ASIC Implementations of the Viterbi Algorithm

Jonathan M. Dobson

A thesis sumbitted for the degree of Doctor of Philosophy

The University of Edinburgh

- November 1998 -

Abstract

The Viterbi Algorithm is a popular method for decoding convolutional codes, receiving signals in the presence of intersymbol-interference, and for channel equalization.

In 1981 the European Telecommunication Administration (CEPT) created the Groupe Special Mobile (GSM) Committee to devise a unified pan-European digital mobile telephone standard. The proposed GSM receiver structure brings together Viterbi decoding and equalization.

This thesis presents three VLSI designs of the Viterbi Algorithm with specific attention paid to the use of such modules within a GSM receiver. The first design uses a technique known as redundant number systems to produce a high speed decoder. The second design uses complementary pass-transistor logic to produce a low-power channel equalizer. The third design is a low area serial equalizer.

In describing the three designs, redundant number systems and complementary pass-transistor logic are examined. It is shown that while signed binary redundant number systems offer significant speed advantages over twos-complement binary, there are other representations such as carry-save arithmetic that can perform equally well, if not better. It is also shown that complementary pass-transistor logic can offer a small improvement for some VLSI functions in terms of power consumption.

Acknowledgements

I would like to thank the following people for their invaluable assistance during the course of this PhD:

- My Supervisor Gerard Blair for his continuous support and guidance throughout this research project. Also for reading and checking this thesis.
- Bernard Mulgrew for his help and knowledge while Gerard was in Japan.
- The other members of Signal Processing Group for their support throughout my PhD. Namely (in alphabetical order) Jon Altuna, Mike Banbrook, Steve Bates, Justin Fackrell, Bill Nailon, Sarat Kumar Patra, Yoo-Sok Saw, Iain Scott, Paul Strauch, Rudolf Tanner, George Taylor.
- Wolfson Microelectronics and the EPSRC for providing financial support.
- Cadence Nvision.
- VIS Interactive plc for their support and employment during the last year.
- Nichola, for everything.

Declaration of Originality

I hereby declare that the research recorded in this thesis, and the thesis itself, is the original and sole work performed by the author while studying in the Department of Electrical Engineering at The University of Edinburgh.

Jonathan M. Dobson

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

A communication system must ensure the reliable transmission and reception of data. Modern communication systems often transfer data as digital signals. Channel effects such as multipath propagation, noise, and intersymbol interference mean that the received signal is a severely distorted version of the transmitted signal.

Before the received data can be used, the effects of this distortion must be removed. One common method is to introduce controlled redundancy into the transmitted signal, which is then removed by the receiver in such a way as to correct the distortion. A receiver using the Viterbi algorithm (VA) [1] is a popular method of correcting the received signal.

1.1.1 The Viterbi algorithm

In 1967, Viterbi proposed a maximum-likelihood sequence estimation algorithm for decoding convolutionally encoded signals [1].

The VA has been examined comprehensively in the literature and it has two main applications:

1. error correction
2. channel equalization.

To correct errors, the VA compares the received digital sequence with each of the possible received sequences. The possible sequence which has least differences from

the actual received sequence is deemed the *most likely* sequence, and this is output. To examine all possible sequences would be expensive in terms of time and silicon area. The VA is an iterative algorithm and reduces the complexity by discarding half of the possible sequences at each iterations.

For channel equalization, the VA is based on a trellis representation of the modulation process, and an estimate of the channel impulse response (CIR) is required [2]. Similar to the algorithm applied to error correction, the received sequence is iteratively compared with possible transmitted sequences filtered through a filter matched to the CIR.

Chapter 2 will describe the VA in detail and both these applications will be reviewed.

1.1.2 The European mobile radio standard: GSM

In 1981, the European Telecommunication Administration (CEPT) created the Groupe Special Mobile (GSM) Committee to devise a unified pan-European digital mobile telephone standard. After entertaining a number of different proposals, a FDMA/TDMA system was settled on in 1987 and the specification for the GSM system was published in 1988 [3].

One of the proposed GSM receiver structures brings together both of the uses of the VA as described above. The data is transmitted in packets which are encoded using block encoding, interleaving, and convolutional encoding. A Viterbi decoder is required to decode the convolutional code. A Viterbi equalizer is also required to combat the channel effects.

The design of a Viterbi decoder is presented in Chapter 4 and the design of a Viterbi equalizer for GSM is presented in Chapter 5.

1.1.3 Redundant Number Systems

Redundant number systems were first presented by Avizienis in 1961 [4]. They offer the possibility of high speed digital arithmetic because addition and subtraction operations using redundant number do not require a long carry propagation chains. Redundant numbers have been used in the literature to develop high speed arithmetic

circuits [5], [6]. In Chapter 3, redundant number systems are reviewed, and it is shown that while they offer significant speed advantages over twos-complement binary, there are other representations that can often perform equally well, if not better.

The Viterbi decoder presented in Chapter 4 uses redundant number systems to provide a high speed implementation.

1.1.4 Complementary pass-transistor logic

In recent years, technological advancements in the production of integrated circuits has resulted in VLSI circuits with smaller and smaller feature sizes. This reduction in size has brought a considerable increase in speed and there are now many complex and fast CMOS ICs available.

In Chapter 6 we review complementary pass-transistor logic (CPL) as a logic style for producing high speed, low power CMOS circuits. CPL will be compared with CMOS and similar circuit styles and it will be shown that only careful use of CPL can offer gain in terms of power consumption.

The CPL implementation of the Viterbi equalizer design proposed in Chapter 5 will be described, and conclusions about the feasibility of the implementation will be presented.

1.2 Objectives

The objective of the research described in this Thesis was to investigate the Viterbi algorithm, which is used twice in a GSM receiver system, and to implement a VLSI Viterbi decoder and Viterbi equalizer. It was decided to investigate a number of issues, such as speed, area, and power consumption which are important when developing a general Viterbi implementation, and also an implementation specific to GSM. The aim of developing the Viterbi decoder design was to put into practice some of the issues which are important when designing a high-speed Viterbi decoder. The Viterbi equalizer design was produced with the aim of putting into practice the issues which are important when designing a Viterbi equalizer specific for GSM.

This thesis will present three VLSI designs of implementations of the VA. The first

design uses a technique known as redundant number systems [4] to produce a high speed decoder. The second design uses complementary pass-transistor logic (CPL) [7] to produce a low-power channel equalizer. The third design is a low area serial equalizer suitable for use in a handheld GSM receiver.

The initial direction of the research was influenced by a Electronic Letters paper by Srinivas and Pahri which described a fast addition circuit which used redundant number systems as an internal representation to sum two twos-complement numbers at high speed [5]. This design used a novel *sign-select circuit* which was used to quickly determine the sign of a redundant number (not a trivial task, as we shall see in Chapter 3). It was noted than this circuit could be used as the basis of a fast add-compare-select unit for a Viterbi implementation (Chapter 4 will describe how the add-compare-select operation in the Viterbi algorithm is generally considered to be the main bottleneck in any implementation). It was decided to base the Viterbi implementation on redundant number systems. However, during the course of the research, the underlying architecture of the Srinivas and Pahri design was shown to be purely twos-complement (this will be described in Chapter 3) and the speed of the circuit was due to hybrid carry-lookahead/carry-select nature of the circuit and *not* due to any inherent speed advantage that redundant number systems may have.

The other main influence on the direction of the research was a paper by Yano *et al* in published in the IEEE Journal of Solid-State Circuits in 1990 which described a VLSI circuit style called *complementary pass-transistor logic* (CPL) [7]. CPL was shown to produce circuits with lower power consumption, lower area requirements, and higher speeds than CMOS. It was decided to use this circuit style to implement a Viterbi Equalizer design, so a library of CPL cells were designed to implement the design. It was discovered during the implementation of the CPL cell library that the improved performance of CPL over CMOS was not as significant as described in [7]. In fact Yano *et al* independently reached the same conclusions and proposed a revised circuit style [8].

1.3 Custom ASIC Design

Finally, one thing that has not been discussed yet, is one of the main elements of the title of the thesis: ASIC implementation.

When designing an electronic system, such as a mobile telephone, there are a number of different routes that can be taken, these include:

- Off-the-shelf components.
- Software solutions using Digital Signal Processors (DSPs).
- Custom ASIC design.

The most expensive solution, at least in the short term, is the last choice, yet this is the route that has been taken in this research. It seems as strange choice, given that the final product needs to be low cost. However, the other options have disadvantages also, these will be discussed in this section.

Using off-the-shelf components is a quick and cheap solution. Many electronics manufacturers have already invested significantly in the Viterbi algorithm, and have implemented components which can be used in many different applications. Using such components would reduce the development time for a product. The problem with this route is that one chip would be required for the Viterbi components of the system. This will increase the chip count of the final product, which will effect overall power consumption. In addition, off-the-shelf components tend to be generic, they often contain additional circuitry (and pins) which are not relevant to the chosen application. This can also increase the power consumption of the systems, and increase the size of the circuit board of the product.

The next possibility for implementation is a software solution using a DSP. To briefly explain, a number of manufactures produce processors which can be used for digital signal processing. These allow algorithms to be written software, and run on the processor. Again, this solution would reduce the development time for a product. This solution may not increase the chip count, because the DSP could be used for many elements of the overall system, not just for the Viterbi algorithm. A DSP solution would require ROM (to store the code) and RAM (to store any dynamic data). It should be noted that DSPs are available with on-board RAM, so additional RAM may not be required. However, DSPs are designed to be flexible, so they may contain redundant elements, which could increase size, and power consumption. In general DSPs come in large packages, which may not be suitable to a hand-held product. Finally, since DSPs are complex, and flexible, they are often relatively expensive, almost certainly more expensive than off-the-shelf components.

The last possibility is custom ASIC design, this involves producing a circuit level implementation of the algorithm, and manufacturing custom chips. The initial development cost, and time for this method is significantly greater than the others, but it does have a number of advantages. The cost decreases as the number of units increases, so the actual cost of the product could be cheaper that if the other methods were used. Specifically, any manufacture or distributor mark-up on using off-the-shelf components does not apply here. In addition, the implementation can be customized to the product, this means that the size of design (and hence power consumption) can be minimised. Also, with regards power consumption, the design could be produced using low-power circuit techniques, resulting in far lower power usage than the other methods. Finally, the Viterbi element of the system does not require its own chip, if custom ASIC design is used, many elements of the system could be integrated onto the same device, thus minimising the chip count.

It can be seen that there are a number of different methods of implementing a Viterbi module within a mobile telephone system. Each of these has advantages, and disadvantages. The method chosen for this research was the custom ASIC design. In addition to the advantages discussed above, this method allowed innovative design styles, and recent advancements in VLSI technology to be investigated, and implemented.

Chapter 2
The Viterbi Algorithm for Decoding and Equalization

2.1 Introduction

2.1.1 Motivation

The initial direction of the work described in this thesis, was to examine algorithms that have a number of different digital applications, and to design and implement ASIC devices that could be used in those applications. One of the algorithms that was investigated in the early period of work was the Viterbi algorithm. The main attraction of this algorithm is that it is used twice within one of the proposed mobile telephone receiver structures for the European mobile telephone standard. In addition, the two applications of the Viterbi algorithm within the receiver are quite different. The results of this initial investigation are presented in this chapter, in the form of a review of the Viterbi algorithm for decoding, and equalization.

2.1.2 Overview

In real time communication systems a common technique for error correction is to introduce redundancy at transmission using convolutional encoding and remove this redundancy at the receiver using a Viterbi decoder [1].

This chapter describes convolutional codes and reviews the Viterbi algorithm (VA) which was developed in 1967 as a method of decoding data from a convolutional encoder [1], [9]. The VA has been extended to correcting data in the presence of intersymbol interference (ISI) [10] and it has also been popular in satellite communications [11], [12]. The VA has also been applied to channel equalization [2]. Both

of these applications will be reviewed in this chapter. Viterbi decoding and Viterbi equalization are popular decoding and equalization techniques for the European mobile radio standard GSM. GSM will be reviewed, with particular attention to the use of Viterbi equalization for removing channel effects.

2.2 Convolutional codes

Convolutional encoding was developed as an alternative to conventional block encoding techniques [13]. The main difference is that they allow continuous encoding of a bit-stream into a redundant representation suitable for transmission.

A convolutional encoder operates serially on a bit-stream of arbitrary length. The encoder in Figure 2.1 consists of a 3 stage shift register, two modulo-2 adders and a 2-1 multiplexor. The constraint length K of the encoder is defined as the number of shifts over which a single input bit can influence the output of the encoder. In Figure 2.1, $K = 3$. The rate r is defined as the ratio of output bits to input bits.

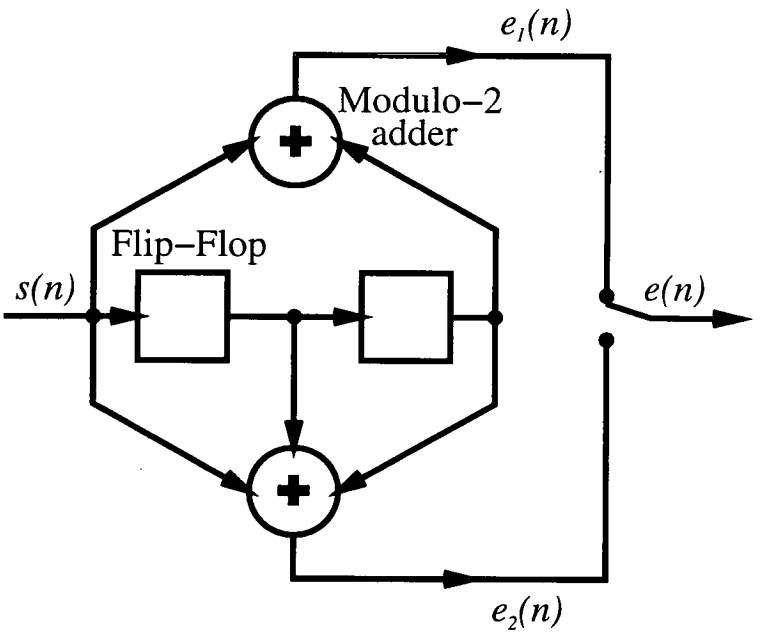


Figure 2.1: A block diagram of a convolutional encoder for $K = 3$ and $r = \frac{1}{2}$

In this section and Section 2.3 we shall use $K = 3, r = \frac{1}{2}$ convolutional codes in examples for simplicity. The techniques described can be extended for use with codes of arbitrary values of K and r.

The behaviour of a convolutional encoder is defined by the impulse responses of the modulo-2 adders. That is, the output sequence from the modulo-2 adders, given an impulse input sequence of $1, 0, 0, ...$ In the case of the encoder in Figure 2.1 there are two impulse responses:

$$g_1(n) = \{101\}$$
$$g_2(n) = \{111\}$$

These are known as the *code generators* of the convolutional encoder. By examining the diagram in Figure 2.1 it can be seen that the code generators define the connections between the shift register and the modulo-2 adders. The code generators are more commonly expressed in octal notation, for this encoder, $g_1 = 5, g_2 = 7$.

The choice of code generators affects the performance of the encoder. The performance of a code is defined as its robustness to channel noise, the property of a convolutional code which describes this robustness is called the *free distance* and is defined as the minimum Hamming distance[1] between any two code words in the code [13].

For different values of K and r there are optimal values for the code generators. These have been determined experimentally [14] and Table 2.1 shows the optimal values for $r = \frac{1}{2}$.

Constraint Length	g_2	g_1
3	5	7
4	15	17
5	23	35
6	53	75
7	133	171
8	247	371
9	561	753
10	1167	1545
11	2335	3661
12	4335	5723
13	10533	17661
14	21675	27123

Table 2.1: Optimal Code Generators for $r = \frac{1}{2}$

[1]The Hamming distance between two binary words is defined as the number of bitwise mismatches.

2.2.1 The trellis representation

An important representation of a convolutional code is a *trellis diagram*. The trellis diagram is also essential to the understanding of the Viterbi decoding algorithm. To draw a trellis diagram for a convolutional code we first must create a state transition table. Table 2.2 shows a state transition table for the convolutional encoder of Figure 2.1.

State	Input symbol	Next State	Output symbol
00	0	00	00
	1	10	11
10	0	01	11
	1	11	00
01	0	00	10
	1	10	01
11	0	01	01
	1	11	10

Table 2.2: State transition table

The states in the first column correspond to the value stored in the shift register. The second column shows the next input symbols that result in a new value *next state* being stored in the shift register. The final column shows the pair of output symbols produced by the encoder for the corresponding input symbol.

From this table of state transitions, a trellis diagram can be constructed. Figure 2.2 shows the trellis diagram which was constructed from this table. The trellis diagram is an interconnected set of *stages*, each stage is indicated by a different time, t.

An input sequence to the encoder is represented in the trellis diagram as a path through the structure, starting at time $t = 0$. A solid line corresponds to a 0 input symbol and a dashed line corresponds to a 1 input symbol. The edges between nodes are labelled with the output symbol pairs.

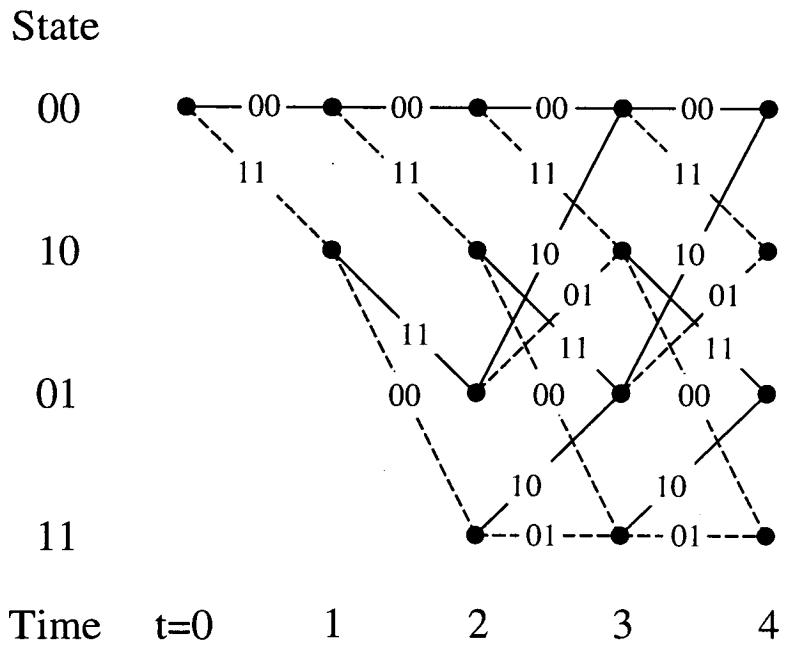


Figure 2.2: A trellis diagram representing the encoder in Figure 2.1

2.3 Maximum-likelihood decoding of convolutional codes

2.3.1 The Viterbi algorithm

The VA is based on the concept of the trellis diagram as described in Section 2.2.1. The algorithm is a maximum-likelihood decoding algorithm which selects the path through the trellis diagram which is the closest to the original input sequence [10]. The branches on this path denote the originally encoded sequence. Figure 2.3 shows a basic communication system using a Viterbi decoder.

Figure 2.3: A basic communication system using a Viterbi decoder

At each stage of the trellis there are m^{K-1} nodes (where m is the alphabet size, and K is the constraint length of the convolutional code). The VA implements one stage of

the trellis diagram directly. For each node, two values are stored:

1. Path Metric Value (PMV): this value indicates the likelihood that this path is the correct one. At each stage the node with the lowest PMV is taken to be the most likely one.

2. Path History (PH): a sequence of bits which corresponds to the branches which led to this node: a dashed line being a 1 and a solid line being a 0.

The VA steps through the trellis evaluating new PMVs for each stage in the following way:

- At each time interval the decoder receives n symbols (where n is the number of modulo-2 adders).
- A Branch Metric Value (BMV) is calculated for each branch, this is the difference (or Hamming distance) between the received symbols and the labels on the branches. This can be defined simply as the number of mismatches between the received signal bits and the branch designation.
- For each node in the current stage, the following steps are performed:
    1. For all the nodes in the previous stage that are connect via a branch to this node:
        - Sum the PMV of the previous node with the BMV of the interconnecting branch.
    2. Choose the lowest of these sums as the PMV for this node.
    3. Append the PH with a 0 or a 1 depending on whether the branch was a solid (0) or dashed (1) line.

At each step the PH of the node with the lowest PMV corresponds to the decoder's prediction for the original unencoded sequence. When implementing the VA, it is not reasonable to store the whole of the message, so a finite number of bits have to be used to store the PH. Simulations by Michelson and Levesque have shown that storing PHs of more than 4 times the constraint length of the convolutional code shows no increase in performance [15].

2.3.2 Correcting bit errors with the Viterbi algorithm

Figure 2.4 shows how the VA can correct bit errors. The diagram shows the steps in the algorithm as the received sequence is decoded. The value on the nodes represent the PMV up to that point. Where there are two values on a node, they represent the addition of the BMVs and PMVs for the upper and lower paths into the node.

In this example the transmitted sequence is $\{00, 11, 11, 10\}$, representing a original data sequence of $\{0, 1, 0, 0\}$. However, a bit error occurs at position 4 so the received sequence is $\{00, 10, 11, 10\}$.

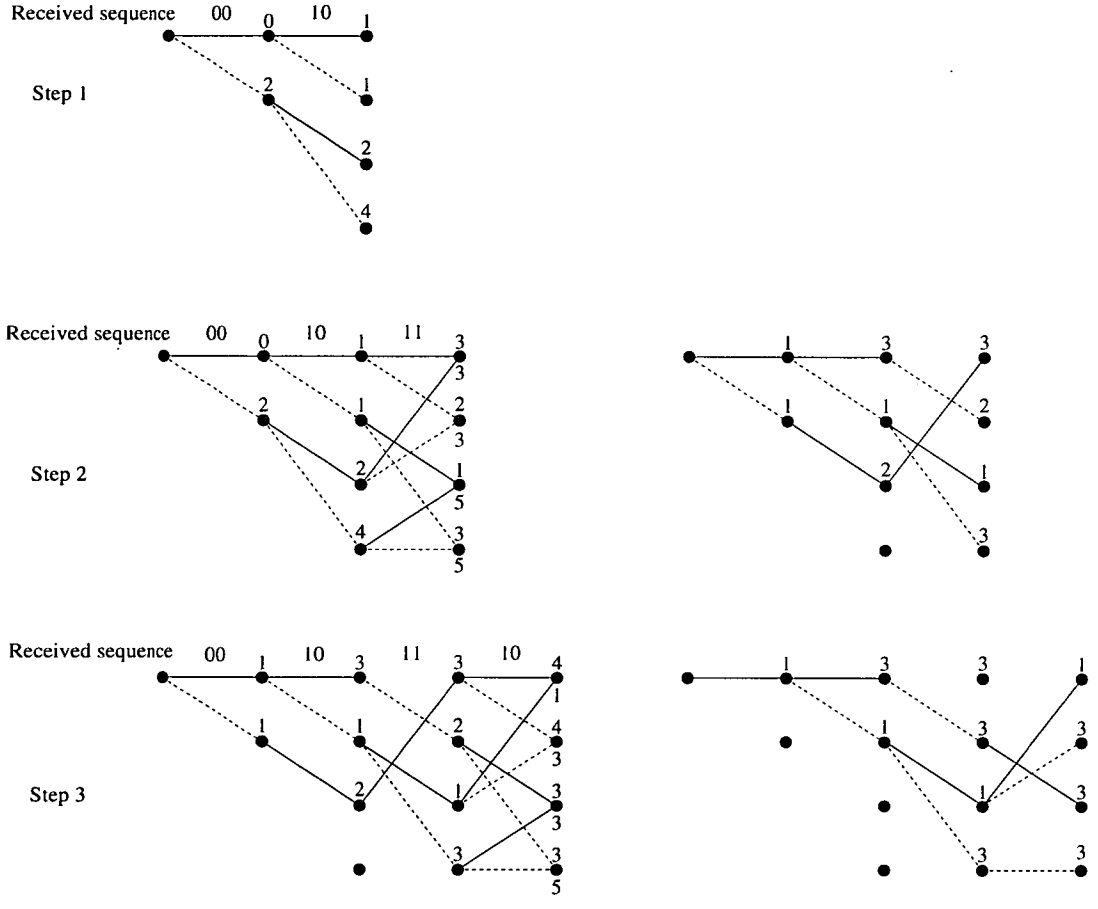


Figure 2.4: Correcting bit errors with the Viterbi Algorithm

At step 1, just after the error has occurred, there are two paths through the trellis with the same value of metric. However, because no errors are present in the subsequent steps, by the time we reach step 3, the metric associated with the correct path is the lowest.

The VA is able to correct the error because the Hamming distance between the received sequence and the transmitted sequence is less than the free distance of the code as defined in Section 2.2.

2.3.3 Failure of the Viterbi algorithm

There are cases where the bit errors cannot be corrected by the VA, or rather the bit errors are incorrectly corrected. This occurs when the Hamming distance between the received sequence and the transmitted sequence is greater than or equal to the free distance of the convolutional code.

Figure 2.5 shows an example of breakdown of the VA due to too many bit errors in the channel.

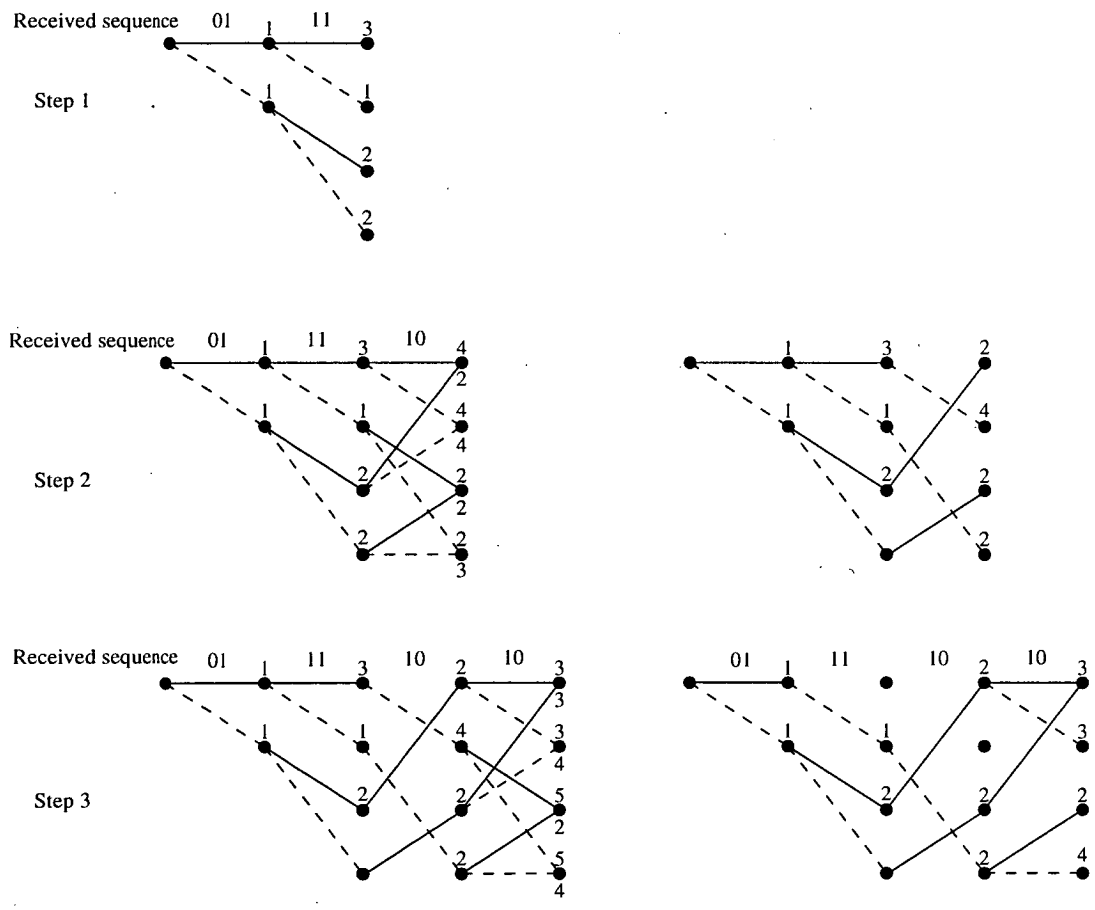


Figure 2.5: Breakdown of the Viterbi Algorithm

The transmitted sequence is $\{00, 11, 11, 10\}$, as in the previous example in Section 2.3.2.

This time there are two bit errors, at positions 1 and 5, so the received sequence is $\{01, 11, 10, 10\}$

The layout of the diagram is the same as before. It can be seen that at step 1 the lowest metric corresponds to the correct path through the trellis. However, at step 2, correct path is discarded because both paths into the 3rd node have metrics of size 2 and the correct path is selected (at random) to be discarded. By the time we get to step three, the algorithm has incorrectly identified the sequence $\{00, 11, 00, 10\}$ which corresponds to a data sequence of $\{0, 1, 1, 0\}$ as being the most likely received sequence

This time the Hamming distance between the received sequence and the transmitted sequence is equal to the free distance of the convolutional code so the VA is equally likely to choose the correct sequence as the nearest incorrect sequence.

2.3.4 The VA applied to intersymbol interference

Soon after the original publication of the VA, Forney applied it to the detection of digital transmissions in the presence of intersymbol interference (ISI) [10].

ISI is a problem in pulse modulation systems and occurs when the a transmitted pulse has not decayed before transmission of the next pulse is started. As an example of ISI, we will consider a simple pulse amplitude modulation (PAM) system, as described by the system shown in Figure 2.6. The transmitted signal is described by equation 2.1.

$$s(t) = \sum_{k=-\infty}^{\infty} x_k h(t - kT), \tag{2.1}$$

where, x_k are the samples of the data sequence, i.e. the modulating waveform; $h(t)$ is the channel impulse response which describes the pulse shaping; and T is the symbol period.

The transmitted signal is then corrupted by additive white Gaussian noise (AWGN) signal $n(t)$, to give the received signal, $r(t)$ as described by equation 2.2.

$$r(t) = s(t) + n(t). \tag{2.2}$$

Figure 2.7 is an example of the ISI within a PAM communication system, Figure 2.7(a)

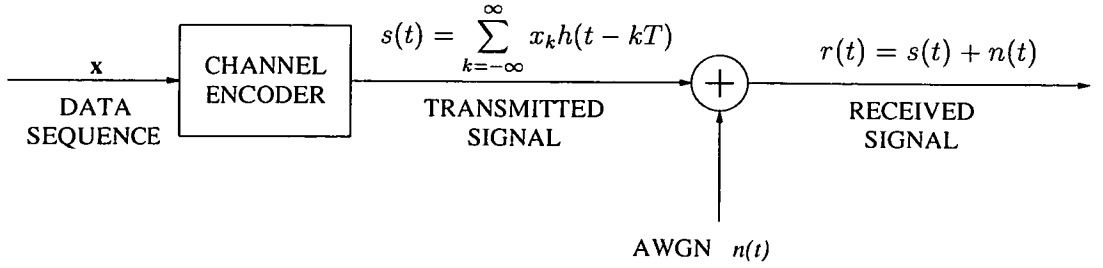


Figure 2.6: A PAM communications sysem

represents the data sequence

$$x(t) = \{1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0\} \tag{2.3}$$

Figure 2.7(b) shows $h(t)$ applied to this pulse sequence, in this example $h(t)$ is a Gaussian shaping function. The transmitted sequence $s(t)$ is shown in Figure 2.7(c).

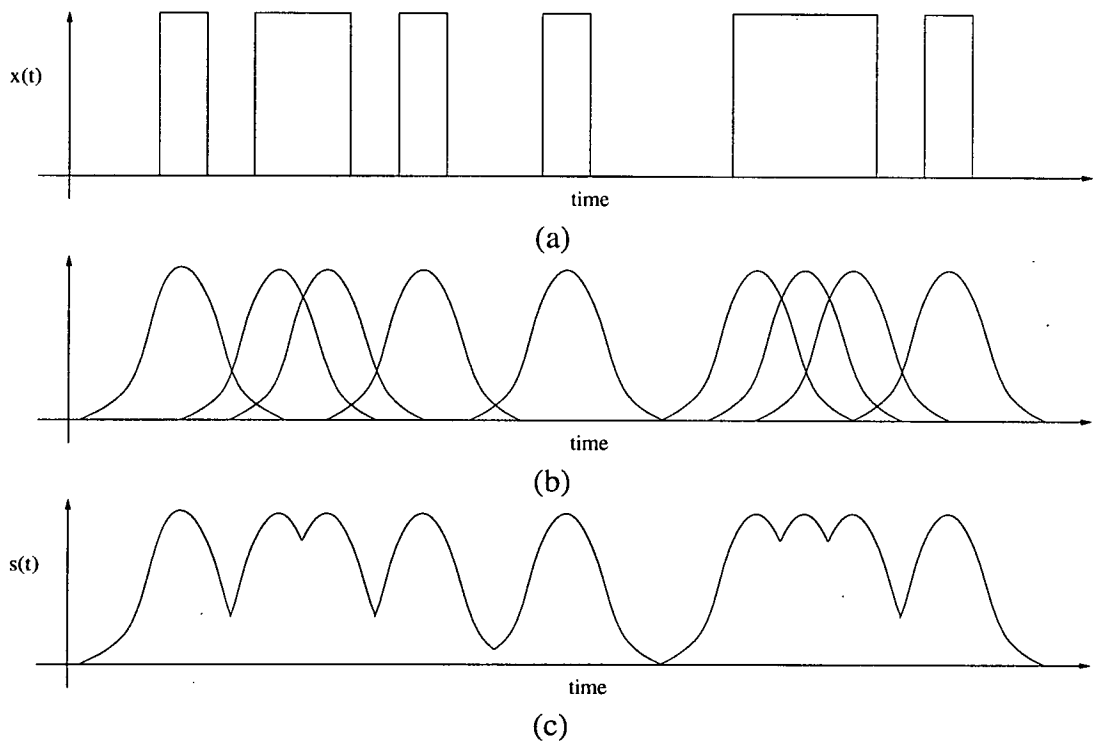


Figure 2.7: An example of intersymbol interference

The decoder structure proposed from [10] is shown in Figure 2.8. The received data is passed through a whitened match filter with response $w(-t)$. This filter is necessary because the VA requires the noise components of each successive received sample to

be statistically independent [16].

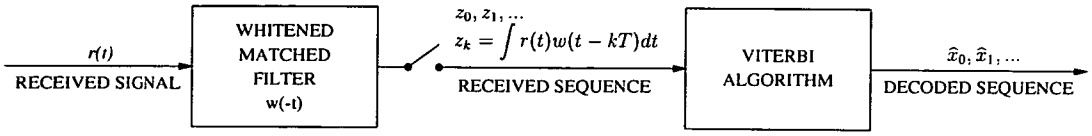


Figure 2.8: A maximum-likelihood decoder for PAM

Forney noted that the VA receiver has a recursive structure with complexity proportional to m^L, where m is the input alphabet size and L is the length of the channel impulse response $h(t)$. He also showed that the structure requires no multiplications and it is optimum for maximum-likelihood sequence estimation, as long as there are no limits placed on the delay through the decoder.

2.4 Channel equalization using the Viterbi algorithm

The VA was suggested in [3] as a method for channel equalization in the European mobile radio standard GSM. Previously the VA had been applied to the equalization of phase shift keying (PSK) modulation by Ungerboeck in 1974 [16], and Acampora extended the application of the VA to the more generalised form of modulation - quadrature amplitude modulation (QAM), of which PSK is a subset [2], [17].

In this section, the application of the VA to the equalization of QAM signals will be reviewed. It will be shown that the VA can be applied to minimum shift keying (MSK) which is also a form of QAM [18], [19].

2.4.1 Quadrature Amplitude Modulation

From [2], we assume that the QAM process can be described by equation 2.4:

$$S(t, \mathbf{d}) = \Re\left\{\sum_{k=0}^{N} [a_k(\mathbf{d}) + jb_k(\mathbf{d})]\, \overline{h}(t - kT)e^{-jw_0 t}\right\}, \tag{2.4}$$

where $\overline{h}(t) = h_R(t) + jh_I(t)$ is a complex waveform, $a_n(\mathbf{d})$ and $b_n(\mathbf{d})$ are real numbers dependent on the data $\mathbf{d}$, T is the symbol period, and $N + 1$ is the number of complex channel symbols used to transmit the data $\mathbf{d}$.

Figure 2.9, from [2] shows a model of a QAM digital communication system. The channel encoder transforms the input data stream **d** into the in-phase and quadrature-phase symbols streams **a** and **b**. These signals can be dependent or independent of each other.

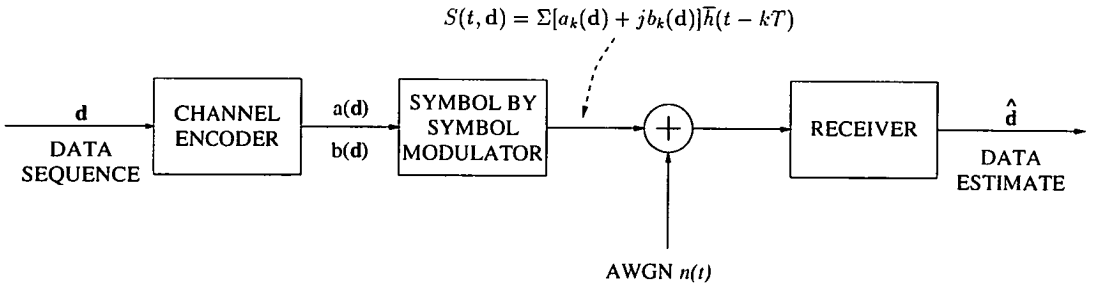


Figure 2.9: Symbol-by-symbol QAM model

2.4.1.1 MSK as a subset of QAM

Figure 2.10 shows a 4-QAM constellation diagram and the state transition diagram for minimum shift keying (MSK). These diagrams represent the operation of the channel encoder in Figure 2.9. Both modulation processes have four phase states. In generalised 4-QAM, an input symbol can result in a transition from the current state to any of the four states.

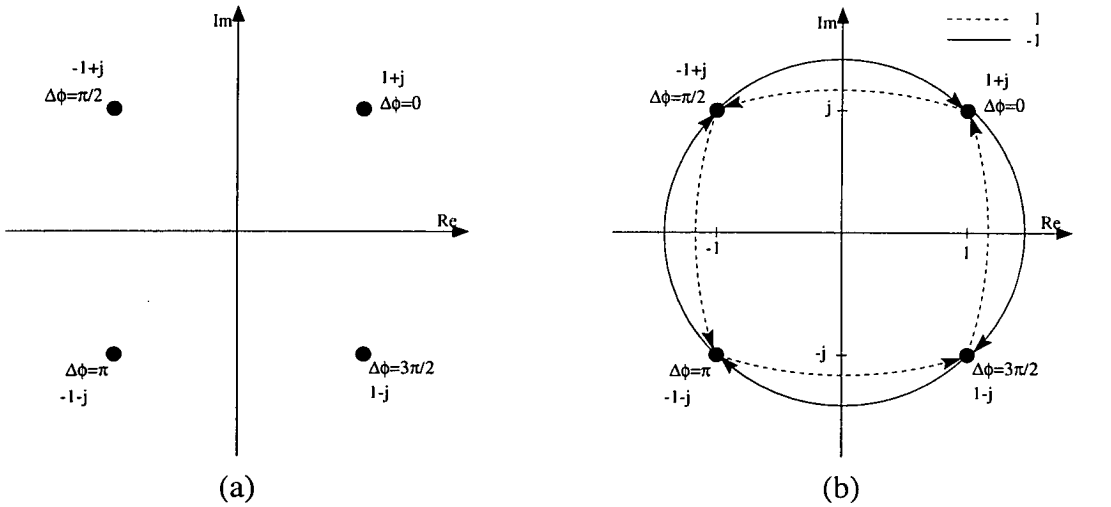


Figure 2.10: 4-QAM constellation and MSK transition diagram

In MSK, transitions between phase states are limited to those defined in the transition

diagram. A solid line represents an input symbol -1 to the encoder and a dashed line represents an input symbol of $+1$.

The real and imaginary coordinates of the phase states represent the in-phase and quadrature-phase samples a_k and b_k in equation 2.4.

As the Figure 2.10 shows, MSK is a subset of 4-QAM where the symbol streams a and b have a dependency on each other as defined by Figure 2.10(b).

2.4.2 Equalization of QAM and MSK using the Viterbi algorithm

2.4.2.1 Deriving the trellis diagram

To apply the VA to equalization of QAM we need to produce a trellis diagram of the modulation process. Figure 2.11(a) shows a trellis diagram for 4-QAM with four bits (two symbols) of ISI. The trellis is constructed from the constellation diagram. The number of states in the trellis diagram is determined by the amount of ISI that is expected to be introduced in the channel and can be tolerated.

Similarly, the trellis diagram for MSK can be derived from the MSK modulation diagram of Figure 2.10(b). Alternatively the 4-QAM trellis diagram can be altered to obtain the MSK trellis. Figure 2.11 shows which of the transitions in the 4-QAM trellis are not possible in the MSK trellis. We can see that the MSK trellis requires only eight states.

It should also be observed that if the we know the starting state of the MSK modulation process, then the number of states in each iteration reduces to $2^{ISI-2} = 4$. This is shown in Figure 2.12.

2.4.2.2 Applying the Viterbi algorithm to QAM

The analysis presented in this section is for QAM but, as explained in Section 2.4.1, the technique can be applied to MSK signals. The transmitted signal is described by equation 2.5:

$$f(t) = \Re[x(t)e^{j\omega_c t}], \tag{2.5}$$

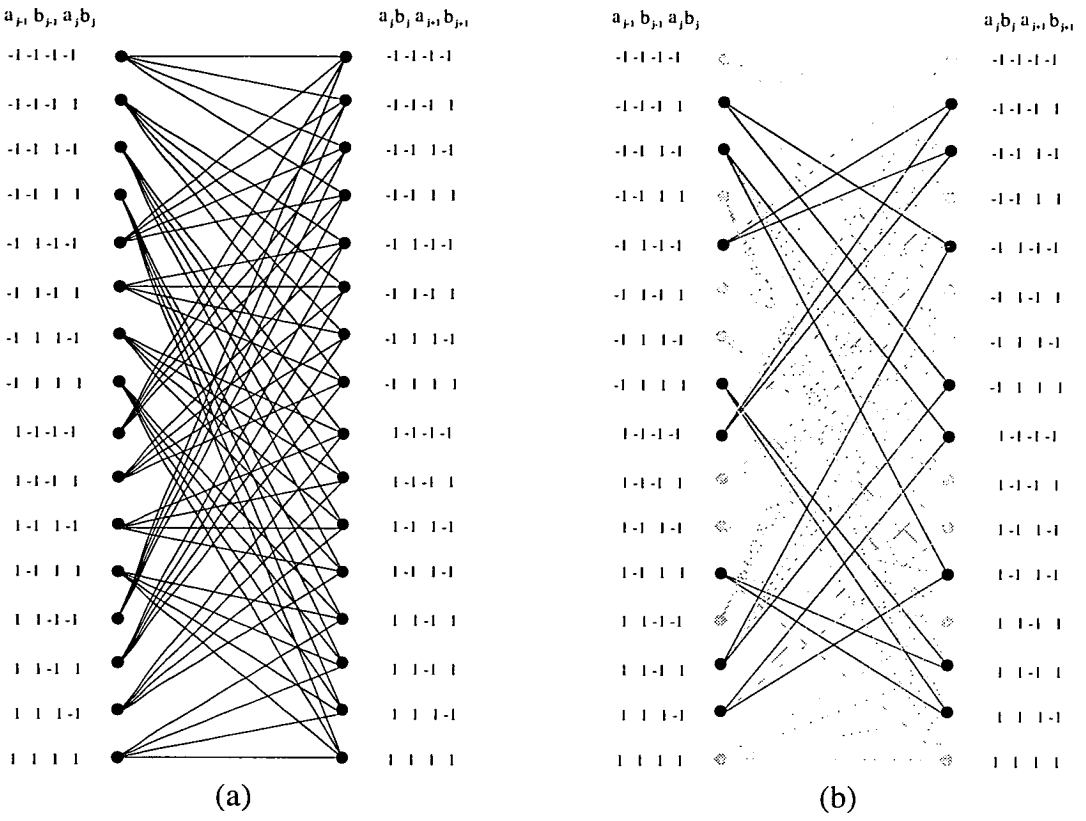


Figure 2.11: 4-QAM and MSK trellis diagrams

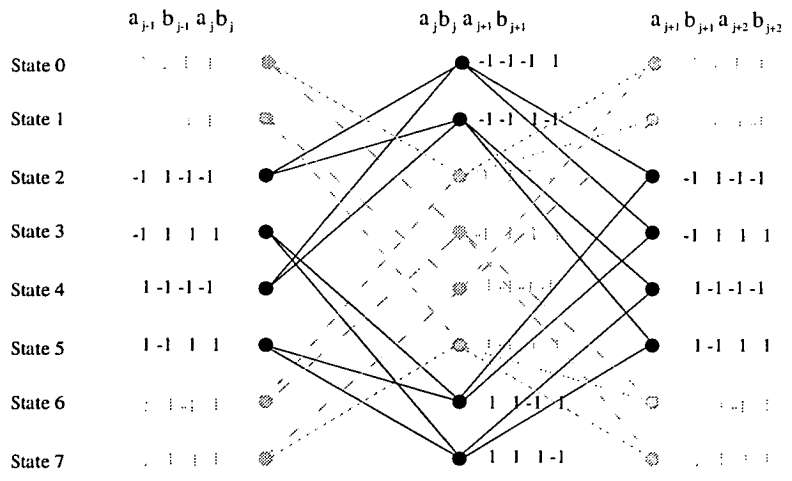


Figure 2.12: MSK trellis diagram

where ω_c is the carrier frequency and $x(t)$ is the transmitted signal consisting of in-phase and quadrature phase data sequences:

$$x(t) = \sum_{k=0}^{N} a_k g(t - 2kT) + j \sum_{k=0}^{N} b_k g(t - 2kT - T), \tag{2.6}$$

where a_k and b_k can assume the values ± 1 and the data rate is $1/T$.

The signal is then transmitted though a channel with complex impulse response $g_c(t)$ and a receiver filter with response $g_r(t)$. Then received signal is:

$$f(t) = \Re[s(t)e^{j\omega_c t}], \tag{2.7}$$

where

$$\begin{aligned} s(t) &= \sum_{k=0}^{N} a_k h(t - 2kT) + n_1(t) + \\ &\quad j \sum_{k=0}^{N} b_k g(t - 2kT - T) + j n_2(t), \end{aligned} \tag{2.8}$$

$n_1(t)$ and $n_2(t)$ are independent AWGN functions. The overall channel impulse response (CIR) $h(t)$ is given by the convolution of the three impulse responses, shown in equation 2.9.

$$h(t) = g(t) \otimes g_c(t) \otimes g_r(t) \tag{2.9}$$

2.4.2.3 The Viterbi algorithm for QAM

To apply the VA to QAM we need to derive an expression for the metric calculation. Each of the 2^{2N+2} possible sequences through the trellis has a metric associated with it. To apply the VA we need to assume that the CIR $h(t)$ is known - or at least can be estimated.

