AN ABSTRACT OF THE THESIS OF

Lupin Chen for the degree of Master of Science in

Electrical and Computer Engineering presented on September 4, 2007.

Title: Modified VLSI Designs for Error Correction Codes

Abstract approved:

Zhongfeng Wang

Nowadays, error correction codes have become an integral part in almost all the modern digital communication and storage systems. With the continuously increasing demands for higher speed and lower power communication systems, efficient VLSI implementations of those error correction codes have great importance for practical applications. In this thesis, several VLSI design issues for Viterbi decoder and Low-Density Parity-Check (LDPC) codes decoder will be discussed. We propose a low-power memory-efficient Viterbi decoder to reduce the memory read operations in the survivor memory unit (SMU) and the memory size of SMU. We develop a parallel Viterbi decoder for high throughput applications. We also propose an efficient early stopping scheme to reduce the number of decoding iterations for LDPC codes decoding.

# Modified VLSI Designs for Error Correction Codes

by

Lupin Chen

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Presented September 4, 2007

Commencement June 2008

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my major professor, Dr. Zhongfeng Wang. I feel very honored and privileged to have worked under his supervision. I greatly benefited from his deep intuition and extensive knowledge of VLSI designs, from his invaluable teaching and research skills, and from his guidance in writing papers/thesis and making presentations.

I would like to thank Dr. Huaping Liu, Dr. Thomas K. Plant, and Dr. James A. Coakley, Jr. for serving on my committee.

I would also like to thank the members of our group, particularly, Zhiqiang Cui, Qingwei Li, Jinjin He and Somya Rathi  for many useful discussions.

I am very grateful to my parents, who demonstrated in so many admirable ways their unconditional love and support throughout my life.

.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

# LIST OF TABLES

# MODIFIED VLSI DESIGNS FOR ERROR CORRECTION CODES

# 1 INTRODUCTION

## 1.1 Overview

This thesis is devoted to the efficient VLSI architecture design for error correction codes. Nowadays, the error correction codes have become an integral part of almost all modern digital communication and storage systems. With the continuously increasing demands for higher speed and lower power communication systems, efficient VLSI implementations of those error correction codes that are currently used in practical applications are of great importance. In this thesis, several VLSI design issues for Viterbi decoder and Low-Density Parity-Check (LDPC) codes decoder are discussed. We propose a low-power memory-efficient Viterbi decoder to reduce the memory read operations in the survivor memory unit (SMU) and reduce the memory size of SMU. We develop a parallel Viterbi decoder for high throughput applications. We also propose an efficient early stopping scheme to reduce the number of decoding iterations for LDPC codes decoding.

## 1.2 Summary of Contributions

### *1.2.1 A Low-Power Memory-Efficient Viterbi Decoder Design*

We propose a new low-power memory-efficient trace back (TB) scheme for high constraint length Viterbi decoders (VD). With the trace back modifications and path merging techniques, more than 50% memory read operations in the survivor memory unit (SMU) can be eliminated. The memory size of SMU can be reduced by 33% and the decoding latency can be reduced by 14%. The simulation results show that, compared to the conventional TB scheme, the performance loss of this scheme is negligible.

### *1.2.2 A Parallel Viterbi Decoder Architecture for High-Throughput Applications*

High-throughput Viterbi decoders for convolutional codes are very attractive for high-data-rate applications. Parallel processing is a powerful technique for high-throughput applications. We present a parallel Viterbi decoder architecture, which breaks the bottleneck of the add-compare select (ACS) unit. The simulation result shows that, compared to the conventional RE scheme, no performance degradation is observed for the parallel RE scheme. The proposed architecture is well suited for high-speed data applications.

### *1.2.3 An Efficient Early Stopping Scheme for LDPC Decoding*

We propose an efficient early stopping scheme for LDPC codes decoding to detect undecodable blocks at early stages and hence to save unnecessary power consumption. The proposed approach thoroughly exploits the convergence of the summation of the sign products computed in the check-to-variable message passing phase. The new approach can significantly reduce the average number of decoding iterations in the low to medium signal to noise ratio (SNR) range while the performance loss is negligible. In the high SNR range, the proposed scheme can turn off early stopping mechanism to avoid performance loss and unnecessary computation. The computation overhead of the proposed scheme is very small.

## 1.3 Outline of the Thesis

This thesis is outlined as follows. In Chapter 2, we introduce the basic elements of a digital communication system. Then, we explain the basic ideas of coding. We continue with an introduction of convolutional codes and their properties, followed by an explanation of Viterbi algorithm and VLSI implementations of Viterbi algorithm. After an introduction of block codes and their properties, LDPC codes are introduced. We also define the Tanner graph, which is a visualization of codes, suited for LDPC codes. Finally, we briefly introduce the encoding and decoding for LDPC codes. In Chapter 3, we present a low-power memory-efficient Viterbi decoder design. An illustration of conventional TB algorithm is included. Then, the proposed TB algorithm and the architecture design are presented. Then, the simulation results and discussions are included. A conclusion is given at the end of this chapter. In Chapter 4, we propose a parallel RE Viterbi decoder architecture. A brief literature survey of high-speed VD design and an introduction of parallel processing technique are given at the beginning of this chapter. Then, the proposed parallel RE decoding scheme is introduced. Then, the proposed architecture and the simulation results are included. A conclusion is given at the end of this chapter. In Chapter 5, we present an efficient early stopping scheme for LDPC decoding. A brief introduction of LDPC codes and LDPC decoding is given at the beginning of this chapter. Then, the proposed early stopping scheme is presented. After the simulation results and discussions, a conclusion is given at the end of this chapter. In chapter 6, we give a final conclusion of this thesis.

# 2 ERROR CORRECTION CODES

In this chapter the basic concepts of digital communications and error correction codes are introduced. This chapter will begin with an overview of digital communication and coding. Then, we will talk about convolutional codes and Viterbi algorithm. A brief introduction of block codes and linear block codes and their characteristics will be followed. After that, we will give a brief introduction of LDPC codes and their characteristics.

## 2.1 Digital Communication

### *2.1.1 Digital Communication System*

A digital communication system is a way of transporting information from an endpoint A to an endpoint B. The system is digital, meaning that the information is represented by a sequence of symbols from a finite discrete alphabet. The sequence is mapped onto analog signals, which is then transmitted through a physical channel. During transmission the signal is distorted by noise, so the received signal is not the same as the sent one. The receiver selects the most likely sequence of symbols and delivers it to the receiving endpoint B.

The transmitter and receiver functions are always performed by different elements. Figure 2.1 shows the basic elements of a digital communication system [1]. First, information signal is sampled and quantized to form a digital sequence, then it passes through the source encoder to remove any unnecessary redundancy in the data. Then, channel encoder codes the information sequence so that it can recover the correct information after passing through the channel. The information sequence is protected by error correction codes such as convolutional codes and LDPC codes. The digital modulator maps the binary sequence onto analog signal waveforms so that it can be efficiently transmitted over the communication channel. The modulator acts as an interface between the digital signal and the channel.

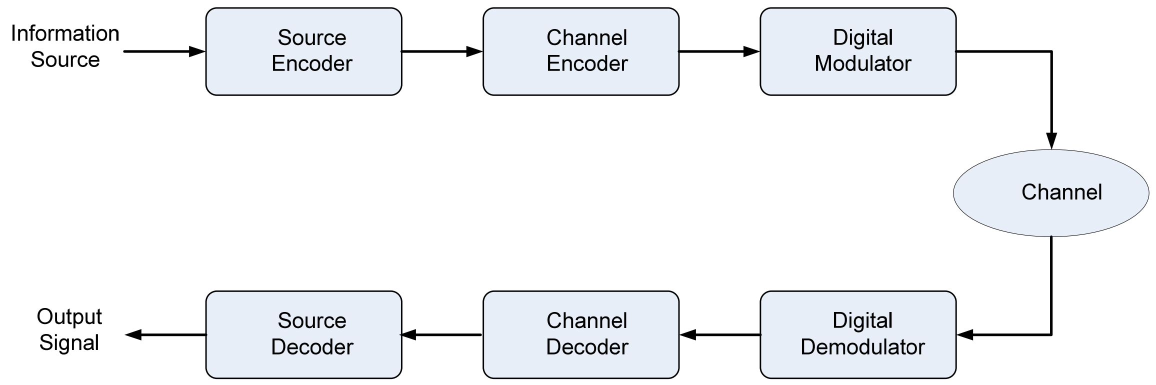


Figure 2.1 Basic elements of a digital communication system.

The communication channel is the physical medium that is used to send the signal from the transmitter to the receiver. The channel attenuates the transmitted signal and introduces noise. The attenuation is generally caused by energy absorption and scattering in the propagation medium. The noise is generated in a random manner by many possible mechanisms such as ambient heat in the transmitter/receiver hardware and the propagation medium, hardware-induced transients, co-channel and adjacent-channel interference from other communication systems, or climatic phenomena. The most commonly assumed noise model is the additive white Gaussian noise (AWGN) model [1].

At the receiving end of the digital communications system, the digital demodulator processes the channel-corrupted transmitted waveform and recovers a sequence of digital values from the waveforms, then feeds it into the channel decoder. The decoder reconstructs the original information sequence by the knowledge of the code used by channel encoder and the redundancy contained in the received data. Channel decoders can be Viterbi decoders [2], LDPC decoders [7], etc. Then, source decoder decompresses the data and retrieves the original information. The probability of having error in the output sequence is a function of the code characteristics, the type of modulation, channel characteristics such as noise and interference level, etc. There is a trade-off between the power of transmission and the bit error rate. Researchers are trying to minimize the power consumption while maintaining a reliable communication. This raises a need for stronger codes with more error correction abilities.

### *2.1.2 Coding*

In 1948 Shannon published a paper regarding basis of the entire field of information theory [48]. In that work, he introduced a metric by which the information can be quantified. This metric allows one to determine the minimum possible number of symbols necessary for the error-free representation of a given message. A longer message containing the same information is said to have redundant symbols. These can lead to the definition of three distinct types of codes [3]:

**Source codes:** These codes are used to remove the uncontrolled redundancy from the information symbols. Source coding reduces the symbol throughput requirement placed upon the transmitter.

**Secrecy codes:** These codes encrypt the information so that it cannot be understood by anyone else except the intended recipient.

**Error control codes (error correction codes or channel codes):** These codes are used to format the transmitted information so that it can increase its immunity to noise. This is achieved by adding controlled redundant information into the transmitted information stream, allowing the receiver to detect and possibly correct those errors.

As we mentioned before, in a communication system, all three types of these codes are used to increase the reliability and performance of the system.

## 2.2 Convolutional Codes

Convolutional codes are widely used in modern digital communication systems including deep space communications and wireless communications due to their powerful error correction capabilities and low decoding latency. Convolutional coding can be applied to a continuous input stream (which cannot be done with block codes), as well as blocks of data.

### *2.2.1 Convolutional Code Representation*

Convolutional codes are usually described using two parameters: the code rate and the constraint length. The code rate, *k/n*, is expressed as a ratio of the number of bits, *k*, into the convolutional encoder to the number of channel symbols, *n*, output by the convolutional encoder in given encoder cycle. The constraint length, *K*, denotes the length of the convolutional encoder, *i.e.*, how many k-bit stages are available to feed the combinational logic that produces the output symbols. Closely related to *K* is the parameter *m*, which indicated how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the convolutional encoder. The *m* parameter can be thought of as the memory length of the encoder.

A convolutional encoder can be considered as a finite state machine and represented by state diagrams, graphs, or trellises. It generates a coded output data stream from an input data stream. It consists of shift registers and a network of Exclusive-OR (XOR) gates as shown in Figure 2.2.

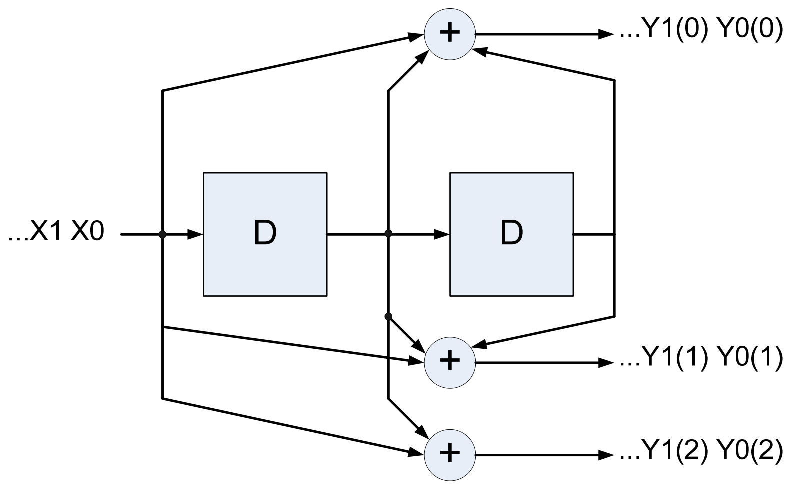


Figure 2.2 A rate-1/3 convolutional encoder.

The encoder in Figure 2.2 produces three bits of encoded information for each bit of input information, so it is called a rate 1/3 encoder. A convolutional encoder is generally characterized in (*n, k, m*) format. The rate of a (*n, k, m*) encoder is *k/n*. The encoder shown in the Figure 2.2 is a (3, 1, 2) encoder with rate 1/3.

