{j\pi \sin\omega}\frac{1}{2}(1+\cos\omega)$ is sampled at the three locations $\omega = \{-\pi, {}^{-\pi}/_{3}, {}^{\pi}/_{3}\}$ (indicated by $\bullet$), interpolated using the interpolation function

$$C(\omega) = e^{jMN\omega}\frac{\sin\frac{MN\omega}{2}}{\left(MN\sin\frac{\omega}{2}\right)} \tag{2.88}$$

then evaluated at the four locations $\omega = \{-\pi, {}^{-\pi}/_{2}, 0, {}^{\pi}/_{2}\}$ (indicated by $*$). The dotted line is the underlying continuous function and the solid line is the interpolated function $\tilde{F}(\omega) = \sum_n F[n]C\left(\omega - {}^{n\pi}/_{N}\right)$.

Any function $C(\omega)$ where

$$C(\omega) = e^{jD\omega}\frac{\sin\frac{D\omega}{2}}{D\sin\frac{\omega}{2}} \qquad \forall \quad \omega = \frac{2\pi m}{M}, \quad m = 0, 1, \ldots, M-1 \tag{2.89}$$

is sufficient for resampling the spectrum at the points $\omega = \omega[m]$. So, for a given M, it may be possible to simplify (2.86). For example, if $M = 4$ a suitable interpolation function is

$$C_{M=4}(\omega) = \tfrac{1}{2}\cos\omega\,(1+\cos\omega) \tag{2.90}$$

which is more simple than (2.89) in the sense that it is of lower frequency. An example of the use of this simplified function used for resampling a spectrum is given in figure 2.19. The underlying function (the dotted line) is sampled at three locations ($\bullet$), interpolated using the function $\frac{1}{2}\cos\omega\,(1+\cos\omega)$, then resampled at four locations ($*$). The solid line is the interpolated function $\sum_n F[n]C\left(\omega - {}^{n\pi}/_{N}\right)$. This interpolation is exact for $M = 4$ and will produce the same SW as the modified sinc interpolation of (2.86).

To reconstruct a continuous-frequency spectrum, independent of the number and location of the output samples, a regular sinc interpolation function is used — the interpolation function no longer needs to be periodic over 2π. This is not the case with the SW transform, which is defined entirely in the

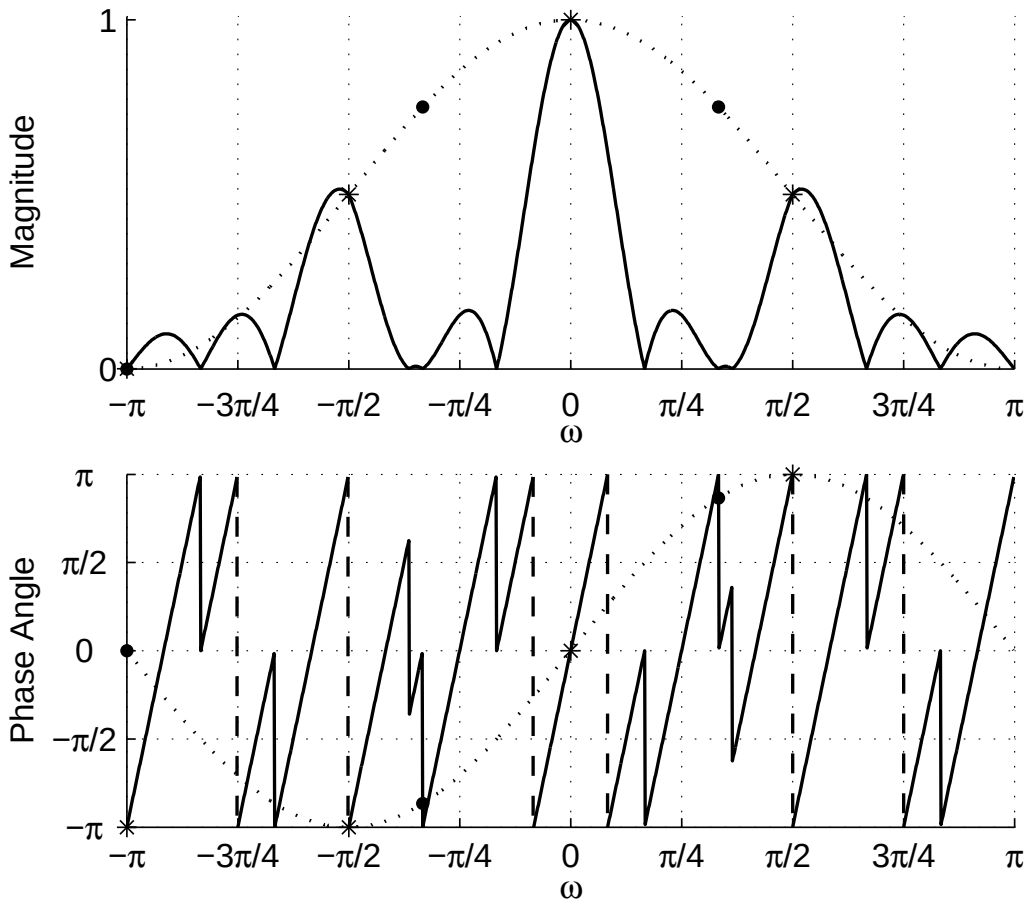

Figure 2.18: **The modified sinc interpolation function used to resample a discrete-time signal.** The dots (•) are the samples of the original sequence (the dotted line is the underlying function) and stars (∗) are the samples of the reconstructed sequence (the solid line is the underlying function of the reconstructed signal).

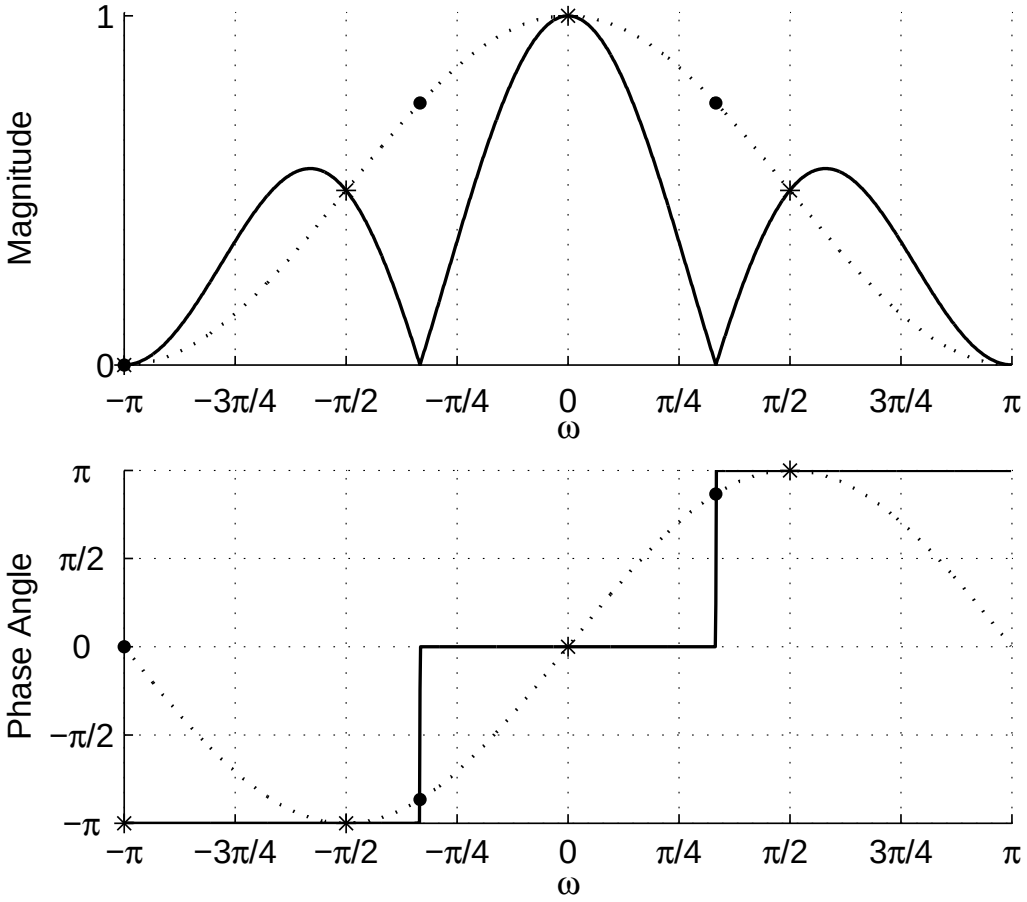

Figure 2.19: **The simplified interpolation function ($M = 4$) used to resample a discrete-time signal.** The dots (•) are the samples of the original sequence (the dotted line is the underlying function) and stars ($*$) are the samples of the reconstructed sequence (the solid line is the underlying function of the reconstructed signal).

digital domain: the SW interpolation functions will always be periodic. However, the integer i in (2.87) can be chosen arbitrarily high to approximate a continuous spectrum.

2.3.7 Sampling the Frequency Domain at Arbitrary Locations

To sample the spectrum at arbitrary locations, the interpolation function is generated as though it were for uniform sampling, with enough sample points to ensure that the value of the interpolated function equals the true value of the spectrum at all the desired output sample locations. To do this, (2.86) is evaluated while setting

$$D = i \operatorname{lcm}\left\{\frac{2\pi}{\gcd\{\omega[m]\}}, N\right\} \tag{2.91}$$

where $\gcd\{\omega[m]\}$ is the greatest common divisor of all the output sample locations.[3]

For example, if a spectrum were initially sampled at intervals of $\pi/4$ and then were to be resampled at $\omega = \{-\pi/3, -\pi/8, 0, \pi/5, \pi/3\}$, then

$$\begin{aligned} D &= \operatorname{lcm}\left\{\frac{2\pi}{\gcd\{-\pi/3, -\pi/8, 0, \pi/5, \pi/3\}}, 8\right\} \\ &= \operatorname{lcm}\{120, 8\} \\ &= \frac{120 \times 8}{\gcd\{120, 8\}} = 120 \end{aligned}$$

and

$$\begin{aligned} F(\omega[m]) &= \sum_n f[n]\, e^{j120(\omega[m]-n\pi/N)} \frac{\sin 60(\omega[m]-n\pi/N)}{120 \sin \frac{(\omega[m]-n\pi/N)}{2}} \\ \omega[m] &= \{-\pi/3, -\pi/8, 0, \pi/5, \pi/3\} \end{aligned}$$

[3]The greatest common divisor of a set of more than two numbers can be reduced using

$$\gcd\{a_0, a_1, \ldots, a_n\} = \gcd\{\gcd\{a_0, a_1\}, a_2, \ldots, a_n\}$$

2.3.8 Constructing the Interpolation Matrix

An interpolation matrix $\mathbf{C}$ is constructed by sampling series of frequency-shifted $C(\omega)$ functions at the desired output frequencies. If $c_{m,n}$ is the n^{th} element of the m^{th} row of $\mathbf{C}$, then

$$c_{m,n} = C\left(\omega - \tfrac{2\pi n}{N}\right)\Big|_{\omega=\hat{\omega}[m]} \tag{2.92}$$

Using the interpolation function of (2.86) gives

$$c_{m,n} = e^{jD(\hat{\omega}[m]-2\pi n/N)}\frac{\sin\frac{D(\hat{\omega}[m]-2\pi n/N)}{2}}{D\sin\frac{\hat{\omega}[m]-2\pi n/N}{2}} \tag{2.93}$$

Each row of $\mathbf{C}$ gives one of the warped frequency samples as a linear combination of the uniformly spaced samples of the standard DFT. As an example, consider the first-order all-pass warping where

$$\hat{\omega}[m] = \tan^{-1}\frac{(1-a^2)\sin\omega[m]}{(1+a^2)\cos\omega[m]-2a} \tag{2.94}$$

Each element of the interpolation matrix can be calculated by substituting this into (2.93):

$$\begin{aligned} c_{m,n} =& e^{jD\left(\tan^{-1}\frac{\left(1-a^2\right)\sin\omega[m]}{(1+a^2)\cos\omega[m]-2a}-\frac{2\pi n}{N}\right)} \quad \times \\ & \sin\left[\frac{D}{2}\left(\tan^{-1}\frac{(1-a^2)\sin\omega[m]}{(1+a^2)\cos\omega[m]-2a}-\frac{\pi n}{N}\right)\right] \quad \times \\ & \frac{1}{D}\sin\left[\frac{1}{2}\left(\tan^{-1}\frac{(1-a^2)\sin\omega[m]}{(1+a^2)\cos\omega[m]-2a}-\frac{\pi n}{N}\right)\right]^{-1} \end{aligned} \tag{2.95}$$

Figures 2.20 and 2.21 show some rows of the $\mathbf{C}$ matrix for the above example. Figure 2.20 shows the modified sinc function (for $a = 1/8$). The markers show the points at which the interpolation function is evaluated. The solid line with filled markers shows the real component and the dotted line with hollow markers shows the imaginary component. In figure 2.21, The 13^{th} row of a 64×16 matrix $\mathbf{C}$ (for $a = 1/8$) is plotted, showing the weights applied to each of the 16 input samples to calculate the 13^{th} output sample.

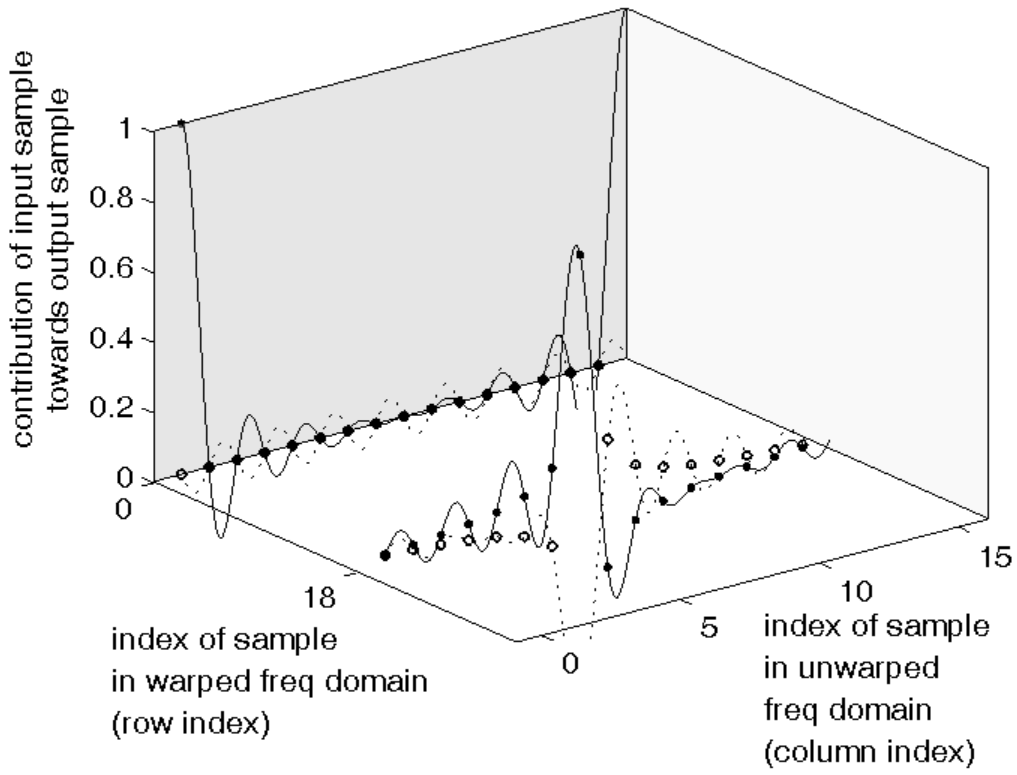

Figure 2.20: **Two example rows of a** 64×16 **interpolation matrix C**. The modified sinc function for warping factor $a = 1/8$ is plotted. The markers show the points at which the interpolation function is evaluated. The solid line with filled markers shows the real component and the dotted line with hollow markers shows the imaginary component.

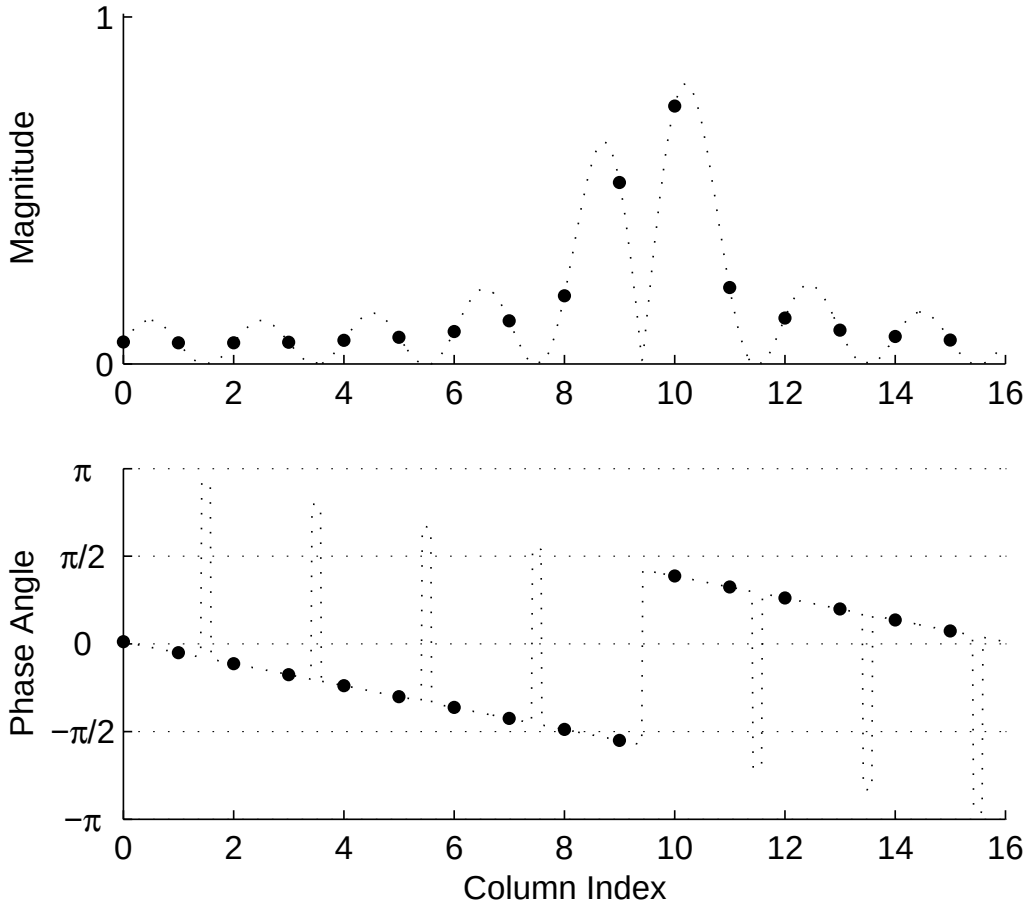

Figure 2.21: **One row of an interpolation matrix.**

The phase term in (2.95) has an interesting effect on the phase of the interpolation samples. It almost completely compensates for the phase reversal between adjacent nodes of the sinc function, resulting in adjacent coefficients having similar phase (as can be seen in figure 2.21).

Having constructed the frequency interpolation matrix $\mathbf{C}$, the z-transform matrix $\mathbf{H}$ and the SW matrix $\mathbf{S}$ can be computed:

$$\mathbf{H}_{M,N} = \mathbf{C}_{M,N}\mathbf{D}_N \tag{2.96}$$

$$\mathbf{S}_{M,N} = \mathbf{D}_M^{-1}\mathbf{C}_{M,N}\mathbf{D}_N \tag{2.97}$$

The SW arrived at using the modified sinc interpolation function (all-pass function) can be generalised by changing the interpolation function/matrix. One generalisation is achieved by using arbitrarily-placed samples around the unit circle (giving an arbitrary warp function) to generate the $\mathbf{H}$ matrix. The interpolation coefficients are then found by solving

$$\mathbf{C}_{M,N} = \mathbf{H}_{M,N}\mathbf{D}_N^{-1} \tag{2.98}$$

This also generates modified sinc interpolation functions.

An example of this approach is shown in figure 2.22, in which the piecewise-linear function of (2.30) is used to construct the $\mathbf{C}$ matrix. For the figure, the all-pass warping matrix has a warping factor of $a = 1/8$, and the piecewise-linear warping uses the mapping

$$\hat{\omega} = \begin{cases} \frac{2\omega + \text{sign}(\omega)\pi}{3}, & |\omega| \geq \frac{\pi}{4} \\ 2\omega & |\omega| \leq \frac{\pi}{4} \end{cases} \tag{2.99}$$

The 256 output samples are calculated from 32 input samples.

2.3.9 Other Interpolation Functions

A second way to generalise the SW is to use an interpolation function other than the modified sinc function. An example of this is given in figure 2.23, which shows some rows of an interpolation matrix based on the simple linear

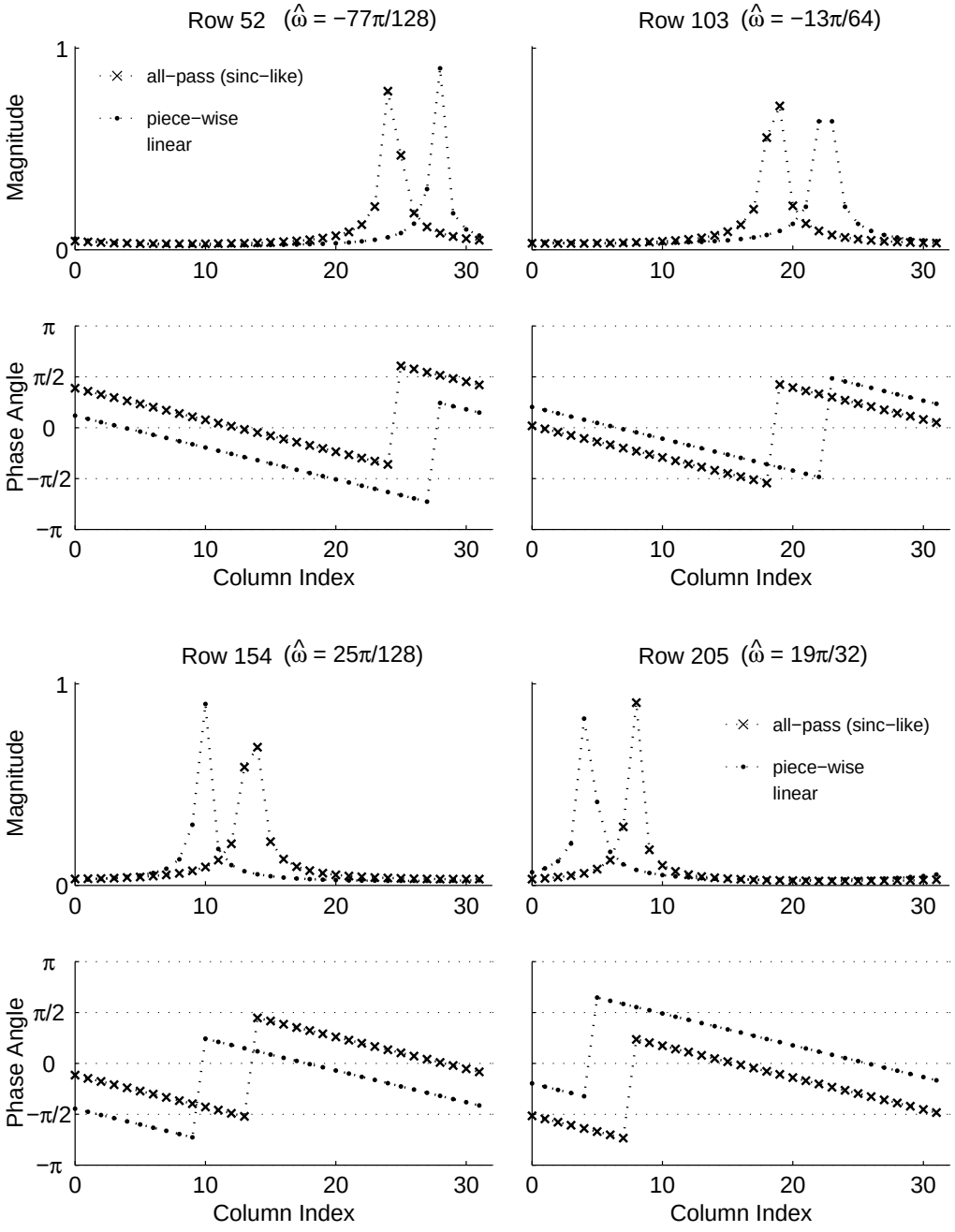

Figure 2.22: **Rows of the interpolation matrices C for all-pass and piece-wise-linear SW.**

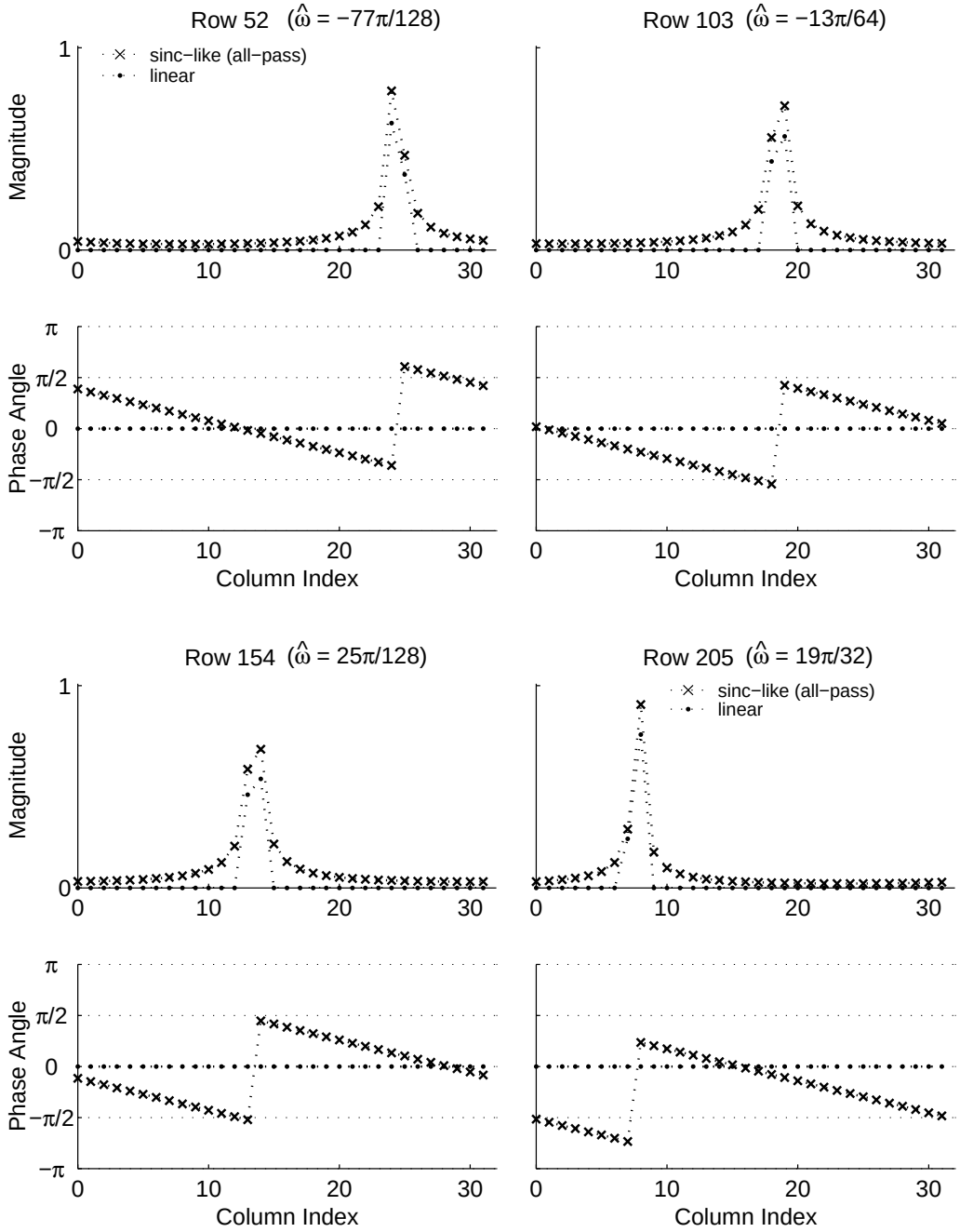

Figure 2.23: **Rows of the modified-sinc and linear interpolation matrices.**

interpolation function

$$C_{linear}(\omega) = \begin{cases} 1 - \frac{N}{2\pi}|\omega + \omega_m|, & |\omega + \omega_m| < \frac{2\pi}{N} \\ 0, & \text{otherwise} \end{cases} \tag{2.100}$$

The matrices for the figure interpolate 32 frequency-domain input samples to evaluate 265 output samples. The crosses show rows of the interpolation matrix for a first-order all-pass warping with a warp parameter of $a = 1/8$. The dots show rows of a matrix that calculates the (magnitude of the) output samples as a linear combination of the two closest input samples. This interpolation matrix is used to calculate the same output frequency locations as the first-order all-pass modified sinc interpolation matrix.

An advantage of using the above type of interpolation function is the small number of calculations required to perform the interpolation. Only two multiplications and one addition are required to calculate each output sample. $\mathbf{C}$ is almost diagonal (depending on the severity of the warping), so an efficient implementation using linear filters is possible.

The figures 2.24 and 2.25 show that the three example warpings (all-pass frequency mapping with modified sinc interpolation, piecewise-linear frequency mapping with modified sinc interpolation, and approximate all-pass frequency mapping with linear interpolation) produce signals with similar frequency content. The piecewise-linear warping increases the duration of each frequency component (and the gap between the components) equally, because the slope of the mapping is the same at the respective frequencies of the components. For the same reason, the final bandwidth of each component is identical. Conversely, time separation occurs in the all-pass and approximate all-pass warpings, also the components change in duration, amplitude and bandwidth by different amounts from each other.

The four pairs of graphs in figure 2.24 show the results in time and frequency of applying SW to a signal consisting of two components that are distinct in both time and frequency. The unwarped signal is shown in the top pair of graphs. The second pair is the signal warped using the interpolation matrix that results from a first-order all-pass mapping function for

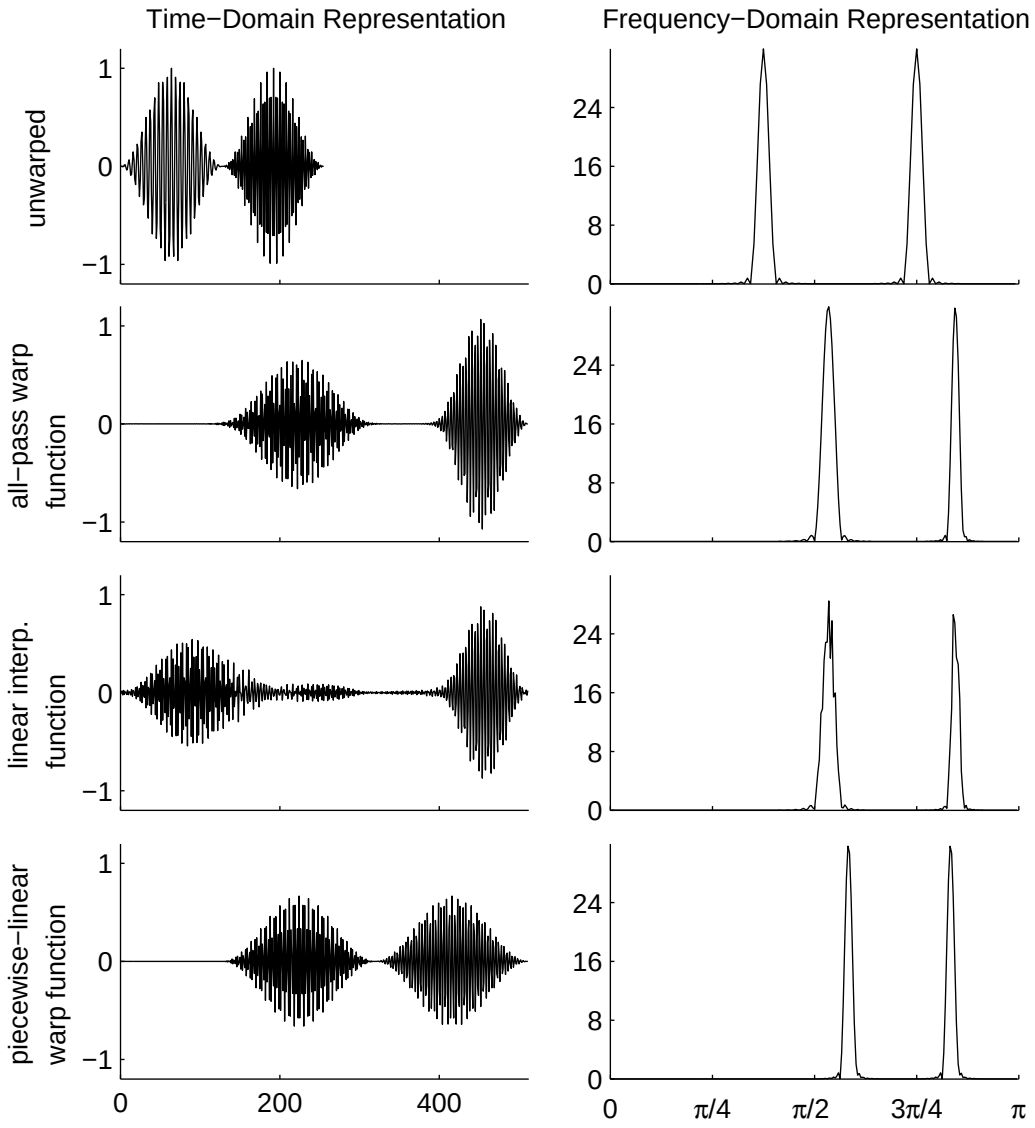

Figure 2.24: **The result of SW using different interpolation matrices.** An unwarped signal is shown in the top pair of graphs. The second pair of graphs plot the signal once it has been warped using the interpolation matrix that results from a first-order all-pass mapping function for which $a = 1/8$. The third is the warping given by the type of linear-interpolation shown in figure 2.23. The bottom graphs show the warping of an interpolation matrix that results from the piecewise-linear function of figure 2.22.

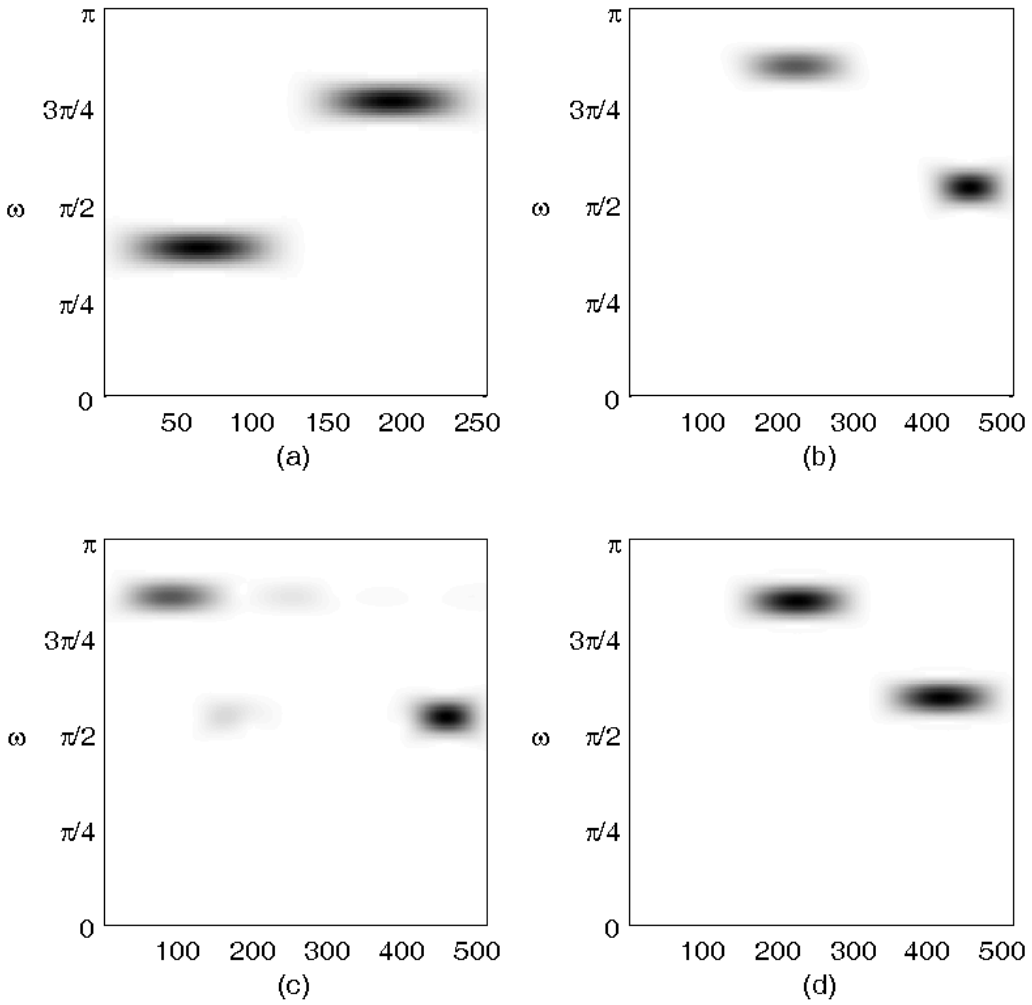

Figure 2.25: **Spectrograms of warped signals from different interpolation matrices.** (a) shows the unwarped signal, (b) shows the all-pass warped signal, (c) shows the signal warped with a linear interpolation matrix and (d) shows the signal warped using a piecewise-linear frequency mapping.

which $a = 1/8$. The third is the warping given by the type of linear-interpolation shown in figure 2.23. The bottom graphs show the warping of an interpolation matrix that results from the piecewise-linear function of figure 2.22.

The spectrograms in figure 2.25 show the same signals as figure 2.24. The four plots are: the unwarped signal, the all-pass warped signal, the signal warped with a linear interpolation matrix and shows the signal warped using a piecewise-linear frequency mapping. Because the linear interpolation functions are very short in the frequency domain (only two samples long), they have a long duration in the time domain, resulting in the *temporal leakage* seen in plot (c). The term temporal leakage, as used here, is analogous to the signal processing meaning of spectral leakage. Spectral leakage is the phenomenon of signal energy being transferred to other frequency components during a time-domain windowing operation. Likewise, temporal leakage is energy being transferred to other time samples during a frequency-domain windowing operation (that is, the DFT being multiplied by a frequency "window" function).

2.3.10 Invertibility

For the spectral warping to be invertible, there must exist an pseudo-inverse SW matrix $\mathbf{S}_{N,M}^{-1}$ (in general, $\mathbf{S}$ is not square, so a true inverse will not exist) for which

$$\mathbf{S}_{N,M}^{-1}\,\mathbf{S}_{M,N} = \mathbf{I}_N \tag{2.101}$$

where $\mathbf{I}_N$ is the $N \times N$ identity matrix. From (2.75):

$$\begin{aligned} \mathbf{S}_{N,M}^{-1}\,\mathbf{S}_{M,N} &= \mathbf{H}_{N,M}^{-1}\mathbf{D}_M\;\mathbf{D}_M^{-1}\mathbf{H}_{M,N} \\ \Rightarrow \qquad \mathbf{S}_{N,M}^{-1} &= \mathbf{H}_{N,M}^{-1}\mathbf{D}_M \end{aligned} \tag{2.102}$$

$\mathbf{H}$ belongs to the class of matrices known as Vandermonde matrices [20, 21, 22]. As such its determinant can be expressed as

$$\det\left(\mathbf{H}_{M,N}\right) = \prod_{1 \le i \le k \le N} \left(e^{-j\,\omega_k} - e^{-j\,\omega_i}\right) \tag{2.103}$$

The determinant of $\mathbf{H}$, therefore, is non-zero (indicating the existence of an inverse) if there are at least N distinct warped frequencies ω_m. The inverse frequency interpolation matrix can also be derived from $\mathbf{H}^{-1}$:

$$\mathbf{C}_{N,M}^{-1} = \mathbf{D}_N \mathbf{H}_{N,M}^{-1} \tag{2.104}$$

From (2.98) and (2.104):

$$\mathbf{C}^{-1}\mathbf{C} = \mathbf{I}_N \tag{2.105}$$

Equation (2.105) implies that $\mathbf{C}$ must be of rank N for such an inverse to exist, which suggests that any additional output frequency samples ($M > N$) are redundant. This is true from a mathematical perspective, but, as explained in section 2.4 and in [23], to completely represent the warped output signal in the time domain in a useful manner, generally more output samples than input samples are required. There is a similar effect that occurs when implementing convolution via the frequency domain [13], for which the input signal must be zero-padded prior to filtering to prevent aliasing in the time domain.

2.4 Effects Due to Sequence Length

Because the duration of some of the signals' frequency components are increased after SW (as discussed in section 2.2.2) the number of output samples M, in general, has to be greater than the number of input samples N in order for the output time-domain signal to have a useful physical meaning. The effect of producing too few output samples is that the signal at frequency components that would have otherwise extended past M samples will become aliased in the time domain. This is illustrated in figure 2.26, where the frequencies stretched most in time have wrapped around to the beginning of the sequence. The plot and spectrogram of figure 2.26 show the same unit impulse used in figure 2.15 after it has been warped using the same number of output samples as input samples. While technically this aliasing

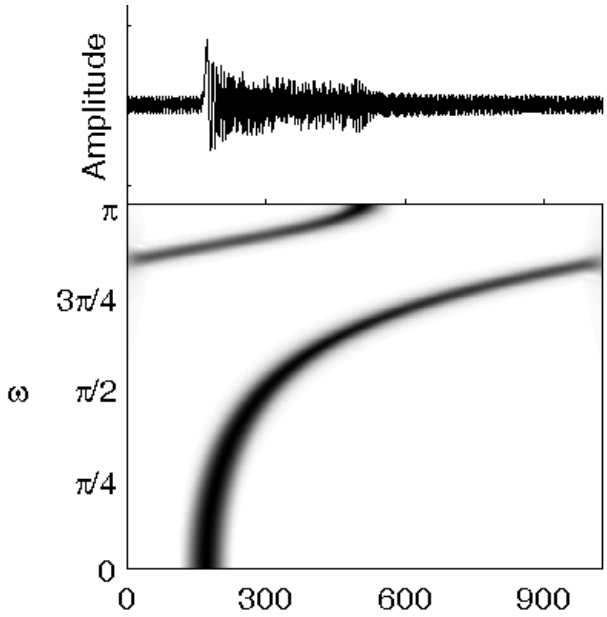

Figure 2.26: **The result of spectrally warping a unit impulse using too few output samples.** The frequencies stretched most in time wrap around to the beginning of the sequence. The plot and spectrogram above show the same unit impulse used in figure 2.15 after it has been warped using the same number of output samples as input samples.

can be recovered (since $\mathbf{H}$ is invertible — as discussed in section 2.3.10) the time-domain signal has little meaning.

When $\mathbf{C}$ is not square, the inverse is not unique because only N of the M frequency sample points are linearly independent. When an all-pass mapping is used for the spectral warping transform, the inverse transform is the same as the forward transform, except with a negative warping factor a. Since a transformation with a increases the number of samples from N to M, the inverse transformation will also have the same time stretching ratio. However, the time stretching of the inverse transform occurs at frequencies that were shrunk in time by the forward transform. Since there are N input samples, provided that M is sufficiently long to represent the entire transformed sequence without aliasing, then the inverse transform will provide only N non-zero output samples.

2.5 Spectral Warping using the Non-Uniform Discrete Fourier Transform

The non-uniform DFT provides a method to evaluate the samples at arbitrary locations on the z-plane [20]. A non-uniform DFT followed by a regular inverse DFT is a SW transform.

The non-uniform DFT of an N-length sequence $x[n]$ is defined in [24] as

$$X[z_k] = \sum_{n=0}^{N-1} x[n] z_k^{-n} \qquad k = 0, 1, \ldots, N-1 \tag{2.106}$$

where $z_0, z_1, \ldots, z_{N-1}$ are distinct points anywhere in the z–plane. By defining a non-uniform DFT matrix $\mathbf{U}$ as

$$\mathbf{U} = \begin{pmatrix} 1 & z_0^{-1} & z_0^{-2} & \cdots & z_0^{-(N-1)} \\ 1 & z_1^{-1} & z_1^{-2} & \cdots & z_1^{-(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{N-1}^{-1} & z_{N-1}^{-2} & \cdots & z_{N-1}^{-(N-1)} \end{pmatrix} \tag{2.107}$$

the non-uniform DFT can be expressed in the form

$$\mathbf{X} = \mathbf{U}\mathbf{x}$$

$$\begin{pmatrix} X(z_0) \\ X(z_1) \\ \vdots \\ X(z_{N-1}) \end{pmatrix} = \begin{pmatrix} 1 & z_0^{-1} & z_0^{-2} & \dots & z_0^{-(N-1)} \\ 1 & z_1^{-1} & z_1^{-2} & \dots & z_1^{-(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_{N-1}^{-1} & z_{N-1}^{-2} & \dots & z_{N-1}^{-(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{pmatrix} \tag{2.108}$$

The non-uniform DFT is invertible ($\mathbf{U}$ is non-singular) if the sampling points $z_0, z_1, \dots, z_{N-1}$ are distinct [24]. The inverse transform is given by

$$\mathbf{x} = \mathbf{U}^{-1}\mathbf{X} \tag{2.109}$$

The non-uniform DFT is a generalisation of the DFT and reduces to the DFT when the N points are evenly spaced around the unit circle. In this case, $\mathbf{U}$ equals the DFT matrix $\mathbf{D}$ given in (2.70).

A special case of the non-uniform DFT is the *Warped Discrete Fourier Transform* [25]. The z–plane sample locations are found by warping evenly spaced sample location according to

$$z^{-1} = B(\hat{z}) \tag{2.110}$$

where $B(\hat{z})$ is an M^{th}-order all-pass function. It is all-pass because it maps points on the unit circle (in the z–plane) to other points on the unit circle (in the $\hat{z}$-plane).

Warped DFT is achieved by warping the z–plane (i.e., applying the mapping of (2.110) to the z–transform of an N-length sequence):

$$X(z) = \sum_{n=0}^{N} x[n]z^{-n} \quad \rightarrow \quad X(\hat{z}) = \sum_{n=0}^{N} x[n]z^{-n} \tag{2.111}$$

and evaluating at evenly spaced locations around the unit circle:

$$\hat{X}[k] = X(\hat{z})\Big|_{\hat{z}=e^{j2\pi k/N}} \tag{2.112}$$

$X(\hat{z})$ can be written as a ratio of two polynomials,

$$X(\hat{z}) = \frac{P_e(\hat{z})}{D_e(\hat{z})} \tag{2.113}$$

where $P_e(\hat{z})$ and $D_e(\hat{z})$ are some polynomials of $\hat{z}$ that solve the equation. By defining

$$\begin{aligned} P(\hat{z}) &= P_e(\hat{z}) \mod \hat{z}^N \\ &= \sum_{n=0}^{N-1} \left[\sum_{m\geq 0, n+mN\leq M(N-1)} p_{e_{nmN}} \right] \hat{z}^{-n} \end{aligned} \tag{2.114}$$

where p_{e_i} is the i^{th} coefficient of $P_e(\hat{z})$, and

$$\begin{aligned} D(\hat{z}) &= D_e(\hat{z}) \mod \hat{z}^N \\ &= \sum_{n=0}^{N-1} \left[\sum_{m\geq 0, n+mN\leq M(N-1)} d_{e_{nmN}} \right] \hat{z}^{-n} \end{aligned} \tag{2.115}$$

$\hat{X}[k]$ can be decomposed to

$$\hat{X}[k] = \frac{P(\hat{z})\Big|_{\hat{z}=e^{j2\pi k/N}}}{D(\hat{z})\Big|_{\hat{z}=e^{j2\pi k/N}}} \tag{2.116}$$

A warped DFT followed by an inverse DFT is equivalent to the all-pass SW transform described in previous sections.