The reconstructed signal for the mth sequence is:

$$s_m(t) = \sum_{k=0}^{N} a_k^m h(t - 2kT) + j \sum_{k=0}^{N} b_k^m g(t - 2kT - T) \tag{2.10}$$

From [3], the most likely transmitted sequence is one which maximises the likelihood

function:

$$\mathcal{L}_m = exp\left[-\frac{2}{\eta_o}\int_{-\infty}^{\infty}|s(t)-s_m(t)|^2\,dt\right] \tag{2.11}$$

where η_o comes from the spectral density of the AWGN functions $n_1(t)$ and $n_2(t)$ in equation 2.8. This is equivalent to maximising:

$$\Lambda_m = 2\left[\int_{-\infty}^{\infty} s(t)s_m^*(t)dt\right] - \int_{-\infty}^{\infty}|s_m(t)|^2\,dt \tag{2.12}$$

After substituting for $s(t)$ and $s_m(t)$ [2], the metric computation becomes:

$$\begin{aligned}\Lambda_m &= 2\sum_{k=0}^{N}(a_k^m y_k + b_k^m z_k) \\ &\quad -\sum_{k=0}^{N}\sum_{i=0}^{N}(a_k^m a_i + b_k^m b_i)\chi_{k-i} \\ &\quad -\sum_{k=0}^{N}\sum_{i=0}^{N}(a_k^m b_i + b_k^m a_i)\zeta_{k-i},\end{aligned} \tag{2.13}$$

where y_k and z_k are the received signals statistics at time kT obtained by passing the received samples though a matched filter matched to $\overline{h}(t)$; and χ_n and ζ_n are the real and imaginary samples of the matched filter's response to $h(t)$ at time nT.

From [2], rewriting (2.13) and assuming that a_k and b_k are 0 for $k < 0$ and $k > N$:

$$\begin{aligned}\Lambda_m &= \sum_{k=0}^{N} a_k^m\left[2y_k - a_k\chi_0 - 2\sum_{i=k-L}^{k-1}(a_i^m\chi_{k-i} + b_i^m\zeta_{k-i})\right] \\ &\quad \sum_{k=0}^{N} b_k^m\left[2z_k - b_k\chi_0 - 2\sum_{i=k-L}^{k-1}(b_i^m\chi_{k-i} + a_i^m\zeta_{k-i})\right]\end{aligned} \tag{2.14}$$

Equation 2.14 can be computed recursively from the previous partial sum as shown in equation 2.15.

$$\begin{aligned}\Lambda_{mn} &= \Lambda_{m(n-1)} + a_n^m\left[y_n - \sum_{i=n-L}^{n-1}(a_i^m\chi_{n-i} + b_i^m\zeta_{n-i})\right] \\ &\quad + b_n^m\left[z_n - \sum_{i=n-L}^{n-1}(b_i^m\chi_{n-i} + a_i^m\zeta_{n-i})\right] \\ &= \Lambda_{m(n-1)} + a_n^m[y_n - I_s] + b_n^m[z_n - Q_s]\end{aligned} \tag{2.15}$$

where y_n and z_n are the in-phase and quadrature received signals samples after passing through a filter matched to an estimate of the channel impulse response (CIR).

The values I_s and Q_s can be pre-computed and are constant for is long as the CIR remains constant. It can be seen that the computational complexity of the Viterbi equalizer for QAM is four additions per state.

2.4.2.4 Extending Viterbi equalization to GMSK modulation

In equation 2.6, the function $g(t)$ describes the pulse shaping of the transmitter. This allows us to extend the analysis to the equalization of Gaussian minimum shift keying (GMSK) which is the modulation technique used in the GSM system [3].

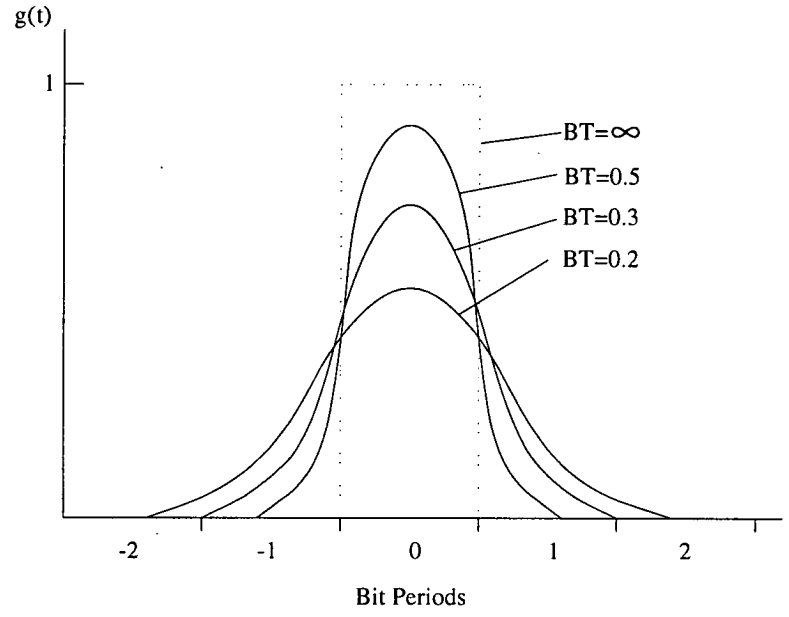


Figure 2.13: Pulse shaping function $g(t)$ for GMSK

Figure 2.13 shows the $g(t)$ waveform for GMSK. From [20] the shaping function is defined by:

$$g(t) = \frac{1}{2T}\left[Q\left(2\pi B\frac{t-\frac{T}{2}}{\sqrt{ln2}}\right) - Q\left(2\pi B\frac{t+\frac{T}{2}}{\sqrt{ln2}}\right)\right], \tag{2.16}$$

for

$$0 \leq BT \leq \infty \tag{2.17}$$

where

$$Q(t) = \int_t^\infty \frac{1}{\sqrt{2\pi}} e^{-\tau^2 2} d\tau \tag{2.18}$$

B is the bandwidth of a low pass filter with a Gaussian shaping spectrum and T is the bit period. Figure 2.13 shows the effect of different values of BT on the shaping

function $g(t)$. This Gaussian shaping introduces controlled ISI.

2.4.3 Performance of the Viterbi Algorithm

Forney and Viterbi have both analysed the performance of the VA, for convolutional codes [9], [21]. With an alphabet size of 2 and symmetric memoryless channels, $P(\epsilon_k)$, the probability of an error occurring at time k is:

$$P(\epsilon_k) \approx N_d 2^{-dD}, \tag{2.19}$$

where d is the free distance as defined in Section 2.2 , N_d is the number of all possible paths, and D is the Bhatacharyya distance [9]:

$$D = log_2 \sum_Z P(z|0)^{1/2} P(z|1)^{1/2}, \tag{2.20}$$

where $\sum_Z$ is the sum over all outputs z is the channel space Z.

For channels with AWGN, the error probability is:

$$P(\epsilon_k) \approx N_d e^{-dr E_b/N_0}, \tag{2.21}$$

where r is the code rate, and E_b/N_0 is the signal-to-noise ratio of the channel.

Heller and Jacobs showed by simulation that the results achieved using 8-level quantization are very close to this upper bound [11]. They also showed that 2-level quantization is 2 dB worse than 8-level.

The performance of the VA for equalization of GMSK signals in the GSM system will be reviewed in Section 2.6.6.

2.5 A comparison of Viterbi decoding and equalization

In this Chapter the use of the VA for decoding of convolutional codes, and the equalization of QAM modulation have both been explained. These applications have different complexities which will be explained in this section.

The construction of the trellis for convolutional codes is straightforward. The trellis is directly related to the code generators and optimum code generators for rate $r = \frac{1}{2}$ are known (see Table 2.1) [14]. For a rate $r = \frac{1}{2}$ code with constraint length (the number of bits in the encoded sequence over which a single bit can influence the output of the encoder) $K = 7$, the trellis diagram will have $2^4 = 64$ states with 2 branches from each state to the next states. For MSK the trellis construction is more complicated, it is dependent on the modulation constellation and the amount of ISI in the channel. However, with an ISI of 6, the trellis diagram for MSK equalization would also have 64 states with 2 branches from each state to the next states. So the complexity of the trellis is similar for equalization and decoding.

The interconnection of the equalizer trellis states becomes more complicated if a QAM modulation system, such as the one shown in Figure 2.10 (a), is used. In such a system, any input symbol to the modulator can result in a transition to any state. In the case of 4-QAM there are four branches from each state to the next stage of states. So, increasing the complexity of the modulation process will increase the complexity of the trellis. This is also true for the decoding of convolutional codes if the rate is changed.

The main difference in complexity is the calculation of the incremental metrics. As discussed in Section 2.3.1, for Viterbi decoding, the branch metrics are computed as the Hamming distance between the received symbol and the labels on the branches. This is a fairly straightforward operation and consists of a bitwise comparison and summation. From [11], an 8-level quantization is satisfactory for the input values, using the distances described in [22], a rate $r = \frac{1}{2}$ decoder has branch metric value consisting of 3 bits, this means that the metric evaluation can be implemented with simple gates (Section 4.2.1 will describe the design of such a circuit).

It can be seen from equation 2.15 that for Viterbi equalization the evaluation of the branch metric is a summation operation consisting of real number additions and subtractions. If we assume that the inputs and coefficients have a resolution of 8 bits [3] and that there are 5 bits of ISI, then the metric calculation operation consists of summing 12, 8-bit numbers. It can be seen that the metric calculation is significantly more complicated for Viterbi equalization than decoding.

It should be noted that the summations only have to be performed for a new CIR estimate. If the CIR estimate is constant then only 2 subtraction operations are performed. This means that to reduce the delay in metric calculation for a Viterbi equalizer it is advisable not to update the CIR estimate at every received signal interval. This means that

equalizer could be pipelined and the summations for the current CIR estimate could be performed while the equalizer is decoding the received symbols for the previous CIR estimate. This method can be applied to GSM (Section 2.6).

Since the incremental metrics are significantly larger for Viterbi equalization, then the path metrics are also. This means that the add-compare operation which has to be performed for each node will be operating on numbers of larger word length. This means that the area required to implement each node will be larger.

In summary a Viterbi equalizer is more complicated that a Viterbi decoder. In Chapter 4 a design of a Viterbi decoder is proposed and in Chapter 5, a Viterbi equalizer design is described. The remainder of this chapter will describe the GSM system which is a common application of Viterbi decoding and equalization.

2.6 The GSM system

The GSM system was developed by the European Telecommunications Standard Institute/Group Special Mobile European Community working group [3]. The GSM system aimed to provide a unified mobile telephone standard for the entire European Community.

Previously there had been 6 different mobile telephone standards operating in 16 different counties [23], and the majority of these system were analogue based. The GSM working group set themselves the task of devising a unified telephone standard that would allow equipment and networks all over the European Community to be compatible with each other.

Before the GSM standard was introduced in 1991 it was decided that the new system should be digital rather that analogue based. A digital system would provide: improved transmission quality, increased security in communications, ISDN services [24], improved bandwidth utilisation, network modularity, and lower operating cost [25].

Nine different proposals were "entertained" by the GSM group; the one chosen was a frequency division multiple access / time division multiple access (FDMA/TDMA) hybrid system using a GMSK modulation process.

The most popular proposal for the GSM receiver structures [3], [26], [27], uses two

distinct Viterbi processors. The first one is a Viterbi equalizer and is used for channel equalization. The second is a Viterbi decoder and is used to decode the data which is encoded using a convolutional code of rate $\frac{1}{2}$ and constraint length 5.

This section will review the GSM system and describe how the Viterbi equalizer and Viterbi decoder fit into such a system.

2.6.1 Specifications

The Time Division Multiple Access (TDMA) transmission operates in the 900 MHz frequency band. The gross bit rate is 270.833kb/s, and the voice data rate achievable is 13kb/s. The data is transmitted using GMSK modulation with a BT product of 0.3 (see Section 2.4.2.4). The GSM system should be able to cope with many severe propagation conditions, and vehicle speeds of up to 250km/h [23].

There are two frequency bands: 890-915 MHz is used for mobile-to-base-station transmission; and 935-950MHz is used for base-station-to-mobile transmission. These two frequency bands are divided into 124 carrier frequencies with 200 kHz guard band spacing between them.

2.6.2 TDMA packet structure

Each of the carrier frequencies described above supports 8 time slots, within a TDMA frame, the structure of which is shown in Figure 2.14. As the diagram shows, the TDMA frame lasts for $4.615ms$ and contains 8 times slots lasting for $0.577ms$ which are reserved for channel packet bursts.

Each normal burst time slot contains 114 data bits, and a 26 bit midamble. This midamble is used by the receiver to estimate the impulse response of the channel. The CIR is obtained by correlating the received midamble with the correct midamble which is known by the receiver. The midamble sequence is composed of a 16 bit word followed by 5 cyclically repeated bits [27]. The GSM groups have identified 8 midamble sequences by computer simulation by optimising their autocorrelation and crosscorrelation properties [27].

The GSM system base stations are arranged in a cellular network and all of these 8

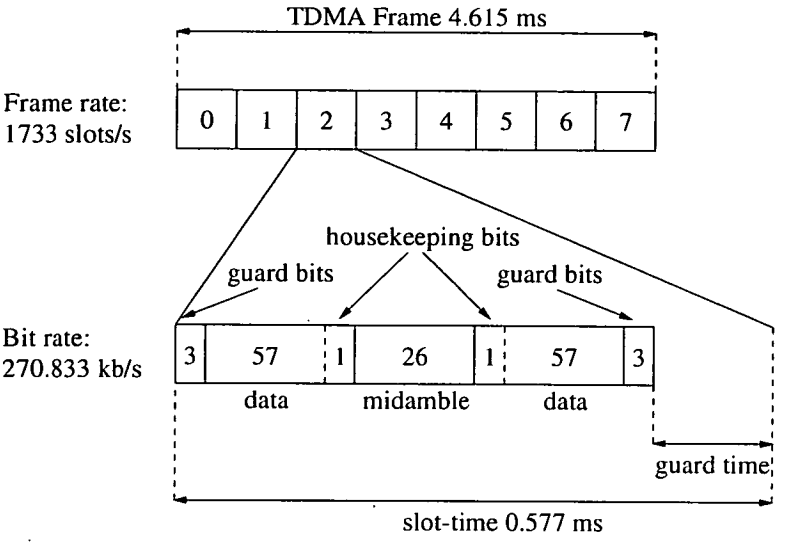


Figure 2.14: The GSM TDMA frame structure

midambles are used as a means of identifying the base station, since adjacent base stations use different midamble sequences [23].

When a mobile station initialises a link with the base station a physical channel is opened. Each physical channel is allocated a TDMA time slot (TN), numbered $0 - 7$ and a TDMA frame number (FN), numbered $0 - 123$, corresponding to the carrier frequency. The GSM system employs *frequency hopping* which helps to overcome the problem of channel fading [23]. This means that the FN can change throughout a channel's existence.

Each mobile station is allowed to operate when they are within a 35km distance from the base station. The time for a signal to travel the the maximum distance of 70km to and from a base station is $233.3\mu s$. This seems to imply that a guard spacing longer than $233.3\mu s$ is required at the end of every slot to ensure that signals from mobile stations do not overlap. In fact, a guard time of $68.25bits$ ($252\mu s$) is only required when the mobile station initially connects to the base station (either during call start up or handover). This is achieved with a special type of burst called an *access burst.* There are five types of GSM burst structure and these are shown in Figure 2.15.

When the mobile station uses the access burst, the base station calculates a 6-bit offset called the *timing advance.* This offset informs the mobile station to transmit subsequent slot packets earlier by multiples of $3.69\mu s$. This ensure that slots arrive at the base station during the correct time period, it also allows the guard time to be reduced

NORMAL BURST

TAIL BITS	DATA BITS	MIDAMBLE	DATA BITS	TAIL BITS	GUARD TIME
3	58	26	58	3	8.25

FREQUENCY CORRECTION BURST

TAIL BITS	FIXED BITS	TAIL BITS	GUARD TIME
3	142	3	8.25

SYNCHRONISATION BURST

TAIL BITS	SYNC BITS	EXTENDED MIDAMBLE	SYNC BITS	TAIL BITS	GUARD TIME
3	39	64	39	3	8.25

ACCESS BURST

TAIL BITS	SYNCHRO SEQUENCE	ENCRYPTED BITS	TAIL BITS	GUARD TIME
8	41	36	3	68.25

DUMMY BURST

TAIL BITS	MIXED BITS	MIDAMBLE	MIXED BITS	TAIL BITS	GUARD TIME
3	58	26	58	3	8.25

Figure 2.15: The 5 difference GSM TDMA burst structures

to $30.36\mu s$ or 8.25 bits [23].

During the call, the base station continually monitors the time that the slot from the mobile station is received and adjusts the timing advance accordingly.

2.6.3 Channel coding operations

Before the speech data is transmitted in the TDMA packet, it is pre-coded. The reason for this coding is twofold: firstly to make the communication of the data more robust; and secondly, to introduce encryption into the data for security reasons. In this section the GSM channel coding operations will be outlined.

The GSM system offers two main modes of communication: speech communication at full rate ($22.8kb/s$) and half rate ($11.4kb/s$); and data communication at full rate ($22.8kb/s$) and half rate ($11.4kb/s$). In addition to these two channels, there are a number of control channels for synchronisation, frequency control, etc.

These data rates correspond to the coded channel data rate, rather than the rate of data actually communicated over the channel. For full rate speech, the rate of the traffic channel is $13kb/s$. The half rate channel is provided for future development of the GSM system, when low bit-rate voice coders are available, the channel capacity can be doubled, with two channels occupying the same time slot at alternate frames.

Information about other channel coding techniques can be found in the literature [23], [28], [3].

2.6.3.1 Speech coding

The block diagram in Figure 2.16 shows the channel coding procedure for full rate speech. The speech is sampled at $8kHz$ which after compression has at a bit rate of $13kb/s$, each speech frame consists of 260 bits and this data is fed into the channel encoder. The data bits are divided into three classes:

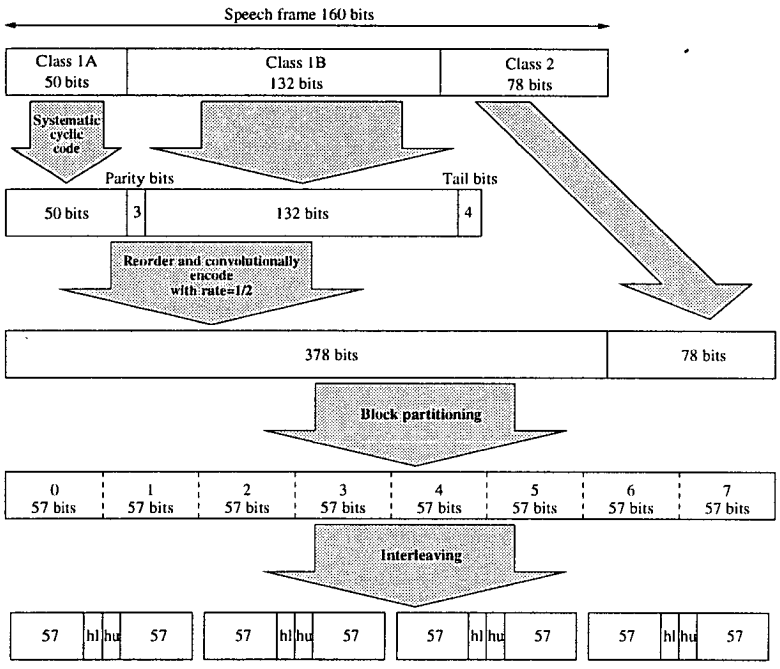


Figure 2.16: The channel coding for GSM full rate speech

- Class 1A

  These bits are encoded using a systematic cyclic code with the generator polynomial of:

  $$G_4(D) = D^3 + D + 1 \tag{2.22}$$

  and are appended with three parity bits.

- Class 1B

  These bits are appended to the encoded Class 1A bits, with four tail bits to flush the Viterbi decoder in the receiver. All of the Class 1 bits are re-ordered as

described in [23] and are convolutionally encoded with the encoder shown in Figure 2.17.

- Class 2
  These bits are left unencoded.

After these encoding operations, there are 456 bits left to be transmitted, these are block partitioned into groups of 57 bits, which are then interleaved with each other, and blocks from the preceding and following speech frames. Each pair of 57 bit blocks make up one normal burst (as outlined in Section 2.6.2) along with two flag bits which indicate whether the information is speech or control [23].

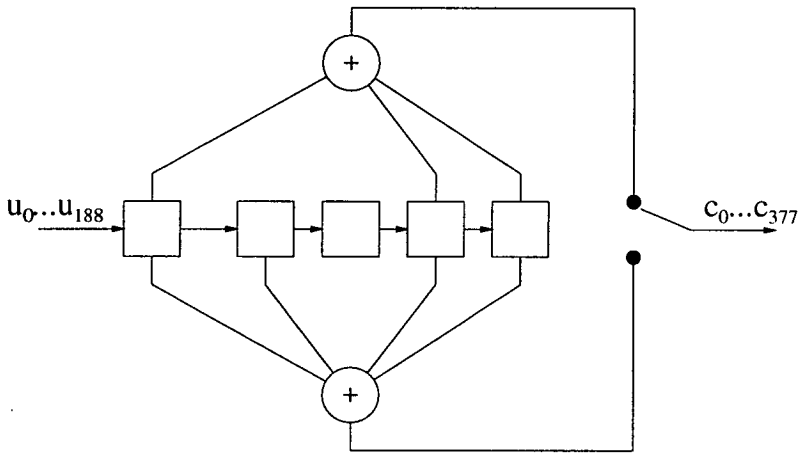


Figure 2.17: The convolutional encoder for GSM full rate speech

2.6.4 Channel specifications

To evaluate the performance of different receivers, eight different channel models have been developed by the COST 207 committee [3], [29]. Table 2.3 shows the definitions of six of these models.

In addition to the channel models in Table 2.3 there are two models based on a tapped delay line, these are the *equaliser model* (shown in Table 2.4) and the *simplified hilly terrain model* (shown in Table 2.5).

COST 207 channel models		
1	static two-path	$H(f) = 1 - bexp[-j2\pi(f - f_0)\tau]$
2	Rayleigh two-path	$H(f) = a - b\alpha exp[-j2\pi(f - f_0)]\tau$
3	flat Rayleigh fading	$H(f) = -b\alpha exp[-j2\pi(f - f_0)]\tau$
4	hilly terrain	$P(t) = \begin{cases} exp(-3.5t) & 0 < t < 2 \\ 0.1exp(15 - t) & 15 < t < 20 \\ 0 & otherwise \end{cases}$
5	rural area	$P(t) = \begin{cases} exp(-9.2t) & 0 < t < 0.7 \\ 0 & otherwise \end{cases}$
6	typical urban area	$P(t) = \begin{cases} exp(-t) & 0 < t < 7 \\ 0 & otherwise \end{cases}$

Table 2.3: COST 207 channel models

2.6.5 A GSM receiver structure

Figure 2.18 shows the communication between a GSM transmitter and a receiver. At the transmitter the signal is precoded (as described in Section 2.6.3) and placed into TDMA packets. These are then transmitted using GMSK modulation and the signal is passed through a channel.

The receiver structure consists of a GMSK demodulator which separates the signal into in-phase and quadrature-phase signals, the signals are synchronised and passed to the equalizer. As discussed earlier on in this chapter, the Viterbi equalizer requires an estimation of the CIR to perform equalization. The CIR is estimated by extracting the midamble, or sounding sequence, from the received packet, and comparing it with the known midamble which is known by the receiver.

The five bits at the start and the end of the received midamble are discarded to avoid transient situations [26], so only 16 received samples are used in the estimation of the CIR:

$$v(t) = (v_0, v_1, ..., v_{15}) \tag{2.23}$$

Tap number	Relative time (μs)
1	0.0
2	3.2
3	6.4
4	9.6
5	12.8
6	16.0

All taps have a doppler spectrum of
$S(f) = \frac{A}{\sqrt{[1-(f/f_d)^2]}}, -f_d < f < +f_d$
and the same relative power.

Table 2.4: COST 207 equalizer channel model

Tap number	Relative time (μs)	Average relative power (dB)
1	0.0	0.0
2	1.7	-2.0
3	12.0	-8.0
4	16.0	-7.0

Table 2.5: COST 207 simplified hilly terrain channel model

There are 26 reference samples, denoted as:

$$x(t) = (x_0, x_1, ..., x_{25}) \tag{2.24}$$

To estimate the CIR, a complex correlation between these sequences is performed:

$$\begin{aligned} R_{xv}(t) &= \int\int v(\tau + t - \alpha)h(\alpha)v^*(\tau)d\tau d\alpha & (2.25)\\ &= \int R_{vv}(t-\alpha)h(\alpha)d\alpha & (2.26)\\ &= R_{vv}(t) \otimes h(t) & (2.27) \end{aligned}$$

So, $R_{xv}(t)$ is the convolution between the impulse response $h(t)$ and the autocorrelation $R_{vv}(t)$. Since the midamble sequence have been chosen so that $R_{vv}(t)$ is a delta function:

$$R_{vv}(t) = \begin{cases} 1 & \text{if } l = 0 \\ 0 & \text{if } -5 \leq l \leq 5 \text{ and } l \neq 0 \end{cases} \tag{2.28}$$

So,

$$R_{xv}(t) \approx h(t) \tag{2.29}$$

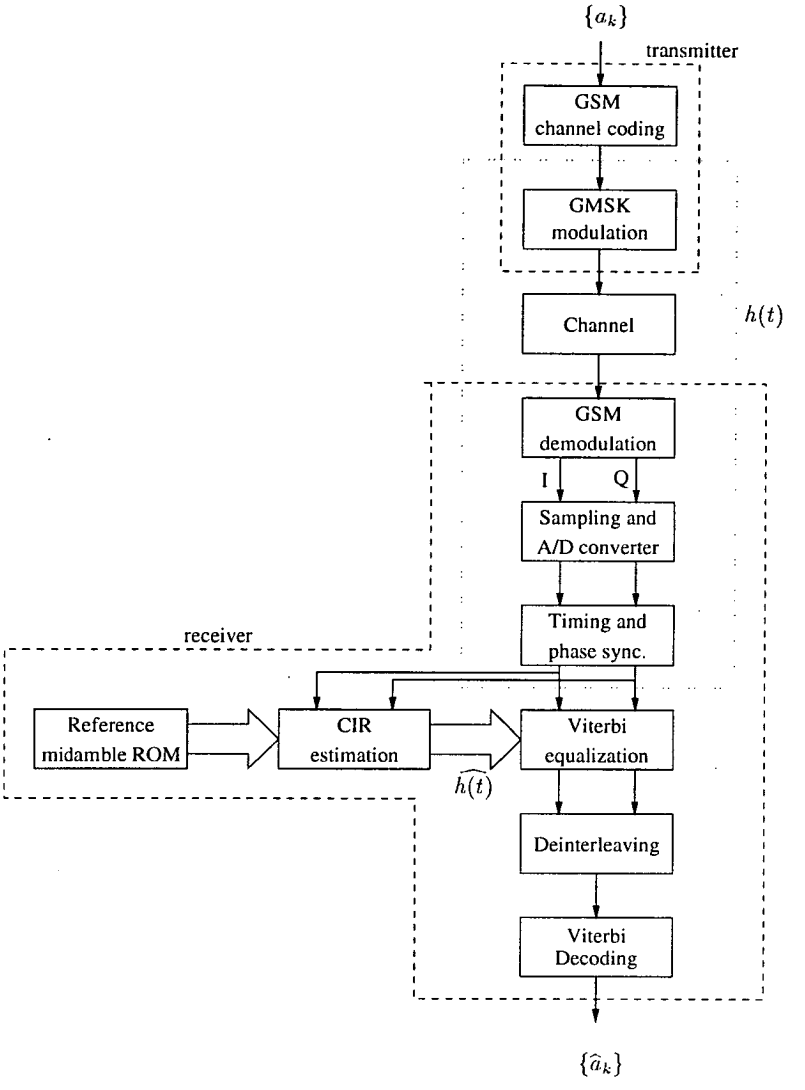


Figure 2.18: A block diagram of the GSM commucination system

The $h(t)$ estimate is then passed to the Viterbi equalizer which implements the algorithm as described in Section 2.4.2.3. Deinterleaving and Viterbi decoding (as described in Section 2.3) [30], [31] to remove the channel coding effects is performed to produce an estimate of the original sequence.

2.6.6 Performance

From simulations performed by the COST 207 group it was shown that a receiver structure using a 32 state Viterbi equalizer can compensate for echo delay up to $15\mu s$

in a severe multipath Rayleigh fading environment [3]. This corresponds to a vehicle speed of up to $200km/h$. The receiver structure discussed here assumes that the CIR is constant for the whole TDMA burst, adaptive Viterbi equalizers that adjusts the coefficients of CIR during the burst have been shown to have increased performance over non-adaptive implementations [32], [33]. It should be pointed out that the computation complexity of the VA for channel equalisation is quite high. Hence a number of alternative receiver structures have been proposed when low implementation costs are important [34].

2.7 Conclusions

This chapter has been a review chapter on the Viterbi algorithm. Convolutional codes have been described, and the trellis representation for a convolutional code has also been presented, this forms the basis for the Viterbi algorithm. The Viterbi algorithm has been described by a worked example, and it has been shown that the algorithm can be used to decode convolutional codes. In addition, the Viterbi algorithm can exploit the fact that only certain sequences are valid convolutional encodings, and correct errors in a convolutionally encoded signal. The amount of errors that can be corrected is determined by the complexity (or rather the constraint length) of the convolutional code. It has been shown that the size of the Viterbi trellis used to decode the code is exponentially proportional to the constraint length. This limits useful convolution codes to ones with relatively small constraint lengths.

It has been shown how the Viterbi algorithm can be extended to channel equalization to correct errors introduced by intersymbol interference (ISI) in quadrature amplitude modulation (QAM) systems. This can be achieved by describing the modulation system as a convolutional code, and designing a Viterbi decoder to decode this modulation system. In the GSM mobile telecommunications system, the modulation process is Gaussian minimum shift keying (GMSK), an extension of QAM, and one of the suggested receiver structures uses Viterbi equalization. This chapter has shown how a Viterbi equalizer fits into a GSM receiver system. The GSM system also convolutionally encodes the data before transmission, so a Viterbi decoder is also required in a GSM receiver. GSM provides a good example of these two applications of the algorithm.

This chapter has been mainly a review chapter. It has contained original work in the

form of the comparison of Viterbi decoding and equalization in Section 2.5. The complexity of the code (in the case of Viterbi decoding) and the modulation process (in the case of Viterbi equalization) have a direct (and large) impact on the size and complexity of the Viterbi implementation. It was noted that the main difference between the two applications is the calculation of the branch metrics, which is significantly more complicated for a Viterbi equalizer. Additionally, the incremental metrics for equalization have a larger resolution than the metrics for decoding, so the add-compare-select units will be larger (and probably slower). With the equalization applications, it was noted that the use of 4-QAM (rather that MSK or GMSK) will produce a heavily interconnected trellis structure. This is probably not desirable for a small device because it will result in a large amount of routing between trellis nodes.

Chapter 3

Redundant Number Systems for High Speed Arithmetic

3.1 Introduction

3.1.1 Motivation

This chapter discusses a technique which was to have an important effect on the research. In addition to investigating different signal processing algorithms, the early months of the research involved examining different techniques for producing "better" VLSI circuit designs. "Better" was defined as faster, more reliable, and lower power consumption. One of the techniques investigated for faster circuit design was redundant number systems. A paper by Srinivas and Parhi had a large impact on the direction of the work [5]. This paper described a fast adder structure, which used redundant number systems to gain a substantial speed increase over conventional methods. At the heart of this adder structure was a "sign-select circuit" for redundant numbers which suggested their suitability to a high-speed add-compare-select unit, which was very much applicable to the Viterbi algorithm, described in the previous chapter. It transpired later on during the research (in fact, after the initial implementations had been produced), that the Srinivas and Parhi design was a red herring, and the use of redundant number systems simply served to disguise the true source of the speed up. This chapter describes this investigation.

3.1.2 Overview

In the previous chapter the Viterbi algorithm (VA) for error correction and equalization was reviewed. The VA requires many addition operations to be performed at each

iteration. In this chapter a redundant number representation [4] will be examined as a possible technique for high speed arithmetic. The arithmetic operations which are required by the Viterbi algorithm: addition, subtraction, and multiplication will be discussed. Division using redundant number will also be examined. Although division is not used in the Viterbi algorithm, reviewing it will allow conclusions to be drawn on the usefulness of redundant number arithmetic in a wider range of digital applications. In Chapter 4 we will go on to propose a Viterbi decoder design which uses the techniques described in this chapter.

The most common operation in arithmetic is addition. In conventional number representations, the addition operation requires the propagation of a carry from the least significant digit, to the most significant digit, this places a limit on the speed of implementation.

Avizienis developed signed-digit number representations (SDNR) in 1961 [4]. In a conventional number representation of radix-r, where $r > 1$, each digit can assume only r values. In an SDNR with radix-r, each digit is allowed to assume more than r values. This means that a value has more than one representation in an SDNR.

For example, a radix-5 SDNR could have the following allowable digits $(\overline{4}, \overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3, 4)$, the overbar indicates that the digit has negative sign. So, the numbers 121 and $2\overline{3}1$ in this radix-5 SDNR are equal to each other, and represent the value 36 in decimal.

Redundant number systems have been used to develop high speed digital circuits [35], [36], [37]. This chapter will examine high speed arithmetic using redundant number systems. Radix-2 SDNR has possible digit values: $\{\overline{1}, 0, 1\}$ and is also known as signed binary number representation (SBNR). Each of the four main arithmetic operations will be explained and the improvement in speed offered by SBNR will be reviewed. Carry-save arithmetic will also be examined as a redundant number system, and its performance relative to SBNR will be assessed.

3.2 Choice of Radix

Before investigating Radix-2 SDNR, the choice of radix 2 redundant numbers should be examined. The main reason for choosing radix-2 was because of routing. Designs which use redundant number sytems with a radix other than 2 often suffer from large

amounts of routing, or require novel placement techniques to overcome this [38]. One of the main goals of this research was to produce two (or more) Viterbi algorithm designs, using automatic routing methods. While it would have been possible to base the designs on a redundant number system with a radix greater than 2, it was decided, for speed of implementation, that SBNR was the prefered choice.

3.3 Converting between SBNR and binary representation

The usefulness of SBNR for implementing arithmetic functions depends on the ease at which SBNR can be converted to and from conventional binary representations. In this section we will examine the techniques for these conversions.

3.3.1 Converting binary into SBNR

As discussed in Section 3.1, SBNR is similar to binary, except that there is an additional possible value for each digit, $\overline{1}$. It can be seen that conventional binary is a subset of SBNR so no conversion is necessary [5], [39].

However, this is not true for twos-complement representation. An n bit twos-complement word $Z = \{z_{n-1}, z_{n-2}, ..., z_0\}$ has the algebraic value, Z, shown in equation 3.1.

$$Z = \sum_{i=0}^{n-2} z_i 2^i - z_{n-1} 2^{n-1} \tag{3.1}$$

As equation 3.1 shows, if the MSB (the sign bit) is set, then the value of the word is negative. It can be seen from equation 3.1 that conversion into SBNR consists of changing the sign bit if it is a 1 into a $\overline{1}$. Otherwise the number is positive and no conversion is necessary.

Hence, conversion from twos-complement to SBNR can be achieved in constant time, independent of word length.

3.3.2 Converting SBNR into binary

Converting from SBNR into binary is not so straightforward. The most common technique [6] is to separate a SBNR number $Z = \{z_0, z_1, ..., z_{n-1}\}$ into two binary numbers $Z^+ = \{z_0^+, z_1^+, ..., z_{n-1}^+\}$ and $Z^- = \{z_0^-, z_1^-, ..., z_{n-1}^-\}$. $z_k^+ = 1$ if $z_k = 1$, and $z_k^+ = 0$, otherwise. Similarly, $z_k^- = 1$ if $z_k = \bar{1}$, and $z_k^- = 0$, otherwise: in effect the positive and negative components of the number.

The binary number Z^- is then subtracted from Z^+ using conventional twos-complement arithmetic to yield the twos-complement representation of the SBNR number Z.

If the subtraction is performed using carry-lookahead techniques the minimum time taken for the conversion would be $O(\log_2(n))$ [6].

It should be noted that the subtraction cells can be simplified since if $z_k^+ = 1$ then $z_k^- = 0$, similarly if $z_k^- = 1$ then $z_k^+ = 0$.

Many designs for converting SBNR into binary have been presented in the literature, these all have similar performance because they rely on carry-lookahead techniques [40], [41]. However, it should be noted that if the SBNR number is produced in a digit-by-digit manner then the conversion to conventional binary can be acheived by "on-the-fly" conversion [42].

3.4 Fast addition using redundant number systems

It has been claimed in the literature that redundant number systems have a major speed improvement over conventional number systems for addition [4], [5]. In this section addition using redundant numbers will be described and a fast twos-complement adder will be presented, based on a design from [5], which does not require the use of redundant number systems.

3.4.1 Totally parallel addition

Figure 3.1 shows a schematic for a totally parallel addition circuit using SDNR. The two cells ① and ② implement the following two equations,

$$rt_{i+1} + w_i = z_i + y_i \tag{3.2}$$

$$s_i = w_i + t_i \tag{3.3}$$

where, for $r = 3$ (for example),

$$\begin{aligned} z &\in \{\overline{2}, \overline{1}, 0, 1, 2\} \\ y &\in \{\overline{2}, \overline{1}, 0, 1, 2\} \\ t &\in \{\overline{1}, 0, 1\} \\ w &\in \{\overline{1}, 0, 1\} \\ s &\in \{\overline{2}, \overline{1}, 0, 1, 2\}. \end{aligned}$$

$Z = (z_{n-1}, z_{n-2}, ..., z_0)$ and $Y = (y_{n-1}, y_{n-2}, ..., y_0)$ are the two numbers to be summed; r is the radix of the number system; w_i is the intermediate sum; and t_i is the transfer digit from the previous stage.

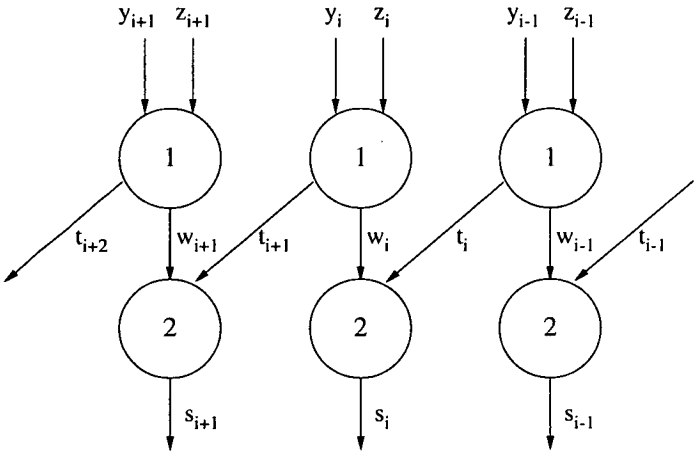


Figure 3.1: A totally parallel adder for SDNR

As the diagram shows each digit in the input word can only affect two digits in the output word, thus eliminating the need for a carry propagation chain. However, for the adition of digits z_i and y_i to be totally parallel, the following two conditions must be

satisfied [4]:

1. The sum digit s_i is a function only of z_i, y_i and the transfer digit t_i.
2. The transfer digit t_{i+1} is a function only of z_i and y_i.

This condition will be satisfied if the range of values of s_i in equation (3.3) does not exceed the range of values in z_i and y_i in equation (3.2) [4]. However, this does not allow totally parallel addition for SBNR. Almost parallel addition for SBNR can be accomplished by allowing the input signal to propagate over a range of three output digits. This means that the following three transfer steps are needed.

$$rt'_{i+1} + w'_i = z'_i + y'_i \tag{3.4}$$

$$rt''_{i+1} + w''_i = w'_i + t'_i \tag{3.5}$$

$$s'_i = w''_i + t''_i, \tag{3.6}$$

where, for $r = 2$,

$$\begin{aligned} z'_i &\in \{\bar{1}, 0, 1\} \\ y'_i &\in \{\bar{1}, 0, 1\} \\ t'_i &\in \{0, 1\} \\ w'_i &\in \{\bar{2}, \bar{1}, 0\} \\ t''_i &\in \{\bar{1}, 0\} \\ w''_i &\in \{0, 1\} \\ s'_i &\in \{\bar{1}, 0, 1\}. \end{aligned}$$

This structure is shown in Figure 3.2, and Figure 3.3 shows each of the three subcells.

These digits are encoded using multiple binary bits, which are indicated in Figure 3.3 by a $*$ superscript. Table 3.1 shows how these values are represented.

So, addition can be achieved with a very small delay, independent of word length, but only if the representation remains as SBNR.

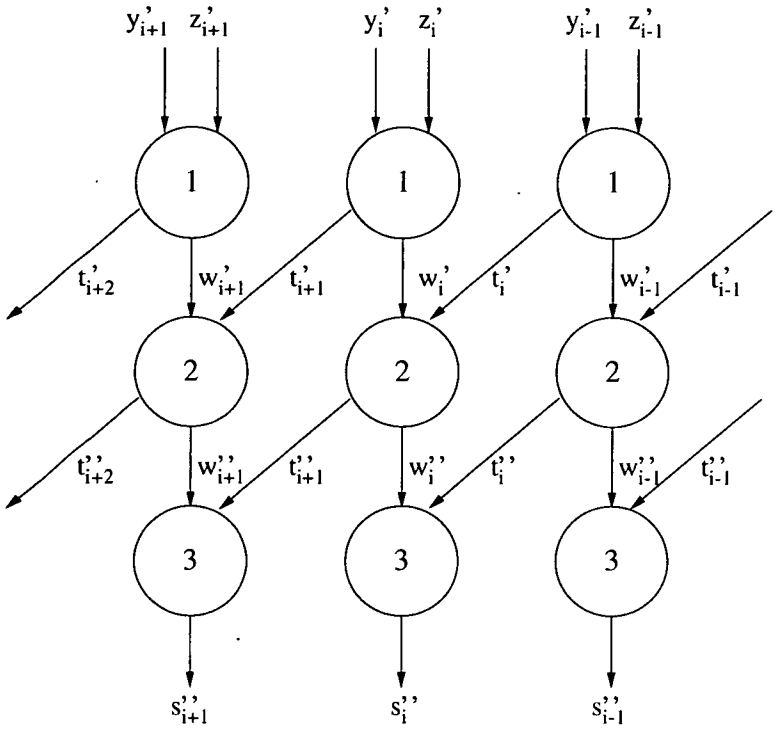


Figure 3.2: An almost totally parallel adder for SBNR

3.4.2 A fast twos-complement VLSI adder design

In 1992 Srinivas and Parhi published a fast VLSI adder architecture for summing two 32-bit twos-complement words [5]. The authors claimed at the time that this design was $20 - 28\%$ faster than the fastest known binary lookahead adder designs.

Figure 3.4 shows Srinivas and Parhi's fast adder design. Since conventional binary is a subset of SBNR, two 32-bit twos-complement numbers $\{a_{31} \ldots a_0\}$ and $\{b_{31} \ldots b_0\}$ can be summed as SBNR numbers (without carry propagation) to yield an 32-digit SBNR number. This is then converted into back into twos-complement format.

In Figure 3.4 it can be seen that the main part of the adder circuit concerns the conversion of the SBNR word into twos-complement format. The circuit consist of converter units which convert the SBNR number in 8 digit blocks. These are basically ripple adders which sum the positive and negative components of the SBNR word to produce a twos-complement result. To avoid having to propagate the carry the full length of the word, two converters are used for each 8 digit block in a similar manner to a carry-select adder. The choice of which converter to take the output from is made by the sign of the previous 8 digit block. This sign is produced by a novel sign-select circuit

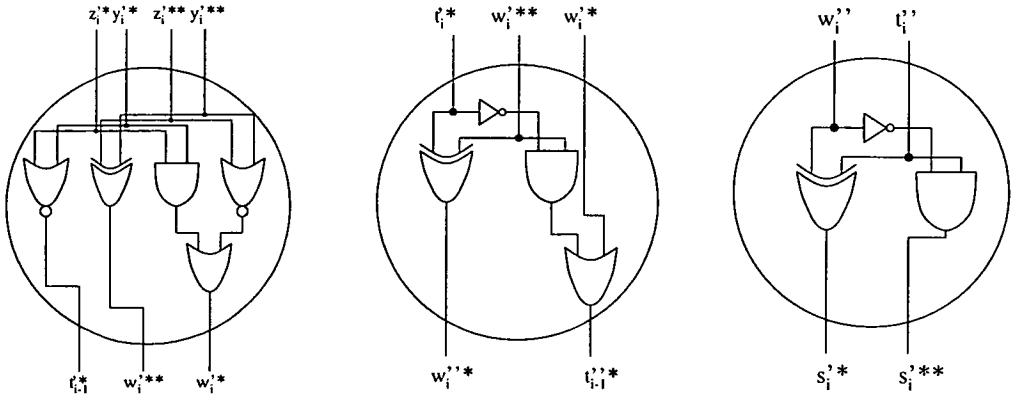


Figure 3.3: The three subcells for SBNR parallel addition

which Srinivas and Parhi proposed for detecting the sign of a SBNR number. Their sign-select cicuit is shown in Figure 3.5.

This remainder of this section proposes a simple static-logic binary-tree carry generator to support high-speed adder implementations with a delay of $\lceil \log_2(N) \rceil + 2$ gates based on the Srinivas and Parhi design.

In examining this design it was observed that the initial SBNR addition block of [5] can be replaced by $\mathrm{xor}(a_i, b_i)$, $\mathrm{xnor}(a_i, b_i)$ and $and(a_i, b_i)$ (conventional propagate and generate signals for a carry-lookahead adder); and the remaining circuit can thus operate directly on twos-complement representation producing the inverse of the sum (easily corrected). With trivial changes, $\mathrm{nand}(a_i, b_i)$ can be used instead of $\mathrm{and}(a_i, b_i)$ and so reduce the gate count of the design by $3N - 1$ and the critical path by 2. More significantly, the design is thus equivalent to the hybrid carry-lookahead/carry-select architecture shown in Figure 3.6 and it is a *purely* twos-complement architecture whose high speed has no reliance upon SBNR.

In Figure 3.6, the pairs of 8-bit additions are performed using carry-lookahead logic and their outputs selected according to the true value of the corresponding carry-in. Unlike the usual carry-select architecture [43], the carry terms are generated by independent logic (the *carry-generator tree*) rather than by using the results of successive adder sub-sections; this is where the speed advantage lies since the Srinivas and Parhi design uses a tree structure to achieve $O(\log(N))$ delay.