A convolutional encoder is a Mealy type state machine, where the output is a function of the current state and the current input as well. It consists of one or more shift registers and multiple XOR gates. The information sequence passes into the linear finite-state shift registers from one end and then is shifted out at the other end. For the optimal location of the shift register stages to be connected to XOR gates, it is based on empirical experience and no theoretical principle. The location of stages as well as the number of shift registers determines the minimum Hamming distance. The maximal number of correctable bits is determined by minimum Hamming distance. Detailed information about the interconnection functions for different rates and different number of memory elements and their minimum Hamming distances are available in [4].

The operation of a convolutional encoder can be easily understood with the aid of a state diagram. Figure 2.3 represents the state diagram of the encoder shown in Figure 2.2. Figure 2.3 depicts state transitions and the corresponding encoded outputs. As this rate-1/3 encoder has two memory cells, there are four possible states. These four states are represented as $S_0$ through $S_3$. The information of each state (*i.e.*, the contents of shift register for the state) along with one input generates an encoded output code. For each state, only two outgoing transitions can be observed: one corresponding to a '0' input bit and the other corresponding to a '1' input bit.

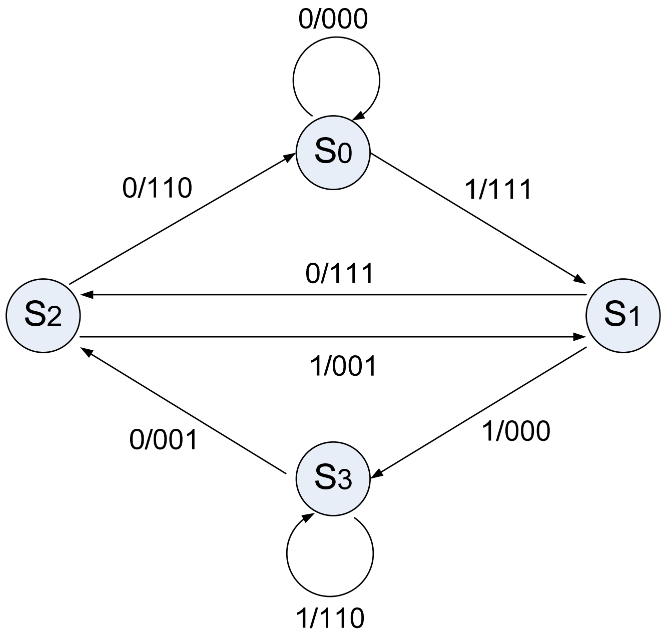


Figure 2.3 State diagram for encoder in Figure 2.1 [3].

A trellis diagram is an extension of a state diagram that explicitly shows the passage of time. In Figure 2.4, the state diagram is extended in time to form a trellis diagram for the encoder given in Figure 2.2. In the trellis diagram, nodes correspond to the states of the encoder. The branches of the trellis diagram are labeled with the output bits corresponding to the associated state transitions. From an initial state ($S_0$) the trellis records the possible transitions to the next states for each possible input pattern in every stage. At the stage t = 1 there are two states $S_0$ and $S_1$, and each state has two transitions corresponding to input bits '0' and '1'. Therefore, the trellis grows up to the maximum number of states or nodes, which is determined by the number of shift register in the encoder. After all the encoded symbols of the information bits are transmitted, the encoder is usually forced back into the initial state by applying a fixed *m* zeros input sequence called force zero sequence. The trellis diagram in Figure 2.4 is for an input length of five bits, in which the last two bits zero represent the force zero sequence. It should be pointed out that, there is a unique path for every codeword that begins and stops at the initial state.

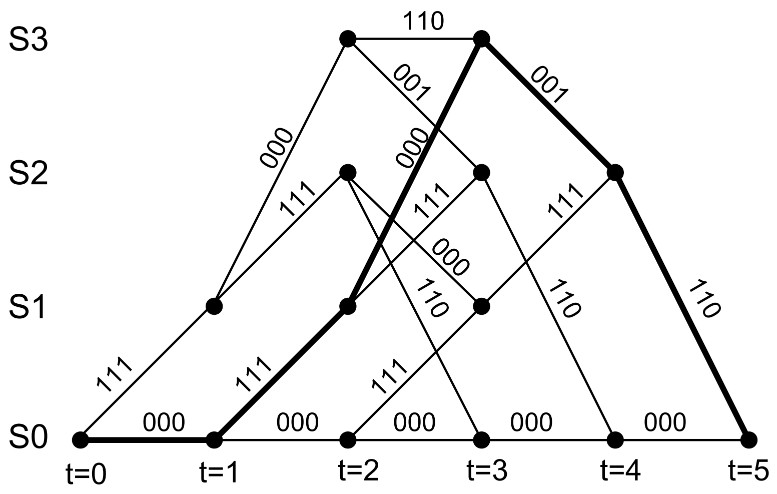


Figure 2.4 Trellis diagram for inputs of length three to the encoder in Figure 2.2.

## 2.3 The Viterbi Algorithm

The Viterbi algorithm (VA), first introduced in [2], is known to be an optimal decoding method for convolution codes. The function of the VA is to find a maximum likelihood sequence in the trellis diagram based on the received symbols. The VA has been used in many digital communication systems such as magnetic recording systems, satellite communication systems, mobile communication systems and video broadcasting systems. The Viterbi decoding algorithm is a decoding method for convolutional codes or trellis codes in a memoryless channel. Figure 2.5 depicts the transmission flow of information over a noisy channel. An information sequence **x** is encoded to form a convolutional codeword **y**, which is transmitted through a noisy channel. The convolutional decoder takes the received vector **r** and tries to extract the transmitted information sequence through a decoding algorithm and generates an estimate **y'** of the transmitted codeword. A decoding algorithm that maximizes the probability *p*(**r**|**y'**) is a **maximum likelihood (ML)** algorithm. An algorithm which maximizes the *p*(**y'**|**r**) through the proper selection of the estimate **y'** is called a **maximum a posteriori (MAP)** algorithm. The two algorithms have identical results when the source information **x** has a uniform distribution.

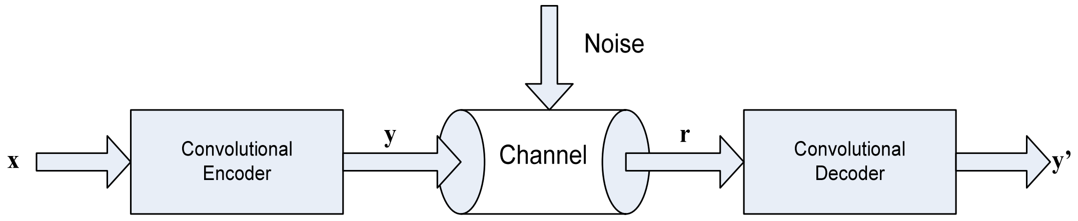


Figure 2.5 The convolutional encoding and decoding system.

Since the received signal is analog, it can be quantized into several levels. If the received signal is converted into two levels, either zero or one, it is called **hard decision**. If the input signal is quantized and processed for more than two levels, it is called **soft decision**. The soft decision captures more information in the input signal consequently performing better than the hard decision at the cost of a higher complexity.

The Viterbi algorithm based on the **ML** algorithm and the hard decision is illustrated in Figure 2.6. The trellis in the Figure 2.6 corresponds to the convolutional encoder given in Figure 2.2. The received code symbols are shown at the bottom of the trellis. The encoder encodes an input sequence (11010100) and generates the codeword (111,000,001,001,111,001,111,110). This codeword is transmitted over a noisy channel, and (1$\underline{0}$1,$\underline{1}$00,001,0$\underline{1}$1,111,$\underline{1}$01,111,110) is received at the other end. As we can see, four errors are introduced during the transmission. As mentioned earlier, the length of the trellis is equal to the length of the input sequence, which consists of the information bits followed by the force zero sequence. The force zero sequence, "00", forces the trellis into the initial state, so that the trace-back can be started at the initial state.

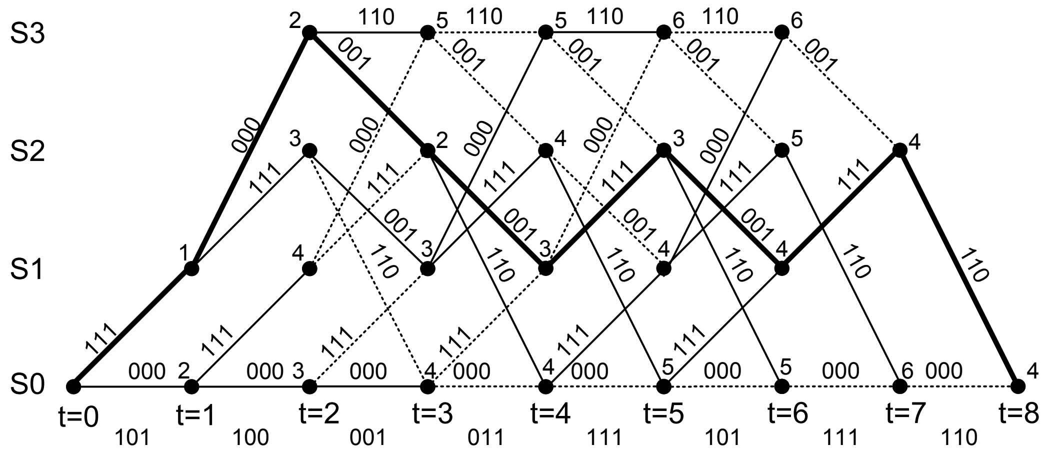


Figure 2.6 Hard-decision Viterbi decoding example [46].

An **ML** path is found with the help of a branch metric and a path metric. A branch metric is the Hamming distance between the estimated and the received code symbol. The branch metrics accumulated along a path form a path metric. A partial path metric at a state, often called as state metric, is the path metric for the path from the initial state to the given state. The surviving paths are those paths with the minimum partial path metric at each node. After surviving branches at all nodes in the trellis have been identified, there exists a unique path starting and ending at the same initial state in the trellis. The decoder generates an output sequence corresponding to the input sequence for this unique path. The procedure is explained below using the trellis diagram in Figure 2.6.

The path metric for state $S_0$ at time t = 0 is initialized to zero. At time t = 1 there is only one incoming branch for state $S_0$. This branch metric is two, which is the Hamming distance between the expected input "000" and the received input "101". The path metric of $S_0$ at time t = 1 is the sum of the path metric of $S_0$ and the current branch metric. Similarly, the path metric of $S_1$ at t = 1 is one. At t = 1 there is only one incoming branch for each node. The single branch is the survivor branch. The same process repeats for t = 2. At t = 3 there are two incoming branches for each node. For instance, at state $S_0$, one incoming branch with the partial path metric six (which is the sum of the path metric 3 of $S_2$ and the branch metric 3) is from $S_2$. The other incoming branch with the partial path metric four is from $S_0$. Compared with the two branches, the branch from $S_0$ survives and the other one is discarded. Surviving branches are described in solid lines and discarded ones are in dotted lines in Figure 2.6.

Once the trellis is tagged with partial path metrics at each node, we perform a trace back process to extract the decoded output sequence from the trellis. We start with state $S_0$ at time t = 8 and trace backward in time. The survivor path leads to state $S_2$ at time t = 7. From state $S_2$ at time t = 7, we trace back to $S_1$ at time t = 6. In this way, a unique path shown in the bold line is identified. It is pointed out that each branch is associated with specific source input bit. For example, the branch from state $S_2$ at time t = 7 to node $S_0$ at time t = 8 corresponds to a bit '0' whose bit position is the seventh in the source input sequence. So while tracing back through the trellis, the decoded output sequence corresponding to these branches is generated.

Consider a general (*n*, *k*) binary convolutional encoder with the number of memory elements *m*, given an input sequence of *kL* bits, the Viterbi algorithm is described as follows [3]. First, let the node corresponding to state $S_j$ at time t be denoted $S_{j,\,t}$. Each node in the trellis is to be assigned a value Val($S_{j,\ t}$). The node values are computed in the following manner.

1. Set Val($S_{0,\ 0}$) = 0 and t = 1;
2. At time t, compute the partial path metrics for all paths entering each node;
3. Set Val($S_{k,\ t}$) equal to the best partial path metric entering the node corresponding to state $S_k$ at time t. The nonsurviving branches are discarded from the trellis;
4. If $t < L + m$, increment t and return to step 2.

Once all node values have been computed, start at state $S_0$, time $t = L + m$, and follow the surviving branches backward through the trellis. The defined path is the maximum likelihood path.

## 2.4 VLSI Implementation of the Viterbi Algorithm

A block diagram of the Viterbi decoder is shown in Figure 2.7. It can be seen that the Viterbi decoder consists of three major units, *i.e.*, (1) a *Branch Metric Unit* (BMU); (2) an *Add Compare and Select Unit* (ACSU); and (3) a *Survivor Memory Unit* (SMU).

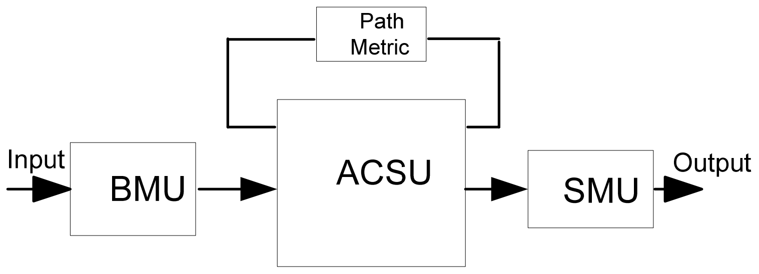


Figure 2.7 Block diagram of Viterbi Decoder.

### 2.4.1 The Branch Metrics Unit

The BMU calculates the branch metrics from the input data. It compares the received code symbol with the expected code symbol and counts the number of different bits. An implementation of the block is shown in Figure 2.8.