Chapter 3

Realising the Spectral Warping Transform

The aim of this chapter is to evaluate algorithms that perform SW and that are suitable for implementing in either hardware or software. These realisations fall into two broad categories: those that calculate each output sample as a linear combination of the input samples — that is, performing SW using FIR filter banks — and, secondly, implementations based on all-pass IIR filters. A generalised heuristic for designing a SW transform realisation is also provided, with an example.

3.1 Direct Multiplication Realisation

Spectral Warping can be realised directly by performing the multiplications and additions implied in the equation $\mathbf{g} = \mathbf{Sf}$ (see (2.74) on page 47), as expanded here:

$$\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{M-1} \end{pmatrix} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,N-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{M-1,0} & s_{M-1,1} & \cdots & s_{M-1,N-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{M-1} \end{pmatrix}$$

$$= \begin{pmatrix} f_0 s_{0,0} & + & f_1 s_{0,1} & + & \cdots & + & f_{N-1} s_{0,N-1} \\ f_0 s_{1,0} & + & f_1 s_{1,1} & + & \cdots & + & f_{N-1} s_{1,N-1} \\ & & & \vdots & & & \\ f_0 s_{M-1,0} & + & f_1 s_{M-1,1} & + & \cdots & + & f_{N-1} s_{M-1,N-1} \end{pmatrix}$$

$$\Rightarrow \qquad \mathbf{g} = f_0 \mathbf{s}_{M,0} + f_1 \mathbf{s}_{M,1} + \cdots + f_{N-1} \mathbf{s}_{M,N-1} \tag{3.1}$$

where

$$\mathbf{s}_{M,n} = \begin{pmatrix} s_{0,n} & s_{1,n} & \cdots & s_{M-1,n} \end{pmatrix}^{\mathrm{T}} \tag{3.2}$$

A schematic diagram of a parallel realisation of these calculations is presented in figure 3.1.

As a hardware implementation, this direct multiplication realisation would be fast (it is highly parallel, with the each path requiring one multiplication and N additions), but expensive in terms of hardware real estate.

3.2 Spectral Warping Using Filters

The SW transform can be realised by using a filter bank. This can be done in two ways. The first way is to treat each input sample as an impulse and pass each impulse through a filter, summing the resulting impulse responses. The second way is to filter the entire input sequence many times, using different filters. One sample from each of the filter outputs is selected and these samples are concatenated to form the output sequence. These methods correspond to filtering using the rows of the **S** matrix, and the columns of the **S** matrix, respectively. The methods are described in detail below.

3.2.1 Filtering Using Columns of S

It can easily be seen from (3.1) and (3.2) that the output sequence is the sum of the time-dependent impulse responses (columns of **S**) scaled by the corresponding input sample. The realisation in figure 3.2 uses these impulse responses as filters — a single input sample is fed through each filter. Each

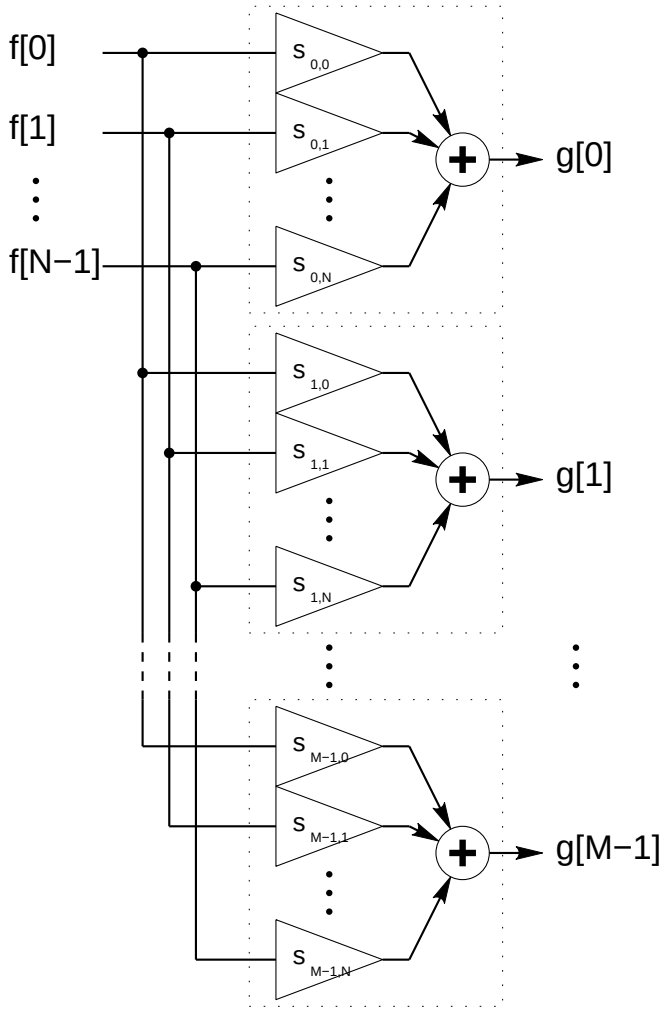

Figure 3.1: **SW as matrix multiplication.** A SW transform, represented by the matrix **S**, can be constructed using combinatorial logic.

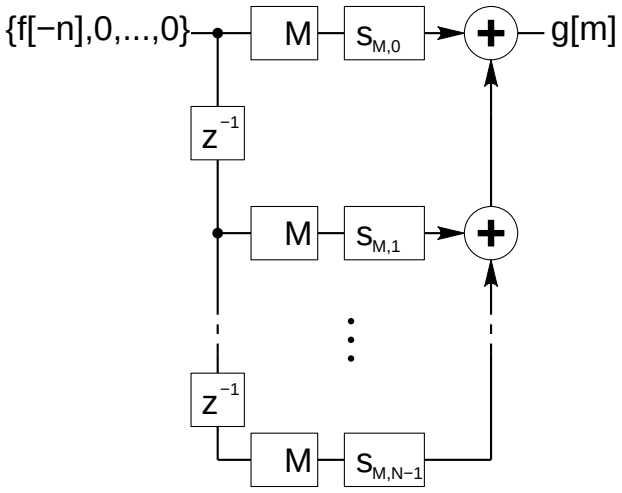

Figure 3.2: **SW as a bank of N length-M filters.**

column of $\mathbf{S}$ (of length M) is scaled by its corresponding input sample. The output sequence is the sum of these scaled impulse responses. Figure 3.2 is a serial equivalent of figure 3.1, presented as a signal block diagram.

Because SW operates on a frame of N input samples, producing a frame of M output samples (M is usually larger than N), each input frame must first be augmented with $M - N$ zeroes. Without this zero padding, the impulse responses from one frame will not have died away before the next frame arrives, causing the output frames to overlap. After the zero padding, the input sequence frame needs to be time reversed for this SW structure to be realisable (time advances become time delays). A further $M - 1$ zeroes must be appended to the (time reversed) signal (this is not shown in figure 3.2). These extra zero samples are necessary because the first real output sample occurs at sample position M of the output sequence. The extra zeroes ensure the remaining output samples are propagated through the filters. The leading $M - 1$ samples of the output sequence are zero and should be discarded. The extra $M - 1$ zeroes only need to be appended at the end of all the input frames, not to every frame.

A MATLAB function that performs spectral warping by filtering by columns

of **S** is shown in figure 3.3. This is not a very efficient MATLAB implementation. However, it serves as a proof of concept for the method, and can be used as a base from which to design hardware or firmware implementations.

3.2.2 Filtering Using Rows of S

The structure of figure 3.2 uses N filters, each of length M. Alternatively, an equivalent structure that uses M length-N filters can be designed — the entire input sequence with rows of **S** is filtered, instead of filtering (multiplying) single input samples with columns of **S**. This new scheme selects a single output sample (the N^{th} sample out of every M-sample frame) from each filter, as opposed to selecting a single input sample to feed into each filter. This design is illustrated in figure 3.4, in which the input sequence is passed through M filters, each relating to a row of the **S** matrix. The input sequence is time-reversed here, although the same result is achieved by time-reversing the filters instead. Note that the down sample blocks each select an output sample on a different clock cycle (they are out of phase from each other). Mathematically, this process relates to

$$g[m] = g_m = \mathbf{f} \cdot \mathbf{s}_{m,\mathrm{N}} \tag{3.3}$$

where

$$\mathbf{s}_{m,N} = \begin{pmatrix} s_{m,0} & s_{m,1} & \cdots & s_{m,N-1} \end{pmatrix} \tag{3.4}$$

A proof-of-concept MATLAB function to perform spectral warping by filtering by rows of **S** is listed in figure 3.5.

3.3 Designing a SW Realisation

To design a SW realisation, the steps listed here may be followed:

1. Choose the M frequencies ω_m that give the locations $e^{-j\,\omega_m}$ at which the z-plane is evaluated. These points, which form the $\mathbf{H}_{M,N}$ matrix,

```
function g = warp_by_columns_of_S( S, f_in )
[M,N] = size( S );
len = M/N*length( f_in );
% Insert M - N zeroes after each input frame to produce
% the right number of output samples and prevent the
% output frames overlapping
f = zeros( len, 1 );
f(mod( [0:len-1], M ) < N) = f_in;
% Time reverse the input, frame by frame
i = [0:len - 1];
f = f(M - mod( i, M ) + M*floor( i/M ));
% Append zeroes to allow time for the signal to propagate
% through the filters
f = [f; zeros( M-1, 1 )];

g = zeros( len + M-1, 1 );
i = ~mod( [1:len + M-1]', M );
for n = 1:N
    % Downsample by M
    x = f .* i;
    % Filter by the nth column of S
    x = filter( S(:,n), 1, x );
    % Add this to the output from the other filters
    g = g + x;
    % Time delay input
    f = [0; f(1:end-1)];
end
g = g(M:end);  % The first M - 1 samples are always zero
```

Figure 3.3: **Matlab code for spectral warping using columns of S as filters.**

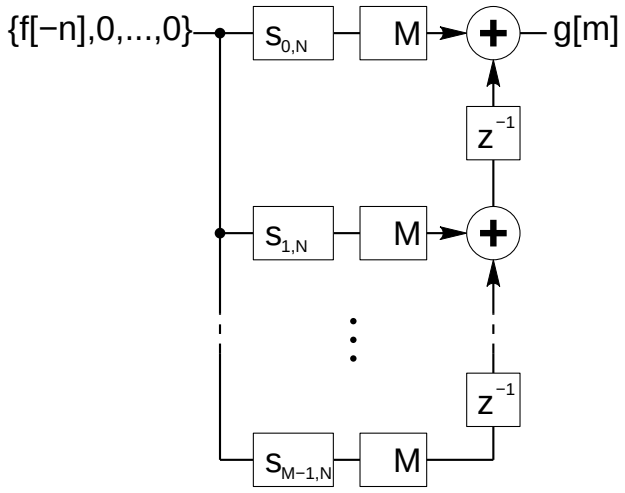

Figure 3.4: **SW as a bank of M length-N filters.**

define the warping. They may or may not have an underlying function $\theta(\omega)$, but careful selection of $\theta(\omega)$ can offer efficient filter designs.

2. Calculate the $\mathbf{S}_{M,N}$ matrix by either

   (a) solving $\mathbf{S}_{M,N} = \mathbf{H}_{M,N}\mathbf{D}_N$ for sinc-like (exact) interpolation, or

   (b) using a set of some other (approximate) interpolation functions, such as the linear interpolation function (2.100) on page 65, to generate the frequency interpolation matrix $\mathbf{C}_{M,N}$, then solve $\mathbf{S}_{M,N} = \mathbf{D}_M^{-1}\mathbf{C}_{M,N}\mathbf{D}_N$.

3. Design the filters, either from the columns of $\mathbf{S}$, for a realisation of the form shown in figure 3.2, or from the rows of $\mathbf{S}$, for the form shown in figure 3.4. Alternatively implement the direct multiplication method of figure 3.1.

The filters used in a SW realisation may either be FIR or IIR filters. IIR filter designs have a disadvantage in that, for exact spectral warping, they have to be reset in between each input frame, otherwise the output of later frames will be affected by the lingering transients of earlier frames.

```
function g = warp_by_rows_of_S( S, f_in )
[M,N] = size( S );
len = M/N*length( f_in );
% Insert M - N zeroes after each input frame to produce
% the right number of output samples and prevent the
% output frames overlapping
f = zeros( len, 1 );
f(mod( [0:len-1], M ) < N) = f_in;
% Time reverse the input, frame by frame
i = [0:len - 1];
f = f(M - mod( i, M ) + M*floor( i/M ));
% Append zeroes to allow time for the signal to propagate
% through the filters
f = [f; zeros( M-1, 1 )];

g = zeros( len + M-1, 1 );
i = ~mod( [1:len + M-1]', M );
for m = M:-1:1
    % Filter by the mth row of S
    x = filter( S(m,:), 1, f );
    % Downsample by M
    x = x .* i;
    % Time delay output
    g = [0; g(1:end-1)];
    % Add this to the output from the other filters
    g = g + x;
end
g = g(M:end);  % The first M - 1 samples are always zero
```

Figure 3.5: **Matlab code for spectral warping using rows of S as filters.**

3.3.1 An Example SW Realisation Design

To illustrate the above procedure, a SW realisation based on the all-pass frequency mapping function

$$\theta(\omega) = \tan^{-1} \frac{(1-a^2)\sin\omega}{(1+a^2)\cos\omega - 2a} \qquad -1 > a > 1 \tag{3.5}$$

is designed here. For simplicity only warping input frames of length 3 are considered. This can be extended at a later stage. The maximum slope of (3.5) is 3 (when $|a|$ approaches unity), so the output frame is set to length to $M = 3 \times 3 = 9$ samples.

$$\begin{aligned}\hat{\boldsymbol{\omega}} &= \left(\theta^{-1}(0), \theta^{-1}\left(\tfrac{2\pi}{9}\right), \theta^{-1}\left(\tfrac{4\pi}{9}\right), \theta^{-1}\left(\tfrac{2\pi}{3}\right), \theta^{-1}\left(\tfrac{8\pi}{9}\right),\right. \\ &\qquad \left.\theta^{-1}\left(\tfrac{-8\pi}{9}\right), \theta^{-1}\left(\tfrac{-2\pi}{3}\right), \theta^{-1}\left(\tfrac{-4\pi}{9}\right), \theta^{-1}\left(\tfrac{-2\pi}{9}\right)\right)^{\mathrm{T}} \\ &= \left(0, \tan^{-1}\frac{(1-a^2)\sin\frac{2\pi}{9}}{(1+a^2)\cos\frac{2\pi}{9} + 2a}, \ldots, \tan^{-1}\frac{(1-a^2)\sin\frac{-2\pi}{9}}{(1+a^2)\cos\frac{-2\pi}{9} + 2a}\right)^{\mathrm{T}}\end{aligned} \tag{3.6}$$

Setting $a = 1/2$ gives the following output frequency vector and corresponding z-transform matrix.

$$\hat{\boldsymbol{\omega}} = \begin{pmatrix} 0 \\ 0.2415 \\ 0.5455 \\ 1.0472 \\ 2.1685 \\ -2.1685 \\ -1.0472 \\ -0.5455 \\ -0.2415 \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & e^{-j\,0.2415} & e^{-j\,0.4829} \\ 1 & e^{-j\,0.5455} & e^{-j\,1.0909} \\ 1 & e^{-j\,1.0472} & e^{-j\,2.0944} \\ 1 & e^{-j\,2.1685} & e^{j\,1.9463} \\ 1 & e^{j\,2.1685} & e^{-j\,1.9463} \\ 1 & e^{j\,1.0472} & e^{j\,2.0944} \\ 1 & e^{j\,0.5455} & e^{j\,1.0909} \\ 1 & e^{j\,0.2415} & e^{j\,0.4829} \end{pmatrix} \tag{3.7}$$

The equation $\mathbf{S} = \mathbf{D}_M^{-1}\mathbf{H}$ is then solved to give

$$\mathbf{S} = \begin{pmatrix} 1 & \frac{86}{171} & \frac{708}{171\times19} \\ 0 & \frac{1}{171} & \frac{-180}{171\times19} \\ 0 & \frac{2}{171} & \frac{-303}{171\times19} \\ 0 & \frac{4}{171} & \frac{-492}{171\times19} \\ 0 & \frac{8}{171} & \frac{-756}{171\times19} \\ 0 & \frac{16}{171} & \frac{-1056}{171\times19} \\ 0 & \frac{32}{171} & \frac{-1200}{171\times19} \\ 0 & \frac{64}{171} & \frac{-576}{171\times19} \\ 0 & \frac{128}{171} & \frac{2496}{171\times19} \end{pmatrix} \tag{3.8}$$

This SW matrix translates into the block diagram shown in figure 3.6 (using the parameters: $N = 3$, $M = 9$ and $a = 1/2$). The down sample blocks in the figure are out of phase from each other (they each select an output sample on a different clock cycle).

This parallel filter bank can be converted into a cascade of filters by dividing each filter by the product of all the previous filters. Figure 3.7 illustrates this procedure. This division produces systems that are ratios of two polynomials. Such systems are easily realised as IIR filters. For example, the second filter of this cascade is

$$\frac{s_{1,N}}{s_{0,N}} = \frac{\frac{1}{171}z^{-1} - \frac{180}{171\times19}z^{-2}}{1 + \frac{86}{171}z^{-1} + \frac{708}{171\times19}z^{-2}} \tag{3.9}$$

where $s_{m,N}$ is the filter represented by the m^{th} row of the $\mathbf{S}$ matrix. This system is illustrated in figure 3.8. The third filter of the cascade is given by

$$\begin{aligned} \frac{s_{2,N}}{s_{1,N}/s_{0,N}} &= \frac{\left(\frac{2}{171}z^{-1} - \frac{303}{171\times19}z^{-2}\right)\left(1 + \frac{86}{171}z^{-1} + \frac{708}{171\times19}z^{-2}\right)}{\frac{1}{171}z^{-1} - \frac{180}{171\times19}z^{-2}} \\ &= \frac{2 - \frac{48545}{171\times19}z^{-1} - \frac{24642}{171\times19}z^{-2} - \frac{214524}{171\times(19)^2}z^{-3}}{1 - \frac{180}{19}z^{-1}} \end{aligned} \tag{3.10}$$

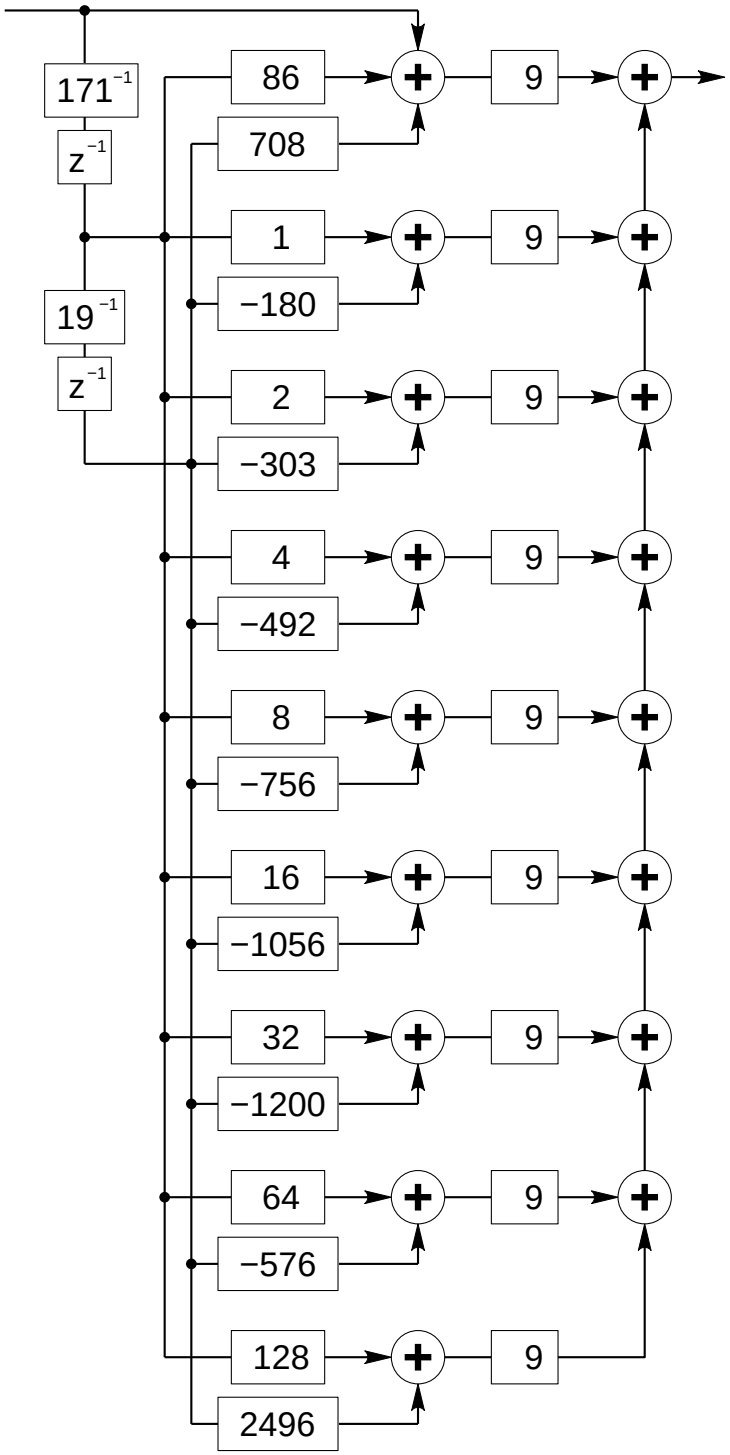

Figure 3.6: **Block diagram for an FIR implementation of an all-pass SW transform.**

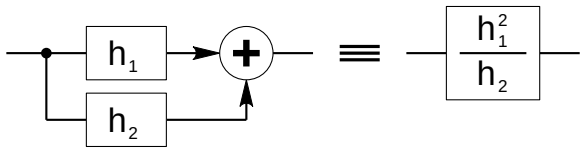

Figure 3.7: **Converting a parallel filter bank into a cascade of filters.**

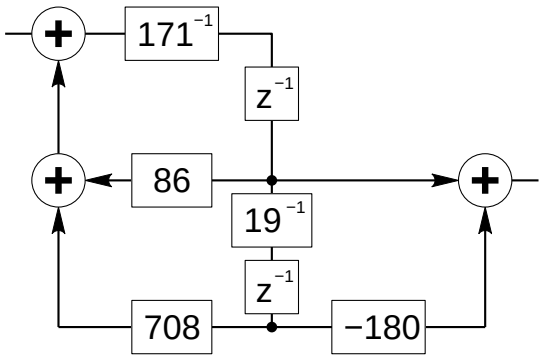

Figure 3.8: **Second filter of the cascade implementation of an all-pass SW transform.**

The fourth filter is given by

$$\begin{aligned}\frac{s_{3,N}s_{1,N}}{s_{2,N}s_{0,N}} &= \frac{\left(\frac{4}{171}z^{-1}-\frac{492}{171\times19}z^{-2}\right)\left(\frac{1}{171}z^{-1}-\frac{180}{171\times19}z^{-2}\right)}{\left(\frac{2}{171}z^{-1}-\frac{303}{171\times19}z^{-2}\right)\left(1+\frac{86}{171}z^{-1}+\frac{708}{171\times19}z^{-2}\right)} \\ &= \frac{\frac{4}{171}z^{-1}-\frac{1212}{171\times19}z^{-2}+\frac{88560}{171\times(19)^2}z^{-3}}{2-\frac{48545}{171\times19}z^{-1}-\frac{24642}{171\times19}z^{-2}-\frac{214524}{171\times(19)^2}z^{-3}}\end{aligned} \tag{3.11}$$

3.4 IIR Method Implementation

An all-pass SW transform design using a cascade of all-pass IIR filters is described in this section. In this scheme all the filters from the third one on are identical, which enables implementations that are efficient in terms of hardware.

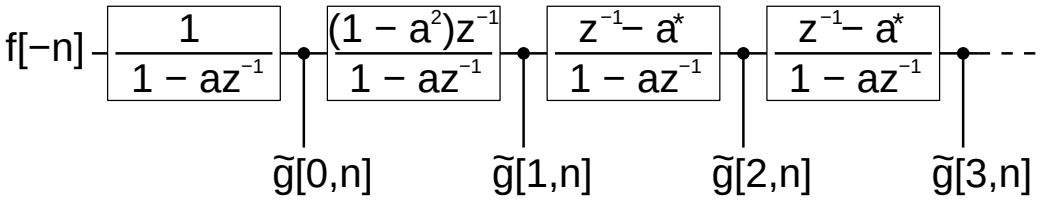

Figure 3.9: **SW using a cascade of all-pass filters.**

3.4.1 All-Pass Filter Realisation

A SW network that is an all-pass filter of the form shown in (2.27) can be realised as a cascade of first-order all-pass filters. Figure 3.9 shows such a network. A new notation is introduced in this figure: $\tilde{g}_m[n]$ denotes the (time reversed) length-N signal after it has passed through $m+1$ all-pass filters. In this SW network, a signal is first time reversed, then passed through the first all-pass filter. The last sample of the filtered sequence becomes the first output sample (that is $g[0] = \tilde{g}_0[N-1]$). The sequence is then passed through another all-pass filter. The last sample of the sequence after the second filter is the second output sample. This process is repeated for each filter in the network, giving the expression

$$g[m] = \tilde{g}_m[N-1] \qquad m = 0, 1, ..., M-1 \tag{3.12}$$

The number of output samples generated is equal to the number of filters used. All of the filters are identical apart from the first two [1].

3.4.2 The Oppenhiem-Johnson Implementation

The IIR all-pass filters shown in figure 3.9 can be realised with a combination of combinatorial logic and an M-word shift register, as shown schematically in figure 3.10. The algorithm that is applied to the hardware blocks of the figure is as follows:

1. The time-reversed input samples are fed into the network via block D.

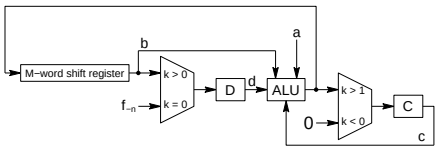

Figure 3.10: **The SW network hardware implementation.**

2. For every (time-reversed) sample that is entered, data passes though the ALU M times, where M is the length of the output sequence.

3. After N samples have been processed, the shift register contains the warped output sequence.

In total, $O(N, M)$ ALU calculations are required for the transform. Oppenheim and Johnson first suggested this implementation in [1].

3.5 ASIC Implementation

At the heart of the Oppenhiem-Johnson implementation is the ALU, which is dedicated to executing the function

$$F = a(b - c) + d \tag{3.13}$$

An application-specific integrated circuit (ASIC) that provides a specialised arithmetic logic unit (ALU) core to calculate (3.13) was designed and fabricated using very-large-scale integration technology (VLSI).

3.5.1 Motivation

Among new areas of possible applications of the SW transform is the electronic testing of mixed-signal circuits and systems. In this context, the use of the SW transform can open up new opportunities in test signal generation and response analysis for both external testing and built-in self testing (BIST). Application of the SW transform for external testing would allow the tester performance to be increased. The use of SW in mixed-signal BIST

is related to the current trend of using digital test tools and methods for both digital and analogue components (so-called *DSP-based testing*), wherein the test signal generation as well as the test response analysis is realised by digital means. The speed of DSP-based testing depends heavily on the speed of the processor. Such dependence is a serious weak point of the method (due to the limitations of the computing resources inherent in built-in test resources).

The main objective of the ALU core is to investigate the use of SW as implemented in CMOS, in particular its use in the characterisation of mixed-signal ICs. It is a proof-of-concept realisation, from which further designs can be derived. It is envisaged that SW techniques will be used as tools in mixed-signal IC characterisation (in fact, characterisation of any unknown arbitrary system) and high-resolution frequency analysis, plus other yet-to-be-discovered applications. It is proposed that the use of the SW transform can significantly increase the speed of DSP-based testing, making it more efficient.

Secondary objectives of the ALU core include evaluating the usefulness of the SW transform implemented with five-bit precision, and experimenting with real-time applications of SW. Five-bit precision was chosen because that was the largest word-length that would allow the chip to be developed in reasonable time with the design tools available.

3.5.2 Methods

The circuit was designed using DSCH2 for the gate-level design and Microwind2 as the layout editor.[1] It was fabricated using 0.18 μm ST CMOS technology, as one section of a multi-purpose chip.

Details of the chip design are given in appendix A.

[1]Both DSCH2 and Microwind2 are written by Etienne Sicard of INSA Toulouse, France.

3.5.3 Results

Simulation tests were run using Microwind2, which showed the chip design to produce correct output when clocked at 25 MHz. Section A.8 gives details of these tests.

A test adapter board for the ProTest chip tester (see section A.9) was developed to test the SW ALU ASIC. On initial tests, it appeared that the chip did not work as intended, but further testing is required to verify if this is the case, and, if so, which part of the chip is faulty.

3.6 Matlab Simulations

The applicability and efficiency of the SW technique in mixed-signal DSP-based testing was verified experimentally. To evaluate the ability and accuracy of the SW technique to find a circuits response, two simple filters were used as devices-under-test (DUTs).

3.6.1 Experiment Set-Up

The first simulation arrangement was done using MATLAB. A software implementation of the SW network used MATLAB's filter function to directly implement the spectral warping network (figure 3.9). Figure 3.11 presents the MATLAB code for this implementation. A test chirp was generated using the SW network and fed into the DUT (a band-pass filter). The chirp response was analysed according to the method described in section 4.3.

In the second experimental set-up, a MATLAB-generated SW chirp signal was used as the input to a filter designed employing PSPICE [26]. The simulation was performed using PSPICE and the simulation output was loaded back into MATLAB for processing. An AC sweep analysis was performed using PSPICE, providing data for comparison. The DUT was an active high-frequency boost filter with a maximum gain of two, using an LM324 Op-Amp as the active component. The netlist for the circuit is found in

```
function g = swn( f, a, M )
% f: input sequence
% a: warping factor
% M: number of filters
%         (= length of output sequence)

N = length( f );
x = zeros( 1, N );
g = zeros( 1, M );

% time reverse
    f = fliplr( f );
% first filter
    x = filter( [1   0], [1   -a], f );
% tap -> 1st output sample
    g(1) = x(end);
% second filter
    x = filter( [0   1-a^2], [1   -a], x );
% tap -> 2nd output sample
    g(2) = x(end);
% filter and tap for remaining filters
    for k = 3:M
        x = filter( [-a   1], [1   -a], x );
        g(k) = x(end);
    end
```

Figure 3.11: **Matlab implementation of an all-pass IIR SW network.**

figure 3.12.

3.6.2 Results

For the first experiment, the SW technique produced a result identical to the known response of the MATLAB filter DUT (except for the very first and the very last samples). The response is shown in figure 3.13, where there was no visible difference between the actual and the calculated frequency response.

When comparing the results from the PSPICE simulation, the magnitude of the Fourier transform of the SW chirp response is very close to the magnitude part of the PSPICE AC sweep analysis, deviating only at the extremely high and low frequencies. This result is illustrated in figure 3.14, in which the solid line is from the PSPICE AC sweep analysis, the dotted line is the Fourier transform of the SW chirp response before phase adjusting and the broken line is the Fourier transform of the SW chirp response after phase adjusting. After correcting for the phase (by warping the result using an inverse warping to that which was used to generate the chirp signal), the phase response agrees with that of the AC sweep within a few degrees for $0.001\pi < \omega < 0.2\pi$ and within eight degrees up to 0.93π. The magnitude becomes less accurate (in this case gaining about 1dB over most of the spectrum) after the phase correction.

It can be concluded from these results that the Fourier transform of a system's SW chirp response is a useful approximation of the true system response. In particular, if only the magnitude of the system response is of interest and no phase correction is required, then the Fourier transform the SW chirp response is essentially equal to the system response over most of the spectrum, suggesting that the SW chirp method has much practical value.

```
** high frequency emphasis filter

* power supplies
    V1 5 0 dc +12V
    V2 6 0 dc -12V
* signal input
    Vin 1 0 AC 1 0
* filter circuit
    R1 0 2 1.2 k
    C1 2 3 33 n
    R2 3 4 1.2 k
    Rload 4 0 10 Meg
    XU 1 1 3 5 6 4 LM324


* LM324 op-amp subcircuit
* connections: non-inverting input
*              | inverting input
*              | | positive power supply
*              | | | negative power supply
*              | | | | output
*              | | | | |
.subckt LM324  1 2 3 4 5
                ... <op-amp sub-circuit>
.ends
```

Figure 3.12: **High frequency emphasis filter PSPICE netlist.**

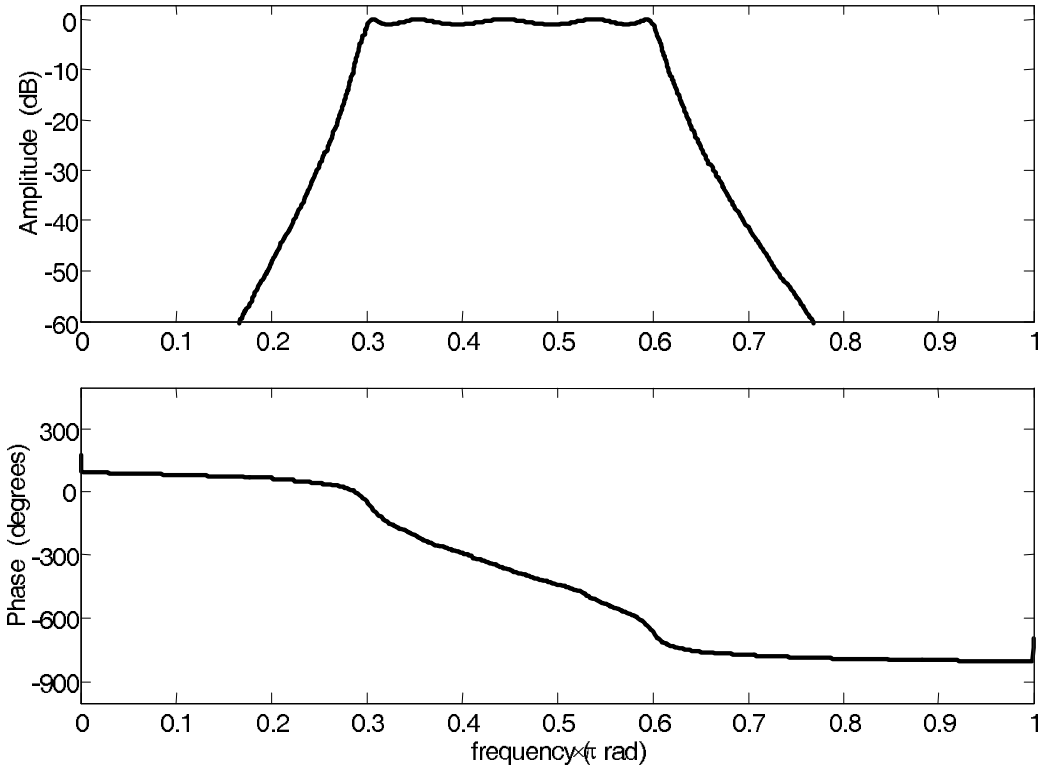

Figure 3.13: **Band-pass filter used to test SW chirp system technique.**

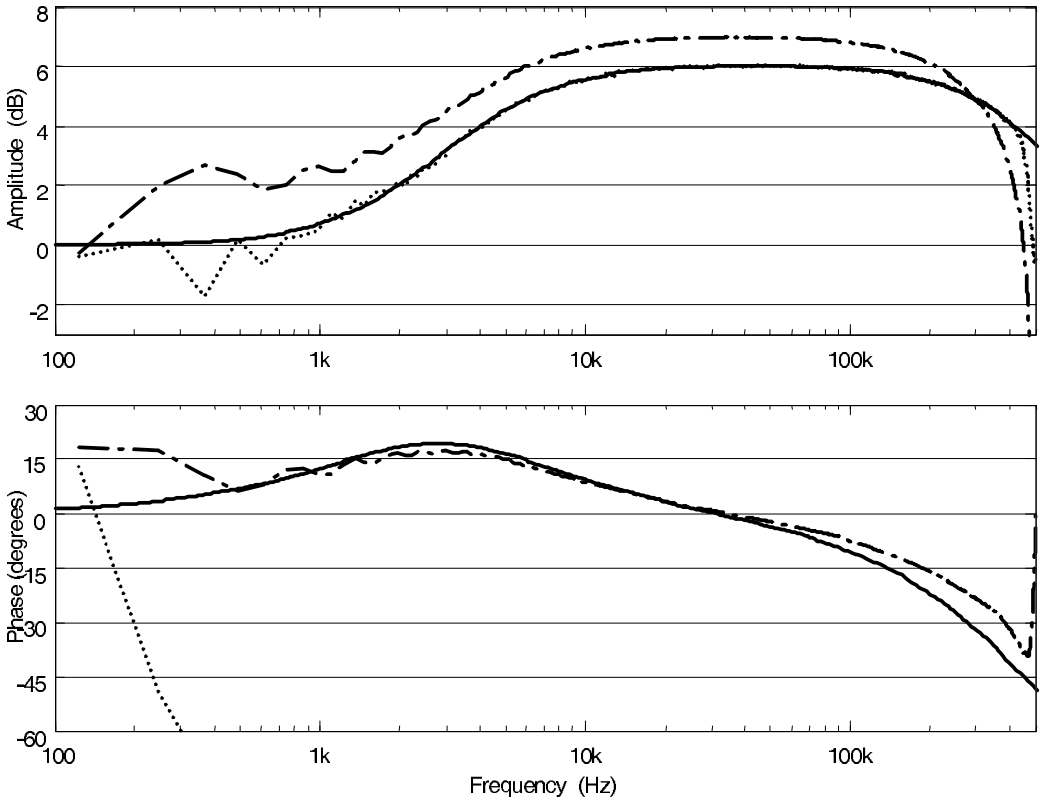

Figure 3.14: **Frequency response of PSPICE simulated filter.** Legend: the solid line is the PSPICE AC sweep analysis trace, the dotted line is the Fourier transform of the SW chirp response before phase adjusting, the broken line is the Fourier transform of the SW chirp response after phase adjusting.

Chapter 4

Example Applications of the Spectral Warping Transform

Some proposed new applications of the SW transform are in the area of Digital Signal Processing (DSP)-based analogue and mixed-signal testing. They include evaluation of the impulse response of the Device Under Test (DUT) without the risk of overloading and damaging the system, as might happen when applying an impulse. A special chirp signal is generated using SW and applied to the system. The output is fed through two SW networks to produce the impulse response. If only the magnitude response is required, then no SW is needed — simply, the Fourier transform of the chirp response is analysed. Further, a quick estimate of the magnitude response can be found by observing the envelope of the output. The proposed method is accurate for all but the very extreme frequencies.

Among the other applications of the SW transform are the characterisation of analogue-to-digital converters using input chirp signals, the generation of test signals having pre-specified frequency spectra, digital filtering with good stop-band attenuation, and unequal-resolution frequency analysis. The application of the SW transform to these areas is investigated in the following sections.

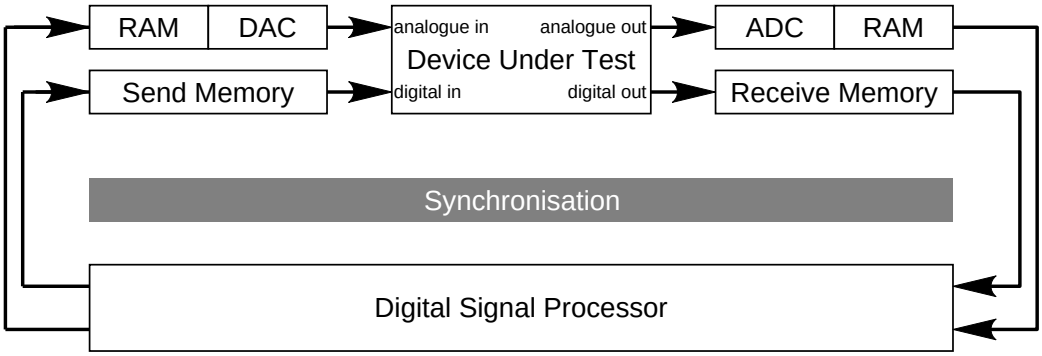

Figure 4.1: **DSP-based testing of an analogue/digital mixed-signal device.**

4.1 Spectral Warping Applications in Analogue and Mixed-Signal Testing

The importance of analogue and mixed-signal testing has grown increasingly during the last decades. The topic has been studied intensively, resulting in a number of techniques and approaches [27, 28, 29, 30, 31, 32, 33, 34]. One of these approaches is Digital Signal Processing (DSP)-based testing, which has proved to be one of the most promising, offering excellent potential for practical use. It involves using digital tools and methods to test both the digital and the analogue components [27, 32, 34]. The basic structure of DSP-based testing is presented in figure 4.1, where it can be seen that both signal generation and output measurement are realised by means of pure digital circuitry [34].

4.2 Non-Uniform Frequency Resolution Signal Generation and Analysis

Often in analogue and mixed signal devices, the frequency domain characteristics or transfer function of a device under test are of interest. Usually

the focus is on a particular region of the frequency spectrum rather than the complete spectrum, so improved resolution is desired in the region of interest. Conventional analysis using an FFT provides equal resolution from DC up to the Nyquist frequency. However, many of the samples produced by the FFT are not required, and more detail is often desired around the frequencies of interest. Improving the resolution requires taking a longer FFT so that the samples are more closely spaced in the frequency domain.

The frequency-shifting property the SW transform allows it to be used as a tool for high-resolution spectral analysis by providing higher resolution in some frequency bands and lower resolution in others. This offers significant benefits in mass-production testing where high throughput of testing is vitally important. The use of frequency spectral warping pre-processing to obtain unequal spectral resolution could reduce total computing times and thus time to perform testing. This could be especially important in testing high-speed high-frequency circuits and systems. Indeed, SW enables the Fast Fourier Transform (FFT) to have unequal resolution along the frequency axis. This relates to taking samples of the z-transform that are unevenly distributed around the unit circle, as discussed earlier (see figure 2.1 on page 22).

To perform such analysis, a signal is pre-warped to move frequency components away from a band of particular interest. An FFT of the warped signal is taken and the result can be shifted back in frequency by applying the inverse of (2.16). The data now reflect the frequency content of the unwarped signal, with the frequency data being closely spaced where the slope of the frequency-mapping curve is the steepest. This technique is most useful in the case when information from one end of the spectrum is important, while the other end of the spectrum can be ignored. Using SW in this manner allows for high-resolution analysis of a limited band requiring significantly fewer calculations than conventional methods (if information from the whole spectrum is required then standard FFT techniques will use fewer calculations than this method).

For example, if it were desired to analyse a signal's frequency content

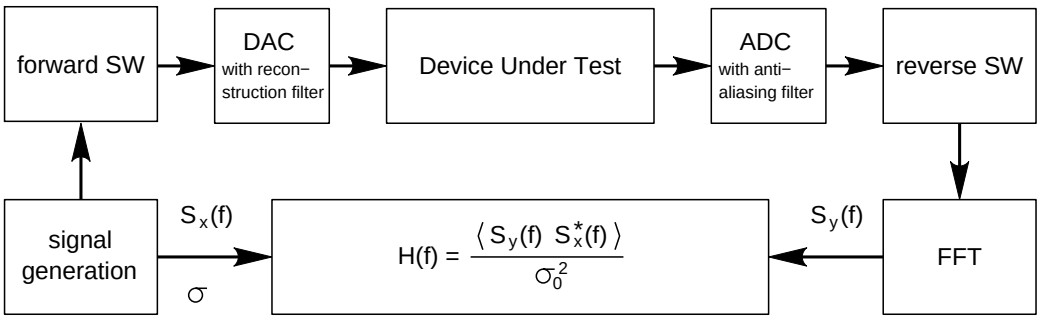

Figure 4.2: **SW-based testing arrangement.**

above 0.9π (and the information about the rest of the spectrum is of no interest), then the signal could be pre-warped using $a = -0.8$. An output sequence of $M = N/2$ could be used as it would not matter that the lower frequency signal components would be truncated. The FFT of the warped signal could then be generated and the result plotted against

$$\bar{\omega} = \theta^{-1}(\hat{\omega}) = \tan^{-1} \frac{(1-a^2)\sin\hat{\omega}}{(1+a^2)\cos\hat{\omega} + 2a} \tag{4.1}$$

where $\hat{\omega} = \theta(\omega)$. So if the frequencies of interest lay in this upper band, then an FFT could be performed on a sequence of half the length, yielding higher resolution while performing fewer calculations.

It is important that SW can be employed for both test response analysis and test signal generation when non-uniform frequency resolution ("focused" or "stretched") is required. This is of particular importance in telecommunication or audio/video circuits and systems testing. The test data flow is shown in figure 4.2. In both the cases (test stimuli generation and response analysis) the amplitude-frequency characteristics of the network, as presented earlier in section 2.2.3, have to be taken into account to obtain correct results from testing.

A signal is applied to the DUT from a signal source of known frequency spectrum $S_x(f)$ and power σ_0^2. The signal output from the DUT is converted from analogue to digital form and transformed by FFT into frequency spectra $S_y(f)$. To limit the measurement errors caused by internal noise, ensemble

averaging can be used, and the transfer function $H(f)$ is found as

$$H(f) = \frac{\langle S_y(f) S_x^*(f) \rangle}{\sigma_0^2} \tag{4.2}$$

where $\langle\rangle$ denotes ensemble averaging and $S_x^*(f)$ is the complex conjugate of $S_x(f)$. SW is used for both the generation of unequally spaced frequencies (with some value of warping parameter a), and for the response analysis (with the warping parameter equal to $-a$, as the frequencies have to be placed back into the uniform mode). Thus the total distortion introduced by the double warping is

$$|H_M(\omega)| = \frac{(1-a^2)^2}{(1+a^2)^2 - 4a^2 \cos^2 \omega} \tag{4.3}$$

To eliminate the amplitude distortion an appropriate correction has to be realised. This can be done, for example, by means of a digital filter that has amplitude-frequency characteristics inverse to $H_M(\omega)$.

4.2.1 Alternative Tools for Non-Uniform Resolution Spectral Analysis

Some other common tools for analysis spectra using non-uniform frequency resolution are outlined here.

The Non-Uniform Discrete Fourier Transform

The non-uniform discrete Fourier transform, proposed by Bagchi and Mitra [20, 24, 35], is closely related to the SW transform. Refer to the discussion in section 2.5 on page 71.

Subband Discrete Fourier Transform

A DFT gives equally spaced samples around the unit circle — i.e., fixed spectral resolution over the whole spectrum. The subband DFT evaluates evenly spaced samples around selected sections of the unit circle [36].

The basic principle of the subband DFT is that the N-length time-domain sequence is separated into its low- and high-frequency subbands and a DFT

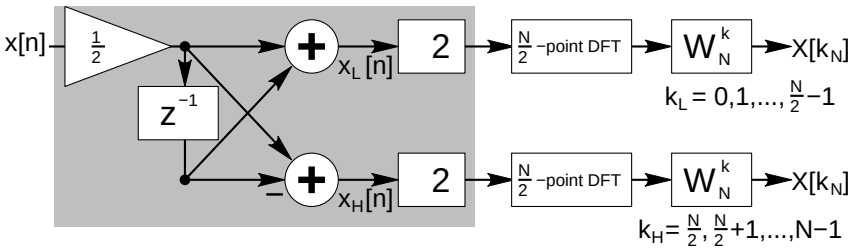

Figure 4.3: **The subband DFT algorithm.** The shaded area shows a single stage of the algorithm [36].

is performed on each of the subbands. These $N/2$-length DFTs may or may not themselves be subband DFTs.