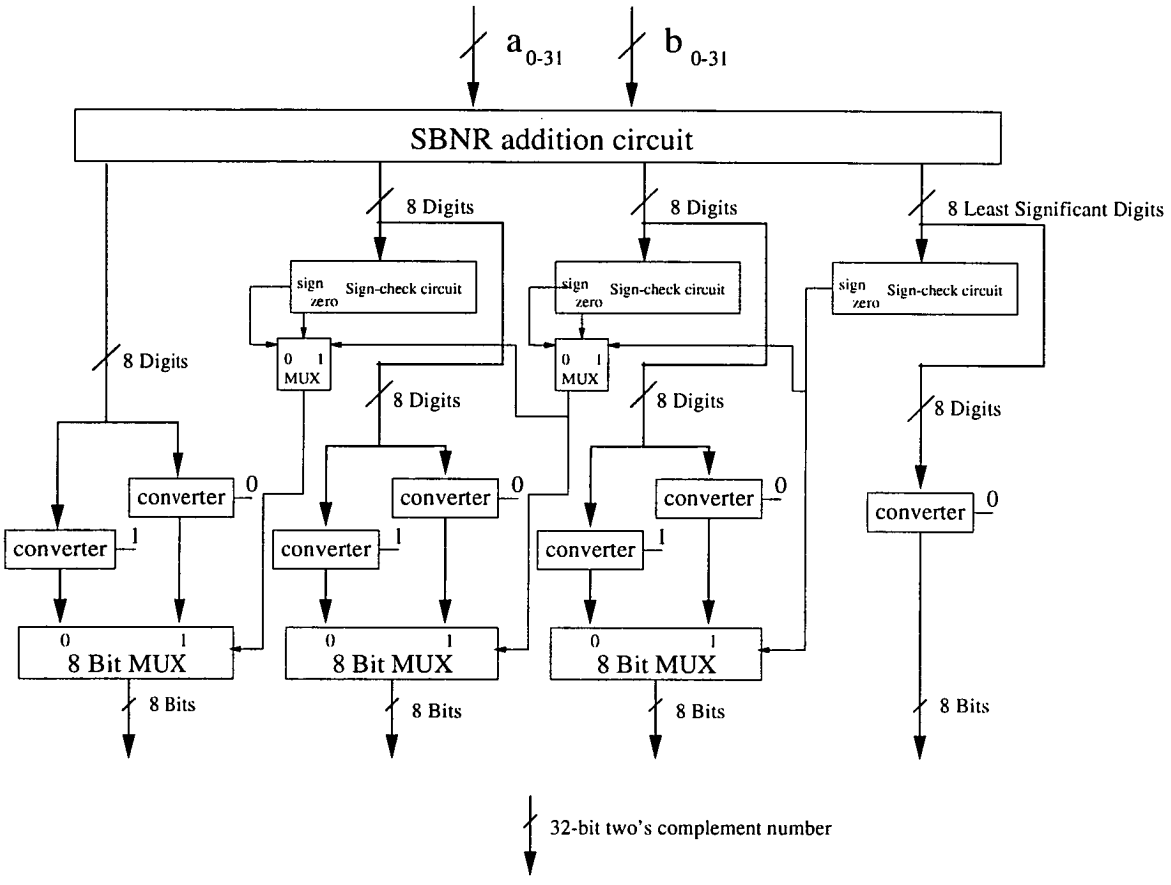


Figure 3.4: The Srinivas and Parhi's fast adder

Digit set $\{\bar{1}, 0, 1\}$		
x	x^{**}	x^{*}
$\bar{1}$	1	1
0	0	0
1	0	1

Digit set $\{\bar{2}, \bar{1}, 0\}$		
x	x^{**}	x^{*}
$\bar{2}$	1	0
$\bar{1}$	0	1
0	0	0

Digit set $\{\bar{1}, 0\}$	
x	x^{*}
$\bar{1}$	1
0	0

Digit set $\{0, 1\}$	
x	x^{*}
1	1
0	0

Table 3.1: Representation of redundant digits

3.4.2.1 Carry-generator tree

The carry-generator is based on the novel 8-digit SBNR sign-checking circuit presented in [5] (Figure 3.5). There are two inputs to the circuit $\{Z_{N-1}, ...Z_0\}$ and $\{S_{N-1}, ...S_0\}$ which in [5] are generated by the addition of 2 twos-complement numbers to form an SBNR sum and interpreted as *zero* and *sign*. We have replaced this pre-logic with $Z_i = \text{xor}(a_i, b_i)$ and $S_i = \text{and}(a_i, b_i)$ (conventional carry-propagate and carry-generate signals in "carry-lookahead" terminology), and the remainder of

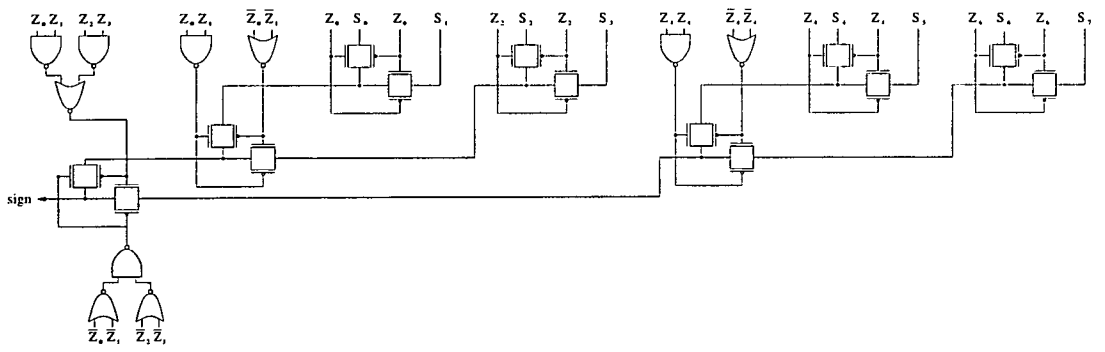


Figure 3.5: The 8-digit SBNR sign select circuit

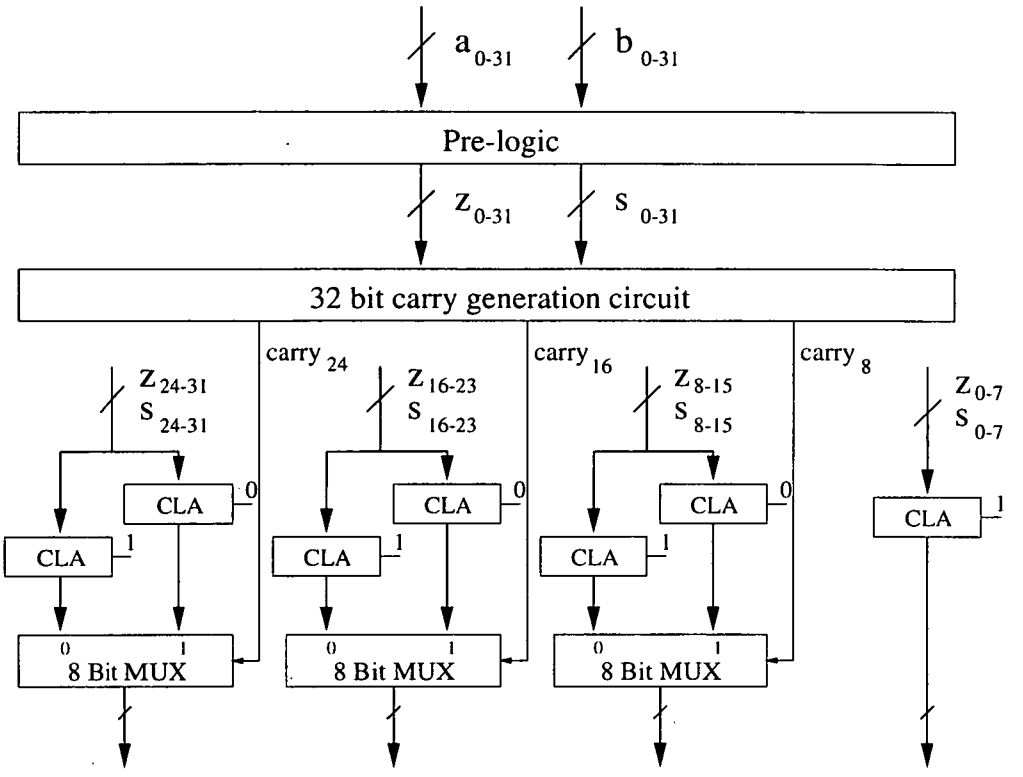


Figure 3.6: The hybrid carry-lookahead/carry-select logic

the circuit can be described in the following way. Define

$$Z_{j,k} = \text{and}(Z_j, Z_{j-1}, ...Z_{k+1}, Z_k) \tag{3.7}$$

and $C_{j,k}$ as the carryOut of the addition of bits j through to k with a zero carryIn (giving $C_{j,j} = S_j$).

We build a carry-generator tree using the following relations at each node:

$$\text{mux}(Z_{p,q}, C_{p,q}, C_{r,p-1}) = C_{r,q} \tag{3.8}$$

$$\text{nand}(Z_{p,q}, Z_{r,p-1}) = \overline{Z_{r,q}} \tag{3.9}$$

$$\text{nor}(\overline{Z_{p,q}}, \overline{Z_{r,p-1}}) = Z_{r,q} \tag{3.10}$$

which effectively combines two adjacent adder sections. The first relation states that the carryOut of combined sections is the same as the carryOut of the more significant section *unless* its inputs are such that its carryIn propagates through it completely. This is so when each bit has a "0" and a "1" input and the resulting carryOut will then

be the carryOut of the less significant section. (In practice, we propagate $\overline{carry}$ to use an initial nand to form $\overline{S_j}$). The second and third relations maintain the function *each bit has a "0" and a "1" input* for the combined inputs at each node; these two relationships are not needed at the leaf nodes.

By applying these relations first to pairs of inputs, and then to pairs of the outputs of such pairs, etc, we build up a simple binary tree. This naturally produces carryOut on boundaries for successive powers of 2 (2, 4, 8, 16 etc); to obtain carryOut on intermediate boundaries, we augment the tree with other nodes (using the same relationship). Figure 3.7 shows this for a 8-bit carry generation tree with 2-bit boundaries. Note that the generation of carry-bit 6 from inside the tree structure shows how the critical delay on each term can be limited to $\lceil log_2(N) \rceil$. It is believed this was not known to Srinivas and Parhi since it allows the use of 4-bit adder sections in their 32-bit example which moves the critical path from the carry-lookahead adders to the carry-generate tree and thus would have reduced their critical path delay.

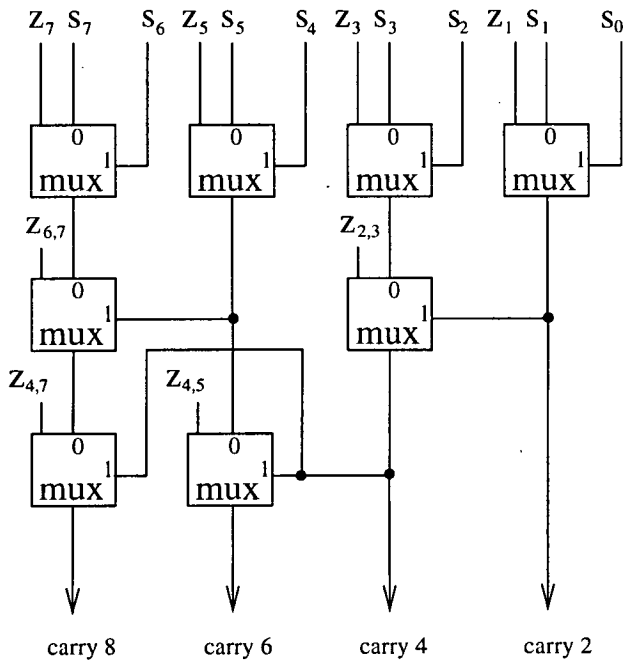


Figure 3.7: An 8-bit carry-generation tree augmented for 2-bit boundaries

The optimal decomposition for this architecture is found by determining the delay through the binary-tree carry generator and then selecting adder sub-sections (not necessarily carry-lookahead) which complete within that time. The only problem is fan-out, but this will be constrained in practice since larger adder tree sizes will lead to larger carry-boundaries.

The carry-generator tree is based upon two-input gates and multiplexors and, we believe, is suitable for the implementation of large adders using circuit techniques such as complementary pass-transistor logic [7] (CPL) which provide both high speed and low power. With the improvements outlined above, this circuit technique leads to a critical path delay of $\lceil \log_2(N) \rceil + 2$ gates. Thus a 32-bit adder, will have a critical path of 7 gate delays, compared to 12 as reported in [5].

3.4.2.2 Discussion

The underlying architecture can now be seen to be similar to that of Lynch and Swartzlander also published in 1992 [44], [45]. Their design creates a tree structure using 4-input dynamic-logic Manchester carry chains and uses ripple adders in place of the carry-lookahead adders. Like Srinivas and Parhi their "modified" tree produces carry signals only on 8-bit boundaries. In contrast, the design presented here provides a simple static-logic implementation based on two-input nodes with greater flexibility in choosing the carry boundaries and with no requirement for a system clock. This design also has similar performance to that reported by Suzuki *et al* but with fewer gates [46]. Finally, it should be noted that the improved circuit can form the basis for a fast SBNR to twos-complement conversion.

3.5 Multiplication using SBNR

As has been shown so far in this chapter, addition using SBNR can be achieved with a relatively small delay that is independent of word length. However, we have also shown that the conversion of SBNR into conventional binary representation is similar to a conventional binary addition operation, so SBNR is not suitable as the basis of fast twos-complement addition circuits with two operands as claimed in [5].

The advantages of SBNR can be seen in algorithms which either do not require the result to be converted back into binary representation, or where multiple addition operations are performed before converting into binary.

The most common arithmetic function in VLSI which consists of many addition operations is multiplication. In this section we will outline a two techniques for multiplying binary numbers using SBNR.

3.5.1 Multiplication using SBNR partial products

The most straightforward technique of performing multiplication is to use SBNR partial products. Figure 3.8 shows the concept on two 4-digit SBNR numbers. Each of the n digits in the multiplier are multiplied by the multiplicand to produce n partial products. This is shown on the left hand side of Figure 3.8. These n partial products are summed using a binary tree of SBNR adders. The final sum represents the product of the two inputs.

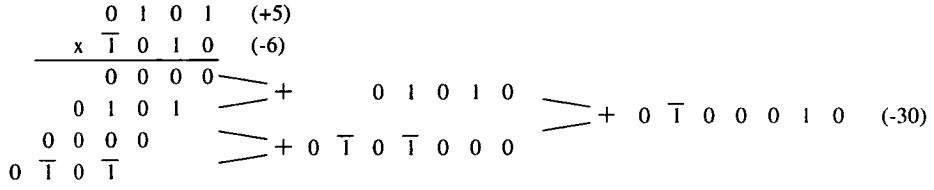


Figure 3.8: SBNR multiplication using SBNR partial products

In this example all of the partial products are implemented using SBNR. This produces a simple regular structure which is shown in Figure 3.9 for a multiplier and multiplicand of 4 bits. Because the structure is based on a binary tree, the delay of the multiplier is $O(\log(n))$. If this structure was to be used for binary multiplication, there would be an additional delay for conversion of SBNR to binary at the output.

In [6] a twos-complement multiplier based on this design was shown to have a gate count of $O(n^2)$ and area requirement of $O(n^2 \log n)$, the same as a Wallace tree multiplier, but with a less complicated structure.

3.5.2 SBNR multiplication using the modified Booth algorithm

A common technique for reducing the delay and complexity of twos-complement multipliers is to use the modified Booth's algorithm [47] which halves the number of partial products to be summed.

The algorithm examines the bits of the multiplier in triplets with one bit overlap between each adjacent triplets. A partial product is produced by performing an operation on the multiplicand depending on the pattern of the triplets.

For a multiplicand $A = \{a_0, a_1, ..., a_{n-1}\}$ and a multiplier $B = \{b_0, b_1, ..., b_{m-1}\}$, the

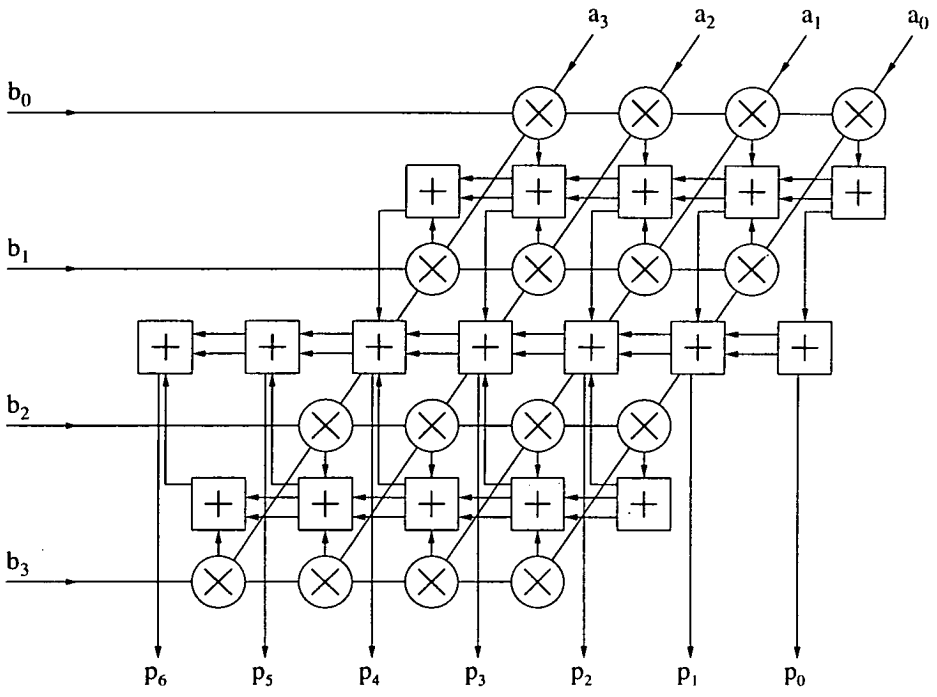


Figure 3.9: SBNR multiplier using SBNR partial products

operations performed to produce each partial are outlined in Table 3.2.

Multiplier triplet			Operation
a_{2i+1}	a_{2i}	a_{2i-1}	
0	0	0	$+0 \times B$
0	0	1	$+B$
0	1	0	$+B$
0	1	1	$+2 \times B$
1	0	0	$-2 \times B$
1	0	1	$-B$
1	1	0	$-B$
1	1	1	$+0 \times B$

Table 3.2: Booth's algorithm for generating partial products

It can be seen that the multiplier can only be divided into triplets in this way if n is odd. If n is even, then the multiplier is sign extended. A 4 bit Booth's multiplier is shown in Figure 3.10, note the sign extension.

It should be noted from Table 3.2 that the generation of the partial products is equivalent to re-coding the multiplier as a radix-4 redundant number with the values $\{\overline{2}, \overline{1}, 0, 1, 2\}$, then multiplying the multiplicand by the digits of the recoded multiplier in the same

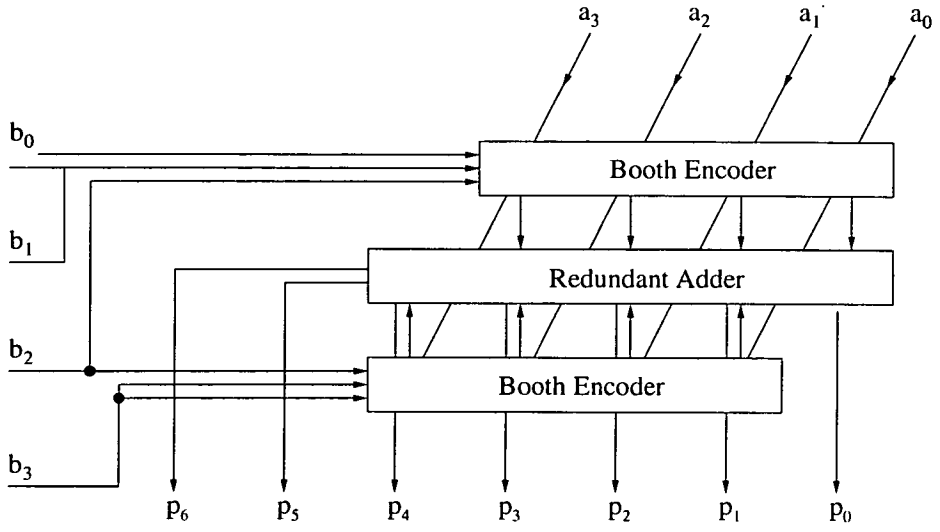


Figure 3.10: SBNR Booth's multiplier

way as described in Section 3.5.1. The number of partial products is halved by this recoding [48].

The main disadvantage with the modified Booth's algorithm for twos-complement numbers is that the generation of the partial products involves a negation operation. For twos-complement numbers a negation operation means inverting the bits and adding 1; this involves carry propagation delay which is dependant on the word length n. This means that there is a greater delay for generating the partial products than with the technique outlined in Section 3.5.1, which can be performed in constant time.

In addition to the delay of the negation operation, power consumption is also a problem. To negate a twos-complement word *all* of the bits need to be inverted, this results in a high switching activity [39].

SBNR can be used to lower the delay and switching activity of the negation operation since to negate a SBNR number, only the signs of those digits which are 1 or $\overline{1}$ need to be inverted. Also, this inversion can be performed in constant time since there is now no carry propagation. It has been shown that SBNR can reduce the average switching activity by 87.5% for partial product generation for 8-bit numbers [39].

3.6 Division using SBNR

In 1958, Robertson presented a new digital division method using redundant number systems as an alternative to non-restoring division using conventional arithmetic [49].

In this section, restoring and non-restoring division is reviewed and Robertson's technique will be shown.

3.6.1 Restoring division

Restoring division is based on the conventional approach to division, it is a recursive algorithm defined by equation 3.11

$$X^{j+1} = r \times (X^j - q_j D), j = 0, 1, ..., m-1, \tag{3.11}$$

where X^0 is the dividend, X^j is the partial remainder for iteration j; $q_j \in \{0, 1\}$ is the jth digit in the quotient; m is the number of digits, radix-r , in the quotient; X^m is the remainder; and D is the divisor, bit aligned with X^0. So the quotient, Q, is:

$$Q = \sum_{i=0}^{m} r^{-i} q_i \tag{3.12}$$

At successive iterations, if $X^j > D$ then $q_j = 1$, otherwise $q_j = 0$.

The main drawback with this approach is that the test $X^j > D$ is slow and since it has to be performed at every iteration it slows down the whole division operation.

3.6.2 Non-restoring division

Non-restoring division removes the test $X^j > D$ and replaces it with a test on the sign of X^j. The algorithm is similar to that for restoring division and is described by equation 3.13:

$$X^{j+1} = r \times (X^j - q_j D), j = 0, 1, ..., m-1 \tag{3.13}$$

where X^0 is the dividend X^j is the partial remainder for iteration j and $-2D < X^j < 2D$; $q_j \in \{\bar{1}, 1\}$ is the jth digit in the quotient; m is the number of digit, radix-r , in the quotient; X^m is the remainder; and D is the divisor, bit aligned with X^0. So the

quotient, Q, is:

$$Q = \sum_{i=0}^{m} r^{-i} q_i \tag{3.14}$$

From [50], at each iteration, the quotient digit q_j is selected such that the partial remainder X^{j+1} lies between $-rD$ and rD. For radix-2 division, this means that the quotient digits can assume the values 1 or $\overline{1}$. In can be shown that if the digits can only have values $1, \overline{1}$, then the quotient Q has a unique representation for all values so this number system is not redundant.

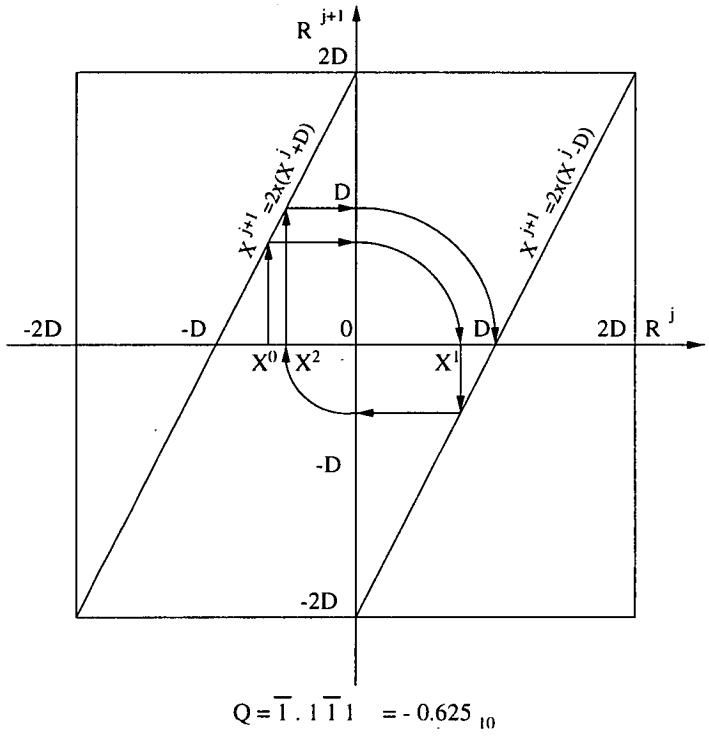


Figure 3.11: A Robertson diagram for non-restoring division

Figure 3.11 shows a Robertson diagram which illustrates this division algorithm. The two diagonal lines at $X^{j+1} = 2(X^j + N)$ and $X^{j+1} = 2(X^j - N)$, represent a quotient digit of -1 and $+1$ (i.e. an addition of $-D$ and $+D$) respectively.

In this example, the dividend X^0 is $-\frac{5}{8}D$. At each iteration, a vertical line from X^j is drawn, it either intersects with the left hand line, in which case $q_j = \overline{1}$, or the right hand line, $q_j = 1$. The division stops when the $j = m$ or when $X^j = 0$, the latter being the case in this example.

It can be seen that for radix-2, the value of the quotient digit q_j is equal to the sign

of the partial remainder X^j. However, because the allowable quotient digits are $1, \bar{1}$, the evaluation of the sign of X^j requires a carry-propagation of $O(n)$ (where n is the word size of the divider) which slows down the iterative step. The whole non-restoring division operation is $O(nm)$.

3.6.3 Non-restoring division with redundant number systems

If the quotient and the partial remainders are stored using SBNR, which allows the digits $\{\bar{1}, 0, 1\}$, the Robertson diagram for the example above becomes Figure 3.12.

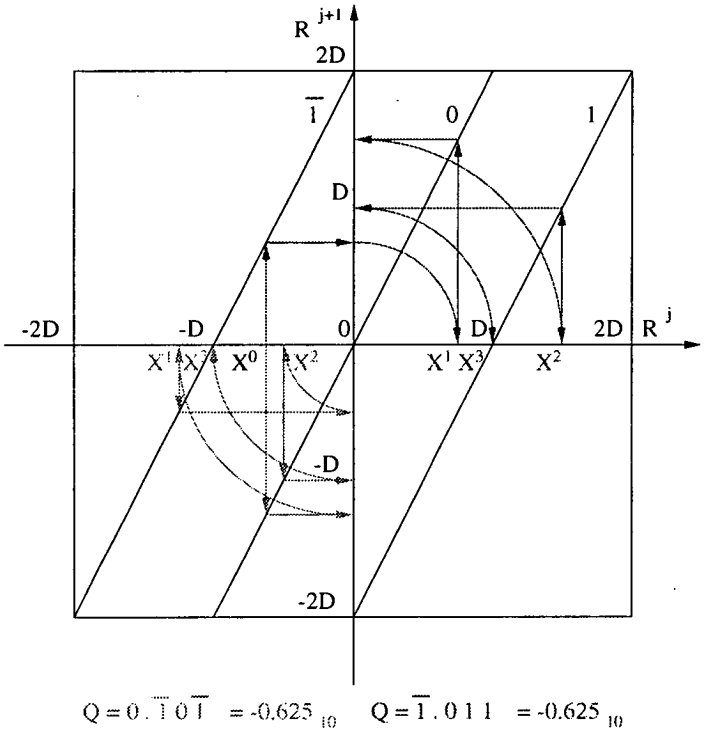


Figure 3.12: A Robertson diagram for non-restoring division with SBNR

As the diagram shows, there is now a third diagonal line, through the origin, which represents a quotient digit of 0. Now, the quotient is evaluated in the following manner:

$$\begin{array}{lll} \text{if} & X^j < 0 \text{ then} & q_j = \bar{1} \\ \text{if} & -D < X^j < D \text{ then} & q_j = 0 \\ \text{if} & X^j > 0 \text{ then} & q_j = 1. \end{array}$$

Since there is an overlap in the diagonal lines, when $-D < X^j < D$ the q_j can be evaluated incorrectly and the final result will still be correct. This is shown in the

diagram. The lightly shaded line represents the correct evaluations, and results in $Q = 0.\bar{1}0\bar{1}$ (-0.625_{10}). The darker shaded line shows what would happen if q_0 is evaluated incorrectly as $\bar{1}$. The final result is $Q = \bar{1}.011$ (-0.625_{10}) which is also correct.

It is sufficient to examine the 3 most significant digits of the X^j to the correct evaluation of the quotient [50], this eliminates the need for carry propagation so the delay for the non-restoring division using SBNR is $O(m)$.

3.7 Comparision using redundant number systems

While not strictly an arithmetic operation, the comparision of two numbers will be briefly described, due to its relavence to the Viterbi algorithm.

A block diagram for a 4-bit binary tree based twos-complement comparator is shown in Figure 3.13. The first level, comprised of type ① cells determines whether the corresponding bits in words a and b are equal.

The second and subsequent levels are comprised of type ② cells which take the results from two adjacent cells and perform the following:

- If most significant inputs are equal then the outputs are the least significant inputs.
- If most significant inputs are not equal then the outputs are the most significant inputs.

The delay through the binary tree comparator is $O(log_2 N)$ where N is the word length.

Because of the redundancy inherent in SBNR, the same technique cannot be extended to redundancy number systems. For example, if we wanted to compare two SBNR numbers: $a = 10\bar{1}0$ and $b = 0111$. The MSD comparision implies that $a > b$. However because, this is incorrect as $b > a$. The method used for comparing 2 SBNR numbers is to perform a subtraction operation, and then to use a sign select circuit, such as the one in Figure 3.5, to compute the boolean result. This comparision technique has a delay of $O(log_2 N) + 3$ where N is the word length, and the 3 comes from the delay through the SBNR addition circuit.

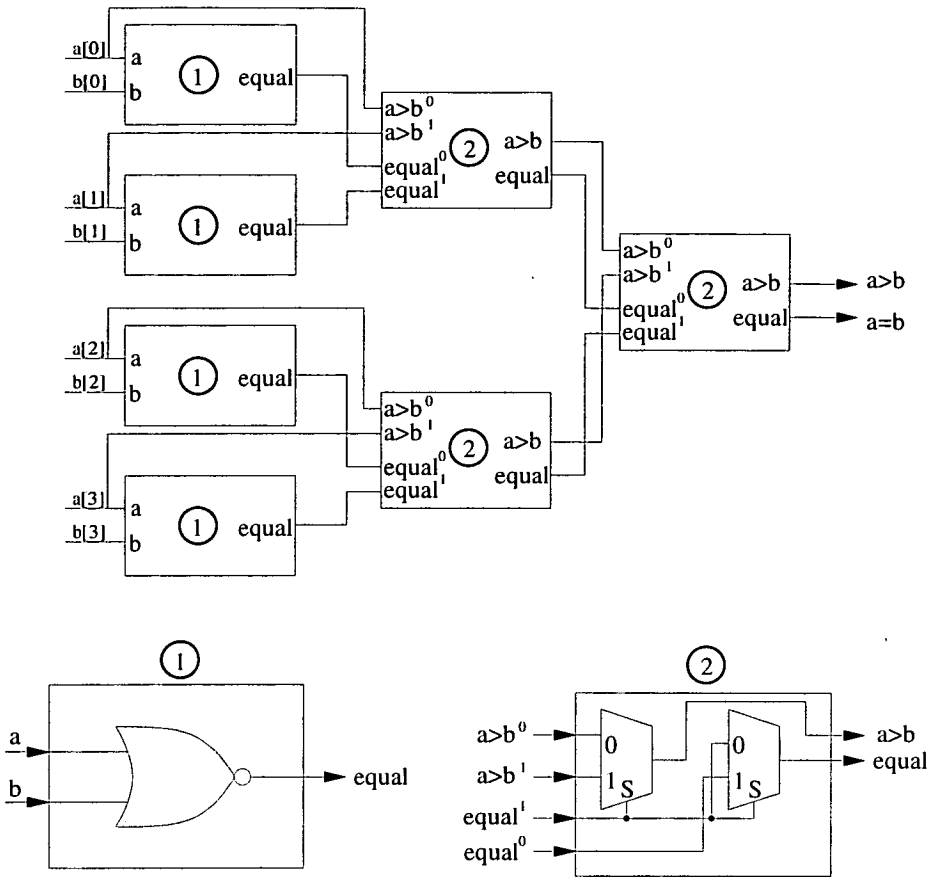


Figure 3.13: A 4-bit binary tree based twos-complement comparison circuit

3.8 Carry-save arithmetic as a redundant number system

A popular redundant number system which is used to produce high speed designs is *carry-save* arithmetic. A ripple adder is a *carry-propagate* adder, this means that the carry signal is passed from each full adder to the next most significant full adder in the current level. In a carry-save adder, the carry signal is passed to the next most significant adder in the *next* level. This is shown in Figure 3.14.

It can be seen that carry-save arithmetic is a redundant number system with each digit represented by binary digits: *carry* and *sum*. This means that each digit can assume 4 values $\{0, 1, 2, 3\}$. In this section we will examine how carry-save arithmetic compares with SBNR as technique for producing high speed arithmetic VLSI designs.

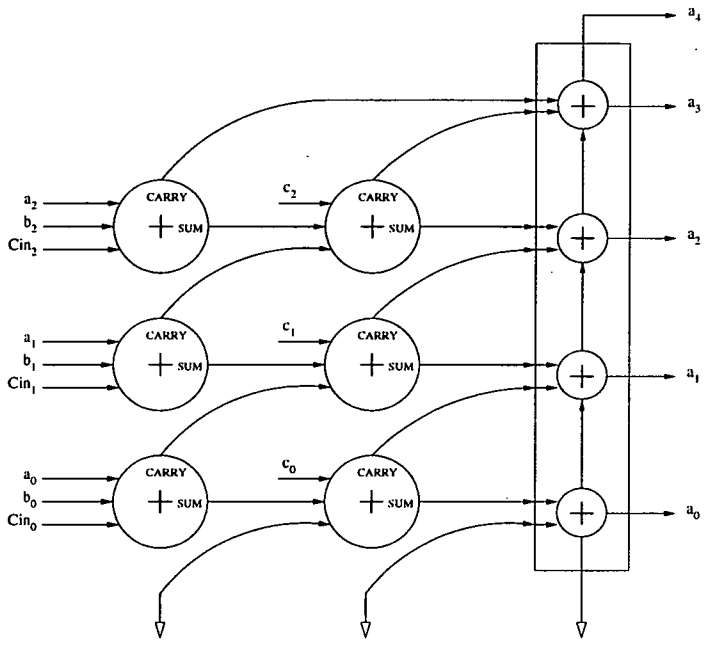


Figure 3.14: A carry-save adder for summing 3 three bit numbers

3.8.1 Converting between binary and carry-save arithmetic

The possible values for binary digits are $\{0, 1\}$ which is a subset of the possible digit values for carry-save arithmetic. This means that, as with SBNR (Section 3.3), no conversion from binary to carry-save is necessary.

Converting from carry-save to binary is also very similar to converting from SBNR to binary. A carry-save number Z, of $n + m$ digits, can be described as a list of pairs: $((c_{-n}, s_{-n}), (c_{-n+1}, s_{-n+1}), ..., (c_{m-1}, s_{m-1}))$. The arithmetic value of Z is:

$$\begin{aligned} Z &= \sum_{i=-n}^{m-1} s_i 2^i + \sum_{i=-n}^{m-1} c_i 2^{i+1} \\ &= S + C, \end{aligned} \tag{3.15}$$

where $S = (s_{-n}, s_{-n+1}, ..., s_{m-1})$, $C = (c_{-n}, c_{-n+1}, ..., c_{m-1})$.

It can be seen from equation 3.15 that conversion from carry-save to binary is equivalent to a twos-complement addition operation of two binary words formed from the *sum* and *carry* bits.

As discussed in Section 3.3, the conversion from SBNR to binary can be optimised since the pairs of bits being subtracted cannot both be equal to 1. With carry-save

arithmetic, both the *carry* and *sum* bits can be either 0 or 1 irrespective of the value of the other, so the operation cannot be optimised.

3.8.2 Carry-save addition

As shown in Figure 3.14, only one adder delay is required to sum two carry-save numbers; this is because there is no carry propagation. It is possible to design a full adder circuit with only one gate delay, so the addition of two carry-save numbers can take only one gate delay, independent of word length. Figure 3.2 shows that the addition of two SBNR numbers requires at least three gate delays because of the transfer digit propagation. So for addition operations, carry-save arithmetic is smaller and faster than SBNR.

3.8.3 Carry-save multiplication

As the diagram in Figure 3.14 shows, carry-save arithmetic is useful for cascaded adders because the carry-propagation only happens once at the output. This implies that carry-save would be useful for multiplication circuits.

The first SBNR based multiplier that was examined in Section 3.5.1 took two twos-complement number as input: n partial products in SBNR were formed by multiplying each of the n digits of the multiplier with the multiplicand. These partial products were then summed and the final results was converted back into twos-complement form.

A carry-save multiplier could be designed in exactly the same way, with the partial products produced in carry-save format, rather than SBNR. As noted in Section 3.8.2, carry-save adders are slightly faster and smaller than SBNR adders so a carry-save multiplier based on this design would also be faster and smaller that a SBNR multiplier.

The second type of SBNR multiplication circuit examined in Section 3.5.2 used Booth's algorithm to reduce the number of partial products formed. It was noted that Booth's algorithm required the multiplicand to be multiplied by $-2, -1, 0, 1$ and 2. The multiplication by 0 and 2 are trivial, requiring the resetting of all the bit and a left shift operation respectively. However, the negating of the multiplicand is not as straightforward. For twos-complement word, a negation operation requires all of the bits to be inverted and 1 to be added to the result. This involves carry propagation. It was noted

that negating an SBNR number involves only inverting those digits which are $\overline{1}$ or 1 so the operation is independent of word length.

The negation of a carry-save word is not trivial. Since the allowed digit values are $\{0, 1, 2, 3\}$ it is not possible to invert a single digit without taking into account the rest of the word. So it appears that a carry propagation the full length of the word is required to negate a carry-save number.

However, if our input words are twos-complement binary then the Booth's recoding stage involves multiplying a twos-complement word by $-2, -1, 0, 1$ or 2 to produce a carry-save number. This can be achieved by inverting all of the twos-complement bits, and adding 1 using carry-save addition. Because carry-save addition has no carry propagation, this operation is also independent of word length.

Thus, a multiplier based on the modified Booth's algorithm would be faster and smaller using carry-save arithmetic than SBNR. However, it should be noted that the negation operation of SBNR has less switching activity than the negation of a twos-complement number [39] so the SBNR multiplier may have lower power consumption inspite of its larger area.

3.8.4 Carry-save division

As discussed in Section 3.6, the advantage of non-restoring division over restoring division is that a "greater than" test is replace by a sign test. The advantage of non-restoring division with redundant arithmetic, as proposed by Robertson, is that the sign test of the partial remainder does not need to be accurate [49]. This is due to the additional redundancy introduced by an extra quotient digit.

The Robertson diagram in Figure 3.12 shows that for each quotient digit q there is a line $X^{j+1} = 2(X^j + qD)$ which represents the subtraction of qD from the current partial remainder to give the next partial remainder. This means that the partial remainder lies between $-D$ and $+D$ and an incorrect evaluation of the sign test is performed, then this can be corrected because the digits $\overline{1}$ and 1 are the inverse of each other.

With carry-save arithmetic the quotient digits are all positive so we cannot exploit the redundancy of carry-save arithmetic and apply the technique proposed by Robertson [49].

We can, however, use *borrow-save* [51] arithmetic. Borrow-save arithmetic is similar to carry-save arithmetic with two bits: *minus*, *borrow*. The borrow-save cells are full subtractors and the borrows are passed to the next most significant full subtractor on the next level. To convert from borrow-save arithmetic into binary, a borrow-propagate, i.e. a binary subtractor (or adder) is required.

As with carry-save there are four possible pairs of bits: $(m, b) \in \{(0,0), (0,1), (1,0), (1,1)\}$, which correspond to the three possible digit values $\{-1, 0, 1\}$. So borrow-save arithmetic is really SBNR. The only difference is the way that the digits are represented with binary values.

So if borrow-save is the same as SBNR, then the division methods described in Section 3.6.2 are equally applicable to borrow-save arithmetic. This has been shown by the implementation in [51].

3.9 On-line arithmetic

This Chapter has been devoted to redundant number systems in parallel arithmetic circuits. However, mention should be made of the relevance of redundant number systems in serial circuits. Specifically, with regards a technique known as *on-line arithmetic*.

On-line arithmetic is an application of redundant number systems that reduces the latency of digit serial circuits. An example of this can been seen in a paper by Ercogovac and Lang which describes the conversion of redundant numbers into conventional binary in a serial manner [42]. The technique described in [42] is applicable to any circuit which produces as redundant result in a digit serial manner, with the most significant digit first. The delay of the circuit is proportional to the size of the word, but since the input is serial with the MSD first, the result is ready one adder delay after the input was ready.

This pipelining technique can be used to produce circuits that, although in some cases may have a significant delay, have a very high throughput.

On-line arithmetic has been extended to a MSD first skew-parallel data format which allows the technique to be applied to digit parallel circuits, with similarly large gains in throughput [36], [52], [53]. This technique shows that redundant arithmetic can be

a powerful method of increasing the speed of arithmetic circuits.

3.10 Conclusions

This chapter has primarily been a review chapter for redundant number systems. Redundant number systems, particularly SBNR have been defined. It has been shown that redundant numbers (especially SBNR) can be used to perform addition and subtraction operations very fast. In fact, the speed of these operations is 2 cell delays for systems other than SBNR, and 3 cell delays for SBNR. Also, the speed is independent of the word-length of the operands. The main problem with redundant numbers, which means that SBNR cannot be used to arbitrary speed-up parallel circuits, is that the conversion between SBNR and conventional twos-complement is expensive (in fact, it is equivalent to summing two twos-complement operands).

This chapter has reviewed SBNR in some detail. All of the major arithmetic operations have been examined, and compared in speed and complexity to the equivalent conventional binary operations. Also, the comparison operation has been reviewed. While this function is not as common, it plays an important part in any implementation of the Viterbi algorithm.

One of the most common redundant number systems, carry-save arithmetic, has also been reviewed, and the similarities between it and SBNR have been discussed.

One of the most important uses of redundant number systems in digital circuits, on-line arithmetic, has been briefly discussed. This technique has been used in the literature to produce novel digital designs with very high throughput rates.

This chapter has also presented the design of a fast twos-complement adder circuit. The design was based on a Srinivas and Parhi fast adder design that used SBNR as an internal representation. Their use of redundant number systems has been shown to be pointless. In fact, the use of SBNR in their design only served to disguise the true nature of the speed-up, which has been revealed here. From that investigation, we have shown that redundant number systems should not be used simply as a arbitrary method for achieving speed-up, as demonstrated in the Srinivas and Parhi design. Unfortunately, this design was used as a stepping stone for the project, and the shortcomings of it were not discovered until later. However, this is an important piece of original

work, because it led to the design of a new fast adder circuit, which is purely twos complement. It has been shown that this design has a lower gate count, and smaller critical path, than existing designs.

Chapter 4

High Speed Viterbi Decoding using Redundant Number Systems

4.1 Introduction

4.1.1 Motivation

As discussed in the opening section of the last chapter, the Srinivas and Parhi paper had suggested that SBNR would be applicable to an implementation of the Viterbi algorithm. The Viterbi algorithm is an iterative algorithm, with an add-compare-select circuit forming the main part of the feedback loop. If this loop could be minimised, perhaps with a very fast add-compare-select circuit, a high speed Viterbi implementation could be produced. It should be noted that while a high-speed Viterbi implementation is not essential to a hand-held telephone system, such a device could be useful in receiver stations where signals could be multiplexed on a single device, reducing the size of the implementation. This chapter describes the VLSI design of the first devices.

4.1.2 Overview

In Chapter 2 Viterbi decoding was reviewed and in Chapter 3 redundant number systems were examined. This chapter describes the application of redundant number systems to a Viterbi decoder. As has been shown, SBNR has the potential for producing high speed arithmetic circuits, particularly when the speed of the addition circuitry is a primary concern. This chapter will present the design of a Viterbi decoder which uses redundant arithmetic as a representation for the path metric values. These metrics are totally internal and never form part of the output from the decoder. This means that the design will not suffer from the drawback of having to convert redundant numbers

into twos-complement arithmetic as covered in Chapter 3. Two final Viterbi decoder designs were produced, with constraint lengths $K = 4$, and $K = 5$, and rate $r = \frac{1}{2}$. In Section 4.2 the decoder design is presented and each of the main modules are described. In Section 4.3 issues regarding Viterbi decoder design for high-bit rates are reviewed. The remainder of the Chapter compares redundant arithmetic with conventional binary techniques for producing a high-speed Viterbi decoder design.

4.2 Viterbi decoder design

To achieve the highest possible throughput, the decoder design is based on a fully parallel architecture with a single add-compare-select (ACS) unit for each node. The block diagram in Figure 4.1 shows the basic design of the decoder. The incoming bit-stream is used to calculate the branch metric values (BMVs) which are then used to increment the path metric values (PMVs).