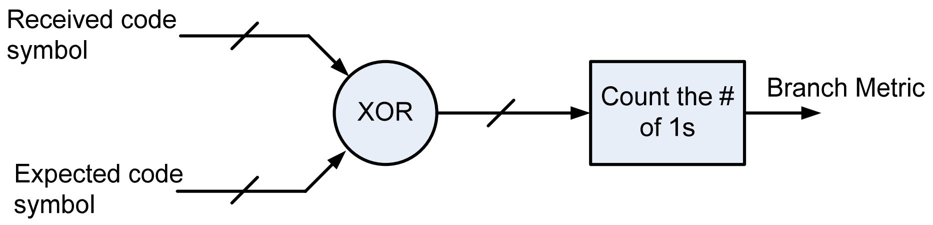


Figure 2.8 The branch metric computation block.

### 2.4.2 The Add-Compare-Select Unit

The add-compare-select (ACS) unit recursively accumulates the branch metrics to path metrics for all the incoming paths of each state and selects the path with minimum path metric as the survivor path. An ACS module is shown in Figure 2.9. The two adders compute the partial path metric of each branch, the comparator compares the two partial metrics, and the selector selects an appropriate branch.

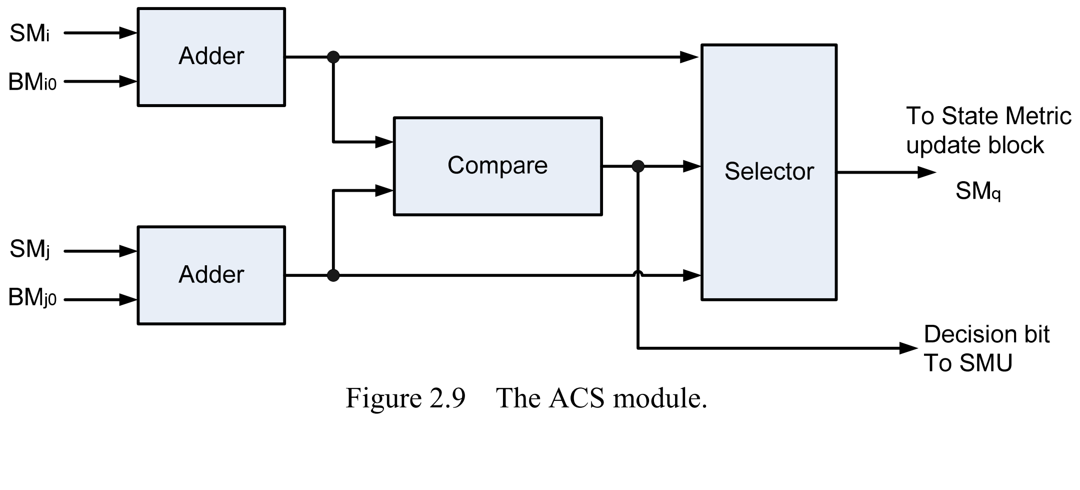


Figure 2.9 The ACS module.

### *2.4.3 The Survivor Memory Unit*

The SMU stores the information which can be used to determine the survivor path and generates the decoded sequence.

In practice, two algorithms are employed for the implementation of SMU [2], *i.e.*, register exchanges algorithm (RE) and trace back algorithm (TB). In RE algorithm, SMU computes the candidate information sequences of the survivor paths corresponding to all states with the decision bits output from ACSU. The register exchange approach assigns a register to each state. The register records the decoded output sequence along the path starting from the initial state to the final state, which is same as the initial state. Consider a trellis diagram shown in Figure 2.4. This approach eliminates the need to trace back, since the register of the final state contains the decoded output sequence. Hence, the approach may offer a high-speed operation, but it is not power efficient due to the need to copy all the registers in a stage to the next stage. The RE hardware architecture is shown in Figure 2.10.

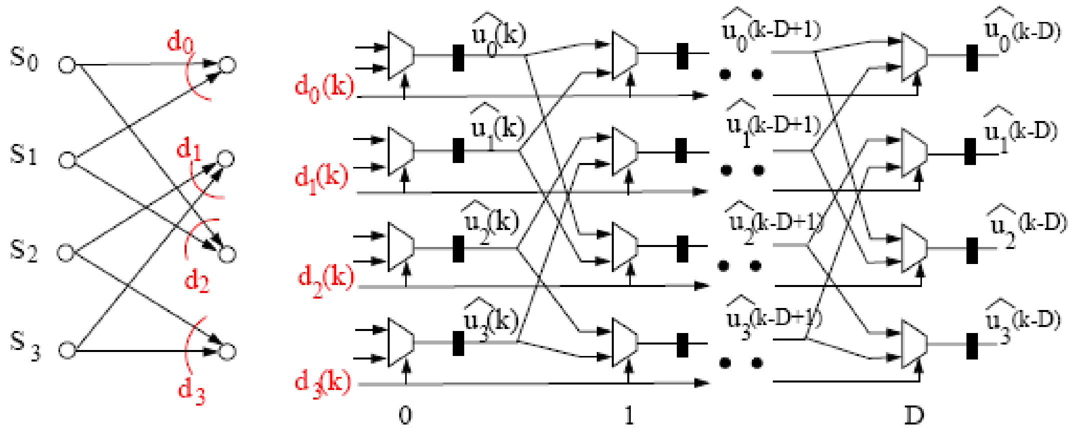


Figure 2.10 The RE architecture for the example code [6].

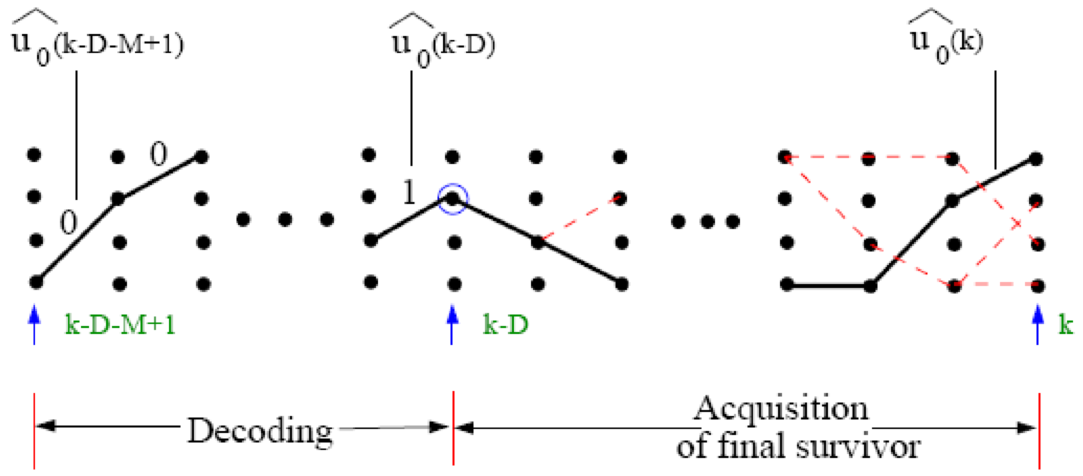

$\hat{u}_0(k\text{-}D\text{-}M\text{+}1)$
$\hat{u}_0(k\text{-}D)$
$\hat{u}_0(k)$
0
0
1
k-D-M+1
k-D
k
Decoding
Acquisition of final survivor

Figure 2.11 An example of TB method [6].

In the TB algorithm, the information sequence is extracted by the SMU with the decision bits stored in it. An example of TB method is shown in Figure 2.11. After trace back for $D$ steps, the path is traced back for $M$ steps further to obtain $M$ symbols that are associated with the final survivor. There will be trade-offs between performance and throughput in choosing $D$ and $M$. It is clear that, with TB algorithm, the decoding latency will be at least $D + M$ instead of $D$ in RE case. Also a last-in-first-out (LIFO) memory is required as block of M symbols are output in reverse order. Compared to the RE approach, the TB method consumes less power when the constraint length is moderately large.

## 2.5 Block Codes

In general, block codes break the data stream up into k-bit blocks, and (n-k) check bits are added to these blocks. The coded sequence will have n bits in total. This is referred to as a (n, k) block code. For each of the $2^{k-1}$ combinations of k-bit input block, the encoder outputs a unique n-bit sequence. The coding rate is k/n.

### *2.5.1 Definition of Block Codes*

There are two main ways to define a linear block code, either through a **generator matrix** *G* or a **parity check matrix** *H*. The relation $c = G^T x$ (module 2 sum) holds for a code defined by a generator matrix. Thus the rows of *G* (the columns of $G^T$) form a basis for the code, and the message *x* is the coordinates for the codeword *c*. In this thesis, however, we will define codes through parity check matrices. Then the set of codewords is given by the relation $Hc = 0$ (module 2 sum). The rows of *H* thus define a set of checks on the codeword *c*. The relation implies that the bits involved in each check must have an even number of ones for the word to be in the code. This definition of a code does not include a mapping between codewords and messages, but often a code is constructed such that the message bits are mapped to certain locations in the codeword. These bits are then called message bits, and the other bits are called parity bits.

### *2.5.2 Systematic Form*

A systematic parity check matrix form can be represented as $H = [P\ I]$, where *I* is the identity matrix. On this form, the parity check matrix is particularly easy to be converted to a generator matrix. By recognizing which parity bits are changed by changing one message bit and keeping the other message bits constant, we can determine the rows of the generator matrix. This leads to the corresponding generator matrix $G = [I\ P^T]$.

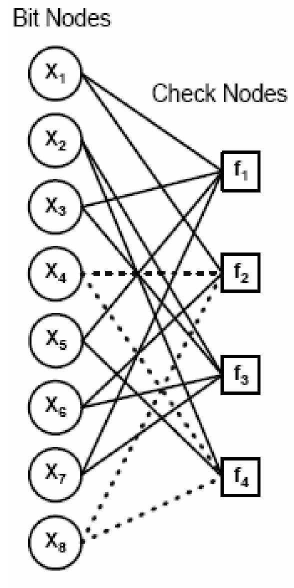
## 2.6 Low-Density Parity-Check (LDPC) Codes

### *2.6.1 Definition of LDPC Codes*

Low-Density Parity-Check codes are a class of linear block codes corresponding to the parity check matrix *H*. Parity check matrix consists of only zeros and ones and is very sparse which means that the density of ones in this matrix is very low. Originally Gallager defined an LDPC matrix as a randomly created matrix with small constant column weights and row weights [7]. For a (*Wc* , *Wr*) **regular** LDPC code each column of the parity check matrix *H* has *Wc* ones and each row has *Wr* ones. If degrees per row or column are not constant, then the code is ***irregular***. Some of the irregular codes have shown better performance than regular ones. But irregularity results in more complex hardware implementation.

### *2.6.2 Tanner Graph*

LDPC codes can be represented effectively by a bi-partite graph called a "Tanner" graph [8], [9]. A bi-partite graph is a graph (nodes or vertices are connected by undirected edges) whose nodes may be separated into two classes, and where edges may only be connecting two nodes not residing in the same class. The two classes of nodes in a Tanner graph are variable nodes and check nodes. Each variable node is associated with a digit of the codeword. Each check node is associated with a parity-check constraint. Figure 2.12 shows a Tanner graph for a simple parity check matrix *H*. In this graph each variable node is connected to two check nodes and each check node has a degree of four.



$$H = \begin{bmatrix} 1&0&1&0&1&0&1&0 \\ 1&0&0&1&0&1&0&1 \\ 0&1&1&0&0&1&1&0 \\ 0&1&0&1&1&0&0&1 \end{bmatrix}$$

Figure 2.12    Tanner graph of a parity check matrix.

*Definition:* Degree of a node is the number of branches that is connected to that node.

*Definition:* A cycle of length *l* in a Tanner graph is a loop comprised of *l* edges. The Tanner graph in the above figure has a cycle of length four which has been shown by dashed lines.

*Definition:* The Girth of a Tanner graph is the minimum cycle length of the graph. The shortest possible cycle in a bipartite graph is clearly a length-4 cycle.

### *2.6.3 Encoding*

Having the parity check matrix of a set of LDPC code, we can draw the corresponding Tanner graph. To give a general perspective about encoding of LDPC codes, we can say that one might first assign each of the information bits to a variable node in the graph, then the values of the remaining variable nodes are determined so that all the parity check constraints are satisfied.

In order to put encoding process in the matrix notation, to encode a message $x$ of $K$ bits with LDPC codes, one might compute c = $xG$ in which c is the $N$ bit codeword and $G_{KxN}$ is the generator matrix of the code in which $GH^T = 0$.

### *2.6.4 Decoding*

In addition to presenting LDPC codes in his seminal work in 1960, Gallager also provided a decoding algorithm that is effectively optimal. Since then, the decoding algorithms have been independently discovered by other researchers. The algorithm iteratively computes the distributions of variables in graph-based models and comes under different names, such as “Message Passing algorithm”, “Sum-Product (SP) algorithm” or “Belief Propagation (BP) algorithm”.  The SP decoder is a type of iterative decoder. The algorithm works by passing messages representing bit and check probabilities over the Tanner graph of the code. For each iteration, the received data is used for calculating the likelihoods of each sent bit, until the set of bits form a valid codeword or a maximum number of iterations is reached. The main strength of the SP decoder is its simplicity and inherent scalability. Every node in the graph can be considered a separate simple processing entity, receiving and sending messages along its edges. Thus, the calculations can be made either in parallel by an element for every node. The weaknesses, on the other hand, are very high memory requirements for storing interim messages, and high wire routing complexity caused by the random nature of the graph.