The low- and high-frequency subbands of a sequence $x[n]$ are found by respectively taking the sum or difference of adjacent sample pairs:

$$x_L[n] = \frac{1}{2}\left(x[2n+1] + x[2n]\right) \tag{4.4}$$

$$x_H[n] = \frac{1}{2}\left(x[2n+1] - x[2n]\right) \tag{4.5}$$

for

$$n = 0, 1, \ldots, N/2 - 1$$

This process is illustrated in figure 4.3.

If N is a power of two, this spectral separation step can be repeated $\log_2 N$ times, giving N frequency coefficients. This is similar to the decimation-in-frequency FFT.

If $\log_2 N$ stages of the subband DFT are performed, then N evenly spaced samples around the unit circle are evaluated. However, if information from only some of the subbands is required, then the DFT of the other subbands does not need to be calculated. An example of when this is useful is the case in which a signal's energy is concentrated in certain frequency bands. After each spectral separation stage, a decision whether or not to further decompose each subband, based on its energy content, can be made.

Goertzel's Algorithm

A method for computing a DFT is the *Goertzel's algorithm* [13, 37]. This is a recursive algorithm that is efficient for applications requiring just a few DFT samples.

Chirp z-Transform

The chirp z-transform is another method to give flexibility of sample locations in the z-plane [38]. The chirp z-transform samples the z-domain on a spiral contour on the z-plane. The sampling points are given by

$$z_k = AW^{-k} \qquad k = 0, 1, \ldots, M-1 \tag{4.6}$$

where M is the number of samples, A controls the starting location of the spiral and W controls the shape of the spiral. A and W are given below.

$$A = A_0 e^{j\theta_0} \tag{4.7}$$

$$W = W_0 e^{j\phi_0} \tag{4.8}$$

If $A = W_0 = 1$ and $\phi_0 = 2\pi/M$, then the chirp z-transform reduces to a DFT. A possible chirp z-transform contour is shown in figure 4.4.

The chirp z-transform can be efficiently calculated with order $(N+M)\log_2(N+M)$ operations (N being the length of the pre-transformed signal) [20].

The chirp z-transform is most useful when used to evaluate the z-transform at evenly spaced locations over part the unit circle (that is, when $|A| = 1$, $W_0 = 1$ and $\phi_0 < 2\pi/M$). In the example in figure 4.5, samples are evaluated only over a quarter of the unit circle. If the frequencies of interest lie within this band, only one quarter of the operations are required to evaluate the chirp z-transform compared with calculating a DFT (using direct algorithms).

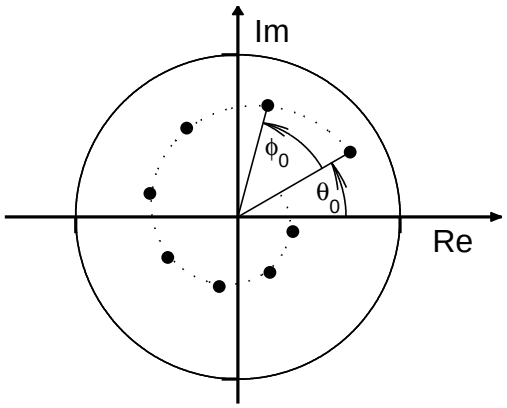

Figure 4.4: **Chirp z–transform sample locations in the z-plane.** $A = A_0 e^{j\theta_0}$ gives the position of the first sample and ϕ_0 is the angle between samples. The chirp z-parameters for this contour are $A = {}^4\!/_5\, e^{j\pi/6}$ and $W = {}^9\!/_8\, e^{-j\pi/4}$.

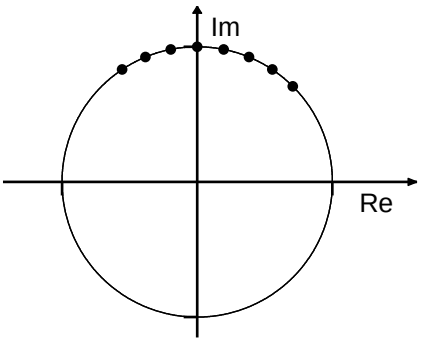

Figure 4.5: ***z*-domain samples from a chirp *z*-transform equal to the DFT for frequencies between $\pi/4$ and $3\pi/4$.** The chirp z-parameters for this contour are $A = e^{j\pi/4}$ and $W = e^{-j\pi/16}$.

4.3 Using SW Chirps to Find a System's Frequency Response

Another way of testing analogue and mixed-signal circuits is based on the evaluation of the frequency response of a DUT. To find the frequency response of a system, one can apply an impulse to the input and measure the response on the output. To provide good accuracy of the measurement, the generated impulse has to be of substantial energy.

However, because all the energy in an impulse signal is concentrated in one point in time (ideally, the pulse has to be infinitesimally short and with infinitely high magnitude), this method can damage delicate electronic circuitry. Spectral warping allows this situation to be avoided by converting an initial pulse into an equivalent chirp signal. Such a signal contains frequencies from the entire spectrum, so the frequency response can also be derived from its chirp response.

An impulse at sample N_0 can be constructed with a linear combination of cosine functions that add at $n = N_0$ and cancel for $n \neq N_0$.

Because superposition holds for the SW transform, the transformation of an impulse can be analysed by observing the transform of the sinusoids that make it up. It is useful to consider the sinusoidal components as spanning the whole sequence, i.e., $n = N_1 = 0$ from to $n = N_2 = N - 1$. Each transformed component still begins at the origin ($\hat{N}_1 = 0s(\omega)$), but lasts until $\hat{N}_2 = N_2 s(\omega)$. The phase of each component at the origin is also unchanged.

The sinusoids no longer add together at one point and cancel each other at all the other points. However, components of similar frequency do, to some extent, constructively interfere at a certain time and destructively interfere at other times. Two illustrate this, consider one of the frequency components that constitute the impulse, labelling it component A. This component originally has frequency ω_A, and moves to frequency $\hat{\omega}_A$ under spectral warping. Time N_0 is the location of the impulse; this is where component A is in phase

with every other component and, consequentially, has its maximum power. The phase angle of component A at time N_0 (before warping) is labelled ϕ_0 (it follows that the phase angle of every frequency component at time N_0 is ϕ_0). After warping, the component has phase ϕ_0 at time $\hat{N}_0(\omega_A)$, as given by

$$\begin{aligned} \hat{N}_0(\omega_A) &= N_0 \tilde{s}(\omega_A) \\ &= N_0 \frac{1 - 2a\cos\omega_A + a^2}{1 - a^2} \end{aligned} \tag{4.9}$$

Section 2.2.2 contains an explanation of the function $\tilde{s}(\omega)$. Now consider the frequency components that have unwarped frequencies close to ω_A. After warping, these components will have phase ϕ_0 at a point in time close to $\hat{N}_0(\omega_A)$, and will constructively interfere with the signal from component A. Because of this constructive interference from its neighbours, component A will have its maximum power at time $\hat{N}_0(\omega_A)$. To generalise this result, when an impulse is spectrally warped, the time in the output sequence at which each unwarped frequency will have its maximum power is given by

$$\hat{N}_0(\omega) = N_0 \frac{1 - 2a\cos\omega + a^2}{1 - a^2} \tag{4.10}$$

The time at which each warped frequency has its maximum power is found by replacing ω with its expression in terms of the distorted frequency $\hat{\omega}$ (from (2.18)).

$$\hat{N}_0(\hat{\omega}) = N_0 \frac{1 - a^2}{1 + a^2 + 2a\cos\hat{\omega}} \tag{4.11}$$

The result is a signal in which the low frequencies have a maximum at one end and the high frequencies have a maximum at the other.

Figure 4.6 shows a chirp signal generated by SW. The warping factor is positive, so the chirp begins with low frequencies and finishes with high frequencies. Towards the end of the chirp only the high-frequency components are present; these remaining components are close in frequency, modulating to cause the low-frequency beating that is seen in the figure.

Up to a certain point all the frequency components are present in the output signal and they cancel each other out. The chirp signal begins when

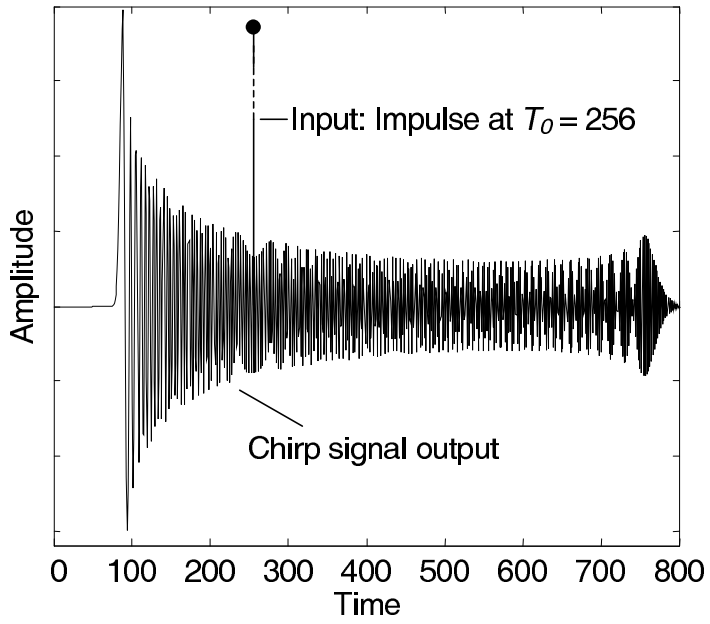

Figure 4.6: **A chirp signal generated by warping a time-shifted impulse with a warping factor of** $a = 0.5$**.** The low-frequency components now occur at an earlier time (they are also amplified) and the high-frequency components occur at a later time.

the first frequency component finishes. This component will be the one relating to either $\omega = 0$ or $\omega = \pi$, depending on the sign of a. If the appropriate value of ω is substituted into (4.11), the start of the chirp, N_1, can be found as a function of the time of the impulse.

$$N_1 = N_0 \times \frac{(1 - |a|)^2}{1 - a^2} \tag{4.12}$$

The end of the chirp is found in a similar manner:

$$N_2 = N_0 \times \frac{(1 + |a|)^2}{1 - a^2} \tag{4.13}$$

To find a system's frequency response, a test chirp is generated by spectrally warping an impulse, using a warping factor a. This signal is then passed through the system or circuit to be tested. This gives the system's *chirp response*.

The Fourier transform of the chirp response has the same magnitude as the system's frequency response. This is because each frequency component of the chirp has equal energy. The phase information is altered due to the time shifting. To recover the phase information the chirp response is warped with $-a$. This produces a warped impulse response that is time-shifted by N_0. Shifting this to start at the origin and then warping it (using the original warping factor a) will make it identical to an impulse response found in the conventional manner. The system's frequency response can be found from the Fourier transform of the impulse response.

It is possible to estimate a system's magnitude response from its chirp response without calculating a Fourier transform. Because the time at which each frequency is dominant in the chirp is known from (4.11), an indication of the magnitude response can be obtained by comparing the envelope of the chirp response with that of the chirp signal. Figure 4.7 shows an example of a chirp response of a band-stop filter with a very narrow stop-band. Only one section of the chirp responses envelope differs significantly from the envelope of the original chirp. This occurs at the time when the stop-band frequency is dominant in the chirp. This frequency can be found by overlaying the

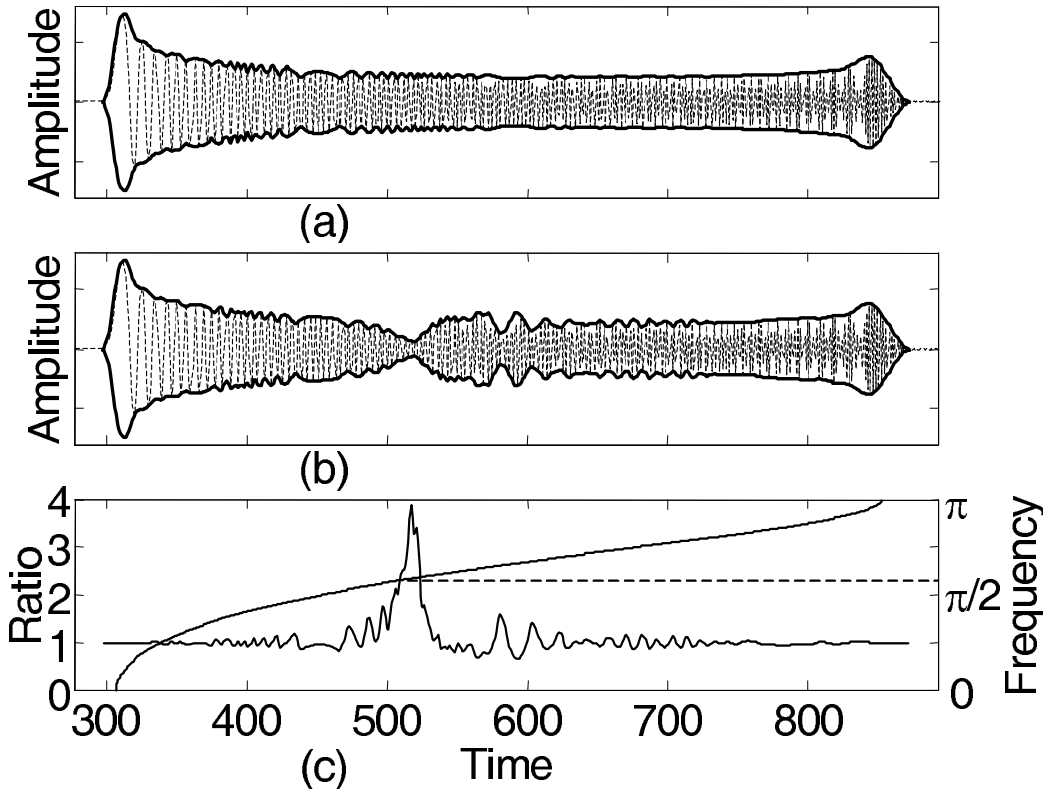

Figure 4.7: **Envelopes of a chirp and chirp response.** The envelope of a SW chirp signal (a) and the resulting chirp response (b) of a band-stop filter. Also shown is the ratio of the two signals overlaid with the frequency versus time curve (c).

frequency versus time curve for the particular chirp. (The known stop band for this example filter is 0.57π to 0.58π). This approximation is only valid if the phase shift introduced by the system is not excessive.

4.4 Accelerated Evaluation of Dependency between Effective Number of Bits of ADC and the Input Signal Frequency

In addition to the evaluation of the frequency response function, the SW transform can benefit testing and characterisation of other parameters of mixed-signal systems. For example, it can accelerate evaluating the dependence between the Effective Number of Bits (ENOB) of an Analogue-to-Digital Converter (ADC) and the input signal frequency. The basic idea of this approach is to bring a full-swing chirp signal to input of the tested ADC where the frequency sweep of the chirp covers the required range of the dynamic test, while the length of the chirp is determined by the required accuracy of the test. The estimation of ENOB with respect to the input signal frequency is then calculated from the series of ADC output samples using an algorithm developed in [39, 40].

The approach provides the same accuracy as a sine-fit test (one of the basic methods for ADC testing as specified by IEEE Standard 1241–2000 [41]). At the same time it requires shorter test time that is a very valuable advantage in ADC final production testing.

It was noted in [40] that a superior chirp generator is required to implement the method. This is due to the fact that most *Direct Digital Synthesiser* (DDS)-based generators are not suitable for such an application as their frequency sweep is built upon discrete frequency increments, thus degrading the test results. The required chirp generator can be implemented employing SW-based techniques that provide the required frequency and amplitude parameters of the test signal.

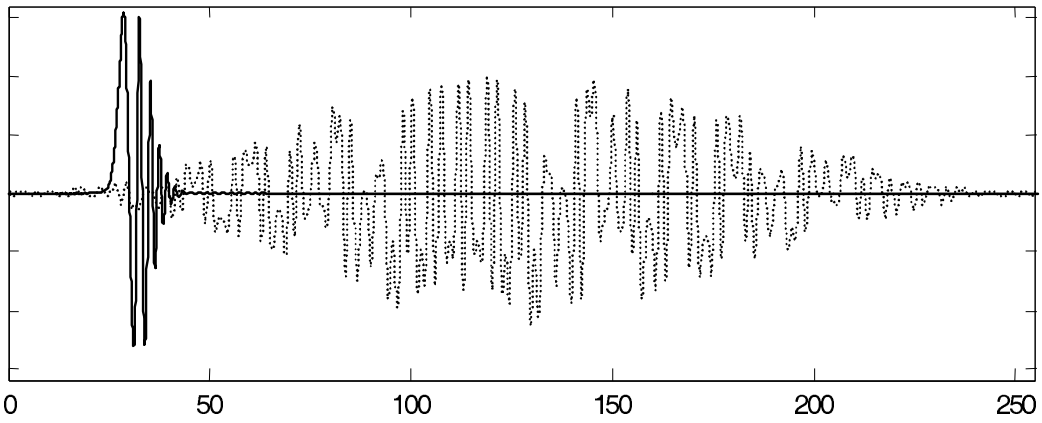

Figure 4.8: **A SW chirp compared with a linear chirp.** The SW chirp signal is plotted as the solid line; the linear chirp signal is the dotted line.

4.5 Using Spectral Warping for Filter Design

SW provides a new efficient way of designing FIR filters. A SW chirp can be upsampled and used as the filter coefficients for a low-pass FIR filter. The low-pass filter can be transformed to a high-pass, band-pass or band-stop filter using appropriate spectral transformation techniques [13]. The process of developing a low-pass filter using the SW procedure is as follows.

A SW chirp is produced using a warping factor a. This chirp is shown in figure 4.8, along with a linear chirp (multiplied by a Hanning window function [42]) for comparison. The chirp signals are then up-sampled by a factor of 32. This produces the filter coefficients. Because the SW chirp is (almost) zero for most of its length, the leading and trailing zero coefficients are removed. In this case, the 501 coefficients remain. The linear chirp filter retains a length of 1892. The SW chirp filter is plotted in figure 4.9. Sharing the same axes is a plot of a Parks-McClellan FIR filter, designed to have the same characteristics as the two chirp filters.

Figure 4.10 shows the frequency response of the three filters (SW chirp filter, linear chirp filter and Parks-McClellan filter). It appears from figure 4.10 that the SW chirp filter can achieve a comparable magnitude response

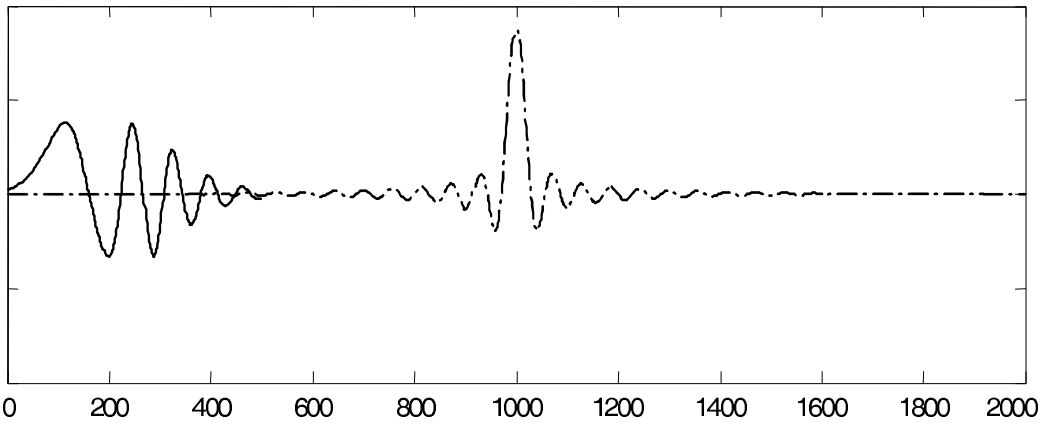

Figure 4.9: **A SW chirp compared with a Parks-McClellan low-pass filter.** The SW chirp signal is the solid line; the Parks-McClellan low-pass filter is the broken line.

to a Parks-McClellan filter of four times the order. A SW chirp filter will give a much better magnitude response than a Parks-McClellan filter of the same order.

The linear chirp filter has much more ripple in the pass-band and requires a much larger order to achieve similar stop-band characteristics as the SW chirp filter. However, that the SW method produces a much greater phase shift. A related disadvantage is that the SW chirp filter produces a large, non-constant group delay, as illustrated in figure 4.11.

In summary, the SW chirp filter provides significant advantages where fast (it can be a smaller order than Parks-McClellan), simple (FIR filters are simpler to implement than IIR ones) and stable (FIR filters are always stable) filtering is required.

Spectral warping methods have also been employed to design IIR filters [43], with particular attention given to their use in audio applications [44].

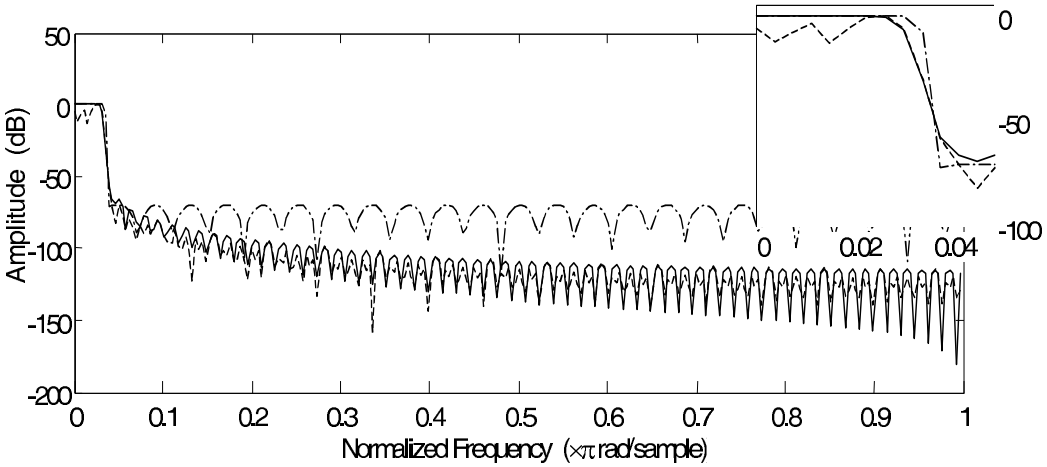

Figure 4.10: **Magnitude frequency response of the SW chirp filter compared with the linear chirp and the Parks-McClellan filters.** The solid, dotted and broken lines show the frequency response of the SW chirp, the linear chirp and the Parks-McClellan filters, respectively.

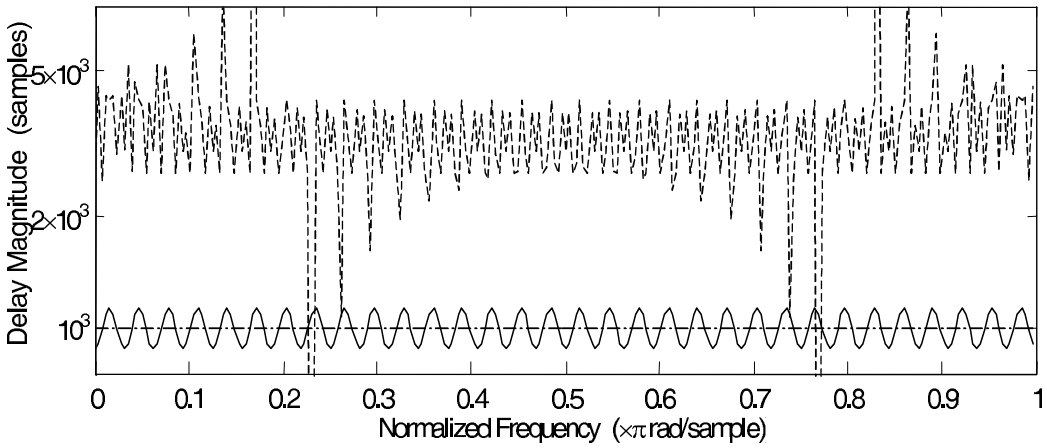

Figure 4.11: **Group delay of the SW chirp, linear chirp and Parks-McClellan filters.** Solid line: SW chirp filter; dotted line: linear chirp filter; broken line: Parks-McClellan filter.

4.6 Harmonic Distortion Measurement using Spectral Warping

Appending a sequence of zeroes to a signal before it undergoes spectral warping causes the various frequency components contained within the signal to become more separated in time after the warping. This effect has been investigated for use in the analysis of the harmonic distortion[1] a system introduces to a signal in [45]. That study concluded that estimating harmonic distortion using the SW transform provides an efficient method that gives moderate accuracy, and could, in some applications, provide a viable alternative to more conventional filtering methods.

4.7 Other Applications

Other authors have investigated applications for SW warping in the areas of designing of audio equalisers [46], warped wavelet design [47] and speech analysis [48].

[1] Harmonic distortion can be defined as the proportion of energy of a sinusoidal signal transferred to the harmonics.

Chapter 5

Conclusions and Future Work

The major results of this thesis, and the conclusions that can be drawn from them, are summarised and discussed below.

5.1 The Types of Spectral Warping Transforms

The SW transform has been defined as a time-domain-to-time-domain transform that shifts the frequency components of a signal along the frequency axis, characterised by a frequency (or z-plane) mapping function. Two important categories of the SW transform were considered: transforms for which the z-plane mapping is an all-pass function

$$\Theta_N(z) = \prod_{n=1}^{N} \frac{z - a_n}{1 - a_n^* z} \qquad |a| < 1 \tag{5.1}$$

and transforms for which the frequency mapping function is piecewise linear. All-pass SW can be further categorised into first-order ($N = 1$) and higher-order ($N > 1$) transforms. It was shown that higher-order all-pass SW transforms introduced a type of frequency aliasing, making them useless for most common applications. First-order all-pass SW transforms do not suffer from this aliasing and were found to be useful, and lent themselves to straight forward analytical analysis.

5.2 Matrix Representation

A significant new result from this work is viewing the SW transform as linear transformation. This view of SW is more general than how the literature had previously considered SW. It enables the many tools and techniques applicable to linear transforms to be applied to the SW transform. A linear transform can be represented as a matrix multiplication. The matrix representation of the SW transform that was proposed effectively decomposes the transformation into three components:

1. a DFT from the original time domain samples into the frequency domain,

2. a warping matrix that interpolates the available frequency samples to give a new set of frequency samples, and

3. an inverse DFT to transform these back into the time domain.

The transformation matrix effectively represents a bank of FIR filters that can be used to implement the warping. From the matrix representation, it is shown that the spectral warping transform is not time invariant, although the impulse responses are readily available from the transformation matrix.

5.3 The Uses of Spectral Warping Transforms

The manipulation of the frequency content of digital signals using SW can be employed to analyse signals' frequency content with non-uniform resolution over the frequency spectrum. This technique is most useful (in the sense that it can reduce the number of calculations required compared with conventional methods) in the case when information from one end of the spectrum is important and the other end of the spectrum can be ignored.

An important consideration that must be accounted for when using the SW transform is that the transform operates on a finite-length "frame" of

input samples, producing a finite-length frame of output samples (that generally has a different length to the input frame). The output frames may be rejoined by simple concatenation, or by some other method, but, in general, there is no correct way to rejoin the output frames — the best method to rejoin the frames (and even whether it is sensible to rejoin the frames at all) is dependent on the particular application.

Another new technique developed through this thesis is the production of spectrally-flat frequency chirps using the SW transform, and the use of such chirps for system characterisation. Although using chirp signals for system characterisation is a standard technique, the use of SW to develop the chirp signals is a new result, which has the potential to be useful in practical situations, due to the nice properties of SW chirp signals. SW chirps were shown to be successful in characterising a system's response both exactly and approximately (by appraising the envelope of the SW chirp response directly in the time domain, thereby using fewer operations).

The SW transform was also successfully used to generate useful FIR filters. A SW low-pass filter was shown to have similar magnitude-response characteristics to a Parks-McClellan filter of four times the order. These superior magnitude-response characteristics come at the expense of poorer phase-response characteristics.

5.4 Realisation and Implementation

FIR and IIR realisations of the SW transform were investigated[1]. There exists an efficient implementation for an IIR realisation of a first-order all-pass SW transform that has the benefit of requiring fewer calculations than a typical FIR implementation.

Both FIR and IIR realisations require that the output sequence be of sufficient length to represent the entire transformed signal. If the output

[1]The IIR realisations are so called because they are based on IIR filters, not because the transform has an infinite response itself.

sequence is too short in an FIR realisation, the signal represented by the truncated samples is superimposed on the initial samples (the signal wraps around to the beginning). If the output sequence is too short in an IIR realisation, the truncated samples are either lost or added to the start of the next output frame, depending on the particular implementation.

5.5 Arithmetic Logic Unit Core Application-Specific Integrated Circuit

Because the SW ALU core ASIC did not seem to function correctly, it is difficult to draw many conclusions from the exercise of producing the chip.

Although no formal analysis of quantisation noise has been carried out, it is evident from simulation results that using five-bit precision for the SW transform results in the quantisation noise to be so large as to almost completely obscure the signal. Hence, one conclusion that can be made from the exercise is that any serious evaluation of SW as a component of DSP-based testing must be done using longer word lengths. An analytical study of quantisation noise verses word-lengths will reveal how many bits precision is needed for a given application.

5.6 Future Work

Areas where further study would benefit the understanding of the SW transform, and its practical use, include:

- Modifications to the FIR filter to improve the phase response.
- Further study of IIR SW filters.
- Analysis of the quantisation noise introduced in various implementations of the SW transform, including the effect of varying the word-length used.

- A comparison of using SW to evaluate a subband with Goertzel's algorithm.

Further testing of the ASIC is also required. This will fully determine where the faults, if any, are. Once faults are isolated, they can be fixed and an improved die fabricated. This is necessary to be able to make statements about the usefulness of such an implementation, and to aid in the development of other implementations using the same algorithm. It could also be interesting to implement both the IIR and FIR implementations on a field programmable gate array (FPGA), to see how that technology would suit using SW for system characterisation.

5.7 Final Conclusions

A valid conclusion to draw from the analysis and experimentation reported in this thesis is that the SW transform is a useful tool for the manipulation of the frequency content of digital signals. In particular, SW is useful and efficient for cases where the signal or system under investigation is band-limited, or where only the frequency content/response over a limited band is of interest. Under these circumstances, the SW transform may be a valuable tool for embedded mixed-signal testing.

Appendix A

SW ALU ASIC Data Sheet

An application-specific integrated circuit (ASIC) that provides a specialised arithmetic logic unit (ALU) core that performs a calculation central to the IIR all-pass network implementation of the SW transform was designed and fabricated using very-large-scale integration technology (VLSI).

The ultimate goal of the core is to investigate the use of SW as implemented in CMOS, in particular its use in the characterisation of mixed-signal ICs It is a proof-of-concept realisation, from which further designs can be derived. It is envisaged that SW techniques will be used as tools in mixed-signal IC characterisation (in fact, characterisation of any unknown arbitrary system) and high-resolution frequency analysis, plus many applications that are waiting to be discovered.

The objectives of the ALU core are to:

1. Evaluate the usefulness of the SW transform implemented with five-bit precision (five-bit precision was chosen because that was the largest word-length that would allow the chip to be developed in reasonable time with the design tools available).

2. Develop a prototype for both the ALU core and the surrounding circuitry, which can be further developed as required.

3. Experiment with real-time applications of SW.

A.1 Acknowledgements

The circuit was designed using DSCH2 for the gate-level design and Microwind2 as the layout editor.[1] It was fabricated using 0.18 μm ST CMOS technology, as one section of a multi-purpose chip.

The development of this core was done in partnership between the Institute of Information Sciences and Technology, Massey University, New Zealand and Institut National des Sciences Appliques, France.

A.2 SW ALU Core Overview

The function the ALU performs is

$$F(a, b, c, d) = a(b - c) + d \tag{A.1}$$

All the operands are five bits long (four bits plus a sign bit). The output is of the same form. An overflow output bit is also provided. This bit indicates whether an overflow has occurred at any stage of the calculation. Note that no overflow occurs during the multiplication stage of the calculation because the product is truncated. The four high-order bits of the product are taken, while the low-order bits are discarded.

The design takes advantage of the fact that identical operations are performed on the data at different points in the calculation. This means that circuitry can be reused within the calculation. To do this, intermediate results are stored in registers. The four input words (a, b, c and d) are loaded at separate times via the same input pads. This significantly reduces the number of pads and, hence, the chip's size.

Because a is a constant, it is loaded into a register at the start of the transform and the data in that register is reused for the subsequent iterations of the function. A notLDA (load a) pad is provided to indicate that the data on the input pads is to be loaded into the a register on the next rising edge

[1] Both DSCH2 and Microwind2 are written by Etienne Sicard of INSA Toulouse, France.

of the clock. notLDA also acts as a reset, clearing the output register and resetting the chip back to the first stage of the calculations. As a result of the truncation in the multiplication stage, a is inherently scaled by 2^{-4}. To cancel this effect a should be pre-scaled by 2^4. This is useful for SW, where $-1 < a < 1$.

A.2.1 ALU Input and Output Nodes

Table A.1 lists the input and output nodes of the ALU. The ALU requires 23 inputs, 6 outputs (plus two power supply inputs).

A.2.2 Top-Level Functional Blocks

The ALU consists of 12 functional blocks. Figure A.1 shows the data flow through these blocks.

Table A.1: **ASIC I/O list.** Input/output list with internal names. See also Table 15 on page 35.

Node	Type	Description	Normal
Vdd	Power	+1.8V power supply.	+2V
Vss	Power	0V power supply.	0V
in_clk	Input (control)	External clock signal.	n/a
in_notLDA	Input (control)	Not load a: allows the warping factor to be loaded into the a-register. Also acts as a reset pin.	HIGH
in_mem	Input (control)	Input mux enable: when this line is HIGH the data is loaded in parallel, when it is LOW the data is loaded sequentially via the A data lines.	n/a
in_da4	Input (data)	(most significant bit; sign bit)	n/a
in_da3			
in_da2		A input data word.	
in_da1			
in_da0		(least significant bit)	
in_db4	Input (data)	(most significant bit; sign bit)	n/a
in_db3			
in_db2		B input data word.	
in_db1			
in_db0		(least significant bit)	

Table A.1 (cont): **ASIC I/O list.**

Node	**Type**	**Description**	**Normal**
in_dc4		(most significant bit; sign bit)	
in_dc3	Input		
in_dc2	(data)	C input data word.	n/a
in_dc1			
in_dc0		(least significant bit)	
in_dd4		(most significant bit; sign bit)	
in_dd3	Input		
in_dd2	(data)	D input data word.	n/a
in_dd1			
in_dd0		(least significant bit)	
oreg_OV	Output	Is HIGH if an overflow has occurred during the last calculation. If this is HIGH, the output data word is set to ± 15.	LOW
oreg_q4		(most significant bit; sign bit)	
oreg_q3	Output		
oreg_q2	(data)	Output data word.	n/a
oreg_q1			
oreg_q0		(least significant bit)	

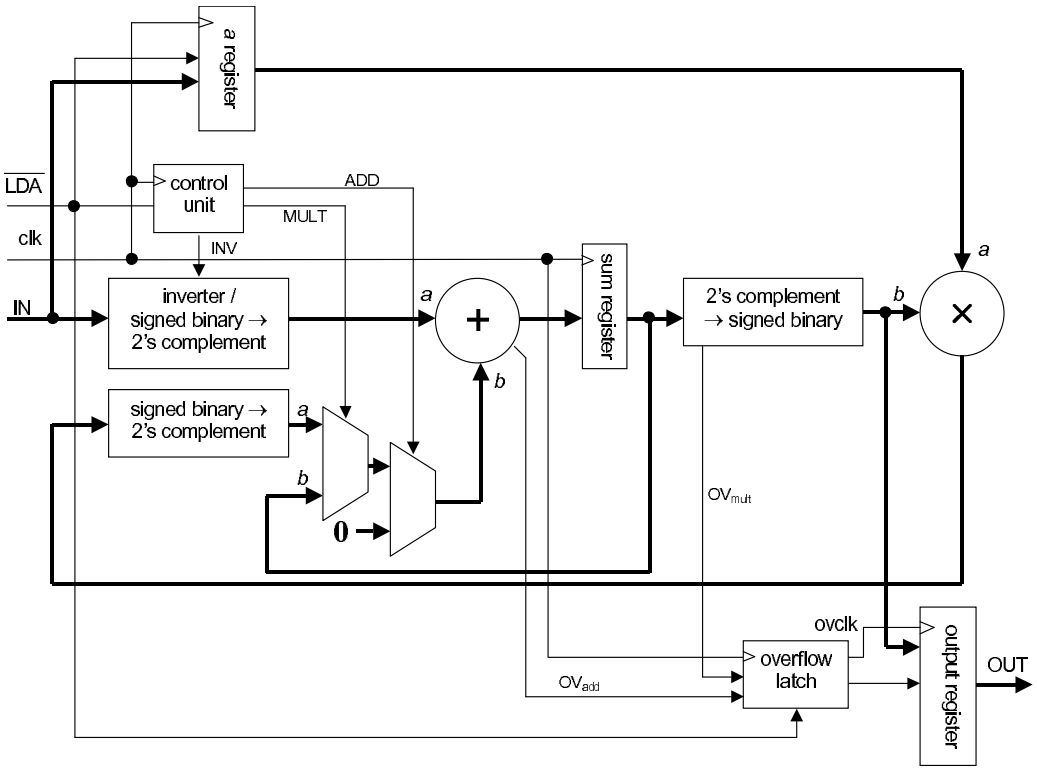

Figure A.1: **ASIC ALU dataflow.** Data flow through the ALU (not including input multiplexer).

The functions of the 12 blocks are listed in table A.2.

Note that a clock period is considered to start on the rising edge of the clock signal. Also, the least significant bit of a parallel data line is denoted by a 0 subscript and, if a sign bit is present, it is denoted by the highest subscript (i.e., 4).

A.2.3 Operation

Each cycle consists of a single four-period calculation, noting that the cycles overlap by one stage (clock period). The control unit directs the data flow through the ALU, according to the current stage. Each stage is considered to finish when the data is clocked into the sum register. Table 3 shows the

Table A.2: **The functional blocks of the ALU.**

Block	Inputs	Outputs
Input multiplexer — inmux If in_men is HIGH selects between the appropriate input data lines, or, if in_men is LOW, always selects the A input.	notLDA in_men muxS ADD INV inmux_dA$_{4\rightarrow0}$	inmux_q$_{4\rightarrow0}$
Control Unit — ctrl Synchronises the calculation performed to the data currently on the input pads. Reset by notLDA.	CLK notLDA	muxS ADD INV
a **Register** — areg Stores the 5-bit *a* constant. Loads input when notLDA is pulled low. Rising edge triggered.	CLK notLDA inmux_q$_{4\rightarrow0}$	areg_q$_{4\rightarrow0}$
Pre-adder 2's Complementor with Inverter — isb2c Converts the data bits of the output of the Input multiplexer to their 2's complement if the sign bit (mux_q$_4$) or INV is set (but not both). If iumux $= -0$ (10000_2) then the output is 00000_2.	INV inmux_q$_{4\rightarrow0}$	isb2c_q$_{4\rightarrow0}$
Pre-adder multiplexer — mux Selects the B operand of the Adder from either the Sum Register (muxS $=$ 0) or the product of the multiplication, via the 2's complementor (muxS $= 1$).	muxS sb2c_q$_{4\rightarrow0}$ mult_q$_{4\rightarrow0}$	mux_q$_{4\rightarrow0}$

Table A.2 (cont): **The functional blocks of the ALU.**

Block	**Inputs**	**Outputs**
Pre-multiplexer 2's Complementor — sb2c Converts the data bits of the output of the Multiplier to their 2's complement if the sign bit (mux_q_4) is set. If mult $= -0$ (10000_2) then the output is 00000_2.	INV $mult\_q_{4\rightarrow 0}$	$sb2c\_q_{4\rightarrow 0}$
Adder — add Adds the A operand to either the B operand (ADD $= 1$), which is either the intermediate result stored in the Sum Register or the product of the multiplication (via the 2's complementor), or adds it to 0 (ADD $= 0$). An overflow bit is set if the result of the addition is greater than $01111_{2s\ cmp}$ or less than $10000_{2s\ cmp}$.	ADD $isb2c\_q_{4\rightarrow 0}$ $mux\_q_{4\rightarrow 0}$	$add\_q_{4\rightarrow 0}$ add_ov
Pre-multiplier 2's Complementor — 2csb The output of the Adder (stored in sreg) is in 2's-complement form. If it is a negative number then the 2's complement of the four data bits is taken — the sign bit is left unchanged. An overflow is generated if $add_q = 10000_{2s\ cmp}$.	$sreg\_q_{4\rightarrow 0}$	$2csb\_q_{4\rightarrow 0}$ 2csb_ov
Multiplier — mult Multiplies two 4-bit numbers and takes the XOR of the sign bits. The product is truncated to be four bits long.	$areg\_q_{4\rightarrow 0}$ $2csb\_q_{4\rightarrow 0}$	$mult\_q_{4\rightarrow 0}$
Sum Register — sreg Holds the output of the addition for one clock period. Rising edge triggered.	CLK $add\_q_{4\rightarrow 0}$	$sreg\_q_{4\rightarrow 0}$

Table A.2 (cont): **The functional blocks of the ALU.**

Block	**Inputs**	**Outputs**
Overflow Latch — ovfl If add_ov or $2csb_ov$ are set during any stage of a complete calculation, then $ovfl_qov$ will be set for the remainder of the calculation. This unit also provides the triggering for the data to be latched into the Output Register.	CLK notLDA INV ADD MULT add_ov 2csb_ov	ovfl_qov ovfl_ovclk
Output Register — oreg Latches out the final value once the each calculation has completed. The register is cleared by notLDA.	$ovfl_ovclk$ $add\_q_{4\rightarrow0}$ notLDA INV $sreg\_q_{4\rightarrow0}$ ovfl_qov	$oreg\_q_{4\rightarrow0}$

steps involved in each calculation.

The control unit directs the data flow through the ALU. For the following explanation of the circuit's operation, each stage (clock period) finishes when the data is clocked into the sum register. A cycle consists of a single four-period calculation.

1. During the first stage of each cycle, the input data (that is the c operand) is converted to two's-complement form, negated, added to zero and stored in the sum register.

2. In the second stage, b is applied to the input pads. Its two's-complement representation is added to the contents of the sum register (selected by the multiplexer). Thus, $b - c$ is stored in the sum register.

3. d is applied to the chip's inputs for the third period. It is converted to two's-complement form and presented at one of the inputs to the adder. The value in the sum register is converted to signed-binary magnitude representation and multiplied by a. The product is converted to two's-

Table A.3: **The stages of loading and calculating each expression.** The result of each operation is stored in the sum register. The output data is presented on the outputs on the falling edge of the third clock period of each iteration.

Stage	Input	Operation	Output
0	a	load a	—
1	c_1	$-c_1$	—
2	b_1	$b_1 - c_1$	—
3	d_1	$a(b_1 - c_1) + d_1$	$a(b_1 - c_1) + d_1$
1	c_2	$-c_2$	$a(b_1 - c_1) + d_1$
2	b_2	$b_2 - c_2$	$a(b_1 - c_1) + d_1$
3	d_2	$a(b_2 - c_2) + d_2$	$a(b_2 - c_2) + d_2$
		$\vdots$	

complement form, passes through the multiplexer and is presented at the other input to the adder. The sum, $a(b - c) + d$, is stored in the sum register.

4. The output of the sum register is converted to signed-binary magnitude representation and presented to the output register. The output register latches the data through to the chip's outputs on the following falling edge. This fourth period is also the first period of the next calculation.

A.2.4 Supply

The ALU core requires a 1.8V Vdd and a 0V Vss (with respect to the substrate) supply. The supply is delivered to the circuitry via the comb routing scheme shown in figure A.2.

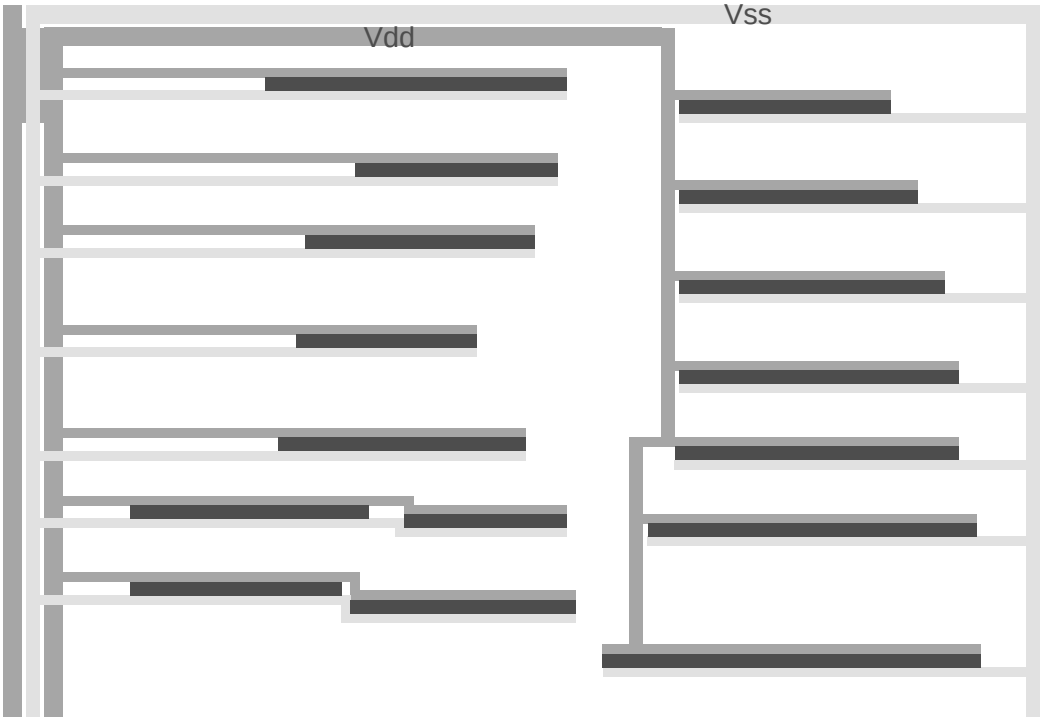

Figure A.2: **Power supply routing.**

Table A.4: **Best order to apply inputs.** The order to change input values to minimise the effect of delays (from capacitance etc.) of the input pads. The new value should be applied immediately after a rising edge.

Stage	**Inputs**				
Number	notLDA	A	B	C	D
-3	0	load			
-2	apply 1	↓		load	
-1	↓	↓	load	↓	
0	↓	ready	↓	↓	load
1	1		↓	ready	↓
2	1		ready	load	↓
3	1		load	↓	ready
1	1		↓	↓	load
2	1		↓	ready	↓

A.2.5 Clocking

The clock line enters the layout from the left side (for the chip orientated as in figure A.2) and runs down the routing channel in the centre of the design. The latches in the design, with the exception of the output register, are rising-edge triggered.

The output value changes at most once every three clock cycles. If the input multiplexer is used, the inputs can also be updated at a third of the clock speed. If the time taken to change the value on the input pads is critical, then it may be important to apply the new value to the pad as soon as possible. In this case the order that the inputs should be updated is shown in table A.4.

It has not yet been determined how fast the clock can be run.

A.2.6 Reset

The notLDA input also acts as a reset. When notLDA is pulled low the following occurs:

1. The control unit will output INV = 1, ADD = 0, MULT = 0 on the next clock rising edge. It will remain in this state until notLDA returns high, after which the output will begin its normal cycle (see Table 9).
2. The input multiplexer selects the A input word.
3. The A input word is clocked into the a register.
4. The output clock goes high after a rising edge and a falling edge of the main clock, after notLDA goes low (see Figure 7).
5. The output register is cleared on the rising edge of the output clock.