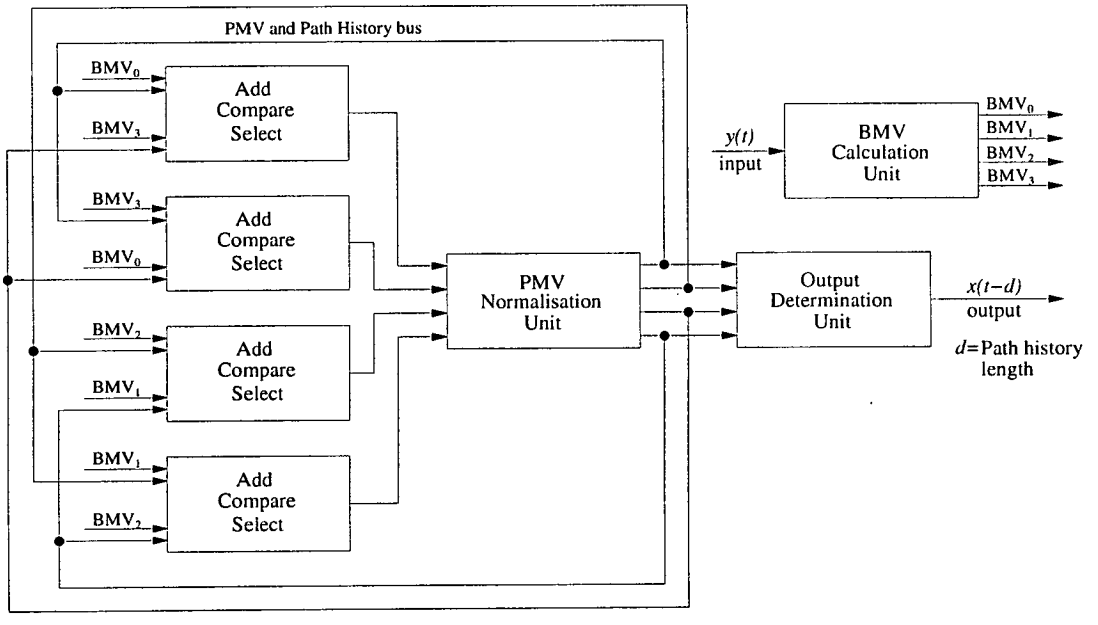


Figure 4.1: A block diagram of a Viterbi decoder

For constraint lengths $K = 4$ and $K = 5$, 7 bits is the minimum resolution for the PMV [22], [54]. As stated in Chapter 3, because of properties peculiar to redundant arithmetic, one extra digit must be stored to achieve the same dynamic range as conventional binary notation [4]. Therefore 8 SBNR digits are required to achieve the same resolution as 7 binary bits, and so we use a PMV resolution of 8 digits.

4.2.1 The branch metric value calculation unit

The BMV calculation unit computes the Euclidian distance between the received symbol pairs, and the possible received value associated with the trellis branches (see Section 2.3.1 for details). 8-level quantization was used for the input symbols in preference to a 2-level (binary) system. This level of quantization is known to provided $2dB$ higher coding gain over 2-level binary inputs [11]. It was decided that 8-level quantization was sufficient since infinite resolution provides an additional increase of only $0.25dB$ [11].

Table 4.1 shows how the Euclidian distance between the received input symbols and the possible input symbols is calculated. The two columns in the table show the input symbol with 8-level quantization. This system is the same as that used in [22] and [11]. The circuit which calculates this is shown in Figure 4.2.

Quantized value	Binary value	Distance from 0	Distance from 1
0	000	0	6
1	001	0	5
2	010	0	3
3	011	0	1
4	100	1	0
5	101	3	0
6	110	5	0
7	111	6	0

Table 4.1: 8-level quantization values for different received symbols

In a rate $r = \frac{1}{2}$ code the input symbols are pairs of bits, so the BMVs are calculated by summing two distances. The BMV calculation circuit for a rate $r = \frac{1}{2}$ decoder is shown in Figure 4.3.

The BMV calculation unit uses conventional binary notation rather that SBNR. As Table 4.1 shows, the maximum Euclidian distance for one bit is 6. So for pairs of bits, the maximum distance is 12, which can be represented with 4 binary bits. Since conventional binary is a subset of SBNR, and since a SBNR adder requires a transfer digit propagation of 3 digits (Section 3.4.1), there is little to be gained, except additional complexity, by using redundant arithmetic for the BMV calculation unit.

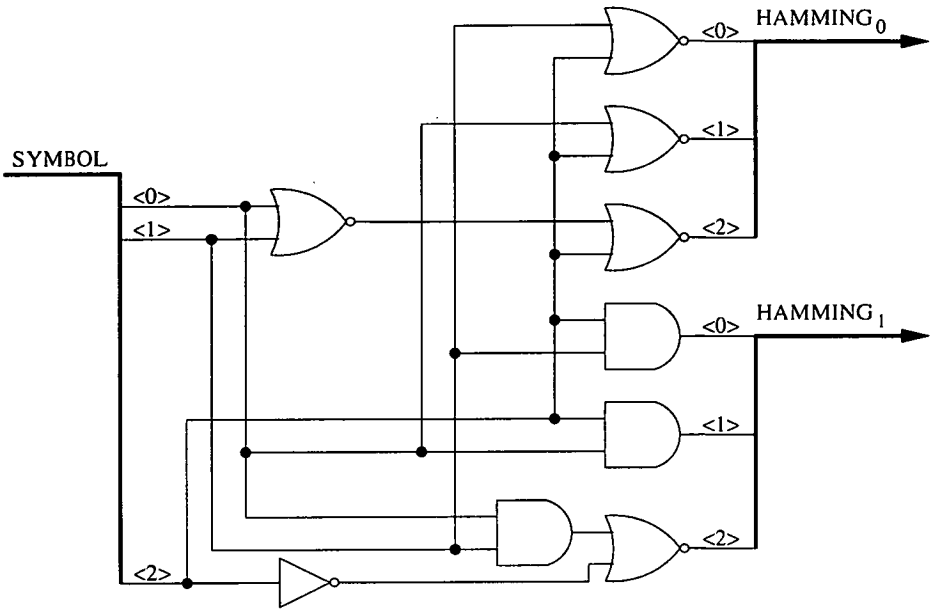


Figure 4.2: Schematic diagram of the bit distance generator

4.2.2 The add-compare-select unit

There is one ACS unit for each of the nodes in the trellis diagram. The ACS unit for state j performs the summation $PMV_i + BMV_{i \to j}$, where $BMV_{i \to j}$ is the BMV associated with a transition from state i to state j, for all permitted transitions $i \to j$.

The ACS unit selects the lowest of these summations as the PMV for state j. For a rate $r = \frac{1}{2}$ code there are two such transitions for each state.

4.2.3 The output determination unit

The path history length depends on the technique which is used for output determination. The standard technique for determining the output bit in the Viterbi algorithm is *minimum metric selection.* It involves selecting the least PMV from all the nodes and output the oldest bit in the corresponding path history.

Michelson and Levesque have simulated a Viterbi decoder using minimum metric se-

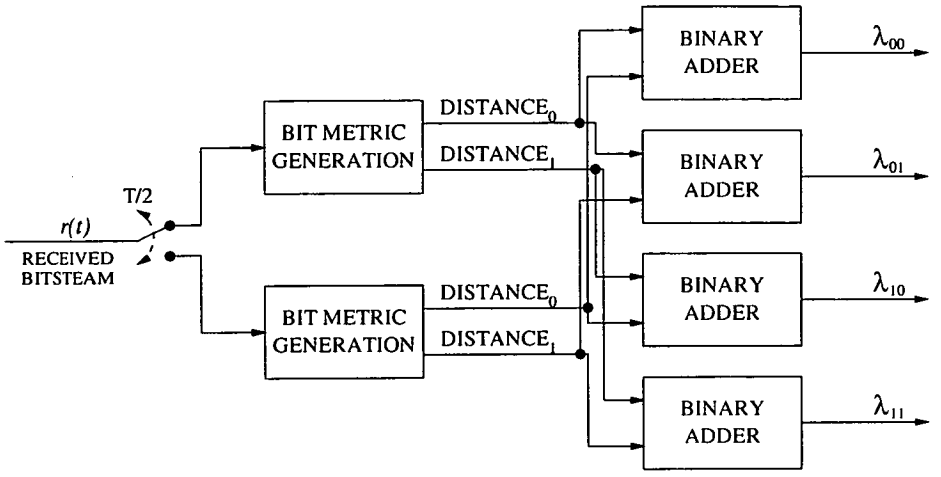


Figure 4.3: Schematic diagram of the BMV calculation unit

lection, with varying sizes of path history [15]. Their results showed that increasing the number of bits stored in the path history to more than 4 times the constraint length resulted in no discernible improvement in performance.

A binary tree based minimum metric selection circuit was initially investigated. This is shown in Figure 4.4.

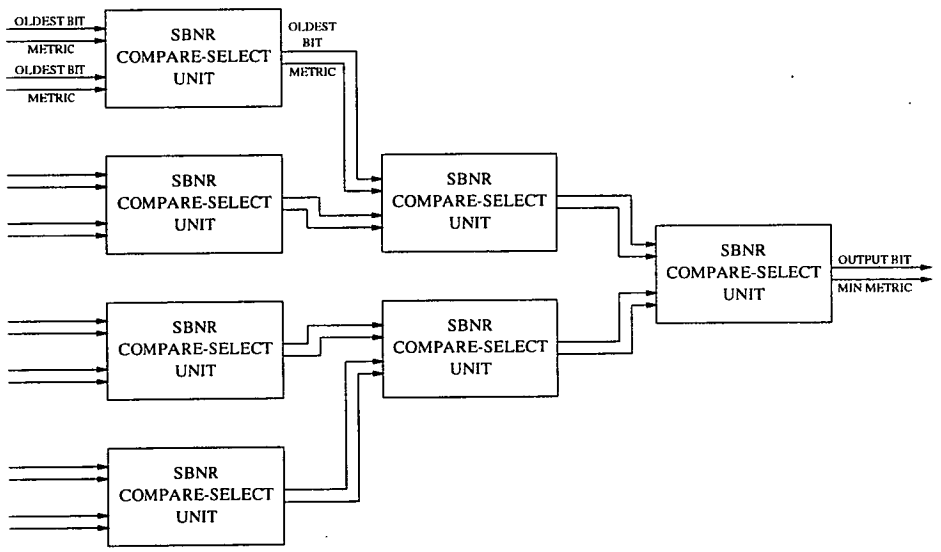


Figure 4.4: Schematic diagram of the minimum metric selection circuit

The circuit in Figure 4.4, which is suitable for an eight state Viterbi decoder with a constraint length of $K = 4$, was implemented using the 1.2μm standard cell library from ES2. The resulting circuit was relatively slow at $10.50ns$, compared with the ACS

units which were $4.56ns$ (as will be explained in Section 4.4.4). While this circuit could have been pipelined to increase the throughput rate, the area requirement was also large at 1073×1322 microns.

It was decided instead to use majority voting [55], [56] to determine the output. This involves examining the oldest bits of all the path histories and outputting a one or a zero depending upon which is the most common.

The oldest bits in the path histories are often the same [57]. This means that the oldest bit in the path history from each ACS unit can be fed into a majority gate to determine the output. The majority voting circuit for $K = 4$ only takes $1.48ns$ and is only requires 113×162 microns of silicon area. The disadvantage of using majority voting is that to achieve the same performance as for minimum path metric selection, the path history length must be increased. Michelson and Levesque's experiments showed that the path history length should be increased to 5 to 6 times the constraint length to provide the same coding gain as minimum metric selection [15]. This means that the path history is 24 bits for $K = 4$ and 30 bits for $K = 5$.

The path history storage has been implemented as one 24 or 30 bit register for each ACS unit. The output of each register is fedback to the input of 2 other ACS units. While other smaller implementations are possible, such a RAM blocks, the intention of the design was to compare SBNR with conventional binary so a basic path history implementation was used.

4.2.4 Metric normalization with redundant arithmetic

As discussed in Chapter 2, a PMV for a node is the sum of all the BMVs along the path to that node. This corresponds to the accumulated error between the received signal, and the individual path. Because of the trellis structure, paths will merge and for each convolutional code there is a maximum difference, Δ, between the smallest PMV and the largest PMV [22].

The number of errors in the transmitted signal can only increase, not decrease. For the amount of errors to decrease, previously incorrectly received bits would have to be received correctly - which is clearly impossible. With this knowledge it can be seen that the PMVs will increase continually if they are not regulated. This is undesirable since they must be stored using a finite number of bits. However, as has been pointed

out, the largest PMV can only be greater than the smallest PMV by Δ so when the largest value exceeds Δ, Δ can be subtracted from all the PMVs. This will not affect the computation of the algorithm because the absolute values of the PMVs are not important, only their relative differences are considered.

A PMV normalisation circuit is used to control and adjust the PMVs. The standard technique for normalisation when using conventional binary notation is to detect when all the MSBs (most significant bits) of the PMVs are set and reset them. If the number of bits used to represent the PMV is chosen carefully (see Table 4.2, based on results from [22]) then this is equivalent to subtracting the next power of 2 above Δ from all the path metric values [22].

PMV length (binary bits)	PMV length (SBNR Digits)	upper constraint length (K)	
		$r = \frac{1}{2}$	$r = \frac{1}{3}$
5	6	1	-
6	7	2	1
7	8	5	3
8	9	10	7
9	10	21	14

Table 4.2: Maximum constraint length for different word resolutions for our chosen quantisation

When using SBNR rather than conventional binary, instead of detecting that the MSB is set we must detect one of the following two conditions:

1. Is the 2nd MSB set to $+1$?
2. Is the MSB set to $+1$ and is the 2nd MSB set to -1?

This means that the normalisation circuitry is slightly more complex for redundant arithmetic.

The normalisation is achieved by the use of a normalisation control circuit. Each of the ACS nodes asserts a *normalisation request* line ($NORM_{req}$) when the PMV for that node satisfies one of the above conditions. The normalisation circuit generates a *normalisation grant* signal ($NORM_{grant}$) if all of the $NORM_{req}$ lines are asserted. When an ACS node received a $NORM_{grant}$ signal, the top two digits of its PMV are reset. This circuit is shown in Figure 4.5.

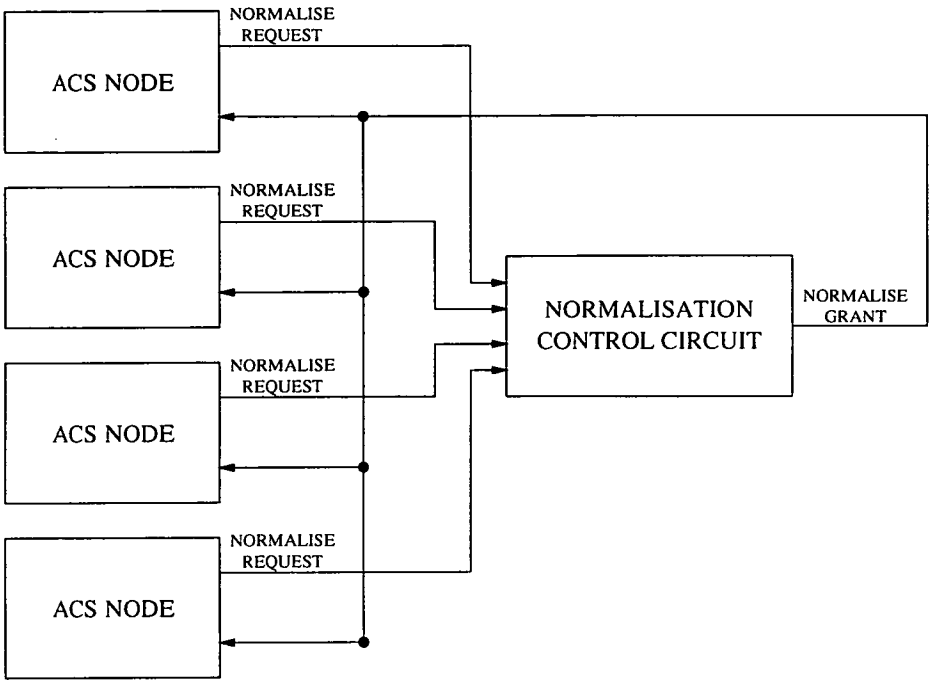


Figure 4.5: Schematic diagram of the normalisation circuitry

4.3 High bit-rate Viterbi Decoder design

In modern day communications systems, it is important to achieve higher, and higher bit rates. Hence it is important to be able to implement a Viterbi decoder that can operate at fast speeds. The fact that the VA is an iterative algorithm means that the throughput of the decoder is limited by the time it take to compute one iteration. This problem will be described in terms of the *add-compare-select bottleneck.*

This section will describe design techniques for producing high bit-rate Viterbi decoders. Both architectural and algorithmic methods will be reviewed.

4.3.1 The add-compare-select bottleneck

The feedback loop in the Viterbi decoder is shown in Figure 4.6. The addition operation of one iteration cannot be performed until the comparison operation of all the nodes of the previous iteration has been completed.

This results in a bottleneck which means that the conventional Viterbi algorithm cannot be pipelined. Hence the data rate of an implementation of the Viterbi algorithm is

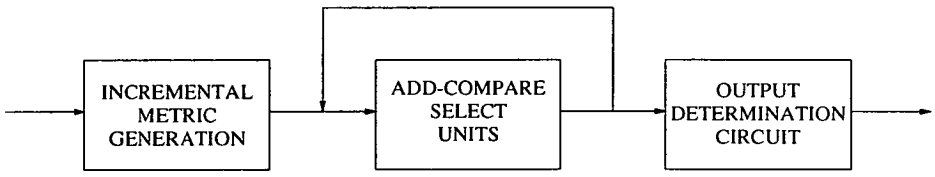


Figure 4.6: The add-compare-select bottleneck

dependant on the delay of the ACS units. To produce a high speed Viterbi decoder, this delay needs to be minimised by the use of architectural improvements, or the feedback loop has to be removed altogether by the use of algorithmic manipulation

4.3.2 Architectural improvements

The first method of producing a high speed implementation of the Viterbi algorithm is to introduce architectural improvements, normally to the ACS unit to minimise the length of time needed for each iteration.

In [58] the design of a 45 Mbps Viterbi decoder in presented for use in digital video applications. This design employs a traceback memory management unit [59], for output determination, which accesses the path histories of the ACS units. To speed up the decoder, multiport memory is used to store the path histories which means that the output determination and the ACS operation can be performed in the same cycle. This reduces the overall cycle time and allows the decoder to operate at high bit rates.

In [60] a 140 Mbps Viterbi decoder for satellite communications is presented. The design of a high speed ACS unit using custom emitter-coupled logic (ECL) is described. The use of ECL has the advantage of much higher speeds than CMOS. However, it has two main drawbacks: the power consumption is very large compared with CMOS; and the size of the design is much greater. In the design presented in [60] each ACS unit is implemented on one ECL gate-array which means that for a 16 state trellis code more than 16 chips would be required to perform the Viterbi algorithm.

The design presented in [61] uses carry-save arithmetic to create an ACS unit that can be pipelined, thus removing the feedback bottleneck. The use of carry-save addition means that the addition section of the ACS unit has no carry chain, and a bit-local carry-save maximum selector [62] is employed which means that the data rate is independent

of word length.

4.3.3 Algorithmic manipulation

An alternative method of producing high speed Viterbi decoders is to remove this bottleneck entirely by algorithmic manipulation.

In [57], [63] and [64] the ACS iterations are combined algebraically to produce an "M-step ACS recursion" which means that word-level parallelisation can be introduced to produce an arbitrary speedup, although the area of the design increases significantly. The design in [64] can operate up to 600 Mbps but requires an area of 170mm^2. In [65] this technique is extended and finer grain pipelining is introduced which can result in a Viterbi decoder up to eight times faster than conventional detectors.

4.4 Redundant number systems for high speed add-compare-select units

The technique that is explored in the remainder of this chapter is using redundant number systems, specifically SBNR, as an architectural method of producing a high speed Viterbi decoder.

Chapter 3 showed that addition circuits using SBNR can be considerably faster than twos-complement addition circuits, especially for large word lengths, because they eliminate the need for a long carry propagation chain. This section describes the results obtained from simulations to determine whether significant speed-ups could be achieved with SBNR ACS units over more conventional ones.

4.4.1 Simulation conditions

The circuits which follow were implemented using Boolean functions in the standard CMOS 0.7 micron library from ES2. Full adder modules were available in this library, but since the SBNR modules were designed from Boolean functions, it was decided that the twos-complement adders should also be designed from Boolean functions to

avoid biasing the results. The areas and delays are based on fully placed and routed units, both 8 bit and 16 bit ACS units have been simulated.

4.4.2 Adder circuits

Table 4.3 shows the results obtained for different adder units. The SBNR adder takes two SBNR numbers and outputs the sum in SBNR. From the table it can been seen that the redundant adder is faster than all other addition units. The fact that it does not require a carry propagation chain means that the 16 digit adder has the same delay as a 8 digit adder. It should also be noted that the redundant adder required only 30% more area than the carry-lookahead adder and is more than twice as fast.

	8 digit		16 Digit	
Adder Type	Delay (ns)	Area ($\times 10^3 \mu m^2$)	Delay (ns)	Area ($\times 10^3 \mu m^2$)
Ripple Adder	4.77	50.58	9.33	92.66
Carry Lookahead	2.27	111.54	2.78	221.67
Carry Select	4.11	78.32	4.38	147.84
SBNR Adder	1.06	143.02	1.06	283.86

Table 4.3: Comparison of 8 and 16 Digit Addition Circuits

4.4.3 Comparator circuits

Table 4.4 shows the area and delays for three types of comparator circuits. All of the circuits operate on two input words and determine which of the two is the larger. The first two comparators operate on twos-complement numbers, and the last circuit operates on two SBNR input words.

	8 digit		16 Digit	
Comparator Type	Delay (ns)	Area ($\times 10^3 \mu m^2$)	Delay (ns)	Area ($\times 10^3 \mu m^2$)
Ripple	4.66	44.47	8.98	82.58
Binary Tree	2.19	46.87	2.86	80.77
SBNR	2.83	164.86	3.37	333.46

Table 4.4: Comparison of 8 and 16 Digit Comparison Circuits

4.4.3.1 Binary comparator

The binary comparator used is based on the 4-bit binary tree twos-complement comparator shown in Figure 3.13, on Page 57.

The delay through the binary tree comparator is $O(log_2 N)$ where N is the word length.

4.4.3.2 Ripple comparator

The second type of twos-complement comparator is a ripple comparator. A block diagram for this is shown in Figure 4.7. The comparator is built up from type ① cells which take the maximum and equality result from the previous bits and compute the maximum and equality result for the current bits. These cells implement the following:

- If the a and b inputs being compared are equal then the new outputs are the same as the previous outputs.
- If the a and b inputs being compared are different then the $equal = 0$ and $max = b$.

The delay through the ripple adder is $O(N)$ from the least significant bits of a and b being available. However, if the values a and b are generated from a ripple adder then the delay though a cascaded ripple adder and ripple comparator would be equivalent to one carry chain. i.e. $O(N + 1)$.

4.4.3.3 SBNR comparator

The final comparator circuit shown in Table 4.5 is a SBNR comparator. While SBNR addition is fast and does not require a carry chain - the same is not true for SBNR comparison. Our comparator consists of a SBNR subtraction circuit followed by an SBNR sign select circuit based on a design from [5] which is shown in Figure 4.8.

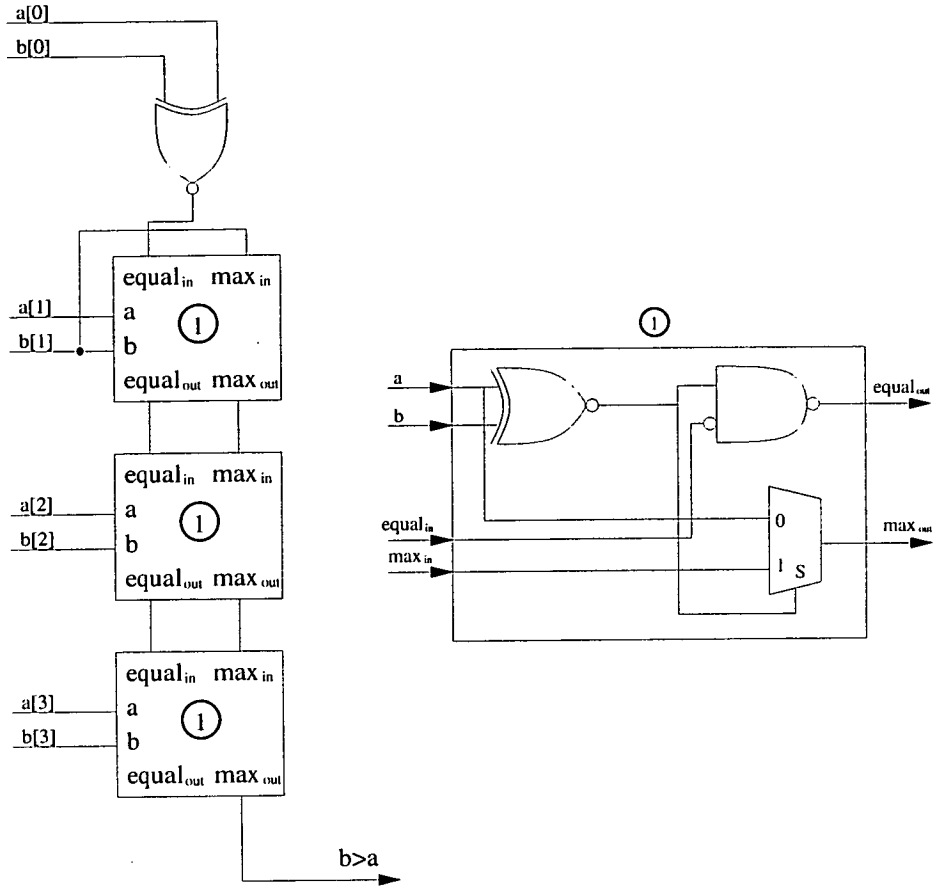


Figure 4.7: A 4-bit twos-complement ripple comparison circuit

4.4.4 Add-compare-select circuits

Table 4.5 shows results from five different ACS circuits comprised of the addition and comparison modules described in Sections 4.4.2 and 4.4.3.

The table confirms the observation made in Section 4.4.3.2 that a ripple adder followed by a ripple comparator would perform faster than the same adder followed by a binary tree comparator. In fact the ripple adder / ripple comparator circuit is approximately 10% faster than the ripple adder / binary comparator circuit.

Table 4.5 also shows that the CLA is roughly 75% larger than the ripple adders for 8 bits and 100% larger for 16 bits but it is significantly faster in both cases.

Finally it can be seen that the speed of SBNR is similar to that of CLA techniques for 8 digit adders but a speed-up of 14% can be achieved for 16 digit adders. However, the SBNR ACS units require almost twice the area of the corresponding CLA 16 bit ACS

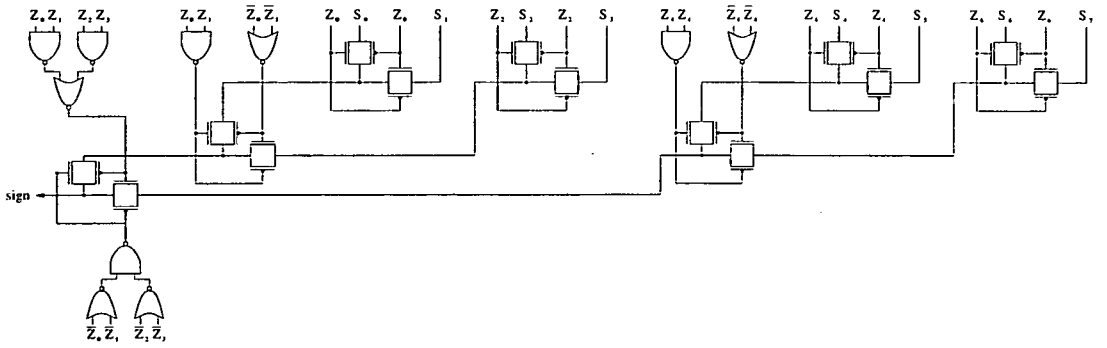


Figure 4.8: The 8-digit SBNR sign select circuit

ACS Unit Type	8 digit		16 Digit	
	Delay (ns)	Area ($\times 10^3 \mu m^2$)	Delay (ns)	Area ($\times 10^3 \mu m^2$)
Ripple add / ripple compare	6.86	193.36	12.81	336.12
Ripple add / binary compare	7.58	209.32	11.42	358.11
CLA add / binary compare	4.62	348.19	5.92	668.25
Carry select add / binary compare	6.40	243.46	7.60	488.56
SBNR add / SBNR compare	4.56	507.26	5.10	1056.65

Table 4.5: Comparison of 8 and 16 Digit add-compare-select units

units.

4.4.5 Discussion

We have shown that an increase in speed of 14% for an ACS unit can be achieved by using SBNR instead of twos-complement number representation. However, the increase in area in considerable, roughly 45%. If SBNR were to be used as an internal representation for the metrics in a Viterbi decoder, we could expect a 14% increase in the bit rate, but the increase in area is probably not practical in a number of applications, certainly SBNR is not applicable to small or low-power design.

4.5 Implementation and layout

Two Viterbi decoders have been designed and simulated, these are $K = 4, r = \frac{1}{2}$ and $K = 5, r = \frac{1}{2}$. They were both produced using the $1.0\mu m$ two-metal standard cell

CMOS process from ES2.

The $K = 4$ design was produced using Cadence Design Framework II schematic capture and logic synthesis tools to optimise the area and speed of the lowest level cells.

For the $K = 5$ decoder, a C program was written. This program generates a Verilog HDL structural description of the decoder. The code accepts the constraint length K as input and generates a Viterbi decoder, rate $r = \frac{1}{2}$, with a trellis based on the optimal convolutional code for the constraint length K, as determined by Larson [14].

The Verilog code which is produced does not contain all of the modules, the rest (such as the SBNR addition modules etc) have been designed in schematics. The Verilog code was then imported into Cadence Design Framework II which produces schematic diagrams and netlists.

The layout was then produced using the Cadence/ES2 automatic place and route tools. The layout of the $K = 4, r = \frac{1}{2}$ decoder occupied 13.5mm^2 of silicon and is shown in Figure 4.9, the $K = 5, r = \frac{1}{2}$ decoder required 26mm^2.

It has been estimated that a similar decoder with $K = 7$ would occupy 100mm^2.

4.6 Simulation and testing

The capacitance of the routed chips were extracted and the decoders were simulated. Both the layouts of the $K = 4$ and $K = 5$ decoders have achieved similar bit rates of up to 100 Mbps.

As the constraint length K is increased, the speed of the decoder does not decrease considerably. So we estimate that a decoder of $K = 7$ based on this design would be able to operate at a similar speed to the $K = 4$ and $K = 5$ decoders.

4.7 Comparisons

This section will review some existing Viterbi decoders designs, and compare them with the design presented in this chapter.

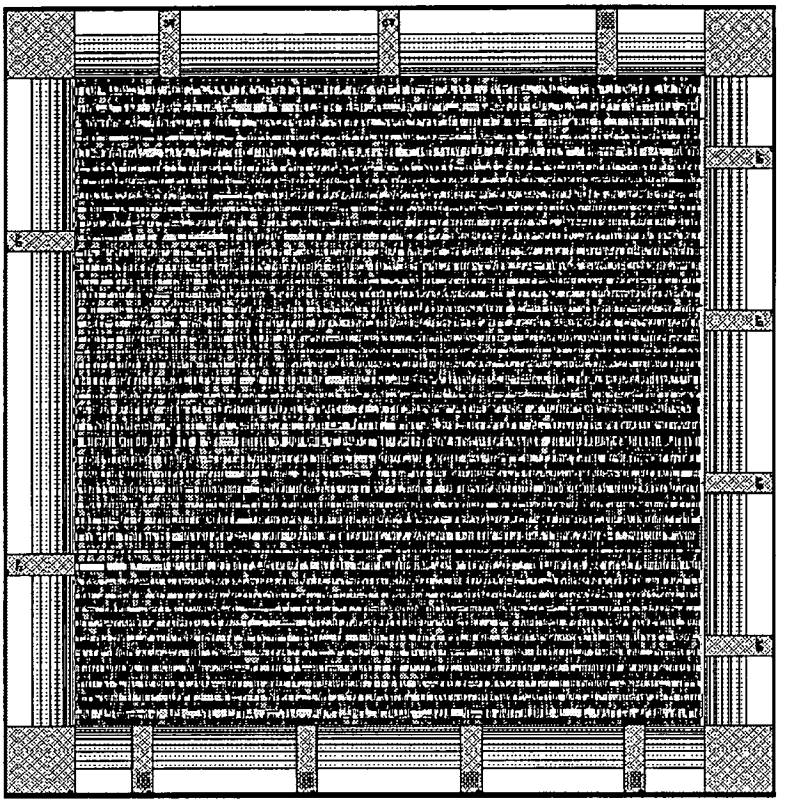


Figure 4.9: The layout of the K=4 decoder

Table 4.6 shows some recent Viterbi decoder designs. Design 1, taken from [58] is a $K = 6$ decoder, and is capable of 45Mbps. The designs presented here, although the biggest of which is only $K = 5$, is extendable to $K = 6$ with an estimated throughput of 100Mbps.

The second design from [60] is designed for satellite communications and is capable of a high throughput of 140Mbps. However, this design is only $K = 4$ and consists of over 16 discrete chips. Also, since the design using emitter-coupled logic (ECL) the power consumption is very large. The 100Mbps $K = 4$ design presented in this

	Reference	No. of states	Area	Speed (Mbps)	Process
1	[58]	64	-	45	0.8μm CMOS
2	[60]	16	>16 chips	140	ECL gate arrays
3	[61]	4	74 mm^2	115	2μm CMOS
4	[64]	4	170 mm^2	600	1.2μm CMOS
5	[38], [66]	32	62 mm^2	140	1.2μm CMOS

Table 4.6: Comparison of recent high speed Viterbi decoders

chapter is CMOS (hence significantly lower in power usage) and is implemented on one chip with a silicon area of 13.5mm^2.

Designs 3 and 4, from [61] and [64] have been developed by Fettweis and Meyr. The implementations use carry-save arithmetic to create an ACS unit that can be pipelined, thus removing the feedback bottleneck. In addition, in [64] the ACS iterations are combined algebraically to produce an arbitrary speedup (as discussed in Section 4.3.3), however, the implementation is very large, at 170mm^2, and only has a 4 state trellis ($K = 3$).

Design 5 has been produced by Black and Meng at Stanford University [38], [66]. It is the most comparable design to the one presented in this section. The design is faster than the one presented here, the throughput is 140Mbps (40% faster than our design). However, the design is considerably larger at 62mm^2, compared to 26mm^2.

Details, such as the chip area, and process type, of commercial implementations are difficult to come by. This is why none are included in Table 4.6. However, in [38], Black and Meng described single chip Viterbi solutions from Stanford Telecommunications and Qualcomm Incorporated. These implementations are based on convential radix-2 techniques and are capable of bit rates in the region of 25Mbps. Signigicantly slower that the implementations presented here.

4.8 Conclusions

This chapter has described the architectural design of a Viterbi decoder which lead to the implementation of two Viterbi decoder devices, one with constraint length $K = 4$, and one with constraint length $K = 5$. The decoder design used redundant number systems as an internal representation for the incremental metrics. One of the attractions of SBNR for this was that the metrics are used internally only, they are never output. This meant that the problem of the slow conversion from SBNR into twos-complement binary format was not an issue.

Specifically, the following components of the Viterbi decoder have been described: the branch metric calculation unit, the add-compare-select unit, the output determination unit, and the metric normalization unit. The metric normalization unit was based on the conventional normalisation technique for binary numbers; it had to be extended

slightly to apply the same method to SBNR. The add-compare-select unit used the sign-select circuit based on the Srinivas and Pahri fast adder described in Chapter 3. This produced a circuit, which was faster than the equivalent twos-complement circuit.

It should be noted that this chapter has contained a large amount of original work. The design of a fast SBNR add-compare-select unit has been produced, it has been shown to give a limited (but significant) speed increase of 14% over conventional carry-lookahead techniques for 16bit values. Also, the implementation of a decoder design using SBNR throughout, specifically using the fast add-compare- select units, has been presented. This design as been shown to produce a very fast implementation that is small (even though the ACS units are large) and compares well against existing designs. In fact, the design shown in Figure 4.9 is better in terms of speed and area than most of the designs reviewed in Section 4.7. A design using the carry-save techniques described in Chapter 3 would be even smaller and faster.

Chapter 5
Digital Architectures for Viterbi Equalization

5.1 Introduction

5.1.1 Motivation

The second application of the Viterbi algorithm within a GSM receiver is for channel equalization. The implementation of the decoder module (presented in Chapter 4) had been quite successful. It seemed logical to extend this design to equalization. Again, a high-speed Viterbi equalizer is not essential to a hand-held telephone system, but such a device could be useful in receiver stations. In addition, a technique for producing low-power VLSI circuits called Complementary Pass-transistor Logic (CPL) [7] had been investigated during the early period of the project. The equalizer was designed with the intention of implementing it using CPL. This chapter describes the architectural design of the Viterbi equalizer.

5.1.2 Overview

As described in Chapter 2, the VA was proposed in the COST 207 report into Digital land mobile radio communications as a possible channel equalization technique for GSM [3]. In this chapter, the design and a low-power implementation of a Viterbi equalizer (VE) suitable for a GSM receiver is described [67]. The techniques presented here have been being implemented for an intersymbol interference (ISI) of 5 bits which (in GSM) represents a multipath delay spread of $15\mu s$ [27].

A number of VE designs for GSM have been presented in the literature. In [68] the design and simulation of a VE is proposed, although not implemented. In [69] a pa-

rameterised VLSI serial implementation of VE is presented and in [70] the simulation of a Verilog HDL implementation is described.

In this chapter the design of a fully parallel VE for GSM will be described, the architecture uses high-speed combinatorial logic to minimise the speed of the internal clock, and is designed to be implemented by complementary pass-transistor logic (CPL) which offers the possibility of low power consumption and low area at a high speed [71].

The parallel architecture will be described, although the design was not implemented due the large area requirements. Chapter 6 will examine why the implementation was halted and why a serial design was developed instead.

5.2 The Viterbi equalizer design

From Chapter 2, for each possible received sequence, $(\mathbf{a}, \mathbf{b}) = ((a_{n-L}, a_{n-L+1}, \ldots, a_{n-1}), (b_{n-L}, b_{n-L+1}, \ldots, b_{n-1}))$, the VE minimises the metric [2]:

$$\begin{aligned}\Lambda_n(\mathbf{a},\mathbf{b}) = \Lambda_{n-1}(\mathbf{a},\mathbf{b}) &+ a_n\left[y_n - \sum_{m=n-L}^{n-1}(a_m\chi_{n-m} + b_m\zeta_{n-m})\right] \\ &+ b_n\left[z_n - \sum_{m=n-L}^{n-1}(b_m\chi_{n-m} + a_m\zeta_{n-m})\right] \qquad (5.1)\end{aligned}$$

$$\Lambda_{mn} = \Lambda_{m(n-1)} + a_n^m\left[y_n - I_s\right] + b_n^m\left[z_n - Q_s\right] \qquad (5.2)$$

$$I_s = \sum_{m=n-L}^{n-1}(a_m\chi_{n-m} + b_m\zeta_{n-m}) \qquad (5.3)$$

$$Q_s = \sum_{m=n-L}^{n-1}(b_m\chi_{n-m} + a_m\zeta_{n-m}) \qquad (5.4)$$

where y_n and z_n are the in-phase and quadrature received signals samples after passing through a filter matched to an estimate of the channel impulse response (CIR), χ_n and ζ_n are the CIR matched filter autocorrelation coefficients, and m represents the unique possible received sequence $(\mathbf{a}, \mathbf{b})$ which results in a path to state S in the trellis diagram.

It can be seen that I_s and Q_s, defined in equations 5.3 and 5.4 depend only on the sequence of possible received signal sequences $(\mathbf{a}, \mathbf{b})$ and the CIR estimate. If the CIR

is changing rapidly then the incremental metric (equation 5.2) will be incorrect. In the GSM system, each $0.577ms$ TDMA burst contains a midamble which is used to estimate the CIR (Section 2.6.2). It is possible that the CIR could be varying so quickly that the CIR estimate is not sufficiently accurate for the whole of the $0.577ms$ TDMA burst. If this is the case then the VE needs to be adaptive [72], [73], [74], [27], [75].

Adaptive equalizers have been shown to be able to operate with multipath delay of $20\mu s$, corresponding to a mobile station speed greater than $200km/h$, [27] which would allow the use of cellular telephone handsets in high speed trains. In fact some adaptive equalizer design using the fast Kalman algorithm [3] can achieved a good signal-to-noise ratio at up to $300km/h$ [72].

In this work, the main objective was to produce a small, low-power design which would be suitable for use in a cheap portable handset. It was decided not to produce an adaptive equalizer which would require significantly more complexity than a non-adaptive one [3]. It has been shown that a non-adaptive VE design with 32 states is capable of equalizing signals with multipath delays of up to $15\mu s$ which corresponds to a vehicle delay of $200km/h$ [75]. It was decided to select this as the specification of the VE design for GSM applications.

The diagram in Figure 5.1 shows the overall structure of a non-adaptive VE for an N state trellis.

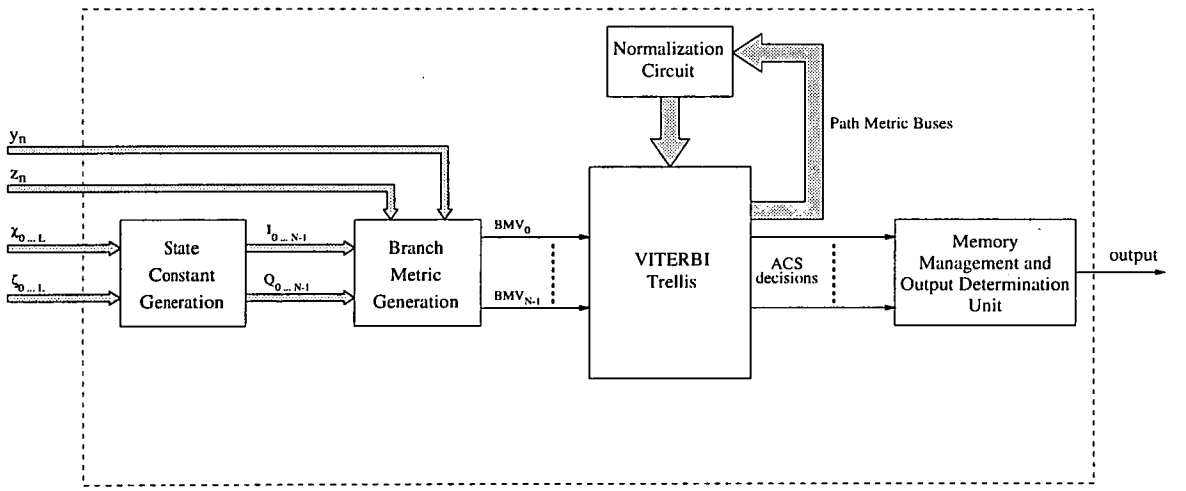


Figure 5.1: Top level block diagram

The resolution for the received sample and the coefficients was chosen to be 8-bit. From the literature, a binary wordlength of 8-bit is necessary to reduce performance

degradation [72].

5.2.1 Viterbi equalizer complexity

To design a VE circuit, the trellis diagram needs to be constructed. In Section 2.4.2, the MSK trellis diagram for an ISI of 2 in-phase/quadrature-phase pairs is derived. A VE which is required to cope with multipath echo delay of $15\mu s$ must be able to cope with an ISI of 5 in-phase/quadrature-phase pairs [27].

Table 5.1 shows the state transition table for the MSK trellis diagram with an ISI of 5. The table shows the $(\mathbf{a}, \mathbf{b})$ sequences which are associated with a path each of the states. It can be seen that the length of the $(\mathbf{a}, \mathbf{b})$ sequence corresponds to the number of symbols of ISI, or the *memory* of the modulation code and the memory of the channel.

A branch into a state represents a path between state on the modulation diagram (as shown in Figure 2.11(b) on page 20). Each of the transitions represents an input symbol to the modulator, these are shown in the "Append" column of Table 5.1 and these are the values that are appended to the path histories for the states.

The trellis consists of 64 states. As noted in Section 2.4.2, if we know the starting state then the number of states in each iteration is halved, in this case the trellis has 32 states at each iteration. Assuming the first state at the modulator (Figure 2.11(b)) is $\Delta\phi = 0$, then the final column shows the cycles (odd or even) that each state is valid.

5.2.2 The summation circuit

Examining equation 5.1 we can see that for each single TDMA packet we are required to evaluate 4 sets of summations:

$$\sum_{m=n-L}^{n-1} a_n \chi_{n-m} \tag{5.5}$$

$$\sum_{m=n-L}^{n-1} b_n \zeta_{n-m} \tag{5.6}$$

$$\sum_{m=n-L}^{n-1} b_n \chi_{n-m} \tag{5.7}$$

State Number	(a, b) sequence										Output to states	Append bit	Cycle
0	1	1	1	-1	-1	-1	-1	1	1	1	48 , 56	-1	even
1	1	1	-1	1	1	1	1	-1	-1	-1	16 , 24	-1	even
2	1	1	1	-1	1	1	1	-1	-1	-1	49 , 57	-1	even
3	1	1	-1	1	-1	-1	-1	1	1	1	17 , 25	-1	even
4	1	1	1	-1	-1	-1	1	-1	-1	-1	50 , 58	-1	even
5	1	1	-1	1	1	1	-1	1	1	1	18 , 26	-1	even
6	1	1	1	-1	1	1	-1	1	1	1	51 , 59	-1	even
7	1	1	-1	1	-1	-1	1	-1	-1	-1	19 , 27	-1	even
8	1	1	1	-1	-1	-1	-1	1	-1	-1	52 , 60	1	even
9	1	1	-1	1	1	1	1	-1	1	1	20 , 28	1	even
10	1	1	1	-1	1	1	1	-1	1	1	53 , 61	1	even
11	1	1	-1	1	-1	-1	-1	1	-1	-1	21 , 29	1	even
12	1	1	1	-1	-1	-1	1	-1	1	1	54 , 62	1	even
13	1	1	-1	1	1	1	-1	1	-1	-1	22 , 30	1	even
14	1	1	1	-1	1	1	-1	1	-1	-1	55 , 63	1	even
15	1	1	-1	1	-1	-1	1	-1	1	1	23 , 31	1	even
16	-1	1	1	1	1	-1	-1	-1	-1	1	0 , 8	-1	odd
17	-1	1	-1	-1	-1	1	1	1	1	-1	32 , 40	-1	odd
18	-1	1	1	1	-1	1	1	1	1	-1	1 , 9	-1	odd
19	-1	1	-1	-1	1	-1	-1	-1	-1	1	33 , 41	-1	odd
20	-1	1	1	1	1	-1	1	1	1	-1	2 , 10	-1	odd
21	-1	1	-1	-1	-1	1	-1	-1	-1	1	34 , 42	-1	odd
22	-1	1	1	1	-1	1	-1	-1	-1	1	3 , 11	-1	odd
23	-1	1	-1	-1	1	-1	1	1	1	-1	35 , 43	-1	odd
24	-1	1	1	1	1	-1	-1	-1	1	-1	4 , 12	1	odd
25	-1	1	-1	-1	-1	1	1	1	-1	1	36 , 44	1	odd
26	-1	1	1	1	-1	1	1	1	-1	1	5 , 13	1	odd
27	-1	1	-1	-1	1	-1	-1	-1	1	-1	37 , 45	1	odd
28	-1	1	1	1	1	-1	1	1	-1	1	6 , 14	1	odd
29	-1	1	-1	-1	-1	1	-1	-1	1	-1	38 , 46	1	odd
30	-1	1	1	1	-1	1	-1	-1	1	-1	7 , 15	1	odd
31	-1	1	-1	-1	1	-1	1	1	-1	1	39 , 47	1	odd
32	-1	-1	-1	1	1	1	1	-1	-1	-1	16 , 24	-1	even
33	-1	-1	1	-1	-1	-1	-1	1	1	1	48 , 56	-1	even
34	-1	-1	-1	1	-1	-1	-1	1	1	1	17 , 25	-1	even
35	-1	-1	1	-1	1	1	1	-1	-1	-1	49 , 57	-1	even
36	-1	-1	-1	1	1	1	-1	1	1	1	18 , 26	-1	even
37	-1	-1	1	-1	-1	-1	1	-1	-1	-1	50 , 58	-1	even
38	-1	-1	-1	1	-1	-1	1	-1	-1	-1	19 , 27	-1	even
39	-1	-1	1	-1	1	1	-1	1	1	1	51 , 59	-1	even
40	-1	-1	-1	1	1	1	1	-1	1	1	20 , 28	1	even
41	-1	-1	1	-1	-1	-1	-1	1	-1	-1	52 , 60	1	even
42	-1	-1	-1	1	-1	-1	-1	1	-1	-1	21 , 29	1	even
43	-1	-1	1	-1	1	1	1	-1	1	1	53 , 61	1	even
44	-1	-1	-1	1	1	1	-1	1	-1	-1	22 , 30	1	even
45	-1	-1	1	-1	-1	-1	1	-1	1	1	54 , 62	1	even
46	-1	-1	-1	1	-1	-1	1	-1	1	1	23 , 31	1	even
47	-1	-1	1	-1	1	1	-1	1	-1	-1	55 , 63	1	even
48	1	-1	-1	-1	-1	1	1	1	1	-1	32 , 40	-1	odd
49	1	-1	1	1	1	-1	-1	-1	-1	1	0 , 8	-1	odd
50	1	-1	-1	-1	1	-1	-1	-1	-1	1	33 , 41	-1	odd
51	1	-1	1	1	-1	1	1	1	1	-1	1 , 9	-1	odd
52	1	-1	-1	-1	-1	1	-1	-1	-1	1	34 , 42	-1	odd
53	1	-1	1	1	1	-1	1	1	1	-1	2 , 10	-1	odd
54	1	-1	-1	-1	1	-1	1	1	1	-1	35 , 43	-1	odd
55	1	-1	1	1	-1	1	-1	-1	-1	1	3 , 11	-1	odd
56	1	-1	-1	-1	-1	1	1	1	-1	1	36 , 44	1	odd
57	1	-1	1	1	1	-1	-1	-1	1	-1	4 , 12	1	odd
58	1	-1	-1	-1	1	-1	-1	-1	1	-1	37 , 45	1	odd
59	1	-1	1	1	-1	1	1	1	-1	1	5 , 13	1	odd
60	1	-1	-1	-1	-1	1	-1	-1	1	-1	38 , 46	1	odd
61	1	-1	1	1	1	-1	1	1	-1	1	6 , 14	1	odd
62	1	-1	-1	-1	1	-1	1	1	-1	1	39 , 47	1	odd
63	1	-1	1	1	-1	1	-1	-1	1	-1	7 , 15	1	odd

Table 5.1: 64 state MSK trellis transition table

$$\sum_{m=n-L}^{n-1} a_n \zeta_{n-m} \tag{5.8}$$

where $\{a_i\} = \pm 1$ and $\{b_i\} = \pm 1$ are state dependant, and the sequences $(\chi_{n-L}, \chi_{n-L+1}, \cdots, \chi_{n-1})$ and $(\zeta_{n-L}, \zeta_{n-L+1}, \cdots, \zeta_{n-1})$ are the same for all states and, for a non-adaptive VE, they are constant for each single TDMA burst.