In the literature, various approximate belief propagation decoding algorithms [45] were proposed to simplify the decoding complexity. The overall decoding procedure of those algorithms is similar to the standard BP algorithm.

Bit flipping algorithm [7] has lower complexity than message passing algorithm at the expense of lower performance. This algorithm works on the hard decision of the received signal. It has very low decoding complexity since only simple logical operations are needed. But it has significant performance degradation compared to those soft decoding algorithms such as the SP algorithm.

# 3 LOW-POWER MEMORY-EFFICIENT VITERBI DESIGN

This chapter presents a new low-power memory-efficient trace back (TB) scheme for high constraint length Viterbi decoder (VD). With the trace-back modifications and path merging techniques, up to 50% memory read operations in the survivor memory unit (SMU) can be eliminated. The memory size of SMU can be reduced by 33% and the decoding latency can be reduced by 14%. The simulation results show that, compared to the conventional TB scheme, the performance loss of the proposed scheme is negligible.

## 3.1 Introduction

Many Viterbi decoders have been implemented in the past for different applications because of the forward error correction (FEC) capability of the decoders [11]–[14]. The constraint length of the decoders in [11] and [12] is very small. Their goal is to achieve very high decode rate through different techniques. In [11], a radix-4 ACS module has been used to achieve higher throughput by applying one level of lookahead technique. The unfolding architecture [15] and lookahead-based architecture [16] are applied for high performance decoder architectures. In [19] and [20], systolic array techniques are applied. Also in [21] and [22], improved MSB-First ACS is proposed. The decoders in [17] and [18] are mainly applied for CDMA mobile system where a convolutional code with a constraint length $K = 9$ is used.

In the literature, variations of TB algorithms have been proposed [13] [14]. They all need a large amount of memory accesses which consume significant amount of power. In [26], a dynamic TB scheme is proposed to reduce the memory access operations based on path merging and prediction techniques. It is shown that 30% of power dissipation can be reduced. However, it still has a large memory requirement and long decoding latency. In

[29], a pre-traceback architecture which reduces the memory read operations of survivor memory by 50% is proposed, but significant extra hardware is introduced, which contributes a large amount of power. In this chapter, we further exploit the existing TB schemes and propose a new low-power memory-efficient TB scheme. The remainder of this chapter is organized as follows: Section 3.2 briefly reviews the existing TB algorithms. The proposed TB scheme and the decoder architecture are presented in Section 3.3. The simulation results and discussions are presented in Section 3.4. The conclusions are drawn in Section 3.4.

## 3.2 Trace Back Algorithm

In the TB algorithm, all survivor paths will merge to the same state if they are continuously traced back for a sufficient number of stages. Some variations of the conventional TB architecture are proposed in [11] [25]. In [26], a 3-point even TB algorithm derived from the *k*-point TB algorithm [23] is proposed, where *k* is the number of read pointers to access the SMU. In the following, we start with the 3-point even algorithm and a new TB scheme will be discussed in later sections.

In the 3-point even algorithm, the survivor memory is divided into 6 banks, each with $L/2$ entries, where $L$ is the TB length. Three operations: 1) decision bit write (WR), 2) trace-back (TB), and 3) decode-read (DC) are performed in parallel at the same clock speed. As shown in Figure 3.1, the manipulation of SMU is explained as follows:

1) The WR process continuously writes decision bits into the survivor memory banks in an increasing order of memory address starting from Bank0.

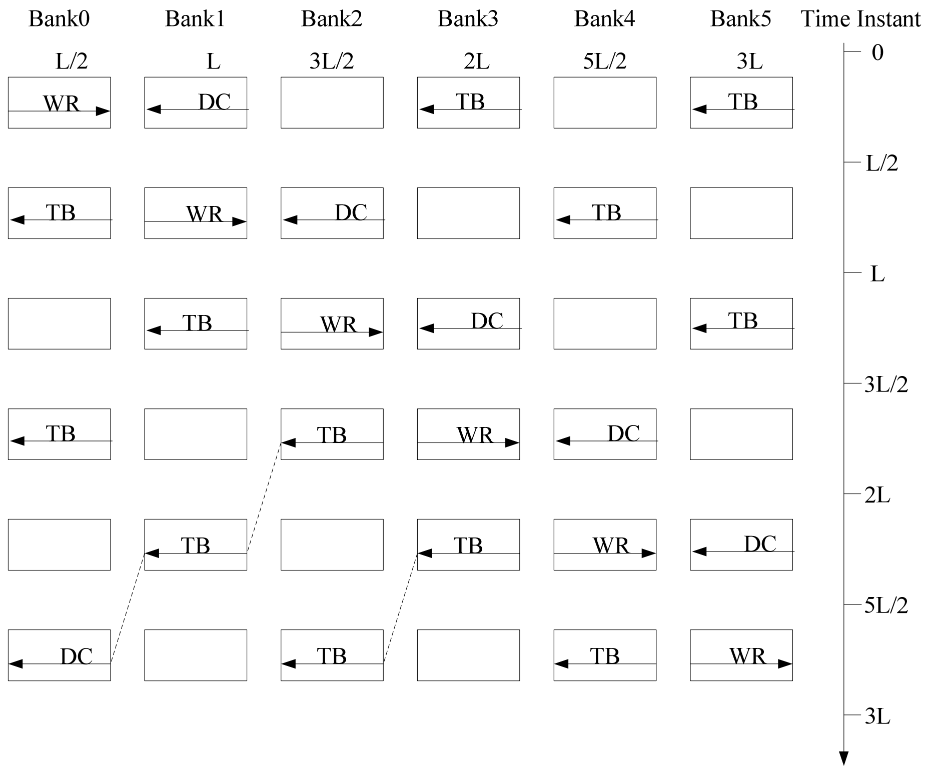


Figure 3.1 The 3-point even TB algorithm.

2) The TB process recursively computes the previous state $S_{n-1}$ based on the current state $S_n$ and the associated decision bit $D_n^s$ obtained from the survivor memory. The computation for the previous state $S_{n-1}$ is $S_{n-1} = D_n^s \| (S_n >> 1)$. In other words, the previous state is obtained by concatenating the decision bit and the current state right shifted by one bit. This process will be repeated for *L* consecutive steps.

3) The DC process starts from the state which is the output state of the $L^{\text{th}}$ TB process to decode the input sequences in the reverse order, thus a last-in-first-out (LIFO) memory is required for reversing the order before outputting the information.

Therefore, the total decoding latency including the LIFO process is 3.5*L*. The WR process performs memory write operations. The DC process and the TB process employ the same memory read operations, but in different time period. Therefore, every decision bit stored in the survivor memory experiences one write operation, which is in the WR process, and three read operations, two of which are in the TB process and one of which is in the DC process, respectively. The large latency and redundant memory read operations are the main issues to be addressed in our proposed scheme.

## 3.3 Proposed Trace Back Algorithm and Architecture Design

In the TB algorithm, all the survivor paths will merge to the same state with a high probability after continuously tracing back a number of time instances. During this process, the SMU tends to trace the same path which has been traced recently, which implies that the SMU can reuse the data which have been used in previous TB operations. We proposed a new low-power memory-efficient TB approach with path merging technique. The key ideas of the improved TB approach are as follows.

When the WR operations are completed for a memory bank in the SMU, the generated decision can be immediately exploited to compute a local TB merged path. The starting state of the local merged path is the state with the best path metric. We found that the local merged paths coincide with the global merged paths (*i.e.*, the TB path computed using the traditional approach) in a very high probability (*i.e.*, over 85% based on our simulation experience).

During the local trace back operation, the tentative decoding result can also be stored. If the local TB state is coincident with a global trace-back state, the tentative decoding result can be taken as the final decoding result.

In order to apply the path merging technique for the proposed TB approach, a buffer-based memory bank architecture is proposed as shown in Fig. 3.2. Based on the buffer-based memory bank, the path merging technique can be described as follows [26]:

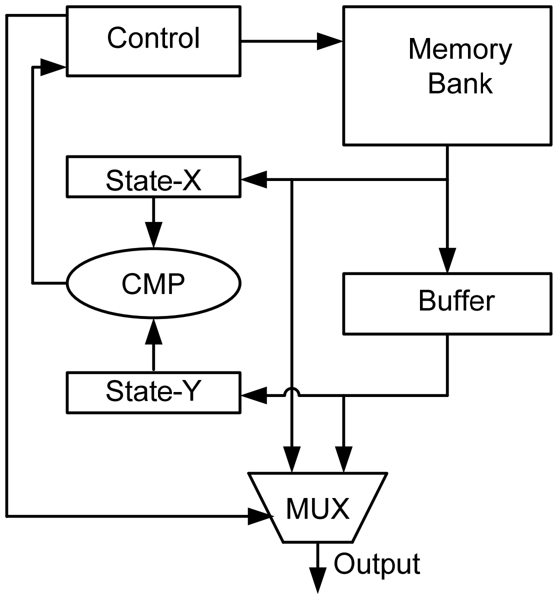


Figure 3.2 Memory bank architecture with a buffer [26].

1. Initialization

    - Survivor paths have been written to survivor memory.
    - Buffer contains the previously traced path.
    - State-X is the state with the optimal path metric.
    - t = 0.

2. The TB process

```
Repeat {
        state-X = TB(t, state-X);
        state-Y = BUF(t, state-Y);
        t = t + 1;
```

} until (t = *M*) or (state-X = state-Y);

3. Done.

In the above, TB(t, state-X) denotes the TB operations performed in memory, BUF(t, state-Y) denotes the TB operations performed in buffer, *M* stands for the TB length.

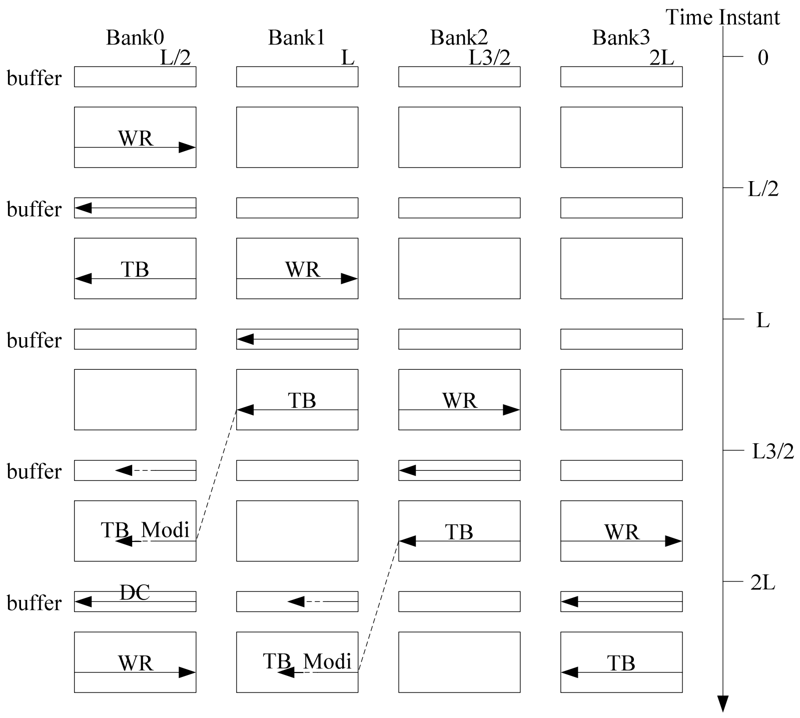


Figure 3.3 The proposed TB algorithm.

The proposed TB algorithm is shown in Figure 3.3. And the algorithm is described as follows. Compared to the 3-point even algorithm which has 6 memory banks, each with *L*/2 entries, the proposed TB algorithm has 4 memory banks, each with *L*/2 entries.

1. $t = 0$: The WR process continuously writes decision bits from ACSU into survivor memory banks in an increasing order of memory address.
2. $L/2 \le t \le L$: The TB process traces Bank0 in a decreasing order of memory address, the

traced path will be recorded into the buffer associated with Bank0.

3. $L \le t \le 3L/2$: The TB process traces Bank1 and records the traced path into the buffer associated with Bank1.
4. $3L/2 \le t \le 2L$: The TB process traces Bank0 again to modify the traced path recorded in the buffer associated with Bank0 in a decreasing order of memory address, where the path merging technique is applied. These TB operations with data modifications are denoted by TB modifications (TB Modi).
5. $2L \le t \le 5L/2$: The DC process decodes the input sequence directly in the buffer associated with Bank0, thus all the memory read operations associated with the DC process (*i.e.*, 50% of the total memory read operations in the TB process) are replaced by buffer read operations, which consume much less power. The WR process simultaneously writes the new decision bits into Bank0.

A last-in-first-out (LIFO) memory is still required for reversing the order before outputting the information sequence in this algorithm. Therefore, the total decoding latency including the LIFO process is 3*L*. The overall comparisons of the memory size and the decoding latency between the proposed scheme and the scheme in [26] are shown in Table 3.1.

TABLE 3.1 MEMORY AND LATENCY COMPARISON BETWEEN THE PROPOSED TB SCHEME AND THE TB SCHEME IN [26].

| | Proposed scheme | The scheme in [26] |
|---|---|---|
| Memory Depth | $2L$ | $3L$ |
| Decoding Latency | $3L$ | $3.5L$ |

## 3.4 Simulation Results and Discussions

In this work, two rate-1/2 convolutional codes are simulated. One has the constraint length of 7 (*i.e.*, $K$ = 7, where K represents the constraint length) with the generator polynomial (133,171), which is used in the WLAN systems. The other has the constraint length of 9 with the generator polynomial (561, 735). For the convenience of discussion, they are labeled as code-I and code-II. In all simulations, AWGN channel is applied and each data is presented in floating-point precision. Three decoders are compared: 1) the optimal Viterbi decoder [2]; 2) the conventional TB scheme [29], and 3) the proposed cache-based TB scheme. For the optimal Viterbi decoder, the TB process starts after the whole signal sequence is received. For the conventional TB decoder, a TB length of $6K$ is used. Therefore, $L$ = 54 when $K$ = 9, and $L$ = 42 when $K$ = 7. It should be mentioned that the scheme presented in [26] has the same decoding performance as the conventional TB scheme.