A.2.7 Design Tools

MATLAB, DSCH2 and Microwind2 were the principal software tools used to develop the ASIC design (figure A.4).

A.3 Circuit Components

A.3.1 Basic Blocks

The design is made up of primitive gates (such as NAND, NOR and transmission gates), and with simple blocks that are combinations of a few primitives. The blocks used are described below.

Edge-Triggered D-Latch

Standard

An RS-latch (with inverted inputs) can be made from two NAND gates (see figure A.5) and has the truth table shown in table A.5.

This can be used to make a D-latch by ensuring $\overline{\text{SET}} \neq \overline{\text{RESET}}$ only when the data is to be latched through (in this case on a rising clock edge). At other times $\overline{\text{SET}} = \overline{\text{RESET}} = 1$. This is illustrated in figure A.6. Table A.6 is the corresponding truth table.

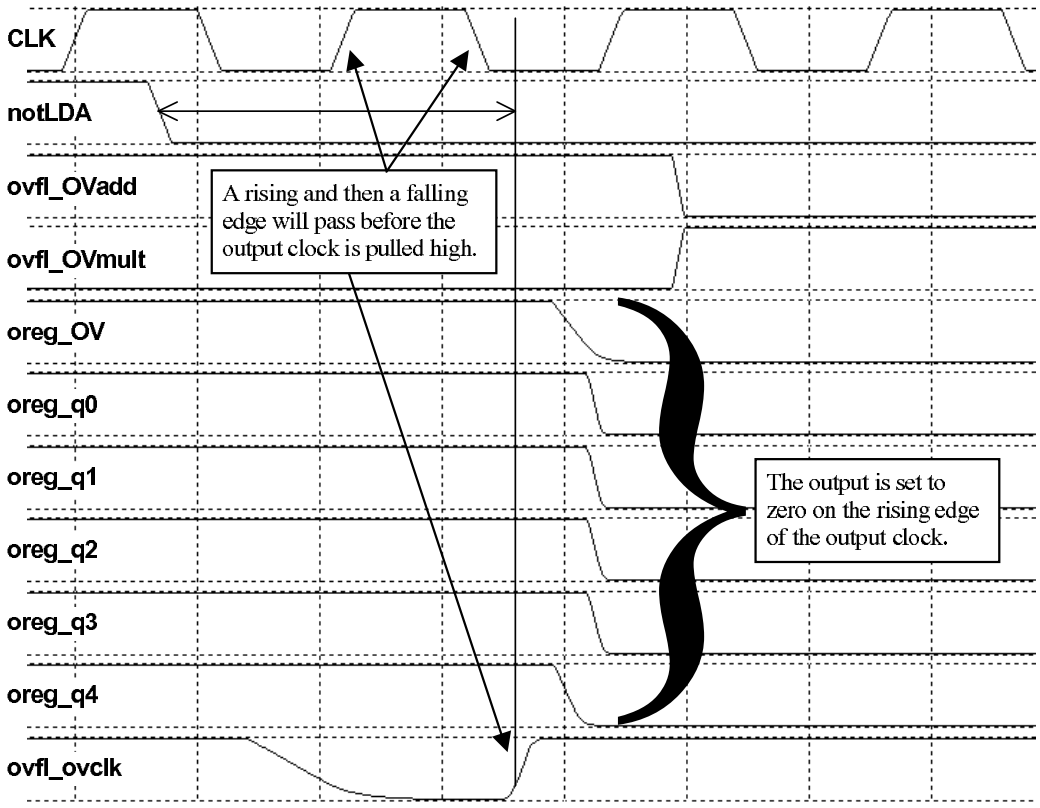

Figure A.3: **Microwind trace showing the effect of setting** notLDA $= 0$ **on the overflow latch and the output register.** Note that there are one and a half clock cycles between notLDA going low and the output being reset.

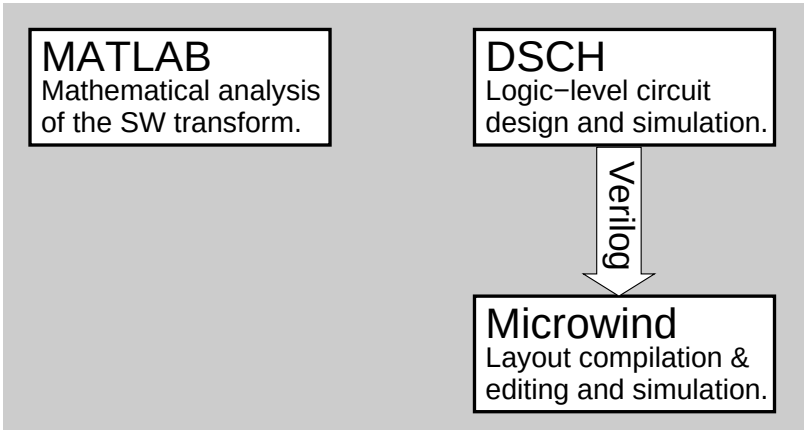

Figure A.4: **CAD tools used in developing the design.**

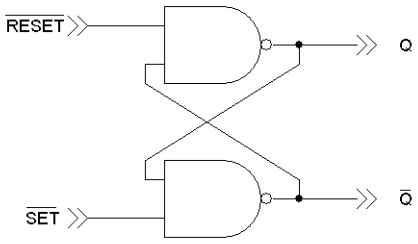

Figure A.5: **RS-latch.**

Table A.5: **Truth table for NAND RS-latch.**

$\overline{\text{SET}}$	$\overline{\text{RESET}}$	Q	$\overline{\text{Q}}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q	$\overline{\text{Q}}$

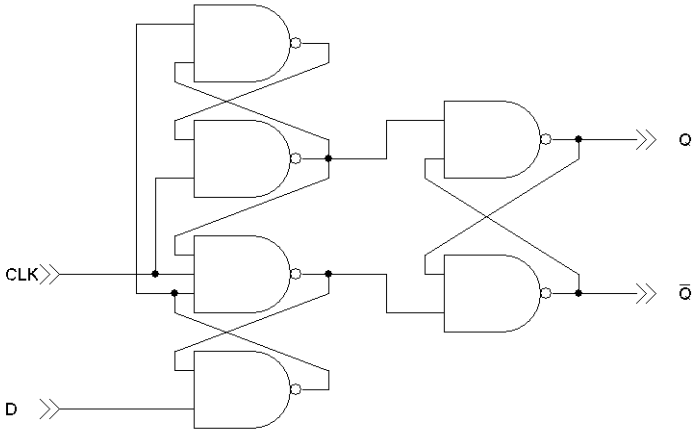

Figure A.6: **Rising edge-triggered D-latch.**

Table A.6: **Truth table for rising edge-triggered D-latch.**

CLK	Q	$\overline{Q}$
0	Q	$\overline{\overline{Q}}$
0→1	D	$\overline{D}$
1	Q	$\overline{Q}$
1→0	Q	$\overline{Q}$

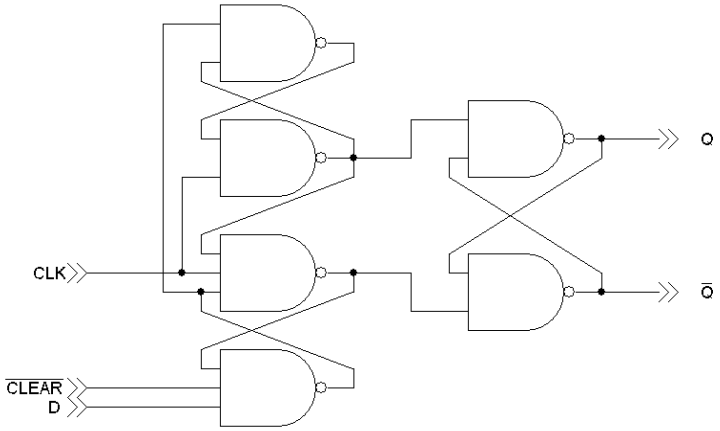

Figure A.7: **D-latch with clear.**

With Clear

The latch can be cleared by ANDing the input with a $\overline{\text{CLEAR}}$ line (this is done using a three-input NAND gate). This is a synchronous clear — it only takes effect on a rising clock edge.

Adders

Half Adder

The half adder circuit (as in figure A.8) is a fundamental block that is used extensively throughout the design.

A half adder with inverted inputs (figure A.9) was also found to be useful. This is made by simply swapping the AND gate with a NOR gate. A third

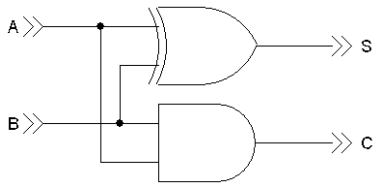

Figure A.8: **Half adder.**

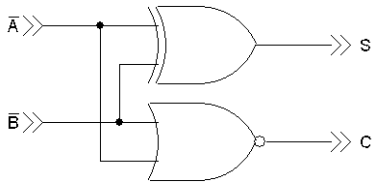

Figure A.9: **Half-adder with inverted inputs.**

version of the half adder that has the carry bit inverted (made with a XOR and a NAND gate) was also used.

Full Adder

To cascade adders, allowing addition of words of an arbitrary bit-length, provision has to be made to include carry-in bits. This is a full adder, the schematic for which is illustrated in figure A.10.

4-Bit Adder

In the design of the multiplier it seemed useful to have a 4-bit adder block. This block can easily be made from five half adders and a NOR gate (figure A.11). Blocks with two and four inverted inputs were also used (made by simply swapping one or both, of the input half adders with inverted-input half adders). A version of the 4-bit adder with an inverted carry-out was used where appropriate. The reason for using inverted inputs and/or output is to reduce the number of inverters in the circuit.

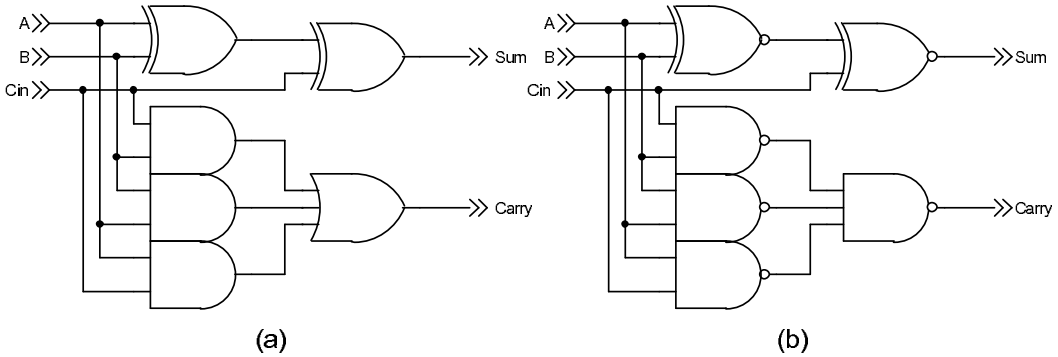

Figure A.10: **Full adder** with (a) AND, OR and XOR gates and (b) a more efficient implementation with NAND and XNOR gates (as in the actual implementation).

Table A.7: **Truth table for full Adder.**

A	B	Cin	carry	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

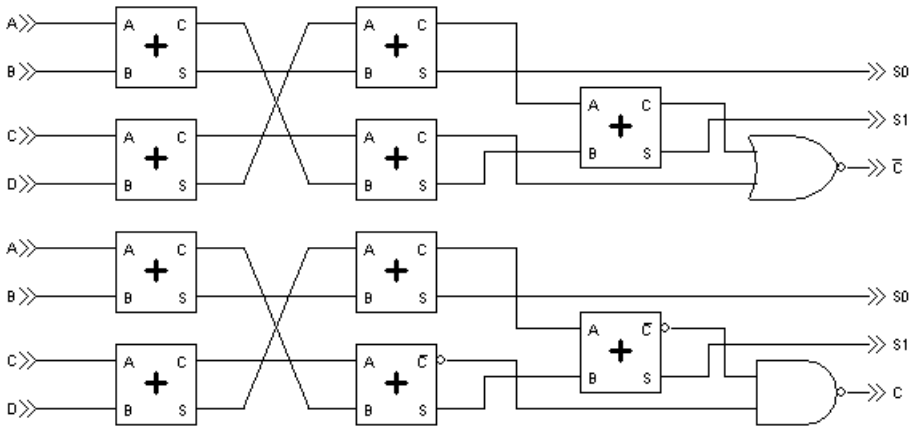

Figure A.11: **4-bit adders** with (top) invert and (bottom) non-invert carry-out.

Multiplexers

A multiplexer selects one of several inputs (all the multiplexers in this project select one of two inputs). The multiplexers are made using transmission gates. When the control line is high, half of the transmission gates allow current to flow through them and the other half act as open circuits. This selects one set of inputs. When the control line goes low the other half of the transmission gates conduct, selecting the other input. The first lot of transmission gates is now in a high-impedance mode. Figure A.12 shows an arbitrary-length multiplexer of this form.

A multiplexer of the form shown in figure A.12 is not an optimal implementation. Each of the transmission gates is implemented (when complied in Microwind) as in figure A.13. Every transmission gate inverts the control line for the gate of the PMOS transistor independently of the others. This is unnecessary because the inverse of the control line is already available. A better implementation is achieved by designing the multiplexers in the transistor level (as in figure A.14), thus saving several inverters.

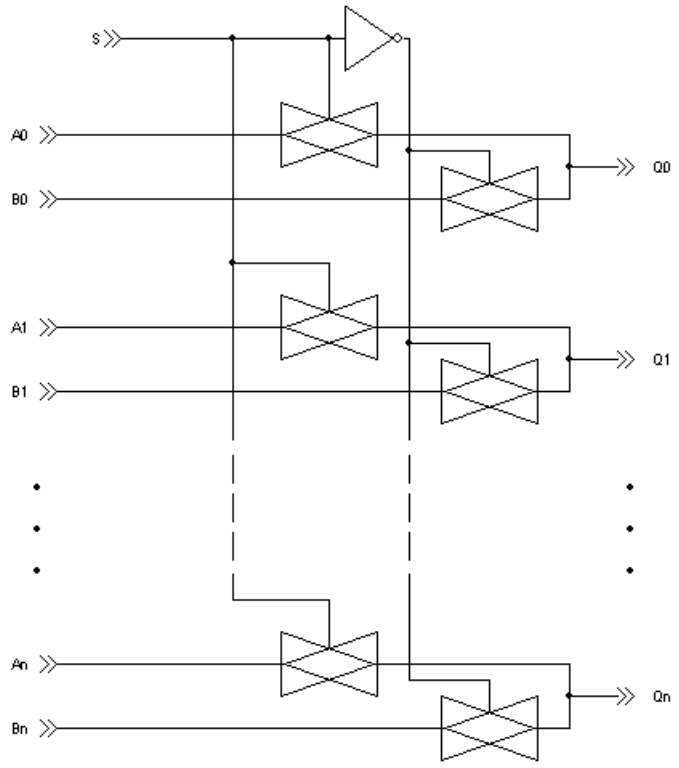

Figure A.12: **multiplexer using transmission gates.**

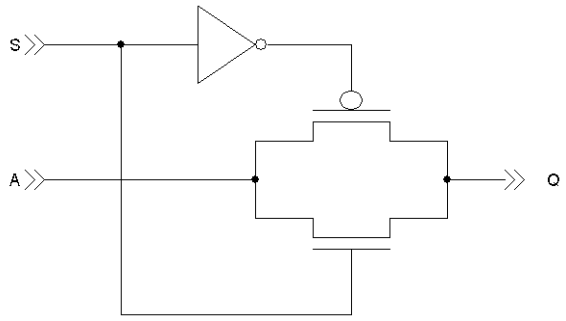

Figure A.13: **Transmission gate.**

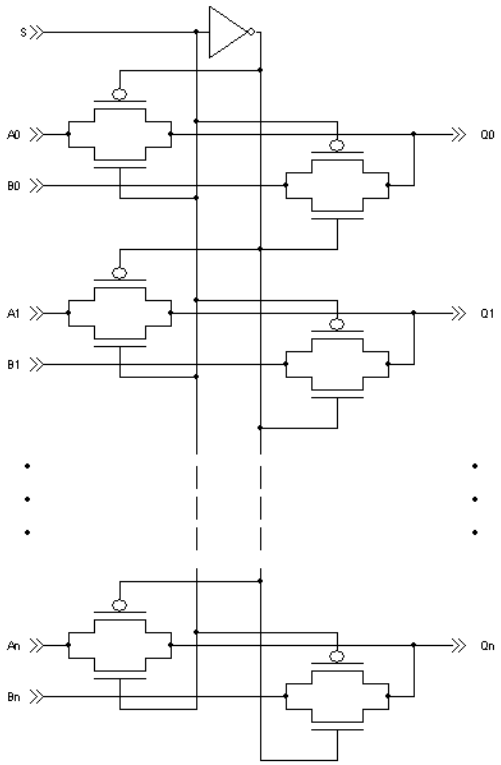

Figure A.14: **multiplexer designed at the transistor level.**

Table A.8: **The outputs of the control unit as a function of the current stage of calculation.**

Stage	INV	ADD	MULT
0	1	0	0
1	0	1	0
2	0	1	1
3	1	0	0

Table A.9: **Control unit truth table for transition on the rising edge of the clock.**

LDA	**Current Stage** n	INV $n+1$	ADD $n+1$	MULT $n+1$	next stage $n+1$
1	0	0	1	0	1
1	1	0	1	1	2
1	2	1	0	0	3
1	3	0	1	0	1
0	x	1	0	0	0

A.3.2 Control Unit

The control unit sets the various control lines according to the current stage of the ALU's calculation cycle (see table A.3). Table A.8 shows the value of the control lines for each stage and table A.9 is the truth table for the control unit. The circuit to achieve this is shown in figure A.15.

A.3.3 Multiplier

The multiplier is a 4-bit basic array multiplier. It is based on the "paper-and-pencil" algorithm illustrated in figure A.16.

Partial products are generated by ANDing the bits of the multiplier and the multiplicand. The four rows of partial products are then added to generate the result.

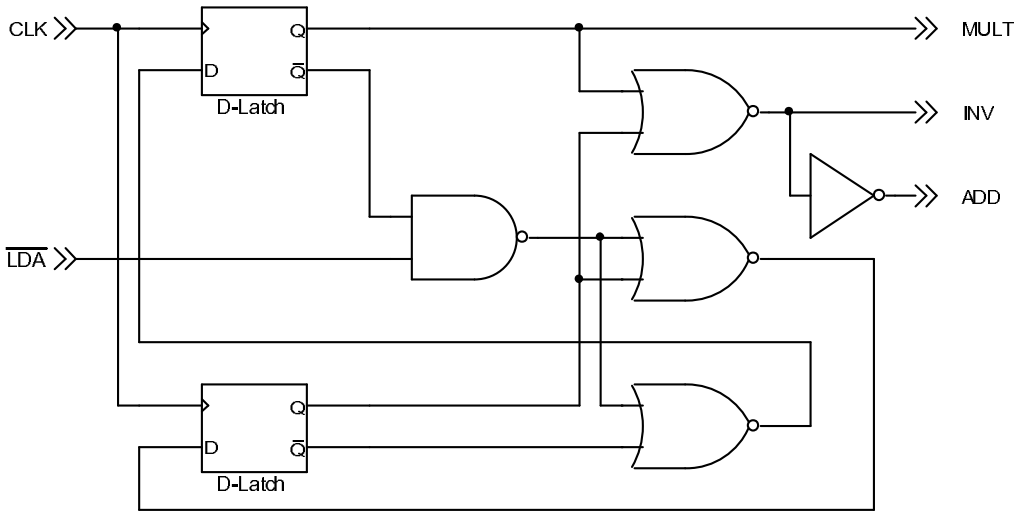

Figure A.15: **Control unit.**

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
\times & & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
+ & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
+ & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
+ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
\hline
 q_7 & q_6 & q_5 & q_4 & q_3 & q_2 & q_1 & q_0
\end{array}
$$

Figure A.16: **Array multiplication using the "paper-and-pencil" algorithm.**

The addition of each output bit is dependent on any carries generated from the addition of the lower order bits. The expression for each output bit is given by the following equations.

$$
\begin{array}{lcl}
q_0 & = & a_0b_0 \\
q_1 & = & a_1b_0 + a_0b_1 \\
q_2 & = & a_2b_0 + a_1b_1 + a_0b_2 + c_{1,1} \\
q_3 & = & a_3b_0 + a_2b_1 + a_1b_2 + c_{2,1} \\
q_4 & = & a_3b_1 + a_2b_2 + a_1b_3 + c_{3,1} + c_{2,2} \\
q_5 & = & a_3b_2 + a_2b_3 + c_{4,1} + c_{3,2} \\
q_6 & = & a_3b_3 + c_{5,1} + c_{4,2} \\
q_7 & = & a_3b_3 + c_{6,1} + c_{5,2}
\end{array}
$$

where $c_{i,j}$ is the j^{th} carry bit from calculating q_i.

The output of the multiplication is truncated to the four most significant bits. For the less significant bits, therefore, only the circuitry to calculate the carry bits was included. The resulting circuit is shown in figure A.17. The exclusive-or of the sign bits of each of the operands is the sign of the product. Note that, wherever it is beneficial, gates have been converted to NAND or NOR gates to reduce the transistor count.

A.3.4 Adder

The adder (as shown in figure A.18) uses the ripple-carry technique. It adds two 5-bit numbers, generating a 6-bit output (including the overflow bit). One of the operands is ANDed with a control line. When the control line is low one operand is added to zero, i.e. it passes through the adder unchanged. If the operands are in two's complement form they can be either positive or negative and the correct sum will be produced.

A.3.5 Two's Complementors

The chip inputs and outputs are in signed-binary magnitude form. To enable the adder to be used as a subtractor, and for it to be able to handle negative

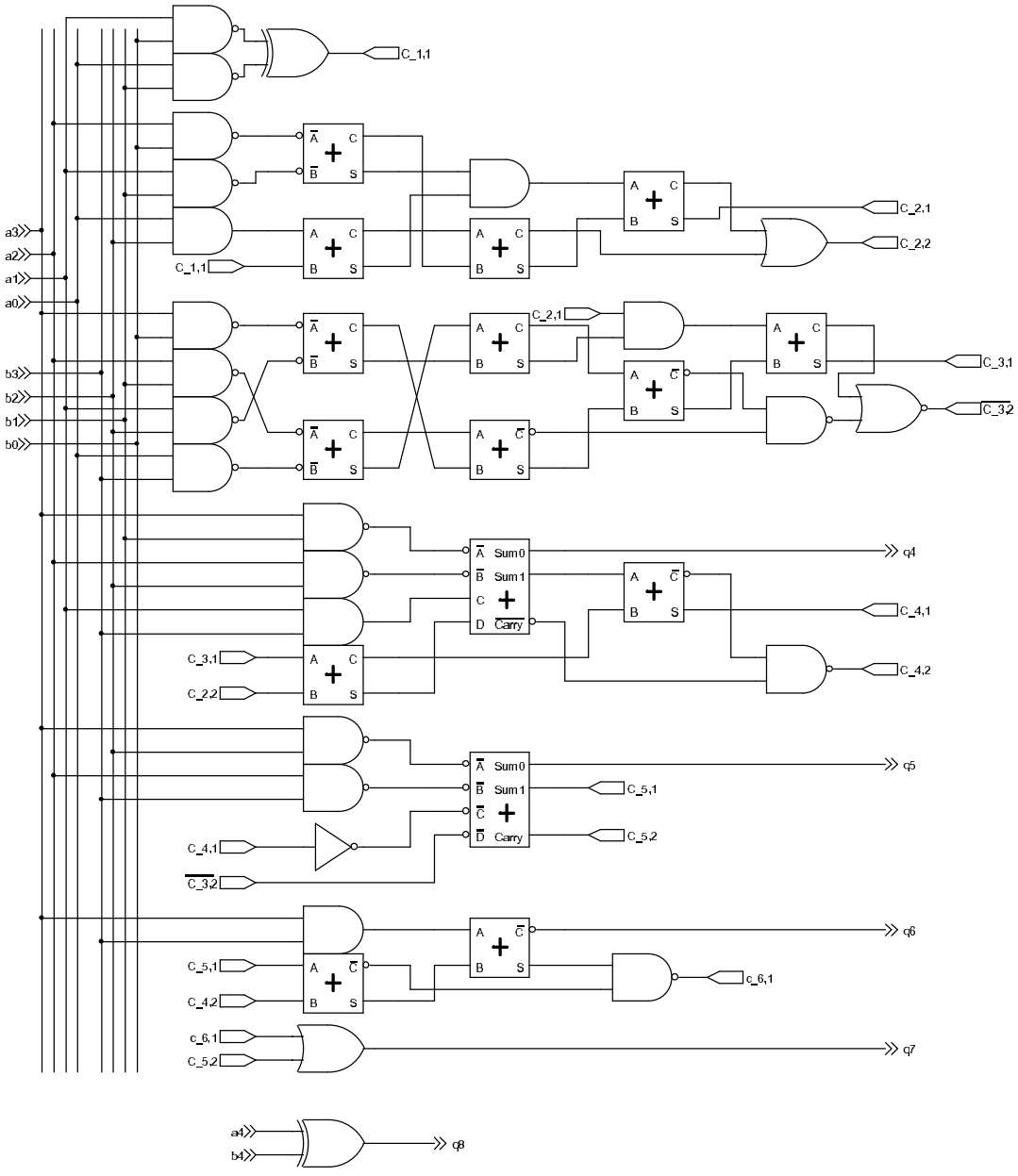

Figure A.17: **Multiplier (least significant bits truncated).**

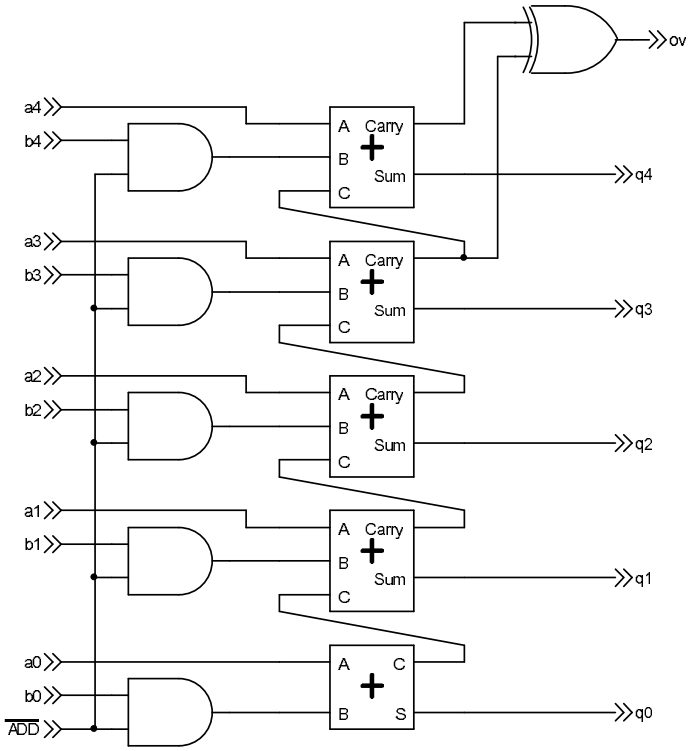

Figure A.18: **5-bit adder with disable-add control line.**

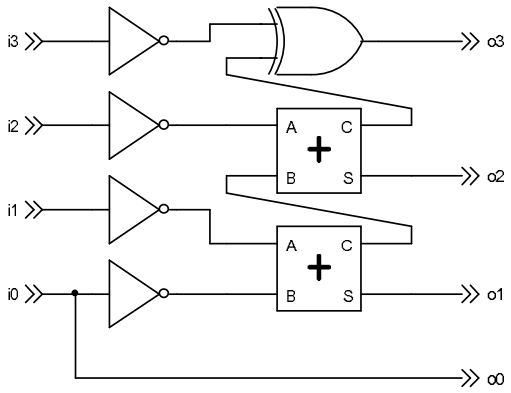

Figure A.19: **Two's complementor.**

numbers, the numbers must be in two's complement form. The multiplier requires its inputs to be in signed-binary magnitude form. Hence, the data must be converted between the two representations at various points in the calculation. Positive numbers are represented identically in the two methods. To convert a negative number from one representation to the other, the sign bit (that is the most significant bit) is unchanged and the other bits are replaced by their two's complement. The two's complement of a binary number is found by adding 1 to the one's complement of the number.

The circuit in figure A.19 produces the two's complement of its input.

Pre-adder Two's Complementor with Inverter

The pre-adder two's complementor converts a signed-binary magnitude number into its two's complement representation, and inverts the number if the INV control line is high. It works out the two's complement of the number and, using a multiplexer, decides whether to output the original number, or its two's complement. The decision is based on the sign of the output, which is the exclusive-or of the input sign and INV. Some circuitry had to be included to swap an input of negative 0 (10000_2) to positive 0 (00000_2). This was because 10000_2 would remain unchanged after going the two's complementor. $10000_{2\text{s cmp}} = -8_{10}$ — obviously an erroneous result. The logic

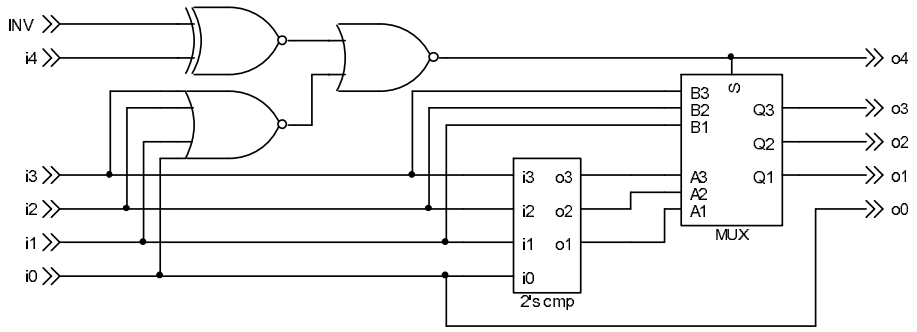

Figure A.20: **Inverter and signed-binary magnitude to two's complement converter.**

behind this circuit says that the output sign can only be negative if at least one of the input data lines is high AND the exclusive-or of the input sign and the INV line is high. The circuit is illustrated figure A.20.

Pre-multiplexer Two's Complementor

A block to convert the signed-binary output of the multiplier into two's-complement form for the addition is placed immediately before the multiplexer. It is identical to the circuit in figure A.20, except that it does not have the INV input and the XNOR is replaced with a NOT gate.

Pre-Multiplier / Output Two's Complementor

The two's complementor before the multiplier and the output latch converts data from a two's complement representation to the equivalent signed-binary magnitude representation. Again the output is the result of a selection between the original data and it's two's complement, depending on the sign bit. An output of negative 0 (which is really -8_{10}) is detected and reported as an overflow. This is shown in figure A.21.

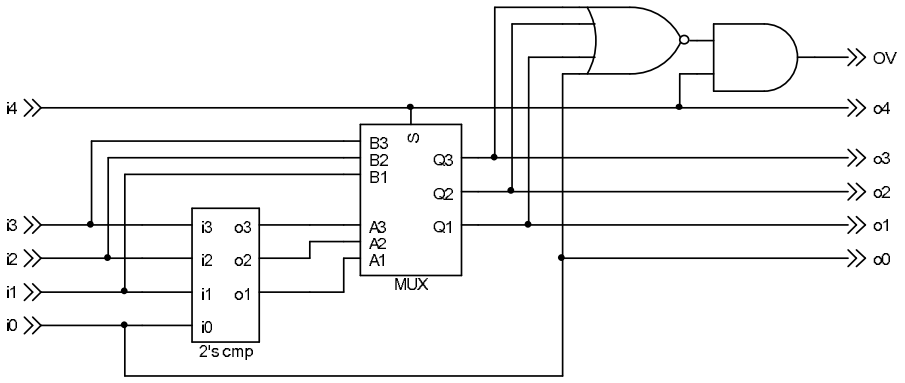

Figure A.21: **Two's complement to signed-binary magnitude converter.**

A.3.6 Registers

The registers in the ALU are cascades of D-latches. Three registers are used — one to store the a constant, one to keep intermediate results and one to hold the final output. These are explained in detail below.

a Register

The *a* register (figure A.22) is made up of five D-latches. The clock is normally suppressed by the notLDA control line. Only when notLDA drops lows can the clock propagate through to latch data into the register. There is no facility to clear the *a* register other than clocking in 0.

Sum Register

The sum register (figure A.23) is so named because its input is the output of the adder. It also is five bits long. It is similar to the *a* register in all ways except that it latches data in on every rising edge.

Output Latch and Register

The output register (figure A.24) keeps the result of the last calculation on the output pads for three clock periods — until the next result arrives. It

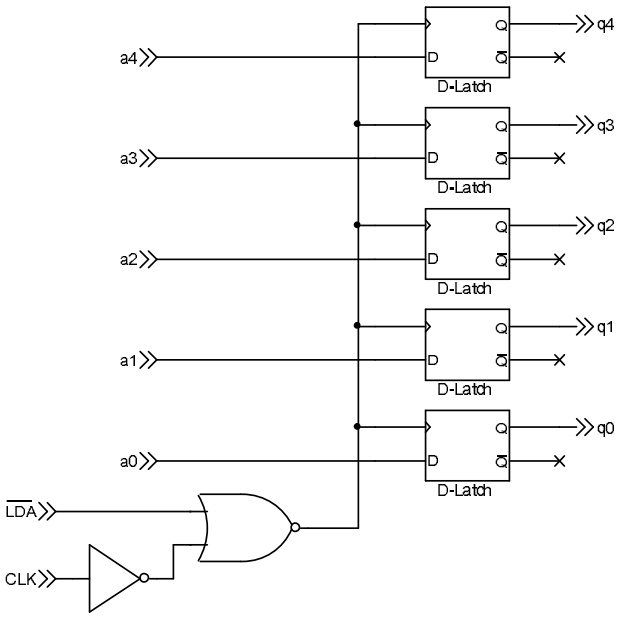

Figure A.22: *a* **register.**

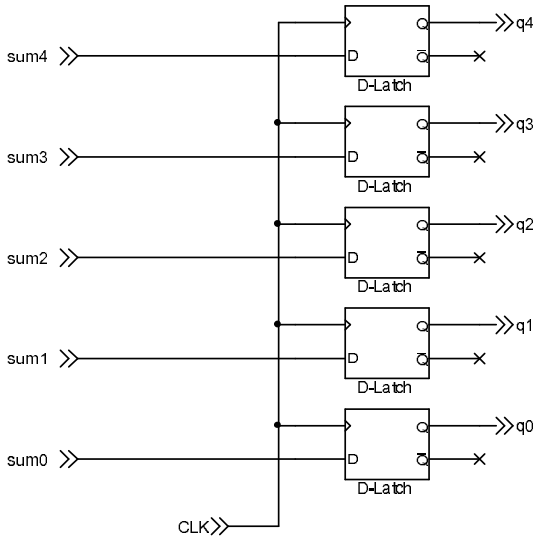

Figure A.23: **Sum register.**

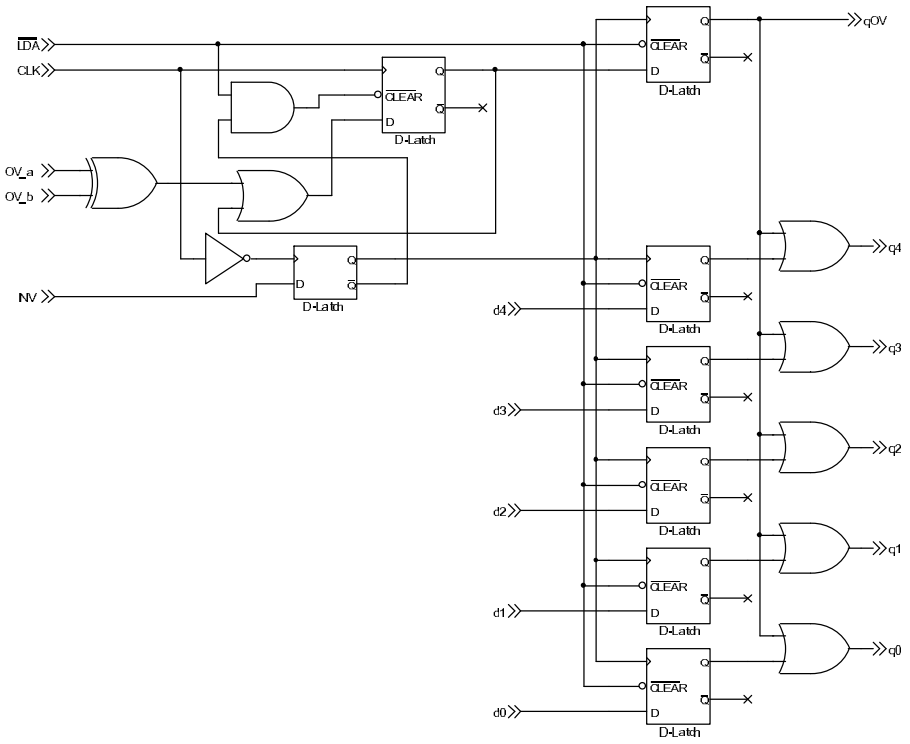

Figure A.24: **Output latch and register.**

clocks the data through on the falling edge of the clock during the forth stage of the calculation. The data is clocked on the falling edge to be allowed half a clock cycle for the signals to propagate from the sum register, through the two's complementor and to the output register. The data is latched through when the INV pin is high.

An intermediate latch is used to store the overflow. If an overflow occurs at any stage of the calculation, it will be stored in the intermediate latch and then latched out at the end of the cycle with the other data. This latch is cleared at the start of each cycle.

The output latch is reset when notLDA is low.

A.3.7 multiplexers

Adder multiplexer

The adder multiplexer chooses data from the input word (i.e. from the input multiplexer) when muxS is low, or the product of the multiplication (muxS high). It is five bits long and of the form shown in figure A.12 on page 140.

Input multiplexer

The input multiplexer is a five-bit, four-to-one multiplexer that selects the input word from one of the four sets of input pads. If it is enabled (MEN high), then it bases its decision as to which set of inputs to select on the current state of the control unit. Thus it presents the appropriate data for the current stage of the calculation. If it is disabled (MEN low) then it always sets the A input lines. In this case the data is loaded sequentially through the one set of input pads. Also if the notLDA control is low, the A inputs will be selected. The logic to translate the current state of the INV, ADD, MULT, MEN and notLDA control lines to the appropriate select control lines is given by the equations below, and expanded as a truth table in table A.10.

$$\begin{aligned}
\mathsf{A} &= \overline{\mathsf{MEN} \times \mathsf{notLDA}} \\
\mathsf{B} &= \overline{\mathsf{ADD}} \times \mathsf{INV} \times \mathsf{MULT} \times \mathsf{A} \\
\mathsf{C} &= \overline{\mathsf{ADD}} \times \mathsf{INV} \times \overline{\mathsf{MULT}} \times \mathsf{A} \\
\mathsf{D} &= \mathsf{ADD} \times \mathsf{INV} \times \overline{\mathsf{MULT}} \times \mathsf{A}
\end{aligned}$$

A.4 Experimental Set-Up

The SW ALU core is design to be included in a five-bit SW network circuit, as shown in figure A.25.

Table A.10: **Truth table for input multiplexer select logic.**

$\overline{\text{MEN} \times \text{notLDA}}$	ADD	INV	MULT	A	B	C	D	stage
0	0	0	0	0	0	0	0	illegal
0	0	0	1	0	0	0	0	illegal
0	0	1	0	0	0	1	0	1
0	0	1	1	0	1	0	0	2
0	1	0	0	0	0	0	0	illegal
0	1	0	1	0	0	0	0	illegal
0	1	1	0	0	0	0	1	3
0	1	1	1	0	0	0	0	illegal
1	0	0	0	1	0	0	0	load a
1	0	0	1	1	0	0	0	load a
1	0	1	0	1	0	0	0	load a
1	0	1	1	1	0	0	0	load a
1	1	0	0	1	0	0	0	load a
1	1	0	1	1	0	0	0	load a
1	1	1	0	1	0	0	0	load a
1	1	1	1	1	0	0	0	load a

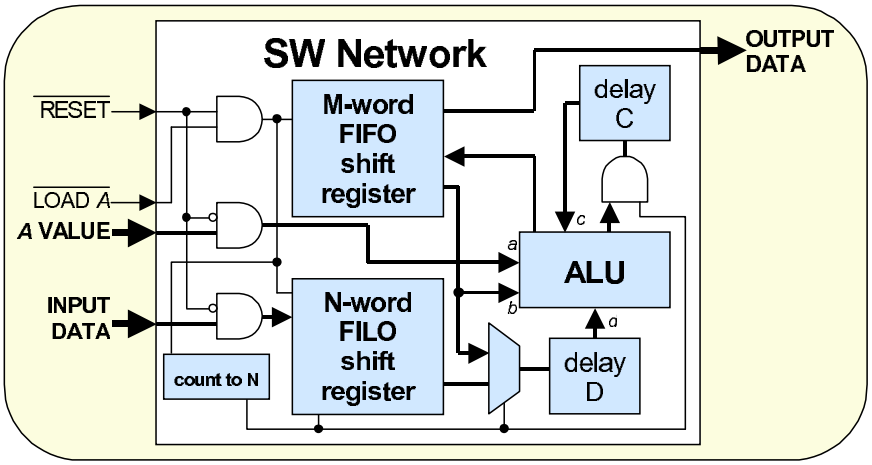

Figure A.25: **The ALU in place in a spectral warping network.** The ALU core forms part of an overall system capable of calculating spectral warping transforms. Compare this with figure 3.9 (page 87).

A.4.1 Elementary Testing

A very elementary test can be performed by holding the notLDA line high and applying random inputs and observing changes on the outputs. Also, it would be good to test that the outputs go to zero soon after notLDA is pulled low. This very simple test will at least show that signals are propagating through the circuit.

A.4.2 Functional Testing

Table A.11 provides some functional test patterns, which can be used to verify the chip's operation. These values can be programmed into a suitable chip tester. These patterns are based on Table A.15, found in section A.8 on page 173. Table A.11 also shows the expected values at certain nodes inside the circuit.

Further functional tests can be performed by emulating the circuit of figure A.25 with a PC, as illustrated if figure A.26.

A.5 Floorplanning of SW ALU Core

Table A.12 lists the input and output pins required for the SW ALU core (compare with table A.1 on 124). All the input pins can be shared with other cores on the same chip (they can be used for other purposes while the SW core is not in use). The output pins cannot be shared.

The ALU core floorplan is shown in figure A.27.

Table A.11: **Some test patterns for the ALU core.** For the latter part of this table, where all four input data buses are used, it is assumed that it takes three clock cycles for the inputs to settle once a value has been applied. Also, for the purposes of this table, it is assumed that the output values appear instantly once they have been clocked through the output register. Because of the clocking of the output register, the output values are delayed by an extra half clock cycle from what is indicated by this table.

in_men	in_notLDA	in_dA	in_dB	in_dC	in_dD	oreg_q	oreg_OV
0	0	00000	×	×	×	–	–
0	0	00100	×	×	×	00000	0
0	1	00011	×	×	×	00000	0
0	1	00111	×	×	×	00000	0
0	1	00010	×	×	×	00000	0
0	1	10110	×	×	×	00011	0
0	1	10001	×	×	×	00011	0
0	1	11010	×	×	×	00011	0
0	1	00000	×	×	×	11001	0
0	0	00011	×	×	×	11001	0
0	0	00100	×	×	×	00000	0
0	1	00101	×	×	×	00000	0
0	1	10010	×	×	×	00000	0
0	1	01111	×	×	×	00000	0
0	1	00000	×	×	×	01100	0
0	0	00000	×	×	×	01100	0
0	0	11001	×	×	×	00000	0
0	1	01100	×	×	×	00000	0
0	1	00001	×	×	×	00000	0
0	1	00010	×	×	×	00000	0
0	1	11101	×	×	×	01000	0
0	1	00011	×	×	×	01000	0
0	1	00000	×	×	×	01000	0
0	1	00110	×	×	×	01111	1
0	1	11001	×	×	×	01111	1
0	1	00011	×	×	×	01111	1
0	1	00001	×	×	×	01011	0
0	1	00101	×	×	×	01011	0
0	1	11011	×	×	×	01011	0
0	1	00000	×	×	×	11101	0
0	0	00000	×	×	×	11101	0

Table A.11 (cont): **Some test patterns for the ALU core.**

in_men	in_notLDA	in_dA	in_dB	in_dC	in_dD	oreg_q	oreg_OV
0	0	01111	×	×	×	00000	0
0	1	00001	×	×	×	00000	0
0	1	11111	×	×	×	00000	0
0	1	00110	×	×	×	00000	0
0	1	01001	×	×	×	01111	1
0	1	00110	×	×	×	01111	1
0	1	10001	×	×	×	01111	1
0	1	00000	×	×	×	10011	0
0	0	00000	×	×	×	10011	0
0	0	00000	×	×	×	00000	0
1	0	00100	×	×	×	–	–
1	0	00100	×	×	×	–	–
1	0	00100	×	×	×	–	–
1	1	00100	×	00011	×	00000	0
1	1	00100	00111	00011	×	00000	0
1	1	00100	00111	00011	00010	00000	0
1	1	×	00111	10110	00010	–	–
1	1	×	10001	10110	00010	–	–
1	1	×	10001	10110	11010	00011	0
1	1	×	10001	00000	11010	00011	0
1	1	×	×	00000	11010	00011	0
1	0	00000	×	00000	×	11001	0
1	0	00000	×	×	×	11001	0
1	0	00000	×	×	×	11001	0
1	0	01000	×	×	×	00000	0
1	0	01000	×	×	×	00000	0
1	0	01000	×	×	×	00000	0
1	1	01000	×	00100	×	00000	0
1	1	01000	10010	00100	×	00000	0
1	1	01000	10010	00100	01111	00000	0
1	1	×	10010	×	01111	–	–
1	1	×	×	×	01111	–	–
1	0	×	×	×	×	01100	0
×	0	×	×	×	×	01100	0
×	0	×	×	×	×	01100	0
×	0	×	×	×	×	00000	0

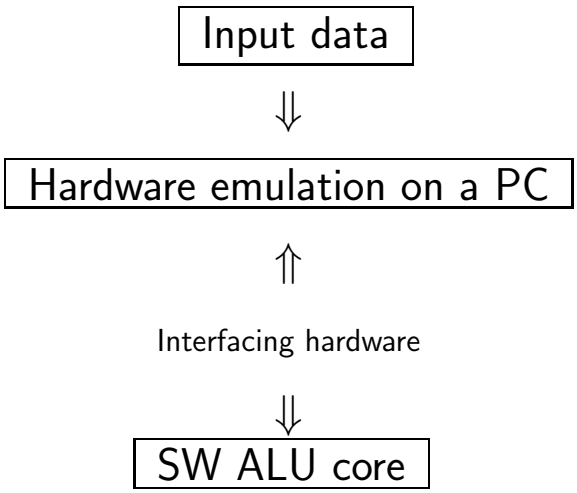

Figure A.26: **The basic concept of using a PC to perform functional tests.** Realistic data is generated and feed into a software programme that mimics the SW network, except for the ALU. The data for the ALU calculations and the resulting values are transmitted between the chip and the computer via custom-built interfacing hardware. The computer would control the clock signal is the set-up, so the ALU would be running at a much slower speed than what it could handle.