The number of states increases with the ISI level, therefore the number of summations required increases exponentially. To reduce the computational complexity we compute the summations using a *differential addition circuit.* This is an architecture which exploits the inherent symmetries in the algorithm as described below.

Table 5.1 shows the following three symmetries:

- Each $(\mathbf{a}, \mathbf{b})$ sequence differs from at least one other sequence by only one digit.
- For each state, the $(\mathbf{a}, \mathbf{b})$ sequence is the exact inverse of the $(\mathbf{a}, \mathbf{b})$ sequence for a different state. For example, for state 0, $(\mathbf{a}, \mathbf{b}) = (1, 1, 1, -1, -1, -1, 1, 1, 1)$ and state 32, $(\mathbf{a}, \mathbf{b}) = (-1, -1, -1, 1, 1, 1, -1, -1, -1)$. In fact, the sequences for states 0 through to 31, are the exact inverses of the sequences for states 32 through 63. The means that we only need to consider evaluating the summations for states 0 through to 31 and the others can be computed simply by negation.
- Each of the $\mathbf{a}$ sequences from one state are the same as the $\mathbf{b}$ sequences on two other states. For example, for state 16, $\mathbf{b} = (1, 1, -1, -1, 1)$, which is the same as the $\mathbf{a}$ sequence for states 0 and 64. In fact, the $\mathbf{a}$ sequences for states 0 through 15 are the same as the $\mathbf{b}$ sequences for states 16 through 31.

This means that the summations in equation 5.7 are the same as the summations in equation 5.5, similarly the summations in equation 5.8 are the same as the summations in equation 5.6.

So, to evaluate equations 5.5 to 5.8 for $(\mathbf{a}, \mathbf{b})$ for the states 0 to 31, we only have to compute equation 5.5 and equation 5.8 for states 0 to 15. Hence, only two summation trees are required, each of which evaluates 16 summations.

Figure 5.2 shows the summation tree for summations $0 - 16$ in equation 5.5. An identical tree with coefficients χ_i replaced by ζ_i is used for calculating summations $0 - 15$ in equation 5.8.

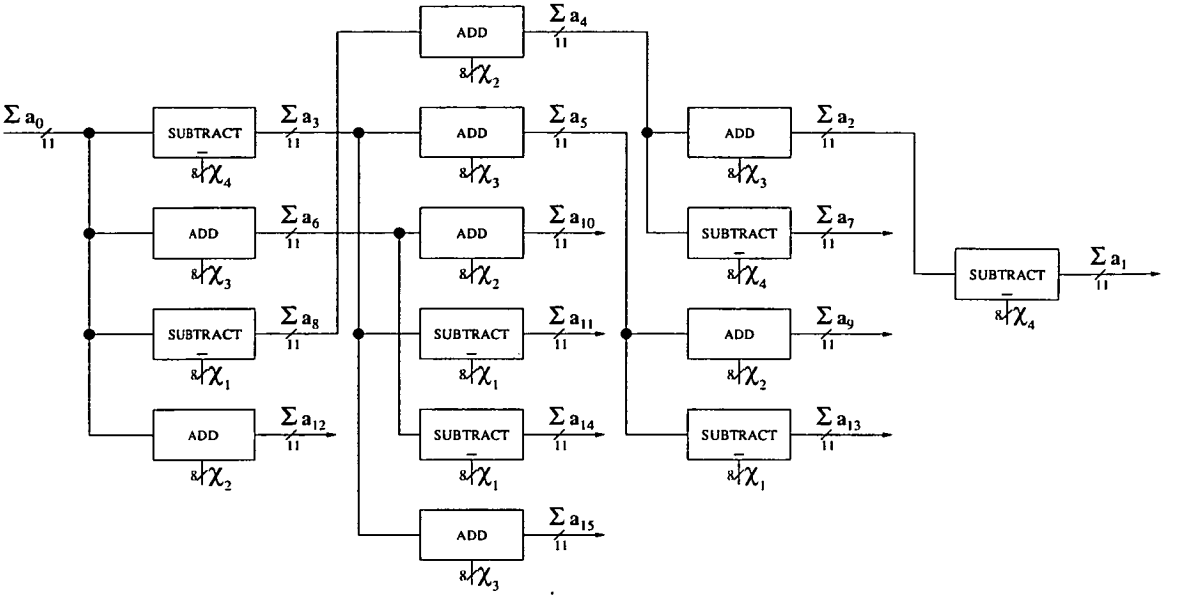


Figure 5.2: Top level block diagram

Only the two summations are evaluated initially, for state 0, and the summations for the remaining states are obtained by adding $\pm 2\chi_i$ or $\pm 2\zeta_i$. Figure 5.3 shows the circuit which performs the initial summation for equation 5.5.

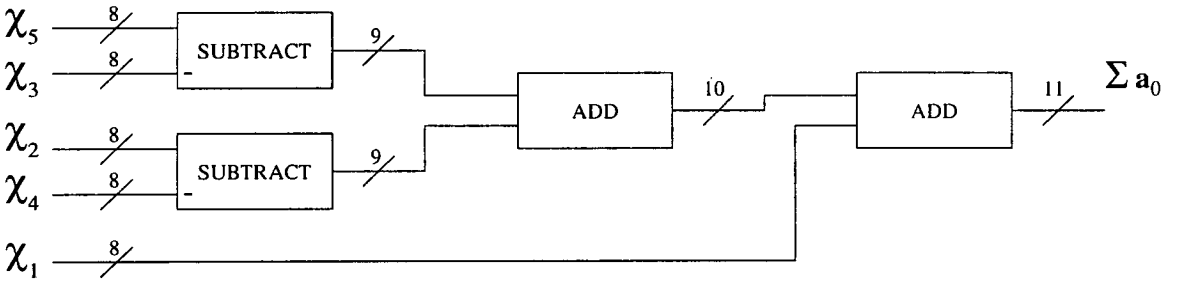


Figure 5.3: Top level block diagram

The complete circuit for generating the constants I_s and Q_s requires 8 addition units to form the two initial summations; 30 11-bit addition/subtraction units for the two summation trees; and 32 12-bit addition units and 32 12-bit negation units to produce the I_s and Q_s values. If the summations were to be evaluated individually in parallel, 1280 addition/subtraction units would be required.

In terms of speed, the two initial summations are formed in 3 adder delays. The summation trees require 4 adder delays. The evaluation of I_s and Q_s for states 0 to 31 by adding together two of these summations requires one adder delay and one negation

delay. Then the evaluation of I_s and Q_s for states 32 to 63 requires the delay of one negation unit.

It was decided to implement the summation circuit using ripple addition because the ripple adder often has the smallest area and lowest power of all the adder architectures [71]. The delay for a ripple adder is $O(n)$, however, the least significant bit (LSB) of the output of a ripple adder is available only 1 full adder delay after the LSB of the input words are available. This means that for cascaded ripple adders, the critical path is the carry chain of the last ripple adder in the sequence, plus the first full adders of all the prior ripple adders. This is shown in Figure 5.4, the critical path through the two cascaded ripple adders is highlighted.

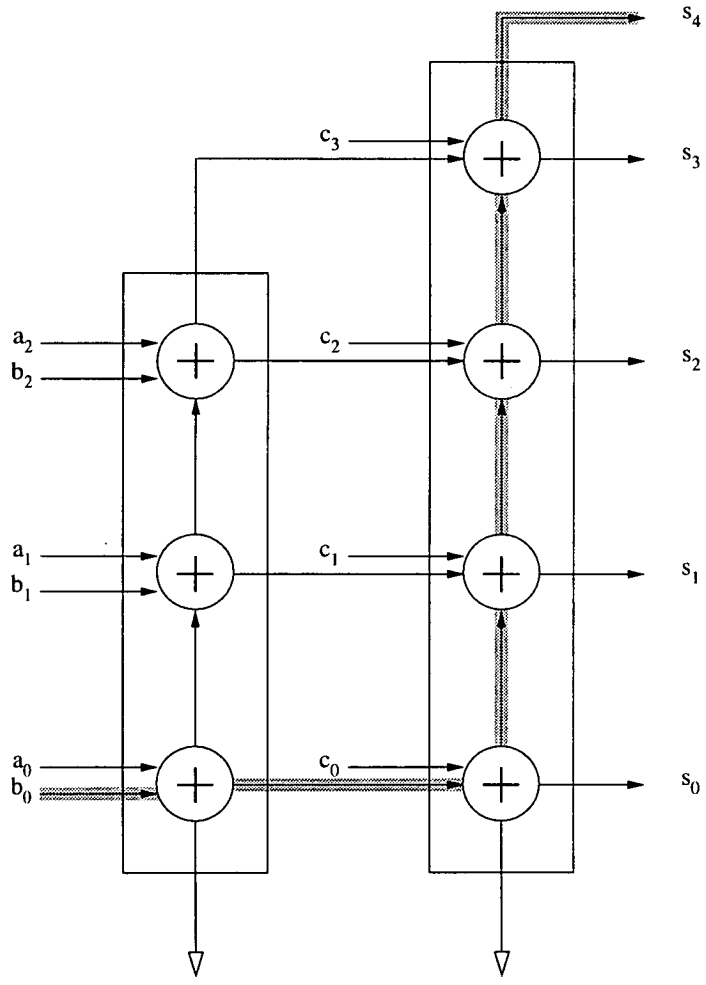


Figure 5.4: The critical path through two cascaded ripple adders

In the state constant generation circuit, assuming full adder cells and one bit negation cells have one gate delay, after the coefficients $\chi_0, \chi_1, ..., \chi_5$ are available, it will take

9 gate delays to evaluate the LSBs for all of the I_s and Q_s and another 12 gate delay for the carry to propagate. This means that the delay for the whole state constant generation circuit is 21 gate delays.

If the fast adder circuit proposed in Section 3.4.2 with a delay of $log_2(n) + 2$ was used instead of ripple adders, with n being 9, 10, and 10, the total delay would be 17 gate delays. However, the circuit would be significantly larger and in this design the speed saving is not necessary.

5.2.2.1 A serial summation unit

An alternative implementation would be to produce the summations serially. Figure 5.5 shows a design of a serial summation circuit which produces the summation for the a sequence for state 0.

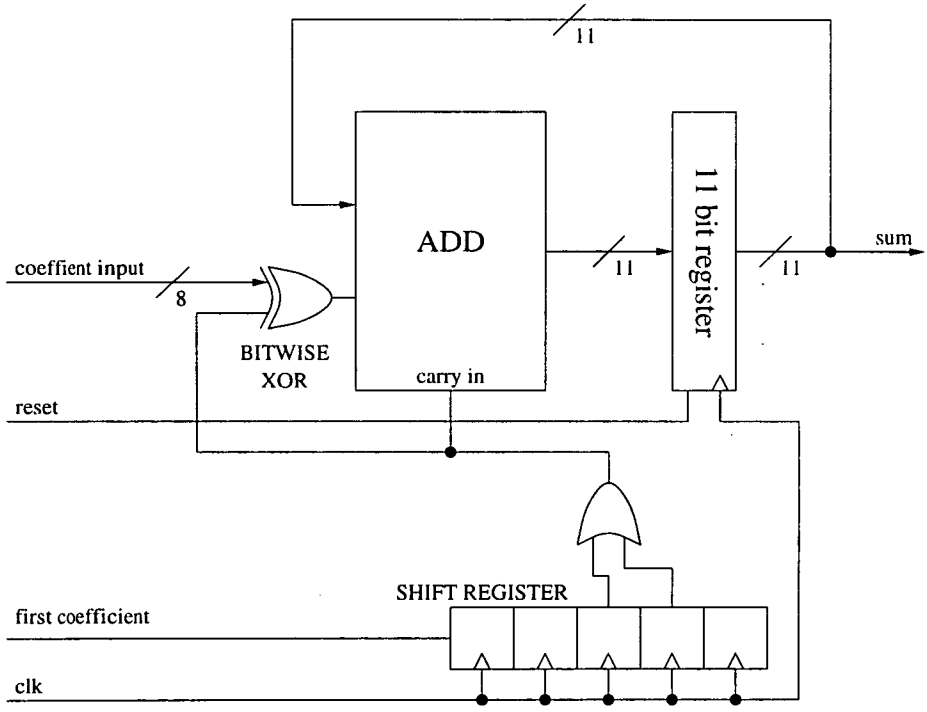


Figure 5.5: A serial summation unit

The coefficients χ_1 to χ_5 are loaded on the 8-bit input line serially, at each cycle of *clk*. First the *reset* line is asserted, to zero the accumulated sum. Then the first coefficient is loaded on to the 8-bit input bus and the reset is de-asserted. At the same time as the first coefficient being loaded onto the bus, the *first coefficient* input is asserted. This is

the input to the shift register which is used to determine which of the 5 coefficients is currently being added to the accumulated sum.

The connections between the shift register and the **xor** gate indicate that the corresponding a_k in the **a** sequence is -1. The connections in Figure 5.5 represent the sequence $(1, 1, -1, -1, 1)$.

If these summation units were used instead of the summation tree described in Section 5.2.2 then the number of addition units would only be reduced from 70 to 64 but it is likely that the amount of routing would also be reduced.

However, the previous summation circuit was fast because ripple addition was use. The same properties that were outlined in Section 5.2.2 cannot be used in the serial summation circuit because each coefficient has to be added to the accumulated sum before the addition of the next coefficient can be started. It is possible to pipeline the serial summation circuit, but this would mean introducing additional latches which would increase the area and power consumption of the circuit.

It would be better to use a carry-save addition unit as described in Section 3.8, with a carry propagation unit on the output. This would reduce the delay at each iteration from 11 full-adder cells to 1, with a delay of 11 full-adder cells at the output for carry-propagation.

In conclusion this serial design is smaller, although slower than the parallel circuit described in Section 5.2.2.

5.2.3 The Viterbi trellis

The trellis is implemented as interconnected add-compare-select (ACS) units. Examining Figure 5.6 we see that only half of the states are used at each time interval, hence we can use only one ACS unit to represent different trellis nodes at odd and even time intervals. The branch metric generation unit ensures that the correct corresponding branch metric values (BMVs) are fed to the ACS units at the correct time interval.

The trellis nodes are paired up so that each ACS unit is shared between two nodes. The pairs are chosen in the following way:

- An even time interval node is paired with an odd time interval node.

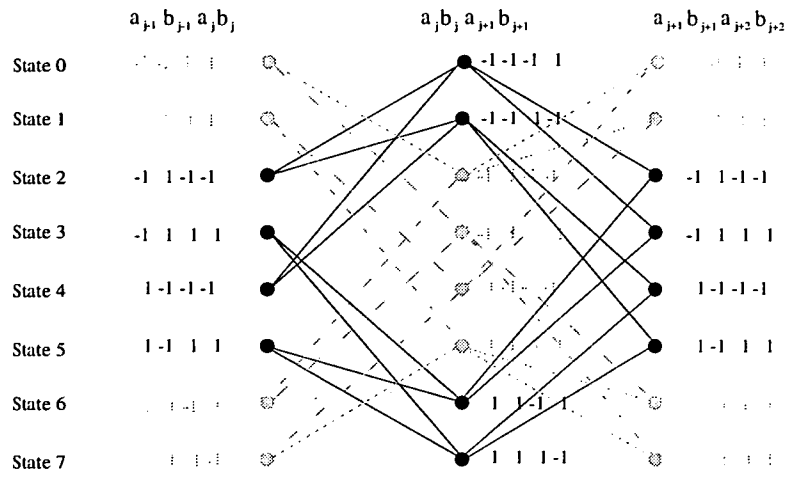


Figure 5.6: MSK trellis diagram

- Each pair of nodes are connected to 4 nodes which themselves form two pairs.

This ensures the minimum amount of routing between ACS units and that the routing between node pairs is never redundant at either odd or even time intervals.

Using a computer program which performed an exhaustive search on the states shown in Table 5.1, it was found that the state pairings are: $(0, 63), (1, 62), (2, 61), ..., (31, 32)$.

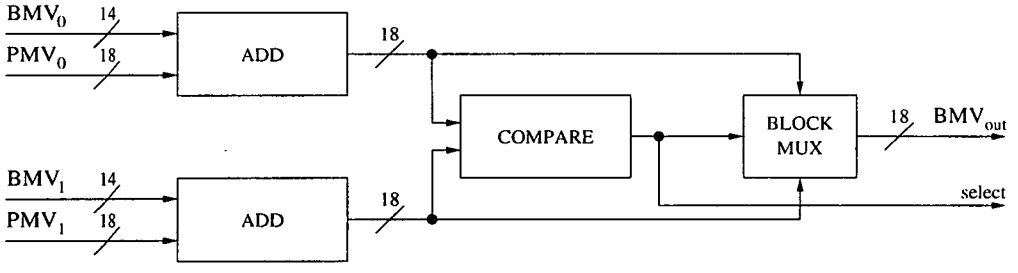


Figure 5.7: An add-compare-select unit

The ACS unit shown in Figure 5.7 implements the following function:

if $((PMV_0 + BMV_0) \geq (PMV_1 + BMV_1))$
  $PMV_{out} = PMV_0 + BMV_0$
  select= 0
else
  $PMV_{out} = PMV_1 + BMV_1$
  select= 1

The *select* signal is fed to the memory management unit which stores the path histories for all the trellis states.

For the reasons outlined in Section 5.2.2, the addition and comparison units are implemented using ripple propagation.

5.2.4 Branch metric calculation

When a new pair of in-phase and quadrature-phase symbols (y_n, z_n) are received, the incremental metric for each branch on the trellis diagram is computed. From equation 5.1 the incremental metric associated with a transition from state p to state q is:

$$\lambda_{p \to q} = a_n(y_n - I_s) + b_n(z_n - Q_s) \tag{5.9}$$

where a_n and b_n are the expected in-phase and quadrature-phase signals which would result in a transition from state p to state q.

From Section 5.2.3, state 0 and state 63 are paired and so they share the same ACS unit in the implementation of the Viterbi trellis. The branch metric values associated with these states can be derived from Table 5.1. These are:

$$\begin{aligned}
\lambda_{16 \to 0} &= -1(y_n - I_{16}) + 1(z_n - Q_{16}) \\
\lambda_{49 \to 0} &= -1(y_n - I_{49}) + 1(z_n - Q_{49}) \\
\lambda_{14 \to 63} &= 1(y_n - I_{14}) + 1(z_n - Q_{14}) \\
\lambda_{47 \to 63} &= 1(y_n - I_{47}) + 1(z_n - Q_{47})
\end{aligned}$$

These two states share two branch metric calculation cells, as well as the same ACS unit. At even samples the metrics for state 0 are computed and at odd samples the metrics for state 63 are computed. Figure 5.8 shows these two branch metric calculation

cells, where *sample* is 0 for even samples and 1 for odd samples.

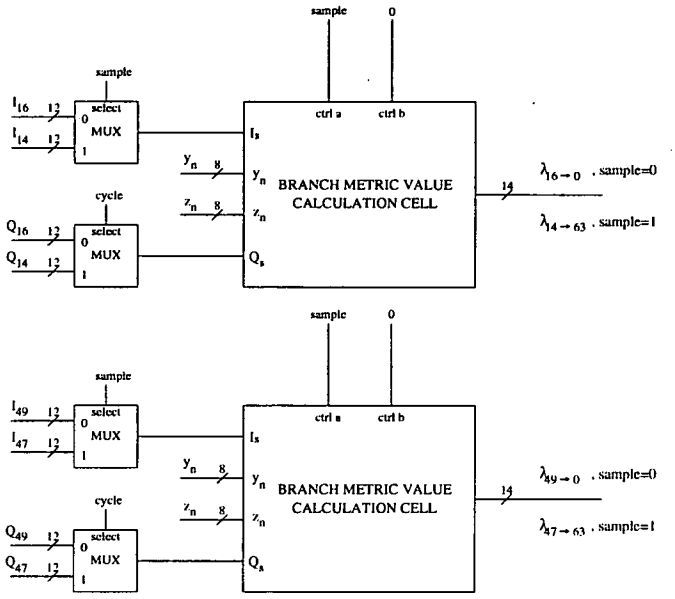


Figure 5.8: The first two cells of the BMV unit for ISI=5

The BMV generation cell is shown in Figure 5.9; it performs the sum $\pm(I_s - y_n) \pm (Q_s - z_n)$ where the $\pm$ operator is chosen by the values of *ctrl a* and *ctrl b*. For some ACS units, the control signals are the same for both trellis states so the logic can be optimized.

As discussed at the end of Section 5.2.3 the adders in the branch metric value cell are cascaded so that the delay through the adders is equivalent to a full-adder and one carry chain.

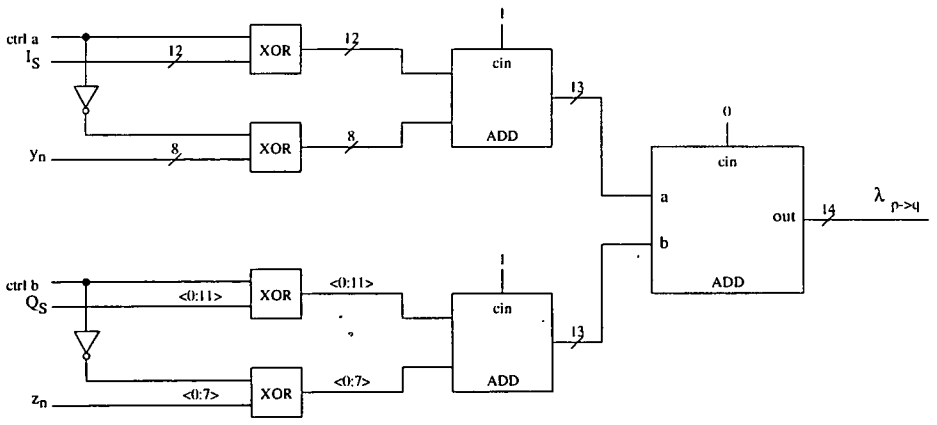


Figure 5.9: BMV cell

5.2.5 Metric normalisation

The metrics must be normalised so that they can be stored using finite length registers. In the conventional Viterbi algorithm, metrics are always positive and are normalised by observing when all the most significant bits are set and resetting them [22]. In the design the metrics can be both positive and negative twos-complement numbers. We normalise metrics of N bits in length if:

- All metrics are positive and $\geq 2^{N-2}$.
- All metrics are negative and $\leq 2^{N-2}$.

In the first case, bit $N-1=0$ and bit $N-2=1$, to normalise we reset bit $N-2$. In the second case, bit $N-1=1$ and bit $N-2=0$, to normalise we set bit $N-2$. The Normalisation unit is shown in Figure 5.10

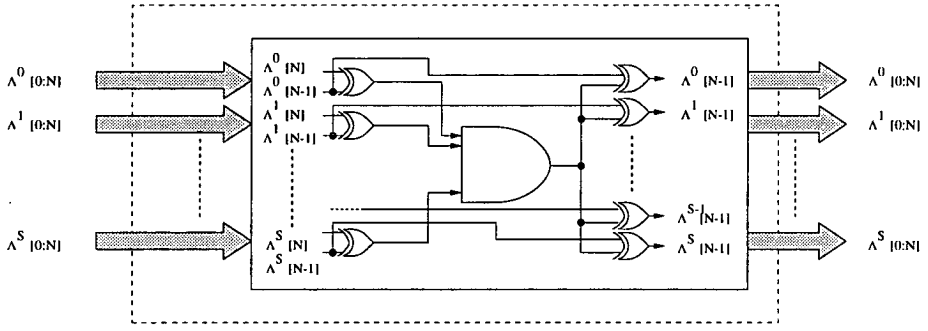


Figure 5.10: Metric Normalisation Unit

In [76], Hekstra proposed an alternative to metric normalisation in Viterbi decoders by exploiting the "wraparound" properties of twos-complement arithmetic. This method cannot be used in this design because we have both positive and negative incremental metrics [76].

For ISI=5, assuming our channel estimation coefficients have an accuracy of 8 bits, our path metric values are required to be stored using 18 bits. This means that normalisation occurs if all of the metrics are larger than 2^{16} or if all of the metrics are lower that -2^{16}.

5.2.6 Memory management

We have used a register exchange method of path history memory management for simplicity (see Section 4.2.3). However, the module has been specified so that it is compatible with a traceback management unit [59] to allow the possibility of future development.

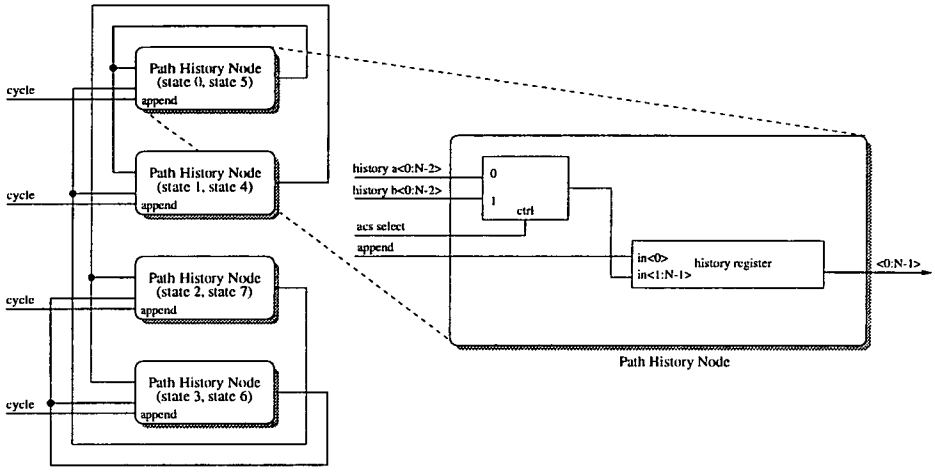


Figure 5.11: Register Exchange Memory Management Unit

Each state in the trellis appends either a 1 or a 0 to the path history. The *append* signal shown in the circuit is dependent on the trellis state and the cycle. For simplicity the output bit is determined by a majority vote on the oldest bit in all of the path histories which means that the path histories are longer but the circuitry for selecting the output bit is simple.

For ISI=5, we have 32 registers in the memory management unit. For majority voting the length of the path history register is 5-6 times the length of the path metric values [22]. This means that the path history registers should be roughly 90 bits in length. This is quite large which would suggest that a traceback method of path history memory management would be more desirable.

5.3 Performance

We need to estimate the delay though the circuit to determine whether CPL [71] will produce a satisfactory implementation. From the specifications of GSM [3] the VE

output is buffered to produce a continuous bitstream at *22.8 kbit/s*, hence the VE has $5ms$ to complete the decoding of a TDMA packet.

Each GSM TDMA packet contains 114 bits of data which need to be passed through the VE, in addition, the VE needs to be flushed with the same amount of bits as in the path history. For these iterations the branch metrics are all set to zero. With a path history of 90 bits, this means that our VE has to perform 204 iterations.

Assuming that the output determination unit has a shorter delay than the Viterbi trellis then the clock period T in CPL gate delays is:

$$T \geq (\Delta_{BMV} + \Delta_{ACS} + \Delta_{NORMALIZE}) \tag{5.10}$$

and the delay through the state constant generator followed by 204 iterations of the VE must be less than $5ms$.

Circuit Element	Delay (CPL gates)
State Constant Generator	20
Branch Metric Generator	16
Add-Compare-Select Unit	15
Normalization Unit	5

Table 5.2: CPL gate delays for various circuit elements

The number of CPL gate-delays through the various circuit elements are summarised in Table 5.2. Using these delays and equation 5.10 we can estimate a maximum gate-delay of $0.5\mu s$ which is clearly realisable.

5.4 Conclusions

This chapter has presented the architectural design of a Viterbi equalizer for GMSK signals as used in the GSM system of mobile radio communications. The design has implemented the theory of using the Viterbi algorithm to equalize signals encoded using an MSK modulation processes that was discussed in Chapter 2.

One of the major parts of design for the equalizer was the state constant generation circuit. The branch metric computation for the equalization requires a large number of summations (based on the impulse response of the communication channel) to be

calculated. The state constant generation circuit performs these summations. Since each packet contains a new estimate for channel, this needs to be performed for each frame. The circuit that was developed, which is presented here, is an incremental tree-based summation circuit. The design uses the relationships between summation sequences to reduce the number of adder units required from 1280 to 70. Also, the evaluation of all 128 sums requires only 7 adder unit delays. The design of a second possible summation circuit was also produced. This serial summation circuit is suitable for implementation using carry-save arithmetic. This circuit would reduce the number of addition units to 64, and have delay of 16 full-adder cells. It was noted during the design that a recent alternative to metric normalization, which exploits the wraparound properties of twos-complement arithmetic, cannot be used for the Viterbi Equalizer design because the incremental metrics can be positive or negative. For this reason the conventional normalization method has been extended to work with positive or negative twos-complement numbers.

Finally, the performance of the decoder design has been analysed and it has been shown that the speed requirement of the GSM communication system can be easily realised by this design.

Chapter 6

Implementation of the Viterbi Equalizer using Complementary Pass-Transistor Logic

6.1 Introduction

6.1.1 Motivation

Yano *et al* had presented the technique of Complementary Pass-transistor Logic (CPL) [7] for producing lower-power, and slightly higher speed circuits when compared with the equivalent CMOS circuit designs. The architectural design of the Viterbi equalizer discussed in the previous chapter had been developed with CPL in mind. A standard cell library of CPL functions was then produced to develop the silicon layout for the design. During the implementation, the performance of individual modules (including size, power, and speed) was not as good as had been expected. The implementation was suspended, and a detailed examination of the CPL technique was undertaken. It was soon discovered that Yano's results were biased, in that two examples presented in [7] (the full adder cell and the **xor** gate) were the only ones that gave increased speed, and decreased power. Even then, it was found that Yano's CMOS full adder cell was not very well optimised, resulting in a poor comparision. This chapter describes the investigation into CPL.

6.1.2 Overview

The VE architecture described in Chapter 5 consists of 32 interconnected add compare select (ACS) units and the path history module consists of 64 interconnected registers.

It can be seen that the design, when implemented, would be quite large. For a portable handheld receiver is would be desirable to minimise this area, and minimise the power consumed. To achieve this without altering the VE architecture it was decided to investigate a number of logic styles which provide low power consumption and low area utilisation, and choose one to implement the VE architecture.

6.1.2.1 Low power logic styles

In recent years, advancements in the production of integrated circuit technology have resulted in VLSI circuits with smaller and smaller feature sizes. This reduction in size has brought a considerable increase in speed and there are now many complex and fast CMOS ICs available. However, the need for high speed digital circuits to implement complex algorithms such as RSA encryption [50], MPEG encoding/decoding and the Viterbi Algorithm [77] is increasing faster than the CMOS technology.

Circuit speed can be increase by architectural improvements such as pipelining and parallelisation [78], but also by new logic circuits such as pass transistor circuits [7] which have been developed as an alternative to conventional CMOS logic design.

In addition, a growing emphasis on low power circuit design has emerged [78], [71]. With organisations such as CEPT developing standards for mobile digital radio communications [3], a need for low power implementations of complex algorithms has become more important, especially since battery technology has not improved significantly in recent years [78].

In the previous chapter, the architectural design for a Viterbi equalizer (VE) was described. It was suggested that logic design style was proposed by Yano *et al* in 1990 called *complementary pass-transistor logic* (CPL) [7] would be a good choice for implementing the VE on silicon. Yano *et al* claimed that CPL offers lower power dissipation than conventional CMOS logic. The equalizer is designed for use in the GSM mobile telephone system so a low-power implementation would be useful for a handheld receiver. In this chapter, logic design using CPL is reviewed. The performance of CPL for various logic functions is analysed relative to CMOS implementations. The development of a CPL standard library is outlined, and the implementation of the VE, proposed in Chapter 5 is described. Finally the results of implementing the VE design using this standard library, and the reasons for discontinuing the design will be explained.

6.2 Complementary pass-transistor logic

CPL features a logic network consisting of N-type transistors only. Since there are no P-type transistors, the complement of the each input and output signal is required. This allows the full range of logic functions to be produced using a small number of transistors. Figure 6.1 shows the generic schematic diagram for a CPL circuit.

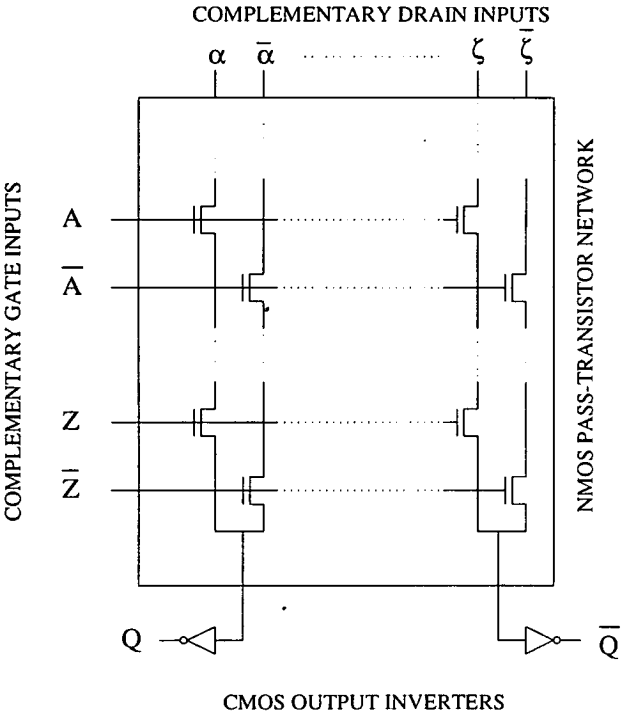


Figure 6.1: A generic CPL schematic diagram

The output of the logic network is buffered using standard CMOS inverters which restore the outputs to a full logic level.

It should be noted that because signals and their complements are available, an **and** gate is exactly the same as a **nand** gate, this is not the case with CMOS. This means that by using CPL more flexibility is available in terms of the choice of logic functions.

Figure 6.2 shows a transistor circuit for a **nand/and** gate in CPL.

2-input boolean functions can be design using the same transistor network of four transistors shown in Figure 6.2. Only the input and output connections need to be altered, all of the internal connections are the same [71].

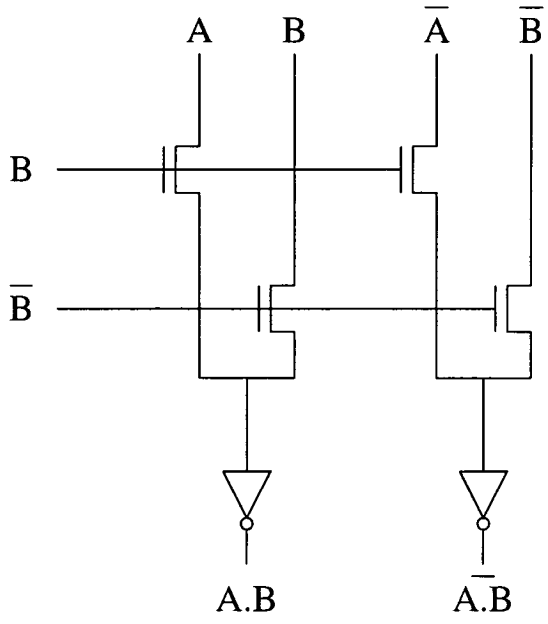


Figure 6.2: A **nand/and** gate in CPL

It should be noted that the standard CPL circuit shown here suffers from sub-threshold current [78] which results in static power dissipation. This is shown in Figure 6.3 which shows a CMOS inverter being driven by a single N-type pass transistor. The threshold drop of the N-type is V_{Tn}, which means that the input to the inverter is $V_{DD} - V_{Tn}$. This means that the gate-to-source voltage of the P-type is $-V_{Tn}$ and the P-type is not completely turned off. This results in a significant sub-threshold dc current, as shown by the dashed line.

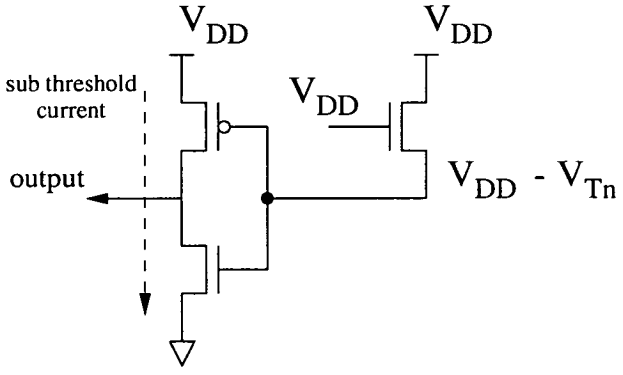


Figure 6.3: Static power consumption in the CPL output inverter

This static power consumption can be eliminated by using a pair of cross-coupled P-type transistors as shown in Figure 6.4, although this increases the circuit area and

decreases the circuit speed [78].

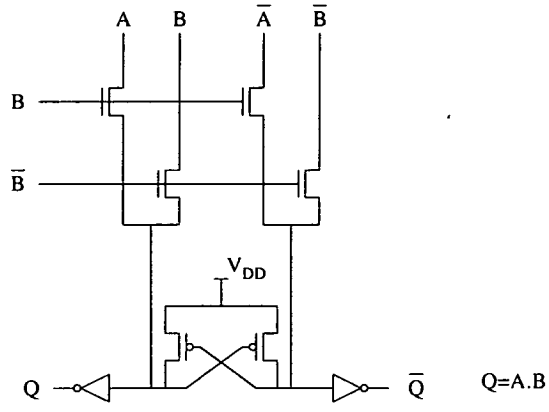


Figure 6.4: A **nand/and** gate in CPL with cross-coupled p-types

6.2.1 Design of standard logic functions in CPL

A CPL gate can be thought of as implementing two separate functions. The schematic diagram for a 2-input **nand/and** gate shown in Figure 6.4 shows that there are two N-type logic networks, one implementing A.B and the other $\overline{\text{A.B}}$.

To design a CPL transistor diagram for any 2-input boolean function, one of the inputs is selected as the *control* input. This control input (and its complement) will form the transistor gate inputs (as shown in Figure 6.1). The output is expressed in terms of this control input and the remaining input. In the case of $A.B$, with B as the control input:

1. if $B = 0$ then $A.B = B$.
2. if $B = 1$ then $A.B = A$.

This gives the lefthand transistor network in Figure 6.4. The righthand network is identical to the lefthand one with the drain inputs inverted.

Figure 6.5 shows the schematic diagram for a 3-input **nand/and** gate. It can be seen from the diagram that similar techniques can be used to create a 3-input CPL gate as were used to create 2-input one. This time two of the 3-inputs are selected as control inputs. The output is expressed in terms of one of the 3-inputs for each of the four possible values of the control signals. In the case of the 3-input **nand/and** gate with control inputs A and B:

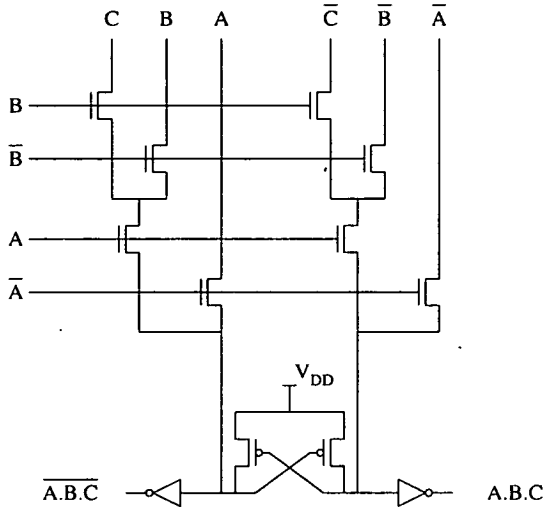


Figure 6.5: A 3-input **nand/and** gate in CPL

if	$A = 0$	$A.B.C = A$
else if	$B = 0$	$A.B.C = B = B$
else		$A.B.C = C$

Which gives the lefthand network in Figure 6.5. Again, the righthand network is identical with the drain inputs inverted.

Alternatively, more complicated 3-input functions can be implemented as cascaded 2-input gates. Figure 6.6 shows a 3-input **xnor/xor** gate which is constructed for two 2-input **xnor/xor** gates. This allows more complex 3-input functions, such as a full adder, to be implemented by decomposing into 2-input functions, which can be cascaded to produce a circuit with a single CPL gate delay.

6.2.2 Comparison of CPL and CMOS logic functions

To asses the merits of this logic style, a number of 2-input and 3-input CPL and CMOS functions have been simulated and compared. The circuits were designed at the transistor level using Cadence Design Framework II and the layout for all the circuits were produced using the Cadence layout synthesis tools (LAS). Simulations of the extracted version of the layout were performed using HSpice.

Table 6.1 shows the results for 2-input CMOS **nand**, **nor**, **xor** gates and Table 6.2 shows the results for 2-input CPL **nand/and**, **or/nor**, **xor/xnor** gates. The power was

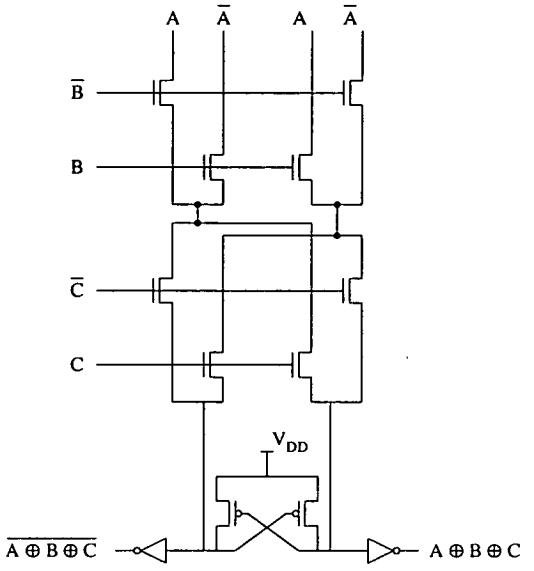


Figure 6.6: A 3-input **xnor/xor** gate in CPL

obtained by simulating a circuit consisting of a tree of seven of these cells, with the same random test vectors presented to all of the circuits. The relative power is shown to provide a straightforward comparison between different circuits. The CMOS **nand** gate was set at a power of 100.00 and all the other powers are shown relative to this.

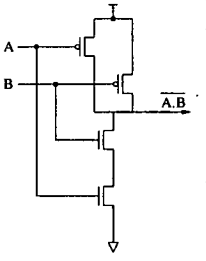
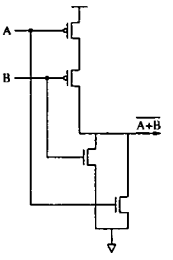
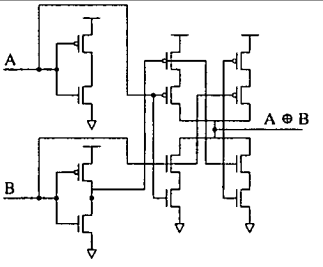
Function	**nand**	**nor**	**xor**
Circuit			
Transistor count	4	4	12
Area	382.03μm^2	412.38μm^2	1336.67μm^2
Delay	1.13ns	1.64ns	2.36ns
Relative Power	100.00	110.93	535.66

Table 6.1: Simulation results for 2-input CMOS logic functions

Comparing the results from Table 6.1 and Table 6.2 it can be seen that for the 2-input **nand** and **nor** gates, the CPL implementations are more than three times the size of the CMOS circuit. The delays of the CPL **nand** and **nor** gates are greater than the

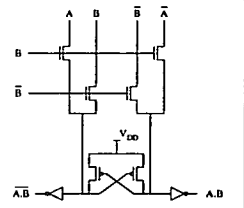
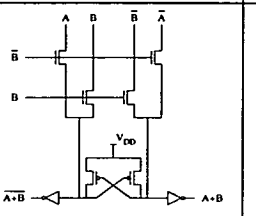
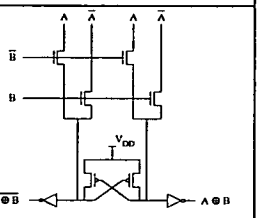
Function	**nand**	**nor**	**xor**
Circuit			
Transistor count	10	10	10
Area	1360.24μm^2	1392.92μm^2	1204.05μm^2
Delay	2.47ns	2.05ns	2.33ns
Relative Power	195.05	286.75	532.33

Table 6.2: Simulation results for 2-input CPL logic functions

CMOS delays. It can be seen that the power dissipation is double for the CPL **nand** gate and for the **nor** gate the power is almost three times that of the corresponding CMOS circuit.