Figure 3.4 shows the bit error rate (BER) performance of the code-I. The conventional TB scheme and the optimal decoder can achieve almost identical decoding performance in the simulated signal to noise ratio (SNR) range. Compared to the optimal decoder, the maximum performance loss of the proposed scheme is only 0.11dB. Thus the performance loss is negligible.

Figure 3.5 shows the BER performance of code-II. Still taking the conventional TB as the base, the performance loss introduced by the proposed cache-based TB is 0.17dB at BER of $10^{-3}$ and 0.13dB at BER of $10^{-4}$. They are slightly larger than code-I shown in Figure 3.5, but can still be negligible.

The performance loss of the proposed scheme is mainly caused by the incomprehensive path merging trace process. Although most of the paths will merge in *L*/2 steps, a small amount of paths will not. Figure 3.6 shows the path merging percentage of the rate-½ convolution code with K of 7. The path merging percentages are examined at SNR = -0.7dB, 0.3dB and 1.3db, respectively. According to Table 3.2, overall, more than 85% of the paths merge naturally if the trace back is started from the state with the optimal path metric. In other words, over 85% of the data recorded in the buffer can be used directly without any modification. Furthermore, more than 98% of the paths can merge in *L*/2 steps.

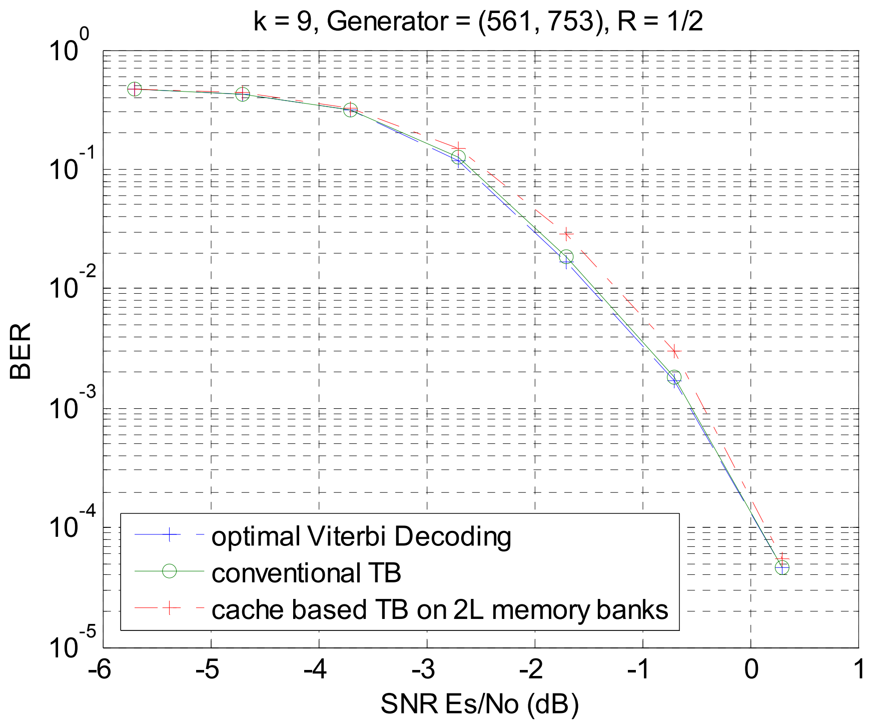


Figure 3.4 BER performance in AWGN channel for a convolutional code of *K*=9.

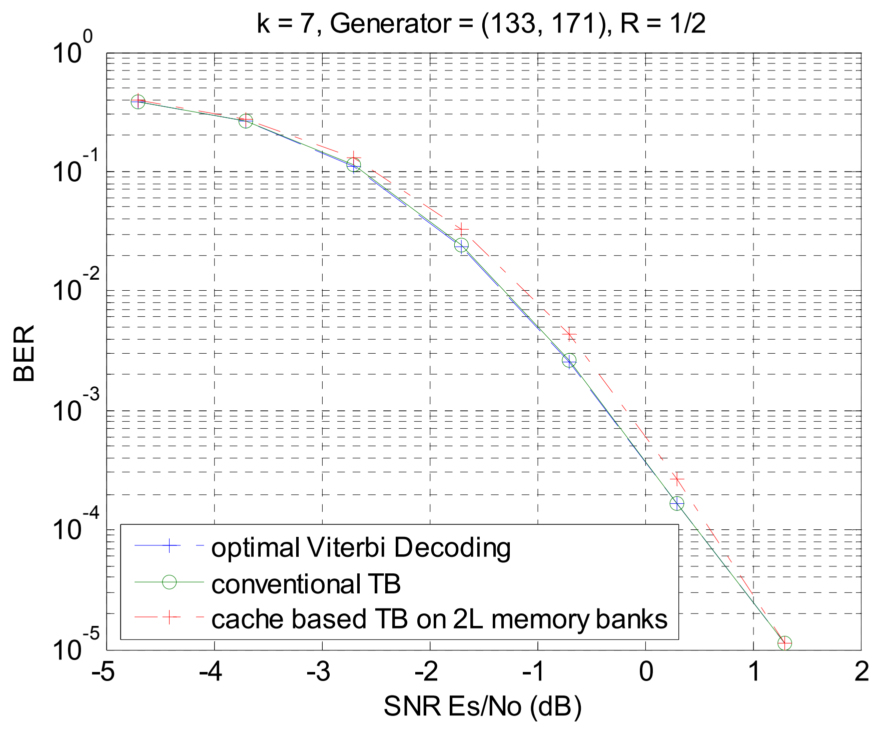


Figure 3.5 BER performance in AWGN channel for a convolutional code of *K*=7.

TABLE 3.2 PATH MERGE PERCENTAGE FOR THE *K*=7 CASE.

| SNR (dB) | -0.7 | 0.3 | 1.3 |
|---|---|---|---|
| Merge without adjusting | 85.86% | 93.80% | 97.83% |
| Merge with adjusting in *L*/2 steps | 13.02% | 6.06% | 2.16% |
| Merge with adjusting in more than *L*/2 steps | 1.12% | 0.14% | 0.01% |

## 3.5 Conclusion

In this chapter, a new low-power memory-efficient TB approach is proposed for high constraint length VD. In the proposed scheme, more than 50% memory read operations in the SMU can be reduced. The memory size of the SMU can be reduced by 33% and the decoding latency can be reduced by 14%. The simulation results show that the path can successfully merge in *L*/2 steps with a very high probability (more than 98%), and the performance loss due to the incomprehensive path merging is negligible (less than 0.2dB).

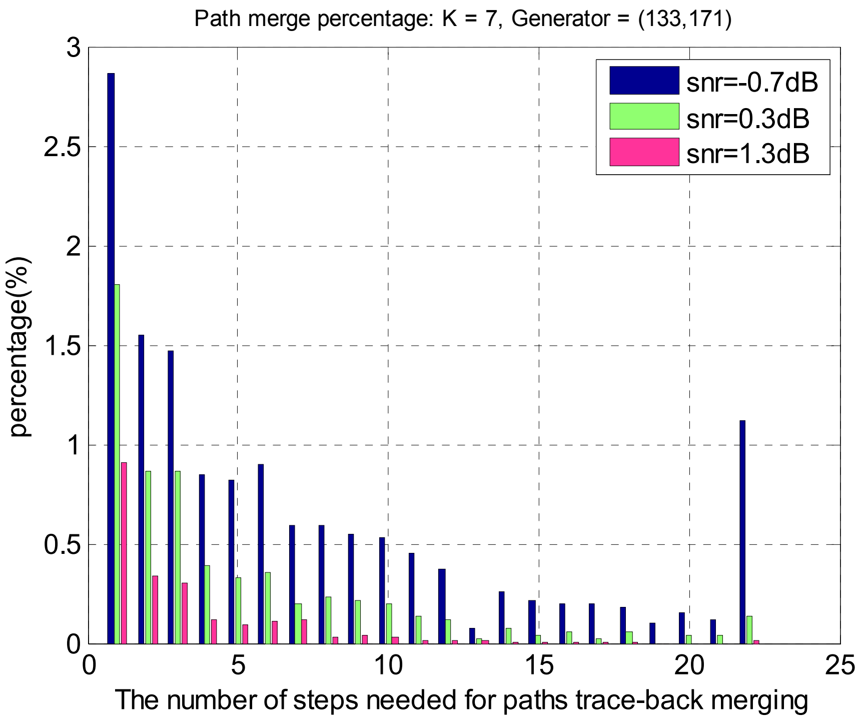


Figure 3.6 Path merging percentage.

# 4 PARALLEL VITERBI DECODER ARCHITECTURE

## 4.1 Introduction

High-speed Viterbi decoders for convolutional codes are of great interest for high-data-rate applications such as ultra-wideband communication systems [22] and data storage systems [30]. Since the add-compare-select (ACS) recursion in the Viterbi decoding algorithm contains feedback loops, the achievable throughput is limited. Several structures have been proposed to speed up the computation of ACS unit [11] [21] [33].

In [11], the lookahead technique was utilized. The throughput of an ACS unit with the $M$-step lookahead can be increased by $M$ times. However, as $M$ increases, the number of the branches in computing a state metric increases exponentially. Hence, the implementation complexity and power consumption are significantly increased if $M$ is large.

The throughput of an ACS unit can be also improved with the optimization in bit level. Usually, least significant bit (LSB) first is used in accumulation operation. But most significant bit (MSB) first computation is more suitable for compare operation. An ACS structure combining MSB-first compare-select with carry-propagation-free addition was proposed in [33]. An improved MSB first bit-level pipelined ACS unit structure was presented in [21], by balancing the settling time of different paths in the ACS unit, the length of the critical path was reduced.

On the other hand, it is possible to implement the high-speed Viterbi decoder with parallel processing techniques. A variety of block based parallel implementations of the Viterbi algorithms have been proposed [32] [34] [35]. In [34] and [35], the extra bit stuffing at the transmitter results in the reduction of the rate at which information is transmitted and received. In [32], an overlap-add decoding scheme which does not place constraints on the transmitted signal is more attractive. Because the conventional trace back method is used in [32], the overall performance of the proposed parallel Viterbi decoder is downgraded. In this chapter, a parallel processing Viterbi decoder architecture with register exchange algorithm is proposed for ultra high-throughput application.

## 4.2 Parallel Processing

The parallel processing technique exploits the concurrency available in the computation. By replicating hardware, parallel processing increases the sampling rate so that multiple inputs can be processed in parallel and multiple outputs are produced in parallel in a clock period. Therefore, the effective sampling speed is increased by the level of parallelism. Parallel processing systems are also referred to as block processing systems and the number of inputs processed in a clock cycle is referred to as the block size [31].

As mentioned above, it has already been shown that parallel processing can increase the throughput. Now, we consider using parallel processing for reducing the power dissipation. Assuming that $N$ processors are used, to maintain the same processing throughput, the clock speed can be reduced by a factor of $N$. As a result, the power dissipation can be reduced.

## 4.3 The Proposed Parallel Processing Scheme

Typically, it has been demonstrated for convolutional codes that the survivor paths merge into the final survivor path with very high probability after tracing back more than $5K$ stages back into the trellis [36], where $K$ denotes the constraint length. Similarly, when starting from unknown initial state metrics, it is found that the state metrics are independent of the starting state after tracing back more than 5*K* stages.

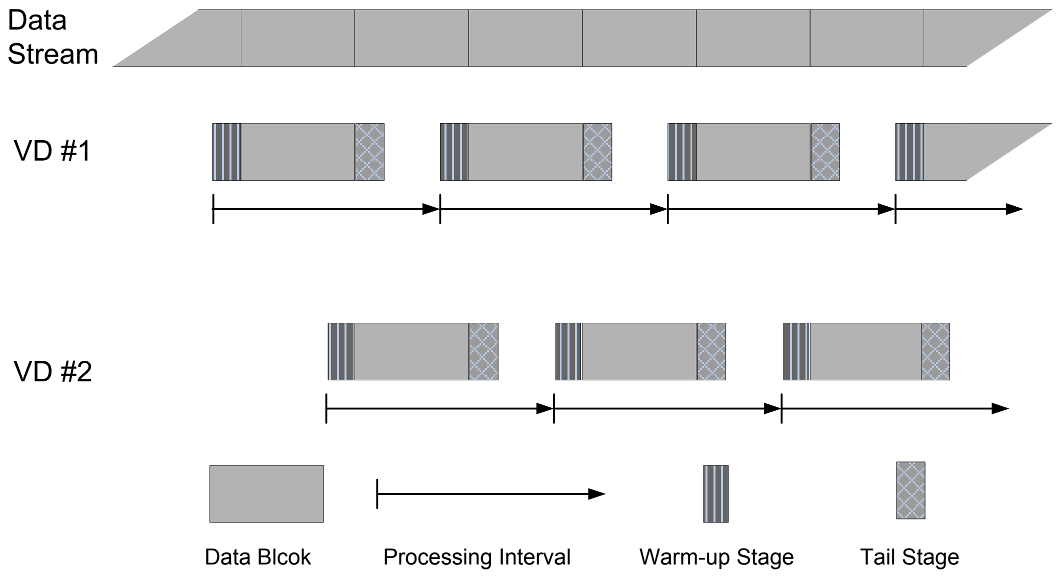


Figure 4.1 Proposed parallel Viterbi decoding scheme.