Table A.12: **List of I/O pins.**

Pin No.	Name	Type	Description
1	VDD1	Supply	1.8V logic supply
25	VSS1	Supply	0V logic supply
2	CLK	In logic	External clock signal
3	LDA	In logic	0–1.8V logic input to internal in_notLDA
4	MEN	In logic	0–1.8V logic input to internal in_men
5	A4	In logic	0–1.8V logic input to internal in_da4
6	A3	In logic	0–1.8V logic input to internal in_da3
7	A2	In logic	0–1.8V logic input to internal in_da2
8	A1	In logic	0–1.8V logic input to internal in_da1
9	A0	In logic	0–1.8V logic input to internal in_da0
10	B4	In logic	0–1.8V logic input to internal in_db4
11	B3	In logic	0–1.8V logic input to internal in_db3
12	B2	In logic	0–1.8V logic input to internal in_db2
13	B1	In logic	0–1.8V logic input to internal in_db1
14	B0	In logic	0–1.8V logic input to internal in_db0
15	C4	In logic	0–1.8V logic input to internal in_dc4
16	C3	In logic	0–1.8V logic input to internal in_dc3
17	C2	In logic	0–1.8V logic input to internal in_dc2
18	C1	In logic	0–1.8V logic input to internal in_dc1
19	C0	In logic	0–1.8V logic input to internal in_dc0
20	D4	In logic	0–1.8V logic input to internal in_dd4
21	D3	In logic	0–1.8V logic input to internal in_dd3
22	D2	In logic	0–1.8V logic input to internal in_dd2
23	D1	In logic	0–1.8V logic input to internal in_dd1
24	D0	In logic	0–1.8V logic input to internal in_dd0
26	OV	Out logic	0–1.8V output of internal oreg_OV
27	Q4	Out logic	0–1.8V output of internal oreg_q4
28	Q3	Out logic	0–1.8V output of internal oreg_q3
29	Q2	Out logic	0–1.8V output of internal oreg_q2
30	Q1	Out logic	0–1.8V output of internal oreg_q1
31	Q0	Out logic	0–1.8V output of internal oreg_q0

A.6 Schematic Diagrams of SW ASIC Components

Below is a list of the gate-level schematics used in the ALU. The numbers assigned to each gate refer to the position of the gate in the layout, from left to right.

1. Control unit: Figure A.28, page 161
2. Five-bit adder: Figure A.29, page 162
3. Pre-adder two's complementor with inverter: Figure A.30, page 163
4. Pre-multiplexer two's complementor: Figure A.32, page 164
5. *a* register: Figure A.33, page 165
6. Sum register: Figure A.34, page 166
7. Output register: Figure A.35, page 167
8. Output latch: Figure A.36, page 168
9. Select control for input multiplexer: Figure A.37, page 169

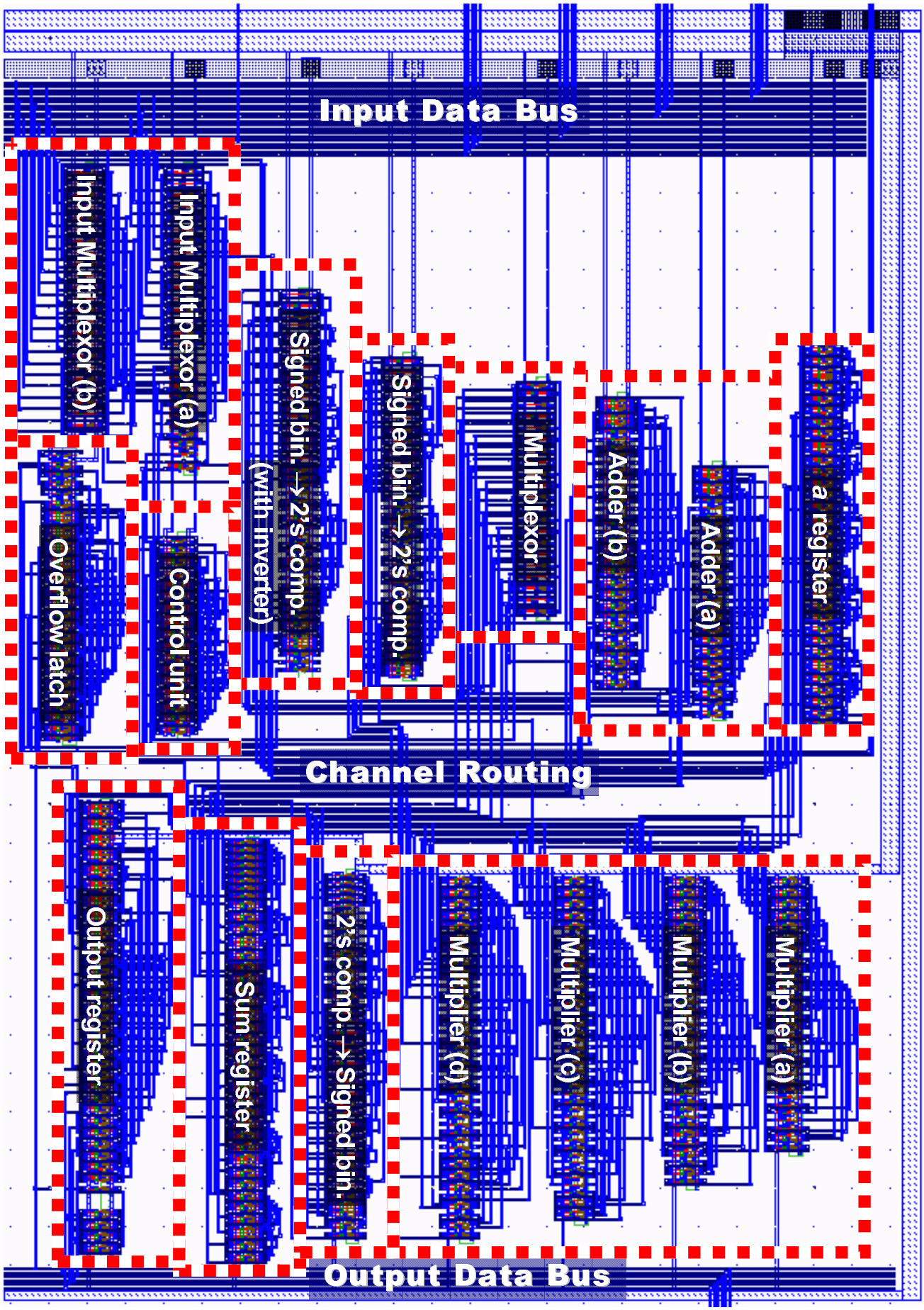

Figure A.27: **Layout diagram of the SW ALU ASIC.**

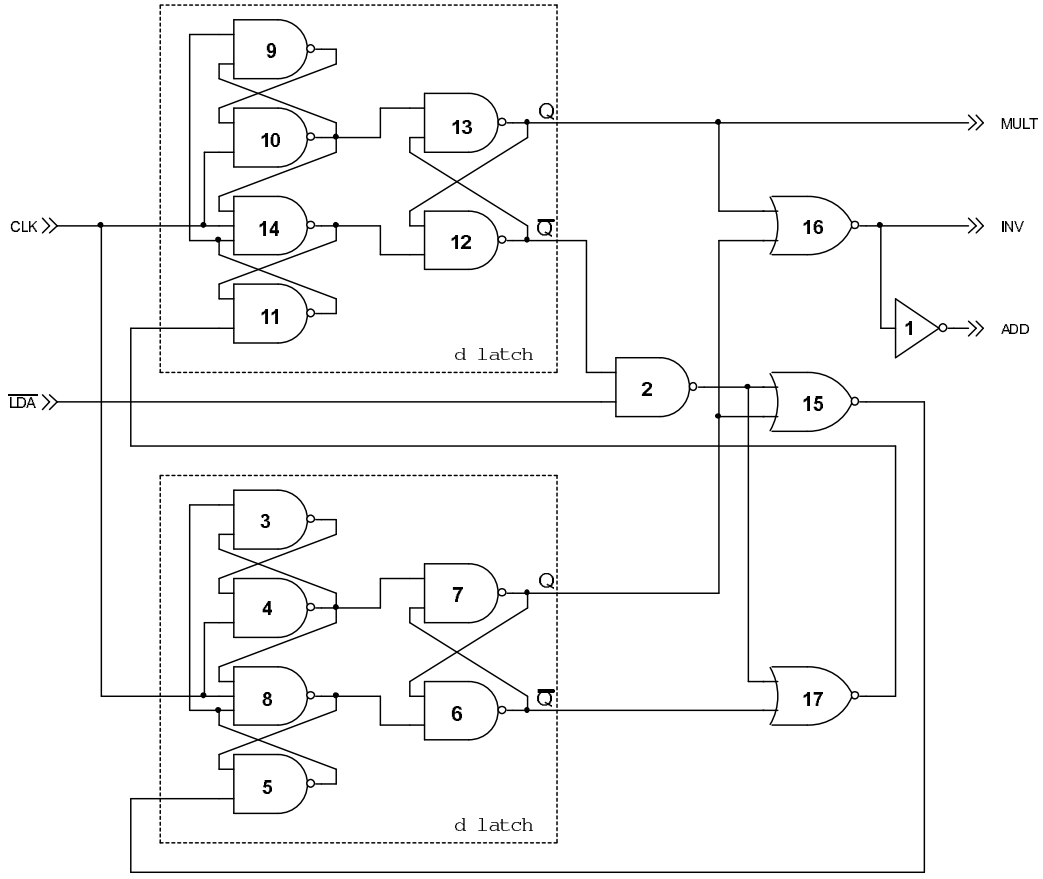

Figure A.28: **Control unit (expanded).**

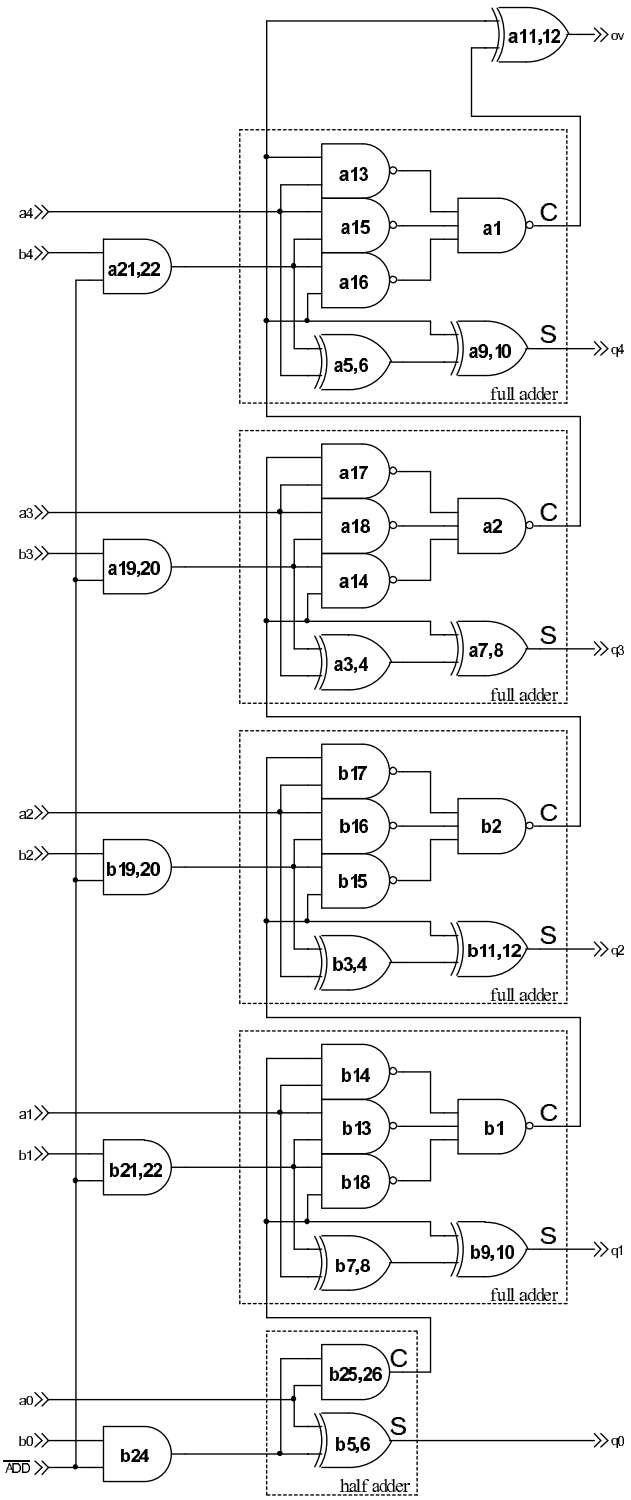

Figure A.29: **Five-bit adder (expanded).** *a* represents the top section and *b* represents the lower section.

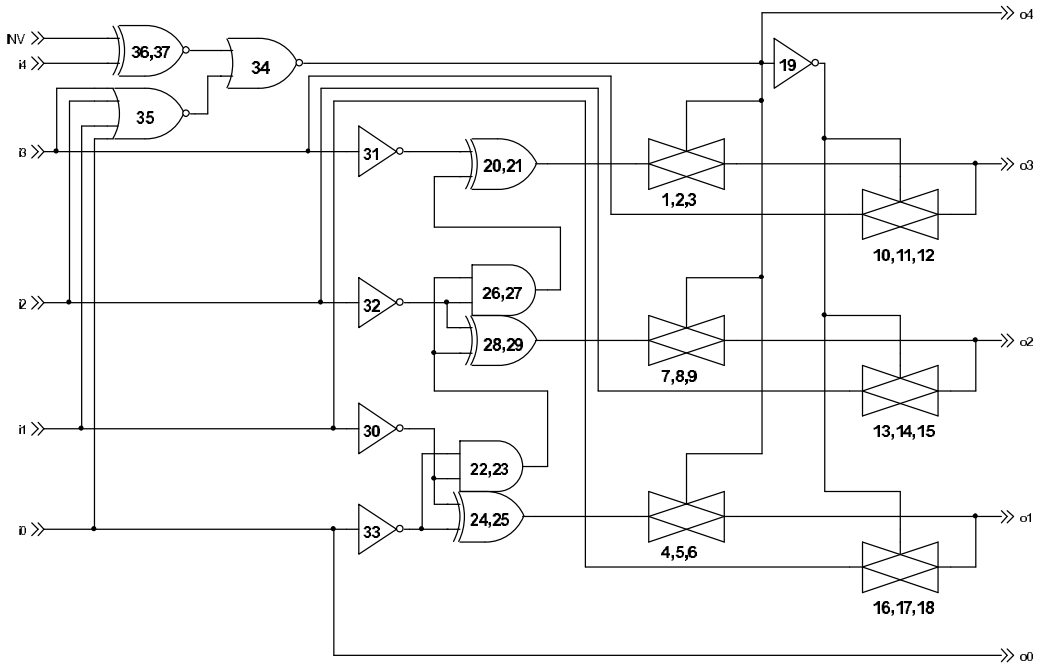

Figure A.30: **Signed binary to two's complement converter with inverter (expanded).** Note that the transmission gates can be implemented more efficiently if they are designed in the transistor lever (as in figure A.32). A slightly more optimal version (fewer gates) can be constructed with the two's complementor of figure A.31. This method would save five primitive gates (NAND, NOR and NOT gates are considered primitive, whereas XOR and XNOR gates are considered to be worth two primitives) — from 14 gates down to 9.

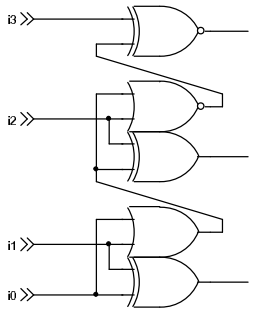

Figure A.31: **Optimised two's complementor.**

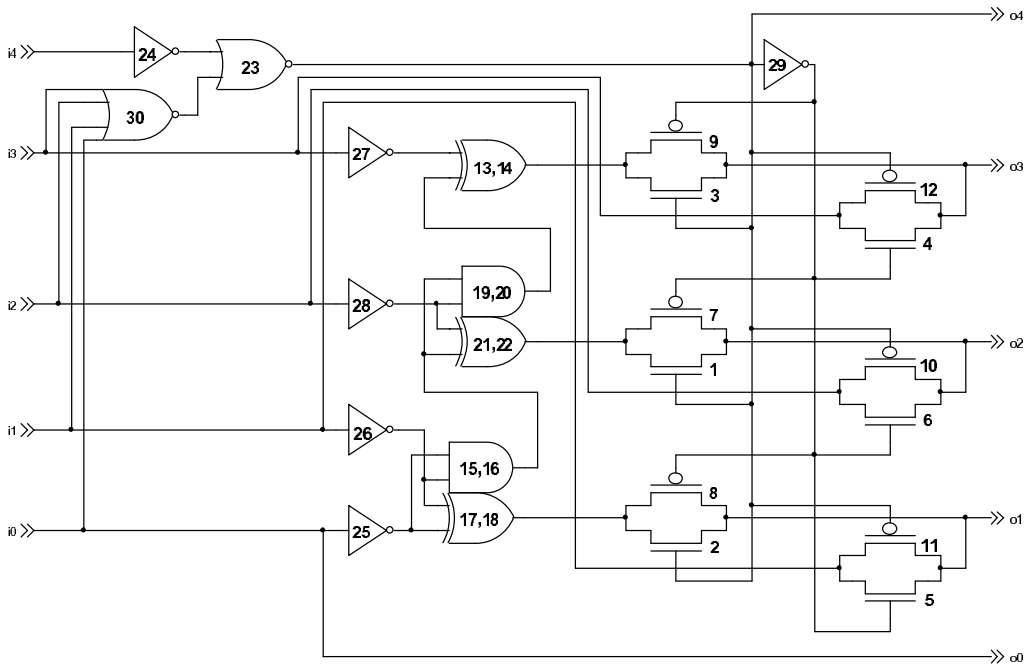

Figure A.32: **Signed binary to two's complement converter without inverter (expanded).**

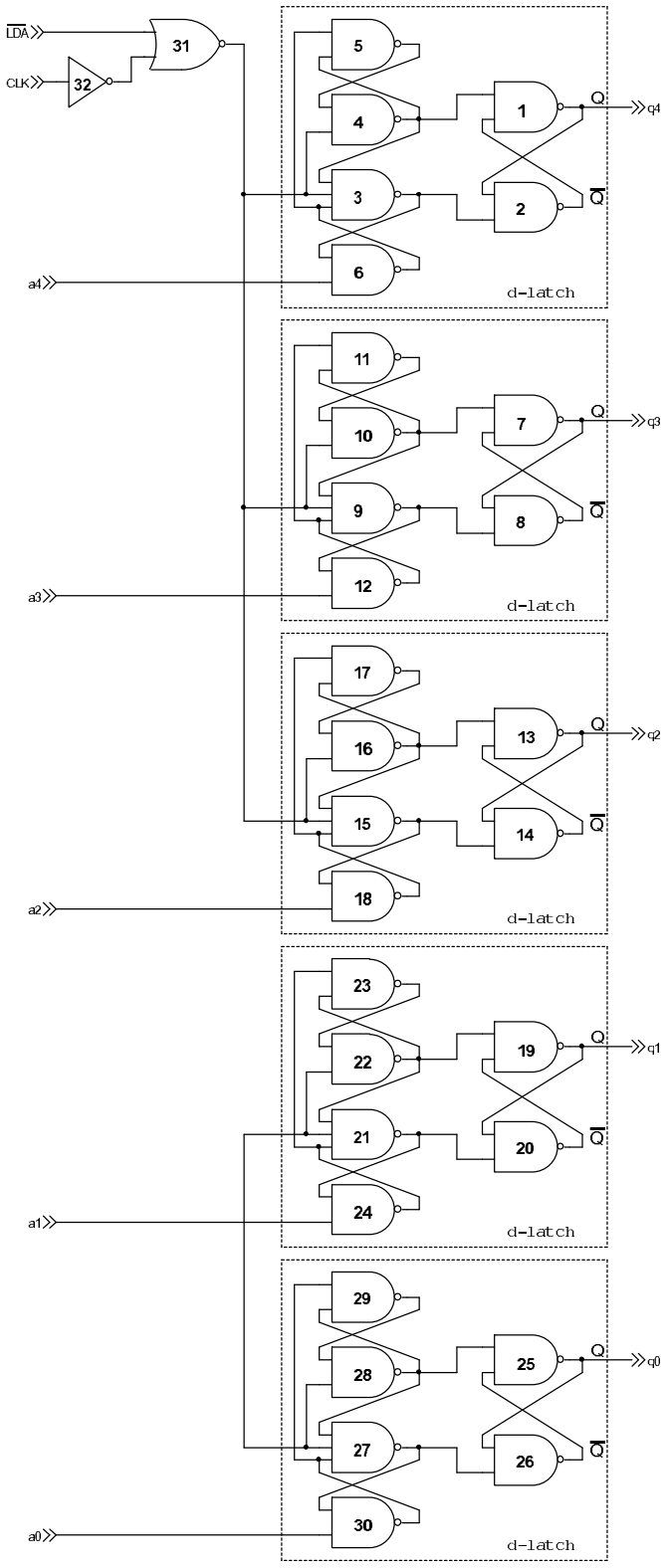

Figure A.33: *a* **register (expanded).**

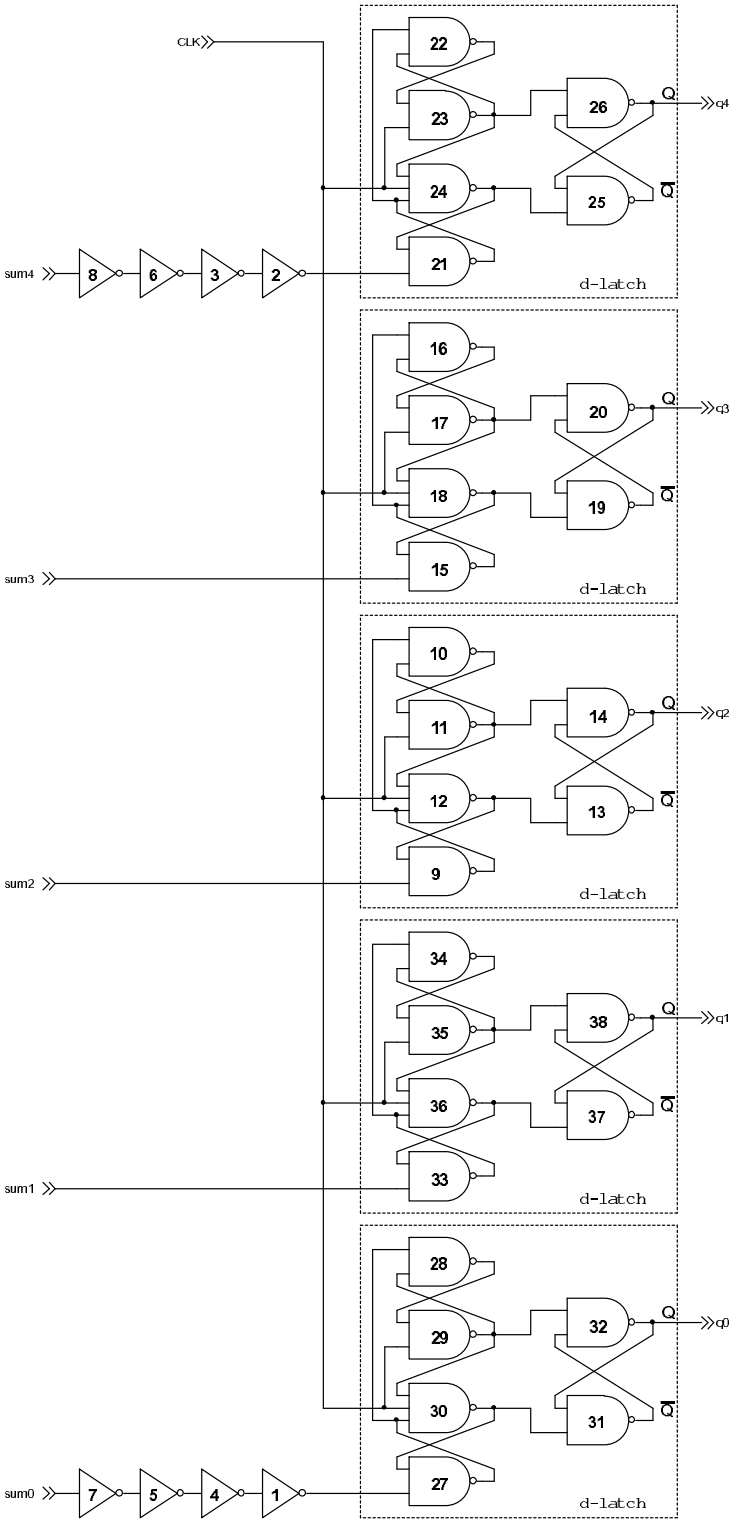

Figure A.34: **Sum register (expanded).** The inverters on the MSB and LSB inputs help to keep the propagation delay even.

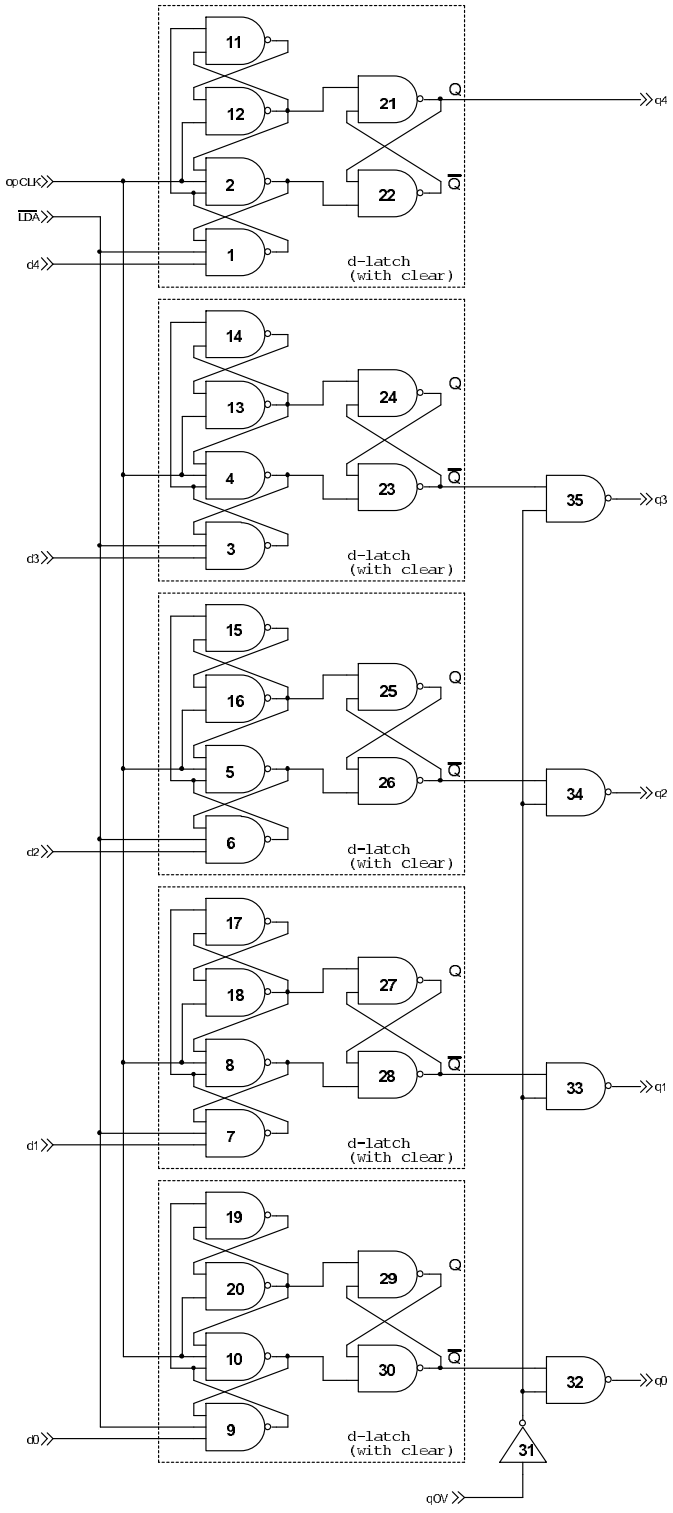

Figure A.35: **Output register (expanded).** If an overflow has occurred the outputs will be forced high (except for the sign bit).

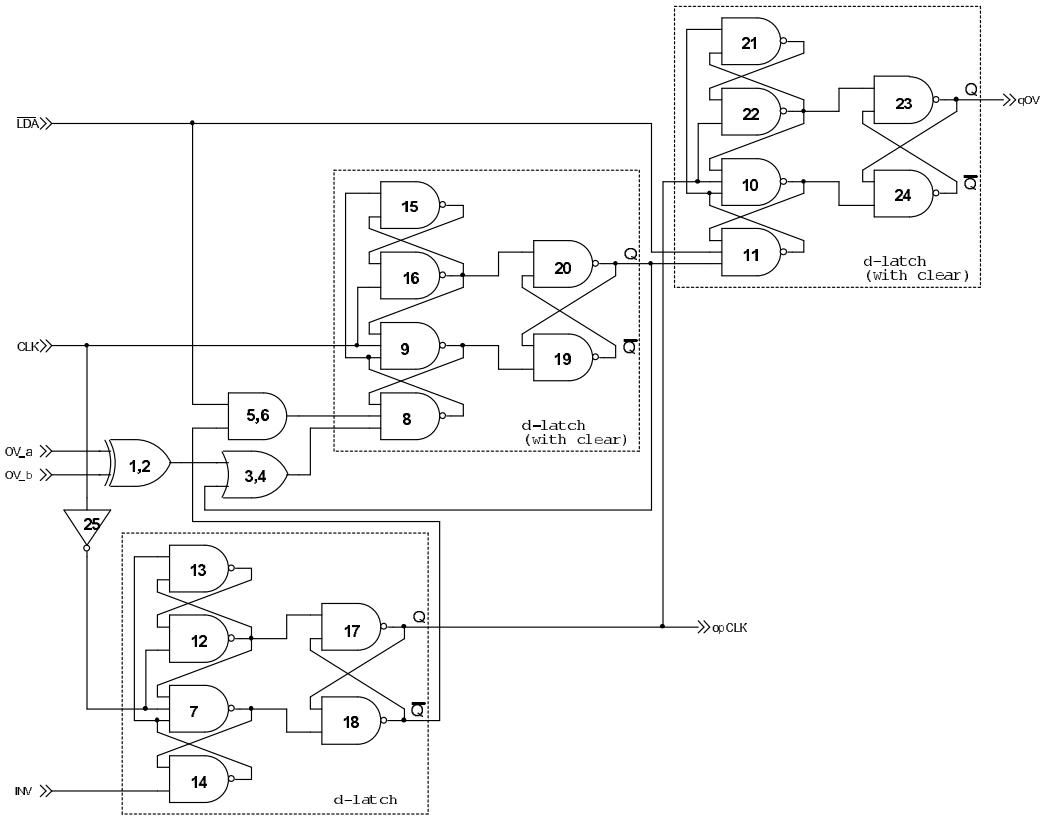

Figure A.36: **Output latch (expanded).**

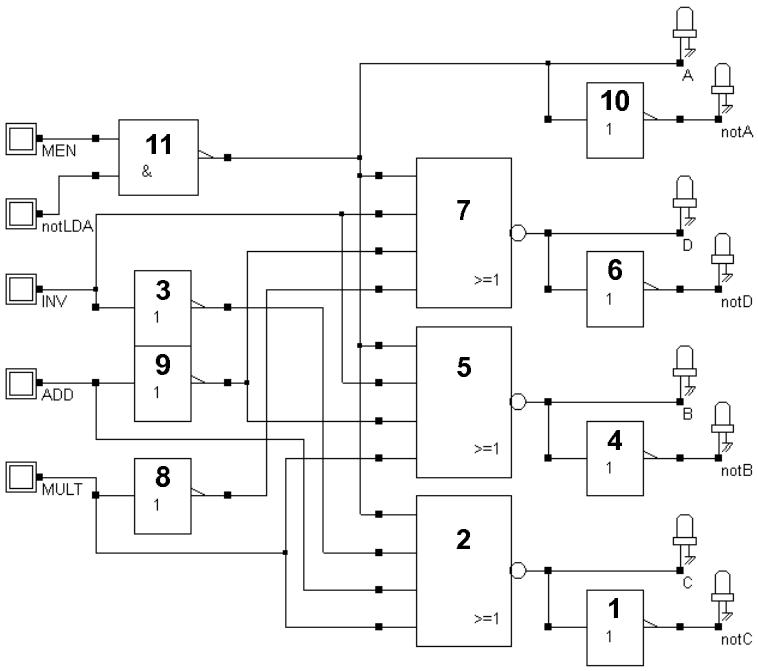

Figure A.37: **Select control logic for the input multiplexer.**

Table A.13: **Abbreviations used for each functional block for labels in the Microwind layout.**

Block	Abbreviation
Control unit	ctrl
Signed binary to two's complement converted with inverter (pre-adder two's complementor)	isb2c
Signed binary to two's complement converted without inverter (pre-multiplexer two's complementor)	sb2c
Two's complement to signed binary converted (pre-multiplier two's complementor)	2csb
multiplexer	mux
Adder	add
Multiplier	mult
a register	areg
Sum register	sreg
Output register	oreg
Overflow latch	ovfl

A.7 Naming Conventions Used in SW ASIC

A.7.1 Labels in Microwind

The labels in the layout files follow the convention of beginning with an abbreviation to indicate what block the label relates to, followed by an underscore and then a symbol to indicate the particular node. For example, 'ctrl_clk' denotes the clock line to the control unit. Table A.13 lists the block abbreviations.

The naming convention of table A.13 leads to nodes having several aliases. The output of one block is the input to one or more other blocks. Table A.14 lists all the aliases.

Table A.14: **Aliases for the labels in the Microwind layout.**

VDD (supply)	no alias				
VSS (ground)	no alias				
in_clk	ctrl_clk	ovfl_clk	sreg_clk		
in_notLDA	ctrl_notLDA	ovfl_notLDA	inmux_notLDA	oreg_notLDA	
ctrl_inv	isb2c_inv	ovfl_inv	inmux_inv		
ctrl_add	add_add		inmux_add		
ctrl_mult		muxS	inmux_mult		
indd0	no alias				
indd1	no alias				
indd2	no alias				
indd3	no alias				
indd4	no alias				
in_men	no alias				
inmux_q0	add_a0	areg_d0	isb2c_d4		
inmux_q1		areg_d1	isb2c_d0	isb2c_q0	
inmux_q2		areg_d2	isb2c_d1		
inmux_q3		areg_d3	isb2c_d2		
inmux_q4		areg_d4	isb2c_d3		
isb2c_q1	add_a1				
isb2c_q2	add_a2				
isb2c_q3	add_a3				
isb2c_q4	add_a4				
add_q0			sreq_d0		
add_q1			sreq_d1		
add_q2			sreq_d2		
add_q3			sreq_d3		
add_q4			sreq_d4		
add_ov	ovfl_ovadd				
sreg_q0	2csb_d0	mux_b0	mult_b0	oreg_d0	2csb_q0
sreg_q1	2csb_d1	mux_b1			
sreg_q2	2csb_d2	mux_b2			
sreg_q3	2csb_d3	mux_b3			
sreg_q4	2csb_d4	mux_b4	mult_b4	oreg_d4	2csb_q4
2csb_q1			mult_b1	oreg_d1	
2csb_q2			mult_b2	oreg_d2	
2csb_q3			mult_b3	oreg_d3	
2csb_ov	ovfl_ovmult				

Table A.14 (cont): **Aliases for the labels in the Microwind layout.**

mult_q0	sb2c_d0	mux_a0			sb2c_q0
mult_q1	sb2c_d1				
mult_q2	sb2c_d3				
mult_q3	sb2c_d3				
mult_q4	sb2c_d4				
sb2c_q1		mux_a1			
sb2c_q2		mux_a2			
sb2c_q3		mux_a3			
sb2c_q4		mux_a4			
mux_q0	add_b0				
mux_q1	add_b1				
mux_q2	add_b2				
mux_q3	add_b3				
mux_q4	add_b4				
areg_q0			mult_a0		
areq_q1			mult_a1		
areg_q2			mult_a2		
areg_q3			mult_a3		
areg_q4			mult_a4		
ovfl_qov				oreg_OV	
ovfl_ovclk				oreg_ovclk	
oreg_q0	no alias				
oreg_q1	no alias				
oreg_q2	no alias				
oreg_q3	no alias				
oreg_q4	no alias				

A.8 Simulation Tests

A.8.1 Microwind Tests

Table A.15 presents a set of input data, with the expected outputs (ovfl and oreg) and intermediate values. The following Microwind simulation traces show that the simulated results agree with the predicted results.

The figures showing the results of simulation test using Microwind are listed below.

1. Output pads:
   (a) Figure A.38, page 177
   (b) Figure A.39, page 178
   (c) Figure A.40, page 179
   (d) Figure A.41, page 180
2. Control unit: Figure A.42, page 181
3. Signed binary to two's complement converter with inverter: Figure A.43, page 182
4. Signed binary to two's complement converter without inverter: Figure A.44, page 183
5. Two's complement to signed binary converter: Figure A.45, page 184
6. multiplexer: Figure A.46, page 185
7. Adder: Figure A.47, page 186
8. Multiplier: Figure A.48, page 187
9. *a* register: Figure A.49, page 188
10. Sum register: Figure A.50, page 189

Table A.15: **Test data and expected results.** The left-hand columns are the inputs, the central columns are the internal nodes and the right-hand columns are the outputs. The output register oreg and latch ovfl are delayed by half a clock period.

notLDA	IN	ctrl_INV ctrl_ADD ctrl_MULT	areg	isb2c	add	sreg	2csb	mult	sb2c	mux	oreg	ovfl
0	00000	× × ×	×	00000	00000	×	×	×	×	×	–	–
	0 (a)			0	0							
0	00100	1 0 0	00000	11100	11100	00000	00000	00000	00000	00000	00000	0
	4 (a)		0	-4	-4	0	0	0	0	0	0	
1	00011	1 0 0	00100	11101	11101	11100	10100	10001	11111	11100	00000	0
	3 (c)		1/4	-3	-3	-4	-4	-1	-1	-4	0	
1	00111	0 1 0	00100	00111	00100	11101	10011	10000	00000	11101	00000	0
	7 (b)		1/4	7	4	-3	-3	-0	0	-3	0	
1	00010	0 1 1	00100	00010	00011	00100	00100	00001	00001	00001	00000	0
	2 (d)		1/4	2	3	4	4	1	1	1	0	
1	10110	1 0 0	00100	00110	00110	00011	00011	00000	00000	00011	00011	0
	-6 (c)		1/4	6	6	3	3	0	0	3	3	
1	10001	0 1 0	00100	10001	00101	00110	00110	00001	00001	00110	00011	0
	-1 (b)		1/4	-1	5	6	6	1	1	6	3	
1	11010	0 1 1	00100	11010	10111	00101	00101	00001	00001	00001	00011	0
	-10(d)		1/4	-10	-9	5	5	1	1	1	3	
1	00000	1 0 0	00100	00000	00000	10111	11001	10001	11111	10111	11001	0
	0 (c)		1/4	0	0	-9	-9	-1	-1	-9	-9	
0	00000	0 1 0	00000	00000	00000	00000	00000	00000	00000	00000	11001	0
	0		0	0	0	0	0	0	0	0	-9	
0	01000	1 0 0	00000	11000	11000	00000	00000	00000	00000	00000	00000	0
	8 (a)		0	-8	-8	0	0	0	0	0	0	
1	00100	1 0 0	01000	11100	11100	11000	11000	10100	11100	11000	00000	0
	4 (c)		1/2	-4	-4	-8	-8	-4	-4	-8	0	
1	10010	0 1 0	01000	11110	11010	11100	10100	10010	11110	11100	00000	0
	-2 (b)		1/2	-2	-6	-4	-4	-2	-2	-4	0	
1	01111	0 1 1	01000	01111	01100	11010	10110	10011	11101	11101	00000	0
	15 (d)		1/2	15	12	-6	-6	-3	-3	-3	0	
1	00000	1 0 0	01000	00000	00000	01100	01100	00110	00110	01100	01100	0
	0 (c)		1/2	0	0	12	12	6	6	12	12	
0	00000	0 1 0	01000	00000	00000	00000	00000	00000	00000	00000	01100	0
	0 (c)		1/2	0	0	0	0	0	0	0	12	

Table A.15 (cont): **Test data and expected results.**

notLDA	IN	ctrl_INV ctrl_ADD ctrl_MULT	areg	isb2c	add	sreg	2csb	mult	sb2c	mux	oreg	ovfl
0	11001	1 0 0	00000	01001	01001	00000	00000	00000	00000	00000	00000	0
	-9 (a)		0	9	9	0	0	0	0	0	0	
1	01100	1 0 0	11001	10100	10100	01001	01001	10101	11011	01001	00000	0
	12 (a)		-9/16	-12	-12	9	9	-5	-5	9	0	
1	00001	0 1 0	11001	00001	10101	10100	11100	00110	00110	10100	00000	0
	1 (b)		-9/16	1	-11	-12	-12	6	6	-12	0	
1	00010	0 1 1	11001	00010	01000	10101	11011	00110	00110	00110	00000	0
	2 (d)		-9/16	2	8	-11	-11	6	6	6	0	
1	11101	1 0 0	11001	01101	01101	01000	01000	10100	11100	01000	01000	0
	-13(c)		-9/16	13	13	8	8	-4	-4	8	8	
1	00011	0 1 0	11001	00011	00000	01101	01101	10111	11000	01101	01000	0
	3 (b)		-9/16	3	ovfl	13	13	-7	-7	13	8	
1	00000	0 1 1	11001	00000	00000	00000	00000	10000	00000	00000	01000	0
	0 (d)		-9/16	0	0	0	0	-0	0	0	8	
1	00110	1 0 0	11001	11010	11010	00000	00000	10000	00000	00000	01111	1
	6 (c)		-9/16	-6	-6	0	0	-0	0	0	overflow	
1	11001	0 1 0	11001	10111	10001	11010	10110	00011	00011	11010	01111	1
	-9 (b)		-9/16	-9	-15	-6	-6	3	3	-6	overflow	
1	00011	0 1 1	11001	00011	01011	10001	11111	01000	01000	01000	01111	1
	3 (d)		-9/16	3	11	-15	-15	8	8	8	overflow	
1	00001	1 0 0	11001	11111	11111	01011	01011	10110	11010	01011	01011	0
	1 (c)		-9/16	-1	-1	11	11	-6	-6	11	11	
1	00101	0 1 0	11001	00101	00100	11111	10001	00000	00000	11111	01011	0
	5 (b)		-9/16	5	4	-1	-1	0	0	-1	11	
1	11011	0 1 1	11001	11011	10011	00100	00100	10010	11110	11110	01011	0
	-11(d)		-9/16	-11	-13	4	4	-2	-2	-2	11	
1	00000	1 0 0	11001	00000	00000	10011	11101	00111	00111	10011	11101	0
	0 (c)		-9/16	0	0	-13	-13	7	7	-13	-13	
0	00000	0 1 0	11001	00000	00000	00000	00000	10000	00000	00000	11101	0
	0 (a)		-9/16	0	0	0	0	-0	0	0	-13	
0	01111	1 0 0	00000	10001	10001	00000	00000	00000	00000	00000	00000	0
	15 (a)		0	-15	-15	0	0	0	0	0	0	
1	00001	1 0 0	01111	11111	11111	10001	11111	11110	10010	10001	00000	0
	1 (c)		15/16	-1	-1	-15	-15	-14	-14	-15	0	
1	11111	0 1 0	01111	10001	10000	11111	10001	10000	00000	11111	00000	0
	-15(b)		15/16	-15	-16	-1	-1	-0	0	-1	0	

Table A.15 (cont): **Test data and expected results.**

notLDA	IN	ctrl_INV ctrl_ADD ctrl_MULT	areg	isb2c	add	sreg	2csb	mult	sb2c	mux	oreg	ovfl
1	00110	0 1 1	01111	00110	00110	10000	10000	10000	00000	00000	00000	0
	6 (d)		15/16	6	6	-16	ovfl	-0	0	0	0	
1	01001	1 0 0	01111	10111	10111	00110	00110	00101	00101	00110	01111	1
	9 (c)		15/16	-9	-9	6	6	5	5	6	overflow	
1	00110	0 1 0	01111	00110	11101	10111	11001	11000	11000	10111	01111	1
	6 (b)		15/16	6	-3	-9	-9	-8	-8	-9	overflow	
1	10001	0 1 1	01111	11111	11101	11101	10011	10010	11110	11110	01111	1
	-1 (d)		15/16	-1	-3	-3	-3	-2	-2	-2	overflow	
1	00000	1 0 0	01111	00000	00000	11101	10011	10010	11110	11101	10011	0
	0		15/16	0	0	-3	-3	-2	-2	-3	-3	
0	00000	0 1 0	01111	00000	00000	00000	00000	00000	00000	00000	10011	0
	0		15/16	0	0	0	0	0	0	0	-3	