Finally, the tables show that the CMOS and CPL **xor** circuits are very similar in terms of performance. The CPL **xor** is comparable to the CMOS **xor** gate for area, delay and power consumption.

These results suggest that for simple 2-input boolean function, CPL does not offer any advantages over CMOS.

Table 6.3 and Table 6.4 show similar results for 3-input CMOS and CPL functions. Again, the power is shown relative to the CMOS **nand** gate. The power is calculated in the same manner as before, with a tree circuit of the relevant gates, and 50 random input vectors.

It can be seen from Table 6.3 that the CMOS **xor** gate (which is constructed from two cascaded 2-input **xor** gates) is over four times as large than the other two gates. In addition, it dissipates 15 times more power than the **nand** gate.

Table 6.4 shows that all three 3-inputs gates require the same number of transistors in CPL. It can also be seen that the **nand** and **nor** gates are roughly four times the size of the corresponding CMOS gates. The CPL **nand** gate is half the speed on the CMOS **and** gate but both the **nor** and **xor** gates are faster, the **xor** gate is 22% faster than the CMOS implementation.

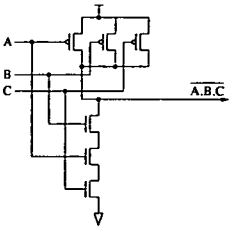
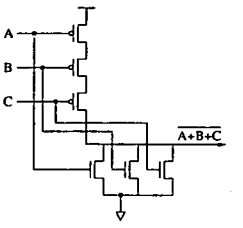
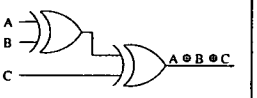
Function	**nand**	**nor**	**xor**
Circuit			
Transistor count	6	6	24
Area	543.40μm^2	548.46μm^2	2746.20μm^2
Delay	1.61ns	3.03ns	4.13ns
Relative Power	100.00	180.55	1514.83

Table 6.3: Simulation results for 3-input CMOS logic functions

The table also shows that the power consumption for the CPL gates is very similar to their CMOS equivalents, except in the case of the **xor** which requires half of the power.

Finally, Table 6.5 shows comparisons for CMOS and CPL full adder cells, which are shown in Figure 6.7 and Figure 6.8 [7], respectively (note that for compactness the cross-coupled p-type transistors are not shown).

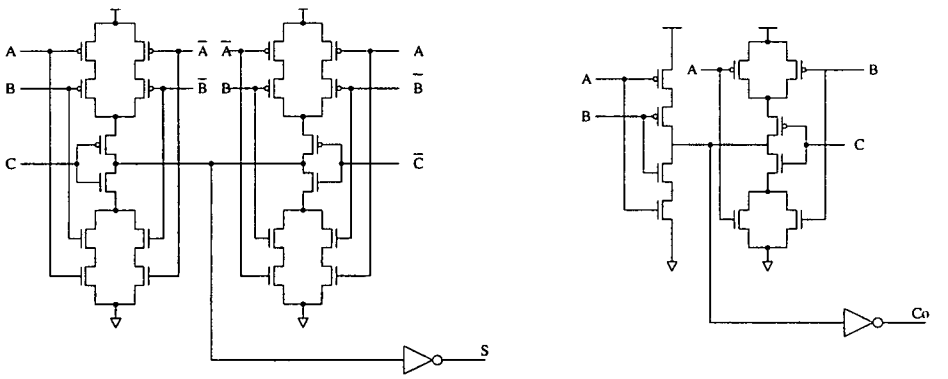


Figure 6.7: A CMOS full adder circuit

The same technique was used for these comparisons as was used for the previous ones. The table shows that the CPL design uses significantly less silicon area, and is 30.8% faster. For each of the three different supply voltages the CMOS full adder has been given a relative power of 100.

In [7], Yano *et al* claimed a reduction in power dissipation of 28.3% at 3V compared to CMOS while our results show an reduction of 10% for 3V, although this rises to an

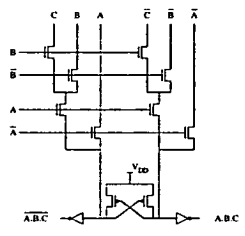
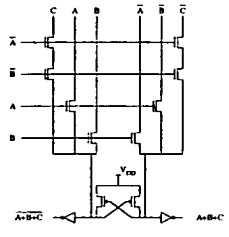
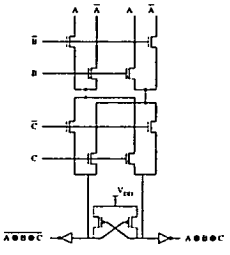
Function	**nand**	**nor**	**xor**
Circuit			
Transistor count	14	14	14
Area	2027.69μm^2	2002.02μm^2	2176.51μm^2
Delay	3.16ns	2.72ns	3.22ns
Relative Power	117.94	188.14	774.79

Table 6.4: Simulation results for 3-input CPL logic functions

CPL and CMOS Full Adder comparisons		
Circuit Style	CMOS	CPL
Transistor count	40	32
Area	7241.52μm^2	5612.96μm^2
Delay (at 3V)	6.62ns	4.59ns
Relative Power (5V)	100	82.31
Relative Power (4V)	100	85.30
Relative Power (3V)	100	90.86

Table 6.5: Simulation results for CPL and CMOS full adders

18% reduction for 5V.

It can be concluded from these results that CPL is best suited to designs which require a large amount of complex multi-input functions, such as **xor** gates. The CPL full adder provides a reduction in power of 10% for a 3V supply and it is significantly smaller and faster.

6.3 Improvements to conventional CPL

Since CPL was first proposed by Yano *et al* [7], a number of similar logic styles based on CPL have been suggested. These will be summarised here to provide an idea of the

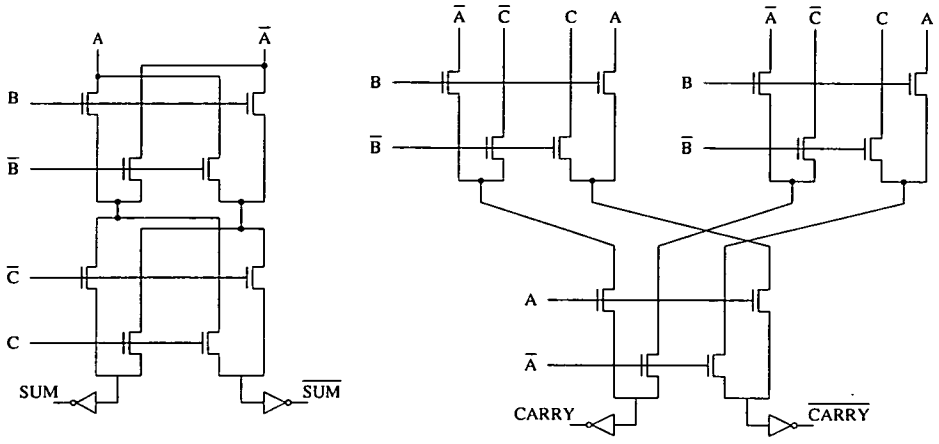


Figure 6.8: A CPL full adder circuit

current "state of the art".

6.3.1 Modified CPL

Rather than use the cross-coupled p-type transistors, a small p-type transistor can be used in the manner depicted in Figure 6.9 [79]. This will reduce the sub-threshold currents, and reduce power consumption. However, the propagation delay is increased.

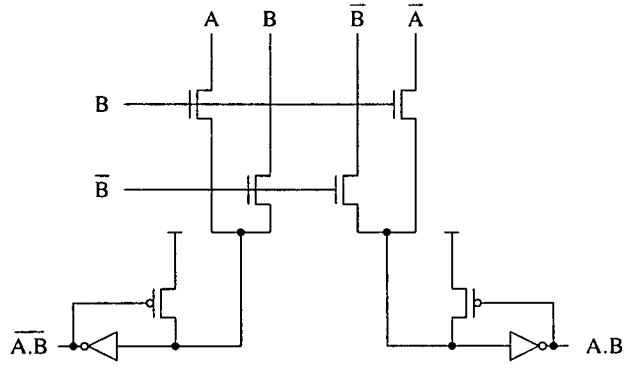


Figure 6.9: A **nand/and** gate in CPL with a p-type feedback transistor

The transistor count, and the power consumption, can be decreased further with the circuit shown in Figure 6.10 [79]. Only one of the two transistor networks is used, an additional inverter is used to generate the complementary output. This reduces the transistor count by half, but increases the propagation delay by the delay of a CMOS inverter. According to Ko *et al* [79] this CPL based circuit can reduce the power

consumption by almost a third while only increasing the delay by roughly 10%.

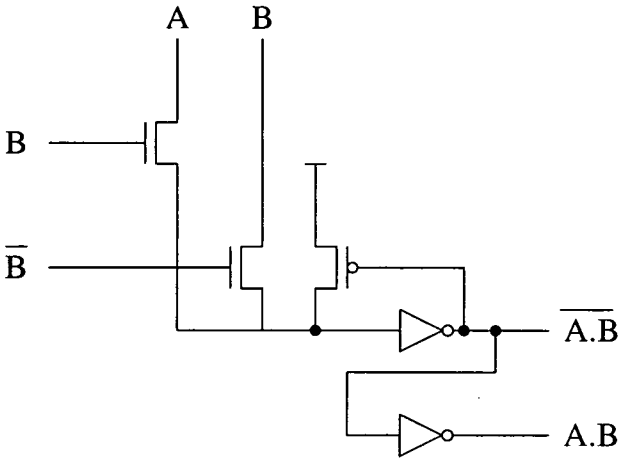


Figure 6.10: A **nand/and** gate in modified low-power CPL with double inverter

6.3.2 Dual pass-transistor logic

In dual pass-transistor logic (DPL) [80], both p-type and n-type transistors are used in the logic network. P-types are used to pass a logic level 1 and n-types are used to pass a logic level 0.

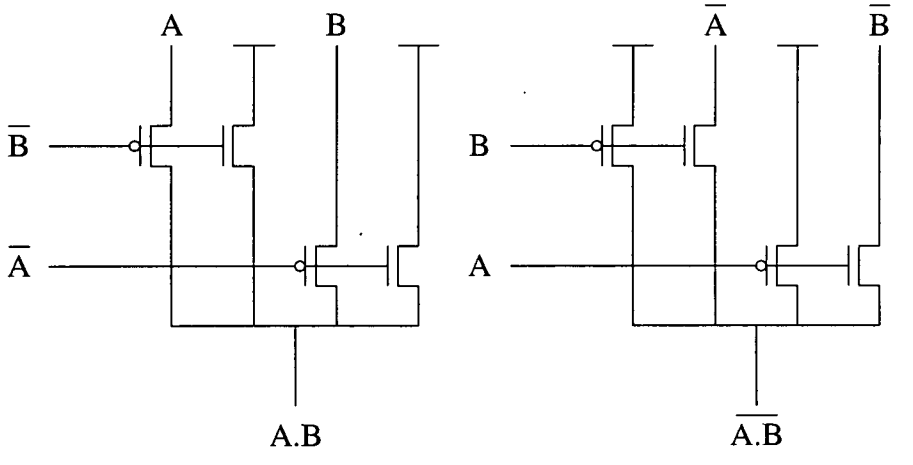


Figure 6.11: A **nand/and** gate in DPL

Figure 6.11 shows a 2-input **nand/and** gate in DPL. Looking at the transistor network on the left it can been seen that for any combination of inputs, there are always two

current paths driving the output. This means the output is also at a full logic level so there are no problems due to threshold voltage drop and there is no need for cross-coupled p-types on the input to the output inverters.

6.3.3 Swing restored pass-transistor logic

The final CPL based circuit style that will be examined is swing restored pass-transistor logic (SRPL) [81]. This is the same as CPL with the output inverters and the cross-coupled p-types replaced by a pair of cross-coupled inverters.

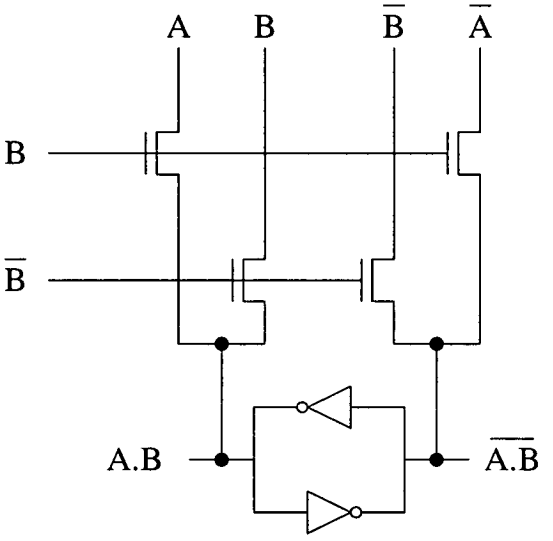


Figure 6.12: A **nand/and** gate in SRPL

Figure 6.12 shows a 2-input **nand/and** gate using SRPL. The sizing of the transistors both in the inverters and the network affects the speed and power dissipation of the circuit greatly [71].

6.3.4 Discussion

Bellaouar and Elmasry compared these circuit styles (except modified CPL) and concluded that CPL and SRPL result in the smallest circuits [71]. It has been shown that modified CPL with two cascaded inverters has the smallest area [79]. According to Bellaouar and Elmasry, SRPL has the lowest power dissipation [71] and DPL has both the largest area and power consumption.

In 1996, during this investigation, Yano *et al* proposed a new methodology for design circuits with pass-transistor logic [8] which utilises the advantages of CPL but with reduced routing since complements are not propagated. A circuit synthesis tool called a *circuit inventor* is used to represent a design in terms of three basic pass-transistor based cells (Yano referred to them as "Y" cells due to the shape of the transistor network, here we will use the term *single-output-PL*, or SO-PL) and four inverters of varying size.

Gates are constructed using these seven cells, and the complements are generated locally using small inverters. The gates do not produce complementary outputs which halves the size of the transistor network compared to conventional CPL. When complementary signals are required, they are generated locally. Results produced by Yano *et al* show that a full adder based on this technique dissipates 53% of the power dissipated by a CMOS full adder [8].

It was decided to implement the VE described in Chapter 5 using a mixture of CPL and SO-PL. The ACS units are connected by 18-bit buses. If full CPL was to be used for the design, these buses would have be 36 bits wide (including complements). The SO-PL cells would be used at the output of the ACS units and local inverters would create the complements at the inputs. The majority of the VE architecture is made up of full adder and **xor** cells, with a small proportion of the design being 2-input **and** and **or** functions. It was decided to use CPL for all of these cells (even though our comparisons favoured CMOS for these simple 2-input functions) because the outputs of the 2-input functions are used by CPL gates which require complementary inputs. An alternative would be to have complements generated from local inverters.

6.4 Developing a standard library of CPL functions

The VE was developed using Verilog HDL and the low level gates required are:

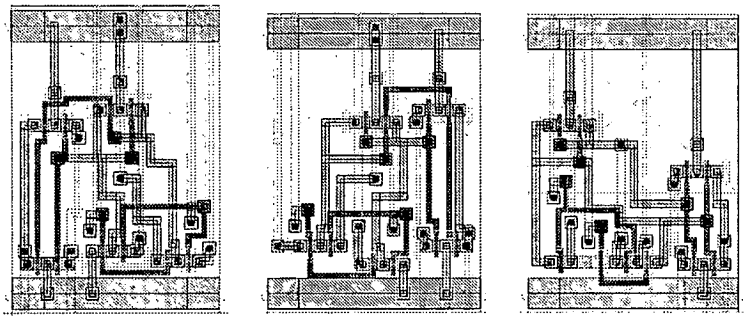
- 2-input **and** gate
- 2-input **or** gate
- 2-input **xor** gate
- 4-input **or** gate

- 2-1 **mux**
- Full adder
- Half adder
- Reset latch

The cells were designed using schematic capture in Cadence Design Framework II. The layout was generated using the layout synthesis tool (LAS) with a $1\mu m$ process, the simulations used typical industrial delays. The automatic place and route tools would be used to generate the VE chip layout.

All of the cells had to have the same separation of power and ground rails, so LAS was used to generate the best layout for each of the cells, the largest power and ground rail separation of all the cells was then noted and LAS was re-run on all the other cells with this separation as a constraint.

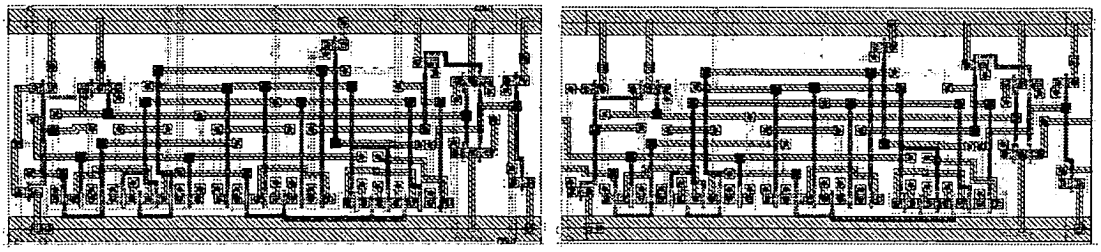
This means that the simple cells (such as the **and** and **or** gates) are inefficient in terms of area, while the larger, more complex cells (such as the adder cells) are more compact. Figure 6.13 shows the layout for the **and**, **or** and **xor** standard cells and Figure 6.14 shows the layout for the full adder cells.



(a) (b) (c)

Figure 6.13: Layout for CPL based **and**, **or** and **xor** standard cells.

Two different full adders and three different half adder cells were designed. To reduce the amount of routing generated in the automatic place and route stage, the carry inputs and outputs from the full adder (and the carry outputs from the half adder) were put on the left and right edges of the cell. This meant that a large ripple adders could be constructed by placing the cells end-to-end.



(a) (b)

Figure 6.14: Layout for the two CPL based full adder standard cells.

Since these connections were in the metal 1 layer (the same as the power and ground rails) additional layout cells for the half and full adders had to be designed with the carry inputs and outputs in the metal 2 layer at the top and bottom of the cells. This can been seen in Figure 6.14 (b) which shows the full adder layout with the carrys at the left and right sides, and Figure 6.14 (a) which shows the full adder layout with carrys at the top and bottom of the cell.

The half adders have been designed in the same way, but there is also a half adder cell with a hardwired carry in of 1 for the subtraction circuits. Table 6.6 shows the full CPL standard library with size and delays.

Note that the cell delays in this table are shorter than the results of the simulations previously shown in this Chapter. This is because in the default setup the layout synthesis tool automatically uses polysilicon routing between transistors if it determines that it is preferably to using metal. This sometimes generates slower circuits (if too much poly routing is used) although often it generates faster ones (since there are less vias). It was decided from the previous simulations that poly routing should be totally disabled, to give a fairer comparison between the logic styles (and not the implementation of those styles). In constructing the standard library, the poly routing options were varied to achieve smaller and faster circuits.

As stated at the beginning of this section, a smaller library of standard cells based on the SO-PL cell methodology [8] was developed. The memory management unit in the VE (see Section 5.2.6) consists of interconnected shift registers, which only pass data between each other at each iteration. Since speed was not critical for this part of the architecture, little would be gained by implementing this in CPL, instead this unit was implemented using the SO-PL cell standard library.

The full CPL standard library			
Cell name	function	Area (μm^2)	Delay (ns)
CPLAND2	2-input **and** gate	1547.15	1.02
CPLOR2	2-input **or** gate	1644.15	1.04
CPLXOR2	2-input **xor** gate	1623.75	1.05
CPLAND4	4-input **and** gate	2982.75	2.55
CPLMUX21	2-1 **mux**	1644.15	1.08
CPLHADD0	Half adder (Cin = 0)	2725.70	1.64
CPLHADD1	Half adder (Cin = 1)	2725.70	1.61
CPLHADD2	Half adder (Top/Bottom)	2725.70	1.10
CPLFADD0	Full adder (Left/Right)	5097.35	1.82
CPLFADD2	Full adder (Top/Bottom)	5097.35	1.83
CPLLATR	Reset latch	2231.00	-

Table 6.6: The full CPL standard library

The Viterbi trellis (see Section 5.2.3) features 32 interconnected add-compare-select units. These units transfer the path metric values, of 18 bits in width, between each other. To reduce the amount of routing the complements of the path metrics are generated locally, and SO-PL cells are used for the block mux (see Figure 5.7), since complementary outputs are no longer necessary. Since the complements are not available for the metric normalisation unit (Section 5.2.5), this unit is also implemented using SO-PL cells. Table 6.7 shows the SO-PL cell standard library.

The SO-PL cell standard library	
CPLYAND2	2-input **and** gate
CPLYAND4	4-input **and** gate
CPLYXOR2	2-input **xor** gate
CPLINV1	small inverter
CPLINV2	large inverter
CPLLATR	reset latch
CPLLATR2	reset latch with complementary outputs
CPLYMUX21	2-1 **mux**

Table 6.7: The CPL SO-PL cell standard library

6.5 Implementation of the Viterbi equalizer using the CPL and SO-PL standard libraries

The Verilog HDL code for the design was imported into Cadence Design Framework II which mapped the cells in the code to the custom cells in the CPL and the SO-PL libraries. The Cell Ensemble automatic place and route tools were used to produce the chip layout.

Rather than place and route the whole chip at one, four subsections were identified and implemented separately: the state constant generator; the branch metric calculation unit; the add compare select trellis; and the path history module.

For the trellis, the placed and unrouted area was large at $7656\mu m \times 3613\mu m$. However, after routing this increased to $10186\mu m \times 24354\mu m$, roughly $5cm^2$. Similarly the placed and routed path history module was also large $5532\mu m \times 18282\mu m$, $1cm^2$. The large area utilisation was due to the wide buses which connect the trellis node (and the path history registers), and also the fact that the architecture was a fully parallel design.

Because of the size of the routed modules, and the fact that CPL no longer appear to produce the power improvement initially expected, it was decided not to proceed with the implementation of the VE design using CPL.

6.6 Conclusions

This chapter has reviewed low power logic styles. Specifically it has described CPL, which attempts to produce low-power designs by using complementary input and output signals. This allows a smaller number of transistors to be used for the same logic functions when compared to CMOS. This lower number of transistors is the main source of any lower power consumption. CPL is flexible, in that many logic functions can be produced with a single gate delay, although this is limited by the voltage drop across the transistors.

CPL and CMOS have been compared like-for-like with a number of common logic functions. It was shown that for simple functions, CMOS is smaller and requires less power. Only for more complex functions such as cascaded **xor** gates and full-adders does CPL begin to show any improvements over CMOS.

Improvements to conventional CPL have also been reviewed. Modified CPL claims to reduce power consumption by one third, and increase delay by 10%. Dual pass-transistor logic uses both n-type and p-type transistors, although a double inverter is required to produce a complementary signal. Finally, SRPL has a pair of cross-coupled p-types on the output instead of inverters, however, speed and power dissipation are very sensitive to transistor sizing which makes circuit design more difficult.

The construction of a standard cell library of CPL has also been reviewed. This library was used to produce an implementation of the design in the previous chapter, although the resulting design was very large, and development was halted.

The major piece of original work in this chapter is the investigation into CPL. It was prompted because the performance improvement of CPL claimed by Yano *et al* in their original paper had been difficult to emulate. On closer investigation it was discover that in many cases, CPL circuits were actually slower, and had greater power consumption than their CMOS counterparts. Specifically, the CMOS full adder circuit that Yano *et al* had used in their paper, which they used to show a 28.3% speed-up for CPL, was not very well optimised. Comparing like with like, using the same layout software and routing algorithms, this research has suggested that a more realistic speed-up is 10%.

The implementation of the Viterbi design also highlighted another problem with CPL circuit styles. The requirement that a signal and its complement is needed, means that routing can become the dominant factor in anything other than the smallest designs.

It should be noted that Yano independently reached the same conclusions when he suggested an alternative to CPL in the form of SO-PL. This logic style uses local inverters to generate complementary signals, only when they are required. It also uses more complex cells than the standard Boolean logic functions, because that is where the advantage of CPL lies.

Chapter 7

A Low Area Serial Viterbi Equalizer Implementation

7.1 Introduction

7.1.1 Motivation

After the initial Viterbi equalizer design was halted, it was decided that an alternative implementation should be developed, and the conclusions of the previous implementations should be considered. A low-power design was still the goal. Using the lessons learned from previous implementations, the preferred circuit style would be SO-PL, however for speed of implementation it was decided to use a conventional CMOS process. In addition, the redundant number system was discarded. This chapter describes the low-area serial Viterbi equalizer.

7.1.2 Overview

The results of implementing the fully parallel Viterbi equalization module using CPL, as described in Chapter 6, showed that a fully parallel module results in a large use of silicon area. The use of CPL also has the effect of increasing the silicon area due to the increase in routing between cells. This additional routing had a substantial effect on the parallel architecture which already had large amounts of routing between the node processors.

When designing a VE module for a hand-held mobile GSM receiver, it is preferable to develop a serial module which has a smaller number of add compare select (ACS) units, than trellis nodes.

In this chapter the design of a serial VE module is described. From the results of Chapter 6 it was concluded that single-output CPL cells, such as those proposed by Yano [8] are the desired logic style since these are smaller and require less powers that conventional CPL cells, and do not require the distribution of a signal and its complement, which resulted in the large amount of routing that was present in the implementation of the fully parallel VE. However, for speed of implementation we have used the ES2 $0.7\mu m$ CMOS process [82].

Section 7.2 describes the overall VE module, the following 8 sections will describe each of the units in the module in more detail. Section 7.11 describes the implementation of the design using Verilog HDL [83], the ES2 design kit, and Cadence Design Framework II. Section 7.11.3 outlines the testing and the layout of the design and Section 7.12 describes the specifications of the implementation. Finally in Section 7.13 some conclusions are drawn.

7.2 Serial Viterbi equalizer overview

The simplified block diagram in Figure 7.1 represents the serial VE module. The method of operation is similar to the fully parallel decoder design which is described in Chapter 5, only the developments for the serial architecture are discussed here.

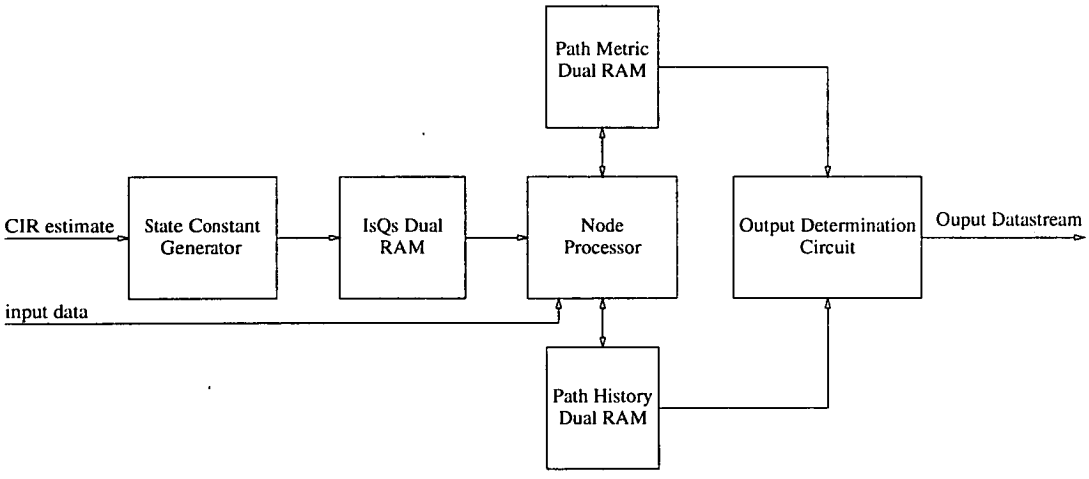


Figure 7.1: A block diagram of the Serial Viterbi Equalizer

The channel impulse response (CIR) is used to generate the state constants I_S and Q_S as described in equation 2.15 on page 22. Once computed, the state constants are stored

in a block of memory out of which they are read by the node processor (NP). Note from the diagram that the memory is a "dual" memory. This means that there is in fact two blocks of memory, while one block is being written to by the state constant generator (SCG), the other block is being read by the NP. This allows the SCG to compute the state constants for the next packet while the NP is decoding the current packet.

To have the SCG and the NP working on different packets, it is assumed that the input and output of the whole module is buffered. This is acceptable because due to the GSM block encoding scheme [3], the packets must be stored at the output of the VE module.

The NP contains a single ACS unit which represents each of the trellis nodes in turn throughout one iteration. This means that the path metrics values (PMVs) and path histories (PHs) cannot be stored at the corresponding node, as with the design in Chapter 5, these values are now stored in two RAM modules external to the NP.

The PMV and PH RAMs are also "dual" RAMs similar to the RAM for the state constants. This is because of the iterative nature of the VA which means that the results of the previous iteration need to be available for all of the required nodes in the subsequent iteration. To achieve this two identical blocks of memory are present in the two RAM modules, during odd and even iterations. Alternative RAMs are used for reading and writing.

Finally, the output determination unit is far simpler than the one required for the fully parallel version. Recall that there are two main methods of determining the output of the VA: majority voting [55] [56] or minimum/maximum metric select [76] [22] (Section 4.2.3). In the parallel version we were unable to use metric selection, even though the performance is considered to be better [15] because of the large area which would be required for a parallel metric selection circuit with 32 inputs. In the serial design we have designed a very small metric selection circuit, which selects the largest metric during the final iteration, and outputs the corresponding PH.

The following sections will describe the operation of each of the modules in more detail.

7.3 The state constant generator

Figure 7.2 shows a schematic diagram of the SCG circuit.

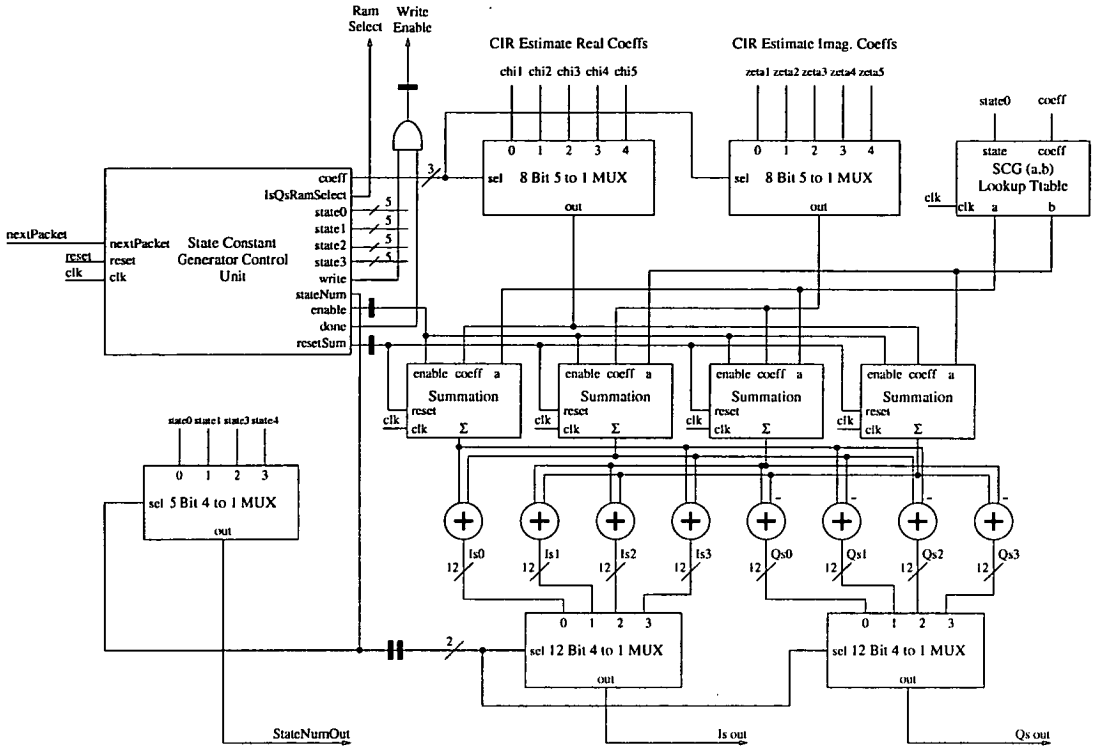


Figure 7.2: The state constant generator

To re-cap, the SCG computes the constants I_S and Q_S as defined by the equation 2.15 on page 22. This involves adding, or subtracting the CIR coefficients $\{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5\}$ and $\{\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5\}$ depending on the sequence $(\mathbf{a}, \mathbf{b})$ which describes the sequence of data corresponding to a path to state S.

The serial SCG shown in Figure 7.2 computes I_S and Q_S for 4 values of S simultaneously, by adding or subtracting the CIR coefficients in turn. When the 4 pairs of constants are computed they are written serially into the IsQs RAM module.

To understand the operation of the circuit, the transition table of the modulation process is shown in Table 7.2 and the control signals generated by the SCG control unit are shown in Table 7.1. Table 7.2 is a re-drawn version of Table 5.1 which shows: all 64 states of the trellis; the state pairs; the $(\mathbf{a}, \mathbf{b})$ sequence associated with each state; the input pair from the previous iteration; the pair of I_S and Q_S constants required for computing the two incoming metrics; the append bit for each state; and the iteration (odd or even) for which each state is valid.

It is known from equation 5.3 that the I_S and Q_S values are based on the $(\mathbf{a}, \mathbf{b})$ coefficients. Since the $(\mathbf{a}, \mathbf{b})$ coefficients for state $S = 32...63$ are the negative of the $(\mathbf{a}, \mathbf{b})$

Signal Names	Description
state0-3	The 4 states numbers , S, for which I_S and Q_S are being evaluated in parallel.
coeff	The index of the coefficients χ_n and ζ_n which are being added or subtracted during the current cycle.
write	0 if the constant are being computed and 1 if they are begin written out to RAM.
stateNum	If *write=1* then writeState is the state number S for which I_S and Q_S are being written to RAM.
enable	1 if the summations are being computed.
done	1 if I_S and Q_S have been evaluated for all values of S
resetSum	At the start of computation of the summation this is held high for one clock cycle to reset the sums.

Table 7.1: The state constant generator control signals

coefficients for states $S = 0...31$ (as in Section 5.2.2), it can be seen that only 64 out of the 128 I_S and Q_S values need to be computed.

In addition, it can be seen from the table that computing the four summations:

$$\sum_{m=n-L}^{n-1} a_n \chi_{n-m} \tag{7.1}$$

$$\sum_{m=n-L}^{n-1} b_n \zeta_{n-m} \tag{7.2}$$

$$\sum_{m=n-L}^{n-1} b_n \chi_{n-m} \tag{7.3}$$

$$\sum_{m=n-L}^{n-1} a_n \zeta_{n-m} \tag{7.4}$$

for state zero, allows the computation of I_S and Q_S for $s = 0, 15, 16, 31$ since $\mathbf{a}_{15} = \mathbf{b}_0$, $\mathbf{b}_{15} = \mathbf{a}_0$, $\mathbf{a}_{16} = -\mathbf{b}_0$, $\mathbf{b}_{16} = \mathbf{a}_0$, $\mathbf{a}_{31} = -\mathbf{a}_0$, and $\mathbf{b}_{31} = \mathbf{b}_0$. The same relationship applies for the quadruplets of states: $(1, 14, 17, 29), (2, 13, 18, 28), ..., (7, 8, 23, 24)$.

The schematic diagram in Figure 7.2 shows the circuit which computes the I_S and Q_S values. The black lines on some of the wires signify that the signals are latched by the system clock.

The control unit generates the timing signals which control the circuit. The inputs to the SCG are the CIR coefficients $\chi_1, \chi_2, ..., \chi_5$ and $\zeta_1, \zeta_2, ..., \zeta_5$ and the control signal *nextPacket*, which is held high for one clock cycle to signify that new CIR coefficients

State Number	Pair No.	(a, b) sequence	Input from pair	IsQs	Append bit	Cycle
0	0	1 1 1 -1 -1 -1 -1 1 1 1	16, 14	16, $\overline{17}$	-1	even
1	1	1 1 -1 1 1 1 1 -1 -1 -1	18, 12	18, $\overline{19}$	-1	even
2	2	1 1 1 -1 1 1 1 -1 -1 -1	20, 10	20, $\overline{21}$	-1	even
3	3	1 1 -1 1 -1 -1 -1 1 1 1	22, 8	22, $\overline{23}$	-1	even
4	4	1 1 1 -1 -1 -1 1 -1 -1 -1	24, 6	24, $\overline{25}$	-1	even
5	5	1 1 -1 1 1 1 -1 1 1 1	26, 4	26, $\overline{27}$	-1	even
6	6	1 1 1 -1 1 1 -1 1 1 1	28, 2	28, $\overline{29}$	-1	even
7	7	1 1 -1 1 -1 -1 1 -1 -1 -1	30, 0	30, $\overline{31}$	-1	even
8	8	1 1 1 -1 -1 -1 -1 1 -1 -1	16, 14	16, $\overline{17}$	1	even
9	9	1 1 -1 1 1 1 1 -1 1 1	18, 12	18, $\overline{19}$	1	even
10	10	1 1 1 -1 1 1 1 -1 1 1	20, 10	20, $\overline{21}$	1	even
11	11	1 1 -1 1 -1 -1 -1 1 -1 -1	22, 8	22, $\overline{23}$	1	even
12	12	1 1 1 -1 -1 -1 1 -1 1 1	24, 6	24, $\overline{25}$	1	even
13	13	1 1 -1 1 1 1 -1 1 -1 -1	26, 4	26, $\overline{27}$	1	even
14	14	1 1 1 -1 1 1 -1 1 -1 -1	28, 2	28, $\overline{29}$	1	even
15	15	1 1 -1 1 -1 -1 1 -1 1 1	30, 0	30, $\overline{31}$	1	even
47	16	-1 -1 1 -1 1 1 -1 1 -1 -1	31, 1	31, $\overline{30}$	1	even
46	17	-1 -1 -1 1 -1 -1 1 -1 1 1	29, 3	29, $\overline{28}$	1	even
45	18	-1 -1 1 -1 -1 -1 1 -1 1 1	27, 5	27, $\overline{26}$	1	even
44	19	-1 -1 -1 1 1 1 -1 1 -1 -1	25, 7	25, $\overline{24}$	1	even
43	20	-1 -1 1 -1 1 1 1 -1 1 1	23, 9	23, $\overline{22}$	1	even
42	21	-1 -1 -1 1 -1 -1 -1 1 -1 -1	21, 11	21, $\overline{20}$	1	even
41	22	-1 -1 1 -1 -1 -1 -1 1 -1 -1	19, 13	19, $\overline{18}$	1	even
40	23	-1 -1 -1 1 1 1 1 -1 1 1	17, 15	17, $\overline{16}$	1	even
39	24	-1 -1 1 -1 1 1 -1 1 1 1	31, 1	31, $\overline{30}$	-1	even
38	25	-1 -1 -1 1 -1 -1 1 -1 -1 -1	29, 3	29, $\overline{28}$	-1	even
37	26	-1 -1 1 -1 -1 -1 1 -1 -1 -1	27, 5	27, $\overline{26}$	-1	even
36	27	-1 -1 -1 1 1 1 -1 1 1 1	25, 7	25, $\overline{24}$	-1	even
35	28	-1 -1 1 -1 1 1 1 -1 -1 -1	23, 9	23, $\overline{22}$	-1	even
34	29	-1 -1 -1 1 -1 -1 -1 1 1 1	21, 11	21, $\overline{20}$	-1	even
33	30	-1 -1 1 -1 -1 -1 -1 1 1 1	19, 13	19, $\overline{18}$	-1	even
32	31	-1 -1 -1 1 1 1 1 -1 -1 -1	17, 15	17, $\overline{16}$	-1	even
63	0	1 -1 1 1 -1 1 -1 -1 1 -1	14, 16	14, $\overline{15}$	1	odd
62	1	1 -1 -1 -1 1 -1 1 1 -1 1	12, 18	12, $\overline{13}$	1	odd
61	2	1 -1 1 1 1 -1 1 1 -1 1	10, 20	10, $\overline{11}$	1	odd
60	3	1 -1 -1 -1 -1 1 -1 -1 1 -1	8, 22	8, $\overline{9}$	1	odd
59	4	1 -1 1 1 -1 1 1 1 -1 1	6, 24	6, $\overline{7}$	1	odd
58	5	1 -1 -1 -1 1 -1 -1 -1 1 -1	4, 26	4, $\overline{5}$	1	odd
57	6	1 -1 1 1 1 -1 -1 -1 1 -1	2, 28	2, $\overline{3}$	1	odd
56	7	1 -1 -1 -1 -1 1 1 1 -1 1	0, 30	0, $\overline{1}$	1	odd
55	8	1 -1 1 1 -1 1 -1 -1 -1 1	14, 16	14, $\overline{15}$	-1	odd
54	9	1 -1 -1 -1 1 -1 1 1 1 -1	12, 18	12, $\overline{13}$	-1	odd
53	10	1 -1 1 1 1 -1 1 1 1 -1	10, 20	10, $\overline{11}$	-1	odd
52	11	1 -1 -1 -1 -1 1 -1 -1 -1 1	8, 22	8, $\overline{9}$	-1	odd
51	12	1 -1 1 1 -1 1 1 1 1 -1	6, 24	6, $\overline{7}$	-1	odd
50	13	1 -1 -1 -1 1 -1 -1 -1 -1 1	4, 26	4, $\overline{5}$	-1	odd
49	14	1 -1 1 1 1 -1 -1 -1 -1 1	2, 28	2, $\overline{3}$	-1	odd
48	15	1 -1 -1 -1 -1 1 1 1 1 -1	0, 30	0, $\overline{1}$	-1	odd
16	16	-1 1 1 1 1 -1 -1 -1 -1 1	1, 31	1, $\overline{0}$	-1	odd
17	17	-1 1 -1 -1 -1 1 1 1 1 -1	3, 29	3, $\overline{2}$	-1	odd
18	18	-1 1 1 1 -1 1 1 1 1 -1	5, 27	5, $\overline{4}$	-1	odd
19	19	-1 1 -1 -1 1 -1 -1 -1 -1 1	7, 25	7, $\overline{6}$	-1	odd
20	20	-1 1 1 1 1 -1 1 1 1 -1	9, 23	9, $\overline{8}$	-1	odd
21	21	-1 1 -1 -1 -1 1 -1 -1 -1 1	11, 21	11, $\overline{10}$	-1	odd
22	22	-1 1 1 1 -1 1 -1 -1 -1 1	13, 19	13, $\overline{12}$	-1	odd
23	23	-1 1 -1 -1 1 -1 1 1 1 -1	15, 17	15, $\overline{14}$	-1	odd
24	24	-1 1 1 1 1 -1 -1 -1 1 -1	1, 31	1, $\overline{0}$	1	odd
25	25	-1 1 -1 -1 -1 1 1 1 -1 1	3, 29	3, $\overline{2}$	1	odd
26	26	-1 1 1 1 -1 1 1 1 -1 1	5, 27	5, $\overline{4}$	1	odd
27	27	-1 1 -1 -1 1 -1 -1 -1 1 -1	7, 25	7, $\overline{6}$	1	odd
28	28	-1 1 1 1 1 -1 1 1 -1 1	9, 23	9, $\overline{8}$	1	odd
29	29	-1 1 -1 -1 -1 1 -1 -1 1 -1	11, 21	11, $\overline{10}$	1	odd
30	30	-1 1 1 1 -1 1 -1 -1 1 -1	13, 19	13, $\overline{12}$	1	odd
31	31	-1 1 -1 -1 1 -1 1 1 -1 1	15, 17	15, $\overline{14}$	1	odd

Table 7.2: 64 state MSK trellis transition table

are available, and computation begins on the next packet.

The control unit cycles through the 8 quadruplets of 4 states on the lines *state0, state1, state2, state3*, and each of the five coefficient indices on the *coeff* line. The summation modules calculate the summation defined in equations 7.1 to 7.4 serially. When the 4 I_S and Q_S values are available the *Write Enable* output is asserted and the values are written out in turn to the IsQs RAM module.

7.4 The IsQs dual RAM module

Figure 7.3 shows the IsQs dual RAM module which is written to by the SCG, and read from by the NP during the equalization of the packet data.

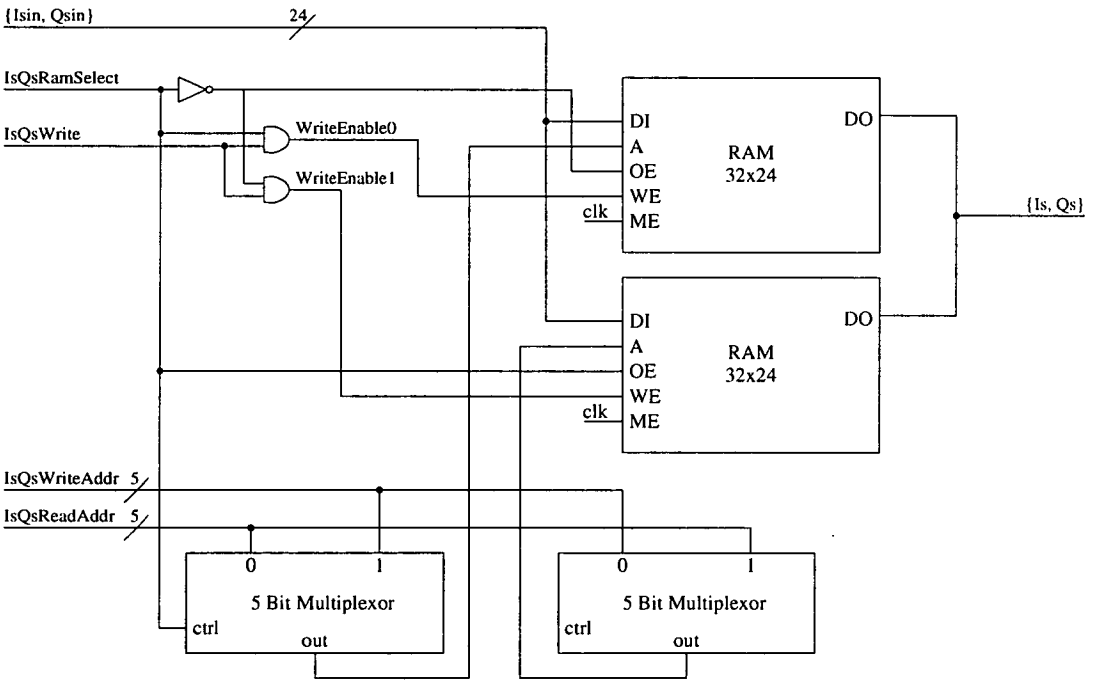


Figure 7.3: The IsQs dual RAM module

The SCG and NP access the RAM at the same time and to ensure that the correct values are available for the NP, two separate RAM banks are used. The RAMs used are from the ES2 design kit [82] megacell compiler and have tri-state outputs.