The proposed parallel Viterbi decoder (VD) scheme is shown in Figure 4.1. Two register exchange (RE) VD units are employed in this scheme. The high speed incoming data stream is divided into data blocks. In order to decode each independent block of $N$ symbols, a block of length $L + N + D$ is required for processing, where $L$ is the length of warm-up stage, $N$ is the block length to be decoded, and $D$ is the survivor path merge depth (*i.e.*, the tail stage in Figure 4.1). As we mentioned above, survivor paths merge into the final survivor path with very high probability if the survivor path merge depth is more than 5*K*. Therefore, the selected value of $D$ is more than 5*K*. In this scheme, each block is processed efficiently through each RE VD unit. The processing interval is the period for single RE VD unit to process one block. The speedup factor $S$ is a ratio between throughput of the proposed parallel VD and that of the single RE VD. Given 2 VD units, the speedup factor $S$ is as follows.

$$S = 2\left(\frac{N}{L + N + D}\right) \tag{4.1}$$

As the block length increases, this approach can ideally achieve a speedup factor equal to the number of VD units in the parallel scheme. In this example, as the number of VD units is 2, the ideal speedup factor is 2. The throughput increase is proportional to the increase in hardware complexity at the expense of longer latency.

## 4.4 The Proposed Architecture and Simulation Results

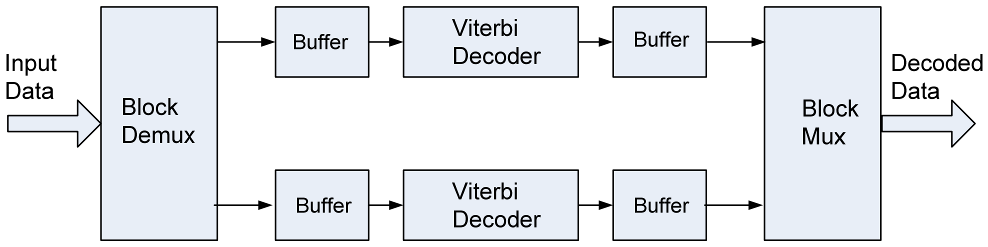


Figure 4.2 Proposed parallel Viterbi decoder architecture.

The proposed architecture for the parallel Viterbi decoding scheme in Figure 4.1 is shown in Figure 4.2. The incoming data stream is first divided into data blocks through the block demultiplexer. Based on the decoding scheme, each block is fed into the corresponding front buffer, which is a first-in-first-out (FIFO) buffer. Each VD unit then receives the data out from the corresponding front buffer for decoding processing. The decoded data popped out from the VD is fed into the rear buffer, which is also a FIFO buffer. Finally, all the decoded data out from the rear buffers are collected by block multiplexer to recover the decoded data into single decoded stream as the same order as input data.

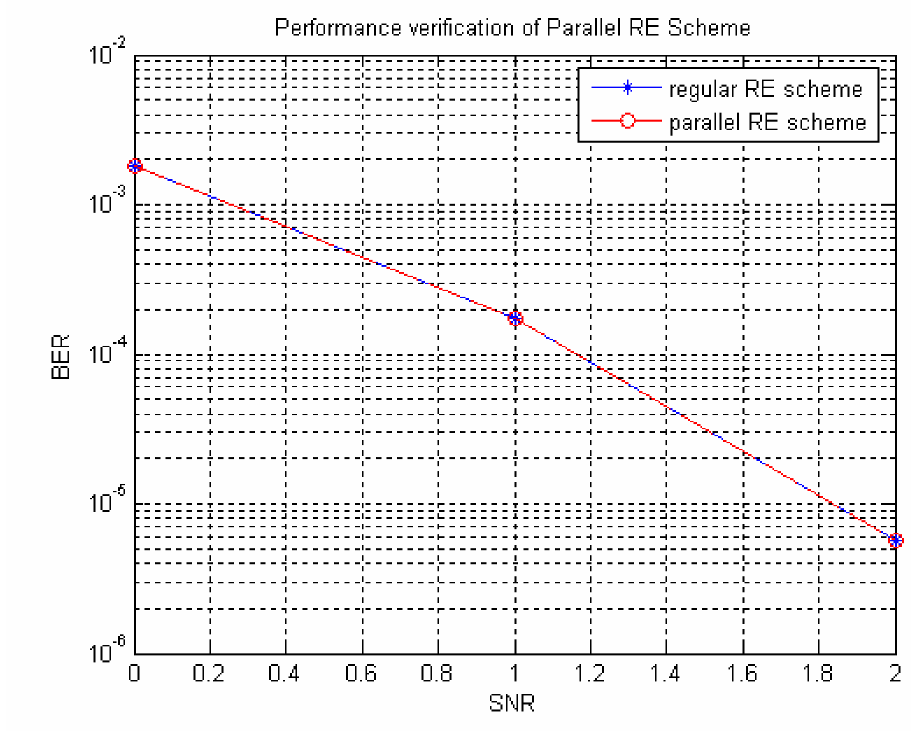


Figure 4.3 Performance comparison of the conventional RE VD with the parallel RE VD.

In this work, a rate-1/2 convolutional code is simulated, which has a constraint length of 7 with the generator polynomial (133,171) and is used in the WLAN systems. An AWGN channel is applied and each data is presented in floating-point precision. Two decoders are compared: 1) the conventional RE VD; 2) the proposed parallel RE VD. For the conventional RE decoder, an RE length of 6$K$ is used. In this case, $6 \times K = 42$ when $K = 7$. The parallel RE decoder uses the same RE length with the conventional one, therefore the tail stage is also $D = 42$. Figure 4.3 shows the bit error rate (BER) performance of the code. The conventional RE decoder and the parallel RE decoder can achieve identical decoding performance in the simulated signal to noise ratio (SNR) range from 0 to 2 dB. No performance loss can be observed.

## 4.5 Conclusion

A parallel RE Viterbi decoder architecture is proposed, which breaks the bottleneck of the ACS unit. The simulation results show that, compared to the conventional RE scheme, no performance degradation is observed for the parallel RE scheme. The proposed architecture is well suited for ultra high-throughput data applications.

# 5 EFFICIENT EARLY STOPPING SCHEME FOR LDPC DECODING

In this chapter, an early stopping scheme for low-density parity-check (LDPC) codes decoding is presented to detect undecodable blocks at early stages and hence to save unnecessary power dissipation. The proposed approach thoroughly exploits the convergence of the summation of the sign products computed in the check-to-variable message passing phase. The new approach can significantly reduce the average number of decoding iterations in the low to medium signal-to-noise ratio (SNR) range while the performance loss is negligible. In the high SNR range, the proposed scheme can turn off early stopping mechanism to avoid performance loss and unnecessary computation. The computation overhead of the proposed scheme is very small.

## 5.1 Introduction

LDPC codes [37] are a class of linear block codes which provide near Shannon limit performance on a large collection of data transmission and storage channels while simultaneously admitting inherently parallelizable decoding scheme. Recently, LDPC codes are considered for many industrial standards of next generation communication systems such as DVB-S2, WLAN (802.11.n), and 10GBaseT (802.3an). With iterative decoding LDPC codes can obtain near capacity performance. If a valid codeword is found by parity checking, the decoder will stop the iterating. Otherwise, it will continue the decoding process until a prescribed maximum iteration number is reached. At low to medium signal-to-noise ratios (SNRs), a phenomenon is frequently observed that a valid codeword cannot be found even though many decoding iterations are processed. Therefore, it is desired in real applications to detect such undecodable cases as early as possible, then terminate the decoding process in order to avoid unnecessary decoding iterations. Various early stopping criteria [39]-[41] for turbo codes decoding have been proposed in the literature. In [42], a comprehensive overview of the early stopping criteria for turbo codes decoding is presented. Because of the similarity between the turbo decoding and LDPC decoding, some existing early stopping criteria for turbo decoding can be used for LDPC decoding. However, considerable performance loss may be caused at high SNRs. A convergence of mean magnitude (CMM) early stopping criterion specially for LDPC decoding was recently presented in [43]. This criterion is based on the evolution of the average magnitude of the log-likelihood ratio (LLR) messages in the decoding process. This approach can effectively detect undecodable cases to avoid unnecessary decoding iterations. However, because it needs the accumulation of the absolute values of all LLR messages and a large bit-width multiplication operation, its computation overhead is very high.

In this chapter, we investigate the convergence and distribution of the summation of sign products computed in the check-to-variable message passing phase. Then, an efficient early stopping scheme is presented based on the investigation. Simulation results show that the proposed scheme can significantly reduce the average number of decoding iterations at low to medium SNRs. The performance loss is very small at all SNRs. In this work, the standard two-phase message passing (TPMP) Sum-Product algorithm (SPA) [38] [44] is used. It can be observed that the proposed approach is also suited for various approximations of SPA [45]. The rest of this chapter is organized as follows. In Section 5.2, the TPMP SPA is introduced. The proposed early stopping scheme and its implementation complexity are discussed in Section 5.3. The simulation results are provided in Section 5.4. Finally, Section 5.5 concludes the chapter.

## 5.2 Decoding of LDPC Codes

Commonly, the conventional TPMP SPA has been considered as the standard LDPC decoding algorithm and is usually implemented in log domain. The check-to-variable messages $R_{cv}$ are computed as (5.1)-(5.2).

$$R_{cv} = S_c \times sign(L_{cv}) \times \Psi\left\{\sum\nolimits_{n \in N(c)} \Psi(L_{cn}) - \Psi(L_{cv})\right\}, \tag{5.1}$$

$$S_c = \prod\nolimits_{n \in N(c)} sign(L_{cn}), \tag{5.2}$$

where *N(c)* denotes the set of variable nodes connected to the check node *c*, and $\Psi(x) = -\log(\tanh(|x|/2))$ is a nonlinear function. The variable-to-check message $L_{cv}$ is computed as (5.3)-(5.4).

$$L_{cv} = L_v - R_{cv}, \tag{5.3}$$

$$L_v = \sum\nolimits_{m \in M(v)} R_{mv} + I_v, \tag{5.4}$$

where $L_v$ is the LLR message of variable node *v* and *M(v)* denotes the set of check nodes connected to the variable node *v*. The intrinsic message corresponding to variable node *v* is $I_v = 2r_v / \sigma^2$, for binary input, AWGN channel, mapping 0 to +1 and 1 to -1, where $r_v$ is the received soft value. The sign of $L_v$ is taken as the estimated codeword bit $c_v$ (mapping +1 to 0 and -1 to 1). The check-sum $P_c$ of parity equation corresponding to check node *c* is computed by (5.5).

$$P_c = \oplus_{v \in N(c)} c_v, \tag{5.5}$$

where $\oplus$ represents binary addition. If $P_c = 0$ for any check node *c*, a valid code is found and the decoding process can be terminated. In VLSI design, (5.2) is implemented in the same way as (5.5).

## 5.3 Efficient Early Stopping Scheme

At low to medium signal-to-noise ratios (SNRs), a phenomenon is frequently observed that a valid codeword cannot be found even though many decoding iterations are processed. It is highly desired in real applications that an efficient scheme needs to be proposed to detect such undecodable cases as early as possible and hence to avoid unnecessary computations. After thoroughly studying the statistic characteristics of extrinsic and reliability messages computed during the decoding process, we observed that the sign of extrinsic messages and reliability messages can be used to predict whether the received block is decodable or not. For the convenience of the following discussion, let us denote $S_S$ as the summation of the binary mapping of every sign product computed in (5.2) (*i.e.*, $S_S = \sum_{c=0}^{M-1} S_c$ ) and $S_P$ as the summation of the check-sum of every parity equation computed in (5.5) (*i.e.*, $S_P = \sum_{c=0}^{M-1} P_c$ ). In LDPC decoding, the value of $S_P$ in the $k^{th}$ iteration, $S_P^k$, decreases as $k$ increases (even though a certain range of fluctuation may occur) if the decoded block is decodable. $S_P$ converges to zero when a valid code is found. It can be observed that the convergence of $S_S^k$ is very similar to that of $S_P^k$ during the decoding process. Both $S_P^k$ and $S_S^k$ can be utilized to detect undecodable blocks. In this design, $S_S^k$ is exploited for the simple hardware implementation purpose. In this section, the convergence of $S_S^k$ is shown in detail to illustrate the proposed early stopping scheme. A received block is most likely decodable if the value of $S_S^k$ monotonically decreases as $k$ increase during the LDPC decoding. In various cases, if $S_S^k$ fluctuates during the decoding, the received block is possibly undecodable.

Figure 5.1 shows the trend of variation of $S_S^k$ at SNR of 1.2 dB for ten randomly picked undecodable blocks. A (4000, 2000) (3, 6) LDPC code is used in the experiment. It can be observed that $S_S^k$ fluctuates in a small range of magnitudes in most cases. Figure 5.2 shows the variation of $S_S^k$ at SNR of 1.2 dB for ten randomly picked decodable blocks. Compared to the cases in Figure 5.1, it is clearly shown that the fluctuation of the cases in Figure 5.2 is short and $S_S^k$ converges to zero along a steep slope in most cases. Even if in the cases that $S_S^k$ keeps fluctuating with a long period, the fluctuant magnitude is much larger than that shown in Figure 5.1. It implies that the convergence of $S_S^k$ can be utilized to predict the decoding convergence before the maximum number of iterations is reached or a valid codeword is found.