11. Output register: Figure A.51, page 190

12. Overflow latch: Figure A.52, page 191

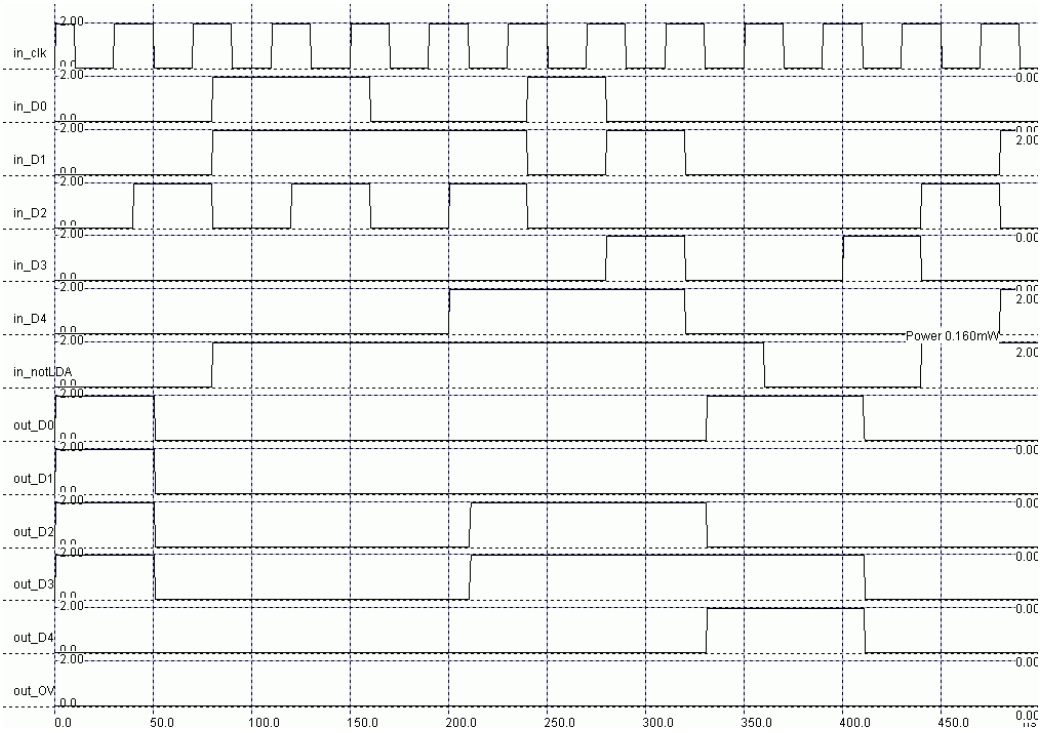

Figure A.38: **Simulation trace of inputs and outputs (a).**

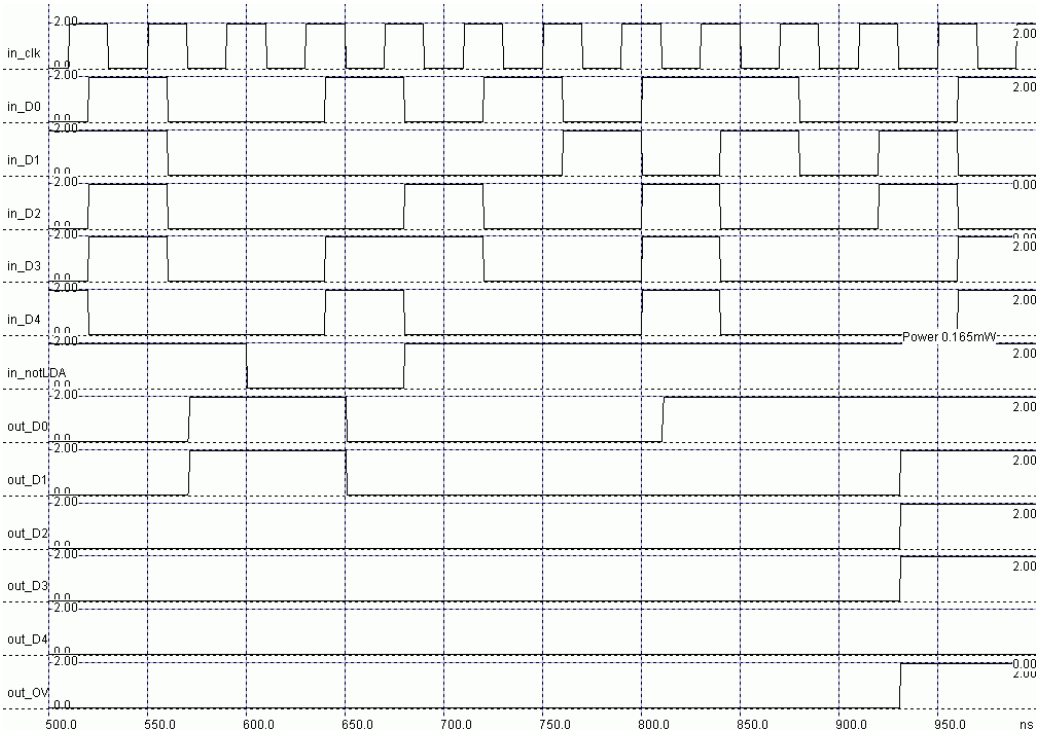

Figure A.39: **Simulation trace of inputs and outputs (b).**

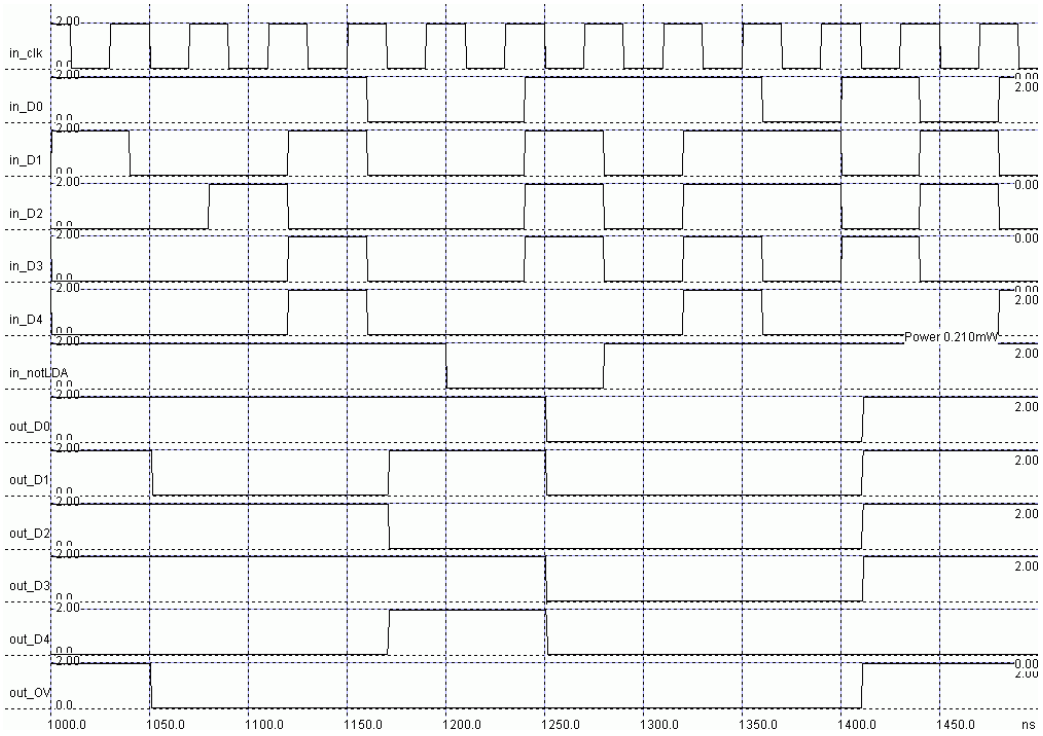

Figure A.40: **Simulation trace of inputs and outputs (c).**

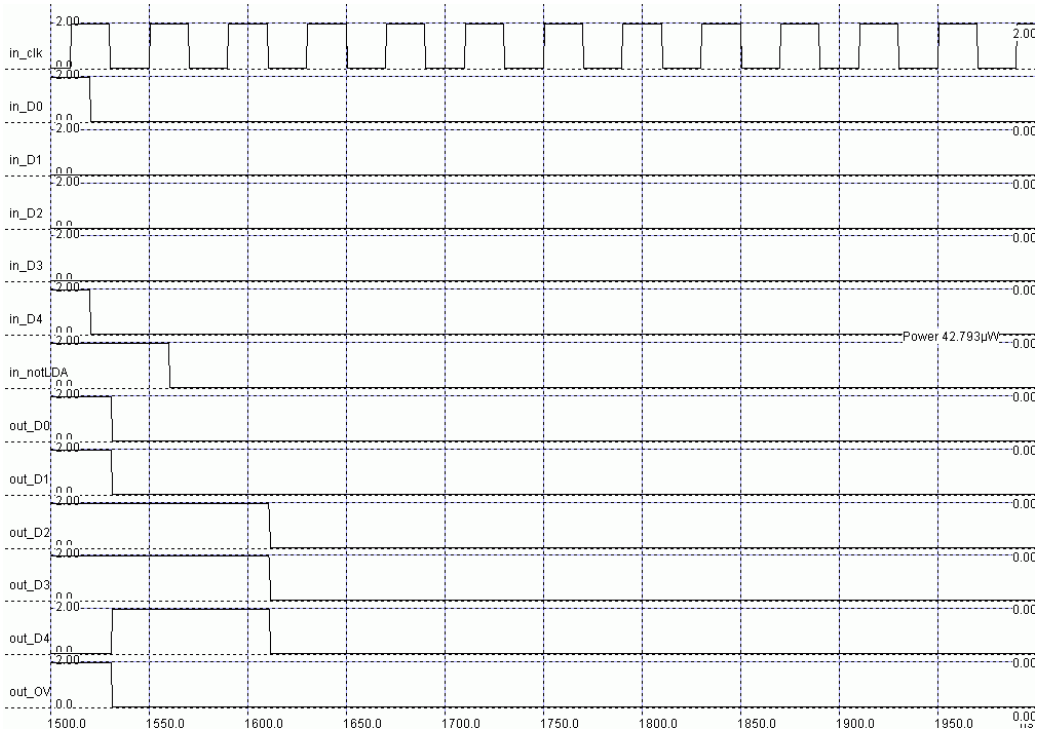

Figure A.41: **Simulation trace of inputs and outputs (d).**

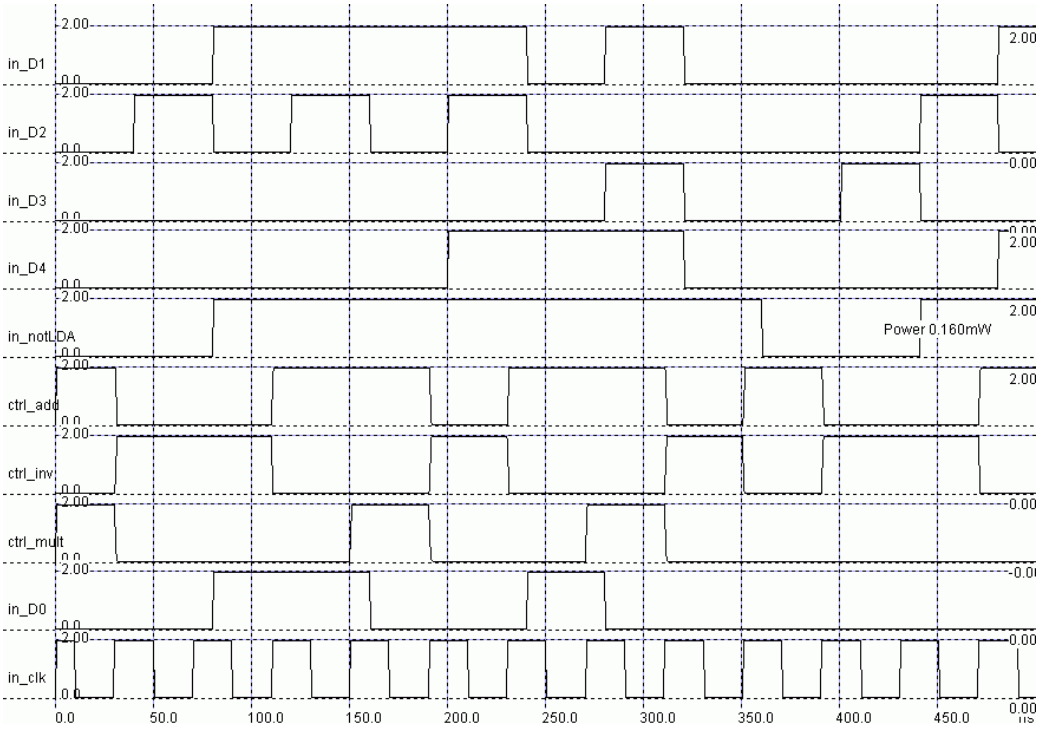

Figure A.42: **Simulation trace of control unit outputs.**

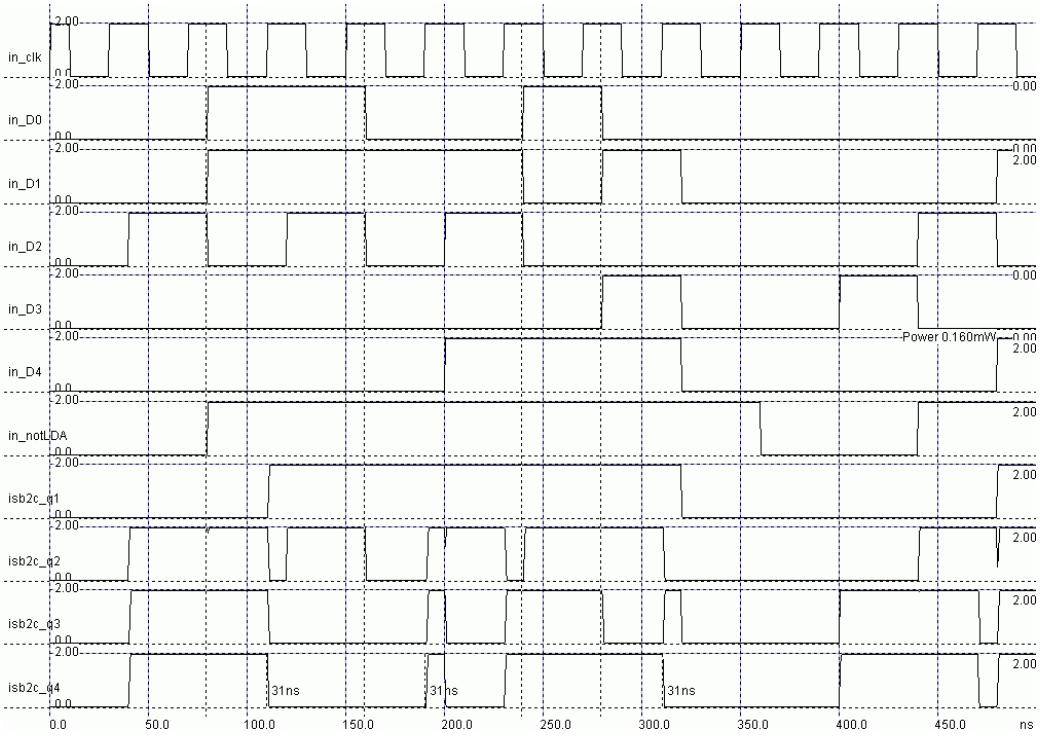

Figure A.43: **Simulation trace of pre-adder two's complementor outputs.**

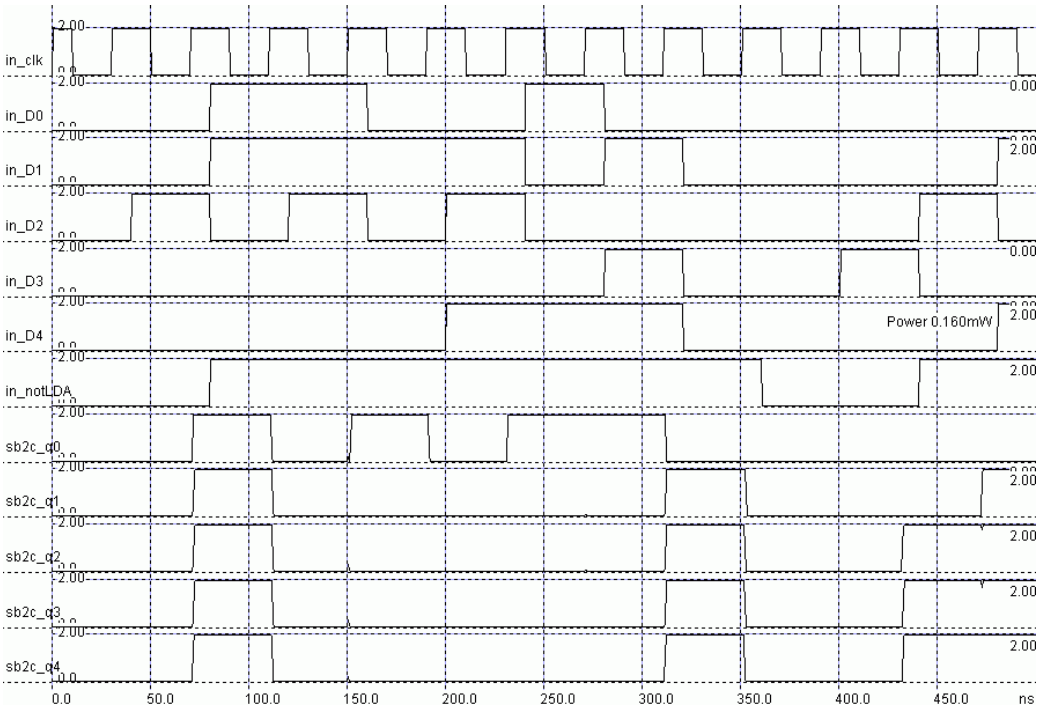

Figure A.44: **Simulation trace of pre-multiplexer two's complementor outputs.**

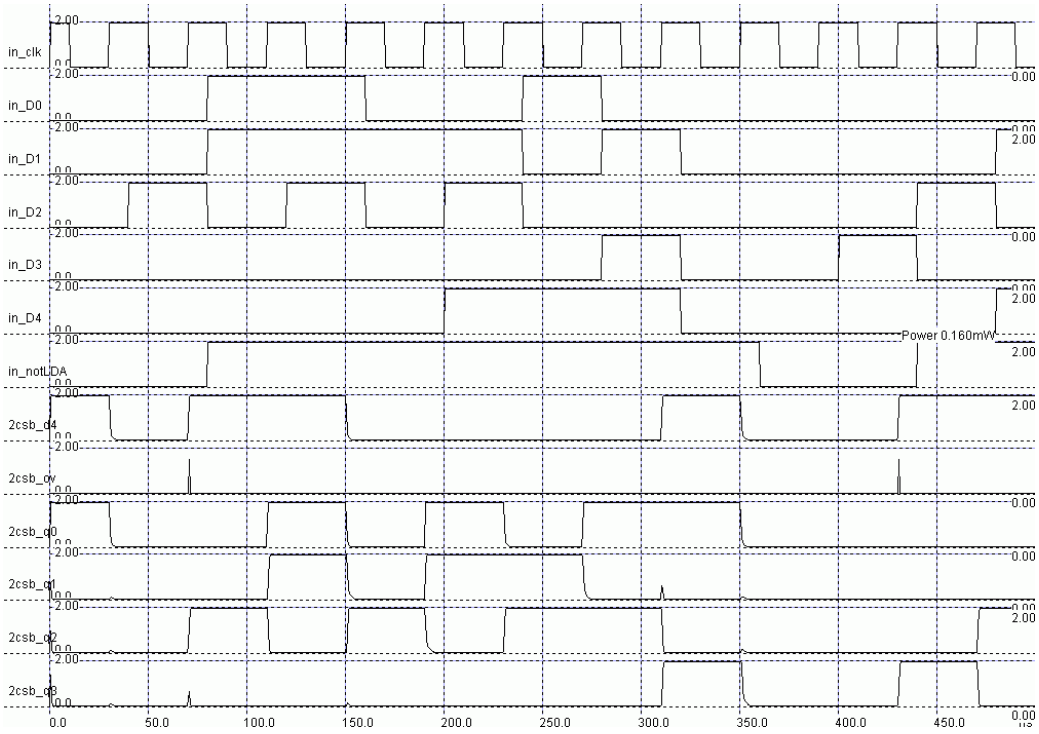

Figure A.45: **Simulation trace of pre-multiplier two's complementor outputs.**

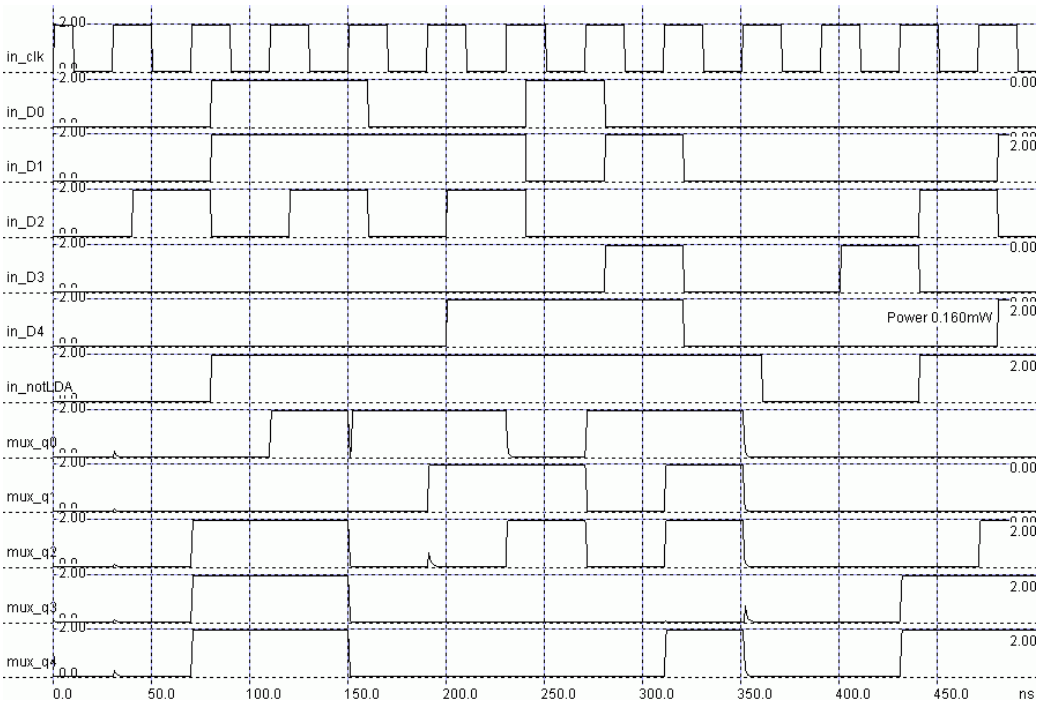

Figure A.46: **Simulation trace of multiplexer outputs.**

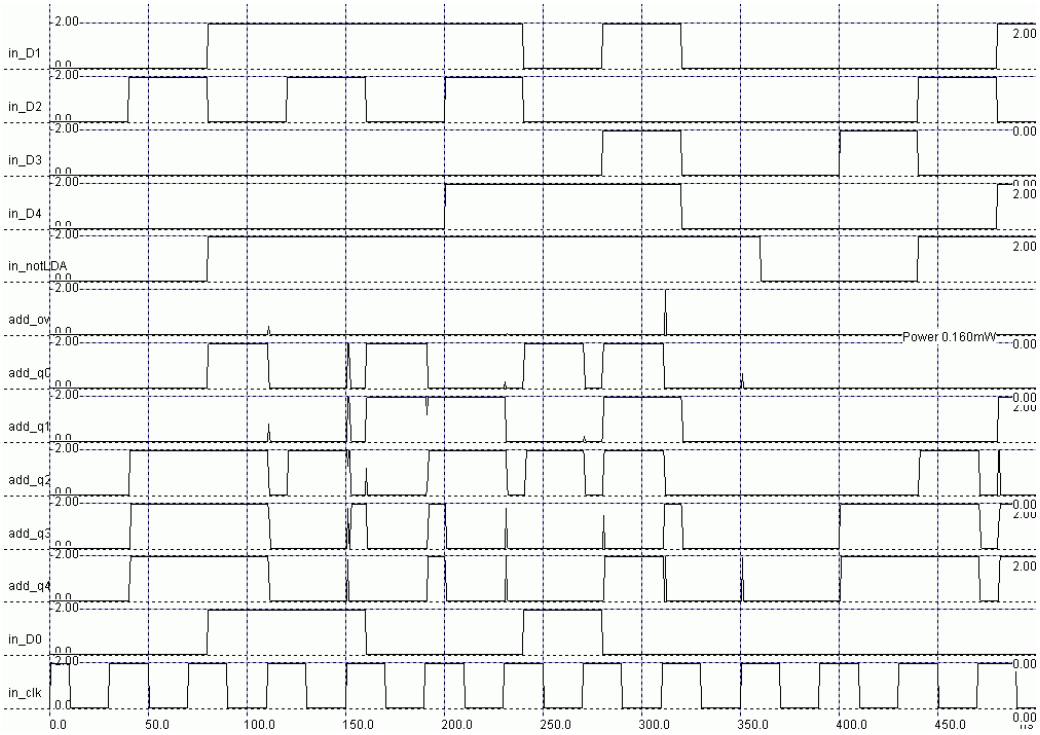

Figure A.47: **Simulation trace of adder outputs.**

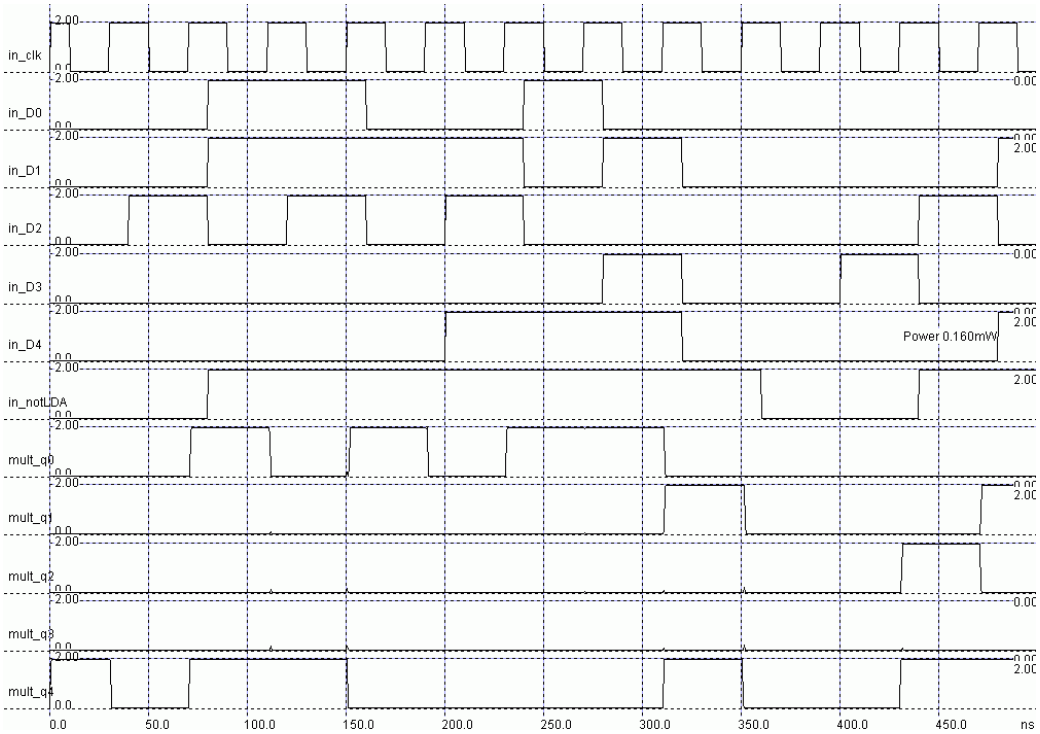

Figure A.48: **Simulation trace of multiplier outputs.**

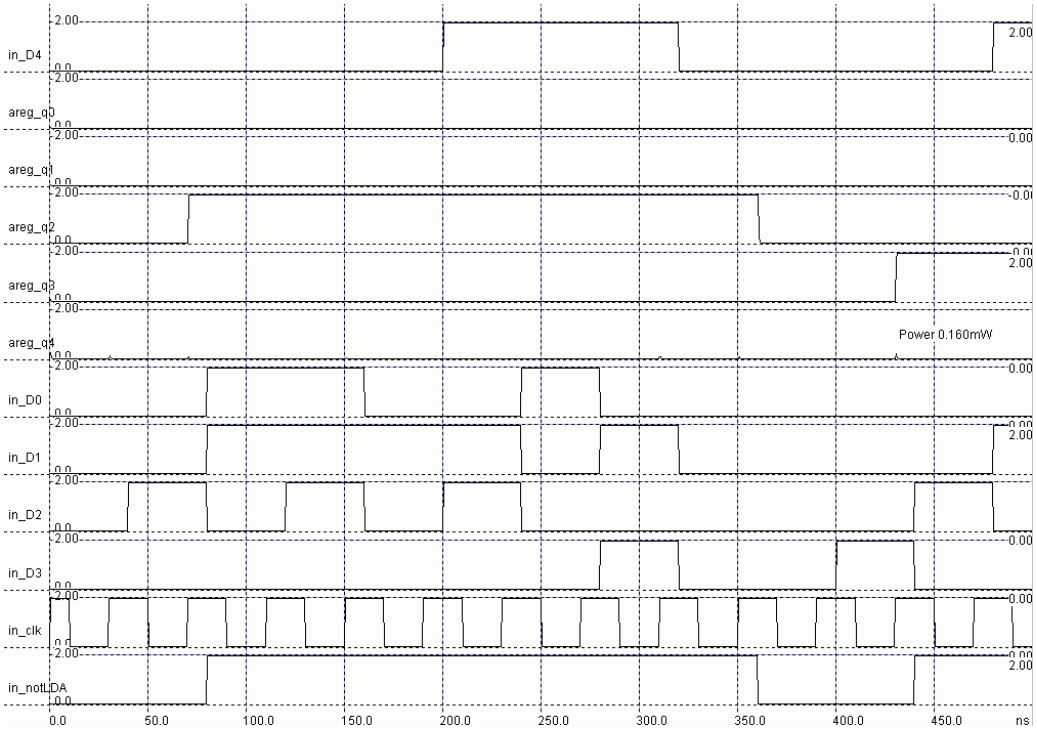

Figure A.49: **Simulation trace of a register outputs.**

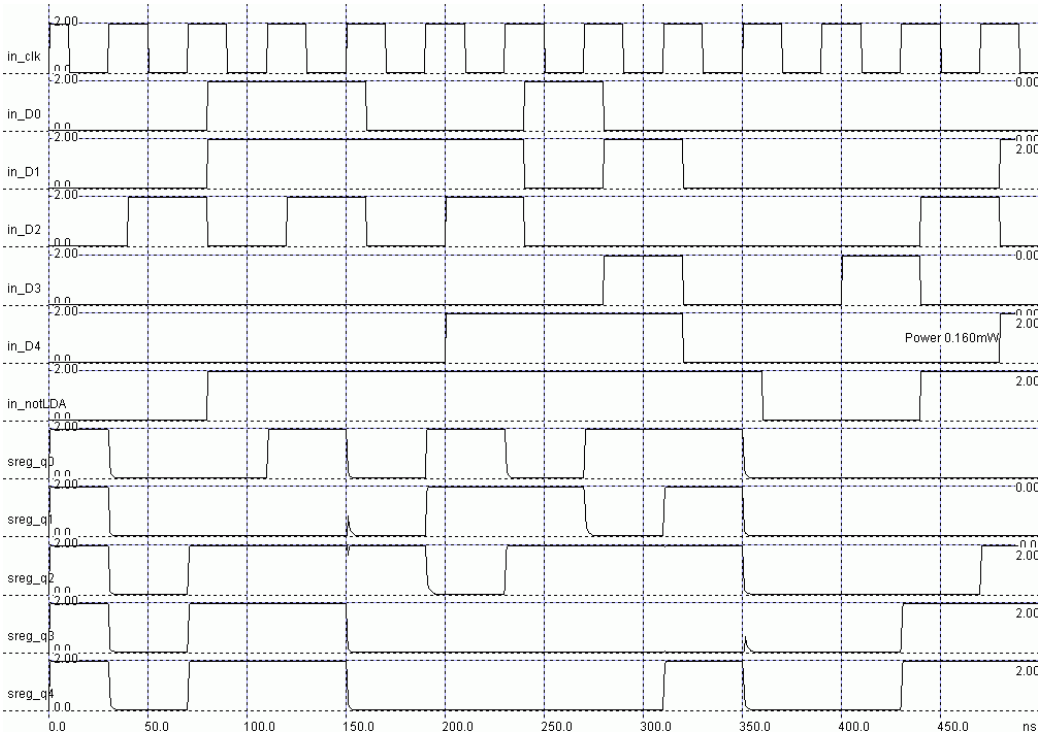

Figure A.50: **Simulation trace of sum register outputs.**

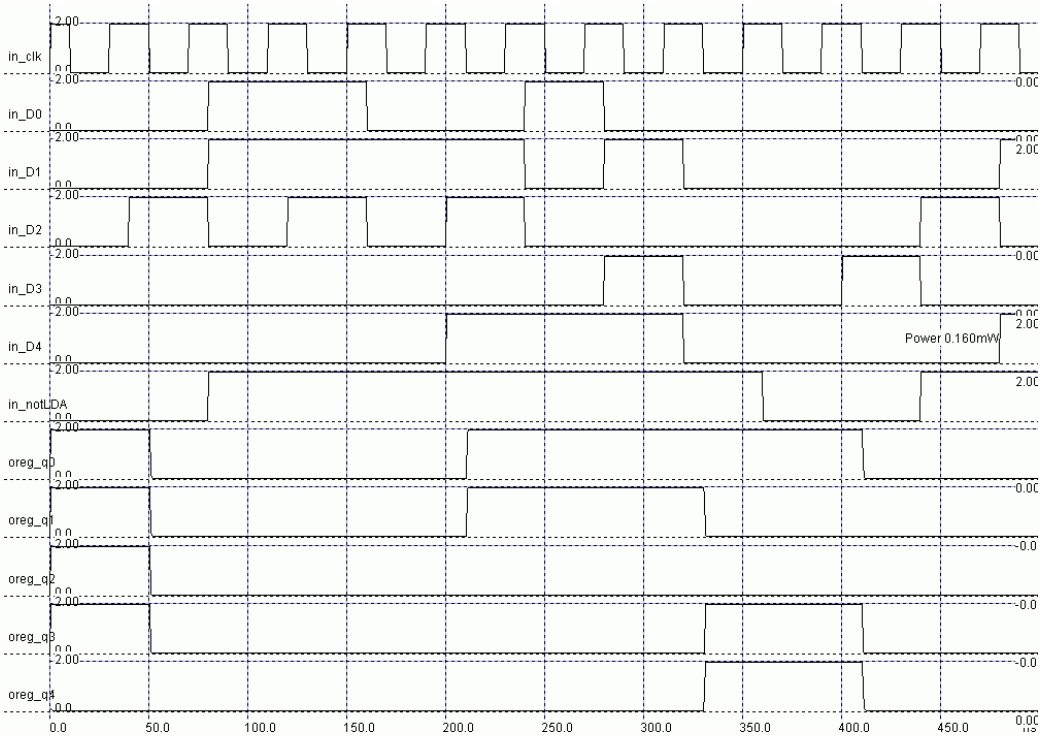

Figure A.51: **Simulation trace of output register outputs.**

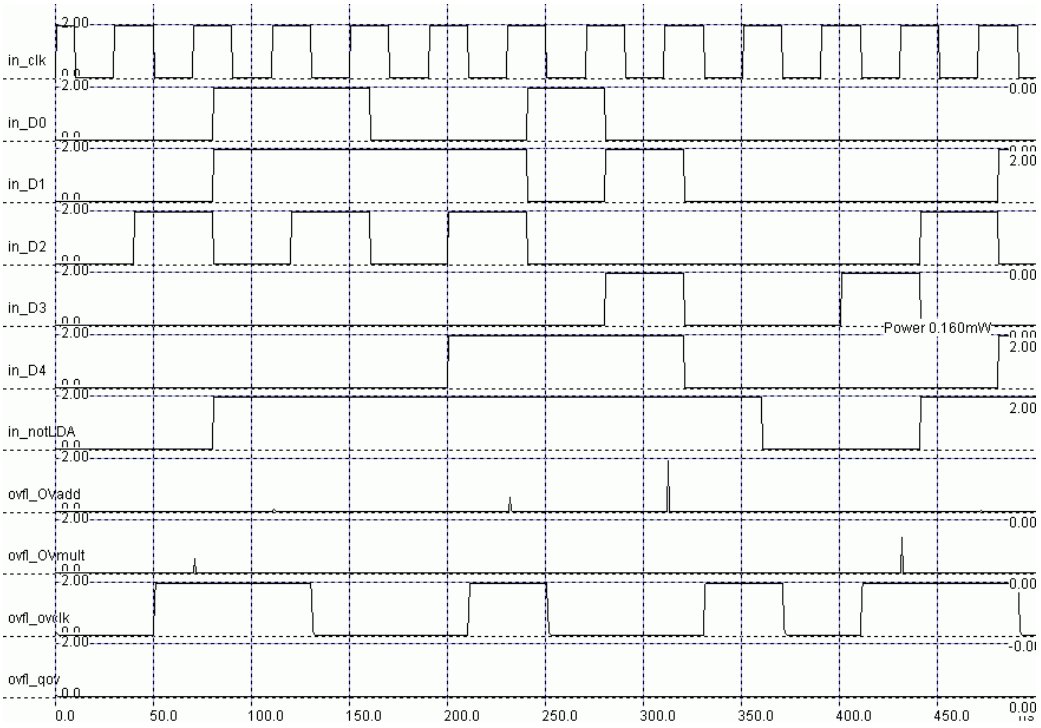

Figure A.52: **Simulation trace of overflow latch outputs.**

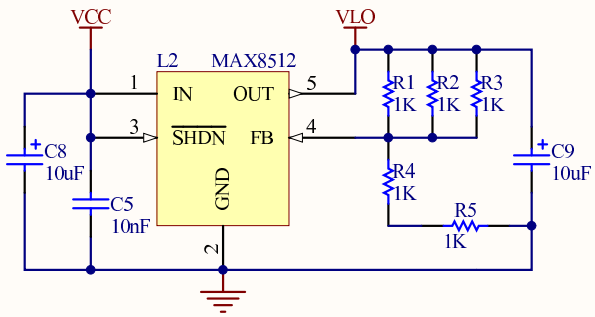

Figure A.53: **Interface circuit — power supply.** The MAX8512 chip is an ultra-low-noise, low-dropout linear regulator, designed to deliver up to 120mA continuous output current.

A.9 Test Board Design

A.9.1 Interface Circuit Design

The circuit to interface between the SW ALU and the ProTest chip tester is illustrated in figures A.53 to A.56.

A.9.2 PCB Layout

To PCB layout for the adaptor board is shown on page 196.

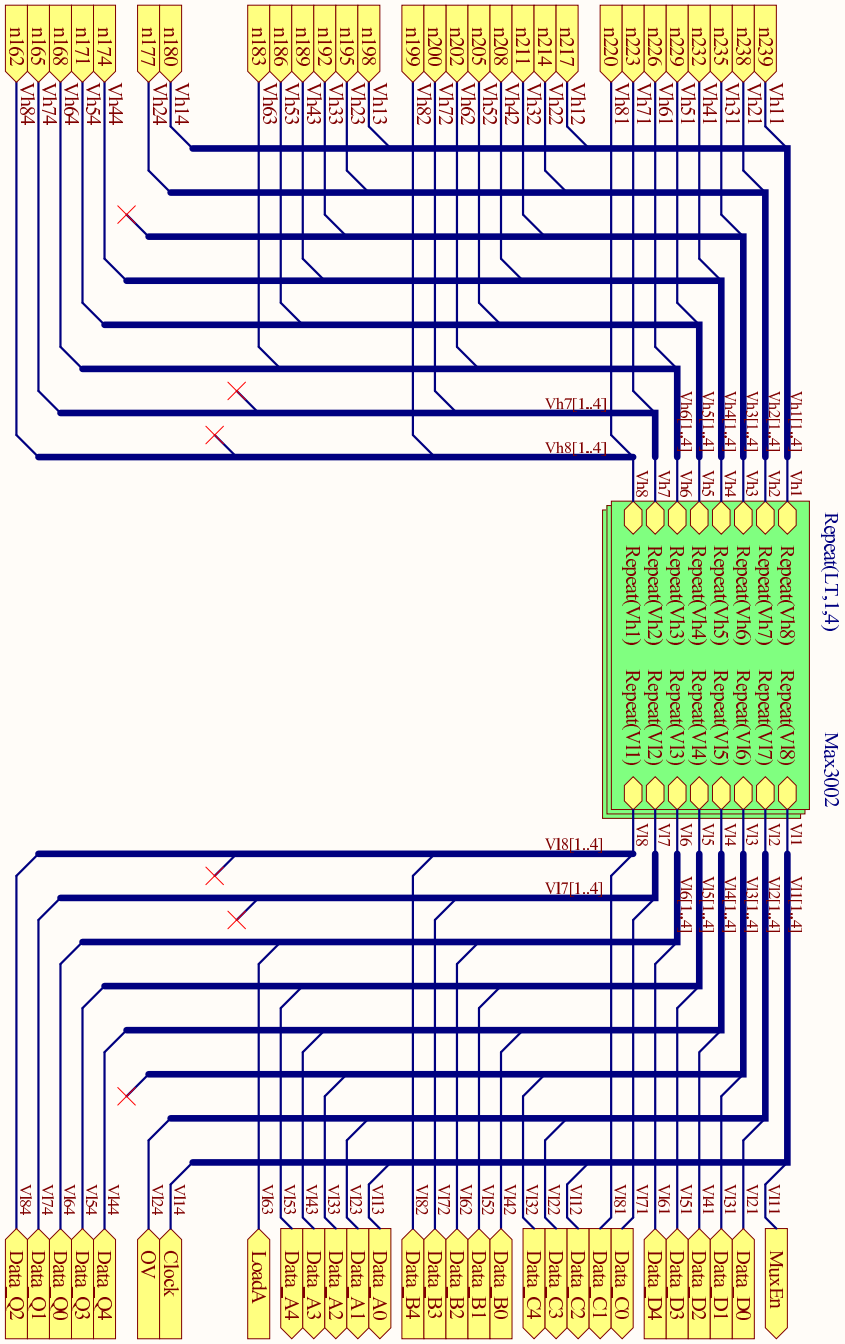

Figure A.54: **Interface circuit — level translator.** The MAX3002 chip is an eight-channel level translator that provides the level shifting necessary to allow data transfer in a multi-voltage system. This is necessary to convert the 5V signals of the tester to the 1.8V level of the ASIC.

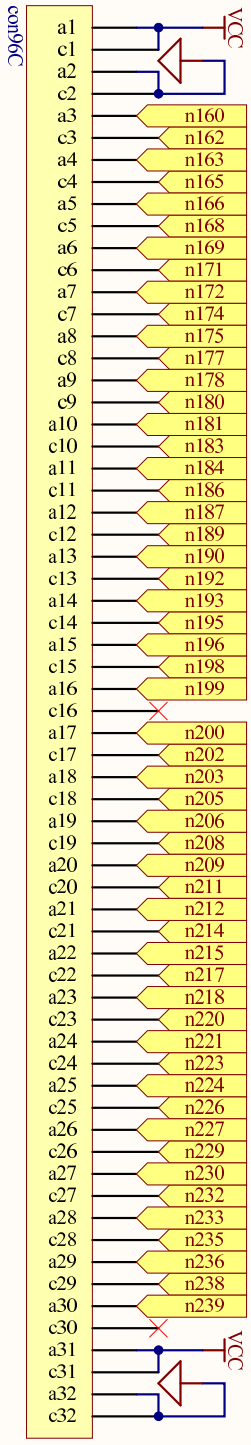

Figure A.55: **Interface circuit — connectors to ProTest tester.**

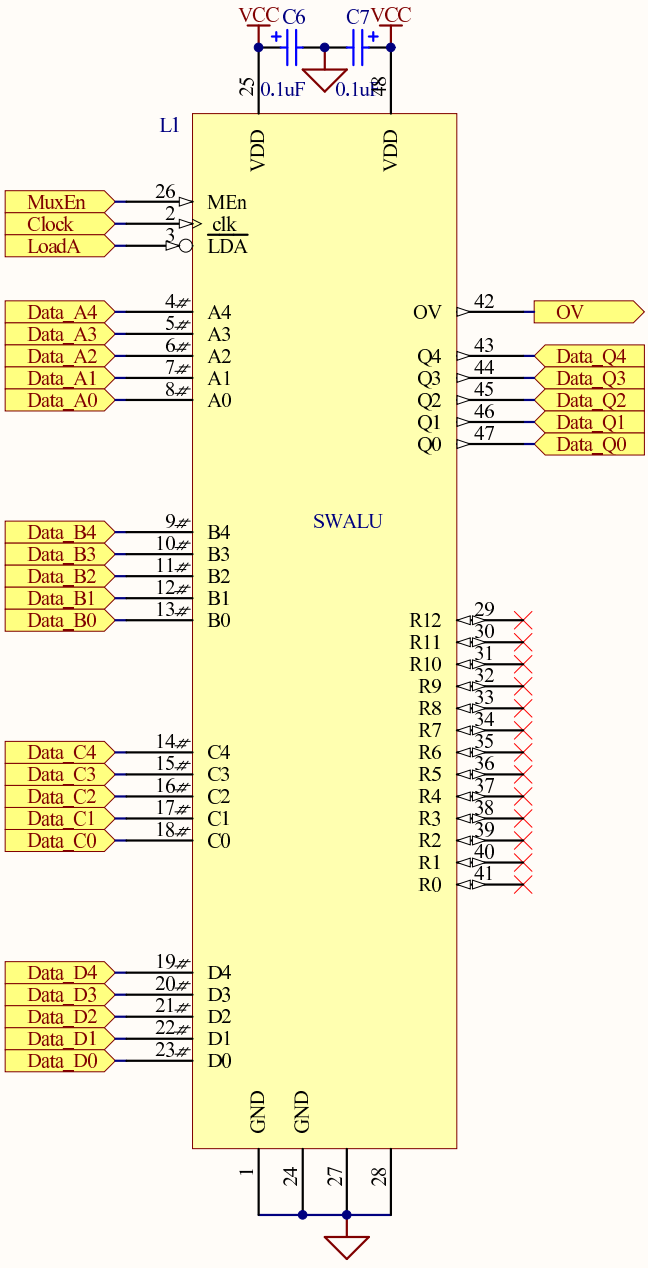

Figure A.56: **Interface circuit — SW ALU ASIC.**

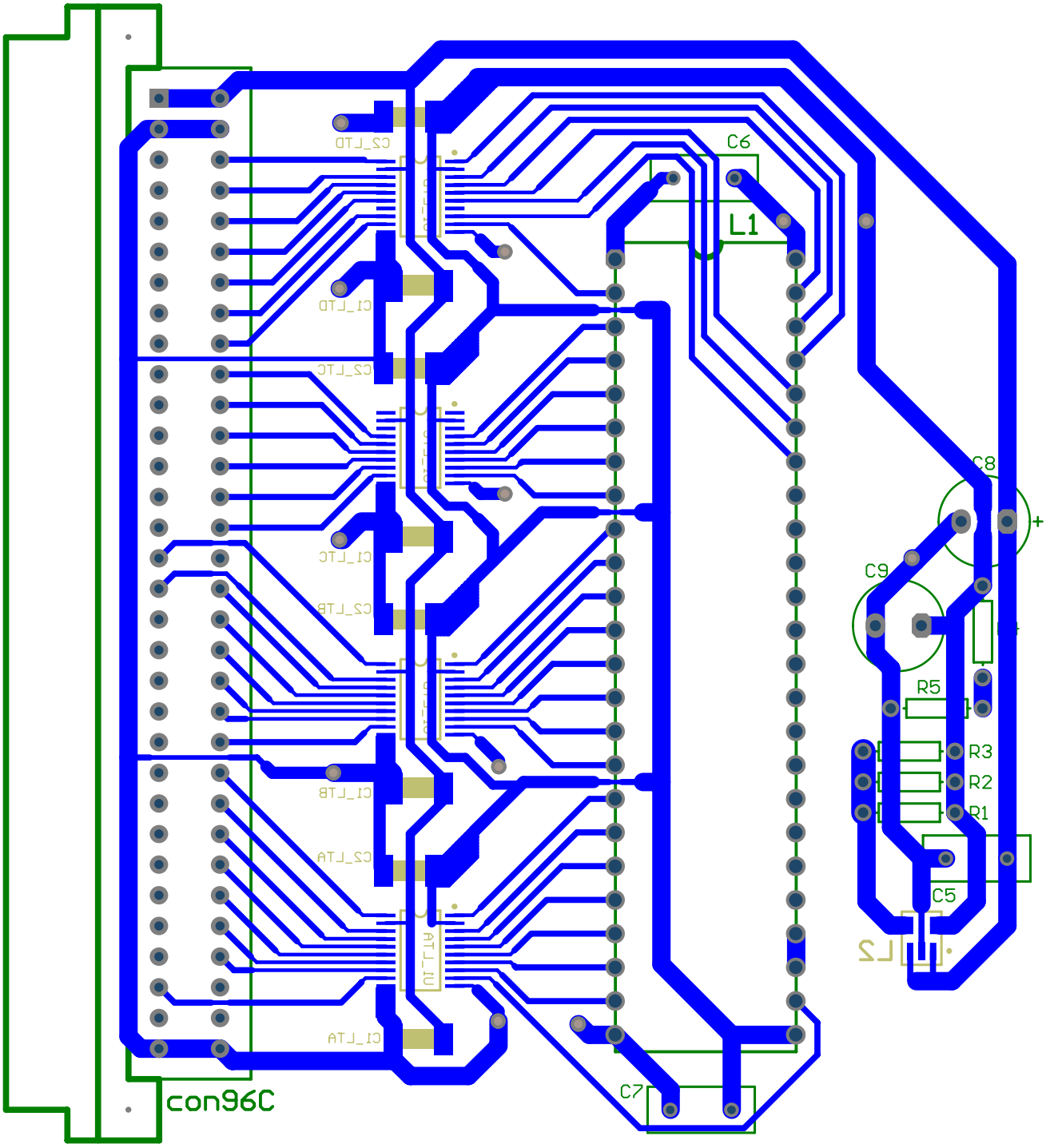

Figure A.57: **PCB layout for test adaptor board.**

Appendix B

List of Abbreviations

AC Alternating Current

ADC Analogue-to-Digital Converter

ALU Arithmetic Logic Unit

ASIC Application-Specific Integrated Circuit

BIST Built-In Self Test

CMOS Complementary Metal-Oxide Semiconductor

DDS Direct Digital Synthesiser

DAC Digital-to-Analogue Converter

DFT Discrete Fourier Transform

DSP Digital Signal Processing

DUT Device Under Test

ENOB Effective Number Of Bits

FFT Fast Fourier Transform

FPGA Field Programmable Gate Array

FIR Finite Impulse Response

IC Integrated Circuit

IIR Infinite Impulse Response

I/O Input / Output

SW Spectral Warping

VLSI Very-Large-Scale Integration technology

Appendix C

Matlab Scripts to Produce Figures

The MATLAB m-files used to generate the figures in this document are listed here.