The control signal *IsQsRamSelect* is generated by the SCG and determines which of the RAM banks is being read and which is being written to. This signal is toggled

when a new packet is received so that the SCG always computes the I_S and Q_S values for the next packet while the NP is decoding the current packet.

7.5 The node processor

The NP shown in Figure 7.4 computes the metric for each of the trellis nodes during each iteration. Table 7.3 explains the control signals which are generated by the NP control unit.

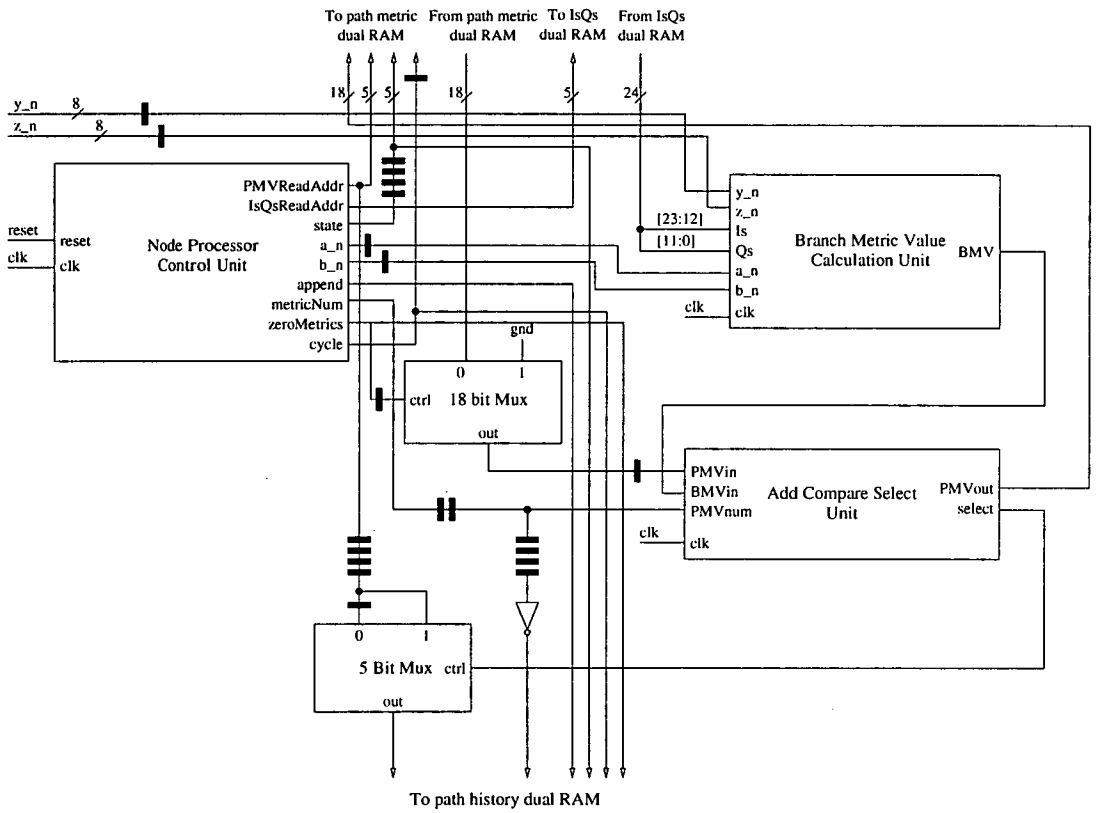


Figure 7.4: The node processor module

The main inputs to the NP are *y_n* and *z_n*, the received data sequences. In addition the module receives inputs from the IsQs RAM module and the PMV RAM module.

The NP cycles through the 32 states pairs and reads the corresponding PMVs from the RAM. The I_S and Q_S values are also read out of RAM and used by the branch metric calculation unit to determine the branch metric values (BMVs).

To ensure that the signals arrive at the correct inputs at the correct time, latches have

Signal Name	Description
state	The current state pair for which the metric is being computed. Cycles from 0 to 31 every 2 clock cycles.
PMVReadAddr	The state in the previous stage which is linked by a branch to this one. Two for each state.
IsQsReadAddr	The number of the I_S and Q_S pairs required (see Table 7.2). Two for each state.
metricNum	0 or 1 indicating the first or second PMV (and I_S Q_S) value.
a_n	Last a in the seqeunce for this state (see Table 7.2).
b_n	Last b in the seqeunce for this state (see Table 7.2).
append	append value for this state (see Table 7.2).
zeroMetric	High for all of the first iteration.
cycle	0 for an even iteration 1 for odd.

Table 7.3: The node processor control signals

been inserted into the circuit. These are indicated by the thick black lines on some of the wires and busses.

The two BMVs and PMVs are added and compared by the ACS unit. The largest of the two sums is then written back into the PMV RAM. In addition, the results of the comparison are passed to the PH RAM module which updates the PH values with the correct append bit.

Because the operation of the NP is pipelined, even though it requires 6 clock cycles to compute each PMV, an average of only 2 clock cycles per state is required to complete all 32 state pairs. However, all of the results of an iteration must be written into memory before the next iteration can commence so an additional 4 cycles must be inserted after the last PMV is computed.

7.6 The path metric value RAM module

Figure 7.5 shows the PMV RAM module. The design of the module is similar to the IsQs RAM module described in Section 7.4, however, as discussed in Section 5.2.5, the metrics need to be normalised to avoid them increasing beyond the resolution of the registers that store them.

The normalisation circuit shown in Figure 7.6 is based on the classic normalisation

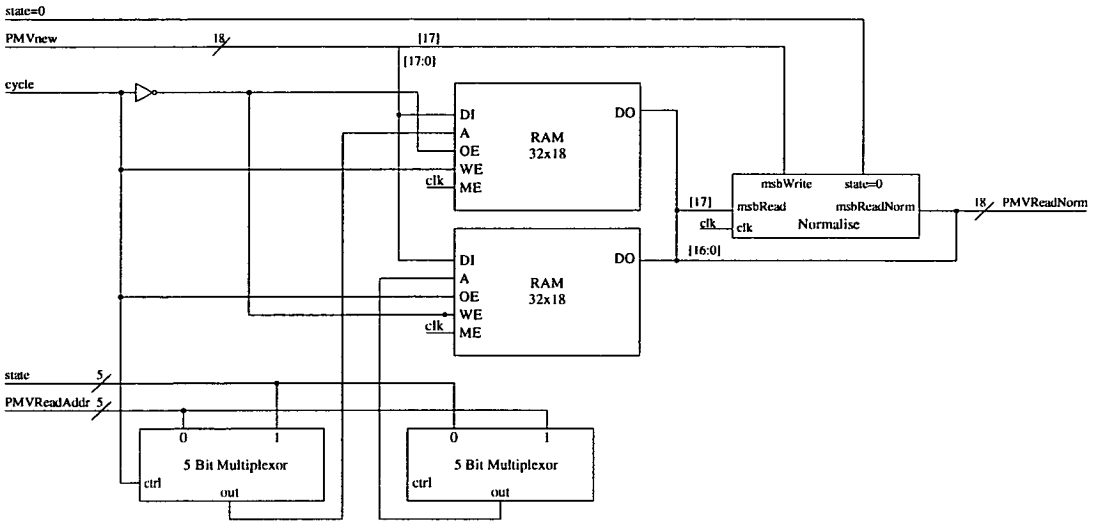


Figure 7.5: The path metric dual RAM module

technique used in the Viterbi algorithm, when the MSBs of all the metrics are set, then reset them [22].

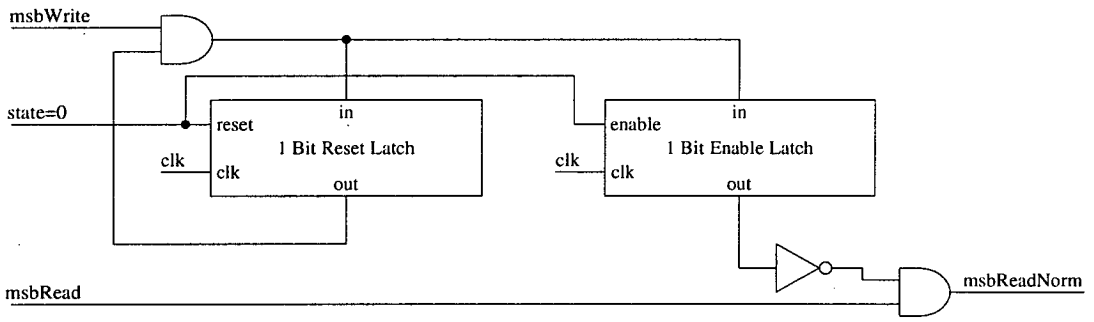


Figure 7.6: The normalisation module

The normalisation module is a serial *and* gate. When the PMVs are being written into memory, the MSBs are all *and*ed together. At the end of the iteration the result is latched and used to conditionally reset the MSBs of the PMVs as they are read from the memory during the next iteration.

7.7 The BMV calculation unit

The BMV calculation unit is shown in Figure 7.7. It can be seen that the circuit is a direct implementation of the incremental metric equation:

$$\lambda_{p \to q} = a_n(y_n - I_S) + b_n(z_n - Q_S) \tag{7.5}$$

The only difference is the inclusion of the two negation units. As was discussed in

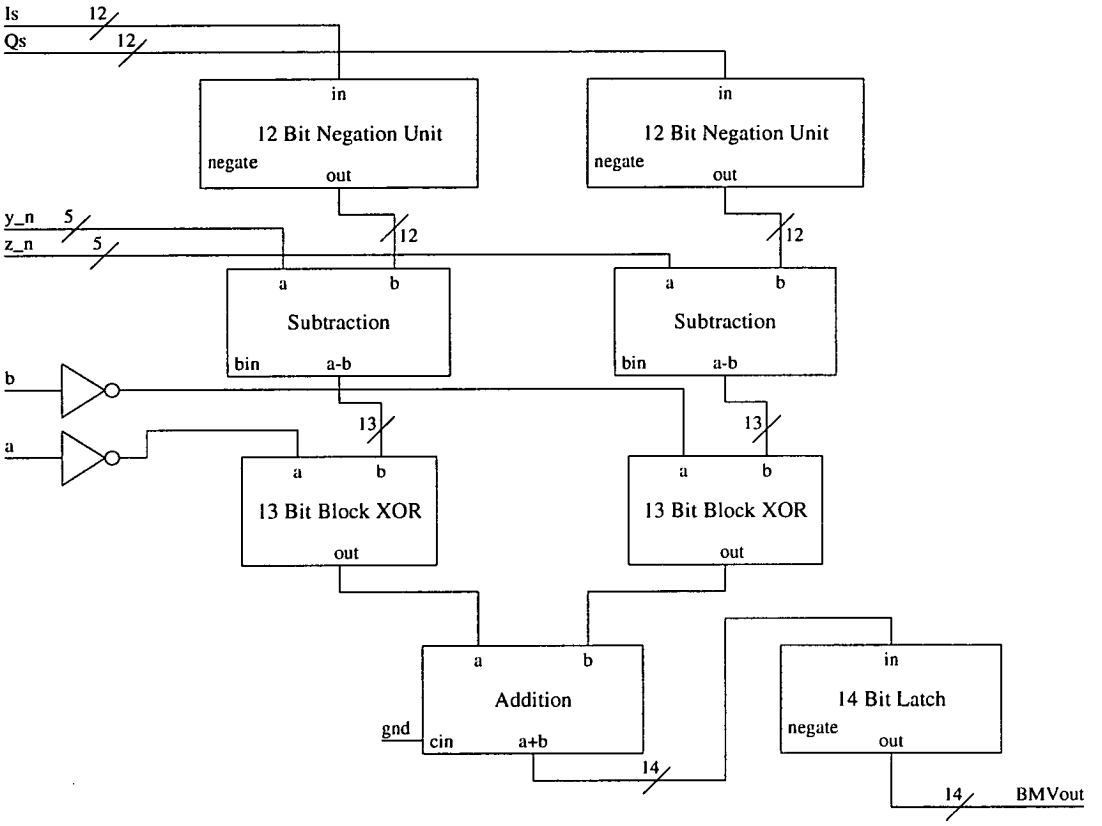


Figure 7.7: The branch metric value calculation unit

Section 7.3, I_S and Q_S for $S = 32...63$ have not been calculated because they are the negative of I_S and Q_S for $S = 0..31$, this means that for the second of the two I_S and Q_S values for each state, the value should be negated. This is achieved by a conditional negation unit with its input connected to the *metricNum* control signal from the NP.

7.8 The add-compare-select unit

Figure 7.8 shows a schematic diagram for the ACS unit. The ACS units compares the sums of 2 PMVs and 2 BMVs, since both PMVs are stored in the same RAM they cannot be accessed simultaneously. The ACS unit performs the two additions and latches the results of the first one for comparison in the next clock cycle. It can be seen that the output of the ACS unit is valid on alternate clock cycles.

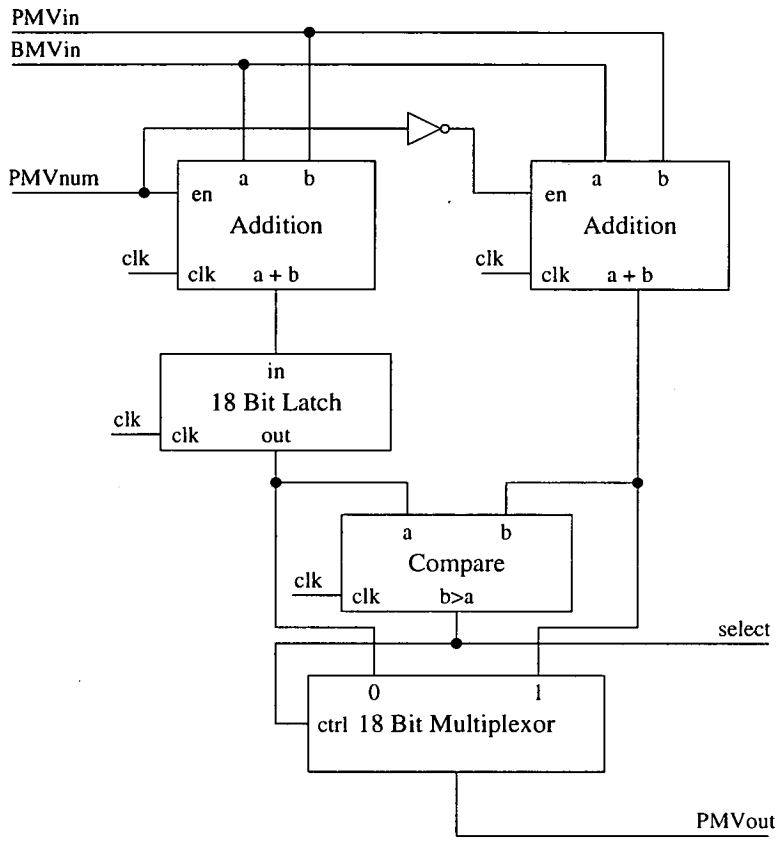


Figure 7.8: The serial add-compare-select unit

7.9 The path history RAM

Figure 7.9 shows the schematic diagram for the PH RAM module. The PH RAM module also controls the updating of the PHs with the correct append bit.

When the NP selects the correct path based on the PMVs, the numbers of the previous state and the current state in the path are passed to the PH RAM module on the inputs

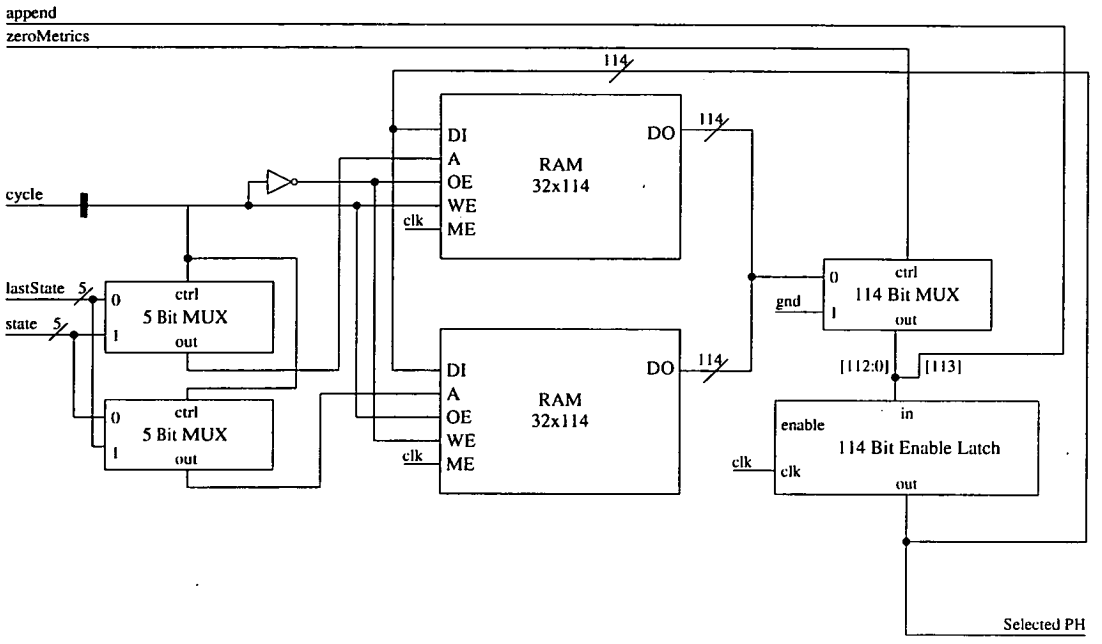


Figure 7.9: The path history dual RAM module

lastState and *state* respectively. The PH RAM module then reads the PH for *lastState* out of the read RAM, appends it with the *append* bit, and writes the result into the PH for *state* in the write RAM. The read and write RAMs are selected by the input *cycle* which is the same signal which is used to determine which PMV RAM is being written to and read from.

The selected PH forms the output of the PH RAM block. This is used by the output determination unit which is described in the next section.

7.10 The output determination unit

Figure 7.10 shows a schematic diagram of the output determination unit.

The unit examines the metrics as they are written into the PMV RAM module (incoming metrics appear on the input *PMVnew*), the *enable* input signifies that a new metric is available. At the start of each iteration (i.e. when the state number is equal to zero) the current maximum metric is reset. When a new PMV is available, this is compared with the current maximum stored in the latch. If the new metric is larger than the current maximum, the current maximum is set to the value of the new PMV.

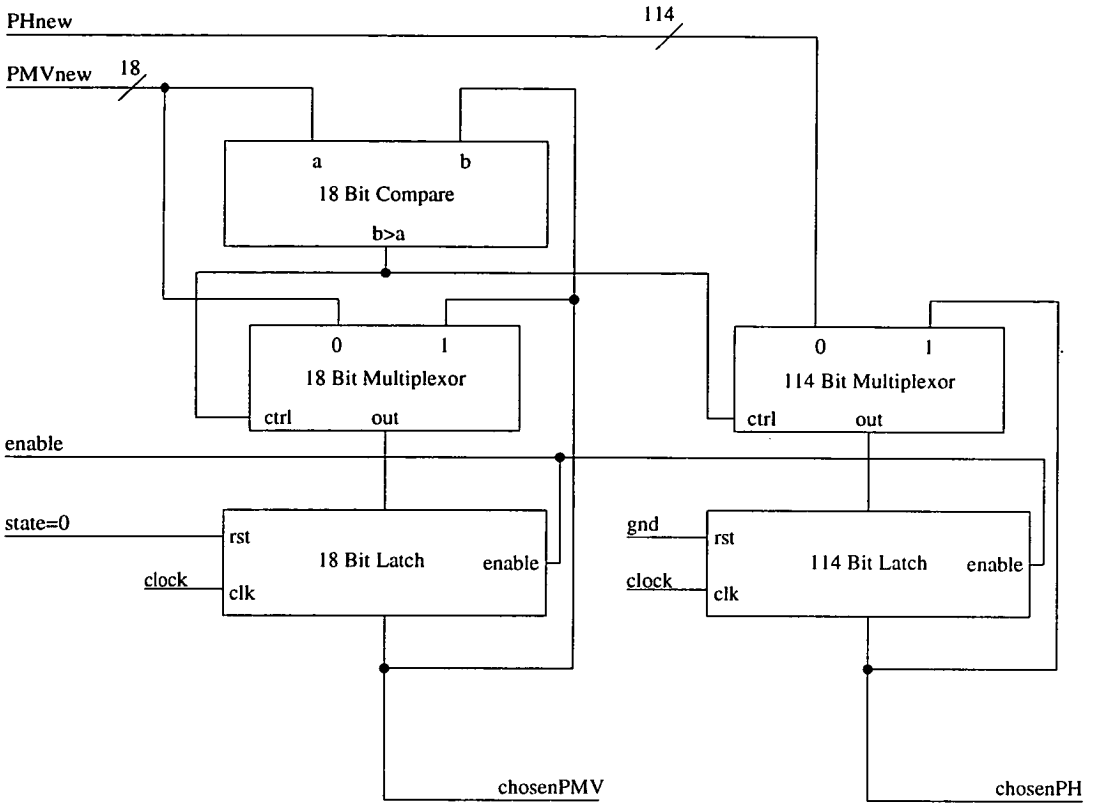


Figure 7.10: The output determination unit

The unit also takes the new PH values from the PH RAM module as input. When a new metric is chosen to be the maximum, the corresponding PH is also stored.

When the packet has been fully decoded, the *SelectedPH* output is equal to the equalized packet.

7.11 Implementation

As discussed in the introduction to this chapter it was decided to implement the serial VE design using the Cadence automatic place and route tools and the ES2 $0.7\mu m$ CMOS design kit.

7.11.1 Development using Verilog HDL

It was decided to use Verilog HDL because the language allows the use of behavioural and structural module descriptions in a straightforward manner.

Initially a Verilog description for the design was produced with the low-level modules described with a behavioural definition and the more high level modules defined with structural definitions.

7.11.2 The Synergy logic synthesis tool

Each of the behavioural modules were imported into the Cadence Synergy logic synthesis tool which was used to produce structural implementations using the ES2 library. This structural description for each module was exported as a Verilog netlist and used to replace the behavioral modules in the original Verilog description file.

7.11.3 Testing and Verification

When the behavioural definition of the design was produced, sets of test vectors were created to verify the design. Synthesised version of each module were used to replace the behavioural versions and the design was retested. This made it straightforward to identify errors which where introduced as a result of the synthesis step.

7.11.4 Layout

The equalizer module was place and routed using the Cadence Cell Ensemble automatic place and route tools. Figure 7.11 shows the layout for the serial VE module. The area of the module is $4392 \times 3233 \mu m$ which makes the design suitable for a hand-held receiver.

In Figure 7.11 the RAM modules are clearly visible at the top and bottom of the circuit. The large RAMs are for the path history storage and the smaller RAMs to the right are the path metric, and state constant RAMs.

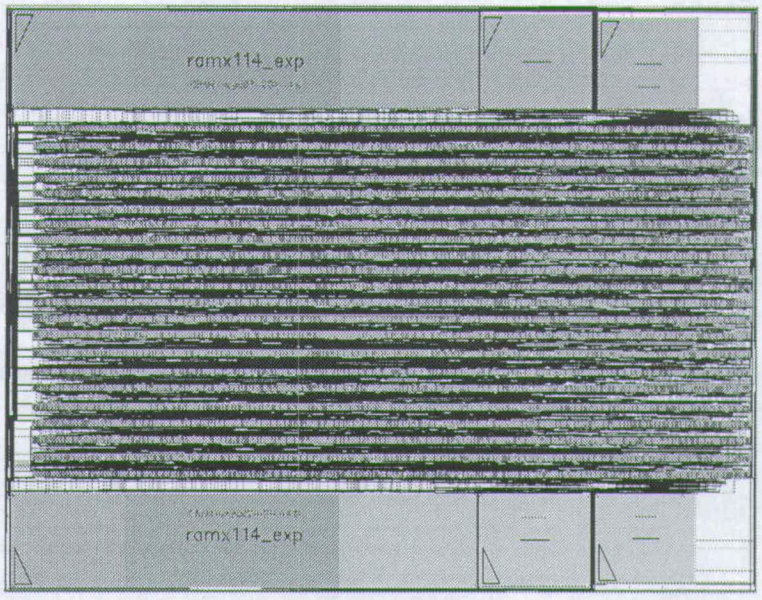


Figure 7.11: The Viterbi Equalizer layout

7.12 The VE module specifications

Table 7.4 describes the input and output ports of the VE module. Simulation showed that the maximum clock speed for the module was $20MHz$ which is well within the requirements for GSM.

Port Name	Input/Output	Description
clk	input	system clock, max. freq $20MHz$
reset	input	system reset
nextPacket	input	indicates the start of a new packet
y_n[0:7]	input	received in-phase signal statistic
z_n[0:7]	input	received quad-phase signal statistic
coeffIn[7:0]	input	coefficient value
coeffSelect	input	selects either χ or ζ coefficients
coeffNum[2:0]	input	index of current coefficient
latchCoeff	input	new coefficient value is ready
dataOut	output	equalized data stream
dataEnd	output	indicates that decoding has finished on the current packet

Table 7.4: The ports of the serial VE module

7.13 Conclusions

This chapter has presented the design of a serial Viterbi equalizer. The major reason for producing this design was because the previous fully parallel design was too large. In addition, while a high-speed design would be useful for a GSM base station (where different transmissions could be multiplexed using a single Viterbi device), a hand-held receiver requires only a very slow bit-rate. The most important requirement is that the design should be small. The lower the transistor count, the lower the power consumption.

The main reason in which the design presented in this chapter differs from the parallel one, is because there is only one node processor. This processor performs the calculations for each of the trellis nodes, in a serial manner. This means that the data cannot be stored locally at the nodes, some sort of memory management system is required. In addition, because the algorithm is computed serially (which is obviously slower than a parallel implementation) the design is heavily pipelined to maintain an acceptable throughput. This makes the design more complicated, although the final implementation is significantly smaller (in fact, the size of the final design is similar to the smaller of the two Viterbi decoders presented in Chapter 4).

The heavily pipelined design features dual RAM blocks where results are written into one section, for reading out in the next cycle. The state constant generator unit exploits the similarities between summations in a similar manner to the circuit described in Chapter 5. In addition, the fact that only half of the trellis states are valid at each stage is exploited in the dual RAM structure.

The pipelining of the design means that the processing of the data has a high throughput. On average only 2 clock cycles are required per state to evaluate each of the 32 states. Because of this pipelining, and the serial manner in which the metric are evaluated, the path history calculation unit is substantially more straightforward that the parallel design. The unit has been simplified to a serial comparison unit, which requires much less silicon area than the parallel equivalent.

The design has been implemented and has been shown to be very small at about $14mm^2$, with a bit- rate of $312.5Kbps$, well above the GSM standard requirement of $22.8Kbps$.

Chapter 8

Conclusions

The thesis has examined implementations of the Viterbi algorithm in VLSI, from that investigation a number of secondary investigations have emerged. They have been concerned with redundant number systems, and complementary pass-transistor logic.

Section 8.1 will provide a summary of the content of the thesis, detailing the work that has been presented. Section 8.2 will discuss the main conclusions that can be drawn from the research.

8.1 Thesis Review

8.1.1 Redundant Number Systems

In Chapter 3 redundant number systems for high-speed arithmetic were reviewed. Specific attention was paid to signed binary number representation (SBNR) [4]. Major arithmetic functions were discussed, and SBNR implementations were compared in some detail with their twos-complement equivalents. The investigation showed that SBNR can be a used to perform addition and subtraction operations at very high speed. It was also shown that the delay of such circuits is independent of the word-length of the operands. One of the main drawbacks of SBNR for VLSI circuit design is that the conversion back into twos-complement is relatively slow, and *is* dependent on the word-length.

During this investigation, a novel static-logic binary-tree based twos-complement VLSI adder design was proposed which used SBNR concepts to produce a purely twos-complement adder with a delay of $\lceil \log_2(N) \rceil + 2$. The design was as fast as previously reported designs.

8.1.2 Viterbi Decoding

In Chapter 4, the design of a Viterbi decoder using SBNR was described. The design uses redundant number systems as an internal representation for the metrics. Because the representation is purely internal, the values never need to be converted into twos-complement representation.

It was shown that the design required an estimated 45% more area than an equivalent implementation using twos-complement arithmetic, although it demonstrated an increase in bit rate of 14% over conventional methods.

The decoder design was compared with a number of designs from the literature in Section 4.7. The implementation compared favourably with similar designs, in most cases it was smaller, and faster.

8.1.3 Complementary Pass-Transistor Logic

Chapter 6 described the use of complementary pass-transistor logic (CPL) to produce low-power VLSI circuits. The technique of CPL had been proposed by Yano *et al* as an alternative to CMOS circuit design [7].

CPL has been investigated thoroughly in this work. A large number of logic functions have been implemented using both CMOS and CPL and their speed, area, and power consumption have been compared. A standard cell library of CPL functions has also been developed using the Cadence design tools.

8.1.4 Viterbi Equalization

In Chapter 5, the design of a low-power implementation of a Viterbi equalizer suitable for a GSM receiver was described. The implementation of this design was not completed because the area requirement for the routing was too large. For the Viterbi trellis, the placed and unrouted area was at $7656\mu m \times 3613\mu m$, however, after routing this increased to $10186\mu m \times 24354\mu m$, roughly $5cm^2$, which is clearly not suitable for a VLSI implementation.

Using the lessons from this implementation. A serial Viterbi equalizer was described

in Chapter 7. The design presented contained only one node processor. This processor performs the calculations for each of the trellis nodes, in a serial manner. This makes the design more complicated, although the final implementation was significantly smaller than the parallel circuit. The area of the entire module was $4392 \times 3233 \mu m$, very much suitable for a handheld receiver. The design is very small, and meets the GSM specifications.

8.2 Final Conclusions

This section will discuss the main achievements of the research, and discuss some possible further work.

8.2.1 Achievements

8.2.1.1 Viterbi Implementation

The original aim of the thesis was to focus on VLSI implementations of the Viterbi algorithm. In producing the three designs, the following conclusions can be reached which are important to consider when produced a VLSI Viterbi decoder or equalizer.

1. The throughput of the algorithm is primarily influenced by the delay of the add-compare-select unit. The algorithm is iterative, and each iteration cannot proceed until the results of the previous iteration are known. To maximise the throughput of the design, a high-speed add-compare-select unit is required.

2. The path metric values are internal to the decoder/equalizer so their representation need not be twos-complement. The choice of representation of the path metric is important and should be chosen based on the speed, power, and area requirements of design. It can be concluded from this work that for high speed designs carry-save arithmetic is preferable while for non speed critical designs, where low power and low area are more important twos-complement representation is a better choice.

3. The Viterbi algorithms complexity increases exponentially as the length of convolutional code, or the required intersymbol interference is increased. Parallel

implementations are practical for low ISI (as the design described in Chapter 4 showed), but for larger values of ISI, for non-speed critical applications, and for small devices, a serial implementation (such as the one described in Chapter 7) is desirable.

8.2.1.2 A fast twos-complement adder

During the investigation into redundant number systems, a fast adder design by Srinivas and Parhi was investigated [5]. The adder circuit used SBNR as an internal representation to compute the sum of two operands, and then converted the result back into twos-complement representation. Detailed analysis of this design showed that the use of redundant number representation was pointless, and it disguised the true nature of the speed increase. Using this knowledge a fast adder design was developed which was purely twos-complement, and was faster and smaller that the original Srinivas and Parhi design. In fact, this design has a lower gate count, and smaller critical path, than existing designs.

This achievement was an important development in the investigation into redundant number systems. It showed that using SBNR arbitrarily is not a sensible method of speeding up arithmetic computations in digital circuits. As with most things, care must be taken to ensure that the technique is suitable for the application. In many cases (particularly parallel designs) carry-save arithmetic (which is, itself a redundant number representation) is probably more useful than SBNR. However, it should be noted that redundant number systems have been used in the literature to achieve large throughput rates in digit-serial, and digit-skewed designs.

8.2.1.3 The failings of CPL

Another important achievement in the research was the results obtained from the detailed investigation into complementary pass-transistor logic. In the original paper on the subject, Yano *et al* had presented the technique as an alternative to CMOS logic design [7]. They claimed that circuits designed using CPL had lower power consumption, and were often faster than the CMOS equivalents.

In this investigation, many logic functions were implemented, using the same layout tools and technology, in both CMOS and CPL. The conclusions were that in most

cases CPL is actually slower, and higher in power consumption than CMOS. The exceptions to this were more complex functions, such as **xor** gates and full-adder cells. Even in these cases the reduction in power, and the increase in speed was quite small, significantly lower than the original claims made by Yano *et al.*

In addition, the implementation of the Viterbi equalizer using CPL raised another problem with the logic style. The requirement that cells need both the original signal, and its complement, means that the routing between cells is doubled. In practice this will more than double the area requirement for the routing, as the routing becomes more complicated. This was the main reason for the massive area requirement for the original Viterbi equalizer design.

It should be noted than Yano himself independently came to the same conclusions. In a follow-up paper, which was published during the latter stages of this research, Yano *et al* presented an extension to CPL that we have called single-output pass transistor logic (SO-PL) [8]. These SO-PL cells use local inverters to generate complementary signals, only when they are required. This reduces the routing requirements. Yano also doesn't suggest replacing common logic functions with SO-PL equivalents. Instead, he presents a methodology for logic circuit design based on a very specific set of primitive cells that are optimised for SO-PL.

8.2.2 Further Work

Further work could include the following:

- An in-depth comparison of redundant number systems and carry-save arithmetic. The work described in Chapter 3 showed that SBNR can offer significant improvements in speed performance for many arithmetic circuits. It was also concluded that carry-save arithmetic could offer greater speed increases than SBNR. Valuable further work would be to examine important applications of SBNR and compare the speed performance of carry-save implementations of the same designs. In addition, the fast-adder design showed that, at least in one case, redundant number systems can be considered redundant and only serve to disguise the underlying concepts. Again, important designs using SBNR should be examined and this finding investigated further.

- Implementation of the fast-adder design.

  The twos-complement adder design presented in Chapter 3 was as fast as previously reported designs and had a smaller gate count. Implementation of this design and a detailed investigation of its performance would allow a better comparison with other adder implementations.

- Implementation of the Viterbi decoder using carry-save arithmetic, and tailoring it for use in a GSM system.

  The Viterbi decoder implementation presented in Chapter 4 was implemented using SBNR, subsequent investigations of carry-save arithmetic described in Chapter 3 showed that a Viterbi decoder design using carry-save arithmetic could be smaller and faster that the SBNR design. This should be investigated. In addition, the decoder design was suitable for decoding the convolutional codes used in the GSM telephone system, but does not contain the necessary interface for placing the decoder within a GSM receiver structure. This should be designed and implemented to produce small Viterbi decoder module for a GSM receiver.

- Implementation of the Viterbi equalizer design using SO-PL.

  The serial Viterbi equalizer design described in Chapter 7 was designed for implementation with SO-PL. Due to time constraints a standard library of CMOS cells was used for the implementation. A SO-PL implementation would allow a performance evaluation of the complete design methodology.

References

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptoically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT13, pp. 260–269, Apr 1967.

[2] A. S. Acampora, "Analysis of maximum-likelihood sequence estimation performance for quadrature amplitude modulation," *The Bell System Technical Journal*, vol. 60, pp. 865–885, July-August 1981.

[3] COST 207 Management Committee, "Digital Land Mobile Radio Communications - Final Report," 1988.

[4] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 389-400, Sep. 1961.

[5] H. R. Srinivas and K. K. Parhi, "A Fast VLSI Adder Archictecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 761-767, May 1992.

[6] N. Takagi, H. Yasuura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Transactions on computers*, vol. C-34, pp. 789–796, Sep 1985.

[7] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu, "A 3.8ns cmos 16x16-b multiplier using complementary pass-transistor logic," *IEEE Journal of Solid-State Circuits*, vol. JSSC-25, pp. 388–395, Apr 1990.

[8] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *IEEE Journal of Solid State Circuits*, vol. 31, pp. 793–803, June 1996.

[9] G. D. Forney, "The Viterbi Algorithm," *IEEE Proceedings*, vol. 61, pp. 268–278, Mar 1973.

[10] G. D. Forney, "Maximum-Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 363–378, May 1972.

[11] J. A. Heller and I. M. Jacobs, "Viterbi decoding for satellite and space communication," *IEEE Transactions on Communication Technology*, vol. COM-19, pp. 832–848, Oct 1971.

[12] S. A. Morley, "Forward error correction applied to intelsat IDR carries," *International Journal of Satellite Communications*, vol. 6, pp. 445–454, 1988.

[13] S. Haykin, *Digital Communications*. Wiley, 1988.

[14] K. J. Larson, "Short Convolutional Codes With Maximal Free Distance for Rates 1/2, 1/3, 1/4," *IEEE Transactions on Information Theory*, vol. IT-19, pp. 371–372, May 1973.

[15] A. M. Michelson and A. H. Levesque, *Error-Control Techniques for Digital Communication*. John Wiley & Sons, 1985.

[16] G. Ungerboeck, "Adaptive Maximum-Likelihood Receiver for Carrier Modulated Data-Transmission Systems," *IEEE Transactions on Communications*, vol. COM-22, pp. 624–636, May 1974.

[17] J. B. Anderson, T. Aulin, and C. Sundberg, *Digital phase modulation*. Plenum Press, 1986.

[18] P. J. McLane, "The Viterbi receiver for correlative encoded MSK signals," *IEEE Transactions on Communcations*, vol. COM-31, pp. 290–295, Feb 1983.

[19] P. G. Gulak and E. Shwedyk, "VLSI structures for Viterbi receivers part ii - encoded MSK modulation," *IEEE Journal on Selected Areas in Communications*, vol. SAC4, pp. 155–159, Jan 1986.

[20] I. J. Wassell and R. Steele, ""Wideband Systems"," in *Mobile Radio Communications* (R. Steele, ed.), IEEE Press, 1992.

[21] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Transactions of Communication Technology*, vol. COM-19, pp. 751–772, Oct 1971.

[22] M. A. Bree, "A Bit-Serial Viterbi Processor," Master's thesis, University of Saskatchewan, Dec 1988.

[23] L. Hanzo and J. Stefanov, "The pan-european digital cellular mobile radio system - known as GSM," in *Mobile Radio Communications* (R. Steele, ed.), IEEE Press, 1992.

[24] F. Halsall, *Data communications, computer networks and open systems.* Addison-Wesley, 4th edition ed., 1995.

[25] P. Vary, "Implementation aspects of the pan-European digital mobile radio system," in *COMPEURO '89 - 3rd Annual European Computer Conference*, vol. 4, pp. 17–22, IEEE, May 1989.

[26] G. Benelli, A. Garzelli, and F. Salvi, "Simplified Viterbi processers for the GSM pan-European cellular communication system," *IEEE Transactions on Vehichular Technology*, vol. 43, pp. 870–878, Nov 1994.

[27] R. D'Avella, L. Moreno, and M. Sant'Agostino, "An Adaptive MLSE Receiver for TDMA Digital Mobile Radio," *IEEE Journal on Selected Areas in Communications*, vol. 7, pp. 122–129, Jan 1989.

[28] G. D'Aria, F. Muratore, and V. Palestini, "Simulation and Performance of the Pan-European Land Mobile Radio System," *IEEE Transcations on Vehicular Technology*, vol. 41, pp. 177–189, May 1992.

[29] G. D'aria, G. Stola, and L. Zingarelli, "Modelling and Simulation of the Propagation Characteristics on the 900 MHz Narrowband-TDMA CEPT/GSM Mobile Radio," in *IEEE Vehicular Technology Conference*, vol. 2, pp. 631–639, 1989.

[30] H. Busschaert, P. P. Reusens, L. Dartois, and L. Desperben, "A power efficient channel coder/decoder chip for GSM terminals," in *IEEE 1991 Custom Integrated Circuits Conference*, 1991.

[31] W. Kaiming and W. Wuiling, "Viterbi hardware implementation for GSM," in *The 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering (TENCON '93)*, pp. 120–122, IEEE, Oct 1993.

[32] J. G. Proakis, "Adaptive equalization for TDMA digital mobile radio," *IEEE Transactions on Vehicular Technology*, vol. 40, pp. 333–341, May 1991.

[33] E. Dahlman, "New Adaptive Viterbi Detector for Fast-Fading Mobile-Radio Channels," *Electronic Letters*, vol. 26, pp. 1572–1573, Sep 1990.

[34] G. Benelli, A. Fioravanti, A. Garzelli, and P. Matteini, "Some Digital Receivers for the GSM Pan-European Cellular Communication System," *IEE Proceedings on Communications*, vol. 141, pp. 168–176, Jun 1994.

[35] N. Takagi and S. Yajima, "Module multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem," *IEEE Transactions on computers*, vol. 41, pp. 887–891, Jul 1992.

[36] S. C. Knowles, J. G. McWhirter, R. F. Woods, and J. V. McCanny, "Bit-level systolic architectures for high performance IIR filtering," *Journal of VLSI Signal Processing*, vol. 1, pp. 9-24, 1989.

[37] H. Makino, Y. Nakase, H. Suzuki, H. Marinaka, H. Shinohara, and K. Mashiko, "An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 773–783, Jun 1996.

[38] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 1877–85, Dec 1992.

[39] Z. Menghui and A. Albicki, "Low power and high speed multiplication design through mixed number representations," in *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pp. 566–570, IEEE, Oct 1995.

[40] S. Yen, C. Laih, C. Chen, and J. Lee, "An efficient redundant-binary number to binary number converter," *IEEE Journal of Solid State Circuits*, vol. 27, no. 1, pp. 109–112, 1992.

[41] A. Herrfeld and S. Hentschke, "Conversion of redundant binary into two's complement representations," *Electronic Letters*, vol. 31, no. 14, pp. 1132–1133, 1995.

[42] M. D. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers*, vol. C-36, no. 7, pp. 895–897, 1987.

[43] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design - A systems perspective.* Addison-Wesley, second edition ed., 1993.

[44] T. Lynch and E. E. S. Jr, "A spanning tree carry lookahead adder," *IEEE Transactions on Computers*, vol. 41, pp. 388–395, Apr 1992.

[45] V. Kantabutra, "A recursive carry-lookahead/carry-select hybrid adder," *IEEE Transactions on Computers*, vol. 42, pp. 1495–1499, Dec 1993.

[46] K. Suzuki, M. Yasashima, T. Nakayama, M. Izumikawa, M. Nomura, H. Igura, H. Heiuchi, J. Goto, T. Inoue, Y. Koseki, H. Abiko, K. Okabe, A. Ono, Y. Yano, and H. Yamada, "A 500mhz, 32 bit, 0.4μmm cmos risc processor," *IEEE Journal of Solid-State Circuits*, vol. JSSC-29, pp. 1464–1473, Dec 1994.

[47] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," *IEEE Transactions on Computers*, vol. C-24, pp. 1014–1015, 1975.

[48] H. Sam and A. Gupta, "A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations," *IEEE Transactions on Computers*, vol. 39, pp. 1006–1015, Aug 1990.

[49] J. E. Robertson, "A new class of digital divison methods," *IRE Transactions on Electronic Computing*, vol. EC-7, pp. 218–222, 1958.

[50] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P. G. Jespers, "A new carry-free division algorithm and its application to a single chip 1024-b RSA processor," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 748–755, Jun 1990.

[51] J. H. P. Zurawski and J. B. Gosling, "Design of high-speed digital divider units," *IEEE Transactions on Computers*, vol. C-30, pp. 691–699, Sep 1981.

[52] M. Yan and J. V. McCanny, "A bit-level systolic architecture for implementing a VQ tree search," *Journal of VLSI Signal Processing*, vol. Vol. 2, pp. 146–158, 1990.

[53] M. Yan and J. V. McCanny, "Systolic inner product arrays with automatic word rounding," *Journal of VLSI Signal Processing*, vol. Vol. 4, pp. 227–242, 1992.

[54] M. A. Bree, D. E. Dodds, R. J. Bolton, S. Kumar, and B. L. F. Daku, "A modular bit-serial architecture for large-constraint-length Viterbi decoding," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 184–190, Feb 1992.

[55] B. Parhami, "Voting networks," *IEEE Transactions on Reliability*, vol. 40, pp. 380–394, Aug 1991.

[56] K. G. Shin and J. W. Dolter, "Alternative majority-voting methods for real-time computing systems," *IEEE Transactions on Reliability*, vol. 38, pp. 58–64, Apr 1989.

[57] G. Fettweis, H. Dawid, and H. Meyr, "Minimized method Viterbi decoding: 600 Mbit/s per chip," in *GLOBECOM '90: IEEE Global Telecommunications Conference and Exhibition*, pp. 1712–16, 1990.

[58] G. Edwards, "A 45-Mbit/sec. VLSI Viterbi decoder for digital video applications," in *Conference Proceedings of the National Telesystems Conference 1993*, pp. 127–39, 1993.

[59] G. Feygin and P. G. Gulak, "Arcitectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Transactions on Communications*, vol. 41, pp. 425–9, Mar 1993.

[60] S. P. Miller, N. Becker, and P. N. Johnson, "Custom ASIC development for high-speed Viterbi decoding," in *MILCOM 90: IEEE Military Communications Conference*, vol. 1, pp. 239–43, 1990.

[61] G. Fettweis and H. Meyr, "A 100MBit/s Viterbi decoder chip: novel architecture and its realization," in *IEEE International Conference on Communications*, vol. 2, pp. 463–467, 1990.

[62] G. Fettweis and H. Meyr, "A systolic array Viterbi processor for high data rates," in *Internation Conference on Systolic Arrays*, pp. 195–204, 1989.

[63] G. Fettweiss and H. Meyr, "Feedforward architectures for parallel Viterbi decoding," *Journal of VLSI Signal Processing*, vol. 3, pp. 105–119, 1991.

[64] G. Fettweiss and H. Meyr, "High-speed parallel Viterbi decoding: Algorithm and VLSI-architecture," *IEEE Communications Magazine*, pp. 46–55, May 1991.

[65] K. K. Parhi, "High-speed VLSI architectures for Huffman and Viterbi decoders," *IEEE Transactions on Circuits and Systems-II: Analogue and Digital Signal Processing*, vol. 39, pp. 385–391, Jun 1992.

[66] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," in *IEEE International Solid-State Circuits Conference*, pp. 70–71, 1992.

[67] J. M. Dobson, G. M. Blair, and B. Mulgrew, "Viterbi equalization: low-power VLSI module design," in *Irish DSP and Control Conference 1996*, pp. 253–260, 1996.

[68] E. D. Re, G. Benelli, G. Castellini, R. Fantacci, L. Pieruccu, and L. Pogliani, "Design of a Digital MLSE Receiver for Mobile Radio Communications," in *IEEE Globecom '88*, vol. 2, pp. 1469–1473, 1988.