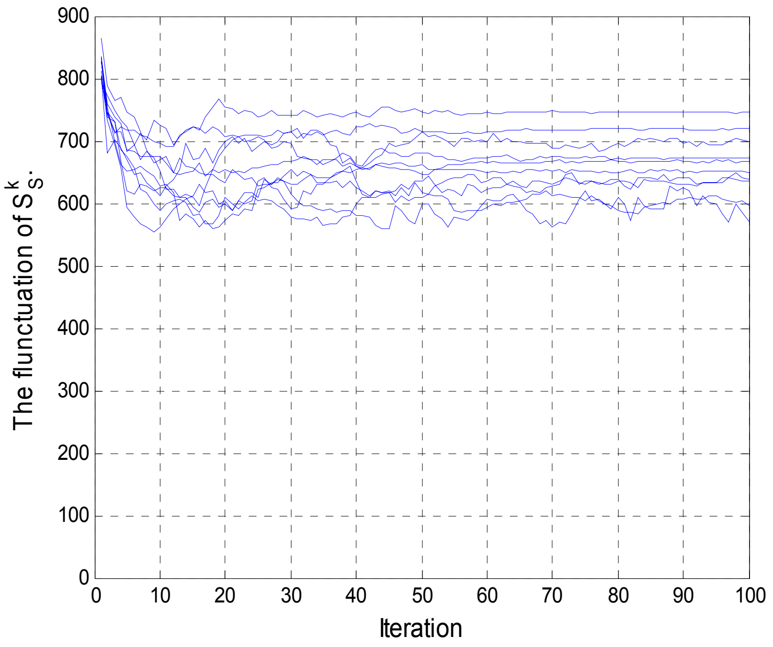


Figure 5.1 The fluctuation of $S_S^k$ for ten undecodable blocks .

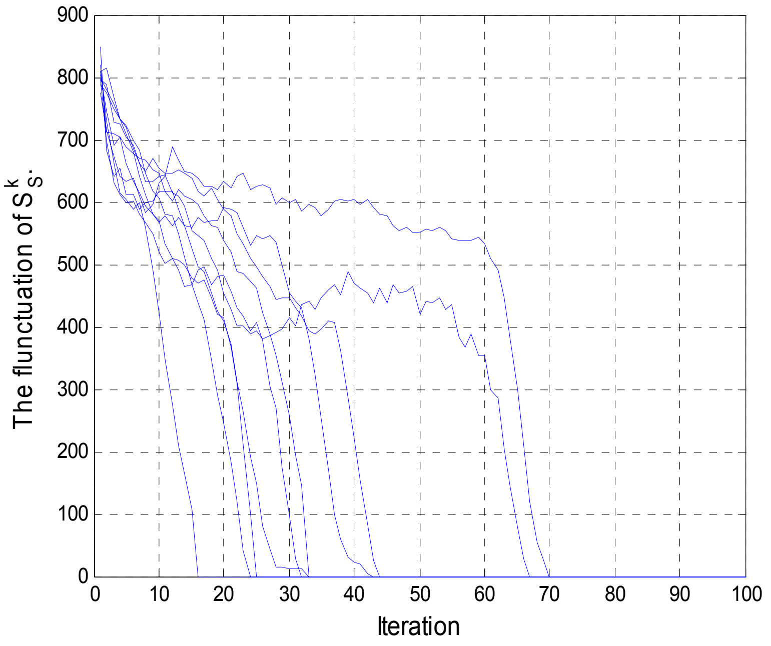


Figure 5.2 The fluctuation of $S_S^k$ for 10 decodable blocks .

It should be pointed out that any individual detection trial may have three possible outcomes, *i.e.*, hit, miss detection, and false alarm. In LDPC decoding, a false alarm causes the performance loss. Thus, early stopping schemes should be optimized to minimize the false alarm rate at all SNRs. However, the block error rate is very small at high SNRs and thus the portion of computation power specifically for undecodable blocks at high SNRs is negligible. The early stopping scheme should be disenabled at high SNRs to avoid performance loss and save computation overhead. Based on the above observation, an early stopping scheme for detecting the undecodable blocks is proposed as follows:

1) Check the value SNR to decide the SNR range in the first iteration. Step 2 is performed if in low to medium range, otherwise directly go to step 3.
2) cnt:=0;

   $\Delta := S_S^{k-1} - S_S^k$ ;

```
flag = 1 if $\Delta < 0$ once (fluctuation occurred)
while (flag = = 1) do
{
  if ( $\Delta > 0$ ) then
      if ( $\Delta < \Delta_{TH}$ ) then
          cnt:=cnt+1;
      else
          cnt:=0;
      end
  end
  if (cnt >T) then
       stop decoding
  else
      go to step 3
  end
}
```

3) Continue to the next iteration.

In step 2, $\Delta_{TH}$ and $T$ are two predetermined thresholds by simulation. $S_S^k$ converges if $\Delta > 0$ is satisfied. The condition of $\Delta < \Delta_{TH}$ indicates that a slow convergence speed occurs. $T$ is for recording the duration of slow convergence. The proposed early stopping scheme can be implemented with a $\log_2 M$ -bit accumulator for counting the number of 1s from the binary mapping of $S_c$ and a small number of additional logic gates. Therefore, the hardware overhead is very small.

The value of $S_S^0$ (the $S_S^k$ obtained in the first iteration) can be utilized to roughly check the SNR region when the channel SNR is unknown. Figure 5.3 shows the distribution density of the value of $S_S^0$ at the SNR of 2.5 dB and 1.0 dB. Totally $10^5$ blocks were simulated in this experiment. From Figure 5.3 we can see that if $S_S^0$ is greater than

780, the probability of SNR > 2.5dB is very small. Thus the detection scheme can be enabled. Otherwise, it should be disenabled to avoid performance loss and save computation overhead.

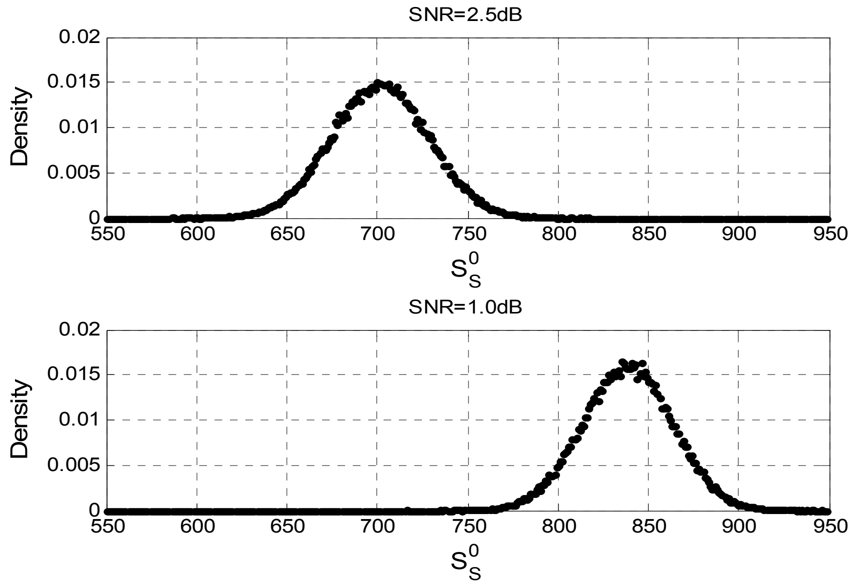


Figure 5.3 The $S_S^0$ distribution density at the SNR of 2.5dB and 1.0dB.

## 5.4 Simulation Results

In this work, two LDPC codes are simulated for the proposed early stopping scheme. One is a 4000-bit (3, 6) rate-0.5 LDPC code and the other is a 1974-bit (5, 10) rate-0.5 LDPC code. TPMP SPA is used for decoding. For practical purpose, we assume that SNR information is not available for the proposed early stopping scheme and $S_S^0$ is utilized to roughly determine the SNR region. In addition, based on our simulation, 780 and 440 are used as thresholds of $S_S^0$ for the decoding of the 4000-bit LDPC code and the 1974-bit code, respectively. The maximum iteration number is set to be 100 in this experiment. During this experiment, the standard approach means that the decoding is stopped only if the maximum iteration number is reached or a valid codeword is found. CMM means the early stopping scheme proposed in [43]. In Figure 5.4, the average number of iterations needed for decoding the 4000-bit LDPC code is shown. It can be observed that the proposed approach can significantly reduce the average iteration number at low to medium SNRs. The decoding performance for the same code is demonstrated in Figure 5.5. The proposed scheme has better decoding performance than CMM criterion at high SNRs. Figure 5.6 and Figure 5.7 show the needed average number of iterations and decoding performance for the 1974-bit code. It demonstrates again that the new early stopping scheme can significantly reduce the average number of iteration at low to medium SNRs. The performance loss is very small at all SNRs.

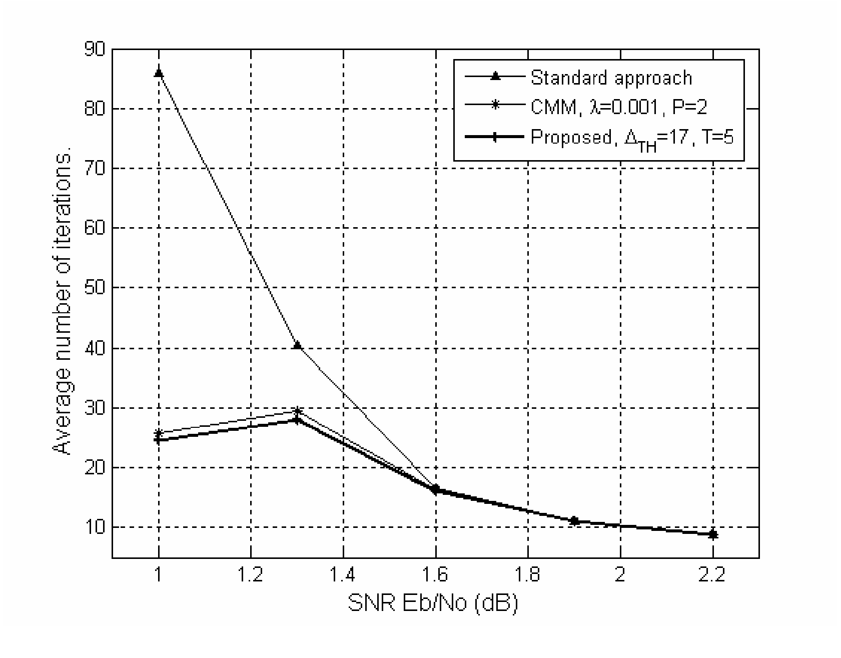


Figure 5.4 The average number of iterations needed for decoding the 4000-bit LDPC code.

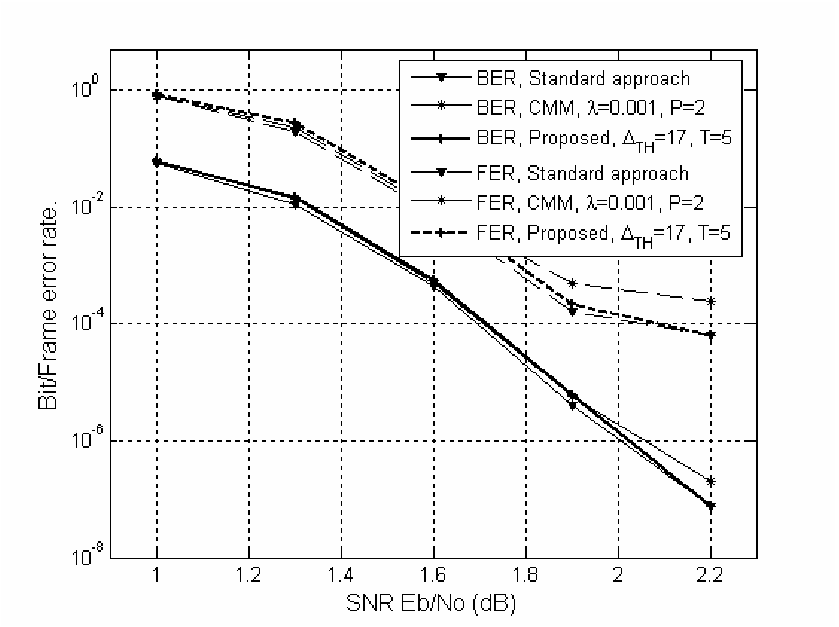


Figure 5.5 Decoding performance for the 4000-bit LDPC code.

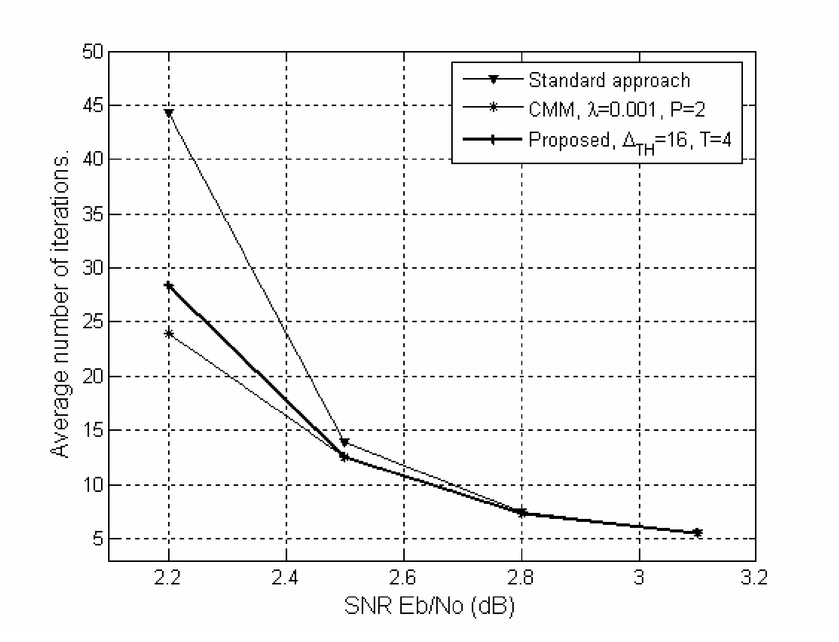


Figure 5.6 The average number of iterations needed for decoding the 1974-bit LDPC code.