Helper Functions

```
function c = cconv( a, b )
%CCONV Circular convolution.
%   C = CCONV( A, B ) advances B by floor( length( B ) )
%   samples then convolves A with B.  The result is wraped
%   around on itself, such that size( C ) = size( A ).
%   Works for one or two dimensions.

% do the full convolution
c = conv2( full( a ), full( b ) );
% wrap the ends around to the start, in both dimensions
sizA = size( a );
sizB = size( b );
c = [c(1:sizB(1)-1,:) + c(sizA(1)+1:end,:); c(sizB(1):sizA(1),:)];
c = [c(:,1:sizB(2)-1) + c(:,sizA(2)+1:end), c(:,sizB(2):sizA(2))];
% realign the matrix so that it looks like B has time=0 in the
% middle
mid = ceil( sizB/2 );
if mid(1) c = [c(mid(1):end,:); c(1:mid(1)-1,:)]; end
if mid(2) c = [c(:,mid(2):end), c(:,1:mid(2)-1)]; end
```

```
function g = circconv( h, f )

lh = length( h );
lf = length( f );
g = conv( h, f );
g(1:lh-1) = g(1:lh-1) + g(lf+1:end);
g = g(1:lf);
```

```
function D = dft_matrix( M )

D = ztransform_matrix( -pi*[-M:2:M-1]/M, M );
```

```
function quotant = div( dividend, divisor )
quotant = Inf*ones( size( dividend ) );
i = find( divisor );
quotant(i) = dividend(i)./divisor(i);
```

```
function [] = drawarrow(pos)
ha = axes(  ...
    'Box', 'off', ...
    'NextPlot', 'add', ...
    'Position', pos, ...
    'XColor', 'w', ...
    'XTick', [], ...
    'YColor', 'w', ...
    'YTick', [], ...
    'Visible', 'on' ...
);
ht = text( 0.55, 0.68, '\rightarrow' );
set( ht, ...
    'FontSize', 30, ...
```

```
    'FontWeight', 'bold', ...
    'HorizontalAlignment', 'center' ...
)
```

```
function h = DRAWZPLOT( fid, z, Z, origin, N, M, side, varargin )
step = 20;
aspectratio = 1.25;

do_axis = 1;   do_grid = 1;   do_vert = 1;   do_horz = 1;
do_circle = 1; do_samples = 1;do_origin = 1;

for i = 1:length( varargin )
    eval( [varargin{i} ';'] );
end

if do_axis
    h.axis = ZPLANEAX( fid, side, aspectratio );
end
if do_grid
    if do_vert
        h.grid.vert = plot( Z(:,1:step:end), 'k:' );
    end
    if do_horz
        h.grid.horz = plot( Z(1:step:end,:).', 'k:' );
    end
end
if do_circle
    h.circle = plot( z, 'b-', 'LineWidth', 0.5 );
end
if do_samples
    h.samples = plot( z(round( 1:N/M:end )), '.', ...
        'MarkerSize', 8 );
end

if isreal( origin ) origin = complex( origin ); end
if do_origin
    h.origin = plot( origin, 'o-' );
end
```

```
function domega_hat = DTHETA( omega, a, warp_type )

% for a matrix, do column at a time
if prod( size( omega ) ) > length( omega )
    domega_hat = zeros( size ( omega ) );
    for i = 1:size( z, 2 )
        domega_hat(:,i) = DTHETA( domega(:,i), a, warp_type );
    end
    return
end

switch warp_type
case 'all-pass'
    l_omega = length( omega );
        l_a = length( a );
    omega = diag( omega )*ones( l_omega, l_a );
    a = ones( l_omega, l_a )*diag( a );
    domega_hat = (1 - abs( a ).^2)./abs( exp( j*omega ) - a ).^2;
    domega_hat = prod( domega_hat, 2 );
otherwise
    error( ['Type "' warp_type '" is unrecognised.'] );
end
```

```
function varargout = FLOATING_LEGEND( pos, labels, varargin )

h = axes( 'Position', pos, 'Box', 'on', 'NextPlot', 'add', ...
    'XLim', [0 1], 'XTick', [], ...
    'YLim', [0 1], 'YTick', [], varargin{:} );
n = size( labels, 1 );
for i = 1:n
    plot( [0.06 0.2], (1 - i/(n + 1)).*[1 1], labels{i,1}{:}, ...
        'Marker', 'none' )
    plot( 0.13, 1 - i/(n + 1), labels{i,1}{:}, ...
        'LineStyle', 'none' )
    text( 0.25, 1 - i/(n + 1), labels{i,2} )
end
```

```
if nargout  varargout = {h}; end
```

```
function d = gcd( a )
%GCD    Greatest common divisor (Euclidian algorithm).

RECURSION_LIMIT = get( 0, 'RecursionLimit' );
set( 0, 'RecursionLimit', 5000 );
TOLERENCE = 1e-14;

a = abs( a(:) );
if length( a ) > 2
    d = gcd( [gcd( a(1:2) ); a(3:end)] );
else
        if a(1)<=TOLERENCE
        d = a(2);
        elseif a(2)<=TOLERENCE
        d = a(1);
        else
        d = gcd( [min( a ); abs( diff( a ) )] );
    end
end

set( 0, 'RecursionLimit', RECURSION_LIMIT );
```

```
function iD = IDFT_MATRIX( M )

iD = ZTRANSFORM_MATRIX( -pi*[-M:2:M-1]/M, M )'/M;
```

```
function a = lcm( a )
%LCM    Least common multiple.

a = a(:);
```

```
while length( a ) > 1
    a = [a(1).*a(2)./gcd( a(1:2) ); a(3:end)];
end
```

```
function C = LINEAR_INTERP_MATRIX( omega_m, N );

R = length( omega_m )/N;
omega_m = [omega_m(:); 2*pi + omega_m(1:R)];

M = length( omega_m );
omega_n = pi*[-N:2:N-1]/N;
d = N/(2*pi) * abs( ones( M, N )*diag( omega_n ) + ...
    diag( omega_m )*ones( M, N ) );
C = (1 - d).*(d < 1);

% wrap-around
C(1:R,:) = C(1:R,:) + C(end-R+1:end,:);
C = C(1:end-R,:);
```

```
function h = OMEGA_OMEGAHAT_AXIS( hf, varargin )

real_quad = 0;
for i = 1:nargin-1
    if length( varargin{i} ) == 9
        if varargin{i}=='real_quad'
            real_quad = 1;
        end
    end
end

figure( hf );
h.Figure = hf;
axis( pi*[-1 1 -1 1] );
h.Axis = gca;
set( gca, ...
    'Box', 'on', ...
```

```
    'NextPlot', 'add', ...
    'XGrid', 'on', ...
    'XTick', pi*[-1:1/4:1], ...
    'XTickLabel', [], ...
    'YGrid', 'on', ...
    'YTick', pi*[-1:1/4:1], ...
    'YTickLabel', [] ...
);
axis square;
h.XAxis = line( 2*pi*[-1 1], [0 0], 'Color', 'k' );
h.YAxis = line( [0 0], 2*pi*[-1 1], 'Color', 'k' );
labels = {'-\pi' '-\pi/2' '0' '\pi/2' '\pi'};

if real_quad
    axis( pi*[0 1 0 1] );
        for i = 3:5
        x = (i-3)*pi/2;
        h.XTickLabel(i-2) = text( x, -0.03*pi, labels{i}, ...
            'HorizontalAlignment', 'center', 'FontSize', 8 );
        h.YTickLabel(i-2) = text( -0.08*pi, x, labels{i}, ...
            'HorizontalAlignment', 'center', 'FontSize', 8 );
        end
        h.XLabel = text( pi/2, -0.1*pi, '\omega', ...
        'HorizontalAlignment', 'center' );
        h.YLabel = [text( -0.18*pi, pi/2, '\omega', ...
                    'HorizontalAlignment', 'center' )
                   text( -0.20*pi, pi/2+0.11, '\^' )]';
else
        for i = 1:5
        x = (i-3)*pi/2;
        h.XTickLabel(i) = text( x, -1.07*pi, labels{i}, ...
            'HorizontalAlignment', 'center', 'FontSize', 8 );
        h.YTickLabel(i) = text( -1.17*pi, x, labels{i}, ...
            'HorizontalAlignment', 'center', 'FontSize', 8 );
        end
        h.XLabel = text( 0, -1.2*pi, '\omega', ...
        'HorizontalAlignment', 'center' );
        h.YLabel = [text( -1.37*pi, 0, '\omega', ...
                    'HorizontalAlignment', 'center' )
                   text( -1.41*pi, 0.11, '\^' )]';
end
```

```
function [varargout] = PLOT_WRAPPED( x, y, varargin )

% plots jumps of 2 pi as dotted lines

x = x(:);
y = y(:);
N = length( y );
% Where there is a jump of 2 pi in the underlying function,
% the actual difference between the samples will be less
% than 2 pi if the slope is the same on both sides of the
% discontinuity (which is the common case).  We can get a
% good idea of how big the jump will be by looking at the
% slope on each side.
dy = diff( y );
i = find( abs( dy ) > pi ); % possible 2 pi jumps
if length( i )    % ignore jumps at the ends
    if i(1) == 1
        i = i(2:end);
    end
end
if length( i )
    if i(end) == length( dy )
        i = i(1:end-1);
    end
end
left_slope  = dy(i - 1);
right_slope = dy(i + 1);
min_jump = 2*pi - abs( left_slope + right_slope );
jump_indices = i(abs( dy(i) ) >= min_jump);
njumps = length( jump_indices );
for i = njumps:-1:1
    jump = jump_indices(i);
    x = [x(1:jump); x(jump); x(jump:end)];
    y = [y(1:jump)
         NaN
         y(jump)-sign( y(jump) )*2*pi
         y(jump+1:end)];
end
```

```
hold_status = get( gca, 'NextPlot' );
h = plot( x, y );
if njumps
    hold on;
    i = find( isnan( y ) );
        x = x(i);
    h = [h; plot( ones( 2, length( x ) )*diag( x ), ...
                  [y(i-1) y(i+1)].' )];
    set( h(2:end), 'Color', 'b', 'LineStyle', '--' )
end
if length( varargin )
        for i = 1:length( h )
        set( h(i), varargin{:} );
        end
end
set( gca, 'NextPlot', hold_status )

if nargout
    varargout = {h};
end
```

```
function v = SELECT_EVERY( u, n )
% SELECT_EVERY  Select elements in a vector
%   SELECT_EVERY(U,N) sets every element in U, except every Nth
%   element, to NaN.

%       Created: Warwick Allen   30 Sep 2003
% Last modified:       "         30 Sep 2003

v = zeros( size( u ) );
warningstate = warning( 'off' );
v(:) = u(:) + 0./~mod( [0:length( u(:) ) - 1].', n );
warning( warningstate );
```

```
function y = SINC( x )
```

```
y = ones( size( x ) );
i = find(x);
y(i) = sin(x(i))./x(i);
```

```
function varargout = SPECGRAM( f, R )

f = f(:);
N = length( f );
t = [0:N-1]';
S = zeros( N );

for n = 1:N
    S(:,n) = fft( f.*WINDOW( t - n, R, 'hanning' ) );
end

colormap gray
h = pcolor( 1 - abs( S(1:N/2,:) ) );
set( h, 'LineStyle', 'none' )
set( gca, 'YTick', [0:N/8:N/2], 'YTickLabels', [] )
yticklabels = {'0', '\pi/4', '\pi/2', '3\pi/4', '\pi'};
y = [0:N/8:N/2];
for i = 1:5
    text( -0.02*N, y(i), yticklabels{i}, ...
        'HorizontalAlignment', 'right' )
end
text( -0.25*N, N/4, '\omega' )

if nargout
    varargout = {S};
end
```

```
function g = SW( f, omega )

N = length( f );
M = length( omega );
```

```
H = ztransform_matrix( omega, N );
Dinv = idft_matrix( M );
g = Dinv * H * f;
```

```
function [] = thesis_figure( hf, name, varargin )

height = 1;
if length( varargin )
    height = varargin{1};
end
figure( hf )
clf
siz = 138;
set( hf, ...
    'Color', [1 1 1], ...
    'Name', name, ...
    'Position', [600 80 2.5*siz siz*height], ...
    'MenuBar', 'figure', ...
    'ToolBar', 'none', ...
    'PaperOrientation', 'portrait', ...
    'PaperUnits', 'points', ...
    'PaperPosition', [0 0 2.5*siz siz*height], ...
    'PaperSize', [2.5*siz siz] ...
);
```

```
function omega_hat = theta( omega, a, warp_type )

% to warp a matrix, warp one column at a time
if prod( size( omega ) ) > length( omega )
    omega_hat = zeros( size ( omega ) );
    for i = 1:size( z, 2 )
        omega_hat(:,i) = theta( omega(:,i), a, warp_type );
    end
    return
end
```

```
switch warp_type
case 'all-pass'
    l_omega = length( omega );
        l_a = length( a );
    omega = diag( omega )*ones( l_omega, l_a );
    a = ones( l_omega, l_a )*diag( a );
        factors = (exp(j*omega) - a).*(1 - a.*exp(-j*omega));
    numerator = prod( factors, 2 );
    omega_hat = atan2( imag( numerator ), real( numerator ) );
case 'piecewise_linear'
    omega_hat = zeros( size( omega ) );
    for segment = a.'
        i = omega >= segment{2}(1) & omega <= segment{2}(2);
        x = eval( segment{1} );
        omega_hat(i) = x(i);
    end
otherwise
    error( ['Type "' warp_type '" is unrecognised.'] );
end
```

```
function z_hat = THETA_( z, a, type )

% to map a matrix, map one column at a time
if prod( size( z ) ) > length( z )
    z_hat = zeros( size (z ) );
    for i = 1:size( z, 2 )
        z_hat(:,i) = THETA_( z(:,i), a, type );
    end
    return
end

z = z(:);     % make sure this is a column vector
a = a(:).';  % make sure this is a row vector
l_z = length( z );
l_a = length( a );

switch type
case 'all-pass'
```

```
    numerator = prod( diag( z )*ones( l_z, l_a ) - ...
            ones( l_z, l_a )*diag( a ), 2 );
    denominator = prod( 1 - ...
            diag( z )*ones( l_z, l_a )*diag( conj( a ) ), 2 );
    z_hat = DIV( numerator, denominator );
otherwise
    error( ['Type "' type '" is unrecognised.'] );
end
```

```
function g = UPSAMPLE( f, p )
% UPSAMPLE  Sinc Interpolation
%   G = UPSAMPLE(F,P) resamples F at P times the original sample
%   rate using sinc interpolation.

%       Created: Warwick Allen  30 Sep 2003
% Last modified:       "        30 Sep 2003

isrow = size( f, 1 )==1;
if isrow f = f.'; end
siz = size( f );
len = siz(1);
siz(1) = len*p;
g = zeros( siz );
for i = 1:siz(2)
    F = fft( f(:,i) );
    F = [F(1:floor( len/2 ))
         zeros( len*(p - 1), 1 )
         F(floor( len/2 )+1:end)];
    g(:,i) = p*ifft( F );
end
if isrow g = g.'; end
% remove imaginary part generated by round-off errors
if isreal( f ) g = real( g ); end
```

```
function g = WARP_BY_COLUMNS_OF_S( S, f_in )
[M,N] = size( S );
```

```
len = M/N*length( f_in );
% Insert M - N zeroes after each input frame to produce
% the right number of output samples and prevent the
% output frames overlapping
f = zeros( len, 1 );
f(mod( [0:len-1], M ) < N) = f_in;
% Time reverse the input, frame by frame
i = [0:len - 1];
f = f(M - mod( i, M ) + M*floor( i/M ));
% Append zeroes to allow time for the signal to propagate
% through the filters
f = [f; zeros( M-1, 1 )];

g = zeros( len + M-1, 1 );
i = ~mod( [1:len + M-1]', M );
for n = 1:N
    % Downsample by M
    x = f .* i;
    % Filter by the nth column of S
    x = filter( S(:,n), 1, x );
    % Add this to the output from the other filters
    g = g + x;
    % Time delay input
    f = [0; f(1:end-1)];
end
g = g(M:end);  % The first M - 1 samples are always zero
```

```
function g = WARP_BY_ROWS_OF_S( S, f_in )
[M,N] = size( S );
len = M/N*length( f_in );
% Insert M - N zeroes after each input frame to produce
% the right number of output samples and prevent the
% output frames overlapping
f = zeros( len, 1 );
f(mod( [0:len-1], M ) < N) = f_in;
% Time reverse the input, frame by frame
i = [0:len - 1];
f = f(M - mod( i, M ) + M*floor( i/M ));
```

```
% Append zeroes to allow time for the signal to propagate
% through the filters
f = [f; zeros( M-1, 1 )];

g = zeros( len + M-1, 1 );
i = ~mod( [1:len + M-1]', M );
for m = M:-1:1
    % Filter by the mth row of S
    x = filter( S(m,:), 1, f );
    % Downsample by M
    x = x .* i;
    % Time delay output
    g = [0; g(1:end-1)];
    % Add this to the output from the other filters
    g = g + x;
end
g = g(M:end);  % The first M-1 samples are always zero
```

```
function w = WINDOW( t, R, type )
switch type
case 'hanning'
    w = (1 + cos( 2*pi/R*t ))/2;
    w(abs( t ) > R/2) = 0;
otherwise
    error( ['Unknown type "' type '"'] );
end
```

```
function ha = ZPLANEAX( hf, side, aspectratio )
figure( hf )
if side(1) == 'l'      % left side
    pos = [0 0.1 0.45 0.9];
elseif side(1) == 'r' % right side
    pos = [0.55 0.1 0.45 0.9];
else                   % middle
    pos = [0.275 0.1 0.45 0.9];
end
```

```
ha = axes(   ...
    'Box', 'off', ...
    'NextPlot', 'add', ...
    'Position', pos, ...
    'XColor', [1 1 1], ...
    'XLim', aspectratio*[-1.15 1.3], ...
    'XTick', [], ...
    'YColor', [1 1 1], ...
    'YLim', [-1.15 1.3], ...
    'YTick', [], ...
    'Visible', 'on' ...
);
len = 0.1;
ticks = [   -1,-len    -1,0
             1,-len     1,0
       -len,-1          0,-1
       -len, 1          0, 1      ];
line( ticks(:,[1 3])', ticks(:,[2 4])', ...
        'Color', 'k', 'LineWidth', 1 );
xlims = get( ha, 'XLim' )/aspectratio*1.25;
ylims = get( ha, 'YLim' );
line( xlims, [0 0], 'Color', 'k', 'LineWidth', 1 );
line( [0 0], ylims, 'Color', 'k', 'LineWidth', 1 );
patch( [xlims(2) 0.96*xlims(2)*[1 1]], ...
        [0 0.02*ylims(2)*[1 -1]], 'k' )
patch( [0 0.02*xlims(2)*[1 -1]], ...
        [ylims(2) 0.96*ylims(2)*[1 1]], 'k' )
text( 1.2, -0.15, 'Re' )
text( 0.1,  1.15, 'Im' )
```

```
function H = ztransform_matrix( omega, N )

H = exp( j*omega(:)*[0:N-1] );
```

Figure 1.1

```
function [] = types_of_signals( NN )

t = [0:NN-1];
omega1 =  4 * 2*pi/NN;
omega2 = 13 * 2*pi/NN;
f = (3*sin( omega1*t ) + sin( omega2*t ))/4;
T = NN/64;  % Sampling period
Q = 4;      % Number of discrete levels in the range [0,1)

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 1.5], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 1.5], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 1.5] );

subplot( 2, 2, 1 )
plot( t, f, 'LineWidth', 1 )
adjust_graph( 1, NN )
title( 'Continuous Domain' )
h = ylabel( ['Continuous' char( 10 ) 'Range'], 'Color', 'k' );
set( h, 'Position', get( h, 'Position' ) - [NN/4 0 0] )

subplot( 2, 2, 2 )
plot( t(1:T:end), f(1:T:end), '.' )
adjust_graph( 2, NN )
title( 'Discrete Domain' )

subplot( 2, 2, 3 )
plot( [0:1/64:NN-1/64], round( Q*UPSAMPLE( f, 64 ) )/Q, '-', ...
    'LineWidth', 1 )
adjust_graph( 3, NN )
h = ylabel( ['Discrete' char( 10 ) 'Range'], 'Color', 'k' );
set( h, 'Position', get( h, 'Position' ) - [NN/4 0 0] )

subplot( 2, 2, 4 )
plot( t(1:T:end), round( Q*f(1:T:end) )/Q, '.' )
adjust_graph( 4, NN )

function [] = adjust_graph( i, NN )
labels = {'(a)' '(b)' '(c)' '(d)'};
axis( [0 NN -1 1.2] )
line( [0 NN], [0 0], 'Color', 'k' )
```

```
set( gca, ...
    'XTick', [], 'YTick', [], ...
    'XColor', 'w', 'YColor', 'w', ...
    'Box', 'off' )
xlabel( labels{i}, 'Color', 'k' )
```

Figure 1.2

```
function [] = decaying_signal( NN )

t = [0:NN-1];
tau = NN/8;
f = exp( -t/tau );

plot( t, f, 'LineWidth', 1 )
axis( [0 NN 0 1] )
xlabel( 'time' )
h = ylabel( 'amplitude' );
pos = get( h, 'Position' );
set( h, 'Position', [pos(1)*0.8 pos(2:3)] )
set( gca, ...
    'XTick', tau, 'XTickLabel', [], ...
    'YTick', [0 exp( -1 ) 1], 'YTickLabel', ['0';' ';'1'] )
ylim = get( gca, 'YLim' );
xlabelheight = ylim(2) - 67/64*diff( ylim );
text( tau, xlabelheight, '\tau', ...
    'HorizontalAlignment', 'center' )
text( -NN/16, exp( -1 ), 'e^{-1}' )
line( [0 tau tau tau], [exp( -1 ) exp( -1 ) 0 0], ...
    'Color', 'k', 'LineStyle', ':' )
set( gca, 'Box', 'off' )
text( tau, exp( -1/2 ), 'e^{-t/\tau}', ...
    'HorizontalAlignment', 'center' )
```

Figure 1.3

```
function [] = impulse_response_example( NN )

t = [0:NN-1].';
tau = NN/5;
omega = pi/tau;
t0 = tau/2;
g = exp( (t0 - t)/tau ) .* ...
    (exp( j*omega*(t0-t) ) + exp( j*omega*(t - t0)))/2;

plot( t, g, 'LineWidth', 1 )
line( [0 NN], [0 0], 'Color', 'k' )
h = xlabel( 'time' );
h = ylabel( 'amplitude' );
pos = get( h, 'Position' );
set( h, 'Position', [pos(1)*0.8 pos(2:3)] )
set( gca, 'Box', 'off', 'XLim', [0 NN], 'YLim', [-0.5 1.1], ...
    'XTick', tau.*[1:6], 'XTickLabel', [] )

ylim = get( gca, 'YLim' );
ylabelheight = ylim(2) - 67/64*diff( ylim );
text( 0, ylabelheight, '0', 'HorizontalAlignment', 'center' )
text( tau, ylabelheight, '\tau', ...
    'HorizontalAlignment', 'center' )
for i = 2:floor( NN/tau )
        text( i*tau, ylabelheight, [num2str( i ) '\tau'], ...
        'HorizontalAlignment', 'center' )
end
```

Figure 1.4

```
function [] = sampling( NN )

t = [0:NN-1];
Ts = 24;
omega = 2*pi/Ts.*[-4:8/NN:4-1/NN];
f = SINC( 10*2*pi/NN*(t-NN/2) ).*WINDOW( t-NN/2, NN, 'hanning' );
F = fftshift( fft( f ) );
h = ~mod( t, Ts );
H = zeros( 1, NN );
```

```
H(round( [1:( NN/Ts ):end] )) = 1;
g = f.*h;
G = fftshift( fft( g ) );

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 3], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 3], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 3] );

subplot( 3, 2, 1 )
plot( t, f, 'LineWidth', 1 )
setTimePlot( 'a', NN )
title( 'Time Domain' )

subplot( 3, 2, 2 )
plot( omega, abs( F ), 'LineWidth', 1 )
setFreqPlot( 'b' )
title( 'Frequency Domain' )

subplot( 3, 2, 3 )
plot( t, h, 'LineWidth', 1 )
setTimePlot( 'c', NN )

subplot( 3, 2, 4 )
plot( omega, H, 'LineWidth', 1 )
setFreqPlot( 'd' )

subplot( 3, 2, 5 )
plot( t, g, 'LineWidth', 1 )
setTimePlot( 'e', NN )

subplot( 3, 2, 6 )
plot( omega, abs( G ), 'LineWidth', 1 )
setFreqPlot( 'f' )


function setTimePlot( label, NN )
set( gca, 'Position', [1 1 1 0.9].*get( gca, 'Position' ), ...
    'XLim', [0 NN], 'Box', 'off' )
xlabel( ['(' label ')'] )
hy = ylabel( 'Amplitude' );
```

```
pos = get( hy, 'Position' );
set( hy, 'Position', [-0.23*NN pos(2:3)] )


function setFreqPlot( label )
Xlim = [-0.3 0.3];
axis tight
set( gca, 'Position', [1 1 1 0.9].*get( gca, 'Position' ), ...
    'XLim', Xlim, 'Box', 'off' )
xlabel( ['(' label ')'] )
hy = ylabel( 'Magnitude' );
pos = get( hy, 'Position' );
set( hy, 'Position', [-0.41 pos(2:3)] )
```

Figure 1.5

```
function [] = sinc_signal( NN )

t = [-NN/2:NN/2-1];
Ts = NN/8;
s = SINC( pi*(t/Ts) );
plot( t, s, 'LineWidth', 1 )
set( gca, 'Box', 'off', ...
    'XLim', [-NN NN]/2, 'XTick', Ts*[-4:4], ...
    'XTickLabel', [], ...
    'YLim', [1.1*min( s ) 1], 'YTick', [0 1] )
line( [-NN 0;NN 0], [0 min( s );0 1], 'Color', 'k' )
xticklabels = {'-4T_s' '-3T_s' '-2T_s' '-T_s' '0' ...
               'T_s' '2T_s' '3T_s' '4T_s'};
for i = 1:9
    text( (i-5)*Ts, 1.1*min( s ), xticklabels{i}, ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'top' )
end
```

Figure 2.1

```
function [] = ...
    graphical_representation_of_steps_involved_in_sw( M, NN )

w = 2*pi*[0:NN-1]/(NN-1);
a =1/2;
z = exp( j*w );
gridlines = [-1:0.01:1];
Z = gridlines(ones( length( gridlines ), 1 ),:);
Z = Z + j*Z.';

zhat = DIV( (z + a), (1 + a'*z) );
Zhat = DIV( (Z - a), (1 - a'*Z) );

set( gcf, ...
    'Position',     get( gcf, 'Position'      ).*[1 1 1.25 1], ...
    'PaperSize',    get( gcf, 'PaperSize'     ).*[1.25 1], ...
    'PaperPosition',get( gcf, 'PaperPosition' ).*[1 1 1.25 1] );

DRAWZPLOT( gcf, zhat, Z, 0, NN, M, 'left', 'do_grid=0', ...
    'do_origin=0', 'aspectratio=1.5625' );
set( gca, 'Position', get( gca, 'Position' )-[0.04 0 0 0] )
xlabel( '(a)', 'Position', [0 -1.18 0], 'Color', 'k' )

DRAWZPLOT( gcf, z, Zhat, -a, NN, M , 'right', 'do_grid=0', ...
    'do_origin=0', 'aspectratio=1.5625' );
set( gca, 'Position', get( gca, 'Position' )+[0.05 0 0 0] )
xlabel( '(c)', 'Position', [0 -1.18 0], 'Color', 'k' )

axes( 'Position', [0.425 0.15 0.15 0.8] )
omega_hat = pi*[-M:2:M-1]./M;
omega = atan2( (1 - a^2)*sin( omega_hat ), ...
               (1 + a^2)*cos( omega_hat ) + 2*a );
plot( [omega; unwrap( omega_hat )], 'b' )
xlabel( '(b)', 'Position', [1.5 -3.6 0] )
yticks = [-pi:pi/2:pi];
ylabels = {'-\pi', '-\pi/2', '0', '\pi/2', '\pi'};
set( gca, 'XTick', [], 'YGrid', 'on', ...
          'YLim', [-pi pi], 'YTick', yticks, 'YTickLabel', [] );
for i = 1:5
```

```
    text( 0.9, yticks(i), ylabels{i}, ...
        'HorizontalAlignment', 'right' )
    text( 2.1, yticks(i), ylabels{i}, ...
        'HorizontalAlignment', 'left' )
end
```

Figure 2.2

```
function [] = order1_zplane_warping( M, NN )

w = 2*pi*[0:NN-1]/(NN-1);
a = 1/2;
zhat = exp( j*w );  % evenly spaced samples in warped plane
gridlines = [-1:0.01:1];
Z = gridlines(ones( length( gridlines ), 1 ),:);
Z = Z + j*Z.';

z = THETA_( zhat, -a, 'all-pass' ); % inverse warp the samples
Zhat = THETA_( Z,  a, 'all-pass' ); % forward warp the grid

DRAWZPLOT( gcf, z, Z, 0, NN, M, 'left' );
xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], 'Color', 'k' )
DRAWZPLOT( gcf, zhat, Zhat, -a, NN, M, 'right' );
xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], 'Color', 'k' )
text( -0.33, -1.22, '\^' )
DRAWARROW( [0.47 0.49 0.06 0.08] )
ahat = THETA_(a,a,'all-pass');
```

Figure 2.3

```
function [] = order1_zplane_warping__complex_( M, NN )

w = 2*pi*[0:NN-1]/(NN-1);
a = 1/4.*(j - 1);
zhat = exp( j*w );  % evenly spaced samples in warped plane
gridlines = [-1:0.01:1];
```

```
Z = gridlines(ones( length( gridlines ), 1 ),:);
Z = Z + j*Z.';

z = THETA_( zhat, -a, 'all-pass' ); % inverse warp the samples
Zhat = THETA_( Z,  a, 'all-pass' ); % forward warp the grid

DRAWZPLOT( gcf, z, Z, 0, NN, M, 'left' );
xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], 'Color', 'k' )
DRAWZPLOT( gcf, zhat, Zhat, -a, NN, M, 'right' );
xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], 'Color', 'k' )
text( -0.33, -1.22, '\^' )
DRAWARROW( [0.47 0.49 0.06 0.08] )
```

Figure 2.4

```
function [] = order1_freq_warping( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf );

omegahat = THETA( omega, 1/2, 'all-pass' );
plot( omega, omegahat, 'LineWidth', 1 )
```

Figure 2.5

```
function [] = order1_freq_warping2( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf );

omegahat = THETA( omega, 3/4, 'all-pass' );
plot( omega, omegahat, 'LineWidth', 1 )
text( -5*pi/8, 5*pi/8, '\ita\rm = 3/4', 'FontSize', 8 )
line( [-2 1]*pi/16, [5 5]*pi/8, 'Color', 'k' )

omegahat = THETA( omega, -1/4, 'all-pass' );
```

```
plot( omega, omegahat, 'LineWidth', 1 )
text( -7*pi/8, -1*pi/8, '\ita\rm = -1/4', 'FontSize', 8 )
line( -[11 10]*pi/16, -[2 3]*pi/8, 'Color', 'k' )
```

Figure 2.6

```
function [] = order1_freq_warping( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf );

omegahat = THETA( omega, 3*(1 - j)/4, 'all-pass' );
PLOT_WRAPPED( omega, omegahat, 'LineWidth', 1 )
```

Figure 2.9

```
function [] = order2_zplane_warping__complex_conj_( M, NN )

a = [1+j 1-j]./(2*sqrt(2));
omega = 2*pi*[0:NN-1]'/(NN-1);
z = exp( j*omega );
zhat = THETA_( z, a, 'all-pass' );

hzplot{1} = DRAWZPLOT( gcf, z, [], 0, ...
    NN, M, 'left', 'do_grid=0' );
xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], ...
    'Color', 'k' )
hzplot{2} = DRAWZPLOT( gcf, zhat, [], prod(a), ...
    NN, M, 'right', 'do_grid=0' );

% draw arrows
omega_start = omega(NN/M*[1 11 15 5] + 1);
omega_end = THETA( omega_start, a, 'all-pass' );
omega_end = mod( omega_end, 2*pi );
domega = (omega_end - omega_start)./NN;
for i = 1:length(omega_end)
```

```
    omega = [omega_start(i):domega(i):omega_end(i)].';
    phi = exp( j*omega );
        dist_from_end   = omega - omega_end(i);
        dist_from_start = omega_start(i) - omega;
    s = (dist_from_start.*dist_from_end).^(1/2);
    s = (-1)^(xor(i>2,mod(i,2))) * s/max( s )/3;
        plot( (3/2).^s.*phi, 'k', 'LineWidth', 0.25 )
    k = find(sign( domega(i) ) * ...
        omega > sign( domega(i) )*omega_end(i) - pi/32);
        arrowhead = ...
        ( [(3/2).^s(k(ceil(end/3))).*phi(k(ceil(end/3)))
           (9/5).^s(k).*phi(k)
           flipud( (5/4).^s(k).*phi(k))] );
    patch( real( arrowhead ), imag( arrowhead ), ...
        'k', 'LineStyle', 'none' )
end

xlabel( '\itz\rm-plane', 'Position', [0 -1.2 0], ...
    'Color', 'k' )
text( -0.33, -1.22, '\^' )
DRAWARROW( [0.47 0.49 0.06 0.08] )
```

Figure 2.7

```
function [] = order2_freq_warping( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf );

omegahat = THETA( omega, [-3/4 3/4], 'all-pass' );
PLOT_WRAPPED( omega, omegahat, 'LineWidth', 1 )
```

Figure 2.8(a)

```
function h = order2_freqwarp__cmplxcnj_a( NN )
```

```
omega = 2*pi/NN*[-NN/2:NN/2-1];
h = OMEGA_OMEGAHAT_AXIS( gcf );

a = [1+j 1-j]/(2*sqrt(2));
omegahat = THETA( omega, a, 'all-pass' );
h.Plot = PLOT_WRAPPED( omega, omegahat, 'LineWidth', 1 );
```

Figure 2.8(b)

```
function h = order2_freqwarp__cmplxcnj_b( NN )

h = order2_freqwarp__cmplxcnj_a( NN );
omega = get( h.Plot(1), 'XData' ).';
omegahat = get( h.Plot(1), 'YData' ).';
set( h.Plot(2), 'Visible', 'off' )
set( h.Plot(3), 'Visible', 'off' )

axis( 2*pi*[-1 1 -1 1] )
labels = {'-2\pi' '-\pi' '0' '\pi' '2\pi'};
set( h.Axis, 'XTick', pi*[-2:1/2:2], 'YTick', pi*[-2:1/2:2] )
for i = 1:length( h.XTickLabel )
    set( h.XTickLabel(i), ...
        'Position', 2*get( h.XTickLabel(i), 'Position' ), ...
        'String', labels{i} );
    set( h.YTickLabel(i), ...
        'Position', 2*get( h.YTickLabel(i), 'Position' ), ...
        'String', labels{i} );
end
set( h.XLabel, 'Position', 2*get( h.XLabel, 'Position' ) )
set( h.YLabel(1), ...
    'Position', 2*get( h.YLabel(1), 'Position' ) )
set( h.YLabel(2), ...
    'Position', 2*get( h.YLabel(2), 'Position' ) )

for omegahat_offset = -2*pi*[-2:2]
        for omega_offset = -2*pi*[-2:2]
        plot( omega+omega_offset, ...
            unwrap( omegahat )+omegahat_offset, ...
            'LineWidth', 0.5 )
```

```
        end
end
```

Figure 2.10

```
function [] = pwlinear_freq_warping( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf );

omegahat = THETA( omega, {
    '(2*omega - pi)/3' [-pi   -pi/4]
    '2*omega'          [-pi/4  pi/4]
    '(2*omega + pi)/3' [pi/4   pi]
    }, 'piecewise_linear' );
plot( omega, omegahat, 'LineWidth', 1 )
```

Figure 2.11

```
function [] = distortion_of_bandwidth( NN )

omega = 2*pi/NN*[-NN/2:NN/2-1];
OMEGA_OMEGAHAT_AXIS( gcf, 'real_quad' );

a = -3/4;
omegahat = THETA( omega, a, 'all-pass' );
plot( omega, omegahat, 'LineWidth', 1 )

L = 48;             % window length
omega0 = 7*pi/8;    % centre frequency
N1 = 100;           % start of window
F = 8*NN/L*( ...
        1/2 * hanning_spectrum( omega - omega0, L, N1 ) + ...
        1/2 * hanning_spectrum( omega + omega0, L, N1 ) ...
    );
```

```
plot( omega, abs( F )/2, 'LineWidth', 1 )
plot( abs( F )/2, omegahat, 'LineWidth', 1 )

% find the indices of the points of interest
i = round( find( omega==omega0 )+[-2*NN/L 0 2*NN/L] );

% draw tangent
s = inline( '(1-2*a*cos(omega)+a^2)/(1-a^2)', 'a', 'omega' );
m = 1/s( a, omega0 );
c = omegahat(i(2)) - m*omega0;
plot( omega([1 end]), m*omega([1 end])+c, 'k-.' )

% draw lines
line( omega(i(1)).*[1;1;0], omegahat(i(1)).*[0;1;1], ...
    'Color', 'k', 'LineStyle', '--' )
line( omega(i(2)).^[1;1;0], omegahat(i(2)).^[0;1;1], ...
    'Color', 'k', 'LineStyle', '--' )
line( omega(i(3)).*[1;1;0], omegahat(i(3)).*[0;1;1], ...
    'Color', 'k', 'LineStyle', '--' )

text( 0.8, 0.6, ['constant slope';' approximation'], ...
    'FontSize', 6 )
line( [1.92 2.4], [0.58 0.46], 'Color', 'k' )


function U = hanning_spectrum( omega, L, N1 );
% Returns the spectrum of a hanning widow of length L,
% starting at sample N1
N = length( omega );
U = exp( -j*omega*(N1 + L/2) ).*(L/N).*(...
    SINC( L/2*omega ) + ...
    SINC( L/2*omega + pi )/2 + ...
    SINC( L/2*omega - pi )/2)/2;
```

Figure 2.12

```
function [] = bw_stretching_S( NN )

set( gcf, ...
```

```
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
colormap gray
xlabels = {'0','\pi/4','\pi/2','3\pi/4','\pi'};
zlabels = {'-3','0',' 3','6','\geq 9'};
omega = pi*[-NN:2:NN-1]'/NN;
L = 2.^[10 7 4];
for i = 1:length(L)
    B = 8*pi/L(i);          % The bandwidth
    phi = B/2;
    a = [-7/8:7/NN:7/8]; % The range of warping factors
    omega_0 = omega;       % The centre freq.
    omega_0 = omega((omega>phi) & (omega<pi-phi));% Max/min freqs

        a = ones(size(omega_0,1),size(a,2))*diag(a);
        omega_0 = diag(omega_0)*ones(size(omega_0,1),size(a,2));
        U = {(1 - a.^2).*sin(omega_0 + phi)
         (1 + a.^2).*cos(omega_0 + phi) - 2*a};
        V = {(1 - a.^2).*sin(omega_0 - phi)
         (1 + a.^2).*cos(omega_0 - phi) - 2*a};
    % The warped bandwidth
        Bhat = abs( atan2( U{:} ) - atan2( V{:} ) );
        % Bandwidth stretching factor
    S = Bhat./B;
        Stilde = DIV( 1 - a.^2, ...
                  1 - 2*a.*cos(omega_0) + a.^2 );



    h.Axis(i) = axes( 'Position', [0.35 1-0.3*i 0.34 0.22] );
    h.Surf(i) = surf( omega_0, a,   S, ...
        'LineStyle', 'none' );
    set( h.Axis(i), ...
        'XLim', [0 pi], ...
        'XTick', [0:pi/8:pi], ...
        'XTickLabel', [], ...
        'YLim', [-0.8 0.8], ...
        'ZLim', [0 4], ...
        'FontSize', 8 ...
        )
```

```
    for k = 1:length( xlabels )
        h.XTickLabel(i,k) = ...
            text( pi/4*(k-1), -1.1,   0, xlabels{k}, ...
            'HorizontalAlignment', 'center', ...
            'FontSize', 8 );
    end
        h.XLabel(i) = text( pi/2, -1.7,   0, ...
        '\omega_0', 'HorizontalAlignment', 'center' );
    h.YLabel(i) = ylabel( 'a', 'Position', ...
        [-0.25 0.2 -3/4], ...
        'FontSize', 10 );
    h.ZLabel(i) = text( -pi/3, 4/3, 2, 'S' );

    pos = get( gca, 'Position' );
    h.Title(i) = axes( ...
        'Position', [pos(1:2)+[-0.2 0.2] 0.15 0.05], ...
        'XTick', [], 'YTick', [], 'Box' ,'off', ...
        'XColor', 'w', 'YColor', 'w' );
    text( 0.1, 0.5, ['B = \pi /' num2str( L(i)/8 )] )
end
axes( h.Title(1) ) % bring to front
```

Figure 2.13

```
function [] = bw_stretching_lnE( NN )

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
colormap gray
xlabels = {'0','\pi/4','\pi/2','3\pi/4','\pi'};
zlabels = {'-3','0',' 3','6','\geq 9'};
omega = pi*[-NN:2:NN-1]'/NN;
L = 2.^[10 7 4];
for i = 1:length(L)
    B = 8*pi/L(i);        % The bandwidth
    phi = B/2;
```

```
a = [-7/8:7/NN:7/8]; % The range of warping factors
omega_0 = omega;      % The centre freq.
omega_0 = omega((omega>phi) & (omega<pi-phi));% Max/min freqs

    a = ones(size(omega_0,1),size(a,2))*diag(a);
    omega_0 = diag(omega_0)*ones(size(omega_0,1),size(a,2));
    U = {(1 - a.^2).*sin(omega_0 + phi)
     (1 + a.^2).*cos(omega_0 + phi) - 2*a};
    V = {(1 - a.^2).*sin(omega_0 - phi)
     (1 + a.^2).*cos(omega_0 - phi) - 2*a};
% The warped bandwidth
    Bhat = abs( atan2( U{:} ) - atan2( V{:} ) );
    % Bandwidth stretching factor
S = Bhat./B;
    Stilde = DIV( 1 - a.^2, ...
                1 - 2*a.*cos(omega_0) + a.^2 );
E = abs(Stilde - S)./S;
lnE = log( E ); lnE(lnE < -20) = NaN;

h.Axis(i) = axes( 'Position', [0.35 1-0.3*i 0.34 0.26] );
h.Surf(i) = surf( omega_0, a, lnE, ...
    'LineStyle', 'none' );
set( h.Axis(i), ...
    'XLim', [0 pi], ...
    'XTick', [0:pi/8:pi], ...
    'XTickLabel', [], ...
    'YLim', [-0.8 0.8], ...
    'ZLim', [-12 0], 'ZTick', [-12:6:0], ...
    'FontSize', 8, ...
    'CameraPosition', [12 3 109] )
ZLim = get( gca, 'ZLim' );
for k = 1:length( xlabels )
    h.XTickLabel(i,k) = ...
        text( pi/4*(k-1), 1.1, -12, xlabels{k}, ...
        'HorizontalAlignment', 'center', ...
        'FontSize', 8 );
end
    h.XLabel(i) = text( pi/2, 3/2, -12, ...
    '\omega_0', 'HorizontalAlignment', 'center' );
h.YLabel(i) = ylabel( 'a', 'Position', ...
    [9*pi/8 0 -16], ...
```

```
        'FontSize', 10 );
    h.ZLabel(i) = text( 5*pi/4, -3/2, -6, 'ln E' );

    pos = get( gca, 'Position' );
    h.Title(i) = axes( ...
        'Position', [pos(1:2)+[-0.2 0.2] 0.15 0.05], ...
        'XTick', [], 'YTick', [], 'Box' ,'off', ...
        'XColor', 'w', 'YColor', 'w' );
    text( 0.1, 0.5, ['B = \pi /' num2str( L(i)/8 )] )
end
axes( h.Title(1) ) % bring to front
```

Figure 2.14

```
function [] = sw_of_three_sinusiods( NN )

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 1.5], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 1.5], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 1.5] );
plot1    = axes( 'Position', [0.1 0.7  0.33 0.3 ] );
plot2    = axes( 'Position', [0.6 0.7  0.33 0.3 ] );
spectro1 = axes( 'Position', [0.1 0.15 0.33 0.55] );
spectro2 = axes( 'Position', [0.6 0.15 0.33 0.55] );

t = [0:NN/2-1]';
omega = pi*[-NN:2:NN-1]/NN;

omega_1 = 3*pi/8;
omega_2 = 6*pi/8;
omega_3 = 7*pi/8;

h_window = (1 - cos( 4*pi/NN*t ))/2;
f = [cos( omega_1*t ).*h_window
    (cos( omega_2*t ) + cos( omega_3*t )).*h_window];
t = [t;  (NN/2 + t)];

axes( plot1 )
plot( t, f )
```

```
axis( [0 NN -2.4 2.4] )
ylabel( 'Amplitude' )
axes( spectro1 )
SPECGRAM( f, 256 )
xlabel( '(a)' )

a = -1/2;
fhat = SW( f, THETA( omega, -a, 'all-pass' ) );

axes( plot2 )
plot( t, fhat )
axis( [0 NN -2.4 2.4] )
ylabel( 'Amplitude' )
axes( spectro2 )
SPECGRAM( fhat.', 256 )
xlabel( '(b)' )

step = 100*ceil( NN/500 );
set( plot1, 'XTick', [1:step:NN], 'XTickLabel', [], ...
    'YTickLabel', [], 'Box', 'off' );
set( plot2, 'XTick', [1:step:NN], 'XTickLabel', [], ...
    'YTickLabel', [], 'Box', 'off' );
set( spectro1, 'XTick', [1:step:NN], 'XTickLabel', [0:step:NN] );
set( spectro2, 'XTick', [1:step:NN], 'XTickLabel', [0:step:NN] );
```

Figure 2.15

```
function [] = ...
    time_domain_plot_spectrogram_of_SWing_unit_impulse( NN, MM )

set( gcf, ...
    'Position',     get( gcf, 'Position'       ).*[1 1 1 1.5], ...
    'PaperSize',    get( gcf, 'PaperSize'      ).*[1 1.5], ...
    'PaperPosition',get( gcf, 'PaperPosition'  ).*[1 1 1 1.5] );
plot1    = axes( 'Position', [0.3  0.7  0.43 0.3 ] );
spectro1 = axes( 'Position', [0.3  0.15 0.43 0.55] );

t = [0:MM-1]';
omegahat = pi*[-MM:2:MM-1]/MM;
```

```
a = 1/2;
omega = THETA( omegahat, -a, 'all-pass' );

N0 = NN/2;
s_min = (1 - abs(a))/(1 + abs(a));
s_max = (1 + abs(a))/(1 - abs(a));
N_first = floor( N0*s_min );
N_last = ceil( N0*s_max );

f = t==N0;
fhat = flipud( sw( f, omega ) );

axes( plot1 )
plot( t, fhat )
axis( [0 MM -1/4 1/4] )
% this line is where the orininal impulse is
line( [N0 N0], [-1 1], 'LineStyle', '--', 'Color', 'k' )
% this line is approximately at the first non-zero sample
line( [N_first N_first], [-1 1], 'LineStyle', ':', 'Color', 'r' )
% this line is approximately at the last non-zero sample
line( [N_last  N_last ], [-1 1], 'LineStyle', ':', 'Color', 'm' )
ylabel( 'Amplitude' )

axes( spectro1 )
SPECGRAM( fhat, 128 )

step = 100*ceil( MM/500 );
set( plot1, 'XTick', [1:step:MM], 'XTickLabel', [], ...
    'YTickLabel', [], 'Box', 'off' );

set( spectro1, 'XTick', [1:step:MM], ...
    'XTickLabel', [0:step:MM], 'XLim', [0 MM] );
```