[69] N. S. Hoult, C. A. Dace, and A. P. Cheer, "Implementation of an equaliser for the GSM system," in *Fifth International Conference on Radio Receivers and Associated Systems*, pp. 182–186, IEE, Jul 1990.

[70] A. Spielberg, J. Stahl, and T. Pagden, "ASIC Receiver for the Pan European Digital Cellular Telephone (GSM)," in *Vehicular Technology Society 42nd VTS Conference*, vol. 2, pp. 919–922, IEEE, 1992.

[71] A. Bellaouar and M. I. Elmasry, *Low-power digital VLSI design*. Kluwer Academic Publishers, 1995.

[72] G. D'Aria, R. Piermarini, and V. Zingarelli, "Fast adaptive equalizers for narrow-band TDMA mobile radio," *IEEE Transcations on Vehicular Technology*, vol. 40, pp. 177–189, May 1991.

[73] L. Moreno and R. D'Avella, "Maximum Likelihood Adaptive Techniques in the Digital Mobile Radio Environment," in *Int. Conf. in the Digital Land Mobile Radio Commum.*, pp. 227–232, June 1987.

[74] E. D. Re, G. Castellini, L. Pierucci, and F. Conti, "A Within-Burst Adaptive MLSI Receiver for Mobile TDMA Cellular Systems," in *IEEE ICASSP-92*, vol. 4, pp. 493–496, 1992.

[75] G. D'Aria and V. Zingarelli, "Results on Fast-Kalman and Viterbi Adaptive Equalizers for Mobile Radio with CEPT/GSM System Characteristics," in *IEEE Globecom '88*, vol. 2, pp. 815–819, 1988.

[76] A. P. Hekstra, "Alternative to metric rescaling in Viterbi decoders," *IEEE Transactions on Communications*, vol. 37, Nov 1989.

[77] M. Shamanna, K. Cameron, and S. R. Whitaker, "Multiple-inpur, multiple-output pass transitor logic," *International Journal of Electronics*, vol. 79, no. 1, pp. 33–45, 1995.

[78] G. M. Blair, "Designing low-power digital CMOS," *IEE Journal of Electronics and Communication Engineering*, vol. 6, pp. 229–236, Oct 1994.

[79] U. Ko, T. Balsara, and W. Lee, "Low-power design techniques for high-performance CMOS adders," *IEEE Transactions on VLSI Systems*, vol. 3, pp. 327–333, June 1995.

[80] M. Suzuki, K. Shinbo, T. Yamanaka, A. Sasaki, and Y. Nakagome, "A 1.5-ns 32-b CMOS ALU in double pass-transistor logic," *IEEE Jounral of Solid-State Circuits*, vol. 28, pp. 1145–1151, 1993.

[81] A. Parameswar, H. Hara, and T. Sakurai, "A high-speed, low-power, swing restored pass-transistor logic based multiply and accumulate circuit for multimedia applications," in *IEEE Custom Integrated Circuits Conference*, pp. 278–281, May 1994.

[82] H. Samiy and M. Willoughby, *The ES2 $0.7\mu m/1\mu m$ CMOS library design kit on Cadence DFWII software.* Rutherford Appleton Laboratory, 1995.

[83] D. E. Thomas and P. Moorby, *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 1991.

Appendix A

Papers

This appendix contains reprints papers that have been produce during the research project. "A Fast Twos-Complement VLSI Adder Design" by J. M. Dobson and G. M. Blair was published in *IEE Electronics Letters*, volume 31, No 20, pp1721-1722. "Viterbi Equalization: Low-Power VLSI Module Design" by J. M. Dobson, G. M. Blair and B. Mulgrew appeared in the proceedings of IDSPCC'96, pp253-260. "A Low Area Serial Viterbi Equalizer Implementation" by J. M. Dobson and G. M. Blair has been submitted to ESSCIRC'97.

A Fast Twos-Complement VLSI Adder Design

Jonathan M. Dobson and Gerard M. Blair
Dept. of Electrical Engineering,
University of Edinburgh, Mayfield Rd., Edinburgh, EH9 3JL
031 650 5655 (Fax 650-6554) jmd@ee.ed.ac.uk

This paper appeared in Electronic Letters, vol. 31, no. 20, pp 1721-1722, Sep 1995.

Abstract

his paper reexamines the design by Srinivas and rhi[1] which used redundant-number adders for fast os-complement addition. The underlying mecha-sm is revealed and improvements are presented hich lead to a static-logic binary-tree carry genera-r to support high-speed adder implementations with delay of $\lceil \log_2(N) \rceil + 2$ *gates.*

Introduction

1992, Srinivas and Parhi[1] presented a design for a os-complement adder which uses Signed Binary umber Representation[2] (SBNR). Since nventional binary is a subset of SBNR, two n-bit os-complement numbers $\{a_{N-1} \cdots a_0\}$ and $_{N-1} \cdots b_0\}$ can be summed as SBNR numbers ithout carry propagation) to yield an N-digit SBNR mber. The Srinivas and Parhi architecture consisted such an addition stage and followed by conversion the SBNR sum back into twos-complement format. e authors claimed that this provided the fastest chitecture for the addition of two twos-complement mbers.

owever, the initial SBNR addition block of [1] can replaced by $xor(a_i, b_i)$, $xnor(a_i, b_i)$ and $and(a_i, b_i)$ onventional propagate and generate signals for a rry-lookahead adder); and the remaining circuit can us operate directly on twos-complement presentation producing the inverse of the sum (easily rrected). With trivial changes, we can also use $nd(a_i, b_i)$ instead of $and(a_i, b_i)$ and so reduce the te count of the design by 3N-1 and the critical path 2. More significantly, the design is thus equivalent our hybrid carry-lookahead/carry-select architecture own in Fig. 1 and it is a *purely* twos-complement chitecture whose high speed has no reliance upon BNR.

Fig. 1 (which corresponds to Fig. 6 and 8 in [1]). e pairs of 8-bit additions are performed using carry-okahead logic and their outputs selected according the true value of the corresponding carry-in. Unlike e usual carry-select architecture[3], the carry terms e generated by independent logic rather than by ing the results of successive adder sub-sections: this where the speed advantage lies since the Srinivas d Parhi design uses a tree structure to achieve $O(log(N))$ delay.

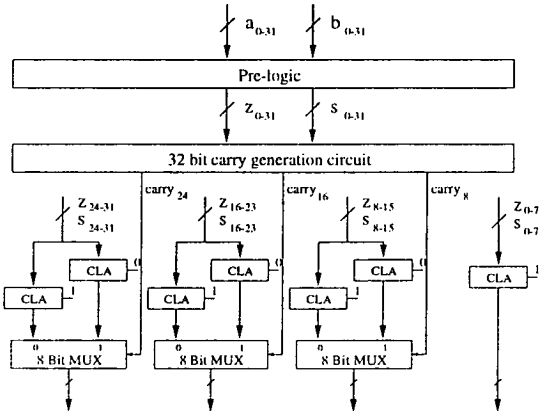


Fig. 1: The hybrid carry-lookahead/carry-select logic.

II. Carry-generator tree

The *carry-generator* is based on a novel 8-digit SBNR sign-checking circuit presented in [1] (Fig. 7 in [1]). There are two inputs to the circuit $\{Z_{N-1}, \cdots Z_0\}$ and $\{S_{N-1}, \cdots S_0\}$ which in [1] are generated by manipulation on an SBNR sum and interpreted as *zero* and *sign*. We have replaced this pre-logic with $Z_i = xor(a_i, b_i)$ and $S_i = and(a_i, b_i)$, and the remainder of the circuit can be described in the following way. Define $Z_{j,k} = and(Z_j, Z_{j-1}, \cdots Z_{k+1}, Z_k)$, and $C_{j,k}$ as the carryOut of the addition of bits j through to k with a zero carryIn (giving $C_{j,j} = S_j$).

We build a carry-generator tree using the following relations at each node:

$$mux\ (Z_{p,q}, C_{p,q}, C_{r,p-1}) = C_{r,q}$$
$$nand(Z_{p,q}, Z_{r,p-1}) = \overline{Z_{r,q}}$$
$$nor\ (\overline{Z_{p,q}}, \overline{Z_{r,p-1}}) = Z_{r,q}$$

which effectively combines two adjacent adder sections. The first relation states that the carryOut of combined sections is the same as the carryOut of the more significant section *unless* its inputs are such that its carryIn propagates through it completely. This is so when each bit has a "0" and a "1" input and the resulting carryOut will then be the carryOut of the less significant section. (In practice we propagate $\overline{carry}$ to

use an initial *nand* to form $\overline{S_j}$). The second and third relations maintain the function *each bit has a "0" and a "1" input* for the combined inputs at each node; these two relationships are not needed at the leaf nodes.

By applying these relations first to pairs of inputs, and then to pairs of the outputs of such pairs, etc, we build up a simple binary tree. This naturally produces carryOut on boundaries for successive powers of 2 (2, 4, 8, 16 etc); to obtain carryOut on intermediate boundaries, we augment the tree with other nodes (using the same relationship). Fig. 2 shows this for a 8-bit carry generation tree with 2-bit boundaries. Note that the generation of carry-bit 6 from inside the tree structure shows how the critical delay on each term can be limited to $\lceil \log_2(N) \rceil$. We believe this was not known to Srinivas and Parhi since it allows the use of 4-bit adder sections in their 32-bit example which moves the critical path from the carry-lookahead adders to the carry-generate tree and thus would have reduced their critical path delay.

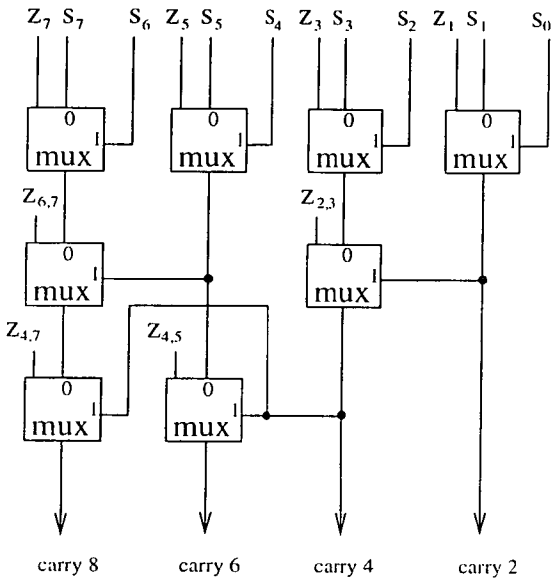


Fig. 2: An 8-bit carry-generation tree augmented for 2-bit boundaries.

The optimal decomposition for this architecture is found by determining the delay through the binary-tree carry generator and then selecting adder sub-sections (not necessarily carry-lookahead) which complete within that time. The only problem is fan-out, but this will be constrained in practice since larger adder tree sizes will lead to larger carry-boundaries.

The carry-generator tree is based upon two-input gates and multiplexors and, we believe, is suitable for the implementation of large adders using circuit techniques such as complementary pass-transistor logic[4] (CPL) which provide both high speed and low power. With the improvements outlined above, this circuit technique leads to a critical path delay of $\lceil \log_2(N) \rceil + 2$ gates. Thus a 32-bit adder, with a sub-block size of 4-bits, will have a critical path of 7 gate delays, compared to 12 as reported in [1].

The underlying architecture can now be seen to be similar to that of Lynch and Swartzlander[5] also published in 1992. Their design creates a tree structure using 4-input dynamic-logic Manchester carry chains and uses ripple adders in place of the carry-lookahead adders. Like Srinivas and Parhi their "modified" tree produces carry signals only on 8-bit boundaries. In contrast, the design presented here provides a simple static-logic implementation based on two-input nodes with greater flexibility in choosing the carry boundaries and with no requirement for a system clock. Our design also has similar performance to that reported by Suzuki *et al* [6] but with fewer gates. Finally, we also note that our improved circuit can form the basis for a fast SBNR to twos-complement conversion.

III. Conclusions

We have revealed the true source of the speed advantage in the Srinivas and Parhi[1] design, and have reduced the number of gates by 3N-1 and the critical path to $\lceil \log_2(N) \rceil + 2$ gate delays.

Acknowledgements

The authors would like to acknowledge the financial support of the Engineering and Physical Sciences Research Council (EPSRC) and Wolfson Microelectronics.

References

[1] H. R. Srinivas and K. K. Parhi, "A fast VLSI adder architecture", IEEE Journal of Solid State Circuits, vol. 7, no. 5, pp 761-767, May 1992.

[2] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic", IRE Transactions on Electronic Computers, vol. EC-10, pp. 389-400, Sep 1961.

[3] N. H. E. Weste and K. Eshraghian, "Principles of CMOS VLSI Design - A Systems Perspective", Second Edition, Addison-Wesley, 1993.

[4] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi and A. Shimizu "A 3.8ns CMOS 16x16-b multiplier using complementary pass-transistor logic", IEEE Journal of Solid-State Circuits, vol. JSSC-25, no. 2, pp. 388-395 Apr 1990.

[5] T. Lynch and E. E. Swartzlander Jr, "A Spanning tree Carry Lookahead Adder", IEEE Trans on Computers, vol. 41, no. 8, pp. 931-939, Aug 1992.

[6] K. Suzuki, M. Yasashima, T. Nakayama, M. Izumikawa, M. Nomura, H. Igura, H. Heiuchi, J. Goto, T. Inoue, Y. Koseki, H. Abiko, K. Okabe, A. Ono, Y. Yano and H. Yamada, "A 500Mhz, 32 bit, 0.4μm CMOS RISC Processor", IEEE Journal of Solid-State Circuits, vol. JSSC-29, no. 12, pp 1464-1473, Dec 1994.

Viterbi Equalization: Low-Power VLSI Module Design

J. M. Dobson, G. M. Blair and B. Mulgrew

The University of Edinburgh
The King's Buildings
Edinburgh EH9 3JL
Scotland, UK.

This paper appeared in the proceedings of IDSPCC'96

Abstract
This paper examines the application of Viterbi equalization for receiving Gaussian minimum shift keying (GMSK) signals with respect to the Group Special Mobile (GSM) system of mobile radio communications. GMSK modulation is described and the theoretical background to Viterbi Equalization for GMSK is reviewed. The VLSI architecture for the Viterbi equalizer is then described.

Keywords
VLSI, LOW POWER, MOBILE COMMUNICATIONS, EQUALIZATION, GSM

1. Introduction

The Viterbi algorithm is a well-accepted technique for receiving signals in the presence of inter-symbol interference [1]. In the last 15 years the algorithm has been applied to the area of channel equalisation for mobile radio communications [2].

Viterbi equalization was proposed in the COST 207 Report into Digital land mobile radio communications as a possible channel equalization technique for GSM [3]. The aim of this project is to design and a low-power implementation of a Viterbi equalizer (VE) suitable for a GSM receiver. The techniques presented here are currently being implemented for an intersymbol interference (ISI) of 5 bits which (in GSM) represents a multipath delay spread of $15\mu s$ [4].

For simplicity we will describe the theory and show diagrams for an ISI of 2 bits. Any important differences for an ISI of 5 bits will be indicated in the text.

The architecture uses high-speed combinatorial logic to minimise the speed of the internal clock, and is implemented by complementary pass-transistor logic (CPL) which offers the possibility of low power consumption and low area at a high speed [5]. We will show that the architecture can be implemented successfully using CPL to provide a low-power implementation.

2. GMSK Modulation

The GSM system uses GMSK as the modulation process. We approximate GMSK as an MSK modulation process and assume that the Gaussian shaping can be modelled as part of the channel impulse response. As the transition diagram in Figure 1 shows MSK is similar to 4-QAM except that only certain transitions between phase states are valid.

The Viterbi algorithm is based on the trellis diagram which is derived from the MSK transition diagram shown in Figure 1. The diagram for ISI=2 is shown in Figure 2. To construct the trellis we must choose, L, the level of ISI in bits, the number of states in the trellis is 2^{L+1}. Each state has

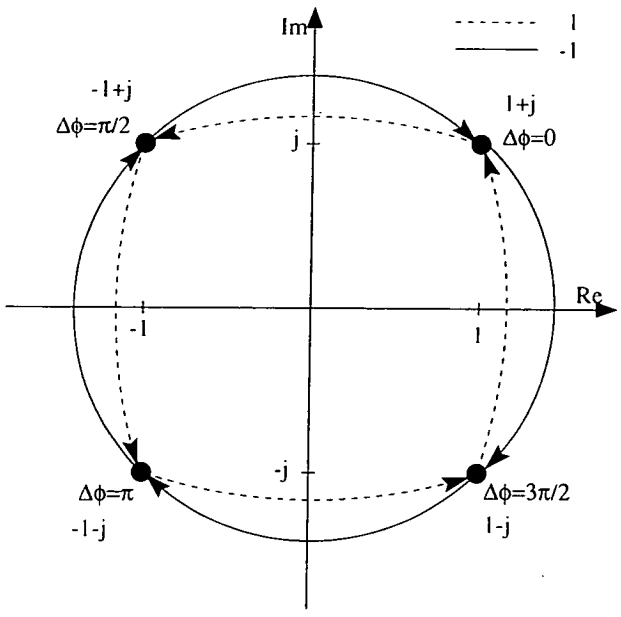


Figure 1: MSK transition diagram

a *correlative state vector* (a, b) = $(a_i, b_i, a_{i+1}, b_{i+1}, ..., a_{i+L-1}, b_{i+L-1})$ associated with it. This vector represents the magnitudes of the in-phase and quadrature-phase coefficients shown in Figure 1, hence a and b can assume the value ± 1. The branches in the trellis diagram are derived from mapping the transitions from Figure 1 onto the trellis diagram. Because of the nature of GMSK, if we know the starting state, only half of the states are valid at each iteration, this is indicated in Figure 1.

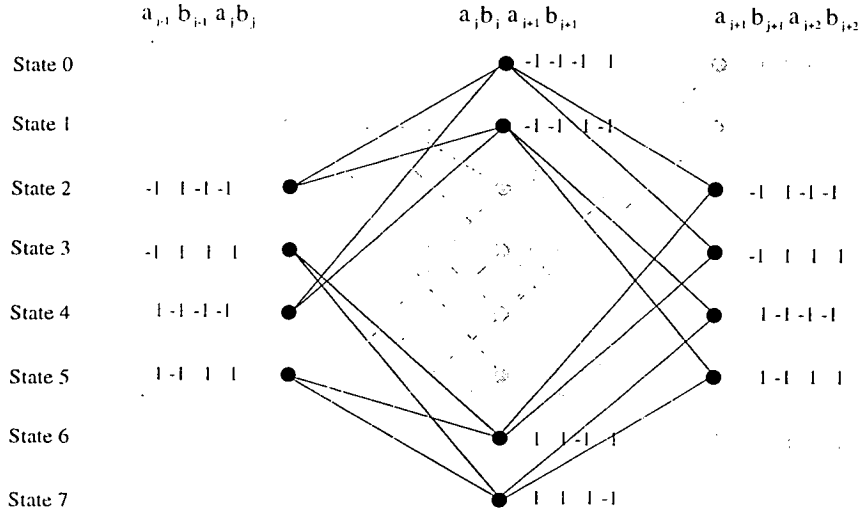


Figure 2: GMSK Trellis for ISI=2

For ISI=5 the resulting trellis diagram consists of 64 states. Again, only half of the trellis states are valid at each iteration.

3. The Viterbi Equalizer

For each possible received sequence, (a, b), the VE minimises the metric [2]:

$$\Lambda_n(\mathbf{a},\mathbf{b}) = \Lambda_{n-1}(\mathbf{a},\mathbf{b}) + a_n\left[y_n - \sum_{m=n-L}^{n-1}(a_m\chi_{n-m} + b_m\zeta_{n-m})\right] + b_n\left[z_n - \sum_{m=n-L}^{n-1}(b_m\chi_{n-m} + a_m\zeta_{n-m})\right] \quad (1)$$

$$\Lambda_n(\mathbf{a},\mathbf{b}) = \Lambda_{n-1}(\mathbf{a},\mathbf{b}) + I_s + Q_s \quad (2)$$

where y_n and z_n are the in-phase and quadrature received signals samples after passing through a filter matched to an estimate of the channel impulse response (CIR), and χ_n and ζ_n are the CIR matched filter autocorrelation coefficients. The CIR estimate remains constant for each single TDMA burst.

The summations I_s and Q_s in the equation (1) are pre-computed for each TDMA packet, S refers to the state in the trellis for which the summation is computed. The diagram in Figure 3 shows the overall structure of the VE for an N state trellis.

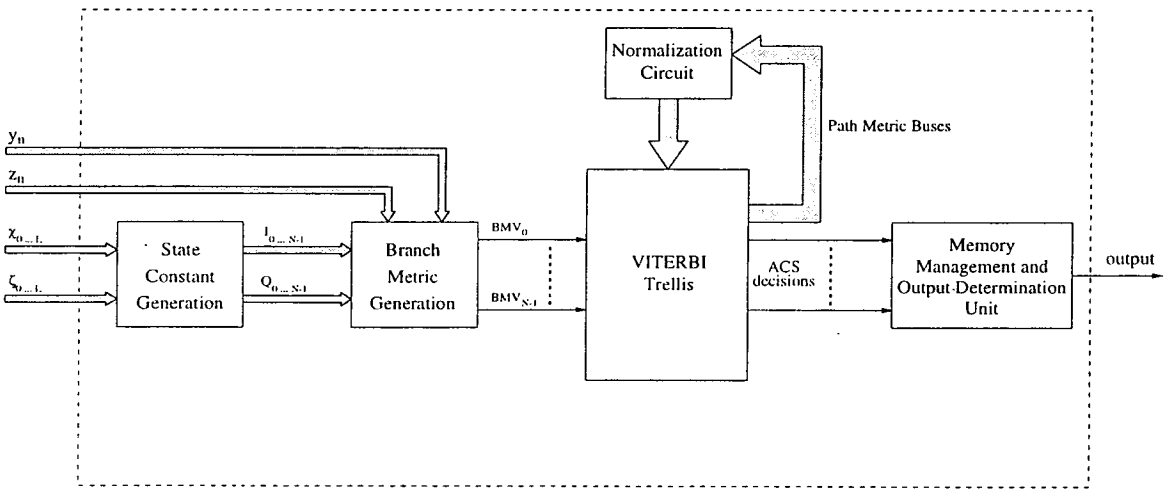


Figure 3: Top level block diagram

3.1 Summation Circuit

Examining equation (1) we can see that for each single TDMA packet we are required to evaluated 4 summations of a form similar to:

$$\sum_{m=n-L}^{n-1} a_n\chi_{n-m} \quad (3)$$

where $\{a_i\}$ is state dependant, $a_i = \pm 1$ and the χ sequence is the same for all states and constant for each single TDMA burst.

The number of states increase with the ISI level and therefore the number of summations required increases exponentially. To reduce the computational complexity we compute the summations using a differential addition circuit. Only one summation is evaluated initially and the remaining summations are obtained by adding $\pm 2\chi_i$.

Table 1 shows how the summations relate to the sequence $\{a_i\}$. Note from Table 1, each of the summations has a duplicate, and also each summation has a negative; which implies that we only need to compute a quarter of the summations.

State	a vector	$\sum_{m=n-L}^{n-1} a_n \chi_{n-m}$
0	-1 1	$-\chi_1 + \chi_2$
1	1 -1	$+\chi_1 - \chi_2$
2	-1 -1	$-\chi_1 - \chi_2$
3	1 1	$+\chi_1 + \chi_2$
4	-1 -1	$-\chi_1 - \chi_2$
5	1 1	$+\chi_1 + \chi_2$
6	-1 1	$-\chi_1 + \chi_2$
7	1 -1	$+\chi_1 - \chi_2$

Table 1: Summation Table for ISI=2

The summation tree for equation (3) is straightforward, and consists of one adder or subtractor. Two such trees are needed to compute I_s and Q_s for every state.

For ISI=5, each of the 2 trees perform 16 summations. All of the I_s and Q_s values can be computed in 6 adder levels and with 94 adder cells and 32 negation units. It would require 640 adder cells to compute these summation individually. An alternative implementation could produce these summations serially using two adder accumulators and the results stored at the branch metric cells.

3.2 Viterbi Trellis

The trellis is implemented as interconnected add-compare-select (ACS) units. Examining Figure 2 we see that only half of the states are used at each time interval, hence we can use only one ACS unit to represent different trellis nodes at odd and even time intervals. The branch metric generation unit ensures that the correct corresponding branch metric values (BMVs) are fed to the ACS units at the correct time interval.

The trellis nodes are paired up so that each ACS unit is shared between two nodes. The pairs are chosen in the following way:

- An even time interval node is paired with an odd time interval node.
- Each pair of nodes are connected to 4 nodes which themselves form two pairs.

This ensures the minimum amount of routing between ACS units and that the routing between node pairs is never redundant at either odd or even time intervals.

One possible pairing of the trellis states from Figure 2 is shown in Table 2.

State	Paired with
0	5
1	4
2	7
3	6

Table 2: State Pairing Table for Figure 3

The ACS unit implements the following function:

if $((PMV_0 + BMV_0) \geq (PMV_1 + BMV_1))$
  $PMV_{out} = PMV_0 + BMV_0$
  $select = 0$
else
  $PMV_{out} = PMV_1 + BMV_1$
  $select = 1$

The *select* signal is fed to the memory management unit which stores the path histories for all the trellis states.

The addition and comparison units are implemented using ripple propagation. The ripple adder's delay is proportional to the word length of the operands. However, the ripple adder often has the smallest area and lowest power of all the adder architectures [5]. In our ACS unit the comparison operation is implemented as a subtractor chain which operates in parallel to the two ripple adders chains. Thus the delay through these cascaded operations is equivalent to a full-adder and one carry chain.

3.3 Branch Metric Calculation

When a new pair of in-phase and quadrature-phase symbols (y_n, z_n) are received, the incremental metric for each branch on the trellis diagram is computed. From equation (1) the incremental metric associated with a transition from state p to state q is:

$$\lambda_{p \to q} = a_n(y_n - I_s) + b_n(z_n - Q_s) \tag{4}$$

where a_n and b_n are the expected in-phase and quadrature-phase signals which would result in a transition from state p to state q.

From Table 2, state 0 and state 5 are paired and so they share the same ACS unit in the implementation of the Viterbi trellis. The branch metric values associated with these states can be derived from Figure 2. These are:

$$\begin{aligned}
\lambda_{2 \to 0} &= -1(y_n - I_2) + 1(z_n - Q_2) \\
\lambda_{4 \to 0} &= -1(y_n - I_4) + 1(z_n - Q_4) \\
\lambda_{1 \to 5} &= 1(y_n - I_1) + 1(z_n - Q_1) \\
\lambda_{7 \to 5} &= 1(y_n - I_7) + 1(z_n - Q_7)
\end{aligned}$$

These two states share two branch metric calculation cells, as well as the same ACS unit. At even samples the metrics for state 0 are computed and at odd samples the metrics for state 5 are computed. Figure 4 shows these two branch metric calculation cells, where *sample* is 0 for even samples and 1 for odd samples.

The BMV generation cell is shown in Figure 5; it performs the sum $\pm(I_s - y_n) \pm (Q_s - z_n)$ where the $\pm$ operator is chosen by the values of *ctrl a* and *ctrl b*. At some nodes the control signals are fixed and so the logic can be optimized.

As discussed at the end of Section 3.2 the adders in the branch metric value cell are cascaded so that the delay through the adders is equivalent to a full-adder and one carry chain.

3.4 Metric Normalisation

The metrics must be normalised so that they can be stored using finite length registers. In the conventional Viterbi algorithm, metrics are always positive and are normalised by observing when all the most significant bits are set and resetting them [6]. In our design the metrics can be both positive and negative 2's complement numbers. We normalise metrics of N bits in length if:

- All metrics are positive and $\geq 2^{N-2}$.
- All metrics are negative and $\leq 2^{N-2}$.

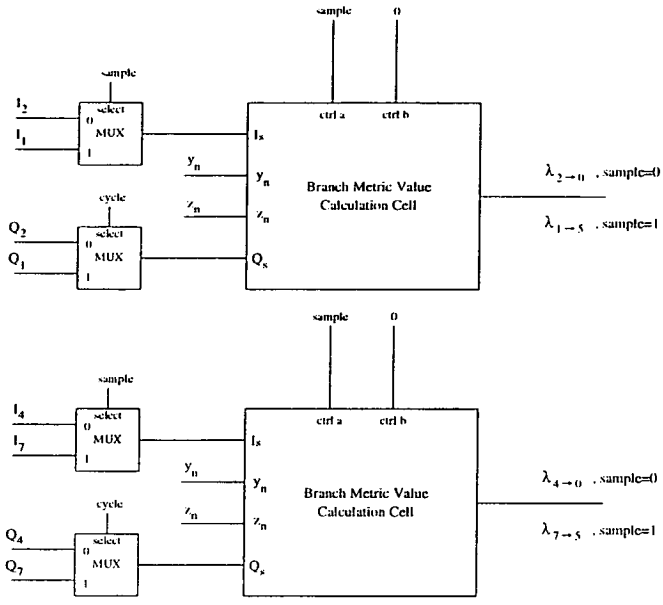


Figure 4: The first two cells of the BMV unit for ISI=2

In the first case, bit $N-1=0$ and bit $N-2=1$, to normalise we reset bit $N-2$. In the second case, bit $N-1=1$ and bit $N-2=0$, to normalise we set bit $N-2$. The Normalisation unit is shown in Figure 6

For ISI=5, assuming our channel estimation coefficients have an accuracy of 8 bits, our path metric values are required to be stored using 18 bits. This means that normalisation occurs if all of the metrics are larger than 2^{16} or if all of the metrics are lower that -2^{16}

3.5 Memory Management

We have used a register exchange method of path history memory management for simplicity. However, the module has been specified so that it is compatible with a traceback management unit [7] to allow the possibility of future development.

Each state in the trellis appends either a 1 or a 0 to the path history. The *append* signal shown in the circuit is dependent on the trellis state and the cycle. For simplicity the output bit is determined

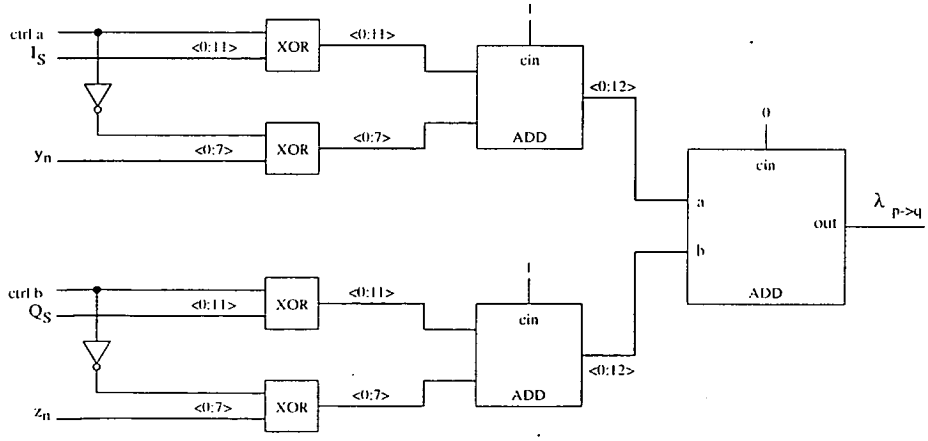


Figure 5: BMV cell

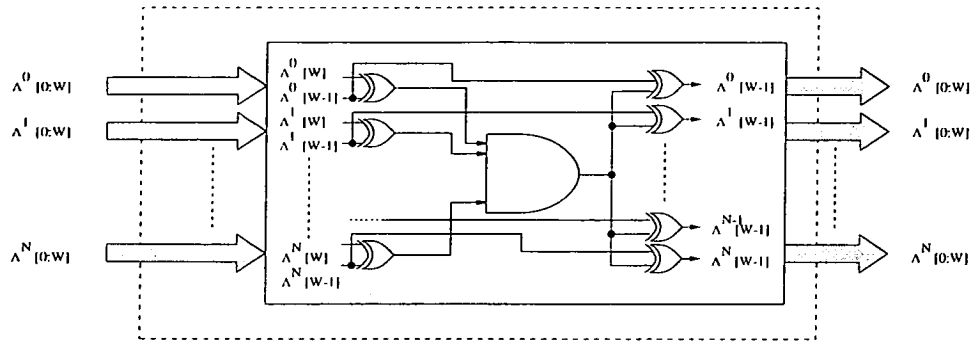


Figure 6: Metric Normalisation Unit

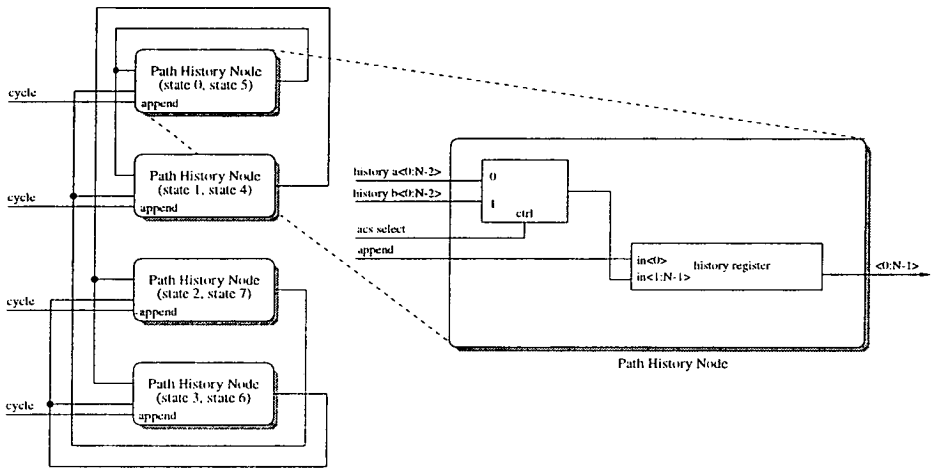


Figure 7: Register Exchange Memory Management Unit

by a majority vote on the oldest bit in all of the path histories.

For ISI=5, we have 32 registers in the memory management unit. For majority voting the the length of the path history register is 5-6 times the length of the path metric values [6]. This means that the path history registers should be roughly 90 bits in length. This is quite large which would suggest that a traceback method of path history memory management would be more desirable.

4. Performance

We need to estimate the delay though the circuit to determine whether CPL [5] will produce a satisfactory implementation. From the specifications of GSM [3] the VE output is buffered to produce a continuous bitstream at *22.8 kbit/s*, hence the VE has *5ms* to complete the decoding of a TDMA packet.

Each GSM TDMA packet contains 114 bits of data which need to be passed through the VE, in addition, the VE needs to be flushed with the same amount of bits as in the path history. For these iterations the branch metrics are all set to zero. With a path history of 90 bits, this means that our VE has to perform 204 iterations.

Assuming that the output determination unit has a shorter delay than the Viterbi trellis then the clock period T in CPL gate delays is:

$$T \geq (\Delta_{BMV} + \Delta_{ACS} + \Delta_{NORMALIZE}) \tag{5}$$

and the delay through the state constant generator followed by 204 iterations of the VE must be less than $5ms$.

Circuit Element	Delay (CPL gates)
State Constant Generator	20
Branch Metric Generator	16
Add-Compare-Select Unit	15
Normalization Unit	5

Table 3: CPL gate delays for various circuit elements

The number of CPL gate-delays through the various circuit elements are summarised in Table 3. Using these delays and equation (5) we can estimate a maximum gate-delate of $0.5\mu s$ which is clearly realisable.

5. Conclusions

In this paper we have reviewed the concepts of Viterbi equalization for GMSK signals as used in the GSM system of mobile radio communications. We have presented the design of a VLSI implementation of a Viterbi Equalizer module using Complementary Pass-transistor Logic, which satisfies the GSM specification and has low-power requirements.

Further work will include implementing the memory management unit using a traceback system rather than register passing. This should further reduce the area requirement and power usage of the design.

Acknowledgements

The authors would like to acknowledge the financial support of the Engineering and Physical Sciences Research Council (EPSRC) and Wolfson Microelectronics Ltd[1].

References

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptoically optimum decoding algorithm", *IEEE Transactions on Information Theory*, vol. IT13, pp. 260–269, Apr 1967.

[2] A. S. Acampora, "Analysis of maximum-likelihood sequence estimation performance for quadrature amplitude modulation", *The Bell System Technical Journal*, vol. 60, no. 6, pp. 865–885, July-August 1981.

[3] COST 207 Management Committee. Digital Land Mobile Radio Communications - Final Report, 1988.

[4] R. D'Avella, L. Moreno, and M. Sant'Agostino, "An Adaptive MLSE Receiver for TDMA Digital Mobile Radio", *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 1, pp. 122–129, Jan 1989.

[5] A. Bellaouar and M. I. Elmasry, *Low-Power Digital VLSI Design*. Kluwer Academic Publishers, 1995.

[6] M. A. Bree. A Bit-Serial Viterbi Processor. Master's thesis, University of Saskatchewan, Dec 1988.

[7] G. Feygin and P. G. Gulak, "Arcitectural tradeoffs for survivor sequence memory management in Viterbi decoders", *IEEE Transactions on Communications*, vol. 41, no. 3, pp. 425–9, Mar 1993.

[1] Wolfson Microelectronics Ltd, Edinburgh EH9 9NX, Scotland, UK

A Low Area Serial Viterbi Equalizer Implementation

Jonathan M. Dobson and Gerard M. Blair
Department of Electrical Engineering,
The University of Edinburgh,
Edinburgh, Scotland
{jmd,gerard}@ee.ed.ac.uk

5th April 1997

Abstract

In recent years the popularity of mobile telephones has risen dramatically. With the introduction of the European digital telephone standard GSM in 1988 the quality of these telephone links has also improved. The Viterbi Algorithm has been an excepted method of decoding convolutional code since its conception in 1967. Recently it has been used to equalizer channel effect in digital transmission, and is a popular equalization technique for GSM receivers. The paper presents the design of a low area Viterbi Equalizer VLSI module for GSM we will conclude that the design is compact and suitable for a handheld GSM receiver.

I Introduction

The Viterbi algorithm is a well-accepted technique for receiving signals in the presence of intersymbol interference [1]. In the last 15 years the algorithm has been applied to the area of channel equalisation for mobile radio communications [2].

Viterbi equalization was proposed in the COST 207 Report into Digital land mobile radio communications as a possible channel equalization technique for GSM [3]. The aim of this project is to design and a low-power implementation of a Viterbi equalizer (VE) suitable for a GSM receiver. The techniques presented are implemented for an intersymbol interference (ISI) of 5 bits which (in GSM) represents a multipath delay spread of $15\mu s$ [4].

When designing a VLSI module for a handheld device it is important to produce a low area, and low power implementation. Previous work in designing a VLSI VE module [5] showed that a fully parallel module results in a large use of silicon area, which is not suitable for a handheld device. When designing a VE module for a hand-held mobile GSM receiver, it is preferable to develop a serial module which has a smaller number of add compare select (ACS) units, than trellis nodes.

In this paper the design of a serial VE module is described. For speed of implementation we have used the ES2 $0.7\mu m$ CMOS process. Section II describes the overall VE module. Section III describes the implementation of the design using Verilog HDL , the ES2 design kit, and Cadence Design Framework II. Section IV outlines the testing and the layout of the design

and Section V describes the specifications of the implementation. Finally in Section VI some conclusions are drawn.

II Architecture

The simplified block diagram in Figure 1 represents the serial VE module. The VE theory will

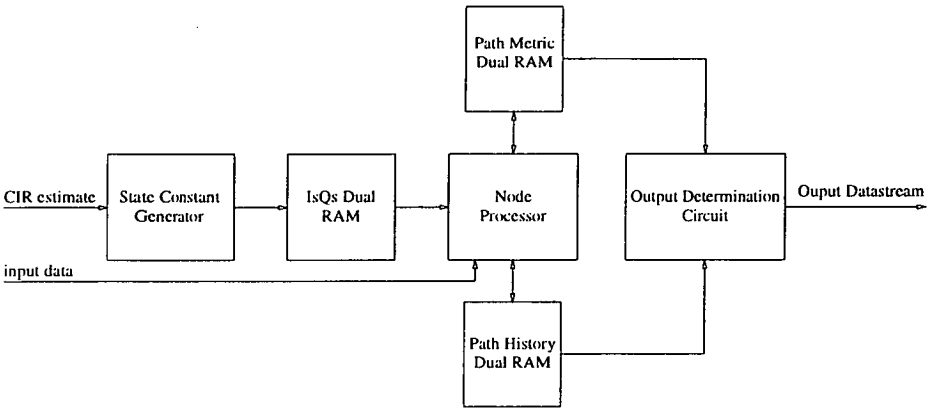


Figure 1: A block diagram of the Serial Viterbi Equalizer

not be described here. Explainations can be found in the literature [3], [2], [5].

For each possible received sequence, $(\mathbf{a}, \mathbf{b})$, the VE minimises the metric [2]:

$$\begin{aligned}\Lambda_{mn} &= \Lambda_{m(n-1)} + a_n^m \left[y_n - \sum_{i=n-L}^{n-1} (a_i^m \chi_{n-i} + b_i^m \zeta_{n-i}) \right] \\ &\quad + b_n^m \left[z_n - \sum_{i=n-L}^{n-1} (b_i^m \chi_{n-i} + a_i^m \zeta_{n-i}) \right] \\ &= \Lambda_{m(n-1)} + a_n^m [y_n - I_s] + b_n^m [z_n - Q_s] \end{aligned} \quad (1)$$

where y_n and z_n are the in-phase and quadrature received signals samples after passing through a filter matched to an estimate of the channel impulse response (CIR), and χ_n and ζ_n are the CIR matched filter autocorrelation coefficients. The CIR estimate remains constant for each single TDMA burst.

The summations I_s and Q_s in the equation (1) are pre-computed for each TDMA packet, S refers to the state in the trellis for which the summation is computed. Once computed, the state constants are stored in a block of memory out of which they are read by the node processor (NP). Note from the diagram that the memory is a "dual" memory. This means that there is in fact two blocks of memory, while one block is being written to by the state constant generator (SCG), the other block is being read by the NP. This allows the SCG to compute the state constants for the next packet while the NP is decoding the current packet.

To have the SCG and the NP working on different packets, it is assumed that the input and output of the whole module is buffered. This is acceptable because due to the GSM block encoding scheme [3], the packets must be stored at the output of the VE module.

The NP contains a single ACS unit which represents each of the trellis nodes in turn throughout one iteration. The path metrics values (PMVs) and path histories (PHs) are now stored in two RAM modules external to the NP.

The PMV and PH RAMs are also "dual" RAMs similar to the RAM for the state constants. This is because of the iterative nature of the VA which means that the results of the previous iteration need to be available for all of the required nodes in the subsequent iteration. To achieve this two identical blocks of memory are present in the two RAM modules, during odd and even iterations, alternative RAMs are used for reading and writing.

Finally, the output determination unit is far simpler than the one required for the fully parallel version [5]. A very small metric selection circuit has been designed, which selects the largest metric during the final iteration, and outputs the corresponding PH.

III Implementation

It was decided to implement the serial VE design using the Cadence automatic place and route tools and the ES2 $0.7\mu m$ CMOS design kit.

Verilog HDL was used because the language allows the combination of behavioural and structural module descriptions in a straightforward manner. Initially a Verilog description for the design was produced with the low-level modules described with behavioural definitions and the more high level modules defined with structural definitions.

Each of the behavioural modules were imported into the Cadence Synergy logic synthesis tool which was used to produce structural implementations using the ES2 library. This structural description for each module was exported as a Verilog netlist used to replace modules in the original Verilog file.

IV Testing and Layout

When the behavioural definition of the design was produced, sets of test vectors were created to verify the design. Synthesised versions of each module were used to replace the behavioural versions and the design was retested. This made it straightforward to identify errors which where introduced as a result of the synthesis step. The equalizer module was place and routed

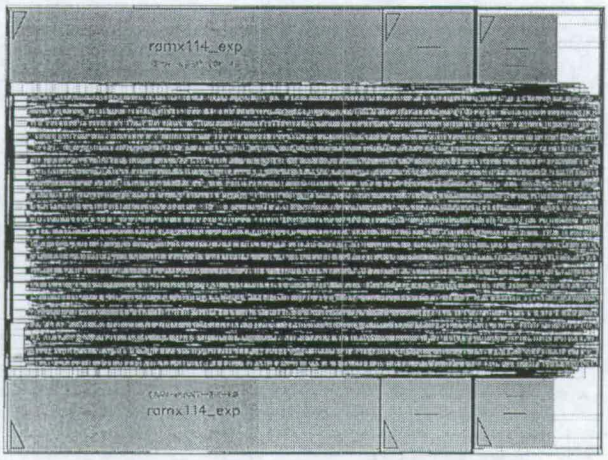


Figure 2: The Viterbi Equalizer layout

using the Cadence Cell Ensemble automatic place and route tools. Figure 2 show the layout for the serial VE module. The area of the module is $4392 \times 3233\mu m$ which makes the design suitable for a handheld receiver.

In Figure 2 the RAM modules are clearly visible at the top and bottom of the circuit. The large RAMs are for the path history storage and the smaller RAMs to the right are the path metric, and state constant RAMs.

V Performance

Table 1 describes the input and output ports of the VE module. Simulation showed that the maximum clock speed for the module was $20MHz$ which is well within the requirements for GSM.

Port Name	Input/Output	Description
clk	input	system clock, max. freq $20MHz$
reset	input	system reset
nextPacket	input	indicates the start of a new packet
y_n[0:7]	input	received in-phase signal statistic
z_n[0:7]	input	received quad-phase signal statistic
coeffIn[7:0]	input	coefficient value
coeffSelect	input	selects either χ or ζ coefficients
coeffNum[2:0]	input	index of current coefficient
latchCoeff	input	new coefficient value is ready
dataOut	output	equalized data stream
dataEnd	output	indicates that decoding has finished on the current packet

Table 1: The ports of the serial VE module

VI Conclusions

In this paper a serial implementation of a Viterbi Equalizer for GSM has been proposed. Using the concepts which were developed in design of the parallel VE in [5] a serial design has been produced which has been implemented and has been shown to have a low area.

VII Acknowledgements

This work was perfomed with the support of the EPSRC and Wolfson Microelectronics Ltd, Edinburgh.

References

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptoically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. IT13, pp. 260–269, Apr 1967.

[2] A. S. Acampora, "Analysis of maximum-likelihood sequence estimation performance for quadrature amplitude modulation," *The Bell System Technical Journal*, vol. 60, pp. 865–885, July-August 1981.

[3] COST 207 Management Committee, "Digital Land Mobile Radio Communications - Final Report," 1988.

[4] R. D'Avella, L. Moreno, and M. Sant'Agostino, "An Adaptive MLSE Receiver for TDMA Digital Mobile Radio," *IEEE Journal on Selected Areas in Communications*, vol. 7, pp. 122–129, Jan 1989.

[5] J. M. Dobson, G. M. Blair, and B. Mulgrew, "Viterbi equalization: low-power VLSI module design," in *Irish DSP and Control Conference 1996*, pp. 253–260, 1996.

[6] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *IEEE Journal of Solid State Circuits*, vol. 31, pp. 793–803, June 1996.