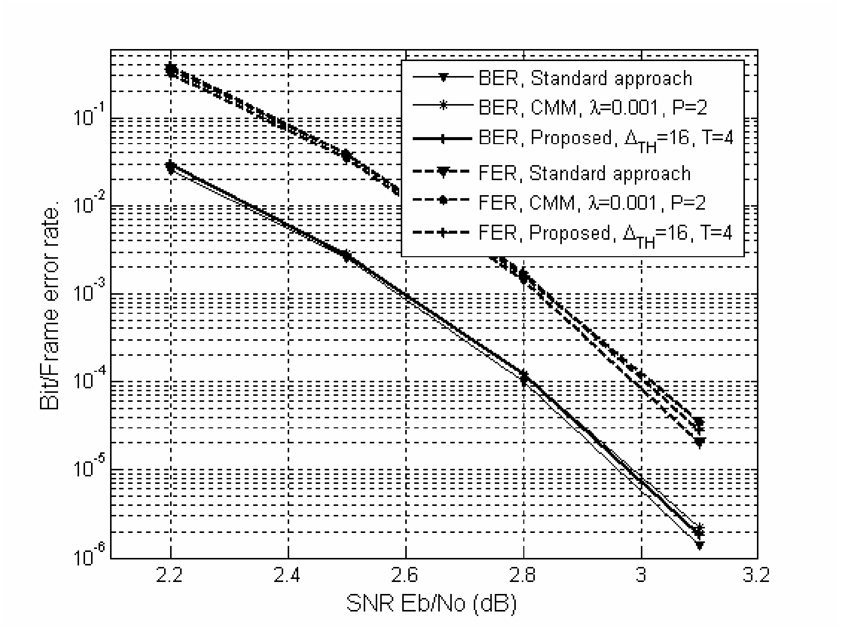


Figure 5.7 Decoding performance for the 1974-bit LDPC code.

## 5.5 Conclusion

In this chapter, an efficient early stopping scheme has been proposed, which exploits the convergence and distribution of the summation of the sign products computed in the check-to-variable message phasing phase. The overhead of hardware implementation is very small. Simulation results have shown that the proposed scheme can significantly reduce the average number of decoding iterations at low to medium SNRs. The performance loss is very small at all SNRs.

# 6 CONCLUSION

This thesis investigates efficient decoding approaches for Viterbi decoders and LDPC decoder. We propose a new low-power memory-efficient trace-back (TB) scheme for high constraint length Viterbi decoder (VD). With the trace back modifications and path merging techniques, more than 50% memory read operations in the survivor memory unit (SMU) can be eliminated. The memory size of SMU can be reduced by 33% and the decoding latency can be reduced by 14%. The simulation results show that, compared to the conventional TB scheme, the performance loss of this scheme is negligible.

A parallel RE Viterbi decoder architecture is proposed, which breaks the bottleneck of the add-compare select (ACS) unit. The simulation results show that, compared to the conventional RE scheme, no performance degradation is observed for the parallel RE. The proposed architecture is well suited for high-throughput data applications.

We also propose an early stopping scheme for LDPC codes decoding to detect undecodable blocks at early stages and hence to save unnecessary power dissipation. The proposed approach thoroughly exploits the convergence of the summation of the sign products computed in the check-to-variable message passing phase. The new approach can significantly reduce the average number of decoding iterations in the low to medium signal to noise ratio (SNR) range while the performance loss is negligible. In the high SNR range, the proposed scheme can turn off early stopping mechanism to avoid performance loss and useless computation. The computation overhead of the proposed scheme is very small.

# 7 PUBLICATIONS

Lupin Chen, Jinjin He, and Zhongfeng Wang, "Design of Low-Power emory- efficient Viterbi Decoder, " *in Proc of IEEE 2007 Workshop on Signal Processing Systems (SiPS),* Oct. 2007.

Zhiqiang Cui, Lupin Chen, and Zhongfeng Wang, "An Efficient Early Stopping Scheme for LDPC Decoding, *" in Proc of 13th NASA Symposium on VLSI Design 2007*, June, 2007

# 8 BIBLIOGRAPH

[1] J. G. Proakis. Digital Communications, volume 3. McGRAW Hill, 1995.

[2] Viterbi. Orthogonal tree codes for communication in the presence of white gaussian noise. IEEE Transactions on Communications, Apr. 1967.

[3] Joy A. Thomas Thomas M. Cover. Elements of Information theory. John Wiley and Sons, 1991.

[4] S.B. Wicker. Error Control Systems for Digital Communication and Storage. Prentice Hall, New Jersey, 1995.

[5] E. Yeo, S. Augsburger, Wm. R. Davis, and B. Nikolic, "Implementation of high throughput soft output viterbi decoders," in IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000, ICASSP'00, pp. 3378-3381, June 2000.

[6] Zhongfeng Wang. High performance, low complexity VLSI design of turbo decoders, Ph.D. thesis, University of Minnesota, Sept. 2000.

[7] Robert Gallager. Low-density parity-check codes. PhD thesis, Massachusetts Institute of Technology, 1963.

[8] David MacKay. Good error-correcting codes based on very sparse matrices. IEEE Transactions on Information Theory, vol. 45(2):399–431, March 1999.

[9] R.M. Tanner. A recursive approach to low complexity codes. IEEE Transactions on Information Theory, 27(5):533{547, Sep 1981.

[10] B.J. Frey F.R. Kschischang and H.A. Loeliger. Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory, 47(2):498{419, Feb. 2001.

[11] Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," IEEE J. Solid-State Circuits, vol. 27, pp. 1877–1885, Dec. 1992.

[12] A. K. Yeung and J. Rabaey, "A 210-Mb/s, radix-4 bit-level pipelined Viterbi decoder," ISSCC Dig. Tech. Papers, pp. 88–89, Feb. 1995.

[13] J. K. Hinderling et al., "CDMA mobile station modem ASIC," IEEE J. Solid-State Circuits, vol. 28, pp. 253–260, Mar. 1993.

[14] I. Kang and A. N. Willson, Jr., "Low-power Viterbi decoder for CDMA mobile terminals," IEEE J. Solid-State Circuits, vol. 33, pp. 473–482, Mar. 1998.

[15] K. K. Parhi, "High-speed VLSI architectures for Huffman and Viterbi decoders," IEEE Trans. Circuits Syst. II, vol. 39, pp. 385–391, June 1992.

[16] H. D. Lin and D. G. Messerschmitt, "Algorithm and architectures for concurrent Viterbi decoding, " IEEE International Conference on Communications, vol. 2, pp. 836 – 840, June 1989.

[17] J. K. Hinderling et al., "CDMA mobile station modem ASIC," IEEE J. Solid-State Circuits, vol. 28, pp. 253–260, Mar. 1993.

[18] I. Kang and A. N. Willson, Jr., "Low-power Viterbi decoder for CDMA mobile terminals," IEEE J. Solid-State Circuits, vol. 33, pp. 473–482, Mar. 1998.

[19] C.-Y. Chang, K. Yao, "Systolic Array Processing of the Viterbi Algorithm," IEEE Transaction on Information Theory, Vol. 35, No. 1, Jan. 1989.

[20] M. Gao, C. Wang, "FPGA Design and Implementation of a Low-Power Systolic Array-Based Adaptive Viterbi Decoder," IEEE Transaction on Circuits and System – 1: Regular papers, Vol. 52, No. 2, Feb 2005.

[21] K. K. Parhi, "An Improved Pipelined MSB-First Add-Compare Select Unit Structure for Viterbi Decoders," IEEE Transaction on Circuits and System – 1: Regular papers, Vol. 51, No. 3, pp. 504 - 511 March 2004.

[22] J. Tang, K. K. Parhi, "Viterbi Decoder for High-Speed Ultra-Wideband Communication Systems," ICASSP 2005, Vol. 5, pp. v/37 - v/40, March 2005.

[23] G. Feygin and G. Gulak, "Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decodes," IEEE Trans. on Commun, Vol. 42, No. 3, pp. 425-429, March 1994.

[24] E. Yeo, S. Augsburger, Wm. R. Davis, and B. Nikolic, "Implementation of high throughput soft output viterbi decoders," in IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000, ICASSP'00, pp. 3378-3381, June 2000.

[25] R. Cypher and C. Shung, "Generalized Trace Back Techniques for Survivor Memory Management in the Vitrbi Algorithm," in Proceedings of the IEEE Global Telecommunications Conference GLOBECOM, (San Diego, California), pp. 707A.1.1-707A.1.5, IEEE, Dec. 1990.

[26] C. C. Lin, Y. H. Shih, H. C. Chang, and C. Y. Lee, "Design of a Power-Reduction Viterbi Decoder for WLAN Applications," IEEE Transaction on Circuits and System – 1: Regular papers, Vol. 52, No. 6, June 2005.

[27] R. Cypher and C. Shung, "Generalized trace back techniques for survivor memory management in the viterbi algorithm," IEEE Global Telecommunications Conference, GLOBECOM, p. 1318 to 1322, Dec. 1990.

[28] R. Henning and C. Chakrabarti, “An approach for adaptively approximating the Viterbi to rduce power consumption while decoding convolutional codes,” IEEE Trans. on Signal Processing, Vol. 52, No. 5, pp. 1443-1451, May 2004.

[29] Y. Gang, A. T. Erdogan, and T. Arslan, “An efficient pre-traceback architecture for the Viterbi decoder targeting wireless communication application,” IEEE Trans. On Circuits and Systems – 1: Regular Papers, Vol. 53, No. 9, pp. 1918-1927, Sept. 2006

[30] F. Sun and T. Zhang, “Quasi-reduced-state soft-output viterbi detector for magnetic recording read channel, ” IEEE Transactions on Magnetics, accepted, 2007

[31] K. K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation. New York: Wiley, 1999, ch. 2.

[32] P. J. Black and T. H.-Y. Meng, “A hardware effcient parallel viterbi algorithm,” Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on 3-6 April 1990 Page(s):893 - 896 vol.2

[33] G. Fettweis and H. Meyr, “High rate Viterbi processor: A systolic array solution,” IEEE J. Select. Areas Commun., vol. 8, pp. 1520–1534, Oct. 1990.

[34] G. Fettweis and H. Meyr. “Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck,” IEEE Trans. Commun., Vol COM-37, No. 8, August 1989, pp 785-790.

[35] K. A. Wen and J. Y. Lee. “Parallel processing for Viterbi algorithm,” Electronics Letters, Vol. 24, No. 17, Aug 1988,

[36] J. A. Heller and LM. Jacbos. “Viterbi decoding for satellite and space communication,” IEEE Trans. Commun. Technol., 61, March 1973, pp. 268-278. Vol. COM-19, NO. 5

[37] J. B. Anderson, “Limited search trellis decoding of convolutional codes,” IEEE Trans. Inf. Theory, vol. 35, pp. 944–955, Sep. 1989.

[38] R. G. Gallager, “Low-density parity-check codes,” IRE Transactions on Information Theory, vol. IT-8, pp. 21-28, Jan. 1962.

[39] R. Y. Shao, S. Lin, and M.P.C.Fossorier, “Two simple stopping criteria for turbo decoding,” IEEE Trans. Comm., vol. 47, no. 8, pp. 1117 – 1120, Aug. 1999.

[40] A. Matache, S. Dolinar, and F. Pollara, “Stopping rules for turbo decoders,” Tech. Rep., Jet Propulsion Laboratory, Pasadena, California, Aug. 2000.

[41] Z.Wang and K. K. Parhi, “Decoding metrics and their applications in VLSI turbo decoders,” in Proc. ICASSP, 2000, pp. 3370– 3373.

[42] Z. Wang, Y. Zhang, and K. K. Parhi, "Study of early stopping criteria for Turbo decoding and their applications in WCDMA systems," in Proc of ICASSP'06, pp. III-1016-1019, May 2006.

[43] J. Li, X. H. You and J. Li, "Early stopping for LDPC decoding: convergence of mean magnitude (CMM)," IEEE Comm. Letters, vol. 10, no. 9, pp. 667 - 669 Sept. 2006.

[44] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inform. Theory, vol. 45, pp. 399-431, Mar. 1999.

[45] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, X. Hu, "Reduced-complexity decoding of LDPC codes," IEEE Trans. on Commun., vol 53, pp. 1288-1299, Aug. 2005.

[46] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," The 23rd IEEE Convention of Electrical and Electronics Engineers in Israel, pp. 223-226, Sept., 2004.

[47] Samirkumar Ranpara, "On a Viterbi decoder design for low power dissipation," Master of Science thesis, the Virginia Polytechnic Institute and State University, 1999.

[48] Claude E. Shannon, "A mathematical theory of communication," Bell System Technical Journal, 27, 1948.