Figure 2.16

```
function hf = columns_of_SW_matrix( N, M, MM )

% M = length of each column
% N = number of columns
```

```
c = MM/M;  % interpolate columns by a factor of c
d = 5;     % plot every dth column
m = [0:M-1];
n = [0:N-1]';
a = 1/8;

omegahat_m = pi*[-M:2:M-1]'/M;
omega_m = THETA( omegahat_m, -a, 'all-pass' );

H = ZTRANSFORM_MATRIX( omega_m, N );
Dinv = IDFT_MATRIX( M );
S = Dinv*H;
SS = UPSAMPLE( S, c );

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
axis( [0 N 0 M 0 1] )
set( gca, 'NextPlot', 'add', ...
    'Position', [0.13 0.195 0.775 0.733] )
patch( [N N -1 -1 N
        N N  N  N  N]', [M M  M  M  M
                         M M  0  0  M]', [0 1  1  0  0
                                          0 1  1  0  0]', ...
    reshape( [0.8 1]'*[1 1 0.6], 1, 2, 3 ), 'FaceAlpha', 0.4 )
set( gca, ...
    'XGrid', 'off', ...
    'XLim', [-1 N], ...
    'XTick', [0:d:N-1], ...
    'YLim', [0 M], ...
    'YTick', [mod(M,4):round(M/4):M], ...
    'YTickLabel', M-[mod(M,4):round(M/4):M], ...
    'ZLim', [0 1] ...
)
xlabel( 'column_index', ...
    'Position', [N/2 0 -0.15] )
ylabel( 'output\newlinesample_index', ...
    'Position', [0 M/2 -0.05] )
zlabel( 'value_of_matrix_element' )
```

```
mM = diag( ( [0:1/c:M-1/c] ) )*ones( MM, N );
nN = ones( MM, N )*diag( SELECT_EVERY( n, d ) );
plot3( nN, M - mM, SS, '-', 'LineWidth', 0.25 )
mM = diag( ( m ) )*ones( M, N );
nN = ones( M, N )*diag( SELECT_EVERY( n, d ) );
plot3( nN, M - mM, S, '.', 'MarkerSize', 10 )
```

Figure 2.17

```
function [] = ...
    phase_corrected_modified_sinc_interp_in_freq_dom( M, N, NN )

NN = lcm( [NN M N] ); % NN must be a multiple of both M and N
% the input sample locations
sN = sparse( ~mod( [0:NN-1].', NN/N ) );
% the output sample locations
sM = sparse( ~mod( [0:NN-1].', NN/M ) );

xtick = [-pi:pi/4:pi];
xticklabel = {'-\pi','-3\pi/4','-\pi/2','-\pi/4','0', ...
              '\pi/4','\pi/2','3\pi/4','\pi'};
ytick = [-pi:pi/2:pi];
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};

omega = pi*[-NN:2:NN-1].'./NN;  % the continuous frequency vector
f = F( omega );                 % the continuous time input signal
H = sinc_interp( omega, N, M ); % the interpolation function

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
axes( 'Position', [0.13 0.6 0.775 0.385] )

plot( omega, abs( H ), 'LineWidth', 1 );

set( gca, ...
    'Box', 'off', ...
    'XGrid', 'on', ...
```

```
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YLim', [0 1.01], ...
    'YTick', [0 1] ...
)
set( xlabel( '\omega' ), 'Position', [0 -0.144 0] )
ylabel( 'Magnitude', 'Position', [-3.8 0.505 0] )
for i = 1:length( xtick )
    text( xtick(i), -0.1, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end

axes( 'Position', [0.13 0.1 0.775 0.385] )

PLOT_WRAPPED( omega, angle( H ), 'LineWidth', 1 );

set( gca, ...
    'Box', 'off', ...
    'XGrid', 'on', ...
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YGrid', 'on', ...
    'YLim', [-pi pi], ...
    'YTick', ytick, ...
    'YTickLabel', [] ...
)
set( xlabel( '\omega' ), 'Position', [0 -4 0] )
ylabel( 'Phase Angle', 'Position', [-3.8 0 0] )
for i = 1:length( xtick )
    text( xtick(i), -pi-0.5, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
for i = 1:length( ytick )
    text( -pi-0.3, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end


% the original spectrum that was sampled
```

```
function F = F( omega );
F = exp(j.*pi*sin( omega )).*(1 + cos( omega ))/2;

% the phase-corrected modified sinc interpolation function
% (periodic in freq domain)
function H = sinc_interp( omega, N, M );
NN = length( omega );
H = ones( NN, 1 );
i = find( N*omega );
D = lcm( [M N] );
H(i) = exp(j*D*omega(i)) .* ...
    sin( D*omega(i)/2 )./(D*sin( omega(i)/2 ));
```

Figure 2.18

```
function [] = ...
    construct_op_samples_using_sinc_interp_in_freq_dom( M, N, NN )

NN = lcm( [NN M N] ); % NN must be a multiple of both M and N
% the input sample locations:
sN = sparse( ~mod( [0:NN-1].', NN/N ) );
% the output sample locations:
sM = sparse( ~mod( [0:NN-1].', NN/M ) );

xtick = [-pi:pi/4:pi];
xticklabel = {'-\pi','-3\pi/4','-\pi/2','-\pi/4','0', ...
              '\pi/4','\pi/2 ','3\pi/4','\pi'};
ytick = [-pi:pi/2:pi];
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};

omega = pi*[-NN:2:NN-1].'./NN;  % the continuous frequency vector
f = F( omega );                 % the continuous time input signal
H = sinc_interp( omega, N, M ); % the interpolation function
G = cconv( H, f.*sM );          % the interpolated function
G = [G(2:end); G(1)]; % we're one sample out

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
```

```
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
axes( 'Position', [0.13 0.6 0.775 0.385] )
hold on
plot( omega, abs( f ), 'b:', 'LineWidth', 1 )
plot( omega(sN), abs( f(sN) ), 'b.', 'MarkerSize', 12 )
plot( omega, abs( G ), 'r-', 'LineWidth', 1 )
plot( omega(sM), abs( G(sM) ), 'r*', 'MarkerSize', 6 )

set( gca, ...
    'Box', 'off', ...
    'XGrid', 'on', ...
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YLim', [0 1.01], ...
    'YTick', [0 1] ...
)
set( xlabel( '\omega' ), 'Position', [0 -0.144 0] )
ylabel( 'Magnitude', 'Position', [-3.8 0.505 0] )
for i = 1:length( xtick )
    text( xtick(i), -0.1, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end

axes( 'Position', [0.13 0.1 0.775 0.385] )
hold on
plot( omega, angle( f ), 'b:', 'LineWidth', 1 )
plot( omega(sN), angle( f(sN) ), 'b.', 'MarkerSize', 12 )
PLOT_WRAPPED( omega, angle( G ), 'Color', 'r', 'LineWidth', 1 )
plot( omega(sM), angle( G(sM) ), 'r*', 'MarkerSize', 6 )

set( gca, ...
    'XGrid', 'on', ...
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YGrid', 'on', ...
    'YLim', [-pi pi], ...
    'YTick', ytick, ...
    'YTickLabel', [] ...
)
```

```
set( xlabel( '\omega' ), 'Position', [0 -4 0] )
ylabel( 'Phase Angle', 'Position', [-3.8 0 0] )
for i = 1:length( xtick )
    text( xtick(i), -pi-0.5, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
for i = 1:length( ytick )
    text( -pi-0.3, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end


% the original spectrum that was sampled
function F = F( omega );
F = exp(j.*pi*sin( omega )).*(1 + cos( omega ))/2;

% the phase-corrected modified sinc interpolation function
% (periodic in freq domain)
function H = sinc_interp( omega, N, M );
NN = length( omega );
H = ones( NN, 1 );
i = find( N*omega );
D = lcm( [M N] );
H(i) = exp(j*D*omega(i)) .* ...
        sin( D*omega(i)/2 )./(D*sin( omega(i)/2 ));
```

Figure 2.19

```
function [] = ...
    construct_op_samples_with_simple_interp_in_freq_dom( M, N, NN )

NN = lcm( [NN M N] ); % NN must be a multiple of both M and N
% the input sample locations:
sN = sparse( ~mod( [0:NN-1].', NN/N ) );
% the output sample locations:
sM = sparse( ~mod( [0:NN-1].', NN/M ) );

xtick = [-pi:pi/4:pi];
xticklabel = {'-\pi','-3\pi/4','-\pi/2','-\pi/4','0', ...
```

```
                '\pi/4','\pi/2 ','3\pi/4','\pi'};
ytick = [-pi:pi/2:pi];
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};

omega = pi*[-NN:2:NN-1].'./NN;  % the continuous frequency vector
f = F( omega );                 % the continuous time input signal
H = interp_M4( omega, N );      % the interpolation function
G = cconv( H, f.*sM );          % the interpolated function
G = [G(2:end); G(1)];           % we're one sample out
G(sM(1)) = G(sM(1))*exp( -j*2*pi );  % makes the graph prettier

set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 2] );
axes( 'Position', [0.13 0.6 0.775 0.385] )
hold on
plot( omega, abs( f ), 'b:', 'LineWidth', 1 )
plot( omega(sN), abs( f(sN) ), 'b.', 'MarkerSize', 12 )
plot( omega, abs( G ), 'r-', 'LineWidth', 1 )
plot( omega(sM), abs( G(sM) ), 'r*', 'MarkerSize', 6 )

set( gca, ...
    'Box', 'off', ...
    'XGrid', 'on', ...
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YLim', [0 1.01], ...
    'YTick', [0 1] ...
)
set( xlabel( '\omega' ), 'Position', [0 -0.144 0] )
ylabel( 'Magnitude', 'Position', [-3.8 0.505 0] )
for i = 1:length( xtick )
    text( xtick(i), -0.1, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end

axes( 'Position', [0.13 0.1 0.775 0.385] )
hold on
plot( omega, angle( f ), 'b:', 'LineWidth', 1 )
```

```
plot( omega(sN), angle( f(sN) ), 'b.', 'MarkerSize', 12 )
PLOT_WRAPPED( omega, angle( G ), 'Color', 'r', 'LineWidth', 1 )
plot( omega(sM), angle( G(sM) ), 'r*', 'MarkerSize', 6 )

set( gca, ...
    'XGrid', 'on', ...
    'XLim', [-pi pi], ...
    'XTick', xtick, ...
    'XTickLabel', [], ...
    'YGrid', 'on', ...
    'YLim', [-pi pi], ...
    'YTick', ytick, ...
    'YTickLabel', [] ...
)
set( xlabel( '\omega' ), 'Position', [0 -4 0] )
ylabel( 'Phase Angle', 'Position', [-3.8 0 0] )
for i = 1:length( xtick )
    text( xtick(i), -pi-0.5, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
for i = 1:length( ytick )
    text( -pi-0.3, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end


% the original spectrum that was sampled
function F = F( omega );
F = exp(j.*pi*sin( omega )).*(1 + cos( omega ))/2;

% the phase-corrected modified sinc interpolation function
% (periodic in freq domain)
function H = interp_M4( omega, N );
H = cos( omega ).*(1 + cos( omega ))/2;
```

Figure 2.20

```
function [] = example_rows_of_interp_matrix( N, M, NN )
```

```
% N = length of each row
% M = number of rows
c = NN/N;  % interpolte rows by a factor of c
d = 18;     % plot every dth row
m = [0:M-1];
n = [0:N-1]';
a = 1/8;

omegahat_m = pi*[-M:2:M-1]'/M;
omega_m = THETA( omegahat_m, -a, 'all-pass' );
omegahat_n = pi*[-N:2:N-1]'/N;

H = ZTRANSFORM_MATRIX( omega_m, N );
Dinv = IDFT_MATRIX( N );
C = H*Dinv;
CC = UPSAMPLE( C.', c ).';

set( gcf, ...
    'Position',      get( gcf, 'Position'       ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'      ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition'  ).*[1 1 1 2] );
axis( [0 N 0 M 0 1] )
set( gca, 'NextPlot', 'add', 'Position', [0.16 0.195 0.775 0.733] )
patch( [N N -1 -1 N
        N N  N  N  N]', [M M  M  M  M
                         M M  0  0  M]', [0 1  1  0  0
                                          0 1  1  0  0]', ...
    reshape( [0.8 1]'*[1 1 0.6], 1, 2, 3 ), 'FaceAlpha', 0.4 )

set( gca, ...
    'XGrid', 'off', ...
    'XLim', [-1 N], ...
    'YLim', [0 M], ...
    'YTick', [mod( M, d ):d:M], ...
    'YTickLabel', M-[mod( M, d ):d:M], ...
    'ZLim', [0 1] ...
)
xlabel( ['index of sample\newline' ...
         'in unwarped\newline' ...
         'freq domain\newline' ...
         '(column index)'], ...
```

```
    'Position', [N/2 0 -0.08] )
ylabel( ['index of sample\newline' ...
         'in warped freq domain\newline' ...
         '(row index)'], ...
    'Position', [-1 M/4 -0.05] )

nN = ones( M, NN )*diag( ( [0:1/c:N-1/c] ) );
mM = diag( SELECT_EVERY( m, d ) )*ones( M, NN );
plot3( nN.', M - mM.', real( CC.' ), '-', 'LineWidth', 0.25 )
plot3( nN.', M - mM.', imag( CC.' ), ':', 'LineWidth', 0.25 )
nN = ones( M, N )*diag( ( n ) );
mM = diag( SELECT_EVERY( m, d ) )*ones( M, N );
plot3( nN.', M - mM.', real( C.' ), '.', 'MarkerSize', 10 )
plot3( nN.', M - mM.', imag( C.' ), 'o', 'MarkerSize', 3 )

h = zlabel( ['contribution of input sample' 10 ...
             'towards output sample'], ...
            'Position', [-81.7 -155 5] );
get( h )
```

Figure 2.21

```
function [] = one_row_of_interp_matrix( N, M, NN )

% N = length of each row
% M = number of rows
c = NN/N;  % interpolte rows by a factor of c
d = 13;     % plot the dth row
a = 1/8;

omegahat_m = pi*[-M:2:M-1]'/M;
omega_m = THETA( omegahat_m, -a, 'all-pass' );
omegahat_n = pi*[-N:2:N-1]'/N;

H = ZTRANSFORM_MATRIX( omega_m, N );
Dinv = IDFT_MATRIX( N );
C = H*Dinv;
CC = UPSAMPLE( C.', c ).';
```

```
ytick = [-pi:pi/2:pi];
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};
set( gcf, ...
    'Position',      get( gcf, 'Position'       ).*[1 1 1 2], ...
    'PaperSize',     get( gcf, 'PaperSize'      ).*[1 2], ...
    'PaperPosition', get( gcf, 'PaperPosition'  ).*[1 1 1 2] );

axes( 'Position', [0.13 0.6 0.775 0.385] )
set( gca, ...
    'XLim', [0 N], ...
    'YLim', [0 1.01], ...
    'YTick', [0 1] ...
)
ylabel( 'Magnitude', 'Position', [-0.102*N 0.505 0] )
hold on

plot( [0:NN-1]/c, abs( CC(d,:) ), ':', 'LineWidth', 0.5 )
plot( [0:N-1], abs( CC(d,1:c:end) ), '.', 'MarkerSize', 12 )

axes( 'Position', [0.13 0.1 0.775 0.385] )
set( gca, ...
    'XLim', [0 N], ...
    'YGrid', 'on', ...
    'YLim', [-pi pi], ...
    'YTick', ytick, ...
    'YTickLabel', [] ...
)
ylabel( 'Phase Angle', 'Position', [-0.102*N 0 0] )
for i = 1:length( ytick )
    text( -0.046*N, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
xlabel( 'Column Index' )
hold on

plot( [0:NN-1]/c, angle( CC(d,:) ), ':', 'LineWidth', 0.5 )
plot( [0:N-1], angle( CC(d,1:c:end) ), '.', 'MarkerSize', 12 )
```

Figure 2.22

```
function [] = comparing_interp_filters1( N, M )

N = 32;
M = 256;
n = [0:N-1];
a = 1/8;
rows = ceil( [1:4]/5*M );

% prepare axes
ax_width = 0.42;   ax_height = 0.16;
ax_x = [0.1  0.56]; ax_y = [0.8  0.58  0.3  0.08];
ytick = [-pi:pi/2:pi];
ylim_mag = [0  1.01];
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};
set( gcf, ...
    'Position',     get( gcf, 'Position'      ).*[1 1 1.25 4], ...
    'PaperSize',    get( gcf, 'PaperSize'     ).*[1.25 4], ...
    'PaperPosition',get( gcf, 'PaperPosition' ).*[1 1 1.25 4] );

h(1,1).axis = axes( 'Position', ...
    [ax_x(1) ax_y(1) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', [0 1] )
ylabel( 'Magnitude', 'Position', [-4 0.505 0] )
hold on

h(1,2).axis = axes( 'Position', ...
    [ax_x(2) ax_y(1) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, ...
    'YTick', [0 1], 'YTickLabel', [] )
hold on

h(2,1).axis = axes( 'Position', ...
    [ax_x(1) ax_y(2) ax_width ax_height] );
set( gca, 'XLim', [0 N], ...
    'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
ylabel( 'Phase_Angle', 'Position', [-4 0 0] )
for i = 1:length( ytick )
    text( -2, ytick(i), yticklabel{i}, ...
```

```
        'HorizontalAlignment', 'center' )
end
hold on

h(2,2).axis = axes( 'Position', ...
    [ax_x(2) ax_y(2) ax_width ax_height] );
set( gca, 'XLim', [0 N], ...
    'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
hold on


h(3,1).axis = axes( 'Position', ...
    [ax_x(1) ax_y(3) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', [0 1] )
ylabel( 'Magnitude', 'Position', [-4 0.505 0] )
hold on

h(3,2).axis = axes( 'Position', ...
    [ax_x(2) ax_y(3) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, ...
    'YTick', [0 1], 'YTickLabel', [] )
hold on

h(4,1).axis = axes( 'Position', ...
    [ax_x(1) ax_y(4) ax_width ax_height] );
set( gca, 'XLim', [0 N], ...
    'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
ylabel( 'Phase_Angle', 'Position', [-4 0 0] )
for i = 1:length( ytick )
    text( -2, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
hold on

h(4,2).axis = axes( 'Position', ...
    [ax_x(2) ax_y(4) ax_width ax_height] );
set( gca, 'XLim', [0 N], ...
    'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
```

```
hold on

% calculate matrices
omegahat_m = pi*[-M:2:M-1]'/M;
omegahat_n = pi*[-N:2:N-1]'/N;
Dinv = IDFT_MATRIX( N );

omega_m = THETA( omegahat_m, -a, 'all-pass' );
H = ZTRANSFORM_MATRIX( omega_m, N );
C.sinclike = H*Dinv;

omega_m = THETA( omegahat_m, {
    '(2*omega - pi)/3' [-pi   -pi/4]
    '2*omega'          [-pi/4  pi/4]
    '(2*omega + pi)/3' [pi/4   pi]
    }, 'piecewise_linear' );
H = ZTRANSFORM_MATRIX( omega_m, N );
C.linear = H*Dinv;

% plot rows of C
for k = [0 2]
    for i = 1:2

axes( h(k+1,i).axis )
h(k+1,i).sinclike = plot( n, abs( C.sinclike(rows(k+i),:)) );
h(k+1,i).linear   = plot( n, abs( C.linear(rows(k+i),:)  ) );
[num,den] = rat( omegahat_m(rows(k+i))/pi );
title( ['Row ' num2str( rows(k+i) ) '                  '] );
text( 16, 1.15, ['(\omega = ' num2str( num ) ...
                 '\pi/' num2str( den ) ')'] )
text( 16.8, 1.18, '\^' )
axes( h(k+2,i).axis )
h(k+2,i).sinclike = ...
    PLOT_WRAPPED( n, angle( C.sinclike(rows(k+i),:)) );
% avoid plotting phase where the amplitude is negligible
l = ones( size( C.linear(rows(k+i),:) ) );
l(abs( C.linear(rows(k+i),:) )<1e-6 ) = NaN;
h(k+2,i).linear  = ...
    PLOT_WRAPPED( n, angle( C.linear(rows(k+i),:) ).*l );

    end
```

```
end

% set plot attributes
for i = 1:8
    set( h(i).sinclike(1), 'LineStyle', ':', ...
        'LineWidth', 1, 'Marker', 'x', 'MarkerSize', 5 )
    set( h(i).linear(1),   'LineStyle', ':', ...
        'LineWidth', 1, 'Marker', '.' )
    axes( h(i).axis )
    if ~mod( i, 2 ) xlabel( 'Column Index' ); end
end

% draw legends
legend_positions = {[0.11 0.87 0.25 0.1], [0.74 0.36 0.25 0.1]};
for i = 1:2
        hl = FLOATING_LEGEND( legend_positions{i}, ...
        {{':x'} 'all-pass (sinc-like)'
         {':.'} ' \newlinepiece-wise\newlinelinear'}, ...
        'XColor', 'w', 'YColor', 'w' );
        hlc = get( hl, 'Children' );
        set( hlc(1), 'FontSize', 8 )
        set( hlc(4), 'FontSize', 8 )
end
```

Figure 2.23

```
function [] = comparing_interp_filters2( N, M )

N = 32;
M = 256;
n = [0:N-1];
a = 1/8;
rows = ceil( [1:4]/5*M );

% prepare axes
ax_width = 0.42;   ax_height = 0.16;
ax_x = [0.1 0.56]; ax_y = [0.8 0.58 0.3 0.08];
ytick = [-pi:pi/2:pi];
ylim_mag = [0 1.01]; ytick_mag = [0 1];
```

```
yticklabel = {'-\pi','-\pi/2','0','\pi/2','\pi'};
set( gcf, ...
    'Position',     get( gcf, 'Position'      ).*[1 1 1.25 4], ...
    'PaperSize',    get( gcf, 'PaperSize'     ).*[1.25 4], ...
    'PaperPosition',get( gcf, 'PaperPosition' ).*[1 1 1.25 4] );

h(1,1).axis = ...
    axes( 'Position', [ax_x(1) ax_y(1) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', ytick_mag )
ylabel( 'Magnitude', 'Position', [-4 0.505 0] )
hold on

h(1,2).axis = ...
    axes( 'Position', [ax_x(2) ax_y(1) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', ytick_mag, ...
    'YTickLabel', [] )
hold on

h(2,1).axis = ...
    axes( 'Position', [ax_x(1) ax_y(2) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
ylabel( 'Phase_Angle', 'Position', [-4 0 0] )
for i = 1:length( ytick )
    text( -2, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
hold on

h(2,2).axis = ...
    axes( 'Position', [ax_x(2) ax_y(2) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
hold on


h(3,1).axis = ...
    axes( 'Position', [ax_x(1) ax_y(3) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', ytick_mag )
ylabel( 'Magnitude', 'Position', [-4 0.505 0] )
hold on
```

```
h(3,2).axis = ...
    axes( 'Position', [ax_x(2) ax_y(3) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YLim', ylim_mag, 'YTick', ytick_mag, ...
    'YTickLabel', [] )
hold on

h(4,1).axis = ...
    axes( 'Position', [ax_x(1) ax_y(4) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
ylabel( 'Phase Angle', 'Position', [-4 0 0] )
for i = 1:length( ytick )
    text( -2, ytick(i), yticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end
hold on

h(4,2).axis = ...
    axes( 'Position', [ax_x(2) ax_y(4) ax_width ax_height] );
set( gca, 'XLim', [0 N], 'YGrid', 'on', 'YLim', [-pi pi], ...
    'YTick', ytick, 'YTickLabel', [] )
hold on

% calculate matrices
omegahat_m = pi*[-M:2:M-1]'/M;
omegahat_n = pi*[-N:2:N-1]'/N;
Dinv = IDFT_MATRIX( N );

omega_m = THETA( omegahat_m, -a, 'all-pass' );
H = ZTRANSFORM_MATRIX( omega_m, N );
C.sinclike = H*Dinv;

C.linear = LINEAR_INTERP_MATRIX( omega_m, N );

% plot rows of C
for k = [0 2]
    for i = 1:2

axes( h(k+1,i).axis )
h(k+1,i).sinclike = plot( n, abs( C.sinclike(rows(k+i),:)) );
```

```
h(k+1,i).linear   = plot( n, abs( C.linear(rows(k+i),:)  ) );
[num,den] = rat( omegahat_m(rows(k+i))/pi );
title( ['Row ' num2str( rows(k+i) ) '                    '] );
text( 16, 1.15, ['(\omega = ' num2str( num ) ...
                 '\pi/' num2str( den ) ')'] )
text( 16.8, 1.18, '\^' )
axes( h(k+2,i).axis )
h(k+2,i).sinclike = ...
    PLOT_WRAPPED( n, angle( C.sinclike(rows(k+i),:)) );
h(k+2,i).linear   = ...
    PLOT_WRAPPED( n, angle( C.linear(rows(k+i),:) ) );

    end
end

% set plot attributes
for i = 1:8
    set( h(i).sinclike(1), 'LineStyle', ':', 'LineWidth', 1, ...
        'Marker', 'x', 'MarkerSize', 5 )
    set( h(i).linear(1),   'LineStyle', ':', 'LineWidth', 1, ...
        'Marker', '.' )
    axes( h(i).axis )
    if ~mod( i, 2 ) xlabel( 'Column Index' ); end
end

% draw legends
legend_positions = {[0.11  0.92   0.25  0.05 ]
                    [0.68  0.425  0.25  0.045]};
for i = 1:2
        hl = FLOATING_LEGEND( legend_positions{i}, ...
        {{':x'} 'sinc-like (all-pass)'
         {':.'} 'linear'}, ...
        'XColor', 'w', 'YColor', 'w' );
        hlc = get( hl, 'Children' );
        set( hlc(1), 'FontSize', 8 )
        set( hlc(4), 'FontSize', 8 )
end
```

Figure 2.24

```
function [] = comparing_interp_filters3( N, M )

a = 1/4;

% prepare axes
set( gcf, ...
    'Position',     get( gcf, 'Position'      ).*[1 1 1.25 3], ...
    'PaperSize',    get( gcf, 'PaperSize'     ).*[1.25 3], ...
    'PaperPosition',get( gcf, 'PaperPosition' ).*[1 1 1.25 3] );
ax_width  = 0.35;
ax_height = 0.21;
ax_x = [0.13 0.55];
ax_y = [0.75 0.52 0.28 0.04];
xtick = [0:pi/4:pi];
xticklabel = {'0', '\pi/4','\pi/2','3\pi/4','\pi'};

for i = 1:4
        h(i,1).axis = axes( 'Position', ...
        [ax_x(1) ax_y(i) ax_width ax_height] );
        set( gca, 'XLim', [0 M], 'XTickLabel', [], ...
        'YLim', [-1.2 1.2], 'YTick', [-1:1:1] )
        hold on
end
set( gca, 'XTickLabel', get( gca, 'XTick' ) )
for i = 1:4
        h(i,2).axis = axes( 'Position', ...
        [ax_x(2) ax_y(i) ax_width ax_height] );
        set( gca, 'XLim', [0 pi], 'XTick', xtick, ...
        'XTickLabel', [], 'YLim', [0 M/16], ...
        'YTick', [0:M/64:M/16-1] )
        hold on
end
for i = 1:length( xtick )
    text( xtick(i), -M/128, xticklabel{i}, ...
        'HorizontalAlignment', 'center' )
end

% calculate matrices
omegahat_m = pi*[-M:2:M-1]'/M;
```

```
omegahat_n = pi*[-N:2:N-1]'/N;
Dinv_M = IDFT_MATRIX( M );
D_N = DFT_MATRIX( N );

omega_m = THETA( omegahat_m, -a, 'all-pass' );
H = ZTRANSFORM_MATRIX( omega_m, N );
S.sinclike = real( Dinv_M*H );
omega_m = THETA( omegahat_m, {
    '(3*omega_+_pi)/2' [-pi   -pi/2]
    'omega/2'          [-pi/2  pi/2]
    '(3*omega_-_pi)/2' [ pi/4  pi  ]
}, 'piecewise_linear' );
% the inverse functn is: (2*omega - pi)/3 [-pi   -pi/4]
%                         2*omega         [-pi/4  pi/4]
%                        (2*omega + pi)/3 [ pi/4  pi  ]
H = ZTRANSFORM_MATRIX( omega_m, N );
S.linear_warp = real( Dinv_M*H );
omega_m = THETA( omegahat_m, -a, 'all-pass' );
S.linear_interp = ...
    real( Dinv_M*LINEAR_INTERP_MATRIX( omega_m, N )*D_N );

% generate test signal and warp
t = [0:N/2-1]';
omega_1 = 3*pi/8;
omega_2 = 6*pi/8;
WINDOW = (1 - cos( 4*pi/N*t ))/2;
f = [cos( omega_1*t ).*WINDOW; cos( omega_2*t ).*WINDOW];
t = [t;  (N/2 + t)];

g.sinclike      = S.sinclike*f;
g.linear_warp   = S.linear_warp*f;
g.linear_interp = S.linear_interp*f;

% plot
axes( h(1,1).axis )
plot( t, f )
hlab = ylabel( 'unwarped' );
pos = get( hlab, 'Position' );
set( hlab, 'Position', pos.*[1.3 1 1] )
title( 'Time-Domain_Representation' )
axes( h(1,2).axis )
```

```
plot( omegahat_n, fftshift( abs( fft( f ) ) ) )
title( 'Frequency-Domain Representation' )

axes( h(2,1).axis )
plot( [0:M-1], g.sinclike )
hlab = ylabel( 'all-pass warp\newlinefunction' );
pos = get( hlab, 'Position' );
set( hlab, 'Position', pos.*[1.6 1 1] )
axes( h(2,2).axis )
plot( omegahat_m, fftshift( abs( fft( g.sinclike ) ) ) )

axes( h(3,1).axis)
plot( [0:M-1], g.linear_interp )
hlab = ylabel( 'linear_interp.\newlinefunction' );
pos = get( hlab, 'Position' );
set( hlab, 'Position', pos.*[1.6 1 1] )
axes( h(3,2).axis )
plot( omegahat_m, fftshift( abs( fft( g.linear_interp ) ) ) )

axes( h(4,1).axis )
plot( [0:M-1], g.linear_warp )
hlab = ylabel( 'piecewise-linear\newlinewarp_function' );
pos = get( hlab, 'Position' );
set( hlab, 'Position', pos.*[1.6 1 1] )
axes( h(4,2).axis )
plot( omegahat_m, fftshift( abs( fft( g.linear_warp ) ) ) )

% the enery of all the signals is approximately equal
disp( ['Energy of original      signal = ' ...
        num2str( sum( abs( f ) ) )] )
disp( ['Energy of sinc-like     signal = ' ...
        num2str( sum( abs( g.sinclike ) ) )] )
disp( ['Energy of linear warp   signal = ' ...
        num2str( sum( abs( g.linear_warp ) ) )] )
disp( ['Energy of linear_interp signal = ' ...
        num2str( sum( abs( g.linear_interp ) ) )] )
```

Figure 2.25

```
function [] = comparing_interp_filters4( N, M )

a = 1/4;

% prepare axes
set( gcf, ...
    'Position',     get( gcf, 'Position'      ).*[1 1 1.25 3], ...
    'PaperSize',    get( gcf, 'PaperSize'     ).*[1.25 3], ...
    'PaperPosition',get( gcf, 'PaperPosition' ).*[1 1 1.25 3] );

% calculate matrices
omegahat_m = pi*[-M:2:M-1]'/M;
omegahat_n = pi*[-N:2:N-1]'/N;
Dinv_M = IDFT_MATRIX( M );
D_N = DFT_MATRIX( N );

omega_m = THETA( omegahat_m, -a, 'all-pass' );
H = ZTRANSFORM_MATRIX( omega_m, N );
S.sinclike = real( Dinv_M*H );
omega_m = THETA( omegahat_m, {
   '(3*omega + pi)/2' [-pi   -pi/2]
   'omega/2'          [-pi/2  pi/2]
   '(3*omega - pi)/2' [ pi/4  pi  ]
}, 'piecewise linear' );
% the inverse functn is: (2*omega - pi)/3 [-pi   -pi/4]
%                         2*omega         [-pi/4  pi/4]
%                        (2*omega + pi)/3 [ pi/4  pi  ]
H = ZTRANSFORM_MATRIX( omega_m, N );
S.linear_warp = real( Dinv_M*H );
omega_m = THETA( omegahat_m, -a, 'all-pass' );
S.linear_interp = ...
    real( Dinv_M*LINEAR_INTERP_MATRIX( omega_m, N )*D_N );

% generate test signal and warp
t = [0:N/2-1]';
omega_1 = 3*pi/8;
omega_2 = 6*pi/8;
WINDOW = (1 - cos( 4*pi/N*t ))/2;
f = [cos( omega_1*t ).*WINDOW; cos( omega_2*t ).*WINDOW];
```

```
t = [t;  (N/2 + t)];

g.sinclike      = S.sinclike*f;
g.linear_warp   = S.linear_warp*f;
g.linear_interp = S.linear_interp*f;

% plot
plot_title = {'(a)' '(b)' '(c)' '(d)'};
func = {f g.sinclike g.linear_interp g.linear_warp};
for i = 1:4
    subplot( 2, 2, i );
    SPECGRAM( func{i}, M/8 )
    xlabel( plot_title{i} )
end
```

Figure 2.26

```
function [] = ...
    result_of_SWing_unit_impulse_using_few_op_samples( NN )

MM = NN;
set( gcf, ...
    'Position',      get( gcf, 'Position'      ).*[1 1 1 1.5], ...
    'PaperSize',     get( gcf, 'PaperSize'     ).*[1 1.5], ...
    'PaperPosition', get( gcf, 'PaperPosition' ).*[1 1 1 1.5] );
plot1    = axes( 'Position', [0.3  0.7  0.43 0.3 ] );
spectro1 = axes( 'Position', [0.3  0.15 0.43 0.55] );

t = [0:MM-1]';
omegahat = pi*[-MM:2:MM-1]/MM;
a = 1/2;
omega = THETA( omegahat, -a, 'all-pass' );

f = t==NN/2;
fhat = flipud( sw( f, omega ) );

axes( plot1 )
plot( t, fhat )
axis( [0 MM -1/4 1/4] )
```

```
ylabel( 'Amplitude' )
axes( spectro1 )
SPECGRAM( fhat, 128 )

step = 100*ceil( MM/500 );
set( plot1, 'XTick', [1:step:MM], 'XTickLabel', [], ...
    'YTickLabel', [], 'Box', 'off' );
set( spectro1, 'XTick', [1:step:MM], ...
    'XTickLabel', [0:step:MM], 'XLim', [0 MM] );
```

Figure 4.4

```
function [] = chirp_z( NN )

ha = ZPLANEAX( gcf, 'middle', 1.25 );

omega = 2*pi*[-NN:2:NN-1]/NN;
z = exp( j*omega );
plot( z, 'k' )

omega_0 = pi/6;
A_0 = 4/5;

phi_0 = -pi/4;
W_0 = 1 + 1/8;

A = A_0*exp( j*omega_0 );
W = W_0*exp( j*phi_0 );

M = 8;
k = [0:M-1];
z_k = A.*W.^(-k);

n = [0:NN-1]';
phi_0 = phi_0*M/NN;
W_0 = 1 + (W_0 - 1)*M/NN;
W = W_0*exp( j*phi_0 );
z = A.*W.^(-n);
```

```
plot( z,   'b:' )
plot( z_k, 'b.', 'MarkerSize', 12 )

line( [0 real( z_k(1) )], [0 imag( z_k(1) )], ...
    'Color', 'k', 'LineStyle', '-' )
line( [0 real( z_k(2) )], [0 imag( z_k(2) )], ...
    'Color', 'k', 'LineStyle', '-' )

% draw arrow
z = (3/5*exp( j*pi/6 )) .* exp( -j*pi/4 ).^omega;
plot( z(angle( z ) <= angle( z_k(2) ) & ...
        angle( z ) >= angle( z_k(1) )), 'k' )
z = (3/5*exp( j*5*pi/12 )) .* (9/8*exp( -j*pi/4 )).^omega;
plot( z(angle( z ) >= angle( z_k(2) ) - pi/12 & ...
        angle( z ) <= angle( z_k(2) )), 'k' )
z = (3/5*exp( j*5*pi/12 )) .* (7/8*exp( -j*pi/4 )).^omega;
plot( z(angle( z ) >= angle( z_k(2) ) - pi/12 & ...
        angle( z ) <= angle( z_k(2) )), 'k' )
z = 4/9*exp( j*3*pi/8 );
text( real( z ), imag( z ), '\phi_0', 'FontSize', 8 )

% draw arrow
z = (2/3*exp( j*pi/6 )) .* exp( j*pi/4 ).^omega;
plot( z(angle( z ) <= angle( z_k(1) ) & ...
        angle( z ) >= 0), 'k' )
z = (2/3*exp( j*pi/6 )) .* (9/8*exp( -j*pi/4 )).^omega;
plot( z(angle( z ) >= angle( z_k(1) ) - pi/12 & ...
        angle( z ) <= angle( z_k(1) )), 'k' )
z = (2/3*exp( j*pi/6 )) .* (7/8*exp( -j*pi/4 )).^omega;
plot( z(angle( z ) >= angle( z_k(1) ) - pi/12 & ...
        angle( z ) <= angle( z_k(1) )), 'k' )
z = 1/2*exp( j*pi/12 );
text( real( z ), imag( z ), '\theta_0', 'FontSize', 8 )
```

Figure 4.5

```
function [] = chirp_z_on_unit_circle( NN )

ha = zplaneax( gcf, 'middle', 1.25 );
```

```
omega = 2*pi*[-NN:2:NN-1]/NN;
z = exp( j*omega );
plot( z, 'k' )

omega_0 = pi/4;
A_0 = 1;

phi_0 = -pi/16;
W_0 = 1;

A = A_0*exp( j*omega_0 );
W = W_0*exp( j*phi_0 );

M = 8;
k = [0:M-1];
z_k = A.*W.^(-k);

n = [0:NN-1]';
phi_0 = phi_0*M/NN;
W_0 = 1 + (W_0 - 1)*M/NN;
W = W_0*exp( j*phi_0 );
z = A.*W.^(-n);

plot( z, 'b:' )
plot( z_k, 'b.', 'MarkerSize', 12 )
```

References

[1] Alan V. Oppenheim and Donald H. Johnson, "Discrete representation of signals", *Proceedings of the IEEE*, vol. 60, no. 6, pp. 681–691, June 1972.

[2] Alan V. Oppenheim, Donald H. Johnson, and Kenneth Steiglitz, "Computation of spectra with unequal resolution using the fast Fourier transform", *Proceedings of the IEEE*, vol. 59, no. 2, pp. 299–301, February 1971.

[3] Alan V. Oppenheim and Alan S. Willsky with Syed Hamid Nawab, *Signals and Systems*, McGraw-Hill, Singpore, 1998.

[4] Wikipedia, "Mixed-signal integrated circuit — wikipedia, the free encyclopedia", `http://en.wikipedia.org/w/index.php?title=Mixed-signal_integrated_circuit&oldid=121727621`, 2007, [Online; accessed 2-June-2007].

[5] Douglas A. Pucknell and Kamran Eshraghian, *Basic VLSI Design*, Prentice Hall, Sydney, third edition, 1994.

[6] Edward O. Wilson, "Cemical communication in the social insects", *Science*, vol. 149, pp. 1064–1071, September 1965.

[7] Charles L. Byrne, *Signal Processing: A Mathematical Approach*, A K Peters, Wellesley, Massachusetts, 2005.

[8] Peter V. O'Neill, *Advanced Engineering Mathematics*, Thomson Brooks/Cole, Pacific Grove, California, fifth edition, 2003.

[9] Claude E. Shannon, "Communication in the presence of noise", *Proc. IRE*, vol. 37, pp. 10–21, 1949.

[10] John M. Whittaker, "The Fourier theory of the cardinal functions", *Proc. Edinburgh Mathematical Society*, vol. 1, pp. 169–176, 1929.

[11] Vladimir A. Kotel'nikov, "On the transmission capacity of "ether" and wire in electrocommunications", *Izd. Red. Upr. Svyazzi RKKA*, 1933.

[12] Micheal Unser, "Sampling — 50 years after Shannon", *Proceedings of the IEEE*, vol. 88, no. 4, pp. 569–587, April 2000.

[13] Sanjit K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, McGraw-Hill, Singpore, 1998.

[14] Wikipedia, "Hadamard transform — wikipedia, the free encyclopedia", `http://en.wikipedia.org/w/index.php?title=Hadamard_transform&oldid=133198004`, 2007, [Online; accessed 2-June-2007].

[15] Olivier Rioul and Martin Vetterli, "Wavelets and signal processing", *Signal Processing Magazine, IEEE*, pp. 14–38, October 1991.

[16] Chi-Tsong Chen, *Signals and Systems*, Oxford University Press, New York, third edition, 2004.

[17] Serge N. Demidenko and Kenneth Lever, "Analysis of the frequency characteristics of a digital spectral warping network", in *International Workshop on Design Methodologies for Microelectronics and Signal Processing*, Gliwice-Cracow, Poland, October 1993, pp. 397–401.

[18] Warwick P. M. Allen, Donald G. Bailey, Serge N. Demidenko, and Vincenzo Piuri, "Analysis and implementation of a spectral warping network", in *International Symposium on Integrated Circuits, Devices & Systems*, Singapore, 2001, pp. 492–496.

[19] Donald E. Knuth, *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*, Addison-Wesley, Massachusetts, third edition, 1997.

[20] Sonali Bagchi and Sanjit K. Mitra, *The Nonuniform Discrete Fourier Transform and its Applications in Signal Processing*, Kluwer Academic Publishers, Massachusetts, 1999.

[21] Wikipedia, "Vandermonde matrix — wikipedia, the free encyclopedia", `http://en.wikipedia.org/w/index.php?title=Vandermonde_matrix&oldid=130967721`, 2007, [Online; accessed 2-June-2007].

[22] Donald G. Bailey, Warwick P. M. Allen, and Serge N. Demidenko, "Spectral warping revisited", in *International Workshop on Electronic Design, Test, and Applications (DELTA)*, Perth, Australia, January 2004, pp. 23–28.

[23] Warwick P. M. Allen, Donald G. Bailey, Serge N. Demidenko, and Vincenzo Piuri, "Frequency response analysis using spectral warping chirp signals", in *International Workshop on Electronic Design, Test, and Applications (DELTA)*, Christchurch, New Zealand, January 2002, pp. 492–495.

[24] Sonali Bagchi and Sanjit K. Mitra, "The nonuniform discrete Fourier transform and its applications in filter design: Part I — 1-D", *IEEE Transactions on Circuits and Systems — II: Analog and Digital Signal Processing*, vol. 43, no. 6, pp. 422–433, June 1996.

[25] Anamitra Makur and Sanjit K. Mitra, "Warped discrete-Fourier transform: Theory and applications", *IEEE Transactions on Circuits and Systems — I: Fundmental Theory and Applications*, vol. 48, no. 9, pp. 1086–1093, September 2001.

[26] Ian M. Wilson, "Analog behavioral modeling using PSPICE", in *Proc. of the 32nd Midwest Symposium on Circuits and Systems*, Champaign, Illinois, August 1989, vol. 2, pp. 981–984.

[27] M. Soma, "Mixed-signal fault models and design for test (I)", in *Proc. IEEE Asian Test Symposium*, November 1996.

[28] C. L. Wey, "Mixed-signal fault models and design for test (II)", in *Proc. IEEE Asian Test Symposium*, November 1996.

[29] "Special issue on the IEEE international mixed signal testing", *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 9, no. 1/2, 2001.

[30] "Special issue on the IEEE international mixed signal testing workshop (IMSTW)", *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 17, no. 5, 2001.

[31] "Analog and mixed-signal boundary-scan: A guide to the IEEE 1149.4 test standard", in *Frontiers in Electronic Testing*, A. Osserian, Ed., Boston, USA, 1999, vol. 16, Kluwer Academic Publishers.

[32] M. Bushnell and V. Agarwal, "Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits", in *Frontiers in Electronic Testing*, Boston, USA, 2000, vol. 17, Kluwer Academic Publishers.

[33] G. W. Roberts and A. K. Lu, *Analog Signal Generation for Built-In Self-Test of Mixed-Signal Integrated Circuits*, Kluwer Academic Publishers, Dordrecht, 1995.

[34] Matthew Mahoney, *DSP-Based Testing of Analog and Mixed-Signal Circuits*, IEEE Computer Society Press, September 1997.

[35] Sonali Bagchi and Sanjit K. Mitra, "The nonuniform disctrete Fourier transform and its applications in filter design: Part II — 2-D", *IEEE*

Transactions on Circuits and Systems — II: Analog and Digital Signal Processing, vol. 43, no. 6, pp. 434–444, June 1996.

[36] Sanjit K. Mitra, O. V. Shentov, and M. R. Petraglia, "A method for fast approximate computation of discrete time transforms", in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, April 1990, vol. 4, pp. 2025–2028.

[37] Gerald Goertzel, "An algorithm for evaluation of finite trigonometric series", *American Mathematical Monthly*, vol. 65, pp. 34–35, January 1958.

[38] L.R. Rabiner, R. W. Schafer, and C. M. Rader, "The chirp z-transform algorithm", *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 2, pp. 86–92, June 1969.

[39] J. Holub, R. Šmíd, and J. Vedral, "A new time-saving method for ADC testing", in *Measurement '99 Conference*, Bratislava, Slovak Republic, July 1999, pp. 55–58.

[40] J. Holub, R. Šmíd, and J. Vedral, "Chirp based method for ADC testing: Simulation and evaluation", in *Advanced A/D and D/A Conversion Techniques and their Applications*, Glasgow, UK, July 1999, pp. 94–96.

[41] "IEEE standard terminology and test methods for analog-to-digital converters", 2000, IEEE Standard.

[42] C. Marven and G. Ewers, *A Simple Approach to Digital Signal Processing*, John Wiley, New York, 1996.

[43] Matti Karjalainen, Aki Härmä, and Unto K. Laine, "Realizable warped IIR filters and their properties", in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1997, vol. 3, pp. 2205–2208.

[44] Matti Karjalainen, Aki Härmä, Unto K. Laine, and J. Huopaniemi, "Warped filters and their audio applications", in *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, October 1997, pp. 298–302.

[45] Donald G. Bailey, "Distortion measurement using spectral warping", in *International Workshop on Electronic Design, Test, and Applications (DELTA)*, Kuala Lumpur, Malaysia, January 2006.

[46] Chalee Asavathiratham, Paul E. Beckmann, and Alan V. Oppenheim, "Frequency warping in the design and implementation of fixed-point audio equalizers", in *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New York, October 1999, pp. 55–58.

[47] Richard G. Baraniuk and Douglas L. Jones, "Warped wavelet bases: Unitary equivalence and signal processing", in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1993, vol. 3, pp. 320–323.

[48] S. Umesh, L. Cohen, N. Marinovic, and D. Nelson, "Frequency-warping in speech", in *Proc. Fourth International Conference on Spoken Language*, October 1996, vol. 1, pp. 414–417.